# Code Execution Guide

**Ⅰ．Overview**

1. This guide explains how to use the code in the paper to predict polycube structures using the ddpm-polycube algorithm.**[1]**
2. The implementation of this code is mainly based on the Python language and 3D software Blender, using Python third-party libraries and Blender's built-in libraries. Below is a list of Python third-party libraries and Blender information used in this workflow.

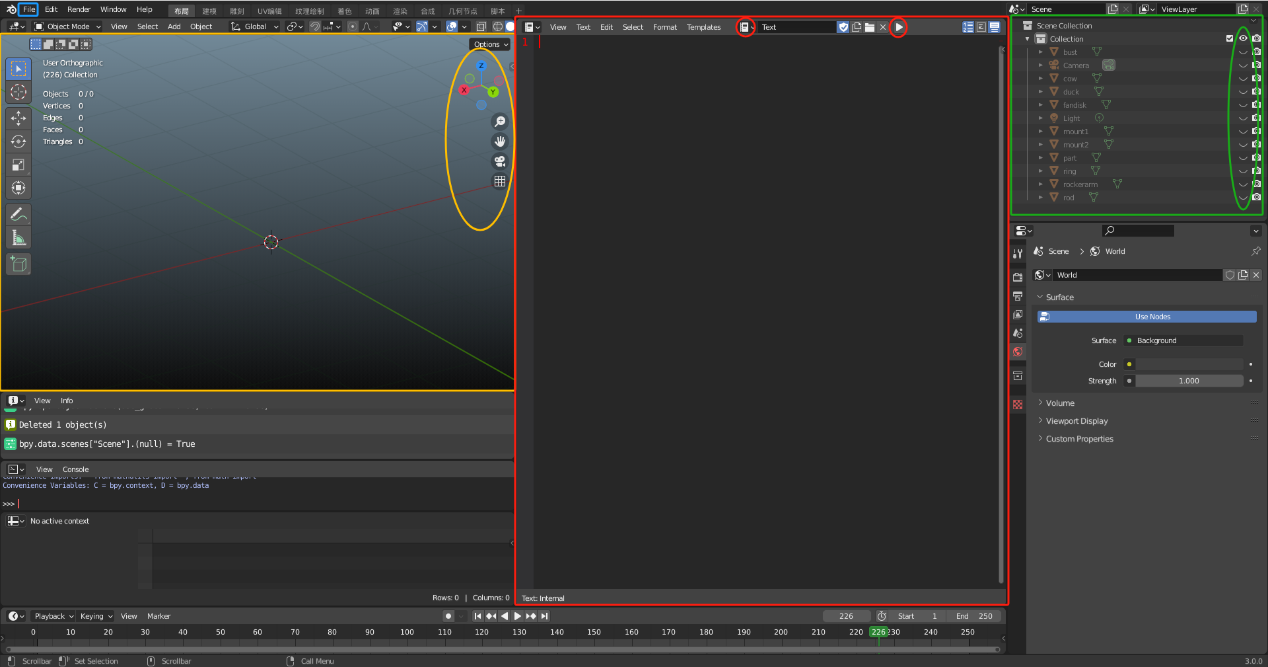
|  |  |  |
| --- | --- | --- |
| Environment/Library/Software | Version | Description |
| Blender | 4.3.2 | Built-in libraries `bpy`. |
| python | 3.9 | / |
| numpy | 1.23.5 | Consistency in version is recommended to avoid issues. |
| torch | 1.10.1+cu102 | / |

1. Introduction to Blender Interface and Basic Operations

The image below shows the basic Blender interface. In this code workflow, we mainly focus on three areas: the viewport (3D Viewport) outlined in yellow, the code area (Text Editor) outlined in red, and the file area (Outliner) outlined in green. Also, the File button outlined in blue is important. Below is an introduction to their uses:

Green rectangle (File Area):\*\* Lists all current models. Focus on the eye icon outlined in the green oval to show/hide models. Double-click the model name with the mouse to rename it.

1. Yellow rectangle (Viewport): Used to display the model and adjust the current view. The yellow oval area achieves view adjustments. Drag the top coordinate axis in the yellow oval with the mouse to rotate the view; similarly, drag the magnifying glass icon to zoom, and drag the hand icon to move the view. These operations allow you to inspect model details in the viewport.
2. Red rectangle (Code Area): Used for switching, browsing, and running code. The left red oval outlines a booklet-like icon for switching between different code files, while the right red oval outlines a triangle icon for executing code.
3. Green rectangle (File Area): Lists all current models. Focus on the eye icon outlined in the green oval to show/hide models. Double-click the model name with the mouse to rename it.
4. Blue rectangle (File): Click this when importing models. For example, to import an FBX file, click `File -> Import -> FBX (.fbx)` and select the desired FBX file to import.



1. Explanation of Subdirectories in This Directory:

|  |  |
| --- | --- |
| Subdirectory | Purpose |
| diff\_32\_steps\_stl | Stores the reconstructed 32-step diffusion triangle mesh result. |
| pics | Stores images showing the diffusion process. |
| src | Stores source code. |
| testing\_models | The fbx format file of the test model is stored. |
| tmp | Mainly stores various intermediate files output during the coding process. |
| training\_data | The constructed training data is stored. |
| v\_cos | The point coordinates of the model used for the training set and their connection relationships are stored. |
| weights | Stores the trained neural network weights. |

**Ⅱ．Execution Process**

The overall code is divided into two parts: one is to execute Python files directly, and the other is to execute Python-based scripts within Blender.

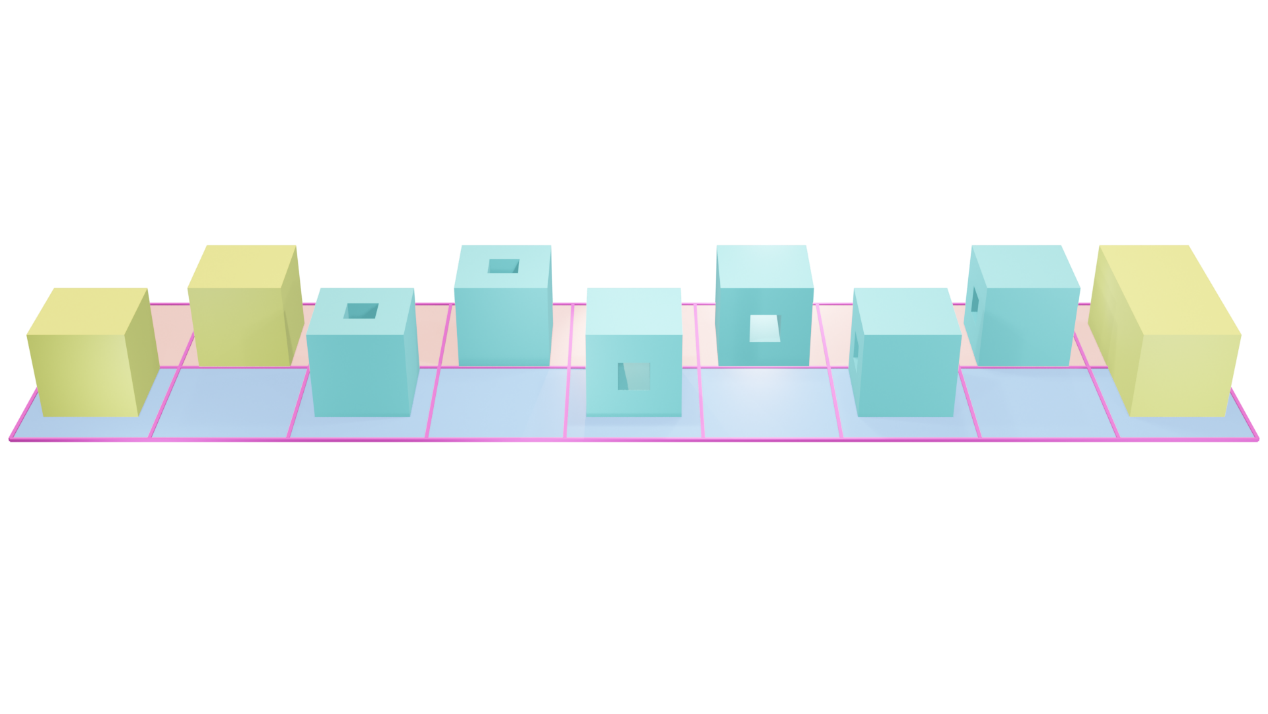
Before executing the code, there are two preparation steps:

- First, double-click to open the Blender file in the `src` directory.

- Second, import the model into Python. As mentioned earlier, use `File -> Import -> FBX (.fbx)` to import a model from the `model` directory into Blender. Note the imported model's name (check the file area); if the name is not `test`, rename it to `test`.

The most important preparation is to include the parameters g1 and g2 when running file 2. Specifically, when testing a model, you can select possible unit combinations based on the model's genus or other available information. For two units, you need to enter g1 and g2. g1 represents the -x-axis portion of the 2x1 grid, and g2 represents the +x-axis portion of the 2x1 grid. If you believe the model consists of only one unit, g1 should be set to -1, and g2 should be filled in with the possible unit number.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **/** |



For example, after importing the rod model in our dataset, because its genus=1, I think its -x part can be a cube and the +x part can be a torus, so I will enter the parameters g1=0 and g2=3 when running file 2. If I import a genus0 model, I can try a single basic unit cube, and in this case, I would adjust the inputs to g1=-1 (since there is only one unit) and g2=1.

It's important to note that because we want to select the best results, we may need to run the program multiple times for each model, selecting different but reasonable basic unit combinations (for example, a genus1 model might use g1=0, g2=3 or g1=0, g2=6, etc.) to try the diffusion process.

After preparation, you can start executing the code. The table below lists the execution order:

|  |  |
| --- | --- |
| Step Number | Specific Execution Content |
| **1** | Execute File 1 in Blender. |
| **2** | Execute Python file 2. Specifically, execute  **python 2\_Testing.py –g1 g1 value –g2 g2 value**`  in the command line. |
| **3** | Execute Python file 3. |
| **4** | Execute File 4 in Blender. |
| **5** | Execute Python file 5. (Optional) |

In general, when testing a model, you first need to calculate genus-related information, enumerate all possible basic unit combinations based on the genus, convert these basic combinations into parameters g1 and g2 (context), and then use the above program to batch generate polycube results for these cases. Finally, based on the polycube results generated with different parameters, you can determine the optimal parameters and corresponding polycube.

In addition, we also provide model training code, which can be used if you need training.

**Ⅲ．Results**

1. The results of the polycube recognition can be viewed in the file step32.k (opened with LS-Dyna) or step32.stl (opened with various 3D modeling software, including Blender) in the diff\_32\_steps\_stl folder. If you executed files 4 and 5, you can also see images of the entire diffusion process in the pics folder.

Note: Some of this code refers to the public sample code of How Diffusion Models Work.

**References**

[1] Y Yu, Y Fang, H Tong, J Liu, YJ Zhang.: DDPM-Polycube: A Denoising Diffusion Probabilistic Model for Polycube-Based Hexahedral Mesh Generation and Volumetric Spline Construction arXiv: 2503.13541 (2025)