Wireframe Shader

amazingassets.world

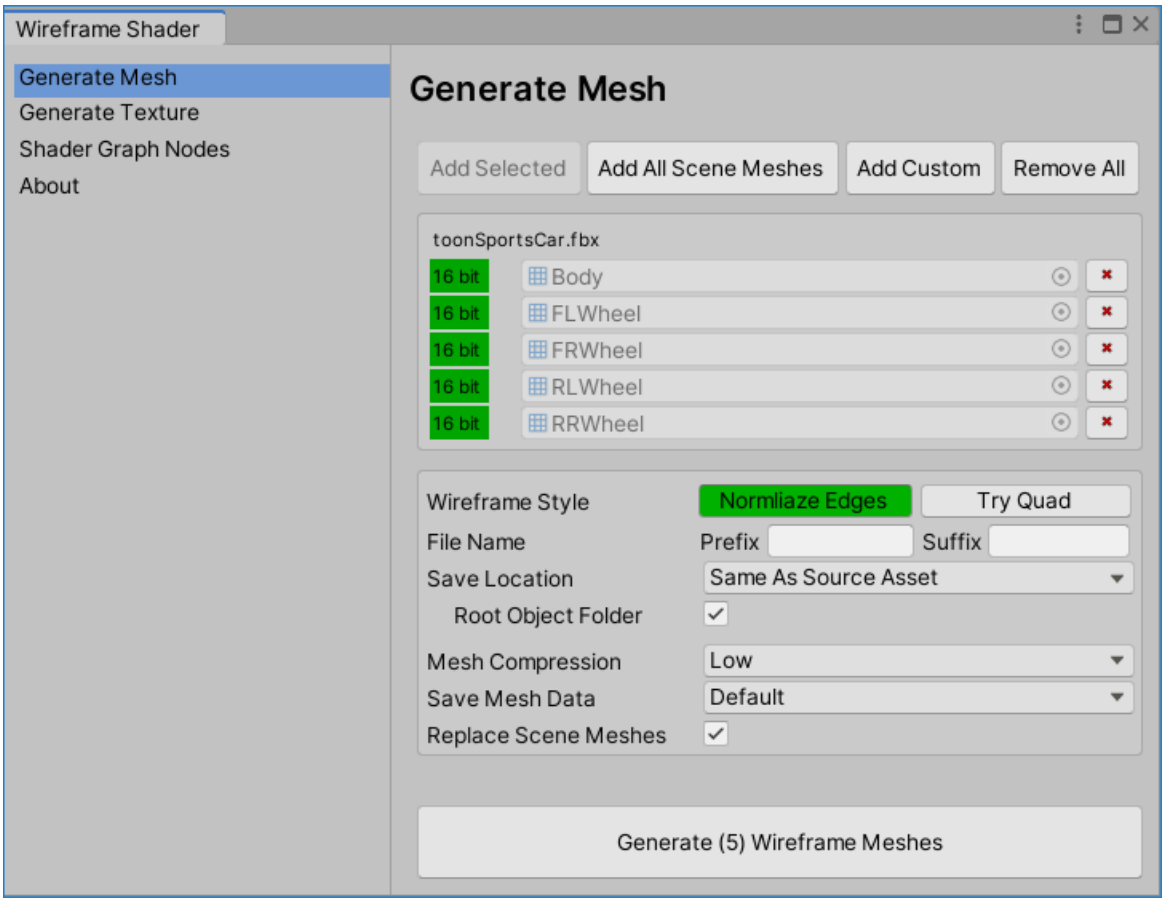
Contents

1. Quick Start

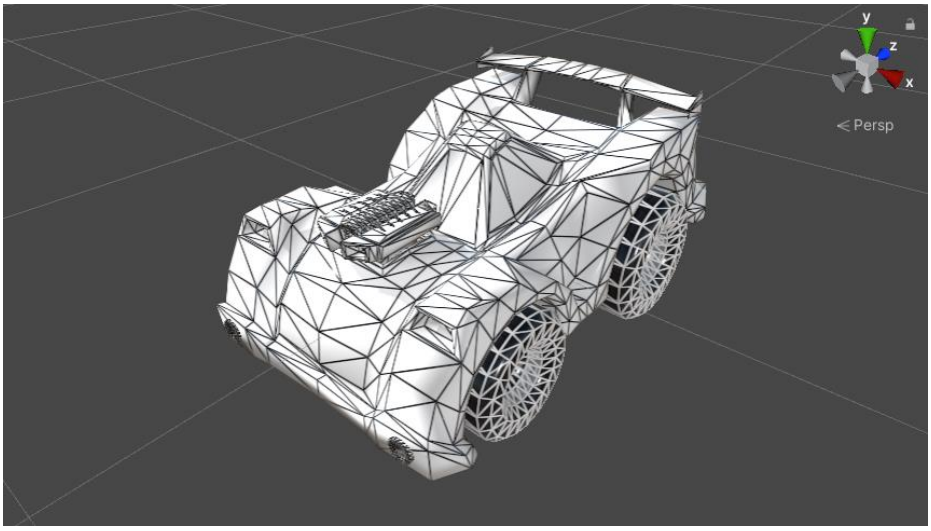
Before rendering wireframe, it is necessary to calculate it and bake inside mesh.

1. For the desired mesh change material shader to the ***Amazing Assets/Wireframe Shader/Standard***. Mesh will be rendered in the solid color (red) without wireframe.

2. It it necessary to bake wireframe inside mesh first. Open editor window from ***Unity Main Menu/Window/Amazing Assets/Wireframe Shader***.
   Choose **Generate Mesh** tab and drag & drop source mesh there.



3. Set properties as on the image above and click on the Generate button.
   If **Replace Scene Meshes** options is enabled then source meshes in the scene will be automatically replaced with the new generated ones. Shader now will render wireframe.



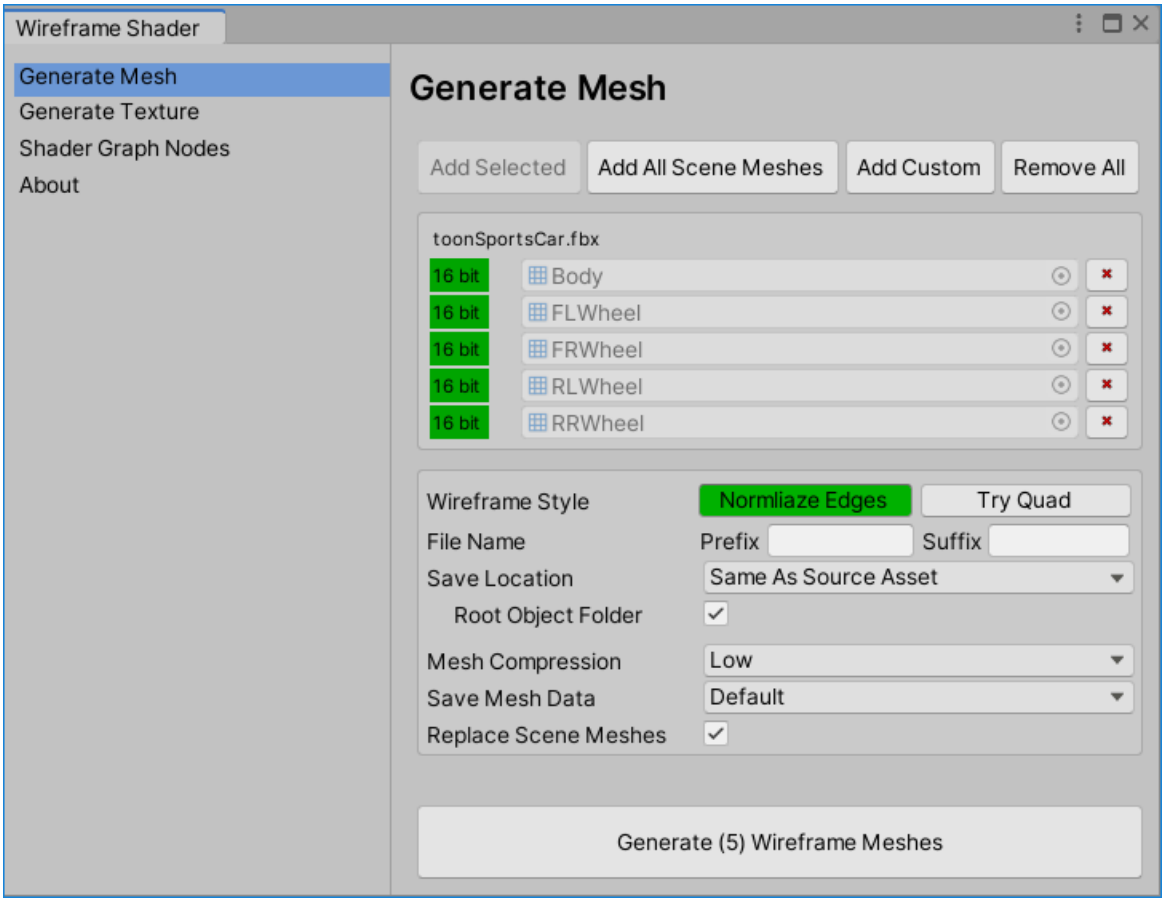
That all.

Package includes various wireframe shaders for different effects that are demonstrated in example scenes. If included shaders are not enough wireframe rendering effect can be easily integrated into any shader, hand-written or created using shader graph tools.

2. Editor window

2.1 Generate Mesh

This tab allows calculating and baking wireframe data inside mesh. Generates wireframe meshes.



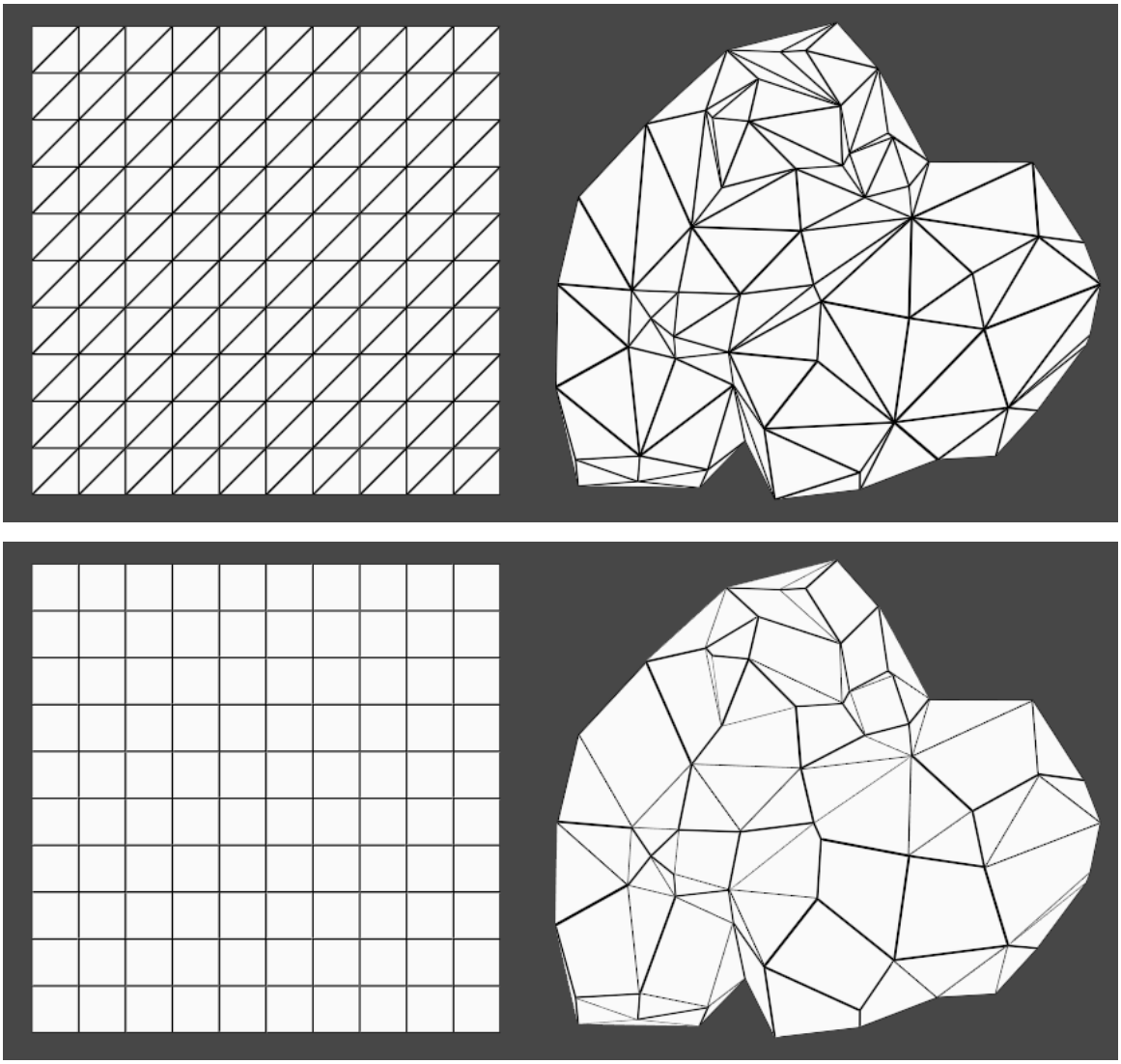
Each wireframe triangle in the mesh is unique and it is necessary to calculate and bake wireframe data per-vertex. Compared to the original, generated mesh will always have the same triangle count, but vertex count equals to ***3 * triangle count***.

16 bit flag on the left side of a mesh indicates that generated mesh will be saved in 16 bit index format. Such meshes support up to 65,535 vertices (21,845 triangles) and can be used almost on any device.
32 bit flag indicates that new mesh will have more than 65,535 vertices and will be saved in 32 bit index format. Such meshes support up to 4 billion vertices, but may not be supported on ***supper low end*** devices. Mostly those are Androids with Mali 400 GPU and before using meshes make sure target device supports them.

Normalize Edges – Wireframe baking algorithm will try to make a width off all edges of the wireframe triangle equal. This effect can be noticeable when using high ***Thickness*** values inside wireframe materials.

Try Quad – Wireframe rendering in quad style is pure approximation and its quality depends on mesh vertex/triangle layout.

Images below demonstrate Triangle and Quad wireframe style rendering for Unity built-in Plane (good vertex/triangle layout) and custom meshes.



File Name Prefix/Suffix – Generated mesh name is same as the source mesh. Adding prefix/suffix to the name makes it easier to search wireframe meshes in a project.

Save Location – Choose generated file location. By default it is the same as the source file location..

Root Object Folder – If multiple meshes are from the same file (FBX and OBJ files can contain several mesh files), they are saved inside one folder with the name of the source file.

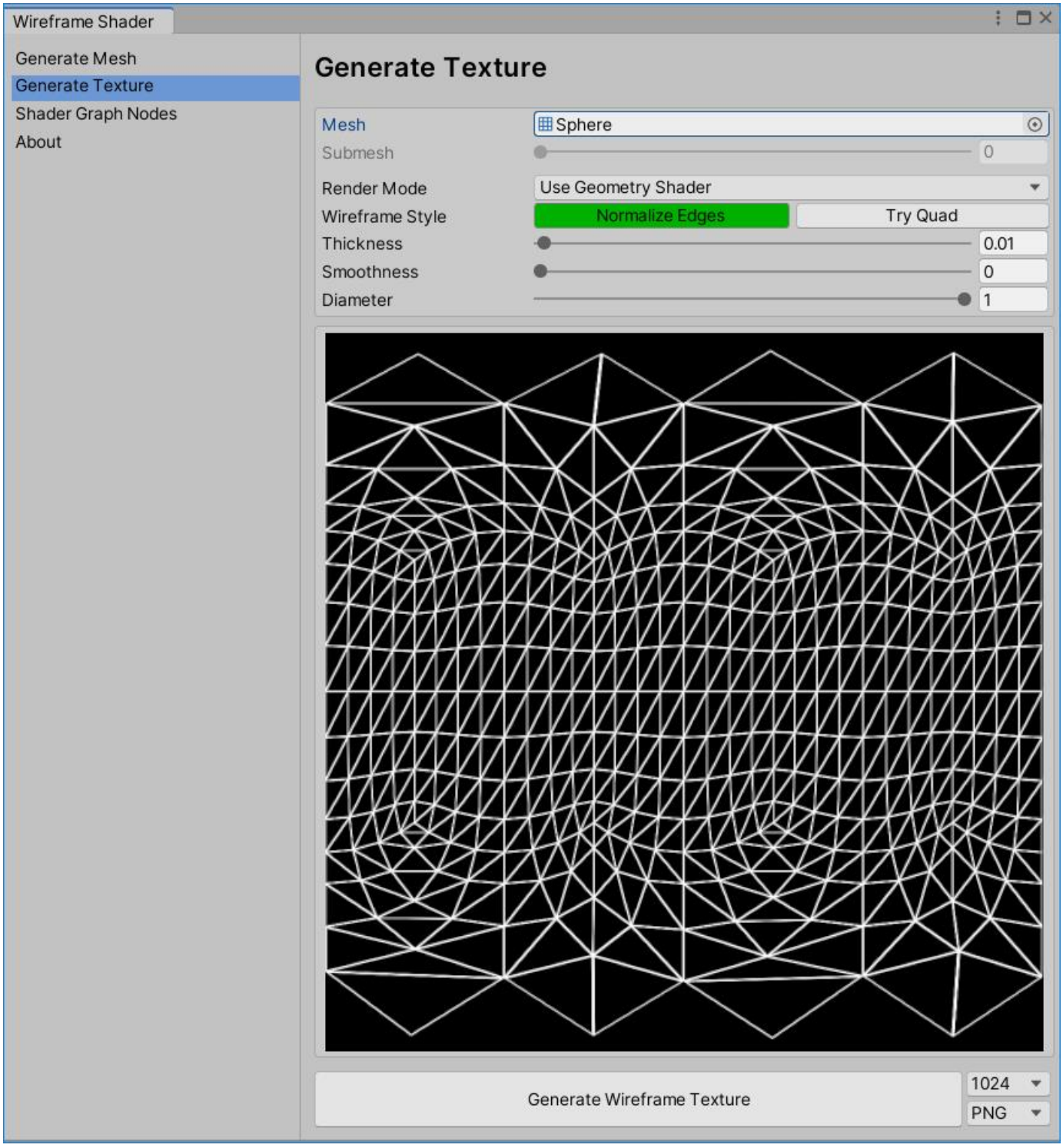
Mesh Compression – Reduces mesh data precision. Instead of 32-bit floats, lower size (the exact size depends on compression quality) fixed number will be used to represent mesh data. As a result generated file size is reduced.

Save Mesh Data – Allows excluding specific data buffers from the generated mesh. As a result generated file size is reduced.

Replace Scene Meshes – After generating wireframe meshes, all scene objects using source meshes will be replaced with newly generated ones. This step can be **Undo**.

2.2 Generate Texture

This tab allows baking wireframe data into a texture.



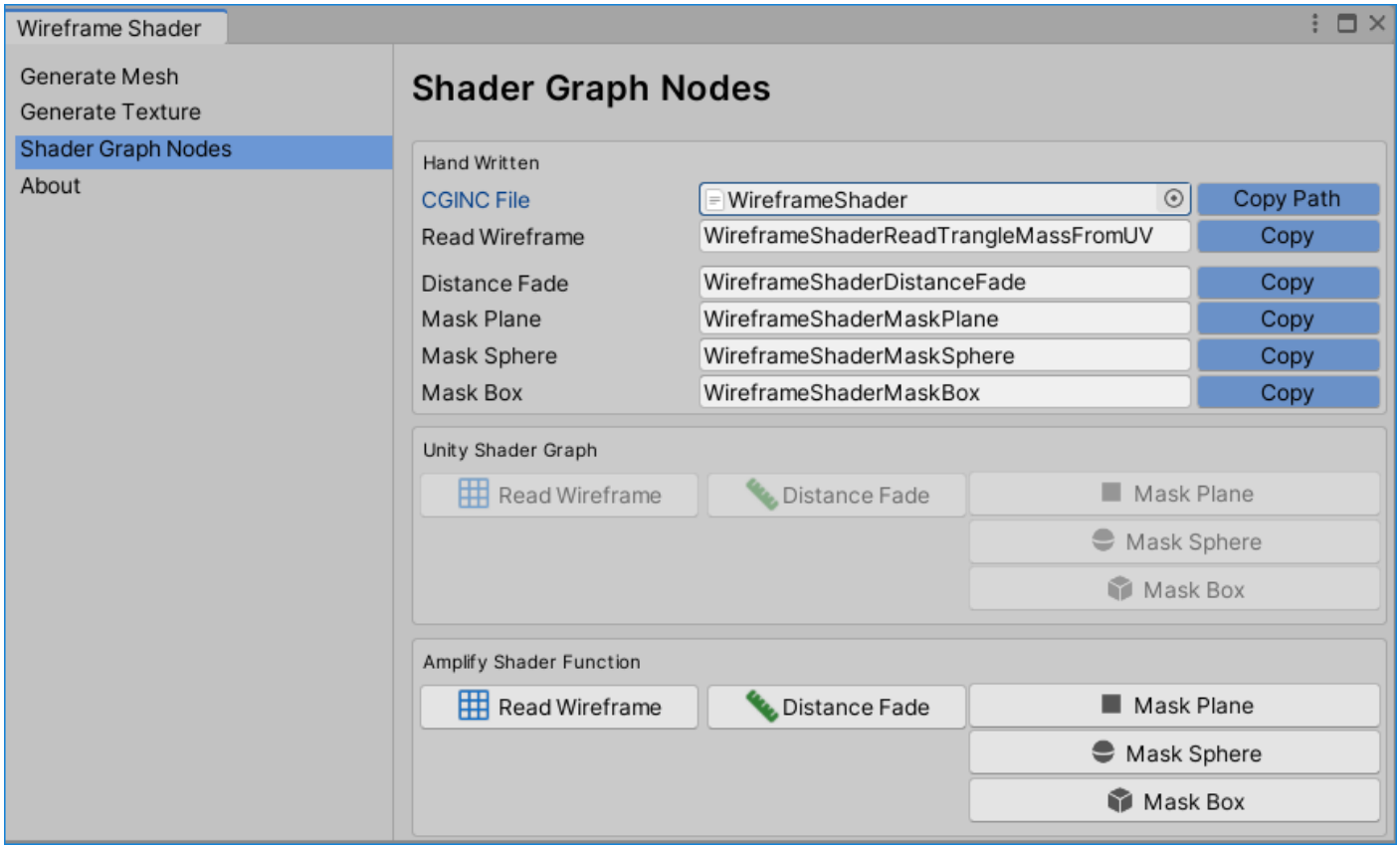
Submesh – Displays submesh count of a mesh. Each submesh has its wireframe data and is exported in separate texture file.

Render Mode – Wireframe data can be read from mesh (if it has such data baked inside) or calculated dynamically using Geometry Shader.

Generated texture can be exported in up to 8K resolution and in JPG, PNG or TGA formats.

2.3 Shader Graph Nodes

Contains references to the **Unity Shader Graph** and **Amplify Shader Editor** nodes. Explained in Chapter 4. Custom Shaders.

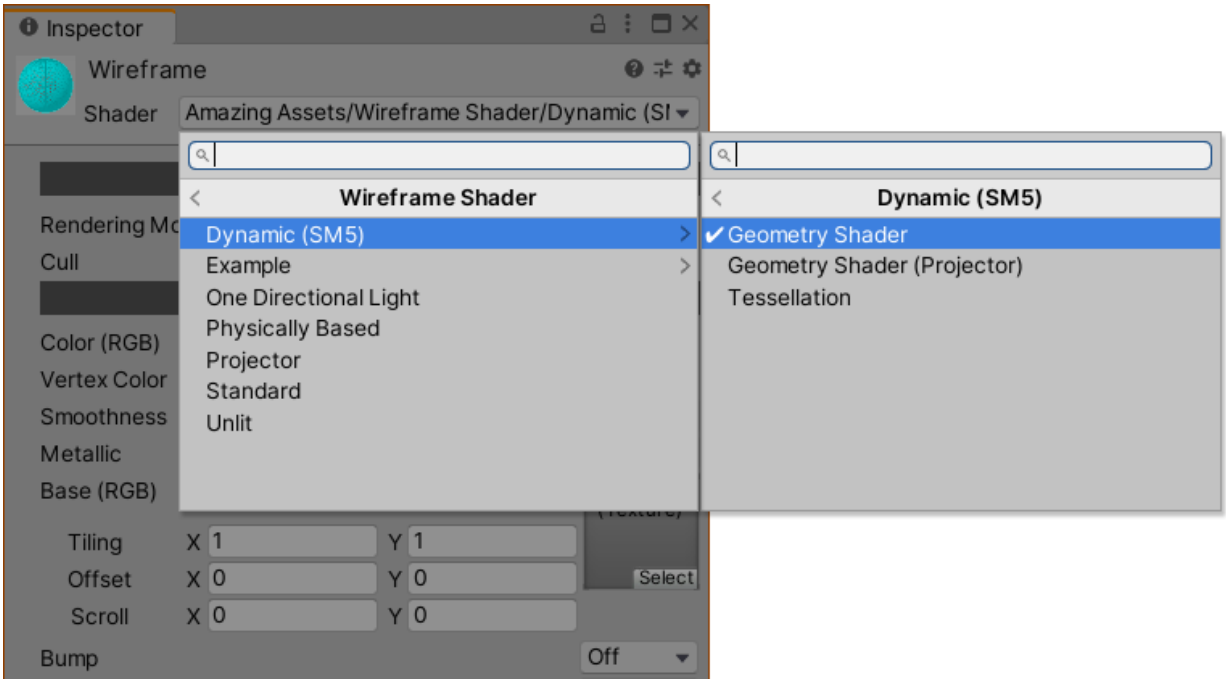


3. Dynamic wireframe rendering

Package contains collection of the Dynamic Wireframe shaders that do not need wireframe data to be baked inside mesh, instead everything is calculated directly by shaders. Those shaders require device with Shader Model 5.0 and Geometry Shaders supports.

Currently dynamic wireframe rendering is supported only by **Built-in** (Standard) render pipeline.

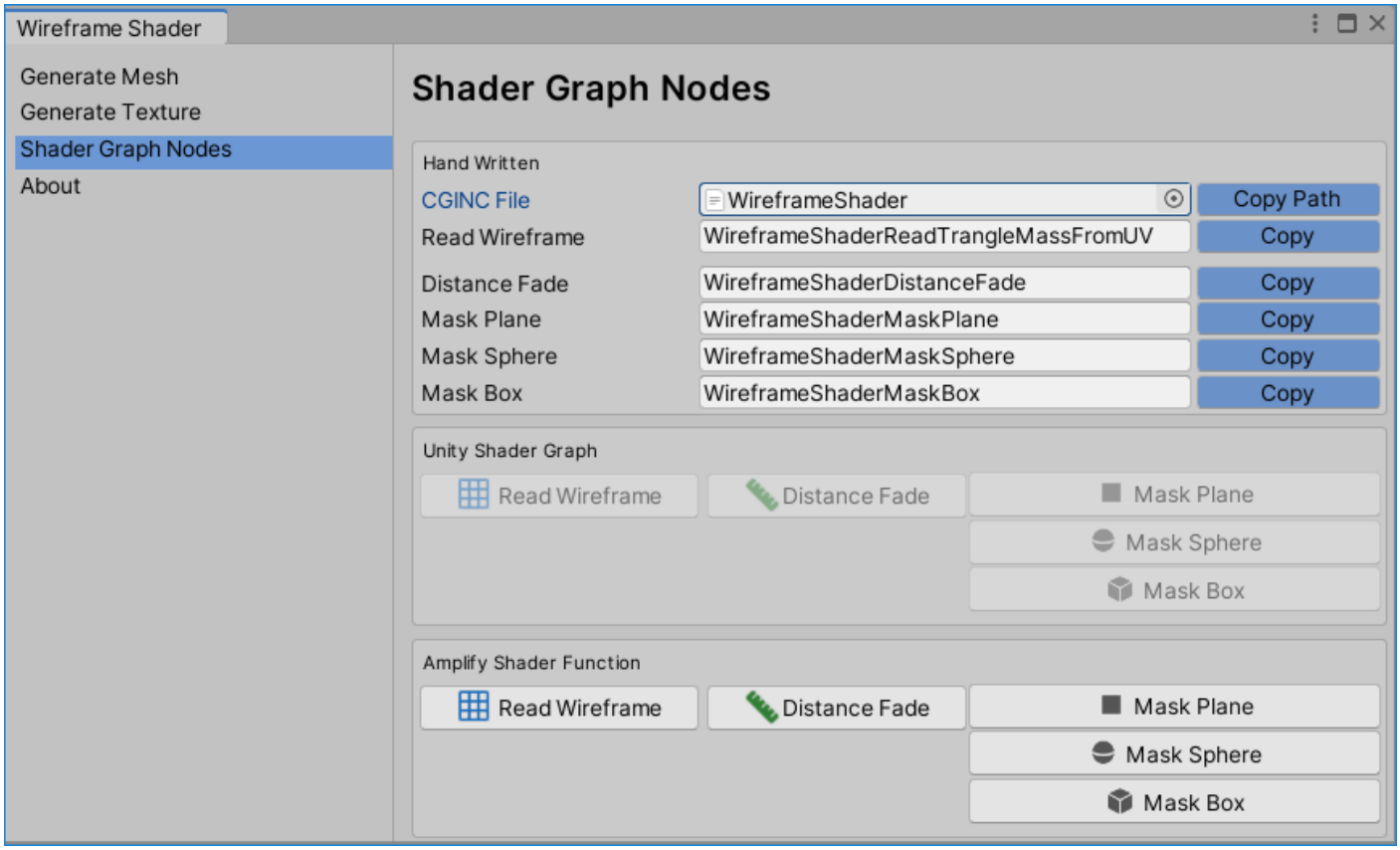
For Universal and High Definition render pipelines above technique can be achieved manually by integrating Geometry Shaders into only hand-written shaders, as ShaderGraph does not support it yet.



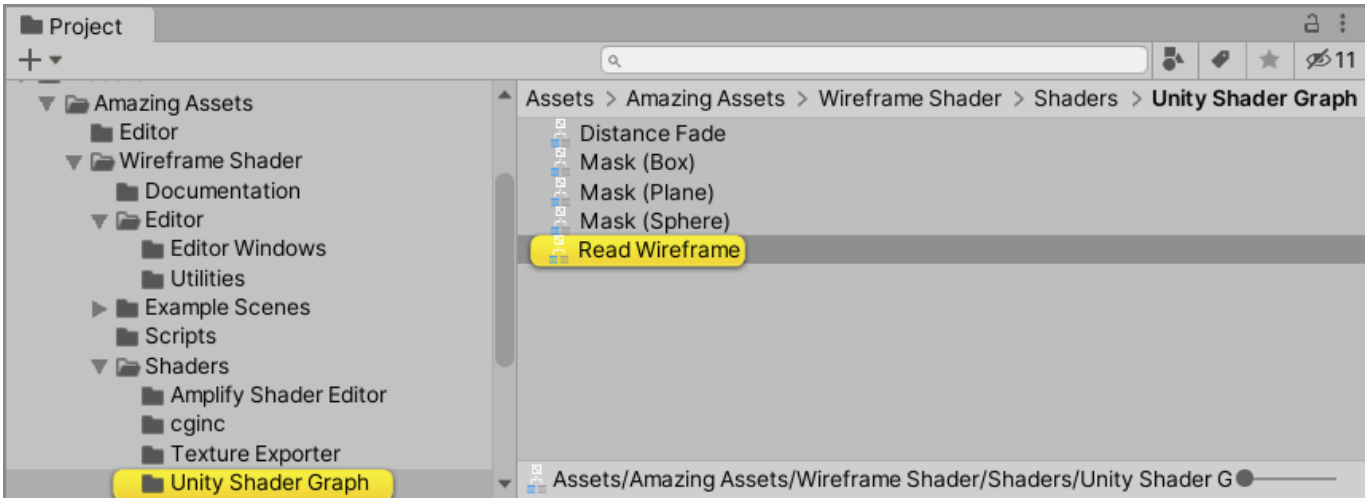
4. Custom shaders

Wireframe rendering effect can be integrated into any shader created using shader graph tools or hand-written.

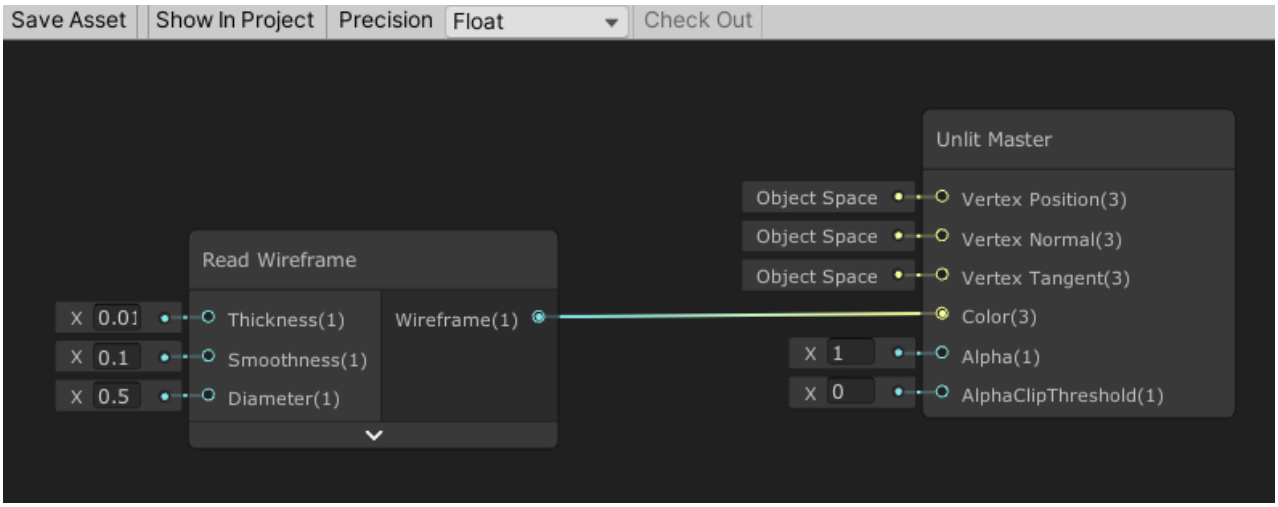
For the Shader Graph and Amplify Shader Editor it is just one node integration. Open Wireframe Shader editor window and go to the **Shader Graph Nodes** tab.



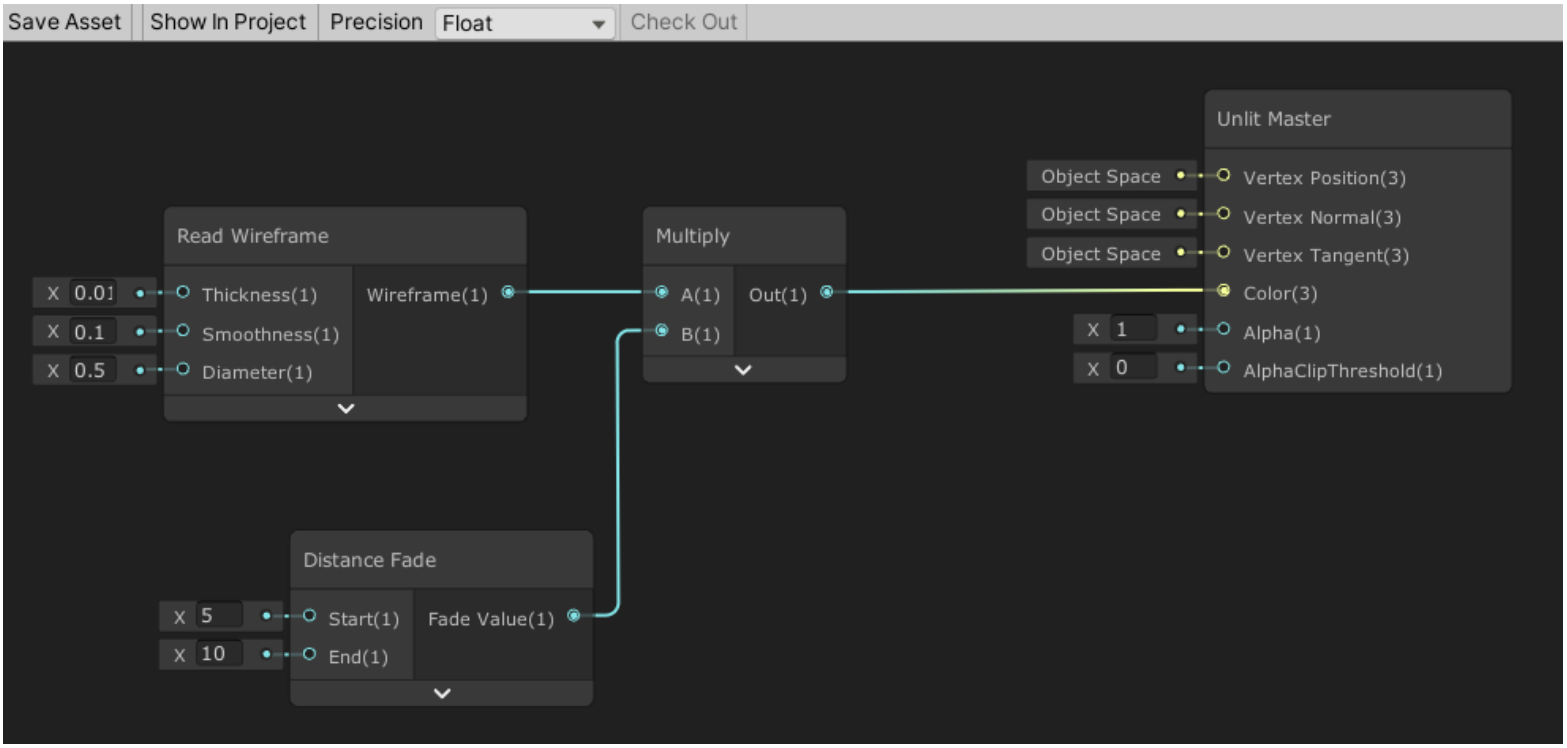
Selecting **Read Wireframe** button will highlight graph node file inside **Project** window.



Drag & drop file inside shader graph tool and use its output. That's all.



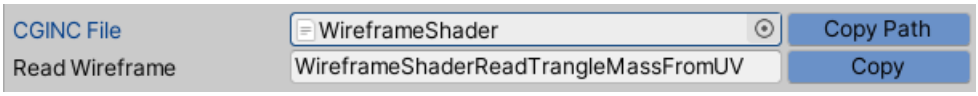
Package includes additional nodes for creating distance fade and dynamic mask effects. Output of those nodes is are the range of 0-1 and can be simply multiplied with **Read Wireframe** (or any other) node.



Using mask nodes (Plane, Sphere or Box) requires material data to be updated using **WireframeMaskController** script.

For the hand-written shaders:

1. Inside shader define path to the `WireframeShader.cginc` file.
   This can be easily done by clicking on the **Copy Path** button inside **Shader Graph Nodes** tab.
   Keyboard memory now will contain path to the file, just paste it into the shader.

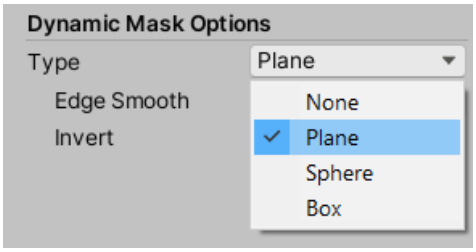


2. Path TEXCOORD3 value from vertex to the fragment stage.
3. Use `WireframeShaderReadTrangleMassFromUV` method to read wireframe from TEXCOORD3.

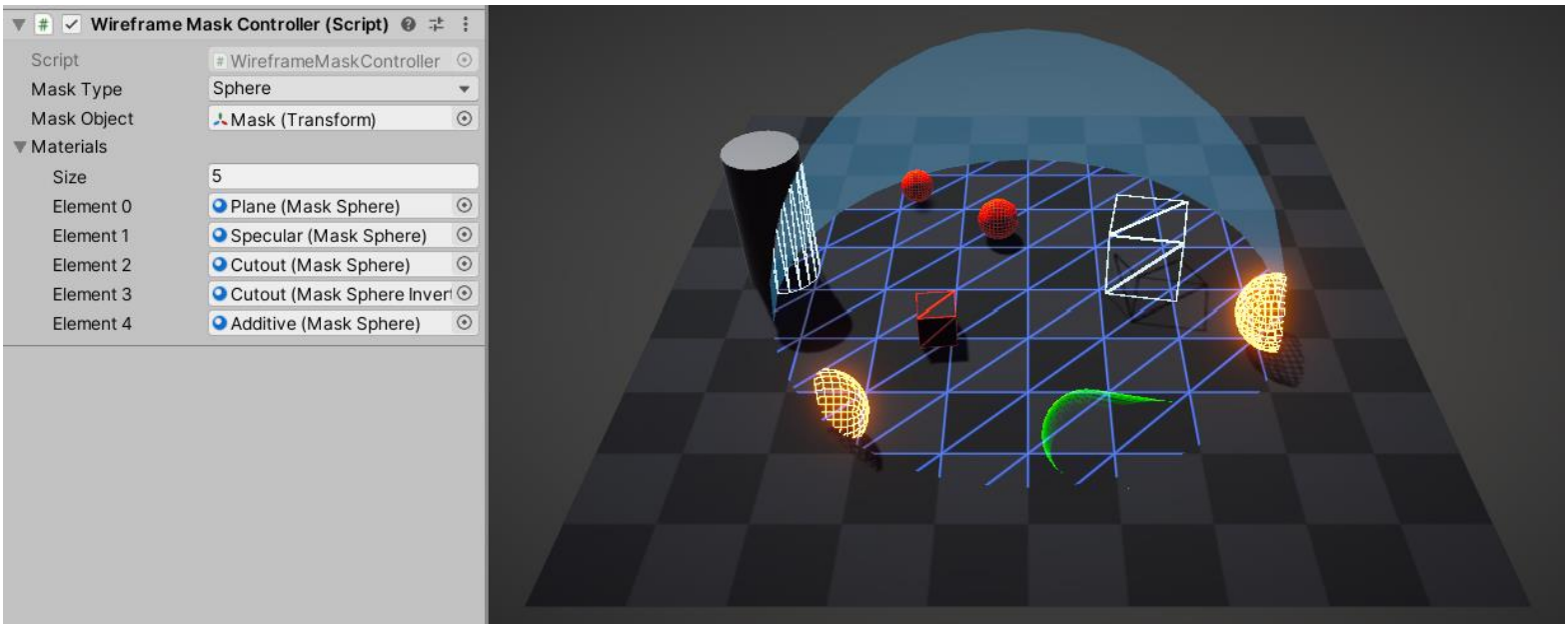
Package includes example shaders (**Amazing Assets\Wireframe Shader\Shaders\Example** folder) for demonstrating and better understanding integration process for the hand-written shaders.

5. Dynamic mask controller

Wireframe shader allows using Plane, Sphere and Box for dynamic masking (hiding) wireframe effect. Most of the included shaders have already integrated this feature and it can be turned on/off from material editor.



Dynamic mask uses Unity built-in meshes and in material editor there are no options for controlling transformation info for those meshes, instead they must be updated from script.



WireframeMaskController is used to update materials mask properties (position, rotation and scale). One instance of this script can update only one type of a mask for multiple materials.
Package includes example scenes demonstrating all 3 type of the dynamic mask.

Included shaders can use only one type of a mask, but for custom shaders it is possible to mix them all together.

6. Runtime API

Wireframe Shader run-time API can be brought into scope with this using directive:

```
C#
using AmazingAssets.WireframeShader;
```

Unity Mesh class now will have 2 additional extension methods:

```
Mesh GenerateWireframeMesh(bool normalizeEdges, bool tryQuad)
```

Generates new mesh with wireframe data baked inside uv4 buffer (note inside shader uv4 coordinate of a mesh is read using TEXCOORD3 semantic). Resultant mesh is in 16 bit index buffer format if final vertex count is less than 65,535 vertices (21,845 triangles), otherwise mesh is in 32 bit index buffer format.

Note, generated mesh currently does not support blend shapes.

```
Texture2D GenerateWireframeTexture(bool useGeometryShader, int submeshIndex,
                                   bool normalizeEdges, bool tryQuad,
                                   float thickness, float smoothness, float diameter,
                                   int resolution)
```

Generates wireframe texture file.

`bool useGeometryShader` – If enabled then Wireframe texture is generated using Geometry Shaders and source mesh does not require wireframe data to be baked inside it. For run-time use project also must include **Amazing Assets/Wireframe Shader/ShadersTexture Exporter/Texture Exporter.shader**.
If option is not used then wireframe texture is calculated from data baked inside mesh.

`int submeshIndex` - Index of a submesh. Wireframe texture is exported only for one submesh at a time.

`bool normalizeEdges` – Wireframe triangle edges will be approximately of the same width.

`bool tryQuad` – Try generating wireframe of the quad style. This highly depends on mesh vertex/ triangle layout.

`float thickness, smoothness, diameter` – Visual characteristic of the rendered wireframe.

`int resolution` - Texture resolution. Must be power of 2, in the range of 32 – 8192.