Training of a 3D GAN

Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modelling

The Task

- Map a latent vector z into 3D objects GAN with transposed 3D convolutional layers
- Get also a discriminator for 3D-Shape recognition (D has to capture the structure of a 3D object during training)
- The space of 3D shapes is high dimensional → **learning is hard**

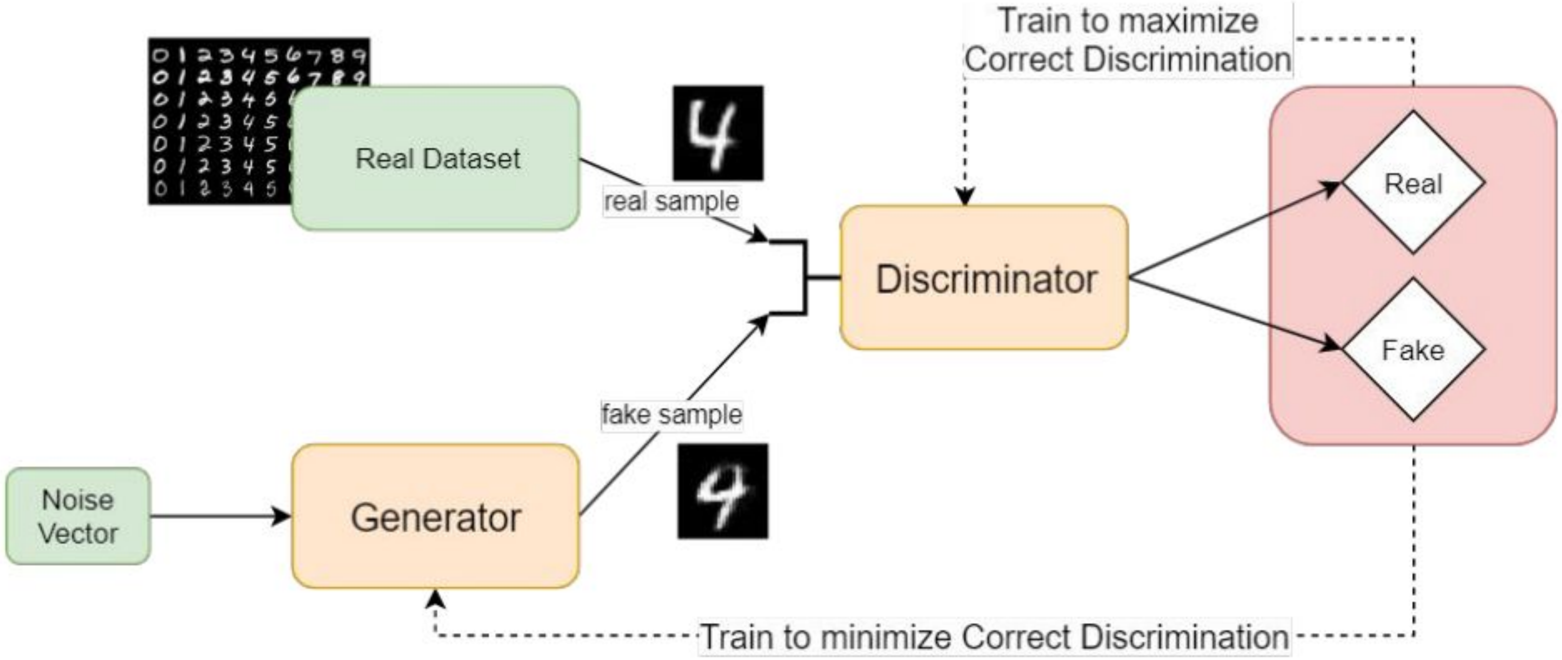
Gan Losses

- standard Generator and Discriminator settings

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

But in practice

- D maximizes: $L^D_{\text{3D-GAN}}$ = log(D(x)) + log (1-D(G(z)))
- G maximizes: $L^G_{\text{3D-GAN}}$ = log(D(G(z))) → not a real zero sum game

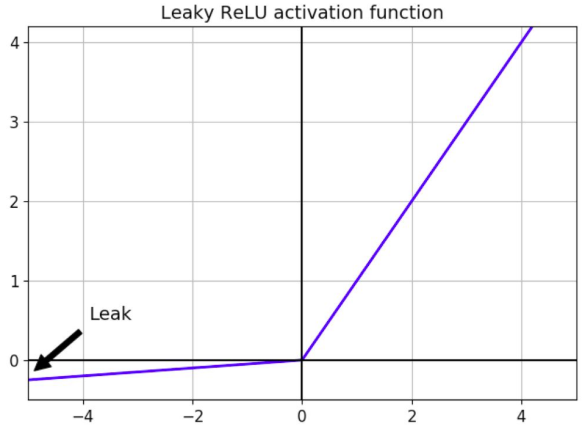

Train to maximize Correct Discrimination
Real Dataset
real sample
Discriminator
Real
Fake
fake sample
Noise Vector
Generator
Train to minimize Correct Discrimination

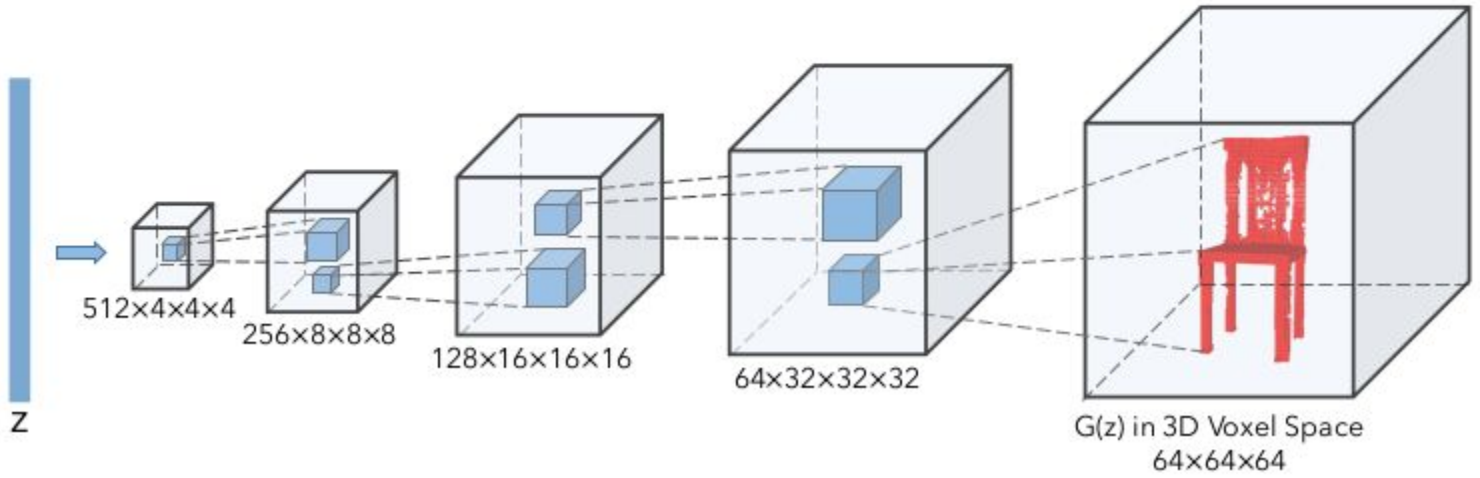
Network details: G

- 5 volumetric convolutional transposed layers of kernel size 4x4x4 and strides 2 followed by:
    - batch normalization
    - ReLU activations
    - except the last layer which does not have batch normalization and has a Sigmoid activation

Network details: D

- 5 volumetric fully convolutional layers of kernel size 4x4x4 and strides 2 followed by:
  - batch normalization
  - **Leaky ReLU** activations and Sigmoid at the end (leak value of 0.2)
  - except the last layer which does not have batch normalization and has a Sigmoid activation




z
512×4×4×4
256×8×8×8
128×16×16×16
64×32×32×32
G(z) in 3D Voxel Space
64×64×64

“Nash equilibrium”

- due to the high dimensionality of the space, D learns much faster than G
- differentiating between generated and real shapes is easy, especially at the beginning when the generator produces poor results
- it is difficult to find an equilibrium between the two
- → the authors propose to update the Discriminator only if its accuracy is less than 0.8

Training

- ModelNet10 dataset, in contrast to the paper which uses ShapeNet
- Attempt to generate shapes of *sofa, table dresser and monitor*
- all the hyperparameters used are the same of the paper, the only one which is changed is the batch size

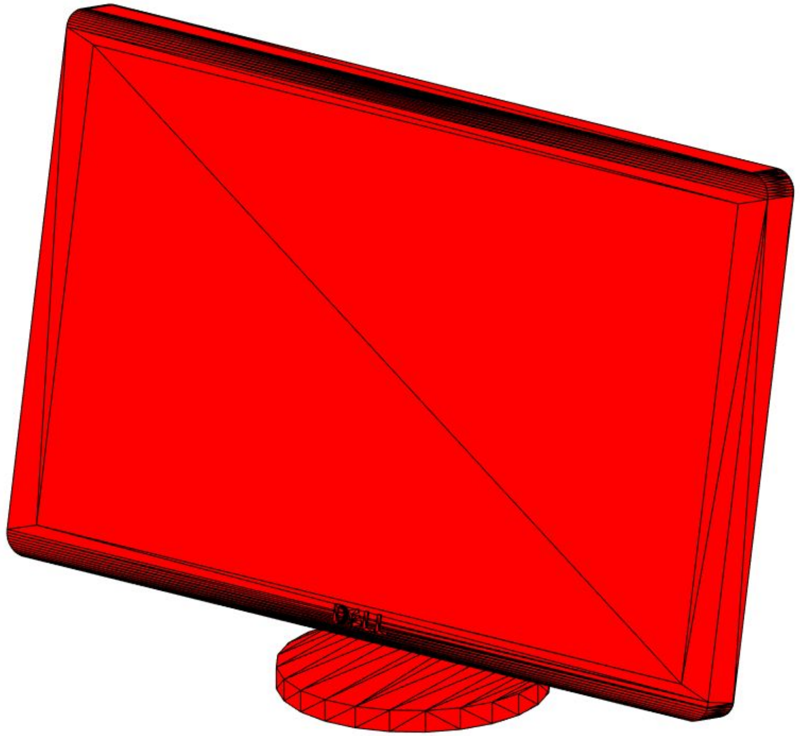
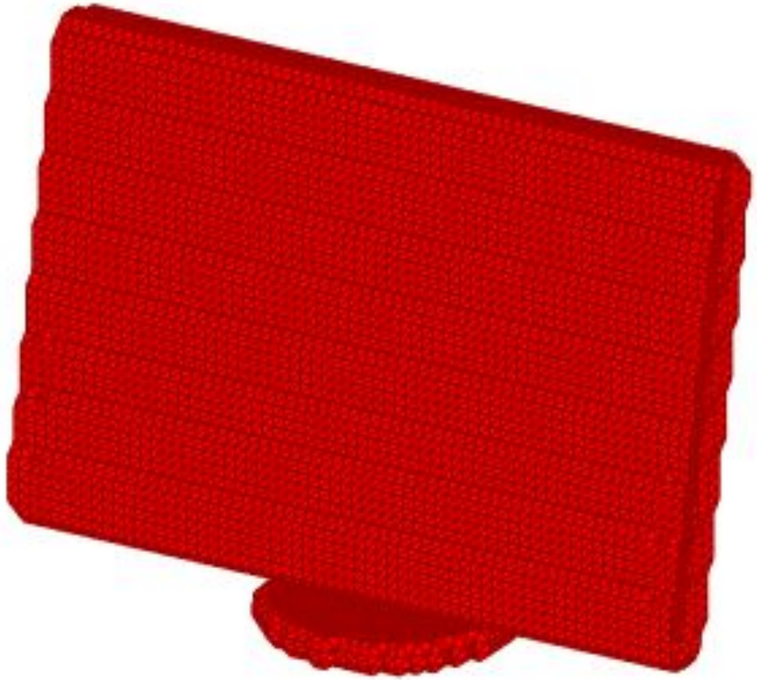
Mesh and Voxel Domains

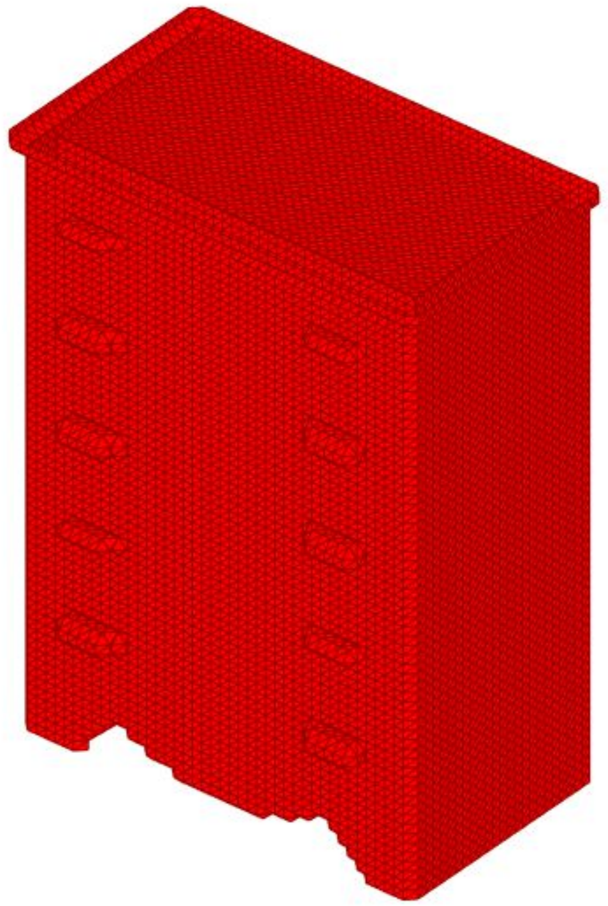
- Triangle Mesh Domain : a mesh file is a collection of n vertices (n vectors, 1x3) stacked with a collection of triangle connections (m vectors of integers, 1x3) which specifies which vertices are on the same triangle

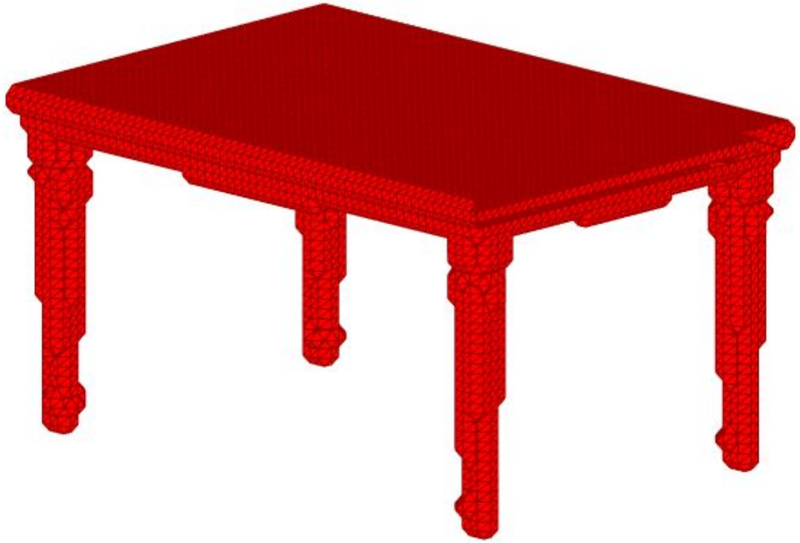
- Voxel Domain: a cube matrix of dimension 64x64x64 that accepts only 1 and 0 as values, 1 means filled and 0 means empty

Dataset preprocessing (Done in Matlab)

- The network works with voxels format of shapes in .mat format
- take 3D triangle meshes in from **ModelNet10** in .off format
- implementation of a function to read the .off mesh files and arrange each one in a data structure containing the vertices and the triangle connectivity
- transform all the shapes from mesh domain into voxel domain with an already implemented function *polygon2voxel* and save these in .mat format.
    - a correction was made to the saving function so that it only saves the volume produced and not any other variable
- implementation of a function to visualize properly corresponding objects before and after the transition

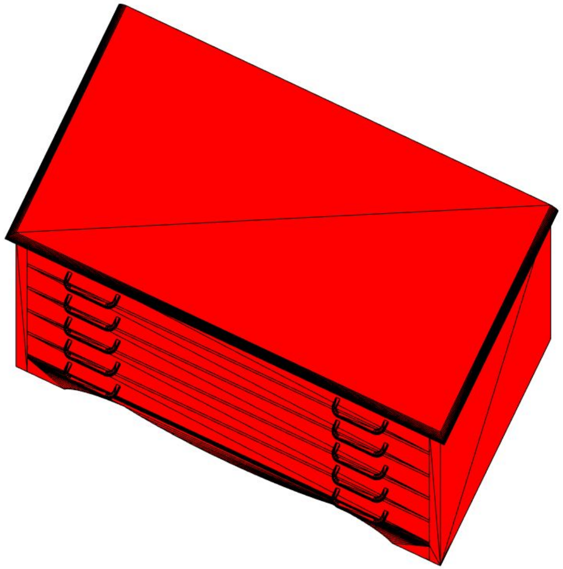



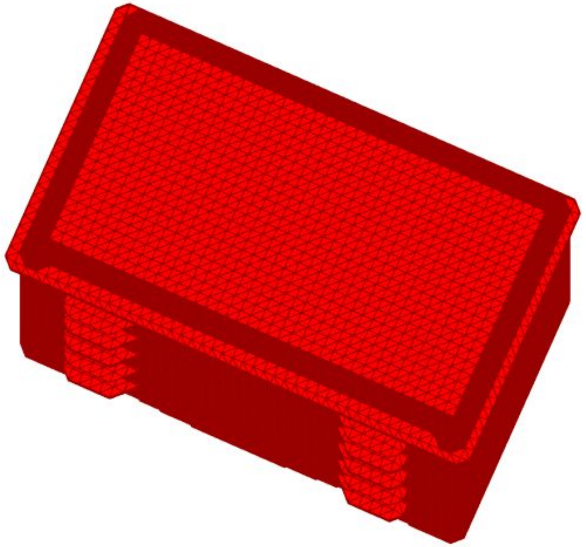




Issue #1

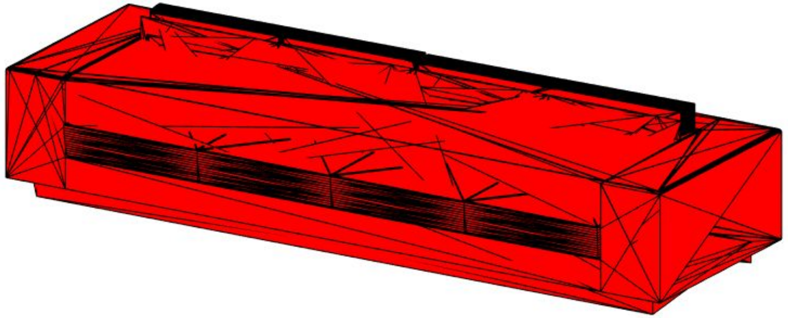
- loss of some details
- edges with many triangles might have holes!

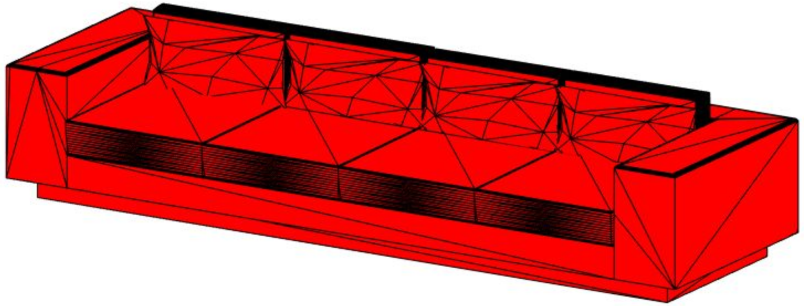




Issue #2

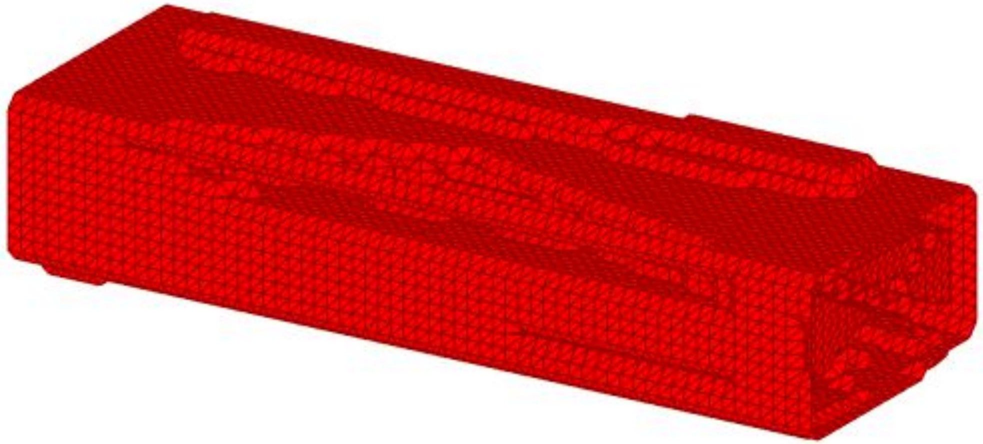
- a few meshes on the dataset are ill-defined → their triangle connection starts from the second vertex





Issue #2

- could not be solved, there is no way to know if the mesh is wrong
- → the result produce will be, which will put noise in the dataset



Training Framework (Colab)

- dataset of voxel shapes is uploaded on Google Drive
- the training is done in a Colab Notebook in which:
  - the github repository with the code is cloned
  - the dataset is copied from Google Drive inside the folder
  - the training is run
  - images of the shapes are saved to monitor the training
  - the output shapes, the logs and the models produced are saved manually during training
  - tensorboard is loaded
  - extra functions to:
    - load the model and continue training
    - visualize tensorboard with the logs from Google Drive

Task 1: produce a sofa

- 780 shapes
- supposed to have an easy probability distribution

Training in PyTorch

- the model is exactly the same but with 2 major differences:
    - 1) label smoothing: the labels for D(X) is uniformly sampled in the interval (0.7, 1.2) and for the D(G(z)) is uniformly sample from (0,0.3)
    - 2) learning rate scheduler: provides several methods to adjust the learning rate based on the number of epochs
    - 3) batch size is set to 32

Adding Label Smoothing

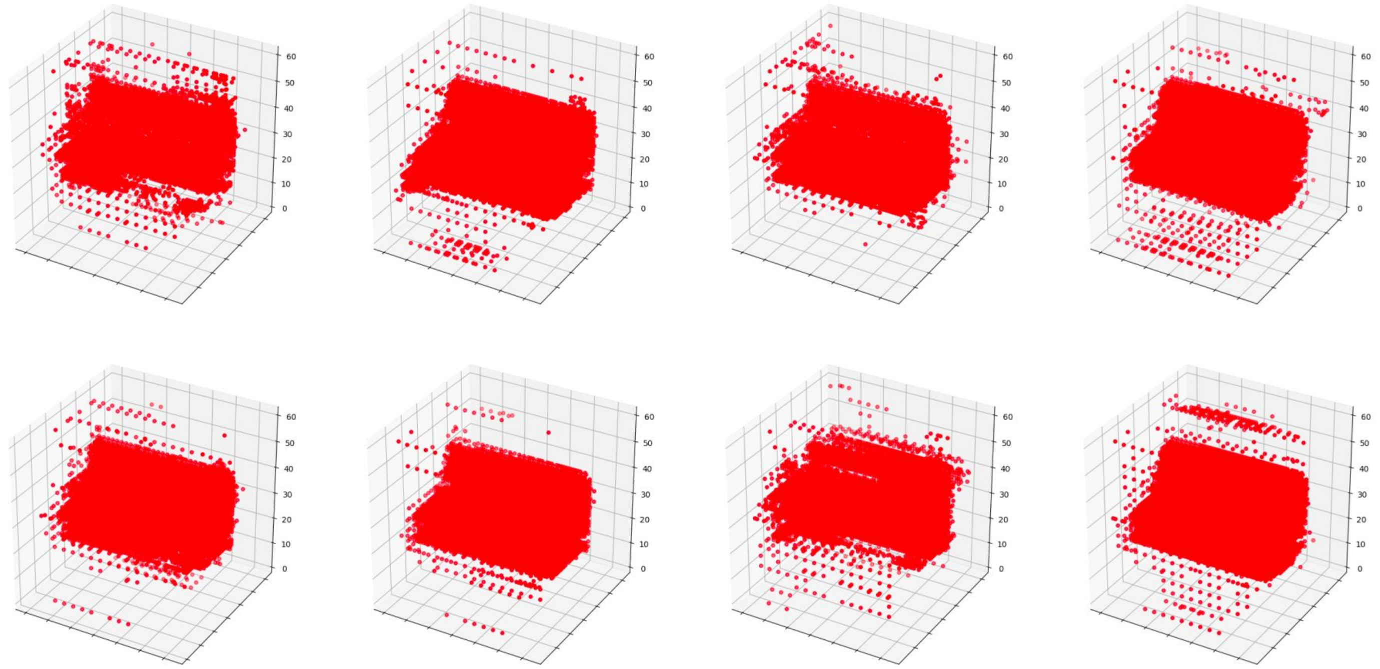
According to the paper *Improved Techniques for Training GANs* it is not suggested to use labels for the fake samples (beta) different from 0

$$D(\boldsymbol{x}) = \frac{\alpha p_{\mathrm{data}}(\boldsymbol{x}) + \beta p_{\mathrm{model}}(\boldsymbol{x})}{p_{\mathrm{data}}(\boldsymbol{x}) + p_{\mathrm{model}}(\boldsymbol{x})}$$

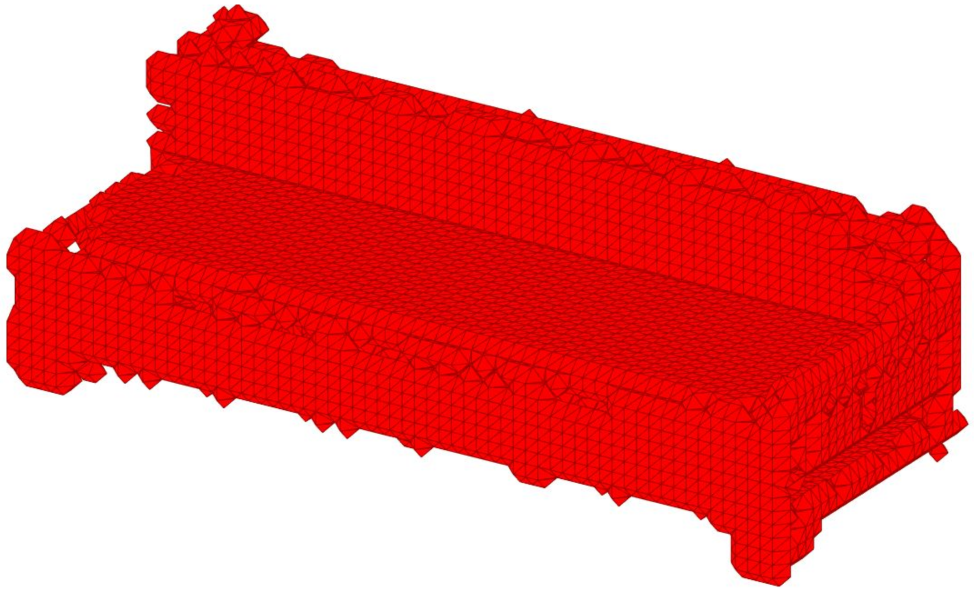
- In practice setting beta to zero give worse results
- The results produced are good but can’t be perfect since the discriminator is not set to learn perfectly (an idea would be to have alfa and beta smoothly and slowly move to 1 and 0 during training)

After only 40 epochs...

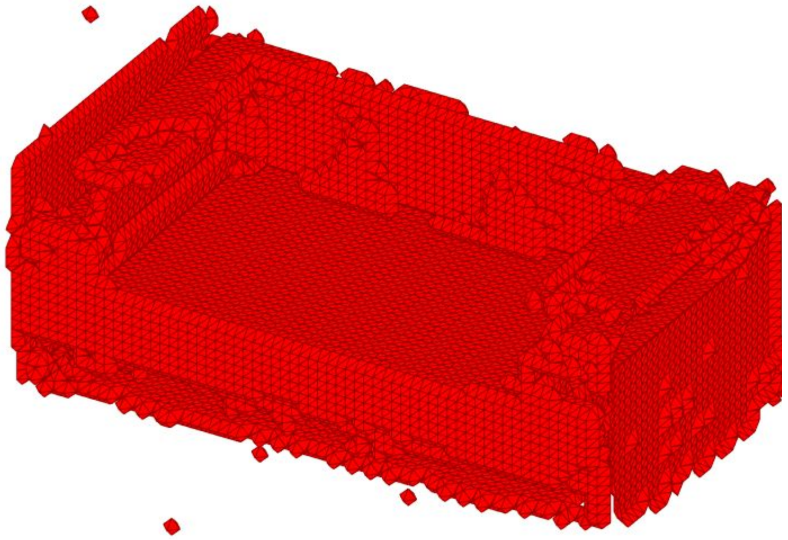
The shape of the sofa starts taking form (first 8 elements of the batch)

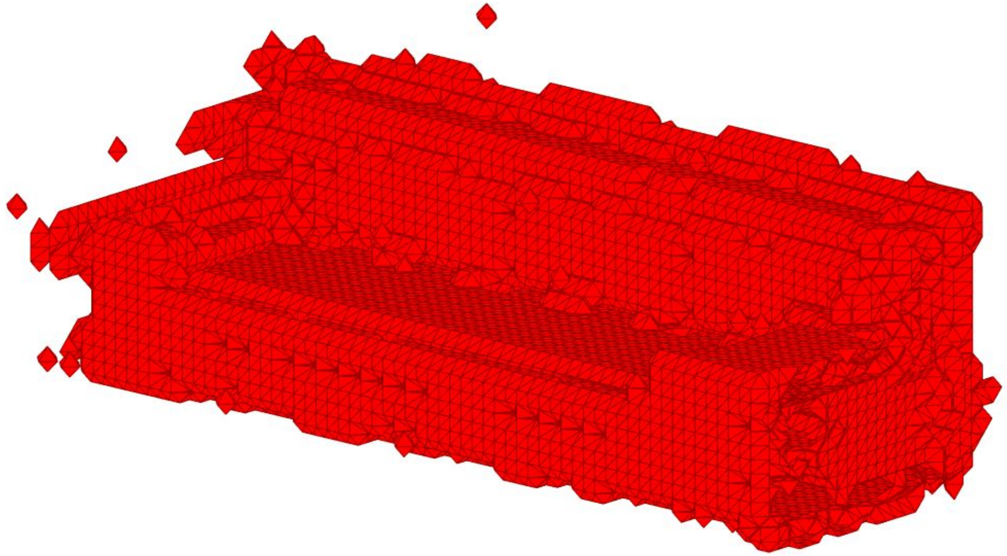


After 500 epochs



After 500 epochs





Notice

- the holes on some edges are not errors due to training but to the voxel transformation
- there might be results at the end of the batch even better

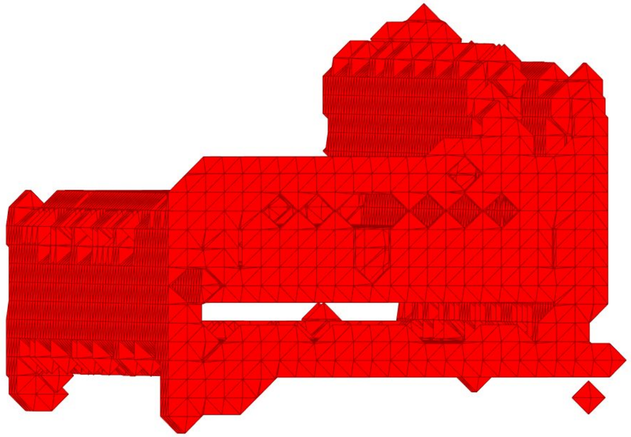
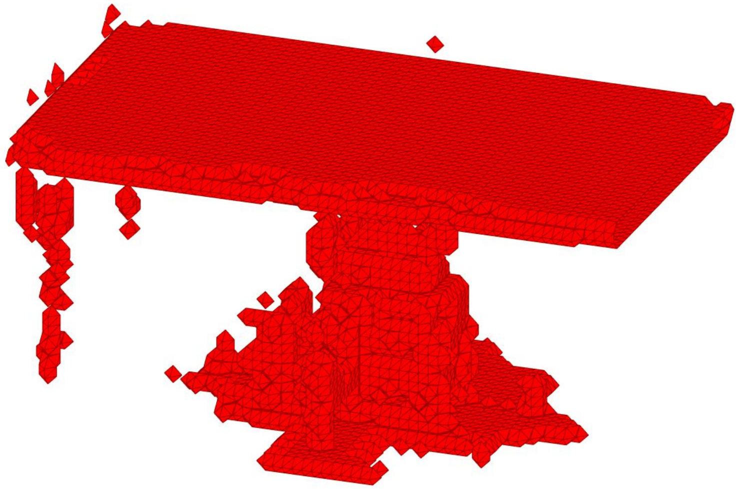
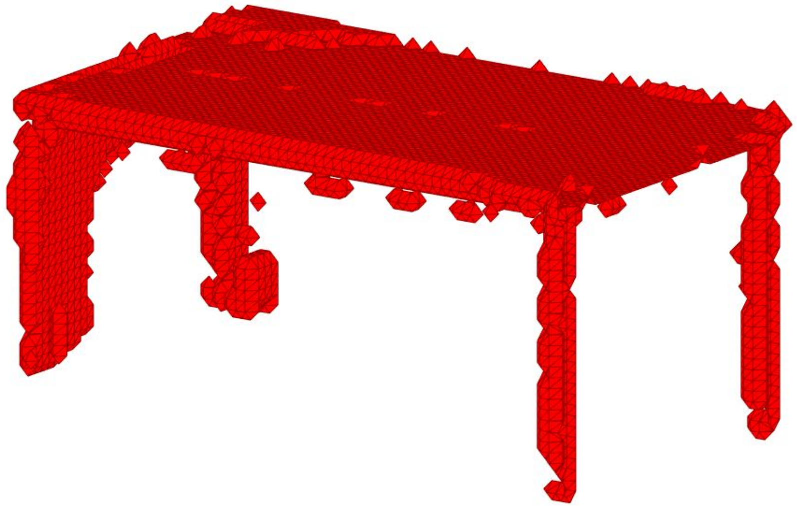
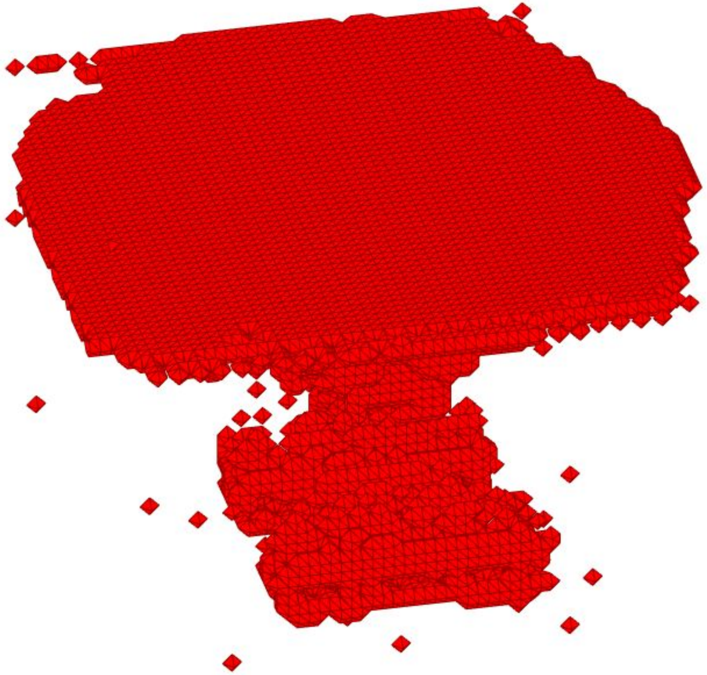


Table (492 shapes)

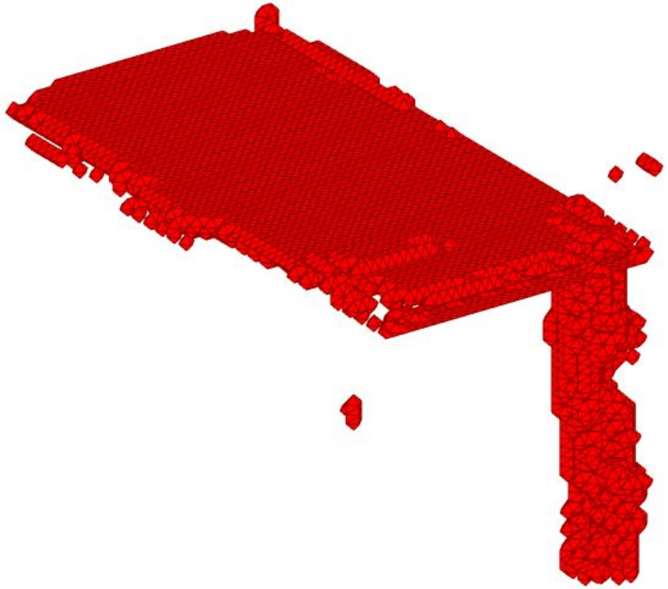
- batch size of 16 (reduced for memory reasons)
- the dataset is challenging
- learns a good variety (single leg - 4 legs) but there is some noise





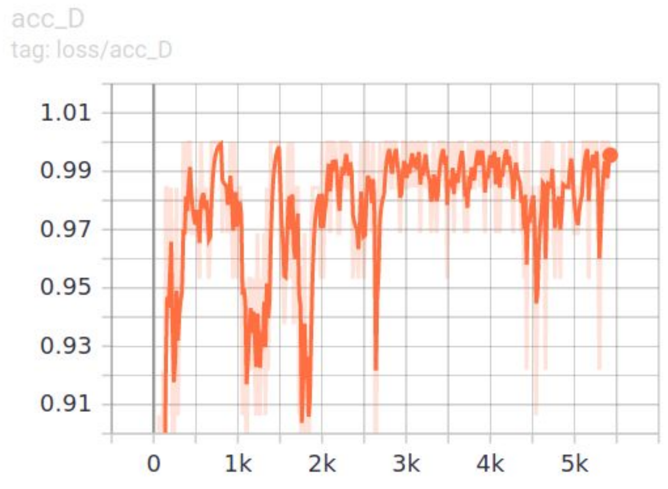


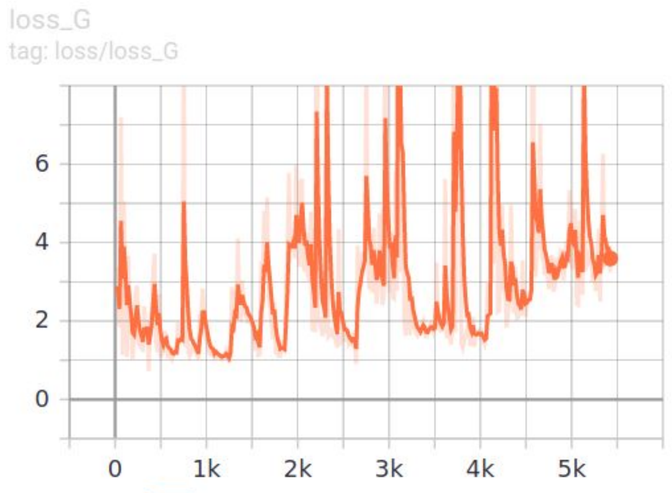
Some failures



Trainings on the Dresser (280 shapes)

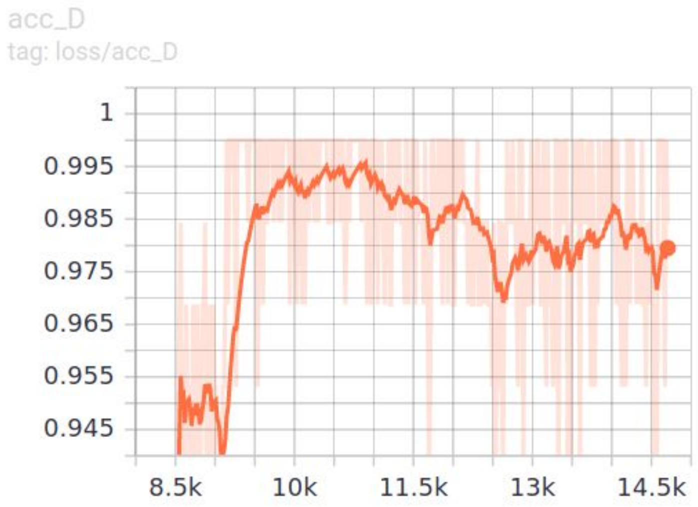
- dresser very restricted dataset
- produces cube-like shapes
- the Discriminator performs too well from the beginning with no possibilities for the Generator to fool D

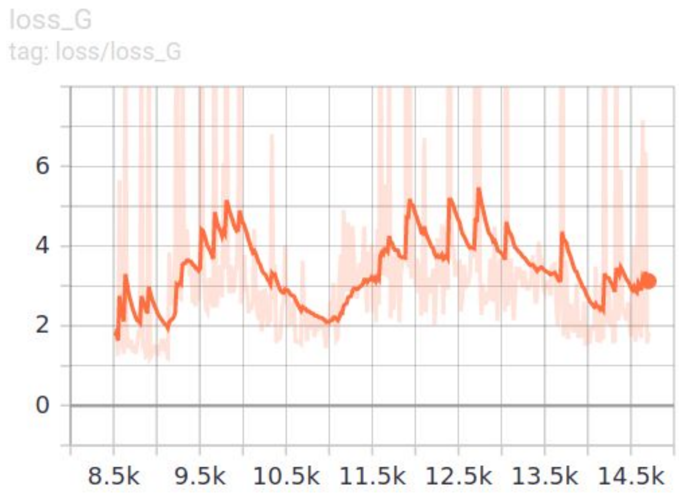




Training on the Monitor (500 shapes)

- → at the beginning produces cube-like shapes
- → later the shapes produced are empty
- → very small area compared to the 64x64x64 cube
- → the Generator is not able to learn, due to the high accuracy of D
- 1000 epochs (#iterations / (dataset size/batch size))

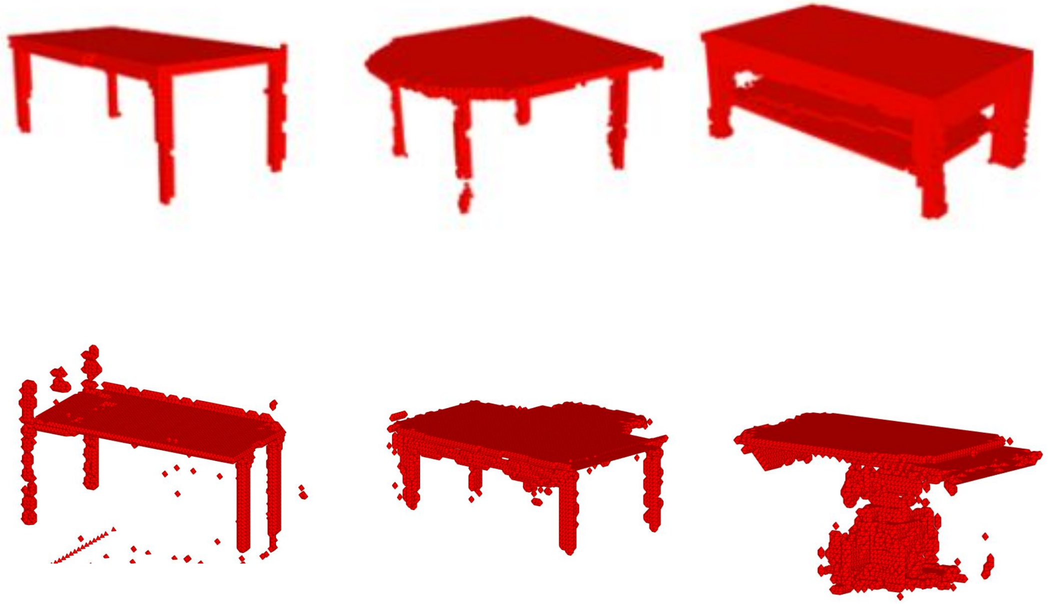




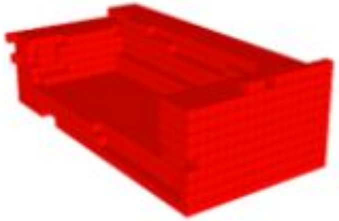
Evaluation

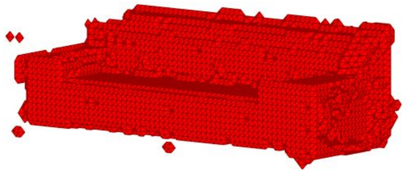
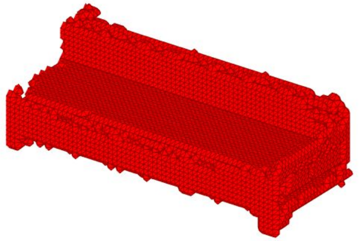
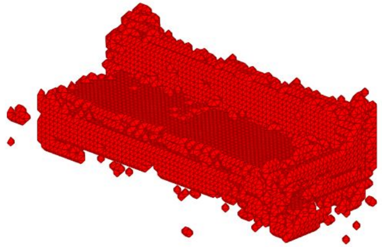
- Human evaluation to evaluate the realism of the shapes produced
- Comparison with the results of the paper
- Comparison with the nearest neighbor to check if the network is just copying from the dataset or if it is producing new samples
    - function implemented in Matlab to get the NN:
        - $\arg\min_{X_i} \| X_i - G(z) \|_1$

Paper results comparison



Paper results comparison



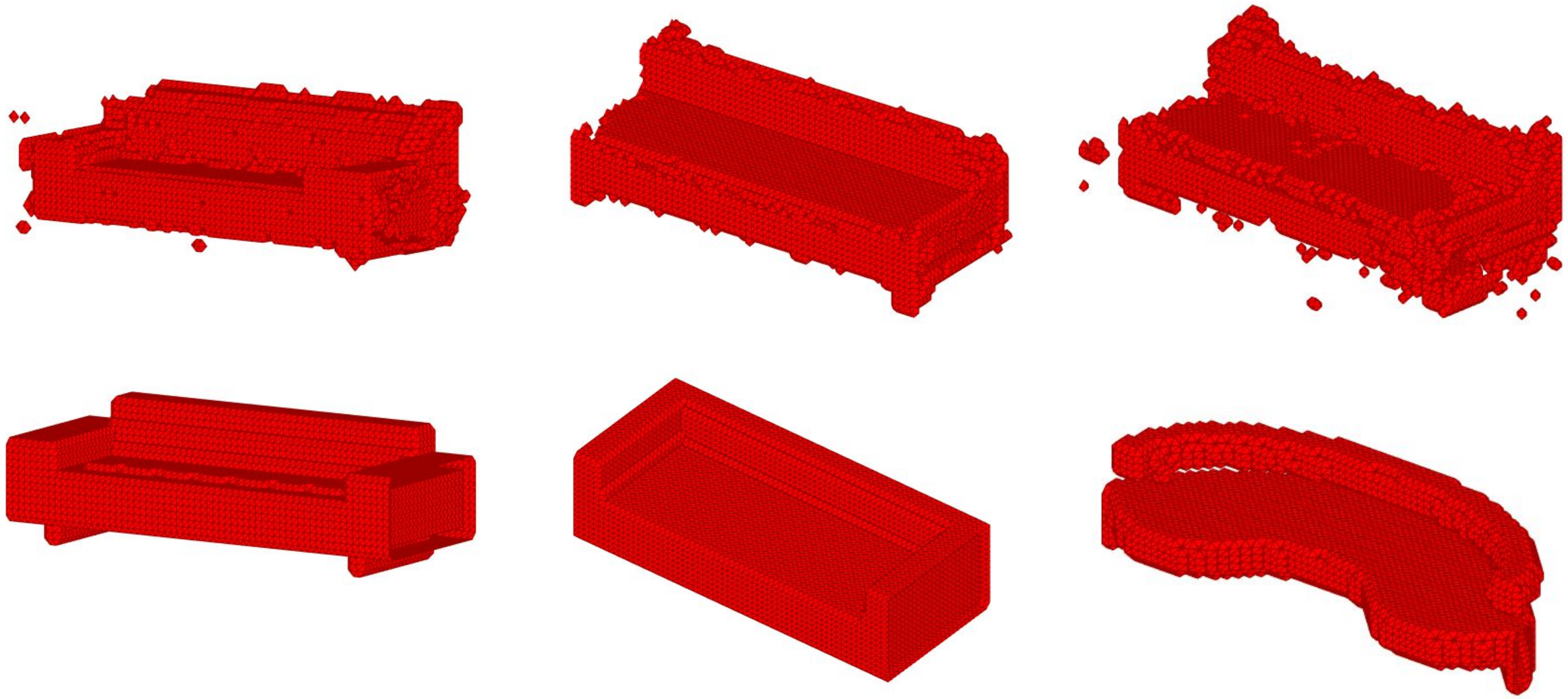
Dataset Comparison

ShapeNet has about 53.000 shapes and 12 categories

ModelNet10 has just about 4900 shapes and 10 categories

→ 10 times less !

Nearest Neighbors



Nearest Neighbors