
Fakultät für Informatik
Institut für Simulation und Graphik


impara GmbH
Magdeburg

Laborpraktikum

# Character Animation

implemented by

Danilo Gulamhussene
Andreas Petermann

Magdeburg, 9. April 2006

Supervisor:
Prof. Dr. Maic Masuch

**Danilo Gulamhussene MNr.: 162795 contact: danilo@impara.de**
**Andreas Petermann MNr.: 162586 contact: pete@impara.de**
*Character Animation*
Laborpraktikum, Otto-von-Guericke-Universität
Magdeburg, 2006.

# Contents

# 1 Task

First aim of the Laborpraktikum was to implement a system that can dynamically animate a 3D Character in Croquet. This seemed to be an important feature for making games with Croquet and we were looking forward to create games or game-like applications in this environment.

In the early design phase we recognized, that it would be nice to have a system that is more general. A system that can dynamically animate any parameter of any object and can be used in other environments too.

**So the following task emerged:**

Implement a system that can animate any parameter of any object and can blend different animations. As proof of functional capability, a demo using this system should be implemented in Croquet. This demo should show a character dynamically animated according to the input of an user.

# 2 Concept of the system

Thats the concept of the system, we came up with to serve the demands of the task (blend able animation of any parameter in any object).

The system can be divided into two parts... a core and its periphery. The core is very abstract and independent. The periphery is more concrete and adapts the system to a special task. For example the animation of 3D-Characters in Croquet.

This structure is summarized in figure 2.1 and the following sections will describe the several parts of the system.

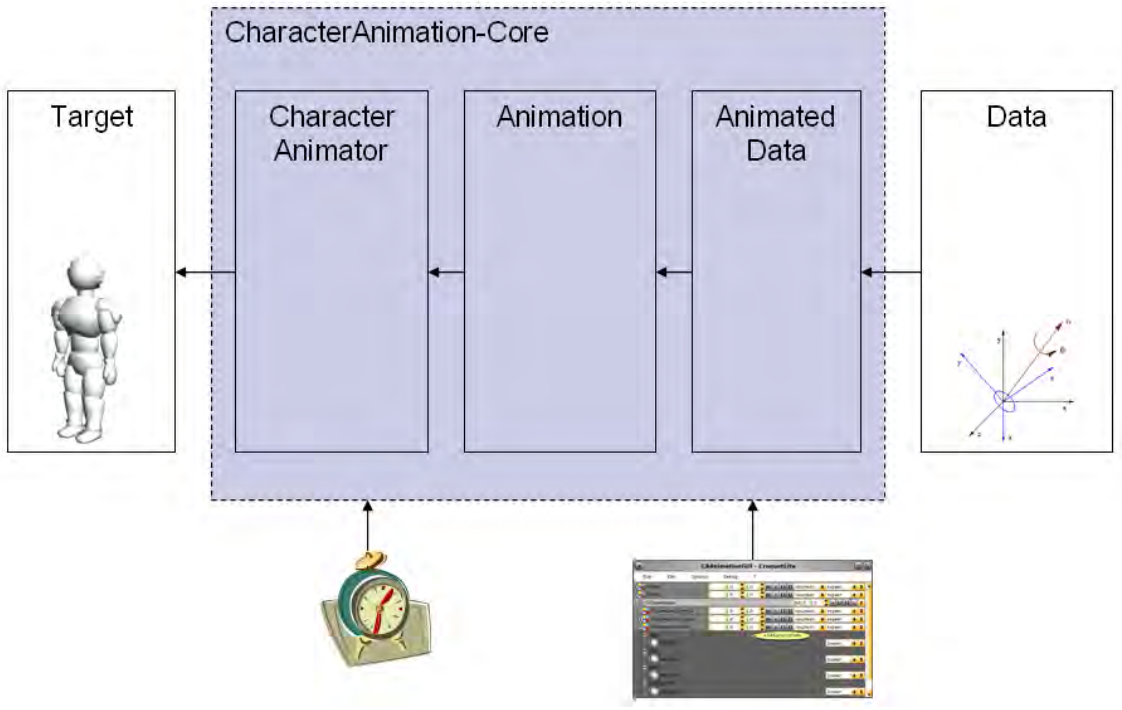



Figure 2.1: Design-Concept

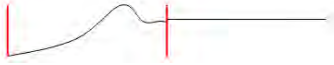
## 2.1 Character Animation Core

As the name lets assume, this is the core of the system. It holds together the data describing the animation and the objects that should be animated accordingly. It is subdivided into three main classes with increasing complexity. These are a relatively simple AnimatedData class, an Animation class that can animate Objects and the most complex of them, the CharacterAnimator class that can dynamically blend multiple animations. These classes are discussed more detailed in the following text. A complete overview of the class structure inside the core-system is shown as UML in figure 2.2.

### 2.1.1 Animated Data

This class contains varying data. Single datasets can be accessed according to a float-value describing the desired position(time). Normally the varying data describes only a known, limited interval. To virtually extend the base-interval, the class supports some behaviors for extrapolating data beyond interval borders. In the current implementation, the request-positions are wrapped and remapped, to achieve this effect.

Animated Data supports the following extrapolation-behaviors:

-  #none
-  #repeat
-  #oscillate
-  #constant

There are two subclasses, that support discrete sampled data. These are SampledData and KeyframedData. The difference between them is, that SampledData assumes, that there is a constant sample rate and so there is no need to save the positions of the samples. KeyframedData does not make this assumption. Both classes do linear key frame interpolation by default, but that's adjustable.

There is at least one AnimatedData for each parameter to be animated in the system, e.g. one for the orientation of the left hand of a character and another for his right hand.

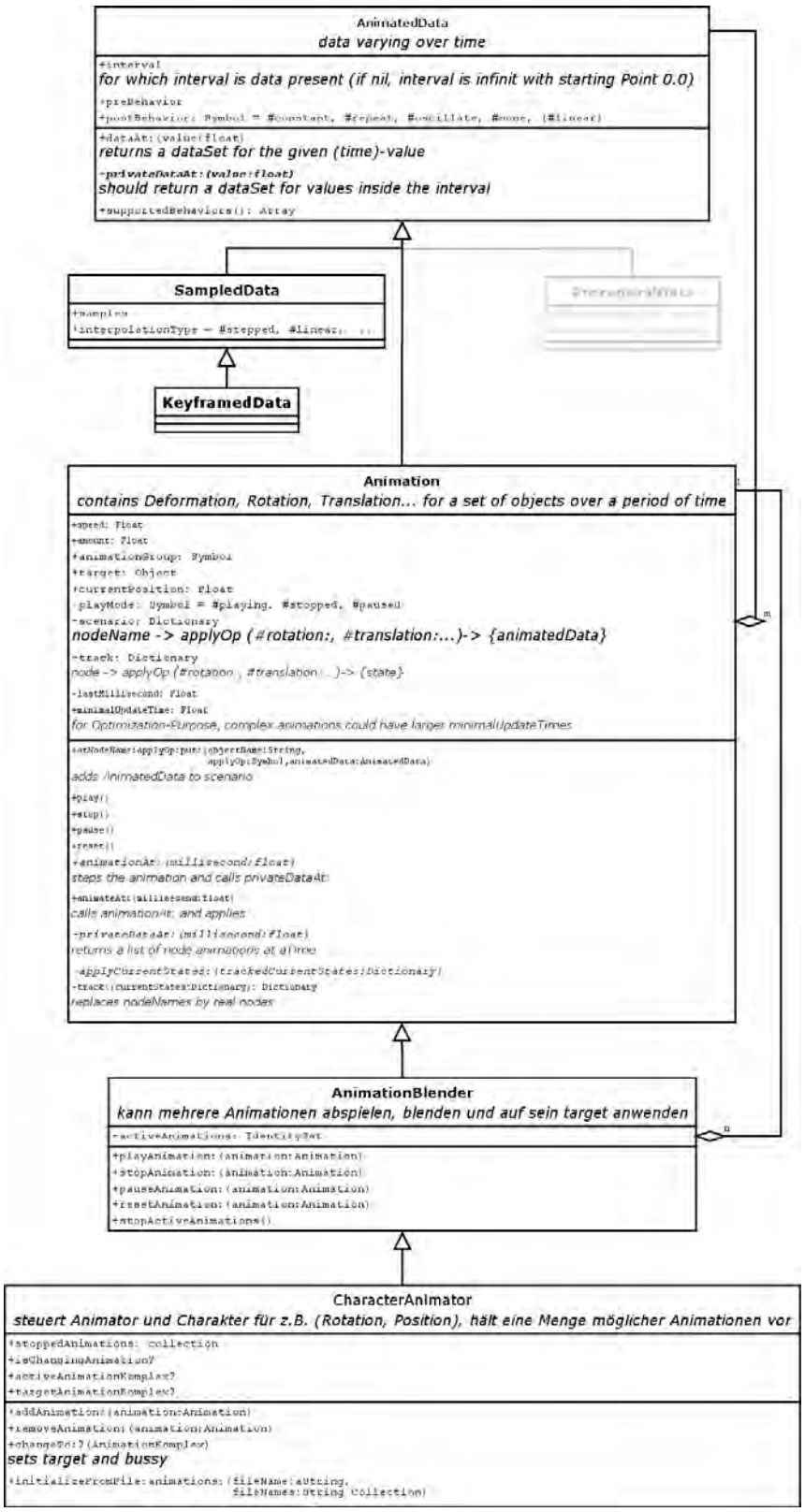



Figure 2.2: UML-Diagramm of CharacterAnimation-Core. The light gray parts mark possible extensions, that are not yet implemented.

If the system should be extended with procedural animations, this is the class, that should be derived from. This means, the system is extended with procedural animations by supporting it with Animated Data that generate datasets procedural.

#### Usage example

```
"initialize a CASampledData"
sampledData := CASampledData new.
sampledData interval: (0.0 to: 10.0).
sampledData samples: #(1.0 2.0 3.0 4.0 5.0).
sampledData interpolationType: #linear.

sampledData dataAt: 1.6. "request a dataset for position 1.6"

"initialize a CAKeyframedData"
keyframedData := CAKeyframedData new.
keyframedData interval: (0.0 to: 10.0).
keyframedData samples: #((0.0 4.0) (8.0 8.0) (10.0 5.0)).
keyframedData interpolationType: #linear.

keyframedData dataAt: 4.0. "request a dataset for position 4.0"
```

### 2.1.2 Animation

Now we have the Animated Data describing the change of parameters, but it needs to be connected to an object, that is animated according to the Animated Data. The role of the Animation class in the system is to serve this task. The Animation class can animate an object(target) by getting datasets for the current time from all its Animated Data's and applying them to the right parameters of the target.

The Animated Data's are organized in a dictionary describing the sub objects and parameters, they belong to. This dictionary is called scenario and has the following structure:

name of sub node → apply operation → base state and Animated Data

To make the Animation more flexibel, there are no direct links to the sub objects, instead there are only names of the sub objects stored. So the animation can be more easily applied to another target, that has a similar structure.

Animations can be played, stopped and paused. Furthermore they support different speeds. Different speeds are achieved by manipulating the time, the Animation is stepped with. To make the speed dynamically adjustable, only the delta time since the last step is scaled according to the current speed of the animation. Based on the scaled delta time, the Animation updates its own virtual time.

Animations have another feature, that is used in the Character Animator to blend different Animations. Animations can be played at different amounts. Imagine, you have an animation of a ball, jumping one meter high. Maybe, you want to reuse the animation, but the ball should only jump only a half meter hight now. When you use this Animation class, you could set the amount to 0.5 to solve this problem. For this, the animation needs the base states in the scenario dictionary. It then blends between base states and Animated Data according to the amount.

Animations provide fade in and fade out by manipulating their amount them self.

#### Usage example

```
"initialize a CASampledData"
sampledData := CASampledData new.
sampledData interval: (0.0 to: 10.0).
sampledData samples: #(1.0 2.0 3.0 4.0 5.0).

"initialize a CAAnimation"
animation := CAAnimation new.
animation atNodeName: 'aNodeName1' applyOp: #setParameter:
put: sampledData.
animation atNodeName: 'aNodeName2' applyOp: #setParameter:
put: sampledData.
animation target: Object new. "you should replace this with an object, that
understands #nodesNamed: and the previously set apply operations (in this case
#setParameter:)"
animation play.

animation animationAt: Time primMillisecondClock."to only request animation
data"
animation animateAt: Time primMillisecondClock."this is the commom way. This
animates the target based on the animation data. But this line only workes if you
have set an compatible target"
```

### 2.1.3 Character Animator

The Character Animator class is a container that can hold many animations. It supports easy access to the animations via names, (e.g. playAnimationNamed: "walk"). The Character Animator can play animations parallel and blend them. The blending is implemented as an addition of all animation states from the different Animations to a base state. When a Character Animator is used, it applies the blended animation and the single Animations are no more applying their states on their own.

Another feature is, that it can automatically cross-fade animations to provide a smooth transition between them.

### Usage example

```
"initialize a CACharacterAnimator"
animator := CACharacterAnimator new.
animator target: Object new.
animator addAnimation: CAAnimation new named: 'testAnim'.

animator playAnimationNamed: 'testAnim'."start an animation"
animator animateAt: Time primMillisecondClock."step the animator"

animator stopAnimationNamed: 'testAnim'."you can stop animations at any time"
```

## 2.2 Periphery

### 2.2.1 Target

That is normally a character. It can have sub objects (nodes), for example head, foot, chest. To let the animation system find these sub nodes, the target should implement "nodesNamed:". This method should take a string and return an OrderedCollection of responding nodes. Implementation of this method is up to you, except you use our system for the animation of 3D-Objects in Croquet, cause for this environment we have already implemented it.

The returned nodes should understand the apply operations you defined in your animations scenario dictionary.

### 2.2.2 Data Types

The Animated Data needs to use some kind of data type to describe the datasets. This could be for example simple floats or quaternions, vertex lists, colors or any other data type, that implements the following operations:

- #interpolateTo:at:
- +
- -

### 2.2.3 Clock

The mechanism to actually run the animation, is to step the animator or animation from the outside. Normally, the animator is stepped with the current system time, but you could use any virtual time (float value) to step the animator. One step is done by calling the method #animationAt: or #animateAt:. These methods could be called for example in the main loop of a game engine.

In our implementation for Croquet, the render process steps the animations for visible animations. The Character Animation GUI has an option to step animations too.

## 2.3 Graphical User Interface

The Character Animation GUI is designed to allow users of the system fast and easy testing of animations without programming. It supports creation of new animations, loading/saving animaitons, exploring them, playing them and customizing them.

The GUI is implemented as a window with menu bar and a list of trees representing Animations and Character Animators. By unfolding the trees, their structure can be explored. Each node is displays buttons and input fields to manipulate its properties and to do some operations with it, e.g. playing it.

A screen shot of the GUI is shown in figure 2.3.

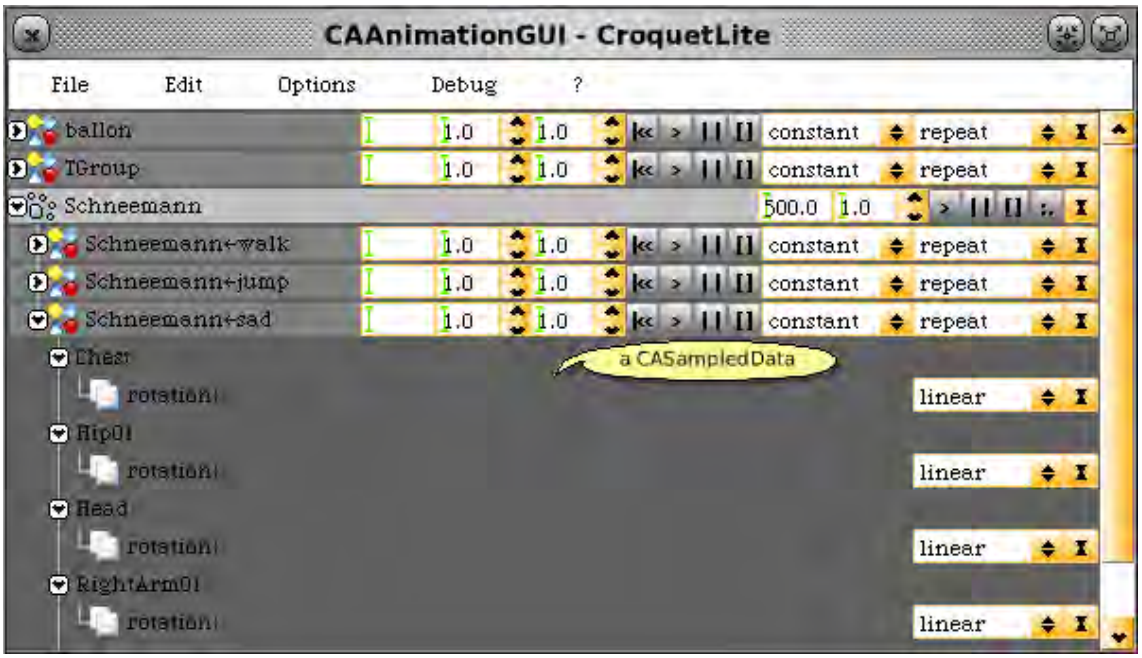



Figure 2.3: Character Animation GUI

# 3 Extensions for Croquet

## 3.1 ASE-Importer

### 3.1.1 Structure of an ASE-file

The form of an ASE-file is similar to a tree data structure. It consists of fieldNames (nodes) and the corresponding fields (subtrees or leaves). The only difference is that an ASE-file has more than one root node. FieldNames of the highest level are for example *SCENE, *MATERIAL_LIST or *GEOMOBJECT. Depending on the type of the fieldName can the corresponding subtree different times be subdevided. The values, the geometry or the animations are stored in the bottom level of this tree, the leaves.

example: extract of an ASE-file

```
...
*SCENE{
    ...
}
*GEOMOBJECT{
    *NODE_NAME "nodename" — leave
    *NODE_TM{
        *NODE_NAME "nodename"
        ...
    }
    *TM_ANIMATION{
        *NODE_NAME "nodename"
        ...
    }
    *MESH_ANIMATION{
        *MESH{
            *TIMEVALUE 0
            *NUM_VERTEX 8
            ...
        }
    }
}
```

...

### 3.1.2 ASE Parser

The procedure of importing an ASE-file takes place in two steps. First the content of the file will be completely scanned and stored as a tree in an OrderedCollection *parseTree* of the following structure:

*parseTree at: i* = fieldName
*parseTree at: i+1* = field (OrderedCollection)
*(with i: 1 to: parseTree size by: 2)*

The in this way generated data structure will now be used in the second step to create the desired TFrame. In our case a CAAnimatedFrame, a special subclass of TFrame.

Out of the structure of this *parseTree* and his leaves will be generated the objects geometry and the animations. The tree is traversed in preorder and in dependencies of the individual fieldNames special methods will be called like *makeGeometry:* or *makeScene:* to analyse the corresponding fields.

### 3.1.3 Add-ons

#### AnimationNode

To be able to parse deformation animations and normals the ase file loading system needed to be upgraded. So the AnimationNode was extended by two new OrderedCollections *deformations* and *normalDeformations*. Just like the corresponding methods *addDeformationSample:vertexList:* and *addNormalDeformationSample:normalList* to fill the data structes with values. Here will be saved a complete list of all vertices or normals with a time value for every keyframe in case of parsing an animation at this moment.

The allready existing method to add rotation samples *addRotSample:x:y:z:angle:* was modified in such a way so that now will only saved the rotation difference to the last position and not the difference to the base position like it is in the ASE-files.

#### AnimationBuilder

The main improvement here is that you can now deliver a base frame to the *asAnimatedFrameFor:targetFrame:* method of the AnimationBuilder. If this base frame is not *nil* then the animations from the AnimationNode will be adapted to the base

positions of this frame so that they are relative to this now. Whether the base frame is *nil* the animations are absolute anymore.

In a simple CAAnimation the animation would be absolute but in a CACharacterAnimator the animations are relative to the base object where they are attached to. Furthermore this method is now able to decide automatically if the animated data objects should be kind of keyframed or sampled data. There for the time value of every keyframe will be analysed and when the successive difference between this time values are constant CASampledData is used.

#### Load3DSMax

To be able to work with these explained add-ons the Load3DSMax class had also been extended. At the one hand deformation animations and normals should be identified and on the other hand this information have to read out and saved. Because of the fact that the rotation and translation animations are not in the same subtree as deformation animations (and normal deformations) in the ASE-file, the method *makeGeometry:* have been modified according to this. For this a new method *makeAnimationFrom:into:* was designed and added to the ASE-file loading system. Thereby we can ensure that maybe allready found animations are not getting lost and new ones simply added to the AnimationNode.

Because the normals are in the subtree *MESH deposited the method *makeMesh:* was adapted to be able to read out and save them. Now the calculation of this normals is no more necessary and can be skipped if normals are found in the ASE-file. But the saved computing time is repayed by much more data that uses much more memory.

The mentioned base frame in the AnimationBuilder have to be declared when initializing the CALoad3DSMax class so that the AnimationBuilder can use the base positions from this frame.

### 3.1.4 Hints

In this place we want to give some hints for working with ASE-files. Perhaps they can be helpful in some situations.

- *TM_ANIMATION node
  - rotation values in *CONTROL_ROT_TRACK are absolute to the base orientation of the corresponding object in the ASE-file
  - in *CONTROL_POS_TRACK the values are also absolute to the base position of this object

- for every object in a ASE-file exist a transformation matrix which is stored in the *NODE_TM node
- the orientation of the coordinate system in Squeak has a different orientation in ASE-files a vertex is stored as (x@y@z)
  → in Squeak we should store it as (x@z@-y)
- bone animation
  - bones are only stored as geometrical objects
    → bones can only be detected by the *NODE_NAME value or some special knowledge of the ASE-file
  - no weight maps available
    → possible identified bones are useless because there are no knowledge about the influences
  - to use bones a new parser should be implement for a file format that supports bones e.g. .GSM (GSkinMesh), .MAX (3DSMax), .ASF (Acclaim Motion Capture), ...

## 3.2 Graphical User Interface

The standard CACharacterAnimationGUI has a very general conceptual design, like the CACharacterAnimationCore. For a better use of this user interface with Croquet the GUI is specialized for working with ASE-files. In this way some new menu entries are added.

The user can now directly import an ASE-file to create a CAAnimation or open a special creation dialog to generate an CACharacterAnimator from multiple files. In this dialog you can choose a base file which includes the geometry information and a list of animation files. The functionality to add an Animation to an existing CACharacterAnimator was extended by the possibility to import them from ASE-files.

The intended support of procedural animations is also added even if it has happend in very simple way. When you are choose Edit → addAnimatedData → Procedural-RotationData a rudimental example can be added to an Animation. This example class is only be able to rotate a node in dependence of the animation time.

# 4 Demo

As proof of concept we implemented a demo in Croquet. It is using the Character Animation system to animate a 3D-Character based on user input and some background objects. The blending functionality of the Character Animator is used for example to create a crouching walk animation, by combining a normal walk cycle and a crouching pose.

The character can be controlled with a keyboard. The keyboard configuration is listed below.

Figure 4.1: Character Animation demo

Keyboard control:

- walk: w, s
- strafe: a, d
- rotate: q, e
- crouch: c
- jump: space
- camera: right mouse button + mouse movement

# 5 Conclusion

The implemented Character Animation system is a multipurpose animation player supporting procedural animations, that can be specialized for different tasks. It proofed its functionality in an interactive demo.

We recognized, that ASE-Files from 3dsMax are limited in their data for bone animations, so that there is an extension of ASE-Files needed to support bone animations. Alternatively another file format should be used for that type of animations.

Another possible application of the system could be to use it for blending between key framed animations and physical animations or animations based on inverse kinematics.

# A Appendix

## A.1 Requirements

For full functionality a 3.8 Squeak-Image with CroquetLite *impara GmbH* and an up to date Tweak is recommended. In the following there is a list of the requirements of the single packages:

- **CharacterAnimation-Core:** Squeak-Image 3.6 or 3.8
- **CharacterAnimation-CroquetLite:** Croquet 0.2 with Animation-extension (Changeset from Takashi) or CroquetLite (in Croquet 0.3 mesh deformation is not supported).
- **CharacterAnimation-GUI-Core:** Squeak-Image 3.8 with up to date Tweak, CharacterAnimation-Core packet
- **CharacterAnimation-GUI-CroquetLite:** CharacterAnimation-GUI-Core and CharacterAnimation-CroquetLite packet
- **CharacterAnimation-Tests:** CharacterAnimation-Core and CharacterAnimation-CroquetLite packet with contents *CAModelle*

## A.2 Creation of ASE-Files with 3DSMax

### A.2.1 Setting up the scene

Customize the base units, cause the exported files will use this as reference. We would suppose to use 1 meter as base unit in 3DSMax, so you can model in real world dimensions. The base units can be adjusted in Customize→Units Setup→System Unit Setup.

### A.2.2 Animating objects

The Character Animation system supports rotations, translations and mesh deformations so far. Other animated parameters will be ignored.

Objects can be animated using rigs. The export of the rig can be suppressed, while the animation of the object is retained.

### A.2.3 Exporting animations to ASE

When you want to create a character with many different animations, it is possible to hold these animations sequenced in one max-file. When single animations should be exported it is important to activate the corresponding time interval in the time line.

There is a known issue. Max optimizes the export of parameters that don not change. There are no key frames generated for these intervals. This appears too, when the first and last frames are the same. This can lead to problems, when the animation should be used as a loop.