3D Feature Point Learning for Fractured Object Reassembly (Instructions)

Du Chaoyu (chaoyu.du@arch.ethz.ch)
Ulrich Steger (ulsteger@student.ethz.ch)
Eshaan Mudga (emudgal@student.ethz.ch)
Florian Hürlimann (fhuerliman@student.ethz.ch)

14.06.2021

1 Data Generation

All code and data can be found: https://github.com/duchaoyu/3d$_fracture_reassmbly.git$.

1.1 Blender and add-on installation

Generation the fracture pieces using 3D Voronoi diagram and fractals requires blender 2.79 and the Add-on file ”000_ShapetoMESH.py” and ”000_fracture_helper.py”.

1. Install Blender 2.79 from https://drive.google.com/drive/folders/0Bwkun-jSiZ-RNDlRS1FURUV0V28. Choose the version that fits your operating system and download it.
2. Manually extract Blender to the desired folder, where you can double-click the executable to run Blender.
3. Go to Blender's user preferences and open the Addons tab.
4. click Install add-on from file at the bottom
5. Navigate to ”000_ShapetoMESH.py” and ”000_fracture_helper.py” and select it
6. The addon is then installed, however not automatically enabled.
7. check the box next to the add-on's name to enable it.

1.2 Fracture modifier

Here is the basic workflow of fracturing a mesh using Blender. The fracture and output process can also be automated by running ”010_blender_fracture.py” and ”011_blender_output.py”.

1. Select a mesh to fracture. This mesh should be non-manifold and water-tight.
2. Go to Physics Tab in Property Editor
3. Select Enable physics for: Fracture
4. Select Voronoi + fractals algorithm and set the proper parameters.
5. Click Execute Fracture and the fracture outcome will show up.
6. Click Convert Objects to separate the fracture pieces.
7. Export the separated fracture pieces in .obj format.

1.3 Clean up input mesh

Note that if you download a mesh, it's recommended to check the mesh before execute the fracture modifier.

1. Uniformity. A uniform mesh gives better result in the fracture process. If the density varies in the mesh, more Voronoi generating points will be placed in the denser area by default. Thus, the size of the fracture pieces will not be ideal. Re-mesh turns the mesh to a more smooth and uniform mesh.
2. Water-tightness. To execute the fracture algorithm properly, the input mesh should be manifold and consist of one closed surface. Separate mesh by loose parts, delete the un-necessary parts and fill the holes.
3. Proper density. The mesh should contain proper number of vertices per unit area. Lower density might leads to failure in fracture. Higher density takes longer computational time.
4. Now the model is ready to be fractured. Run the python files in the above section and you can automatically achieve different fracture results.

1.4 Data processing and visualization

We use COMPAS framework for geometry processing and visualization. Installation instruction can be found in the following link:
https://compas.dev/compas/latest/installation.html
https://compas.dev/compas$_view2/latest/gettingstarted.html.$

1. Visualization. Run "020_vis_meshes.py" to visualize the fracture pieces of an object. Run "021_vis_mesh.py" to visualize a specific fracture piece.
2. Separate loose parts. When the Voronoi generating point is close to the mesh boundary, the output mesh can contain multiple loose parts. Run "031_seperate_loose_parts.py" to separate the parts.

3. Subdivison. Run "030_subd.py" to average the vertices numbers of the fracture pieces of an object.
4. Output. Run "040_npy_output.py" to output the information of vertices and normals to .npy format.

2 Keypoint detector

2_keypoint_detector.rar contains the following folders/files:

- *keypoints:* contains the generated key points of these objects
- *modelnet-pretrained:* contains the pretrained model of USIP, trained on modelnet by the authors of USIP
- *trained:* contains the trained models, trained on our generated data by us.
- *training_ data:* contains the (empty) folder structure necessary for "generate_trainingpairs.py" to work
- *USIP-changes:* contains the files of USIP, that where changed by us
- *Generate_ trainingpairs.py:* program to generate the training pairs

This readme assumes, that Pytorch is installed with Cuda enabled.

2.1 Set up the custom USIP installation

1. Install additional Python libraries:
   - `pip install visdom`
   - `pip install dominate`
2. Download the USIP code from Github:
   - `git clone –recursive https://github.com/lijx10/USIP`
3. go to folder "USIP/models/index_max_ext" and run `python setup.py install`
4. go to folder "USIP/models/ball_query_ext" and run `python setup.py install`
5. Copy the content of the folder "USIP-changes" to the USIP directory, overwrite all files.

2.2 Generate the training data

1. In the file "generate_trainingpairs.py" specify the folder of the point cloud data (data_folder), the folder to save the trainingpairs in (out_folder) and the list of datasets to use (names).
2. Run `python generate_trainingpairs.py`

2.3 Train USIP

1. In "USIP/modelnet/options_detector.py" change the argument "dataroot" to the path where the training pairs are stored.
   Note: The program expects two subfolders named "train" and "test" in this folder, containing a train and test split of the training pairs. The rest of the path should be the same as used by "generate_trainingpairs.py" (e.g. <dataroot>/train/connected_1vN/fragments_1/0.npy)

2. In "data/modelnet_shrec_loader.py":

   - In line 38 and 40 modify the variable "folder" to match the training pairs, on which the training should be done ("connected_1vN", "connected_1vN" or "flipped_normals")
   - In line 46 and 48 change the variable "nsamples" to the number of training pairs, that are used

3. In models/losses.py in line 103 change the output of the function to:

   - "return 2.0 * forward_loss, . . . " when training on the "connected_1vN" training pairs
   - "return forward_loss + backward_loss, . . . " when training on the other training pairs

4. In "USIP/modelnet/train_detector.py" change the variable "detector_model_path" to the path, where the pretrained model (trained on modelnet) is stored.

5. Run `python USIP/modelnet/train_detector.py` to start the training

2.4 Generate the keypoints

1. "data/modelnet_rotated_loader.py":

   - change the range of the for loop in line 23 to the number of fragments to generate keypoints for.
   - Modify the variable "item" in line 25 to match the filenames of the fragments to generate keypoints for

2. In "evaluation/save_keypoints.py":

   - Change the variable "object" to the dataset, which should be used (== the folder name of the raw point clouds)
   - Change the variable "model" to the trained model to use (MN, 1v1, 1vN or FN)
   - Change the variable "root" to the path, where the raw point cloud data is stored

- Change the variable “output_folder” to the path, where the key points should be stored

3. Run `python USIP/evaluation/save_keypoints.py` to generate the keypoints

3 Generate the features

3_keypoint_descriptor.zip contains the following folders/files:

- *data:* Containing all the data required
  - *keypoints:*
    - 3d_fracture_reassmbly-main:Containing pointclouf for "cube_6"
    - encoded_desc: Folder containing the encoded descriptors for different objects
    - features: Folder containing features from PointNet++ for different objects
    - keypoints_4: Folder containing keypoints for different objects
    - real_data_npy: Folder containing pointcloud for "brick"
    - triplets: folder containing the triplet pairs for different objects
- *data_ utils:* Utils files for PointNet++
- *log:* Containing pretrained model for Pointnet++
- *models:* Model definition for PointNet++
- *get_ neighbors.py:* Helper functions for computing neighbors for weighted averaging
- *features.py:* To extract the features for the pointcloud
- *find_ distances.py:* To find distances between keypoints of each fragment and generate triplet pairs
- *triplets_ train.py:* To load all the triplets for the data and train the siamese network
- *encode_ descriptors.py:* To load the trained siamese network and encode all the descriptors using the learned embedding

1. Create a virtual environment and install the following packages: (example shown with conda)

```
conda install pytorch==1.6.0 cudatoolkit=10.1 -c pytorch
```

2. Run "features.py" in order to generate features. Need to change the OBJECT_TYPE and specify the USIP_TYPE and run the file to extract the features. The script iterates through all the keypoints for the specified object and USIP type , extracts the features for the pointcloud and computes the features by the weighted averaging and stores it. The script automatically generates the folder under
"data/keypoints/features/OBJECT_NAME_USIP_TYPE".

3. Run "find_distances.py" to compute the distances between keypoints and generate the triplet pairs under the folder
"data/keypoints/triplets/OBJECT_NAME_USIP_TYPE".
Need to specify the names of the objects to iterate over in the "names" list and the "USIP_TYPE"

4. Run "triplets_train.py" to load all the triplet pairs and train the network. The trained network is stored in the project directory.

5. Run "encode_descriptors.py" to encode the keypoints using the trained network. The encodings are stored in
"data/keypoints/encoded_desc/OBJECT_NAME_USIP_TYPE".

4 3D reassembly optimization

4.1 Software requirements

- MATLAB R2021a (https://www.mathworks.com)
- YALMIP (https://yalmip.github.io)
- MOSEK (https://www.mosek.com)

4.2 Workflow

- Configure directories for point cloud, keypoint and feature data in 'import_3d.m'. Data should be stored in individual .npy files for each fragment. File format is 3 columns (x, y, z coordinates), n or 128 rows (points) for point clouds and keypoints. 32 columns (feature vector), 128 rows (points) for features.
- Run `main_assembly.m;` or alternatively execute functions individually:
- Run `[f,kp,ft] = import_3d;` to import point clouds, keypoints, features.
- Run `[f,kp,kp_index,kp_corr,frag_corr] = comp_kp(f,kp,0.5);` to retrieve keypoint correspondences based on euclidean distance in xyz (ground truth) space. This step is required if reassembly based on ground truth keypoint proximity is chosen instead of based on feature space proximity.

The last input argument denotes the search radius for nearest neighbour search. For the 'brick' data use 0.5, for the 'cube_6' data use 0.08. A correlation plot keypoint ID vs. keypoint ID denoting keypoint pairs will be generated by default.

- Run `[f_out, kp_out] = assy_3d(f,kp,kp_corr,frag_corr);` to perform 3D reassembly optimization. Set the following flags for configuration of the reassembly optimization:
  - use_ground_truth (set to 0 to determine keypoint correspondence pairs based on feature space, set to 1 to use xyz space (ground truth)
  - use_rigid_transformation (set 1 to perform pairwise matching with rigid body transformation based on inliers found by bnb optimization algorithm. Set to 0 to use transformation resulting from bnb optimization algorith directly. However, this might result in non-affine transformations)
  - add_isolated_fragments (not all fragments may have a triple-wise match with other fragments. These isolated fragments can be attached to the other fragments by using the adjacency information found during the pairwise matching. However, this functionality should only be used if few isolated fragments exist.
- A MATLAB workspace with the output of the reassembly optimization for the 'cube_6' model is stored in 'workspace_cube_6_result.mat'
- A video demo is available on YouTube
  https://youtu.be/lK_sCNtZFhs