SUMMARY OF ANIMATION/MODELING RULES

## 1.0 Goals

This document is intended to give an overview of the current procedures relating to generating content for The Sims. Ideally, new hires will use this document as well as direction from their supervisor to more quickly come up to speed on understanding the constraints and requirements of this project. Current team members should use this document to resolve any questions regarding the conformation of existing or new content to the standards outlined below.

## 2.0 Animation

The animation system used in The Sims is called VitaBoy, a quaternion-rotation system of bones which comprise a skeleton. Suits are collections of meshes, called skins, which are appended to the skeletons and move in the desired fashion. More detail regarding the VitaBoy system can be found on the CTG website, at xxxxxx

## 2.1 Skins

The skins are the actual geometric meshes which are rendered in the application;the people we actually see.

### 2.1.1 Polygon limits

The current working polygon limit for all visible skins is currently set at an arbitrary 450 triangular polygonal faces. This number may well increase in the future.

### 2.1.2 Textures

Each skin mesh may carry its own 256 color texture, which should be delivered in BMP format, at one of the following sizes: 32x32, 64x64, 128x128

## 2.2 Animation Templates and Registration

To assure consistency of starting and ending poses, and to assure the proper calculations of character centers and orientation, certain global requirements have been imposed on all animation files. Those max files which contain the necessary setup information for animation content creation are called animation masters. In almost all cases, new animations will be created by opening one or more such animation templates and modifying it.

### 2.2.1 Named Poses

The following named poses are used to assure consistent transitions between animation sequences (more may be defined in the future):

- Standing
- Standing-carry
- Sitting
- Sitting-carry
- Walking-left-foot-down

### 2.2.2 Orientation

In all animations, the skeleton must be facing so that the **face of the head** is visible in the **Back viewport** of Max, which is facing down the positive y World axis.

### 2.2.3 Registration

To ensure registration with the sprites of the object a skill is associated with, animation must be created in the same file as the model and lined up accordingly. The object model must conform to all constraints outlined below in section 3, Modeling. Thus, the file which contains both the animation and the model is the master for the export of both.

## 2.3 Naming Conventions

All animations follow the same naming convention, which is as follows:
"Biped"-(name of skeleton/character)-(owning object)-(action)-(modifier0)-…-(modifierN)

example: Biped-adult-stove-stirfood-leftburner-loop
If there is no owning object, such as a walk or a standard reach animation, substitute the word "stock" for the owning object.

## 2.4 Tagging

When an animation hs been completed, the relevant information regarding the necessary behaviors of the animation must be encoded into the MAX file, or **tagged**. The essential information that is to be derived from these animations are the motions, or **skills**, which the skeletons will perform. Tags are keys inserted into Notetracks for the relevant bone (most will go on the root) which carry script elements inside their note windows. To create a notetrack for any bone, it is necessary to first highlight the **name**, not the object icon, for the relevant bone, and then press the "create notetrack" button on the trackview toolbar. Notetracks may be appended onto notetracks;they will recurse to the bone that underlies the lowest notetrack. Tags placed on bones will refer to all other bones that depend on the tagged bone;for that reason, full-body animations are tagged on the root, and partial-body animations are tagged at the highest bone involved in that animation. The following is a list of tags and explanations of their use. This lexicon will be expanded in the future as needs arise.

- Skeleton=
  This tag identifies the root name of the skeleton at issue in the animation

- Suit=(name)
  This tag identifies the suit of skins as (name). Suit=(name) is not used in an animation that will be exported for motion;it is a special tag for a file used only to establish the relationship between skins and a skeleton. Such a file is called a **suit file**. The skins must be present in such a .max file, unlike animation files.

- absolute
  This tag identifies the animation as an absolute animation, i.e., one where the root's motion is tracked from beginning to end, in contrast to "moving" (see below). An example would be for sitting down into a chair. This tag must be included in the same note as the beginskill= tag (see below).

- yorigin=(value)
  This tag identifies the **ending** world y-coordinate for the root in an absolute animation. X and z coordinates may also be specified by using xorigin=(value) or zorigin=(value).

- moving
  This tag identifies the skill as one which locomotes the skeleton from one beginning point to and ending point, allowing the system to properly update the location of the skeleton as it moves about the world. An example would be for a walk loop. Like absolute (see above) this tag must be included in the note which contains the beginskill=(value) note.

- xevt=(value)
  This tage is used to synchronize some sprite event that must occur as the skill plays. Examples might be picking up/putting down an object, or playing a sprite animation, where each xevt=(value) refers to the next item on a series of sprite frames, like opening a door.

- anchor=(value)

  This tag specified that a certain bone should not **translate** during a specified point in time. This tag is placed in a note which is appended to the bone which should be locked into place. The value 0 is the default, and denotes not being anchored. The value of 1 indicates anchoring. The most common use for this tag is in walking, when the toes of a foot must remain held in place (translation) for a particular time while the other foot moves to the next position.

- beginskill=(name)

  This tag is one of two (see endskill=(value), below) that defines the beginning and ending of any particular skill. Beginskill=(name) both identifies the name of the skill, and the first frame of its range. Beginskill=(name) must be paired with an endskill=(name) that contains a (name) parameter identical to that specified in the beginskill=(name) tag, or there will be an exporter error.

- endskill=

  This tag is one of two (see beginskill=(value), below) that defines the beginning and ending of any particular skill. Endskill=(name) both identifies the name of the skill, and the last frame of its range. The endskill=(name) tag should actually be placed on the next-to-last frame of the skill, to prevent duplicate endpositions (seen as pauses). Endskill=(name) must be paired with a beginskill=(name) that contains a (name) parameter identical to that specified in the endskill=(name) tag, or there will be an exporter error.

- includebone/excludebone

This tag is placed on a bone which is to be specifically included or excluded in a skill. It is not recursive.

## 2.5 Exporting

TO BE DESCRIBED

# 3.0 Modeling

Because of the prerendered nature of the sprites in The Sims, there are few real restrictions on modeling other than what looks good in the render. However, for means of registration and correct lighting there are some rules.

### 3.0.1 The Cage

All objects must be placed within the cage, which is a space enclosed by the (x,y) world coordinates (1.5,1.5),(1.5,-1.5),(-1.5,-1.5),(-1.5,1.5), defining a square 3''on a side.

### 3.0.2 Orientation

All objects must be placed so that their **front** is showing in the **Front Max viewport** (facing down the negative y world axis). This is checked by making sure that the 1st frame of render shows the object facing **away and to the right.**

## 3.1 Materials

Any material is permissible, so long as the render reduces well to 256 colors.

## 3.2 Palettes

Each object may carry its own 256 color palette, but all sprites in that object should share the same palette. For instance, the microwave might have a different palette from the sofabed, but it would share a palette with the microwave door.

## 3.3 Rendering

Each rendered object produces a z-buffered .tga file from the z-sprite plugin. The control rollout for the z-renderer is found under the tools rollout of max. All frames relevant to the sprite must be included in the start frame/end frame fields.

## 3.4 Delivery

The .tgas are used to derive the z-buffer information, but must be reduced in color, stripped of their z-buffer and saved as .bmp's in order to deliver the pattern information. Touchup by pixelpushing on the z-buffer is often necessary, but must be done with extreme care. Consult someone who has done it before.