Grammar-based incremental generation of 3D-animation scripts and its generic implementation in LISP

Andreas Butz, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, butz@dfki.uni-sb.de

October 27, 1995

Abstract

This paper describes the automatic generation of 3D animation sequences within the CATHI [1] system, expressing given communicative contents based on a grammar-like system of visualization rules.

It emphasizes the view of animation generation as a process that bears similarities with the generation of natural language and allows, to a certain extent, the transfer of ideas and methods from this field.

The generation process works incrementally to allow an interactive presentation system to generate animations and show them with a limited delay, which is important for short system response times. We will show which type of incrementality is supported by the inherent structure of animation scripts and describe an efficient and generic implementation of the generation process in LISP.

Depending on the available graphical output devices and facilities the system may produce animation scripts for simple wire-frame graphics or sophisticated lighting and shading models. The availability of such features as color or spotlights is already considered in the process of script generation. This allows optimal use of the available expressiveness, thus making CATHI a resource-adaptive system.

[1] Computer Animation Tool for Help and Information systems

1 Why do we need this?

The analysis and generation of natural language is a well recognized subject in the research area of Artificial Intelligence. It has even spawned a new research area between computer science and classical linguistics: the computational linguistics. The generation of syntactically correct and meaningful sentences or coherent texts is based on grammatical models and rhetorical theories sometimes dating from the time long before computer science and there exist well investigated theories of syntax, semantics and pragmatics of natural language.

The increasing graphical capabilities of available computer hardware and the resulting tendency towards the use of multimodal presentation of information give rise to a need for generation of not only text, but also graphics and animation. The problem is that the structural rules of visual communication are far less formalized than those of verbal communication. Nevertheless we will see throughout the text that it is possible to formalize several aspects of this communication in a grammar-like system and to generate animation scripts from this grammar.

The generation process proposed here works partially incrementally and allows to start the rendering process as well as the output of the animation while further parts of the animation script are still being generated. This strategy is also applied in newer NL generation systems [KF95], that face the need to generate output under real-time conditions, and it makes the most important difference between CATHI and other existing animation generation approaches [KF93, Bu94a, Bu94b].

The CATHI system is being developed in the framework of the $\mathcal{PPP}$ system, an intelligent multimedia presentation system [Aa93, An93] that presents information about the usage and function of technical devices and produces interactive operating instructions that are adapted to the person interacting with the system.

1.1 Why create it from scratch?

In the early days of language generation, texts were simply chosen from preformulated sentences in a database, thus allowing a quick composition of small dialogs. This approach suffices for simple dialog situations with few possible variations and a limited set of presentation situations, but its main drawback is the lack of flexibility. As soon as the information to be presented becomes more complex and/or the presentation situation is subject to

changes, there's a need to formulate utterances from scratch dependent on the respective communicative context. The base data for this text generation is always the communicative content of the intended utterance as well as a more or less sophisticated grammatical model of the language used.

The generation of 3D animation as an expressive medium hasn't yet reached this elaborate level. Most of the existing information systems are restricted to the use of preproduced animation sequences corresponding to the canned text in early language generation and this suffers from the same disadvantages, namely the lack of flexibility and adaptivity to the communicative context. The main reason for this situation can be seen in the relative newness of computer animation compared to human language as a communication medium, and in the absence of grammatical models and theories for this kind of visual communication.

1.2 Is there an animation grammar?

On one hand, this situation makes it difficult to generate syntactically correct and meaningful animation sequences since there's no exhaustive and well defined syntax or semantics of 3D animation (or even filmmaking), but on the other hand it simplifies the process, because as long as an animation sequence clearly communicates its message there's not much restriction on how it is done. Faults in visual communication aren't judged with the same severity as in verbal communication, in the sense that there are few things definitely wrong in the generation of an animation[2]. However, there are certainly different ways of presenting an information and some of them look better or worse and support the communication more or less.

Thus the kind of grammar for 3D animations used in this work doesn't claim to be a sound or complete description of animated visual communication, but it's meant as a formalization of some possible ways of communication. It would for example never be possible to parse films or animations with such a grammar, since, as stated above, in a certain sense everything is possible in visual communication and there's no exhaustive description of all the possibilities to communicate a certain message, but the visualization rules used by the CATHI system describe a certain subset of these possibilities thus allowing the system to formulate animation sequences to transport the communicative contents.

[2]For a discussion of things to avoid see [Ka90]

2 Animation production and language generation

The 3D animation sequences currently used in multimedia systems for information presentation, help systems or education are preproduced sequences that were designed and scripted by humans and whose images were then generated by animation rendering systems. Normally this production process is achieved in certain steps [Ge93, Ho94].

2.1 The communicative content (what to say)

First the designer of the animation or of the multimedia document decides what he/she wants to communicate through the animation sequence. This decision (what to say) forms the basis for the creation of the text, graphics and animation as well. It finds its direct correspondence in the separation of NL systems into what-to-say and how-to-say components.

In NL generation systems the what-to-say problem can be solved by text-grammars or text-planners that rely for example on rhetorical structure theory [An93, An95]. This approach can be adapted to the case of animation sequences as well. In our case the main presentation planner of the $\mathcal{PPP}$ system decides on the contents to be communicated in different parts of the presentation and their organization. Thus the decision what to say is already made when the generation of the respective animation sequence starts and the task of CATHI is to decide on the exact form in which the given contents are expressed in a 3D animation sequence, i.e. to formulate the animation.

2.2 Exposé, Storyboard and Script (how to say)

The next step in the conventional production of an animation is the writing (or at least mental formulation) of a description of the logical progress of the sequence and the coordination of the actions shown. Eventually this description is refined into a storyboard, a collection of images and text that already show key scenes of the final animation and describe what happens between these snapshots. Based on this storyboard a human animator defines the positions and movements of objects in the scene as a function of time. This is done either in a scripting language or interactively in a scripting system. The exact form and the level of detail of this script depend on the respective animation software, but the process always presumes explicit decisions about camera positions and settings, lighting setup, object positions and movements as well as their coordination with each other.

Transferred to the situation of automatic generation, this could for example mean that the $\mathcal{PPP}$ presentation planner orders a sequence showing the location of a certain part of a technical device and CATHI will transform this into a script for an animation showing the whole device and then zooming into the part in question. The necessary camera positions and the light setup are computed on the basis of the given geometrical models while the syntactical aspects of the sequence such as visual continuity are encoded in the generation rules of the animation grammar.

At a certain level of abstraction one can say that the formulation of all these aspects of an animation corresponds to the surface generation of a natural language utterance. The decisions about explicit positions and settings of camera and lights corresponds to the choice and flection of words and the coordination of these decisions can be defined in a certain way by a grammar in both cases. While a NL grammar describes such aspects as congruence, dependency and logical focus of a sentence, an animation grammar must treat aspects like visual consistency of a sequence of shots and visual focus in a shot. Some of these topics are discussed in more detail in [Ka90]. However the inherent structure of animation descriptions differs substantially from the structure of natural language, which makes the application of most NL generation approaches impossible.

3 How can we describe animations?

Whereas natural language structures can be suitably described by feature structures defining all their syntactical aspects (by either filling in slots explicitly or defining coreferences between them), this form of description is not particularly useful for the structure of animation sequences. The most important and obvious reason is a strong need for arithmetic in geometrical and temporal descriptions not only dealing with linear precedence, but with duration and arbitrary temporal relations. This arithmetic can neither be expressed nor be efficiently computed in conventional feature structure formalisms and unification grammars.

Instead, we will use a description of animation sequences which emphasizes a certain way of looking at their structure which is proposed throughout the current literature. This method will allow the description of arbitrarily complex animations as well as their generation in an efficient but still very flexible manner.

3.1 The script tree

An animation script in the CATHI system is described by a tree, built of subsequences. The root of the tree is the top-level sequence while the leaves are the elementary subsequences which explicitly define the positions, properties or movements of the camera, objects or lights at given intervals of the overall animation time. Each non-elementary sequence contains a set of subsequences and classifies their temporal relation roughly as either parallel or sequential (We will see later why this is useful and necessary).

By describing animation scripts in the form of a tree we emphasize the view of animation descriptions as hierarchical structures. Depending on the point of view, one can say that they either abstract from explicit geometrical specifications to more global descriptions terminating in the top-level content description (bottom-up view) or refine a given top-level description into more detailed descriptions of its subparts resulting in the explicit specification of geometry (top-down view).

This way of looking at animation descriptions as essentially hierarchical structures is proposed by several people working in the field of computer graphics, especially in [BBZ90, Ba90, BPW93, KF93, Th90, Fo90, Wa90] and seems to have become the consensus. It also formed the basis for the earlier work on the BETTY system in [Bu94a, Bu94b], although the concept and its consequences weren't so explicit at the time.

Another advantage of such a description is the canonical method of generating it by a grammar. Such a grammar must only specify types of nodes and the splitting of the tree at these nodes, i.e. the subsequences of a certain type of sequence and the relations between them, either in time or concerning other aspects like objects, positions and properties.

Dependencies between more than two levels of subsequences can be handled by explicitly specifying them in all the intermediate levels. This mechanism is not, for example, as general as the arbitrary relations throughout whole structures that can be expressed in a formalism like the Relational Language System [Wi92], but it saves computational complexity in the generation process and strongly supports the efficient generation and partially incremental processing of script trees.

3.2 Movements and alterations

Object and camera movements or alterations of their properties (size, color, shape, texture, ...) are usually described in one of two ways in conventional

animation systems: Either their start and end states are defined and intermediate states are computed by linear interpolation, or the whole motion or alteration is described by a process or data describing the motion path (e.g. a movement function or a spline).

3.2.1 Keyframing

Keyframing is the simpler solution of the two since it normally needs less data, but since the intermediate positions are only interpolated without the direct influence of the animator, complex motions either need many keyframes or are even impossible to describe. Therefore keyframing is especially useful for simple movements or alterations such as a static camera position, a zoom or for simple camera motions like dolly shots [Ho94]. The animation grammar rules used by CATHI use keyframing to describe camera positions and settings.

3.2.2 Motion-path

The more complex motions of objects in the scenery, possibly describing complex mechanical dependencies, are best modeled using more sophisticated motion descriptions, for example splines. These motion descriptions are not computed by the animation planning process, but must be modeled in the same way as the object geometries since they are in a certain sense part of the modeled world. The purpose of the CATHI system is not to simulate complex physical processes, but to show given objects and actions in an effective and unambiguous way. Object motions are therefore described by their corresponding splines in the animation scripts generated by CATHI and in its model world.

On the other hand such motion descriptions, at least in the case of technical devices, can often be obtained from the construction process. This also holds for the object models. In fact, we've provided a way for the CATHI system to include models from a common CAD data format which is supported by several (if not most) CAD packages. This will allow the use of a wide variety of domains for testing and evaluation of the CATHI system in the future.

4 The quick generation of animation scripts

If one wants to generate adapted multimodal presentations in an interactive system, these presentations and their parts must be generated in real time or at least with a delay that isn't apparent to the user. Thus there is not much time to investigate many different designs, span a search space or do other time intensive tasks. The generation process has to be efficient and quick on one hand, to keep the system working interactively, but on the other hand it has to be as flexible and powerful as possible to be able to fulfill the needs of the specific presentation situation.

4.1 Incrementality and search space

When an animation has to be generated and shown immediately, ways must be found to reduce the processing time apparent to the user. This can be achieved by an incremental generation strategy in which the already generated parts of an animation script are processed by the graphical realization component (renderer) while the later parts of the script are still being generated. The concept of incremental generation is used successfully in the real-time generation of natural language [KF95] and can be adapted to the generation of animation descriptions as well. Ideally the processes of script generation and realization should run with about the same throughput and the steps (increments) in which the script is passed on should be as small as possible. (In practice there are certain limitations to this which we shall see soon.)

In most of the cases there are many different possibilities for presenting certain information. Several of these solutions may be formalized in the set of grammar rules, but only one of them can be chosen to create the final animation. This decision has to be made at a certain point in the generation process and cannot be reversible, otherwise we would lose the ability to process the beginning of a script further while the end is still being generated because this beginning could contain invalid subsequences that are retracted by later generation. This is the obvious reason why a global backtracking strategy with a large search space as in a conventional operator-based planning system would usually be a bad idea for the quick generation of animations.

In practice there is one problem with this concept of incremental generation which arises from the twodimensional structure of the script tree: If any sequence of the script, especially the top-level sequence, contains subsequences that may be parallel in time, there is no easy way to decide if the

generation process has already completed a certain temporal portion of the sequence unless all the subsequences are generated.

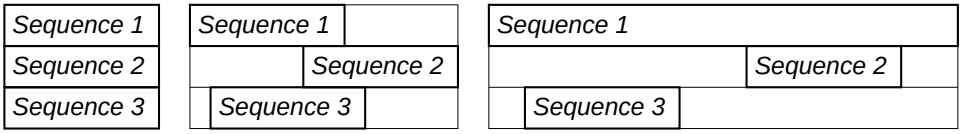

Figure 1: different kinds of parallel subsequences

For example: If from time a to time b an object is moved and the camera and the lights track its motion, these 3 subsequences all have to be generated all before they can be passed on to the realization process and it must be ensured that there won't be another subsequence generated later that would describe things in the same time.

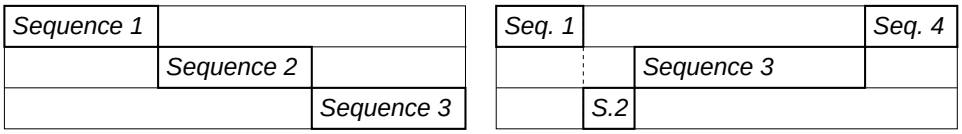

Figure 2: different kinds of strictly sequential subsequences

Consequently forwarding subsequences to the renderer one after another is only possible if they are strictly sequential in time (which is the reason for putting this information into the script as stated earlier). It is the responsibility of the grammar-writer to write visualization rules that use strictly sequential subsequences at the higher levels of the script to support the incrementality of the system.

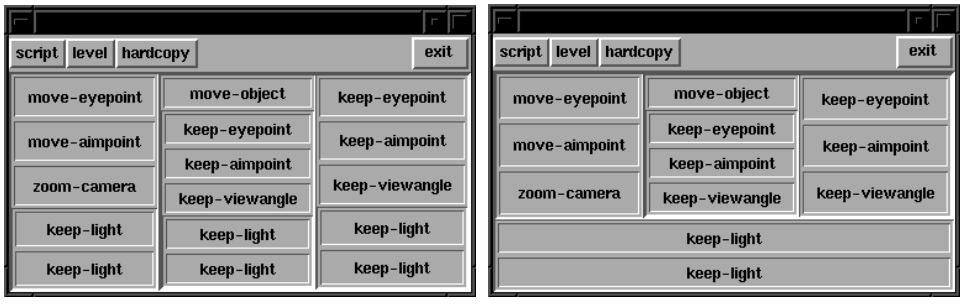

Figure 3: good and bad representation for incremental processing

The worst case for our generation strategy would be a top-level script containing parallel subsequences which have the same duration and start time as the whole animation, which would completely disable any incremental forwarding and processing by the renderer. These cases can be avoided by only specifying parallel subsequences at lower levels in the script tree. A simple example of two different scripts describing the same animation is shown[3] in figure 3. Although the description at the right looks more elegant and efficient, its generation totally prevents its incremental processing, since the subsequences describing the light positions are generated only at the end, but start at the same time as the whole animation. So there is no possibility to start the rendering process before the generation is completed.

In the script at the left side the description of the animation progress is complete for the first third of the animation as soon as this first third is generated. It can therefore be passed on to the rendering process, while the generation of the second third of the script continues.

This phenomenon is another important difference between the structure of animation scripts and the structure of natural language. While spoken or written text extends only linearly in time, animation scripts extend in two dimensions in the sense that they may contain parallel actions.

4.2 Geometrical computations

Just as a text generation system has to compute exact verbal forms from syntactical descriptions in the last step of the generation of the text surface, an animation generation process must eventually provide explicit geometrical descriptions for details of the animation. This means camera positions and settings, light positions, colors and intensities and object positions and properties. This geometrical data is dependent on the geometrical models in question and it can therefore not be included in the grammar definition, since one animation grammar should yield scripts for many possible domains.

The solution to this is the definition of procedures that compute the actual values from the model world and that operate on the geometrical models. These procedures are used as black boxes in the generation process and for the present it suffices to know that, for example, there is a way to compute a camera position in front of an object or a focal length that shows the entire

[3]The images shown are hardcopies produced with SCRED , a simple display tool that was written for the display of hierarchical animation scripts. The metaphor used for this twodimensional display stems from SYMBOLICS' S-DYNAMICS animation software. Time is on the horizontal axis, while parallel sequences are distributed over the vertical axis.

object filling the frame from a given position. How these procedures exactly work is described in [An90, Sc94] and the algorithms were the subject of much work in our earlier projects. The procedures used in the CATHI system are based on these earlier investigations, but modify the algorithms where the need for quick decisions prevents complex computations.

The values produced by these geometrical computations constitute the key values for the elementary subsequences that describe the animation details. This way of describing camera and light setup by keyframing has proved to be not only very efficient to generate, but also powerful enough to produce good looking results.

4.3 Context management

In addition to the actual geometrical models, the current state of the scenery (as the result of the generation until that time) is considered in the generation process. In a context database the generator keeps symbolic information about the scenery (for example which objects can be seen from the current camera position) as well as explicit geometrical data such as object and light positions. This context is modified and kept up to date whenever a geometrical procedure is called (for example, a change of the camera position implies most probably also a change in the set of visible objects), but its changes can also be explicitly specified in the grammar rules.

This concept of a symbolic description of the scene at the current state of generation yields the basis for the generation decisions. Grammar rules can be specified to produce different results dependent on the actual context entries and thus, in different situations, the application of the same grammar rule may result in completely different sets of subsequences.

Since the generator can't retract any earlier decisions (for the reasons stated above), it always tries to make the best out of the current situation and care must be taken to consider the different possible situations (which is the same problem as in NL processing).

4.4 The use of given resources

The context management just described allows yet another degree of flexibility and adaptivity in the generation process: If the context database contains information about what graphical facilities are available, the animation script can be generated accordingly. As an example consider the following situation: If the system has to generate animations for an output device with

high graphical capacity and the context database contains this information, CATHI can decide to make use of this capacity and use, for example, spotlights to support the visual focus in the scenery, or color changes to express temperature changes.

On the other hand under heavy system load another process could make an entry to the context database and the generator could decide not to use shaded graphics, but to fall back to simple wire frame output in order to keep step with the speed of the output device. All these dependencies can be expressed by grammar rules that check the corresponding entries in the context and yield different scripts or subsequences depending on this information.

5 Architecture of CATHI

The CATHI system consists of several components as shown in figure 4. The primary input to the script generator are the communicative contents of the animation supplied by the main $\mathcal{PPP}$ presentation planner (from left). The generation process uses the set of grammar rules and the context database (which it also modifies) to generate an adapted animation script for the rendering control process (to the right).

Geometrical calculations are carried out by a set of algorithms operating on the 3D models and the graphics system and their results are used as parameters and key values in the animation scripts. The generated scripts are forwarded to a rendering controller that directly controls the graphics and rendering package as well as the timing of the animation. This component is needed to separate the generation process cleanly from the graphical output and to provide some buffering between the two.

6 The implementation in Common Lisp

The CATHI system is implemented in COMMON LISP. The guiding principle behind the implementation of the CATHI system was to obtain maximum efficiency in the generation process while still using a declarative description of the animation grammar. In order to achieve this, the declarations of grammar rules are transformed into an internal procedural form. They are managed like LISP functions in their own namespace (i.e. package). This has several major advantages over a declarative representation as data structures.

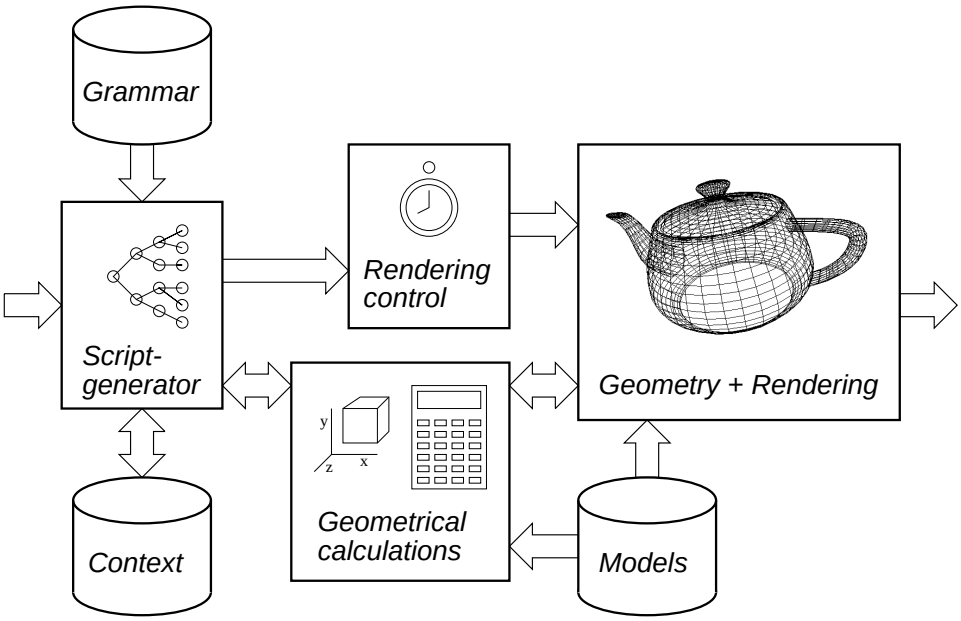

Figure 4: Architecture of the CATHI system

6.1 Efficiency

The organization of the transformed rule base is handled by the LISP interpreter itself which naturally provides an efficient management of function definitions. Most LISP implementations also provide a compiler for LISP code, which now can be used for our grammar rules as well. A compiled grammar generates scripts at a speed that is several times higher than the speed with interpreted code. The exact gain depends on the type of compiler provided. Since the generation time is limited by the interactiveness of the system, this means that we can generate substantially complexer scripts in the same amount of time.

Using these basic LISP functionalities also guarantees a maximum use of the underlying computing power, since every LISP implementation is (or at least should be :-) optimized for the specific machine or operating system.

6.2 Expressiveness

Since the grammar rules are internally represented as LISP functions, it is quite easy to mix these two levels and to integrate LISP constructs into the

declaration of grammar rules, such as local variables, arbitrary arithmetical expressions or the kind of "procedural attachment" that is needed for the use of the external geometrical computations described earlier.

The disadvantage of this expressiveness is that it is possible to write arbitrarily non-declarative grammars if one wants, but to have this possibility and to use it with circumspection seems to be still better than the complete impossibility to express certain things for the pure reason of an elegant formalism.

In this sense the implementation of the script generation process in CATHI follows a very pragmatic approach that will probably provoke some critique from people in the more formally oriented AI community, but the approach is in our opinion fully justified by its effectiveness and speed which allows the generation of much complexer animations in limited time.

6.3 Some examples

The declaration of the animation grammar rules looks very much like the definition of LISP functions, which supports its efficient management and processing in a LISP environment. The given example shows the top-level rule that generates the script on the right side of figure 3.

The rules take two kinds of parameters, some that are declared explicitly like keyword parameters in LISP and can use default values, and three common parameters, that every rule takes: `start-time`, `end-time` and `duration`. Only one or two of the latter must be given as long as they contain the complete information, and the remaining are filled in automatically on the basis of the current planning context and time. If only `duration` is given, the start-time is assumed to be the actual time.

```
(defrule move-camera-to-object-and-move-it-with-static-lights
  (object movement (lights *current-lights*))
  (parallel
   (move-camera-to-object-and-move-it
        :object object :movement movement :duration duration)
   (keep-lights :lights lights :duration duration)))
```

At the beginning of the planning process the whole scenery is shown in the frame and the resulting animation script zooms into the object in question, shows its movement and keeps a still image for some seconds afterwards. During all this time the lighting setup is kept unchanged. Snapshots from a corresponding animation are shown in figure 5. Of course this is a very simple

example while the grammar rules in the working system can become quite complex. To see how a grammar rule can yield different results depending on the context, consider the following (simplified) example:

```
(defrule show-object-motion-dependent-on-features
  (object movement (lights *current-lights*))
  (if *spotlights-available*
      (parallel
       (move-camera-to-object-and-move-it
          :object object :movement movement :duration duration)
       (track-object-movements-with-spotlight
          :object object :movement movement
          :lights lights :duration duration))
  ;else
      (parallel
       (move-camera-to-object-and-move-it
          :object object :movement movement :duration duration)
       (keep-lights :lights lights :duration duration))))
```

This rule shows an object motion and decides to track it with a spotlight as soon as this feature is available. It uses the LISP construct `if` to result in different sets of subsequences depending on the value of the context variable `*spotlights-available*`.

7 The realization of the generated scripts

For the realization of the generated scripts we use a powerful visualization package[4], that is freely available on different platforms and makes full use of their respective hardware capabilities.

Besides the wide availability of GEOMVIEW the main reason for its use is the graphical speed and the clean and simple interface it provides. Since the package wasn't designed specifically for animation tasks, it provides only a minimal support for animation by itself. This gap is filled by the rendering controller (see figure 4) that computes specifications for the single frames and sends them to Geomview for displaying. In principle it is possible to animate the scripts generated by CATHI with every animation system as long as a

[4]The GEOMVIEW package was written at the Geometry Center of the University of Minneapolis, USA, and runs on SGI machines with or without Z-Buffer hardware, on NeXT systems and on almost any machine using X-Windows.

translation to the respective scripting language is provided, since the script trees contain all geometrical and temporal information that is needed to fully specify an animation.

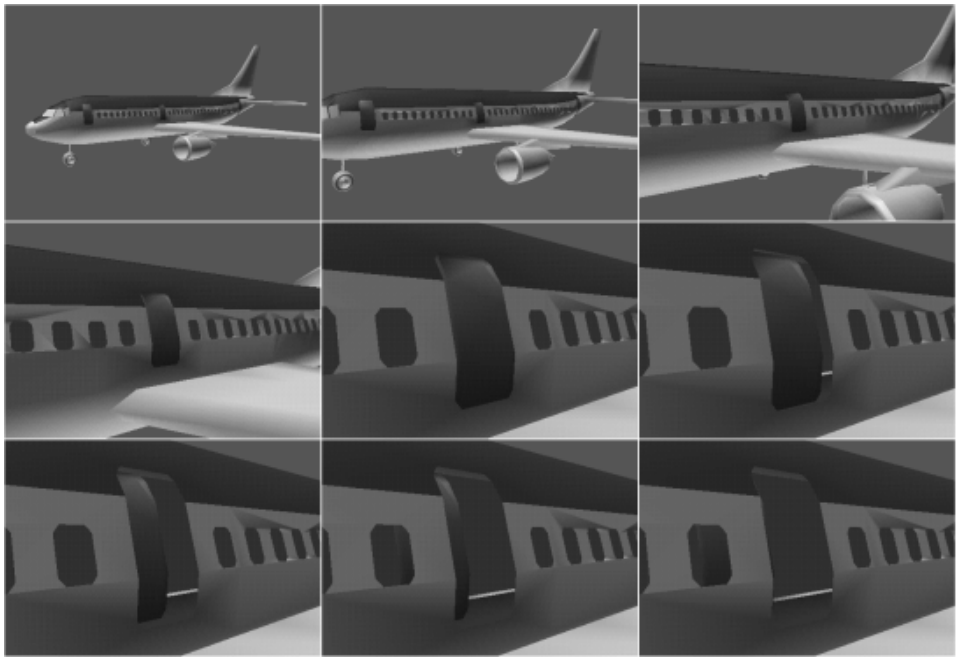

Figure 5: snapshots from a sequence that localizes the door of the plane and shows how it opens

8 Potential applications and portability

The CATHI system was developed using partially commercial software products, and in the framework of an intelligent multimedia presentation system, but this by no means restricts its use in other applications or working environments. The control and management of complex technical devices and installations could, for example, benefit much from the possibility of presenting information in an user-friendly and user-adapted way. As a front end for diagnostic systems, CATHI could for example animate certain information about the location of a failure in an installation or about its reparation.

CATHI has been developed and tested on SUN and SGI machines under ACL and CLISP and on standard PC hardware under LINUX and CLISP.

It makes no special assumptions about any specialized LISP implementation, hardware or operating system (besides the availability of COMMON LISP, GEOMVIEW and the pipe mechanism for their communication) and it is therefore highly portable. Since it is possible to run it completely on shareware systems (LINUX, CLISP and GEOMVIEW are shareware products) there is no need to buy expensive underlying software packages to run this application.

We've also provided a way for the CATHI system to include models in the DXF format, supported by a wide variety of CAD software, and from Autodesk's 3D-Studio format, which seems to be the most widespread format for 3D models found on the Internet. Those models can be converted into the OOGL language used by GEOMVIEW and this allows CATHI to use them as model worlds.

9 Summary

In this paper we have discussed the similarities and differences between the generation of natural language and 3D animation scripts. We have seen that certain concepts from NL generation can be applied with few changes to the generation of animations, while others are not applicable at all because of the inherent structure of animation descriptions.

We have presented the CATHI system, which uses a suitable kind of grammar for animation sequences and generates fully specified animation scripts from top-level descriptions of the communicative content of the animation.

As in the real-time generation of natural language, the need to generate and show animations with a limited delay leads to an incremental processing strategy. We have discussed which kind of incrementality is supported by the structure of animation descriptions in CATHI and how the formulation of the animation grammar influences this property.

Furthermore, the formalism used in the CATHI system supports the flexible generation of animation scripts adapted to external parameters such as, for example, the available output facilities. Finally, we have described a canonical implementation of the generation process in LISP and discussed some aspects of its expressiveness and efficiency.

References

[An90] Elisabeth André, Thomas Rist: „Wissensbasierte Perspektivenwahl für die automatische Erzeugung von 3D-Objektdarstellungen“ in K. Kansy, P.Wißkirchen (Ed.): „Graphik und KI“, 48-57, Springer Verlag Berlin, Heidelberg 1990

[An93] Elisabeth André, Thomas Rist: „The Design of illustrated Documents as a Planning Task“ in: M. Maybury (Ed.): „Intelligent Multimedia Interfaces“94-116, AAAI Press 1993, also as DFKI Research Report RR-92-45

[An95] E. André: „Ein planbasierter Ansatz zur Generierung multimedialer Präsentationen“, Technische Fakultät, Universität des Saarlandes, Saarbrücken, 1995

[Aa93] Elisabeth André, Winfried Graf, Jochen Heinsohn, Bernhard Nebel, Hans-Jürgen Profitlich, Wolfgang Wahlster: „PPP - Personalized Plan-Based Presenter“, DFKI Saarbrücken 1993

[Ba90] Norman I.Badler, Bonnie L. Webber, Jugal Kalita, Jeffrey Esakov: „Animation from Instructions“ in [BBZ90], 50-93

[BBZ90] Norman Badler, Brian Barsky, David Zeltzer: „Making Them Move: Mechanics, Control and Animation of Articulated Figures“, Morgan Kaufmann 1990, ISBN 1-55860-106-6

[BPW93] Norman Badler, Cary Phillips, Bonnie Webber: „Simulating Humans: Computer Graphics Animation and Control“, Oxford University Press, New York / Oxford 1993, ISBN 0-19-507359-2

[Bu94a] Andreas Butz: „BETTY : Planning and Generating Animations for the Visualization of Movements and Spatial Relations“, Proceedings of the 2nd International Workshop on Advanced Visual Interfaces AVI '94, ACM Press

[Bu94b] Andreas Butz: „BETTY : Ein System zur Planung und Generierung informativer Animationssequenzen“, Diplomarbeit Universität des Saarlandes FB 14, 1994, also as DFKI Document D-95-02

[Fo90] James D. Foley, Andries van Dam, John F. Hughes, Steven K. Feiner: „Computer Graphics: Principles and Practice“, 2nd Edition, Addison-Wesley 1990, ISBN 0-201-60921-5

[Ge93] Rainer Geyer, Jörg Sonneborn: „3D Studio Animationsdesign“, IWT Verlag, Vaterstetten 1990, ISBN 3-88322-433-2

[Ho94] Robert Hone, Margy Kuntz: „Making Movies with your PC“, Prima Publishing, Rocklin 1994

[Ka90] Peter Karp, Steven Feiner: „Issues in the Automated Generation of Animated Presentations“, Proc. Graphics Interface 1990, 39-48

[Sc94] Georg Schneider: „TOPAS: Eine Werkbank zur Erzeugung von 3D-Illustrationen“, DFKI Saarbrücken 1994, also as DFKI document D-95-05

[Th90] Nadia Magnenat-Thalmann, Daniel Thalmann: „Synthetic Actors in Computer generated 3D Films“, Springer Verlag, 1990

[KF93] Peter Karp, Steven Feiner: „Automated Presentation Planning of Animation using Task Decomposition with Heuristic Reasonig“, Proc. Graphics Interface 1993, 118-127

[KF95] Anne Kilger, Wolfgang Finkler: „Incremental Generation for Real-Time Applications“, DFKI Research Report RR-95-11, to appear in „Computational Linguistics“ in 1996.

[Wi92] Kent Wittenburg: „The Relational Language System“, Technical Memorandum TM-ARH-022353, Bellcore, Morristown NJ 07962-1910, USA, December 1992

[Wa90] Alan Watt, Mark Watt: „Advanced Animation and Rendering Techniques, Theory and Practice“, Addison-Wesley 1992,
ISBN 0-201-544121-1