BCry Exporter

Document

Version 0.9.1

Contents

Introduction

BCry Exporter *was originally developed by* ***AFS Studio*** *(http://bcry.afcstudio.org) and the GitHub address is https://github.com/AFCStudio/BCRYExporter*

Leonidas White did wonderful work to update the plugin to work with Blender 2.8+. The GitHub address is https://github.com/LeonidasWhite/BCRYExporter

This repository is forked from Leonidas White's version and available on my GitHub page: https://github.com/brickengineer/BCRYExporter

My name is Wang, Zhen and I am an indie game developer from Cryengine community. I forked this new repository because it was no longer compatible with Blender 2.9+. Also, I want to improve it and provide long-term maintenance.

The AFS Studio's website has been offline for a very long time. Fortunately, I happen to have a rough copy of the parts of documents when it was still online. So, I decided to organize and improve these documents for this repository.

NOTE: This version does **NOT** support Blender 2.8 or earlier because python in Blender 2.9+ has been updated to python 3.

*Currently this document is still rough. But it should be able to help users to get used to the BCry Exporter plugin work pipeline. Or at least get started to work with Blender and Cryengine.

Installation

Install BCRY Exporter:

Just copy **io_bcry_exporter** folder to *Blender|Scripts|Addons* directory.

Directory Example:

D:|Program Files|Steam|SteamApps|common|Blender|<version>|scripts|addons

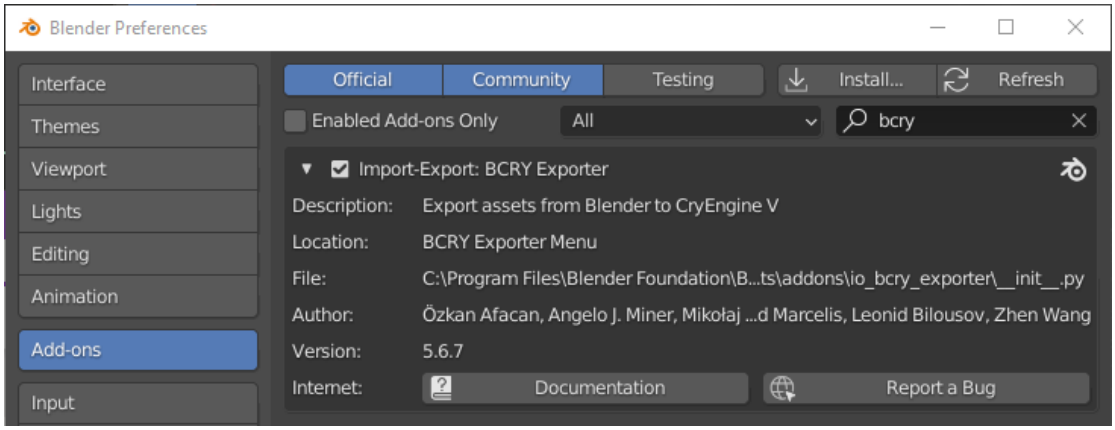
Or *%appdata%|Blender Foundation|Blender|<version>|scripts|addons*

Or your custom Blender installed location.

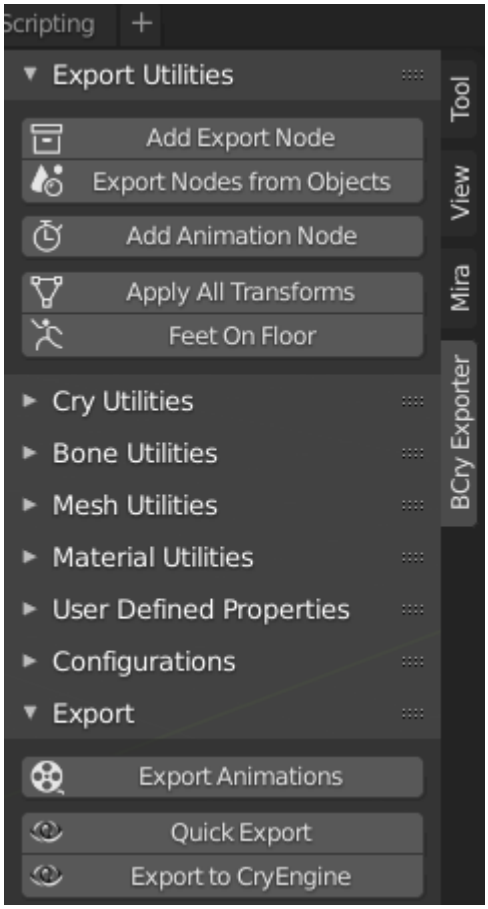
After you copy the BCRY Exporter on one of the above locations. Now, you need to enable it.

Enable BCRY Exporter:

1. Open Blender (If already opened, close and reopen it)
2. Open **Edit/Preferences/Add-ons** menu.
3. Type **BCRY** to **search** bar.
4. Enable tick to activate **BCRY Exporter**.



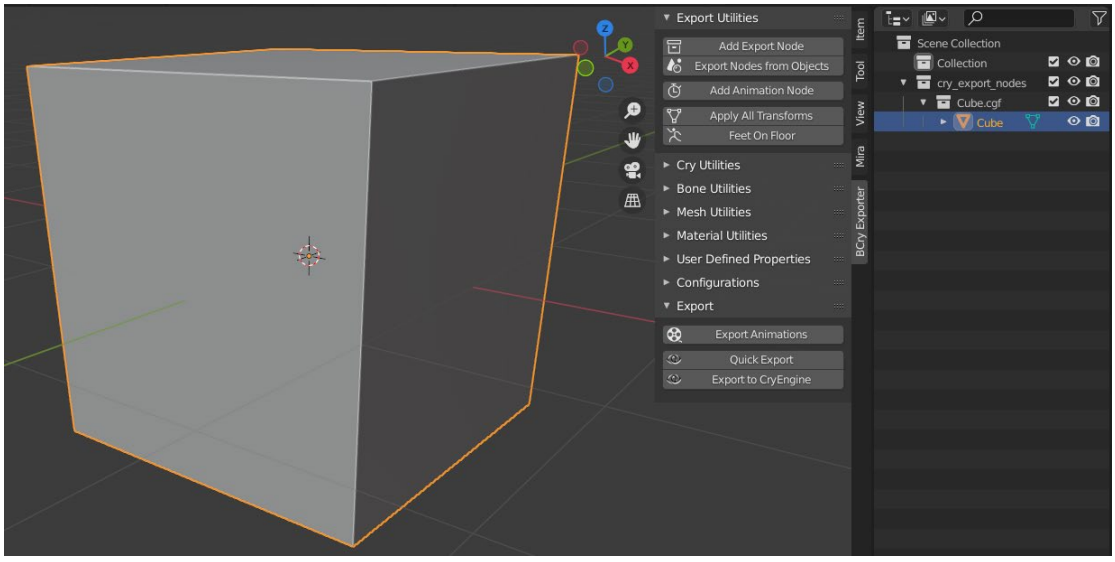
If the plugin is successfully enabled, you can access it from the tool shelf panel on the right side of perspective view. (If you can’t see this panel at all, press ‘N’.)



Static Mesh Exporting

Steps:

- Select the object to be exported.
- Click **Add Export Node** from the BCry Exporter tool panel.
- Select **CGF** as **Type** and enter node name (the name which actually appears in CryEngine).
- Press **Alt + G, Alt + R, Alt + S** to clear its transformation
- Press **OK**.
- Click **Export to Cryengine** from Bcry Exporter menu and select destination path



Animated Geometry Exporting

The animated geometry files are represented with **CGA** (Crytek animation file) extension in CryEngine.

To export CGA file:

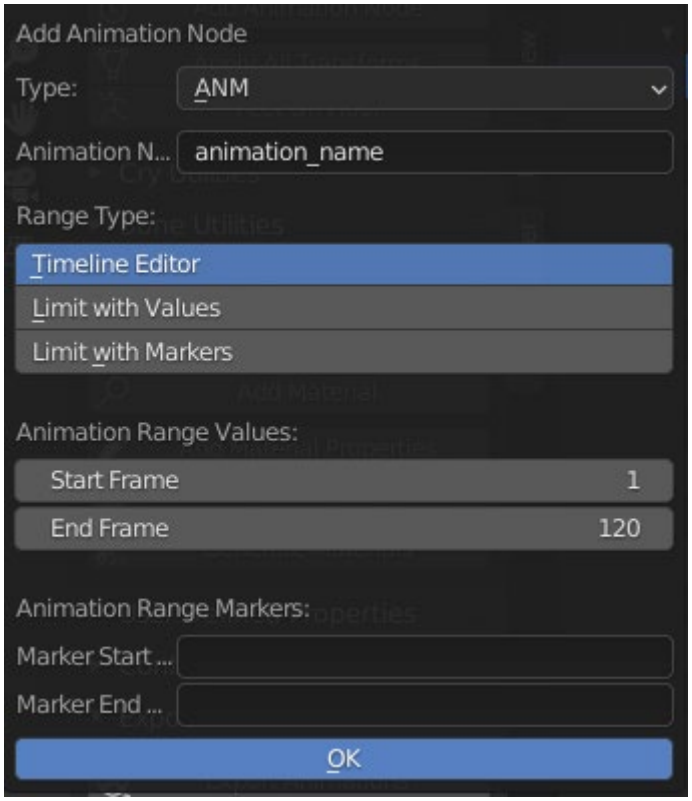
- Move to the first frame where animation starts
- Select object or objects to be exported.
- Select **Add Export Node** from Bcry Exporter menu.
- Select **CGA** as **Type** and enter node name (the name which will appear in CryEngine).
- Press **OK**.
- Click **Export to Cryengine** from BCry Exporter menu and select destination path

To Export Animations:

By Cryengine's default, each **CGA** file can store one animation which is named as *Default* (in CryEngine). Other animations are stored in **ANM** files. However, to preserve consistency in BCRY Exporter, **CGA** files do not store any animation. They only hold mesh information. If you want to use *Default* animation, you can add a **ANM** which named as *Default*. To be short, you **always** need ANM files to export animations even if there is only one animation.

- Select your CGA object.
- Select **Add Animation Node** from Bcry Exporter menu.
- Select **ANM** as **Type** and enter node name (the name which will appear in CryEngine).
- Enter range of animation.
- If you select Limit with Values as range type, set start and end frames.
- Press **OK**.
- You can repeat above steps for each animation.

- Click **Export Animations** from BCry Exporter menu and select destination path (ANM files must stay under the same folder with CGA file).
- Press **Export Animations**.



Set Up in Cryengine:

- Root mesh (including single mesh) will not play move animation in Character Tool (rotation animation will play). But it will play correctly when you put it into game world. This is not a bug.
- If you drag an animated mesh (.CGA file) from asset browser to game world directly. The engine will create an entity with **Animated Mesh Component** and **Rigid Body Component** to for it. DO MAKE SURE YOU CHECKED **Use Raw Animation Name** (Then select any other entity and then select back to this entity, the animation selection are should be changed to a drop down select menu).

Additional Information:

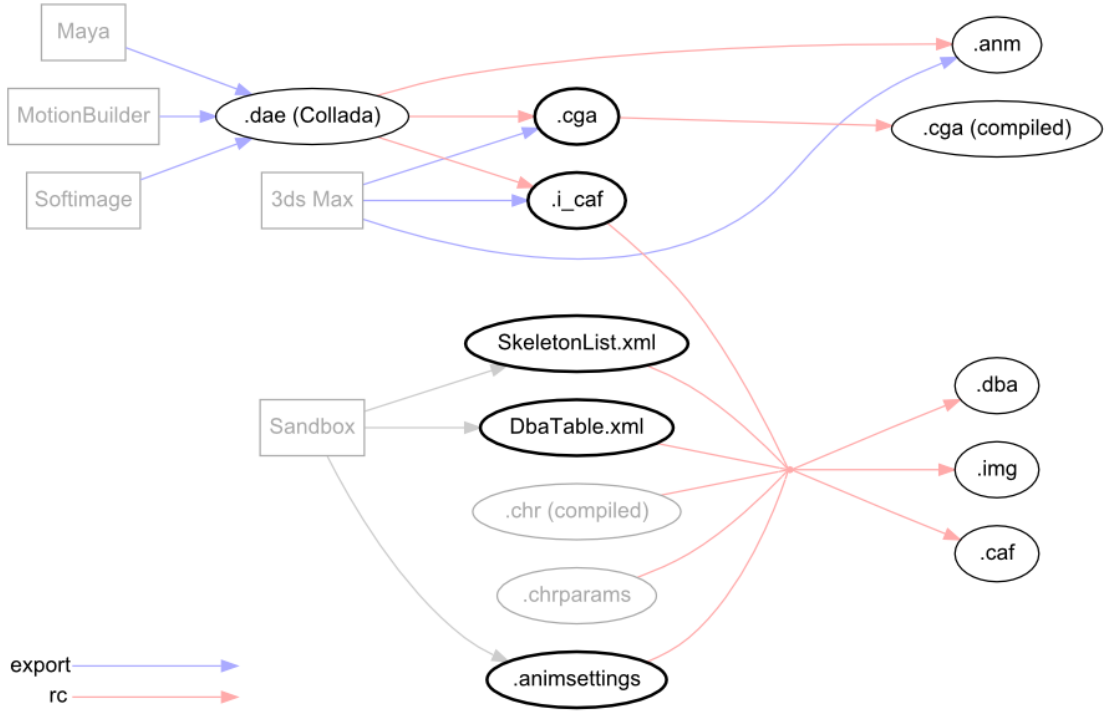
Each ANM file requires a specific naming convention. As it says in CryEngine official documents:

- **ANM** files need to be contained in the same folder as the CGA object they were exported from and require a specific naming convention to work with them. Each additional ANM file added needs to contain the parent CGA's name as a prefix. For example:
  - **Vehicle_b.cga** – Vehicle object(s) and default animation.
  - **Vehicle_b_door1_exit.anm** – Animation keys stored for the door open animation when a character exits the vehicle.
  - **Vehicle_b_door1_enter.anm** – Animation keys stored for the door open animation when a character enters the vehicle.
  - **Vehicle_b_door2_exit.anm** etc.

Fortunately, BCry Exporter can do name convention for you. When you export ANM files with BCRY, final animation name will be *CGANodeName_ANMNodeName.anm*

If your objects don't have a CGA node, the animation name will be converted to *ObjectName_ANMNodeName.anm*

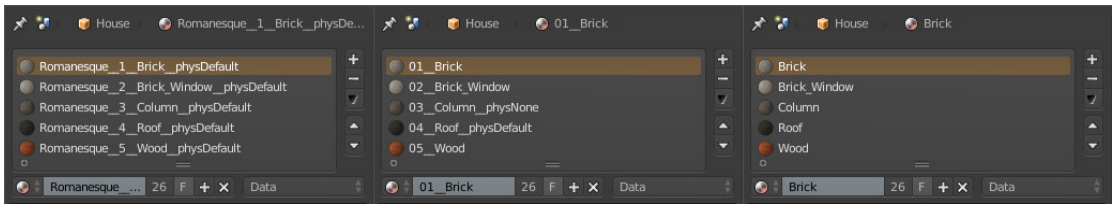
Animation convert process:



Generate Materials

BCRY support 3 types of material exporting:

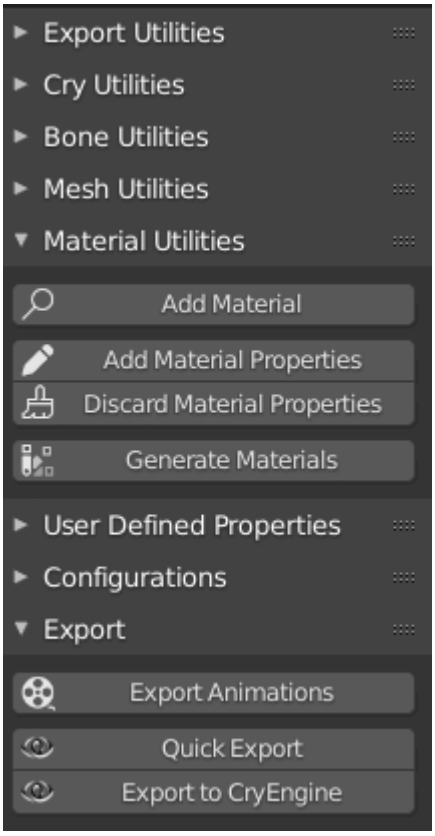
1. *parentName__materialNumber__materialName__physicalProxy*
2. materialNumber__materialName
3. *materialName* (In this case, material index is assigned automatically.)



Only one naming rule can be used for each single Blender project file. You can use **Material Utilities -> Add Material Properties** tool to convert material names for active object.

Generate Materials

When you choose material exporting type and set ready materials, you can export materials one of two way:



1 – Generate Materials Tool

You can use **Generate Materials** tool under **Material Utilities** Menu to create your materials whenever you want. That tool works like Maya or 3DS Max Generate Material Tool.

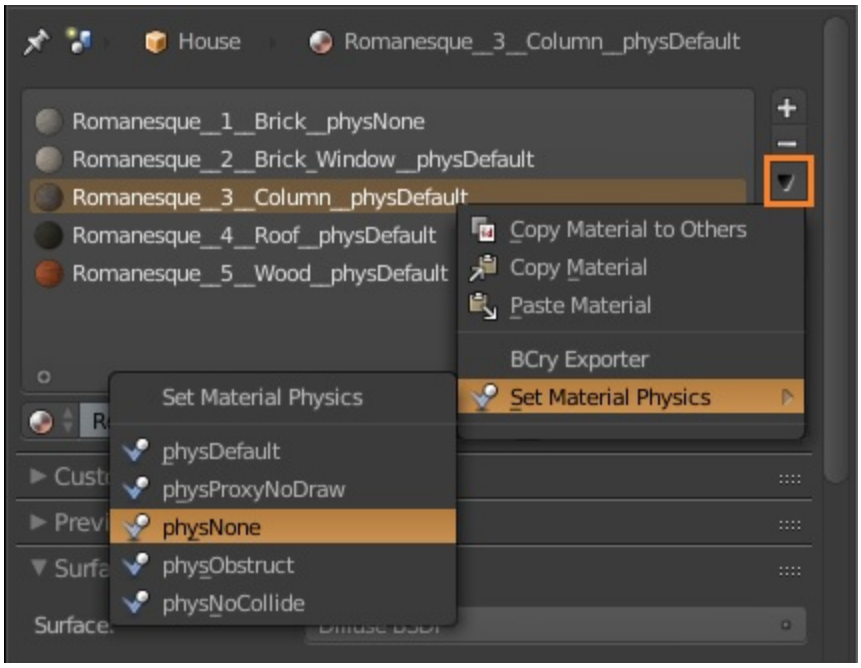
2 – Generate Materials while Exporting Assets

While exporting object from **Export to CryEngine** Tool, you can use **Generate Materials** checkbox to create materials. That

way may be handy when you export object firstly. That tool can work with **Export Selected Nodes** option.

Material Physics

You can set or change material physics for active material from **Set Material Physics** menu.



Textures

**NOTE:* *Textures exporting has not been tested yet, the following information is from original document. Import textures separately if this does not work as expected.*

BCry Exporter **Generate Materials** tool can also export texture paths (for Diffuse, Specular, Normal) both of Blender Renderer and Cycles Renderer materials. To do that, you need to consider some rules:

Blender Renderer:

You can use texture tab from properties editor to specify your textures for **Blender Renderer**. You can use influence panel to decide which texture to use which purpose (Diffuse, Specular, Normal).

- If **Color** checkbox is active under **Diffuse** tab, BCry export that texture as diffuse map.
- If **Color** checkbox is active under **Specular** tab, BCry export that texture as specular map.
- If **Normal** checkbox is active under **Geometry** tab, BCry export that texture as normal map.

Cycles Renderer:

You can use Texture Image Nodes from node editor to specify your textures for **Cycles Renderer**. To specify texture type, you have to set **Image Texture** node name as correctly for BCRY . You can set name active Image Texture node from Node tab under Node Editor **Properties Panel**.

- If node name is **Diffuse** or **Image Texture**, BCry export that texture as diffuse map.
- If node name is **Specular**, BCry export that texture as specular map.
- If node name is **Normal**, BCry export that texture as normal map.

Colors

BCRY **Generate Materials** tool also can export colors both of Blender and Cycles Renderers.

Blender Renderer:

You can set diffuse, specular colors, smoothness (hardness), alpha values from Material Properties.

Cycles Renderer:

You can set diffuse, specular colors, smoothness (hardness), alpha values from **Settings** tab under Material Properties for Cycles Renderer.

Smoothing

NOTE: *Currently* ***Custom Split Normals Data*** *does not work as expected after exported to Cryengine. It behaves like common auto smooth with a fixed limited angle (if checked). I will try to fix this. If you really need it, use FBX export for static meshes.*

BCry Exporter exports surface normals completely as you see in Blender. You may use only the smooth shading for your objects, or you may prefer to use composite of smooth and flat faces. Also, you can use sharp edges with **Auto Smooth** or **Edge Split Modifiers**.

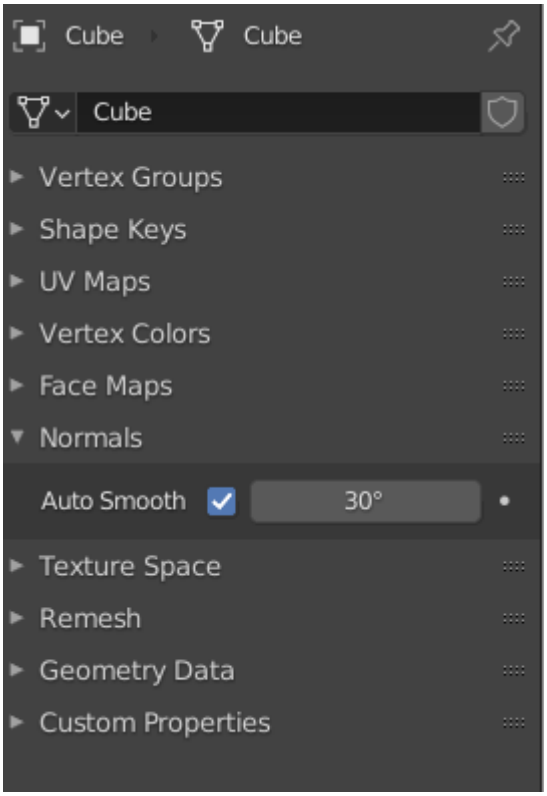
Composite Shading in Blender

Auto Smooth:

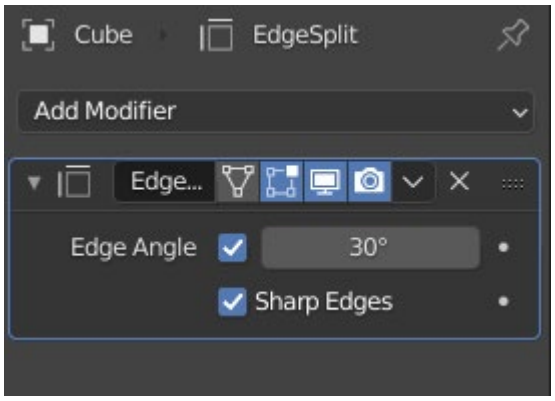
That is very handy and quick property, if you want to specify a smooth angle for your object. Auto smooth property also recognizes sharp edges automatically.

Edge Split Modifier:

With edge split modifier, you may specify the smooth angle. And you can enable or disable angle and sharp edges smoothing.



You can activate or deactivate edge modifier with using display modifier in viewport option.



Note: To use auto smooth or edge split modifier, the object shading type must be specified as **Smooth**. You can do that through **Object->Shade Smooth.**

Also, you can arrange the selected faces as smooth or flat. To achieve that:

1. Go to **Edit Mode**.
2. Select faces you want to make smooth or flat.
3. Run **Shade Smooth** or **Shade Flat** tool from **Mesh -> Faces menu**.

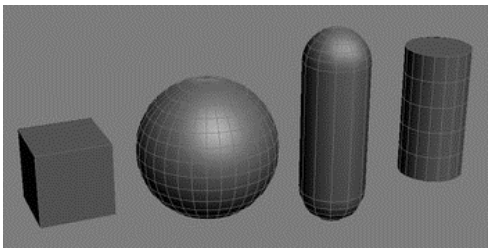
As you finish your shading process, you can export your object with **Export to CryEngine** tool.

Physical Proxy

Note: This document was directly copied from [CryEngine Physics Proxy](#) **document, only DCC** section has been modified according to **Blender** and **BCRY Exporter**.

The physics proxy is the geometry that is used for collision detection. It can be part of the visible geometry or linked to it as a separate node. Usually, the physics proxy geometry is a simplified version of the render geometry, but it is also possible to use the render geometry directly for physics. However, for performance reasons the collision geometry should be kept as simple as possible since checking for intersections on complex geometry is very expensive, especially if it happens often. A physics proxy is set up in the DCC tool exclusively. The only setup needed in Sandbox is assigning the surface type.

Using Primitives



CRYENGINE will automatically try to replace physics proxies with parametric primitives by comparing the proxy geometry against the supported primitive shapes and approximating the potential primitive within some calculative limits. The shape is generated around the general bounding box of the geometry and uses the original pivot point to attempt to rotate the primitive to match the geometry as good as possible.

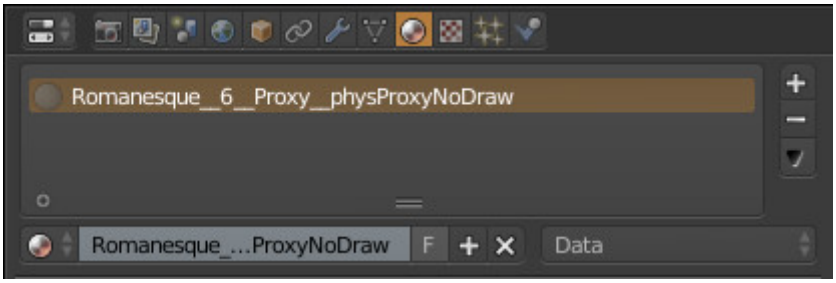
Primitives are a lot more efficient performance-wise than custom collision geometry (one single face of custom geometry is as expensive as a single primitive), so it is important to use them as much as possible.

It is possible to explicitly specify a collision primitive that should be used. In this case the engine will replace the physics proxy by the specified primitive. With

BCRY Exporter the primitive can be specified in the **User Defined Property** dialog of a node.

You may use **Add Physics Proxy** tool to add primitive proxy to your mesh.

Note: After you add proxy, you need to set material name convention (parent name and number) correctly for primitive.



BCRY Exporter Physic Setup

BCRY Exporter deducts proxy type from its material name suffix. You can use one of four proxy types per material. If you use a separate mesh object for proxy, there are three rules to export physical proxies via **BCRY Exporter.**

1. Proxy meshes need to be inside export node (cgf or anm).
2. Proxy meshes need to be same center (pivot point) with render mesh.
3. Your proxy mesh material names need to end with **__physProxyNoDraw**.

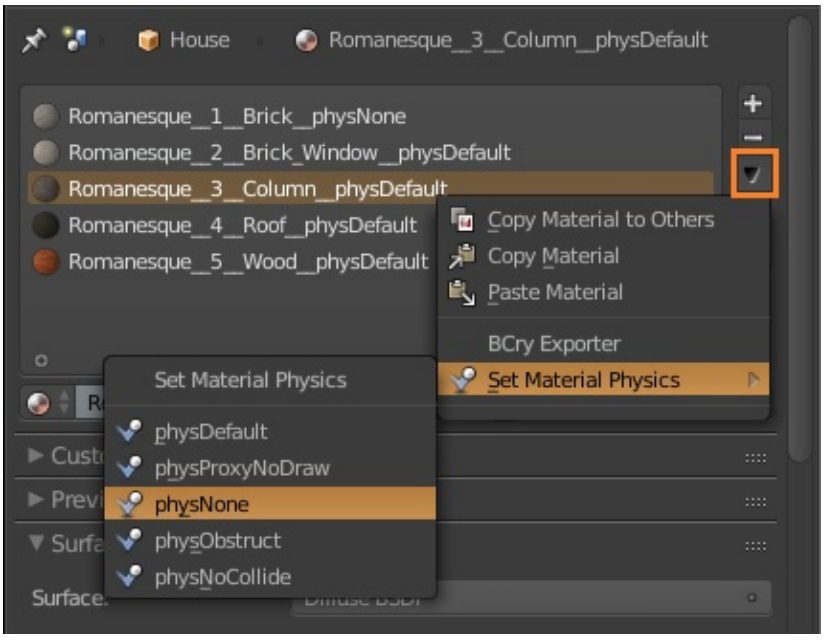
Current Allowed Proxy Settings

The following physics proxy settings are available.

Value	Entity/Player Interaction	Details
Default (None)	Entities are blocked	The render geometry is used as physics proxy. This is expensive for complex objects, so use this only for simple objects like cubes or if

		you really need to fully physicalize an object.
Physical Proxy (NoDraw)	Entities are blocked	Mesh is used exclusively for collision detection and is not rendered.
No Collide	Entities can pass through object	Special purpose proxy which is used by the engine to detect player interaction (e.g. for vegetation touch bending).
No Collide	Entities can pass through object	Special purpose proxy which is used by the engine to detect player interaction (e.g. for vegetation touch bending).
Obstruct	Entities can pass through object; AI View is blocked	Used for *Soft Cover* to block AI view (i.e. on dense foliage).

These properties are assigned in the materials in each 3D package:



The physics proxy is not allowed to have **open edges**. Open edges can confuse the physics engine and have a negative effect on performance. It is helpful to assign an extreme color (like a strong red) to the proxy to keep track of it.

Level of Detail Settings

Proxies are only created for LOD0. Every successive LOD will automatically take the proxy from LOD0. The same happens if different quality settings are used (e.g. Lowspec).

Complexity of Geometry

The physics proxies of environment objects (fences, crates, containers, trees, rocks, ladders, stairs, etc.) should be as simple as possible.

Avoiding geometric complexity for physics proxies is important to reduce redundant memory requirements and physics computations but also for making player movement smoother. The more complicated a proxy is, the more memory it will take and the more performance is lost when checking collisions against its polygons. This affects both single player and multiplayer, including the performance of a dedicated server. Besides the performance issues, a complex proxy with a lot of concavity increases chances that the player can get stuck or bounces undesirably against the proxy.

An ideal proxy is always a primitive (i.e. a box, a sphere, a capsule or a cylinder). CryENGINE recognizes primitives from meshes but the default tolerance is very low; in order to force the recognition, put the corresponding keyword (“box”, “sphere”, etc) into the node’s user defined properties. Note, however, that meshes with several surface types cannot be turned into primitives. Primitives should be considered as an option even for more complex objects. In most cases it is preferable to have a multipart object (i.e. “merge nodes” off) with primitive parts instead of a single part mesh object.

The proxy is used for blocking character movement as well as first pass tracing of projectiles (bullets and decals). If a hit was detected against the physics proxy, projectile impact and decal locations are refined using the render mesh. The render mesh should be fully encapsulated by the physics proxy, so that the player camera does not intersect the render geometry and first pass projectile culling does not miss the physical part of the object even though it hits the visual part of the object. There is also support for creating a special raytrace proxy that, if provided, will be used for projectiles. This would then allow the main proxy to not have to encapsulate the render mesh and thus the proxy could be even simpler.

Simple objects like for example crates and fences can usually be approximated with a simple box which has 6 sides (12 triangles). The top of stairs should usually be simple ramps, resulting in just 2 triangles. More organic or irregularly shaped objects like rocks and trees can still be approximated with a fairly simple hull by allowing slight and acceptable inaccuracies between the render mesh and the physics proxy.

Linking Objects to Simplify Export

The physics proxy can be part of the render object (in 3ds Max as an Element) or as a separate object, linked to the render object.

Setup in Sandbox

In Sandbox you need to assign a surface type to your physics proxy in the Material Editor, as well as to your render geometry. The surface type gives information about sound and the particle effects of your surface.

You also need to assign a NoDraw shader in the material editor. Make sure that no textures are assigned to your proxy sub material. The physics proxy will never be rendered in the Sandbox Editor without debug view. Even if you assign an illum shader it will stay invisible.

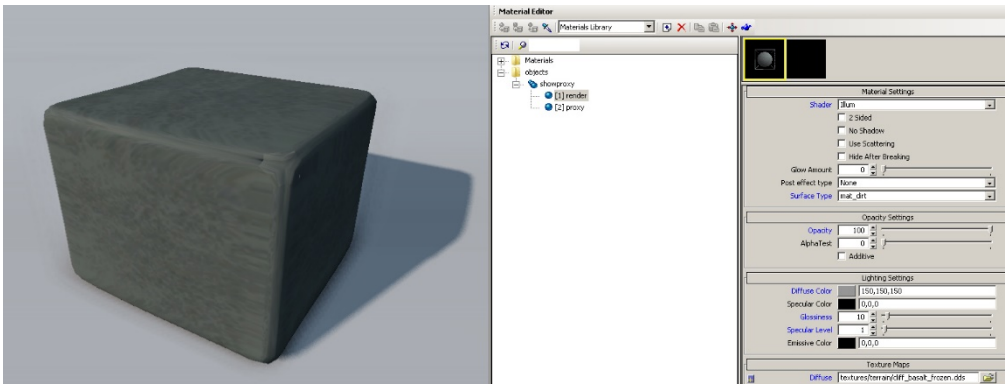
To reload the physics proxy you need to reload your object, delete it, and undo delete.

Debugging

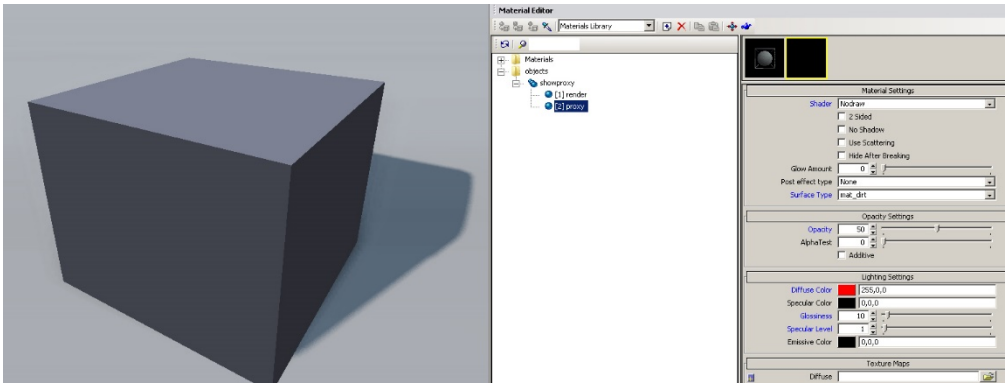
p_draw_helpers

Same as p_draw_helpers_num, but encoded in letters
Usage [Entity_Types]_[Helper_Types] - [t|s|r|R|l|i|g|a|y|e]_[g|c|b|l|t(#)]
Entity Types:
t - show terrain
s - show static entities
r - show sleeping rigid bodies
R - show active rigid bodies
l - show living entities
i - show independent entities
g - show triggers
a - show areas
y - show rays in RayWorldIntersection
e - show explosion occlusion maps
Helper Types
g - show geometry
c - show contact points
b - show bounding boxes
l - show tetrahedra lattices for breakable objects
j - show structural joints (will force translucency on the main geometry)
t(#) - show bounding volume trees up to the level #
f(#) - only show geometries with this bit flag set (multiple f's stack)
Example: p_draw_helpers larRis_g - show geometry for static, sleeping, active, independent entities and areas

Example picture without physics proxy visualization:



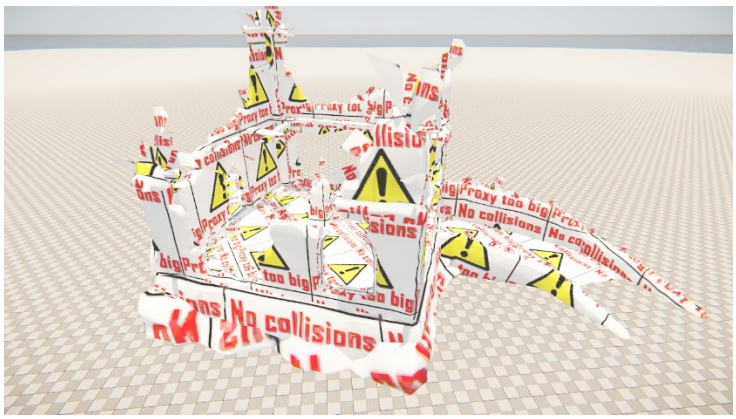
The same object with physics proxy visualization:



e_PhysProxyTriLimit

Maximum allowed triangle count for phys proxies

If you notice your assets are wrapped in a message stating “No Collisions, Proxy Too Big!” then the physics proxy for that asset is over the triangle count specified in **e_PhysProxyTriLimit**. It will look something like this:



Level of Details

Level of Detail objects are used to improve the rendering performance of objects located far away from the camera picture plane. With BCry Exporter, you can create LODs for your object with one tool.

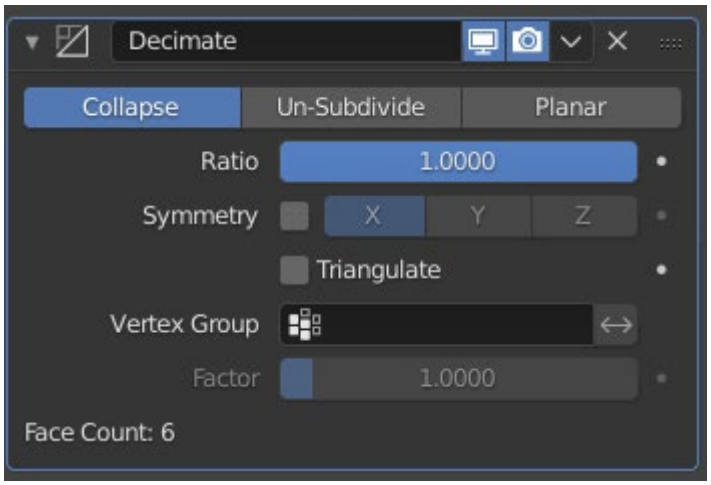
Generate LODs:

1. Select object.
2. Run **Generate LODs** tool from **Mesh Utilities** menu.
3. Set your LODs count and decimate ratio from Tool Shelf panel.

BCRY Exporter is going to create LODs, and it is going to add decimate modifier on each of them. Also, you can set distance of LODs object in scene with View Offset parameter.

Note: Finally, all LODs need to share same world location with your main object. You may set View Offset to zero for that or you may manually align them in a later time. (**Alt + G, Alt + R, Alt + S** to clear transformation)

You can set individual decimate ratio for per LOD from Modifier Panel in Properties Editor. You can tweak the Decimate modifier as you like to achieve best look at certain level of detail with limited vertices.



Note: If you use Generate LODs tool before add object to an export node, you need to add your LODs objects to export node while adding object to node.

Before exporting your object, you must apply decimate modifiers for per LOD object, or you must use check **Apply Modifiers** checkbox when do **Export to CryEngine** panel.

Finally, you can export objects with **Export to CryEngine** tool.

Create LODs Manually

If you want to create LODs manually or want to use your own specific LOD objects, there is one rule to know: The LODs names must be suffixed **_LODx**, which x should be changed with a number.

Example:

- *Cup*
- *Cup_LOD1*
- *Cup_LOD2*

BCRY finds your LODs object if there are.

Debugging LODs:

There are various console commands which help debugging LOD's in Sandbox of in game mode.

e_Lods *Load and use LOD models for static geometry*

e_LodMin *Min LOD for objects*

e_LodMax *Max LOD for objects*

e_DebugDraw

Draw helpers with information for each object (same number negative hides the text)

- *1: Name of the used cgf, polycount, used LOD*
- *2: Color coded polygon count*
- *3: Show color coded LODs count, flashing color indicates no Lod*

- *4: Display object texture memory usage*
- *5: Display color coded number of render materials*
- *6: Display ambient color*
- *7: Display tri count, number of render materials, texture memory*
- *8: RenderWorld statistics (with view cones)*
- *9: RenderWorld statistics (with view cones without lights)*
- *10: Render geometry with simple lines and triangles*
- *11: Render occlusion geometry additionally*
- *12: Render occlusion geometry without render geometry*
- *13: Display occlusion amount (used during AO computations). Warning: can take a long time to calculate, depending on level size!*
- *15: Display helpers*
- *16: Display debug gun*
- *17: Streaming info (buffer sizes)*
- *18: Streaming info (required streaming speed)*
- *19: Physics proxy triangle count*
- *20: Display object instant texture memory usage*
- *21: Display animated object distance to camera*
- *22: Display object's current LOD vertex count*

User Defined Properties

User Defined Properties (**UDP**s) are used to add special information on your mesh.

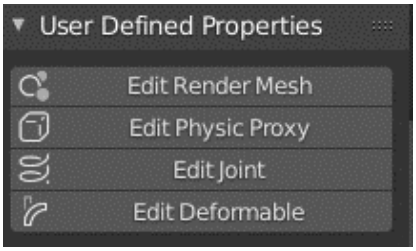
From CryEngine Page:

A user-defined property (UDP) is a property that is defined when you construct a message flow by using the Message Flow editor.

The advantage of UDP's is that their values can be changed by operational staff at deployment and run time, without having to change the code at the message node level.

In CryENGINE, UDP's are used for affecting and changing the Physics Properties for static objects and especially breakable assets!

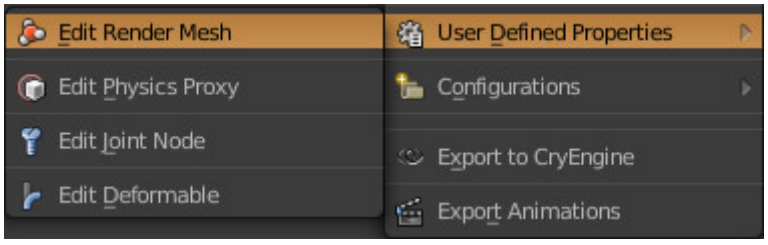
UDP Setup in Blender



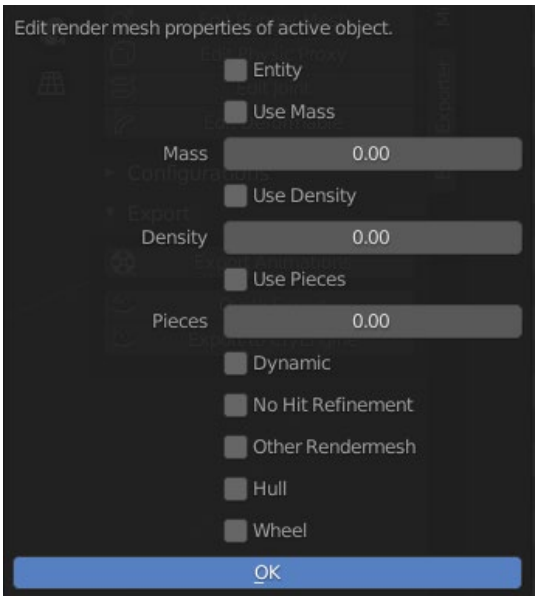
BCRY has interactive and fully described **UDP system**. You can interactively add CryEngine UDP settings to your mesh. To do that:

1. Select your mesh in your scene.
2. Run one of four UDP tools from **User Defined Properties** menu.
3. Enable tick that you want to use property, and set it value from under (If it has variable setting)
4. Press **OK** from panel.
5. Export your mesh as normally, with **Export to CryEngine** tool.

Render Mesh Properties



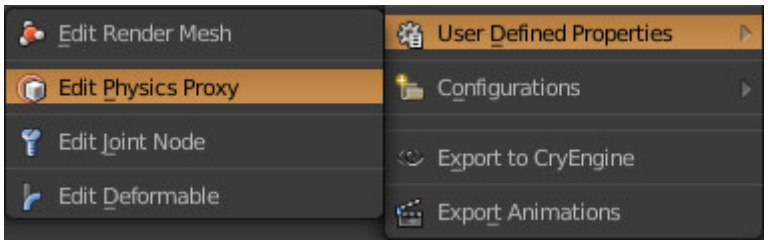
Render mesh UDPs are used generally render and mass settings.



Descriptions of UDP

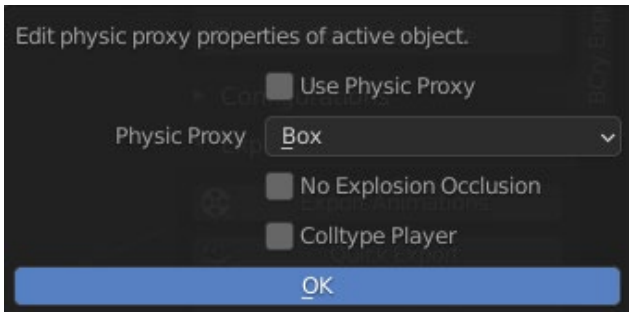
BCRY includes all UDP descriptions copied from [CryEngine manual](). **To show description you may hold cursor over property you want to know.**

Physics Properties for Object properties/UDP



CGF/CGA/CHR

Physic proxy UDPs are used for mesh physic settings.

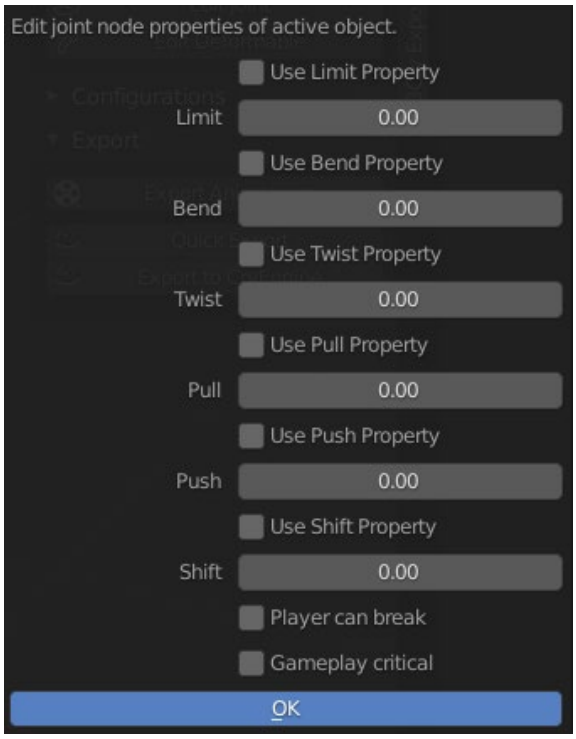


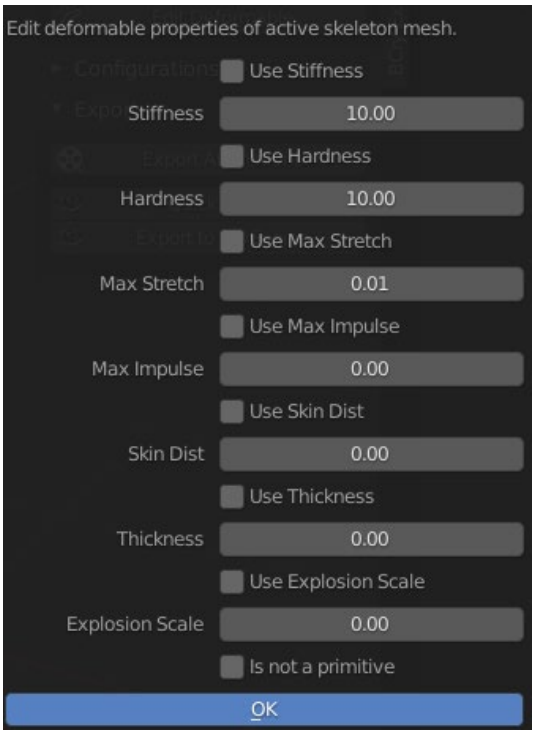
Jointed (pre-broken) Breakable

Joint node UDPs are used for breakable settings.

Deformable

Deformable UDP are used to specify deformable mesh values.





Note: You can also learn more about UDPs from CryEngine Manual [UDP Settings](#).

Blend Layer

NOTE: *This chapter has NOT been tested yet.*

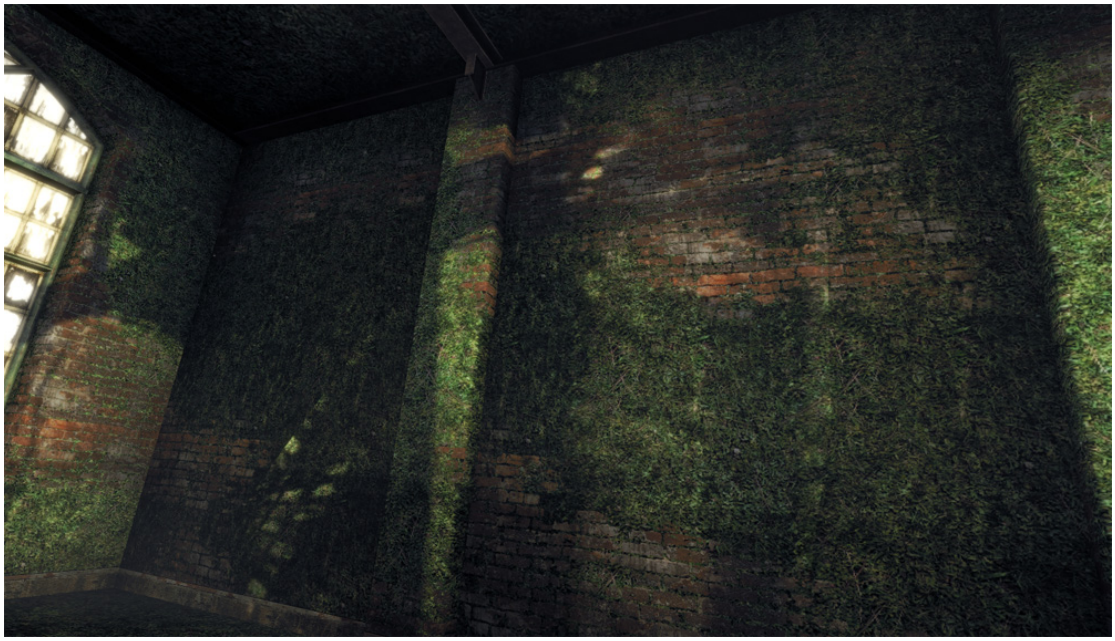
The blend layer feature allows mixing a second set of textures with the base set to get more variation on tiled surfaces.

The blending is primarily based on the vertex alpha channel and a generic blend map. The vertex alpha determines where the blend layer should be visible on the surface of the mesh.

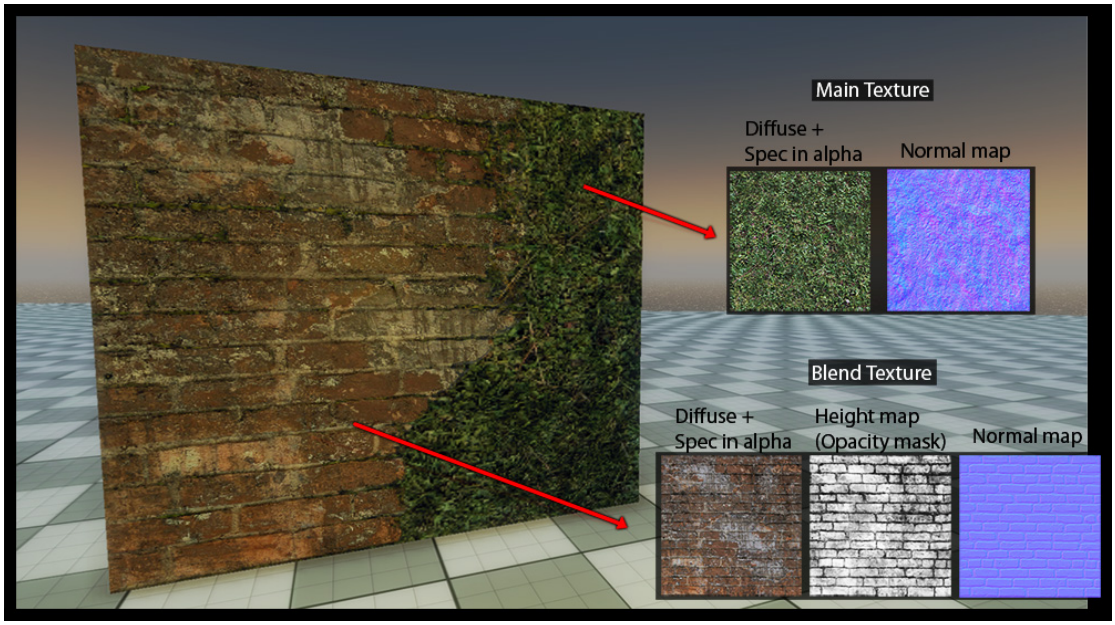
Independent from the geometry tessellation, you can achieve smooth as well as very sharp transitions between the layers. A low number of vertices is sufficient for good looking results.

A very useful feature of the blend layer is that a heightmap can be used as blend map. This makes the blending follow the structure of the base texture.

Example of the blend layer in an indoor environment.



Example of the blend layer on a brick wall with green moss.



In game asset with Illum shader and blend layer feature enabled.

The blend layer feature does not combine two generic materials, it rather blends specific textures of a single material. This means that the basic material settings apply to both layers.

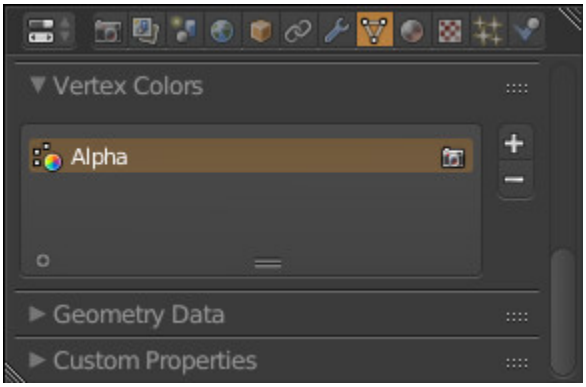
The blend layer can have an individual diffuse map, specular mask (in the alpha channel of the diffuse map) and a separate normal map.

Usually, this gives enough control to get great results with good performance. Shader values (spec intensity, glossiness) are applied to both layers (the base and the blend layer).

Blender Asset Setup

As default, **Blender** doesn't have alpha channel for vertex colors which used blend layers at **CryEngine**.

We can add a vertex color channel named **Alpha** for that. BCRY converts RGB color information to alpha color, whenever active layer named as **Alpha**.



Note: BCRY only export active channel, so if there are one more color layer, you have to activate alpha channel.

Sandbox Material Setup

The blending can be enabled with the **Blendlayer** checkbox. Once active, several new shader parameters will show up:

Shader Param	Description
Blend Factor	Changes the overall blend factor. Use this to fade in and out the blend layer.

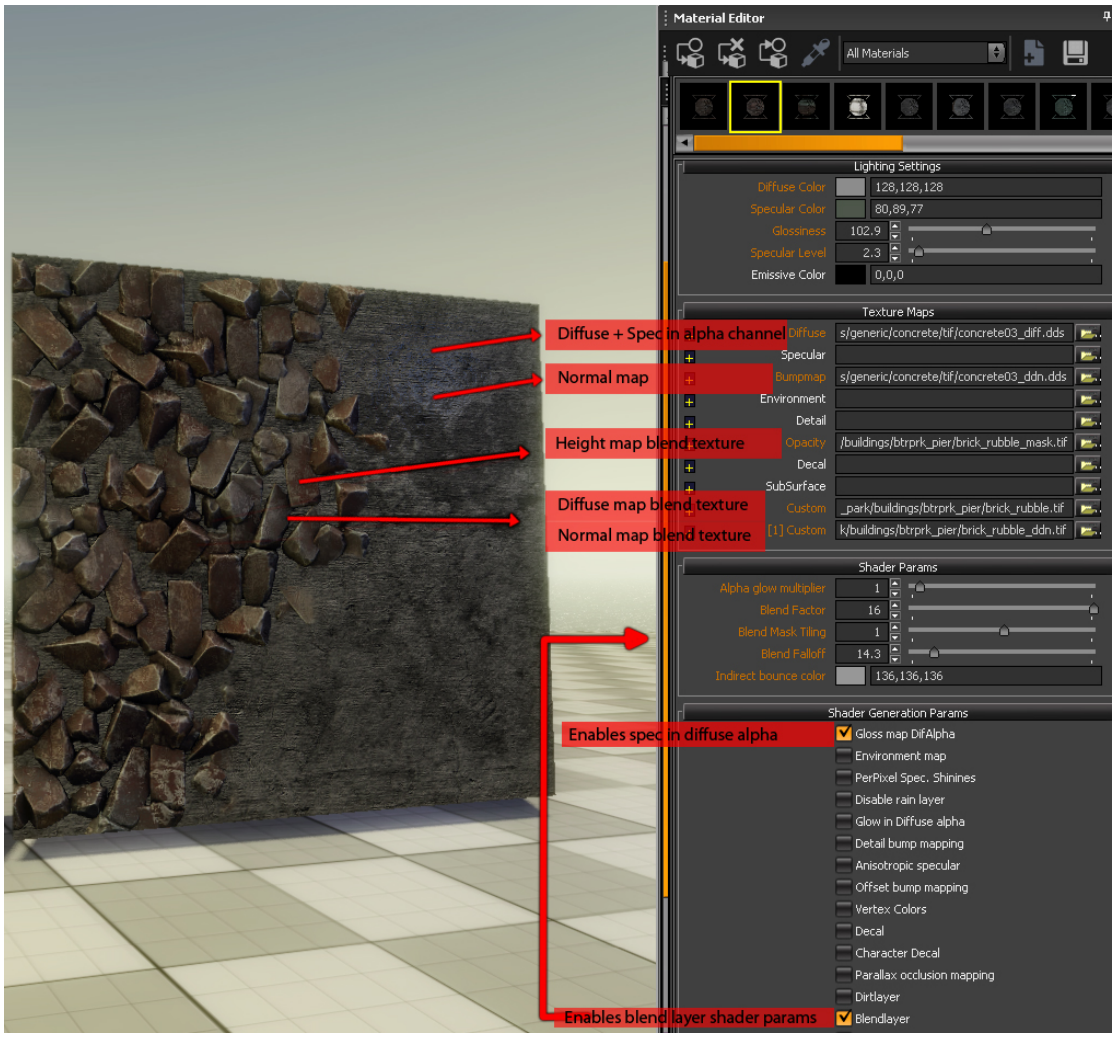
Blend Falloff	Specifies how smooth the transition between the layers should be. Use a high value for a sharp transition.
Blend Layer 2 Tiling	Change tiling of second blend layer.
Blend Mask Tiling	**DEPRECATED –** Specifies how often the blend layer textures are repeated (tiled). If you use heightmap that aligns with your base normal map, this parameter should usually be set to exactly 1.0, otherwise the blend layer textures are rotated to make them appear less repetitive.
Blend Mask Rotation	**DEPRECATED –** Change rotation of blend mask.

Once Blendlayer is enabled, the available texture slots will update:

Texture Map	Description
Second Diffuse Map	Diffuse map of blend layer (with specular mask in alpha). *In CRYENGINE 3.4 or earlier, this is the "Custom" slot.*
Second Bump Map	Normal map of blend layer. *In CRYENGINE 3.4 or earlier, this is the "[1] Custom" slot.*
Second Height Map	Displacement map of the blend layer.
Blending Map	The blend map (a generic grayscale texture, e.g. a noise texture, or a heightmap). *In CRYENGINE 3.4 or earlier, this is the "Opacity" slot.*

The blend map can be interpreted as if it would specify some height. This way you can have moss growing between the bricks.

If the blend factor is increased, by changing the vertex alpha, blend map or strength slider, the moss will slowly begin to cover the surface of the bricks.



Example of Blend Layer setup in the Sandbox Material Editor.

The falloff can be useful to simulate different material types. With a high falloff value you can get a hard blend to simulate for example rust on a metal surface. A low value gives a much smoother transition, as you would have it for snow on the ground.

The blend factor should normally be set to 16. If you want to tweak the blending, you should do it in 3ds Max via the Vertex Alpha. If you lower the blend factor you will lose range and control. A blend factor below 16 never allows you to reach the maximum blending.

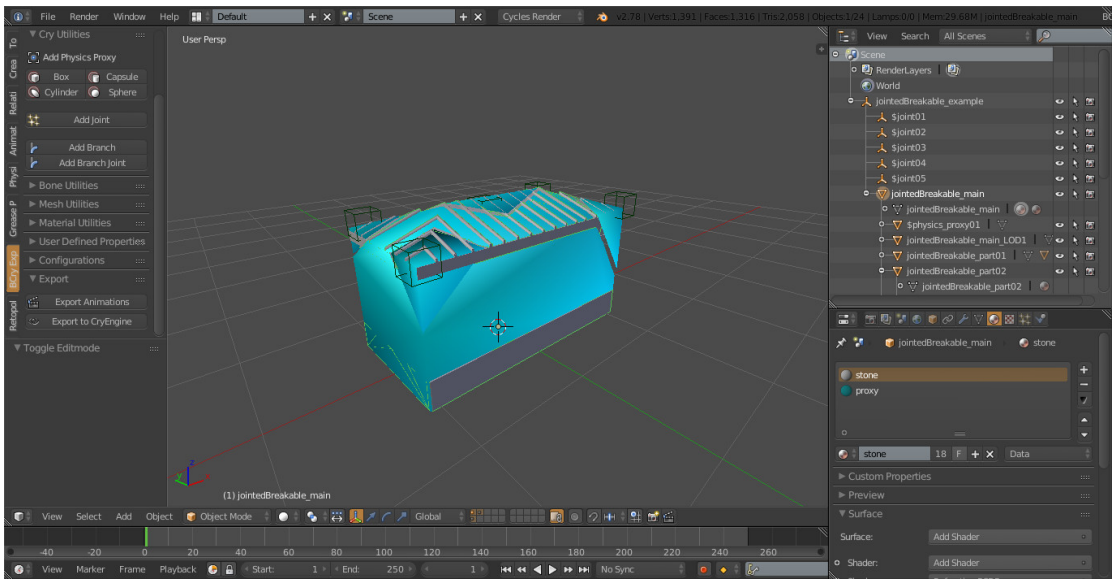
Note: This document copied from **CRYENGINE** official Blend Layer page. Just DCC section has been changed according to **Blender**.

Breakable Objects

Note: This document was directly copied from [CryEngine 3DS Max Breakable Setup]() **document, just DCC** section has been modified according to **BCRY Exporter**.

Overview

A jointed breakable object is a pre-broken object that uses joints to hold the individual pieces together. As soon as a certain amount of force (defined in the joints UDP Settings) is applied to the joint it will break the connection between the pieces and spawn a particle effect defined in the surface type of the proxy. A jointed breakable is a complex object that has a big impact on performance regarding drawcalls, memory impact and physics calculations. Whenever building one, be smart and use as few pieces you can get away with. Besides just breaking parts off your model, you can also make pieces disappear or spawn one or several new pieces when the joint gets broken.



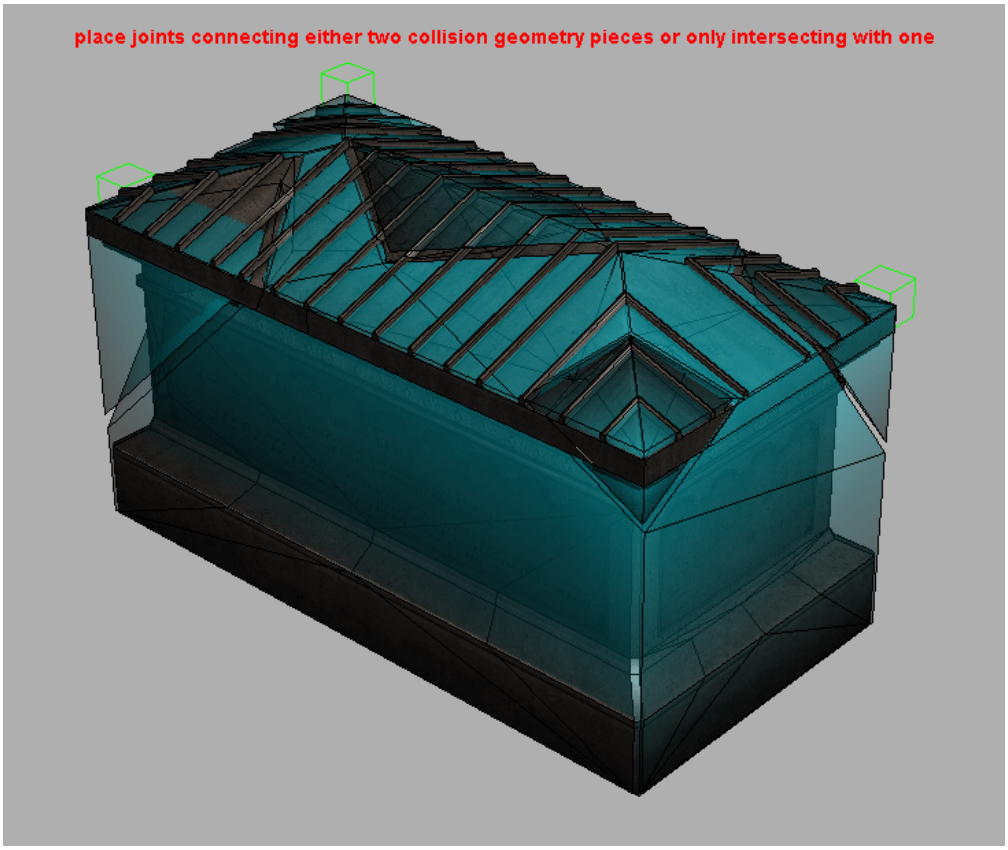
Object setup in Blender for a simple jointed breakable

- Once you cut your object into several pieces, you need to identify which one will be the static remaining piece when the joints gets broken. This is your base piece and becomes the parent render mesh for your hierarchy setup which is a child of your export node. Give it a unique nodename and enter "**mass=0**" and "**entity**" in its UDP's. Create LOD's and collision geometry for it. Center the pivot to the export dummy helper and make sure that it does not have any transformations applied.

- Link all other pieces as children to this static piece and give them a mass in their UDP's (depending on their weight in kg, i.e. "**mass=15**" meaning the object has a mass of 15 kilograms). Give the pieces unique nodenames and create LOD's and collision geometry for them. Make sure the LOD's line up perfectly with all other LOD's on each level so that no gaps are visible. Only create LOD´s for pieces that have more then 100 triangles. Center the pivots of all objects to your export node and make sure they do not have

any transformations applied. For information on how to add UDPs, please refer to the UDP Settings section.

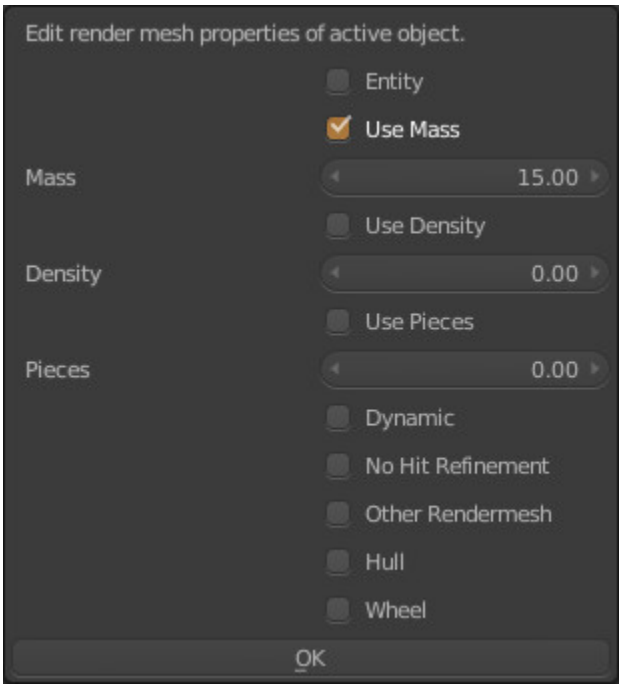
- Create one joint for each piece and call it **_jointxx** (xx=00,01,02,⋯). Set a limit to your joints (use 500 for testing, for more explanation in detail refer to joint values). Make sure the joints intersect with only one piece of collision geometry (connecting this piece to the world, making the joint red in debug game mode) or two different pieces of collision geometry (interconnecting those two pieces, making the joint light grey in debug game mode). The joints must not be scaled and should not be rotated.



- Export your object by adding the dummy helper to the export list with “Export File per Node” enabled and “Merge All Nodes” disabled
- Place the object as a brush in Sandbox and shoot it. It should now break.

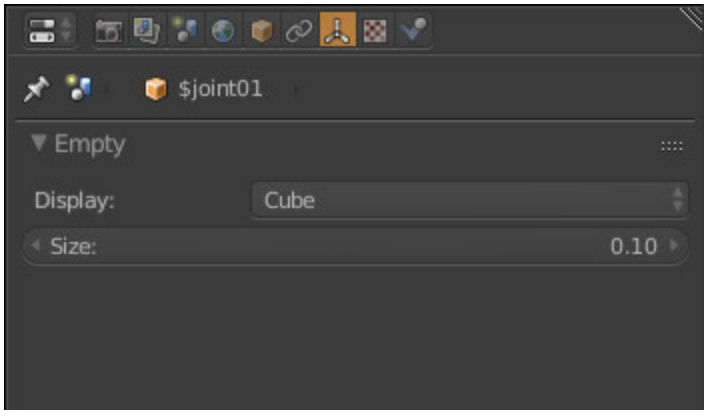
Render Mesh UDP

You can add or set mass property to your empty/dummy object from **Edit Render Mesh** UDP Panel.



Change Empty/Dummy Object Size

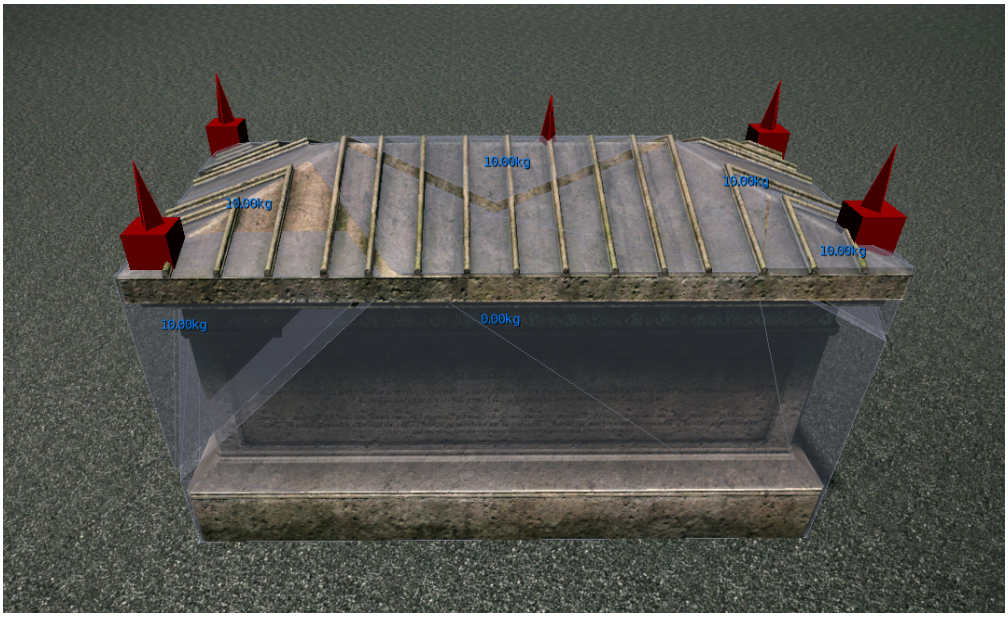
You can change your empty object size from properties panel. You mustn't scale it to change size.



Debugging

Enable **p_draw_helpers 1** and **p_debug_joints 1** in the console.

Joints connected to the world will be displayed in red. Joints interconnecting two pieces will be displayed in light grey. The “spikes” on the joint objects show the orientation. For consistent results, they should point into the z-Axis (up).

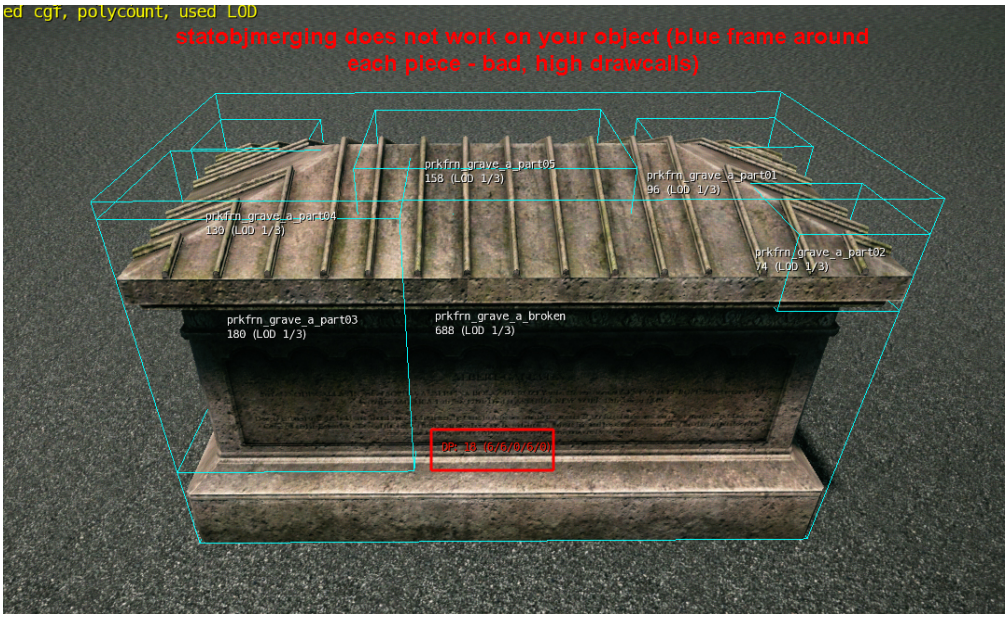


When a force is applied to a joint, the direction is shown (in this case shift and bend), together with the strength and the limit of the particular joint. Green text represents an impact with an impulse much lower then the joint limit. Yellow represents values closer to the joint limit. Red represents values which were higher than the limit – in this case the joint is broken.

The mass of your pieces will be displayed in blue centered on the piece.

To check for drawcalls and statobjmerging, enable **r_stats 6** and **e_debug_draw 1**.

If statobjmerging does work, your entire object will be displayed in a yellow or orange frame with a low number of drawcalls.



If statobjmerging doesn’t work, the individual pieces will be displayed in a blue frame with a high number of drawcalls. Statobjmerging must always work on your object.

Troubleshooting

Q: The joints do not show up.

A: Check naming convention ($joint01, $joint02,⋯). Check if they have transformations on them. If so, recreate them and reposition them.

Q: My object does not break when shot.

A: Check that the weapon you are using is strong enough by enabling p_draw_helpers 1 and p_debug_joints 1. If not, lower the joint limit in the UDP Settings.

Q: My object breaks but random pieces disappear.

A: One of your joints is intersecting with more then two pieces of collision geometry. One of your pieces is missing a joint. Check that all pieces have a mass value and all joints have a limit value (or any other from the Joint Properties section). Check that your joins are big enough and intersecting with at least one piece of collision geometry.

Q: My objects jitter when shot.

A: There is not enough space in between your collision geometry pieces.

Q: After the pieces have been shot, they disappear.

A: This is the default behavior to save on physics calculations. If you need your pieces to not disappear, add "entity" to their UDP Settings.

Q: Statobjmerging does not work on my jointed breakable.

A: Check that all surfacetypes in your material are set to non breakable ones (especially check for glass and wood surfacetypes). If you need breakable glass, separate the glass into its own cgf and create a new material for it. Check for the polycount on your base object – If it is higher then 3000, separate the breakable pieces into their own cgf (refer to Restrictions).

Q: One of the pieces remains floating after I shoot the model into its separate parts.

A: The model needs a grounded piece. Place a joint on a part of the model where it will be touching the terrain to 'ground' the model.

Restrictions

- A joint can only connect two pieces with each other (joint displayed in light grey) or one piece to the world (Joint displayed in red).
- Statobjmerging will only work until a certain polygon threshold is reached. Above that threshold, it becomes too expensive to use this feature. This happens often when using huge buildings with a very polycount heavy base piece and only small chunks that break. To circumvent this issue, you can create a new .cgf that only contains the breakable pieces without the base piece and place it manually in the editor.
- Each piece must be its own node with its own lods and collision geometry.
- Collision geometry of pieces must not intersect with each other. There must be a gap in between different collision geometry pieces.
- Each joint must have a limit UDP set, each piece must have a mass or density UDP set.
- Never scale a joint! If you need a new size, create a new joint. the scaling value will be passed on to the engine and causes wrong results in the simulation.
- Each Jointed breakable object only supports the material it has been exported with. If you assign a new material to it in Sandbox, it will revert to the default material as soon as a joint is broken.
- Jointed breakable objects support up to 64 individual pieces, however you should try to keep the number much lower in order for statobjmerging to work.
- If the material of your breakable object contains a submatid with a breakable surfacetype (mat_glass_breakable_thin_small, mat_glass_breakable_thin_large, mat_glass_breakable_safetyglass_small, mat_glass_breakable_safetyglass_large,

mat_glass_unbreakable_roosevelt, mat_glass_unbreakable_spiderweb, mat_wood_breakable_medium, mat_wood_breakable_thin), statobjmerging will not work.

- Do not use physics geometry merged into the rendermesh and physics geometry separated into its own $physics_proxy node. Always separate physics to $physics_proxy when working with jointed breakables.
- Only add UDP's to your rendermesh – LOD's and collision geometry do not need them and will inherit their parents values.
- Use only one $physics_proxy node per piece. If you need more, merge them into one node (you cannot use primitives in this case).

Destroyable Objects

Note: This document was directly copied from CryEngine Maya Destroyable Setup **document, just DCC** section has been modified according to **BCRY Exporter**.

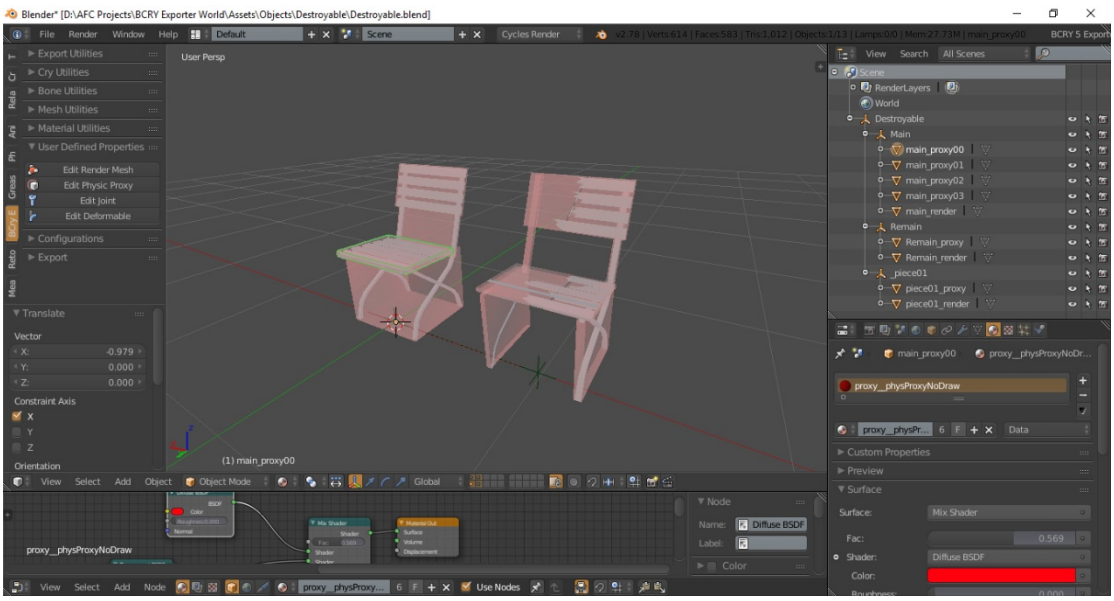
Destroyable objects are structures that contain the original object and pre-created pieces that appear when the original object is destroyed. Its setup is similar to a jointed breakable but it doesn't use any bones. Instead it uses the model in its unbroken state and the pieces in the broken state embedded in one cgf. It can be used to completely destroy specific types of objects in the environment. It shatters into the pre-created pieces by taking more damage then the specified "health" property of the object. It is placed using the DestroyableObject Entity in the engine.

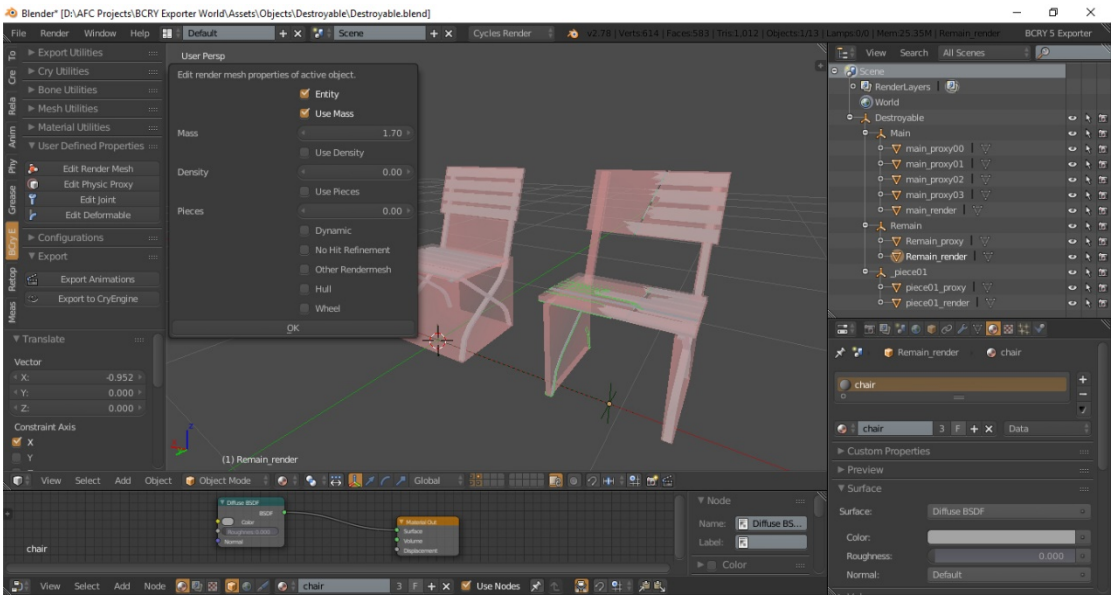
A DestroyableObject can be in two states, an "Alive" or a "Dead" state. In the "Alive" state, it acts exactly like a normal physical entity. It can be set up to be a rigid body or a static physical entity. After taking more damage then the specified "health", it will go into the Dead state. When going into the Dead state, it can optionally generate a physical explosion and apply area damage on the surrounding entities, spawn a particle effect (for example, an explosion), and replace the original geometry of the entity with either destroyed geometry and/or pre-broken pieces of the original geometry. If the object breaks due to a hit (bullet), that hit impulse is applied to the pieces in addition to any explosion (outward) impulse.

Destroyable Objects Setup

- Create your object with physics geometry, LOD's and a pickup helper if needed.
- Duplicate your render mesh. Cut it into the amount of pieces that you want (Be reasonable – This can become expensive quite fast. 8 Pieces should be enough for most objects, the smaller the object the less pieces you need).

- Create LOD’s and collision for your pieces.
- Call your unbroken piece **main** and call your broken pieces **remain_xx** (xx=01, 02, 03, etc.)
- Enter mass a UDP to the main piece and enter mass and entity UDP’s to your broken pieces. The amount of mass of your broken pieces should ideally add up to the mass of your main, unbroken piece.
- Center all pivots to your export helper.
- Export with merge all nodes off and Export file per node on.
- Place your destroyable object as DestroyableObjects Entity in the editor, set up the correct values and shoot it. It should now break.





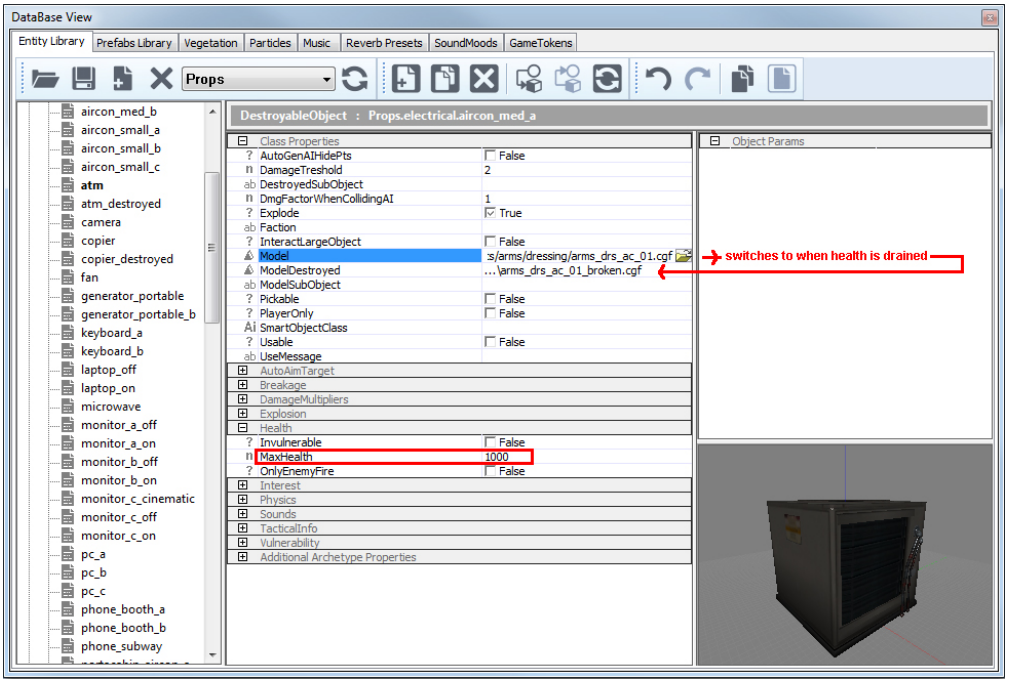
[Download Destroyable.blend example…]

Using plane cgfs without embedded pieces

Instead of embedding the pieces into one cgf, you can create two different cgf's. One in its undestroyed state, one in its destroyed state.

Place the object as a destroyable Object Entity and specify the cgf's and a health value. It will switch when the object has taken more damage then specified in the health value.

When not specifying anything in the ModelDestroyed tab and not using a cgf that has pieces embedded, the object will disappear when the health is drained.



After you've created your Destroyable object cgf(s), add a particle effect, health values and explosion effects. I've used **Props.electrical.aircon_med_a** in the example screenshot.

General Rules

A CGF containing broken object pieces as sub-models must be created for the object. Depending on how you want to set it up, the CGF can also contain the main unbroken model as a separate sub-model. Each sub-model should also have a physics proxy geometry.

The sub-model's name and text properties determine its behavior, as follows:

- The object must consist of the main original piece named Main that will act as an alive geometry. It is the (only) pre-destruction sub-model.
- It should also consist of the destroyed geometry named Remain that will replace the original geometry and acts as the dead geometry. It is the permanent post-destruction sub-model, which replaces Main. Otherwise, it is a destruction piece.
- It should consist of different pieces that will be spawned as particles or entities.
- Every piece needs to have a physical proxy.

- The orientation in world coordinates should be 0/0/0 (see Debug section).
- Turn OFF “Merge All Nodes” on export.
- Objects should be placed as Destroyable entity.
- Spawn Location: By default, the pieces spawn in the position in which they are placed in the CGF, relative to the original model. The following text properties alter this behavior:

- bone = name: (Only for breakable skinned characters) – This is the name of the bone that this piece is attached to. The sub-model’s origin will be placed at the bone origin, and the sub-model’s X-axis will align with the bone direction.
- generic = count: This causes the piece to be spawned multiple times in random locations, throughout the original model. The count specifies how many times it is spawned. There can be multiple generic pieces.
- sizevar = var: For generic pieces, this randomizes the size of each piece, by a scale of 1-var to 1+var.
- rotaxes axes: For generic pieces, this generates random rotation. Set this to the axis letter(s); for example, =z or xyz, to cause the piece to rotate randomly about the selected local axis/axes. If not set, the pieces will spawn in their authored rotation.
- entity: If this is set, the piece is spawned as a persistent entity. Otherwise, it is spawned as a particle.
- density (density or =mass) = mass: This overrides either the density or the mass of the piece. Otherwise, it uses the same density as the whole object.
- Surface Material Setting: It’s important to make sure that each piece has an appropriate surface type (for example, wood or ice) because this is used to spawn secondary breakage effects.

A plain CGF, with only one sub-model, can also be destroyed. Its model will disappear, but if the model's surface material specifies a destroy effect, it will be spawned. For breakable skinned characters, the CGF should not contain Main or Remain pieces; only "bone" and "generic" pieces.

Creating Pickable Destroyable Objects

The setup for making your destroyable object pickable is fairly simple. Refer to the Pickable Objects documentation for more information.

Secondary Particle Effects

When the object breaks, by default, secondary particle effects are automatically spawned, based on each piece's surface material type. The effects are read from the material script, and there are two kinds of effects that can be specified:

- **breakage.particle_effect:** This specifies an effect to spawn on each broken piece of this material type, around its edges.
- **destroy.particle_effect:** This specifies an effect to spawn throughout the unbroken object volume, at the point where it is destroyed. This effect is useful for destroying simple objects that don't have a multi-part CGF. The object disappears and is replaced by the destroy.particle_effect. However, the destroy effect can also be used along with the breakage effects.

Troubleshooting

Q: The object doesn't break.

A: Check the health value in your Destroyable object entity. Check your object for correct use of naming conventions (main, remain_xx). Check that every piece has its own collision geometry.

Q: The object spawns no pieces or they get spawned in wrong positions.

A: Set all your pivots on the export helper and reset transformations.

Q: The pieces jitter when broken.

A: The physics geometry pieces intersect with each other. Make sure that there are gaps between the collision.

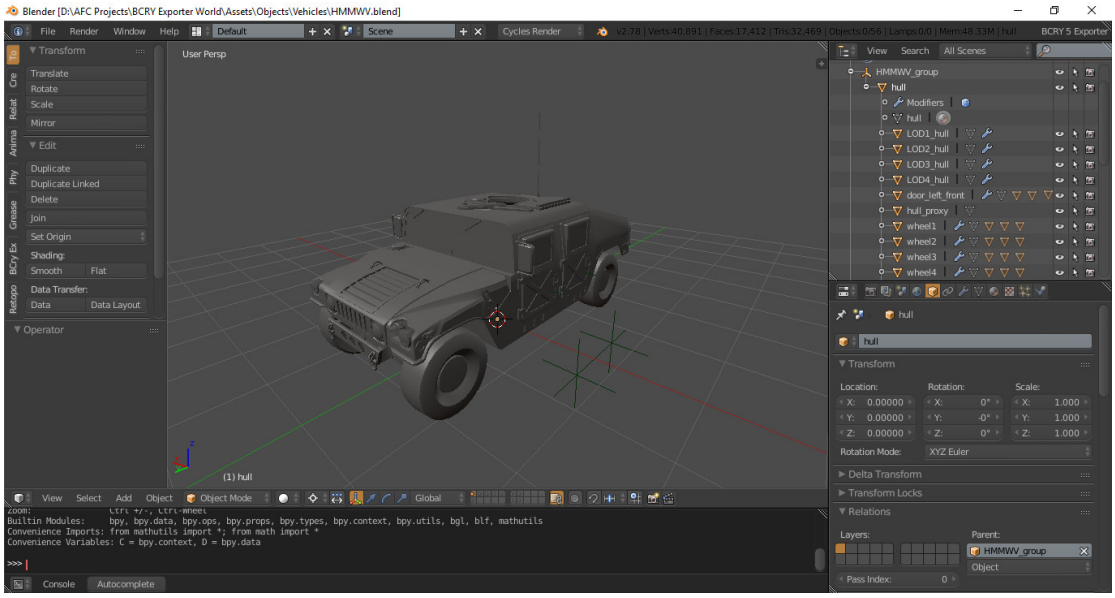
Properties

Please see the Physics Entities document for information on placing a DestroyableObject and their property values.

Vehicle Setup

CAUTION: This chapter has NOT been tested yet. No guarantee for now.

Vehicles are complex objects that need careful setup in the 3d package. Basically, the setup should be applicable to any package. Polycounts need to be carefully calculated, as extended functionality increase object (drawcall) and polycount drastically.

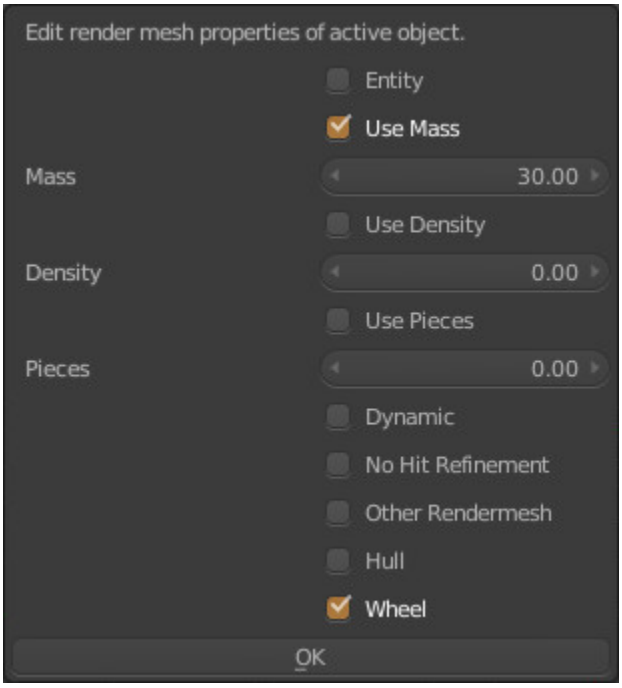


Every moving part needs to be a separate object within the 3d package. The pivot needs to be set correctly within the 3d package as the engine will use the object's individual pivot to make transformations.

Parented objects have the advantage to control objects in scene.

Hull object main part of the vehicle.

Naming is needed for scripts to identify the objects and interact with them. The examples below reflect the asset names used in Crysis – they can be different for other games.

- 

Place the wheel objects in the correct position.

The wheel proxy must be simple enough to be recognized as cylinder by physics. Support for an additional detailed proxy (if the wheel's shape demands it) will follow.

You can follow general material setup rules. There is no any specific rule for Vehicles.

Just be sure you set your proxies material as *__physProxyNoDraw*.

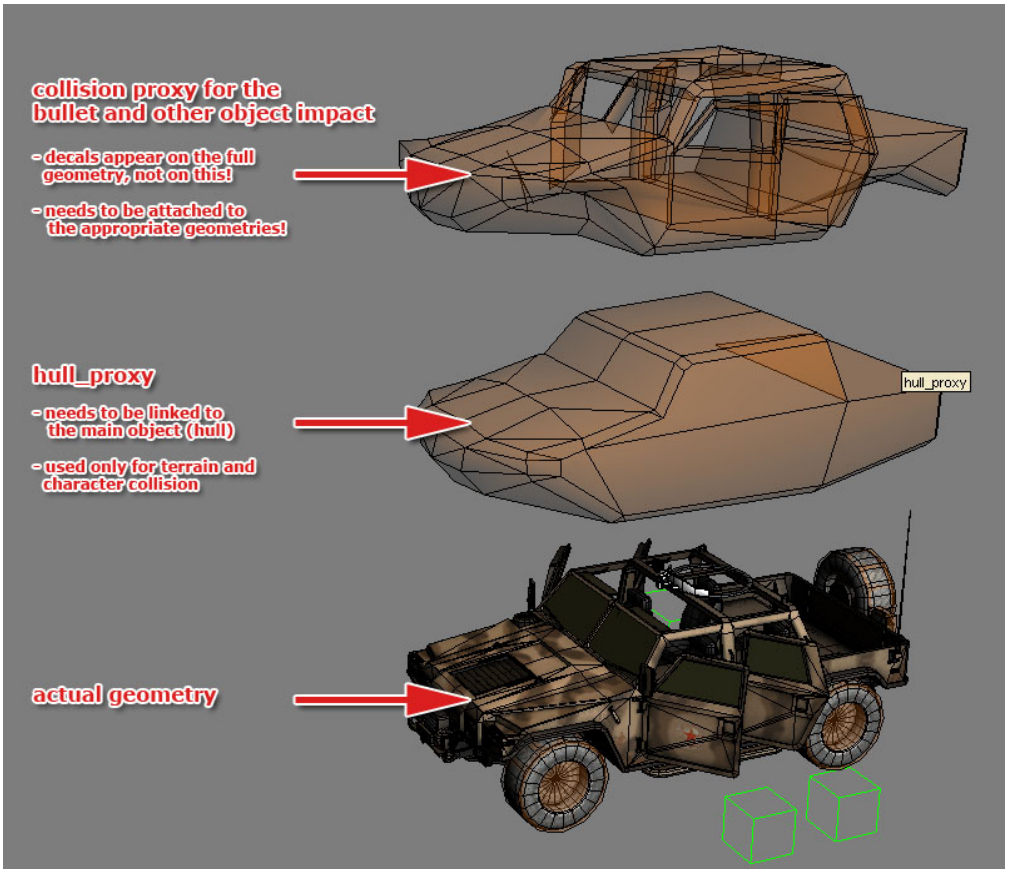
Physics Setup

Vehicles are complex objects and simplifying collision calculation is very important for performance. Therefore it is advisable to create two physics proxies. one is for collision with the terrain and characters, one for collision with bullets, grenades…

- Vehicles must have a very simple physics proxy which is called "**hull_proxy**". This is a separate, collision object for character and terrain collision. It should be slightly larger than the enclosed detailed proxies. Polycount for this proxy must be fairly low. For land-based vehicles like cars, tanks and hovercrafts, the proxy should not have parts sticking out in

places which can frequently bounce into and slide across walls, like the front and side parts of a truck cabin.

- For collision detection of bullets, grenades, the hull, etc – object should contain a more detailed embedded physics proxy. It should resemble the shape of the vehicle more closely, small elements could be left out. It should certainly include cutouts, like holes for the doors, windows, and other open areas.



- Attached parts like doors and bumpers need an own simplified proxy in case the embedded proxy is too detailed and calculation heavy for terrain/water collision. In this case, add the suffix "**_proxy**" to the object name.

- Proxy material must have **__physProxyNoDraw** physic options.

- LODs do not have to contain proxies, as it will be used from the LOD0. In versions with lower performance settings, where LOD1 is used as the base mesh, the proxy from the High Spec LOD0 is taken into account.
- Damaged model has to contain its own proxy to follow deformed / different geometry (LODs do not need a proxy)

LODs

- For general information refer to the Creating LODs section for specialties in vehicle creation, please see below.
- Because of the asset complexity it is recommended to link subsequent LOD's to the LOD0 version and include the LOD0 name in the suffix (i.e. "hull" has LODs linked to it called LOD1_hull; LOD2_hull).

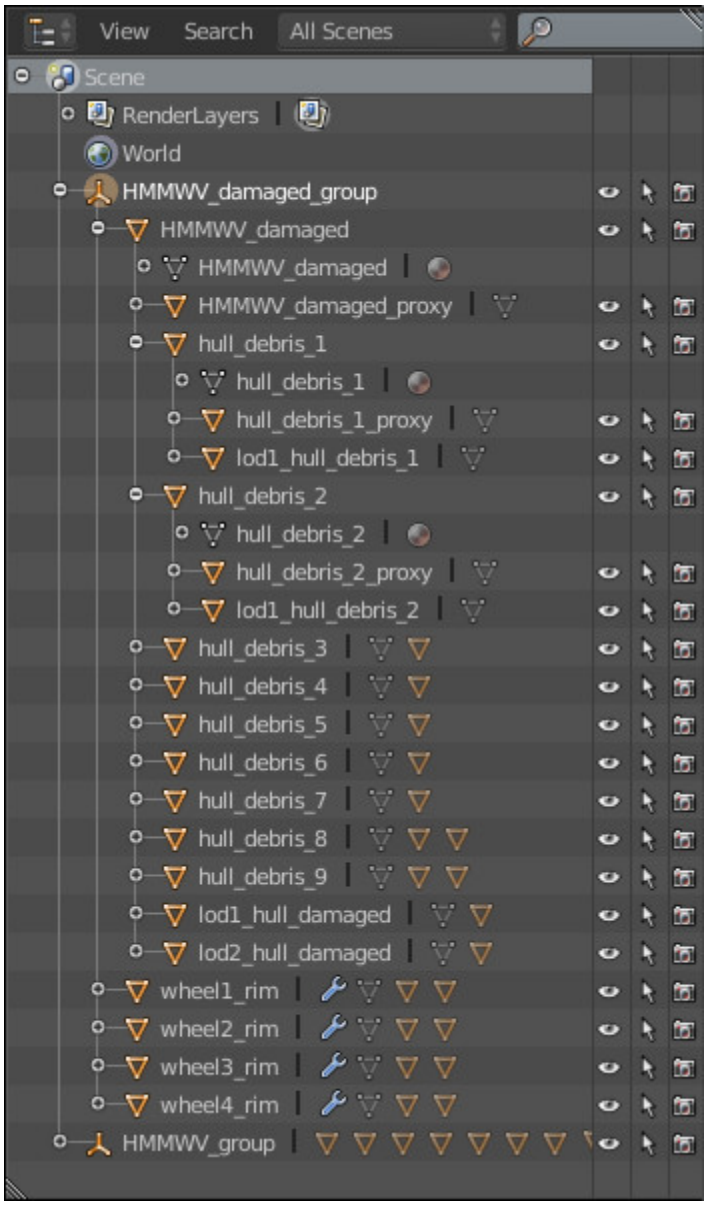
  Note: You can have different number of LOD levels for different parts of the vehicle. i.e; the chassis of a car may have 4 LOD levels whereas the door, being a more simple object, has only 2.
  LOD switching is based on screensize, smaller objects will switch earlier than big ones. Keep this in mind when doing LOD's.

Vehicle Damage Model

The damage model for vehicles is a separate asset which replaces the vehicles on destruction. It can be a simple static model or a complex pre-broken object.

- It replaces the original model and therefore needs a separate proxy model.
- The damaged objects needs the suffix "**_damaged**" to its name (like hull -> hull_damaged, door_left_1 -> door_left_1_damaged, etc).
- To have some extra debris (if one object is breaking apart to more debris parts), you can add further objects to the damaged model. These should be called the object's name + "*debris_*" suffix. (hull -> hull_debris_1,

hull_debris_2, etc.) These will spawn and fly away from the blast when the vehicle explodes.



Note: It happens only when the SpawnDebris damage behavior is set on a vehicle. Please refer to the Sandbox Vehicle Editor guide for this!

- The damage model can also include LOD's.

  Note: To manage objects easier, it is recommended to sort LOD's different layers in scene according to LOD levels.

  ### Animations

- Animations have to be saved as **ANM** files. To save certain animations, you must select the objects you would like to save the animations of, and add to the export list. If you want to animate the doors closing, then you have to export with only the doors in the export list. File names has to start with the object names + the animation's name. If the vehicle's main file is called **car.cga**, then the animation should be called like **car_door_opening.anm** for example.
- Once you animated your objects, you have to export the default state. It has to be called the same like the object .cga file (if the object is tank.cga then the default animation has to be tank.anm). This default state has to contain all the animated objects.
- Please see exporter documentation for further exporting features and parameters!

  Note: You can test if your animations are exported and playing correctly by loading the vehicle to the character editor, and clicking on the animation on the right.

  ### Position Helpers for Characters

- Character sitting and enter positions need to be indicated with position helpers in the 3D package (i.e. **Empty** objects in Blender).

  ### Export Options

  Vehicles are very complex objects and can cause major problems, if the exporting is not done carefully. To avoid complications in the scene:

- All exported objects have to be in BCRY **CGA** node.
- Vehicles except Tanks: export all parts as one **cga**. Uncheck and "merge all nodes".
- Tanks: export all parts except the tank treads together as one **cga**. Uncheck and "merge all nodes"

- Tank Treads: Export right and left tank tread separately as chr. uncheck “merge all nodes”; if you like. Choose only the tank treads object, not the wheels or the bones.

  ### Setup in Sandbox

  In order to work properly, a vehicle script is required. Otherwise the vehicle can only be judged visually as a Geom Entity.

- Drag and drop the item from the entity section in the RollupBar or the Entity Library (in the Database View) into the level.
- If no script is available, drag and drop the asset from the Geom Entity section in the RollupBar into the level.

  Please refer to the Vehicle Editor section for information on the vehicle editor.

  ### Debugging

  There are some useful commands you can use to check your asset, and see if it works as expected. Some of these are listed below (you can turn off any of these drawing modes by entering 0 after the command instead of the number given here!).

CVar	Description
p_draw_helpers 1	Used to check physique proxies.
e_debug_draw 1	Used to check LOD’s. Using this variable you can see the bounding box of the object, see the name of it, number of LOD levels available for the object, the one which is currently displaying, and the polycount of the current LOD.
e_debug_draw 2	Displays polycount only for the given LOD.

e_debug_draw 3	Different LOD levels are represented in different colors, so you can clearly see the transition between two LOD levels. Also see the total number of LOD's and the one currently displaying. Blinking one shows up if the given object does not have an LOD.
e_debug_draw 5	Shows how many material IDs are in use on the given objects.
e_lod_min 0 – 6	Shows the specified LOD number as LOD0 – very handy for debugging LOD's in the engine.

Character Exporting

Note: *Character related chapters may need more detailed introduction. I will add them as soon as possible when I got some time.*

The character definition files are represented with **CDF** (Character Definition File). A **CDF** file defines your final character which is composite of skeleton (.chr), skins, and attachments.

Skeleton (Armature) Exporting

The skeleton files are represented with **CHR** (character) extension in CryEngine.

To export a character file, you need to achieve some prerequisites on your skeleton:

1. The skeleton has to be Zero-Transformation.
2. The skeleton has to has a Root Bone.
3. The skeleton has to be skinned with a Primitive Mesh.

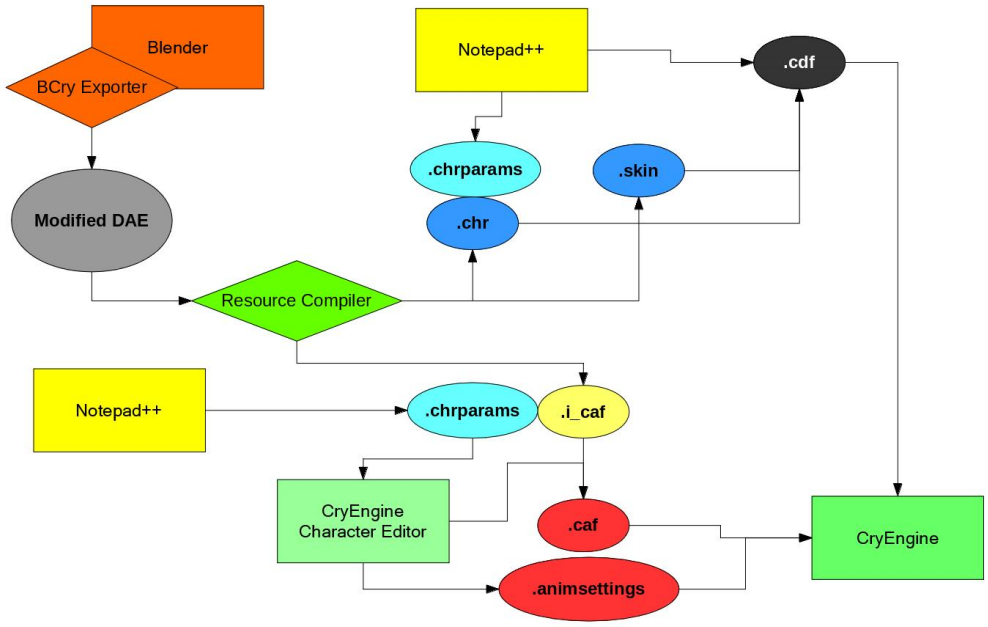
After you make ready prerequisites for your skeleton, you can export it easily with following steps.

To export a skeleton:

1. Select your primitive mesh which weighted to your skeleton.
2. From **BCry Exporter** select **Add Export Node** menu.
3. Select **CHR** as Type and enter node name.
4. From **BCry Exporter** menu click **Export to CryEngine**.
5. Select folder that you want to export the object.
6. Click **Export to CryEngine**.

CHR file will be created for you.

CHR files are just used for skeleton informations. The CryEngine Characters (defined with **CDF**) are composite of chr, skin, and material files. That information are stored in **CDF** (Character Defination File) in CryEngine.



Skin Exporting

Skin files are used to add weighted skins to your characters.
The objects which are skinned to a skeleton are represented with **SKIN** files in CryEngine.

Prerequisites:

- Your skeleton(armature) has to be Zero-Transformation.
- Your skeleton has to be has a Root Bone.
- All vertices have to be weighted at least to a bone for your skin mesh.
- Each vertex has to be assigned with maximum 8 bone. (For CryEngine 3.8.6 and before maximum 4 bone)
- All vertices have to be normalized for your skin mesh.

To Export Skin:

1. Select your weighted mesh.
2. From **BCry Exporter** Menu click **Add Export Node**.
3. Type name as you wish and select **SKIN** as node type.
4. From **BCry Exporter** Menu click **Export to CryEngine**.
5. Check **Generate Materials** box to (re)generate material file (Optional)
6. Click **Export to CryEngine** button.

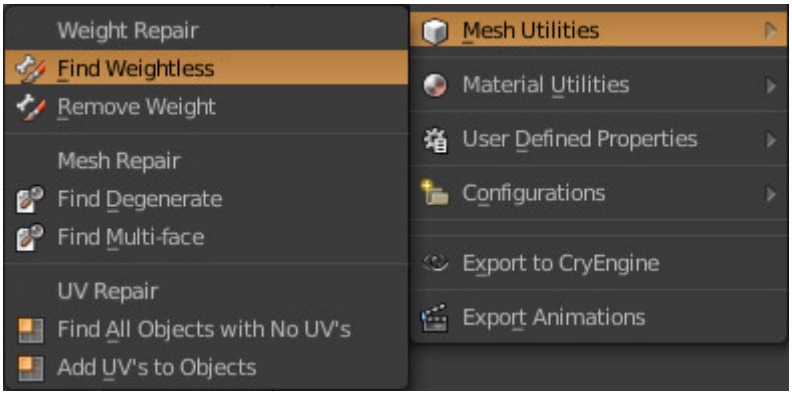
In order to add your exported skin to your character (**CDF** file), you can use CryEngine **Character Tool**.

Find Weightless Bones

All vertices have to be weighted at least to a bone for your skin mesh. If your mesh has vertices that unassigned a bone, resource compiler cannot create your skin file. And it will return an error:

To fix that:

1. Select your skin mesh from scene.
2. Run **Find Weightless** Tool from **BCry Exporter -> Mesh Utilities Menu**.

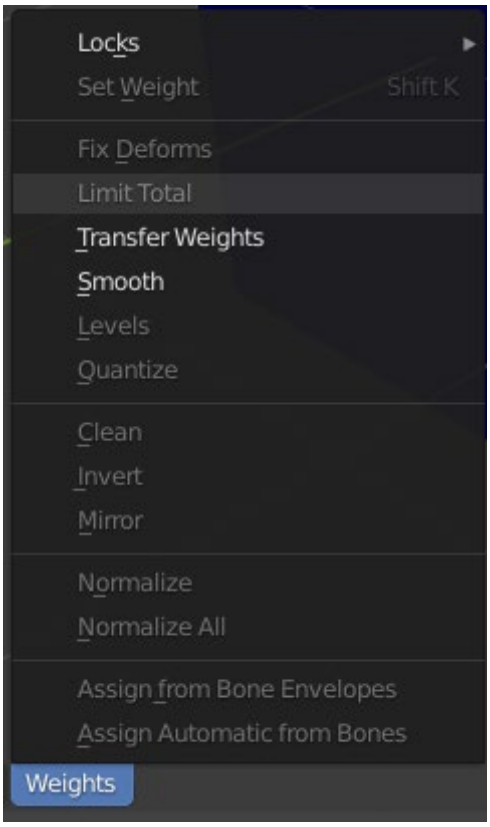


BCry is going to find and select unassigned vertices for you. Now, you can assign them

to any bone (almost nearly) by click Assign from **Properties Editor -> Object Data -> Vertex Groups**. You should use 1.0 as **weight** value.

Weight Limit Total

Each vertex has to be assigned at maximum 8 bone. (For CryEngine 3.8.6 and before maximum 4 bone). To limit bone assignment:

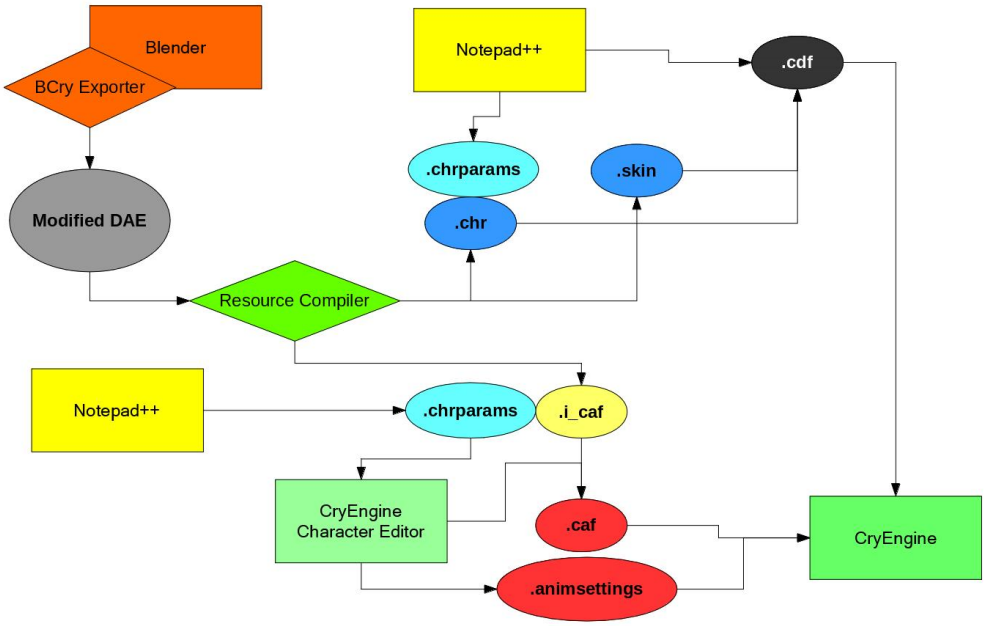


1. Select your skin mesh from scene.
2. Change mode to **Weight Paint** from 3D View Menu Bar.
3. Click **Limit Total** Tool from **Weights**.
4. Enter 8 for Limit value from Tool Shelf panel. (If you use CryEngine 3.8.6 or earlier version, you have to enter 4 as Limit value)
5. Run **Normalize All** Tool to normalize vertices from **Weights**.

Weight Normalize

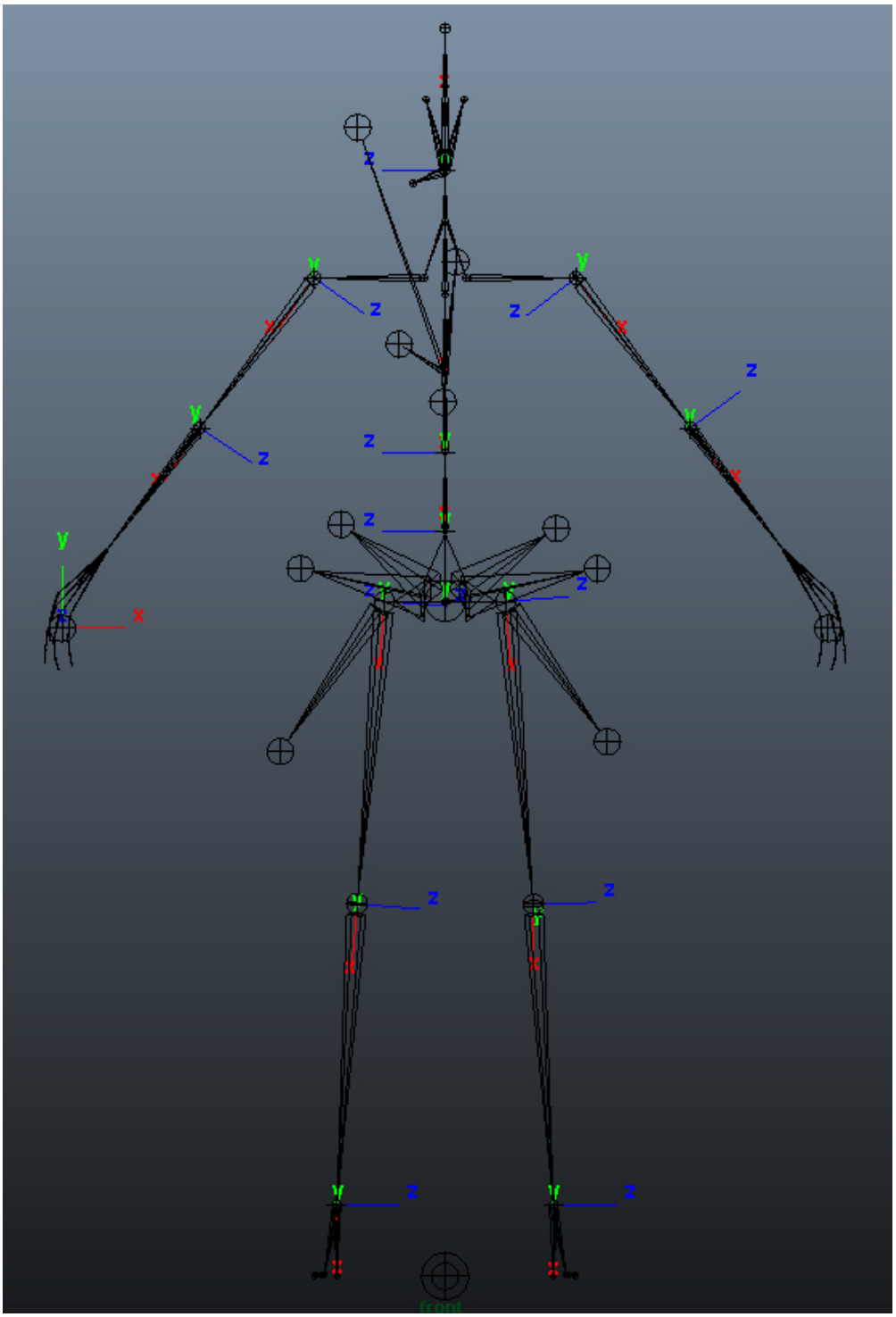
All vertices have to normalized in your skin mesh, to normalize vertices weight:

1. Select your skin mesh from scene.
2. Change mode to Weight Paint from 3D View Menu Bar.
3. Run **Normalize All** Tool to normalize vertices from **Weights**.


Blender
BCry Exporter
Modified DAE
Notepad++
.cdf
.chrparams
.skin
.chr
Resource Compiler
Notepad++
.chrparams
.i_caf
CryEngine
Character Editor
.caf
CryEngine
.animsettings

Axis Orientation

If you want to use **CryEngine** SDK animations for your custom characters, you have to set correctly bone orientations for your skeleton. **CryEngine** uses **X-Axis** bone orientation for SDK characters as you have seen below image.

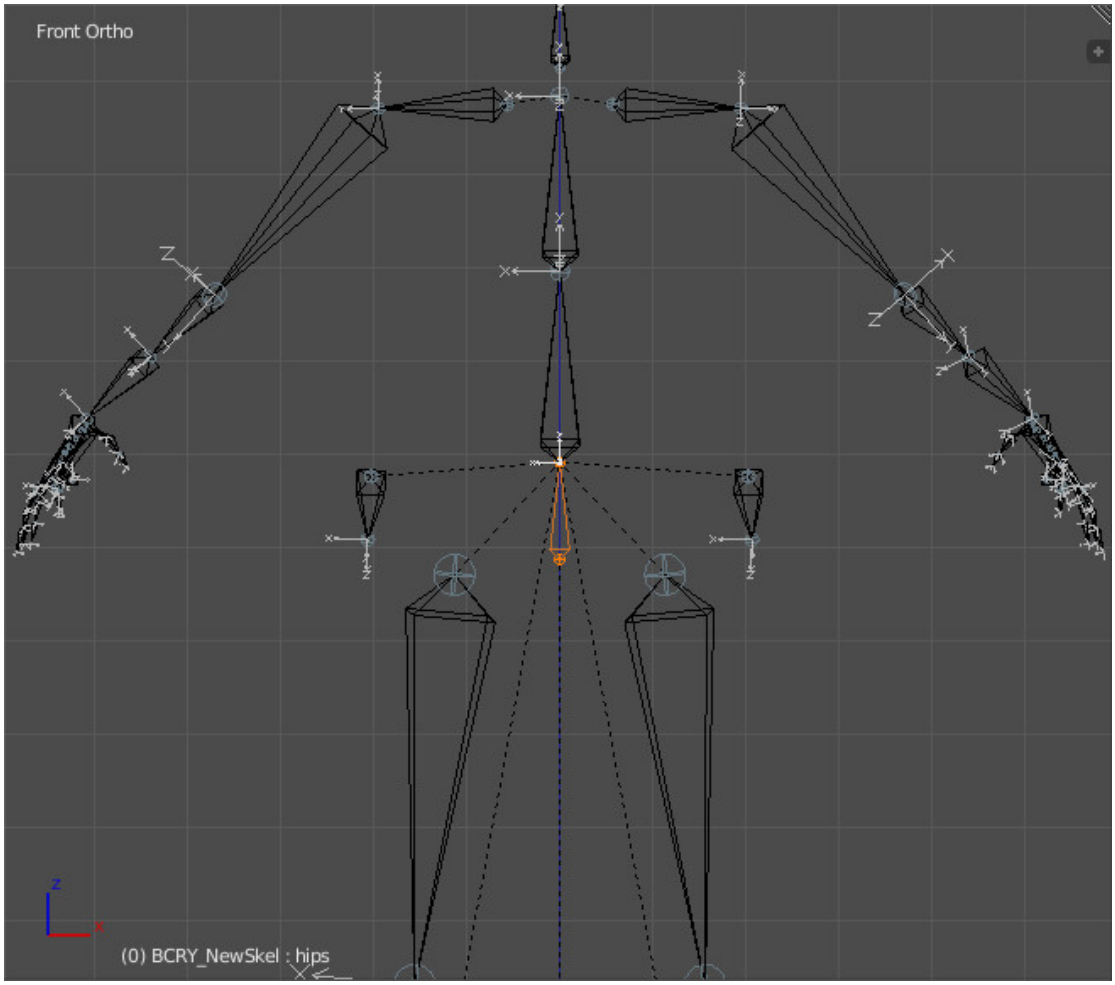


CryEngine SDK character bone orientation

The joint orientations should be aligned as described:

- Arms: z= forward, X= downward along arm
- Legs: z= forward, x= downward along legs
- Eyes: Y= forward, z= along vertical axis

Unfortunately, **Blender** only supports **Y-Axis** as primary orientation for bones. Although you import a skeleton which has **X-Axis** oriented bones, Blender converts it to **Y-Axis** oriented bones form at importing stage.



To solve that problem, **BCRY Exporter** changes bone orientations from **Y-Axis** to **X-Axis** at exporting stage.

Conversion bone orientation at exporting stage:

- **Y-Axis** are converted to **X-Axis**
- **X-Axis** are converted to **Y-Axis**
- **Z-Axis** are converted to **–Z-Axis**

You should be noticed that Z axis changed to -Z axis. That requirement occurs after swapping of X-Y axis.

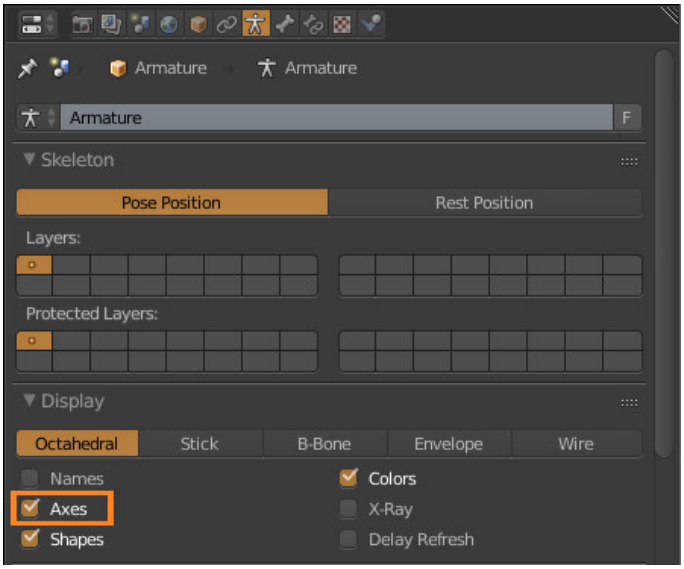
Custom Skeleton

If you already use **BCRY Exporter Player** SDK skeleton, you do not care much about those. Bone orientations have already set for you.

But if you want to use your own custom skeleton and you want to use as well CryEngine SDK animations you have to set your bone orientations by considering those conversions.

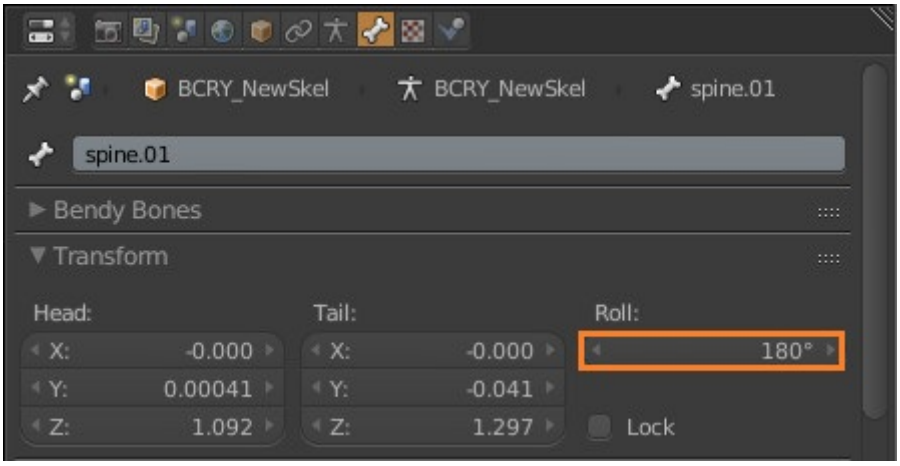
Blender only allows you to set **Roll** for bone orientations. Before to do that, you should make bone axes visible:

1. Select your skeleton from scene.
2. Go to **Armature** tab from **Properties Editor**.
3. Check **Axes** box under **Display** panel.



Bone axes become visible at **EDIT MODE**. Now, you can set bone rolls:

1. Select your skeleton from scene.
2. Change mode to **Edit Mode**.
3. Select bone you want to set orientation.
4. Go to **Bone** tab from **Properties Editor**.
5. Change **Roll** value from **Transform** panel.

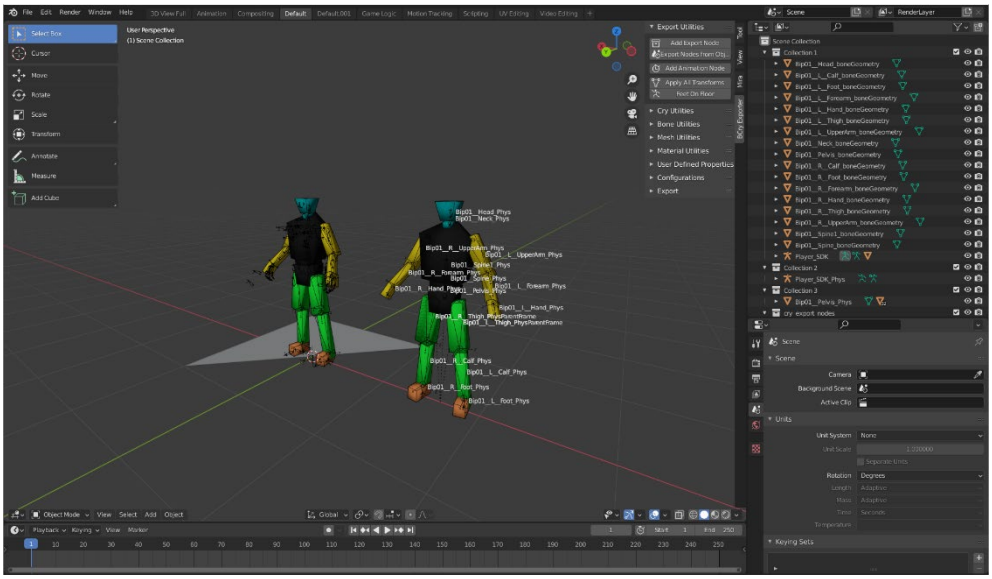


When you finish arrangement, you can export your skeleton as documented.

Player SDK

Note: *The Player SDK skeleton is shipped with original BCry Exporter 5 from AFS Studio. The following content are original from them.*

BCRY Exporter has **CryEngine Character SDK** (under **examples/Player SDK**), which you can use as reference when you create your skeletons or you can directly use it for your skin meshes, to use CryEngine SDK Animations.



Note: If you want to use CryEngine SDK Animations with your custom skeletons you need to consider Bone Axis Orientations.

Important Note:

We're almost fit completely all bone position and orientation with physical proxies except facial bones. Although all our works, we have one last problem about physical proxies' behavior when Player SDK under shoot. We can't fix that yet. IKs seen completely seem in CryEngine Character Tool. We use damping and parent frame settings from Agent.ma. Character Tool doesn't facility to see them actual values. If you have any suggestion to help with that problem, please don't hesitate to contact us.

Character Physic and IKs

CryEngine uses two skeletons for each character that you want to become live in your game. First of those is the **live skeleton**, the second is **ragdoll skeleton** which defines physics for character. The ragdoll is also used for the death characters. When a character is death, its live skeleton is changed to ragdoll skeleton.

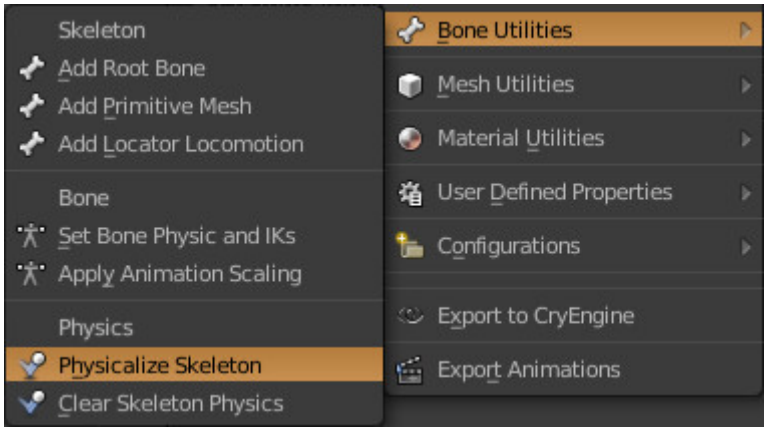
For detail information, I advise you to read [CryEngine documentation]().

To create physics for a character is a bit cumbersome and long procedure:

1. You have to create radgoll skeleton.
2. You have to create bone physical proxies, and assign materials to those.
3. You have to identify primitive shape for your bone physical proxies.
4. You have to set Inverse Kinematics Limits.
5. You have to set stiffness, stiffness tension, and damping, for each of your bone that have a proxy.

Fortunately, **BCRY Exporter** has **Physicalize Skeleton** tool which is dealing all of those instead of you.

To Physicalize Skeleton:



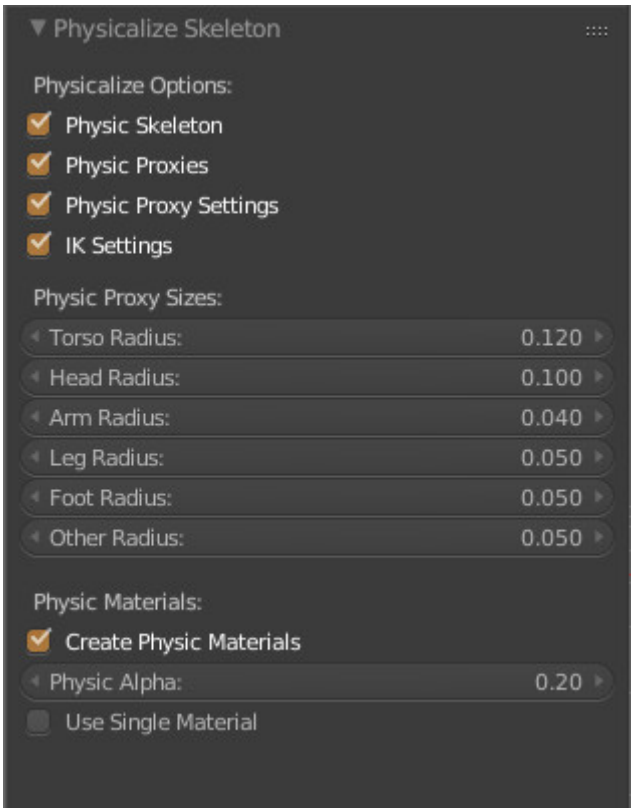
1. Select your skeleton from scene.
2. Change mode to **Pose Mode** from 3D View menu bar.

3. Select bones which you want to consist of **ragdoll** skeleton.
4. Run **Physicalize Skeleton** tool from **BCry Exporter -> Bone Utilities** menu.

Your ragdoll skeleton is going to be created for you.

Note: You have to include main (pelvis or hips) bone for the physic, otherwise your proxy doesn't seem correctly.

After your physic skeleton has been created, you may set physical properties from Blender **Tool Shelf** panel.



Physic Skeleton: Use to generate physic skeleton with ends with **__Phys**.

Physic Proxies: Use to generate physic bone geometries.

Physic Proxy Settings: Writes physic information to bone. (capsule, cylinder, box …)

IK Settings: Writes default inverse kinematics information to bone.

Physic Proxy Sizes: Use to set radius of proxy bones.

Create Physic Materials: Adds materials to proxy geometries.

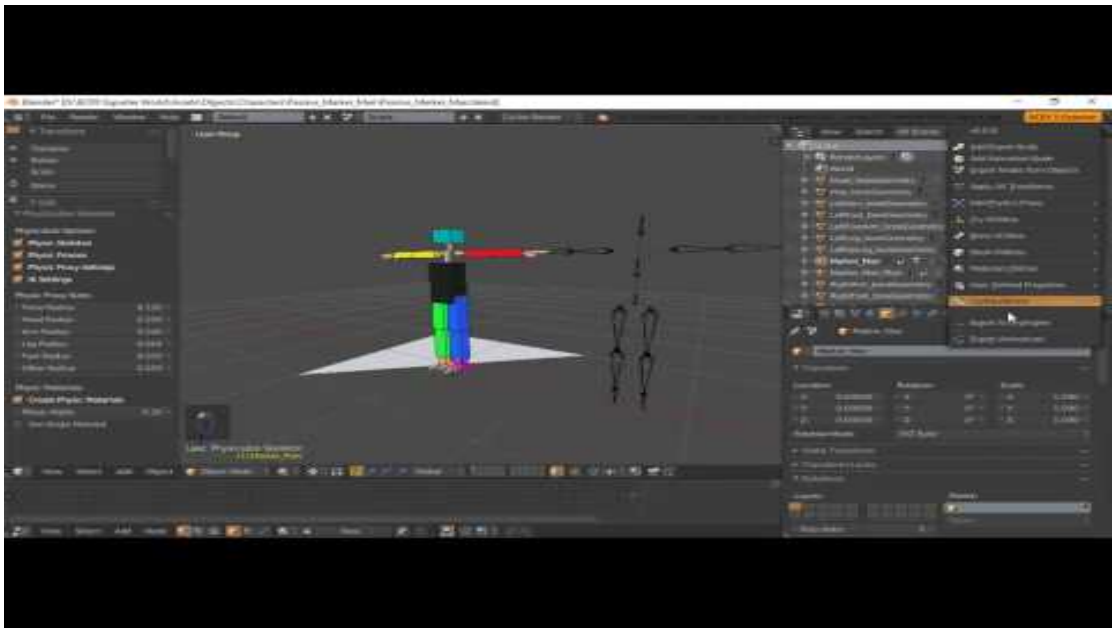
Use Single Material: Use to create single material for all proxy geometries.

Both of live and ragdoll skeletons are stored in a CHR file. So, you have to set them together in a blender scene.

Right after get ready physics for your skeleton, just reexport it like before.

Resource Compiler creates your **CHR** file including character physics.

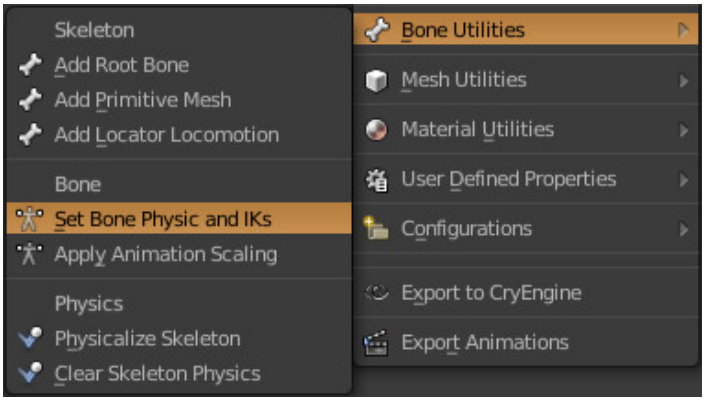
Note: Ragdolls are only stored in CHR files. You shouldn't add it to a SKIN node.



Video Link

Edit Bone Physic and IKs

If you want to manually set physic or IK values for a bone, you may use **Set Bone Physic and IKs** tool for that.



To do that:

1. Select your skeleton from scene.
2. Go to **Pose Mode**.
3. Select you want to set bone in **Pose Mode**.
4. Run **Set Bone Physic and IKs** tool from **Bone Utilities** menu.
5. From opened panel, arrange your bone settings as you wish.

When you finish your arrangement, click **OK** button.

You can also use Inverse Kinematics panel from **Bone Properties** in the **Properties Editor**.

VCloth

VCloth 2.0 is the cloth simulation feature of CRYENGINE. Using VCloth, your character's cloth can be simulated and animated automatically, according to their actual movement. This can make it look much more realistic.

Example Video: Different Cloth Parameter

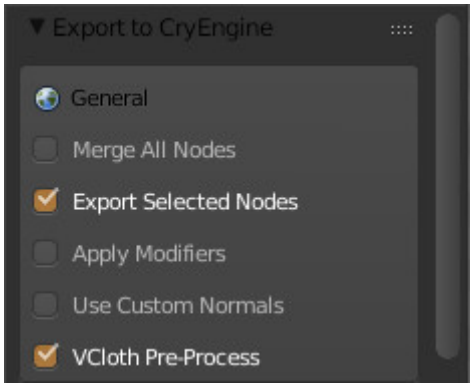
Example Video: Many Characters

For more information you must look [CryEngine Manual](#).

VCloth V2 Export

BCRY Exporter can be export VCloth skins for **CryEngine**. To do that:

1. You must create your **skin** as normally and export it with BCRY. That is going to be your VCloth **render mesh**.
2. Secondly, create or retopologize or duplicate (CryEngine do not advice to use high polygon for VCloth) your **render mesh** for VCloth **simulation mesh** which do not rendered only used for simulation your main mesh.
3. Again, make ready that mesh as normally as **SKIN** meshes (You must use a **SKIN** node different from render mesh).
4. Change mode to **Vertex Paint** mode and paint your VCloth simulation object as you wish. (Black color vertices are fully controlled by weighted bones.)
5. Select your **simulation mesh** from **Blender** scene.

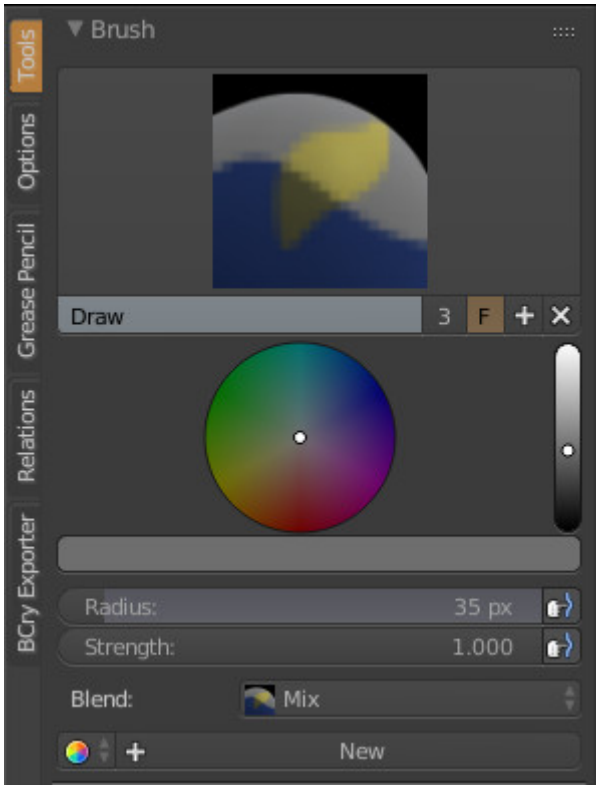


6. Run **Export to CryEngine** tool from **BCRY Exporter** menu.
7. Tick **Export Selected Nodes** and **VCloth Pre-Process** boxes. (Most important)
8. Press **Export to CryEngine**.
9. It's done.

Note: Simulation VCloth mesh must be exported as alone. Therefore we use **Export Selected Nodes** box while exporting. Otherwise your render mesh processed as simulation as well.

Paint Vertices

VCloth use vertex colors, to define what vertices, how much, are controlled by bones or VCloth. Vertices fully colored black are fully controlled by weighted bones. As contrary, vertices fully colored with white are fully controlled by VCloth options. If your vertex is a tone of gray, in that situation, both of bones and VCloth options control the vertex as color proportional.



You can paint your vertices in Blender while in **Vertex Paint** mode:

1. Select your VCloth mesh from scene.
2. Change mode to **Vertex Paint** mode from **View Editor**.
3. Set color from **Tool Shelf** panel.

4. Paint your mesh as you wish.

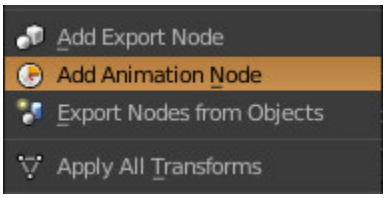
VCloth V2 Importing to CryEngine

After you export your VCloth **render** and **simulation mesh** you can import them to your character with **CryEngine Character Tool**. To add VCloth to your character you may follow Tutorial – VCloth 2.0 Setup from CryEngine Manual.

Character Animation Exporting

The skeleton animation files are represented with **CAF** (Crytek animation file) extension in CryEngine. The **CAF** file is compressed version of animation file. Initially Resource Compiler create a **I_CAF** file for animation then it's compressed to **CAF** file according to compression settings which may be set from Character Tool in CryEngine.

To export a skeleton animation:

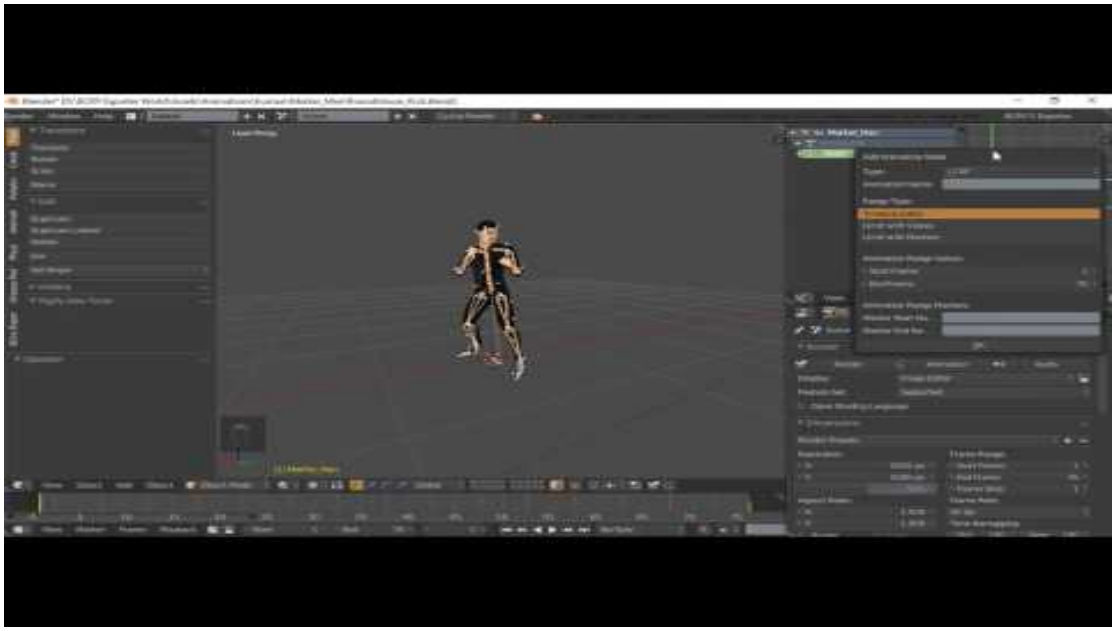


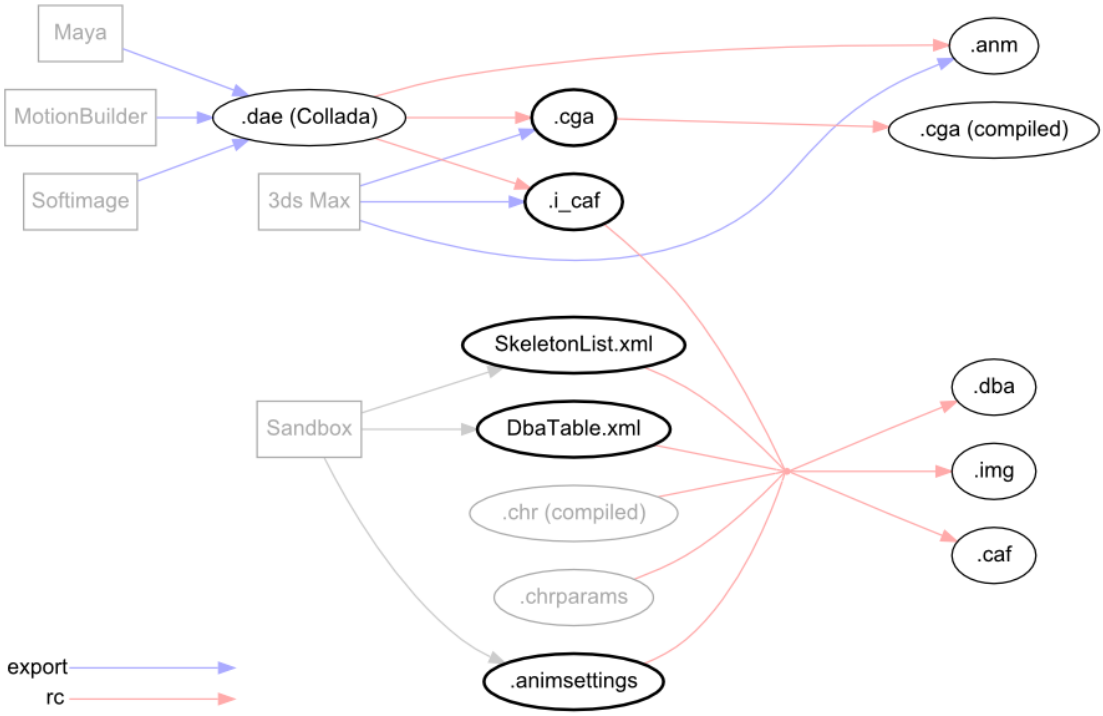
1. Select armature (Blender skeleton) you want to export animation.
2. From **Bcry Exporter** select **Add Animation Node** menu.
3. Select **I_CAF** as Type and enter node name (that name reel name which shows in CryEngine).
4. Set **Timeline Editor** as Range Type.
5. Press **OK**.
6. From **Bcry Exporter** menu click **Export Animations**.
7. Select folder that you want to export object.
8. Press **Export Animations**.

I_CAF file will be created for animation.

CryEngine cannot recognize which animation is exported for which skeleton (**chr** file) unless you define. To define connections between animation and skeleton, you must enter animation name and path to **chrparams** file of skeleton.

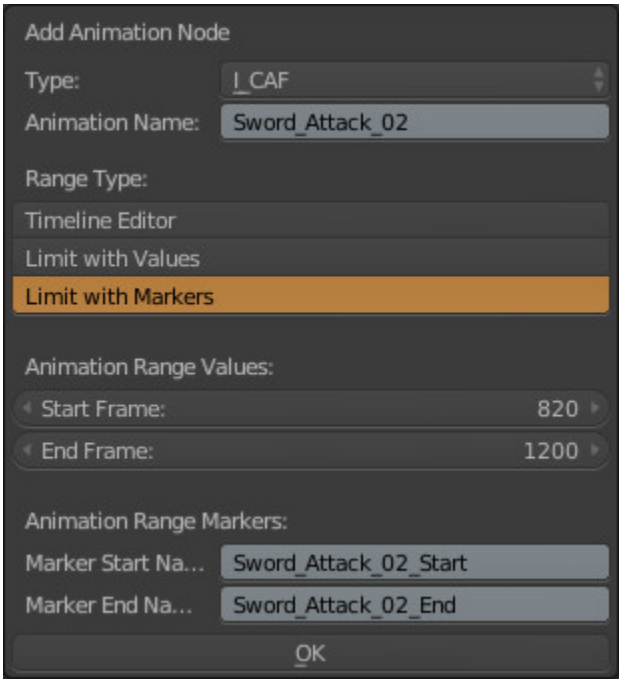
Video Link





To export multiple animation:

You can add more animation node to a skeleton with the previous **To export a skeleton animation** steps. You can set range your animation with **Range Type** from **Add Animation Node** menu (step 4). *(#TODO: Introduce skeleton extension animation export process and tips.*)



Range Types:

Range type is used to determine range of animation. You can use one of three range types for per animation:

Timeline Editor: Animation range is set from Timeline Editor. You may directly change start and end frame from Timeline Editor. This is best choice for single animation per file.

Limit with Values: Animation range is stored in custom property values. You must enter animation range values. You can change start or end frame from object custom properties. This is ideal for multiple animations in a blender project.

Limit with Markers: Animation range is stored on animation markers. Markers can directly be set on Timeline Editor to change frame range. You must identify start and end markers for that property. If there already are markers with that name, those are used directly. If you enter markers name that had not created, new markers are added at Animation Range Values.

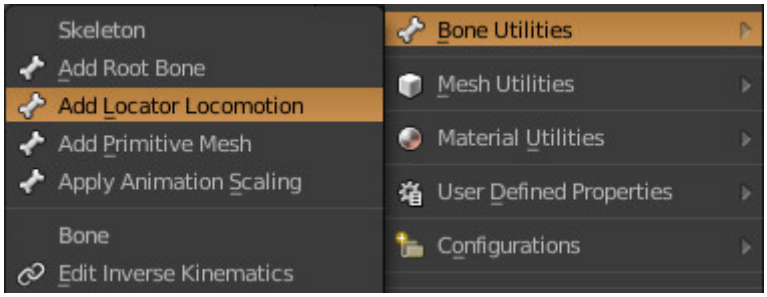
Locator Locomotion

In CryEngine, animations cannot directly change character entity world location or rotation. The **Locator_Locomotion** bone is used to determine character location and rotation offset in game mode at CryEngine. CryEngine get move speed and turning information from Locator_Locomotion bone. Then use that information at Blend Spaces to move character entity.

You can easily add Locator_Locomotion to your skeleton with use **Add Locator Locomotion** tool from BCry menu. That tool also set some CryEngine requirements of the Locator_Locomotion bone for you.

Add Locator Locomotion Tool:

1. Select armature/skeleton that you want to add Locator Locomotion.
2. Select **Add Locator Locomotion** from **Bcry Exporter -> Bone Utilities** menu.

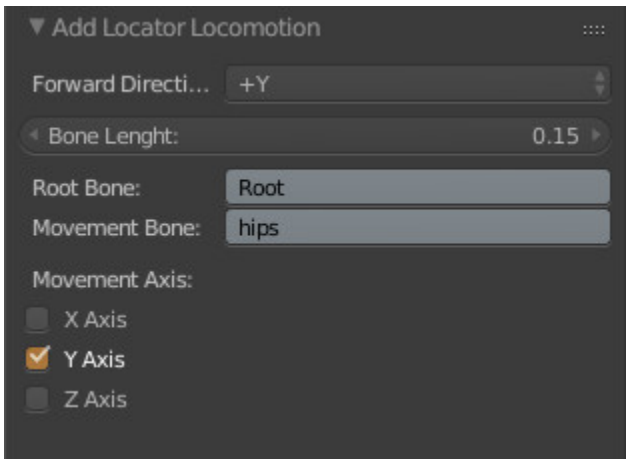


BCryExporter is going to create a bone named **Locator_Locomotion**.
And that bone use Copy Location constraint which copied **Y Axis** location animations from hips to your LM bone.

You can set **Locator Locomotion** settings as you wish from **Tool Shelf Panel**:

1. Set your bone direction and bone length as you wish.
2. Be sure your root bone name and movement reference bone name is set correctly.

3. Set your axis movement.



If you don't want to use hips bone information for **LM**, you can uncheck all axis from **Tool Shelf Panel**. Then you can animate your LM bone as you wish.

If you use **Copy Location** constraint, which default behavior of **LM** tool, you have to bake it to skeleton before export animation.

Bake Animation:

1. Select your animated skeleton from scene.
2. Change mode to **Pose Mode**.
3. Select all bones.
4. Click **Bake Action…** from **Object -> Animation** menu.

Note: When you use bake animation tool you must be carry about that: All other keyframes are going to be deleted which be out of Bake Range.

After you have finished Locator Locomotion animations.
You can export animation.

Note: Locator_Locomotion bone used only for animated skeletons (armatures). You should **NOT** add it to your CHR file.