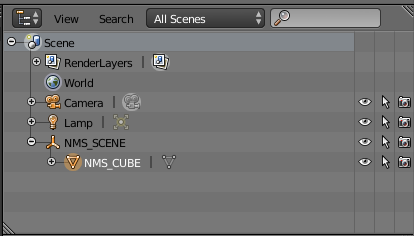
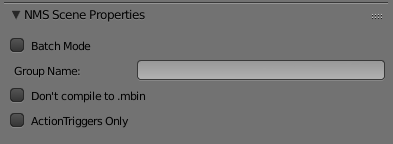
**NMSDK tutorial**

**Installation:**

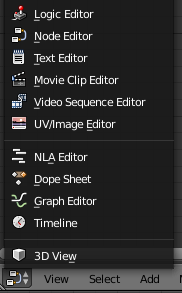
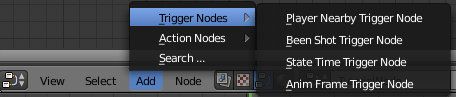
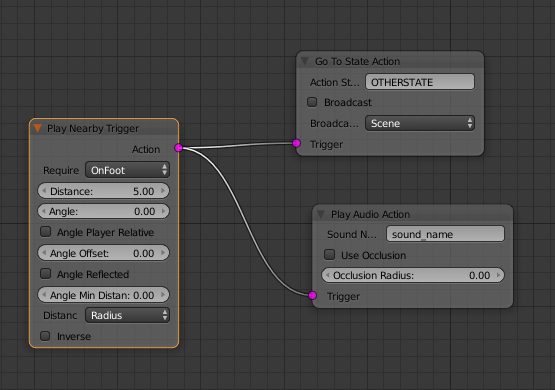
1. Place all files downloaded from GitHub in your Blender folder (usually located at C:/Program Files/Blender Foundation/Blender)
2. Open blender, and open the User Preferences (ctrl+alt+U), then press ‘install from file’ and select the addon\_script.py file.  
   
3. Enable the tickbox to activate the plugin, then close the window. You are now ready!

**Usage:**

1. **Every** scene produced in blender that you want to export using NMSDK **must** have a NMS\_SCENE object which every other object is a child of. This can be any object type, just as long as it is placed at the origin with all transforms set as default (0 across the board).  
   
2. All objects in the scene need to have the naming convention *NMS\_<name>* where *<name>* is whatever you want the object to be called in the games files. It is good practice to keep this name at less than 16 characters (not including the *NMS\_*), never use spaces in the name, and try and name the object something descriptive/memorable.
3. In NMS there are various object types. When you create a new object in blender with the prefix *NMS\_* blender will recognise that you are creating a NMS object, and create some extra panels that will allow you to enter data.  
   The *NMS\_SCENE* object has its own special panel:  
     
   *- Batch Mode*: If selected, each direct child of *NMS\_SCENE* will be exported separately.  
   *- Group Name:* If set, the exported objects will have one of their paths changed to the group name. This is to allow grouping of similar scenes in the games files if required.  
   *- Don’t compile to .mbin*: Pretty self-explanatory. All files produced except the geometry file will stay as .exml files. Note that due to the use of a built-in xml parser, these files are a single line, and have no formatting. There is a *prettify.py* file that can be used to convert these if required, or you can convert to mbin and back to exml if you cannot run this program (requires the lxml python package to run)  
   - *ActionTriggers Only*: If you want to just export the associated ActionTriggers (more on this later!)  
     
   All other NMS objects have this panel:  
     
   as well as one or more panels below it depending on the object type selected.  
   There is no restriction on what type of object is used in blender, however a *Mesh* and *Collision* type object should always be assigned to something with actual mesh data otherwise the export process will have an error. It is recommended to just use an empty axis object in blender for the *Locator*, *Reference* and *Joint* object types, and a point lamp for the *Light* object.
4. In any scene, you only really require one object in the scene to have an entity. This can be enabled in either the *Mesh* or *Locator* panels by selecting the ‘Requires Entity’ tickbox.  
   At the moment the *Entity* panel has only two options:  
   - *Is animation controller*: Select as true on **only one** entity in the scene. This property is required if animations are to be used.  
   - *Has ActionTrigger*: Select if you wish this entity to receive all the ActionTrigger data.

**Action Triggers:**

To create custom action triggers in blender, we make use of the node editor that is usually used to modify shaders. Here we have created a custom set of nodes that allow for visual creation of ActionTriggers.

1. To make use of this feature, first open the Node Editor window:  
   
2. Next, select the NMS node editor type from the shown options (highlighted in blue):  
     
   and press new to create a new ActionTrigger which you need to give a name. This name you enter is the name of the ActionTrigger itself. So try and keep it less than 16 characters and have no spaces in the name. You may be using this name in other action triggers to call it, so again, make it descriptive if possible. It is recommended that you give the ActionTriggers names that are entirely in upper case.  
   
3. ActionTriggers are made up of two parts; a *trigger* and an *action*. The *trigger* is what the game checks to see if the action(s) is (are) called. The *actions* are what the game does if the *trigger* has been triggered.  
   We create *actions* and *triggers* by pressing the ‘Add’ button:  
     
   Each ActionTrigger can only have one *trigger*, but as many actions as you want.  
   Add the required *actions* and *trigger* for this ActionTrigger and join the purple dots together (not required, just for a visual aid…)  
     
   As you can see there are quite a few different variables here. I will not go into any of them really, however if you hold your mouse over any of them, a description will appear. Note that some of the effects are unknown, so the description may indicate that.

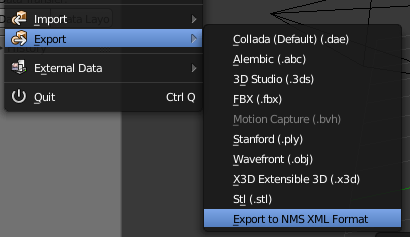
**Animations:**

Animations can be added to NMS scenes by creating an animation in Blender.  
Due to how NMS organises objects that are animated, a few guidelines must be followed for animations to work correctly:

1. Any object that is to be animated **must** be the direct child of a *Joint* object. It is this joint object that the animation is applied to, not the actual object itself. This sounds odd, but if you create the *Joint*, then add whatever you want to be animated as a child to this object, then create the animation on the joint, the children will move as expected.

It should be noted that at the moment only simple animations are possible. I.e animations that only involve translation, rotation and scaling of a mesh. The mesh itself cannot be modified in any way directly. Any direct modification such as moving any number of vertexes for example will simply be ignored by the exporter.

**Exporting your scene:**

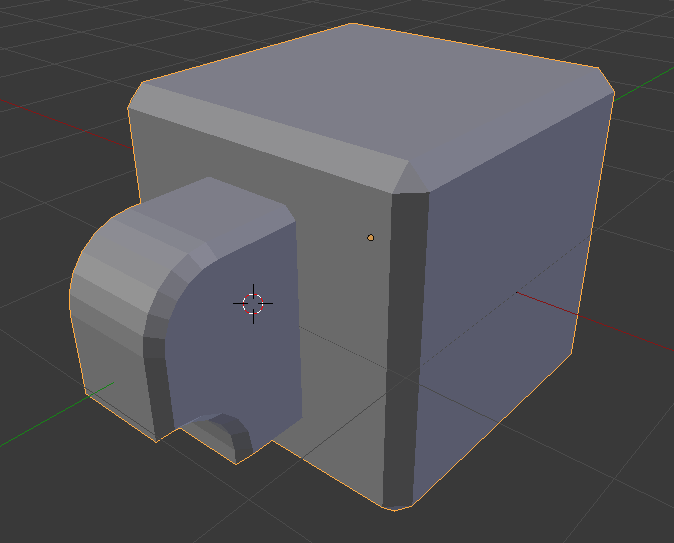
To export your scene, simply select the file dropdown menu and click the export option:  
  
This will open up a dialogue box letting you chose where to save the exported data and what the name is.  
Enter the required information and press ‘Export to NMS XML Format’ and wait for your export to complete!

**Modelling Best Practices**

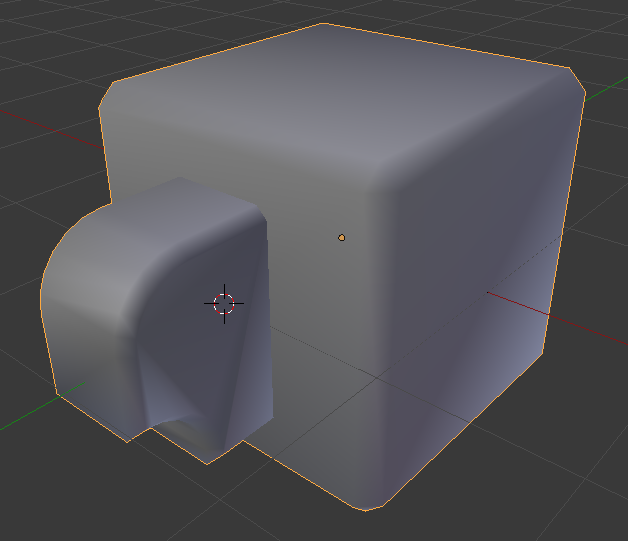
The following are general hints for the construction and/or import of custom meshes via NMSDK. The goal of these hints is to further refine one’s modelling process in order to make better-performing, storage-efficient models that are glitch-free and does not require the modeller to rely on more experimentation to find out what’s wrong. These hints are also based on observed methods Hello Games has created their own models.

1. **EDGE-SMOOTHING**

Edge smoothing is an essential component to look at when creating models, especially hard-surface models. Game engines such as Unity and Unreal Engine have methods for dealing with edge smoothing upon importing a new asset but Blender and NMS does not. A common error is for one to build a model with hard edges that looks like this:



Upon import, however, it instead renders like this in-game:



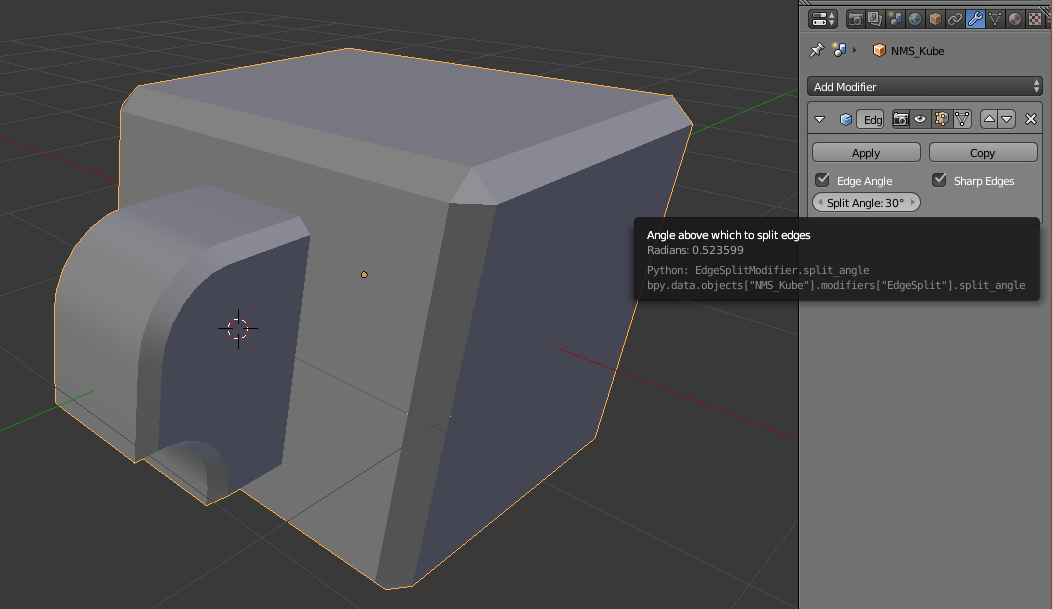
This is caused by the lack recognition for smoothing groups or sharpened edges within NMS’s geometry format. The way the NMS geometry format ‘fakes’ smoothing is actually purely done by the modeller’s method of construction. Contiguous faces in a geometry are file are considered *entirely smooth* but a method of cheating them is by **ripping** edges, imagine using a scissor to cut a seam perfectly through an object to force a sharpened edge to appear.

There are several solutions available. First, select your object and type **Shade Smooth** in the text context menu:



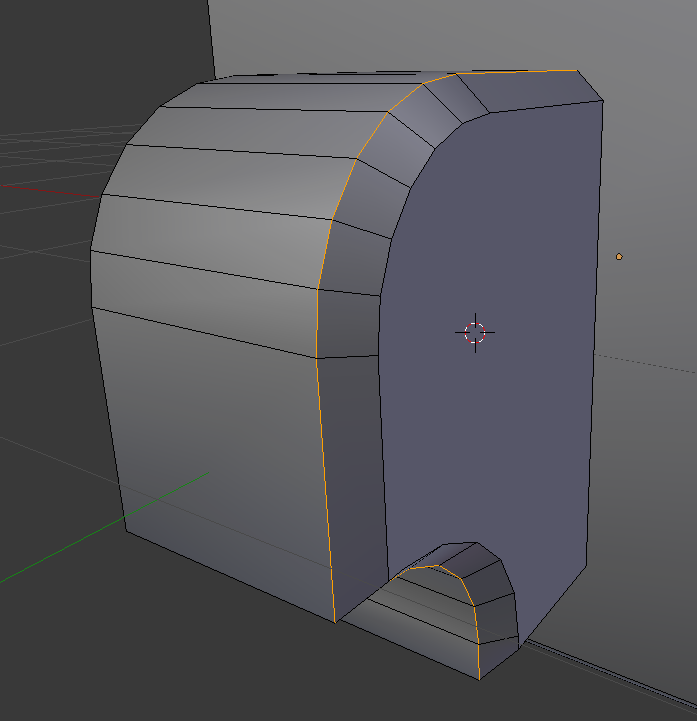
Enabling shade smooth will allow you to preview your object in the way it will actually be rendered in-game.

1. Use the **Edge-Split** modifier to automatically generate initial splits for your object.

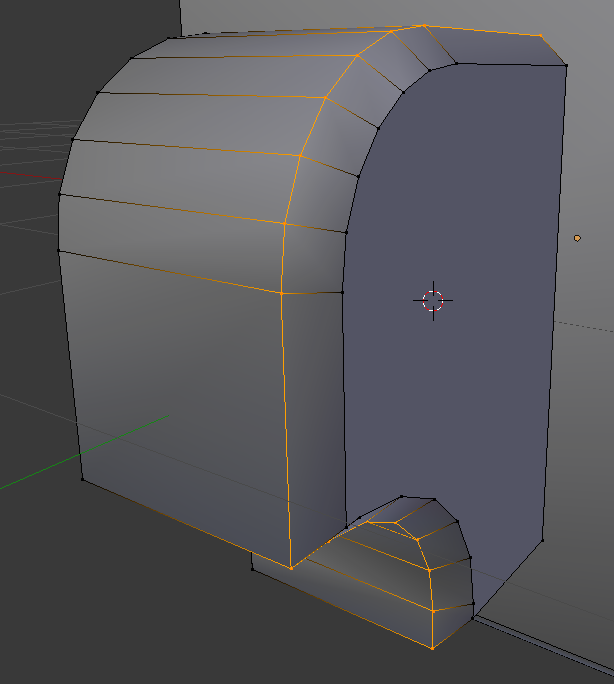


As visible here, **with just a simple modifier**, your strange ‘melted’ looking object now contains sharpened edges and smoothened contours wherever needed. Be sure to **apply** the modifier before hitting export.

1. Use the **Rip (shortcut: V)** command to cut selected edges manually. This can be done after you have applied your Edge-Split modifier.



You can perform the reverse by using **Remove Doubles**. Select the vertices you wish merged. Remember that because you ripped the faces, there are now multiple overlapping vertices on all ripped edges. This is perfectly fine but sometimes, the Edge Split modifier may create splits where you don’t want them.



Before exporting a model that uses edge-splitting like this, be sure to **NOT CHECK THE CREATE TANGENTS CHECKBOX** or the model importer will explode your mesh into individual faces and you will lose all smoothing information entirely.

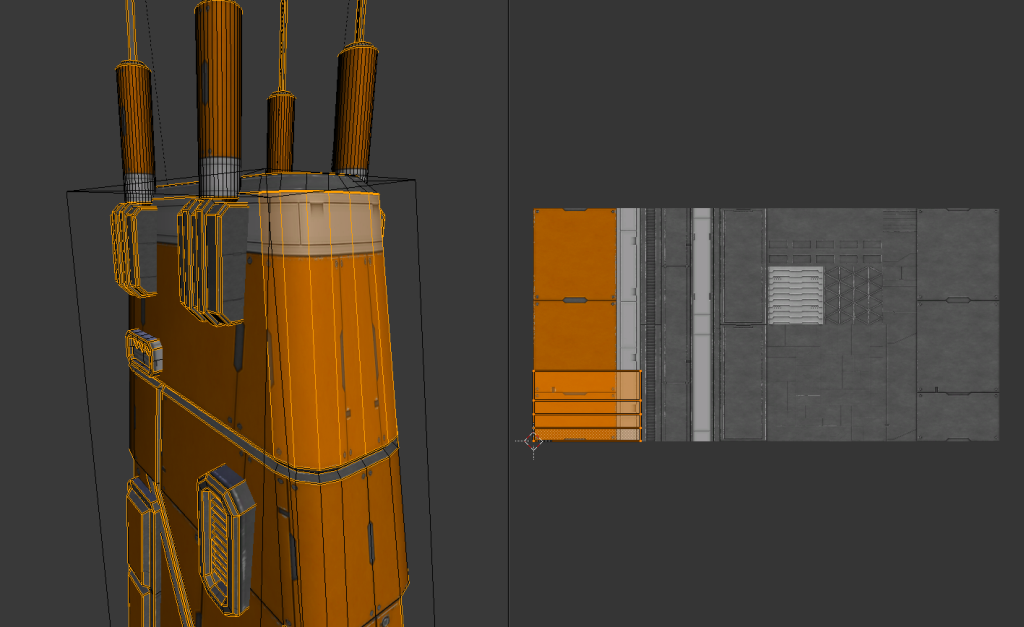
1. **USE TILING TEXTURES WHENEVER POSSIBLE**

NMS prides in keeping file storage requirements extremely low. **It is only proper for custom models and textures to do the same.**

Reliance on tiling textures requires a completely different method of unwrapping and texturing objects whereas…

1. **Instead of unwrapping first, create the textures first.**
2. **Instead of simply using unwrap, you will use Blender’s Follow Active Quads method of unwrapping to map your meshes faces against that of an existing tiling texture.**
3. **Maybe if it’s not necessary, just use a texture that already exists from the game.**

The following example is a custom model that uses the game’s pre-existing procedural freighter material. Instead of having a model with a **25MB** footprint for the geometry itself, and 2k x 2k PBR materials (albedo map, gloss map, metalness map, normalmap), this model only uses **200-300kb** of storage space after PSARC (PAK) compression.



Almost all of Hello Games’ original models are created this way with few exceptions.

1. **LOWER POLY COUNT = BETTER PERFORMANCE**

Despite modern graphics being entirely capable of rendering millions of triangles on-screen with even mobile GPU’s beginning to match up to the power of PC’s, **polycount budget is not infinite.** Representing your model with the **least amount of polygons is always the best case scenario on anything that will involve your model having to appear in real-time.** To set benchmarks:

1. A standard lush tree prop in NMS has 5k – 7k triangles.
2. A standard ship in NMS has 20-30k triangles.
3. A single Vy’keen warrior has 35-45k triangles.

Reducing your polycount will not only allow more objects to render and improve performance in-game, it will also **reduce your** **export compile time.** Budget your polygon count in the context in which your model will be used – a prop that is expected to populate your screen 500x over will strictly require far, far fewer polygons while a massive single structure in view can possess far more polygons as an individual object.