3D Shape Modeling

1 Introduction

This practical work is on 3D shape modeling using binary silhouettes images. Given such 2D silhouettes we can estimate the visual hull of the corresponding 3D shape. The visual hull is, by definition, the maximal volume compatible with a given set of silhouettes and it corresponds to the intersection of the 3D visual cones defined by the silhouette regions.

In the first part, the visual hull will be estimated by an approach called voxel carving. The idea is to consider a grid of elementary cells in 3D and to *carve* the cells that do project outside the silhouettes in the images. In the second part, a multi-layer perceptron (MLP) will be trained to learn the shape occupancy in 3D, as defined by the silhouettes, in the form of an implicit function $f(x, y, z) = 0, 1$ with $(x, y, z) \in \mathcal{R}^3$. The objective is to investigate the ability of a MLP to learn the 3D shape with a low dimensional representation that is the network itself.

In the TP folder you have the file *al.off* of the 3D model Al, which contains a mesh description of the geometry (see Fig. 1). You also have the script *show_mesh.py* that visualizes the geometry in a 3D viewer. Try running

```
python show_mesh.py al.off
```

in the terminal. You should be able to see the 3D geometry.

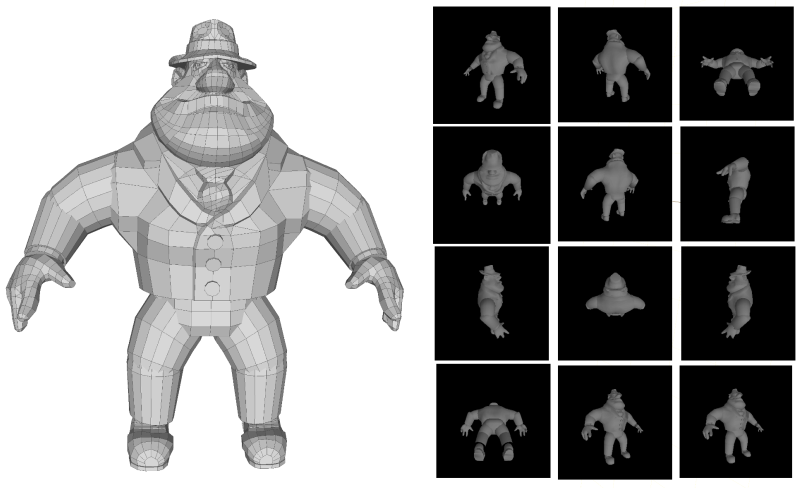

Figure 1: The 3D shape (al.off in the archive) and the 12 image projections.

sergi.pujades@inria.fr

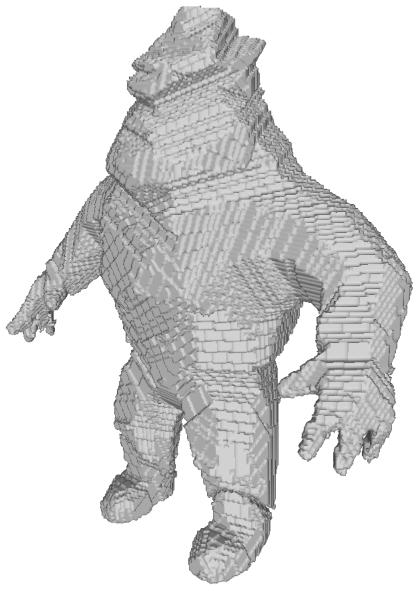

Figure 2: The visual hull with a grid of size $300 \times 300 \times 150$.

In the *images* folder, you have the projections of the geometry in each of the 12 different viewpoints (image0.pgm, ..., image11.pgm). These can be used as binary silhouettes, by checking if a pixel is totally black (intensity equal 0), or not.

2 Voxel Carving

In this part the objective is to build the 3D voxel representation of the visual hull of Al as defined by the 12 silhouettes. At the end of this part, you should get a 3D representation similar to the image in Fig. 2.

Open the file *voxcarv3D.py* which contains the program to be completed. At the beginning of the file the calibration matrices for the 12 silhouette cameras are stored in the array $calib$. Each matrix corresponds to a 3×4 linear transformation from 3D to 2D. Then the voxel 3D coordinate arrays: X, Y, Z and the associated occupancy grid $occupancy$, that shall be filled with 0 or 1, are defined. Note that the grid resolution can be modified with the parameter resolution. If you program is too slow, you can reduce the resolution. The code can be run with

```
python voxcarv3D.py
```

Once the algorithm is run, the program uses the marching cube algorithm to transform the occupancy grid into a 3D mesh that can be exported in a standard format. The resulting mesh *alvoxels.off* will be saved in the output folder. The numeric occupancy grid will also be exported into a numpy array *occupancy.npy*. The current program invites you to visualize the obtained results by running

```
python show_mesh.py output/alvoxels.off
```

2.1 Implementation

1. Complete the program so that the voxels that projects within the silhouette 1 (image1.pgm) are preserved. These voxels define the visual cone associated to image1.

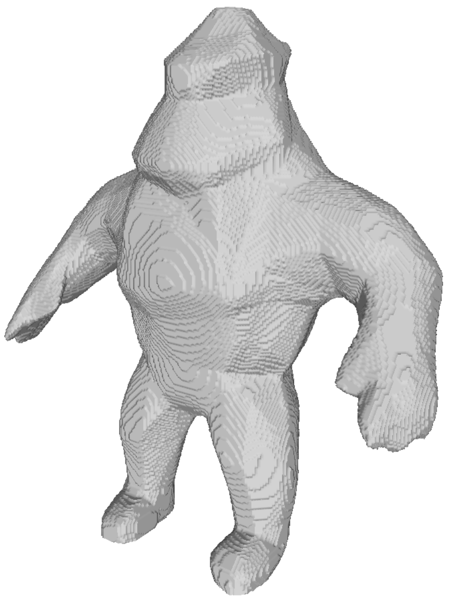

Figure 3: The neural implicit function trained with the $300 \times 300 \times 150$ regular grid points.

2. Complete the program to account for the 12 images and to preserve then only the voxels that belong to the visual hull. Note that projections can be performed in an efficient way using numpy array operations.
3. Open both models (al.off and alvoxels.off) side by side and discuss their differences.

3 Neural Implicit Representation

Now we will explore how a MLP can be trained to learn the 3D occupancy defined by the 12 silhouettes of Al. Open the file *MLPimplicit3D.py* which contains the corresponding program. The principle here is to use a set of 3D points with known occupancy to train a MLP that considers as input 3 coordinates x, y and z and outputs the occupancy, 0 or 1, at the 3D location (x, y, z). We will start with the stored occupancy from the previous part.

3.1 Programming Strategy

First look at the program (no need to run or train) and answer the following questions:

1. Draw the architecture of the MLP (input, output, layers, activation function).
2. How are the training data (X, Y, Z, occupancy) formatted for the training ?
3. In the training function (nif_train), what is the loss function used ?
4. Explain the normalization used to weight losses associated to inside and outside points in the training loss ?
5. During the training how is the data organized into batches ?
6. What does the function binary_acc evaluate ? Is it used for the training ?

7. How is the MLP used to generate a result to be visualized ?

3.2 GPU connection and environment

To run and edit the program, we need a GPU computer. The practical takes place on the Ensimag / Grenoble INP educational GPU cluster and it's user manual can be found in this link. In a nutshell, first you log on the slurm server:

```
ssh -YK nash.ensimag.fr
```

Once connected, you need to make sure your environment has all the python packages installed. A virtualenv with all needed python dependencies is available. You can activate it by typing the following command in your terminal

```
. /matieres/5MMVORF/venv/bin/activate
```

Then, you can execute pytorch scripts by prefixing them with srun command as follows:

```
srun --gres=shard:1 --cpus-per-task=8 --mem=12GB --x11=all python3 train.py
```

In our case, you should use the *MPLImplicit3D.py* file.

3.3 Program Running and Editing

Run the program once and

1. inspect the computed 3D model (*output/alimplicit.off* and compare it to the original and the voxel carved models.

Then

1. Add a line that uses torch.save to save the trained model and run the code again.
2. What is the memory size of the MLP ? how does it compare with: (i) A voxel occupancy grid; (ii) The original image set plus the calibration ?

Then play with the program:

1. Instead of using a regular grid of points for the training modify your program to generate random points $Xrand, Yrand, Zrand$ in 3D. Note that the MLP can still be evaluated on the regular grid points X, Y, Z as before for comparison purposes.
2. The difference between the number of outside and inside points is compensated with a weighting scheme during the training. A more efficient strategy for the training is to reduce the set of outside points before the training. Propose and implement such a strategy.
3. Modify the MLP architecture to see the impact of increasing or reducing the number of parameters through: (i) the number of layers and (ii) the layer dimension.