Poly Shape - 2D Mesh Editor

v 1.0.0 - Flying Banana

23-Jan-2016

1. Thank you!

Thank you for buying Poly Shape! Or if you have obtained this asset by some other means, please support me by purchase it on the Asset Store if you think it's useful!

If you have any questions, suggestions, comments or feature requests, please email me at jiacong.xu@foxmail.com!

2. Overview

Poly Shape is a 2D mesh generator and editor. You define a list of paths and holes, and the script triangulates the resulting shape and generates a mesh for rendering. First, the geometry goes through pre-processing scripts (which may apply smoothing, for example). It is then parsed into a file stored under `Assets/PolyShape/Triangle/` folder. A background process (an algorithm called Triangle) then calculates the mesh. Next, the output files from Triangle are parsed into a Unity Mesh object. The resulting mesh is then passed into a series of user-defined post-processing scripts and finally assigned to the Mesh Filter component.

Poly Shape is designed to be flexible. The pre- and post-processing section describes ways to manipulate the mesh generation to create great procedurally generated graphics.

3. Set up

To setup, simply import all files from the package. Please do not move or rename the folder after importing the package, because Poly Shape uses a fixed relative path for I/O. In case you absolutely have to change the folder, you need to edit `TriangleAPI.cs` file and change the variable `trianglePath` to the appropriate new path relative to the Assets folder.

To set up a Poly Shape on a newly created Game Object:

1. add Poly Shape component,
2. add Mesh Renderer component,
3. set material of Mesh Renderer to Sprites-Default, and
4. add Polygon Collider 2D if you want auto-generated collision.

4. Usage

4.1 Geometry and editing

Poly Shape editor operates in two modes: a normal view mode (active when the component is expanded in the Inspector), and an editing mode (active when any of the Edit buttons in the Geometry section is toggled on).

Click the New Path button to create a path. Click anywhere in the Scene viewport to place vertices. Yellow lines are a preview where the mesh boundary would fall if a vertex is placed at the current mouse position, while green lines depict mesh boundary defined by the placed vertices so far. All the paths are automatically closed, with the closing edge drawn in yellow.

Apart from adding new points from the end of the line, you may also insert vertices in the middle of an edge by moving your mouse close enough to the edge until you see the mouse is connected by yellow lines from both vertices.

Drag to move the vertices around, and Shift-click to delete. Click Edit button again or press the Esc key to exit editing mode.

4.2 Mesh editing and generation settings

You can change the vertex size and colors for both editing and normal modes in case the mesh you are generating does not suit the current settings well.

If Generate Collider flag is on, collision data is automatically relayed to the first attached Polygon Collider 2D component.

If Auto-Rebuild flag is on, the mesh will be recalculated every time a change is made. While it is recommended to keep this on, Unity may get laggy with huge meshes.

Triangle settings affect how the geometry is triangulated. The two exposed properties are min angle (the minimum angle for each generated triangle in degrees) and max area (the maximum area for each generated triangle, in units defined in the Inspector). For example, if you want to generated a mesh with minimum number of triangles, you might want to set min angle to 0 and max area to 1000. However, if you want a nice and even-sized triangulated mesh, you might want to set min angle to 20 and max area to 10. Note that maximum area depends on the actual size of your mesh, but setting it too small *will* cause Unity to hang.

If you wish to have greater control over arguments sent to Triangle, you may consult this page and then edit the `ExecuteTriangle` method in `TriangleAPI.cs` file.

4.3 Pre-processing

Poly Shape can be extended by writing your own pre-processing scripts. The pre-processing scripts are able to “intercept” the geometry data before it is passed into the mesh generator. For example, we can apply a smoothing algorithm so the resulting mesh looks smooth.

Included in the package are two pre-processing scripts, `BezierSmoothing.cs` and `CornerCutter.cs`. To use them, add a pre-processing script as a component, then click Add Script button from pre-processing section of Poly Shape component, and finally drag the script to the added slot. Perform a rebuild mesh to see the effect.

To write your own pre-processing script, extend `PolyShape.Preprocessing` and override `Preprocess` method. Take a look at the example scripts to get an idea of what to do!

4.4 Post-processing

The post-processing scripts works in a similar way, but is probably best suited for applying stunning vertex colors that may not be achieved with textures. The purpose of the scripts is to modify the generated mesh after every mesh rebuild. After the mesh is built, it is passed into a list of post-processing scripts in order, alongside with the paths and holes used to generate it. The simplest script could assign each vertex a random color. The result would be a colored mesh instead of white mesh.

Included in the package are two post-processing scripts, `AlternateFade.cs` and `DistinctTriangle.cs`. To use them, add a post-processing script as a component, then click Add Script button frompost-processing section of Poly Shape component, and finally drag the script to the added slot. Perform a rebuild mesh to see the effect.

To write your own post-processing script, extend `PolyShape.Postprocessing` and override `Postprocess` method. Take a look at the example scripts to get started!

5. Implementation

This section describes the implementation of the editor in case you would like to customize it further.

`PolyShape`
The controller component that links everything together. It contains all the settings for each shape. It generates collision and the mesh (through calls to `TriangleAPI` class).

`PolyShapeEditor`
The custom editor script for vertex editing and property display. This shouldn't need to be touched.

`Shape`
Model object containing a name and a list of vertices. The shape is always assumed to be closed.

`Preprocessing`
This is the pre-processing base class. All pre-processing scripts should extend this class.

`Postprocessing`
This is the post-processing base class. It provides some convenient methods that may be useful in calculations. All post-processing scripts should extend this class.

`TriangleAPI`
The parser for Triangle program. This file can be modified to change the directory to store the temporary files, where the program is located (in the same folder at default), and how the program is called. You can also change the command line arguments in `ExecuteTriangle` method to gain custom control over mesh generation.

`PSLG`
This is a planar straight-line graph model. It contains information about vertices, segments, and holes. This is the intermediate representation of the paths and holes entered in the editor before it is converted into a text file and passed into Triangle.

`Polygon2D`
This is the intermediate representation of a mesh after the text file output given by Triangle is parsed by `TriangleAPI`. It contains vertex and triangle data.

6. Known issues

Currently inserting vertex between two vertices in editing mode is functional, but when you click drag, the editor graphics are not being updated until the mouse stops dragging. The vertex is still inserted at the correct position, though.

Adding a post-processing script will not trigger an auto-rebuild even if the flag is on.

If you find the compiled Triangle program to be incompatible with your platform, please download the source code here, compile it on your machine, and modify the file path variable as outlined in the setup section. If you need help with this step please send me an email. (Please first make sure you attached a Mesh Renderer component and set the correct material!)

7. Inquiries, suggestions and comments

Please email me at jiacong.xu@foxmail.com! I hope this asset will be helpful for your need!

8. Credits

The underlying software for triangulation is Triangle, written by Jonathan Richard Shewchuk. There is *no* source code or object files of Triangle contained in this package.

The Unity parsing and processing API is taken and modified from this blog post, written by Jens-Kristian Nielsen.