Animationen in MeVisLab - Konzept und Funktionsweise

Konrad Mühler

19. September 2006

Inhaltsverzeichnis

Dieses Dokument beschreibt das Konzept und die Funktionsweise der Möglichkeiten zur Animationsgenerierung in MeVisLab, die in Magdeburg entwickelt werden. Das Konzept der zugrunde liegenden Skriptsprache wird im ersten Teil dieses Dokuments behandelt. Im zweiten Teil werden die entwickelten Animations-Module in MeVisLab und deren Funktionsweise vorgestellt. Der dritte Teil beschreibt alle bisher implementierten Anweisungen.

1 Einleitung

Um eine Animation erzeugen zu können, muss diese in einem Skript beschrieben werden. Ein solches Skript enthält Anweisungen, die die Veränderung einzelnen Parameter über die Zeit beschreiben. Dies können Eigenschaften von einzelnen Objekten wie Farbe und Transparenz sein oder Parameter von Kameras und Viewern. Ein Skript wird als Textdatei eingelesen und ausgeführt. Die entstehende Animation kann direkt visualisiert werden (*Realtime*) oder als Video bzw. Einzelbilder gespeichert werden.

Ein Grundgedanke der Skriptsprache ist der Einsatz von mehreren Abstraktionsebenen (*Levels*) bei der Definition einer Anweisung. Auf unterster Ebene stehen die *LowLevel*-Anweisungen. Diese sind atomar und werden direkt ausgeführt. Mehrere Anweisungen können zu abstrakteren *HighLevel*-Anweisungen zusammengefasst werden, um den Entwurf von Skripten zu erleichtern (siehe **??**). Gleiches gilt für Objektbezeichnungen und Parameter. Auch diese können abstrahiert werden (siehe **??** und **??**).

Weiterführende Informationen vor allem zum Konzept:

- **Webseite zum Animationsprojekt mit Beispielanimationen**
  http://isgwww.cs.uni-magdeburg.de/cv/projects/animation/
- **Paper MICCAI 2006:**
  "Adaptive script based animations for intervention planning"
  (Auf Anfrage)
- **Technical Report 2006:**
  "Adaptive script based animations for medical education and intervention planning"
  http://wwwisg.cs.uni-magdeburg.de/cv/pub/files/Muehler06-TechnicalReport.pdf
- **Diplomarbeit Konrad Mühler:**
  "Skriptbasierte Generierung von Animationen für die medizinische Aus- und Weiterbildung"
  http://wwwisg.cs.uni-magdeburg.de/cv/pub/files/DA_KonradMuehler.pdf

- **Paper BVM 2006:**
  "Skriptbasierte Animationen für die Operationsplanung und Ausbildung"
  http://wwwisg.cs.uni-magdeburg.de/cv/pub/files/BVM-2006_Muehler.pdf
- **Poster CURAC 2005**
  "Animation und interaktive Visualisierung patientenspezifischer Daten in einer Lernumgebung für die Planung von Leberoperationen"
  http://isgwww.cs.uni-magdeburg.de/cv/gallery/Poster/Poster_AniLST-Curac05.pdf

2 Skriptaufbau

Ein Skript gliedert sich in zwei Bereiche: einen *Initialisierungsteil* und einen *Anweisungsteil*. Im *Initialisierungsteil* werden Einstellungen vorgenommen, die die Länge der Animation betreffen. Im *Anweisungsteil* folgen die Anweisungen zur Animationsgenerierung.

2.1 Initialisierungsteil

Im Initialisierungsteil wird der zeitliche Rahmen einer Animation festegelgt. Dies geschieht mit Hilfe der Angabe der Zeiteinheiten (`LengthTimeUnit`) und der Länge der Animation in Sekunden (`LengthSeconds`).

> **Achtung:** Sollten im Anweisungsteil Zeitangaben genutzt werden, die größer als `LengthTimeUnit` sind, kommt es zum Crash
> (Dies wird bald so geändert, dass die Länge in Zeiteinheiten aus den Skriptanweisungen selbst extrahiert wird.)

Weiterhin wird durch die Angabe des Parameters `RealTime` (`=yes` oder `=no`) angegeben, ob das Skript in Echtzeit ausgeführt werden soll oder nicht. Eine Ausführung in Echtzeit bedeutet zum Einen, dass am Ende kein Video erzeugt wird und zum Anderen, dass von der Standardframerate 14 nach oben und unten abgewichen wird, je nach verfügbarer Rechenpower und Umfang der Anweisungen.

```
[Ini]
LengthTimeUnit=14
LengthSeconds=10
RealTime=no
...
```

2.2 Syntax der Anweisungen

Die Anweisungen der Skriptsprache gliedern sich in vier Bereiche. Jede Anweisung besteht dabei aus einer Zeitangabe, die den Zeitraum oder Zeitpunkt der Ausführung der Anweisung bestimmt, einer Objektbezeichnung, die das Objekt festlegt, auf das sich die Anweisung bezieht, der Befehlsbezeichnung und möglichen Parametern. Die vier Bereiche sind jeweils durch ein Leerzeichen getrennt.

```
Time Object Command [Parameter]
```

```
[0,10] 'Liver' setColor blue
```

2.2.1 Zeitangaben

Es gibt zwei verschiedene Möglichkeiten der Zeitangabe: die Angabe eines **Zeitpunktes** oder die eines **Zeitbereiches**. Die Zeitangabe erfolgt in eckigen Klammern, wobei bei der Angabe eines Zeitbereiches dessen Start- und Endwert durch ein Komma getrennt wird. Bei der Angabe eines Zeitpunktes wird die Anweisung zu diesem ausgeführt. Im Fall eines Zeitbereiches wird versucht, die durch die Anweisung hervorgerufene Änderung in diesem Zeitbereich zu interpolieren. Ist dies nicht möglich, zum Beispiel weil es sich um eine diskrete Zustandsänderung handelt (Zeichenketten, Wahrheitswerte), wird die Anweisung zu Beginn des Zeitbereiches ausgeführt. Die Angabe der Zeiten erfolgt in abstrakten Zeiteinheiten. Die Umrechnung der Zeiteinheiten in reale Sekunden ergibt sich aus den Angaben im Initialisierungsteil.

```
[0,10] ...
[5] ...
```

2.2.2 Ersetzung von Objektbezeichnungen

An zweiter Stelle innerhalb einer Anweisung erfolgt die Bezeichnung des Objektes, auf welches die Anweisung angewendet werden soll. Um Objektbezeichnungen mit Leerzeichen zu ermöglichen wird diese in Hochkommata (') gesetzt. Auf der *LowLevel*-Ebene entsprechen die Objektbezeichnungen den Bezeichnungen, die den Objekten bei der Segmentierung der Daten zugewiesen wurden bzw. die sie innerhalb des Datensatzes haben. Unteilbare Bezeichnungen, also die Bezeichnungen direkt im Datensatz werden als atomare Bezeichnungen bezeichnet.

Im *HighLevel*-Skript können dagegen andere Bezeichnungen gewählt werden. Diese werden anhand von Ersetzungsvorschriften in atomare Bezeichnungen überführt. Dabei kann eine einfache Ersetzung erfolgen (z.B. kann die Bezeichnung `'Liver'` durch `'Liver,`

segm' substituiert werden). Oder eine Bezeichnung kann durch mehrere Bezeichnungen ersetzt werden (z.B. 'Lung' durch 'Lung left' und 'Lung right'). **Erfolgt eine Ersetzung hin zu mehreren Objektbezeichnungen, so wird die Anweisung für das jeweilige Objekt im Ganzen kopiert.** Aus einer Anweisung für das Objekt 'Lung' würden im Ersetzungsvorgang daher je eine Anweisung für 'Lung left' und 'Lung right' entstehen. Mithilfe dieser Ersetzungen ist es möglich, auf der Ebene des *HighLevel*-Skriptes eine Anweisung für ganze Objektgruppen oder alle Objekte anzuwenden.

Zusätzlich zu den atomaren Bezeichnungen der Objekte im Datensatz gibt es zwei systemseitige Bezeichnungen: 'Cam' und 'System'. Diese kommen bei Anweisungen zur Anwendung, die eine Manipulation der Kamera (z.B. eine Rotation) oder der gesamten Szene (z.B. eine Änderung des Hintergrundes) ermöglichen.

Die Ersetzungsvorschriften werden in einer XML-Struktur abgelegt.

Anweisung mit einer zu ersetzenden Objektbezeichnung:

```
[0,10] 'Lung' on
```

Auszug aus der XML-Struktur mit den Ersetzungsvorschriften:

```
<object name="Lung">
        <object>Lung left</object>
        <object>Lung right</object>
</object>
<object name="Lung left">
        <object>kind2, LiLungeSeg, m Int</object>
</object>
<object name="Lung right">
        <object>kind2, ReLungeSeg, m Int</object>
</object>
```

Zwei Anweisungen als Ergebnis der Ersetzungen:

```
[0,10] 'kind2, LiLungeSeg, m Int' on
[0,10] 'kind2, ReLungeSeg, m Int' on
```

Achtung: Es werden nur Objektbezeichnungen ersetzt, die an der object-Position einer Anweisung stehen. Objektbezeichnungen in den Parametern werden bei der Umwandlung von *High-* in *LowLevel*-Skripte nicht ersetzt! Objektbezeichnungen die in Parametern stehen, werden erst bei der Interpretation des Skriptes im Verlauf einer Animation ersetzt, da erst zu diesem Zeitpunkt beispielsweise evtl. benötigte BoundingBoxen bekannt sind.

Achtung: Die Anweisungen werden zuerst ersetzt und erst dann die Objektbezeichnungen!

2.2.3 Ersetzung von Befehlen

Während eine Anweisung die Gesamtheit aus Zeitangabe, Objektbezeichnung, Befehlsbezeichnung und Parameter darstellt, wird im Weiteren als *Befehl* nur die eigentliche Befehlsbezeichnung betrachtet. Die Befehle des *LowLevel*-Skriptes sind atomar, das heißt, sie sind direkte Repräsentationen von Befehlen der Skriptschnittstelle. Eine Aufstellung elementarer Befehle findet sich Abschnitt **??**. Auf diesen Befehlen aufbauend werden die *HighLevel*-Befehle definiert. Ähnlich den Objektbezeichnungen kommt es auch hier zu Ersetzungen von *HighLevel*-Befehlen in einen oder mehrere andere Befehle. Ebenso können die Ersetzungen hierarchisch strukturiert werden, sodass nicht jeder *HighLevel*-Befehl sofort auf einen atomaren *LowLevel*-Befehl abgebildet werden muss.

Die Ersetzungsvorschriften für Befehle sind ebenfalls in einer XML-Struktur abgelegt. Da die Ersetzung eines Befehls durch einen oder mehrere Befehle aber Auswirkungen auf die gesamte Anweisung (z.B. die Zeitangaben oder Objektbezeichnungen) haben kann, gibt es einige Unterschiede zum Prozess der Objetkbezeichnungsersetzungen.

Die Ersetzungsvorschrift umfasst eine oder mehrere komplette Anweisungen. Als Zeitangabe für die ersetzenden Anweisungen kann wahlweise eine konkrete Zeit, wie auch im Skript, angegeben werden oder aber durch ein `T` die Zeitangabe aus der zu ersetzenden Anweisung übernommen werden. Ähnliches gilt für die Objektbezeichnung, die entweder konkret eine Objektbezeichnung oder aber ein `O` als Platzhalter darstellen kann. Bei einem Platzhalter `O` würde die Objektbezeichnung aus der zu ersetzenden Anweisung genutzt werden. Mit Unterstützung dieser Notation kann die Objektbezeichnung bei der Ersetzung auch als Parameter in die neue Anweisung übernommen werden. Mithilfe eines `P` ist es außerdem möglich, auch die in der zu ersetzenden Anweisung eventuell angegebenen Parameter in die neue Anweisung zu übernehmen. Bei der Entwicklung der atomaren und *HighLevel*-Anweisungen ist darauf zu achten, dass die Platzhalter `T`,`O` und `P` nicht als Parameter in Frage kommen.

Beispiel 1:
Ausgangsanweisung:

```
[0,10] 'Liver' on
```

Auszug aus der XML-Struktur mit den Ersetzungsvorschriften:

```
<command commandStr="on">
        <command>T 0 setVisible true</command>
        <command>T 0 setColor red</command>
</command>
```

Ergebnis nach der Ersetzung:

```
[0,10] 'Liver' setVisible true
[0,10] 'Liver' setColor red
```

Beispiel 2:
Ausgangsanweisung:

```
[0,10] 'Liver' view sagittal
```

Auszug aus der XML-Struktur mit den Ersetzungsvorschriften:

```
<command commandStr="view">
        <command>T 0 on</command>
        <command>T 'Cam' move 0 P 1.0</command>
</command>
```

Ergebnis nach der Ersetzung:

```
[0,10] 'Cam' move 'Liver' sagittal 1.0
```

2.2.4 Ersetzung von Parametern

Als Parameter einer Anweisung werden all' diejenigen Angaben gewertet, die nach der Befehlsbezeichnung folgen. Wird mehr als ein Parameter angegeben, erfolgt die Trennung durch Leerzeichen. Enthält ein einzelner Parameter ein oder mehrere Leerzeichen, so wird dieser Parameter in Hochkommata notiert.

Auch für die Parameter sind Ersetzungen beim Umwandlungsprozess der Skripte vorgesehen. Dabei wird zwischen zwei Arten der Ersetzung unterschieden: der Einzelwertersetzung und den Geschwindigkeitsangaben.

Bei Ersetzungen, die auf der ersten Art basieren, wird lediglich ein Austausch der Zeichenkette des Parameters durch eine andere Zeichenkette vorgenommen. So kann der Parameter `red` durch den RGB-Wert `255,0,0` ersetzt werden. Die Definition dieser Ersetzungsvorschriften erfolgt in der XML-Struktur für die Befehlsersetzungen. Die Ersetzung

ist befehlsabhängig und wird in der Struktur als zusätzliches Element der Befehlsersetzung angegeben (siehe Listing **??**).

```
<command commandStr="setColor">
        ...
  <parameter paramStr="red" singleValue="255,0,0" />
  <parameter paramStr="yellow" singleValue="255,255,0" />
  <parameter paramStr="blue" singleValue="0,0,255" />
  <parameter paramStr="green" singleValue="0,255,0" />
</command>
```

Handelt es sich bei der Parameterersetzung um eine Einzelwertersetzung, so wird der zu ersetzende Wert mit `singleValue` angegeben.

Es gibt nur Einzelwertersetzungen! Der Kram mit der Geschwindigkeit war eher ein Zugeständnis im Diplom, hat sich in der Praxis aber als untauglich erwiesen.

3 Funktion der Module in MeVisLab

inklusive aller Felder der Module ...

4 Implementierte Anweisungen

4.1 LowLevel-Anweisungen

4.1.1 Bewegung der Kamera mit `move`

Diese Anweisung bewegt die Kamera.

Parameter:
`target_object camera_sphere angle_sagittal angle_axial zoom`

Beispiele:
```
[0,10] 'System' move 'Tumor' 'All' 90 0 1
[5] 'System' move 'Lymphnode' 'All' 45 45 0.6
```

`target_object`

`target_object` bezeichnet das Objekt, auf welches die Kamera von ihrer Position aus blicken soll.

`camera_sphere`

Die Ziel-Position befindet sich dabei auf einer Kugel, die eine BoundingBox umgibt. Das Objekt (die Objektgruppe) dieser Boundingbox wird im Parameter `camera_sphere` angegeben. Klassicherweise ist es das Objekt `'All'` welches eine die gesamte Szene umschließende Kugel ergibt (Wenn denn dieses Objekt in der `ObjectNames.xml` definiert ist.).

`angle_sagittal` und `angle_axial`

Die Position der Kamera auf der Kugel wird durch Kugelkoordinaten angegeben. Diese bestimmen die Position in Grad vom oberen Pol der umschließenden Kugel aus, und zwar in sagittale und in axiale Richtung. Die Koordinaten `90 0` geben gemeinhin die Sicht genau von vorne an.

`zoom`

`Zoom` gibt den Zoom-Faktor der Kamera an. 1 ist gleichbedeutend mit "Objekt ist vollständig im Viewer sichtbar". Werte kleiner 1 stehen für ein geringere, Werte größer 1 für größere Entfernungen. Die Position der Kamera vom Zoom-Faktor nicht berührt.

4.1.2 Rotation der Kamera um ein Objekt mit `rotate`

Diese Anweisung rotiert die Kamera um eine Achse, die durch den Mittelpunkt eines (Ziel)Objektes verläuft.

Parameter:
`object axis angle`

Beispiel:
`[0,10] 'System' rotate 'Kreuzband' 0,0,1 90`

`object`

Zielobjekt, durch dessen Mittelpunkt (der BoundingBox) die Rotationsachse verlaufen soll.

`axis`

Rotationsachse. Werte wie `axial (0,0,-1)`,`coronal (0,-1,0)`,`sagittal (1,0,0)` können beispielsweise in der `HighLevel-Commands.xml` definiert werden bzw. sind in den Beispieldateien schon so definiert und können verwendet werden.

`angle`

Zu rotierender Winkel in Grad

4.1.3 Rotation der Kamera um ihre Z-Achse mit `rotateCamZ/rotateCamZTo`

Diese Anweisungen rotieren die Kamera um ihre eigene Z-Achse, kippt sie also. Die Rotation kann absolut erfolgen (`rotateCamZTo`) oder relativ zur momentanen Neigung (`rotateCamZ`).

Parameter:
`angle`

Beispiele:
`[0,10] 'System' rotateCamZ 45`
`[0,10] 'System' rotateCamZTo 75`

`angle`

Zu rotierender Winkel in Grad

4.1.4 Veränderung des Szenenhintergrundes mit `setBackground`

`setBackground R,G,B [ramp]`

Beispiel: `[0] 'System' setBackground red ramp`
(red wird laut `HighLevel-Commands.xml` zu `255,0,0` ersetzt)

Setzt die Hintergrundfarbe der Szene auf einen RGB-Wert. Wird der Parameter `ramp` angegeben, so wird der Hintergrund im Verlauf von dieser Farbe zu schwarz dargestellt.

4.1.5 Veränderung der Farbe eines Objektes mit `setColor`

```
setColor R,G,B
```

Beispiel: `[0,10] 'Kreuzband' setColor 255,255,0`

Setzt die Farbe eines Objektes auf den angegebenen RGB-Wert, wobei die einzelnen Werte sich jeweils im Bereich zwischen 0 und 255 bewegen.

4.1.6 Erzeugen eines Videos am Ende einer Animation mit `setCreateVideo`

```
setCreateVideo true|false
```

Beispiel: `[0] 'System' setCreateVideo true`

Erlaubt oder verhindert das Erzeugen eines Videos am Ende einer Animationsgenerierung. Im RealTime-Modus ist die Videoerzeugung standardmäßig abgeschaltet (kann aber durch die Anweisung eingeschaltet werden). Bei `realtime=no` ist die Videoerzeugung standardmäßig aktiviert.

4.1.7 Veränderung der Darstellungsqualität eines Objektes mit `setQuality`

```
setQuality value
```

Beispiel: `[0,10] 'Kreuzband' setQuality 0.5`

Setzt die Darstellungsqualität eines Objektes auf einen Wert zwischen 0 und 1.

4.1.8 Selektions eines Objektes mit `setSelected`

```
setSelected true|false
```

Beispiel: `[0] 'Kreuzband' setSelected true`

Selektiert oder deselektiert ein Objekt.

4.1.9 Veränderung des Darstellungsstils eines Objektes mit `setStyle`

```
setStyle Filled|Lines|Points
```

Beispiel: `[0] 'Kreuzband' setStyle Lines`

Setzt den Darstellungsstil auf eine gefüllte (Filled), eine gitternetzige (Lines) oder eine gepunktete (Points) Darstellung.

4.1.10 Veränderung der Transparenz eines Objektes mit `setTransparency`

```
setTransparency value
```

Beispiel: `[0,10] 'Kreuzband' setTransparency low`
(`low` wird laut `HighLevel-Commands.xml` zu `0.2` ersetzt)

Setzt die Transparenz eines Objektes auf einen angegebenen Wert zwischen 0 und 1.

4.1.11 Veränderung der Sichtbarkeit eines Objektes mit `setVisible`

```
setVisible true|false
```

Beispiel: `[0] 'Kreuzband' setVisible true`

Setzt die Sichtbarkeitseigenschaft eines Objektes.

4.2 HighLevel-Anweisungen

4.2.1 Ändern der Sichtbarkeit eines Objektes mit `on/off`

```
object on
object off
```

Beispiel: `[0] 'Kreuzband' on`

```
Anweisungsersetzung in XML:

<command commandStr="on">
  <command>T 0 setVisible TRUE</command>
</command>
<command commandStr="off">
  <command>T 0 setVisible FALSE</command>
</command>
```

4.2.2 Selektieren eines Objektes mit `select/deselect`

```
object select
object deselect
```

Beispiel: `[0] 'Kreuzband' select`

```
Anweisungsersetzung in XML:

<command commandStr="select">
  <command>T 0 setSelected TRUE</command>
</command>
<command commandStr="deselect">
  <command>T 0 setSelected FALSE</command>
</command>
```

4.2.3 Kamera einfach positionieren mit `view`

```
object view direction zoom
```

Beispiel: `[0,10] 'Kreuzband' view front 1`

```
Anweisungsersetzung in XML:

<command commandStr ="view">
  <command>T 'Cam' rotateCamZTo 0</command>
        <command>T 'Cam' move 0 P</command>
</command>
```

Der Parameter `front` wird hierbei nochmals durch `90 0` ersetzt.

4.2.4 Objekt hervorheben mit `emphasize`

```
object emphasize
```

Beispiel: `[0,10] 'Kreuzband' emphasize`

```
Anweisungsersetzung in XML:

<command commandStr ="emphasize">
  <command>T 0 view front 1</command>
  <command>T 0 setColor red</command>
  <command>T 0 setTransparency low</command>
</command>
```

4.2.5 Initialisierung mit `init`

```
init
```

Beispiel: `[0] 'System' init`

Die init-Anweisung initialisiert eine Animation mit den Grundeinstellungen für alle Objekte und die Kamera. Sie ist im Gegensatz zu den anderen HighLevel-Anweisungen datensatzabhängig. Nicht unbedingt von jedem Patienten, aber von der Art der Daten (Knie, Hals, ...) wenn innerhalb eines Datensatzstypes immer die gleichen Bezeichnungen verwendet werden.

```
Anweisungsersetzung in XML:

<command commandStr="init">
  <command>T 'All' on</command>
  <command>T 'All' setTransparency normal</command>
  <command>T 'Knochen' setTransparency high</command>
  <command>T 'System' setBackground white</command>
  <command>T 'All' view front 1</command>
  <command>T 'System' setBaseColors</command>
</command>
```