TP
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.

First download the archive 3DShapeModeling.tar. We will consider the 3D shape Al (see the file Al.off in the archive) that is observed from 12 different viewpoints. The corresponding binary silhouettes are stored in the image files image0.pgm, ..., image11.pgm. The archive provides 2 python programs to be completed and which correspond to the 2 parts mentioned before.

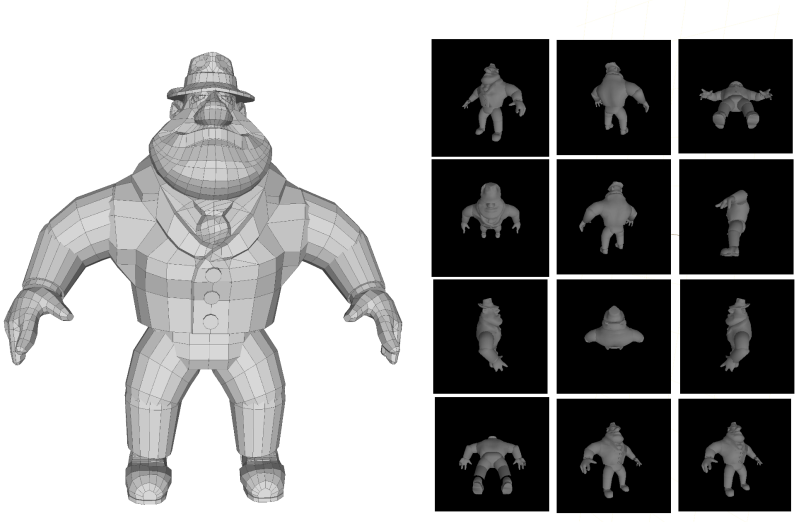

Figure 1: The 3D shape (Al.off in the archive) and the 12 image projections.

Jean-Sebastien.Franco@inria.fr,Edmond.Boyer@inria.fr

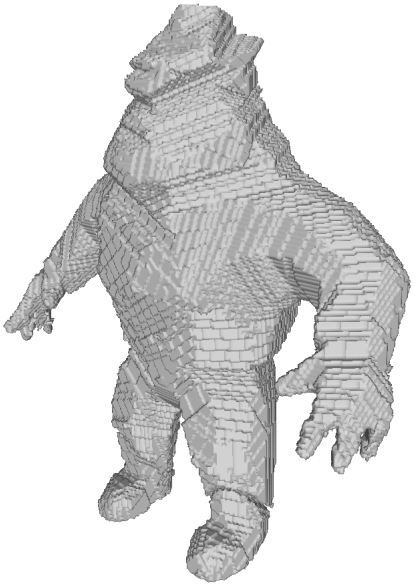

Figure 2: The visual hull with a grid of size $300 \times 300 \times 150$.

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. To this purpose open the file voxcarv3D.py which contains a 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. In order to visualize the occupancy grid the program uses the marching cube algorithm to transform the occupancy grid into a 3D mesh that can be exported in a standard format, in practice the off format in the program (use meshlab to display the 3D mesh).

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.

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.

Jean-Sebastien.Franco@inria.fr,Edmond.Boyer@inria.fr

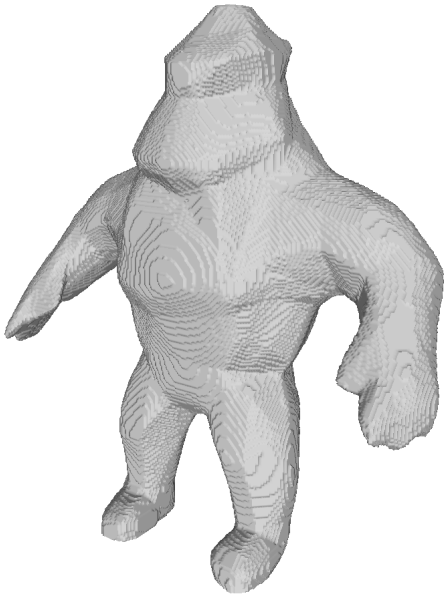

Figure 3: The neural implicit function trained with the $300 \times 300 \times 150$ regular grid points.

3 Neural Implicit Modeling

We will consider in this part a MLP that will 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 occupancies 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). As in the previous part, we need to estimate the occupancy at various locations to build the training point set. The program builds therefore on what was done in the previous part.

3.1 Programming Strategy

First look at the program 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 ?
8. 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 ?

Jean-Sebastien.Franco@inria.fr,Edmond.Boyer@inria.fr

3.2 Program Editing

Then complete the program:

1. In the main function, the first part is supposed to generate the training set. The grid points X, Y, Z are defined but, as before, their occupancies must be estimated. Complete with the code from the previous program and test it.

2. 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.

3. 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.

4. 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.

Jean-Sebastien.Franco@inria.fr,Edmond.Boyer@inria.fr