Introduction

MilkShape is a 3D modelling program from chUmbaLum sOft oriented towards building art for games. MilkShape can be used to build 3D shapes for the Torque Game Engine. MilkShape is shareware and a free 30 day trial is available. It can be registered for $20. The exporter ms2dtsExporterPlus.dll is an extension to MilkShape which allows it to export the DTS files used by the Torque Game Engine. Note that you must have at least Milkshape version 1.7.8 installed to use this exporter.

To use the exporter, just copy ms2dtsExporterPlus.dll to your Milkshape install directory. The next time you start Milkshape, it will appear under the export list as "Torque DTS Plus...". You should also copy the ms2dtsplus.chm help file to the Milkshape install directory to allow access to the documentation directly from the exporter dialogs. The new exporter can co-exist with the old exporter (ms2dtsExporter.dll). It should operate similarly to the original, and most models can be exported with very few changes. You can now view and edit meshes, materials and sequences before you export them. Changes will be applied to the model unless you hit Cancel.

This exporter is not yet completed. Some features are still missing, and it is possible that there are problems with those features that are implemented. If you find an error, you can help the development of this tool by providing a description of the problem, and if possible, the .ms3d, .dts, and dump.dmp files involved. Some features are listed as **UNTESTED**, these may or may not be 100% functional.

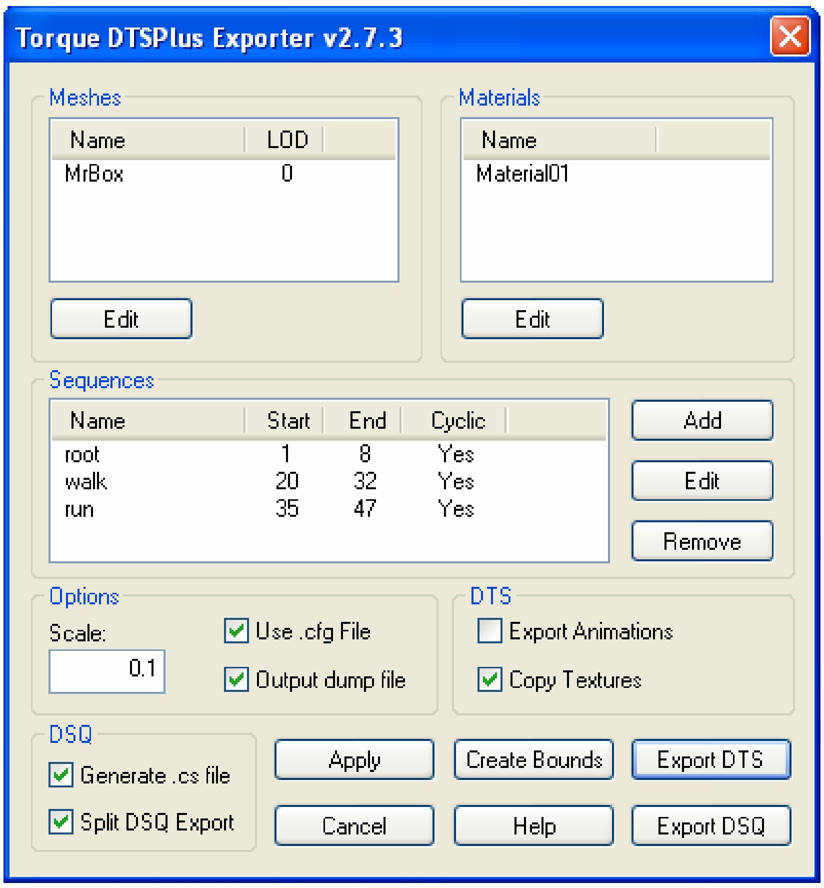
Chris Robertson

Main Dialog

The main exporter dialog box appears when you select the "Torque DTS Plus..." exporter from the File->Export list. All meshes, materials and animation sequences that are to be exported are listed here, and many of the properties of each can be modified before exporting.

To edit a mesh, material or sequence, select the object by clicking the name in the first column of the list. Then press the Edit button to open the relevant edit dialog. Any changes made while the main dialog box is open are applied to the model unless you select Cancel.

The Milkshape SDK does not support shared vertices between mesh groups, so after exporting, seams may appear in the milkshape model that are not present in the exported model. This is easily resolved by selecting the vertices and rewelding.



Name	LOD
MrBox	0

Name
Material01

Name	Start	End	Cyclic
root	1	8	Yes
walk	20	32	Yes
run	35	47	Yes

Scale: 0.1

☑ Use .cfg File

☑ Output dump file

☐ Export Animations

☑ Copy Textures

☑ Generate .cs file

☑ Split DSQ Export

Meshes

Lists all meshes that will be exported, as well as their detail level. Edit a mesh by clicking its name, then selecting Edit.

Materials
: Lists all materials that will be exported. Materials in the model that are not attached to any mesh are not included. Edit a material by clicking its name, then selecting Edit.

Sequences
: Lists all defined animation sequences, as well as some of their properties. Edit a sequence by clicking its name, then selecting Edit.

Add
: Add a new sequence. The sequence editor dialog box will open so you can edit the new sequence. New sequences will be added to the Milkshape model unless you press Cancel.

Remove
: Removes the selected sequence. Removed sequences will be removed from the Milkshape model unless you press Cancel.

Scale
: Global scale factor applied to the model when it is exported.

Use .cfg File
: If checked, the exporter will search for a config file with the same name as the exported shape. eg shape.cfg for the exported shape.dts. If unchecked, the default configuration will be used. See Default Configuration.

Output Dump File
: If checked, a file called dump.dmp will be created in the same directory as the exported shape. See Dump Files

Export Animations
: If checked, animation information will be written to the DTS shape. This flag is ignored when exporting DSQ files.

Copy Textures
: If checked, all textures used in the exported shape will be copied to the export directory. This flag is ignored when exporting DSQ files.

Generate .cs file
: If checked, a TorqueScript .cs file will be created that can be used to load the shape (with DSQ animations) in TGE. This flag is ignored when exporting DTS files.

Split DSQ export
: If checked, each animation will be stored in a separate DSQ file. The name of each file is *base_animname.dsq*. Where *base* is the name chosen in the 'Save As' dialog, and *animname* is the name of the animation. If this flag is unchecked, all animations will be stored in the same DSQ file. This flag is ignored when exporting DTS files.

Apply
: Apply changes to the Milkshape model. The exporter dialog box will remain open.

Cancel
: Close the exporter dialog box without applying any changes.

Help

Display this page.

Create Bounds Mesh

Creates the bounding box mesh and Root bone if they do not already exist. The bounding box is a cube 1 Milkshape unit larger than the extents of the current model. The changes will be applied to the model unless you press Cancel. See Shape Structure.

Export DTS

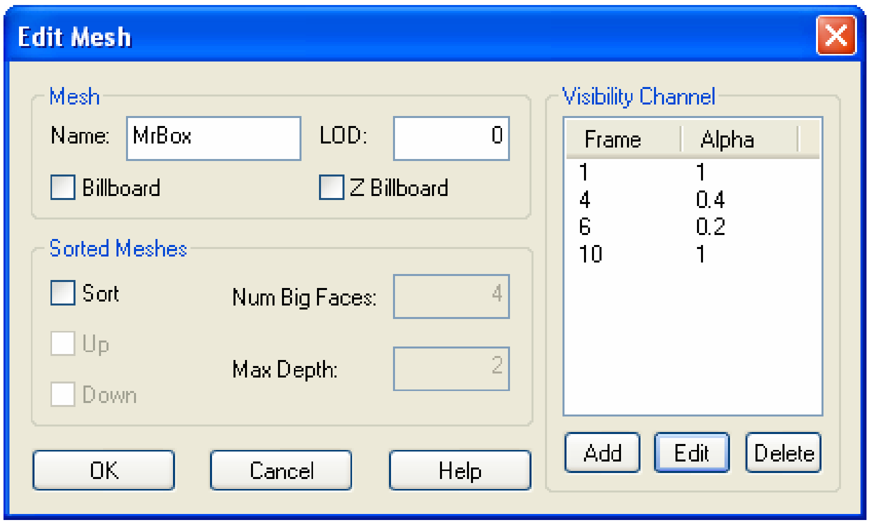
Export the current shape to a dts file.

Export DSQ

Export all animation sequences to a dsq file. See DSQ Export.

Meshes

Most properties of a mesh can be edited using the Mesh Edit dialog box shown below.



Name: MrBox
LOD: 0
☐ Billboard
☐ Z Billboard
☐ Sort
☐ Up
☐ Down
Num Big Faces: 4
Max Depth: 2

Frame	Alpha
1	1
4	0.4
6	0.2
10	1

Name

Name of the mesh, not including the LOD number which is automatically appended to the end of the name.

LOD

Detail level for this mesh. The detail level indicates to the exporter what mesh is to be drawn at a given distance. The number corresponds to the pixel size in the game engine at which the shape will draw with these meshes. Meshes with negative detail levels will be exported, but not drawn. If your mesh has only one detail level, use 0.

Billboard

Checked if this mesh is a billboard. See Billboards for more details.

Z Billboard

Checked if this mesh is a Z billboard. See Billboards for more details.

Sort

Checked if this mesh should be sorted. See Sorted Meshes for more details.

Visibility Channel

This list box defines keyframes for the meshes visibility channel. See Visibility for more details.

Level of Detail

Detail levels indicate to the exporter what mesh is to be drawn at a given distance. The number corresponds to the pixel size in the game engine at which the mesh will be rendered. This is done by naming different detail levels of the same mesh with the same base name but a different trailing number.

e.g. If you have meshes named 'head2' and 'head36', then when the size is 36 or greater the head36 mesh would be drawn. When the size is between 2 and 36 head2 would be drawn, and when the size was less than 2 nothing would draw.

Pixel sizes are inversely proportional to the distance an object is from the camera, so a larger value (like 36) indicates the object is much closer to the camera than a smaller value (like 2).

The original Milkshape exporter output shapes with all visible meshes at detail level 0 (ie render the same mesh no matter how far away it is from the camera). If you are not making use of LOD, this is the best value to give to visible meshes.

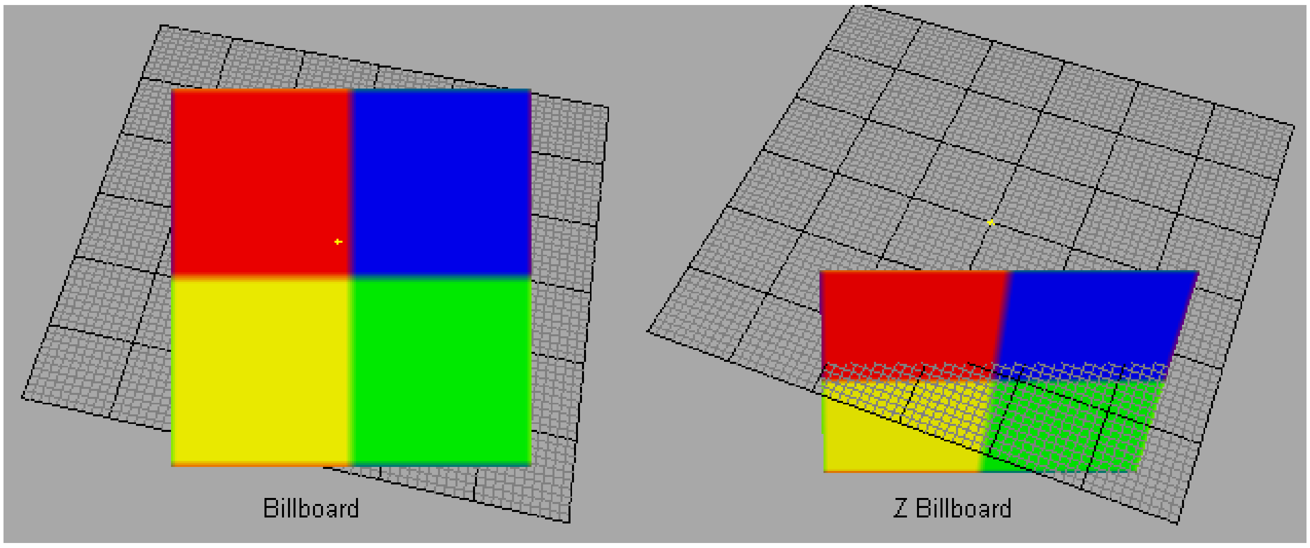
*Note: The exporter treats a trailing underscore character ('_') as a minus sign ('-'). Because of this, an underscore at the end of a mesh name will negate the detail level. eg. a mesh named **MyMesh_2** will be renamed to **MyMesh** with detail level **-2**. Underscores in the middle of the name are not affected. eg. **My_Mesh2** denotes a mesh called **My_Mesh** with detail level **2**.*

Billboards

Parts of a shape can be billboard objects (i.e., they always face the camera). So, for example, you could have an explosion in which shrapnel flies out from the center and also have little explosion balls fly out that are just flat polygons that always face you.

You make an object a billboard object by ticking the Billboard or Z Billboard check boxes in the Edit Mesh dialog box. Note that not all detail levels of the object need to be billboard objects, so the highest detail level of a shape could be a complicated 3d shape, whereas the lowest detail could just be a billboard. Z billboards are the same as regular billboards except that they only rotate about the z (vertical) axis.

Note: These objects tend to have strange sorting properties if translucent materials are used.



Sorted Meshes

Objects with translucent textures often times appear to sort improperly in the engine. On modern graphics hardware, drawing on the screen amounts to storing values on the graphics card for the red, green, and blue channel, and also storing values for the distance of the fragment from the camera. The later value is often referred to as the "depth-value" or "z-value". The depth value is important for determining what should be drawn in front of what.

To understand how this works, you have to understand one basic point: polygons are always drawn in an order. One is drawn first, another second, etc. So when the second is being drawn, the value of the first polygon is sitting in the frame buffer (the place on the graphics card that holds what you are drawing on the screen). This means that the graphics hardware can simply compare the depth value of the incoming pixel against the depth value of the stored pixel, and only update the frame buffer if the incoming pixel is in front of the stored pixel. That is exactly what happens.

Drawing translucent fragments also requires a combination of what is in the frame buffer already and the incoming fragment. With translucency, the incoming fragment has an "alpha-value" in addition to red, green, and blue, and the alpha value is used to blend the fragment with the framebuffer. An alpha of 1 means to over-write what's in the buffer, an alpha of 0 means not to touch the frame buffer, and an alpha of 0.5 means to mix them equally.

Translucent drawing with depth tests gets very tricky. If polygons are drawn back to front, depth tests and translucency behave well together. But when some polygons in the front are drawn first, things start to get very messy. Imagine what would happen if you had a fully translucent texture (alpha of 0) drawn first, and that it fully covered the camera and was in front of everything else. Since the alpha value is zero everywhere, it would not draw to the RGB channels. But the depth value would still be updated for the entire screen. Now everything that was drawn would fail the depth test. The result is that you would see a blank screen no matter what you draw behind the phantom polygon.

Because of this issue, translucent polygons are normally drawn with special care: the depth value is not saved but the depth test is still used. Translucent polygons are drawn after non-translucent polygons, and translucent polygons are drawn from back to front. The result is that translucent polygons behave when they overlap each other because they are drawn back to front. Translucent polygons behave when overlapping non-translucent polygons because they only drawn when they are in front of the non-translucent polygons (remember, the depth test is still carried out, the depth value just isn't stored). The phantom polygon issue is avoided because the depth value isn't stored.

One consequence of all this is that any object that draws translucent polygons must do so with special care. Furthermore, the engine itself must take special care to draw everything in the right order. In particular, the most accurate way for the game to draw the scene is to first draw the non-translucent polygons of all objects, then draw the translucent polygons of each object from furthest to closest to the camera. Each object, then, is only responsible for drawing it's own polygons so that they can sort amongst themselves.

Three space has several mechanisms built in to handle the sorting of polygons. First, parts with only non-translucent polygons are drawn first, then parts with a mixture of translucent and non-translucent polygons, and then translucent parts. Note that if you have several parts with mixed polygon types, you will likely get some inappropriate sorting, so don't do this. These are all the measures 3space takes by default. However, there are special objects that do a little more sorting on their own. These are the sort objects described below. What these objects do is order the polygons so that they will always draw back to front. Believe it or not, it is often possible to do this for all camera angles. This however, it is not always possible. In those cases, the object has different orderings for different angles (usually only a few are needed) and in really bad cases, polygons have to be split. The latter can sometimes lead to large file size. If you see this happening, you should redesign the shape.

The faces of these objects are presorted so that faces are drawn from back to front. This is used to force the sorting order of translucent objects (which are not z-buffered) This sometimes involves splitting faces and sometimes involves different orders depending on where the camera is.

To make an object a sort object, tick the Sort checkbox in the Edit Mesh dialog box. Other detail levels of this object do not have to be sort objects. You can also give the exporter some hints on how to create the sort objects. You supply these hints by editing the sort fields in the Edit Mesh dialog. The fields are:

Up

Used to sort objects with 'leaves' that are layered from top to bottom facing slightly up.

Down

Used to sort objects with 'leaves' that are layered from top to bottom facing slightly down.

NumBigFaces

TODO Default 4.

Max Depth

Maximum recursion depth when sorting mesh. Default 2.

UNTESTED

Visibility

Visibility keyframes can be defined to control the 'alpha' value of a mesh when it is rendered. Frames between keyframes are interpolated, frames outside the keyframe range are clipped to the keyframe range. Visibility ranges from 0 (invisible) to 1 (fully opaque). A sequence must have Enable Visibility set to use a meshes visibility channel.

Note: Only rigid meshes (ie meshes attached to a single bone) can have their visibility animated.

Collision Meshes

Any mesh whose name begins with 'Collision' will be used ingame as a collision mesh. Collision meshes are normally given a negative detail level from -1 to -8 so that they are not drawn, but you can make the collision mesh visible by giving it a positive detail level.

Collision meshes should use as few polygons as possible, and **must** be convex. The more polygons contained in the collision mesh, the greater the CPU load in determining collisions with other objects.

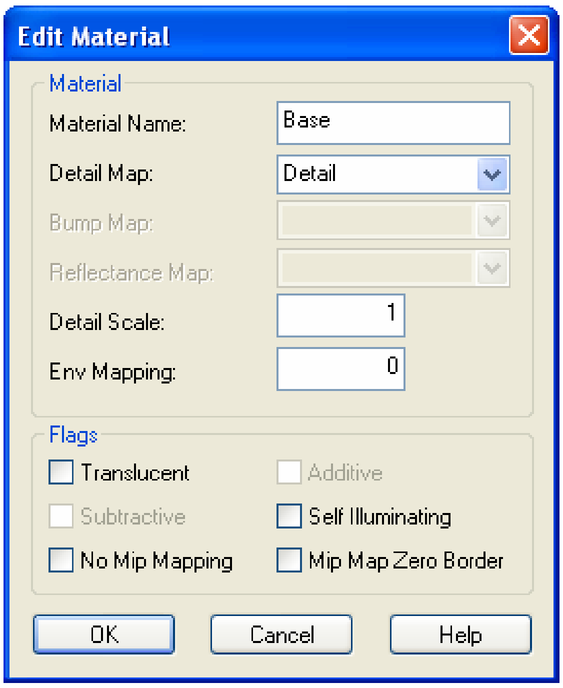
LOS Collision Meshes

Any mesh whose name begins with 'LOSCol' will be used as a line of sight collision mesh. These meshes are used for line of sight collision tests such as checking if a bullet will hit the model. These meshes are normally given a negative detail level from -9 to -16 so that they are not drawn. You can view the LOS collision mesh ingame by giving it a positive detail level.

Like regular collision meshes, LOS meshes should use as few polygons as possible, and **must** be convex.

Materials

Most properties of a material can be edited using the Material Edit dialog box shown below.



Edit Material

Material

Material Name: Base

Detail Map: Detail

Bump Map:

Reflectance Map:

Detail Scale: 1

Env Mapping: 0

Flags

[ ] Translucent
[ ] Additive
[ ] Subtractive
[ ] Self Illuminating
[ ] No Mip Mapping
[ ] Mip Map Zero Border

Material Name

Name of the material. This is used internally by the DTS shape and does not affect the actual texture used.

Detail Map

Name of the Milkshape material to use as a detail map. See Detail Mapping.

Bump Map

Name of the Milkshape material to use as a bump map. *Note that TGE does not yet support bumpmapped DTS shapes.*

Reflectance Map

Name of the Milkshape material to use as a reflectance map. *Not supported*

Detail Scale

Scale of the detail map. See Detail Mapping.

Environment Mapping

Amount of environment mapping to apply. 0 for none. This value is a scaler (range 0-1) which is applied to the alpha channel of the texture to determine the level of environment mapping at each point.

Translucent

Enable transparency

Additive

Enable additive transparency (only valid if translucent flag is checked)

Subtractive

Enable subtractive transparency (only valid if translucent flag is checked)

Self Illuminating

Enable self-illumination (lighting doesn't affect it)

No Mip Mapping

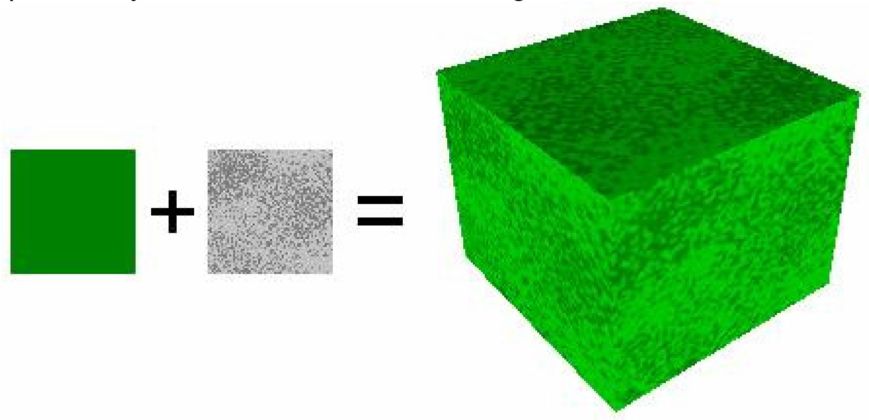
Disable mip-mapping for this material

Mip Map Zero Border

TODO

Detail Mapping

Detail maps allow you to blend two textures together as shown below:



The detail material is scaled by the detail scale setting before being blended with the base material. The easiest method is to make the detail texture the same size as the base texture, and set detail scale to 1. You can find an example of a shape using detail mapping in the examples folder.

Note: The detail material is stored as the Milkshape material index, so if you delete materials, you may need to set the name again.

IFL Materials

An IFL file is a text file that describes which texture to use at each frame for a DTS shape. Animation sequences can be defined that use this information to switch textures automatically while the animation is playing.

IFL materials are defined in Milkshape by specifying a texture with a special name in the texture field of the material. The name of the texture is the same as the IFL text file, except it has _ifl appended.

eg. An IFL file, player.ifl, is shown below:

```
texture1 2
texture2 3
texture3 1
texture4 6
```

Each line describes the texture to use, and the duration (in frames) to display it.

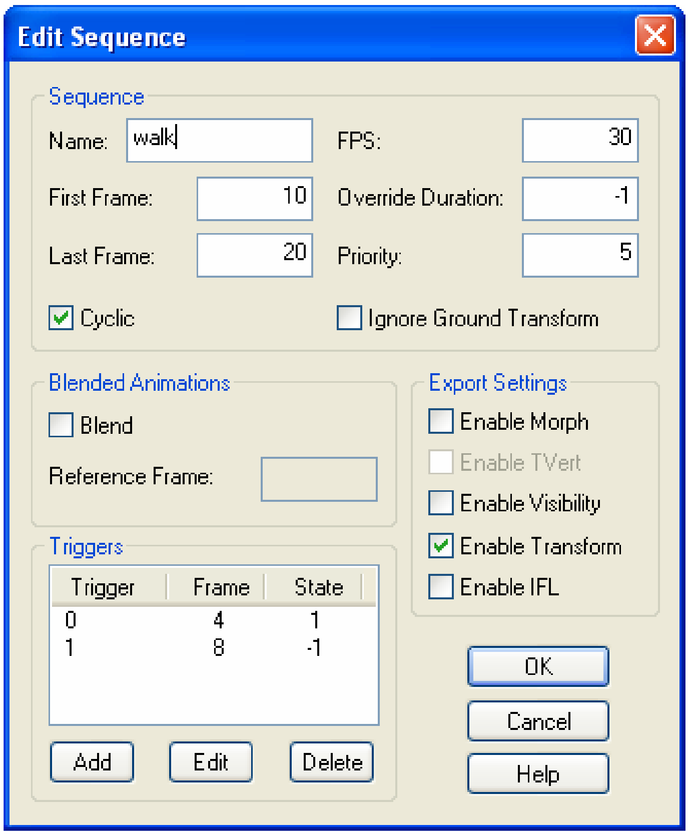
To use the IFL material in Milkshape, a copy of the first image (texture1) is made and renamed to player_ifl. This new texture is used for uv mapping, and tells the exporter the name of the IFL file to use. It is only required during export, and is not actually used by the DTS shape.

A sequence must have the EnableIFL flag set to make use of an IFL material. You can find an example of a shape using an IFL material in the examples folder.

Note: IFL animations are not affected by the frame rate of the sequence in which they are played. The durations specified in the file are assumed to be at a frame rate of 30 fps.

Animation

MilkShape only provides a single animation timeline, but the Torque Engine supports multiple animation sequences, each of which can be named and have different properties. Multiple sequences in MilkShape are animated on the main timeline and are split into separate sequences by the exporter. For this to happen, animation sequences must be declared indicating where each sequence starts and ends on the master timeline. This is done through materials with special names (a '*' at the start of a material name indicates that it is a sequence description). The easiest way to define sequences is using the export dialog box:



Name: walk
FPS: 30
First Frame: 10
Override Duration: -1
Last Frame: 20
Priority: 5
[✓] Cyclic
[ ] Ignore Ground Transform
[ ] Blend
Reference Frame:
[ ] Enable Morph
[ ] Enable TVert
[ ] Enable Visibility
[✓] Enable Transform
[ ] Enable IFL

Trigger	Frame	State
0	4	1
1	8	-1

Name

Name of the sequence.

First Frame

First frame (inclusive) in the sequence. This number should match the frame number in the milkshape animation timeline.

Last Frame

Last frame (inclusive) in the sequence. This number should match the frame number in the milkshape animation timeline.

Cyclic

If turned on, the sequence will loop (e.g. walk and run animations). If turned off, the sequence will play once then stop (e.g. death animations).

FPS

Frames per second for this animation. This does not affect the number of keyframes, only how fast they will be played back.

Priority

Controls what sequence will affect a node when two sequences want to control the same node. The sequence with higher priority will control the node.

Override Duration

If you override the sequence duration, it will change the duration of the sequence when it plays in the game at time scale 1, but it won't otherwise change the animation data (same keyframes will be used, they'll just play at different times). This is useful for altering the speed of the ground transform of an object without scaling the animation. Most of the time, this is not used, and should be set to -1.

Ignore Ground

Don't export a ground transform for this sequence. This should usually be false. See Ground Transforms for more details.

Blend

Makes the sequence a blend animation. See Blended Animations for details.

Blend Reference Frame

The reference frame number for the blend animation. Only valid if the blend flag is set. See Blended Animations for more details.

Triggers

Set of trigger keyframes and states. See Triggers for details.

EnableMorph

This will force the exporter to export all mesh animations as a series of mesh snapshots. This is useful for certain types of animations (e.g. flags), but it will produce large files and does not contain animated nodes. **UNTESTED**

EnableTVert

Enables animated texture coordinates. See Texture Animations for details.

EnableVis

Enables use of the visibility channel. See Visibility Channel for details.

EnableTransform

Enables transform (eg translation and rotation) animation. Normally this setting is enabled.

EnableIFL

Enables IFL animation. See IFL Materials for details.

Ground Transforms

Animation sequences that move the character must export a ground transform. The engine knows that the character has a specific velocity in all directions (this is set in script). When the animations are being played, the engine is aware of what the distance covered is and plays the appropriate animation. If, for instance, the forward velocity of the character increases past the point of a walk animation to the speed of a run, it will transition to the run.

The exporter figures out the ground transform (meters per second over a given distance) by determining how much the bounding box has moved over the course of the animation in the ms3d file. This is done automatically on export.

If you have no ground transform, the animation will not play properly when the character moves. In the Torque Engine with the default character, the forward ground transform is approx=4m/sec.

Note: The bounding box is simply a mesh with the name 'Bounds'. It is normally attached to the Root bone. You can use the main export dialog to create the bounds mesh for you automatically.

Blended Animations

Blend animations allow additive animation on the node structure of the shape. These will not conflict with other threads, and can be played on top of the node animation contained in other threads. Such animations are relative. Blends only read the changes that occur over the course of the animation and not the absolute position of the nodes. This means that if a node is transformed by a blend animation, it includes only the transform information for that node, and it will add that transformation on top of the existing position in the base shape (the DTS).

Bear in mind that a blend can be played as a normal sequence, or it can be played on top of other sequences. When another sequence is playing, it will alter the root position, and the blend will be applied on top of that.

If you try to do a blend sequence where the root position is different than the 'normal' root (in the default root animation), you might expect that the blend will blend it to the new root (the position the character is positioned in during the blend animation). However, it does not work this way. Since nothing would actually be animating, it doesn't move the bones to the new position. What is contained in the blend sequence is only transform offsets from the blend sequence root position.

It is not a good idea to have a different root position in your 'normal' animations and your blends, as they can easily get out of sync.

You can determine the position that the blend animation uses for the animation offset by using the blend reference frame.

The values added from the blend animation are based on the root position in the DTS/DSQ file. This root position does not have to be the beginning of the animation. You can pick any position for the blend animation to reference.

This is useful, because you can have a blend animation that can have a reference position that is the 'root' position. For animation like hip twists and arm movements (as in the 'look' animation), the character can be in a natural default state. In this way, you can have one animation control the character through the base pose to an extreme in either direction while referencing the default 'base' state, which will exist somewhere in the middle of the blend animation.

Texture Animations

This is useful for things where the texture itself must animate. Scrolling computer monitors, waterfalls, and tank treads are just a few of the applications for animated texture coordinates.

Note: Texture animation is not yet supported by this exporter. Non-smooth texture animation can be faked using IFL materials. See IFL Materials for details.

Visibility Channel

A mesh can define a visibility channel (see Meshes). Sequences that have the enableVis flag set can use this set of keyframes to control the transparency of the mesh during the sequence. This is useful for parts of the model that you may only wish to show during certain animations.

Triggers

Triggers are arbitrary markers that can be used to call events on specific frames in a sequence. An example of a triggered event is calling footstep sounds and footprints during walk and run animations.

Triggers can be added to and removed from a sequence by using the Add and Remove buttons in the Edit Sequence dialog. You may define up to 32 triggers per sequence.

TriggerFrame is the frame number on which a trigger event occurs.

TriggerState defines the state of a trigger. There can be up to 32 trigger states each with their respective on (1 to 32) and off (-1 to -32) values. What each of those trigger states means is up to you. You should work with your programmer to define what the trigger states mean and how you should use them.

For example, you could have one trigger for each foot of a character that creates a footprint when the foot is down on the ground. Let's say that a triggerState of 1 is the left foot down and a triggerState of 2 is the right foot down. When the sequence plays the frame during which the left foot touches the ground, you could have a trigger on that frame that has a triggerState of 1 to create a footprint. You would then create another trigger with a triggerState of 2 for the right foot. You don't necessarily need to turn off the footprints (let's assume that the programmer will turn them off when it is necessary), but you could by creating two more triggers with triggerStates -1 and -2.

There is one triggerFrame and triggerState per trigger. Trigger numbering starts at 0. For example, triggerFrame0 and triggerState0 are the first trigger, triggerFrame1 and triggerState1 are the second trigger, etc. Note that when you delete triggers from the list, all of the remaining triggers are renumbered starting from 0. Their frame and state attributes are retained.

Any sequence that makes use of triggers must have the ignoreGround checkbox cleared, or the triggers will not work ingame.

DSQ Export

Exporting animations to a DSQ file allows you to share animations with multiple DTS shapes. DSQ files are loaded at runtime via script. eg.

```
datablock TSShapeConstructor(PlayerDts)
{
   baseShape = "./player.dts";
   sequence0 = "./player_root.dsq root";
   sequence1 = "./player_forward.dsq run";
   sequence2 = "./player_back.dsq back";
}
```

The 'run' sequence can now be played as if it were part of the original DTS shape. A DSQ file may contain more than one animation, and is accessed like this:

```
datablock TSShapeConstructor(PlayerDts)
{
   baseShape = "./player.dts";
   sequence0 = "./player_anim.dsq root";
   sequence1 = "./player_anim.dsq run";
   sequence2 = "./player_anim.dsq back";
}
```

For a DSQ file to be compatible with a DTS shape, all nodes that they have in common must be in **exactly** the same base position and rotation. Only animated nodes need to be exported to the DSQ file.

Note: Milkshape normalises all bone rotations. This can be seen by opening the 'Mr Box' example file, then rotating any bone. You will notice that ***all*** *of the bones change their rotation. This is a milkshape issue, and has nothing to do with the exporter. The result of this normalisation process is that animations produced by milkshape may not be compatible with the default Orc player. DSQ files exported from milkshape should be compatible with DTS files exported from milkshape however.*

Bone Weights

The exporter supports up to 3 bone weights per vertex to be exported. Using bone weighting is the best way to achieve more natual animations, and helps prevent the stretching and distortion of meshes around joints (such as elbows in a humanoid mesh).

Milkshape originally supported only 1 bone weight per vertex, so a plugin is required to access the extra 2 weights. The only plugin currently available is the *Sims2 UniMesh* plugin that is included with the Milkshape 1.7.8 install.

Note that this plugin is not related to or required by this exporter - it merely provides a way to edit the 3 bone weights, which may then be exported to a DTS shape. Please refer to the documentation for the *Sims2 UniMesh* plugin for details on its use.

By default (if no extra weights are set) shapes are still exported with only 1 weight per vertex.

Additional Information

Shape Structure

Many 3D modelling programs support some kind of tree structure that controls the hierarchy of various elements within the shape. Unfortunately, Milkshape does not support a node hierarchy so the exporter attempts to fit the Milkshape model to the following structure:

```
ROOT
|
|-__mainTree
|    |
|    |-LOD Markers
|    |-__meshes
|         |
|         |-skeleton (including Root bone)
|           |
|           |-rigid meshes
|
|-skinned meshes
|-animation sequences
|-bounds mesh
```

- The '__mainTree' and '__meshes' nodes are dummy nodes created automatically by the exporter. They are part of the default NeverExport list, so are not present in the exported shape. If you use your own configuration file, you should add these two nodes to the NeverExport list.
- LOD markers are created automatically for any detail levels that have been defined in the model. They are written into the DTS shape, and have the form:
  - DetailN for regular mesh details (size N)
  - LOSN for line of sight collision mesh details
  - CollisionN for collision mesh details
- Rigid meshes are those that have all their vertices attached to one bone. The 'Root' bone is automatically created by the exporter if it does not already exist in the shape, and is used to catch vertices that are not attached to any bone. These meshes appear in the shape hierarchy below the bone to which they are attached.
- Skinned meshes are those that have their vertices attached to more than one bone. Vertices not attached to a bone are automatically attached to the 'Root' bone.
- The 'bounds' mesh is a box that contains all objects in the Milkshape model. You may define your own bounding box by creating a mesh called

'Bounds'. If no such mesh exists, it will be created automatically by the exporter.

You can check the structure of the exported shape by looking at the dump file.

Note: The bounds mesh and root bone are automatically created by the exporter if they do not already exist in the model. After the export process, they are automatically removed so the model remains unchanged. The bounds mesh and root bone can be retained by selecting 'Create Bounds Mesh' from the export dialog, then exporting or pressing 'Apply'.

Default Configuration

The exporter supports configuration files. If a configuration file is not found, the following default configuration is used:

```
+Error::AllowUnusedMeshes

-Materials::NoMipMap
-Materials::NoMipMapTranslucent
+Materials::ZapBorder

+Param::SequenceExport
-Param::CollapseTransforms

=Params::AnimationDelta 0.0001
=Params::SkinWeightThreshhold 0.001
=Params::SameVertTOL 0.00005
=Params::SameTVertTOL 0.00005
=Params::weightsPerVertex 1

+Dump::NodeCollection
+Dump::ShapeConstruction
+Dump::NodeCulling
+Dump::NodeStates
+Dump::NodeStateDetails
+Dump::ObjectStates
+Dump::ObjectStateDetails
+Dump::ObjectOffsets
+Dump::SequenceDetails
+Dump::ShapeHierarchy

NeverExport
__mainTree
__meshes
```

A '+' sets the setting to true, '-' sets it to false, and '=' is used to set the value of a setting. Nodes in the 'NeverExport' list are not written to the DTS file. This list is mostly used for DSQ export to exclude non-animating nodes. Names in the NeverExport list can include wildcards (*). eg 'leg*' matches both 'leg1' and 'leg2'.

Error::AllowUnusedMeshes
: If true, unused meshes will not cause an exporter error.

Materials::NoMipMap
: Disable mip-mapping on all textures.

Materials::NoMipMapTranslucent
: Disable mip-mapping on translucent textures only.

Materials::ZapBorder
: If set, translucent, non-tiling materials will automatically have the MipMapZeroBorder flag set. See Materials.

Param::SequenceExport
: Allow animation sequences to be exported.

Param::CollapseTransforms
: If set, nodes that do not contain any objects are removed.

Params::AnimationDelta
: Minimum change in position or scale required for a node transform to be recognised as different to the previous transform.

Params::SkinWeightThreshhold
: Minimum bone weighting for a vertex to be affected by that bone. *By default, if only one bone is attached to a vertex it will have weight 1, and bones not attached have weight 0.*

Params::SameVertTOL
: Minimum distance between vertices for them to be considered unique. The DTS file format stores the X,Y,Z position of each vertex in a table, then each triangle in a mesh uses indices into the vert table. Vertices closer together than the minimum distance will store only a single entry in the vert table (the shared position will be the first vertex found). Note that this has no effect on texture mapping, and is not the same as welding two vertices together in Milkshape. Set this parameter to 0 to disable it.

Params::SameTVertTOL
: Minimum distance between texture coordinates for them to be considered unique. The DTS file format stores texture coordinates in a table. Coordinates closer together than this minimum distance will store only a single entry in the tvert table (the shared coordinates will be the first tvert found).

Params::weightsPerVertex
: Maximum number of bone weights per vertex. Since version 2.6.1, the ms2dtsExporter supports up to 3 weights per vertex.

Dump::NodeCollection
: Output details of the node collection process to the dump file.

Dump::ShapeConstruction
: Output details of the shape construction process to the dump file.

Dump::NodeCulling
Output details of which nodes have been culled to the dump file.
Dump::NodeStates
Output node states to the dump file.
Dump::NodeStateDetails
Output node state information to the dump file.
Dump::ObjectStates
Output object states to the dump file.
Dump::ObjectStateDetails
Output object state information to the dump file.
Dump::ObjectOffsets
Output object offset information to the dump file.
Dump::SequenceDetails
Output sequence details to the dump file.
Dump::ShapeHierarchy
Output the shape hierarchy to the dump file.

Dump Files

When the shape is exported, a file called dump.dmp may be created in the same directory as the dts file. This text file contains details of the export process, as well as the final structure of the exported shape. It may be useful to track down problems with the shape or the export process.

Comment Strings

Versions of milkshape before 1.7.4 did not provide any means of storing additional user information in the model. The original ms2dtsExporter stored a small number of properties in the name of the mesh or material. eg. seq:walk=1-4,cyclic. Previous versions of ms2dtsExporterPlus continued this practice, although because there were far more properties to store, they were packed into a binary form, resulting in names that looked like this: *Walk=@!]!!!?&!b!!.

All user properties are now stored in the comment string of the model, mesh and material objects. There are 3 types of properties, floating point, integer and boolean (true/false). Each group stores the number of properties in that group, then a list of name=value pairs, each on a new line.

eg. The comment string of an animation sequence may look like this:

```
1
frameRate=30
4
endFrame=4
startFrame=1
numTriggers=1
triggerFrame0=1
triggerState0=-1
1
cyclic=1
```

Comment strings can be edited manually, but very little validation is performed when they are read by the exporter, so manual editing should be avoided. The export dialog boxes provide a much better way to edit object properties.

Credits

This exporter would not be where it is now without the following people who have been invaluable in it's development. Thanks for all of your bug reports, testing and enthusiastic support!

- David Korsgaard
- David 'Rex' Whalen
- Edward Maurina
- Matt Fairfax
- Melvin Ewing

Change Log

Version 2.7.3 - 30/01/07

- Fixed seams appearing between groups in exported model (thanks Jon Orantes for the bug report and test case)
- Updated documentation

Version 2.7.2 - 04/12/06

- Fixed a texture coordinate export bug introduced in the previous version.
- Updated documentation

Version 2.7.1 - 29/11/06

- Fixed unwelding problem (seams appearing) in exported model and in milkshape after exporting (thanks Rex for helping track down the cause!).
- Updated documentation

Version 2.7.0 - 09/11/06

- Fixed error when exporting after adding sequences or the bounds mesh to the shape via the exporter dialog. If you have seen seemingly random crashes with the exporter, this may have been the problem. (thanks Gordon Marsh for the bug report)
- Added a progress bar
- Updated documentation

Version 2.6.2 - 07/11/06

- Fixed error in shape hierarchy: rigid meshes are now added as children of the bone to which they are attached. (thanks Simon Duggan)
- Updated documentation

Version 2.6.1 - 18/06/06

- Updated documentation

Version 2.6.0 - 10/06/06

- Updated to milkshape SDK 1.7.7
- Added support for up to 3 bone weights per vertex

Version 2.5.0 - 24/12/05

- Added option to split DSQ export into multiple files
- Added option to generate .cs file for DSQ export
- Updated documentation

Version 2.4.0 - 17/12/05

- Added support back for detail maps
- Added option to copy textures to export directory
- Fixed bug that prevented creation of collision detail levels (thanks Simon Duggan)
- Updated documentation

Version 2.3.0 - 03/10/05

- Fixed bug that prevented Z billboards from being exported correctly
- Added billboards image to documentation
- Updated documentation

Version 2.2.0 - 02/05/05

- Fixed bug where NeverEnvMap was not being cleared for materials using environment mapping
- Made main export dialog box modal
- Updated documentation

Version 2.1.0 - 01/05/05

- Fixed bug concerning mesh names ending in -1

Version 2.0.0 - 17/04/05

- Changed to new milkshape SDK (1.7.4)
- Changed naming convention to store properties in comment string
- Made visibility keyframes persistant
- Removed sequences are now removed from the model
- Added 'Output dump file' option to export dialog
- Added 'Create Bounds Mesh' button to export dialog
- Various small code fixes
- Added online help
- Updated documentation

Version 1.9.0 - 23/01/05

- Fixed LOSCol mesh name bug (thanks Ed Maurina)
- Updated documentation

Version 1.8.0 - 16/11/04

- Removed auxilary map (bump, detail, reflectance) support.
- Added fix from Matt Fairfax for normal calculation
- Updated documentation

Version 1.7.0 - 27/10/04

- General code tidy up
- Updated documentation

Version 1.6.0 - 23/09/04

- Added support for custom bounding box
- Fixed broken IFL support
- Fixed problem with unattached vertices being attached to the wrong bone
- Various small code fixes
- Updated documentation

Version 1.5.0 - 31/08/04

- Fixed 'fps' animation setting
- Added initial DSQ support

Version 1.4.0 - 24/07/04

- Added player example to doc/examples folder
- Added base skeleton to doc/examples folder

Version 1.3.0 - 21/07/04

- Fixed animation triggers not being exported
- Added support for detail mapped materials
- Added detail map example to doc/examples folder
- Updated documentation

Version 1.2.0 - 30/06/04

- Added support for IFL materials
- Added IFL example to doc/examples folder
- Fixed unlinked mesh error when using custom config file
- Updated documentation

Version 1.1.0 - 22/06/04

- Fixed scale setting not being applied by main dialog
- Fixed dump file not working with custom config file
- Added support for independent position/rotation keyframes
- Updated documentation

Version 1.0.0 - 19/06/04

- First public release