Anymation with CATHI

Andreas Butz
Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI)
Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany
butz@dfki.uni-sb.de

Abstract

This paper presents an incremental approach to the automated generation of 3D animation clips for the explanation of technical devices. It describes the system CATHI, which is part of the intelligent multimedia presentation system $\mathcal{PPP}$. CATHI generates animation clips in the context of a coordinated multimedia document.

The system is able to generate animation scripts and to present the corresponding animations with a minimal delay, i.e. within a few seconds. This allows it to be used for on-line and on-demand generation of presentations.

The structure of the generated animation depends not only on the given communicative content. It is also influenced by a number of generation parameters. Furthermore, the design of the final animation depends on resource limitations, such as the available computing capacity and the graphical capacity of the output medium.

The more powerful the machine is, the more complex the animation scripts generated by CATHI tend to be, but a fluent output within a given amount of time is achieved on (nearly) any machine – hence the name 'anymation'.

Introduction

The CATHI system (Computer Animation Tool for Help and Information Systems) is part of the intelligent multimedia presentation system $\mathcal{PPP}$ (André, Müller, & Rist 1996). $\mathcal{PPP}$ presents information about the usage and function of technical devices in the form of interactive operating instructions, which are adapted to the presentation situation and to the person interacting with the system. This approach requires the generation of adapted presentation components, such as text, graphics, and animation, from scratch.

Since no preproduced animation clips are used in $\mathcal{PPP}$, the whole design of the animation must be computed, including camera motions, cuts, lighting setup, the coordination of these settings with object motions in the model world, and visual effects, such as depth of field or spotlights. The model world as well as the possible actions in it (resulting in object motions) are given in the form of 3D geometry and motion descriptions. One input to the CATHI system is the communicative goal that is to be fulfilled by the animation. The other inputs are generation parameters that influence the structure of the resulting clip. As a first step, CATHI generates a script for the animation, that specifies camera positions and settings, lights, object positions and motions, all at the lowest level of description (matrices, absolute coordinates). This script is translated into commands for a specific renderer, rendered into an animation clip and immediately presented on the screen. The following sections will discuss several aspects of the approach in more detail and show results produced by the system. There will also be a short comparison to other work in related areas and an outlook upon future extensions.

System input

When the $\mathcal{PPP}$ system plans a multimedia presentation, it decides which pieces of information should be encoded in which parts of the presentation, and how they will interact in the final document. It assigns communicative goals to the presentation parts, and triggers their generation by the corresponding realization modules (NLG for the text parts, automated graphics generation for illustrations, CATHI for animations).

Communicative goals

The specification of the communicative goal is the starting point for the generation of a 3D animation. It is done in the form of a simple statement naming a communicative function of the clip (such as 'localize object' or 'show action') and the object(s) and action(s) to which this statement refers. An example for this kind of top-level specification is the following:

```
(localize-object :object "cylinder-group")
```

This statement indicates, that the object (group) named 'cylinder-group' should be localized in an animation clip, i.e. the spectator should know where the object is after he has seen the animation. Other examples are

```
(show-relation :obj-1 "arm" :obj-2 "shaft")
(show-action :action "open" :object "door")
```

Model worlds

In order to produce 3D animations in which objects of a certain domain appear and act, CATHI uses descriptions of the geometry of these objects as well as descriptions of the actions they can execute. The geometries are given in the form of polygon models, more precisely in the OOGL language defined in (Phillips 1996). Objects are identified by their names, and while their sets of polygons are needed for the production of the final animation, CATHI's script generator only requires object names and a raw abstraction in the form of a bounding box plus a few material properties.

Actions, which can happen in the model world, are described by their corresponding object motions (i.e. translations, rotations, ...). Although other types of actions (e.g., lighting up a lamp) are not yet implemented, it is relatively straightforward to represent and treat them in a similar way.

Presentation parameters

$\mathcal{PPP}$'s presentation planner also specifies generation parameters for the realization modules, such as the target language for the text parts, visual complexity of illustrations, duration and a set of stylistic restrictions for the 3D animations. It can, for example, advise CATHI to use or not to use certain illustration techniques, such as exploded views, depth of field effects, color, opacity or lighting effects.

Known resource limitations

One final piece of input information is the graphical capacity of the output medium, such as the availability of color and opacity control or types of light. If a certain feature (e.g., spotlights) is not available in the output medium (e.g., a black and white wire frame display), the visual effects that rely on this feature cannot be used in the animation clip and must be substituted, if possible, by other effects. This kind of limitation is called *known resource limitations*, in contrast to *arising resource limitations*, which will be discussed later.

System output

The output of the system is produced in two steps. Most of the work goes into the generation of an animation script in a neutral, i.e. not renderer-specific, description language.

Animation scripts

According to a widely accepted position (Badler, Barsky, & Zeltzer 1990; Badler, Phillips, & Webber 1993; Karp & Feiner 1993; Thalmann & Thalmann 1993; Watt & Watt 1992), animation scripts are structured hierarchically. The leaves of the script tree are called *elementary sequences*. They are lowest level descriptions of object or camera motions during a certain time span of the entire animation time. An example for such an elementary sequence is the following:

```
(:translate-object 1.0 7.0
  :object "shaft-1"
  :startposition (71.0 79.5 30.0)
  :endposition (71.0 -451.72 30.0))
```

This sequence specifies that object 'shaft-1' should translate from position (71.0 79.5 30.0) to position (71.0 -451.72 30.0) from time 1.0 sec. until 7.0 sec.. Likewise, all the object and camera motions, as well as light positions, settings, object colors and opacity are specified by elementary sequences. On the way from the leaves of the script tree towards its root, these elementary sequences are grouped into sets of sequences, which appear parallel (overlapping) or sequentially in time. An example:

```
(:parallel
  (:translate-object 1.0 7.0
     :object "shaft-1" ...)
  (:translate-object 1.0 7.0
     :object "shaft-2" ...) ...)
```

The resulting groups are called the *non-elementary sequences* and are grouped again in the same way, until the top-level group keeps all the sequences of a script together.

Animation clips

The scripts described above are translated into renderer-specific animation descriptions. Currently it is possible to generate output for the visualization tool Geomview and in the RenderMan graphics description language. The renderers then produce a series of single frames, which build up the animation. In the case of Geomview, the animation is displayed directly on the screen in more or less fluent motion, depending on the specific 3D models and the graphics hardware. In the case of RenderMan, the images are stored in files and converted into an MPEG movie file after the rendering process is finished. While the production of animations in real time only works with Geomview as the output renderer, RenderMan allows a much wider range of graphical effects. This enables the generation of much more expressive animations in situations where real-time presentation is not required.

The generation process

In order to generate the kind of animation scripts described above from communicative goals and model worlds, CATHI uses a context-guided expansion and decomposition strategy.

The context

The context, which guides the generation process, contains information about the actual state of the model world (e.g., current camera and object positions), information about the generation parameters (e.g., shall we use light effects?) and limitations of the output medium. In addition it keeps track of the time passing in the script, in the generation process and in the presentation, which is possibly already running.

This information allows the script generator to make decisions about how (by which visual effects) to achieve a certain communicative goal. It allows decisions about cuts and camera motions and enables the generation of computationally cheaper solutions, as soon as generation time becomes scarce.

The script grammar

The structure of the animation scripts produced by CATHI is defined in a set of decomposition rules, that represent a context-sensitive grammar. The initial communicative goal triggers a decomposition rule, which specifies, depending on the context, how the goal can be achieved by a set of animation sequences. These sequences can be specified as either parallel (overlapping) or sequential in time, and they trigger their corresponding decomposition rules in turn. The following example of a simple decomposition rule specifies that an object motion is shown by moving the object while keeping camera and light settings fixed.

```
(defrule show-object-motion (object motion)
  (parallel
    (move-object :object object
      :motion motion :duration duration)
    (steady-shot :duration duration)))
```

Three parameters are implicitly declared for every rule: start-time, end-time and duration. Only one or two of them have to be specified, as long as this specification is unambiguous. If, for example, only the duration is given, start-time defaults to the current time and end-time is computed accordingly. Overlapping subsequences can be declared as parallel sequences with different starting times and/or durations.

At every level of decomposition, different subsets of sequences can be selected, depending on the current state of the context. The next example shows a decomposition rule, which selects different sets of subsequences, depending on the context state.

```
(defrule show-relation (object1 object2)
  (if (feature :use-light-effects)
      (shine-from-object-to-object
        :object1 object1 :object2 object2
        :duration duration)
    (move-camera-from-object-to-object
      :object1 object1 :object2 object2
      :duration duration)))
```

At the leaves of the decomposition tree the corresponding rules don't decompose the sequences any further, but instead directly specify the corresponding elementary script sequences (in LISP macro syntax).

```
(defrule keep-camera (transform)
  `(:keep-camera ,start-time ,end-time
                 :transform ,transform))
```

The expansion is done in a depth-first order and chronologically, so that the generation of early elementary script sequences can modify the context according to their content. A camera motion, for example, will change the current camera position stored in the context, so that later decomposition rules can decide, e.g., whether it makes sense to move the camera to a new position smoothly, or to make a cut.

Incrementality

The decomposition strategy above produces the information that a certain portion of the script is completely specified (and won't be changed by later generation) as a side effect. This information is used to forward the piece of script that has just been generated incrementally to the translation and rendering processes.

Every time such an *increment* is generated, it is passed on for presentation, which means that the presentation begins immediately after the generation of the first increment of a script. This overlapping of generation and presentation time shortens the delay between script generation and presentation drastically. The timing problems that can arise from this procedure will be discussed shortly.

Procedural attachment

Some information necessary for specifying an entire animation script is not declared explicitly in the script grammar. It is computed by procedures called by the generation process during the evaluation of decomposition rules. Such calculations include the choice of camera and light coordinates, the identification of 3D object overlappings or the object motions needed for an exploded view.

The calculation of a global scene illumination, for example, is also done by such a procedure. It positions some light sources in the scene, generates some test frames, evaluates them for brightness and adjusts the

light sources accordingly. In addition, spotlights can be positioned to illuminate certain parts of the scenery.

Arising resource limitations

After the first increment has been generated, the presentation of the animation is started. As long as the generation process keeps step with the rendering and presentation process, and supplies a new increment each time the former one has been presented, CATHI produces fluent animation output. As soon as the generation of an increment takes longer than the time left until the scheduled start of its presentation, output will pause until the generation is finished.

To avoid this phenomenon, each time an increment is sent to the renderer, CATHI stores information about the current generation and presentation time in the context. This information is used to select computationally simpler decompositions or attached procedures in order to finish the next increment in time. This kind of resource limitations are called *arising resource limitations* in contrast to the *known resource limitations* discussed earlier.

Timing effects

The consideration of time constraints in the generation process makes the system output depend on the underlying hardware, as well as on the current machine load and other factors. The faster a machine is (or the longer time a presentation is allowed), the more complex the animation scripts tend to be. This fact makes the results of the system vary with the machine it runs on. As a result a fluid animation output is achieved on any type of machine from '486 PCs to SGI workstations – hence the name 'anymation'.

Backtracking, Look-back and Look-ahead

Although the decomposition strategy described above works recursively, it doesn't involve any backtracking in the sense of trial and error. Taking back earlier decisions would impair the ability of the system to decide at what point an increment has been completely generated. The generation context stores not only information about the current state of the world, but also about the state before the last modification of each parameter. This 'look-back' by one step can be used to simulate a 'look-ahead' of one step, too. On one hand, these facilities are quite limited and the underlying philosophy of the generation process is 'to make the best of the current situation' without taking back earlier decisions. On the other hand, it is the responsibility of the grammar author – who has a complete overview – to encode the idioms of the film language (see (Arijon 1976)) properly and to take care of, for example, line-crossing errors or continuous movement.

The presentation process

There are two different ways to transform the scripts generated by CATHI into usable animation clips. They are implemented in the respective output modules for Geomview and RenderMan. The Geomview program directly supports a limited set of animation commands. It is, for example, possible to make objects, or the camera, translate or rotate by a certain amount within a certain time span. These commands can be used to animate elementary script sequences describing object or camera motions. In principle each elementary sequence can be translated into one or several commands in the target description language. Thus the animation can be specified sequence by sequence, leaving the rest of the work to the renderer. While this procedure is adequate for description languages with basic support for animation, such as the Geomview command language (Phillips 1996) or VRML 2.0 (SGI 1996), things are more complicated in the following case: The RenderMan interface, as described in (Upstill 1990), is a static description format for 3D scenes (like Postscript for 2D). It completely specifies the spatial constellation of a single frame, but doesn't include any animation support. This situation makes it necessary to completely specify each single frame of the animation. However the resulting process can still be carried out incrementally and is in principle nothing more than what systems like Geomview do by themselves. Despite the fact, that rendering RenderMan files is currently not possible within reasonable time limits for real-time applications, the procedure fits well with the incremental design of the whole CATHI system.

Other Properties of the system

The current implementation of the CATHI system possesses several other properties, which didn't fit directly into any of the above sections.

Object abstractions

From VRML and animation systems we have learned that a fully detailed representation of an object does not always yield the best results. In VRML, full object geometries can be replaced by simpler ones depending on the distance between the object and the camera. Since simpler geometries are computationally cheaper, rendering speed and frame rate increase and result in a more fluent animation. In CATHI, this strategy of object abstraction is further elaborated. There is an elementary sequence type which controls the level of detail of objects. The decomposition rules of the script grammar specify whether certain objects in the model world shall be shown detailed or at a higher abstraction level. In addition to speeding up the rendering process,

this strategy makes it possible, for example, to direct the viewer's attention away from unimportant objects (shown abstracted) towards the 'main actors' which are shown detailed. The object abstractions needed for this technique are derived automatically from the original models. For a closer description, see (Butz & Krüger 1996).

Metagraphics

One interesting technique in 3D animations is the use of metagraphical objects. These can be short textual annotations or arrows pointing to objects. The arrows and strings are placed in the 3D object world, and their positioning is computed by attached procedures in the same way as camera positions or object occlusions. Metagraphics in 3D animations represent a powerful extension to the classical film language.

Different grammars

By changing the set of decomposition rules used by CATHI, different styles of generated animation clips can be selected. In the current implementation, there are, for example two different grammars. The 'standard grammar' produces simple, straightforward, easy-to-understand explanatory animation clips. These fit well in technical documentation.

The second grammar contains decomposition rules for the same top-level presentation goals, but it specifies different decompositions and calls attached procedures with different parameters. The resulting animations have a 'fresh' design with fast cuts, quick camera motions and many visual effects, reminding the spectator of MTV video clips. Each of these grammars consists of approximately 70 decomposition rules.

Implementation

The system is implemented in Common LISP and CLOS and runs on several UNIX platforms from LINUX PCs to SGIs. For testing and development there is a small user interface which allows the formulation of communicative goals as the system input, and the setting of generation parameters. One interesting aspect of the implementation is that the decomposition rules of the animation grammar are transformed into an internal procedural form, which can be compiled by the LISP compiler. This speeds up the generation process substantially and allows the system to generate much more complex animation scripts within a limited time.

Examples

The following image sequences show snapshots from simple animations generated by CATHI. The communicative goal of the first animation is to show how to

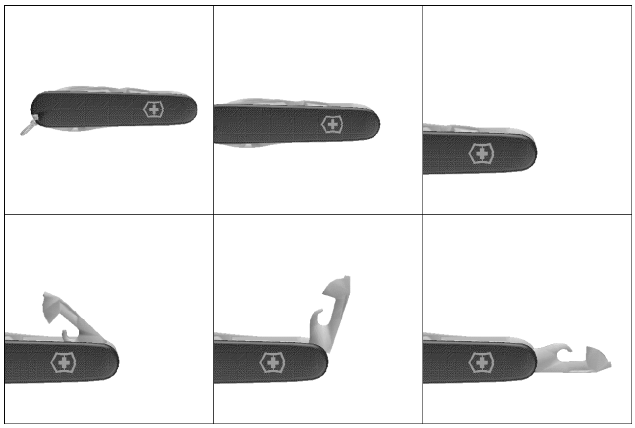

Figure 1: Show how to open the can opener

open the can-opener of the swiss knife. Initially, the whole knife is shown in the frame in order to establish the visual context. Then the camera position is adjusted in such a way that the ensuing object motion can be entirely viewed using a stationary camera.

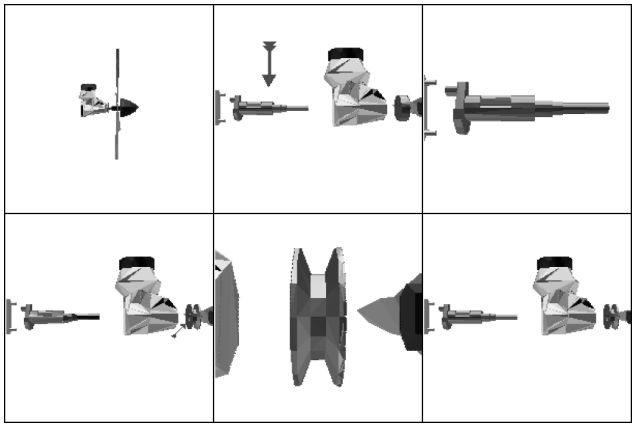

Figure 2: Show the parts of a constructive group

In the second sequence, the animation shows the parts (two in this case) of a constructive group by first creating an exploded view and then zooming in and out on the single objects. While these are very simple examples, CATHI is able to generate much more sophisticated animations, using spotlights or depth of field effects, which unfortunately can not be shown here by a few small-sized keyframes.

Related work

A different solution to a similar task is presented in the work of (Karp & Feiner 1993). Their ESPLANADE system generates animation scripts using a hierarchical planning approach. The generation of the scripts is

done in a breadth-first order, which allows for a complete look-ahead and look-back, but not for incremental generation. Other work, such as in (Badler, Barsky, & Zeltzer 1990) or (Badler, Phillips, & Webber 1993) concentrates more on motion planning of objects in the model world, but relies heavily on defaults for the camera and light settings. In contrast to this, the choice of camera perspective for statical graphics is quite well examined in the work of (Rist & André 1990). I have adopted their basic strategies for the selection of camera positions and modified them, if necessary, in order to make the system work in real-time.

Basic issues for cutting and camera settings in animations are discussed in (Karp & Feiner 1990). These issues have largely been observed in the formulation of the current script grammars. Finally the work described in (Christianson *et al.* 1996; He, Cohen, & Salesin 1996) is inspired much by the same idea, namely the formalization of a certain subset of the film language (as described in (Arijon 1976)) in a declarative way, but while their system emphasizes on camera planning with aspects of story-telling, their formalization fails to yield a fitting light setup and to make use of light effects or animation-specific techniques, such as spotlights, metagraphics and levels of detail.

Future work

There is at least one possible extension that seems interesting at the current state of implementation.

Interactivity

If the output medium allows a direct interaction with the spectator, it would be interesting to make use of this feature. A newly generated animation could, for example, start from the 3D constellation left-over by former user interaction, instead of choosing a new starting position by itself. This can easily be realized by changing the script grammar in such a way that every time a new animation clip is started, the current 3D state is read out from the graphics system instead of resetting it. The selection of objects in the graphics window could be used to start new animations. If the spectator clicks on an object, an animation could be started, showing possible actions with this object. These extensions of the system go beyond its current use in $\mathcal{PPP}$, but could be interesting in the future.

Acknowledgements

The work presented in this article was funded by the Ph.D. program 'Landesgraduiertenförderung des Saarlandes' at the University of Saarbrücken. The Geomview visualization software was developed at the Geometry Center of the University of Minnesota, in Minneapolis.

References

André, E.; Müller, J.; and Rist, T. 1996. Wip/ppp: Automatic generation of personalized multimedia presentations. *ACM Multimedia* 407–408.

Arijon, D. 1976. *Grammar of the Film Language.* Silman-James Press.

Badler, N.; Barsky, B.; and Zeltzer, D. 1990. *Making Them Move: Mechanics, Control and Animation of Articulated Figures.* Morgan Kaufmann.

Badler, N. I.; Phillips, C. B.; and Webber, B. L. 1993. *Simulating Humans. Computer Graphics Animation and Control.* Oxford University Press.

Butz, A., and Krüger, A. 1996. Lean modeling - the intelligent use of geometrical abstraction in 3d animations. In Wahlster, W., ed., *Proceedings of ECAI 96.* John Wiley & Sons, Ltd.

Christianson, D. B.; Anderson, S. E.; wei He, L.; Salesin, D. H.; Weld, D. S.; and Cohen, M. F. 1996. Declarative camera control for automatic cinematography. In *Proceedings of AAAI '96*, 148–155.

He, L.; Cohen, M. F.; and Salesin, D. H. 1996. The virtual cinematographer: A paradigm for automatic real-time camera control and directing. In *Proceedings of SIGGRAPH '96.*

Karp, P., and Feiner, S. 1990. Issues in the Automated Generation of Animated Presentations. In *Graphics Interface '90*, 39–48.

Karp, P., and Feiner, S. 1993. Automated Presentation Planning of Animation Using Task Decomposition with Heuristic Reasoning. In *Graphics Interface '93*, 17–21.

Phillips, M. 1996. *Geomview Manual.* distributed with the software at http://www.geom.umn.edu/.

Rist, T., and André, E. 1990. Wissensbasierte Perspektivenwahl für die automatische Erzeugung von 3D-Objektdarstellungen. In Kansy, K., and Wißkirchen, P., eds., *Graphik und KI*, volume IFB 239. Berlin, Heidelberg: Springer. 48–57.

SGI. 1996. *VRML 2.0.* Specification by Silicon Graphics Inc.: http://vrml.sgi.com/moving-worlds/.

Thalmann, N. M., and Thalmann, D., eds. 1993. *Models and Techniques in Computer Animation.* Computer Animation Series. Springer.

Upstill, S. 1990. *The RenderMan companion.* Addison-Wesley.

Watt, A., and Watt, M. 1992. *Advanced Animation and Rendering Techniques, Theory and Practice.* Addison-Wesley.