VitaBoy Documentation

By Don Hopkins, Maxis.

This document describes VitaBoy, the skeletal character animation system in The Sims, written by Don Hopkins at Maxis.

VitaBoy combines several different types of data together to render the animated characters in the game, including skeletons, skills, suits and texture maps.

Artists create the skeletons, skills and suits in 3D Studio Max, and the texture maps in Photoshop.

The CMX Exporter is a 3D Studio Max plug-in and MaxScript user interface, which allows artists to export skeletons, skills and suits from Max files into CMX files that the game can read.

Character Studio is another 3D Studio Max plug-in, that allows artists to animated a Biped skeleton, and to attach deformable mesh suits to it with Physique. The CMX Exporter knows how to support Character Studio Biped and Physique, but it can be used with other kinds of skeletons and suits as well.

The way the CMX Exporter knows what to export from a Max file, is by looking for note tracks on the bones, for keys containing tags that control the exporter. The artist inserts note track keys into the Max file, to mark up the skeletons, suits, skills and events. The tags in the note track keys tell the exporter what to export from the Max file.

The Access database tells the exporter which skeletons, skills and suits are defined, which Max files contain them, and where to export them. The artist can select the name of a skeleton, skill or suit from a scrolling list, and automatically load, validate and export the correct Max file to the correct destination. The exporter can also check the exported files out from and into SourceSafe. The artist can use the exporter manually without the database, but the database is extremely useful for avoiding accidents when there is a lot of content to manage.

- Installing the CMX Exporter.
  - The "official" distribution site of the CMX exporter is \\Elmo\ctgr\pub\DIST\MaxScript CMX Exporter.
  - The CMX Exporter is implemented as a MaxScript plug-in (maxiscrp.dlx) and a MaxScript startup script (maxis-maxscript.ms). It's not a normal 3D Studio Max "exporter" plug-in, so you will not find it on the File/Export menu, instead it has its own user interface utility panel.
  - To install the CMX exporter, first quit Max, then copy the plug-in maxiscrp.dlx to your 3dsmax\Plugins directory, and copy the script maxis-maxscript.ms to your 3dsmax\Scripts\Startup directory.
  - Finally, restart Max.

- Updating the Database Cache.
  - The old CMX Exporter used to dynamically download the database from Access via OLE Automation, but that had problems and was slow, so we switched to a simpler more reliable approach of generating a text file from Access that MaxScript can read quickly, instead using OLE.
  - In order to use the CMX Exporter with the Access objects database, you must open up the objects database in Access, which lives at [??? TODO…], and run a macro called [??? TODO…].
  - This macro that exports a cache of the interesting parts of the database as MaxScript code that reads in very quickly.
  - You must run it every time part of the database that matters to you changes, and then you must quit and re-start Max in order to read the updated database cache. (Dynamically reloading the cache would be an easy feature to add, though.)
  - After running the macro, you can quit Access if you want to be polite.

- Running the CMX Exporter.
  - To use the CMX exporter, first you open up Max's "Utilities" panel, then select the "MaxScript" utility.
  - MaxScript will initialize and automatically load in the scripts in the Startup directory.
  - Next, select the "CMX Exporter" item from the Utilities menu in the MaxScript panel.
  - Then, the "Load CMX Exporter" panel will open up below.
  - The "Load CMX Exporter" panel has a button called "Load CMX Exporter Cache", that loads the animation database from cache files, which an Access database macro wrote out into the local temporary file directory.
    - In order to use animation database, you must first open up the Objects database in Access, and execute the macro called [TODO: ???], which updates the cache from the database.
    - After it loads the database cache, the "CMX Exporter Turbo-Deluxe" utility window will open up, with a scrolling list that lets you select any animation or suit in the database.

    - If you don’t need to use the database, and want to run the exporter manually, press the button called “Open CMX Exporter”.
        - The “CMX Exporter Turbo-Deluxe” utility window will open up, but the scrolling list will not have any items in it. You can still operate the exporter manually, with the other buttons in the window.
    - Now you can close the Utilities, MaxScript and “Load CMX Exporter” panels to leave more room for the CMX exporter.

- Using the CMX Exporter.
    - The scrolling list in the CMX exporter contains the names of all skeletons, suits, and skills in the database that are checked for export.
    - Select the name of a skill or suit or skeleton from the list.
    - The Max file name and the status are shown in text fields below the list.
    - Press "Load This Max File" to load the max file containing the currently selected database entry.
    - Press "Load And Export This Max File" to both load and export the selected database entry.
    - The CMX directory and file name are shown below the buttons.
    - There is a "Write Enable" check box that you can uncheck to prevent the exporter from writing any files.
    - Good for a dry-run before the output files are checked out, to see which files will be exported.
    - The Check In and Check Out buttons attempt the check the output files in and out of SourceSafe.
    - They don't check in or out the Max source file, however -- you have to do that yourself.
    - The "Export Current Max File Here" button exports the currently loaded max file to the location as specified by the selected database entry.
    - Make sure the right database entry is selected.
    - If the max file name and the cmx file name are different, you are warned and given a chance to abort.
    - The max file name and the cmx file name should always be the same (except for their extensions),
    - and there is no reason to make them different, so usually something's wrong if you get that warning
    - (like a different entry selected than max file loaded).
    - The "Export Current Max File To..." will prompt you for the cmx file and directory to export the currently loaded max file, ignoring the selected database entry. This button will work even if the database is not loaded.
    - The "Export Max Files in List" button lets you list the names of a bunch of max files to export in a text file.
    - It will load and export each of those files (the names can be absolute or relative to the directory containing the text file of names).
    - The "Filter" field allows you to specify a database "Keywords" filter.

- The 3DRT and Animation databases have a "Keywords" field that is for exporter filtering keywords.
- If you type a string into "Filter", the exporter will only export database entries that have that string in their Keywords field.
- Look at the databases to see the keywords that have been defined.
- If you need to repeatedly export a group of animations or suits, you can make keywords to describe them, and enter them into the database,
- so it's easy to use the exporter to batch export just the subset you're interested in (like all suits and animations associated with a named character, would use the name as a keyword).
- The "Export Files In Database" button loads and exports all max files in the database (subject to the filter), including skeletons, suits, and skills.
- It prints out a summary report of the successful exports and failed exports with error messages.
- The "Load Files In Database" button just loads all max files in the database, to make sure they're there and valid.
- It prints out a summary report of the successfully and unsucessfully loaded file names.
- TODO: Describe how to batch export all the Max files in the database in chunks, to work around 3D Studio Max corruption.

- Marking Up Note Tracks and Tags
  - Max allows you to attach a note track to any object in the scene, and place keys on the note track in time, whose string value you can edit in a little text editor window.
  - Note track keys attached to bones at certain times allow you to control the exporter, and insert user definable events into animations.
  - The CMX exporter parses the note tracks as a TimeProps. Each note track key exists at a particular time, and contains a multi-line text field.
  - The text field is parsed as a Props, each line in the format “tag=value”.
  - The format “tag” (without an = sign) is a shorthand for “tag=”, which defines the tag to be a zero length string. You can use certain shorthand tags like “moving”, “absolute” and “relative”.
  - Be careful not to include any blank lines in the multi text editor, or the CMX text file reader might get confused (this is a bug that should be fixed).

  - Skeletons
    - The “skeleton=name” tag marks the root of a skeleton, used for exporting suits or skills. A normal skeleton is not exported like a masterskeleton. You have to tag the skeleton, so the exporter knows where to look for Skills and Suits.
    - The “masterskeleton=name” tag marks the root of the master skeleton, which is actually exported. There should only be one master skeleton of each name (currently “adult” and “child” in The Sims).

        - The “cantranslate=value” tag marks a bone as having its translation animated, which is false for all bones by default. For Biped, this includes the root and the pelvis. The value should be 1 or 0. The CMX exporter sets this automatically for Biped skeletons, but if you make your own skeleton you might need to use it.
        - The “canrotate=value” tag marks a bone as having its rotation animated, which is true for all bones by default. The value should be 1 or 0.
        - The “canblend=value” tag marks a bone as being blended, which is true for all bones by default. The value should be 1 or 0.
        - The “wiggle=canWiggle wigglePower” tag sets the canWiggle flag (1 or 0) and wigglePower value (a floating point number) of the bone. The format is “int float”. This adds Quaternion Perlin noise to the bone rotation. This is not currently used.
        - The CMX exporter has special support for Biped skeletons. Put the “skeleton=name” tag on the biped root (usually called “Bip01”). The “Bip01” root is renamed “ROOT” in the export process, and all the bones are stripped of their “Bip01 ” prefix and canonicalized. Dummy and Footstep nodes are ignored.
        - The CMX exporter also supports Bones or other 3D Objects as skeletons. The trick is to name the bones using the same conventions as the Biped bones: the root should be tagged with “skeleton” or “masterskeleton”, and be named “Bip01” (or anything else, as long as you’re consistent), and all the bones under that are named “Bip01 BoneName” (don’t forget the space). The exporter recognizes the bones under the root by their prefix (the name of the root followed by space), and assumes anything that doesn’t have that prefix is a skin (a mesh to be rendered, instead of a bone of the skeleton). So every bone of the skeleton should have the prefix of the root (plus a space), so the prefixes are stripped off, and the root is renamed “ROOT”. Kind of weird, but that’s how Biped does it.

    - Suits
        - To export a suit, attach a “suit=name” tag to the root of the skeleton, at the time you want it to export.
        - You can animate objects over time and export several suits on the same skeleton at different times, which snapshot its different states at the times of the suit tags.
        - Rigid Meshes
            - Rigid 3D meshes can be attached to parent bones, or other rigid meshes descendent from a bone. All rigid meshes descendent from a bone are exported in the bone’s coordinate system, and move with the bone.
        - Deformable Meshes

        - Deformable meshes can straddle more than one bone of the skeleton, so they bend smoothly with the skeletal animation.
        - Different vertices of the same mesh are attached to different bones, so the mesh moves with the skeleton.
        - Some vertices may be attached to two bones at once, with a different weight for each bone, so that the seam between bones deforms smoothly.
        - The CMX exporter supports using Physique to bind deformable meshes to Biped skeletons, and knows how to read out the vertex to bone bindings and their weights. It does not support the fancy vertex binding types (bulges and tendons, etc), just the rigid weighted style, attached to no more than two bones.
    - The "type=value" tag controls the suit type, defaulting to 0. The normal type 0 Suit has faces, and the type 1 Suit just transforms vertices, and is used by censorship for bounding box calculation.
    - The "flags=value" tag on a skin sets the skin flags, which make it possible to filter out a set of skins when dressing a suit, and is used by censorship to select which bounding boxes of a type 1 suit to censor.
    - The CMX Exporter writes out the texture map coordinates of the vertices, and the name of the texture map, which should be installed separately or bundled with an the object.
    - Put the texture map into a bitmap file (.BMP). Name the material that refers to the bitmap the same as the bitmap file, without the .BMP suffix. That's the name the exporter will write out, so it had better be the same.
    - The CMX exporter handles smoothing groups, when calculating vertex normals. Just create smoothing groups as you usually do, and it will do the right thing.
    - It optimizes shared vertex usage, by collapsing vertices that are at the same 3d position with the same texture map coordinate and normal.

- Skills
    - A skill has a beginning and an end in time, and applies to one or more bones of the skeleton.
    - The beginning of a skill is marked by a "beginskill=name" tag on a bone of the skeleton at a certain time, and the end of the skill is marked by an "endskill=name" tag on the same bone at a later time. The names of the beginskill and endskill must match.
    - Full Body skills are attached to the root of the skeleton, and record the animation of the root and all bones below it, unless overridden.
    - Partial Body skills can be attached to a lower bone of the skeleton, and affect that bone and all bones below it, unless overridden.

- In the same note track key as the “beginskill=name”, you can specify other parameters for the skill.
- You can override the bones animated by a skill, by listing “includebone=name” and “excludebone=name” tags with the parameters to the beginskill. You can list as many includebone or exclude bone tags as you need, each with different bone names, to specify the bones you want to animate. If you use includebone, the children are not automatically included, so you have to list each one you want. If you use excludebone, the children are automatically included, except for the ones you list. Don’t use includebone and excludebone at the same time, or it might get confused.
- The “xorigin=value”, “yorigin=value” and “zorigin=value” and “spin=value” tags control the position and orientation of the animation’s coordinate system.
- The “absolute”, “relative” and “moving” tags mark an animation as being in an absolute coordinate system, a relative coordinate system (starting at the origin), or a moving animation (walking, running or swimming).
- Events can be delivered to any bone in a skill, at any time between its beginskill and an endskill (inclusive). Events are simply tags that the exporter doesn’t recognize. It removes the tags that it recognizes, inserting the leftover events into the skills to be delivered to the game at run time. Unknown events are simply ignored by the game. See the description of SAnimator to learn about all the different events that are supported.
- The “skill=name” tag is used to associate the entire note track with a certain skill, so other overlapping skills don’t pick up those events. Not generally very useful or often used.

- Technical Notes
  - Requirements for setting 3D Studio Max's Master Scale, Master Unit Type, Master Unit Scale.
    - You need to set 3D Studio Max's master unit scale to the right value (feet). Ask Charles for instructions. [TODO…]
  - How the CMX Exporter Compiles and optimizes meshes.
    - Describe how the exporter reads weighted vertices, handles texture map coordinate, smoothing groups, computes normal vectors, and optimizes the meshes. [TODO…]
  - Texture mapping.
    - Notes related to texture mapping. Where to put the textures, what format to use, how to name the files, how to set up textured materials in Max, and other conventions and constraints. [TODO…]
  - Smoothing groups and normals.
    - How to use smoothing groups to create creases and edges, and control the smoothing of polygonal edges. [TODO…]
  - Time Scale.
    - How to set up the time scale of the animation. Standard values. [TODO…]
  - The CMX Exporter maps new Biped bone names to old bone names. The new version of Character Studio Biped had more medically correct bone names, but we needed to regress to the old names since we still had a lot of old content around. So the CMX Exporter automatically maps new bone names to old bone names, as follows:
    - CALF => LEG1
    - THIGH => LEG
    - UPPERARM => ARM1
    - FOREARM => ARM2
    - CLAVICLE => ARM
  - Error messages are reported with offending file names. All the code that could fail reading or writing a file remembers the name of the file in which the error occurred, so the exporter can report more informative error messages. Tools should check for errors and report the problem file names to the user.
  - Histogram for tuning compression algorithm.
    - Describe low frequency floating point number compression algorithm. Document how to gather statistics on compression from batch exports, and how to tune the compression algorithm. [TODO…]
  - Pose analysis.
    - Document how to gather pose statistics from batch exported animations, and analyze the animation pose statistics, to find glitches and misaligned animations. [TODO…]
  - The CMX Exporter write enable flag disables the exporter writing files when false, so you can test a dry run and see what will happen. The

exporter uses this internally to generate a list of files that will be modified, to check them out and in to SourceSafe.
    - Access Database.
        - Describe how the exporter uses the Access database. Document which fields it depends on, and how they are used. [TODO…]
    - Source Safe.
        - Describe how the exporter uses SourceSafe. Document the configuration variables. [TODO…]

- Animation Compiler
    - The Animation Compiler reads in batches of un-indexed text animation files, and writes out one FAR file full of indexed binary animations.
        - Describe what the animation compiler does, and how it's used in the build process. [TODO…]

- File Formats
    - CMX files (text vitaboy files)
        - Should phase out vitaboy text files, because of other languages that use comma as a floating point decimal symbol. FaceLift currently writes out text files, as does the exporter. The Animation Compiler translates the text files to binary, and compiles them into a FAR file. FaceLift should use libraries that read and write binary files, and the exporter should write out binary files by default. TODO…
    - BCF files (binary vitaboy files)
        - Documented in "SimsFileFormat.txt".
    - SKN files (text DeformableMesh files)
        - Should phase out text files. See note above.
    - BMF files (binary DeformableMesh files)
        - Documented in "SimsFileFormat.txt".
    - BIN files (binary uncompressed floating point)
        - Raw binary floating point numbers in Windows format.
    - CFP files (binary compressed floating point)
        - Document floating point compression algorithm and file format. [TODO…]
    - NDX files (binary vitaboy index file)
        - Index to all CMX files in a FAR file, for pre-loading the cache. Document binary file format, that only appears in FAR files. [TODO…]
    - FAR files (archive directories)
        - Archive of files and directories. Document FAR file format. [TODO…]

- Libraries
    - Skeleton.cpp
        - Props
            - A reference counted property list, containing key/value pairs that are strings.
            - PropsAssociation is used internally by Props.
            - PropsIterator is an iterator for looping over Props.
        - TimeProps
            - A reference counted timeline of Property lists, containing key/value pairs, the keys of which are integers (time), and the values of which are Props (property lists).
            - TimePropsAssociation is used internally by TimeProps.
            - TimePropsIterator is an iterator for looping over TimeProps.
        - Skeleton
            - A reference counted Skeleton keeps track of a list of Bones, threading them into a tree.
            - It also manages a vector of Practices that bind Skills to it, and a list of Dressings that bind Suits to it.
            - It has a name by which it is known, and a cmxFileName from which it was loaded.
            - It has a transform to place it in the world (or group).
            - It has a Bone vector, and keeps track of its root Bone. The Bones have pointers to their parents, siblings and children.
            - The practicesChanged flag allows the skeleton to efficiently figure out which practices it should apply.
            - The void data flag is not currently used.
            - The bboxesValid flag and bbox are used to keep track of the bouding box.
        - Bone
            - A set of Bones are assembled into a skeletal tree, in a threaded list.
            - A Bone represents a translated and rotated coordinate system.
            - It has pointers to other parent, sibling and child Bones, which represent the hierarchy through which translation and rotation are inherited.
            - Each Bone inherits its parent's coordinate system, then adds its translation followed by its rotation, to calculate the coordinate system in which the skins are rendered, then passes that transformation on to its children.
            - Bones are not reference counted, since the Skeleton manages them.
            - Each Bone has a name, as well as a parent name of the bone to which it's attached.

        - It has a (possible NULL) pointer to an optional Props, used to store properties.
        - It has a three dimensional vector trans to represent its local translation, and globalTranslation to represent its global translation.
        - It has a four dimensional quaternion rot to represent its local rotation, and globalRotation to represent its global rotation.
        - It has a transform and a rotMat to keep figure out its transformation to world space.
        - It has an integer priority that keeps track of the highest priority opaque practice bound to it, used for optimization.
        - It has an extreme flag to control whether it should be used for skeleton bounding box calculation, as well as a bboxValid flag and a bbox, used for calculating its bounding box.
        - It has flags canTranslate and canRotate to control whether it supports translation and/or rotation.
        - It has a canBlend flag to control whether motions can be blended together on that bone.
        - It has a canWiggle flag and wigglePower float to control its Perlin noise QuatNoise wiggler (not currently used). Animations could have events that set the canWiggle and wigglePower values of the bones, to automatically add smooth quaternion Perlin noise to the bones. Needs to be tuned and adjusted to determine the best values to use.
    - Registration
        - A Registration has a pointer to a Bone, and a Transform relative to that bone, and it is used to simplify figuring out where to register other 3D objects relative to the skeleton.
    - Skill
        - A Skill represents a reference counted named set of Motions that can be applied to the Bones of a Skeleton by creating a Practice.
        - It has a name by which it's called, the fileName of a file containing compressed floating point numbers, the cmxFileName from which it was loaded, the errorFileName to report if something was missing, the resourceSite from which it was loaded, and a loaded flag.
        - It contains a vector of Motions, which are bound to bones of a Skeleton by a Dressing.
        - It holds all the translations and rotations for the Motions it contains, which are counted in numTranslations and numRotations and stored in translations and rotations.
        - It has a duration (the duration of a Practice played at normal speed), a distance (the distance a walking loop

should travel forward), an isMoving flag to tell if it's moving (like a walking loop), an isTurning flag to tell if it's turning, a 3 dimensional offset to tell how much it moves in 3 dimensions, a quaternion turn to tell how much it turns (not used).
        - It keeps track of its activePractices and its lastUnbindTime, so it can automatically unload when it's not needed.
- Motion
    - A Motion has a Bone name to which it should be bound, as well as a pointer to the Skill that contains it.
    - It refers to a particular number of frames of animation data, and has hasRotation and hasTranslation flags, and translationsOffset and rotationsOffset indices into the Skill's translations and rotations.
- Practice
    - A Practice represents the binding of a Skill to a Skeleton, that associates the Motions of a Skill with the Bones of a Skeleton, and makes a TimePropsIterator for each Motion/Bone binding that has a TimeProps stream of events, so Practice::Apply can read out and handle the events.
    - Practices are not reference counted, since they're managed by a Skeleton.
    - A Practice keeps track of the Skeleton and Skills that it binds, as well as a PracticeAssociation vector (each item containing a Bone, a Motion, and a TimePropsIterator for keeping track of events).
    - It has a user definable priority that effects the order in which they are applied to the Skeleton.
    - It keeps track of the currentTime, the lastTime and the count of frames.
    - Its time scale controls how fast it plays. 1.0 is normal speed, 0.5 is half speed, 0.0 is frozen, etc. The duration depends on the scale times the duration of the skill.
    - The propsTime and lastPropsTime are used to keep track of event delivery.
    - The isOver flag turns to true for the last frame of a Practice.
    - The repeatMode controls what the Practice does when it hits the end. Use 0 for hold, 1 for loop, 2 for ping pong, and 3 for fade.
    - It can be faded in and out by automatically adjusting its weight, which is controlled by the variables fading, fadeStartTime, fadeDuration, fadeStartWeight and fadeEndWeight.

        - It can be flagged as opaque, so it completely covers up lower priority practices. The opaque flag is automatically set when the weight is 1.0.
        - The interpolationMode controls how quaternions are blended. Use 0 to snap to the closest quat. Use 1 to quickly compute a weighted average then normalize. Use 2 to SLERP a spherical linear interpolation.
        - The skipFrames integer is used to test animations out at lower temporal resolutions, and should normally be 0.
        - The stopWhenFaded flag will automatically stop a practice and dispose of it when the fade out is finished.
        - The interruptable flag tells if it's allowed to be interrupted, and can be changed by interruptable events in the event stream.
        - The anchor bone points to the toe bone planted on the ground, and the anchored flag tells if it should be anchored.
        - The mixRootTranslation and mixRootRotation flags are used to inhibit the mixing of root translations and rotations, to avoid snapping problems.
        - The void data pointer isn't currently used.
    - Suit
        - A Suit represents a named set of Skins that can be applied to the Bones of a Skeleton.
        - A Suit is reference counted, and automatically unloaded when it has not been used recently.
        - It has a name, a cmxFileName to keep track of the name of the CMX file from which it was loaded, an errorFileName to report errors when something is missing, and a resourceSite to keep track of where it was loaded from.
        - The type controls whether it is normal (0) or a bouding box (1) not to be drawn but used for censorship.
        - It has an optinal Props to associate user defined data.
        - It keeps track of its activeDressings and lastUnbindTime, so it can automatically unload when it's not needed any more.
    - Skin
        - A Skin binds a mesh to a bone. Or at least, it used to, when all we had were rigid meshes.
        - Actually, now that we have deformable meshes, a Skin just points to a more complicated DeformableMesh, that can attach to a bunch of different bones on its own. The bone of a skin doesn't matter any more (as long as it exists in the skeleton), because the bone names in the DeformableMesh are the ones actually used.

        - It has a Bone name, which must exist, but is otherwise ignored. (If it doesn't exist, the Skin won't be dressed onto the Skeleton, but nothing will go wrong).
        - It has a name, which is the name of a file containing the actual DeformableMesh data.
        - It has some flags, used to control which Skins of a Suit are actually dressed on a Skeleton. This is used by censorship to dress the appropriate subset of the censorship bounding boxes onto the Skeleton. Each skin is marked with a different flag (powers of two), and a mask is passed into Dress to determine which skins should be dressed.
        - It has an optional Props, used to store auxiliary information.
        - It has a pointer to the Suit that contains it.
- Dressing
    - A Dressing is a reference counted run-time structure created when you dress a Suit on a particular Skeleton. It keeps track of the bindings of the shared Suit's Skins with the particular Skeleton's Bones, and indirectly holds other structures needed to draw the character.
    - It has a pointer to the Suit and Skeleton that it binds together.
    - It has a DressingAssociation vector, each item containing a Skin pointer, a Bone pointer, and a void data pointer (that points to a SkeletonBinding).
    - It has a void data pointer, that isn't currently used.
- DeformableMesh
    - Represents a deformable mesh that can be bound to a skeleton in the game at run-time. It keeps track of its file name, its texture name, the skin that it's associated with, a vector of bone names, a DeformableFace vector, a BoneBinding vector, a TextureVertex vector, a BlendData vector, and a NormalVertex vector.
- TextureVertex
    - A two-dimensional texture map location, two floating point numbers u and v ranging from 0.0 to 1.0.
- NakedVertex
    - A three-dimensional vertex, three floating point numbers x, y and z.
- NormalVertex
    - A three-dimensional NakedVertex representing the position, with a another three-dimensional NakedVertex representing the normal.
- BlendData
    - Used to keep track of a vertex that's blended between two bones.

        - The weight is an int32 fixed point value (16.16) because of a historical misunderstanding, and should have been floating point (but it doesn't really matter).
        - The otherVert is the index of the other vertex that the BlendedVertex containing this BlendData is to be blended with.
- DeformableFace
    - Represents a triangle, defined by the indices of three vertices, integers a, b and c.
- BoneBinding
    - Keeps track of the binding of rigid and blended DeformableMesh vertices to a bone.
    - It refers to the bones, rigid and blended vertices of the DeformableMesh that contains them all.
    - It contains the index of the bone to which it binds, the index of the first rigid vertex, the rigid vertex count, the index of the first blended vertex, and the blended vertex count.
- SkeletonBinding
    - Used in the game at run-time, to attach a DeformableMesh to a RenderMesh and an actual skeleton. One SkeletonBinding is created for every Skin of a Suit when you dress a suit on a skeleton, and the Dressing keeps track of it indirectly. It keeps track of a vector of Bones, as well as the bounding box of the mesh on the skeleton.
- UsedVertex
    - Only used by the exporter.
    - Keeps track of each time a vertex is used by a face, so it can handle smoothing groups and optimize texture coordinates.
- BoundVertex
    - Only used by the exporter.
    - Holds some extra information needed for the sorting and compilation phase, and a UsedVertex vector to keep track of the different faces using this vertex.
- BlendedVertex
    - Only used by the exporter.
    - A subclass of BoundVertex that also has a BlendData to keep track of blended vertices.
- PerlinNoise
    - A 1-dimensional Perlin noise generator.
    - Used by QuaternionNoise.
- QuaternionNoise
    - A 4-dimensional quaternion Perlin noise generator.

- VitaBoy.cpp
  - Envelope Template
    - Used as a wrapper to automatically index, load and unload Skeletons, Skills and Suits, so they can be sorted into a set for fast lookup.
  - VitaRenderGroup
    - Subclass of RenderGroup that overrides some of the automatic behavior of IsBoundingBoxDirty, since we're taking care of that stuff ourselves.
  - VBAnimMgr
    - Keeps track of the shared VitaBoy animation data, including Skeletons, Skills, and Suits.
    - It uses maps of Envelope templates around all the Skeletons, Suits and Skills.
    - It keeps track of the Device3D and the RenderObjectFactory.
    - It can load and save animations and indices from text and binary files and streams.
  - VitaBoy
    - Wrapper around Skeleton, that knows about device specific stuff, like how to render the Skins with the 3D library.
    - Keeps track of the VBAnimMgr, the Device3D, the Skeleton and the RenderGroup.
    - Has a realTime flag that's not used.
    - Has a void data pointer that's used to point to the next higher level of abstraction (SAnimator).
    - Has a name, that's not used.
    - Has a rootTransform and rootScale that's used to place its RenderGroup in the world.
    - Has a ghost flag that's used to draw it transparently.

- SAnimator.cpp
  - DressingRecord
    - Used by SAnimator to keep track of Dressings with suit and texture names.
  - HandleVitaBoyAnimEvent
    - Event handler that sends events back to the SAnimator buried under several levels of void pointers.
  - SAnimator
    - Higher level animation controller, below Person but above VitaBoy.
    - Handles mixing and sequencing several parallel layers of animations, dressing suits on the skeleton, walking, and lots of other stuff.
    - Sets up search paths for loading animation data.
    - Manages strings used to name Skeleton, Suits, etc.
    - Resolves Suit and Skill names.
    - Has a reference to the containing Person, and to the contained VitaBoy.
    - It can start and stop the VitaBoy, and save and restore all the animation state, so it's possible to stop all the VitaBoys, close down all the animation files, reload them, and start all the VitaBoys back up again.
    - Keeps track of Suits for body, head, hands and accessories.
    - Keeps track of Outfits
      - Normal, Naked, Swim Suit, Job, Formal, Sleep, Skeleton, SkeletonNeg.
    - Saves and restores state of Practices and Dressings.
    - Keeps track of the location and direction of the person on the grid.
    - Plays object and personal animations, and handles and distributes events to tree code.
    - Supports walking, running and swimming, with a hairy complex state machine.
    - Follows a path created by the router, by sequencing walking and turning animations.
    - Handles footstep sound effects.
    - Adjusts walking and animation speed according to person's mood and user's cursor interactions.
    - Handles reaching, carrying, object, personal, background and foreground animations.
    - Handles animation events.
      - xevt event sends numeric argument to animate primitive false branch.
      - interruptable and interruptible events set practice interruptable flag.
      - anchor event on bone anchors that bone.

    - dress event dresses named suit on skeleton.
    - undress event undresses named suit from skeleton.
    - lefthand event sets left hand to integer argument.
    - righthand event sets right hand to integer argument.
    - censor event sets censorship mask.
    - sound event plays named sound.
    - selectedsound event plays named sound if character is selected.
    - delselectedsound event plays named sound if character is not selected.
    - footstep event plays footstep, integer argument tells if left or right, but is ignored.
    - discontinuity event tells us to expect a snap in root location or rotation, so kill the last practice and don't blend.
- Handles head faking, to turn the head towards interesting objects and other people.
- Handles rendering and bounding box optimizations so it doesn't have to draw when off screen.
- Keeps track of Registration Points, used to position held objects, though balloons, and selected person highlight.
    - Right Hand, Head, Balloon.
- Anchors feet to reduce moon walking (currently disabled).
- Handles censorship bounding box calculation and rendering.
- Makes snapshots of person's body.
- Logs animation and walking state events for debugging.
- States
    - Animating, Sitting, SittingFloor, Standing, StandingTurn, StandingAdjust, WalkingStart, WalkingStop, WalkingTurn, Walking, Running, RunningStart, RunningStop.
- Popup Head Support
    - StartPopupHead, StopPopupHead, StartPopupHeadSubmenu, StopPopupHeadSubmenu, NodPopupHead, GetPopupHeadLocation, RenderPopupHead, SetPopupHeadAttitude.
- Person Picking Support
    - PickedPerson, SetPickedPerson, GetPickingPerson, SetPickingPerson.
- Tweakable Class Variables
    - gPrintEvents controls if animation events are printed out to ctgDump, for debugging.
    - gPinPersonToNextDest is not used.

- gFadeMils controls how long it takes to fade animations out.
- gInterpolationMode controls the interpolation mode of the practices.
- gDoFading controls whether or not we fade practices out.
- gWalkingScale controls the scale that walking animations are played at, by default.
- gWalkingSpeedUp controls how fast walking speeds up, in response to the user pointing at characters.
- gWalkingSlowDown controls how fast walking slows down, in response to the user pointing at characters.
- gStepDistance is the step distance along the path, for route following. It if it's too small, route following will be inefficient. If it's too large, route following will cut corners to abruptly.
- gLookAheadDistance is not used.
- gSmallDistance is used to figure out when to play an adjust animation.
- gStopWalkingTurn is used to figure out when to stop walking and turn in place.
- gMaxWalkingTurn is used to limit how much we will turn while walking.
- gSmallTurn is used to figure out if we need to play a standing turn animation.
- gStepTurn is not used.
- gCutOff45Turn is used to figure out when to use the 45 degree turn animation.
- gCutOff90Turn is used to figure out when to use the 90 degree turn animation.
- gCutOff180Turn is used to figure out when to use the 180 degree turn animation.
- gAdjustDistance is not used.
- gMinStopWalkingTime is used to figure out when and how to stop walking.
- gMinStopWalkingDist is used to figure out how close and how to stop walking.
- gMinStopRunningDist is used to figure out how close and how to stop running.
- gMinAdjustWeight is used to limit the weight for adjusting the walk loop animation to stop walking.
- gAnchorFeet is used to turn on and off the foot anchoring behavior.

- CMXExporter.cpp
  - CMXExporter class interfaces to 3D Studio Max, creates Skeletons, Skills and Suits, and exports them to files.
  - MySceneEntry class wraps a 3D Studio Max object, and connects it to the VitaBoy objects that created for export.
- Maxis-MaxScript.ms
  - Implements the user interface to the exporter and the animation database.
  - Control panel has a list of all skills and suits in the database, allowing you to select any of them, automatically load the correct max file fresh from SourceSafe, check the target files out of SourceSafe, and export it to the right directory, and check the target files back into SourceSafe. Also lets you filter them by different criteria, and batch export or verify the things that pass the filter.
  - Animation database is stored in Access. The exporter used to read the animation database directly out of Access via OLE Automation. Now it loads faster cached data from text files of MaxScript code, exported from Access by a macro.
  - Loads and saves user environment variables.
    - accessCmd
    - sourceSafeDir
    - sourceSafeProgDir
    - sourceSafeServerDir
    - tempDir
    - blowingChunks
    - chunkSize
    - skipToRecord
    - processRecords
  - The verbose flag can be set to false to keep the output terse.
  - Batch exports files in chunks to work around Max corruption bug.
  - Integrated with SourceSafe, to get fresh Max files, and check exported files out and back in (or else undo checkout if an error occurred).
  - Compression statistics and histogram analysis and reporting.
  - Pose analysis and reporting.
  - Old unused code for CSM file bone name mapping and ball transplanting, an experiment in mapping a motion capture skeleton to Biped.
  - Old unused code for batch renaming skills.