The Visual Computer

A technique for motion specification in computer animation

C.W.A.M. van Overveld

Eindhoven University of Technology, P.O. Box 513, 5600 MB, Eindhoven, The Netherlands

A technique is discussed for the interactive specification and real-time evaluation of the movements of geometrical objects with several degrees of freedom. The technique is a midway between a faithful simulation of the dynamics and kinematics of the object and completely free user control, and as such it is well suited for animated drawings, for example. The method works in two steps: first, the motion of a relevant subobject (the skeleton) is specified interactively in real time. Next, the entire object is deformed and oriented in space such as to match the form of the skeleton. Depending on the complexity of the object, this second step either takes place in real time, or as a batch process. Several forms of geometrical constraints, as well as stretching and squeezing, are supported.

Key words: Animation – Interactive motion specification – Approximated dynamical simulation – Distance weighted interpolation

1 Introduction

One of the most challenging subjects in computer animation is motion specification. A well-known method of specifying the motion of a geometric object for computer animation is *key framing* (Reeves 1981). With key framing, the form and orientation of the object are given at certain instances of time. Next, using some more or less sophisticated interpolation rule, the object's form and position are computed for the intermediate instances. From a mechanical point of view, when the key framing technique is applied the object's motion may be said to be specified by the motion equation

$$r_i = r_i(t) \tag{1}$$

where t is the time and i distinguishes between the points with position vectors r_i of the object.
The conceptually more advanced, so-called *procedural, methods* for motion programming (Reynolds 1982) are related to key framing; they also work by explicitly evaluating Eq. (1). Whereas key framing and procedural methods may be quite appropriate to model a certain class of motions that are not too complex or need not bear too much resemblance to reality, it turns out that, using these techniques, a very skillful animatior is needed even to program reasonably realistic movements of nontrivial objects (such as falling or tumbling objects with internal degrees of freedom), let alone human or animal locomotion. In these cases, several authors (Barzel and Barr 1988; Hahn 1988; Moore and Wilhelms 1988; Platt and Barr 1988; Terzopoulos and Fleischer 1988; Wyvill 1988) advocate an alternative way of describing the motion of objects, namely *dynamical simulation* based on the application of Newton's law:

$$\begin{aligned} \ddot{r}_i(t) &= F_i(t)/m_i, \\ \dot{r}_i(0) &= v_{i;0}, \\ r_i(0) &= r_{i;0}. \end{aligned} \tag{2}$$

Here, F_i and m_i are the forces that act on point i and its mass, respectively; $v_{i;0}$ and $r_{i;0}$ are its initial speed and its initial location. These equations hold for every individual rigid component of an articulated object. The majority of the relevant forces, responsible for the relative motion of the components and for the motion of the center of gravity of the entire object, are time-dependent constraint forces. Constraint forces operate on hinge points between the components of the object in order to keep the components assembled. Since these forces are generally not known a priori, Eq. (2) in turn needs to be derived. Two methods can be used:

The Visual Computer (1990) 6:106–116

1. Variational calculus starting from a Lagrangian of the dynamical system yields the so-called Euler equations (Sneddon 1976). This technique holds essentially for frictionless motions only; moreover, the mathematical derivations depend heavily on the topology of the system, i.e., on the way in which its degrees of freedom are coupled. In some very special cases the motion equations thus derived may be solved analytically (e.g., the harmonic oscillator, the Kepler problem). In the more general case, only numerical integration of the motion equations can apply. Moreover, in the case of geometrical constraints on the points, i.e., when the points are coupled, for example with stiff rods or hinged joints, even the mere derivation of the equations may be a quite cumbersome task, needing the application of a symbolic algebra software package. (For a report on the animation of a tumbling human figure, performed both by direct user control and by deriving the complete motion equations, see Wyvill 1988).
2. Using *inverse dynamics*, the forces can be derived from the observation that the articulated components of an object move in such a way as to keep the object assembled while moving. Apparently, there are forces that compensate for the differences in acceleration between the several components of the objects, and these are precisely the constraint forces introduced above. They are the unknown forces from Eq. (2). This method is used in by Platt and Barr (1988) and Hahn (1988). According to Barzel and Barr (1988), the time-dependent effect of the constraint forces can be controlled; this allows the animated object to "assemble itself" while moving.

In both methods, the motions of the components of the moving object are dealt with in a coupled manner. In method 1, the coupling resulting from the way in which the constraints are described in the Lagrangian gives rise to a set of coupled nonlinear differential equations. In method 2, both the unknown constraint forces and the accelerations they cause have to be define from a set of coupled nonlinear algebraic equations for every time instance.
As a result of these couplings, animation computations based on simulation of Newtonian dynamics are both conceptually complicated and computationally expensive. They are, therefore, not (yet) well suited for practical animation work: motion planning, which is often a matter of trial and error, should preferably take place in an interactive and user-friendly manner.
In this paper a method for motion specification and evaluation is described that attempts to bridge the gap between a complete simulation of the dynamical equations and a completely free user control of the motion. The method is based upon two premises:
First, a distinction is being made between a skeleton and the rest of the object. The skeleton is the part that is actually animated, using real-time interaction and/or a procedural approach to specify the motion of some of its points combined with dynamical simulation to compute the motion of the rest of the skeleton. The motions of the rest of the object are directed by the motions of this skeleton. The idea is to reduce the number of degrees of freedom, and hence the amount of computational effort, and to avoid the complexity of dealing with continuum mechanics. [Terzopoulos and Fleischer (1988) and Patt and Barr (1988)] used continuum mechanics to simulate inelastic deformations, viscoelasticity, etc.]
Next, in order to provide real-time performance of the dynamical simulation, the evaluation of the constraint forces that control the motion of the skeleton takes place in an approximate way instead of in the exact way. The approximation consists of first neglecting the coupling between the degrees of freedom, and afterwards applying a correction to compensate for the neglect.

The method is characterized by the following four properties:

0. One and the same formalism is used for every kind of wire frame object, irrespective of its topology;
1. Every point i either has a fully controlled motion $r_i = r_i(t)$, or its motion follows from the evaluation of a set of equations equivalent to Eq. (2), where the F_i's account for both internally or externally applied forces (muscular force, gravity, etc.) and reaction forces due to geometrical constraints (constraint forces). The essential difference between our method and the exact method based on the Euler equations is that in our method the location of every point for a later time instance is computed independently of all other points, and the geometrical constraints (which of course are then violated) are re-established by later adapting the velocities in an iterative manner. This is done by first estimating the reaction forces that are caused by the viola-

tion of constraints, and next treating these reaction forces as if they were acting on the points in an uncoupled manner. The decoupling of the motion equations thus achieved allows for an essentially faster evaluation. To date, three questions are still open:
1.0. Does this iteration converge in all cases?
1.1. Does a faster converging scheme exist?
1.2. If it converges, does this approach converge to the exact solution of the Euler equations?
Experiments show that in practice a rapid convergence occurs all the time. Moreover, experiments with systems of linear chains with several components (series of linked rods) show that the solutions for the motion equations obtained with this algorithm asymptotically converge to the (numerical) solutions of the set of exact nonlinear Euler equations for the system. This convergence turns out to be of first order in the time step. Although this does not prove the correctness (in a physical sense) of the method in general, it at least shows that for a certain class of dynamical systems the approximation seems to be a valid one.

2. Three sorts of geometrical constraints are supported:
2.0. A stiff rod or a telescopic rod with limited length interval;
2.1. A hinged joint with a limited angular interval;
2.2. An elastic wall.

3. Computing the reaction forces in order to reestablish the geometrical constraints takes place in an iterative scheme. This has the following consequences:
3.0. By setting a limit to the number of iterations, a maximum evaluation time per frame may be guaranteed.
3.1. When the iteration has not completely converged, the constraints have not exactly been met. This turns out to give an effect as if the stiff rods and hinged joints are replaced by damped springs. The result is a nicely adjustable amount of squeeze and stretch effect; no unwanted oscillations occur. [The effect gained is similar to the effect of the time-dependent constraints of Barzel and Barr (1988); it is achieved, however, in an entirely different way].
3.2. As stated above, it is not yet clear whether the decoupled approach converges in all cases to the exact solution. It will be shown, however, that both internal momentum and internal angular momentum of the system are conserved (provided that no internal forces and torques are applied), as they should be in an exact solution. It appears that these conservation laws are essential to a realistic motion.

In Section 2 of this paper, the algorithm for the motion evaluation is presented. Section 3 deals with the problem of how to use an animated skeleton to direct the motion of a more complete geometrical object. Finally, Section 4 gives the results of some experiments with a preliminary implementation of the algorithm and some directions for future work.

2 The algorithm

The skeleton to be animated is a nonrigid wire frame figure. In every point of the wire frame the actual location, the velocity, the mass and the new location are recorded. Initially, the locations are given by the skeleton in rest position. (In the implementation that was build for generating the example in this paper, the skeleton could be drawn using an interactive wire frame editor.) The initial values for the velocities may be zero, or some arbitrary values may be assigned to the velocities. It is assumed that during the motion parts of the skeleton move under external control (either interactively or using a script or procedural method; in the current implementation a mouse might be used to steer the trajectory of one point, whereas the motions of all but this one point were controlled by the algorithm, or the entire skeleton moved completely under control of the dynamic simulation algorithm. In both cases, a constant gravitational field was applied to provide for external forces, in addition to the constraint forces computed by the algorithm). The wire frame is represented by the following data structure (the syntax and semantics of the language C is used to write down data structures and algorithms):

```
int NP,NR,NJ;          /* the number of points, rods and
                          joints */

typedef struct {
  float x,y,z;
  } pnt;
typedef struct {
  pnt r,v,n;           /* r: the actual location;
                          v: the momentum
                          (=velocity * mass);
                          n: the location to be computed
                          for the next time slot */
```

```
  float m;                    /* the mass of the point*/
  boolean s;                  /* s: the point's location is con-
                                 trolled by the user or by a
                                 script;
                                 not s: the point's location is
                                 computed from (2); */
  } point;
point p[NP];                  /* the points of the skeleton */
typedef struct {
  int e1,e2;                  /* the numbers of the two points
                                 connected by a rod */
  float l1,l2;                /* the interval in which the length
                                 of the rod may vary */
  } rod_constraint;
typedef struct {
  int e1,e2,e3;               /* the numbers of the three points
                                 forming a hinged joint (e2 is
                                 the hinge) */
  float a1,a2;                /* the interval in which the angle
                                 of the joint may vary */
  } joint_constraint;
rod_constraint r[NR];         /* the rod constraints of the skele-
                                 ton */
joint_constraint j[NJ];       /* its joint constraints */
pnt pc[NP];                   /* the positions of those points
                                 having s=true */
```

The algorithm is presented in a top-down manner. Statements and guards to be expanded in successive refinement steps are indicated by S and G, respectively. The form of these statements and the form of the guards may be derived from the assertions that surround it (the pre and post conditions). For simplicity, the assignment, addition and subtraction operators are defined to work for both scalars and vectors. The cross product and the matrix-vector product are notated by the operator '×.' The scalar-vector product is denoted by the usual multiplication symbol '*.' Next, the algorithm for the entire dynamic simulation scheme is presented. Statement 1 accounts for the infinite loop that causes the simulation to be repeated for every time slot. The computations for one time slot comprise the following steps:

- Applying external forces (gravity, etc.) to adjust all momenta (statement 2).
- Computing new locations for all points, either by assigning the externally given values to them (for the interaction- or script-driven points), or computing new locations by using the current momenta, i.e., using a first-order Euler method for solving the uncoupled motion equations while neglecting the constraints (statements 3, 4).
- Computing in an iterative manner corrections to all momenta in order to fulfill the constraints (statements 5 to 17). The following actions are taken in this iteration:
- Initializing the counter that limits the number of iteration steps ('loop_count'), the boolean that indicates whether the constraints have been met ('ok') and looping (statements 5 to 8 and 15).
- Checking and, if necessary, taking care for every rod constraint. Note that all rod constraints are dealt with independently of each other (statements 9, 10).
- Checking, and, if necessary, taking care of every joint constraint. Note that all joint constraints are dealt with independently of each other (statements 11, 12).
- For all points, moving under control of the algorithm, the new locations are computed using the corrected momenta (statements 13, 14).
- Copying the newly computed locations to the actual locations (statement 16).
- Dealing with collisions with rigid walls (statement 17).

```
                /* initially, all constraints are being met for all
                   p[i].r */
1   while (true)
    {
                /* all constraints are being met (within toler-
                   ance) for all p[i].r */
2     S0;
                /* all internal and external forces have been
                   applied, resulting in changed momenta */
3     for (i=0; i<NP; i=i+1)
4       if (p[i].s) p[i].n=pc[i];
          else p[i].n=p[i].r+p[i].v/p[i].m;
                /* points p[i] that are under direct user con-
                   trol have their proper new value p[i].n; for
                   other points the motion equations have been
                   integrated independently of another one
                   time step, thereby possibly violating the con-
                   straints. */
                /* start iteration loop to re-establish the con-
                   straints */
5     loop_count=0;
6     do
      {
7       ok=true;
8       loop_count=loop_count+1;
9       for(i=0; i<NR; i=i+1)
10      if (G1(i)){ok=false; S1(i);}
                /* rod constraint i is not fulfilled for
                   p[r[i].e1].n and p[r[i].e2].n; the momenta
                   of the points involved are adapted such that
```

```
                rod constraint i is fulfilled, thereby possibly
                violating other constraints */
11      for(i=0; i<NJ; i=i+1)
12      if (G2(i)){ok=false; S2(i);}
                /* joint constraint i is not fulfilled for
                p[j[i].e1].n, p[j[i].e2].n, and p[j[i].e3].n;
                the momenta of the points involved are
                adapted such that joint constraint i is ful-
                filled, thereby possibly violating other con-
                straints */
13      for(i=0; i<NP; i=i+1)
14      if (!p[i].s) p[i].n=p[i].r+p[i].v/p[i].m;
                /* the adapted momentum is used to compute
                a new guess for the new location */
15    } while ((!ok)&&
            (loop_count<max_loop_count))
                /* all momenta have such values that for all
                points p[i] for both p[i].r and p[i].n all
                constraints are fulfilled, or the maximal
                number of iterations has been reached. In
                the latter case, stretch or squeeze occurs. */
16    for(i=0; i<NP; i=i+1) p[i].r=p[i].n;
                /* for all p[i].p all constraints are fulfilled, or
                stretch or squeeze occurs. */
17    S3;
                /* the constraints corresponding with collisions
                with the walls have been taken into ac-
                count */
    };
```

Next, the guards G1 and G2 and the statements S1 and S2 are given (the statement S0 is both application specific and trivial and will not be given explicitly here).

```
float l;
l=euclidean_distance(p[r[i].e1].n, p[r[i].e2].n);
```

Guard G1(i):

```
1  G1=((l<r[i].l1)||(l>r[i].l2));
```

Statement S1(i):

```
2  float d;
3  pnt c;
4  if (l<r[i].l1) d=(l-r[i].l1)/l;
     else d=(l-r[i].l2)/l;
5  c=d*(p[r[i].e1].n-p[r[i].e2].n);
6  p[r[i].e1].v=p[r[i].e1].v-c/2;
7  p[r[i].e2].v=p[r[i].e2].v+c/2;
```

In statement 1 of the above code fragment, the actual distance, l, between the two endpoints r[i].e1 and r[i].e2 is checked against the allowed minimal and maximal lengths, r[i].l1, and r[i].l2 of the rod constraint i. The value of d as declared in statement 2 and computed in statement 4 stands for the relative length excess (either positive or negative). This length excess is assumed to introduce a reaction force with magnitude proportional to d, directed along the rod. It is represented by the vector c (statements 3, 5). This reaction force, in turn, is assumed to change the momenta of the two endpoints (statements 6, 7).

The reaction force as computed above is depicted in Fig. 1; it is such that it introduces no total momentum (the momentum corrections, $-c/2$ and $+c/2$, add up to zero) and also no total angular

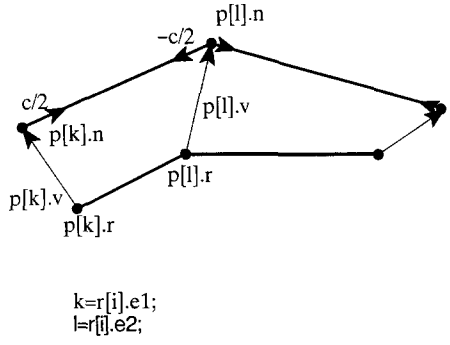


Fig. 1. The correction of velocities to re-establish a rod (distance) constraint

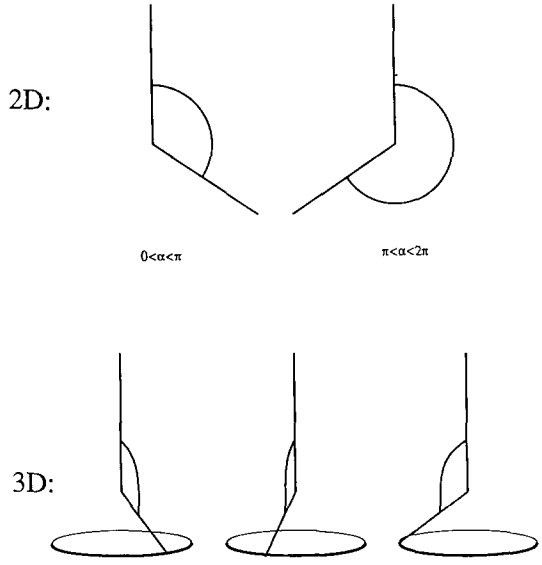


Fig. 2. The distinction between positive and negative angles is only relevant in 2 dimensions

momentum (indeed: the two vectors $-c/2$ and $+c/2$ work along the same line).
A similar approach is used to derive guard G2 and statement S2 for the joint constraints (see Fig. 3):

```
pnt h1,h2,h3,h12,h23;
float a;

h2 = p[j[i].e2.n;        /* the hinge point */
h1 = p[j[i].e1].n;       /* the point before the hinge */
h3 = p[j[i].e3].n;       /* the point behind the hinge */
h12 = h2 - h1;
h23 = h3 - h2;
                         /* first compute the angle between
                            h1 - h2 and h3 - h2 */
a = arccos(inner_product(h23, h12)/
      (length(h12) * length(h23));
                         /* in the 2D case, a distinction between
                            angles between 0 and π and between
                            π and 2π should be made, as is de-
                            picted in Fig. 2. This distinction can
                            be made by evaluating the z-compo-
                            nent of the cross product of h12 and
                            h23: */
```

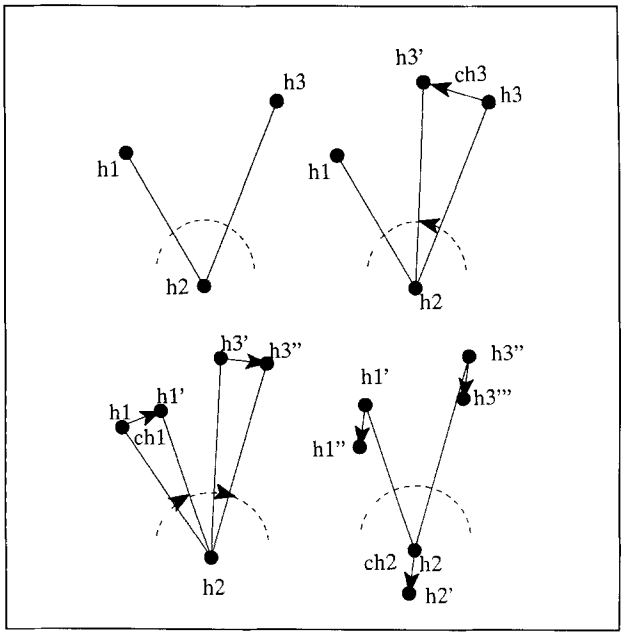


Fig. 3a–d. The correction of velocities in order to (re-)establish a joint takes place in three steps: **a** h1, h2, h3 form the original hinge which configuration violates the joint constraint; **b** h3′ is the rotated image of h3; **c** h1′, h3″ are the rotated images of h1, h3′, respectively, to compensate the angular momentum; **d** h1″, h2′, h3‴ are the shifted images of h1′, h2′, h3″, respectively, to compensate for the momentum

```
if (h12.x * h23.y < h12.y * h23.x) a = 2 * π - a;

Guard G2(i):
                         /* check a against j[i].a1 and j[i].a2 */
1   G2 = ((a < j[i].a1) || (a > j[i].a2));

Statement S2(i):
    pnt ch1,ch2,ch3,torque,t_corr,m_corr;
    float da;
                         /* first, compute a correction momen-
                            tum vector ch3 to be added to h3 to
                            re-establish the joint constraint that
                            corresponds simply to rotation of
                            h3 - h2 round the hinge h2 */
2   if (a > j[i].a2) da = j[i].a2 - a;
      else da = j[i].a1 - a;
3   ch3 = rotation_matrix(angle = da; center = h2;
      axis = (h12 × h23)) × h3 - h3;
                         /* this correction may have introduced
                            torque. Its size is: */
4   torque = ch3 × h23;
                         /* correct for the torque by rotating h1
                            and h3 round h2. The rotation will
                            be effectuated again by adding cor-
                            rection vectors ch1 and ch3 to h1 and
                            h3, respectively. Using congruence,
                            the rotation reduces to: */
5   t_corr = torque * (1/(length(h12)^2
      + length(h23)^2));
6   ch3 = ch3 + h23 × t_corr,
7   ch1 = h12 × t_corr;
                         /* the above corrections (rotations)
                            might have introduced momentum. Its
                            size is: */
8   m_corr = ch3 + ch1;
                         /* compensate for it by adding
                            -m_corr/3 to the momentum correc-
                            tion vectors: */
9   ch1 = ch1 - m_corr/3;
10  ch2 =     - m_corr/3;
11  ch3 = ch3 - m_corr/3;
                         /* this completes the angular momentum
                            and momentum corrections */
12  p[j[i].e1].v = p[j[i].e1].v + ch1;
13  p[j[i].e2].v = p[j[i].e2].v + ch2;
14  p[j[i].e3].v = p[j[i].e3].v + ch3;
                         /* this completes the derivation of
                            statement S2 */
```

In the above code fragment, statement 1 checks whether the actual angle between the two edges forming the hinge falls within the allowed interval. In case it does not, an angle excess da is computed in statement 2, and the point h3 is rotated as to correct for this angle excess in statement 3. In statements 4 to 6 corrections are made in order to elimi-

nate the torque that is introduced with this rotation. Finally, in statements 8 to 11, corrections are made to eliminate the momentum that is introduced both with the rotation and with the subsequent correction. As with statement S1(i), statement S2(i) concludes with assignment of the new values to the velocities of the points involved (statements 12 to 14). Statement S3, which accounts for collisions with walls is not derived here; again, as with S0 it is trivial and also depends heavily on the geometry of the walls. In the examples that accompany this paper, a configuration was chosen where the walls form a rectangular box.

3 Transforming the skeleton into an object

The algorithm to compute the motion of a set of points as outlined in Sect. 2 has a time complexity that is proportional to the number of points to be accounted for. For a nontrivial object, the number of points that is needed to fully specify its geometry is far larger than the number of points that specify its internal degrees of freedom. In order to keep the computational effort of the motion computations within reasonable bounds, a distinction is therefore made between a *skeleton* and an *object*. A skeleton is a wire frame structure; its deformations are animated autonomously as outlined above. The object is a geometrical structure that is also deformed but whose deformation is dictated by the deformation of the skeleton. In this way it is not necessary to do the dynamic computations on all points of the object. The underlying 'physical' idea is that both the mass and the internal reaction forces of the object are concentrated in the skeleton and that the rest of the object has been made of a massless, infinitely flexible jelly.

Several methods could be used to couple the motions of the skeleton to those of the object. Burtnyk and Wein (1976) have developed a method to achieve this effect by assuming that every point of the object is expressed as a linear combination of precisely two skeleton-fixed unit vectors. When these skeleton vectors move, the associated object points move accordingly. Although this method is conceptually very simple and reasonably efficient, it introduces C^1 discontinuities in the objects on the boundaries of the skeleton-fixed unit squares, and it is therefore not suited to yield the impression of elastically deformable objects aimed at with this dynamic simulation model. Instead, a global model should be used where *every* point of the skeleton influences *every* point of the object in the sense that remote skeleton points have less influence on a given object point than nearby skeleton points.

This idea requires the application of distance-weighted interpolants. Assume that at a given time t the skeleton points have been removed from their original locations. Their new locations may be described by the old locations plus a *displacement field*. This displacement field ϕ is a function of the location, to be indicated by r, and of the time t. When this same displacement field is added to the points of the object, a deformed object results which nicely displays the smooth deformation induced by the skeleton.

Let $r_i(t)$ be the location of points of the skeleton at time t, and $r_{i;0}$ the location of points of the skeleton in its initial, undeformed state. The geometry of the object is given by $q_j(t)$ and $q_{j;0}$, respectively. These points may be thought of either as vertices of polygons, if the object is bounded by planar faces, or as control points if the object is bounded by curved surfaces. The properties of the field as introduced above are as follows:

0. Let the field be denoted by $\phi(r;t)$, $\phi:\mathbf{R}^3\times\mathbf{N}\rightarrow\mathbf{R}^3$. Then the meaning of ϕ is $q_i(t)=q_{i;0}+\phi(q_{i;0};t)$.

1. For points q_j and points r_i with $q_{j;0}=r_{i;0}$, ϕ should be such that $q_j(t)=r_i(t)$. From this, it follows that

$$r_i(t)=q_j(t)=q_{j;0}+\phi(q_{j;0};t)$$
$$=r_{i;0}+\phi(r_{i;0};t).$$

Thus

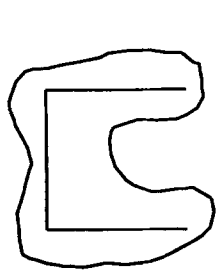
$$\phi(r_{i;0};t)=r_i(t)-r_{i;0}.$$

2. Let i distinguish points of the skeleton and j of the object. For points q_j and points r_i for which $q_{j;0}$ is near to $r_{i;0}$, the location of $q_j(t)$ should be near to $r_i(t)$, and the influence of $r_i(t)$ should decrease when the argument r of $\phi(r;t)$ is further away from $r_{i;0}$. When evaluated, r successively takes the values of all rest points $q_{j;0}$ of the object.

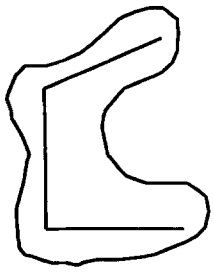
3. The field should be invariant under translations, that is if all locations $r_i(t)$ are shifted over the same vector δ, the field ϕ transforms to $\phi+\delta$.

4. The field should behave smoothly for large values of $|r|$.

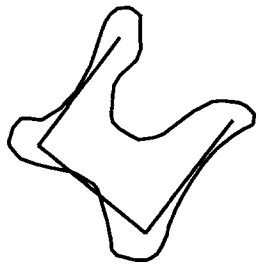
5. For smooth deformations, ϕ should be smooth.



4



5



6

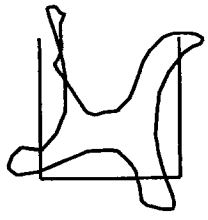
Fig. 4. A skeleton and an object in undeformed state

Fig. 5. Using Eq. (3), the deformation of the skeleton causes a deformation of the object

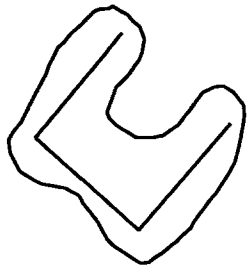
Fig. 6. A rotation of the undeformed skeleton causes unwanted deformations of the object when Eq. (3) is used

Fig. 7. The effect of the unwanted deformation increases with larger rotation angles

Fig. 8. When the translational and rotational components of the deformation field are treated separately, the unwanted deformations of the object are eliminated



7



8

A field function ϕ such that properties 1–5 hold is the following:

$$\phi(r;t)=\frac{\sum_i (r_i(t)-r_{i;0})\,|r-r_{i;0}|^{-2}}{\sum_i |r-r_{i;0}|^{-2}}. \tag{3}$$

This function is evaluated for values of the parameter r equal to all points $q_{j;0}$ of the object. This function is an adaptation made by Barnhill to a distance-weighted interpolation formula of Shephard (Boehm et al. 1984). The validity of properties 2–5 may easily be verified; property 1 may be shown by taking a limit from r to $r_{i;0}$.

In Fig. 4, both a skeleton and an object are drawn for $t=0$. The skeleton consists of four points, connected by three line segments forming roughly the shape of a letter C. The object is the 50-sided polygon surrounding it.

In Fig. 5, the situation is depicted at another time point.

One of the points of the skeleton is shifted to a new location, and the application of field ϕ from Eq. (3) to every one of the points of the objects produces the deformed object from Fig. 5. The deformation is quite acceptable, matching with what one would intuitively expect to happen. In particular, note the occurrence of the smooth concave bend in the upper part of the figure. Had the method of Burtnyk been applied, a first-order discontinuity would have occurred. In the case of a *rotation*, however, unwanted deformations of the object occur (Figs. 6, 7).

This is because ϕ, however invariant under translations, is not invariant under rotations. A remedy may be found by applying the following procedure:

0. Establish the center of gravity c_g of the skeleton, both at time $t=0$ ($c_{g;0}$) and at time t ($c_g(t)$).
1. Reduce all displacement vectors $r_i(t)-r_{i;0}$ by the vector $c_g(t)-c_{g;0}$.
2. For the resulting reduced displacement vectors, $r_i(t)-r_{i;0}-c_g(t)+c_{g;0}$, establish the *rotation vector* with respect to c_g, say ρ_i. A vector Δ, attached to a point p, has a rotation vector with the direction of $p\times\Delta$ and a length equal to $\arccos\dfrac{(p,p+\Delta)}{|p|\,|p+\Delta|}$ (i.e. the angle between p and $p+\Delta$). In this case, Δ is

the reduced displacement vector. Since, in the case of two dimensions, all vectors are in the same plane, the rotation vector reduces to a mere rotation angle.
3. Find the average rotation vector, $\rho_{ave}=\sum_i \rho_i/n$, where n is the number of points of the skeleton.
4. Reduce all displacement vectors $r_i(t)-r_{i;0}-c_g(t)+c_{g;0}$ by the displacement vectors caused by a rotation of the skeleton round axis ρ_{ave}, with respect to c_g, over an angle $|\rho_{ave}|$. If the original displacement of the skeleton was merely a combination of a translation and a rotation round the center of gravity, the remaining displacement vectors by now all vanish. The remaining nonzero displacement vectors account for internal deformations of the skeleton.
5. Apply (3) with the rotated and shifted skeleton to account for the internal deformations of the skeleton to produce a deformed object.
6. Rotate this object using the rotation vector ρ_{ave}.
7. Shift this object over the vector $c_g(t)-c_{g;0}$.

The result of this approach is an extension to the Barnhill formula that is invariant both under translations and rotations round the center of gravity of the skeleton. This is demonstrated in Fig. 8, where the form of the object is computed according to the above prescription.
Although the above manipulations seem to be quite elaborate and therefore time-consuming, one should bear in mind that in general a complex, composite object is decomposed into several smaller objects where every object moves in accordance with a part of the skeleton only. This causes the loops over i in the computation of $\phi(r;t)$ to comprise, in general, only three or four terms at most. Moreover, the task of removing the rotational component of the field may be performed at every third time step, for example instead of every time step.

4 Results and conclusions

4.1 Results

Using the methods described in the previous sections, an implementation for the 2D case is built. It is tested for a simple human figure as depicted in Fig. 9.
The skeleton consists of 16 points and 16 line segments. A rod constraint is associated with all line segments. Both limits of the length interval of this rod constraint are set to the length of the associated line segment in the initial state. Also, 9 joint constraints are included. The user interface for the system is arranged in such a way that a cursor, controlled by a mouse, can be used to move an arbitrary point of the skeleton along the screen. The locations and velocities of all other points of the skeleton are computed accordingly. If only the skeleton is updated, and if the maximum number of iterations is set at 7, and update rate of roughly

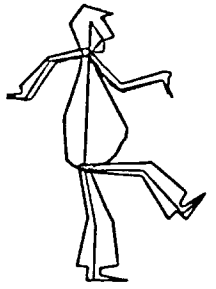


Fig. 9. The figure used for the demonstration exists of a skeleton with 16 edges and 5 objects, with a total of over 90 points

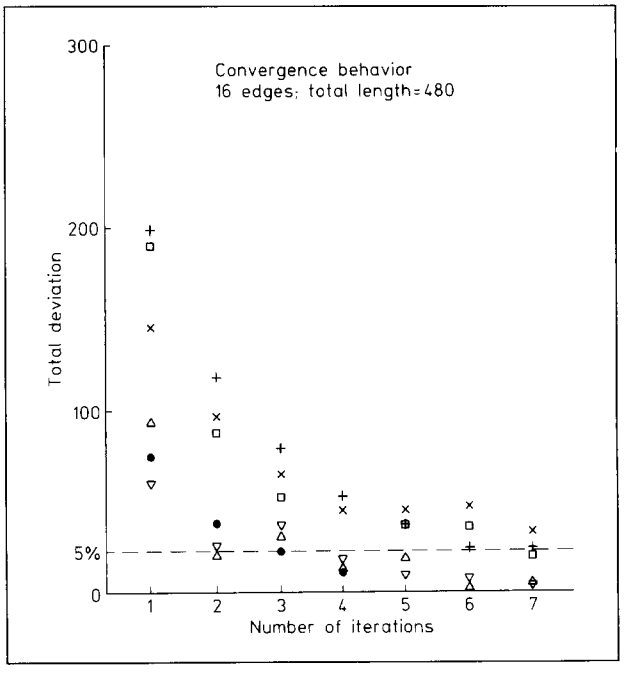


Fig. 10. A graph depicting the convergence behavior of the iteration scheme for 6 arbitrary frame updates (distinguished by different marker symbols). The deviation is plotted as a function of the number of iteration steps

10 frames per s can be achieved on a SUN-4 workstation, which gives a very realistic interactive behavior: one can actually manipulate the figure (throwing it in the air, catching it, having it do somersaults, etc.) as if it were a material object. The convergence behavior of the method is depicted in Fig. 10 for 6 arbitrary frame updates. As to the composition of the objects, there were five distinguished parts of the skeleton (the trunk with the head, both arms and both legs) and there was a different object associated with every part. The objects consisted of approximately 25 points each; they were designed by free-hand drawing with an interactive line drawing tool. The computational cost of the dynamical simulation is roughly equal to that of the computation of the objects.

In this graph, the total deviation of all rod constraints is plotted as a function of the number of iterations. It is observed that, even with initial total deviations as large as 40%, the deviation is reduced to less than 7% within seven iterations.

In Fig. 11 a short animation, composed of 30 frames arranged in columnwise order, is depicted. No interaction takes place in the animation: the figure is maneuvred into its horizontal position, above the floor surface, and dropped. An external gravity force causes the figure to fall; the collision with the floor surface takes place with almost no loss of energy, which causes the figure to do a backward somersault.

4.2 Conclusions

Using a numerical approximation scheme, a method for the simulation of the motion of dynamical mechanical systems has been developed. Although both conceptually and computationally much simpler than a method based on the Euler-Lagrange equations, the essential features of realistic motions are reproduced. Much of this realism is due to the fact that momentum and angular mo-

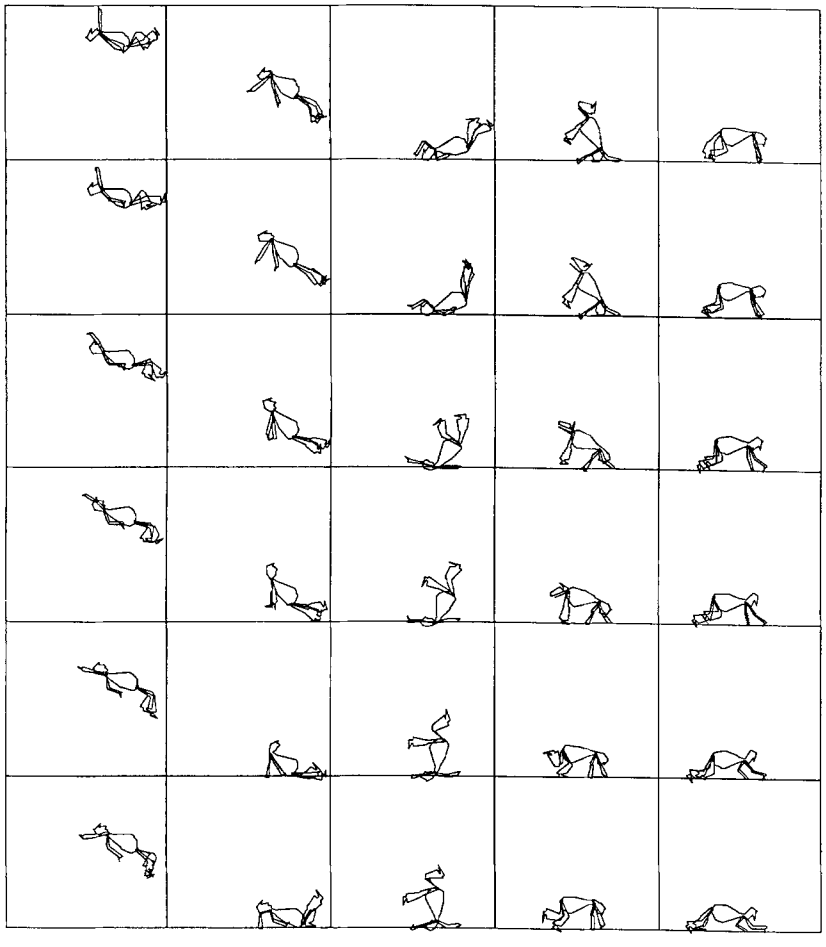


Fig. 11. A 30-frame animation sequence of the human figure shown in Fig. 9. The animation starts *upper left* and subsequent frames are arranged in columnwise order

mentum are conserved. The motion and deformation of a more complex object may be obtained by using this method to animate a skleton, and subsequently using the form and location of this skeleton to dictate the form and location of the object. A distance-weighted interpolation field has been used to describe the coupling between the skeleton and the object.

Although essentially sufficient in 2D, the description of the joint constraints only gives limited freedom to express complex rotational constraints in 3D. The extension of the rotational constraints to cover constraints that occur, e.g., in real human articulation, also must be a subject of future research. Future research must also consider the question of whether a faster (and maybe more accurate) convergence can be achieved using alternative iteration schemes or not, and the question of what the relation is between the motion thus achieved and the motion as resulting from the exact solution of the Euler-Lagrange equations.

Acknowledgements. The author wishes to thank Marloes van Lierop and Huub van de Wetering for careful proofreading of the manuscript.

References

Barzel R, Barr AH (1988) A modeling system based on dynamic constraints. ACM Comput Graphics (Proc SIGGRAPH) 22:179–188

Boehm W, Farin G, Kahman J (1984) A survey of curve and surface methods in CAGD. Computer Aided Geometric Design 1:1–60

Burtnyk N, Wein M (1976) Interactive skeleton techniques for enhancing motion dynamics in key frame animation. Commun ACM 19:564–569

Hahn JK (1988) Realistic animation of rigid bodies. ACM Comput Graph (Proc SIGGRAPH) 22:299–308

Moore M, Wilhelms J (1988) Collision detection and response for computer animation. ACM Comput Graph (Proc SIGGRAPH) 22:289–298

Platt JC, Barr AH (1988) Constraint methods for flexible models. ACM Comput Graph (Proc SIGGRAPH) 22:279–288

Reeves W (1981) Inbetweening for computer animation using moving point constraints. ACM Comput Graph (Proc SIGGRAPH) 15:263–269

Reynolds C (1982) Computer animation with scripts and actors. Comput Graph (Proc SIGGRAPH) 16:289–296

Sneddon IN (ed) (1976) Encyclopaedic dictionary of mathematics for engineers and applied scientists. Glasgow Univ, Pergamon Press, Oxford, pp 707–709 (and references)

Terzopoulos D, Fleischer K (1988) Modeling inelastic deformation: viscoelasticity, plasticity, fracture. ACM Comput Graph (Proc SIGGRAPH) 22:269–278

Wyvill B (1988) Navigating the animation jungle. Presented at the International Summer Institute: State of the Art in Computer Graphics, Exeter, UK (4–8 July 1988)

CORNELIUS W.A.M. VAN OVERVELD is working in computer graphics as of spring 1985 as a staff teacher and researcher in the Department of Mathematics and Computing Science of Eindhoven University of Technology. He has a MSc in physics and a PhD in nuclear physics, also from EUT. In graphics, his main interests are in fundamental aspects of raster algorithms (discretization, rendering), computer animation and user interfaces.

Note added in proof

Since the original manuscript of this paper was submitted for publication, an extension of the method to support 3D-linked articulated objects has been developed. It turns out that with the same hardware, frame update rates, of roughly 10 frames/s may again be reached for 3D objects consisting of 20–30 points, with several internal degrees of freedom and allowing for the same sort of interactive manipulation.