Generalized deep learning model for 3D model construction from 2D images

Duc Tran
NYU
dt2259@nyu.edu

Ishwari Nalgirkar,
NYU
ian4133@nyu.edu

Problem Statement

Coming across a recent research related to computer graphics where 2D image can be reconstructed to 3D object using deep learning technique approach. Our project explores the pipeline discussed in the paper ”Learning to Predict 3D Objects with an Interpolation-based Differentiable Renderer” published by NVIDIA in 2019 and ”Accelerating 3D Deep Learning with PyTorch3D” published by Facebook AI in 2020. Although the two papers focus on introducing the (Differentiable Renderer) DIB-R and Pytorch3D engine for deep learning graphics pipeline, our primary goal is to use this engine to implement various neural network models trained to reconstruct 3D objects from a single image and discuss various impacts these models have on the result as well as explore the exciting field of 3D deep learning.

Data Set

3D to 3D: One of the datasets we will be using is 3D object from clara.io. The camera from our graphics engine will take pictures of the object from different view points and use it as input to train our network and mold the 3D sphere into our desired shape. As of now, we are considering 24 camera view points as our input. We are also considering the dataset Shapenet with 51000 different 3D models to test how well the objects with a higher structural complexity are generated.

2D to 3D: In practice, having a camera that captures the object from multiple view points and stores these images is inefficient. Therefore we use Style based GANs to generate input images for our differentiable renderer engine.To train our GAN, we are using the bmw10_release dataset, the GAN is also pretrained on the LSUN dataset (transfer learning) as abundant images of our object may not be available and computational expensive to train.

Segmentation: Lastly we use a pre-trained deeplabv3_resnet50 models to perform image segmentation to create a silhouette of our targeted object.

Models

As we dived deeper into the project, we learned that the 3D deep learning pipeline using Pytorch3D has already implemented many of the functionalities as well as abstract away the complexities to help us achieve 3D model reconstruction. DIB-R is also a similar engine, however installation requires an NVIDIA GPU to run. These libraries are a collection various techniques that take many years of research from different universities and organizations. Building these techniques from scratch would pose a huge challenge and we would be reinventing the wheel. Thus we try to build a pipeline integrating these neural nets to understanding Graph-CNN, the neural network model of Pytorch3D and the 4 loss functions along with SGD and Adam optimizers that support the accuracy of moving the mesh vertices or pointclouds to the desired locations,

We also would like to implement Quadric Loss function as mentioned in ”Learning Embedding of 3D models with Quadric Loss” since Pytorch3D does not provide this function and compare it with the chamfer, edge, normal, laplacian loss provided in the Pytorch3D library. The next neural networks we consider is Style-GANs, a type of Generative Adversarial Network which uses adaptive instance normalization, it changes the input levels at each layer and generate high resolution synthetic images. We will experiment with different parameter to generate our dataset to reconstruct 3D model from single 2D image.

Convolutional Neural Nets are proven to perform very well with 2D images due to its property of shift invariance and localization on pixels data. However with 3D model, object are represented as a densely connected points, therefore Graph-CNN is used to process connection between datapoints in 3D space. We won’t dive deep into Graph-CNN as it is a part of Pytorch3D engine to help us shape our object mesh and we are not able to tune its parameter.

Molding Mesh Pipeline

Much like a human making a sculpture, the pipeline of inverse (Figure 1) graphics starts with having a sculptor look at the targeted object, taking photo memory of the object from different angle then start working on changing the shape of the clay. What happen if we only have one image of one angle of the object? The human brain starts to imagine different angle based on previous experience of what the object is (StyleGAN)

When looking at the object, only the data of the object is registered and not the surrounding. We use pre-trained deeplabv3-resnet50 to achieve segmentation of object from background.

By checking the desired picture and the picture of the molded sphere, the neural network will learn to move the vertices of the sphere such that the picture of molded sphere matches the desired image. However, the graphics pipeline originally used optimized the process of rasterization in such a way that gradient based techniques did not perform well, the original graphics pipeline does not effectively connect pixels back to the 3D objects which hampers the ability of the model to understand the vertices. The differentiable rasterization technique allows the produced images to be used to fully backpropagate through all the predicted vertices.

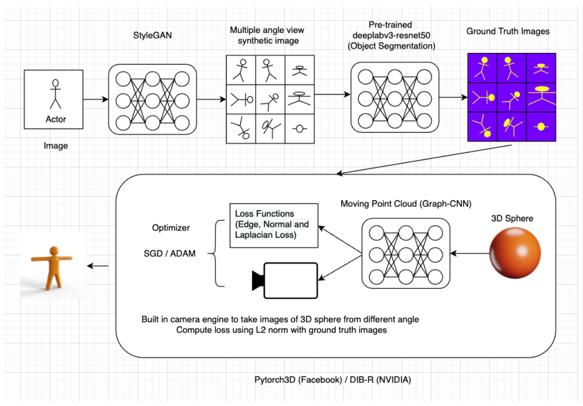

StyleGAN
Multiple angle view synthetic image
Pre-trained deeplabv3-resnet50 (Object Segmentation)
Ground Truth Images
Actor
Image
Moving Point Cloud (Graph-CNN)
Optimizer
Loss Functions (Edge, Normal and Laplacian Loss)
SGD / ADAM
3D Sphere
Built in camera engine to take images of 3D sphere from different angle
Compute loss using L2 norm with ground truth images
Pytorch3D (Facebook) / DIB-R (NVIDIA)

Figure 1: 2D to 3D Pipeline

Synthetic Images With StyleGAN2

The StyleGAN2 - ADA produces state of the art results in terms of resolution and closeness to the ground truth, it improves over the initial StyleGAN by improving normalization over latent vectors in generators, This is done to improve style mixing and feeding a different latent vector to different layers at inference time. The StyleGAN2 is able to increase mixing without reducing meaningful interaction between data.

The StyleGANs are able to change the viewpoints of the image by manipulating the latent code and style mixing. The first few layers of the StyleGAN2 ADA architecture seem to be responsible for the camera viewpoint and can be tweaked to produce various viewpoints of the same object.

We train our StyleGAN2 on our car object images and then resume the training from that point on a pre-trained model trained on the LSUN car dataset, it helps us in generating meaningful images using the state of the art model. To generate the images of the same object from different viewpoints, we need to add an encoder. This is done by generating a high level space using a generated image and the object.

The procedure up till now only gives us the model of the GAN , even though we have generated synthetic images, we are not replicating the original project image with its different viewpoints. To get the final output we have to use an encoder to test our model on the object we intend to apply to our differential rendering engine.

We are working with pre-trained state of the art models which do not require much hyper parameter tuning to perform well. In our next step when dealing with the differential renderer and loss functions we will be able to play around more with tuning and apply different parameters to experiment with the outputs.

Below is a comparison between traditional GAN and StyleGAN taken from the orignal StyleGAN paper ”A Style-Based Generator Architecture for Generative Adversarial Networks”, according to the author, the StyleGAN model has 26.2M while tradition GAN has 23.1M trainable parameter. Here the inputs is mapped through 8 layers FC, which then feed into a latent space W. A is the affine transformation where GAN will learned to generate different images while B. is just a scaling factor when Gaussian noise is applied. The synthesis network has total of 18 layers dealing with different image resolution such as from 4x4 to 1024x1024.

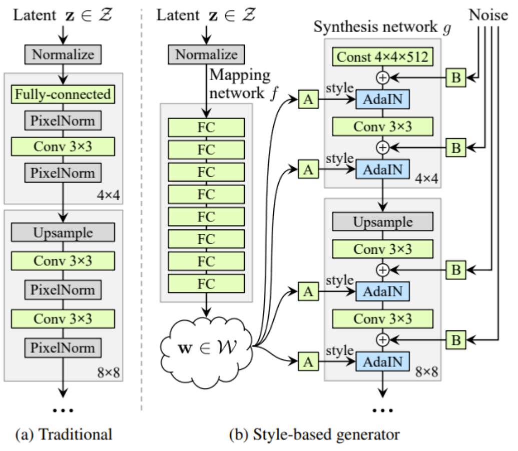


Figure 2: Generator StyleGANs

Loss Function

According to the paper ”Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images”, there are 4 losses they consider in their models to generate good results: Chamfer Loss, Normal Loss, Edge Loss and Laplacian Loss. The paper does not go into details why these losses are useful, therefore we would like to attempt explaining the impact these losses have on the models. (Note: All result shown in this section regarding loss function is performed by using a 3D object and

3D sphere to mold and is not yet involved synthetic images using StyleGAN)

Chamfer Loss:

$$\sum_p min_q ||p-q||_2^2 + \sum_q min_p ||p-q||_2^2$$

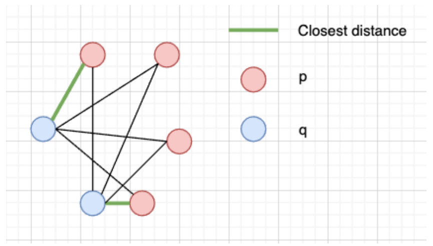


Figure 3: Chamfer Loss illustration

Suppose p is data points on the sphere and q is the ground truth data points. At every point in p, the Chamfer loss looks for points in q with the closest distance using l2 norm between p and q. The same applied for every points in q, hence the second summation. The total loss is the sum of the closest distance between p and q and vice versa. However, finding this loss by using matrices multiplication is computationally expansive, Pytorch3D applied K Nearest Neighbors (KNN) to speed up finding Chamfer loss as well as saving memory when dealing with a very large number of vertices. The Chamfer loss is crucial in finding the overall structure of the target object. Here is the result where we adjust Chamfer loss to demonstrate its impact:

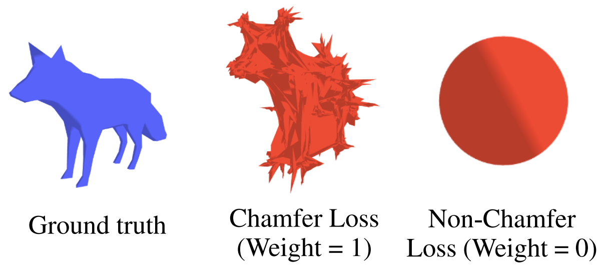


Figure 4: Chamfer Loss Impact

We can see that without Chamfer loss, the sphere object does not change at all. When the weight of Chamfer loss equals 1, the sphere being molded to the structure close to the ground truth object. However, there are many spikes happen along the surface.

Edge Loss:

$$\sum_p \sum_{k \in N(p)} ||p-k||_2^2$$

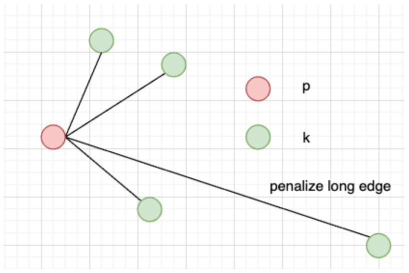


Figure 5: Edge Loss illustration

Distance between 2 points form an edge, therefore, for every neighboring points k of p ($k \in N(p)$), we will calculate the distance using l2 norm and sum up all the distance to penalize the models by edges. If there is a very long edge, the model will be penalized. The edge loss is used to resolve these sudden spikes along the surface. Figure 4 demonstrates the result of Chamfer loss combines with Edge loss.

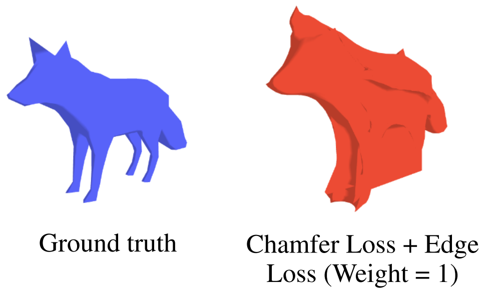


Figure 6: Edge Loss Impact

As the result, we can see the surface is smoother, no more sudden spikes however there are still corner details that requires more tuning to match with ground truth model

Normal Loss:

$$\sum_p \sum_{q=min_q ||p-q||_2^2} || < p-k, n_q > ||_2^2, k \in N(p)$$

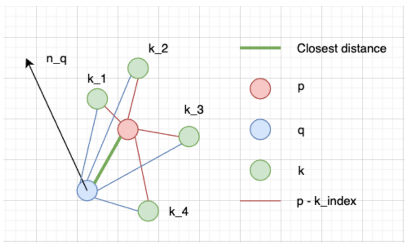


Figure 7: Normal Loss illustration

The normal is a vector perpendicular to a plane that contains information about the direction of the plane. It is useful in computing a smooth shading such as Phong or Gouraud. However in our model, we consider normal loss to keep the direction of our plane consistent and better tuning our result. From figure 5 as well as the equation, we can see the normal loss is considered between point p and q where points q is closest distance to

p. Surrounding q is multiple nearest neighbor points k. When two vectors are perpendicular, its dot product is 0. We know that the normal is a vector perpendicular to the plane. Therefore, the dot product between the edge (red line) between point p and k (p - k) and the ground truth normal (n_q) must be 0 if point p (red dot) is overlaying point q (blue dot). By minimizing the distance between p (red dot) and q (blue dot), we are expected to get a better model. Here is our result:

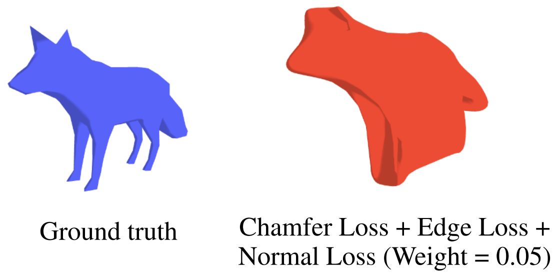


Figure 8: Normal Loss Impact

As we observe from figure 6, our result is much better with smoother surface and completely eliminate any strong spikes as seen from figure 4.

Laplacian Loss:

$$uniform = \delta p = p - \sum_{k \in N(p)} \frac{1}{||N(p)||} k$$

$$Laplacian\ loss = \sum_{p} = ||\delta' p - \delta p||_2^2$$

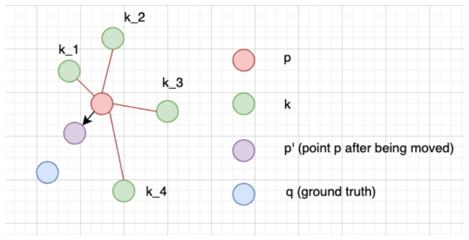


Figure 9: Laplacian Loss illustration

The goal of applying the Laplacian Loss is to keep the point p on the sphere from moving too far away from its neighboring points. Laplacian value is calculated by summing up all neightborings points and treat these points with a uniform weight ($\frac{1}{||N(p)||}$). There are different weight we can apply such as cotangent and co-curvature weight. By calculate the difference of Laplacian of the current points and its next position, we get the Laplacian loss.

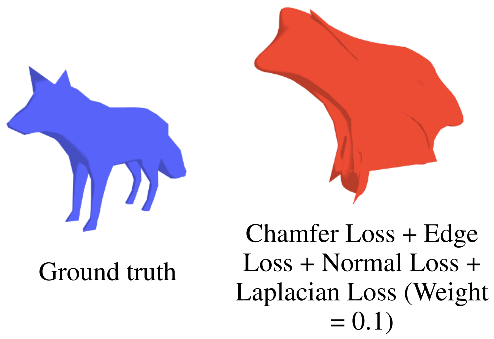

Ground truth
Chamfer Loss + Edge Loss + Normal Loss + Laplacian Loss (Weight = 0.1)

Figure 10: Laplacian Loss Impact

Quadric Loss:

$$\sum_{s \epsilon Vout\ t \epsilon Vin} = s^T Q_t s$$

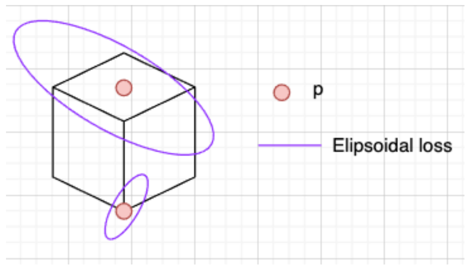


Figure 11: Quadric Loss illustration

The goal of applying the Quadric Loss is to keep the point p from moving too far away along the sharp edges and enhance object details. The equation takes in a Q matrix which compute by using the plane on the surface of ground truth object. S is the vertices of the newly generated sphere after each itertation. By taking the transpose of s multiply by Q matrix and non transpose s, we get the ellipsoidal loss. The ellipsoid size will be very large or infinite if points are on a flat plan. This means the points can lie anywhere on the plane. While when points are along a sharp edge, the ellipsoid size is small and limit the degree of freedom the points can move. According to ”Learning Embedding of 3D models with Quadric Loss” paper, this will be greatly improve the details along the objects edges. We have implemented Quadric loss to our Pytorch3D pipeline to confirm the result stated in the paper.

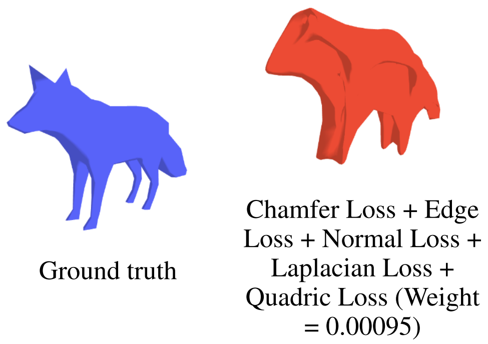

Ground truth
Chamfer Loss + Edge Loss + Normal Loss + Laplacian Loss + Quadric Loss (Weight = 0.00095)

Figure 12: Quadric Loss Impact

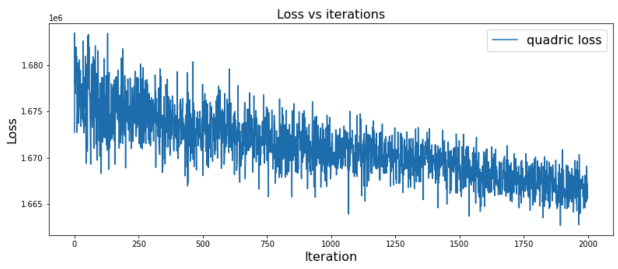


Figure 13: Quadric loss

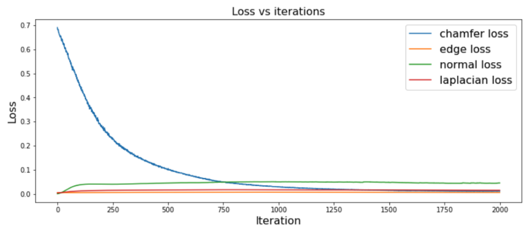


Figure 14: Other losses

To our surprise, the 3D output is much closer to the ground truth when quadric loss is applied. The area between the fox leg and tail is clearly shown. Since the object has lots of flat plane along the curve surface and not alot of edges, the quadric loss return value is huge, therefore we have to apply a very small weight to accurately tuning the object. From the graph, we can see quadric loss generate lots of noises but it constantly goes down per iteration. The chamfer loss and other losses have small value and also goes down very quickly in each iteration.

Optimizer

Another factor in these neural nets is the optimizer used, we are using two optimizers to update the gradients for the meshes formed at each iterations, we try out the SGD optimizer which only uses a subset of the generated meshes to update the gradients and compare it with Adams optimizer which adds a stochiastic property and is supposed to train the nets faster, as we are back propagating, speed should also be optimized. We can see considerable difference between the performance of these two optimizers. SGD is able to form moulds closer to ground truth as compared to Adams when we have the same learning rate, however when we reduce the rate in the range 0.01 - 0.05 Adam seems to perform well if not better than SGD. We aim to learn more as to why there is such a considerable difference, and how can we tune the parameters to optimize the gradients better.

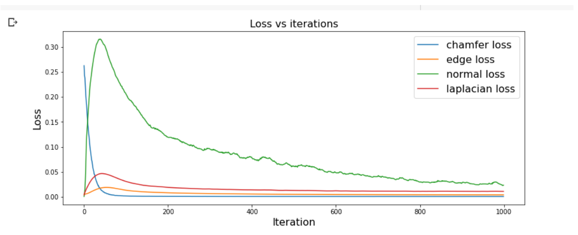


Figure 15: Adam performance

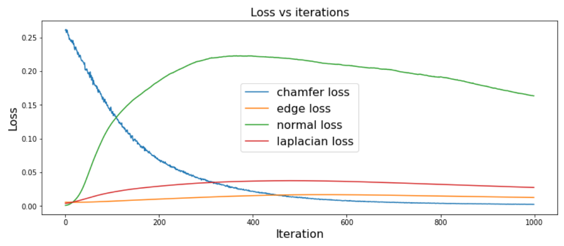


Figure 16: SGD

Generate 3D Object From Rendered Images

Once we have our model and options for optimizer, the next step in our pipeline is to try modeling our sphere with images. Before using StyleGAN, we would like to test if our model works well for some easy obtain and stable ground truth images. Pytorch3D provides a built in camera rendering system which we can use.

According to Pytorch3D tutorial, the camera engine is able to render images of 3D objects with either meshes or silhouette mode. We focus only on silhouette mode as our main focus is shaping the 3D object and not worry about the texture and lighting of objects. Using the the silhouette camera. One of our challenge here is our fox object coordinates are very big therefore it took us a long time debugging the issue and figure out how to scale the object down to fit with our camera coordinates and sphere. Our camera will move from ground up (0 to 180 degree) and left to right (-180 to 180 degree) to be able to capture object angle. Here is the 20 angle images generated of the fox object.

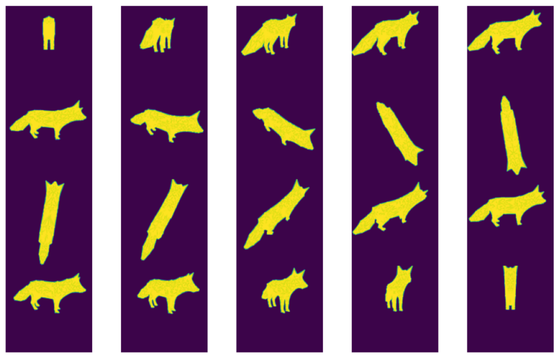
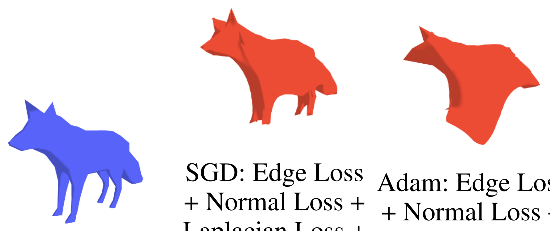


Figure 17: Fox Silhouette

Next we modify our model to include a camera that will constantly move around the object and capture the exact 20 angles of the sphere and compare it with our ground truth images. While comparing, we have new loss function called the silhouette loss which is a simple L2 norm between the sphere silhouette and fox silhouette. This loss function will guide the way the sphere is shaped. We also don't consider chamfer loss and quadric loss as those 2 losses can only make impact when using a 3D object as ground truth and not images. Here is our result:



Ground truth

SGD: Edge Loss + Normal Loss + Laplacian Loss + Silhouette Loss (Weight = 1.0, lr = 1.0, momentum=0.9)

Adam: Edge Loss + Normal Loss + Laplacian Loss + Silhouette Loss (Weight = 1.0, lr = 0.001)

Figure 18: Silhouette Loss Impact

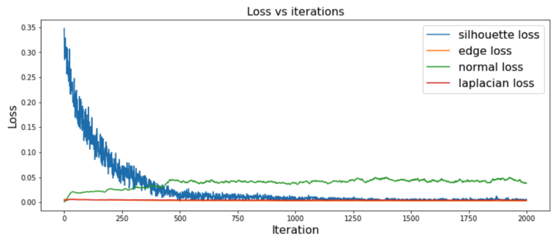


Figure 19: 3D object from images SGD performance

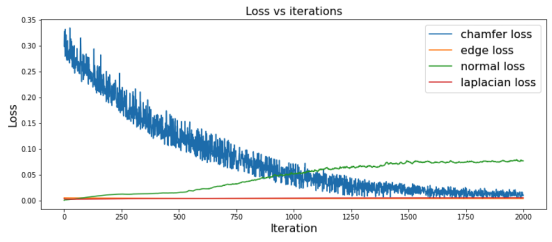


Figure 20: 3D object from images Adam performance

Interestingly, molding object from images yields better result than molding object from 3D object. The silhouette loss in combination with edge, normal and laplacian loss along with SGD optimizer performs extremely well. The sphere object is closer to the ground truth than ever. Although the normal loss drastically increase, we don't see a huge amount of spike along the object surface. The silhouette is noisy but constantly drops as our object is forming into the ground truth shape. In the beginning of class, we learned that SGD performs better than other optimizer despite its well-known issues for unknown reason. That fact is clearly shown here where Adam optimizer did very poorly on estimate the shape of object. While the two loss functions trend shows similarity, however the shape of object generated using Adam is not close the ground truth object as well as forming a very flat shape instead of 3D like. The silhouette loss also goes down alot slower than SGD.

One more interesting point that we observe is the training time is a lot longer for images compare to 3D object. Almost 10 times longer, with 3D object we take 1:30 minutes on average while with images the training time take 10 minutes.

Generate 3D Object From StyleGAN2 Images

From walking through all the steps and features provided within the differential renderer engine, finally we will use images generated from our StyleGAN2 to feed in our current models. Since training StyleGAN is computationally expensive and require a very large dataset, we are only able to generate a small subset different cars with limited amount of different angle. Even with limited result from StyleGAN, we woud like to experiment with our models to see if it will yield any interesting result.

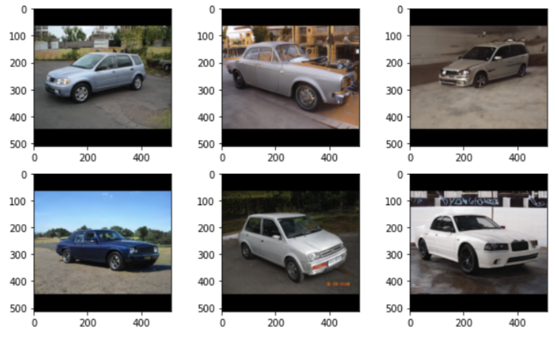


Figure 21: Cars Image Generate By StyleGAN

Segmentation: The images generate by StyleGAN are mixed with background, we need to to find away to segment our object from the background in order to generate silhouette images similar to the above step. According to GANVerse3D by NVIDIA, which they use a DatasetGAN for accurately perform object segmentation along with all the components of the objects. Luckily, Pytorch Vision has an abundant resources on pre-trained models which we can borrow and allow us to simplify this process. We use deeplabv3-resnet50 as our pre-trained segmentation models.

One of the problems with StyleGAN generated images is that it does not focus on the objects even while training, it aims at producing high resolution images with intricacies. This problem can be also approached by feeding our GAN an already blurred images. Here is our segmentation result with deeplabv3-resnet50:

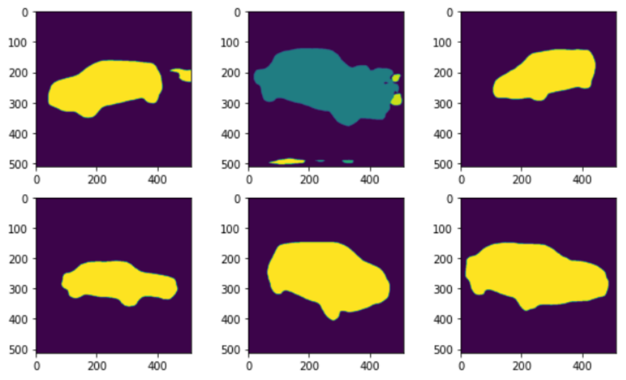
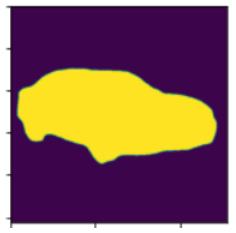


Figure 22: Silhouette Image Generate By Resnet

Final Result:



Car Ground Truth Image

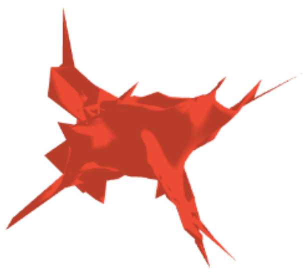


Figure 23: 3D Object Using StyleGAN Inputs

Figure 24: StyleGAN Inputs

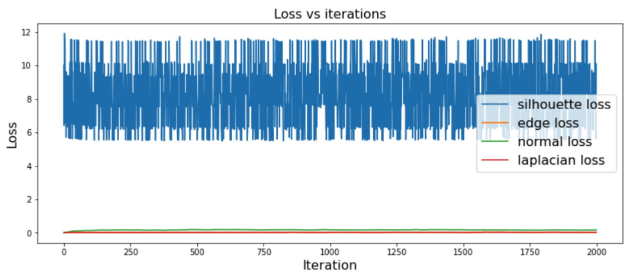


Figure 25: StyleGAN Inputs Performance Graph

Unfortunately our final result is not what we expected, our suspect is we don't have enough training images of cars from different angle to yield an accurate result. Also our camera movement to take pictures of sphere also requires configuration to creating a better model. Our silhouette loss clearly shows it fails to formulate the correct shape as the curve is very noisy across but does not decrease. Another reason for this could be the background noise in the input images.

Conclusion

As the result, through both theory as well as implementation, we have achieve a good understand of the field of 3D deep learning. There are many components along the pipeline, each requires very specific tuning to make the model behave as we expected. We have tried with many different parameter for our models regardin the weight for loss functions, optimizer, learning rate and momentum. The result shown here is based on our best model that we have tested through each step in the pipeline. Along the way, there are moments of excitement to see the sphere shape into our desire target object, putting theory into practice by analyzing loss function and learn the impact each of them make on the model, implementing quadric loss to see how much it will improve the current model. Our final result is not what we expected but we believe in the future we will get StyleGAN to generate our desire inputs and able to achieve a better result. We have learned a lot through various research as well as happy to see our knowledge in class can be applied and helped us understand the latest trend in deep learning. Overall we have achieved and able to perform all the tasks that we are intended to do for this project.

Challenge

We face many issue with understanding the pipeline as it is a combination of various neural network techniques, each is a field of its own and requires extensive study to further understand. Configuration is also a problem as these implementations varies across different programming framework, we have to extract and convert them to fit with our Pytorch3D pipeline. Computational cost is one of the challenge we face when using GoogleColab as our access to GPU is limited and often timeout on our account after certain amount of used time. We have to switch through 5 different accounts to keep our GPU access going.

Acknowledgement

We give many thanks and credits to Pytorch3D tutorials, Quadric Loss paper and implementation, Pytorch Vision tutorial and professor Hedge Chinmay for providing us with the tools and knowledge to help us implement and explore the exciting field of 3D deep learning.

Github

All code and implementation can be found at: `https://github.com/duketran1996/inverse-graphics`

Literature Survey

- ”Learning to Predict 3D Objects with an Interpolation-based Differentiable Renderer (DIB-R)” introduces the pipeline of constructing 3D objects from a single 2D image using StyleGAN and presents a simple training model.
- ”Accelerating 3D Deep Learning with PyTorch3D” introduces the functionalities and architecture of Pytorch3D.

- ”Learning Embedding of 3D models with Quadric Loss” applies Quadric Loss to improve the predicted 3D shape along sharp edges.
- ”Deep Convolutional Inverse Graphics Network” represents a deep convolutions inverse graphics network to learn rendered images, we aim to understand and devise an efficient similar deep learning model to work with DIB-R.
- ”Image GANS meet differentiable rendering for inverse graphics and interpretable 3D nerual rendering” introduces StyleGAN which can generate different camera view points from one single image.
- ”A Style-Based Generator Architecture for Generative Adversarial Networks”

Online Resources

- **DIB-R**
  `https://arxiv.org/pdf/1908.01210.pdf`
- **Pytorch3D**
  `https://arxiv.org/pdf/2007.08501.pdf`
  `https://pytorch3d.org/tutorials/`
- **Convolutional Inverse Graphics Network**
  `https://proceedings.neurips.cc/paper/2015/file/ced556cd9f9c0c8315cfbe0744a3baf0-Paper.pdf`
- **Image GANS meet differentiable rendering for inverse graphics and interpretable 3D nerual rendering**
  `https://arxiv.org/pdf/2010.09125.pdf`
- **Clara Io Dataset**
  `https://clara.io/library`
- **Shapenet Dataset**
  `https://shapenet.org/`
- **A Style-Based Generator Architecture for Generative Adversarial Networks**
  `https://arxiv.org/pdf/1812.04948.pdf`
- **Learning Embedding of 3D models with Quadric Loss**
  `https://bmvc2019.org/wp-content/uploads/papers/0452-paper.pdf`
  `https://github.com/nitinagarwal/QuadricLoss`
- **StyledGan2-ADA Guide**
  `https://colab.research.google.com/github/dvschultz/ml-art-colabs/blob/master/Stylegan2_ada_Custom_Training.ipynb?fbclid=IwAR3As5Qff4xHUtRVqNa20W9bwyudlOJMpD0eNjNhh4URcs_5IVPq5qWuS6c`