StyleTransfer - Mesh deformation

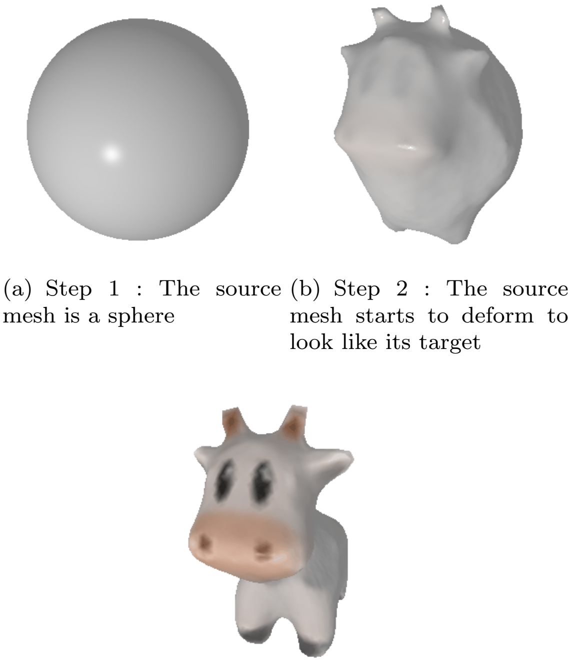
Enguerrand DE SMET · **Laurine LAFONTAINE**

Abstract Mesh is an important and powerful type of data for 3D shapes and widely studied in the field of computer vision and computer graphics thus its data is complex and irregular. With the introduction of machine learning, a lot of programs for image treatment has been emerging. These algorithms make predictions in 2D, ignoring the 3D structure of the world. Working with 3D could be giving a new axis on what we can do with deep learning as we could modify object in world-dimension. After working with 2D deep learning structure, we wanted to give a shot at 3D deep learning that remained relatively under explored for us. Our aim is to deform a source mesh to form a new mesh, based on a target, which is referred as morph target animation, using 3D loss functions. The final goal would be to apply an artistic style to meshes after processing. Our system, called StyleTransfer is based on the library PyTorch3d [1], which is used for deep learning with 3D data.

Keywords reinforcement learning · pytorch · mesh · deformation · 3d model · morph target animation · papers

1 Introduction

Nowadays, video game companies, VFX and even applications in virtual and augmented reality create a demand for rapid creation and easy access to large sets of 3D models. Same as the video game industry, who always want to work faster. In order to satisfy this demand, artists need to edit or deform existing 3D models based on a reference. This process is known as morph targeting. With the introduction of deforming meshes using deep learning, they could work faster and more efficiently. Significant advances in 3D shape understanding have been made with neural networks.



(a) Step 1 : The source mesh is a sphere

(b) Step 2 : The source mesh starts to deform to look like its target

(c) Step 3 : The source mesh is completely deformed to the shape of the target

Fig. 1 Steps of the deformation

Artists are able to form high-level interpretations of 3D objects. How machines can be as good as human ? If we want to use machines as developing a modeling-assistant tool, they must be as efficient as humans (or at least close to). Intelligent agents needs to learn 3D modeling like human does. First, the shape must be seen as a basic geometric primitives to approximate the shape. Then, the mesh needs to be edited, based on the primitives using specific operations to create more detailed geometry. The algorithm needs to learn to take a primitive as input an edit it to have the wanted mesh.

Deforming meshes can be used to represent a collection of objects with different shapes and poses. Even years after, there is still challenging problems to solve.

Enguerrand DE SMET
E-mail: desmet.enguerrand@gmail.com

Laurine LAFONTAINE
E-mail: laurine.lafontaine@outlook.fr

Indeed, meshes may differ in their number of vertices and faces, and their topology. Such heterogeneity makes it difficult to efficiently implement batched operations on 3D data using operators provided by standard deep learning toolkits like PyTorch and Tensorflow. In our case study, as we use the library PyTorch3D, the mesh is a *load_obj* which is a file format containing the description of a 3D geometry. In this paper, we have selected a few of the best-ranked 3D networks and we draw on state-of-the-art methods for 3D shape prediction to build a mesh deformation system.

2 Related work

With an impressive breakthrough made during the 2012 ILSVR challenge[2], machine learning based methods became the new go-to for image restoration and segmentation algorithms. We based our work on some papers that use 3D shapes.

2.1 Variational Autoencoders (VAE) for Deforming 3D Mesh models

This paper suggests a new framework called Mesh Variational Autoencoders[3], which aim is to produce a generative model capable of analysing model collections and synthesising new ones. It needs to explore the latent space behind deforming 3D shapes and with all these information, is able to generate new models not existing in the original dataset.

Instead of using representations with image-like connectivity, the team that works on it uses a surface representation called RIMD (Rotation Invariant Mesh Difference) [4], which is a rigid motion invariant mesh representation based on discrete forms defined on the mesh. A RIMD representation is linearly combined and the mean reconstruction error is extremely small. It can easily represent deformations, along with a variational autoencoder (VAE) [5]. A VAE is a particular type of autoencoder whose training is regularised to avoid overfitting and ensure that the latent space has good properties that enable generative process.

Their algorithm can have multiple application : shape generation, shape interpolation, shape space embedding or even shape exploration. It is a fully-connected network, along with a simple reconstruction loss based on Mean Square Error (MSE). It can be used with small number of training and it is trained using a collection of 3D shapes with the same connectivity.

This algorithm needs to have some pre-processing with extraction features, that are represented with the RIMD mesh feature, which can be the output of the mesh VAE. The mesh model can be reconstructed efficiently with some optimisation.

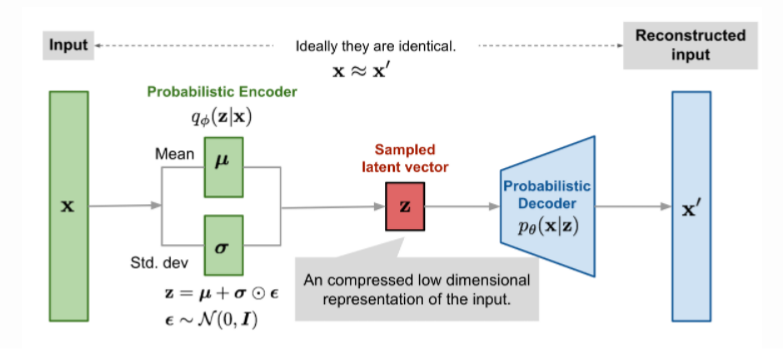

Input
Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Reconstructed input
Probabilistic Encoder
$q_\phi(\mathbf{z}|\mathbf{x})$
Mean
$\boldsymbol{\mu}$
Std. dev
$\boldsymbol{\sigma}$
$\mathbf{x}$
Sampled latent vector
$\mathbf{z}$
Probabilistic Decoder
$p_\theta(\mathbf{x}|\mathbf{z})$
$\mathbf{x}'$
$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$
$\boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I})$
An compressed low dimensional representation of the input.

Fig. 2 VAE network

For the activation function, they use an hyperbolic tangent (tanh). They state stay their encoder aims to map the posterior distribution from datapoint x to the latent vector z, and the decoder produces a plausible corresponding datapoint x from a latent vector z. All of this allows to generate new models, which are generally plausible.

2.2 Accelerating 3D Deep Learning with PyTorch3D

PyTorch3D is an open-source library of modular, efficient, and differentiable operators for 3D deep learning that is built on PyTorch. It improves the state-of-the-art for unsupervised 3D mesh and point cloud prediction from 2D images on ShapeNet[6]. As PyTorch3D is open-source, it helped a lot to accelerate research in 3D deep learning. This is thanks to this library that we were able to do our algorithm.

PyTorch3D allow to easily handle 3D data structures, to manage batches of meshes and point clouds which allow conversion between different tensor-based representations (list, packed, padded) needed for various operations. This library comes with multiple 3D operators such as Chamfer loss, Graph convolution and K Nearest Neighbours.

Thanks to this library, we can also be able to do some differentiable rendering, which is relatively new and exciting for research. This renderer is based on CUDA. Speaking of it, we can have a differentiable mesh renderer that propagate gradients backward from rendered images to scene information, allowing rendering to be embedded into deep learning pipelines. This renderer comes with a rasterizer that uses a camera to transform meshes from world to view coordinates and shaders that consume the Fragment data produced by the rasterizer and compute pixel values of the rendered image. There is also a differentiable point cloud renderer that follows the same design as the mesh renderer.

Finally, using PyTorch3D made improvements in speed and memory and is up to 10 times better compared with original PyTorch operator.

2.3 Mesh R-CNN

Based on Mask R-CNN[7], which detects objects in an image while simultaneously generating a high-quality segmentation mask for each instance. Based on this 2D recognition algorithm, the Mesh R-CNN team done their algorithm, augmenting it with a mesh prediction branch that outputs an high-resolution triangle meshes.

Mask R-CNN inputs a single RGB image and outputs a bounding box, category label, and segmentation mask for each detected object. The image is first passed through a backbone network (ResNet-50-FPN); next a region proposal network (RPN) : gives object proposals which are processed with object classification and mask prediction branches. Mesh R-CNN[10] is based on Detectron2[8] and PyTorch3D[1].

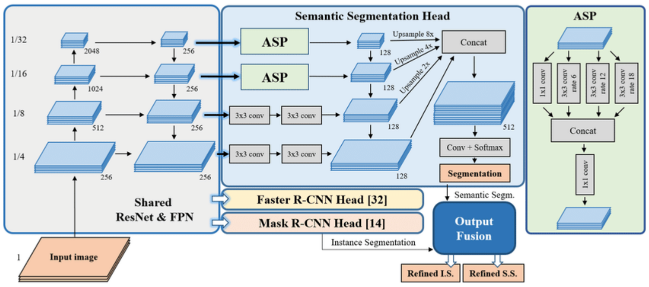


Fig. 3 ResNet-FPN network

Earlier papers that works on mesh prediction with deep networks has been constrained to deform from fixed mesh templates, limiting them to fixed mesh topologies. To overcome this, they first predicts coarse voxels, which are refined for accurate mesh predictions.

The Mesh R-CNN team outlined a framework that takes as input a real-world RGB image, detects the objects within the image, and outputs a category label, bounding box, segmentation mask, and a 3D triangle mesh, giving the full 3D shape of each detected objects. Predicted meshes must be able to capture the 3D structure of diverse, real-world objects. Predicted meshes ought to subsequently powerfully shift their complexity, topology, and geometry.

A 3D shape is created with a novel mesh predictor, composed of a voxel branch and a mesh refinement branch. The voxel branch first estimates a coarse 3D voxelization of an object, which is converted to an initial triangle mesh. The mesh refinement branch then alters the vertex positions of this initial mesh using a sequence of graph convolution layers operating over the edges of the mesh.

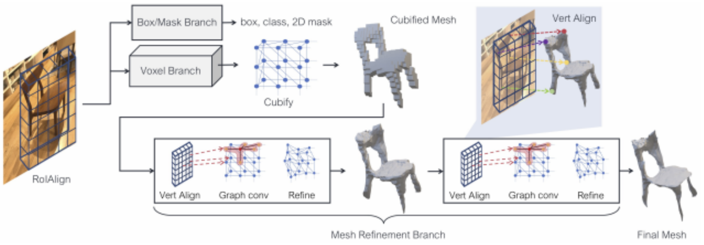


Fig. 4 Mesh RCNN network

Concerning the mesh predictor it consists of the voxel branch and mesh refinement branch : receives convolutional features aligned to an object's bounding box and predicts a triangle mesh giving the object's full 3D shape. Each predicted mesh must have instance-specific topology (number of vertices, faces, connected components) and geometry (vertex positions) The output of this predictor is a triangle mesh $T = (V, F)$ for each object.

As they work with voxels, they need to cubify in order to convert shapes to meshes. Their algorithm is based on a lot of steps which can now be bypassed.

2.4 Modeling 3D Shapes by Reinforcement Learning

Reinforcement learning differs from the supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task. Within the nonappearance of a training dataset, the network need to learn from its experience.

- Agent :
  - The learner and the decision maker.
- Environment :
  - Where the agent learns and decides what actions to perform.
- Action :
  - A set of actions which the agent can perform.
- State :
  - The state of the agent in the environment.
- Reward :
  - For each action selected by the agent the environment provides a reward. Usually a scalar value.

Modeling 3D Shapes by Reinforcement Learning[9] introduce a two-step reinforcement learning for shape analysis and geometry editing. Their agents can produce regular and structure-aware mesh models to capture the fundamental geometry of 3D shapes.

They worked on primitive-based shape abstraction, which use the prim-Agent to understand the part-based structure of a shape by collaborating with the environment. Agents continually change the primitives based on the feedback to achieve the goal. Iteratively, each primitives are visited, to test all the potential actions for the primitive and execute the one which can obtain the best reward. Amid the primary half of the method, they do not consider any erase operations but alter the corners. This is often to empower all the primitives to fit the target shape first. At that point within the second half, they autorize primitives erasing to dispose of repetition.

Their algorithm edit mesh editing by edge loops so they can edit a group of vertices and control an integral geometric unit instead of editing each vertex separately, which preserves the mesh regularity and improves the efficiency. Iteratively, each edge loop are visited to test all the potential actions for the edge loop and execute the one which can obtain the best reward.

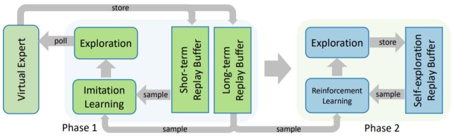


Fig. 5 Modeling 3D shapes with reinforcement learning training flow

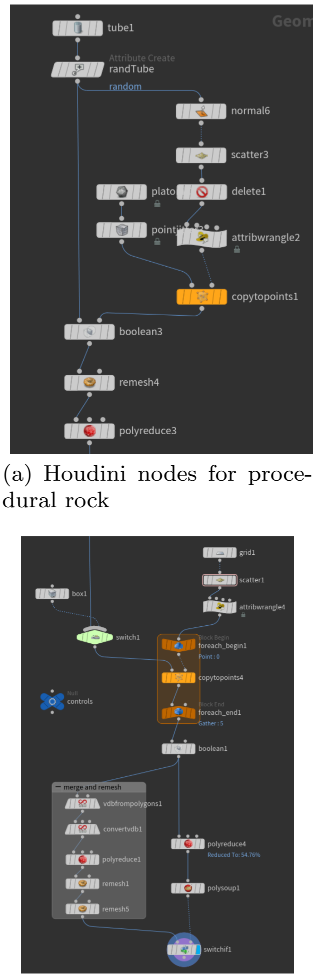
The network of this algorithm is based on the Double DQN Reinforcement learning[11] by self-exploration. As they are limited with data, they use dataset aggregation, which need to be supervised by a human but is efficient for the lacking data. Agents retain a part of the memory from the expert but also gain new experiences by their own exploration. This allows the agents to potentially compare the actions learned from the expert and this is how they succeed to have an efficient algorithm.

3 Pre-processing

As we wanted to appropriate our algorithm, we decided to generate our own dataset.

3.1 Dataset generation with Houdini

In order to test these deformation algorithms we generated some meshes to work with. That's why we decided to use the well-known software Houdini which allow us to generate procedurally both input and target meshes for our deformation algorithm.



(a) Houdini nodes for procedural rock

(b) Houdini nodes : scatter and switch primitives

Fig. 6 Houdini steps

We use the nodal system of Houdini to generate rocks from several rectangles resulting from a random scattering step. With this system, we can expose various parameters in order to export both the rectangles and the corresponding rocks. Therefore, it will be possible to quantify the quality of the deformation.

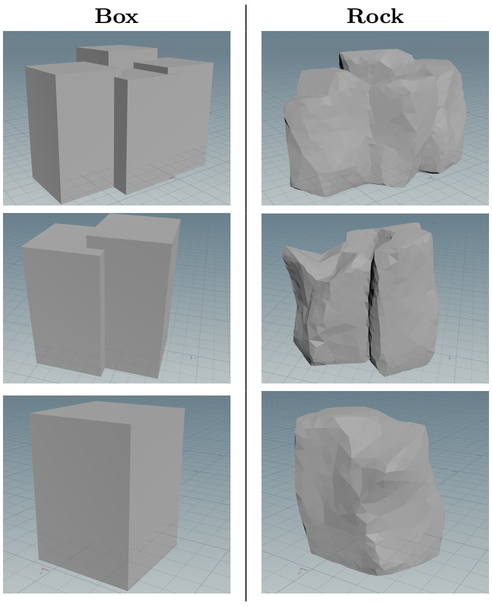


Fig. 7 Meshes generated with Houdini

3.2 Subdivide meshes

An additional step was necessary so that the algorithm has enough vertices on the source mesh. We used a Python library called Trimesh to subdivide the source mesh faster so that it can be deformed properly afterwards. We first compute the max edge of our mesh and we do a re-triangulation of existing meshes by subdividing our mesh until every edge is shorter than a the max edge length.

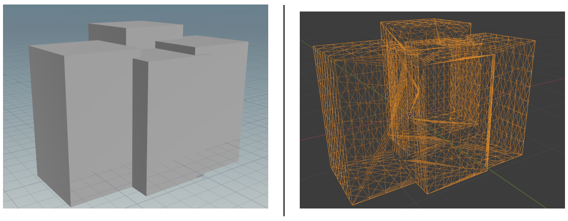


Fig. 8 Mesh subdivided

4 Method

After all the pre-processing, we can start using our data. As said earlier, our algorithm is based on PyTorch3D to be efficient.

4.1 Primitive-based shape

The idea is to reproduce a complex mesh by deforming primitives that give a rough volume of the space where the target mesh should be generated. The algorithm reads in input some rectangles primitives, which is our source mesh from a *.obj* file extension using the *load_obj* method. We also load the target mesh.

For both these objects, we store our vertices and faces. Vertices is a *FloatTensor* of shape $(V, 3)$ where V is the number of vertices in the mesh. *Faces* is an object which contains the following *LongTensors*, *verts_idx*, *normals_idx* and *textures_idx*. We scale, normalize and center the target meshes and we construct a meshes structure for the source and target mesh.

This one will then generate a second subdivided mesh allowing to have a sufficient number of vertices to work with.

4.2 Agent Training Algorithm

The algorithm will then sample uniformly a certain number of points from the surface of the source mesh (5000k) and will apply successive deformations in order to minimize a weighted error function well chosen from different distance measures (chamfer loss, normal consistency, ...) between the source and target mesh.

As the optimizer for our training pass, we use a stochastic gradient descent, which is an iterative method for optimizing an objective function with suitable smoothness properties. It is a modification of the gradient descent, where is calculated the gradient using just a random small part of the observations instead of all of them. In some cases, this approach can reduce computation time. The gradients are calculated and the decision variables are updated iteratively with subsets of all observations, called minibatches.

$$\begin{aligned}\text{Loss} = & w_{\text{c}} \ .\text{chamfer}_{\text{loss}} + w_{\text{n}} \ .\text{normal}_{\text{loss}} \\ & + w_{\text{e}} \ .\text{edge}_{\text{loss}} + w_{\text{l}} \ .\text{laplacian}_{\text{loss}}\end{aligned} \tag{1}$$

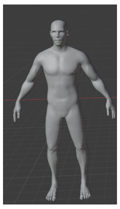
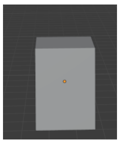
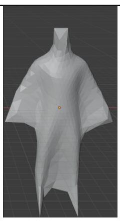
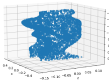
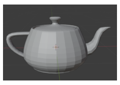
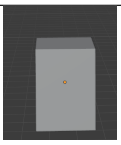
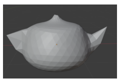
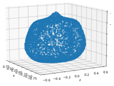
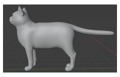
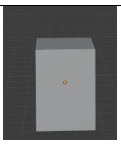
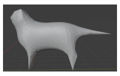
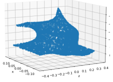
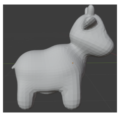
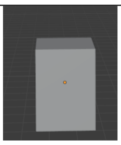
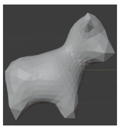
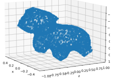
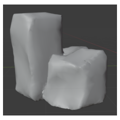
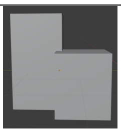
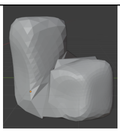
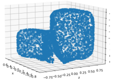
Here's our parameter for the loss equation : $w_c = 1.0$, $w_e = 1.0$, $w_n = 0.01$, and $w_l = 0.1$.

The network learns to deform the source mesh by offsetting its vertices and observe errors from the sample points. The shape generated from the deformation parameters is equal to the total number of vertices of the source mesh, which is why we had previously subdivided the source mesh.

The last step consists in recovering the vertices and faces of the final predicted mesh, normalising to the scale of the source mesh to return to the original target size and finally saving the generated mesh.

5 Results

We provided some results that mix our own data generation and open-source 3D models.

Target	Source	Output	Point cloud
			
			
			
			
			

Each time the algorithm is running, the loss is calculated. For the last example in the table, the loss value is 0.008 for 5000 epochs, which is a really small values. As it won't be readable, we won't write every loss that we had, but the average is around 0.015.

5.1 Distance between two surfaces

When we want to check if an image is well-denoised, we use SRN. Likewise, we wanted to see if our output was similar to our target. For that, we use KDTree to calculate the average thickness between two surfaces. We can compute the thickness between the two surfaces using a few different methods, we focused on the nearest neighbour distance method to compare the distance between each point of the upper surface and the nearest neighbour of the lower surface.

A k-d tree is a space partitioning data structure that allows points to be stored, and searches to be made more quickly than by linearly traversing the point array. each of these nodes represents an axis-aligned hyperrectangle. Each node specifies an axis and splits the set of points based on whether their coordinate along that axis is greater than or less than a particular value. This won't be the exact surface to surface distance, but it will be noticeably faster than a ray trace, especially for large surfaces.

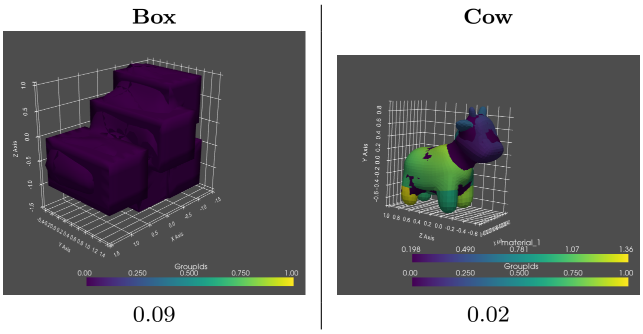


Fig. 9 Distance between target mesh and output

6 Conclusion

In this paper, we introduce various methods of mesh deformation. We wanted to explore machine learning and more specifically, how to teach machines to act like a 3D artist. We learned a lot of things into the process of meshes creation. In order to mimick artists in the best way, our algorithm is based of a source-to-target training. Given an target reference, our input meshes (a primitive) learns to deform to fit the target. To effectively train these modeling agents, we used PyTorch3D. Ultimately, we hope to continue this project by fitting textures onto our deformed meshes.

References

1. Nikhila Ravi. Jeremy Reizenstein. David Novotny. Taylor Gordon. Wan-Yen Lo. Justin Johnson. Georgia Gkioxari, Accelerating 3D Deep Learning with PyTorch3D, Facebook AI Research (2020)
2. Alex Krizhevsky. Ilya Sutskever. Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks (2012)
3. Qingyang Tan. Lin Gao. Yu-Kun Lai. Shihong Xia, Variational Autoencoders for Deforming 3D Mesh Models (2018)
4. Lin Gao. Yu-Kun Lai. Dun Liang. Shu-Yu Chen. Shihong Xia. Efficient and Flexible Deformation Representation for Data-Driven Surface Modeling (2016)
5. Diederik P. Kingma. Max Welling. Auto-Encoding Variational Bayes (2014)
6. Angel X. Chang. Thomas Funkhouser. Leonidas Guibas. Pat Hanrahan. Qixing Huang. Zimo Li. Silvio Savarese. Manolis Savva. Shuran Song. Hao Su. Jianxiong Xiao. Li Yi. Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository (2015)
7. Kaiming He. Georgia Gkioxari. Piotr Dollàr. Ross Girshick. Mask R-CNN (2018)

8. Yuxin Wu. Alexander Kirillov. Francisco Massa. Wan-Yen Lo. Ross Girshick. Detectron2 (2019)
9. Cheng Lin. Tingxiang Fan. Wenping Wang. Matthias Nießner, Modeling 3D Shapes by Reinforcement Learning (2020)
10. Georgia Gkioxari. Jitendra Malik. Justin Johnson, Mesh R-CNN, Facebook AI Research (2020)
11. Hado Van Hasselt. Arthur Guez. David Silver. Deep Reinforcement Learning with Double Q-learning, Google DeepMind (2015)