JJSweep: A 3D Reconstruction Using Sweeping

Group JJS
Joshua Fantillo, John Ordoyo, Silas Myrrh
December 9th, 2024
Simon Fraser University

Abstract. Representing 3D models with minimal information while capturing the geometric features is a hard problem. Primitive shapes such as sweep surfaces to encode 3D models can highly be effective at encoded geometries of varying complexity. There is an art in striking a balance between simple models with high efficiency and lower accuracy vs heavier but more powerful and complex shape abstraction. Introducing JJSweep a sweeping primitive architecture optimized for use in 3D physics simulations and game development. We demonstrate 3D abstraction that can capture concave geometries of complex shapes and is computationally efficient. Scoring metrics such as Chamfer Distance, Hausdorff Distance, and Mean Squared Error validate the high fidelity and efficiency of the reconstructed models, achieving up to a 93% reduction in Chamfer Distance compared to the baseline.

Introduction

3D geometries require heavy data structures such as point clouds, triangle meshes and voxel grids to comprehensively capture fine details. There are many cutting edge techniques to get more use out of each bit of information in the 3D representation. Primitives shapes such as cuboid, superquadrics, parametric surfaces, convex shapes, neural partsm sketch and extrude, and are popular due to their simplicity and efficient space complexity. The problem is that alone these primitives are not expressive. The challenge then lies in applying these primitive shapes to capture as much detail as possible while minimizing the number of primitive shapes. Solving this problem offers many opportunities in a wide variety of applications. Physics simulations can utilize the simplicity of primitive shapes to detect collisions between objects in vast scenes while preserving the intricate behaviour of complex shapes. Game Development could see use of automatic object collider generation allowing for the abstraction of novel user created objects that are physically functional shapes without sacrificing runtime efficiency. VR/AR/MR applications will benefit immensely with this algorithm, especially since devices running these applications are not as computationally powerful.

The design of JJSweep aims to create 3D representations that functionally represent complex 3D geometry and can be used to generate 3D object colliders for applications in simulations and game development. Our implementation first convert 3D meshes into voxel grids for further processing. Next the voxel grid is segmented into an arbitrary k number of parts using k means. This is a critical step in ensuring our model performs well at capturing the complete geometry of the model while effectively minimizing space complexity. The manually choice of k poses challenges for generalization of our approach. We therefore choose k to be twenty unless otherwise stated throughout this paper to allow for ease of interpretation and comparison of other modeling parameters. Following part segmentation of the voxel grid we abstract each part separately using our primitive sweeping technique Finally the swept surfaces can be converted back into simpler meshes for use in simulation environments such as Unity.

We successfully demonstrated that part segmentation of voxel grids using k means aids in the sweepers expressiveness of the generated representation Visual, alignment scores, accuracy metrics. The sweeping implementation itself as the core component of our model performs better in every aspect compared to the baseline model we use.

Related Work

Primitive-based shape abstraction is a widely researched topic in geometric modeling. At its core, it is driven by the pursuit of understanding not only how to reconstruct geometry but also how geometry can be interpreted. Various primitives employ novel techniques designed to encode 3D models. Shape abstraction not only enables new ways for machines to understand geometry but also enhances interactivity and semantic insight. SweepNet [8] is a neural implementation that utilizes parametric sweep surfaces. Symmetry can also aid in part segmentation, facilitating structural understanding of 3D geometry.

RANSAC employs simple iterative subsampling techniques to fit various geometries, including lines, curves, planes, spheres, cubes, and cylinders. Similarly, ICP is a fitting technique used to geometrically align two potentially diverse datasets. With so many techniques available, it can be challenging to determine which primitive techniques are best suited to a specific challenge.

Various metrics are commonly used to evaluate 3D shape fitting and quantitatively compare algorithms. Chamfer Distance measures the average distance between the closest points in a matched dataset, providing a metric for alignment accuracy. Voxel overlap determines the proportion of shared voxels between two voxel grids, offering a measure of volumetric similarity.

Method

Our approach uses a combination of clustering, and sweeping to represent 3D models with cylindrical primitives and neural network-based spherical approximations. SphereNet, introduced as part of our baseline, was leveraged for comparison purposes and is parameterized to approximate models using spheres[3]. For our pipeline, we extended this approach to use cylindrical primitives parameterized by their circular cross-sections. A cylinder can be mathematically described by a cross-section in the xy-plane with a radius r, swept along a path defined by the voxel centers. This parameterization allows for flexible adaptation to the 3D structure of clusters. Mathematically, the circle is defined as:

$$x = r \cdot \cos(\theta), \quad y = r \cdot \sin(\theta), \quad z = 0$$

where θ ranges from 0 to 2π, and r is scaled according to the voxel grid size. The trajectory of the cylinder is defined by the cluster's voxel center points, forming a continuous path in 3D space. Depth values are added to ensure the cylinders form closed volumes suitable for visualization and further analysis. In parallel, SphereNet[3] introduces spherical primitives parameterized by their center (xc,yc,zc)and radius rs. These parameters minimize the Signed Distance Field (SDF)[6] discrepancy between the input model and the reconstructed geometry, defined as:

$$\mathrm{SDF}(\mathbf{q}) = |\mathbf{q} - \mathbf{c}| - r_s$$

where q is a query point, and c is the center of the sphere.

The algorithm begins with argument parsing to allow dynamic configuration of parameters such as the number of clusters, voxel grid size, and file paths. This flexibility ensures adaptability to models of varying complexity, enabling granular control over the reconstruction process. The pipeline then imports the 3D model using the Trimesh[1] library, which supports formats like .obj and .stl. Trimesh[1] constructs a mesh object that encapsulates the vertices, faces, and edges of the model. This mesh serves as the foundation for all subsequent steps and can be visualized interactively for debugging or validation. Following import, the model undergoes voxelization, which converts the continuous surface into a grid of discrete cubic voxels. This step simplifies geometric complexity and is pivotal for clustering. By using a configurable grid size, the resolution of the voxelization can be tuned; smaller grid sizes yield finer detail at the expense of computational cost.

After voxelization, the voxel points are segmented into k clusters using the K-means algorithm provided by Scikit-learn[2]. Clustering minimizes intra-cluster variance, dividing the voxelized model into spatially coherent substructures. Each cluster is assigned a unique label, enabling the pipeline to handle localized reconstruction for individual segments of the model. These clusters are then processed further through a sweeping mechanism. Sweeping begins by defining a circular cross-section for each cluster. The cross-section is scaled dynamically based on the grid size to ensure it matches the resolution of the voxel grid. This circular profile is swept along the trajectory defined by the cluster's voxel centers. For each adjacent pair of points along the trajectory, the cross-sections are connected with faces, forming a seamless tubular surface. To ensure the mesh is not hollow, a depth parameter is introduced, which creates additional layers to close the volume at the start and end of the path. Reversing the mesh and appending the inverted version to the original geometry resolves visualization issues caused by missing normals, ensuring the model appears solid from all angles.

SphereNet[3] complements this pipeline by leveraging a neural network-based approach to refine the reconstruction process using spherical primitives. The encoder in SphereNet[3] employs DGCNN to extract global features from the surface point cloud. These features are passed through a decoder that predicts sphere parameters optimized to minimize the SDF error, smoothness loss, and L2 regularization. Smoothness loss ensures spatial consistency between adjacent primitives:

$$\mathcal{L}_{\text{smooth}} = \frac{1}{N-1}\sum_{i=1}^{N-1}\left(|\mathbf{c_i} - \mathbf{c_{i-1}}| + \left|r_{s,i} - r_{s,i-1}\right|\right)$$

where ci and rs,i represent the center and radius of the i-th sphere. L2 regularization is applied to discourage large parameter values and enhance compactness:

$$\mathcal{L}\text{reg} = \frac{1}{N}\sum_{i=1}^{N}\left(|\mathbf{c_i}|^2 + r_{s,i}^2\right)$$

The total loss combines SDF optimization, smoothness loss, and L2 regularization:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{SDF}} + \lambda_{\text{smooth}}\mathcal{L}_{\text{smooth}} + \lambda_{\text{reg}}\,\mathcal{L}\text{reg}$$

where λsmooth and λreg are weights for the respective terms.

The implementation relies on key libraries such as Trimesh for mesh processing, Scikit-learn for clustering, PyTorch[5] for neural network operations, and NumPy[4] for efficient numerical computations. Technical challenges included resolving overlapping cluster boundaries, mitigating meshing artifacts such as holes, and ensuring smooth alignment between cylindrical and spherical primitives. SphereNet[3] was adapted from its original use in CMPT464/764 Assignment 1 to serve as a neural network-based refinement stage. Its integration highlights the method's extensibility and demonstrates its effectiveness in producing high-fidelity reconstructions through a combination of clustering-based primitives and deep learning techniques.

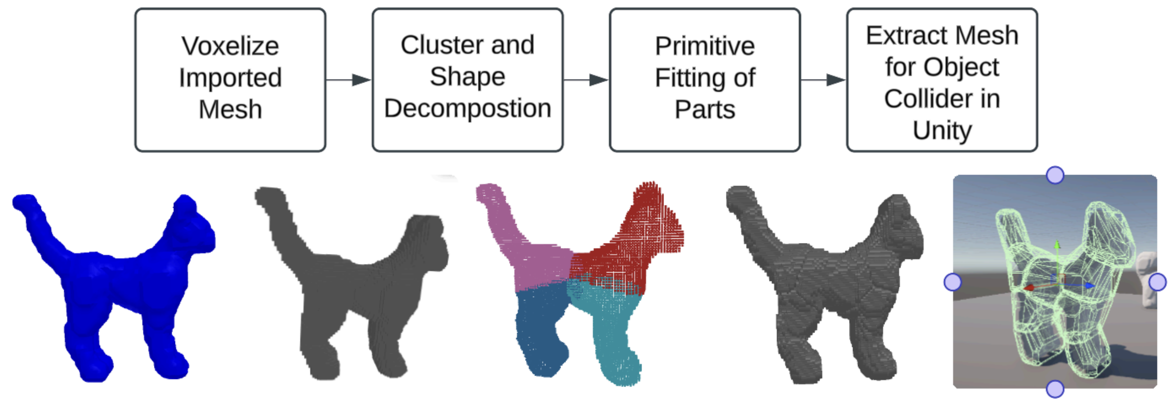
Voxelize Imported Mesh
Cluster and Shape Decompostion
Primitive Fitting of Parts
Extract Mesh for Object Collider in Unity

Figure 1: The pipeline converts 3D mesh to voxel grid then k means is used to cluster and segment into part decomposition before JJSweep abstracts each part individually. The resulting representations are merged into a final 3D mesh with practical uses in environments such as Unity.

Upon completing the reconstruction process, the final object is exported to Unity for further analysis, visualization, and interaction. This step is crucial for real-world applications, as Unity's robust physics engine and rendering capabilities allow us to test the reconstructed models in practical scenarios. The reconstructed mesh, exported as a .obj file, retains all geometric details, ensuring compatibility with Unity's 3D environment while preserving the structural fidelity of the original model.

Experiments

Our experimental evaluation focuses on testing the efficacy of the pipeline on a dataset of 3D models, comparing performance against a baseline, and analyzing the results through various metrics and visualizations.

The dataset comprises five 3D models which include a dog, hand, pot, rod, and sofa. These models were provided as 3D meshes in the .obj file format, which is supported by the Trimesh library[1]. While point cloud and voxel representations were also available, our pipeline exclusively used the mesh inputs. These models are not publicly available and were specifically provided for this project. The processing steps involved importing the meshes, voxelizing the models with user-defined grid sizes, clustering the voxel points, and reconstructing the geometry using sweeping and mesh generation techniques. These steps adhered to the pipeline design detailed earlier and ensured a consistent preprocessing methodology across all experiments.

To evaluate the performance of our method, we employed four metrics: Chamfer Distance, Hausdorff Distance, and Mean Squared Error (MSE) and the time to generate the model. Chamfer Distance was selected to measure the average squared distance between sampled points from the original and reconstructed models. This metric provides a reliable indication of the overall alignment between the two sets of points[6]. Hausdorff Distance, on the other hand, quantifies the maximum distance between the closest points in both sets, thereby capturing outlier behavior and the worst-case alignment. Mean Squared Error was chosen to evaluate voxel alignment accuracy by comparing binary voxel occupancy between the original and reconstructed models. These metrics collectively reveal the fidelity of our reconstructions. The choice of these metrics reflects their widespread use in shape approximation and reconstruction tasks. Implementing these metrics was challenging in some cases, particularly in ensuring consistent sampling of points and voxel alignment between the original and reconstructed models. Chamfer and Hausdorff distances were computed using point sampling techniques, while voxel MSE required precise alignment of voxel grids.

For the baseline comparison, we employed SphereNet, a method adapted from CMPT464/764 Assignment 1. SphereNet[3] uses neural network-based spherical primitives for shape representation and optimization. It reconstructs models by minimizing Signed Distance Field (SDF) errors, alongside smoothness and L2 regularization losses. The baseline provides an established method for spherical primitive-based reconstruction, which we compare against our cylindrical sweeping approach. The implementation details of SphereNet include training using PyTorch[5] and employing a DGCNN-based encoder to extract features from the voxelized models[3]. While SphereNet[3] effectively captures global shape features, it faces limitations in reconstructing elongated structures and complex topologies, areas where our pipeline excels.

To further analyze our pipeline, we conducted ablation studies. Although we retained K-means for clustering (See Figure 2), we varied the number of clusters and voxel grid sizes to observe the impact on the reconstruction[2]. Increasing the cluster count resulted in finer segmentation but also increased processing time. Smaller grid sizes captured finer details but came at the cost of higher computational overhead. We also tested the pipeline without creating double-sided meshes. This revealed a significant drawback, as the reconstructed models appeared see-through when viewed from certain angles due to missing normals. The double-sided meshes resolved this issue, ensuring correct visualization and surface rendering.

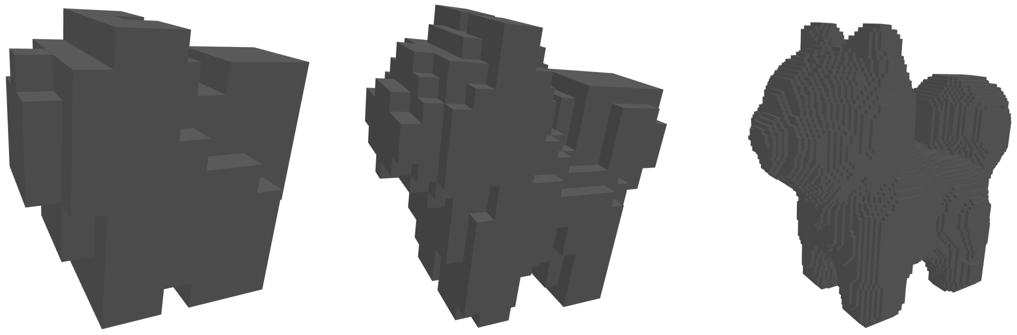

Figure 2: Results of different K and Grid values

Quantitative results demonstrate the advantages of our method over the baseline. Across all models, Chamfer and Hausdorff distances showed significant improvement, with reductions of up to 93% and 79%, respectively. The hand, pot, and sofa models exhibited the most substantial gains in Chamfer Distance, while the rod and dog models showed consistent improvements in all metrics. MSE results were more variable, with a modest 7% improvement for the pot model and a slight decline for the rod model. These variations can be attributed to the inherent geometric complexities of the models. Time measurements indicate that our method is computationally efficient, achieving an average reduction in processing time of over 50% compared to SphereNet[3].

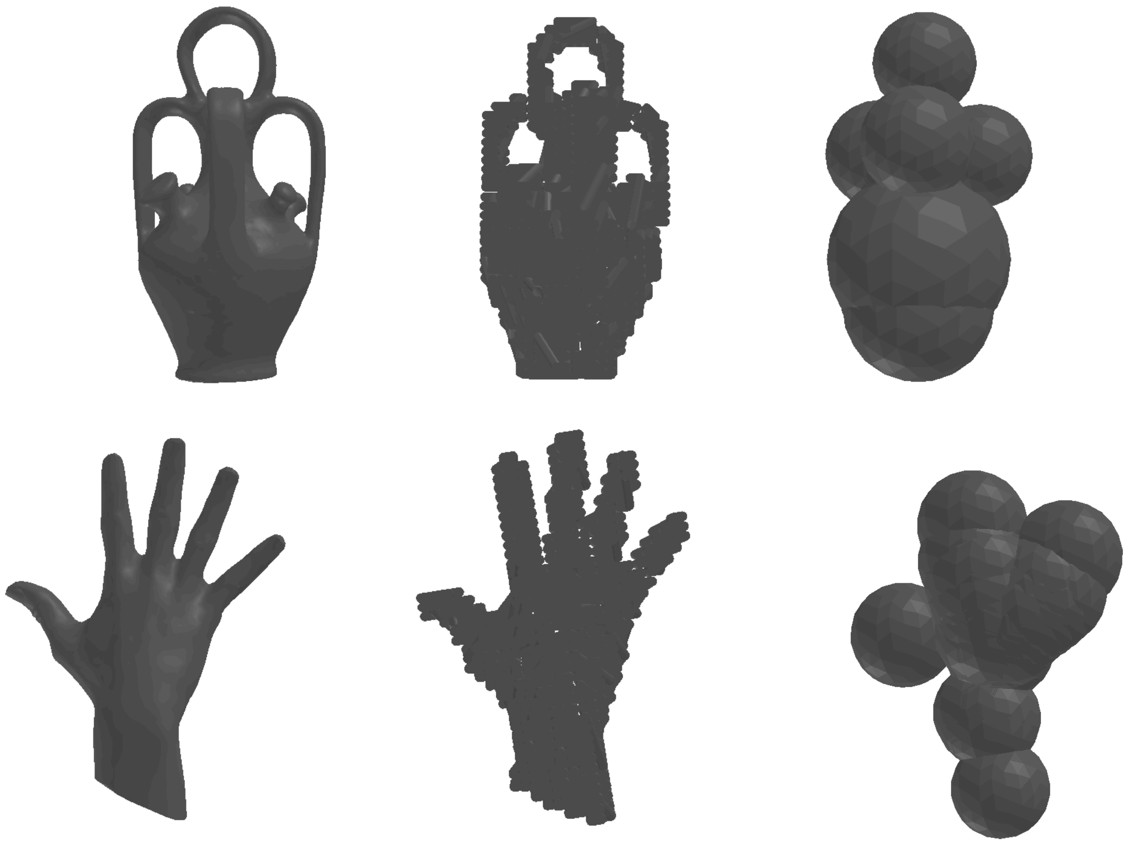

Figure 3: Original, Reconstructed, and Baseline rendering of the Pot and Hand

Qualitative visualizations further highlight the strengths and limitations of our pipeline. Images of the baseline, reconstructed, and original meshes illustrate the differences in fidelity and alignment. The dog, sofa, and rod models exemplify successful reconstructions, with smooth surfaces and minimal deviation from the original shapes (See Figure 4). Conversely, the pot and hand models revealed challenges in capturing intricate details and handling overlapping clusters (See Figure 3). These visualizations provide valuable insights into the pipeline's performance and areas for improvement.

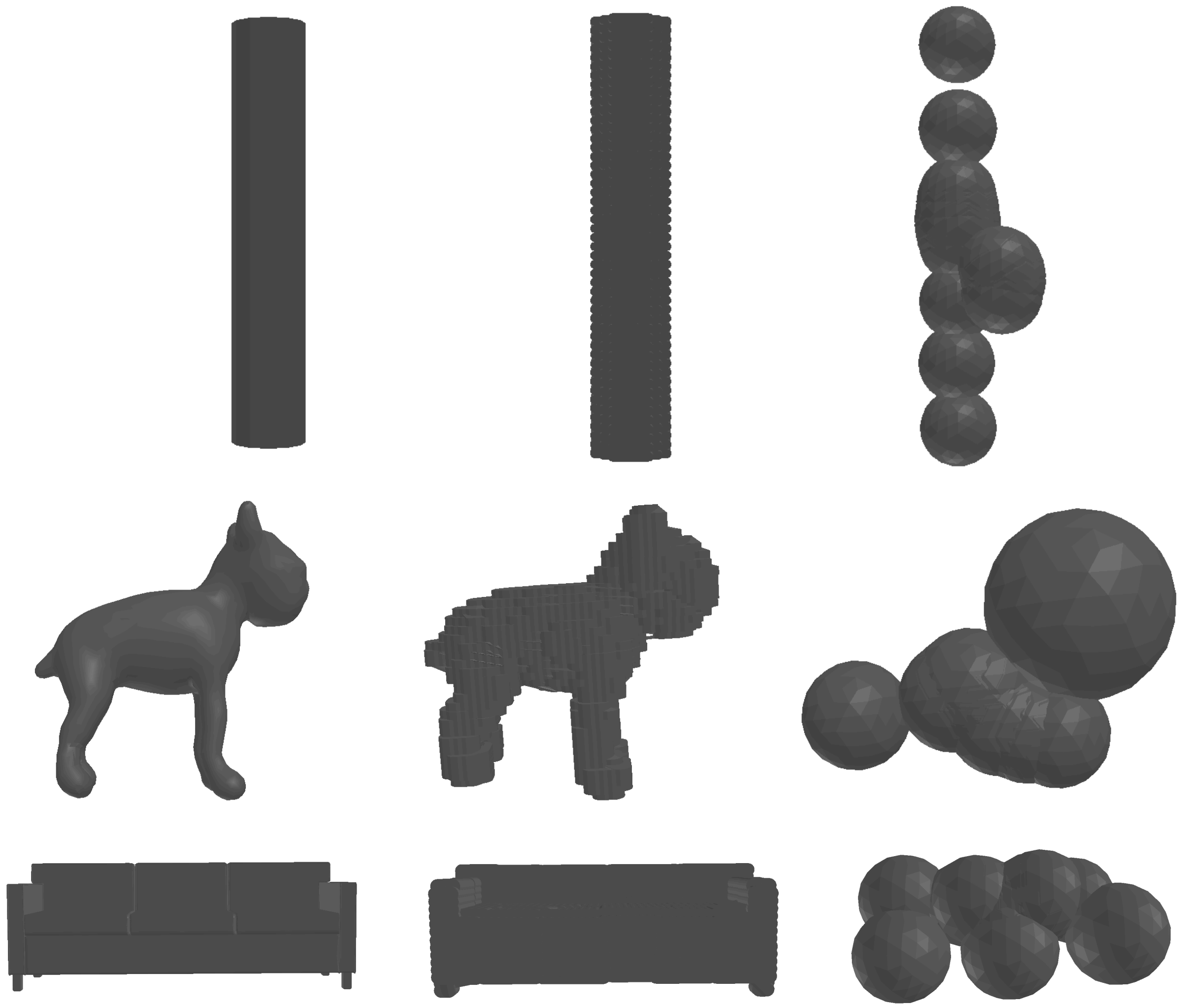

Figure 4: Original, Reconstructed, and Baseline rendering of the Rod, Dog, and Sofa

Future improvements include integrating an advanced axis-fitting algorithm to optimize cross-section alignment and exploring alternative clustering methods such as hierarchical or density-based clustering for better handling of complex geometries. Additionally, leveraging multi-scale voxelization could enhance detail capture without compromising computational efficiency. These enhancements, combined with the robust foundation of our pipeline, hold promise for advancing the state-of-the-art in 3D shape reconstruction.

Unity Collision Benchmarks

As an add on to our project we also tested the performance of the cluster generation as colliders for Unity to use, as the original paper we referenced aimed to solve collider generation efficiency and accuracy. For these clusters to function in Unity, we would need to convert each cluster to a convex hull to enable dynamic collisions (i.e. moving rigidbodies). When generated in Unity we get the following results.

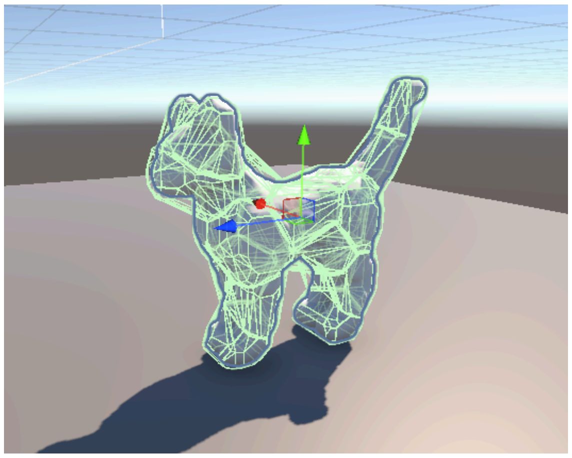
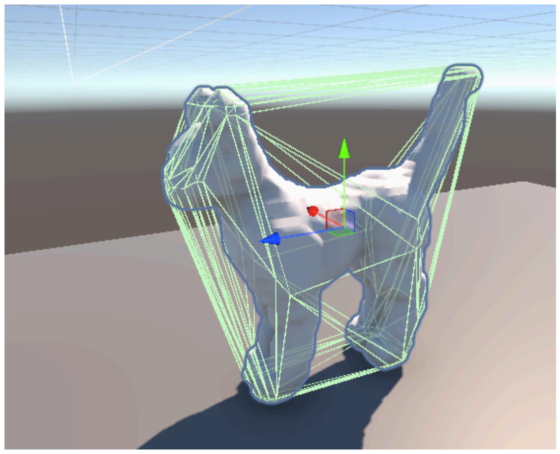
Figure 5: The cat_model.obj with k = 20 clusters (left), generated using our algorithm and the cat_model.obj using Unity's generated convex hull on the original mesh (right)

When segmenting the mesh into sweeps, we can get a composition of convex hulls that closely represents the shape of a given mesh, while still preserving convex properties for efficient collision calculation. The spaces between the cat's legs and the valley between the cat's head and tail are well represented in the composition of convex hulls.

Using Unity's generated convex hull for the original mesh we do get an efficient collider but one that does not best represent the mesh's shape. When it comes to concavities on a mesh they are not well represented when generating one convex hull for an entire mesh, unlike the generated convex hulls per sweep in our algorithm.

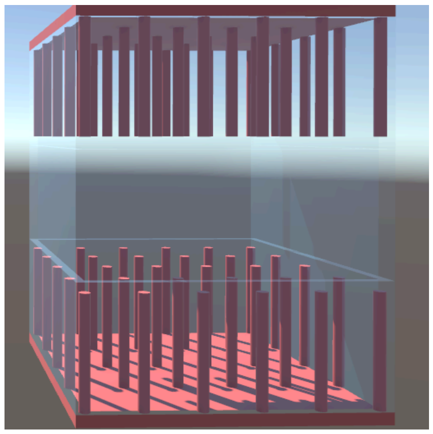
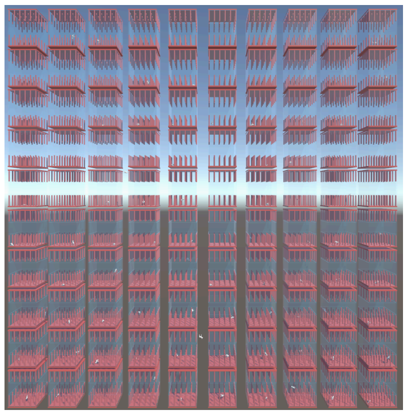
Figure 6: The encased environment (left) we used to test the collider performance, and a testing environment (right) to simulate 100 collisions benchmarks at once.

For benchmarking we measured the performance of the colliders by testing on 100 instances colliding within closed environments, for both Unity's generated convex collider and our composited convex hulls. We opted for cylindrical pipes on the floor and top to best handle collision directly onto holes and concavities of a collider. Testing the data within our simulated environment we get the following results:

100 Instances	Dog	Hand	Pot	Rod	Sofa	Cat	Shiba	Starfuis
Convex	60fps	60fps	60fps	60fps	60fps	34pfs	25fps	27fps
JJSweep	60fps	60fps	60fps	60fps	60fps	60fps	60fps	60fps
K Cluster	20	30	45	1	25	15	25	10

Table 1: Benchmark results on a machine with an RTX 4070 SUPER, intel i5-13600KF, on a 72 Hz locker monitor. Each object here is given a defined k to provide the best visual representation per mesh.

The data for Dog, Hand, Pot, Rod, and Sofa all yield similar performances for both Unity's single convex hull and our composited convex hull generation, however the data for Cat, Shiba, and Starfish. We see that for those tests, the composited convex hull generation doubles the performance of Unity's single convex hull generation.

There are several reasons as to why we believe this is occurring on the data that we have for those specific datasets:

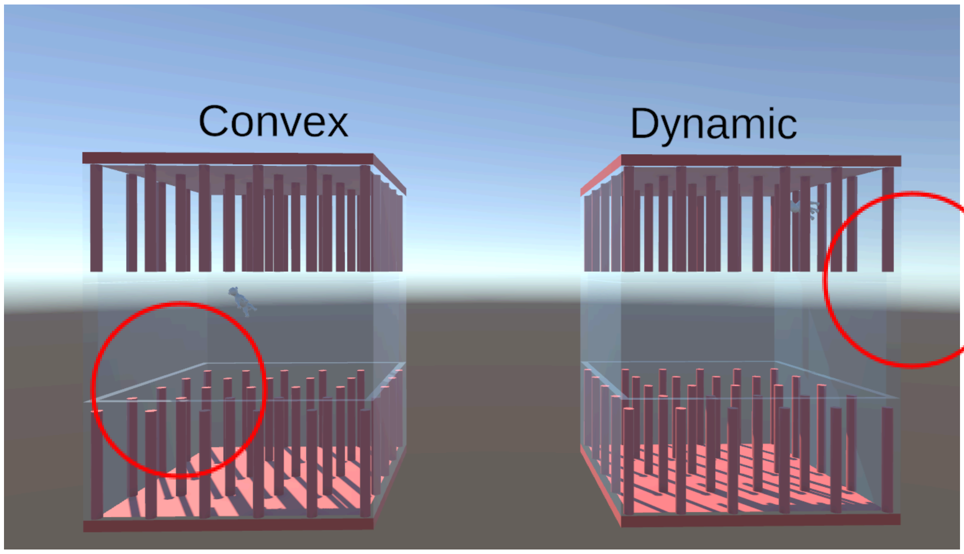

Figure 7: The dog object tested in the environment, as highlighted in the red circle

The meshes, when generated into Unity were small enough to fit through the spaces. Therefore the concavities and holes were not as thoroughly tested given that the meshes would fit right in between the cylinders. The Dog, Hand, Pot, Rod, and Sofa were meshes that could fit right in between the cylinders, unlike the Cat, Shiba, and Starfish.

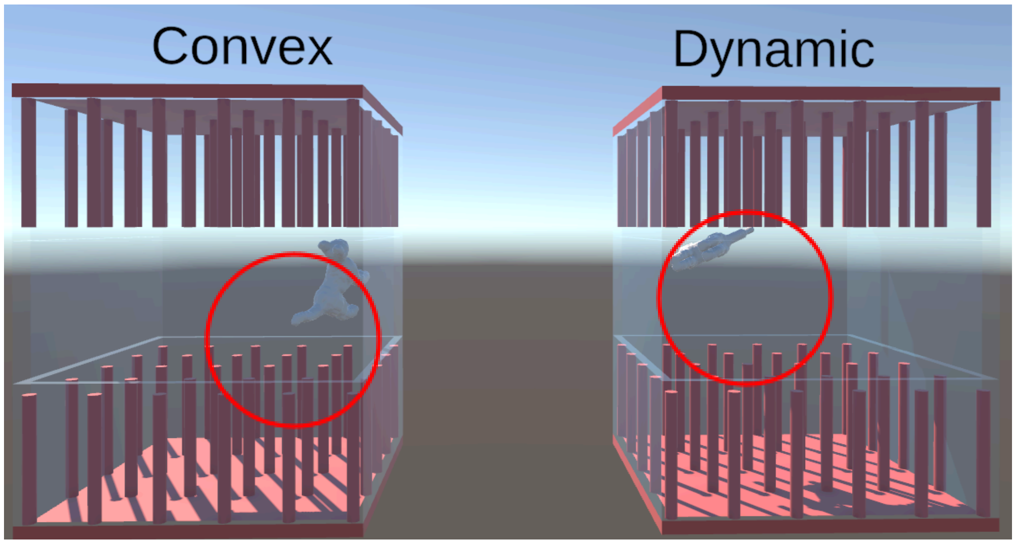

Figure 8: The cat object tested in the environment, as highlighted in the red circle. The cats are large enough to not fit through the spaces of the cylinders, so concavities can be tested.

Given that the Cat, Shiba, and Starfish models are large enough to test concavity collision detection in the test environment, and the convex hull generated by Unity does not represent concavities well, unnecessary concave collision detection is bound to happen.

When a cylinder comes in contact in a concave region, Unity's convex hull will consider that as a collision, when the mesh itself has that space empty. Referring to the Cat model earlier, the space between the legs and the space between the back of the head and tail are considered as collision areas for the object, when the geometry of the Cat model says otherwise.

This is where our algorithm excels, as it labels those concavities as non-collision areas, so collision calculations on those regions are not needed.

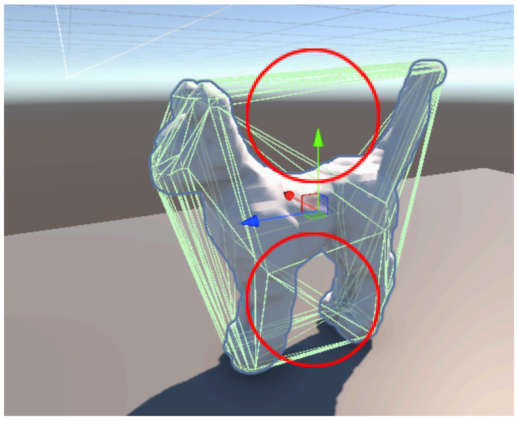

Figure 9: The cat object's convex hull generated by Unity, with the concavities highlighted in red

Discussion

Compared to the baseline method derived from SphereNet[3], which uses spherical primitives for shape reconstruction, our cylindrical primitive-based approach demonstrated superior accuracy and efficiency across all tested models, showcasing the adaptability and precision of our implementation.

Chamfer Distance was a critical measure of the average squared distance between points sampled from the original and reconstructed geometries, showed consistent improvement across all tested models. For example, the Hand model achieved a 93% reduction in Chamfer Distance, indicating a significant enhancement in the alignment of reconstructed points with the original geometry. Similarly, the Pot, Rod, Sofa, and Dog models demonstrated improvements ranging from 79% to 91% (See Table 2), underscoring the effectiveness of the cylindrical sweeping approach for complex shapes. These results highlight the robustness of our pipeline in accurately capturing geometric details. The use of Trimesh[1]. for operations like voxelization and mesh generation played a key role in achieving these high levels of geometric fidelity.

	Hand	Pot	Rod	Sofa	Dog
Improved	0.000526	0.000761	0.001205	0.001317	0.001014
Baseline	0.007227	0.005787	0.005831	0.014253	0.007122
% Increase	93%	87%	79%	91%	86%

Table 2

Hausdorff Distance, which measures the maximum deviation between the original and reconstructed models, also showed substantial improvements, with reductions ranging from 57% to 79% across the tested objects (See Table 3). This metric provides insight into the precision of the reconstructed surfaces and highlights the reduction of outlier deviations achieved by our method. For instance, the Dog model achieved a 65% improvement, demonstrating our pipeline's ability to minimize geometric inconsistencies effectively. The reductions in both Chamfer and Hausdorff Distance collectively reflect the superiority of our approach in reconstructing surfaces that closely resemble the original models.

	Hand	Pot	Rod	Sofa	Dog
Improved	0.038507	0.047865	0.079801	0.073622	0.069249
Baseline	0.180848	0.156359	0.183944	0.236088	0.199468
% Increase	79%	69%	57%	69%	65%

Table 3

The Mean Squared Error (MSE), which compares voxel occupancy between the original and reconstructed models, exhibited varied results. While significant improvements were observed for most models, such as a 60% reduction for the Hand model and 37% for the Sofa model, a slight degradation was noted for the Rod model (-2%) (See Table 4). This result is likely attributed to the uniform and simple structure of the rod shape, where the baseline SphereNet[3] may have leveraged its neural network's efficiency to approximate the geometry effectively. Future refinements, such as adaptive resolutions for simpler shapes, may address this limitation. Overall, the MSE results demonstrate the pipeline's capability to provide robust volumetric accuracy for most geometries.

	Hand	Pot	Rod	Sofa	Dog
Improved	0.083387	0.219846	0.180952	0.357561	0.227654
Baseline	0.210407	0.235294	0.177857	0.569173	0.286111
% Increase	60%	7%	-2%	37%	20%

Table 4

In terms of computation time, our method showcased a dramatic reduction, with improvements ranging from 49% to 63%. For instance, the Hand model required only 2.14 seconds compared to 4.78 seconds for the baseline (See Table 5). This improvement is attributable to the optimized voxelization and clustering processes combined with the efficient sweeping mechanism. The reduced computation time highlights the pipeline's practical applicability for real-time and large-scale 3D modeling tasks. The implementation of clustering using Scikit-learn[2], combined with NumPy[4] for efficient numerical computations, streamlined this process and contributed to the overall time efficiency.

	Hand	Pot	Rod	Sofa	Dog
Improved	2.14320588	2.76999092	2.35037756	4.32928324	2.4977212
Baseline	4.78311491	6.76881623	4.6201489	11.3507924	6.7606132
% Decrease	55%	59%	49%	62%	63%

Table 5

Despite these achievements, certain limitations persist. The current implementation assumes a uniform circular cross-section for sweeping, which may not capture irregular or highly concave geometries effectively. Additionally, overlapping clusters can lead to intersecting or redundant meshes, requiring post-processing steps to mitigate artifacts. While the cylindrical primitives performed well for most shapes, they may struggle with geometries exhibiting highly anisotropic features. As well the fixed resolution of the circular cross-sections and sweeping path may lead to a loss of detail in regions demanding finer reconstruction. These limitations suggest potential areas for improvement to enhance the pipeline's robustness and generalizability.

Future improvements can address these limitations through various strategies. One possible direction is the incorporation of a more advanced axis-fitting algorithms, such as Principal Component Analysis (PCA) with RANSAC for outlier removal. This approach could refine the sweeping trajectory, particularly for elongated or irregular shapes, improving alignment fidelity. Additionally, implementing dynamic cross-section adaptation based on local geometry can enable the use of non-circular shapes or variable radii, ensuring better conformance to the original model.

Multi-primitive integration represents another promising enhancement. By combining cylindrical primitives with other shapes, such as superquadrics or ellipsoids, the pipeline could better handle a wider variety of geometries, from organic to mechanical forms. Finally, post-processing techniques like mesh repair and Boolean operations could resolve overlapping cluster boundaries, ensuring seamless transitions between reconstructed segments.

References

1. Trimesh is utilized for voxelization, mesh generation, and transformations. Available: https://trimsh.org
2. Scikit-learn’s K-means implementation is leveraged for clustering. Available: https://scikit-learn.org
3. SphereNet (CMPT464/764 Assignment 1), Neural network-based spherical primitive fitting for 3D models, focusing on Signed Distance Field optimization. Available via assignment handouts.
4. Numerical computations and array manipulations. Available: https://numpy.org
5. SphereNet’s neural network training and optimization use PyTorch. Available: https://pytorch.org
6. Inspiration for Signed Distance Field representations. Available: https://arxiv.org/pdf/2004.02869
7. C. Snyder, A. Gungormusler, A. Allen, A. Ayvazov, and D. Yordanov, "Approximating 3D poly mesh virtual objects with primitive shape colliders," *Technical Disclosure Commons*, Nov. 1, 2019. [Online]. Available: https://www.tdcommons.org/dpubs_series/2633.
8. M. Zhao, Y. Wang, F. Yu, C. Zou, and A. Mahdavi-Amiri, "SweepNet: Unsupervised Learning Shape Abstraction via Neural Sweepers," *European Conference on Computer Vision*, 2024. https://mingrui-zhao.github.io/SweepNet/