Sketching 3D Animations

Jean-Francis Balaguer and Enrico Gobbetti

CRS4
Center for Advanced Studies, Research, and Development in Sardinia
Scientific Visualization Group
Via Nazario Sauro 10
09123 Cagliari
Italy

E-mail: `{balaguer|gobbetti}@crs4.it`

Abstract

We are interested in providing animators with a general-purpose tool allowing them to create animations using straight-ahead actions as well as pose-to-pose techniques. Our approach seeks to bring the expressiveness of real-time motion capture systems into a general-purpose multi-track system running on a graphics workstation. We emphasize the use of high-bandwidth interaction with 3D objects together with specific data reduction techniques for the automatic construction of editable representations of interactively sketched continuous parameter evolution. In this paper, we concentrate on providing a solution to the problem of applying data reduction techniques in an animation context. The requirements that must be fulfilled by the data reduction algorithm are analyzed. From the Lyche and Mørken knot removal strategy, we derive an incremental algorithm that computes a B-spline approximation to the original curve by considering only a small piece of the total curve at any time. This algorithm allows the processing of the user's captured motion in parallel with its specification, and guarantees constant latency time and memory needs for input motions composed of any number of samples. After showing the results obtained by applying our incremental algorithm to 3D animation paths, we describe an integrated environment to visually construct 3D animations, where all interaction is done directly in three dimensions. By recording the effects of user's manipulations and taking into account the temporal aspect of the interaction, straight-ahead animations can be defined. Our algorithm is automatically applied to continuous parameter evolution in order to obtain editable representations. The paper concludes with a presentation of future work.

Keywords:

Data Reduction, 3D Animation, 3D Interaction, Performance-Driven Animation

1. Introduction

Keyframing is by far the most prevalent motion control technique offered by current animation systems [14]. The animation of continuous parameters is defined by interactively specifying a sequence of key values and their associated time. In-between values are automatically generated by using an interpolation method in order to obtain a smooth animation. Keyframe computer animation is strongly related to the pose-to-pose style of traditional animations, with timing and pose control of extremes and in-betweens. However, with the computerized tool, the animator has much less control on the in-betweens, since the shape and timing of the resulting parameter curve are largely determined by the underlying interpolation scheme. The desired final animation is obtained at the expense of a very time-consuming process, where the parameter curves are iteratively edited until the real-time playback of the animation is judged satisfactory. This inability to specify the timing of an animation in an interactive way is a major drawback in all cases where the spontaneity of the animated object's behavior is important [42]. In these cases, defining the animation by straight-ahead actions is clearly a better solution [21]

The problem of defining animations using straight-ahead actions is addressed by performance animation systems, where an actor's performance is captured in real-time by means of dedicated devices and is used to drive the animation of a synthetic character [42; 41; 39]. The sensed values are put in direct correspondence with parameters of the animated objects and the actor constantly evaluates his performance based on real-time visual feedback. While the entire process is kept very creative and spontaneous, the incredibly large amount of data associated with a live performance makes it impossible to efficiently edit the captured animation. This is the major problem faced by current performance animation systems [42]. In order to edit real-time captured performances, we need to obtain a more compact representation of the captured data. In particular, building a spline representation would have the advantage of providing a mean to seamlessly integrate real-time capture capabilities with keyframing systems.

We are interested in providing animators with a general-purpose tool allowing them to create animations using straight-ahead actions as well as pose-to-pose techniques. Our approach seeks to bring the expressiveness of real-time motion capture systems into a general purpose multi-track system running on a graphics workstation. Essential characteristics of such a system should be:

- high-bandwidth interaction with 3D objects for the continuous control of all kind of attributes. This can be achieved by means of devices allowing the control of multiple degrees of freedom, as in virtual environments [9; 32; 46] and performance animation systems [42; 41], or with more traditional device configurations together with interaction metaphors exploiting the greater possibilities of 3D [19; 12; 15; 27; 37; 17];
- automatic construction of editable representations of interactively sketched animations of continuous parameters. This can be obtained by applying data reduction techniques to the recorded data in order to compute a spline that reproduces precisely the geometry and timing of the sketched animation and that can be effectively edited with standard spline editing tools.

In this paper, we concentrate on the problem of applying data reduction techniques in an animation context. First, we analyze the requirements that must be fulfilled by the data reduction algorithm. Then, we derive an incremental algorithm that guarantees constant latency time and memory needs. After presenting examples of animation paths processed with our algorithm, we describe a visual construction environment allowing the definition of 3D animations using pose-to-pose techniques and straight-ahead actions.

2. Data Reduction for Animation

In a 3D animation system, the variety of data that must be processed and the type of information that must be preserved, as well as the various sources that might be used to generate the initial data, introduce specific requirements that must be fulfilled by the data reduction algorithm. In particular, the following points must be considered.

- **Preservation of geometry and timing**: The computed curve must not only precisely approximate the geometry, as it is required in modeling and drafting tools, but also the timing of the initial data. In the case of interactively specified data, both components are defined with a limited precision and should be efficiently smoothed by the approximating curve while preserving the correspondence between the geometry and timing. This implies that the data reduction algorithm must treat both components simultaneously, since treating them independently may introduce errors difficult to control.
- **Generality**: The algorithm may be used to compute approximations of data generated by various sources. In particular, we shall consider the cases of interactively specified data presenting a non negligible amount of noise and imprecision, as well as highly accurate data generated by evaluating a mathematical model, as when using data reduction to convert the output of a simulation engine to a keyframing representation. Additionally, the algorithm must be able to treat data of various dimensions.
- **Reduction factor**: When applying the data reduction algorithm, we are interested in obtaining an approximation curve being optimal for edition. Therefore, we are more interested in obtaining a uniform distribution rather than a minimum number of curve controls.
- **Compatibility**: The performance based approach that we promote in this paper is intended to complement rather than replace more traditional keyframe techniques. For the sake of integration, the approximating curves computed by the data reduction algorithm should be compatible with those used by a track system, so that the same tools and techniques can be used to perform editing tasks.
- **Usability**: The approximation curve must be made available in times compatible with the hard constraint of responsiveness of an interactive system. In practice, application response is perceived as immediate if the result is made available in less than one second after the end of the user action [30]. Additionally, there should not be an upper limit to the number of samples than can be treated.

If data reduction or curve fitting techniques have been successfully applied in drafting and modeling tools for the interactive specification of curves or surfaces [7; 6; 33; 34; 36; 38; 40; 44] , their use to process animation data appears to have been left mostly unexplored. In [2], a digitizing tablet was used to sketch the evolution over time of mono-dimensional parameters, but no data reduction or curve fitting technique was used to process the sampled data. In [40], a digitizing tablet is used to sketch the projection of the path geometry onto the scene's ground. A curve is then fitted to the 2D path but the timing of the specification is not taken into account, thus missing the advantages of performance approaches. Current performance animation systems use standard keyframe interfaces to manipulate the sampled data directly, clearly an inadequate approach [42; 41].

There is a wealth of literature on curve fitting or data reduction algorithms [10; 36; 7; 23; 34; 13; 44]. Methods for fitting curves to data points have traditionally attempted to reduce the error in one of the L^p norms, using more and more polynomial segments until the resulting fit does not exceed some given error bound [10; 36; 34; 13]. These schemes are generally based on 2D or 3D geometric considerations and lack the generality we require. Alternatively, Lyche and Mørken have developed an algorithm (the LM algorithm) that begins with a piecewise linear B-spline with open end conditions passing through the data points to be approximated [23; 24; 25]. This curve is re-represented through degree elevation as a curve of degree k-1 with knots of multiplicity k-1 at each data point, k being the user-specified order for the approximation curve. This curve is used as the initial curve and knots are removed, according to some global requirements, until the user-specified error tolerance is met.

The LM algorithm is a very general scheme that can approximate data of various dimensions with B-spline curves of arbitrary degree. When selecting the knots to be removed, some provision is made to efficiently distribute the removed knots across the entire curve. The approximation curve defined over the reduced knot sequence is computed by solving a least square minimization problem, but the approximation is accepted upon the measurement of the error with respect to the initial curve using the L^∞ norm. That way, the reduced curve respects the value and parameterization of each data point within the user-specified tolerance.

All these characteristics make the LM algorithm very well suited for our particular data reduction problem. However, each step of the search for the shortest knot sequence involves solving a global minimization problem that requires $O\left(N^3\right)$ operations [35]. This makes it difficult to directly apply the LM algorithm in the context of an interactive application and an incremental version of the algorithm must be considered. Banks and Cohen [7; 6] developed an incremental version of the LM algorithm that could be used in a performance animation context, but their method is unable to ensure constant latency times and memory needs.

We have developed an algorithm that incrementally builds, from the input sequence, a parametric B-spline preserving value and time of each input sample within a given tolerance. It is an incremental version of the LM algorithm that works in parallel with the interactive specification by considering, as in [7], only a small portion of the input curve at any time. Latency time and memory requirements for handling each portion of the curve are constant and easily determined. Data reduction may therefore be performed concurrently with interactive parameter input, and the responsiveness of the application can be ensured when handling animations defined by any number of samples. In the following sections, we rapidly describe the LM algorithm and we discuss our incremental version.

2.1. The Lyche and Mørken Algorithm

The algorithm is composed of three distinct steps termed **rank**, **remove** and **approximate**:

- **rank** consists in evaluating the importance of each interior knot using the approximate distance between the curve and a curve defined on the knot sequence without the ranked knot.
- **remove** exploits the ranking information to suppress the knots in the order that will perturb the curve the least amount. The result is usually a much shorter knot vector for which an approximating curve of the initial data can be computed without exceeding the maximum error tolerance.
- **approximate** determines the new coefficients of the approximating curve by computing an approximation of the initial curve over the new knot vector.

Let *f* be the piecewise degree *k*-1 B-spline interpolating the data points, and ***t*** be the initial knot vector. Then the algorithm can be briefly described as follows:

```
$\tau^0 = \boldsymbol{t}$
$g^0 = f$
for $i = 0,1,2,\ldots$
    if $\left|\tau^i\right| = 2k$ then stop; No more interior knots to remove.
    for $j = 0,1,2,\ldots,\left|\tau^i\right|$
        $w_j^i = \mathbf{rank}\left(\tau_j^i, g^i\right)$
    end
    $\tau^{i+1} = \mathbf{remove}\left(\boldsymbol{w}^i, \tau^i\right)$ such as $\left\| f - \mathbf{approximate}\left(f, \tau^{i+1}\right)\right\| \le \varepsilon$
    if $\left|\tau^{i+1}\right| = \left|\tau^i\right|$ then stop; No knots could be removed
    $g^{i+1} = \mathbf{approximate}\left(f, \tau^{i+1}\right)$
end
```

It is an iterative algorithm that, starting from the initial curve ($g^0 = f$), produces, at the ith iteration, an approximation curve g^i defined over the reduced knot vector τ^i. At the next iteration, the ranking phase determines the importance of each interior knot of τ^i with respect to the previous approximation g^i and not to the initial curve *f*. This allows the importance of each knot to evolve as neighboring knots are removed. However, when computing the approximation of *f* over the new knot vector τ^i, the error is measured by comparing the computed approximation to the initial curve *f*, and not to the previous approximation g^i. The algorithm stops either when every interior knot has been removed, or when no interior knot could be removed. In practice, an upper limit is also set for the number of iterations as a large number of knots are removed during the first iterations. The approximation curve is typically computed by solving a weighted least-squares minimization problem.

2.2. Incremental Algorithm

The basic idea of the incremental knot removal algorithm is to incrementally compute an approximation to the original curve by considering only a small piece, called the window, of the total curve at any time [7]. That way, as data arrives from the input device, the same sequence of control points and knot values are appended to the initial curve *f* and to the accumulated approximation curve *g*. When enough data points have been sampled, the knots within the window are ranked and considered for removal. The reduced segment is then spliced back into the accumulated curve with C^{k-2} continuity. As sketching begins, the portion of the initial and the approximation curves are the same but, after several window reductions, the approximation contains fewer knots and control points than the initial curve. At the end of the algorithm, the starting index of the window is modified according to the number of knots that could be removed in the current window. If all the knots were removed then the index is left unchanged. Alternatively, if no knots could be removed the starting index is moved to the end of the current window. Therefore, each knot is considered only once for removal.

In order to isolate the section considered for reduction from the rest of the curve, as well as to give context to the data reduction process, the window is extended with two overlapping regions at the beginning and the end of the curve section. This is done by extending the parameter interval with *k* knots at each extremity. The knots within the overlapping region are not candidates for removal. The first k knots come from the accumulated approximation, and therefore have been already reduced. The last *k* knots come from the initial curve and will be processed with the following window.

2.2.1 Ranking and Removing the Knots

The incremental algorithm computes the weight of each interior knot of the window as done in the original LM algorithm. In effect, it can be shown [7] that this process uses only local information and therefore the ranking phase can be easily performed for small sections of the curve.

The LM algorithm makes the hypothesis that the number of knots to remove will be relatively large and that many of the knots will be clustered into groups with small differences in their weight values. Therefore, a scheme is introduced to select the knots in a group such that the removed knots are uniformly distributed across the entire parameter range. The incremental algorithm considers small windows of knots at a time, and therefore effectively distributes the removed knots across the entire curve. Moreover, since the window size is usually small with respect to the total number of knots, there is little benefit in grouping the knots, which can thus be removed strictly in order of increasing weights.

To find the shortest knot sequence over which an approximation within the given tolerance can be built, the LM algorithm uses a binary search through the possible knot vectors based on the assumption that the error increases with the number of knots to remove. If this hypothesis may seem reasonable for large knot sequences, it is more questionable for short sequences and we would rather test all possibilities until the given tolerance is met. However, if m is the size of the window, then the solution will be found on average in $m/2$ tries against $\log_2 m$ for the binary search, and we will proceed as the LM algorithm for large windows.

2.2.2 Computing the Approximation

Let f be the initial order k-1 B-spline curve defined by the knot vector $\boldsymbol{t}$ and the control polygon $\boldsymbol{c}$, and let g be the approximation curve defined by the knot vector τ and the control polygon $\boldsymbol{d}$. Let m be the size of the window and p the starting index of the current window. Let g^* be the section of g defined by the knot sequence $\tau^* = [\tau_p, \tau_{p+2k+m-1}]$ and the control polygon $\{d_p, d_{p+k+m-1}\}$. Similarly, let f^* be the portion of f defined over $\boldsymbol{t} = [t_1^*, t_2^*]$ where $t_1^* = \tau_p$ and $t_2^* = \tau_{p+2k+m-1}$

Bank's Approach

In the incremental algorithm proposed by Banks, the approximation h^* of the curve section f^* is computed as in the LM algorithm. However, because of the local nature of the algorithm, we now consider only a section f^* defined over $\boldsymbol{t}^*$ of the initial curve f, and a section g^* defined over τ^* of the accumulated approximation g. The knot vectors $\boldsymbol{t}^*$ and τ^* have general end conditions and the remove step of the algorithm can be expressed as:

$$\kappa^* = \mathbf{remove}(w^*, \tau^*) \text{ such as } \left\| f^* - \mathbf{approximate}(f^*, \tau^*) \right\| \le \varepsilon$$

There are several disadvantages to comparing against f^* when computing the section of the approximation curve h^*. First, the more knots removed, the more the time to treat the following windows increases because the knot vector $\boldsymbol{t}^*$ corresponding parametrically to τ^* will be composed of ever more knots. Therefore, when computing the approximation, the linear system is composed of more and more equations, computation time increases and memory requirements grow. Additionally, the first k knots of τ^* being knots treated by the previous window, the k first knots of $\boldsymbol{t}^*$ and τ^* are not necessarily the same and some complications may arise when computing the knot insertion matrix with the Oslo algorithm [11].

Our Approach

Since $\boldsymbol{t}^*$ and τ^* differ only by their first k knots, we may consider computing the approximation with respect to g^* instead of f^*. The remove phase of the algorithm becomes:

$$\kappa^* = \mathbf{remove}(w^*, \tau^*) \text{ such as } \left\| g^* - \mathbf{approximate}(g^*, \tau^*) \right\| \le \varepsilon .$$

This approach has several advantages. First, whatever the window being considered, the number of knots in τ^* is fixed, and therefore the number of equations is known a priori. Similarly, the computation time required to process one section of the curve has an upper bound, corresponding to the case where only one knot can be removed. Finally, the memory requirements are totally predetermined. Data reduction may therefore be performed concurrently with the sampling of the initial data, and the responsiveness of the application can be ensured when handling animations defined by any number of samples.

One important aspect of the LM algorithm is the warranty that the error between the initial and the approximation spline lies within the tolerance for every data point. With an incremental algorithm, controlling the error over the entire curve is more difficult. As stated by Banks, extending the window with k knots of past context allows essentially isolating from the rest of the curve the error introduced by reducing the current window [7]. The risk exists however, that the modifications of the k control points corresponding to the overlapping knots introduce an additional error in the previous curve segment.

When computing the approximation with respect to g^*, supposing that k overlapping knots are enough to effectively isolate the error from the rest of the curve would result in allowing twice the tolerance for the points associated with the overlapping knots. In effect, g^* is defined by the knot vector $\tau^* = \left[\tau_1^*, \dots, \tau_{m+2k-1}^*\right]$, where $\tau_1^* = \tau_p$ and $\tau_{m+2k-1}^* = \tau_{p+m+2k-1}$, and by the control polygon $\boldsymbol{d}^* = \left\{d_1^*, \dots, d_{m+k}^*\right\} = \left\{d_p, \dots, d_{p+m+k-1}\right\}$. The first k control points $d_1^*, \dots d_k^*$ are members of the control polygon of the accumulated approximation curve, and are therefore defined with a tolerance ε with respect to the initial curve f. When computing the approximation, the LM algorithm considers that the tolerance is the same for all the data points. As the knots $\tau_p, \dots, \tau_{p+k-1}$ participate in two successive windows, this will allow a greater tolerance for the control points associated with the overlapping knots. The method used to compute the approximation has to be modified in order to integrate the fact that the first k points of the control polygon $\boldsymbol{d}^*$ and of g^* are known and must be preserved. We do so by extending the linear system with k equations expressing that the first k control points of the approximated section h^* must be the same as the first k control points of g^*. Additionally, we apply a smaller tolerance to the last k control points in order to prevent too important a modification of the segment's end tangent.

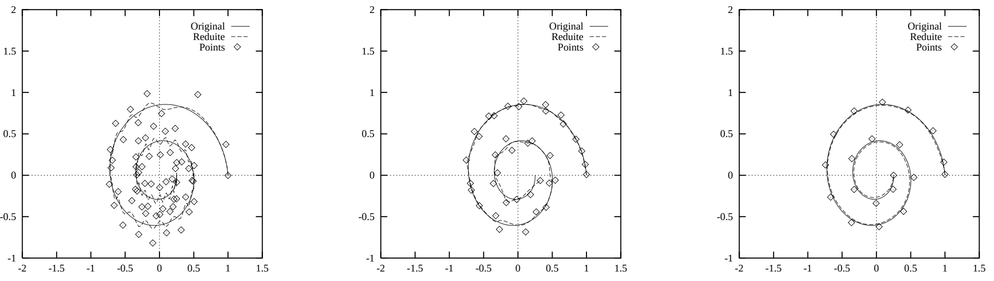


Figure 1. *Plots of the data points and of the approximation splines for the spiral of the example 3.1 of [24] for n = 1001 points, a window size of 10 knots, and the tolerance $\varepsilon = (0.1, 0.1)$.*

Figures 1 and 2 show the effects of introducing constraints in the computation of the approximation when processing the spiral of the example 3.1 from [24]. We applied our incremental algorithm with a window size of 10 knots and a tolerance $\varepsilon = (0.1, 0.1)$. The left curve has been obtained using an algorithm computing the approximation with respect to g^* without any considerations for start and end overlapping regions. The plot exhibits numerous oscillations whose frequency increase with the curvature. For the curve in the middle, additional equations have been introduced to anchor the first k control points of each section. The curve still exhibits oscillations and the error plot shows that the specified tolerance is not met for some localized sections of the curve. For the curve on the right, additional constraints limit the modifications of the last k control points. The error is now kept within the tolerance, while the control points are well distributed over the entire curve.

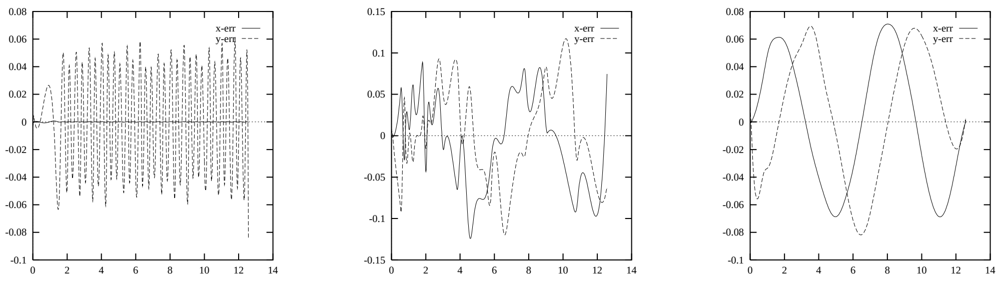


Figure 2. *Plots of the error components for the spiral of the example 3.1 of [24] for* $n = 1001$ *points, a window size of* 10 *knots, and the tolerance* $\varepsilon = (0.1, 0.1)$.

2.2.3 Incremental Algorithm

Let f be the initial curve defined over the knot vector $\boldsymbol{t}$ and the control polygon $\boldsymbol{c}$. Let g be the approximation curve defined over the knot vector τ and the control polygon $\boldsymbol{d}$. Let m be the window size and p the knot index of the current starting point of the window. Let g^* be the portion of g restricted to the interval $\tau^* = [\tau_1^*, \tau_2^*]$ with polygon $\{d_p, \ldots, d_{p+m+k-1}\}$, where $\tau_1^* = \tau_p$ and $\tau_2^* = \tau_{p+2k+m-1}$. Our incremental algorithm can be described as follows:

```
if $|\tau| - p \ge 2k + m$ then
        $\tau^0 = \tau^*$
        $g^0 = g^*$
        for $i = 0, 1, 2, \ldots$
                if $|\tau^i| = 2k$ then end; There are no more interior knots to remove
                for $j = k+1, \ldots, |\tau^i| - k$
                        $w_j^i = \textbf{rank}(\tau_j^i, g^i)$
                end
                $\tau^{i+1} = \textbf{remove}(w^i, g^i)$ such as $\|g^* - \textbf{approximate}(g^*, \tau^{i+1})\| \le \varepsilon$
                if $|\tau^{i+1}| = |\tau^i|$ then end; No interior knots could be removed
                $g^{i+1} = \textbf{approximate}(g^*, \tau^{i+1})$
        end
        if $|\tau^*| - |\tau^{i+1}| > 0$ then $g = \textbf{splice}(g^{i+1}, g)$
        $p = p + m - (|\tau^*| - |\tau^{i+1}|)$
else
        accumulate data points
end
```

Since we compare against the accumulated approximation curve g, we do not need to maintain the initial curve f and, as data arrives, the new knots and control points are added only to the current approximation g. The window is extended with k knots for past context and k knots for future context. Only the interior knots of

τ^* are candidate to removal. The window is processed iteratively and an approximation g^1 defined over τ^1 is built from the initial section of the curve $g^0 = g^*$. During the ith iteration, the ranking phase determines the importance of the knots of τ^i with respect to the current approximation g^i, while the error is checked with respect to the initial curve g^0. The iterative algorithm stops when all the candidate knots have been removed or when no knots could be removed. At the end of the algorithm, *p*, the starting index of the window, is modified according to the number of knots that has been removed, as in the incremental algorithm of Banks.

Our incremental algorithm differs from the algorithm of Banks in the method used to compute the approximation and by the iterative processing of the window. Since we compare to the approximation curve g^*, the processing time is much smaller than with the algorithm of Banks, as the minimization problem to solve is usually composed by a much smaller number of equations. Therefore, we can treat much bigger windows (typically 50 knots at a time) in times compatible with the responsiveness of the application.

2.2.4 Application of the Algorithm to Sketching Three-dimensional Paths

The LM algorithm takes as a starting point a linear B-spline that interpolates the data points. The algorithm is applied a first time with a tolerance ε_1 to this linear spline. The linear approximation curve is re-represented through degree elevation as a spline of the final desired degree. The algorithm is applied a second time to this spline with a tolerance $\varepsilon_2 = \varepsilon - \varepsilon_1$. The result is a spline of the desired degree approximating the data points within the tolerance ε.

The LM algorithm was designed to treat data known with the precision of some data collection device. However, we consider data that is essentially specified interactively and which is therefore far from being exact. In particular, the density and the accuracy of the samples may vary with the speed and accuracy of the user's specification. This aspect is particularly important when sampling animations, since the speed of the specification has here a meaning that must be preserved in the computed approximation. Additionally, when specifying three-dimensional animation paths, the user has to specify simultaneously the position, the orientation and their timing. The specification of such a large amount of information will clearly be the source of additional imprecisions.

We assume that the sampled data points are close enough together so that the B-spline built using them as control polygon is a curve representing the initial data within a small tolerance. The times at which each data point is sampled are used to build the initial knot vector. This curve is used as the starting point and the algorithm is applied only once.

Example: Jump Motion

In order to illustrate the method, we present the results obtained for the interactive definition of a jump motion in a computer generated movie [20]. Additional 2D and 3D examples can be found in [4]. The environment is a simplified version of the movie scene's database and is composed essentially by a square room of eight meters side. The user can control the position and orientation of the camera using a Spaceball and an eye-in-hand metaphor [43]. The beginning and the end of the motion sampling are specified by pressing and releasing a mouse button. The sampling frequency is set to 20 Hz, a low-pass filter is applied to the sampled data and then duplicate values are removed.

The initial data is represented by two cubic parametric B-splines, one for positions $\boldsymbol{f}_p$, and one for orientations $\boldsymbol{f}_r$, both curves being defined over the same knot vector $\boldsymbol{t}$. The control polygon of $\boldsymbol{f}_p$ is defined by the sequence of camera positions, while that of $\boldsymbol{f}_r$ is defined by the sequence of camera orientations represented using Euler angles. Therefore, $\boldsymbol{f}_p$ and $\boldsymbol{f}_r$ are defined by:

$$\boldsymbol{f}_p = \sum_{i=1}^{n} \boldsymbol{c}_i^p B_{i,4,\boldsymbol{t}} \text{ where } \begin{cases} \boldsymbol{t} = (t_1, t_1, t_1, t_1; t_3, \ldots, t_{n-2}; t_n, t_n, t_n, t_n) \\ \boldsymbol{c}_i^p = \begin{bmatrix} x_{t_i}^p & y_{t_i}^p & z_{t_i}^p \end{bmatrix} \text{ for } i = 1, \ldots, n \end{cases}$$

$$\boldsymbol{f}_r = \sum_{i=1}^{n} \boldsymbol{c}_i^r B_{i,4,\boldsymbol{t}} \text{ where } \begin{cases} \boldsymbol{t} = (t_1, t_1, t_1, t_1; t_3, \ldots, t_{n-2}; t_n, t_n, t_n, t_n) \\ \boldsymbol{c}_i^r = \begin{bmatrix} x_{t_i}^r & y_{t_i}^r & z_{t_i}^r \end{bmatrix} \text{ for } i = 1, \ldots, n \end{cases}$$

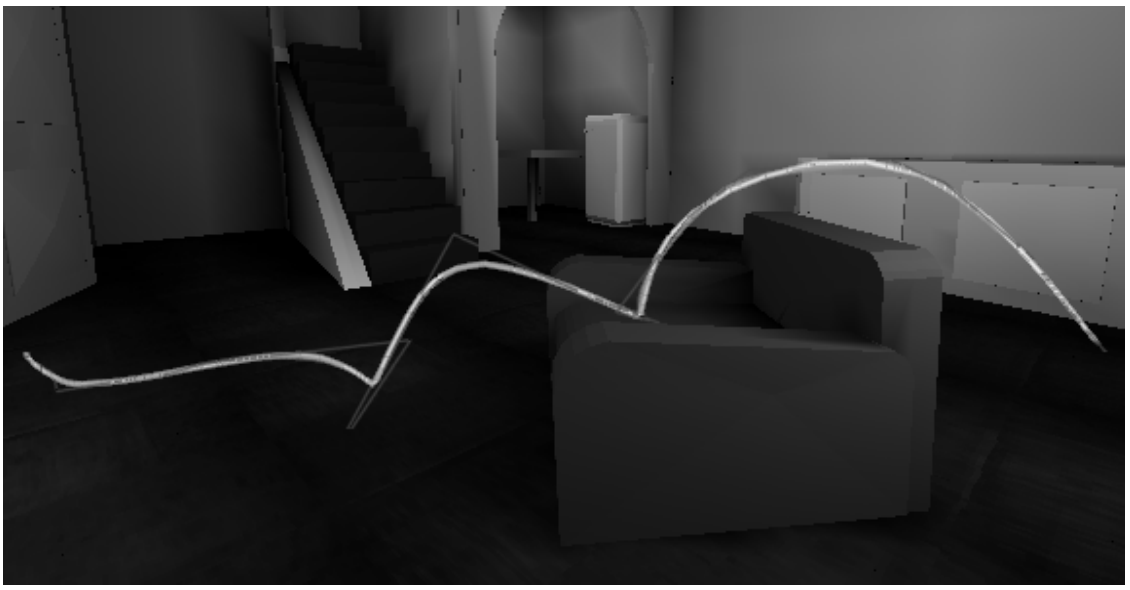


Figure 3. Sketched and initial jump motions.

The jump motion lasts 5.9 s and the initial plot is composed of 111 samples. The data reduction algorithm was applied with a window size of 20 knots, a position tolerance of 40 mm for each dimension, and an orientation tolerance of 2 degrees (0.035 radian). The position spline was reduced to 24 control points, while the orientation spline $\boldsymbol{f}_r$ was reduced to 8 control points. On a *R4000 SGI Indigo²*, the average processing time for each window of the position spline $\boldsymbol{f}_p$ was of 174 ms with a total processing time of 0.9 s, while the average time for rotation windows was of 72 ms with a total processing time of 0.36 s. In comparison, with the LM algorithm, the position spline was reduced to 16 control points in 2.1 s, and the orientation spline to 5 control points in 0.6 s.

Figure 3 shows the initial position spline $\boldsymbol{f}_p$ together with the reduced version $\boldsymbol{h}_p$. Figures 5 and 6 show the plots of initial and reduced position and orientation components. Figure 7 shows the plots of position and orientation error components, and figure 4 shows the plots of chord lengths as a function of time for the initial and reduced position splines.

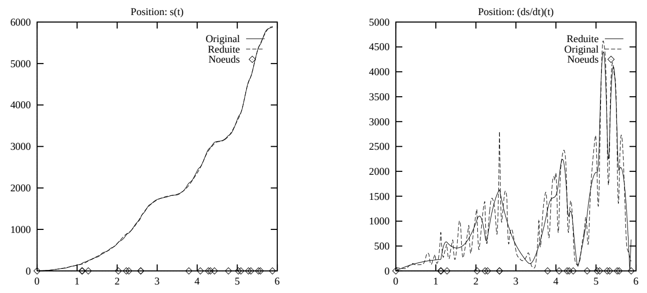


Figure 4. *Plot of chord length as a function of time and speed profiles for the initial and reduced position splines.*

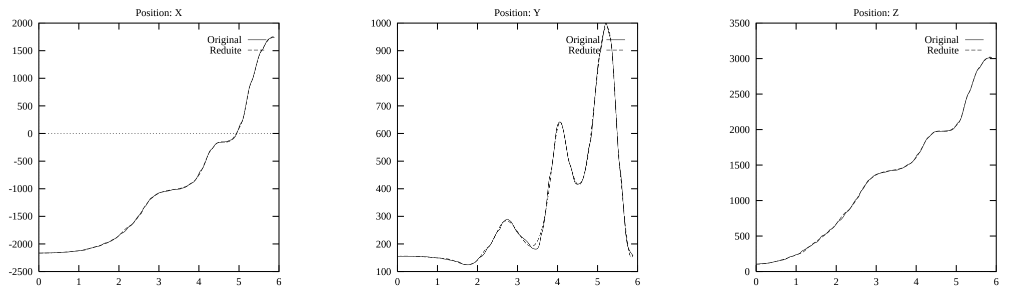


Figure 5. Plot of the components of the initial and reduced position splines.

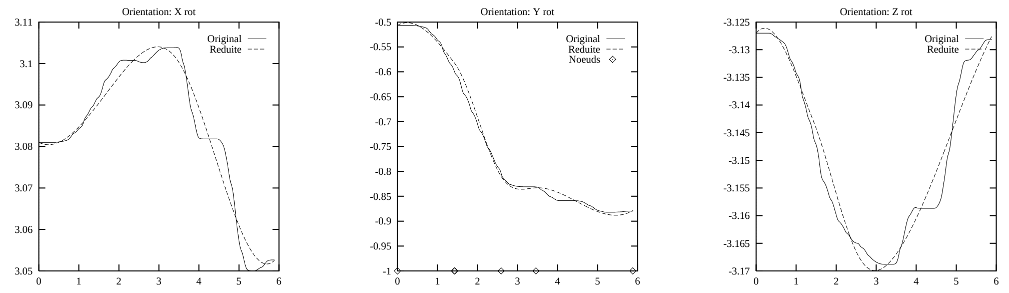


Figure 6. Plot of the components of the initial and reduced orientation splines.

From the plots of the components presented in figure 5 and 6, we can see that the reduced curves preserve the intention of the sampled motion while smoothing it efficiently. In particular, for the position curve, it is interesting to note that the discontinuities corresponding to each jump impulsion are well preserved, even though the algorithm does not make any provision for the processing of discontinuities and pauses. In figure 4, the plot of the chord length of the position curves shows that there is no error accumulation, since the reduced path is traveled very similarly to the initial data. The reduced knots are distributed over the entire parameter range. The speed profile has been correctly smoothed; better results are obtained with bigger windows. Finally, figure 7 shows that the error components are efficiently kept within the desired tolerances over the parameter range. In the video tape [3], the initial and reduced motions are shown in parallel. It can be noted that the geometry and timing of the initial motion has been preserved while efficiently smoothed.

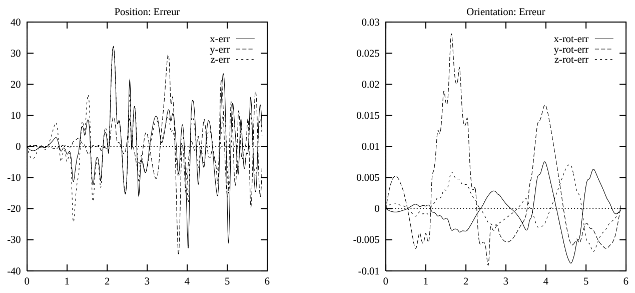


Figure 7. Plot of the error components $f^1(t_i)-h^1(t_i)$, $f^2(t_i)-h^2(t_i)$, and $f^3(t_i)-h^3(t_i)$.

In table 8, we present the results obtained by applying our incremental algorithm with different window sizes. About the reduction factor, we note that the bigger the window, the shorter the knot sequence, even though the results are comparable for all window sizes. The average processing time of each window clearly augments with the number of knots being considered, while the total processing times are similar and very inferior to the specification time (5.9 s). Processing the data in parallel with the specification makes it possible to keep the latency time, corresponding to the processing time of a single window, inferior to the responsiveness requirement of one second, and so for all tested window sizes, even if we process the windows for the position and orientation curves sequentially.

Window Size	Control points position	Control points rotation	Average window time position (ms)	Average window time rotation (ms)	Total time position (ms)	Total time rotation (ms)
5	39	11	37	18	750	440
10	40	10	64	30	640	300
20	24	8	174	72	870	360
30	25	8	258	132	1030	530
50	24	6	525	140	1050	280

Table 8. Results obtained by processing the jump motion using different window sizes.

Example: Long Motion

The advantage of our incremental algorithm over the global method is much more obvious when processing sequences composed of hundreds of samples. This time, we consider a very slow camera motion inside the same environment as in the previous example. The same methodology as before is used to define and sample the motion.

The motion lasts 29 s and the initial plot is composed of 482 samples. Table 10 shows the results obtained with our incremental algorithm for different window sizes. In comparison, with the global algorithm, the reduced position curve is defined by 6 points with a processing time of 13.8 s, while the reduced orientation curve is defined by 6 points with a processing time of 13.7 s.

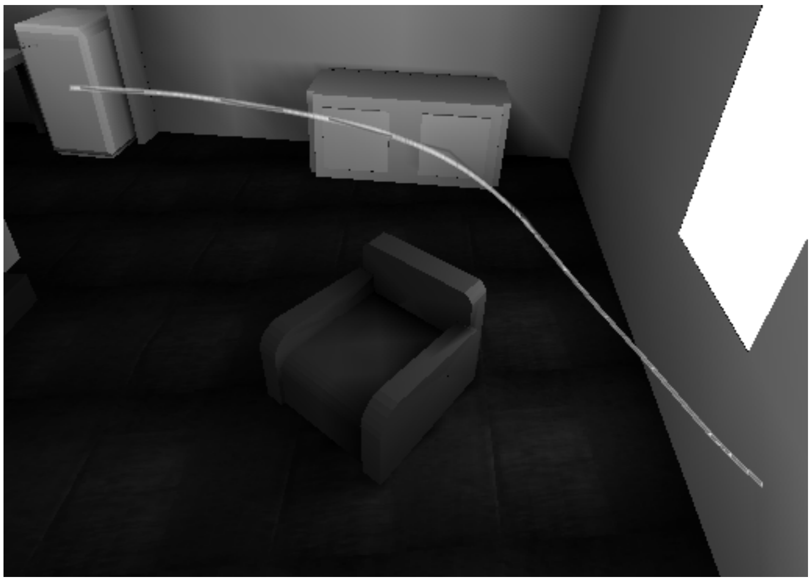


Figure 9. *Initial and reduced path for the slow camera motion.*

Window size	Control points position	Control points rotation	Average window time position (ms)	Average window time rotation (ms)	Total time position (ms)	Total time rotation (ms)
10	20	10	46	28	2200	1300
20	16	13	63	53	1500	1300
30	16	12	110	69	1800	1100
50	12	10	193	122	1900	1200
100	8	10	432	328	2200	1600

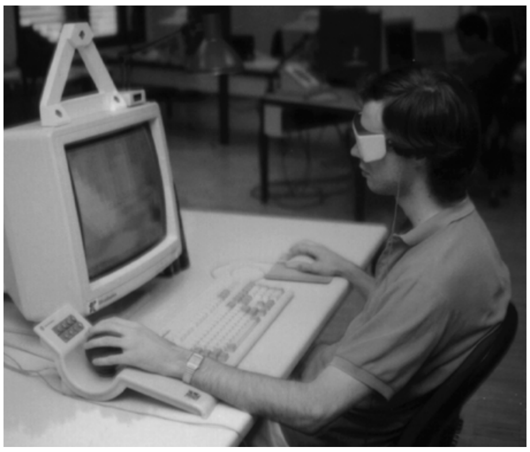
Table 10. *Results obtained by processing the long motion using different window sizes.*

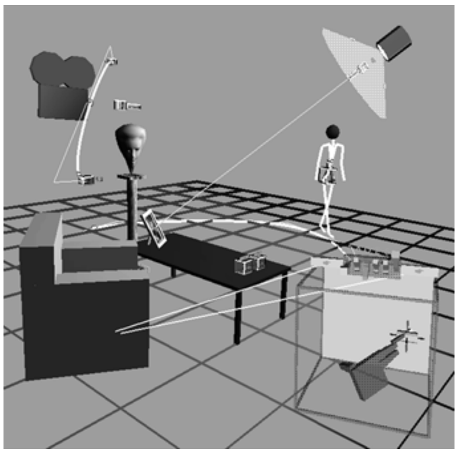
Again, the total processing times are very similar for all window sizes and are kept very inferior to the time needed to specify the motion. Also, for all tested window sizes, the responsiveness requirement can be satisfied when processing the sampled data in parallel with its specification.

The algorithm was implemented in *Eiffel* 2.3 [29] using the linear algebra libraries *BLAS* [22] and *LAPACK* [1]. When computing an approximation, the least-square minimization problem is solved using *QR* decomposition [35]. All times were obtained on a *R4000 SGI Indigo*^{2}.

3. An Integrated Environment to Visually Construct 3D Animations

Our data reduction algorithm has been incorporated in *Virtual Studio*, a prototype 3D animation environment where all the interaction is done directly in three dimensions [5; 18; 4]. 3D devices allow the specification of complex 3D motion, while virtual tools are visible mediators that provide interaction metaphors to continuously control attributes of application objects [17; 16]. By recording the effects of user's manipulations and taking into account the temporal aspect of the interaction, straight-ahead animations can be defined. The algorithm is automatically applied to continuous parameter evolution to obtain editable representations.





(a) (b)

Figure 11a. *Virtual Studio's desktop configuration.*
Figure 11b. *Interactive 3D animation environment.*

3.1. System Principles

Virtual Studio is built on top of *VB2*, a graphics architecture based on objects and constraints [17; 16]. Its desktop configuration uses a *Spaceball* and a mouse as input devices, and *LCD* shutter glasses for binocular perception of the synthetic world. The *Spaceball* is used for the continuous specification of spatial transformations, while the mouse is used as a picking device.

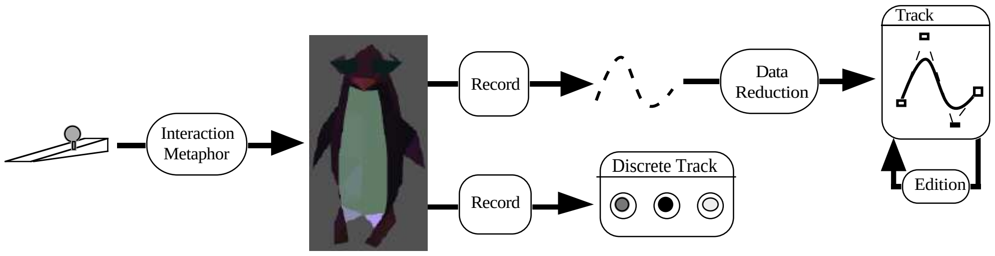


Figure 12. *Principle of interactive animation specification.*

Controller objects are connected to each animatable model and are responsible for monitoring the model's state changes and updating the model's state during animation playback. Controller objects are composed of a collection of continuous and discrete tracks that are used to store the evolution of continuous and discrete parameters. While recording, all model state changes are handled by the controller to feed the animation tracks. Continuous tracks apply our incremental data reduction algorithm each time enough data has been collected or when the recording session terminates, while discrete tracks simply store change value events.

As when manipulating a model using a virtual tool [17], the desire to animate a model is expressed by binding a controller to a model. This action results in the activation of a set of multi-way constraints between the controller's and the model's active variables. When all constraints have been successfully activated, the model is ready to be animated. In the current state of the system, animation controllers are not graphical objects and thus the binding action cannot be performed interactively.

3.2 Sketching 3D paths

A 3D cursor, controlled by the *Spaceball*, is used to select and manipulate objects in the synthetic world. Since the device offers continuous and simultaneous specification of three-dimensional transformations, it is possible to describe with the 3D cursor complex spatial paths that animated objects have to follow. In this way, the animator has continuous control over the animation shape and timing, while keyframing techniques offer control only at a limited number of points. In particular, subtle synchronizations between the position and orientation components of an animation path can be directly specified. As noted in [28; 45], such a guiding technique can provide the mean for interactively describing a large variety of animations if the interaction with the controlled objects is performed at the task level, meaning that the animated objects interpret the path as a goal. This allows to treat animated objects as virtual actors and to automatically obtain secondary animations. In *Virtual Studio*, we currently distinguish between three kinds of primitive objects with respect to their response to guiding: rigid objects, whose motion is an exact replication of the input, articulated chains, whose motion is computed by an inverse kinematics algorithm [26], and virtual humans, which exhibit a walking behavior [8].

3.3 Sketching All Continuous Attributes

The interactive sketching of animated behaviors is not limited to the specification of 3D paths, but can be extended to the definition of all kind of attributes by using mediator objects that continuously map user's actions to changes in manipulated models. The application of data reduction to all kinds of attributes is possible because our incremental algorithm is not based on geometric constructions and is able to handle splines of various dimensions with different levels of continuity.

The use of controller objects that monitor changes in application objects and automatically feed animation tracks makes it possible to record animations that are generated in a number of ways and to convert them to a manageable form, allowing this way further editing. In particular, all the expressiveness of the system's user interface can be exploited when defining animations.

The bi-directionality of the relationship between user-interface and application objects makes it possible to keep the system consistent during playback and to seamlessly integrate interaction and animation features. During playback, information propagates from the animation tracks through the controllers and down to the models. All connections are realized by multi-way constraints. Since the priority of playback constraints is lower than that of interaction constraints, the user can take interactive control over animated models during playback. Therefore, animations which take into account the evolution of the environment and which present complex synchronizations with the animation of other models can be easily specified by interacting with the system during playback. That way, we can support a layered approach to animation specification, where each new piece is automatically synchronized with the rest of the animation.

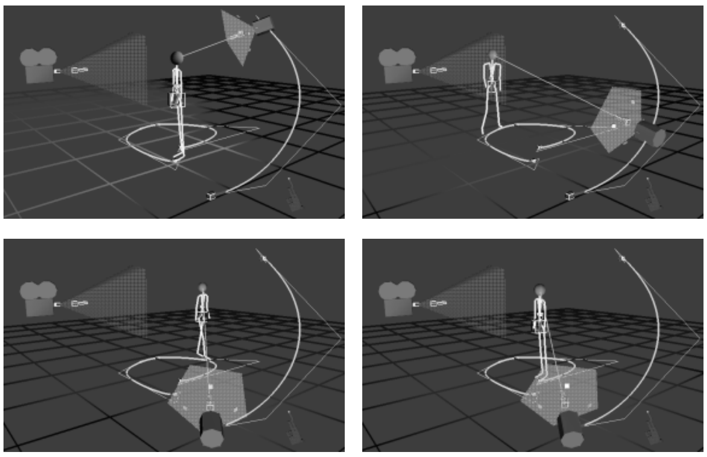


Figure 13. *Sketching the light's path during animation playback using constrained manipulation.*

4. Conclusions and Future Work

We have presented a data reduction algorithm that incrementally builds, from the input sequence, a parametric B-spline preserving value and time of each input sample within a given tolerance. The algorithm considers only a small piece of the total curve at any time and the processing of the user's captured motion may be performed in parallel with its specification. By constraining the approximation, we are able to guarantee constant latency time and memory needs, and so for input motions composed of any number of samples. The algorithm is used inside an integrated environment for the visual construction of 3D animations in order to automatically obtain editable representations of continuous parameters' evolution.

Our future work will concentrate on improving the handling of discontinuities and orientation curves. In its current state, the algorithm does not make any provision to preserve key features such as discontinuities and pauses, and a preprocessing step should be introduced in order to identify them automatically as in [34]. Also, the representation of orientation curves using Euler angles is not satisfactory for free spatial rotation motion. We are investigating the extension of the data reduction algorithm to quaternion B-splines. We plan to solve the approximation problem with iterative techniques similar to those presented in [31] to compute quaternion interpolation B-splines.

With our research, we strive to bring the expressiveness of real-time motion capture systems into a general purpose multi-track system running on a graphics workstation. Data reduction is an essential step towards this goal.

Acknowledgments

The authors would like to thank Ronan Boulic for providing the walking engine.

This research was conducted by the authors at the Swiss Federal Institute of Technology in Lausanne and was partly sponsored by *Le Fonds National Suisse pour la Recherche Scientifique*.

References

1. Anderson E, Bai Z, Bischof C, Demmel J,, DuCroz J, Greenbaum A, Hammarling S, McKenney A, Sorensen D (1990) *LAPACK: A Portable Linear Algebra Library for High-Performance Computers*. CS-90-105, Computer Science Department, University of Tennessee, Knoxville.
2. Baecker RM (1969) Picture-driven Animation. *Proc. Spring Joint Computer Conference* 34: 273-288.
3. Balaguer JF, Gobbetti E (1993) *Data Reduction for Animation*. Demonstration Video, Computer Graphics Lab, Swiss Federal Institute of Technology, Lausanne, 7 min.
4. Balaguer JF (1993) *Virtual Studio: Un système d'animation en environnement virtuel*. PhD Thesis 1178. EPFL DI-LIG, Switzerland.
5. Balaguer JF, Gobbetti E (1995) Animating Spaceland. To appear in *IEEE Computer Special Issue on Real World Virtual Environments* 28(7).
6. Banks MJ (1989) *A User Interface Model and Tools for Geometric Design*. CS-TR-89-001, Computer Science Department, University of Utah, USA.
7. Banks MJ, Cohen E (1990) Realtime Spline Curves from Interactively Sketched Data. *Proc. SIGGRAPH Symposium on Interactive 3D Graphics*: 99-107.
8. Boulic R, Magnenat-Thalmann N, Thalmann D (1990) A Global Human Walking Model with Real-time Kinematic Personification. *The Visual Computer* 6(6) : 344-358.
9. Bryson S, Pausch R, Robinett W, van Dam A (1993) *Implementing Virtual Reality*. SIGGRAPH Course 43.
10. Chou JJ, Piegl LA (1992) Data Reduction Using Cubic Rational Splines. *IEEE Computer Graphics & Applications*: 60-68.
11. Cohen E, Lyche T, Riesenfield R (1980) Discrete B-splines and Subdivision Techniques in Computer-Aided Geometric Design. *Computer Graphics and Image Processing* 14: 87-111.
12. Conner DB, Snibbe SS, Herndon KP, Robbins DC, Zeleznik RC, Van Dam A (1992) Three-Dimensional Widgets. *SIGGRAPH Symposium on Interactive 3D Graphics*: 183-188.
13. Dierckx P (1982) Algorithms for Smoothing Data with Periodic and Parametric Splines. *Computer Graphics and Image Processing* 20: 171-184.
14. Forcade T (1993) Evaluating 3D on the High End. *Computer Graphics World*, Oct/Nov.
15. Gleicher M, Witkin A (1992) Through-the-Lens Camera Control. *Proc. SIGGRAPH:* 331-340.
16. Gobbetti E (1993) *Virtuality Builder II: Vers une architecture pour l'interaction avec des mondes synthétiques*. PhD Thesis 1191. EPFL DI-LIG, Switzerland.
17. Gobbetti E, Balaguer JF (1993) VB2: A Framework for Interaction in Synthetic Worlds. *Proc. UIST*: 167-178.
18. Gobbetti E, Balaguer JF (1995) An Integrated Environment to Visually Construct 3D Animations. *Proc. SIGGRAPH*.
19. Herndon KP, van Dam A, Gleicher M (1994) Report: Workshop on the Challenges of 3D Interaction. *CHI* Bulletin, Oct.
20. Klein D (1990) *Le bruit du frigo*. Computer Animated Movie, EPFL-DI LIG.
21. Lasseter J (1987) Principles of Traditional Animation Applied to 3D Computer Animation. *Proc SIGGRAPH*: 35-44.
22. Lawson, Hanson, Kincaid, Krogh (1979) Basic Linear Algebra Subprograms for FORTRAN Usage. *ACM Transactions on Mathematical Software*, Vol. 5: 308-323.
23. Lyche T, Mørken K (1987) A Discrete Approach to Knot Removal and Degree Reduction Algorithms for Splines. In Mason JC, Cox MG (Ed) *Algorithms for Approximation*. Clarendon Press, Oxford: 67-82.
24. Lyche T, Mørken K (1987) Knot Removal for Parametric B-spline Curves and Surfaces. *Computer Aided Geometric Design 4*: 217-230.
25. Lyche T, Mørken K (1988) A Data Reduction Strategy for Splines with Applications to the Approximation of Functions and Data. *IMA Journal of Numerical Analysis 8*: 185-208.
26. Maciejewski AA (1990) Dealing with Ill-Conditioned Equations of Motion for Articulated Figures.

IEEE Computer Graphics and Applications 10(3): 63-71.

27. Mackinlay JD, Robertson GG, Card SK (1991) The Perspective Wall: Detail and Context Smoothly Integrated. *Proc. SIGCHI*: 173-179.
28. McKenna M, Pieper S, Zeltzer D (1990) Control of a Virtual Actor: The Roach. *Proc. SIGGRAPH Symposium on Interactive 3D Graphics*: 165-174.
29. Meyer B (1992) *Eiffel: The Language*. Prentice-Hall.
30. Newell A (1987) *Unified Theories of Cognition*. Harvard University Press.
31. Nielson GM, Heiland RW (1992) Animated Rotations using Quaternions and Splines on a 4D Sphere. *Programming and Computer Software*, Plenum Publishing Corporation, New York.
32. NSF (1992) Research Directions in Virtual Environments. *NSF Invitational Workshop,* UNC at Chapel Hill: 154-177.
33. Ostby E (1986) Describing Free-Form 3D Surfaces for Animation, *Proc. SIGGRAPH Symposium on Interactive 3D Graphics:* 251-258.
34. Plass M, Stone M (1983) Curve Fitting with Piecewise Parametric Cubics. *Proc. SIGGRAPH*: 229-239.
35. Press WT, Teukolsky SA, Vetterling WT, Flannery BP (1992) *Numerical Recipes in C.* Second edition, Cambridge University Press.
36. Pudet T (1994) Real Time Fitting of Hand Sketched Pressure Brushstrokes. *Proc. EUROGRAPHICS*: 205-220.
37. Robertson GG, Mackinlay JD, Card SK (1991) Cone Trees: Animated 3D Visualizations of Hierarchical Information. *Proc. SIGCHI*: 189-194.
38. Schneider PJ (1988) *Phoenix: An Interactive Curve Design System Based on the Automatic Fitting of Hand-Sketched Curves*. Master's Thesis, University of Washington.
39. Schreiber G (1989) Goopy Real Time Muppets. *Animation Magazine* 2(3):56-57.
40. Shelley KL, Greenberg DP (1982) Path Specification and Path Coherence. *Proc. SIGGRAPH:* 157-166.
41. Tice S (1993) *VActor* Animation Creation Systems. *SIGGRAPH Course #01*.
42. Walters G (1993) Performance Animation at PDI. *SIGGRAPH Course #01*.
43. Ware C, Osborne S (1990) Exploration and Virtual Camera Control in Virtual Three Dimensional Environments. *Proc. SIGGRAPH Workshop on Interactive 3D Graphics*: 175-183.
44. Wu SC, Abel J, Greenberg D (1977) An Interactive Approach to Surface Representation. *CACM* 20(10): 703-712.
45. Zeltzer D (1991) Task-level Graphical Simulation: Abstraction, Representation, Control. In Badler NI, Barsky BA, Zeltzer D (Editors) *Making Them Move*, Morgan Kaufmann, San Mateo, CA, USA.
46. Zeltzer D, Pieper S, Sturman DJ (1989) An Integrated Graphical Simulation Platform. *Proc. Graphics Interface:* 266-274.