



Using the Crytek Exporter plugin for 3DMax

9/8/2004

Introduction.....	2
Installation	2
File types	3
Users Reference	5
Preparing the scene and objects.....	5
Crytek Shaders	7
Exporting the Data	7
Useful tips on exporting	9
Good modelling practice for exporting to CryEngine.....	10
Plugin interface reference.....	12
External tools reference	18
Cgfdump.exe	18
RC.exe.....	18
MotionOptimizer.....	18
Programmers Reference.....	19
File Structures	19
Plug-in Extension Modules	23
Appendix.....	25
Version History	25
Complete Programmers Reference.....	28

Introduction

This plugin, which runs under 3D Studio MAX version 4.x and higher, is used to export scene information from 3D Studio MAX into proprietary CryEngine formats. The interior structures of these file formats are explained in the programmers reference section of this document.

You can export the following types of information of a MAX scene:

- Bone Animation and Biped Data (as *.caf files)
- Geometric Information (as *.cgf or *.cga files)

Installation

To install the plugin, copy the following files to the plugins directory of your MAX installation.

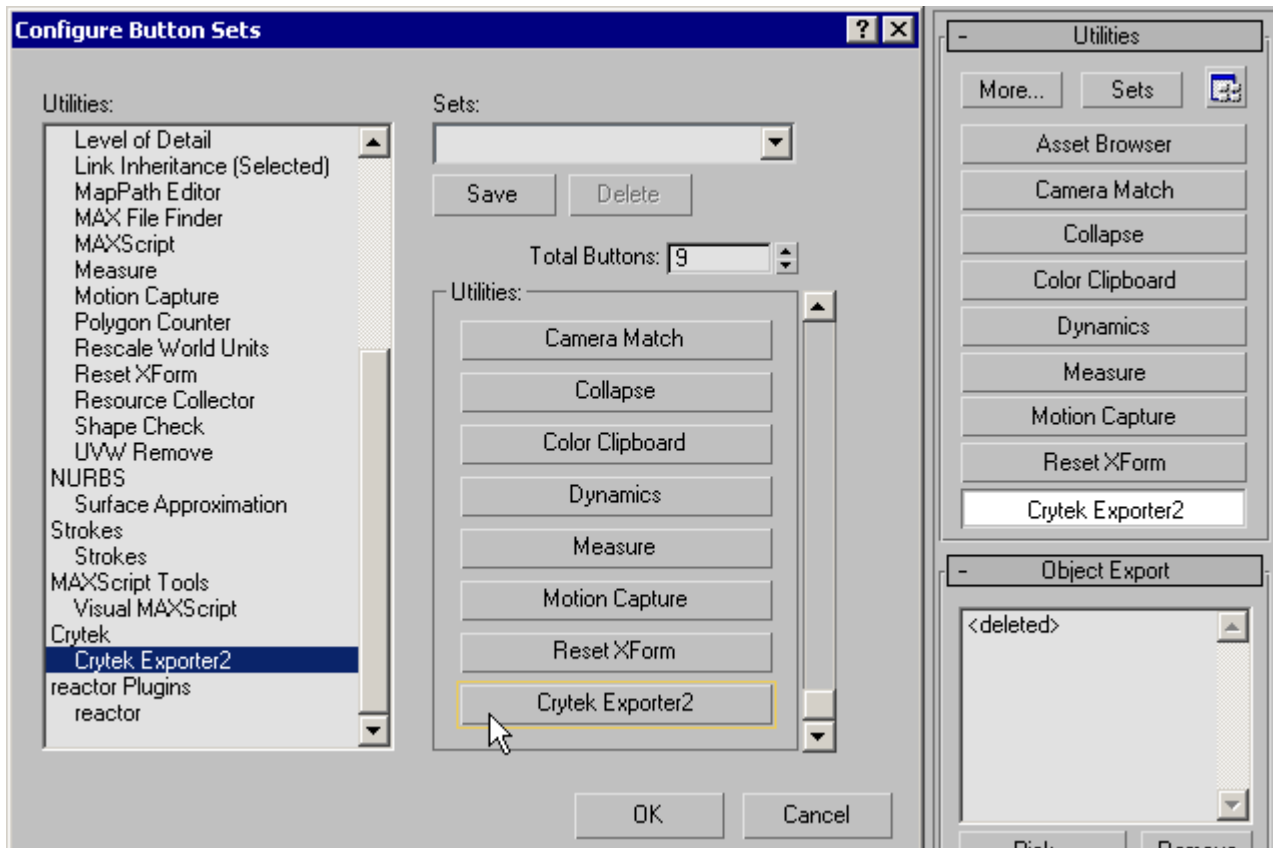
- CryExport.dlu – the exporter plugin
- msvcp70.dll – a Microsoft dll
- msucr70.dll – a Microsoft dll
- cryexport.ini – text file storing shader and material presets

When you restart MAX, open the 'Utilities' rollup, click the 'More...' button and select the 'Crytek Exporter2' utility. Click 'OK' and the plugin palette is activated.




Run the Crytek Exporter plugin

For quicker access you may also want to use 'configure button sets' option to assign a button for the plug-in as shown below (see MAX reference for explanation of this feature if necessary).



Add a Crytek exporter button

Click the 'configure button sets' button  to activate the Utilities button set. Increment the total number of buttons, locate the Crytek plugin and then drag the utility on to the new button. The button automatically gets the name of the utility (orange highlight on picture).

- Another file associated with this plugin is a settings file labelled "*.ccf" stored in the "plugcfg" folder of 3DSMAX. This file stores the plugin settings and is created by the plugin when the settings are saved. Load or save settings on the options pane of the plugin.

File types

*.caf files

*.caf files are Crytek animation files, used to store bone animation and biped data. These files are exported by the plugin in a proprietary format and contain the following information:

- Complete skeleton or any branch of a skeleton structure (biped or non biped), including additional bones added by the user, such as jaws or extra legs.
- Animation of the skeleton sampled at adjustable intervals and/or key times.
- User defined properties per bone (for tagging bones or defining some extra info about each bone).

*.cal files

*.cal files are Crytek animation lists. These files are simple, manually generated text lists of the available *.caf animations. They are only strictly necessary for animations that are shared; for example the "human_male" animations are used by many different characters in the game.

.cal files are also used to relate the actual *.caf file name to the name of the animation as understood by the CryEngine (alias). The lists take the following form:

```
lwalkfwd = human_male\lwalkfwd_loop.caf
```

The alias lwalkfwd can be used in the engine to refer to the actual animation file labelled lwalkfwd_loop.caf.

The default path to the .caf file is up two directories and then down to animations (..\..\animations). Crytek uses a naming convention for the animations, for example 'lwalkfwd' as shown above, which is necessary for the engine to play the animations properly. A full discussion of the naming convention is discussed in the file "Naming Character Animations.doc".

*.cgf files

- *.cgf files are Crytek geometry files, used to store geometric information including the following:
 - Static mesh objects with user defined properties, texture coordinates, material id (per object or per face).
 - Cgf geometries can be animatable or not. Animatable geometries contain bone binding and skin weighting data. Animatable cgf geometries can be used as static.
 - An animatable geometry "objectname.cgf" can only be used as animatable when there is an "objectname.cal" (Crytek animation list) near the .cgf or when the individual animations "objectname_*.caf" are near the cgf.
 - Option to export vertex positions in Local space.
 - Physique bone-vertex assignment values for bone deformed meshes (skin deformation info). There are options to determine one or many links per vertex. Only RIGID vertex types are supported, Deformable vertices are converted to RIGID.
 - Vertex animation for mesh objects
 - Lights
 - Point and Dummy helper objects.
 - Materials.
 - Textures and animated UV coordinates
 - User defined information for the scene and nodes.

*.cga files

*.cga files are Crytek animated geometry files. They are useful for animated map objects that do not require deformation of the mesh by bones.

- Export animated scenes from MAX directly into this format.
- Export format includes node hierarchy as well as the rigid body animations.
- Optimal way to produce rigid body animations such as an oil pump.
- Uses MAX TCB interpolation for keyframes.
- Also useful for animations that will have many instances repeated at one time, eg. winged bugs, because no bone deformations are required which keeps the file size down.

Users Reference

Preparing the scene and objects

Although we tried to cover all of the scene conditions and thus make the export process as transparent as possible, there are still some limitations of which you should be aware. This section describes the scene preparation process and mentions some important points and limitations.

General

Limitations

- Keep object names less than 64 characters. Object name, shader name and material name are concatenated to form the internal reference in the output file. This can cause an error during exporting, and in the game, if the total is longer than 63 characters. The only way around this is to make the object's name shorter. If the internal reference is too long, the material name will be truncated and the object will not work properly in the game.
- Keep texture file names less than 32 characters (including the extension but excluding the path).

User defined scene properties

You can enter any number of additional information for exported data for each MAX scene.

1. Click **File>Properties**.
2. Go to 'Custom' Tab.
3. Enter a name into the 'name' section. Do not use more than 32 characters for the name.
4. **Important:** keep the type field as Text.
5. Enter the value you want into the "value" section. Do not enter more than 64 characters.

Mesh Objects

Limitations

- Use only "Map Channel 1" for texture mapping.
- Do not apply space warps (they will be ignored during export process).
- Try to do the mapping coordinate assignment in MAX, although this is not strictly essential. Mapping coordinates generated by other packages may produce some problems.
- Avoid Off-axis non-uniform scaling. That is, try to avoid applying non-uniform scaling to a node (MAX warns you when you try). If you have to do non-uniform scaling: apply it on local axis space of the object. Non uniform scaling on other axes would shear the object and make it impossible to export. It is valid to use "Xform" modifier though.

User Defined Object Properties

1. You can add extra information to each of the mesh objects via 'Object Properties' dialog box.
2. Make a selection and right-click it.
3. Select 'Properties'.
4. In the dialog box select 'User Defined' tab.
5. Enter the object's custom information. The syntax and content of the information you enter there should be determined by the main/engine programmers because they will use this information. There is no limitation to the length of the string you type here.

Biped and Bone Systems

For bone animations you can use any kind of bone hierarchy, including Character Studio Biped figures or regular IK chains. You can add extra bones to biped figures or create your own hierarchy using MAX bones or other geometries. We recommend you use 'bone objects' instead of mesh objects (like boxes).

Limitations

- If you use a dummy object as a bone, you should uncheck the 'Ignore Dummies' check box on the exporter plugin palette. This option eliminates unnecessary bone creation for biped dummy objects attached to the ends of biped figure bones.
- Each *.caf file may contain only one skeletal system. Individual (distinct) skeleton systems are exported as separate files.

User Defined Bone Properties

You can add extra information to each of the bones via the 'Object Properties' dialog box.

1. Select the bone(s) and right-click them.
2. Select 'Properties'.
3. In the dialog box select 'User Defined' tab.
4. Enter the bone's custom information. The syntax and content of the information you enter there should be determined by the main/engine programmers because they will use this information. You should not enter more than 32 characters here.

Physique

Physique is a modifier which is particular to Character Studio. For bone deformed mesh animations, we export Physique Modifier data for the mesh.

Limitations

- **Important:** Only rigid type of link assignments can be exported.
- Only 1 mesh can be exported with 1 physique modifier

Any deformable links will be converted to rigid right before the export process. Therefore, if you use deformable links, the exported animation (which will be seen in the game in real time) will not be the same as the one you've set in MAX.

It is possible to get better animations with deformable vertices but, since 3D Studio MAX does not support exporting them (their calculations are too complex to fit in a real-time game), your animation will eventually be converted to rigid. We recommend that you to set up your character using rigid vertices and try to refine the links and get the most out of it. By doing this you'll be able to see the animation in MAX exactly as it will be seen in the game.

To use Rigid links select Rigid Vertex - Link assignment during the initialization process of Physique. You are free to choose any Blending option (No blending, N links etc.).

Materials

Please check the below limitations. Only the information covered by the limitation section will be exported, other information will be ignored.

Limitations

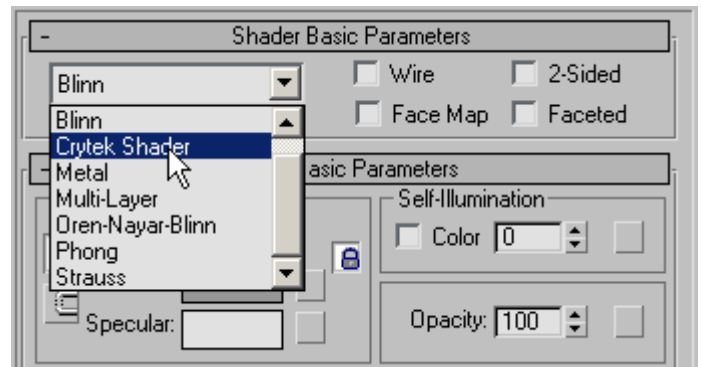
- Only Standard and Multi/Sub-object materials are supported. Do not use other materials.
- Do not use nested Multi/Sub-object materials. It is meaningless both for MAX and for our exported data.
- Only texture mapping channel 1 is supported. Do not use texture channels other than 1.
- Only Ambient, Diffuse and Specular color values are exported.
- Only Diffuse, Bump and Opacity maps are supported.

- Only Bitmap textures are supported.
- Material name should not exceed 64 characters.
- Bitmap texture file names should not exceed 32 characters (including the extension but not the path).

Crytek Shaders

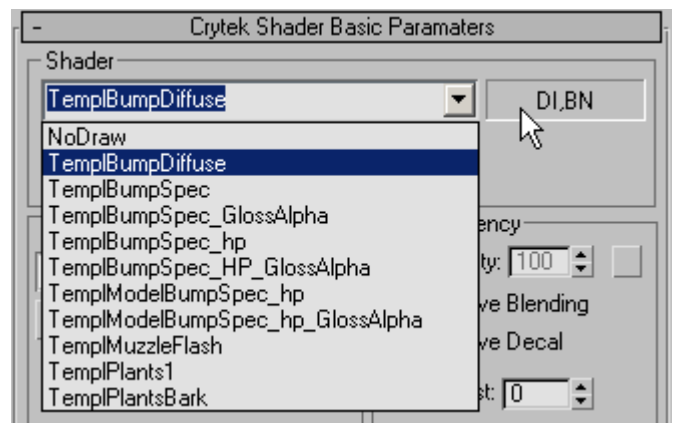
When creating objects for the CryEngine, use the Crytek shaders and materials on the objects. The CryEngine is set up to recognise and process these properties.

If the plugin is loaded, the Crytek shaders are available in the materials editor in MAX.

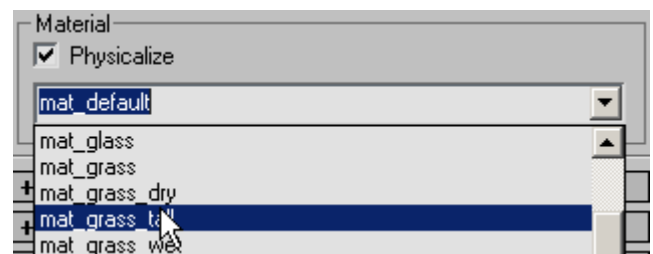


Select the object's shader

With 'Crytek Shader' selected, shaders and materials are loaded from the CryExport.ini file. This makes applying shaders and physical materials very easy.



Select a shader



Select a material and flag 'Physicalize'

Exporting the Data

Before exporting the data please be sure that your scene and objects satisfy the requirements stated in "Preparing the scene and objects" section.

The plugin interface has two main rollups. The 'Object Export' rollup creates .cgf or *.cga files and the Bone Export rollup creates *.caf files.

CGF (Crytek geometry file)

The file describes 4 types of MAX scene objects:

- Static mesh objects.
- Bone-deformed mesh objects (meshes with Physique modifier on them).
- Vertex animated mesh objects (including non-Physique deformations).
- Non-mesh objects: lights, helpers, materials.

CGA (Crytek animated geometry file)

- Like .cgf but it contains node animations using TCB splines exported from MAX.
- Can be used like a character except it has no bone or skinning information.
- Generated in the same way as .cgf files, except you should choose file type '*.cga' when prompted by the save dialog.

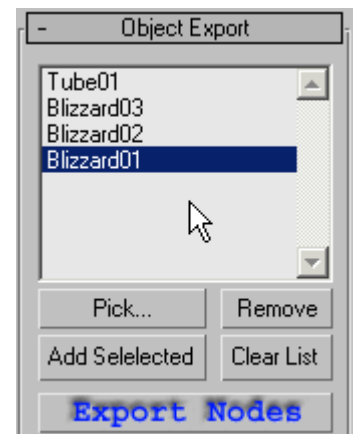
CAF (Crytek animation file)

- Holds bone structure and key framed animation data.

Exporting .CGF files**Select nodes**

The list in the top of the Object Export palette contains the nodes which will be exported from MAX. Add to this list by clicking 'Pick...' and then clicking a node, or by making a selection and then clicking 'Add Selected'.

Use 'Remove' to take individual nodes out of the list, or 'Clear List' to remove all nodes.



Selecting nodes – this pic to be replaced – also twice in appx

Nodes which can not be exported are filtered out during selection. You can not pick such nodes and they will not be added to the list when using 'Add Selected', even if they are selected. Take care when including objects in the list because some bones can be selected as geometries. Bones should be exported separately.

All of the objects included in the list will be written into a single cgf file. If you would prefer to place them in separate files, export them one at a time.

Export to .cgf

Export the selected nodes by clicking the 'Export Nodes' button. You'll be prompted to name the .cgf file (with the MAX project file name as the default value).

Click 'OK' and the plugin will export the data. A progress bar displays during the export operation. Any problems that are encountered will be displayed in a message box which prompts you to continue or cancel.

Exporting using name property

When you name an object within the scene, the .cgf file created by the exporter for that object will be named after the object.

1. Right-click the object and select **Properties**.
2. In the 'General' tab, the first field is for the name of the object. Enter the object name as you wish the .cgf to be named.

You can create directories within the export destination folder by using the string /dirname/objectname in the object's name property. This is particularly useful when exporting the selection to individual .cgf files.

Another thing to remember is that the cgfdump.exe tool (discussed later in this document) works from a command prompt and you should be careful about using spaces in the object's name.

Exporting .CAF files

Exporting bone hierarchy and animation is similar to object export. You need to select the bones and add them to the bone list before clicking the 'Export Bones' button.

There is one major difference to the workflow: If 'Add Related Bones' is checked (in the Object Export rollout), any related bones are automatically added to the bone list when geometry is added to the object list. If the 'Add whole skeleton' option is also checked, the entire skeleton system will be exported.

In other words if 'Add whole skeleton' is OFF, only the bone branches responsible for the deformation of the selected objects will be added to the bone list. If it is ON, the entire skeleton system (starting from the root bone) will be added to the bone list. More information on these options can be found in the appendix of this document.

Please note that if you use dummy objects as bones you should uncheck the 'Ignore Dummies' option. This eliminates the unnecessary biped end nodes which are not used to deform objects.

The bone list displays only the root nodes of the branches that will be exported. When you click the 'Export Bones' button all of the bones in the list and all of their children will be exported.

If the bone list has more than one branch, the branches will be exported in separate files because each .caf file can hold only one skeleton system. In this case you will be prompted, during the export process, for file names for each of the branches found in the bone list. The default .caf file name for each branch is "MAX file name" followed by an underscore character and the name of the root bone for that branch. For example, if "Bip01" is the root bone in "Harley.max", the default .caf file name will be "Harley_Bip01.caf".

Animation

You have the choice to sample the animation at regular intervals and/or at bone keys along with the automatic or manual time range selection. If the 'manual range' option is OFF, the current MAX time range will be used.

Optimization

CAF files can be optimized during export with a simple linear fitting algorithm. Although CAF files can contain B-Spline data, this format can only be made by Motion Optimizer out of non-optimized CAF files.

Reduce .caf file size by enabling the Key cleanup option. Other methods include decreasing the animation sampling rate, or sampling only at certain keys.

Useful tips on exporting

Tip: Facial expressions can also be exported. Use the 'Morpher' modifier in MAX to create the facial expression and pick the phonemes to export. The morph targets can be exported and previewed as usual. When they are exported, morph animations have a # mark appended to the front of the filename, otherwise they are used in the same way as other animations.

Finally, here's the tip: Set the MorphTarget minimum offset property to 0.0001 for facial expressions and other small mesh deformations. Normally this optimisation for large animations is set to a higher value.

Tip: Check the error logs after exporting to discover any problems with the model. Use the console commands to debug the model and work out the errors. This will save a lot of time later.

Tip: You can drag a .cgf model into MAX and it will load the mesh geometry as it was in the MAX file. You will lose texturing information and bones, but this is a handy way to recover some of your work if you lose the source .max file of the object.

Tip: You can type commands into the editor or previewer console, some of which are useful when debugging your objects. Try some from the following list:

ca_drawbones 1 – this command draws all the exported bones, in a contrasting color. it is best used with 'Wireframe' view.

Good modelling practice for exporting to CryEngine

Materials

- Use the minimum possible number of different materials per individual object.
- Where an object has areas of different materials, try to keep each material as continuous as possible. In other words, avoid islands of different material by applying materials to continuous regions of triangles whenever possible.
- Number of polygons used for each material should not be less than 200.
- Never use more than 3 materials on vegetation objects. Grass and ferns should only need 1 while trees may need up to 3.

Modelling

- Avoid modelling objects with open edges. Either they don't work at all or they work very slowly with shadow volumes.
- Use the least number of sides possible for the object (minimise the geometry). For example, tree bark requires only single sided polygons. Drawing the unnecessary interior surface of tree bark is bad for the fill rate and slows down the renderer and thus the frame rate.
- The model must be smoothed wherever possible.
- Avoid strong surface bending where possible.

Skeletons

- To avoid random errors, export all LODs (levels of detail) with the same bone structure. The best way to do this is to have one MAX file with 3 skins on one biped. This saves time and many, but not all, models are already kept in this way.
- Whenever possible, avoid using an excessive number of bones in models. At present, all bones are recalculated even if they don't participate in the mesh deformation.
- The recommended number of bones in a dead body skeleton is 10-14, and in alive skeleton 20-23 (having less than this number is acceptable).
- Skeleton should be physically valid in order to behave properly, because geometry shape and mass are taken into account.
- Heavy-light-heavy patterns should be avoided (important for dead bodies). A typical example from a biped is spine-neck-clavicle-upper arm connection with heavy spine, light neck and clavicle, and relatively heavy upper arm. Exclude neck (not necessarily neck1,2, though) and clavicles from dead bodies.
- Geometries of dead skeletons should be simpler than those of alive ones. An alive skeleton should have around 12-35 polygons and should not have sharp corners near joints.
- In *dead* character models, avoid limb geometries that are thick near the joint. At joints, the child bone should be thinner than the parent, or both should be thinner at the joint than in the middle.

Good joint 

Bad joint 

- Avoid using joints with all 3 axes locked. Such joints slow down the simulation for no reason.

Plugin interface reference

Object Export Rollup

Object List

This list holds the names of nodes (mesh objects, lights, helper object etc) that will be saved into the .cgf file(s) during the export process. To export nodes just add them to the list and click the 'Export Nodes' button. The node types that you can add to this list are:

- Any objects which are either mesh or can be converted to mesh
- Omni, Target Spot, Free Spot, Target Direct and Free Direct lights.
- Dummy helpers
- Point helpers

Any other object type will be rejected if you try to add to this list.

Pick...

This button is used to add a single object to the Object List by clicking it in any of the MAX view ports. You can not pick objects of unsupported types or the ones already contained in the Object List.

Remove

This button removes the selected objects from the Object List.

Add Selected

This button adds the selected objects from MAX into the Object List. Unsupported objects in the selection will be ignored.

Clear List

This button empties the Object List.

Export Nodes

When you hit this button, the objects contained in the Object List will be saved in a '.cgf' file. You'll be asked to give a name and a path for the file. The MAX file name will be used as the default name in the file save dialog box. You can accept it or give a new name.

Preview

This option is deprecated. The button currently replicates the 'Export Nodes' function.

Game

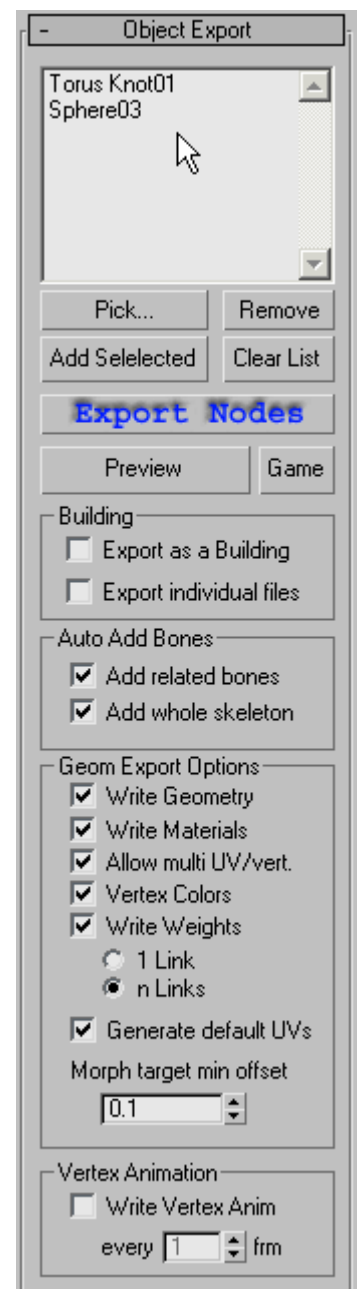
This option is deprecated. The button currently replicates the 'Export Nodes' function.

Export as a building

Exports the selected nodes as a *.bld file instead of the usual *.cgf. *.bld files are no longer used or supported (Build 1014). This option is deprecated.

Export individual files

Exports the list of nodes as individual files. When exporting with this option you will be prompted for a target directory to which all the files will be written. Use the name property of each node to define the output file name as well as directory. Directories referenced in the node name property which don't yet exist will be created during the export.



Add related bones

If this option is ON, all of the objects in the list are checked to whether a Physique modifier is applied to them or not. For objects that have Physique, the bones that are used to deform the object are added to the Bone List automatically.

Add whole skeleton

This option determines which part of the bone hierarchy will be added to the Bone list if the 'Add related bones' option is ON.

If this option is turned OFF, individual bones or branches are added to the Bone List. Please note that only the root bone of the deforming branch is added to the Bone list; any bones that are members of listed branches will not be listed separately in the Bone List.

For example, assume that you have added both of the arms of a mesh character to the Object List. Let the bones affecting these meshes be Upper L/R Arms, Lower L/R arms and L/R hands. If 'Add whole skeleton' option is off, only the bones Upper L Arm and Upper R Arm will be added to the Bone List. Since other bones are contained in the branches rooted by "Upper L Arm" and "Upper R Arm", they will not be added to the list.

If "Add whole skeleton" is ON, on the other hand, the very root of the skeleton will be added to the Bone List (eg. Bip01)

Write Geometry

This option is not yet fully implemented.

Write Materials

Determines whether the materials applied to the exported objects will be listed in the exported file or not. Please note that only Standard and Multi Sub materials are supported.

Allow multi UV/Vert

This option determines whether or not multiple UV co-ordinates per vertex are allowed. Normally keep it turned on. Ask the engine programmer about his requirements.

Vertex Colors

If this option is enabled, the vertex color info of the mesh file will be exported (if the mesh has vertex colors assigned).

Write Weights

This option determines whether the Bone-Vertex assignment data of the Physique modifier will be saved to the .cgf file. Keep it turned on unless you want your Physique-deformed objects to be exported as static (non-animated) objects.

1 Link/ n Links

These options determine if multiple bones can be used to deform vertices or not. This parameter is used as a limiter to the Physique settings. If in Physique you have chosen 1 link and you select n Links here, only 1 link per vertex will be used.

Generate default UVs

If no UV co-ordinates are applied to the object being exported, this object will normally appear black in the CryEngine. When the 'Generate default UVs' option is enabled, dummy UV co-ordinates will be generated so that the artist will at least be able to see the default placeholder texture on the object.

Morph target min offset

This option is used to reduce the number of vertices exported for vertex animations, such as facial expressions. This is because the morph modifier is not perfect - sometimes it can report all or a significant number of the vertices as morphable while they're not. This would represent a great deal of memory waste in the engine if all vertices were exported.

The value sets a threshold which determines the minimum size of vertex movements which are exported. Any vertex whose displacement from one position to the next exceeds this offset will be included in the exported data. Vertices moving with a lower amplitude than this setting will not be exported at all.

Write Vertex Animation

If this option is turned on, vertex positions will be sampled in specified time periods and be written into the .cgf file. This is basically a raw vertex animation data. The next parameters determine the sampling period and range. Please note that vertex animation data is quite big in size compared to the bone deformed animation.

every # frm

This parameter determines the sampling period (in frames) used for vertex animation.

Bone Export Rollup

Bone List

This list holds the names of the root bones of the bone branches that will be exported. The bones in the list and their sub-tree members will be written in the .caf file.

If the newly added bone is a parent or ancestor of a bone already listed, the existing bone will be removed from the list because it will be exported within the sub-tree of the newly added parent/ancestor.

Pick...

This button is used to add a single bone to the Bone List by clicking it in any of the MAX view ports. You can not pick bones that are already contained in the Bone List.

Remove

This button removes the selected bones from the Bone List.

Add Selected

This button adds the bones that are selected in MAX to the Bone List. If the selection contains bones that are already contained in the Bone List the repeated bones will be ignored.

Clear List

This button empties the Bone List.

Export Bones

When you hit this button, the bones contained in the Bone List will be saved in .caf files. Each branch in the list will be saved into a separate .caf file.

For each branch you'll be prompted for a file name and path. The MAX file name, combined with the branch's root bone name will be used as the default name in the save dialog. eg. Object_rootbone.caf

Ignore Dummies

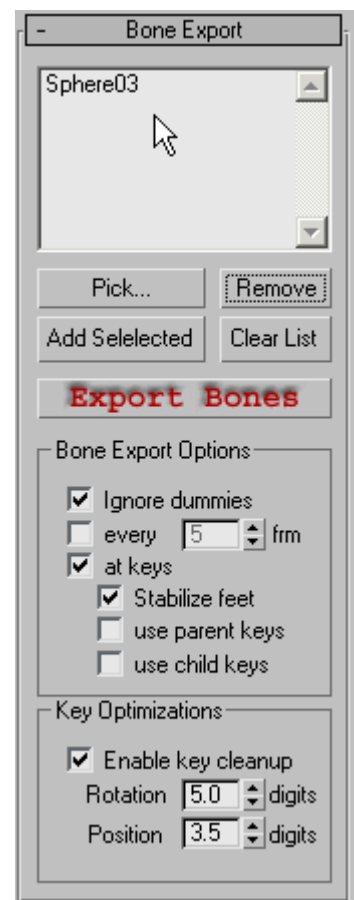
If this option is ON, any Dummy Helper object used as a bone, and its sub-tree, will be ignored during the export process. This eliminates unnecessary bone creation for the terminating dummy bones used in Biped figures.

These dummies are used only for user IK manipulation and are not used for mesh deformations. If you have used dummies in your skeletal system explicitly and if they are used to deform meshes, turn this option OFF. Otherwise keep it turned ON.

every # frm

If this option is turned on, the bone animation will be sampled at specified regular periods. The number entered here is in frames. We have tried some techniques to avoid regular sampling periods since it produces unnecessary keys even for non-animated bones or time sections. Try to keep this option turned OFF unless you can not get desired precision with other sampling options.

If you choose to get the exact MAX animation into your .caf file, despite the cost in file size, you can set the sampler to record keys for every single (set to 1) frame. Such animations can be optimised



during export using the linear 'Enable key cleanup' optimisation, or externally using the Motion Optimiser tool. The Motion Optimiser uses B-spline interpolation to optimise .caf files.

at keys

When this option is turned ON, the animation of each bone will be sampled at the transformation (rotation, translation and scaling) key frame times of that bone. You may also choose to sample the bone's animation at key frame times of its parent or child bones, as described below.

Stabilize feet

When this option is turned ON, all of the leg bones' animation will be sampled at the same time values. In other words, all of the key times of the bones that belong to a leg will be combined and all of these bones will be sampled at these combined key times.

This will produce the same degree of animation curve for each of the bones in the leg hierarchy, thus eliminating small displacements of the feet (slides) which are expected to be fixed on the floor.

Please note only Biped figures are supported. For multi-pad creatures you may want to turn on both 'use parent keys' and 'use child keys' to stabilize the non-rear feet.

Use parent keys

If this option is turned ON, every bone will be sampled at its parents' key times as well as its own key times. This is used to eliminate unwanted slides at the extremes. Please note that this option will produce more keys and thus make the .caf file bigger.

Use child keys

If this option is turned ON, every bone will be sampled at its children's key times as well as its own key times. This is used to eliminate unwanted slides at the extremes. Please note that this option will produce more keys and thus make the .caf file bigger.

Enable key cleanup

This option uses simple, linear key reduction to remove unnecessary keys and thus help optimise the animation.

The number of keyframes in a sampled animation is set by the 'every # frm' option in the Bone Export Options. This gives rise to sequences of rotations and positions generated by the controller sampler. If this option is enabled, keys will be removed from the sampled sequences if they represent near linear change.

When 'Enable key cleanup' is not checked, **all** the keyframes generated by the sampler will be exported in the .caf file.

The tolerated deviation from linearity is set by the 'Rotation' and 'Position' options.

Rotation

This option sets the tolerated deviation from linear change while the angle changes during an animation. It sets the tolerance or error margin when the exporter attempts to reduce the keyframe density of the bone rotation animation.

This number sets the deviation or error margin used when removing 'unnecessary' keys from the animation. For example, if the deviation is set to 2 digits the error margin is $1/10^2 = 0.01$ (two decimal places accuracy in change in radians).

Any change in radians that falls within this tolerance, when compared with the adjacent change, will be assumed to be a linear change and will thus be removed. Changes outside this tolerance will be ignored during the reduction as they are significantly large to be considered important.

Position

This option sets the tolerated deviation from linear change while the position changes during an animation. It sets the tolerance or error margin attempts to reduce the keyframe density of the bone translation animation. The number of keys used to produce the animation is reduced in the exported file.

This number sets the deviation or error margin used when removing 'unnecessary' keys from the animation. For example, if the deviation is set to 2 digits the error margin is $1/10^2$ (two decimal places accuracy in change in spatial units).

Any change in spatial units that falls within this tolerance, when compared with the adjacent change, will be assumed to be a linear change and will thus be removed. Changes outside this tolerance will be ignored during the reduction as they are significantly large to be considered important.

Timing Rollup

This rollup controls the animation range covered by the exporter. You may also enter sub-ranges in a table. These values are saved with the max file so they will be retained for when you next open the MAX file.

Manual Range

If this option is disabled, the exporter uses MAX's currently active Animation Range for exporting. Enable this option if you want to specify the time range manually.

Start, End

Enter the start and end of the manual animation range here. These parameters are enabled only if the Manual range option is enabled.

Edit Sub-Ranges

This button activates the editor for named animation Sub-Ranges. You can edit the named sub-ranges in this editor which stays open as long as the main UI of the plugin is visible.

You can use MAX commands and go back and forth in the MAX animation range without closing this dialog.

Options Rollup

Load...

Loads a saved configuration file. Configurations are saved in files with .ccf (Crytek Config File) extensions. The default directory for configuration files is the 'plugcfg' directory of MAX.

Save...

Saves the current configuration in a file. Configurations are saved in files with ccf (Crytek Config File) extensions. The default directory for configuration files is the plugcfg directory of MAX.

Save as Default

Every time you open the exporter plugin interface, the default configuration file named '_default.ccf' is loaded if it exists in MAX's plugcfg directory. Click this button if you want to save the the current configuration as the default.

Reset

Resets all of the parameters to the defaults.

Auto Save

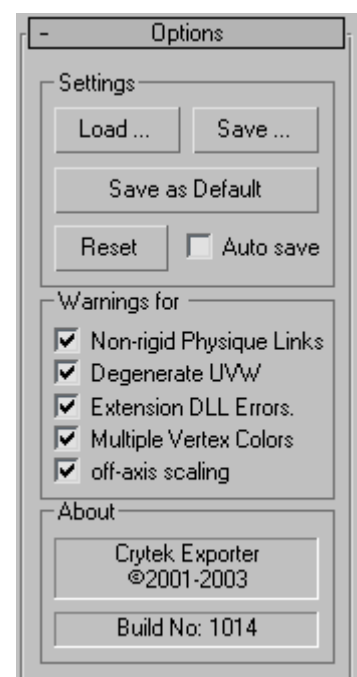
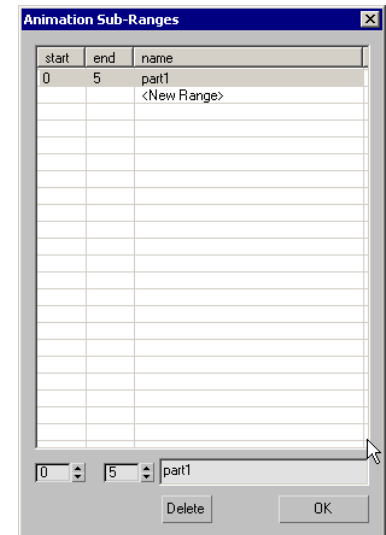
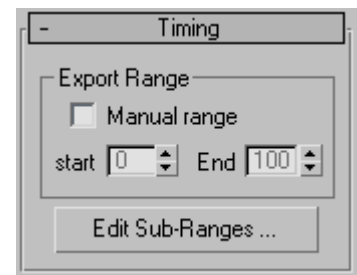
If this option is checked, the current configuration will be saved to the '_default.ccf' file every time plugin interface is closed.

Non-rigid Physique Links

If this option is checked, a warning box will be displayed whenever an object with deformable vertex types are set in Physique modifier.

Degenerate UVW

Sometimes texture vertices became orphaned or are used by multiple (non-related) geometric vertices. This is especially true for hand-edited UVW coordinates.



This may cause trouble for the real time engine, so this option is used to warn whenever objects with such mapping coordinates are being exported.

Please ask the engine programmers about the current state of the engine's texture vertex co-ordinate input requirements to clarify the issue.

Extension DLL Errors

The exporter plugin's functionality can be extended with external DLL modules.

Upon completion of the export process (for either .cgf or .caf files), the DLL module(s) placed in the plugcfg directory will be executed. During this external DLL execution time, errors may occur or the DLL may return an error code. If you want to be informed about these errors keep this option checked.

Multiple Vertex Colors

If this option is enabled, a warning box will be shown whenever vertices with more than one assigned vertex color are encountered.

Off-Axis Scaling

This option enables the warning for nodes that have "off axis non-uniform scaling".

External tools reference

Cgfdump.exe

This is a command line utility which takes a .caf or .cgf file as the input parameter and returns chunk information and the chunk list to the screen. You can use the DOS directive '>' to save the results into a text file.

To get help on cgfdump.exe commands, simply type `cgfdump` at the command prompt.

Usage: `cgfdump [-option1] [-option2] [more options...] filename.cgf(or caf) > Output-file-name.txt`

Cgfdump command line switches

Option switch	What it does
-briefInfo	Prints only the summary information about the file and the timing very essential chunks (timing).
-keys	Prints detailed information about keys.
-mesh	Prints all detailed information about mesh.
-meshVerts	Prints mesh chunk vertices.
-meshFaces	Prints mesh chunk face lists.
-meshUVs	Prints mesh chunk UV coordinate lists.
-meshTexFaces	Prints texture face lists.
-meshBones	Prints bone binding info.
-meshVertCols	Prints vertex color lists.
-collectTextures	Observes all textures used by cgf file and lists them at the bottom of the output.
-collectTexturesForCopying	Outputs textures prepared to copy somewhere as a set of commands.

RC.exe

ResourceCompiler is a command line interface used to compile geometries and textures. More information can be found in the document titled 'How to Use the ResourceCompiler.doc'.

MotionOptimizer

This tool uses spline interpolation between keyframes to reduce the keyframe density of the animation. It is used to optimise the *.caf files for enhanced animation in the CryEngine®.

The best option for optimising the exported animations is to export the animation from MAX with every keyframe (set the every...frame option to 1) included (no optimisation with the plugin). Use MotionOptimizer tool on the resulting .caf to optimise the animation.

The MotionOptimizer tool is discussed in the document entitled "Using MotionOptimizer".

Programmers Reference

File Structures

There are two file types we export: cgf (Crytek Geometry File) and caf (Crytek Animation File). The inner structure of both files are the same: both of them have chunk based contents. Files may have any number of chunks which do not appear in a fixed order. Also chunks do not have fixed sizes: every chunk has its header at the beginning followed by the actual data.

There is a table at the end of each file which lists all of the chunks along with their positions in the file. So by reading this chunk table you can randomly access the chunks that interest you. The position of the chunk table is stored in the header of the file.

The file structure comprises three main elements; file header, chunks and chunk table.

File Header

File header consists of one FILE_HEADER structure which is defined in the "Headers.h" file.

```
struct FILE_HEADER
{
    char        Signature[7];
    FileTypes   FileType;
    int         Version;
    int         ChunkTableOffset;
};
```

Signature

Should hold the constant FILE_SIGNATURE defined in the "Headers.h" file. You can test if a file is caf or cgf file by checking this value. The definition of FILE_SIGNATURE is:

```
#define FILE_SIGNATURE "CryTek"
```

FileType

This value indicates the if the file is CAF or CGF. possible values are:

```
enum FileTypes
{
    FileType_Geom= 0xFFFF0000,
    FileType_Anim,
};
```

Version

This holds the version of the file. This should match the version constants defined in "Headers.h" file. Current values are defined as:

```
#define GeomFileVersion          0x????
#define AnimFileVersion         0x????
```

ChunkTableOffset

This integer gives the offset position of the Chunk Table.

Chunks

Each chunk in the file holds different type of data associated with the chunk type. Every chunk begins with a chunk descriptor which is different for each chunk type. But every chunk descriptor begins with CHUNK_HEADER structure which is a copy of the chunk header found in the chunk table. You can

double check if the chunk being read is correct by comparing this info with the corresponding values found in the chunk table. Some chunks hold all of its data within the Chunk Descriptor structure while the others (usually the ones with variable length of information) has some more information saved just after the chunk descriptor.

Chunk_Header

Chunk header structure is stored in two places: in the chunk table and the beginning of the each chunk descriptor.

```
struct CHUNK_HEADER
{
    ChunkTypes  ChunkType;
    int         ChunkVersion;
    int         FileOffset;
    int         ChunkID;    //new in version 0x0744
};
```

ChunkType

is one of the constants defined as below in "Headers.h" file

```
enum ChunkTypes
{
    ChunkType_ANY           = 0,
    ChunkType_Mesh          = 0xCCCC0000,
    ChunkType_Helper,
    ChunkType_VertAnim,
    ChunkType_BoneAnim,
    ChunkType_GeomNameList,    //Obsolute
    ChunkType_BoneNameList,
    ChunkType_MtlList,        //obsolute
    ChunkType_MRM,           //obsolute
    ChunkType_SceneProps,
    ChunkType_Light,
    ChunkType_PatchMesh,     //not implemented
    ChunkType_Node,
    ChunkType_Mtl,
    ChunkType_Controller,
    ChunkType_Timing,
};
```

ChunkVersion

Is the matching constant defined in "Headers.h" file (with the current versions).

```
MESH_CHUNK_DESC_VERSION
SCENEPROPS_CHUNK_DESC_VERSION
MRM_CHUNK_DESC_VERSION
HELPER_CHUNK_DESC_VERSION
VERTANIM_CHUNK_DESC_VERSION
BONEANIM_CHUNK_DESC_VERSION
GEOMNAMELIST_CHUNK_DESC_VERSION
BONENAMELIST_CHUNK_DESC_VERSION
MTLLIST_CHUNK_DESC_VERSION
LIGHT_CHUNK_DESC_VERSION
```

NODE_CHUNK_DESC_VERSION
 MTL_CHUNK_DESC_VERSION
 CONTROLLER_CHUNK_DESC_VERSION
 TIMING_CHUNK_DESC_VERSION

FileOffset

This gives the position offset of the actual chunk in the file. To read the actual chunk you should fseek to the position indicated here.

ChunkID

Every chunk in the file has given a unique ID. Some chunks reference to other chunks with this id. for example node chunks reference the objects that this node has by giving the id of the chunk that the object is stored within.

Chunk Data

Every chunk has its own data format according to its chunk type. Depending on the chunk type given in the header file this section holds the actual data. Please follow the below links to get information on each chunk type.

Chunk types

ChunkType	ChunkType Constant	Status
Mesh Chunk	ChunkType_Mesh	
Multi Resolution Mesh Chunk	ChunkType_MRM	
Helper Chunk	ChunkType_Helper	
Vertex Animation Chunk	ChunkType_VertAnim	
Bone Animation Chunk	ChunkType_BoneAnim	
Geom Name List Chunk	ChunkType_GeomNameList	Removed
Bone Name List Chunk	ChunkType_BoneNameList	
Material List Chunk	ChunkType_MtlList	Removed
Scene Properties Chunk	ChunkType_SceneProps	
Light Chunk	ChunkType_Light	
PatchMesh Chunk	ChunkType_PatchMesh	Not Implemented
Node Chunk	ChunkType_Node	New
Material Chunk	ChunkType_Mtl	New
Controller Chunk	ChunkType_Controller	New
Timing Chunk	ChunkType_Timing	New

Chunk Table

This table is located at the end of the file and lists all of the chunks that the file holds. The location of this table is given in the header of each file. The structure of the table is as below

nChunks

Chunk Item 1

Chunk Item 2

Chunk Item 3

...

Chunk Item [nChunks]

nChunks

This integer value gives the number of chunks that the table (therefore the file) holds.

Chunk Items

Chunks Items are of the type `CHUNK_HEADER` which is defined in "Headers.h" file.

Plug-in Extension Modules

Introduction

The plug-in is designed so that it can be extended easily with external simple DLL routines. After exporting the caf or cgf files, the plug-in searches the MAX's plugcfg directory for DLL's named Crytek*.DLL. Plug-in then calls each of the DLL's exported functions named "GeomCallback" and "AnimCallback" for cgf and caf files respectively.

So if you have any task that should be performed for each cgf or caf file, you can implement it as a DLL function and place it in plugcfg directory. Since the process will be executed each time a caf/cgf exported, keeping the files up-to-date will be quite easy.

You can have as many DLL's in the plugcfg directory as you want: just give them different names which starts with Crytek (the extension should be DLL).

DLL structure

These extension modules are simple DLL's with one or two exported functions named "GeomCallback" and "AnimCallback" for cgf and caf files respectively. Each of these functions take two parameters: plug-in's window handle for general usage and the exported file's full path name. You can open the files within these functions and do whatever you like; analyze it, convert it to other file type, delete sections, change data etc.

For the return values of these functions 0 indicates no error. Any non-zero value will be considered as an error value and a warning dialog box will be fired (if the user has enabled the warning options).

Here is the sources of a skeleton Extension DLL.

CryExtDLL.cpp file

```
#include <windows.h>
BOOL APIENTRY DllMain(HANDLE hModule, DWORD ul_reason_for_call, LPVOID lpReserved)
{
    switch (ul_reason_for_call)
    {
        case DLL_PROCESS_ATTACH:
        case DLL_THREAD_ATTACH:
        case DLL_THREAD_DETACH:
        case DLL_PROCESS_DETACH:
            break;
    }
    return TRUE;
}

__declspec( dllexport ) int GeomCallback(HWND hWnd, char *filename)
{
    return 0;
}

__declspec( dllexport ) int AnimCallback(HWND hWnd, char *filename)
{
    return 0;
}
```

CryExtDLL.def file

```
LIBRARY CryExtensionDLL
EXPORTS
    GeomCallback @1
    AnimCallback @2
SECTIONS
    .data READ WRITE
```


Appendix

Version History

20.08.2003 (Build 1014)

- *.bld files are no longer used. Export as building option is deprecated.
- CryViewer.exe no longer exists
- Game and Preview buttons are no longer in use, now they both work as the 'Export Nodes' button.
- Cgfdump.exe tool added
- Optimization tools added
- Export as individual files option added
- New document created from original .chm document
- *.cga file type added
- Motion Optimiser and ResourceCompiler tools added

What's Changed after Build 744

- Every chunk descriptor structure now begins with with CHUNK_HEADER structure.
- CHUNK_HEADER structure is changed
- Every chunk in the file has now given a unique ID. Some chunks reference to other chunks with this id. for example node chunks reference the objects that this node has by giving the id of the chunk that the object is stored within.
- Every object in the exported scene is now described with two related chunks: a Node chunk and an object chunk. Node chunk holds the info such as positioning, name, user defined properties, material assignment etc.. Object Chunks (such as Mesh chunk, Light Chunk) on the other hand holds the info about the inner structure of the object. This enables exporting instanced geometry: one object chunk may be referenced/used by more than one node chunks in the scene.
- Node Property strings are now saved in the Node Chunks. not in the Mesh Chunks anymore.
- Names of the objects are now saved in the Node Chunks
- Positioning information are removed from all of the chunks in the cgf files.. They are stored in Node Chunks now.
- Timing information is removed from the vertex animation chunks. Timing chunks hold that info now.
- Bone Animation chunks no longer stores animation. They only store bone hierarchy.. Animations are stored in the Controller Chunks whose id are given in the BONE_ENTITY structures given in the bone hierarchy.
- light Chunks no longer have the positioning and name info.
- Geom Name Chunks are removed. the object names are stored in the Node Chunks for each object individually.
- Material List Chunks are removed. Material Chunks are doing the job for each material.
- Texture's UV coordinates and their animation are now exported.
- MAT_ENTITY structure is obsolete
- Controller chunks now define all of the key framed animations (except the vertex animation which is saved as used to be).

23.04.2001

Please check Changes section for changes that the engine programmers should know. The below part is a non-organized history notes that I kept during the development.

- Named animation sub-ranges can now be defined within the plug-in and exported with caf and cgf files.
- Time limiting function (Time Bomb) is removed.
- Vertex animation save progress is now shown in in the progress bar
- a bug fix in Vertex Animation export
- Exporting the vertex positions in "World Space" option is removed since the instanced object export making this impossible.
- Off-axis non-uniform scaling of the nodes are now detected and an optional warning is displayed in such cases.
- If Negative Scaling is detected in Node transformation, we do not correct the vertex orders anymore. This is because if we have two nodes using the same instanced mesh, and only one having neg scaling: this would cause problems.. So engine coders should take care of this.
- Texture UV coordinates and their animations are now exported.
- Timing chunk is added to both caf and cgf files.
- SceneProps chunk is added to the caf files also
- SceneProps chunk is corrected for wide character usage
- Timing information is removed from other chunks
- Node Animation Controllers are now exported
- Animation Controllers with 11 types of key info are now implemented as separate chunks
- Viewer is updated to read new caf and cgf files properly
- Node TM's are now exported as "relative to parent" instead of "absolute". This allows us to use group and hierarchy info of nodes to be exported.
- Nested Groups are exported.
- Nested materials can now be exported.
- Node hierarchy is imported properly
- Helper size type is changed from float to CryPoint3
- Materials are now saved in individual chunks and multi materials have links to chunks of the sub ones.
- nodes and objects are saved separately. nodes have references to the object, material chunks
- all of the chunks now have a copy of the chunk header (that is found in the chunk table)
- every chunk is given a chunk id
- All of chunks are now prepared before the exporting. this lets us to use chunk id's in other chunks as cross-reference to them.
- DummyChunk is removed. HelperChunk does the job

23.02.2001

- Patch support implementation is stared but unfortunately some control points are exported erroneously: I'll try to solve this problem.
- Patch Mesh Chunk is added.

- Since the patch support is (partly) implemented: now any patch object will be exported in a Patch Mesh Chunk rather than Mesh Chunk. If you want to export patch object as regular triangle meshes, you should convert it into mesh (you can do this by applying an edit mesh as the top-most modifier)

01.01.2001

- cgf file version is changed (you need to re-compile your codes using the new header)
- Vertex Colors are added to the Mesh Chunk.
- Vertex Color export option is added to the UI
- Multiple vertex color per vertex warning option is added.
- A severe bug in Unify_UV extension DLL is fixed.
- CryViewer now can display the objects in isometric display in arbitrary camera position.
- CryViewer mouse behaviors is changed: left drag: rotate, right drag: pan, left+right drag: zoom

29.11.2000

- Raw vertex animation chunks are implemented
- Load/Save configuration scheme is added
- AutoSave default config option is added
- Orphan tvert warning along (with other warnings) is now optional
- There can now be more than one Extension DLL's. [click here for details](#).
- Extension DLL exported functions now takes window handle for general usage
- Extension DLLs can now return error values.
- A workaround for MAX crash bug is found and implemented. Previously if you exit the plugin interface while picking is active it was crashing.
- A sample Extension DLL is created which splits vertices and texture vertices so that 1uv/vert criteria is satisfied.
- Some documentation mistakes are corrected.
- Command reference section is added to the documentation.

Complete Programmers Reference

Bone Animation Chunk.....	29
BONEANIM_CHUNK_DESC.....	29
BONE_ENTITY.....	29
Bone Name List Chunk.....	30
BONENAMELIST_CHUNK_DESC.....	30
Controller Chunk.....	30
CONTROLLER_CHUNK_DESC.....	31
Chunk Table.....	31
CryFace.....	32
CryIRGB.....	32
CryKey.....	32
CryLink.....	33
CryMatrix.....	33
CryPoint3.....	33
CryTexFace.....	34
CryTexPatch.....	34
CryUV.....	34
CryVertex.....	34
File Header.....	35
Geom Name List Chunk.....	35
GEOMNAMELIST_CHUNK_DESC.....	35
Helper Chunk.....	36
HELPER_CHUNK_DESC.....	36
KEY_HEADER.....	36
Light Chunk.....	36
LIGHT_CHUNK_DESC.....	36
Material List Chunk.....	38
MAT_ENTITY.....	38
MESH_CHUNK_DESC.....	39
Mesh Chunk.....	40
MRM_CHUNK_DESC.....	41
MRM Chunk.....	41
MTL_CHUNK_DESC.....	41
Material Chunk.....	42
MTLIST_CHUNK_DESC.....	43
NODE_CHUNK_DESC.....	43
Node Chunk.....	44
PATCHMESH_CHUNK_DESC.....	45
Patch Mesh Chunk.....	46
RANGE_ENTITY.....	47
SCENEPROPS_CHUNK_DESC.....	47
SCENEPROP_ENTITY.....	47
TextureMap.....	48
TIMING_CHUNK_DESC.....	49
Timing Chunk.....	49
VERTANIM_CHUNK_DESC.....	49
Vertex Animation Chunk.....	50

Bone Animation Chunk

This chunk type (ChunkType_BoneAnim) may be found in CAF files only and defines both the bone hierarchy.

Bone Anim Chunk Descriptor

This section is in the type of BONEANIM_CHUNK_DESC which is defined in "Headers.h" file.

Bone Hierarchy

In this section all of the bone structure is presented in hierarchical way such that parent bones are placed first followed the child bones (in recursive manner). So you can easily read this section in a recursive function this makes easy also the pointer assignments between the parents and children. Structure is schematically as follows (in the order of appearance in the file).

Bone 0 (root bone) definition

 Bone 0.0 definition

 Bone 0.0.0 definition

 Bone 0.0.1 definition

 Bone 0.0.2 definition

 Bone 0.1 definition

 Bone 0.2 definition

 Bone 0.2.0 definition

 Bone 0.2.1 definition

 Bone 0.2.1.0 definition

 Bone 0.2.1.1 definition

etc.

each bone definition is stored in BONE_ENTITY structure

BONEANIM_CHUNK_DESC

```
struct BONEANIM_CHUNK_DESC
{
    CHUNK_HEADER    chdr;
    int             nBones;
};
```

chdr

This is the copy of chunk header that is found in the Chunk Table. This gives basic information about the chunk type etc and useful for double checking.

nBones

Total number of bones that this skeleton has.

Note: Although the bones are stored in recursive hierarchic way and can be loaded in similar recursive manner without knowing the # of bones in the beginning, this value is useful if you want to pre-allocate memory for the skeleton.

BONE_ENTITY

```
struct BONE_ENTITY
{
    int    BoneID;
    int    ParentID;
    int    nChildren;
    int    ControllerID;
```

```

        char prop[32];
};

```

BoneID

Is the unique ID of the bone accros the file. This ID is used to match the corresponding Bone Name List Chunk entity.

ParentID

This is the ID of the parent of the bone. This ID is used to match the corresponding Bone Name List Chunk entity.

nChildren

Gives the number of children bones of this bone.

ControllerID

Gives the id of the controller chunk which defines the animation of this bone.

prop

This field hold the object's string entry given in 3D Studio MAX in "Object properties -> User defined" section. This may be used to tag the bones as special ones or give them some properties. The max size is 32 bytes.

Bone Name List Chunk

This chunk lists all of the bone names sorted in ascending order according to bone IDs. Names are the ones used in 3D Studio scene.

Bone Name List Chunk Descriptor

This field is in the type of BONENAMELIST_CHUNK_DESC.

Bone Names

This section hold bone names for all of the bones. There are BONENAMELIST_CHUNK_DESC.nEntities entities in this section each of which are in NAME_ENTITY type.

BONENAMELIST_CHUNK_DESC

```

struct BONENAMELIST_CHUNK_DESC
{
    CHUNK_HEADER      chdr;
    int                nEntities;
};

```

chdr

This is the copy of chunk header that is found in the Chunk Table. This gives basic information about the chunk type etc and useful for double checking.

nEntities

Gives number of bone name entities in the list

Controller Chunk

This chunk type (ChunkType_Controller) may be found in CGF and CAF files and defines the key framed animation of any animatable quantity. Controllers can define animation for floating point values, 3D Point values or Quaternion values.. Linear, Bezier and TCB (Hermite) interpolation is supported.

Controller Chunk Descriptor

Holds the basic properties of the controller being described. This section is in the type of CONTROLLER_CHUNK_DESC which is defined in "Headers.h" file.

Keys

This section holds the actual keys that this controller has. the number of the keys are given in the are CONTROLLER_CHUNK_DESC structure's nKeys field. The keys are of the corresponding key type of CONTROLLER_CHUNK_DESC's type field.

CONTROLLER_CHUNK_DESC

```
struct CONTROLLER_CHUNK_DESC
{
    CHUNK_HEADER      chdr;
    CtrlTypes         type;        // one of the CtrlTypes values
    int               nKeys;       // # of keys this controller has
    int               nSubCtrl;    // # of sub controllers
};
```

chdr

This is the copy of chunk header that is found in the Chunk Table. This gives basic information about the chunk type etc and useful for double checking.

type

one of the CtrlTypes values

```
enum CtrlTypes
{
    CTRL_NONE,
    CTRL_CRYBONE,
    CTRL_LINEER1, CTRL_LINEER3, CTRL_LINEERQ,
    CTRL_BEZIER1, CTRL_BEZIER3, CTRL_BEZIERQ,
    CTRL_TCB1, CTRL_TCB3, CTRL_TCBQ
};
```

The naming convention is that LINEER, BEZIER and TCB indicates the interpolation method while the 1,3 and Q suffix indicates the values as float, 3D point and Quaternion respectively.

nKeys

number of the keys this controller has..

nSubCtrl

of sub controller this controller may have.. Currently this value is not used and always 0

Chunk Table

This table is located at the end of the file and lists all of the chunks that the file holds. The location of this table is given in the header of each file.

Table Sections

nChunks

Chunk Item 1

Chunk Item 2

Chunk Item 3

...

Chunk Item [nChunks]

nChunks

This integer value gives the number of chunks that the table (therefore the file) holds

Chunk Items

Chunks Items are of the type CHUNK_HEADER which is defined in "Headers.h" file

CryFace

```
struct CryFace
{
    int v0,v1,v2;        //vertex indices
    int MatID;          //mat ID
    int SmGroup;        //smoothing group
};
```

v0, v1, v2

These give the indices to the vertices (as appered in Vertices field in the Mesh Chunk) in the right hand order.

MatID

This gives the material ID of the face. If the object has "multi/sub object" material applied, each face may have differnt matID's. In this every sub material is listed in the Material List Chunk as separate entities (the main material is also given in the list).

SmGroup

This gives the smoothing group bit fields of the face.

CryIRGB

```
struct CryIRGB
{
    unsigned char r,g,b;
};
```

r, g, b

color components in 0..255 range

CryKey

```
struct CryKey
{
    int                time;           //in ticks
    //CryMatrix        absmat;         //in world coordinates
    CryPoint3          abspos;         //absolute position;
    //CryQuat          absquat;        //absolute quaternion
    //CryMatrix        relmat;         //relaive to Parent
    CryPoint3          relpos;         //relative position;
    CryQuat            relquat;        //relative quaternion
};
```

Currently there are many redundant data in this field which may help the programmer to check the correctness of his calculation.

time

gives the key time in ticks. refer to Bone Animation Chunk's chunk descriptor section ofr various timing conversion constants.

absmat (removed)

4x4 Trasformation matrix in CryMatrix type at the key time relative to World Coordinate System

abspos

x,y,z position in CryPoint3 type at the key time of the bone relative to World Coordinate System

absquat (removed)

Quaternion in CryQuat type at the key time of the bone relative to World Coordinate System

relmat (removed)

4x4 Trasformation matrix in CryMatrix type at the key time relative to parent bone

relpos

x,y,z position in CryPoint3 type at the key time of the bone relative to parent bone

relquat

Quaternion in CryQuat type at the key time of the bone relative to parent bone

CryLink

```
struct CryLink
{
    int          BoneID;
    CryPoint3    offset;
    float        Blending;
}
```

BoneID

This field gives the ID of the bone associated with this link. This id is used to locate the bone name listed in the Bone Name List Chunk.

offset

This offset vector gives the position of the vertex in the local coordinates of the bone (relative to the bone).

Blending

This value gives the affect weight of the bone for the given vertex. The sum of the weigts for a vertex is always 1.0

CryMatrix

```
struct CryMatrix
{
    float        data[16];    //in GL format
};
```

data

This defines the matrix in OpenGL ordering

CryPoint3

```
struct CryPoint3
{
    float x,y,z;
};
```

x,y,z

cartesian positions

CryQuat

```
struct CryQuat
{
    float x,y,z,w;
};
```

x,y,z

Imaginary part of the Quaternion

w

Real part of the Quaternion??

CryTexFace

```
struct CryTexFace
{
    int t0,t1,t2;    //texture vertex indices
};
```

t0, t1, t2

These give the indices to the vertices (as appered in Texture Vertices field in the Mesh Chunk). There are the same number of Texture Faces with the actual (geometric) Faces that correspond to each ther.

CryTexPatch

```
struct CryTexPatch
{
    int t0,t1,t2,t3; //texture vertex indices
};
```

t0, t1, t2,t3

These give the indices to the vertices (as appered in Texture Vertices field in the Mesh Chunk). There are the same number of Texture Patches with the actual (geometric) Patches. Texture coordinates are stored only for the corner vertices of patches texture index vertex index correspondance is as follows

t0: v[0] of patch

t1: v[3] of patch

t2: v[6] of patch

t3: v[9] of patch (quad patches only)

CryUV

```
struct CryUV
{
    float u,v;    //texture coordinates
};
```

u,v

gives the texture mapping coordinates.

CryVertex

```
struct CryVertex
{
    CryPoint3 p;    //position
    CryPoint3 n;    //normal vector
};
```

p

This gives the position of the vertex (either in world space or in object space depending on the InWorldSpace field of Mesh Chunk Descriptor section or Vertex Anim Chunk Descriptor section)

n

This is the calculated normal for the vertex (for the pose of the object at the time it is exported).

File Header

File header consists of one FILE_HEADER structure which is defined in the "Headers.h" file.

```
struct FILE_HEADER
{
    char        Signature[7];
    FileTypes   FileType;
    int         Version;
    int         ChunkTableOffset;
};
```

Signature

should hold the constant FILE_SIGNATURE defined in the "Headers.h" file. You can test if a file is caf or cgf file by checking this value.

The definition of FILE_SIGNATURE is:

```
#define FILE_SIGNATURE "CryTek"
```

FileType

This value indicates the if the file is CAF or CGF. possible values are

```
enum FileTypes
{
    FileType_Geom = 0xFFFF0000,
    FileType_Anim,
};
```

Version

This holds the version of the file. This should match the version constants defined in "Headers.h" file. Current values are defined as

```
#define GeomFileVersion      0x????
#define AnimFileVersion      0x????
```

ChunkTableOffset

This integer gives the offset position of the Chunk Table

Geom Name List Chunk

Important: This chunk is removed after plug-in Build 744

This chunk lists all of the mesh object names sorted in ascending order according to bone IDs. Names are the ones used in 3D Studio scene.

Geom Name List Chunk Descriptor

This field is in the type of GEOMMELIST_CHUNK_DESC.

Geom Names

This section hold names for all of the meshes. There are GEOMNAMELIST_CHUNK_DESC.nEntities entities in this section each of which are in NAME_ENTITY type.

GEOMNAMELIST_CHUNK_DESC

```
struct GEOMNAMELIST_CHUNK_DESC
{
    int    ChunkType; //must be ChunkType_GeomNameList
    int    ChunkVersion;
    int    nEntities;
};
```

ChunkType

Should be ChunkType_GeomNameList

ChunkVersion

Should be GEOMNAMELIST_CHUNK_DESC_VERSION

nEntities

Gives number of bone name entities in the list

Helper Chunk

This chunk type (ChunkType_Helper) may be found in CGF files only and defines the parameters of helper objects that are exported. There are separate chunk for each exported helper object.

Helper Chunk Descriptor

Holds all of the properties of the helper object being described. This section is in the type of HELPER_CHUNK_DESC which is defined in "Headers.h" file.

HELPER_CHUNK_DESC

```
struct HELPER_CHUNK_DESC
{
    CHUNK_HEADER      chdr;
    HelperTypes       type;           // one of the HelperTypes values
    CryPoint3         size;           // size in local x,y,z axes (for dummy only)
};
```

chdr

This is the copy of chunk header that is found in the Chunk Table. This gives basic information about the chunk type etc and useful for double checking.

type

One of the following values indicating the type of the helper:

```
enum HelperTypes { HP_POINT, HP_DUMMY};
```

size

This holds the bounding box size of the Dummy objects. For Point helpers this is {0,0,0}

KEY_HEADER

```
struct KEY_HEADER
{
    int    KeyTime;    //in ticks
};
```

KeyTime

Gives the time of the key in Ticks. For tick, frame and second conversion constanst please see VERTANIM_CHUNK_DESC structure's SecsPerTick and TicksPerFrame fields.

Light Chunk

This chunk type (ChunkType_Light) may be found in CGF files only and defines the positions and parameters of lights that are exported. There are separate chunk for each exported light.

Light Chunk Descriptor

Holds all of the properties of the light being described. This section is in the type of LIGHT_CHUNK_DESC which is defined in "Headers.h" file.

LIGHT_CHUNK_DESC

```
struct LIGHT_CHUNK_DESC
```

```

{
    CHUNK_HEADER      chdr;
    LightTypes        type;           // one of the LightTypes values
    bool              on;             // light is on
    CryIRGB           color;         // RGB color
    float             intens;        // multiplier value
    float             hotsize;       // for spot and direct lights hotspot value
    float             fallsize;     // for spot and direct lights falloff value
    bool              useNearAtten;  // near atten. on
    float             nearAttenStart; // near attenuation ranges
    float             nearAttenEnd;
    bool              useAtten;      // far atteniation
    float             attenStart;    // far attenuation ranges
    float             attenEnd;
    bool              shadow;        // shadow is on
};

```

```
struct LIGHT_CHUNK_DESC
```

```

{
    int               ChunkType;      //must be ChunkType_Light
    int               ChunkVersion;
    char              name[64];      // light name
    int               type;          // one of the LightTypes values
    BOOL              on;            // light is on
    CryPoint3         pos;           // world position
    CryMatrix         tm;            // transformation matrix
    CryIRGB           color;         // RGB color
    float             intens;        // multiplier value
    float             hotsize;       // for spot and direct lights hotspot value
    float             fallsize;     // for spot and direct lights falloff value
    bool              useNearAtten;  // near atten. on
    float             nearAttenStart; // near attenuation ranges
    float             nearAttenEnd;
    bool              useAtten;      // far atteniation
    float             attenStart;    // far attenuation ranges
    float             attenEnd;
    bool              shadow;        // shadow is on
};

```

chdr

This is the copy of chunk header that is found in the Chunk Table. This gives basic information about the chunk type etc and useful for double checking.

type

One of the following values indicating the type of the light:

```
enum LightTypes { LT_OMNI, LT_SPOT, LT_DIRECT, LT_AMBIENT };
```

LT_OMNI	Omni Light
LT_SPOT	Spot Light
LT_DIRECT	Direct Light
LT_AMBIENT	Ambient Light (not used)

on

True if the light is active. False if it is turned off.

color

Color of the light in RGB.

intens

Intensity of the light. This value is the strenght multiplier. Usual value is 1.0

hotsize

For spot light gives the hotspot angle. For directional lights gives the hotspot radius in world units

fallsize

For spot light gives the falloff angle. For directional lights gives the falloff radius in world units

useNearAtten

If true light has near attenuation turned on and the nearAttenStart and nearAttenEnd fields are active.

nearAttenStart

Near attenuation start range. This value is not used if useNearAtten is false.

nearAttenEnd

Near attenuation end range. This value is not used if useNearAtten is false.

useAtten

If true light has far attenuation turned on and the AttenStart and AttenEnd fields are active.

attenStart

Far attenuation start range. This value is not used if useNearAtten is false.

attenEnd

Far attenuation end range. This value is not used if useNearAtten is false.

shadow

If true light casts shadow. If false shadow casting is turned off.

Material List Chunk

Important: This chunk is removed after plug-in Build 744

This chunk lists all of the materials that are used by the exported meshes.

Material List Chunk Descriptor

This field is in the type of MTLIST_CHUNK_DESC.

Materials

This section holds a list of materials. The size of the list is MTLIST_CHUNK_DESC.nEntities. Each of these entities are in the type of MAT_ENTITY.

MAT_ENTITY

Important: This structure is removed after plug-in Build 744

```
struct MAT_ENTITY
{
    char        name[64];           //material name
    bool        IsStdMat;          //if false the below values are meaningless
    CryIRGB    col_d;              //diffuse color
    CryIRGB    col_s;              //specular color
    CryIRGB    col_a;              //ambient color
    char        map_d[32];         //diffuse map file name
}
```

```

    char    map_o[32];        //opacity map file name
    char    map_b[32];        //bump map file name
    float   Dyn_Bounce;
    float   Dyn_StaticFriction;
    float   Dyn_SlidingFriction;
};

```

name

This is the name of the material as used in 3D Studio MAX.

IsStdMat

This field indicated if this material is a standard material. If the material is not a standard material, then the below parameters are not used. This occurs, for example, for "multi/sub object materials" used for the meshes.

col_d, col_s, col_a

Diffuse, Specular and Ambient color each of CryIRGB type.

map_d, map_o, map_b

These strings hold the file path and name of the Diffuse, Opacity and Bump maps if used in the material.

Dyn_Bounce, Dyn_StaticFriction, Dyn_SlidingFriction

These are the "dynamics properties" which are entered in the "Materials Editor" of 3DS MAX.

MESH_CHUNK_DESC

```

struct MESH_CHUNK_DESC
{
    CHUNK_HEADER    chdr;
    bool            HasBoneInfo;
    bool            HasVertexCol;
    int             nVerts;
    int             nTVerts;    // # of texture vertices
    int             nFaces;
    int             VertAnimID; // id of the related vertAnim chunk if present.
                                //otherwise it is -1
};

```

chdr

This is the copy of chunk header that is found in the Chunk Table. This gives basic information about the chunk type etc and useful for double checking.

HasBoneInfo

Indicates if the chunk includes vertex-bone link data.

HasVertexCol

Indicates if the chunk includes vertex colors. Vertex colors are organized such that only one vertex color per vertex exists. If the original mesh has more than one vertex color assigned for a vertex, and the user ignored the warning dialog that the plug-in displays, the average color will be written to the cgf file for that vertex.

nVerts

Gives the number of vertices that the mesh has.

nTVerts

Gives the number of texture vertices that the mesh has. This value can differ from the nVerts value only if the "Allow Multi UV/vert" option of the plug-in is turned on and the mesh has multiple tverts for each geometric vertex.

nFaces

Gives the number of faces that this mesh has.

VertAnimID

This is the id of the related vertAnim chunk of this mesh object if present. otherwise it is -1

Mesh Chunk

This chunk type (ChunkType_Mesh) may be found in CGF files only and defines the geometry, topology, material, and texture coordinates of the each exported mesh object from 3D Studio MAX. Each cgf file may include one or more Mesh Chunks.

Chunk Sections

Mesh Chunk Descriptor

Vertices

Faces

*Texture Vertices if (header.nTVerts != 0)

*Texture Faces if ((header.nTVerts != 0) && (header.nTVerts != header.nVerts))

*Phsyique Data if (header.HasBoneInfo == true)

*Vertex Colors if (header.HasVertexCol == true)

(*) optional data fields (the condition is given next to item)

Mesh Chunk Descriptor

Holds the basic properties of the mesh being described. This section is in the type of MESH_CHUNK_DESC which is defined in "Headers.h" file.

Vertices

this section holds CryVertex data for each vertex. The number of vertices is given in Mesh Chunk Descriptor section's nVerts field.

Faces

this section holds CryFace data for each face. The number of vertices is given in Mesh Chunk Descriptor section's nFaces field.

Texture Vertices

This section exists only if the nTVerts fields of Mesh Chunk Descriptor section is nonzero.

This section holds CryUV data for each texture vertex. The number of vertices is given in Mesh Chunk Descriptor section's nTVerts field.

Texture Faces

This section exists only if the nTVerts field of Mesh Chunk Descriptor section is nonzero and the nVerts and nTVerts fields are different.

this section holds CryTexFace data for each texture face. The number of texture faces are the same as the number of faces which is given in Mesh Chunk Descriptor section's nFaces field.

Phsyique Data

This section exists only if the HasBoneInfo field of Mesh Chunk Descriptor section is true.

This section holds data for each vertex (that is nVerts vertices) that defines the vertex-bone connection. Each vertex may have 1 one or more links to the bones. So the sub-structure of this section for each vertex is as follows

nLinks

Link Info 0

Link Info 1
 Link Info 2
 ...
 Link Info [nLinks-1]

"nLinks" is in integer type and gives the number of bone connections this vertex has

Each "Link Info" is in CryLink type and defines the nth bone link.

To calculate the deformed position of the vertex use the below formula

$$p' = w1 * TM1 * ov1 + w2 * TM2 * ov2 + w3 * TM3 * ov3 + \dots$$

where:

w: weigh of the bone given in CryLink Blending field

TM: Updated transformation matrix of the bone

ov: Offset of the vertex relative to the bone given in CryLink offset field

Vertex Colors

This section exists only if the HasVertexCol fields of Mesh Chunk Descriptor section is nonzero.

This section holds CryIRGB data for each texture vertex. The number of vertices is given in Mesh Chunk Descriptor section's nTVerts field. If the original mesh has more than one vertex color assigned for a vertex, their average color will be written to the cgf file for that vertex.

MRM_CHUNK_DESC

```
struct MRM_CHUNK_DESC
{
    CHUNK_HEADER    chdr;
    int             GeomID;    //id of the related mesh chunk
};
```

chdr

This is the copy of chunk header that is found in the Chunk Table. This gives basic information about the chunk type etc and useful for double checking.

GeomID

Each mesh listed in the file is given a id number started from 0 in the order of the apperiances of the meshes in the file. This ID can be used for cross references after the info is loaded into the memory.

MRM Chunk

This chunk type (ChunkType_MRM) may be found in CGF files only and defines the Multi Resolution Mesh data for each exported mesh object from 3D Studio MAX. Each cfg file may include one or more MRM Chunks.

MRM Chunk Descriptor

Holds the basic properties of the mesh being described. This section is in the type of MRM_CHUNK_DESC which is defined in "Headers.h" file.

MTL_CHUNK_DESC

```
struct MTL_CHUNK_DESC
{
    CHUNK_HEADER    chdr;
    char            name[64];    //material name
    MtlTypes        MtlType;    //one of the MtlTypes enum values
};
```

```

union
{
    struct
    {
        CryIRGB        col_d;           //diffuse color
        CryIRGB        col_s;           //specular color
        CryIRGB        col_a;           //ambient color

        TextureMap tex_d; //Diffuse Texture settings
        TextureMap tex_o; //opacity Texture settings
        TextureMap tex_b; //Bump Texture settings

        float          Dyn_Bounce;
        float          Dyn_StaticFriction;
        float          Dyn_SlidingFriction;
    } std;

    struct
    {
        int            nChildren;
    } multi;
};
};

```

chdr

This is the copy of chunk header that is found in the Chunk Table. This gives basic information about the chunk type etc and useful for double checking.

name

Name of the materials as appears in the MAX scene

MtlType

One of the MtlTypes enum values

```
enum MtlTypes { MTL_UNKNOWN, MTL_STANDARD, MTL_MULTI, MTL_2SIDED};
```

col_d, col_s, col_a

Diffuse, Ambient and Specular color values of the material as set in the MAX scene. values are in CryIRGB type.

tex_d, tex_o, tex_b

Holds settings of Diffuse, Opacity and Bump textures that this material may have. If the related texture is not set in the material then the name field of the texture is empty. These values are in the TextureMap type.

Dyn_Bounce, Dyn_StaticFriction, Dyn_SlidingFriction

These are the "dynamics properties" which are entered in the "Materials Editor" of 3DS MAX.

nChildren

if the MtlType is MTL_MULTI, then this field gives the # of sub-materials.

Material Chunk

This chunk type (ChunkType_Node) may be found in CGF files only and defines the position, name, material, user defined properties of the nodes found 3D Studio MAX scenes. Each cgf file may include one or more Node Chunks.

Chunk Sections

Mtl Chunk Descriptor

*Sub Material id's if (header.nChildren != 0)

(*) optional data fields (the condition is given next to item)

Mtl Chunk Descriptor

Holds the basic properties of the material being described. This section is in the type of MTL_CHUNK_DESC which is defined in "Headers.h" file.

Sub Material ID's

This section holds the list of integers which define the chunk id's of the sub materials that this node has. number of sub materials can be found in the MTL_CHUNK_DESC structure's nChildren field

MTLLIST_CHUNK_DESC

```
struct MTLLIST_CHUNK_DESC
{
    int          ChunkType;          //must be ChunkType_MtlList
    int          ChunkVersion;
    int          nEntities;
};
```

ChunkType

Should be ChunkType_MtlList

ChunkVersion

Should be MTLLIST_CHUNK_DESC_VERSION

nEntities

Gives number of material entities stored in the list

NODE_CHUNK_DESC

```
struct NODE_CHUNK_DESC
{
    CHUNK_HEADER    chdr;

    char            name[64];

    int ObjectID; // ID of this node's object chunk (if present)
    int            ParentID;          // chunk ID of the parent Node's chunk
    int            nChildren;        // # of children Nodes
    int            MatID;            // Material chunk No

    bool           IsGroupHead;
    bool           IsGroupMember;

    CryMatrix      tm;              // transformation matrix
    CryPoint3      pos;             // pos component of matrix
    CryQuat        rot;             // rotation component of matrix
    CryPoint3      scl;             // scale component of matrix

    int            pos_cont_id;      // position controller chunk id
    int            rot_cont_id;      // rotation controller chunk id
    int            scl_cont_id;      // scale controller chunk id
};
```

```

        int          PropStrLen;    // lenght of the property string
};

```

chdr

This is the copy of chunk header that is found in the Chunk Table. This gives basic information about the chunk type etc and useful for double checking.

name

Name of the node as appears in the MAX scene

ObjectID

id of the chunk that holds the object information this node refers to.. Note that each object chunk may be used by one or more node chunks.

ParentID

id of the chunk that holds the parent node of this node. If the node has no parent (i.e. root level node) then this value is -1

nChildren

number of child nodes that this node has.. in no children case this value is 0.

MatID

id of the chunk that holds the material information of this node. If the node has no material associated with then this value is -1

IsGroupHead

if the node is a head node of a group, then this value is non-zero

IsGroupMember

if the node belongs to a group, this value is non-zero. you can use the group head node of the group by treversing "up" in the hierarchy until reaching a node with IsGroupHead flag set.

tm

Transformation matrix of the node. Note that this matrix is relative to the parent node if present. if there is no parent node, then tm is the "absolute" transformation matrix.

pos, rot, scl

Default potion (translation), rotation and scaling parts of tm.. These are also relative to the parent. If pos_cont_id, rot_cont_id, scl_cont_id values are differnt from -1, then the animated position, rotation or scaling values are defined by the controller chunks.

pos_cont_id, rot_cont_id, scl_cont_id

these are the id of the chunks that defines the animation of the node. If the node's related channel has no animation the related value is -1. in this case values defined in the pos, rot, scl fields should be used.

PropStrLen

This is the length of the property string of the node.. If the node has no property string defined by the user this value is 0.

Node Chunk

This chunk type (ChunkType_Node) may be found in CGF files only and defines the position, name , material, user defined properties of the nodes found 3D Studio MAX scenes. Each cgf file may include one or more Node Chunks.

Chunk Sections

Node Chunk Descriptor

*Property String if (header.PropStrLen!= 0)

*Children Node id's if (header.nChildren != 0)

(*) optional data fields (the condition is given next to item)

Node Chunk Descriptor

Holds the basic properties of the node being described. This section is in the type of NODE_CHUNK_DESC which is defined in "Headers.h" file.

Property String

This section holds the property string that the user defined in the MAX scene. length of the string is given in NODE_CHUNK_DESC structure's PropStrLen field.

Children Node ID's

This section holds the list of integers which define the chunk id's of the children node's that this node. number of children can be found in the NODE_CHUNK_DESC structure's nChildren field

PATCHMESH_CHUNK_DESC

```
struct PATCHMESH_CHUNK_DESC
{
    int    ChunkType;           //must be ChunkType_Patch
    int    ChunkVersion;
    int    GeomID;
    bool   HasBoneInfo;
    bool   InWorldSpace;
    int    nVerts;
    int    nTVerts;           // # of texture vertices
    int    nPatches;
    int    MatID;
    int    PropStrLen;       //length of the property string
};
```

ChunkType

Should be ChunkType_PatchMesh

ChunkVersion

Should be PATCHMESH_CHUNK_DESC_VERSION

GeomID

Each mesh or patch mesh listed in the file is given a id number started from 0 in the order of the appearances of the meshes in the file. This ID can be used for cross references after the info is loaded into the memory. See GeomNameList chunk for getting the names of the meshes.

HasBoneInfo

Indicates if the chunk includes vertex-bone link data.

InWorldSpace

Indicates is the vertex positions are given in the world coordinate system (space) or in the object space. The plug-in has a check box which defines this behaviour.

nVerts

Gives the number of vertices that the mesh has.

nTVerts

Gives the number of texture vertices that the patch mesh has.

nPatches

Gives the number of patches that this patch mesh has.

MatID

Gives the ID of the object's material. This ID is used to locate the material listed in the Material List Chunk. This value is -1 if the object has no material applied on.

PropStrLen

This field gives the length (in bytes) of the Property String section in the Patch Mesh Chunk.

Patch Mesh Chunk

Important: This chunk is not implemented fully and the source code is commented out

This chunk type (ChunkType_PatchMesh) may be found in CGF files only and defines the geometry, topology, material, and texture coordinates of the each exported patch object from 3D Studio MAX. Each cgf file may include one or more Patch Mesh Chunks. Each patch mesh chunk includes one or more patches (very similar to triangle faces of a polygon mesh). Patches themselves may be QuadPatch or TriPatch.

Chunk Sections

PatchMesh Chunk Descriptor

*Property String if (header.PropStrLen > 0) Vertices

Patches

*Texture Vertices if (header.nTVerts != 0)

*Texture Patches if (header.nTVerts != 0)

*Phsyique Data if (header.HasBoneInfo == true)

(*) optional data fields (the condition is given next to item)

Mesh Chunk Descriptor

Holds the basic properties of the patch mesh being described. This section is in the type of PATCHMESH_CHUNK_DESC which is defined in "Headers.h" file.

Property String

This field hold the objects string entry given in 3D Studio MAX in "Object properties -> User defined" section. The length of this string is given in the PatchMesh Chunk Descriptor section.

Vertices

this section holds CryPoint3 data for each vertex. The number of vertices is given in PatchMesh Chunk Descriptor section's nVerts field.

Patches

This section holds control vertex indices of each patch. Every patch data begins with an integer which defines the type of the Patch (either TRI_PATCH or QUAD_PATCH). After the patch type, there is either a CryTriPatch or a CryQuadPatch structure. The number of patches is given in PatchMesh Chunk Descriptor section's nPatches field.

Texture Vertices

This section exists only if the nTVerts fields of PatchMesh Chunk Descriptor section is nonzero.

This section holds CryUV data for each texture vertex. The number of vertices is given in PatchMesh Chunk Descriptor section's nTVerts field.

Texture Patches

This section exists only if the nTVerts field of PatchMesh Chunk Descriptor section is nonzero.

this section holds CryTexPatch data for each texture face. The number of texture faces are the same as the number of patches which is given in PatchMesh Chunk Descriptor section's nPatches field.

Phsyique Data

This section exists only if the HasBoneInfo field of PatchMesh Chunk Descriptor section is true.

This section holds data for each vertex (that is nVerts vertices) that defines the vertex-bone connection. Each vertex may have one or more links to the bones. So the sub-structure of this section for each vertex is as follows

```
nLinks
Link Info 0
Link Info 1
Link Info 2
...
Link Info [nLinks-1]
```

"nLinks" is in integer type and gives the number of bone connections this vertex has

Each "Link Info" is in CryLink type and defines the nth bone link.

To calculate the deformed position of the vertex use the below formula

$$p' = w1*TM1*ov1 + w2*TM2*ov2 + w3*TM3*ov3 + \dots$$

where:

w: weigh of the bone given in CryLink Blending field

TM: Updated transformation matrix of the bone

ov: Offset of the vertex relative to the bone given in CryLink offset field

RANGE_ENTITY

```
struct RANGE_ENTITY
{
    char name[32];
    int start;
    int end;
};
```

name

this is the name of the range as given by the user.

start, end

These are the starting and ending frames that define the range. These values are in frames (not in ticks).

SCENEPROPS_CHUNK_DESC

```
struct SCENEPROPS_CHUNK_DESC_0744
{
    CHUNK_HEADER    chdr;
    int              nProps;
};
```

chdr

This is the copy of chunk header that is found in the Chunk Table. This gives basic information about the chunk type etc and useful for double checking.

nProps

Gives number of properties stored in the list

SCENEPROP_ENTITY

```
struct SCENEPROP_ENTITY
{
```

```

    char    name[32];
    char    value[64];
};

```

name

name of the property as given in MAX

value

Value of the property as given in MAX. User should define only "Text" type properties

TextureMap

```

struct TextureMap
{
    char    name[32];

    //tiling and mirror
    bool    utile;
    bool    umirror;
    bool    vtile;
    bool    vmirror;

    //texture position values
    float   uoff_val;
    float   uscl_val;
    float   urot_val;
    float   voff_val;
    float   vscl_val;
    float   vrot_val;
    float   wrot_val;

    //texture position controller chunk id's (if not animated they are -1)
    int     uoff_ctrlID;
    int     uscl_ctrlID;
    int     urot_ctrlID;
    int     voff_ctrlID;
    int     vscl_ctrlID;
    int     vrot_ctrlID;
    int     wrot_ctrlID;
};

```

name

File name of the bitmap texture. This is only the file name and does not contain file path as in MAX scene

utile, umirror, vtile, vmirror

These boolean values indicates whether the tiling and mirror settings of u and v directions of the texture are set in the MAX's material editor for this texture.

uoff_val, uscl_val, urot_val

holds the offset, scale and rotation values in the u direction as given in MAX's material editor

voff_val, vscl_val, vrot_val

holds the offset, scale and rotation values in the v direction as given in MAX's material editor

wrot_val

holds the rotation value in the w direction as given in MAX's material editor

uoff_ctrlID, uscl_ctrlID, urot_ctrlID, voff_ctrlID, vscl_ctrlID, vrot_ctrlID, wrot_ctrlID

ID's of the controller chunks for offset, scaling and rotation animations of u, v, and w axis' of the texture. If the related channel has no animation, it'll have the value of -1

TIMING_CHUNK_DESC

```
struct TIMING_CHUNK_DESC
{
    CHUNK_HEADER    chdr;

    float           SecsPerTick;           // seconds/ticks
    int             TicksPerFrame;        // ticks/Frame

    RANGE_ENTITY    global_range; // covers all of the time ranges
    int             nSubRanges;
};
```

chdr

This is the copy of chunk header that is found in the Chunk Table. This gives basic information about the chunk type etc and useful for double checking.

SecsPerTick

The timing unit used in the caf and cgf files are usually in sub-frame unit called Tick. SecsPerTick value of this structure defines the physical time of a single Tick in seconds.

TicksPerFrame

This value also defines the tick in terms of the ratio of a frame time. that is how many ticks is equal to a frame time. This is useful for converting the frame values into ticks and vice-versa.

global_range

This give the time range for which the animations is exported for bones and vertex animations. Since the key framed animation controllers other than the bone animation controllers are not re-sampled, they are exported till the last keys they contain. So this value is only for Bone animations and vertex animations.

nSubRanges

Gives number of sub named ranges that the user entered using the plug-in's Sub Range editor. Sub ranges are stored in the file jus after this chunk in RANGE_ENTITY structures.

Timing Chunk

This chunk type (ChunkType_Timing) may be found in CGF and CAF files and defines the animation ranges. "Names sub ranges" edited by the user are also stored in this chunk

Timing Chunk Descriptor

Holds the basic properties of the controller being described. This section is in the type of TIMING_CHUNK_DESC which is defined in "Headers.h" file.

Named Sub Ranges

This section holds the named Sub animation ranges that the user entered in the plug-in's sub range editor. The number of the sub ranges are given in TIMING_CHUNK_DESC structure's nSubRanges field. Each sub range is stored in RANGE_ENTITY structure..

VERTANIM_CHUNK_DESC

```
struct VERTANIM_CHUNK_DESC
{
```

```

CHUNK_HEADER    chdr;

int GeomID; // ID of the related mesh chunk
int      nKeys; // # of keys
int nVerts; // # of vertices this object has
int nFaces; // # of faces this object has (for double check purpose)
};

```

chdr

This is the copy of chunk header that is found in the Chunk Table. This gives basic information about the chunk type etc and useful for double checking.

GeomID

id of the Mesh Chunk that holds the static mesh information for this animated mesh.

nKeys

Gives the number of keys that this chunk contains.

nVerts

Gives the number of vertices of the object.

nFaces

Gives the number of faces of the object.

Vertex Animation Chunk

This chunk type (ChunkType_VertAnim) may be found in CGF files only and defines the vertex animation by giving the coordinates of the vertices at regular time intervals.

Vertex Anim Chunk Descriptor

Holds the basic properties of the mesh being described. This section is in the type of VERTANIM_CHUNK_DESC which is defined in "Headers.h" file.

Vertex Animation Keys

This section contains the one or many vertex animation keys. The number of keys can be found in nKeys field of the VERTANIM_CHUNK_DESC structure given just before this section. Each key begins with a key header which is in KEY_HEADER type. After the "Key Header"s vertex positions for all the vertices of the object will be listed. The number of vertices listed in each key can be found in nVerts field of the VERTANIM_CHUNK_DESC. The vertices are listed in vertex number order beginning with vertex 0 and ending with vertex[nVerts-1]. The positions of the vertices are in type of CryVertex. Visually the "Vertex Animation Keys" section structure is as follows:

```

Key (0)
    Key Header for Key 0 (in KEY_HEADER type)
    Vertex(0) position (in CryPoint3 type)
    Vertex(1) position (in CryPoint3 type)
    ...
    Vertex(nVerts-1) position (in CryPoint3 type)
Key(1)
    Key Header for Key 1 (in KEY_HEADER type)
    Vertex(0) position (in CryPoint3 type)
    Vertex(1) position (in CryPoint3 type)
    ...
    Vertex(nVerts-1) position (in CryPoint3 type)
Key(2)

```

...

Key(nKeys-1)

Key Header for Key 1 (in KEY_HEADER type)

Vertex(0) position (in CryPoint3 type)

Vertex(1) position (in CryPoint3 type)

...

Vertex(nVerts-1) position (in CryPoint3 type)