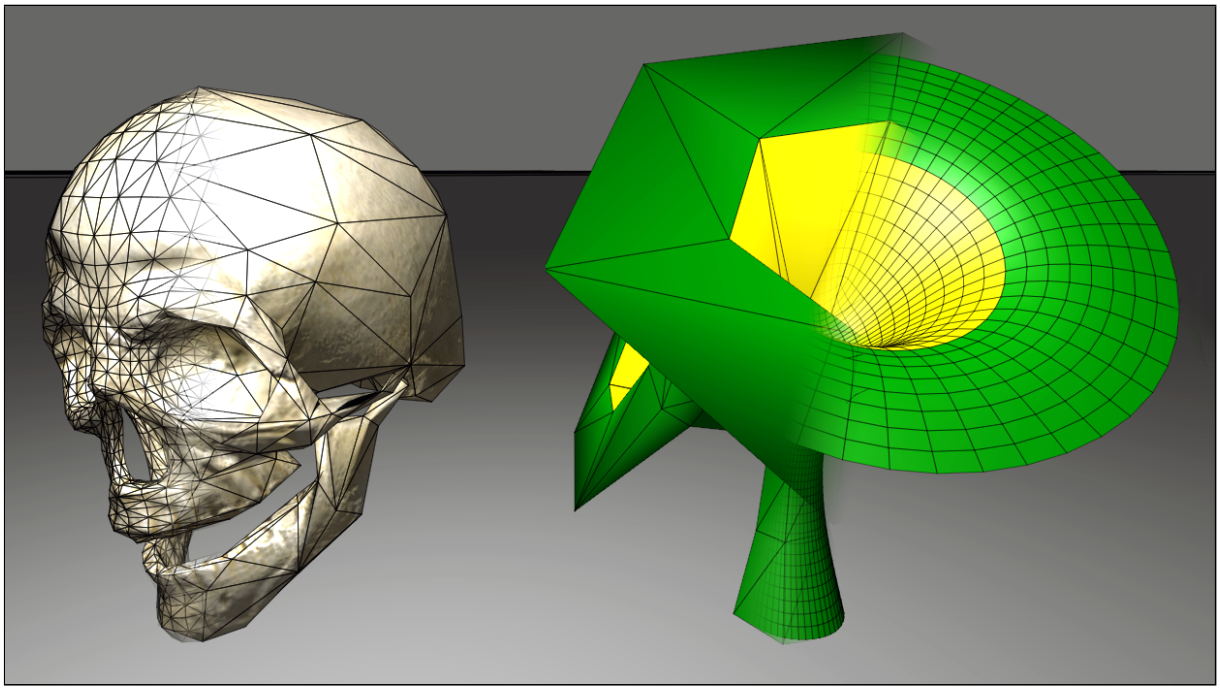
Krabl Mesh Processors v1.0.283 Manual

Mesh Processing Framework for Unity3D

http://krabl.com

Manual written with LaTeX on December 7, 2013

1 Introduction



“Krabl Mesh Processors” is an add-on for the Unity3D game engine to perform mesh processing. It consists of a C# mesh processing library and an editor extension which provides a mesh input processing workflow inside the Unity3D editor.

Different kinds of mesh processors can be chained together right inside the Unity3D mesh inspector. These import programs are executed every time Unity imports a mesh (inside a model).

Processors include mesh simplification (polygon reduction) and mesh subdivision. The system is designed to be extended with more processors and workflows in the future.

Main features:

- C# mesh processing library with Unity3D editor integration.
- Supports Unity3D Free and Pro on all platforms. Requires Unity v4.0 or newer.
- High quality mesh simplification (polygon reduction).
- Quad-based mesh subdivision.
- Mesh attributes filtering, crease detection.
- Extends the Unity3D mesh inspector to work like the texture inspector.
- Adds UV displays to the mesh inspector.
- Automated mesh import programs with build target dependent settings.
- Comes with full source code (C#) and a utility class to easily process meshes in your own scripts (in any language supported by Unity3D) during runtime.

Contents

2 End User License Agreement

END-USER LICENSE AGREEMENT FOR krabl.com

This krabl.com End-User License Agreement (“EULA”) is a legal agreement between you (either an individual or a single entity) and krabl.com for the software accompanying this EULA, which includes computer software and electronic documentation (“SOFTWARE PRODUCT” or “SOFTWARE”). By exercising your rights to make and use copies of the SOFTWARE PRODUCT, you agree to be bound by the terms of this EULA. If you do not agree to the terms of this EULA, you may not use the SOFTWARE PRODUCT.

DISCLAIMER OF WARRANTY

This product is provided on an “AS IS” basis, without warranty of any kind, expressed or implied, including any warranties of fitness for a particular purpose. The authors shall not be liable for damages of any kind. Use of this software indicates you agree to this.

SOFTWARE PRODUCT LICENCE

The SOFTWARE PRODUCT is protected by copyright laws and international copyright treaties, as well as other intellectual property laws and treaties. The SOFTWARE PRODUCT is licensed, not sold.

GRANT OF LICENSE

Installation and Use:
You may install and use copies of the SOFTWARE PRODUCT on all computers you own.
Reproduction and Distribution:
You may not reproduce or distribute the SOFTWARE PRODUCT except to make backup copies.

DESCRIPTION OF OTHER RIGHTS AND LIMITATIONS

Termination:
Without prejudice to any other rights, krabl.com may terminate this EULA if you fail to comply with the terms and conditions of this EULA. In such event, you must destroy all copies of the SOFTWARE PRODUCT and all of its component parts.

COPYRIGHT

All title and copyrights in and to the SOFTWARE PRODUCT (including any images, and text incorporated into the SOFTWARE PRODUCT) and any copies of the SOFTWARE PRODUCT are owned by krabl.com or its suppliers.

3 Package Contents/Directory Structure

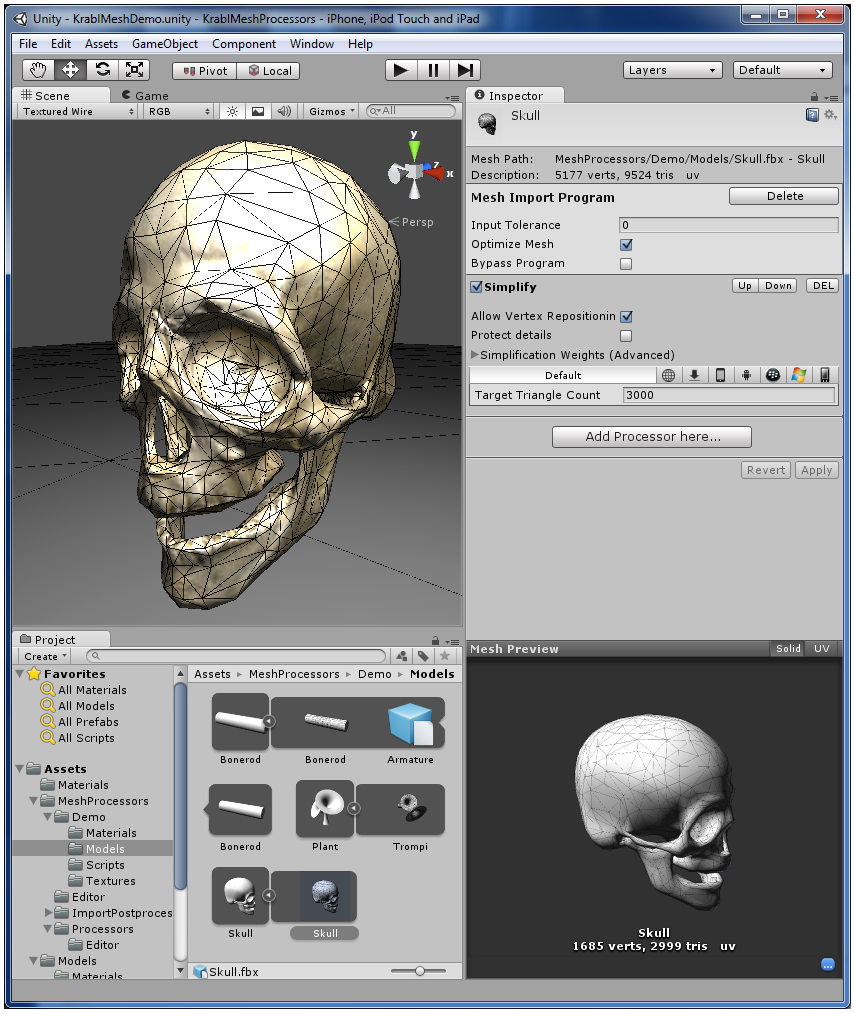
- Assets
  - MeshProcessors
    - Demo — Content to demonstrate the capabilities of the package. This folder can safely be deleted if its contents are not needed.
      - KrablMeshDemo.scene — The tutorial demo scene.
      - Materials
      - Models
      - Scripts
        - KMDemoBoneRot.cs — The animation script used in the skinned mesh (Bonerod) example.
      - Textures
    - Documentation
      - Krabl Mesh Processors Manual.pdf — The user manual you are reading.
      - Krabl Mesh Lib Reference HTML.zip — Doxygen generated zipped html reference for the most useful calls and classes of the krabl mesh library. Unzip and open the enclosed index.html to browse it.
    - ImportPostprocess — The C# classes used for the editor integration and the post processing workflow.
    - MeshImportProcessorSettings.asset — This file is not included but generated once a mesh import processor program is added to any mesh. It contains all the import programs. This should never be changed or duplicated by the user. If it is erased, all import programs are deleted.
    - Processors — The source code of the processors than can be used in import programs.
  - Plugins
    - KrablMeshLibrary — The mesh processing library files. If you just want to use the library in your scripts, this folder is all you need. It is placed inside the global Unity3D Plugins folder to allow accessing the library from UnityScript and Boo scripts.
- Library
  - krablmeshrevisions — This file is not included but generated once import processor programs are executed. It contains revision numbers for each model that are needed to keep the meshes current if a project is modified on different machines.

4 Tutorial

This tutorial shows how to simplify a static mesh, how to simplify a skinned mesh and how to apply subdivision and a number of changes to a low poly mesh during mesh import. To begin, open the demo scene by double-clicking KrablMeshDemo.scene inside MeshProcessors/Demo.

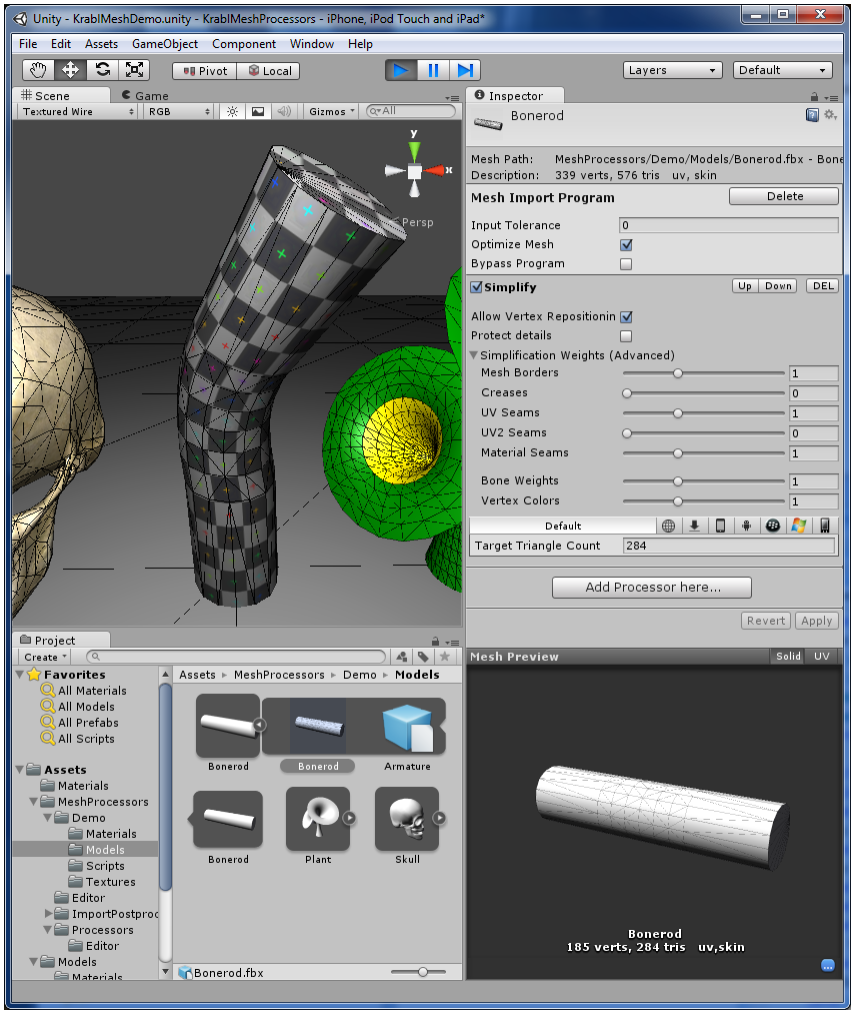
4.1 Import simplification of a static high-poly mesh (skull).

1. Navigate to the “Skull” model in the project folder (inside MeshProcessors/Demo/Models) and drill down to the “Skull” mesh and select it.
2. In the mesh inspector, click on “Add Mesh Import Program” to create a mesh import program for the skull mesh.
3. Click on “Add Processor here...” and choose “Simplify” to create a simplification mesh processor.
4. Enter 3000 in the “Target Triangle Count” field and click “Apply” at the bottom of the program to reimport the mesh and calculate the program. You have now reduced the polygon count of the skull mesh to 3000 triangles.



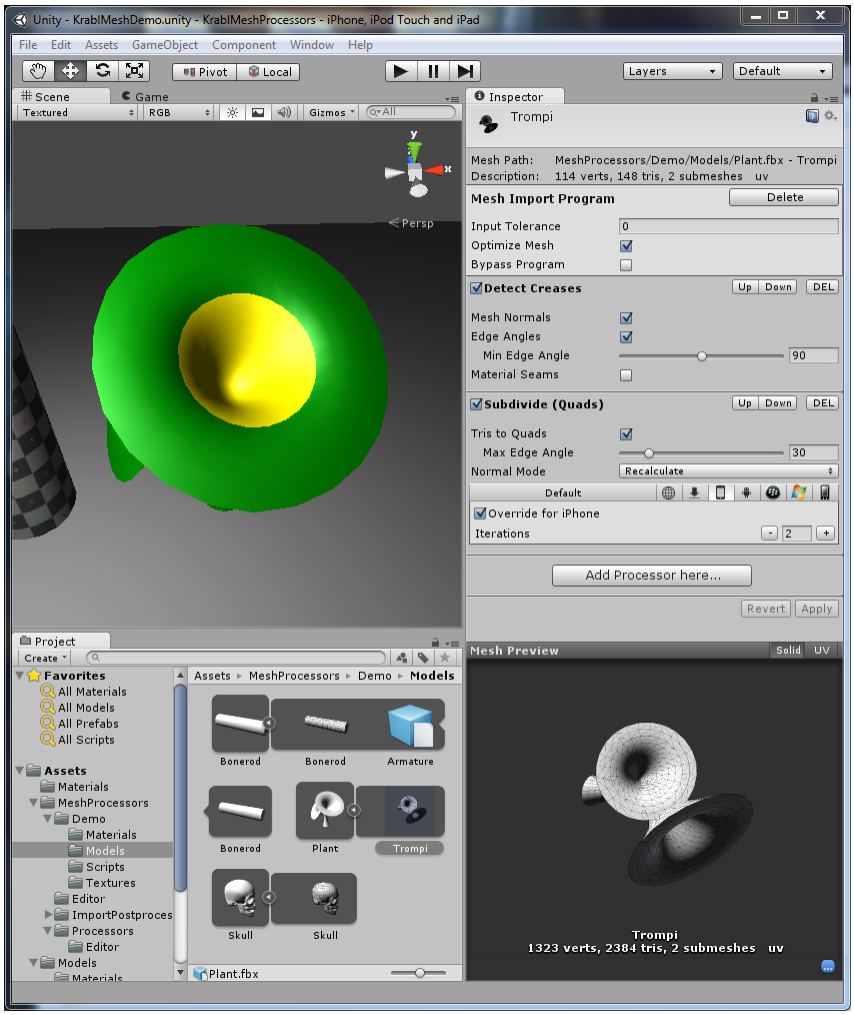
4.2 Import simplification of a skinned mesh (animated bonerod).

1. Hit the main play button to start gameplay in Unity3D. There is a rotation script on the center rod and the rod will start deforming. The rod has many redundant faces from subdivision, especially at its top and bottom.
2. Navigate to the “Bonerod” model in the project folder and select the bonerod mesh inside it. There are multiple items inside the model because this is a skinned mesh. Make sure to select just the mesh.
3. Create a program using “Add Mesh Import Program” and add a “Simplify” processor to it.
4. Enter 284 as the target triangle count and hit “Apply” at the bottom of the program.
5. With 284 triangles, the rod looks still exactly the same. The simplification has determined what faces are deformed and protected them during the process.
6. Click on the triangle to the left of “Simplification Weights (Advanced)” to expand the weighting section of the simplification processor. The “Bone Weights” setting has been set to 1.0 by default and this is what made simplification consider the bone weights of the mesh.



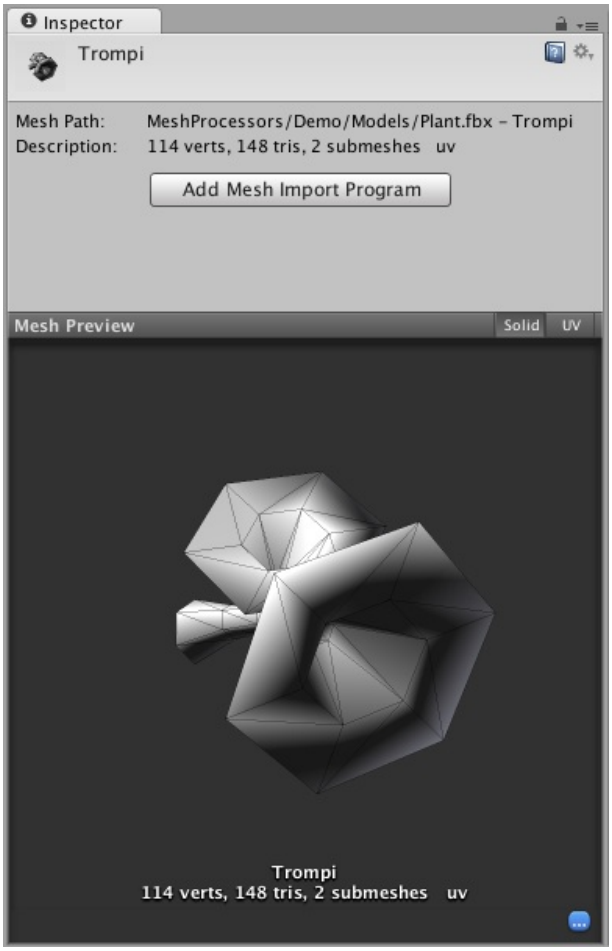
4.3 Import subdivision with crease detection and platform-dependent simplification of a static low-poly mesh.

1. Locate the “Trompi” mesh inside the “Plant” model in the project folder. The framework contains a shortcut. Select the plant object in the scene view and hit Shift-Ctrl-M (Win) / Shift-Command-M (Mac). This will open the mesh inspector for the selected objects first mesh.
2. Create the mesh import program as in the previous tutorials.
3. Add a “Subdivide (Quads)” processor and click “Apply”. The mesh will be subdivided once.
4. Add a “Detect Creases” processor, then click on “Up” to move it above the subdivision processor so it is executed before the subdivision.
5. In “Detect Creases” Enable the “Edge Angles” option and set the “Min Edge Angle” to 90, then click “Apply” again. The mesh will now have a sharp rim on top.
6. Go back to the subdivision processor and increase the iterations to 3, then click “Apply” again. The mesh is now very smooth in the scene view and consists of 9536 triangles.
7. Click the iPhone (iOS) icon above the iterations field. Click on “Override for iPhone” and set the iterations to 2 and click “Apply”. If you now switch Unity3D to build for iPhone (under “File/Build Settings...”), the model will be reimported automatically and will only be subdivided with 2 iterations while keeping the 3 subdivisions for all other build target platforms.



5 Mesh Inspector

5.1 Mesh Information/Adding programs



Krabl Mesh Processors extends the Unity3D built-in mesh inspector.

Mesh Path This is the path to the model file that contains the mesh relative to the project assets folder. At the end of the line, the mesh name is displayed after a "-".

Description The description of the mesh in its original state before any processing is applied by the mesh processors. This allows you to compare the basic mesh parameters before and after processing.

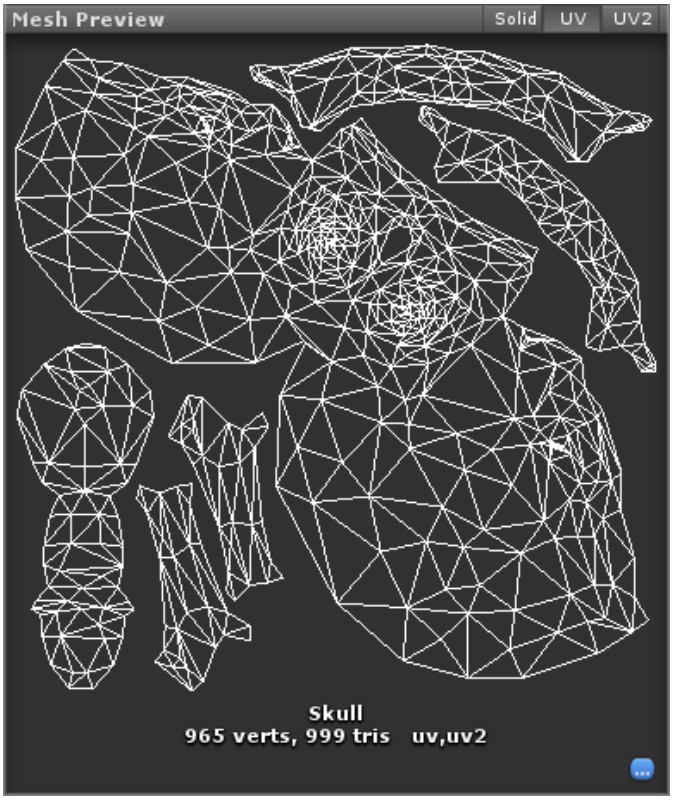
Add Mesh Import Program' This button creates a mesh import program if the mesh does not have one yet. This will also create the MeshImportProcessorSettings file if it does not exist.

5.2 Menu/Keyboard Shortcut

Krabl Mesh Processors includes an editor script (KMLocateMesh.cs) which adds shortcuts to get to the mesh asset inspector belonging to a selected object. It adds a menu entry to the “GameObject” menu called ‘nspect Mesh Importer for selected Mesh”. The same function can be triggered with a keyboard shortcut: shift-ctrl-M on Windows and shift-command-M on Mac OS X.

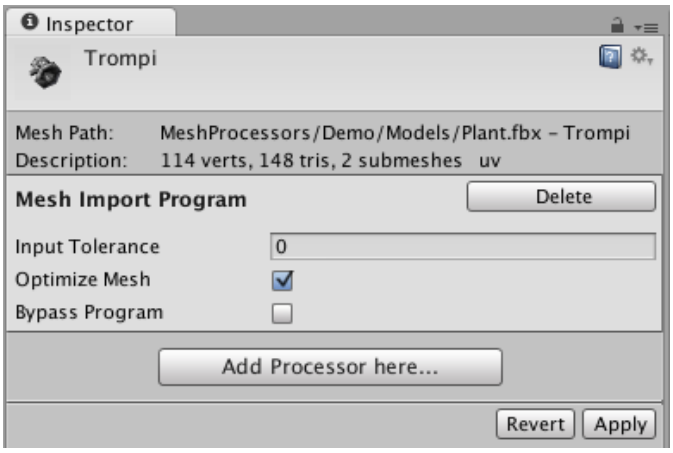
To quickly get to the import settings for a mesh visible in the scene, select the GameObject with that mesh and hit the keyboard shortcut . The correct mesh inspector will show up in Unity’s Inspector pane. This works best if only one mesh is inside the selection.

5.3 UV displays



The mesh inspector extension includes the option to view the UV coordinates of your meshes if they have any. On the right side of the Mesh Preview title you get the option to switch between the regular (Solid) view as well as UV and UV2 displays. If a mesh has UV coordinates outside the 0-1 range, the displays zoom out, otherwise they always show the 0-1 range.

5.4 Program settings



Once a program has been created for a mesh, the basic program settings are displayed.

Delete Delete the mesh import program for this mesh. This is not undoable and thus you will be prompted to confirm in case any processors (with settings) have been created.

Input Tolerance The maximum difference considered equal for vertices and normals when they are fed into the processor chain. Usually 0 should work fine, but apparently some modeling software produces only nearly equal vertices/normals. If weird topology is produced, try setting the value to something like 1e-6.

Optimize Mesh Let Unity3D improve the mesh element order after processing to render faster on the GPU. It is quick and highly recommended.

Bypass Program Do not do any processing and just output the input mesh. This is useful to compare the original mesh with the processed mesh.

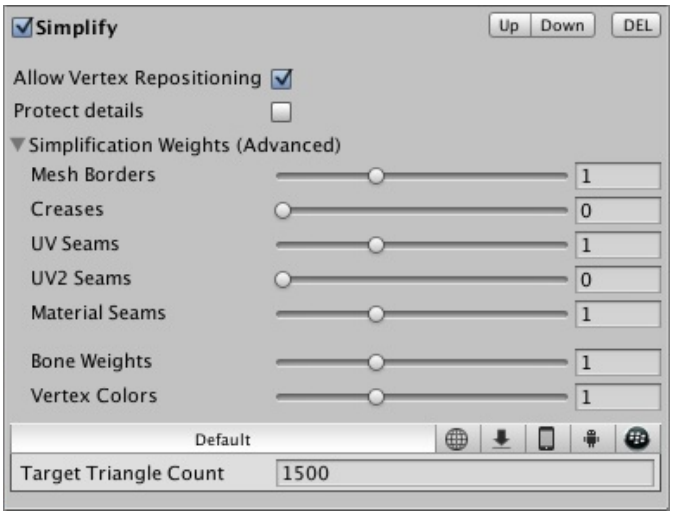
Revert Revert the current changes to this mesh import program. This include processor settings and order as well as global mesh settings, but not creation/deletion of the program.

Apply Apply the current changes to this mesh import program. This will save the changes to disc and trigger a reimport of the model which contains the mesh.

6 Import Processors

The processors described here can be used in any order in any mesh import program. They are created by first creating a mesh import program in the mesh inspector and then clicking of “Add processor here...” and choosing the processor to create.

6.1 Simplify



The “Simplify” processor performs mesh simplification based on edge collapses. It can simplify a mesh to a target number of triangles. The processing of mesh attributes can be configured. The target triangle count can be overridden per build target platform.

Simplify (Up|Down) (DEL) Click the processor title or the checkbox on its left side to enable/disable the processor in the chain. Use Up and Down to move this processor in the chain and change the processing order (always top to bottom). Click DEL to remove this processor.

Allow Vertex Repositioning Calculating new vertex positions leads to a more accurate mesh shape but can have a negative impact on mesh attributes (UVs, skin, colors, ...) because interpolation is needed. Disabling Repositioning will speed up calculations.

Protect Details Enable to prevent small independent mesh features to be totally removed. Technically this will prevent the creation of non-manifold edges during the simplification. Protecting details is not recommended for low target polygon counts.

Simplification Weights(Advanced) Extend the weights section by clicking on the small triangle to the left. This is a group of parameters that affect how the mesh is simplified. They determine which edges get collapsed first and allow to preserve special features of a mesh.

Mesh Borders The amount of protection for edges at mesh borders (edges only connected to one face at the borders of a mesh).

Creases The Amount of protection for edges that have been marked as creases. At the beginning of processing all edges with normal breaks are automatically marked as creases. Different edges can be marked as creases with the “Detect Creases” processor.

UV Seams The amount of protection for UV seam edges. These are edges which are part of multiple separate UV islands. If this value is set very high, the shape of UV islands will not change at all.

UV2 Seams The amount of protection for UV2 seam edges. These are edges which are part of multiple separate UV2 islands. If this value is set very high, the shape of UV2 islands will not change at all.

Material Seams The amount of protection for edges connecting faces with different materials.

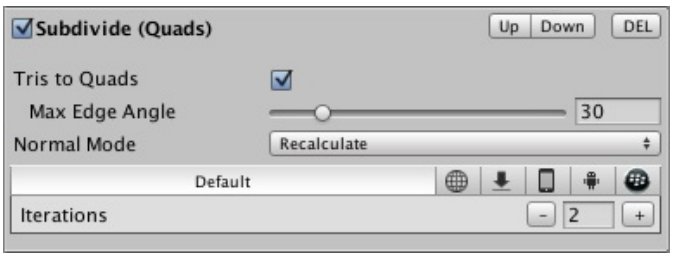
Bone Weights The amount of protection for edges which have different bone weights at their two vertices. The higher this value, the less flexible areas of skinned meshes get simplified.

Vertex Colors The amount of protection for edges which have different vertex colors at their two vertices. The higher this value, the less areas with vertex color changes get simplified.

Target Triangle Count The number of triangles the result mesh should have. Simplification stops once the mesh has this amount or less triangles. As an edge collapse usually removes two triangles, the result mesh might have a few triangles less than this value.

Above the target triangle count field is a build target platform selection bar where you can override this settings to different values for each build target platform Unity3D supports.

6.2 Subdivide (Quads)



The “Subdivide (Quads)” processor applies iterations of quads-based subdivision to the mesh. Optionally it can merge triangles to quads before the subdivision. The number of iterations can be overridden per platform.

Tris to Quads If this settings is enabled, the processor will merge triangles to quads before the subdivision. Quads generally lead to better subdivision results. Quads are produced by merging two neighbouring triangles by dissolving their connecting edge.

Max Edge Angle If “Tris to Quads” is enabled, this defines the maximum angle between two triangles that can be merged to a quad.

Normal Mode How to calculate the vertex normals after subdivision.

Interpolate Linearly interpolate normals of the input mesh during the subdivision. If there is something special about the input normals, this option will most likely preserve it.

Recalculate Calculate new normals based on the mesh shape once the subdivision is complete. This is the recommended option as it leads to the normals conforming to the mesh shape.

Iterations How many times to execute the subdivision algorithm. For every execution, each face is split into four triangles. High iteration counts lead to meshes with more vertices/faces than Unity3D can handle (The limit is 65k).

6.3 Detect Creases

The “Crease Detect” processor marks edges as creases on adjustable conditions. It will recalculate the vertex normals of the mesh if necessary. It is especially powerful when placed above s subdivision processor as the crease information greatly affects subdivision.

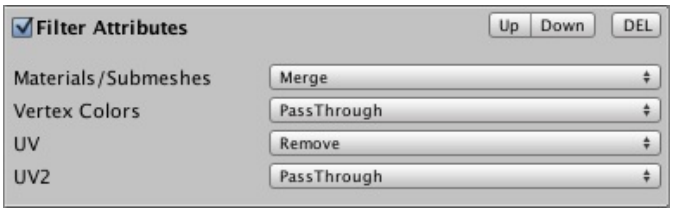
Mesh Normals Compare the vertex normals of the faces connected to each edge. If they are not the same, mark the edge as crease. This is already automatically done at the beginning of mesh processing. It preserves what it known as “smoothing group“ or “edge split”. The normal comparison is sensitive to the programs “Input Tolerance” setting.

Edge Angles Calculate the angle between the faces connected to each edge and mark every edge with an angle above the threshold as crease.

Min Edge Angle If “Edge Angles” is enabled this determines the minimum edge angle to mark as a crease.

Material Seams Every edge connected to faces of different materials/submeshes is marked as a crease.

6.4 Filter Attributes



The “Filter Attributes” processor allows to remove certain mesh attributes.

Materials/Submeshes If set to “Merge”, all faces of the mesh will be set to the same material/submesh.

Vertex Colors If set to “Remove”, vertex colors are removed from the mesh.

UV If set to “Remove”, UV coordinates are removed from the mesh. Most shaders need UV coordinates.

UV2 If set to “Remove”, UV2 coordinates are removed from the mesh. UV2 coordinates are usually used for lightmaps in Unity.

7 Krabl Mesh Library Scripting

The following example scripts show how to process meshes with the KrablMeshUtility class inside C#, UnityScript and Boo scripts. More more in-depth scripting information, please read the included Krabl Mesh Library html documentation.

The C# example simplifies the mesh of a MeshFilter on the same GameObject to a fraction of its triangle count when the game is started:

```
1  using UnityEngine;
2
3  [RequireComponent(typeof(MeshFilter))]
4  public class KMDemoSimplify: MonoBehaviour {
5          [Range(0.0f, 1.0f)]
6          public float factor = 0.5f;
7
8          void Start () {
9                  MeshFilter mf = GetComponent<MeshFilter>();
10                 Mesh mesh = mf.mesh;
11                 int numTriangles = mesh.triangles.Length/3;
12                 KrablMeshUtility.SimplifyMesh(mesh, ((int)(factor*((float)numTriangles))));
13
14                 mf.mesh = mesh;
15         }
16 }
```

Listing 1: KMDemoSimplify.cs (C#)

The UnityScript example subdivides an attached MeshFilter/Mesh on game start:

```
1  #pragma strict
2
3  @script RequireComponent(MeshFilter)
4
5  @Range(1, 4)
6  var iterations : int = 1;
7
8  function Start () {
9          var mf : MeshFilter = GetComponent(MeshFilter);
10         var mesh : Mesh = mf.mesh;
11         KrablMeshUtility.SubdivideQuadsMesh(mesh, iterations, true);
12         mf.mesh = mesh;
13 }
```

Listing 2: KMDemoSubdivide.js (UnityScript)

The Boo example converts the mesh normals to produce a flat shaded look on game start:

```
1  import UnityEngine
2
3  [RequireComponent(typeof(MeshFilter))]
4  class KMDemoFlatShade (MonoBehaviour):
5
6          def Start ():
7                  mf as MeshFilter = GetComponent[of MeshFilter]()
8                  mesh as Mesh = mf.mesh
9                  KrablMeshUtility.FlatShadeMesh(mesh)
10                 mf.mesh = mesh
```

Listing 3: KMDemoFlatShade.boo (Boo)

For a glimpse at how the library works in more detail, here is the part of KrablMeshUtility which provides the simplification method. In essence, the library always needs to convert a mesh coming from Unity to its own data-structure, then do the processing and then convert back.

```
1   public static void SimplifyMesh(Mesh unityMesh, int targetFaceCount, bool highQuality = true) {
2       KrablMesh.MeshEdges kmesh = new KrablMesh.MeshEdges();
3       KrablMesh.Simplify sim = new KrablMesh.Simplify();
4       KrablMesh.SimplifyParameters simpars = new KrablMesh.SimplifyParameters();
5
6       KrablMesh.ImportExport.UnityMeshToMeshEdges(unityMesh, kmesh);
7       simpars.targetFaceCount = targetFaceCount;
8       simpars.recalculateVertexPositions = highQuality;
9       simpars.checkTopology = !highQuality;
10      simpars.maxEdgesPerVertex = highQuality ? 18 : 0;
11      if (highQuality == false) {
12          simpars.preventNonManifoldEdges = false;
13          simpars.boneWeightProtection = 0.0f;
14          simpars.vertexColorProtection = 0.0f;
15      }
16
17      sim.Execute(ref kmesh, simpars);
18      KrablMesh.ImportExport.MeshEdgesToUnityMesh(kmesh, unityMesh);
19  }
```

Listing 4: SimplifyMesh method of KrablMeshUtility.cs (C#)

If you need more information about library scripting, please check out the included “Krabl Mesh Lib Reference HTML.zip” or browse the C# source code. The latest library reference can also be found here: `http://www.krabl.com/meshprocessors/librarydocs`.