Shape Deformation through 2D guidance and Adversarial Training

Andrea Gervasio[1]

Abstract
Rendering is the process of generating a 2D image from 3D models using a computer program which simulates the physical process happening inside a camera. By inverting this process, one could reconstruct the 3D information of the scene represented in a 2D image.

In recent years, there has been a great progress in this field, thanks to the development of differentiable renderers which have been able to substitute the non-differentiable rasterization process with gradient-based methods.

My work expands on that, using the differentiable renderer as a generator in a GAN architecture. The discriminator is a deep neural network with residual connections and a self-attention mechanism.

Experiments show that my model achieves better results than the ones produced by using the differentiable renderer on its own.

Keywords
differentiable rendering, shape deformation, 2D guidance, adversarial training

1. HIGHLIGHTS

1. Create a dataset of multiple views using different objects;
2. Implement a differentiable renderer for shape deformation;
3. Use the renderer as a generator in a GAN architecture;
4. Implement a discriminator using residual connections and self-attention;
5. Obtain more accurate results faster than using the renderer on its own.

2. Introduction

Reconstructing 3D objects and scenes from 2D images enables the creation of digital representations of real-world objects, which can be manipulated and analyzed for various purposes. 3D shape reconstruction has always been one of the main goals in computer vision because it has practical applications in a wide range of fields, such as virtual and augmented reality, computer animation and medical imaging. In virtual and augmented reality, accurate 3D models are essential for seamlessly integrating virtual objects into the real-world setting, which will enhance the user's immersive experience and their interaction with both the virtual and the real environment. In computer animation, it helps create more realistic characters and environment because high-fidelity models will create a more immersive experience for the spectator. In the medical field, the 3D reconstruction of anatomical structures from imaging data such as CT scans or MRI can help in the diagnosis and surgical planning, enabling more precise and less invasive procedures and improving patient outcomes. Given the importance of these applications and the growing demand for high-quality 3D content, the need for robust and versatile techniques to obtain accurate deformations becomes increasingly important. Advances in this field could lead to new applications and improvements in existing technologies, driving innovation and enhancing user experiences across multiple domains.

Despite the advancements made in recent years, 3D shape reconstruction still faces several challenges. Methods like point cloud processing, which involves converting 3D points into surfaces, voxel-based techniques, which use a volumetric grid to represent objects, and mesh-based approaches, which represent object as collections of vertices and edges, struggle with handling complex and diverse shapes because many of the techniques developed are designed for specific object categories, so they lack the ability to generalize to a wider range of geometries. In particular, to produce high-quality surfaces, point cloud methods require dense and accurate point data which is often not available or requires extensive pre-processing. Voxel-based methods are limited by resolution constraints and memory requirements, making it difficult to capture fine details in large or complex objects. Mesh based approaches require precise control over vertex positions and connectivity, which can be challenging to achieve automatically. These, combined with the computational resources required to obtain reasonable results, pose a significant challenge in the development of new models. Moreover, real-world scenario often involve

Woodstock'21: Symposium on the irreproducible science, June 07–11, 2021, Woodstock, NY
gervasio.1883259@studenti.uniroma1.it (A. Gervasio)

dynamic environments, partial occlusions and varying lightning, which traditional methods struggle to handle.

Traditional approaches like the ones mentioned above are still commonly used, but they often require significant manual intervention, and they do not scale well to complex or highly detailed objects. This lack of generalization hinders the development of versatile and robust models that can adapt to various scenarios. For example, a model trained to reconstruct cars may not perform well when tasked with reconstructing human figures or animals. Therefore, there is a pressing need for reconstruction models that can handle diverse objects and maintain high accuracy across different shapes and textures. Such models would be capable of adapting to new objects without requiring extensive retraining or manual intervention, making them more practical for real-life scenarios.

For this reason, researchers have turned to machine learning and deep learning approaches, revolutionizing the field of 3D shape reconstruction. Traditional methods rely heavily on handcrafted features and heuristics, which often limited their flexibility and generalization capabilities. In contrast, deep learning approaches, especially those leveraging convolutional neural networks, have demonstrated remarkable success in capturing complex pattern and representations directly from data. These models have the advantage of learning hierarchical features automatically, which can be particularly beneficial for understanding and reconstructing intricate 3D shapes. CNNs have been employed to improve the accuracy and efficiency of reconstruction processes, exploiting their capabilities to deal with large quantities of data to learn the underlying patterns of object shapes. Additionally, the use of neural networks allows for end-to-end optimization, enabling the integration of various constraints and priors directly into the learning process. All of this has led to the development of more robust and versatile 3D reconstruction methods that can handle a wider range of object categories and environmental conditions, leading to automated and more precise reconstructions.

An emerging trend in 3D shape reconstruction is the integration of multi-view information to enhance the accuracy and realism of the generated outputs. Multi-view reconstruction techniques leverage images captured from different angles to resolve ambiguities and capture details that may be missed in a single view approach, which is particularly advantageous for complex or occluded regions of the object (for example, the back of the object). This allows the model to build a comprehensive representation of the object. Recent advancements in neural networks have made it possible to effectively fuse information from multiple viewpoints. Traditional approaches leverage this high capacity of neural networks to learn intricate spatial relationships to aggregate multi-view data into coherent 3D shapes.

Moreover, the development of differentiable rendering techniques has further improved the fidelity of 3D shape reconstructions. Differentiable rendering allows the gradients of the rendering process to be backpropagated through the network, enabling end-to-end training directly from images observations to 3D outputs. This ensures that generated shapes are not only geometrically accurate, but also visually consistent with the input images.

The synergy between multi-view information and advanced neural network architectures represents a significant step forward in the field of 3D reconstruction. It allows for the creation of detailed and accurate models from relatively sparse data, making it feasible to reconstruct objects in real-world scenario, where ideal conditions are never met.

This paper presents a novel approach to 3D shape reconstruction, using a generative adversarial network on the shape deformation task. The method involves an iterative training process where the model is sequentially trained on different objects, forcing it to adapt to various shapes and textures. Using an adversarial discriminator enhances the reconstructed shapes. Additionally, texture optimization is integrated into the process, resulting in more detailed reconstructions.

The approach presented in this paper aims to overcome the current limitations of 3D shape reconstruction by enhancing the model's generalization capabilities and improving the quality of the reconstructed meshes. The method has shown encouraging results in increasing the quality of the generated shapes by training the model to distinguish between real and synthetic data.

The key contribution of this work are: a novel training methodology that iteratively adapts the model to different objects, improving its generalization abilities across diverse shapes and textures; the integration of adversarial training to enhance the reconstructed shapes; a comprehensive texture optimization process; and an experimental evaluation demonstrating the effectiveness of the proposed method in improving the accuracy and the quality of 3D shape reconstruction.

The paper is organized as follows: section 3 discusses related works, highlighting their strength and limitations, and outlines the context for the approach detailed in this paper; section 4 details the methodology, describing in details the architecture of the model and the training procedure, including the adversarial training process and the texture optimization techniques; section 5 presents the experimental setup, the evaluation metrics and the results; section 6 will conclude the paper with a discussion on the implication of the findings, the challenges found and potential future work.

3. Related works

In recent years, significant advancements have been made in the field of 3D reconstruction and mesh generation from 2D images. Several approaches have been proposed, each contributing with different methodologies and innovations to enhance the accuracy and efficiency of 3D shape reconstruction. We will now see some of the most relevant ones, focusing on their strengths and limitations.

One of the pioneering works in this domain is Soft Rasterizer (SoftRas) by Liu et al. (2019) [1], which introduces SoftRas, a novel differentiable rendering framework that enables end-to-end optimization for 3D deep learning tasks. SoftRas bridges the gap between traditional computer graphics and deep learning by providing a renderer that can backpropagate gradients through the entire pipeline. This capability allows for direct optimization of 3D object properties based on image observations. The key strengths of SoftRas include its ability to produce smooth, differentiable outputs and to handle occlusions and visibility in a probabilistic manner, which enhances its robustness and flexibility in various 3D reasoning tasks. Additionally, SoftRas demonstrates superior performance in applications like 3D shape reconstruction and pose estimation compared to existing methods. However, the model also has limitations, such as potential inefficiencies in handling large-scale scenes and complexities arising from the need for extensive computational resources.

Pixel2Mesh, proposed by Wang et al. (2020) [2], presents a novel approach for generating 3D mesh models directly from single RGB images. The core idea involves progressively deforming an ellipsoid mesh to match the shape depicted in the input image through a series of graph convolutional networks (GCNs). This method leverages image features extracted by a convolutional neural network (CNN) to guide the mesh deformation process, effectively combining the strengths of image-based and mesh-based representations. Strengths of Pixel2Mesh include its ability to generate detailed and high-fidelity 3D shapes with relatively low computational complexity, as well as its end-to-end trainable framework which simplifies the process. However, the model has limitations such as difficulties in capturing fine details and textures, challenges in handling complex topologies, and dependency on the quality of the initial mesh, which may restrict its applicability in more diverse and intricate 3D modeling tasks.

Additionally, the work by Wen et al. (2021) [3] focuses on the reconstruction of deformable objects using a novel encoder-decoder architecture that combines implicit and explicit shape representations. Their work addresses the challenge of reconstructing both the 3D shape and surface color from a single 2D image. The proposed model improves upon the Soft Rasterizer method by incorporating an encoder-decoder structure where features are extracted and fed into separate shape and color generators. A differentiable renderer is used to produce the final 3D model. Notably, the introduction of an attention mechanism (CBAM) in the encoder enhances the detail and accuracy of the reconstruction. Experimental results demonstrate the model's superior performance compared to previous methods, with a significant improvement in intersection-over-union (IOU), structural similarity (SSIM), and mean square error (MSE). However, the complexity of simultaneously reconstructing shape and color poses a significant training challenge. The model's strength lies in its detailed reconstruction capabilities, but its complexity and training difficulty are notable limitations

The work of Nicolet, Jacobson, and Jakob [4] introduces a novel method for generating high-quality 3D meshes from images using a combination of differentiable rendering and preconditioning techniques. The proposed method addresses the instability and high computational cost typically associated with gradient-based optimization in differentiable rendering. By incorporating a sparse Cholesky factorization, the approach achieves more stable and efficient optimization, even with a limited number of viewpoints. This results in high-quality reconstructions that maintain detail and smoothness, surpassing traditional regularization methods. A key strength of the model is its ability to handle noisy gradient estimates effectively, which is crucial for rendering complex scenes. However, the method's dependency on sparse Cholesky factorization can introduce additional computational overhead, and the approach may still struggle with highly ambiguous or non-convex optimization problems.

The 3Deformer paper by Gong et al. (2023) [5] introduces a novel framework designed to address the challenge of image-guided 3D mesh deformation. The 3Deformer framework leverages the power of neural networks to deform 3D meshes based on 2D images, achieving impressive results in preserving fine geometric details and ensuring high fidelity to the input image. The authors highlight the strength of 3Deformer in its ability to generalize across different types of meshes and its effectiveness in handling complex deformations. Additionally, the model demonstrates robustness in maintaining mesh integrity and detail during the deformation process. However, the paper also acknowledges certain limitations, including the computational intensity required for training the neural network and potential difficulties in handling highly intricate or densely packed mesh structures.

The field of 3D shape reconstruction has seen a significant progress in recent years, with various approaches contributing to the enhancement of mesh generation techniques. Despite their advancements, challenges such as handling complex textures, non-rigid deformations, and computational efficiency remain.

4. Implementation

In this study, I employed a series of state-of-the-art techniques for 3D mesh rendering, manipulation, and optimization using the PyTorch3D library [6]. The methodology involves the collection and normalization of 3D mesh data, followed by the application of various rendering techniques to visualize the meshes from multiple perspectives. I used adversarial training to enhance the fidelity of the mesh representations. This section describes the detailed steps taken from data preparation to the adversarial training loop, ending with the optimization of 3D meshes.

4.1. Setup

4.1.1. Data preparation

Due to a still pending request for access to the ShapeNet dataset, the initial step in my process involved acquiring 3D models in the OBJ file format. I sourced a diverse set of models with a free license to use them and they were obtained from an online repository [7].

Each model typically comes with an OBJ file that defines the geometry of the 3D object, an MTL file that specifies material properties, and associated texture files. Upon downloading, these files were organized into corresponding subdirectories under a main data directory. This organization facilitated efficient loading and processing of the meshes using PyTorch3D's IO utilities.

4.1.2. Mesh loading and normalization

The PyTorch3D library provides a convenient method, `load_objs_as_meshes`, which reads the geometry and material properties from the OBJ and MTL files, respectively. This method constructs a `Meshes` object, which is a core data structure in PyTorch3D designed to handle batches of meshes with varying sizes and properties.

For convenience reasons, the meshes were normalized: they were appropriately scaled and centered for further processing. Each mesh is scaled to fit within a unit sphere centered at the origin. This normalization step involves calculating the centroid of the vertices and the maximum distance of any vertex from the centroid. The vertices are then translated to the origin and scaled by the inverse of this maximum distance. This normalization ensures uniformity and simplifies the optimization process.

4.2. Rendering setup

To visualize and deform the meshes, I set up a rendering pipeline using PyTorch3D's rendering utilities. Rendering involves converting the 3D mesh data into 2D images from various viewpoints. This process is facilitated by defining camera settings, light sources, and rasterization settings [8].

4.2.1. Camera and lighting configuration

I defined multiple viewpoints for rendering by setting up a series of cameras. The cameras were placed at different angles around the mesh to capture comprehensive views. The `look_at_view_transform` function in PyTorch3D was utilized to generate the rotation and translation matrices for the cameras. These matrices were then used to create instances of `FoVPerspectiveCameras`, which are perspective cameras with a field of view (FoV) projection.

Lighting is another crucial aspect of rendering. I used a `PointLights` object to define the position and intensity of the light source. In my setup, the light source was positioned slightly away from the mesh to cast realistic shadows and highlights, enhancing the visual quality of the rendered images.

4.2.2. Rasterization settings

Rasterization is the process of converting 3D mesh data into a 2D image. I defined rasterization settings using the `RasterizationSettings` object. Key parameters included the image size, the blur radius for soft rasterization, and the number of faces per pixel. These settings were adjusted to balance between rendering quality and computational efficiency.

I also set up a silhouette renderer to generate binary images that represent the outlines of the meshes. This was achieved by configuring a separate set of rasterization settings optimized for silhouette rendering.

4.2.3. Mesh rendering

With the camera, lighting, and rasterization settings in place, I used the `MeshRenderer` to render the meshes. The renderer is composed of a rasterizer and a shader. The rasterizer converts the 3D mesh data into fragments, while the shader determines the final color of each pixel based on the material properties and lighting conditions.

For standard rendering, I used the `SoftPhongShader`, which implements the Phong reflection model to compute pixel colors based on ambient, diffuse, and specular reflections. For silhouette rendering, I used the `SoftSilhouetteShader`, which produces binary images highlighting the contours of the meshes.

4.3. Dataset creation

To facilitate the training and evaluation of my models, I generated a dataset of rendered images and silhouettes from multiple viewpoints. Each mesh was rendered from a series of predefined angles, resulting in a comprehensive set of images that captured the mesh from different perspectives. These images were stored in lists, with each list corresponding to a particular mesh.

The dataset includes both RGB images and silhouettes, and I used this set of data for training adversarial models and optimizing mesh representations. The use of multiple viewpoints ensures that the models learn to generate realistic and consistent 3D representations.

4.4. Generator loss functions

Loss functions play a pivotal role in guiding the optimization process. I defined several loss functions to capture different aspects of the mesh quality:

1. **RGB Loss**: Measures the pixel-wise difference between the rendered RGB images and the target images, as the squared L2 distance between the two;
2. **Silhouette Loss**: Measures the pixel-wise difference between the rendered silhouettes and the target silhouettes, as the squared L2 distance between the two;
3. **Edge Loss**: Encourages the preservation of the mesh edges during deformation, promoting smooth and realistic surface geometry;
4. **Normal Consistency Loss**: Ensures that the normals of adjacent faces are consistent, leading to smoother surfaces;
5. **Laplacian Smoothing Loss**: Regularizes the mesh by penalizing large deviations from the original vertex positions, promoting smooth deformations.

These loss functions were combined with different weights to form a composite loss to guide the optimization process.

4.5. The discriminator

To enhance the fidelity of my mesh representations, I employed adversarial training using a discriminator. The discriminator is designed to distinguish between real and generated images, providing feedback to the generator to improve the realism of the generated meshes.

4.5.1. Dense blocks

Dense blocks are a key component of the discriminator, featuring densely connected convolutional layers. Each layer receives input from all preceding layers, promoting feature reuse and improving gradient flow through the network. This dense connectivity allows the discriminator to learn more robust and detailed representations of the images, enhancing its ability to distinguish between real and generated data.

4.5.2. Self-attention layers

To capture long-range dependencies and interactions within the images, the discriminator incorporates self-attention layers. These layers compute attention scores for each pixel relative to all other pixels, enabling the network to focus on critical regions and features. The self-attention mechanism enhances the discriminator's capability to detect subtle differences and inconsistencies in the generated images.

4.5.3. Discriminator architecture

The discriminator is a convolutional neural network featuring a combination of dense blocks and self-attention mechanisms to effectively capture fine-grained details and spatial dependencies in the input images (Figure 1). It begins with an initial convolutional layer comprising 64 filters with a kernel size of 4, stride of 2, and padding of 1, followed by a LeakyReLU activation function. This is followed by two dense blocks, each consisting of 4 layers, and two self-attention layers, alternating between them. Each layer within a dense block includes a convolutional layer with spectral normalization, a kernel size of 3, padding of 1, batch normalization, and a ReLU activation function. The first dense block expands the feature maps from 64 to 192 channels by adding 32 channels per layer (growth rate of 32). After the first dense block, a self-attention mechanism is applied with a query, key, and value dimension of 8 channels. The output of the self-attention mechanism is scaled and added to the input feature maps. The second dense block further expands the feature maps from 192 to 320 channels, again adding 32 channels per layer. Another self-attention mechanism is then applied, this time with a query, key, and value dimension of 24 channels. The architecture concludes with a final convolutional layer having a single filter, a kernel size of 4, and no padding, which reduces the spatial dimensions to a single scalar output through a sigmoid activation function.

4.6. Training loop

The training loop iterates over a predefined number of iterations, alternating between generator and discriminator updates. In each iteration, a subset of viewpoints is randomly selected to compute the losses, promoting robustness to different perspectives. The losses are aggregated and used to update the model parameters.

In order to monitor the training process, I generate visualizations and loss plots periodically. The visualizations are the rendered images of the generated meshes,

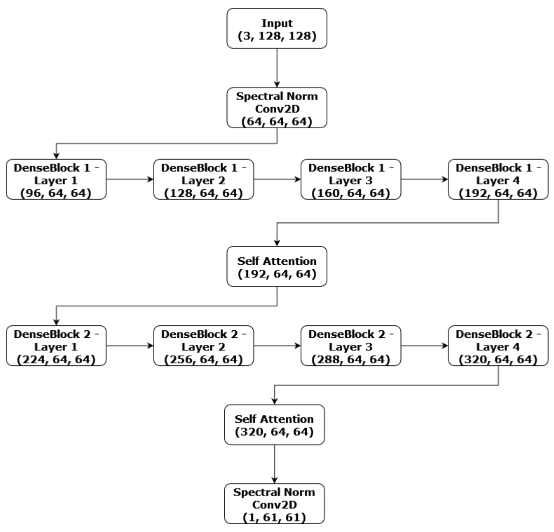

Figure 1: The architecture of the discriminator.

compared to the target images. The loss plots track the evolution of the different loss components.

4.6.1. Adversarial training loop

The training process involves alternating between optimizing the generator and the discriminator. The generator aims to deform the mesh and update the vertex colors to produce realistic images, while the discriminator learns to differentiate between real and generated images.

4.6.2. Generator optimization

The generator is initialized with a sphere mesh, which is progressively deformed to match the target images. Vertex colors are also optimized to match the target textures. The optimization is performed using stochastic gradient descent (SGD) with momentum, which accelerates convergence by considering the gradient history.

4.6.3. Discriminator optimization

The discriminator is optimized using the Adam optimizer, which adapts the learning rate based on the gradient statistics. During each iteration, the discriminator is presented with a batch of real and generated images. The loss is computed as the sum of the binary cross-entropy losses for real and fake images.

4.7. Implementation details

The entire process was implemented in Python using the PyTorch and PyTorch3D libraries. The code is modular, with separate functions for loading data, normalizing meshes, setting up the renderer, defining loss functions, and running the training loop.

The training was performed on Google Colab, with a GPU runtime. This accelerates the computation, especially for the rendering and optimization steps, which involve large matrix operations.

4.8. Summary

The proposed methodology effectively optimizes 3D meshes to produce realistic and high-fidelity representations. The combination of differentiable rendering and adversarial training allows for fine-grained control over the mesh deformations and textures. The use of multiple loss functions ensures that various aspects of the mesh quality are preserved, resulting in smooth and visually coherent surfaces.

The discriminator enhances the realism of the generated images by providing adversarial feedback to the generator. This feedback loop drives the generator to produce meshes that are indistinguishable from real ones, as perceived by the discriminator.

Periodic visualizations and loss plots demonstrate the effectiveness of the training process. The generated meshes closely match the target images, and the losses converge to stable values, indicating successful optimization.

5. Results

In this section, I will present the experimental setup of my work, with the evaluation metrics and the results obtained by my model.

5.1. Experimental Setup

Because of the restricted access of the ShapeNet dataset, I manually created a dataset for the experiments by downloading free-to-use models from online repositories [7]. These models were preprocessed to fit inside a unit sphere to ensure uniformity in scale and orientation.

The code was tested on Google Colab, using a GPU runtime. The PyTorch3D library is automatically downloaded by the code.

The model was trained with two different numbers of iterations, 2000 and 10000. This section will present only the results obtained by the former. See Appendix A for more detailed results.

The optimization of the generator was performed using SGD with learning rate 1.0 and momentum 0.9, while

Table 1
Comparison of the losses of the PyTorch3D renderer and my model when trained for 2000 iterations.

Model	Cow	Humanoid	Fish	Sword	Hand
`PyTorch3D`	0.025049	0.014272	0.010222	0.010509	0.023533
`Mine`	0.025008	0.016262	0.012460	0.009601	0.018882

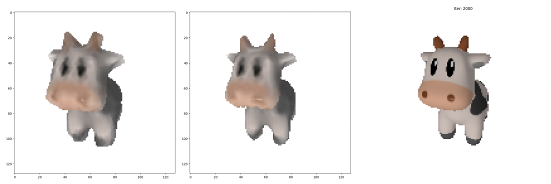

Figure 2: Comparison between the model produced by the PyTorch3D renderer (left) and my model (center) after 2000 iterations. The images on the right are the references used to train.

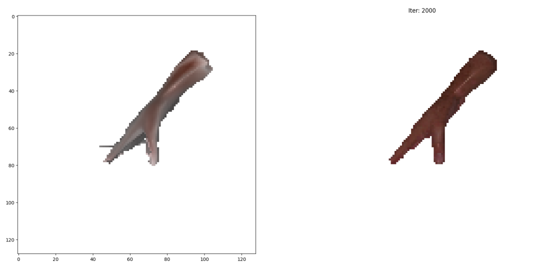

Figure 3: The shape deformed by the PyTorch3D renderer using the hand as reference.

the discriminator was optimized using the Adam optimizer, setting the learning rate to 0.0002, β_1 and β_2 to 0.5 and 0.999 respectively.

5.2. Evaluation metrics

In this work, both quantitative and qualitative metrics were used to evaluate the model.

The quantitative ones are the losses defined in section 4.4 for the generator and the binary cross-entropy loss for the discriminator. The total generator loss is defined as a weighted sum of the single components, with all the weights being 1 except for the normal consistency loss, which weight is 0.01.

Other quantitative metrics to evaluate the model are the Chamfer distance, which computes the similarity between two sets of points, and the intersection over union (IoU), which measures the overlap between the generated mesh and the target one.

The qualitative analysis has been done by comparing visually the deformed meshes produced by the generator and the ones made using the PyTorch3D renderer on its own.

5.3. Experimental results

Now, we will see the results of the work implemented.

The comparison between the losses is shown in Table 1. My model reaches comparable values of the loss in all five objects compared to the PyTorch3D renderer. Despite this, Figure 2 shows that the results obtained by my model are visually more similar to the targets.

Table 2 shows a comparison between the Chamfer distances computed by the PyTorch3D renderer and my model. The results are again similar, except for the hand object, where my model reaches a significantly lower value of the distance. This is probably because the PyTorch3D renderer struggles to deform the mesh in the fingers' area, as shown in Figure 3.

The IoU values are reported in Table 3. This is the metric that shows more variance. My model reaches a lower IoU value only in the fish object, while it shows considerably higher values in the humanoid, sword and hand objects. The sword is also the one where both models have the most difficulty, most likely because it has a lot of little grooves which are not renderer well at the resolution used in this work.

Figure 2 shows a comparison between the mesh deformed by the PyTorch 3D renderer and the one deformed by my model after 2000 iterations. It can be seen clearly how my model produces a shape more similar to the target one, in particular on the ears, the horns and the back of the cow.

6. Conclusion

This study demonstrated the effectiveness of integrating a Generative Adversarial Network with a 3D renderer for the mesh deformation task. By employing an adversarial training approach, the developed model outperformed the PyTorch3D renderer across several key metrics, such as Chamfer distance, and Intersection over Union.

The experimental results showed that the proposed model achieved comparable loss and Chamfer distance values in all tested objects after both 2000 and 10000 iterations. Moreover, the model exhibited significant

Table 2
Comparison of the Chamfer distance computed by the PyTorch3D renderer and my model when trained for 2000 iterations.

Model	Cow	Humanoid	Fish	Sword	Hand
`PyTorch3D`	0.002217	0.000951	0.000457	0.000275	0.001569
`Mine`	0.002212	0.000913	0.000460	0.000275	0.001021

Table 3
Comparison of the intersection over union computed by the PyTorch3D renderer and my model when trained for 2000 iterations.

Model	Cow	Humanoid	Fish	Sword	Hand
`PyTorch3D`	0.971966	0.865181	0.956609	0.618411	0.848967
`Mine`	0.976176	0.918443	0.929015	0.634727	0.941730

improvements in the IoU metrics, suggesting enhanced geometric accuracy and better surface coverage of the deformed meshes. These improvements were particularly evident in complex shapes, like the humanoid and the hand.

The qualitative analysis confirmed the superiority of the proposed approach. Visual comparisons highlighted that the GAN-based model produced more realistic and detailed deformations, closely resembling the target meshes. The enhanced capability to capture fine details and intricate structures underscores the potential of adversarial training in 3D rendering applications.

Given all this, this study has shown that the integration of GANs with traditional rendering techniques presents a promising direction for achieving more accurate and realistic 3D mesh deformations, paving the way for advanced applications in computer graphics, virtual reality, and beyond.

References

[1] S. Liu, T. Li, W. Chen, H. Li, Soft rasterizer: A differentiable renderer for image-based 3d reasoning, 2019. URL: https://arxiv.org/abs/1904.01786. arXiv:1904.01786.

[2] N. Wang, Y. Zhang, Z. Li, Y. Fu, H. Yu, W. Liu, X. Xue, Y.-G. Jiang, Pixel2mesh: 3d mesh model generation via image guided deformation, IEEE Transactions on Pattern Analysis and Machine Intelligence 43 (2021) 3600–3613. doi:10.1109/TPAMI.2020.2984232.

[3] Y. Zhu, Y. Zhang, Q. Feng, Colorful 3d reconstruction from a single image based on deep learning, in: Proceedings of the 2020 3rd International Conference on Algorithms, Computing and Artificial Intelligence, ACAI '20, Association for Computing Machinery, New York, NY, USA, 2021. URL: https://doi.org/10.1145/3446132.3446157. doi:10.1145/3446132.3446157.

[4] B. Nicolet, A. Jacobson, W. Jakob, Large steps in inverse rendering of geometry, ACM Trans. Graph. 40 (2021). URL: https://doi.org/10.1145/3478513.3480501. doi:10.1145/3478513.3480501.

[5] H. Su, X. Liu, J. Niu, J. Wan, X. Wu, 3deformer: A common framework for image-guided mesh deformation, 2023. URL: https://arxiv.org/abs/2307.09892. arXiv:2307.09892.

[6] N. Ravi, J. Reizenstein, D. Novotny, T. Gordon, W.-Y. Lo, J. Johnson, G. Gkioxari, Accelerating 3d deep learning with pytorch3d, arXiv:2007.08501 (2020).

[7] Free3D.Com, 3d models for free, 2024. URL: https://free3d.com/3d-models/obj.

[8] PyTorch3D, Renderer architecture, 2024. URL: https://pytorch3d.org/docs/renderer_getting_started.

A. Detailed results

This appendix contains the results obtained by training the models for 10000 iterations, as well as more visual comparisons between objects when the models are trained for 2000 iterations.

A.1. Loss graphs

Figure 4 shows the evolution of the losses during the training process. In particular, it represents the losses when deforming a sphere to obtain a cow for 2000 iterations. All the graphs have more or less the same shape, so this is the only one that will be reported.

A.2. Evaluation metrics at 10000 training iterations

Tables 4, 5 and 6 contain respectively the values of the losses, the Chamfer distances and IoU when the models are trained for 10000 iterations.

Table 4
Comparison of the losses of the PyTorch3D renderer and my model when trained for 10000 iterations.

Model	Cow	Humanoid	Fish	Sword	Hand
`PyTorch3D`	0.022570	0.013449	0.010238	0.010739	0.021070
`Mine`	0.021360	0.012507	0.010399	0.010575	0.018868

Table 5
Comparison of the Chamfer distance computed by the PyTorch3D renderer and my model when trained for 10000 iterations.

Model	Cow	Humanoid	Fish	Sword	Hand
`PyTorch3D`	0.002212	0.000838	0.000444	0.000278	0.001299
`Mine`	0.002204	0.000747	0.000446	0.000278	0.000951

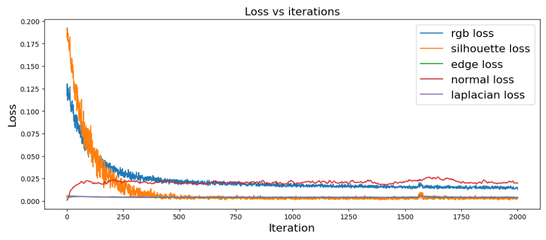

Figure 4: A typical graph showing the losses' evolution during the training process.

The loss values in Table 4 show that both models keep learning when trained for more iterations, but my model learns more than the PyTorch3D renderer. The same can be seen in Tables 5 and 6.

This proves that using a GAN architecture can help in the shape deformation task, producing more realistic results.

A.3. Qualitative results at 10000 training iterations

Figures 5 and 6 show the visual comparison of the meshes deformed by the PyTorch3D renderer and my model when they are trained for 2000 and 10000 iterations respectively.

The images show that my models generates more realistic results in both training scenarios, even if it still struggles with textures that change multiple times in a small space, like the fish and the sword objects.

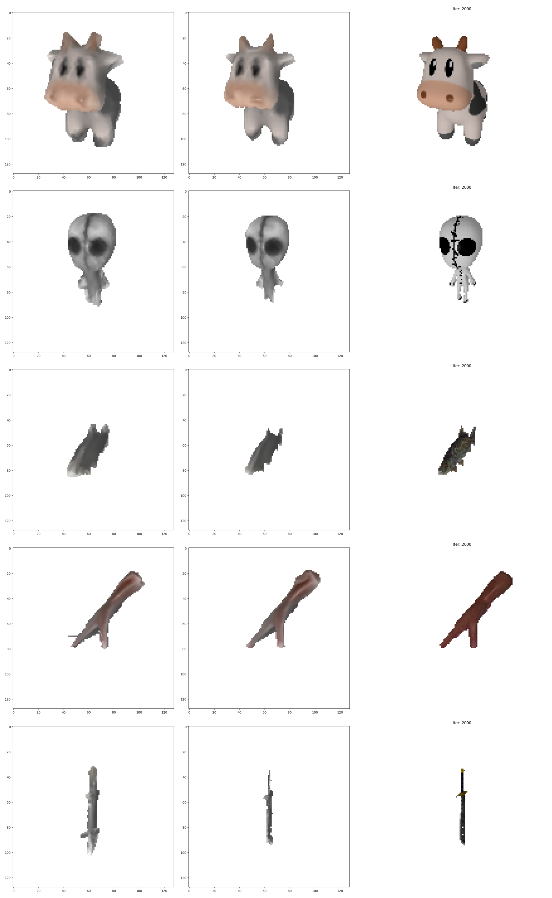

Figure 5: The meshes deformed by the PyTorch3D renderer (left) and by my model (center) when they are trained for 2000 iterations. On the right there are the reference images.

Table 6
Comparison of the intersection over union computed by the PyTorch3D renderer and my model when trained for 10000 iterations.

Model	Cow	Humanoid	Fish	Sword	Hand
`PyTorch3D`	0.973512	0.894125	0.934347	0.626193	0.888504
`Mine`	0.988430	0.922987	0.932218	0.677852	0.960046

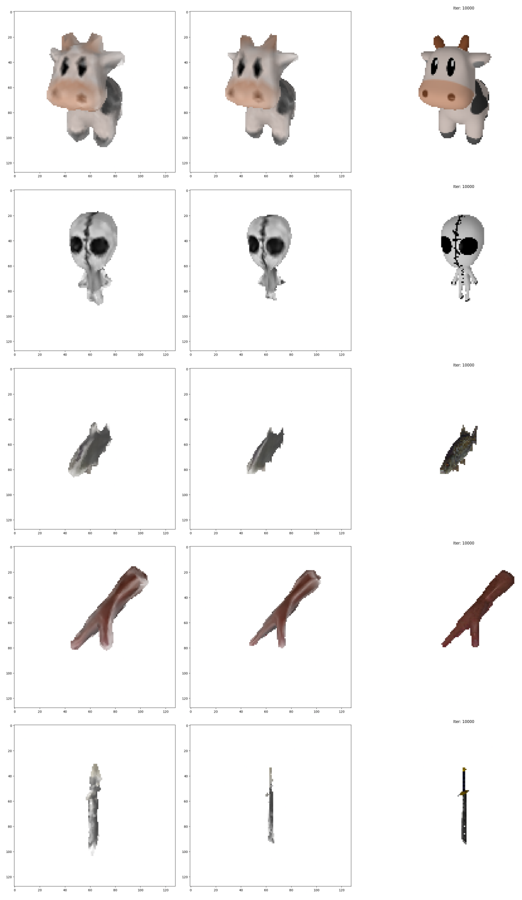

Figure 6: The meshes deformed by the PyTorch3D renderer (left) and by my model (center) when they are trained for 10000 iterations. On the right there are the reference images.