The Virtual Cameraman: an Interval Constraint Based Approach

E. Languenou F. Benhamou F. Goualard
M. Christie

Institut de Recherche en Informatique de Nantes
Université de Nantes, Rue de la Houssinière
BP 92208, 44322 Nantes Cedex 3, FRANCE
{langueno,benhamou,goualard,christie}@irin.univ-nantes.fr

April 2, 1998

Abstract

This article presents our current work on a virtual cameraman which provides the user with camera movements satisfying user defined constraints specified in the image space and/or constraints on the objects of the scene.

A first implementation [19] was using a basic interval solver based on recursive subdivision of the search domain. This technique was not efficient enough to tackle complex camera movements. The work presented in this paper focuses on the use of more sophisticated constraint techniques to handle the non-linear constraints related to the modelling of camera movements and screen-space constraints. Specifics of the application have led us to study a particular extension of the core local-consistency algorithm to process universally quantified time constraints. Prototype experiments are implemented in `Declic`, our interval-based Constraint Logic Programming language compiler.

In particular, there is neither keyframing nor interpolation, thereby, for the solutions obtained, the satisfaction of the specified constraints can be precisely characterized. Moreover, the high level programming related to the use of a CLP language allowed us to design a system in which the user can easily devise new image space constraints.

1 Introduction

A simple glance at our world should convince anyone that image synthesis is becoming more and more popular. Compared to image rendering or geometric modelling domains, rather little work has been done to help computer graphics designers to easily create camera movements.

Our goal is to release the computer graphics artist from the need to understand perfectly the various coordinate systems, the interpolation curve concept nor the velocity graph concept. We want to make possible a natural camera movement specification. The Generate-and-Test process (create control points;

test them ; modify control points; repeat) in order to obtain the desired animation (see figure 1) must be avoided for two reasons: first because these actions can be time-consuming and second because the artist could stop this process, not when the desired animation is attained, but when the "fed-up moment" is reached.

After a presentation of related work, we explain the necessary mathematical models for the scene objects, the camera and the treated movements. After this, the description process and the properties used to specify a desired sequence are described. The constraint logic programming language used for the implementation, `Declic` is briefly presented. Finally, we state some preliminary remarks on the necessary design of universally quantified constraints to handle properties that are required to hold on a specified interval of values (e.g. position over time, stable equilibrium in a range of 3D positions, etc.). The paper ends with some future research directions.

2 Related work

2.1 Control and interpolation curves

Most of the work in camera specification is concerned with obtaining continuous movements and smooth transitions (see figure 1) between user defined keyframes [3, 8, 22, 2]. Unified library of camera control[15], physical behavior[23], input device driven control[25] are still very close to the camera representation leading to difficulty for a user (therefore for an artist) to predict the image animation.

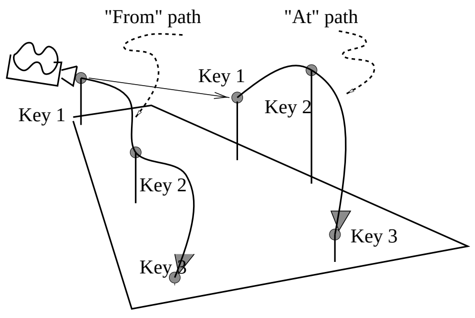

Figure 1: Keyframes and interpolation

2.2 Image space specification

The very first attempt at translating the desired image (image space constraint) into a camera location and orientation were done by Blinn [7]. The work is based on the re-writing of the unknown vectors in term of vectors that are known. His work is rather specific and does not treat multiple constraints.

Another interesting attempt in screen-space specification related work is the *Through the lense camera control* system [16]. The set of accepted constraints includes position of a 3D-point on the screen, orientation of two points on the screen and distance between two points in the image. Higher level of control is attained by the ability to bound a point within a region of the image or to bound the size of an object. Constrained optimization, based on the desired changes, permits the computation of virtual time derivatives of the camera parameters. Their system allows the user to change some camera parameters while maintaining the constraints. The method is limited to a static camera. An advantage of this technique is the ability to explore the solution space since it is connex.

Goal specific camera modules based on the concept of *shot* in cinematography have been devised by [13]. Constraints are then specified via these modules: follow a character, pan to a character. Camera motion under constraint seems to be limited to simple movements. concerning complex camera motion, a path-planning process uses point to point specification (which is not a screen defined constraint), the obtained camera location is then given to the optimization process. Consequently, the solving process is not global. Several drawbacks can be exposed, the system only provides a single solution related to a user given optimization function. There is no solution space exploration. Camera movements (location and view direction) are not handled as a whole concept. The system calculates a static camera solution for each frame. Thus, the optimization process must be performed at each frame (interpolation between computed keyframes is of no use because of the risk of constraint violation). A last drawback is concerned with the need for a user given initialization. However, this work presents a rather complete set of screen space constraints.

[9] developped a paradigm for automatically generating camera specifications in real-time. This work focuses more on the *cutting* part of filmmaking than on the shooting part.

2.3 Goal

Our system is concerned with a virtual cameraman. The system presented in this paper computes camera movements corresponding to user-specified desired properties in the image space. We believe in a process that provides the user with multiple solutions if any. Consequently, the use of a single solution optimization process is not suitable. Moreover, our aim is to help the artist to generate complex camera movements that could not be obtained, at reasonable cost, by a generate-and-test stage.

We suppose that the artist has already modelled the scene (object shape and movement). The objects behavior is therefore known completely by the system.

The solving process presented is global and computes camera movements which satisfy multiple constraints. In addition, the calculation is not performed frame to frame, nor for some keyframes, but for the complete animation.

The use of `Declic` permits an easy extension of properties definition together with reduced computational time.

3 Mathematical models

The used mathematical models for Camera representation and scene objects for both static and dynamic approaches are detailed in the following.

3.1 Fixed camera model

The constraint solving process must manage any camera model even if we present the most commonly used (see figure 2) :

- eye position: three scalars (x, y, z);
- view direction: three scalars (panning: θ, tilting: ϕ, rolling: ψ);
- focal distance: one scalar $(focal)$;
- image aspect ratio: one scalar $(ratio)$.

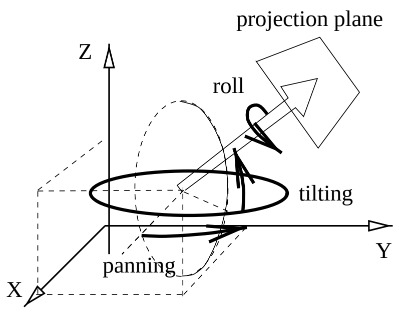

Figure 2: Camera Model

3.2 Fixed Object model

Let us recall that the scene is considered as a data for the problem. We suppose that the user has already set the position, orientation and movement of each object.

First, a 3D-point could be attached to any moving or fixed object or just defined by its global coordinates. Second, a bounding box (in the local object coordinate system) is attached to each scene object. In addition, bounding boxes can be associated to any set of objects.

The user may want to see the right part of an object or the top. Thus, to be able to specify object orientation on the image, three vectors are defined for each object (see figure 3) : $\overrightarrow{Front}$, $\overrightarrow{Up}$ and $\overrightarrow{Right}$. $\overrightarrow{Up}$ is the natural object axis if the object posseses one.

3.3 Animated objects and camera

3.3.1 Elementary shots and Desired camera movements

Most films are made of a large number of short elementary shots. This remark leads us to reject, for the moment, very complicated camera movements like the movements a cameraman could perform when carrying the camera on his shoulder. Our goal is to help an artist to obtain easily simple camera movements with a precise control on object position within the image space. An editor could then assemble the calculated shots (this problem is not currently addressed).

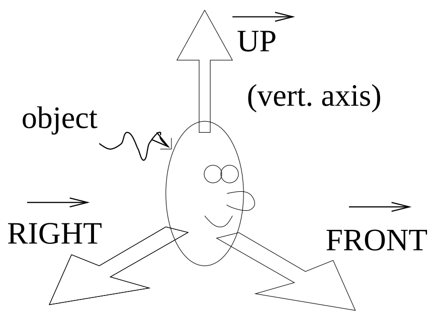
$\overrightarrow{\text{UP}}$
(vert. axis)
object
$\overrightarrow{\text{RIGHT}}$
$\overrightarrow{\text{FRONT}}$

Figure 3: Object axis

In this article, we shall not enumerate all the camera movements that one animator could need. The reader can refer with profit to the most comprehensive book by Arijon[1]. However, for clarity sake, let us recall the main camera movements (see figure 4):

- panoramic shot; Usually the camera location is constant:
  - horizontal panoramic shot: the camera rotates around a vertical axis
  - vertical panoramic shot: the camera rotates around an horizontal axis
- tracking shot or travelling; General term for the case of a change of the camera location :
  - dolly shot: the camera is placed on a dolly and moved from left to right (or reciprocally) or from above to front (or reciprocally);
  - tracking: for movements on the side of objects. Usually dolly term is not used for such movements.
  - arcing: the camera moves along a circle and looks towards the circle center;

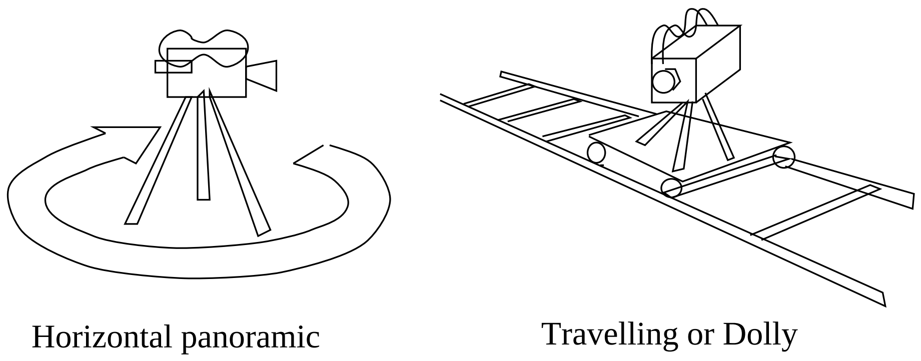

Figure 4: Camera movements

3.3.2 Modelling the parameter changes

We use parametrized movements. In order to obtain the set of movements, the solver must compute the parameters that satisfy the desired constraints.

An easy and powerful formulation is the polynomial formulation. In our tool, every parameter of any object of the scene and each parameter of the camera is allowed to change with respect to time in a linear, parabolic or cubic manner with respectively 2,3 and 4 unknowns.

Let us take the panning parameter θ, if θ is set to parabolic ($\theta(t) = a_\theta t^2 + b_\theta t + c_\theta$), the unknowns would become $a_\theta, b_\theta, c_\theta$. The solver will search their domains to obtain the solution triplets which satisfy the constraints.

The domains of the new parameters have to be chosen with care to allow maximum range of movements.

3.3.3 Object movements

For the sake of simplicity, we have restricted, for instance, the scene objects to perform *simple* movements. These movements are specified by the evolution formula of each parameter of the location and orientation of each object. The only restrictions on the use of complicated formulas are the available interval functions of the solver.

For the moment, there is no visibility testing (just a Z component comparison) and no collision detection.

4 Using constraints to specify the image

This section deals with the various means that can be used to define a desired image.

4.1 Frame

Let us define a *frame* (similar to [14]) as a rectangle whose borders are parallel to the screen borders (see figure 5). This rectangle can be inside, outside or partially inside the screen. Usually, it is fully inside the screen.

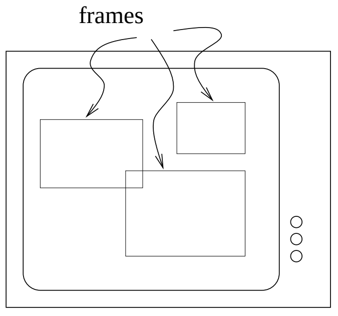

Figure 5: Frame concept

A *frame* helps to restrain the projection zone of an object (or group of objects).

4.2 Constraints on the camera

Constraints can be separated into three categories. The first category contains camera description functions. Objects are not used in these constraints. The second category consists of 2D-projection of the object constrained over frames. The last family is concerned with constraints on objects and constraints between objects. The following paragraphs deal with these three categories.

Some simple constraints on the camera are listed below.

- *fixed location panoramic shot*; The camera location is constant, but the direction view is allowed to change;
- *pure travelling*; View direction remains constant during the movement while the location of the camera is changing.
- *high-angle* or *low-angle*: Quantitative constraints. This allows us to specify whether the camera view direction looks from above or beneath the horizon.

4.3 Constraints on the projected objects

The next constraints are functions of both an object and a frame. They are used to place the object in the image space. The user (interactively or off-line) defines frames, then chooses properties and selects a frame together with an object. The following are some examples of the constraints we have defined.

- *fully included in*; The object bounding box must appear within the frame. This is the main property.
- *partially included in*; The object bounding box is allowed to appear partially out of the frame.
- *fully excluded of*; Could be used for example to specify that a given object must not appear in the image.

4.4 Constraints on objects

- *orientation of the object axis*; This orientation is specified with respect to the screen border; The specified object must be naturally axial. The constraint is based on the $\overrightarrow{Up}$ vector, defined in section 3.
- *upside down*; *stand up*: is used to specify that the given object appears upside down (respectively standing up); The object must be naturally orientated;
- *front side visible*: is used to force the object to face the camera.
- *closer than*: Function of two objects; This constraint compares the Z componants of the transformed objects bounding box.

4.5 Duration of a constraint

Another important property of our work is the ability to specify the duration of any constraint. In the virtual cameraman, any constraint on the camera or on any projected object can last for the whole animation or only for a given time interval. This allows us to, for example, specify a starting image and a final image and to leave the camera movement totally free between these two images. By default, a constraint is specified for the whole time interval.

5 The `Declic` language

`Declic` (Declarative Language with Interval constraints) is a new CLP(intervals) language [4]. Designed with clp(FD) [11] as a starting point, it implements the cooperation of two main solvers: one enforcing hull-consistency over some primitive constraints [10, 6], and the other enforcing box-consistency over global complex constraints [5, 24]. `Declic` is able to handle problems merging integer, boolean and real constraints, and supports closed as well as opened domains for the variables.

5.1 `Declic` in a nutshell

`Declic` is based on the constraint logic programming framework [12, 17, 18] and more precisely on clp(FD), developped by D. Diaz and P. Codognet [11]. It inherits from this last language all the Prolog engine along with both the ability to interpret as well as to compile programs into bytecodes executables [4].

In `Declic`, a variable need not to be initialized, all the typing is done dynamically with constraints. Initialisation of variable may be done with a `X in R` construction, where `R` denotes a range. Opened and closed domains are accepted. By default, the domain of a variable ranges over the real numbers.

There are three classes of relations, whose symbols, $\{=, \backslash =, <, >, <=, >=\}$, are prefixed by #, \$ or \$\$. These classes of relations allow the user to specify the corresponding solver. Basically, `Declic` introduces a cooperation framework between primitive-based interval-consistency and global interval Newton-based techniques. These techniques are essentially interval extensions of well known local consistency techniques such as arc and path consistency [20, 21], widely studied in artificial intelligence.

Besides the primitive constraints, `Declic` implements some traditional high-level constraints. Moreover, the user has still the ability to write its own constraints directly in the engine by interfacing C code with Prolog predicates.

5.2 Universally quantified constraints

In a mobile camera context, it is important to specify that an object (or group of objects) remain in a given screen frame during the whole animation or a user-defined interval of time. Since time is, in our model, represented as a constrained variable, this requirement implies the introduction (and processing) in a traditionally existential and conjunctive constraint world, of universally quantified constraints.

This process is achieved by the transformation of the standard interval propagation algorithm[10, 6] as follows: first the solution space is approximated thanks to Newton-based propagation techniques and the range of the universally quantified variables is checked. If the domains of the considered variables after approximation are strictly included in the desired Cartesian product, constraints are then negated in turn and interior partial approximations are computed. The set of fixed points, computed by successive interval bisections finally gives an inner approximation of the intended solution set. Further local (box) consistency checks are applied at each step to prune the remaining search space. This last algorithm is not currently implemented in `Declic`.

5.3 Constraint for camera movements in `Declic`

In this section, we give the relations, written in Declic language, corresponding to some of the properties listed above.

5.3.1 Basic relations

The main relation is concerned with the perspective transformation. This transformation uses 3D translation, 3D rotation and projection. Therefore, We first provide the system with 3D translation/9, rotations around x, y, z.

```
translation(X1,Y1,Z1,X2,Y2,Z2,X3,Y3,Z3):-
X1+X2 #= X3,
Y1+Y2 #= Y3,
Z1+Z2 #= Z3.
x_rotation(X1,Y1,Z1,A,X1,Y3,Z3):-
 CA $= cos(a), SA $= sin(a),
 Y3 $$= (CA * Y1) - (SA * Z1),
 Z3  $$= (SA * Y1) + (CA * Z1).
```

Y rotation and Z rotation are written in the same manner.

The projection along Y axis relation is then written.

```
Yprojection(X1,Y1,Z1,AX,AY,X2,Y2):-
 TX $= tan(AX), TY $= tan(AY),
 X2 $$= (X1*AX)/Y1,
 Y2 $$=(Z1*AY)/Y1.
```

And last the general perspectve relation:

```
perspective(XC,YC,ZC,TH,PH,PS,
            AX,AY,X1,Y1,Z1,X2,Y2):-
translation(X1,Y1,Z1,XC,YC,ZC,
                              X3,Y3,Z3),
Z_rotation(X3,Y3,Z3,TH,X4,Y4,Z4),
X_rotation(X4,Y4,Z4,PH,X5,Y5,Z5),
Y_rotation(X5,Y5,Z5,PS,X6,Y6,Z6),
Yprojection(X6,Y6,Z6,AX,AY,X2,Y2).
```

5.3.2 Properties relations

To bound a 3D point within a Frame in the screen, the perspective relation is simply called.

```
framepoint(XC,YC,ZC,TH,PH,PS,
            AX,AY,X1,Y1,Z1,
   XMIN,XMAX,YMIN,YMAX):-
perspective(XC,YC,ZC,TH,PH,PS,
                  AX,AY,X1,Y1,Z1,X2,Y2),
X2 $> XMIN, X2 $< XMAX,
Y2 $> YMIN, Y2 $< YMAX.
```

This relation permits to obtain the camera position (`XC,YC,ZC`) and the rotation angles (`TH,PH,PS`) given the appertures on x and y axis (`AX,AY`) and the position (`X1,Y1,Z1`) of the point to bound within the frame (`XMIN,XMAX,YMIN,YMAX`).

5.3.3 Animated objects and camera

This description can be extended to handle animated objects and camera. This can be done by introducing a specific variable representing time and to model the movements in the same manner with respect to time in a linear, parabolic or cubic manner, as already mentioned in the section 3.3.2.

6 Future Work and Conclusion

Our goal was to conceive and develop a helping tool for the camera positionning in a synthetic image film creation context. Our first implementation of the virtual cameraman has shown the feasibility of such a tool.

Using this tool, an artist specifies the desired properties within the image space, without any knowledge of the camera model. The artist can, therefore, focus on creativity and forget internal mathematical models.

The presented tool provides a safe approximation of the whole set of solutions for the camera movement problem instead of a single movement strongly depending on an optimization process. Different kinds of camera movement modelisation have been proposed and interval computation has been shown to be a good solution to avoid keyframing and interpolation.

In the second phase of this project, as developped in this paper, we are interested in (a) improving the performance of the resolution process in order to extend the number of camera degrees of freedom that can be handled by the system, (b) to program the system in a high level, constraint-oriented language for flexibility and ease of programming/maintenance, (c) to study and devise an appropriate and efficient way of dealing with constraints that must hold in a given interval (e.g. of time) and to implement this process as a built-in in the Declic language. Finally, we intend to demonstrate the usability of such a tool for artists and more generally graphists without involving them in camera movements mathematical descriptions. The final implementation will be clearly oriented towards creativity, minimizing the amount of repetitive tasks and the need for image representation models knowledge.

References

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

[2] Barr, A. H., and Currin, B. et al. Smooth interpolation of orientations with angular velocity constraints using quaternions. *Computer Graphics*, 2(26), 1992.

[3] B. Barsky. Local control of bias and tension in beta-splines. *Computer Graphics*, 3(17), 1983.

[4] F. Benhamou, F. Goualard, and L. Granvilliers. Programming with the Declic Language. In *Proceedings of the International Workshop on Interval Constraints*, pages 1–13, Port Jefferson, USA, 1997.

[5] F. Benhamou, D. McAllester, and P. Van Hentenryck. CLP(Intervals) Revisited. In *Proceedings of ILPS'94*, Ithaca, NY, USA, 1994.

[6] F. Benhamou and W. J. Older. Applying Interval Arithmetic to Real, Integer and Boolean Constraints. *Journal of Logic Programming*, 32(1):1–24, 1997.

[7] J. Blinn. Where am i? what am i looking at? *IEEE Computer Graphics and Applications*, (13), 1988.

[8] Brotman, L. S. and A. Netravali . Motion interpolation by optimal control. *Computer Graphics*, 4(22), 1988.

[9] Christianson, Anderson, He, Salesin, Weld, and Cohen. Declarative camera control for automatic cinematography. In *proceedings of AAAI'96*, 1996.

[10] J. G. Cleary. Logical Arithmetic. *Future Computing Systems*, 2(2):125–149, 1987.

[11] P. Codognet and D. Diaz. Compiling Constraints in `clp(fd)` . *Journal of Logic Programming*, 27(3):185–226, 1996.

[12] A. Colmerauer. An Introduction to Prolog III. *Communications of the ACM*, 33(7):69–90, July 1990.

[13] S. M. Drucker and D. Zeltzer. Intelligent camera control for virtual environments. In *Proceedings of Graphics Interface '94*, 1994.

[14] S.M. Drucker. *Intelligent Camera Control for Graphical Environments.* PhD thesis, MIT Media Lab, 1994.

[15] Drucker S. and T. Galyean et al. Cinema: A system for procedural camera movements. In ACM Press, editor, *Proceedings of 1992 ACM Workshop on Interactive 3D Graphics*, 1992.

[16] Gleicher, M. and Witkin, A. Through-the-lens camera control. *ACM Computer Graphics*, 2(26), 1992.

[17] J. Jaffar and J.-L. Lassez. Constraint Logic Programming. In *Proceedings of POPL'87*, pages 111–119, Munich, 1987.

[18] J. Jaffar and M. Maher. Constraint Logic Programming: A Survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

[19] F. Jardillier and E. Languenou. Screen-space constraints for camera movements : The virtual cameraman. In *to appear in Proc. 1998 Eurographics Conference*, September 1998.

[20] A. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[21] U. Montanari. Networks of Constraints: Fundamental Properties and Applications to Picture Processing. *Information Science*, 7(2):95–132, 1974.

[22] K. Shoemake. Animation rotation with quaternion curves. *Computer Graphics, Proceedings of SIGGRAPH 85*, 3(19), 1985.

[23] Turner, Russell, F.Balaguer, E.Gobbetti, and D. Thalmann. Physically-based interactive camera motion control using 3d input devices. *Computer Graphics International*, 1991.

[24] P. Van Hentenryck, L. Michel, and F. Benhamou. `Newton` - Constraint Programming over non-linear Constraints. *Science of Programming*, 1997. (Forthcoming).

[25] Ware, C. and Osborn, S. Exploration and virtual camera control in virtual three dimensional environments. In ACM Press, editor, *Proc. 1990 Symposium on Interactive 3D Graphics*, 1990.