TABLE OF CONTENTS

This Exporter was Created by
Mike Jarosch and Matt Summers
of Dark Industries

Much of the basic DTS feature documentation in this file was assembled in part from other tutorials and docs created by various members of the Garage Games Community, most notably Joe Maruschak, Logan Foster, Tim Gift and Clark Fagot. Also, special thanks to Pavol from Caligari, Stan Melax and the beta testers like Craig Fortune who weren't afraid our software would erase their hard drives.

Getting Started 2
- General Information 2
- Installation 2
- Using the Scene Editor 3
- Using Object notes 3
- Exporting to .x to lock object scale 4
- Skeletal vs Object Animation 4
- Positioning the axes of objects 4
- Manipluating axes in gameSpace 5
- Parenting Objects 5
- Parenting with Bones 5
- Parenting with Groups 5

Configuring DTS Material Properties 6
- Object Triangulation 6
- Materials For Torque 6
- Default Material Setup 6
- Texture Priority 6
- DTS Material Properties 7
- Detail Mapping 7
- Mipmap Control 7

Using the features of the DTS Format 8
- The DTS Hierarchy 8
- The Bounds 9
- Detail Levels 9
- AutoDetail Levels 9
- Collision Objects 10
- Billboards 10
- Billboard Detail Levels 10
- AutoBillboard Detail Levels 10
- Mesh Sorting 11

DTS Animation 11
- Types of Animation Supported 11
- Animating for Torque 12
- Sequence Markers 12
- Sequence Marker Attributes 12
- Enable/Force Animations 13
- Visibility Animation 13
- IFL Animation 13
- Ground Transform 13
- Threads 14
- Blend Animations 14
- Blend Reference Time 14
- Configuration Files (.cfg) 15
- Player Sequences 16
- Vehicles Sequences 16
- Weapon Sequences 16
- Triggers 17

Named Nodes 18
- Using Named Nodes 18
- Player Nodes 18
- Vehicle Nodes 18
- Weapon Nodes 19

Using .CS files 19
- The .CS Script file 19

Using the show tool 20
- The ShowTool 20
- Loading a Shape 20
- Detail Levels 20
- Viewing Animations 21
- Transitions 21

General Information

This document assumes a basic level of familiarity with gameSpace or trueSpace and is simply a general reference for creating and exporting shapes using the gameSpace/ truespace DTS exporter. It is not a quick start guide for how to use gameSpace nor is it a detailed tutorial on how to build and animate shapes in gameSpace for the Torque Game Engine.

There are two file types that you can export: shapes (DTS) and sequences (DSQ). A DTS file contains meshes, detail levels, the bounding box, all necessary nodes for a particular shape and optionally animation data can also be included in the DTS file.

DSQ files only contain animation data and require a matching DTS file that contains all of the necessary nodes for the shape. There can be multiple DSQ files for any one DTS file. Also, DSQ files can be used on multiple DTS files provided that the animation nodes in the DTS files match exactly.

The exporter generates a log file named dump.dmp in the directory you export your object to. If your export fails, check the dump file first to see where it is crashing in the export process. Most of the time it is something simple, like spelling the word “bounds” wrong.

If a particular model is continually crashing on export, corrupted meshes, bad texture vertices, or double faces in the model might be the cause. Check the dump file, and if it stops on a particular mesh, do a test by adding “_” to beginning of the object name, then try to export again. (objects with _ at the beginning are ignored by the exporter)

After a model is successfully exported, check your shapes in the Torque ShowTool first before bringing them into your game. If it works in the ShowTool but not in the game, the file exported properly and the problem exists elsewhere. What is required for a shape to work “in the game” is not the same as what is required for a shape to export.

Installation

Run the installer for the gameSpace/trueSpace DTS exporter and then run gameSpace or trueSpace and click on the install extension icon (the 3d plugin icon). From the dialog select the DTS plugin from the directory you installed it to and click the open button. You should now see the DTS icon in your tool bar.

Left clicking the exporter icon will export all of the objects in your scene as a DTS file and any sequences setup in the file will be stored inside the DTS file. Right clicking the exporter icon will export all of the sequences in the scene as one .DSQ file containing all of the sequences setup in your scene.

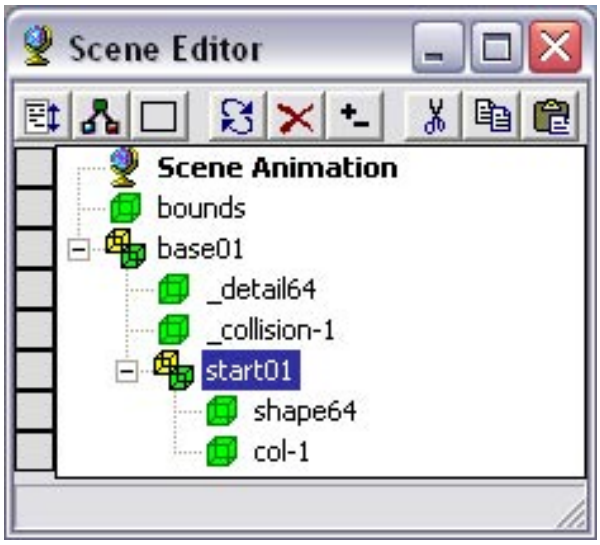
Using the Scene Editor

Using this exporter requires that you learn how to create a DTS hierarchy using object grouping in gameSpace or trueSpace. The easiest way to do this is by using the Scene Editor. By default the Scene Editor icon is located in the upper left corner of the screen near the animation playback controls.

With the Scene Editor open on the left pane you will see a listing of all of the objects in your scene. Dragging any object in the Scene Editor onto another object in the Scene Editor will create a group that contains both objects. You can also drag other objects from the scene root onto the group to add more objects to the group or drag one group onto any other group. Also, in order to move a grouped object to a new group you must first drag the object to the scene root and then to your target group.

You need at least two objects in order to create a group in the Scene Editor, but sometimes your shape setup may require you create a group that contains only one object. To do this create a mesh object such as a simple box and name it "_dummy". Objects named "_dummy" are ignored by the exporter. Use this dummy object to create any grouping you need in order to export your DTS shape properly.

If you don not see your changes in the Scene Editor right away, you can right click on the scene view pane and select the option "refresh" to update the scene editor view port.



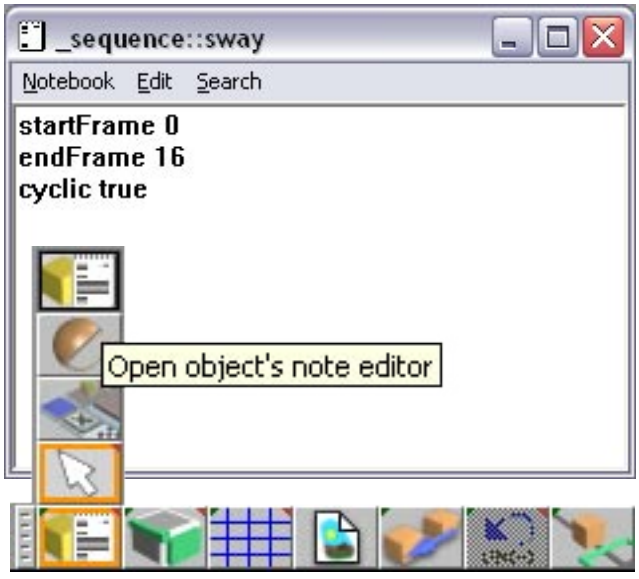
Using Object notes

There are many custom object properties that must be configured in order to export your DTS shape and DSQ animation files. These custom properties are stored in the object notes of the various objects and helper objects in your scene.

To open the object notes editor click and hold down on the Object tool icon. (the arrow icon) Select the top icon from the flyout menu to open the object note editor. If you also have the Scene Editor open clicking on any object in the scene editor will display the object notes for the current selected object.

There are two types of object properties that can appear in the object notes. A property name followed by a space and a number. For example: "startFrame 0" Or a property name followed by a boolean state (true or false). For example: "twoSided true".

Only one property name and value pair should appear on each line in the object notes. You can also create comments for yourself in the object notes by adding "//" to the beginning of the line. Everything after // is ignored to the end of the line. For example. "// Note to self.....".



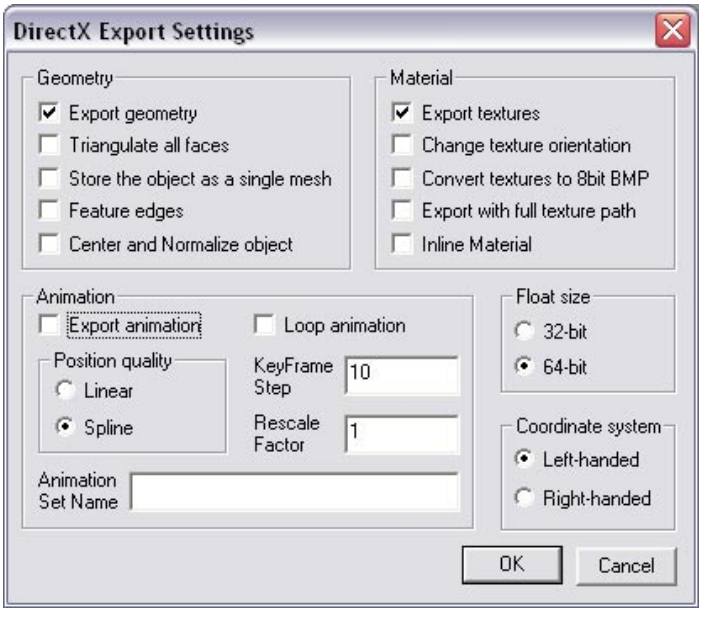
Exporting to .X to lock object scale

Using the object manipulation tools in gameSpace or trueSpace doesn't actually change the physical size of an object, instead these tools apply a scale factor to the X, Y, and Z axes of all objects.

Since there is apparently no way in gameSpace or trueSpace to freeze the scale transforms of an object before export you must export your objects (including the bonds) from gameSpace to the .X format and then load the objects back into gameSpace. The .X exporter recalculates the vertex positions and recalculates the scale transforms applied to exported objects. Doing this will allow your objects to export at the proper size.

To export your object to .X format. Select an object in your scene or group all of the objects in your scene and select the object group then click on the file menu and choose: "save as object" and choose the .X format from the file type selector. Click the settings button in the save object window and adjust the .X exporter settings to fit your needs. The settings below are recommended as they seem to work well.

After exporting your object load the exported .X object back into gameSpace and delete your original objects from your scene. The imported file is now a fixed size and no longer contains scale transform information.



Skeletal vs object animation

There are two primary ways to create animation in the DTS format.

Skeletal animation is where bones (nodes) deform a mesh using vertex weights. (The orc character from the Torque demo is a good example of skeletal animation)

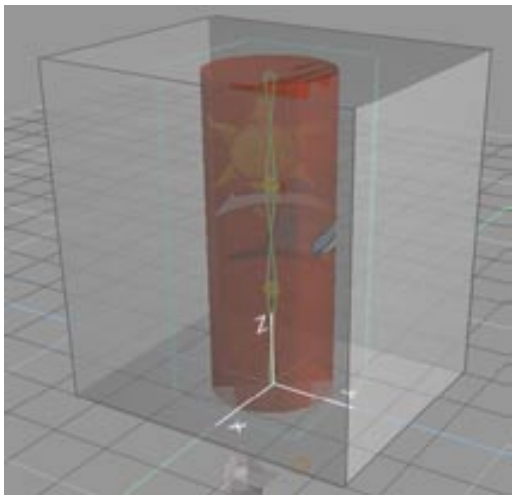
Object animation is where the individual objects that make up your DTS shape can each have independent scale, rotation, translation and visibility animation. (The crossbow weapon from the Torque demo is a good example of object animation)

Positioning the axes of objects

When animating objects often times you will need to move the axes of your object(s) from its center to the point in your object that it should rotate from.

Your bounds object should have its axes centered and at the bottom (usually 0,0,0) as this is the point where the object will come to rest on the terrain.

When using skeletal animation the axes of your highest detail mesh should have its axes at the same z value as your bounds, other wise you can get node offsets where the mesh is lower or higher than the nodes that should be deforming it.



Axes of skined mesh and bounds at same z axes

Manipulating axes in gameSpace

The axes tool will display the axes for the currently selected object. Once you use this tool, the axes become visible and are themselves treated as an object; they are available for rotation or scaling, without affecting the object itself. The axes may also be moved as desired.

The object cannot be manipulated while this tool is active. When activated, you are actually inside the hierarchical structure of the object. In order to return to editing the object, you must move up the hierarchy of the object, returning to object edit mode. You may also use the up arrow on your keyboard to accomplish this.

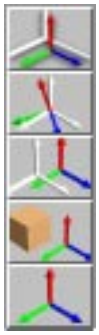
There may be times when you will have a multitude of axes visible for a number of objects. They can visibly clutter your scene. If this is the case, provided you are finished with them, you can activate the Axes Tool, which will select the axes for you. Once selected you can use the Axes Tool again to cause the axes to disappear or use the Delete Tool to delete the axes. This will not affect the object, it merely removes the axes from sight. The axes can be brought back at any time by using the Axes Tool again.

The normalize rotation tool is used to orient an object or an object's axes to the orientation of the World Axis. If the object you are working with has been rotated and you wish to return the object to a normalized state, select the object and then use the Normalize Rotation tool to return it to the World Axis orientation.

You may also normalize the rotation on an object's axis only (as opposed to the object itself). Selecting the Normalize Rotation tool will normalize the rotation of the axes, without affecting the object.

The normalize location tool will move either the object itself or its axes to world origin (0,0,0). It will not affect the object's rotation or that of its axes.

The move axes to center of object tool will move the axes of an object to the center of the object.



Parenting Objects

In certain objects, vehicles and player objects it becomes necessary to parent objects or nodes to other objects or nodes. In gameSpace there is no way to directly parent an object to another object so there are several work arounds you can employ to overcome this.

When you export a object to DTS two things are created for each mesh object, a node that holds position and transform (at the axes) and the mesh object attached to it. When you cull out a node using a .cfg the mesh object attached to it is then attached to the nodes parent.

There are only two types of objects in gameSpace that can be considered parents in the scene hierarchy, Bones and Groups.

Parenting with Bones

When an object is attached to a bone, that bone is considered the objects parent. When exporting non-skin meshes attached to bones to create a parent child structure you can add the nodes of your objects to the "never export" section of your .cfg file and then your mesh objects will be attached to their respective parent bones creating a parent child grouping of the objects.

Parenting with Groups

When an object is in group in the hierarchy, that group is considered the objects parent. Groups are normally culled out by the exporter because they are not mesh objects, but they do contain translation information and you can see there location by selecting the group and clicking on the axes icon to show its axes.

Just like you can do with rigid objects attached to bones you can add the meshes node to the "never export" list in the .cfg file and then add the groups name to the "always export" list in your .cfg file. You mesh object will now be attached to its parent (the group) and any additional nested groups containing objects that are added to the appropriate places if the .cfg file would be part of the parent child hierarchy of your DTS shape as well.

Object Triangulation

Triangulation is handled automatically by the exporter. unless you prefer implicit control of the triangles in your model it is fine to leave the objects in the scene as quads as the exporter uses functions in the TSX SDK to return a triangulated version of the objects in the scene at export and the objects in you scene are left as they are.

Materials For Torque

All materials must use image textures in the Color channel. Procedural textures and material colors are not supported in Torque. Torque supports JPG, TGA, and PNG texture formats.

Image file dimensions must be powers of two (e.g. 256x256, 64x64, etc). The maximum size supported is 512x512. Textures do not have to be square (e.g. 64x128). Multiple textures on a single mesh is supported.

Your shape should be properly UV mapped and textured with image based textures. Procedural textures, decals, and solid colors are not supported in the DTS format.

Default Material Setup

Open the material editor and if its not already, expand it by clicking on the blue triangle on the right side of the material editor window.

Right click on the color shader and load the texture map shader from the color shader selector. Left click on the texture shader to load the texture for your object into the texture map shader.

Click on the blue sphere at the top of the color shader and select plain transparency from the menu.

Right click on the reflectance shader and select an environment map shader from the reflectance shader selector.

In your environment map shader use these settings
Set the luminance slider to 0
Set the diffusion slider to max
Set the shininess slider to 0
Set the specular slider to 0

Note: This exporter supports the use multiple textures on your DTS object. On aper face or per distinct object basis. In game performance issues can arise depending on the size and number of textures on your object.

This will create a neutral texture only setup for the DTS export. Leaving the color shader as a solid color will create a DTS without materials.

Texture Priority

When working with materials it is important to note that the DTS format does not store the file extension of textures in the DTS file, only texture name. Because of this the presence of a JPG file and a PNG file image file with the same name (like “surface.png” and “surface.jpg”) will cause the engine to choose the texture file type by texture priority.

Since JPG has higher priority than PNG it will always be used first. This can cause problems if you are attempting to use transparent textures because the JPG is used first and it contains no transparency information.

DTS Material Properties

Self Illumination
Set the luminance slider to max, Self Illuminate is turned on

Environment Mapping
Set the reflection to max ; reflection is turned on
Reflectivity is based off of the alpha channel of texture loaded into the environment map shader.

Translucency/Transparency
To make an object translucent, the texture on the object must have an alpha channel. The amount of transparency is controlled by the images alpha channel. Applying an image file with an alpha channel to the Color attribute of a material will automatically apply the images alpha channel to the same materials Transparency attribute.

Material Transparency
Set the transparency slider to 0.5 to turn on material transparency. Transparency is based off of the textures map alpha channel

Subtractive Transparency
Set the transparency slider to 0.5 to turn on transparency.
Set specular > 0 but < 1 ; Then Subtractive is on
Transparency is based off of the texture map alpha channel

Additive Transparency
Set the transparency slider to 0.5 to turn on transparency.
Set specular = 1(max) Then Additive is turned on
Transparency is based off of the texture map alpha channel

wrapU false/true
By adding the property “wrapU true” or “wrapU false” to the object notes of any mesh shape in your scene you can enable or disable texture tiling in the U direction.

wrapV false/true
By adding the property “wrapV true” or “wrapV false” to the object notes of any mesh shape in your scene you can enable or disable texture tiling in the U direction.

twoSided true/false
Double-sided materials are supported by adding “twoSided true” to the object notes of your target object. The twoSided attribute is a boolean value that determines whether or not the shape will export with double-sided materials.

Detail Mapping

Detail maps allow you to blend two textures together The detail material is scaled by the detail scale setting before being blended with the base material. (not supported yet)

Mipmap Control

By default, all textures are mipmapped but you can add properties to the .cfg file to change how mipmaps are generated.

+Materials::NoMipMap
Use this if you don’t want to mipmap any textures.

+Materials::NoMipMapTranslucent
If you want textures to mipmap, but not translucent textures.

Translucent textures that do not tile are now, by default, considered “zeroborder” textures -- that is, we assume that there is a blackborder around the texture and we enforce this in the mipmap. This is normally what one wants, because otherwise streaks occur when off-tile parts of the face are drawn.

-Materials::ZapBorder
If you don’t want to do this to your non-tiling translucent textures, then you can set: (as mentioned it is true by default).

Envrionment Mapping

Transparency

The DTS Hierarchy

All Torque scenes must contain the following objects at the scene root level in order to export properly: a bounding box named bounds and the DTS hierarchy.

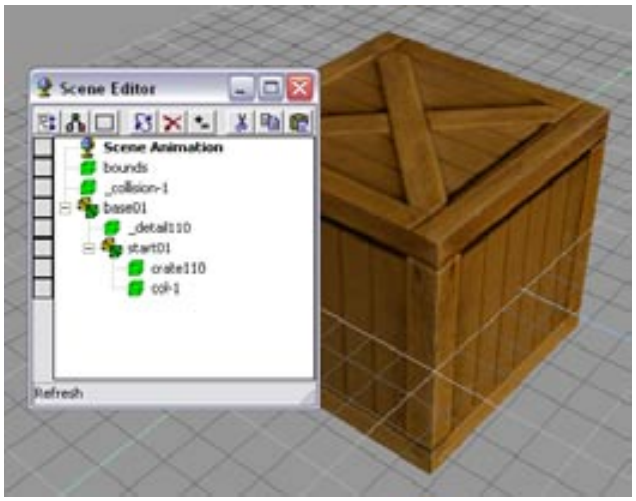
The bounding box defines the shapes orientation and position in the world. Without the bounding box, the scene will not export.

The DTS hierarchy is a group in the scene root (base01) that contains at least one detail level marker (_detail#) and at least one additional group with children (start01) that has geometry somewhere in its sub-hierarchy and/ or a IK group that deforms a mesh.

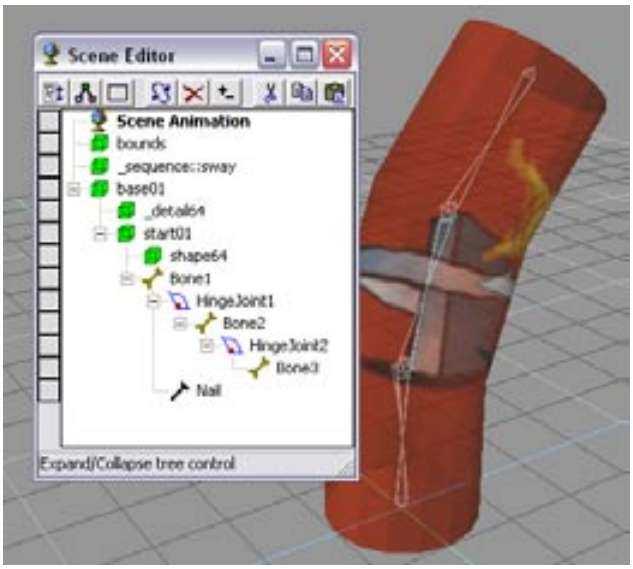
A detail level marker indicates to the exporter what detail level mesh should be drawn at a given distance. The number following the name of an object corresponds to the pixel size in the game engine at which the shape will be drawn.

The shape branch of the DTS hierarchy corresponds to the actual DTS shape that will be exported. The whole subtree can be under one branch or there can be multiple branches. (shape01, shape02, etc.)

If you have one shape at the root level that you want to export (e.g. a static shape), an example scene hierarchy should look like this:



The shape branch of the DTS hierarchy will look different if you are using a skeleton to deform a mesh (e.g. a single mesh character). In this case the IK group becomes the shape branch along the mesh and any detail levels should be at the scene root level. Here is what a simple hierarchy with a skeleton might look like:



To create a proper DTS hierarchy shapes must be properly named and in the right place in the scene hierarchy. Here are the locations where shapes and helper objects should go in the DTS hierarchy.

The bounds, sequence helpers and secondary detail meshes should all be at the scene root.

The first group in the DTS hierarchy (base01) should contain detail level markers (_detail64), collision markers (_collision-1), loscollision markers (_los-9) and the shape branch group. (start01)

The shape branch group (start01) should contain your highest detail meshes, IK groups, collision meshes (col-1), loscollision meshes (loscol-9) and named nodes. (_mount0)

The Bounds

Each scene must contain a bounding box. If you fail to create a bounding box within your scene, the scene will not export. You must also name the bounding box bounds.

When you create the bounding box, you are creating a mini world that defines how the object will orient itself in the game. It is important to note that the orientation of the bounding box defines the orientation of the object. Positive Y is forward, and positive Z is up.

The bounding box should be a box that completely encloses the shape at all points in the animation. When you export your scene, the pivot point of the bounding box will become the pivot point of your shape in the exported scene.

Ground transform is based on the animation of the bounding box. So if you want to export a running person, you would place the bounding box around the person at time 0, with the origin of the box at the character's feet. As the person runs, you would animate the bounding box to keep pace with the person.

Detail Levels

All shapes that are to be rendered in the Torque engine must also have a detail number at the end of their name. This number corresponds to the pixel height at which the shape will draw.

For example: Shape128 – this naming tells the exporter that this object named "shape" is drawn when the object is greater than 128 pixels high on screen.

Each detail number must have a corresponding detail marker that shares the same number. The base name for all the shapes must be the same, with the only difference being the detail number. Detail markers are named "_detail#" and there must be one marker for each unique detail number in your scene.

Only the highest detail meshes should be linked to the shape subtree. The other detail meshes should be left at the scene root level. At export time, the "loose" meshes are collected and the transforms of the detail meshes are discarded and the transform of the corresponding shape in the subtree is used.

For example: if you have three detail meshes for a particular shape named shape128 , shape 64, and shape2, you must also have three detail markers named _detail128 , _detail64, and _detail2. Shape128 should be a child of start01, while shape64 and shape2 are at the scene root level. All of the detail markers should be added to the group base01.

When the shapes size on screen is 128 pixels or greater, the shape with the detail number of 128 will be drawn. When the size is between 64 and 128, the shape with the detail number of 64 will be drawn. Likewise, when the shapes size is between 2 and 64, the shape with the detail number of 2 will be drawn. When the size is less than 2, nothing is drawn.

AutoDetail Levels

AutoLod works for rigid object and skinned objects. Create your detail markers like you normally would. In the object notes for the highest detail object add

numAutoDetails #
autoDetailSize# size
autoDetailPercent# 0-1

You need to have an entry for every detail level that you don't create by hand.

If you have a shape Sphere128 and billboard Sphere2 with the following detail markers 128,64,32,2 you would add the following to the object notes of your highest detail

numAutoDetails 2
autoDetailSize0 64
autoDetailPercent0 0.8
autoDetailSize1 32
autoDetailPercent1 0.6

Collision Objects

There is no correct way to make collision objects for the Torque engine. It all depends on how your programmer wants to implement collision detection. Several games have used this engine and they all used different collision schemes. Different collision schemes can be different for different shapes as well. Some shapes use simple sphere collision that is derived from the bounding box, some have custom-built collision shapes. Vehicles tend to have custom collision shapes.

Custom collision shapes can be created by assigning a negative detail number to the shape and creating a corresponding collision marker. Shapes with negative numbers will export but not draw.

The Torque Game Engine presently uses detail markers named _collision-# with the mesh shapes named col-#. The shapes must be convex hulls (no concave surfaces). Collision markers must be children of base01 (same level as detail markers). Collision meshes should be in the shape subtree. (start01)

Here are the naming conventions:

Collision-1 to **Collision-9**
These are collision markers. (linked to base01)

Col-1 to **Col-9**
These are the actual collision meshes. (linked to start01)

LOS-9 to **LOS-15**
Markers for line of sight collision (linked to base01)

LOScol-9 to **LOScol-15**
Collision geometry for los collision. (linked to start01)

Keep the collision meshes as low in polygon count as possible, because collision can be processor intensive.

Vehicles are limited to ONE collision mesh for the collision shape Also, pay special attention to the way the collision geometry is shaped to minimize collision with small bumps/hills/slopes etc. Changing this shape will dramatically effect whether or not the vehicle is prone to getting stuck while driving over bumps.

Billboards

Billboard are objects (usually 2d) that always face the player. Adding a billboard or billboardz flag to an object name will cause the exported object to be exported as a billboard and always face the player.

bb:: Tells the exporter an object always faces the player.

bbz:: Tells the exporter that an objectson always faces the player on the Z axes.

Parts of a shape can be billboard objects (i.e., they always face camera). 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.

Billboard Detail Levels

The highest detail level of a shape could be a complicated 3d shape, whereas the lowest detail could just be a billboard. If the highest detail level of an object is called “Obj256” say, then”BB::Obj64” would be a lower detail level of that shape. Similarly, if the highest detail level is “BB::Obj256” then”Obj64” would be a lower detail level version of this.

AutoBillboard Detail Levels

Auto billboard generation is configured using several properties that can be added to the object notes of your Highest detail level.

bb::equator_steps #
bb::polar_steps #
bb::polar_angle #
bb::dl # (size)
bb::dim #
bb::include_poles true/false

Mesh Sorting

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 our 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.

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, begin its name with "SORT: :". Other detail levels of this object do not have to be sort objects, so "SORT::Head128" and "Head64" would be considered detail levels of "Head", the first being a sort object but the second not. You can also give the exporter some hints on how to create the sort objects.

You supply these hints by adding properties to the object notes of your sorted objects. The properties are:

max_depth #
num_big_faces #
write_z true/false

(Default values for these are 2, 4, and false respectively.)

z_layer_up true/false
z_layer_down true/false

Used to sort objects with "leaves" that are layered from top to bottom either facing slightly up or down. Will usually be used with MAX_DEPTH.

Types of Animation Supported

Transform animation:
Allows the export of object animation (position and rotation) and node animation. (skeletal animation)

Morph animation:
This will force the exporter to export all mesh animation as a series of mesh snapshots. (vertex animation) This is useful for certain types of animations but produces huge files and doesn't contain animated nodes (bones). You should never use morph animation unless you are absolutely sure you need it.

Visibility animation:
Allows animation of object visibility. Objects in the DTS file can disappear and reappear or blend from solid to transparent and back.

Scale Animation:
This is the animation of object scale. Don't use this unless you need it, as it adds additional strain on the animation system.

Texture animation:
Enables animation of Texture Coordinates. 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.

IFL animation:
Allows you to use IFLs, or Image File Lists. These are a sequence of image files that are loaded frame by frame to create animated textures.

Animating for Torque

There are two methods for exporting animation into the game. One method is to animate all of the sequences in one file and export it as a DTS file. The other method is to separate each sequence into its own scene file, export them as DSQ files, and then merge them together at run-time with a .CS file.

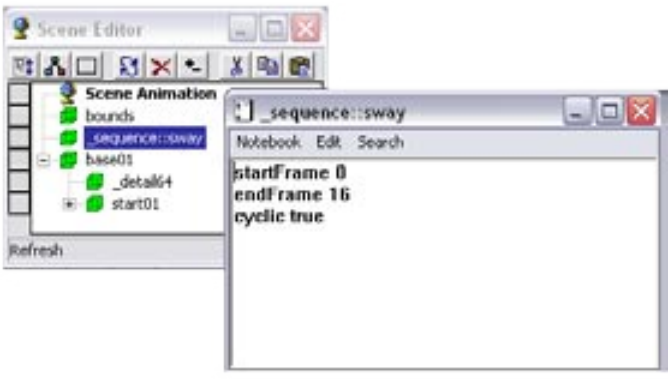
Having the base mesh and skeleton in one scene file and having each sequence contained in its own scene file is the preferred way to do this because it gives more control of the nodes being exported and allows more control of the character.

For simple shapes with only a few animations, putting all of the sequences in the main scene file will work fine. For characters, it is recommended that you save each animation sequence in its own file and export the sequences as DSQs.

Sequence Markers

Sequence markers tell the exporter how to export animation data over a given range of frames. They are required if animation is to be exported. The sequence marker contains a variety of information such as the name of the animation sequence, start and end frames, cycling and blend information, and frame rate.

Sequence Markers are actually just mesh objects with custom properties and are placed at the scene root level. The naming convention for sequence nodes is _sequence: :<name>. Sequence nodes must always be prefixed with _sequence::, otherwise they will not be exported.



To create a Sequence Marker, create any mesh object (like a box) and rename the object "_sequence::<name>". Select the object and open the object notes to add your custom node attributes.

Sequence Marker Attributes

startFrame # (default is 0)
The first frame of the sequence.

endFrame # (default is 0)
The last frame of the sequence.

cyclic true/false (default is true)
If turned on, the sequence will loop. If turned off, the sequence will play once then stop.

blend true/false (default is false)
Makes the sequence a blend animation.

blendReferenceFrame # (default is 0)
The reference frame number for the blend animation.

overrideDuration # (default is -1)
Changing this attribute alters the speed and duration of the ground transform of the shape. The same keyframes will be used, but they will be played at different times.

numFrames # (default is 0)
Number of keyframes for the sequence.

frameRate # (default is 30)
The frame rate for the sequence.

groundNumFrames # (default is 0)
Number of keyframes for the ground transform.

groundFrameRate # (default is 10)
Frame rate for the ground transform.

priority # (default is 5)
Controls what sequence will control a node when two sequences want to control the same node.

ignoreGround true/false (default is false)
Should ground transform be exported for this sequence.

Enable/Force Animations

enableMorph true/false (default is false)
This will force the exporter to export all mesh animations as a series of mesh snapshots. This is useful for certain types of animations, but it will produce large files and does not contain animated nodes.

enableTVert true/false (default is false)
Enables or disables animation of texture coordinates.

enableTransform true/false (default is true)
Enables or disables export of transform animation.

enableVis true/false (default is true)
Enables or disables visibility animation.

enableScale true/false (default is true)
Enables or disables object scale animation.

enableUniformScale true/false (default is true)
Enables or disables export of uniform scaling.

enableAlignedScale true/false (default is true)
Enables or disables export of aligned scaling.

enableArbitraryScale true/false (default is true)
Enables or disables export of arbitraty scaling.

enableIFL true/false (default is true)
Enables or disables export of IFL animation. These are text files that list sequences of images files.

forceMorph true/false (default is false)
forceTVert true/false (default is false)
forceVis true/false (default is false)
forceTransform true/false (default is false)
forceScale true/false (default is false)

Visibility Animation

Visibility animation is created by adding visibility keyframe states to an objects notes or by keyframing object visibility. Visibility animation is supported by adding a custom attribute named visibility to the object notes. The visibility attribute should be a float value with a minimum value of 0 and a maximum value of 1. The syntax for using visibility is "visibility(keyframe) value"

Visibility animation example:
visibility0 0.2
visibility1 0.3
visibility2 0.4
visibility3 0.5
visibility4 0.6
visibility5 0.75

IFL Animation

An IFL is text file where each line describes the texture to use, and the duration (in frames) to display it.

Example IFL file contents:
texture001 2
texture002 3
texture003 1

To use IFL: copy the first image in the sequence and rename it to match the name of your .ifl file (example: explode_ifl.jpg) apply this texture to your object. The actual .ifl file must exist in the same directory as you scene file to export properly.

Ground Transform

Animation sequences that move the character must have 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 (meter per sec over x distance) by determining how much the bounding box has moved over the course of the animation. This is done automatically on export. You can allow the exporter to set the keys (frameRate), or you can set it to sample the distance covered explicitly by telling the sequence to use N frames and sample at the beginning and end of the animation (groundFrameRate). Use N frames is on by default and should be sufficient for most applications.

If you have no ground transform, the animation will not play when the character moves. The default character's forward ground transform is approximately 4 m/sec.

Threads

To fully understand the animation system in the engine, one must realize that several animations (called threads in the engine) can be played concurrently, at different speeds in both directions, and can control different parts of the hierarchy. If two threads try to control the same node, the sequence priority will determine which thread controls a particular node.

In practice, the best way to go about doing things is to export different types of animation to control different parts of the character, and then have them hooked up to different controls. In the game, you can look around while running. Instead of having a run-look-left, run -look-middle, or run-look-right, there are individual run and look animations that are being played at the same time. In this way, you can get a great deal of flexibility out of very few animations.

The look animations are controlled by the mouse (running on one thread) while movement is controlled on another thread which is playing the lower body animations (forward, back, side). Celebration and death animations control the whole body and are played on the same thread as the body movement animations.

Viewing multiple threads is the ShowTool will be covered in greater detail in the ShowTool section.

Blend Animations

There are special sequences that can be marked as blend animations. These 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.

Blend 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).

If a sequence is a blend, the transforms will be added on top of the other animations playing in the engine on a node by node basis. Only the animation values are added.

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

Blend Reference Time

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

The values added from the blend animation are based off of the root position in the 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.

You can set the blend reference position either by entering the blend reference frame number in the appropriate sequence node or by leaving your character in the root position at frame 0 (the default reference position) and then animating the extreme positions after that in a sequence that starts after frame 1.

Configuration Files

Configuration (.CFG) files are text files that allow you to specify how shapes and animations are exported. When you export a shape, the exporter looks for a file called <fileName>.cfg in the same directory as your DTS file is being exported to. The configuration file can contain several keywords that determine which object are exported and which objects are ignored as well as export parameters.

There are three lists that can be in the configuration file: AlwaysExport, NeverExport, and NeverAnimate. Node and object names are placed into one of those three lists. By default, all objects are implicitly placed into the AlwaysExport list. Names can include wildcards (*) to simplify writing configuration files. Lines with + or - along with the parameter name turn on and turn off boolean parameters. Lines with = set the value of valued parameters (note: the = occurs at the beginning of a line).
If a node's name matches a name on the NeverExport list, that node will not be exported. Any meshes that are children of that node will be parented to that node's parent (or its parent's parent if the parent is also on the NeverExport list). If a node is on the AlwaysExport list, it will be exported even if there are no meshes on the node, and even if the name matches a name on the NeverExport list. The AlwaysExport list takes priority over the NeverExport list. If a node is on the NeverAnimate list, its animation will not be exported even if it contains animations.

For example, assume that the following lines are in a configuration file:

```
AlwaysExport:
mesh*

NeverExport:
submesh*
```

This configuration file would export all nodes that have names beginning with "mesh" and ignore all nodes beginning with "submesh".

An example configuration file with all parameter names are included with the sample files.

Using .CFG files to control DSQ node structure
By default, all transform animation in a shape during a sequence is exported. If a node differs from the default position in a sequence then the transform on that node is considered animated even if it is unchanging throughout the sequence. The default position for all nodes is determined by the node position at frame 0 in the scene file. If Collapse Transforms is on, non-animating nodes will be collapsed out. If not, all nodes will be exported whether they animate or not, and as such, will be considered animating.

It is preferable to use .CFG files to determine which nodes are being exported to a particular thread, force export of needed nodes, manually cull out useless nodes (or ones with potential conflicts), and otherwise take steps to ensure that threads do not fight for control of the same nodes.

By using .CFG files, you can, for instance, animate and export animation for the lower body only and then other animation for the upper body only. When played in the ShowTool, the two threads will play without conflicts.

By explicitly culling out certain nodes, you can ensure that conflicts do not occur. Often the nodes that do not appear animated are indeed animating. This is especially true when using IK to animate, as the transform values of a node may be animating, but it might not be visible.

So, the ideal situation is to separate your animations into different types: full body, upper body only, lower body only, arms only, hands only, facial animation only, etc.

Create a separate directory for each animation type and make a custom .CFG for each type, culling out the un-needed nodes in the NeverExport list. Remember that the .CFG does not need to be named dtsScene.cfg. You can (and should) name them appropriately for the type of animations you are exporting.

Alternately, you can use Sequence Priority to determine which thread controls a node when conflicts occur, but this is a somewhat clunky way to deal with the situation. It is better to use the .CFG files to control the node structure of your .DSQ files to ensure there are no conflicts.

It is a good idea for all of the shapes to be in or pass through a root pose that is the same in the DTS and all of the DSQ files. This will insure that all the animations line up properly. This can be as simple as making sure that the pose in frame 0 of all of the scene files are exactly the same.

Player Sequences

What follows is a listing of the different types of animations that were used to create the default player.

Normal Full Body
These animations are what you would consider to be 'normal' animations, with all the nodes exporting and controlling the entire skeleton.

Root
All Sitting
All Death Animations
All Celebrations
All Salutes
All Taunts

Lower Body Only
These animations are exported with node control to isolate the nodes in the lower body of the character.

Forward
Backward
Side (this animation is played forward or reverse)
Fall
Land
All Jumps

Blends
These animations are blend animations that are designed to be played on top of movement animations and exist in the same directory as the full body animations, but are set to blend in the sequence node.

All Look
Head left/right
Head up/down
Recoil

Note that these sequences go from one extreme pose to another with the blend setting. The sequence is usually initialized to start in the middle and play either forward or backward from this state by the engine. The engine determines the position of the animation based on the mouse look.

Vehicles Sequences

turn: Each wheel needs a turn sequence, this controls the steering.

spring: Each wheel needs a spring sequence, This controls the up/down motion of the wheel (like shocks)

wheel: Each wheel needs a wheel sequence, ie. Wheel0, Wheel1, etc. This controls the spinning of the wheel.

Weapon Sequences

activate: Animation that is played when the weapon is activated and placed on the player

deactivate: Animation that is played when the weapon is deactivated and put away.

fire: Animation that is played when the player would fire the weapon.

noammo: Animation that is played when the player has no available ammo.

reload: Animation that is played when the player is reloading the weapon.

Crossbow Activate

Triggers

Triggers are arbitrary markers to 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. Because triggers are related to animations, they are added to their respective sequence markers object notes.

numTriggers #
The number of trigger state changes in this sequence.

triggerFrame(keyframe#) #
The frame number on which the trigger occurs.

triggerState(state#) #
The state of the trigger. (on or off)

numTriggers is the number of trigger state changes associated with a given sequence.

triggerFrame is the frame number in the sequence on which a trigger event occurs.

triggerState defines the state of a trigger on a given triggerFrame. 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 work with 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 is down.

You don't necessarily need to turn off the footprints (let's assume that the programmer will turn off when it is necessary), but you could by creating two more triggers with triggerStates -1 and -2 to turn off the triggers.

The attributes for triggers will change depending on the number of triggers on the sequence node.

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.

Trigger Example:

```
// # of trigger state changes
numTriggers 4

// first state: keyframe 3 trigger 1 on
triggerFrame0 3
triggerState0 1

// second state: keyframe 4 trigger 1 off
triggerFrame1 4
triggerState1 -1

// thrid state: keyframe 5 trigger 2 on
triggerFrame2 5
triggerState2 2

// fourth state: keyframe 6 trigger 2 off
triggerFrame3 6
triggerState3 -2
```

Foot prints on the terrain

Named Nodes

Named nodes are position markers in your DTS shape that can be either static or animated and are used to create mountpoints, camera positions and other special location markers.

Named nodes are created by adding a simple mesh object (like a box) to you scene, naming the object with an _ at the beginning of the object name (_mount0) and adding the object to the shape branch of the DTS hierarchy. (start01) Only the position, rotation and any animation of the object is exported, the mesh part of this object is culled out on export. You must also add named nodes to the always export list in the .cfg file for your scene, otherwise they will not be created when you export your DTS shape.

Eye and cam nodes are required for getting your character working correctly in the default Torque engine. These are special nodes referenced in code that determine where the player point of view (POV) is and the rotation point for the orbiting death camera. The eye node is used for the first-person POV; the cam node is used for the third-person POV. Both nodes can be parented to the head of the character (or wherever is most appropriate).

These are not required for the character to export, but they are required in order for the character to be dropped in the game and work as a player correctly. If they are not included, the eye node will default to origin of the bounding box.

For weapons to mount correctly, the model must contain mount points. The weapon is mounted to mount0. Without it, the weapon will mount at the players bounding box origin.

Additional mount points may be added as needed. For example, the default player in the Torque Game Engine contains mount0, mount1, ski1, and ski2 nodes. Consult with your programmer to determine what is needed for your game.

Remember that all nodes (eye, cam, mount, etc) must be present in both the base shape (DTS) and all sequence files (DSQ). It is sometimes helpful to view the scene through a node to make sure the node is oriented properly. You can resize nodes as you wish only placement and orientation are important.

Player Nodes

_cam: Required only for playable characters. The “cam” node is used to tell the engine where to view the model from if the camera perspective is placed into 3rd person mode. Without this node you will view from the transform coordinates of the “bounds” shape.

_eye: Required for playable characters, this node is essentially just a set of transform coordinates that tells the engine where to place the camera for viewing from 1st person perspective.

_light#: Light Emitter node

_mount#: Mount nodes are used to tell the engine that something can be attached to your model at this position. For example you would use a mount node to tell the engine where to place a weapon on your player model.

_ski#: Ski Marker node

Vehicle Nodes

_cam: Required only for playable characters. The “cam” node is used to tell the engine where to view the model from if the camera perspective is placed into 3rd person mode.

_contrail#: This emitter creates contrails, which are those little wisps that you see that emit from the tips of wings on airplanes.

_eye: Required for playable characters, this node is essentially just a set of transform coordinates that tells the engine where to place the camera for viewing from 1st person perspective.

_hub: For wheeled vehicles. Hub nodes are used to define where a wheel DTS shape will attach to the mesh when it is in the engine. Example use: hub0, hub1, hub2, hub3

_jetnozzle: This node tells the engine where to emit a jet engine exhaust from.

_light: Light Emitter node

_smoke_node: These nodes denote the location of a particle system to the engine. This particular particle system emits particles based on the amount of damage that your vehicle has taken.

_ground#: Each wheel needs a ground node, i.e. Ground0, Ground1, etc. These tells the wheels where they touch the ground.

_mass: Each vehicle needs a mass node. This should be positioned at the center of mass for the whole shape. This is used in the driving and flying dynamics.

_mount: Each vehicle needs mount nodes. These are where the players and weapons mount. Here are the names of the mount nodes and their function:

mount0 - pilot
mount1 - navigator\gunner
mount2 - passenger
mount3 - passenger
etc..
mount10 - gun
mount9 - bomb mount

Weapon Nodes

_ejectpoint: This node tells the engine where to emit the shell casing DTS file from when
the weapon is fired.

_mountpoint: This node tells the engine where to attach the weapon to the player model. The weapon is attached specifically at the location of the "mountpoint" to the corresponding "mount" node on the player character.

If there is no mountpoint in the DTS file or if the node is not exported, the engine will default to the origin (0,0,0) of the DTS file when mounting.

_muzzlepoint: Weapon Muzzle Flash node. The "muzzlepoint" node tells the engine where to show the 'flash' or 'bang' DTS file when the weapon is fired.

The .CS Script file

At runtime, the DTS and DSQ shapes are merged together to create a new shape that contains the mesh and all it's associated data (mountpoints, etc…) and the animations.

This is done by including a .CS file for the shape in the directory with the DTS and DSQ file. The name of the .CS file for the shape should be the same as the DTS file. For example, player.dts has player.cs.

To construct a shape in engine, you start off with this at the beginning of the file:

datablock TSShapeConstructor(PlayerDTS)

This tells the engine the name of the shape it is constructing and is called in the engine.

Next the shape and animations are added.

```
{
baseShape = "./player.DTS";
sequence0 = "./player_root.DSQ root";
sequence1 = "./player_forward.DSQ run";
sequence2 = "./player_back.DSQ back";
sequence3 = "./player_side.DSQ side";
sequence4 = "./player_lookde.DSQ look";
sequence5 = "./player_head.DSQ head";
sequence6 = "./player_fall.DSQ fall";
sequence7 = "./player_land.DSQ land";
sequence8 = "./player_jump. jump";
};
```

The base shape is added, and then all the sequences are added to the shape and given a number. All of the numbered sequences reference a file.

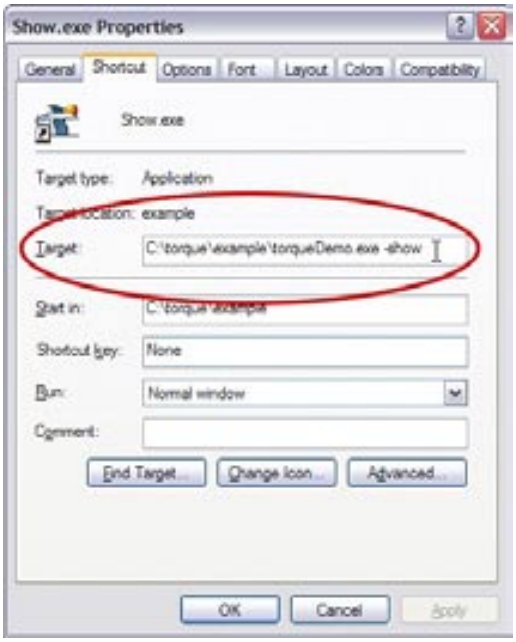
Names can be associated with the animations. You can see these names in the ShowTool in 'thread' control. For example, Sequence 0 is created in the shape, it uses player_root.DSQ, and the name of the sequence is root.

When you create a .CS file for your shape, it is important that the sequences are entered into the list in the correct order. Animations are called by the engine by index number, not by name.

The ShowTool

The ShowTool is simply a compiled Torque application that is running without game information and with a very simple UI. It is used to load and preview shapes and animations

The ShowTool can be accessed by putting the argument -show when you launch the application. This can be done from the command line, but it is easier to just add the switch into a shortcut.



Loading a Shape

In order to view shapes in the ShowTool, the shape and texture files must reside in a directory inside the Torque project. If you are using the demo, you can place the files in demo\data\shapes inside the Torque example directory.

After launching the ShowTool, you will see a black screen with several buttons on the left-hand side.

To load a shape into the ShowTool, click the Load Shape button, select a shape from the list, then click the Load button.Your shape will appear in the middle of the window. You can navigate around the shape by using the W, S, A, D, E, C, X, and Z keys.

Show Tool Movement Controls:

W: zoom in
S: zoom out
A: rotate left
D: rotate right
E: rotate camera up
C: rotate camera down
Z: rotate camera left
X: rotate camera right

Detail Levels

To manually view detail levels in the ShowTool, click the Detail Control button to open the Detail Control window.

By default the detail control will be set to Slider Sets Detail Level. Drag the slider to display the different detail levels.

Click the arrow button to switch to Auto Detail Using Distance. As you zoom in and out, you will see the different details levels change as the object gets closer to and further from the camera.

Other useful information is contained in the panel, including which detail level is being displayed and the polygon count.

Viewing Animations

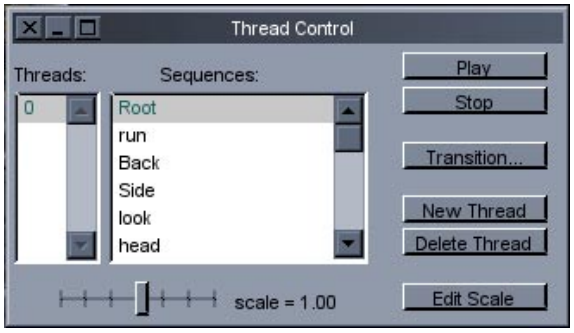
You can manually load animations onto shapes by clicking the Load Sequence button. As long as the animation sequence matches the shape, then the shape will animate correctly

If the shape already has animations and is using a .CS file to merge DTS and DSQ files, then the animations are automatically loaded into a thread. To view animation threads, click the Thread Control button to open the Thread Control window.

The Thread Control window allows you to playback and test all of the animations in your shape. The sequences list displays each animation for your shape using the name and order that was designated in the .CS file.

You can view all of the shape's animations by clicking through the list of sequences. You can also add another thread by clicking the New Thread button. This will give you a second thread that can run another animation simultaneously to the animation running in the first thread. Normally, multiple animation threads can be created for different parts of the body. You can have one thread controlling the lower body, a second thread for arms, a third for the head, etc. Multiple full body threads are not normally used and can behave unusually.

The threads can be controlled individually, started and stopped individually, played at different speeds (using the Edit Scale button), and transitioned individually on a thread by thread basis. In code, animations can be controlled to respond to the player input, play in reverse, play at different speeds based on player input, and transitioned at different speeds and over varied lengths of time, again, on a thread by thread basis.



Transitions

Although you cannot explicitly export or set transition parameters from gameSpace, the engine supports transitions. Transitions will transition from one sequence to another on a node by node basis over a set period of time (which can be varied). The transition is linear.

Using transitions allows the artist to animate without hitting the root position accurately. The engine will transition from one sequence to the other. This also allows for the engine to transition from one sequence to another mid-sequence (not returning to the root position). In this way, a character can go from a run to a walk mid-sequence (e.g. the character does not need to complete the animation) and the engine will interpolate the node positions during the change. Note that this can introduce some strange behaviors, because the engine will take the shortest path to translate the nodes to the next state.

To test how one sequence transitions into another, open the Thread Control window then click the Transition button to bring up the Transition Control window.

In the Window, the first arrow button will be set to Set Sequence. This means that the transition will pop to each new animation as you click through the list in the Thread Control window. Click the button to change the setting to Transition to Sequence. The sequences will now transition from one sequence to the next as you select them in the thread control window.

The other two arrow buttons allow you to play with how the transition happens. By default, the second button is set to Transition to Synched Position. This transition between the sequences using a default value. You can press this button to change the mode to Transition to Slider Position, which allows you to use the slider to determine where on the target sequence you want the transition to go to.

The last arrow button allows you to have the target sequence either play or pause during the transition. This is rarely used and usually works better if the target sequence is playing during the transition.

The Edit Duration button allows you to change how much the transition takes to go from one animation to another.