Numerical Techniques for Cloth Simulation

Michael Hauth
WSI/GRIS, University of Tübingen
www.gris.uni-tuebingen.de/staff/Michael_Hauth.html

October 29, 2003

1 Spatial discretisation

The modelling of physical systems often leads to partial or directly to ordinary differential equations. The solution of these equations usually is a dominant part of the total computational costs for a simulation or animation, therefore being the main focus of this chapter. Before addressing the numerical solution of ODEs - the temporal discretization or (time) integration - we take a brief look at the numerical techniques used for spatial discretization. Three techniques, not entirely unrelated, dominate the field. They are normally classified as particle systems, finite difference and finite element methods.

1.1 Particle Systems

When using the particle system paradigm, the discretisation is already a part of the physical modelling process, as the continuous object is immediately represented as a set of discrete points with finite masses. Physical properties are specified by directly defining forces between these mass points. Typical representatives of this approach, that is very popular in cloth simulations, are mass-spring-damper systems [21, 26] and particle systems with forces defined directly by measured curves[6] or low order polynomial fits of this data[2].

1.2 Finite Differences

Another physical modelling concept is to specify physical behavior by minimizing some energy functionals defined on a continuous solid. The arising equations normally contain derivatives with respect to space and time variables. Replacing the spatial derivatives with finite differences, e.g.

$$\frac{\partial}{\partial x} f \rightarrow \frac{f(x+h) - f(x-h)}{2h} \tag{1}$$

and

$$\frac{\partial^2}{\partial x^2} f \rightarrow \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \tag{2}$$

leads to finite difference formulations. This replacement is easily accomplished in 1-space or on structured grids in any dimension. It becomes harder to define finite difference approximations on unstructured meshes[18], often finite element techniques are used for deriving appropriate schemes[19]. An important trait of finite difference schemes is, that the terms on the right hand side of the ODE are given for a point and its neighbors, i.e. are specified on the *edges* of the discretization. The resulting equations are structured very similar to these from particle systems, indeed finite difference techniques can be used to derive a particle system from continuous equations[9].

1.3 Finite Elements

Finite elements, in their beginning, were designed to overcome the difficulties of finite differences with unstructured meshes. They also start with a continuous model, usually given as an integral equation in a weak form, e.g. resulting form a variational ansatz. The

functions are then replaced by a piecewise polynomial approximation over the unstructured discretization of the domain. The number of nodes per element and the polynomial degree of the shape functions can be varied on a single grid. Curvilinear grids allow a very good shape fitting. For visual simulations one usually prefers piecewise linear approximations over triangle, quad-, hexa- or tetrahedral meshes. The finite element method is the preferred solution technique in numerical analysis and engineering applications because of its versatility, sound derivation and superior convergence properties with respect to integral norms, often natural for the problem at hand. The arising equations are given by terms formed over the area *elements* of the mesh. Their drawback is that these improvements are paid for by an increased computational effort, compared to the previous techniques. Also the masses may not be concentrated on the points of the mesh, as in the previous cases, leading to an implicit ODE

$$Mx' = f(x) \tag{3}$$

with a non-diagonal and even singular mass matrix M. A common technique is to use mass lumping, to make M diagonal and reduce computational costs[24, 8].

For the following discussion we assume that the ODE is given in the general explicit form

$$x' = f(x) \text{ or } x'' = f(x, x') \tag{4}$$

2 Methods for Numerical Integration

As we have seen in the previous sections mechanical systems are often given as a second order ordinary differential equation accompanied by initial values

$$\begin{aligned} x''(t) &= f_v\big(t, x(t), x'(t)\big), \\ x(t_0) &= x_0, x'(t_0) = v_0. \end{aligned} \tag{5}$$

The differential equation can be transformed into a first order system by introducing velocities as a separate variable:

$$\begin{bmatrix} x(t) \\ v(t) \end{bmatrix}' = \begin{bmatrix} v(t) \\ f_v\big(t, x(t), v(t)\big) \end{bmatrix}, \quad \begin{bmatrix} x(t_0) \\ v(t_0) \end{bmatrix} = \begin{bmatrix} x_0 \\ v_0 \end{bmatrix}. \tag{6}$$

For the next few sections it will be convenient to write this ODE in the more abstract form

$$y'(t) = f\big(t, y(t)\big), \quad y(t_0) = y_0, \tag{7}$$

before we will come back to the special setting (6) for eventually gaining computational advantages.

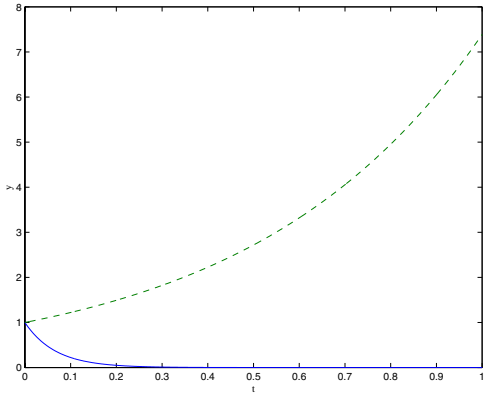


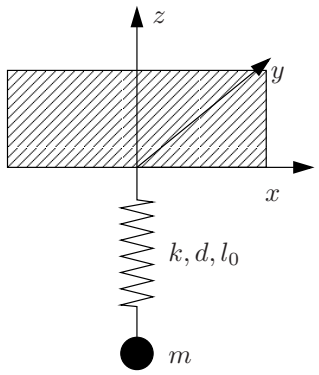
Figure 1: Solutions of example 1 for $\lambda = 2$ (dashed) and $\lambda = -15$ (solid)

Throughout the following discussion we will use the following examples:

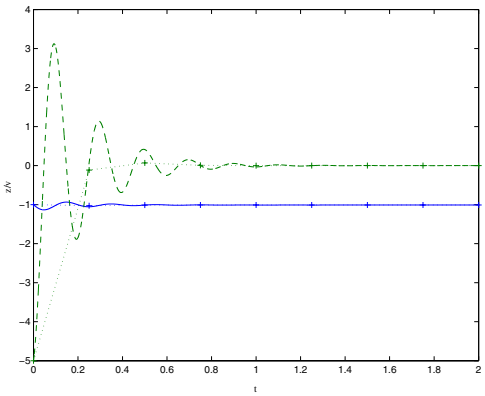
1. $y' = \lambda y$, $y(0) = 1$ with $\lambda = 2, -15$ for $t \in [0, 1]$ (figure 1).

2. The overdamped wave equation $y'' = \lambda/2y + \lambda y'$ with $\lambda = -5$ for $t \in [0.10]$ and starting values $y(0) = 0, y'(0) = 1$. It has the analytical solution $y(t) = 1/15\sqrt{15}e^{1/2\left(-5+\sqrt{15}\right)t} - 1/15\sqrt{15}e^{-1/2\left(5+\sqrt{15}\right)t}$

3. This example is based on a simple mechanical system (figure 2(a)): A particle p with mass m connected to the origin using a spring with stiffness k, damping d and rest length l_0, is pulled down by gravity. This setting is described by the



(a) The mechanical system of example 3.



(b) Exact solution of example 3: z (solid), z' (dashed) and an implicit Euler solution (see below) for large timesteps.

Figure 2: Example 3.

ODE

$$\frac{d^2z}{dt^2} = \frac{k\,(l_0 - z)}{m} - \frac{d}{m}\frac{dz}{dt} + g_z. \tag{8}$$

We set the parameters $m = 0.1$, $k = 100$, $l_0 = -1$, $d = 1$, $g = -10\, v_{0,z} = -5$ and simulate the interval $t \in [0, 2]$ (figure 2(b)).

2.1 Explicit methods

The oldest and most simple method of integration is the so called forward or explicit Euler method. Time is discretised into slices of length h. To get a formula for advancing a time step h, the differential quotient on the left hand side of (7) is replaced by the forward difference quotient

$$\frac{y(t+h) - y(t)}{h} \approx y'(t) = f(t, y(t)). \tag{9}$$

Thus we get the integration formula for advancing a single timestep

$$y(t+h) = y(t) + hf(t, y(t)). \tag{10}$$

Iterating this method gives a sequence of numerical approximations $Y_n \approx y(t_n) =: y(t_0 + nh)$. Geometrically this method can be interpreted as straightly following the tangent of the solution and then recalculating the slope for the next step.

There are several criteria for evaluating an integration method:

- convergence
- accuracy
- stability
- efficiency

Convergence means that for $h \to 0$ the numerical solutions Y_n meet the analytical. All useful methods must be convergent, so we won't discuss non-convergent methods or criteria for convergence. More interesting is the accuracy or order of a method. By this we mean how fast a method converges for $h \to 0$, or with other words how accurate the solution is for

a given h. By using a taylor expansion for the exact solution after a single time step

$$y(t+h) = y(t) + hy'(t) + h^2/2y''(t) + O(h^3) \quad (11)$$

we find that for the numerical approximation Y_1 produced by an explicit Euler step

$$y(t_1) - Y_1 = O(h^2). \quad (12)$$

If we continue the method using the *numerical solution* Y_1 as a starting value for the next time step we lose[13] a power of h for the global error

$$y(t_n) - Y_n = O(h). \quad (13)$$

This means that the explicit Euler method converges linearly or has *order* 1. We will analyze the stability and efficiency of the method later.

As a next step we introduce methods of higher order. For this a centered difference estimation for $y'(t+h/2)$ (7) is used

$$\frac{y(t+h) - y(t)}{h} \approx y'(t+h/2) = f(t+h/2, y(t+h/2)). \quad (14)$$

resulting in the iteration scheme

$$Y_{n+1} = Y_n + hf(t+h/2, y(t_n + h/2)). \quad (15)$$

But how do we find $k_1 \approx y(t_n + h/2)$? For an estimation we use an explicit Euler step to get

$$k_1 = Y_n + \tfrac{h}{2} f(t, Y_n) \quad (16)$$
$$Y_{n+1} = Y_n + hf(t+h/2, k_1), \quad (17)$$

the so called *explicit midpoint* rule. The estimation by forward Euler, although not very accurate, is good enough, as the function evaluation is multiplied by the timestep to advance to the next approximation. So by a taylor expansion we find a local error of $O(h^3)$ leading to a global error of

$$Y_n - y(t_n) = O(h^2) \quad (18)$$

for the explicit midpoint rule.

Generalizing the idea of using function evaluations at s intermediate points $t + c_j h$ leads to Runge-Kutta methods. They are defined by a Runge-Kutta matrix (a_{ij}), weights b_i, abscissae c_j and the equations

$$k_i = Y_n + h \sum_{j=1}^{s} a_{ij} k'_j$$
$$\text{with } k'_i = f(t_n + c_i h, k_i) \text{ for } \quad i = 1, \ldots, s$$
$$Y_{n+1} = Y_n + h \sum_{i=1}^{s} b_i k'_i \quad (19)$$

The coefficient set can comfortably be specified as shown in table 1. If the matrix (a_{ij}) is strictly upper,

c_1	a_{11}	a_{12}	$\cdots$	a_{1s}
c_2	a_{12}	a_{22}	$\cdots$	a_{2s}
$\vdots$	$\vdots$		$\ddots$	$\vdots$
c_s	a_{s1}	a_{s2}	$\cdots$	a_{ss}
	b_1	b_2	$\cdots$	b_s

Table 1: General Runge–Kutta method

all inner stages k_i only depend on k_j with $j < i$ and thus can be computed one after the other.

The most famous scheme is the method by Runge and Kutta given in table 2(b). Table 2(a) shows the explicit midpoint rule interpreted as a Runge-Kutta method. The method by Runge and Kutta possesses order 4.

0	0	0
$\frac{1}{2}$	$\frac{1}{2}$	0
	0	1

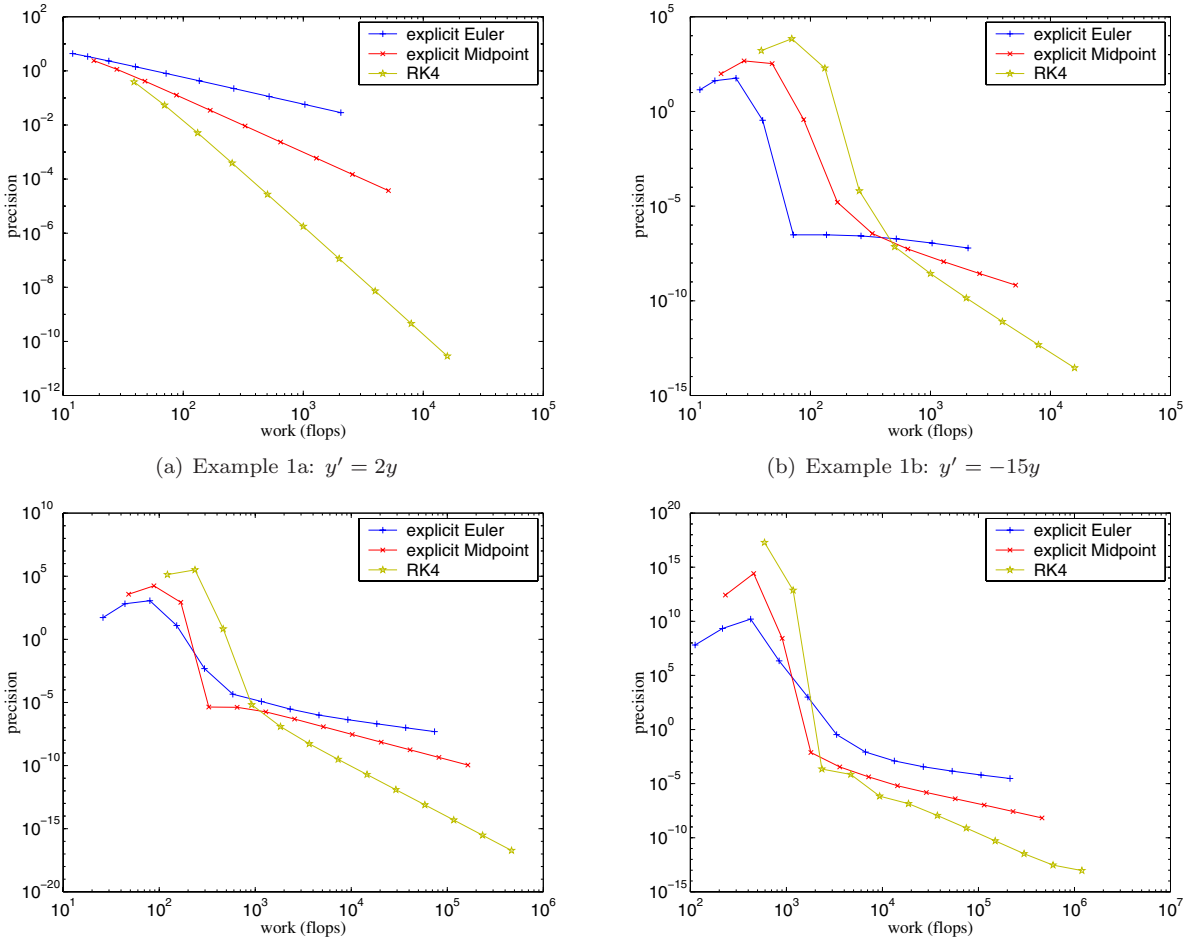
(a) Midpoint

0				
1/2	1/2			
1/2	0	1/2		
1	0	0	1	
	1/6	2/6	2/6	1/6

(b) RK4

Table 2: Explicit midpoint and "the" Runge-Kutta method.

By using algebraic relations for the coefficients, it is possible to construct explicit Runge-Kutta methods of arbitrary high order resulting in many inner stages with numerous evaluations. For most practical applications order 4 is sufficient.



(a) Example 1a: $y' = 2y$

(b) Example 1b: $y' = -15y$

(c) Example 2: $y'' = -5y' - 5y$

(d) Example 3

Figure 3: Work precision diagrams for the explicit Euler, explicit midpoint and RK4 methods.

Having constructed all these method we apply them to our examples.

The plots in figure 3 were produced by solving the examples using different timesteps and measuring the number of floating point operations needed for achieving the specified accuracy when compared with the (analytical) reference solution. In the work-precision diagram the y-axis shows the error $\|Y_{t_{end}} - y(t_{end})\|$ as a function of the required number of floating point operations. The first example with $\lambda = 2$ (figure 3(a)) shows exactly the expected behavior: when reducing the time step and thus investing more work, the numerical solutions converge towards the reference solution. Moreover the slope of the curves in the double logarithmic plot exactly matches the order of the method. In all other examples (figure 3(b)-3(d)) this behavior only shows up after an initial phase, where the solver produces completely wrong results. This is the point where stability comes into play. We will now analyze this by using the simplest example where it occurs, i.e. example 1 with $\lambda < 0$.

2.2 Stability

The equation for example 1 is called *Dahlquist's test equation*

$$y' = \lambda y, \qquad \lambda \in \mathbb{C}. \tag{20}$$

Its exact solution for an initial value $y(0) = y_0$ is given by

$$y(t) = e^{\lambda t} y_0. \tag{21}$$

This equation is a tool for understanding and evaluating the stability of integration methods. We have seen, that in the damped case characterized by $\mathrm{Re}\,\lambda < 0$ convergence is only achieved for very small time steps. In this case, since the exponent is negative, the analytical solution is bounded for $t \to \infty$. Therefore one expects from a meaningful numerical method to deliver a bounded solution. An integration scheme that yields a bounded solution is called *stable*.

If we apply the forward Euler method with a fixed step size h to (20), the n-th point of the numerical solution is given by:

$$Y_n = (1 + h\lambda)^n y_0 \tag{22}$$

It is bounded if and only if $|1 + h\lambda| < 1$, i.e. for $h\lambda$ in the unit ball around -1. A similar analysis can be carried out for the other methods and also results in restrictions of the admissible step size.

This analysis explains the sharp bend in figures 3(b)-3(d). Only when the step size drops below a certain limit dictated by λ, i.e. by $h < \lambda^{-1}$ in case of the forward Euler method, the numerical solutions can converge. If the damping is increased, i.e. $\mathrm{Re}\,\lambda \to -\infty$, then for the explicit Euler necessarily $h \to 0$ for the solution to be stable. This means the step size is artificially limited and it cannot be increased beyond the stability limit. This limits the flexibility of balancing work against accuracy.

2.3 The implicit Euler method

To construct a method that better suits our needs we go back to (7) and substitute the differential quotient by a *backward* difference quotient for $y(t + h)$

$$\frac{y(t+h) - y(t)}{h} \approx y'(t+h) = f(t+h, y(t+h)). \tag{23}$$

This results in the integration formula

$$Y_{n+1} = Y_n + hf(t + h, Y_{n+1}), \tag{24}$$

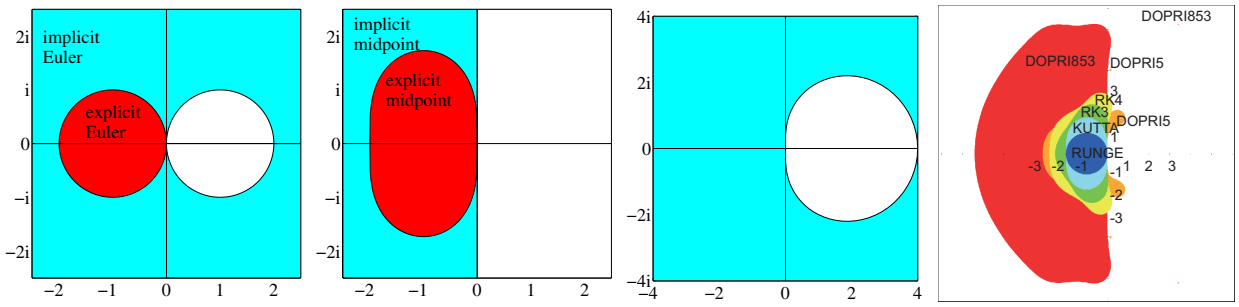
the so called backward or implicit Euler method. As its explicit variant this method can be shown to have order 1. Now the numerical solution only is given implicitly by the solution of the possibly nonlinear equation

$$Y_{n+1} - hf(t + h, Y_{n+1}) - Y_n = 0. \tag{25}$$

If we apply this method to the Dahlquist equation we get the recurrence formula

$$Y_n = (1 - h\lambda)^{-n} y_0. \tag{26}$$

The numerical solution Y_n remains bounded for $|(1 - h\lambda)^{-1}| < 1$. If we assume $\lambda < 0$, this holds for arbitrary $h > 0$. Thus there is no restriction on stepsize, the method is *unconditionally stable*. Figure 5 shows the work-precision diagrams for the implicit Euler method and our examples. We observe that we never loose stability and we especially do no miss the



(a) Euler methods (b) midpoint rules (c) BDF(2) (d) RK methods

Figure 4: Stability regions (shaded) of the methods.

solution by several orders of magnitude compared to the explicit methods. Of course the method looses accuracy when the time steps become large.

As a useful tool for visualizing the stability properties of a method we define the *stability region* $\mathcal{S}$ to be the set of parameters, for which the integration method yields a bounded solution:

$$\mathcal{S} := \{z := h\lambda \in \mathbb{C} : \text{ the numerical integration of equation (20) withstep size } h \text{ and parameter } \lambda \text{ is stable}\}. \quad (27)$$

Methods that contain the complete left half-plane in $\mathcal{S}$ are called *A-stable* or *unconditionally stable*. They are well suited for the stable integration of stiff equations. Obviously, the implicit Euler scheme is A–stable, whereas its explicit counterpart is not. The stability regions of all presented methods are shown in figure 4.

After reviewing the process that led us to the definition of the stability region, we can outline a more general idea that will allow us to determine the stability of more complex methods. The idea for analysing both Euler methods applied to (20) was to find a closed expression describing the *stability function* $\mathcal{R}$. This function maps the initial value y_0 to the value Y_1, performing a single step of the method

$$\mathcal{R} : y_0 \mapsto Y_1. \quad (28)$$

Thus $Y_n = R(h\lambda)^n y_0$. For the explicit Euler method we found in (22)

$$\mathcal{R}(z) = 1 + z, \quad (29)$$

for the implicit version in (26)

$$\mathcal{R}(z) = \frac{1}{1-z}. \quad (30)$$

The definition for the stability region now reads

$$\mathcal{S} = \{z \in \mathbb{C} : |\mathcal{R}(z)| < 1\}. \quad (31)$$

2.4 Methods of higher order

To find a higher order method, we go back to equation (15) and insert a linear interpolation term for $y(t + h/2)$. The resulting formula is taken as an implicit definition of $y(t + h)$. We get the *implicit midpoint rule*

$$Y_1 = Y_0 + hf\Big(t + h/2, \frac{Y_1 + Y_0}{2}\Big), \quad (32)$$

using a simplified notation for advancing one step, i.e. writing Y_0 and Y_1 instead of Y_n and Y_{n+1}.

Alternatively the midpoint rule can be derived as a *collocation method*[12] with $s = 1$ internal nodes, i.e. by constructing a polynomial interpolating the particle trajectories at a given, fixed set of s nodes[12]. This idea allows for the construction of implicit Runge-Kutta methods with arbitrary order. In contrast to explicit methods the matrix (a_{ij}) ceases to

be strictly lower triangular. These methods are computationally more expensive, so we just stick to the midpoint rule. Its stability function is given by

$$\mathcal{R} = \frac{1 + z/2}{1 - z/2}. \tag{33}$$

As $\mathcal{R} \leq 1$ for any $\operatorname{Re} z < 0$ the implicit midpoint rule is A-stable.

As another possible choice we now introduce *multistep methods*. They are computationally inexpensive because they have no inner stages and some of them are A-stable. A multistep method with k steps is of the general form

$$\sum_{j=0}^{k} \alpha_j Y_{k-j+1} = h \sum_{j=0}^{k} \beta_j f_{k-j+1}, \tag{34}$$

with $f_{n+j} := f(t_{n+j}, Y_{n+j})$. Here we also have 'history points' with negative indices. The coefficient α_k is required to be nonzero; for variable time step sizes the coefficients depend on the last stepsizes, which we have omitted here for the ease of demonstration. Important special cases are the class of *Adams methods* where $\alpha_0 = \cdots = \alpha_{k-2} = 0$:

$$Y_1 = Y_0 + h \sum_{j=0}^{k} \beta_j f_{k-j+1} \tag{35}$$

and the class of *BDF–methods* (**b**ackward **d**ifferentiation **f**ormulas) with $\beta_0 = \cdots = \beta_{k-1} = 0$:

$$\sum_{j=0}^{k} \alpha_j Y_{k-j+1} = h \beta_k f_1. \tag{36}$$

If the formula involves the right-hand side f_1 at the new approximation point Y_1 the method is said to be implicit. BDF-methods are always implicit. The coefficients can again be constructed by a collocation approach. BDF-methods exist up to order 6, higher order methods loose consistency for any choice of coefficients[12].
The stability regions of implicit and explicit Adams methods are bounded and located around the origin, thus they are not interesting for large time steps.
BDF-methods were the first to be developed to deal with stiff equations and possess an unbounded stability region covering a sector within the negative complex half-plane. Therefore they are among the most widely used methods today. For $k+1$ points, these methods possess order $k+1$ and only one nonlinear system has to be solved, whereas s coupled systems have to be solved for an s-stage implicit Runge-Kutta method.

The BDF-method for $k = 1$ is just the implicit Euler method, for k=2 the method is given as

$$Y_1 = \frac{4}{3} Y_0 - \frac{1}{3} Y_{-1} + \frac{2}{3} h f(t + h, Y_1) \tag{37}$$

The coefficients for higher order methods are given in table 3. The stability region of BDF(2) and the other implicit methods are shown in figure 4.

α_0	α_1	α_2	α_3	α_4	α_5	α_6	β
$\frac{3}{2}$	-2	$\frac{1}{2}$					1
$\frac{11}{6}$	-3	$\frac{3}{2}$	$-\frac{1}{3}$				1
$\frac{25}{12}$	-4	3	$-\frac{4}{3}$	$\frac{1}{4}$			1
$\frac{137}{60}$	-5	5	$-\frac{10}{3}$	$\frac{5}{4}$	$-\frac{1}{5}$		1
$\frac{147}{60}$	-6	$\frac{15}{2}$	$-\frac{20}{3}$	$\frac{15}{4}$	$-\frac{6}{5}$	$\frac{1}{6}$	1

Table 3: BDF Methods

2.5 The Verlet method

As a last method we will discuss a scheme commonly referred to as leapfrog or Stoermer-Verlet method. It is especially efficient if (5) is given as the second order system

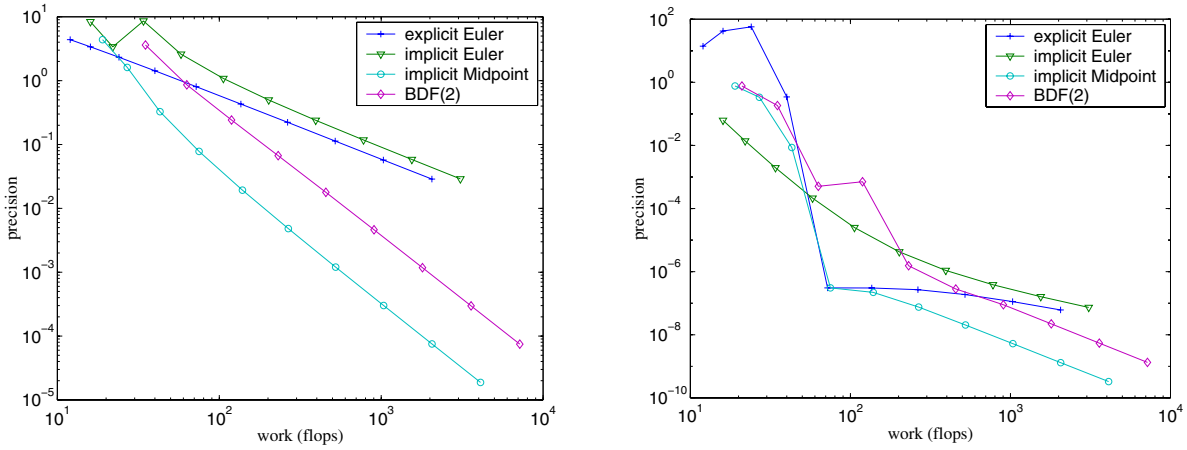
$$x''(t) = f_v\big(t, x(t)\big), \tag{38}$$

i.e. $f_v\big(t, x(t), x'(t)\big) = f_v\big(t, x(t)\big)$. It is not applicable to general first order systems of the form (7).

To derive it, we use centered differences at a staggered grid (figure 6) i.e. we now approximate v at $t + (2i+1)h/2$ and x at $t + ih$ by centered differences

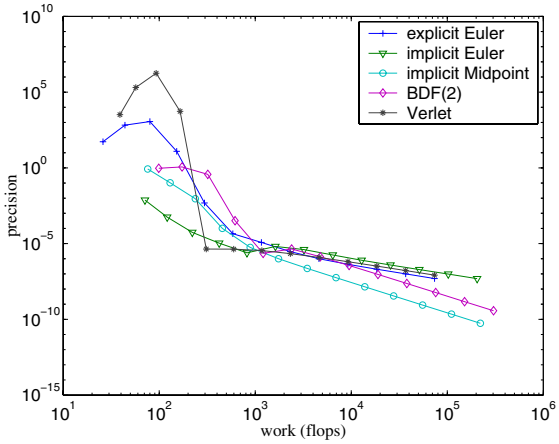
$$\frac{v_{n+1/2} - v_{n-1/2}}{h} = f(x_n) \tag{39}$$

$$\frac{x_{n+1} - x_n}{h} = v_{n+1/2} \tag{40}$$

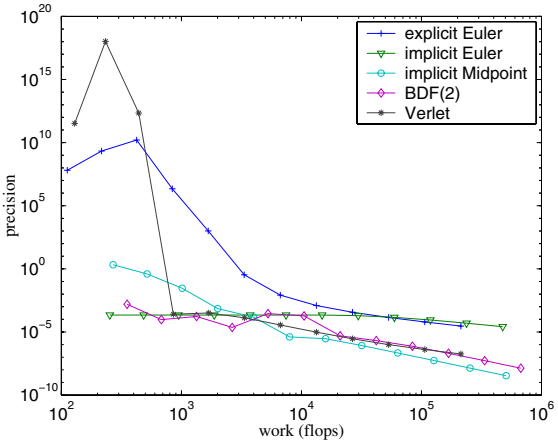


(a) Example 1a: $y' = 2y$

(b) Example 1b: $y' = -15y$



(c) Example 2: $y'' = -5y' - 5y$



(d) Example 3

Figure 5: Work precision diagrams for the implicit Euler, implicit midpoint, BDF(2) and Verlet (whenever possible) methods. The results for Euler are for comparison and the same as in figure 3.

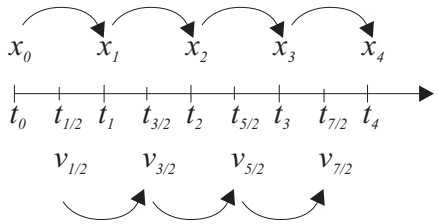


Figure 6: Staggered grids for the Verlet method.

thus

$$v_{n+1/2} = v_{n-1/2} + hf(x_n) \tag{41}$$
$$x_{n+1} = x_n + hv_{n+1/2}. \tag{42}$$

The method possesses order 2 as one can see by substituting (41) into (42) resulting in the second order centered difference

$$\frac{x_{n+1} - 2x_n + x_{n-1}}{h^2} = f(x_n). \tag{43}$$

From this equation an alternative formulation of the Verlet scheme as a multistep method can be derived

$$v_n - v_{n-1} = hf(x_n) \tag{44}$$
$$x_{n+1} - x_n = hv_n, \tag{45}$$

which omits the half steps and staggered grids from above. Now for second order equations which do not possess the form of (38) one may replace $f(x_n)$ by $f(x_n, v_{n-1})$ at the expense of some stability. Correctly the replacement had to be with $f(x_n, v_n)$ but this would result in a implicit method. Now we can apply the method to examples 2 and 3.

Figure 5 shows the work precision diagrams for the implicit methods. Even for large time steps these methods give approximations to the exact solution. After a somewhat uneven convergence phase all the implicit methods converge smoothly for decreasing time step to the exact solution with a slope given by their order.

Up to now we have omitted a discussion of the stability properties of the leapfrog method. From the examples it can be observed that the method cannot be unconditionally stable. Indeed some more delicate computations show that the method only delivers a bounded solution for arbitrary $h > 0$ for purely oscillatoric equations, i.e. for second order ordinary differential equations of the form (38) (with no damping term) and a contractive right hand side. Nevertheless the method remains well behaved in the presence of low damping. This explains its importance in molecular and celestial dynamics. In figure 7 we applied the Verlet method to the undamped wave equation $y'' = 5y$ and it is clearly one of the best choices over a wide range of accuracy requirements.

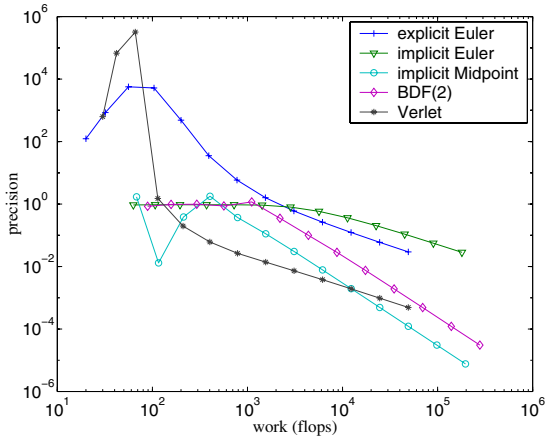


Figure 7: Work precision diagram for the explcit/implicit Euler, implicit midpoint, BDF(2) and Verlet methods for the wave equation $y'' = -5y$ without damping.

2.6 Methods used in clothing literature

Terzopoulos et. al.[24] used a finite element formulation and a simple implicit Euler scheme to solve the arising ODE. Later publications focused on explicit integration methods, e.g. Eberhardt et al.[6] preferred RK4 and the Burlisch-Stoer extrapolation method as suggested in Press et al.[20]. Volino[4] used an explicit midpoint rule.

Implicit methods again became popular with the work of Baraff and Witkin. They used a linearized implicit Euler method and achieved simulations about an order of magnitude faster than explicit

methods. Although nonlinear constraints are formulated in the model, they only use their linear approximation to obtain a linear system of equations. This way the system to be solved in each time step also becomes linear and can be solved efficiently by a *cg*-method. This method corresponds to the solution of a nonlinear system with only one Newton iteration. Because the nonlinear part is not integrated, with high stiffness one may encounter similar slow downs as observed by Volino[28] and Eberhardt[7].

Provot[21] proposed a simple model only incorporating linear springs, combining it with an explicit method. This model was used by Desbrun et. al. [5] who also use only a linearized implicit method. But instead of linearizing the whole system they split it in a linear and nonlinear part and use a precomputed inverse of A for solving the linear part of the equations. They don't aim at solving the equation completely, as they don't integrate the nonlinear term explicitly. Instead the angular momentum is corrected to account for the nonlinear part. With this algorithm one can neither change the stepsize h nor deal with an A depending on t.

Based on this work Kang et al.[17] did some further simplification to avoid solving the linear system. In order to update the solution vector in one step they divide by diagonal entry of the matrix of the linear system. Therfore they just do a single iteration of a Jacobi-like scheme for solving the linear equations. Again this may lead to artificial slowdowns.

Recently more advanced methods gain importance. Hauth et. al.[15] used BDF and the implicit midpoint rule, and Choi and Ko[3] also used BDF, both combining it with an iterative cg solver. In Hauth et. al. [16] there the complete BDF-2 algorithm, including variable step sizes, is presented in pseudocode, derived in the presented framework.

2.7 Selecting an efficient method

Which method is best for a certain application? This question is nearly impossible to answer a priori. The only choice is to try a set of methods and to evaluate which one performs best. Choosing the methods to try, though can be done based on theoretical considerations and observations of the problem at hand. A possible strategy is shown in figure 8.

The same statement holds for predicting the efficiency of a method. Generally implicit methods require more work per step. On the other hand one may be able to use time steps that are several magnitudes larger than the ones explicit methods would allow. Although accuracy will suffer, the integration won't be unstable (see figure 2(b) for example 3). If evaluations of the right hand side function are cheap, a step with RK(4) is faster than an implicit step with BDF(4). On the other hand if it is cheap to compute a good sparse approximation to the jacobian, it may be more efficient to solve the linear system with a few cg iterations than to perform 4 full function evaluations.

3 Solving nonlinear systems

All implicit methods require the solution of a nonlinear system. The implicit Euler method for example reduces our integration problem to the solution of the nonlinear system

$$Y_1 - hf(Y_1) - Y_0 = 0. \tag{46}$$

The other methods yield a system of similar form, namely

$$Y_1 - hf(\tfrac{1}{2}(Y_1 + Y_0)) - Y_0 = 0 \tag{47}$$

$$Y_1 - \tfrac{2}{3}f(Y_1) + \left(-\tfrac{4}{3}Y_0 + \tfrac{1}{3}Y_{-1}\right) = 0 \tag{48}$$

for the midpoint and BDF(2) rule, respectively. This is a nonlinear system of dimension $6N$. It must be solved with Newton's method to allow arbitrary step sizes independent of λ. Simpler methods for nonlinear systems would compensate the advantage of A-stability because the number of iterations would increase proportionally to the stiffness parameter $|\lambda|$. We now will work out an approach for implementing Newton's method efficiently.

3.1 Newton's method

For the nonlinear system $G(Y) = 0$ we compute a numerical solution by the following algorithm:

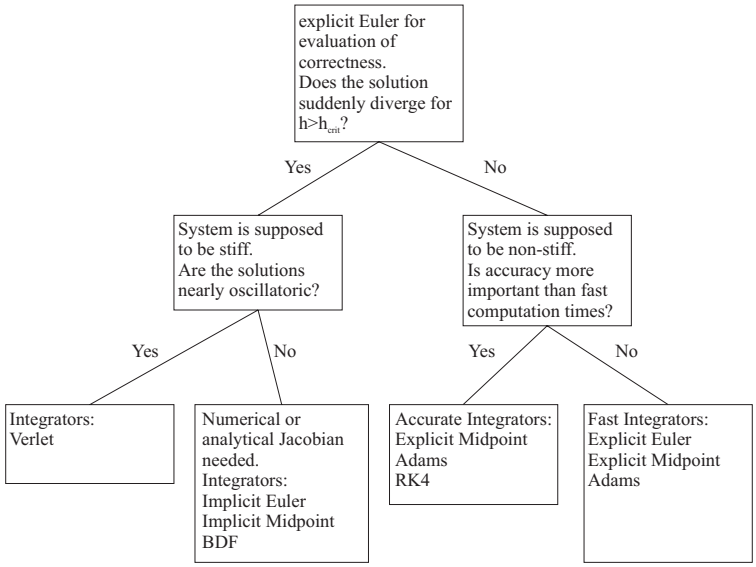


Figure 8: Selecting a method.

Algorithm 1: Newton's Method

(1) **for** $k = 1, 2, \ldots$ **until** convergence **do**
(2) Compute $G(Y^{(k)})$.
(3) Compute $J^{(k)} = \frac{\partial}{\partial Y} G(Y^{(k)})$.
(4) Solve $J^{(k)} s^{(k)} = -G(Y^{(k)})$.
(5) $Y^{(k+1)} := Y^{(k)} + s^{(k)}$
end

Applying Newton's method reduces the problem to the successive solution of linear systems. In a classical Newton method this is achieved by Gaussian elimination. This introduces a lot of non-zero elements into the factors. Although reordering techniques alleviate the effects, this approach may be too expensive for the present application.

The majority of authors[1, 27, 15, 3] use iterative methods to solve the linear system. We will also use the conjugate gradient method here to solve the linear systems in each Newton step. Unfortunately this changes the convergence behaviour of the outer Newton method, which is referred to as an inexact Newton method[22], given by algorithm 2.

Algorithm 2: Inexact Newton Method

(1) **for** $k = 1, 2, \ldots$ **until** convergence **do**
(2) Compute $G(Y^{(k)})$.
(3) Compute $J^{(k)} = \frac{\partial}{\partial y}(Y^{(k)})$.
(4) Find $s^{(k)}$ with $J^{(k)} s^{(k)} = -G(Y^{(k)})) + r^{(k)}$, such that $\|r^{(k)}\| \leq \eta_k \|G(Y^{(k)})\|$.
(5) $Y^{(k+1)} := Y^{(k)} + s^{(k)}$
end

3.2 Residual control

The error of the iterative solution of the linear system is formulated in terms of the residual, which is easily computationally accessible, whereas the actual error cannot be computed. The tolerance of the linear iteration is decreased proportionally to the monotonically decreasing residual of the nonlinear iteration. An analysis of this method[22] shows that it converges under rather weak additional assumptions. If the classical Newton method converges and the scalar tolerances $\eta^{(k)}$ are uniformly bounded by an $\eta < 1$, the inexact method converges. In literature the $\eta^{(k)}$ are referred to as *forcing terms*. Note that this additional assumption is also necessary: For $\eta = 1$,

$s^{(k)} = 0$ would be admissible and the iteration would stagnate.

The inexact method then at least converges linearly, whereas Newton converges superlinearly. By choosing the η_k to converge to zero sufficiently fast[22], the convergence of the inexact Newton method can be forced to have an order > 1. In a neighbourhood of the solution the convergence usually speeds up. By extrapolating the solution of the previous time step we obtain a good initial value for the new solution and the method converges quickly using the constant bound $\eta = 0.02$ without imposing a too strict tolerance on the linear solver.

3.3 Inexact simplified Newton methods

The efficiency of the Newton method can be further improved by another approximation. In the simplified version of Newton's method the Jacobian $J^{(k)}$ is approximated by $J^{(0)}$. Such a scheme can be rewritten in the form of an inexact Newton method, if the linear system is written as follows and J is chosen as approximation to $J^{(k)}$

$$\begin{aligned} Js^{(k)} &= -G(Y^{(k)}) + (J - J^{(k)})s^{(k)} + r^{(k)} \\ &=: -G(Y^{(k)}) + \tilde{r}^{(k)} \end{aligned} \tag{49}$$

The residual $r^{(k)}$ is replaced by the larger $\tilde{r}^{(k)}$, which can be bounded if $J \approx J^{(k)}$. By choosing $\tilde{\eta}^{(k)}$ appropriately, the method still converges. In fact, we trade some accuracy approximating $J^{(k)}$ against accuracy in solving the linear system and up to a certain limit the method still behaves as before.

This degree of freedom can be further exploited be even not computing $J^{(0)}$ but a sparser approximation of it. Hauth et. al.[15] exploit the idea previously used by Desbrun et al.[5] and approximate the Jacobian by the linear expression

$$F_{\text{lin}}(x, v) = \sum_{j|(i,j)\in E} \left[\frac{k_{ij}}{l_{ij}^2}(x_i - x_j) \; + \frac{d_{ij}}{l_{ij}^2}(v_i - v_j) \right]. \tag{50}$$

This choice of the Jacobian has two major advantages over the full Jacobian. First, J is inexpensive to compute and only changes when either the material constants or the step size changes. Second, we reduce the entries in the Jacobian to approximately a third of the entries in the sparsity pattern of the full Jacobian. Hence an iteration of the linear solver only requires a third of the original time. Obviously this is a major speed-up for the solver. The resulting algorithm is surprisingly simple.

Algorithm 3: Inexact Simplified Newton's Method

(1) Compute $J \approx \frac{\partial}{\partial Y} G(Y^{(0)})$.
(2) **for** $k = 1, 2, \ldots$ **until** convergence **do**
(3) Compute $G(Y^{(k)})$.
(4) Find $s^{(k)}$ with $Js^{(k)} = -G(Y^{(k)})) + r^{(k)}$, such that $\|r^{(k)}\| \leq \tilde{\eta}_k \|G(Y^{(k)})\|$.
(5) Update $Y^{(k+1)} := Y^{(k)} + s^{(k)}$
end

3.4 Adaptive time stepping

Newton's method can also be used to control the step size of the ODE solver. If the convergence of Newton's method is poor, the time step h is reduced such that the solution of the previous time step is a better start value for the current time step and achieves a faster convergence. On the other hand, the number of Newton iterations necessary is a criteria for the behaviour of the integrator and has already been used for an order selection algorithm[14] of the Radau integrator. In our implementation the number of Newton iterations decides whether to increase or decrease the time step h.

4 Linear Systems

In a last section we briefly present the last part of the puzzle, the solution of the remaining nonlinear system. We restrict the discussion to the most important iterative method used, the method of conjugate gradients. A more profound discussion can be found in the standard texts of Trefthen[25], Greenbaum[11] or Saad[23]. It uses only matrix-vector multiplications to solve a symmetric system of the form

$$Ax = b, \tag{51}$$

with $A \in \mathbb{R}^{N\times N}, x, b \in \mathbb{R}^N$. Thus the expenses for a matrix inversion is reduced from $O(N^3)$ to a hopefully small number of multiplications with a complexity of $O(N^2)$. In the present application, A is the Jacobian arising in Newton's method, e.g. $A = I - h\frac{\partial}{\partial Y} f(Y_0)$ for the implicit Euler method.

4.1 Preconditioning

What lets Krylov methods like cg fly, is the use of a cheaply invertible preconditioner $M \approx A$ and the solution of

$$M^{-1}Ax = M^{-1}b, \tag{52}$$

which is equivalent to (51). The most prominent preconditioners in computer science literature are

- The diagonal preconditioner

  $$M := \mathrm{diag}(A). \tag{53}$$

  This is the choice considered in the systems of Baraff[1] and Volino[27], also called diagonal scaling. This preconditioner is easily available and inexpensive to apply, as each application is just a division of each vector-component by the corresponding diagonal entry, resulting in N flops per application.

- The incomplete Cholesky factorisation (ICC) is computed by carrying out an approximate Cholesky factorisation of A, i.e. formally factorising A and dropping all intermediate values not fitting the sparsity pattern. It is more expensive to compute and to apply, resulting in almost $2\,\mathrm{nnz}(A)$ flops per application, where nnz denotes the set entries of A.

- The successive-symmetric over-relaxation preconditioner (SSOR) is another iterative solving scheme for linear equations. The matrix formulation of one SSOR-step is given by the multiplication by

  $$M := (D - L)D(D - L^T), \tag{54}$$

  where $A = D - L - L^T$ and $D = \mathrm{diag}(A)$, L strictly lower tridiagonal. Note that the inversion can be realised by inverting two triangular systems, as in practice one product with D is precomputed. Thus the SSOR-preconditioner is inexpensive to compute and the cost of an application is about the same as for the ICC-preconditioner.

- The block diagonal preconditioner, a generalisation of diagonal scaling where typically 3×3 blocks of A are inverted.

4.2 Enforcing constraints for collision response

The incorporation of collisions distinguishes animation from classical problems in numerical analysis. The effects cannot be modelled a priori in the differential equation, since the collision reaction depends on the collisions that are detected in each time step. Therefore, the ODE solver has to incorporate a collision response according to the current collisions. This requires the ODE to be modified during run time in each time step.

A first and useful technique is to use constraints, i.e. to constrain the motion of a particle p_i during a time step in the direction c_i, e.g. to allow no movement in the (negative) normal direction of a colliding surface.

Baraff and Witkin [1] presented a very efficient method to enforce constraints inside a *cg*-method. The cg method, like Newton's method, is an update method. It starts with an initial guess and adds scalar multiples of a search direction. If we insure, that the initial guess obeys the constraints and each update is orthogonal to the constraint directions, then the final solution will also fulfill the constraints. Therefore in each iteration of the *cg*-method the new direction is filtered such that the solution does not alter in a constrained direction. We will give some theoretical background to illuminate this method.

Constraining positions and particles is equivalent to adding algebraic equations to the system. Therefore, a differential algebraic system has to be solved with different constraints (algebraic equations) in each step. Collision forces are given implicitly, i.e. the predefined positions and velocities result in constraint forces, which can be computed after each step.

The constraints can be implemented by multiplying the linear system matrix with the rank-deficient projector matrix $P := I - \sum_i d_i d_i^T$. The vectors $d_i \in \mathbb{R}^{3N}$ are constructed by inserting the constrained direction $c_i \in \mathbb{R}^3$ to constrain particle j at the position $3j$. Thus we seek an admissible solution of the system

$$PAPx = Pb. \tag{55}$$

Here the multiplication on the right restricts the forces to admissible forces, the left multiplication with P filters the velocities, the multiplication with A is the linearized transformation to accelerations, which are again filtered by P. This system ceases to be positive definite, in fact it is nonnegative definite, but of course not of full rank. A unique solution exists if the system is expanded by the equation

$$(I - P)x = 0, \tag{56}$$

which states that the solution is admissible, i.e. has no components outside the nullspace $\ker(P)$. Effectively, the equality constrained least square problem

$$\|PAPx - Pb\| \to \min, \quad \text{where } (I - P)x = 0 \tag{57}$$

is solved. Since $Pb \in \text{range}(PAP)$ the minimum is 0 and the *cg* method applied to PAP is able to find a solution of this singular linear system[10].

Algorithm 4: Filtered cg

while $\frac{\|r\|}{\|b\|} \leq \epsilon$
  $i = i + 1$
  solve $Mz = r$
  if $i = 0$
    $\rho = \langle r, z \rangle$
  else
    $\beta = \frac{\rho}{\rho_1}$
    $p = z + \beta p$
  $p = Pp$
  $q = Ap$
  $q = Pq$
  $\alpha = \frac{\rho}{\langle p, q \rangle}$
  $v = v + \alpha p$
  $r = r - \alpha q$
  $\rho_1 = \rho$
end

References

[1] David Baraff and Andrew Witkin. Large steps in cloth simulation. In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, pages 43–54, July 1998. 12, 14

[2] David E. Breen, Donald H. House, and Michael J. Wozny. Predicting the drape of woven cloth using interacting particles. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94*, pages 365–372. ACM SIGGRAPH, ACM Press, July 1994. 1

[3] Kwang-Jin Choi and Hyeong-Seok Ko. Stable but responsive cloth. In *SIGGRAPH 2002 Conference Proceedings*, pages 604–611. ACM Press/ACM SIGGRAPH, 2002. 11, 12

[4] Martin Courshesnes, Pascal Volino, and Nadia Magnenat Thalmann. Versatile and efficient techniques for simulating cloth and other deformable objects. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 137–144. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995. 10

[5] Mathieu Desbrun, Peter Schröder, and Alan Barr. Interactive animation of structured deformable objects. In *Graphics Interface*, pages 1–8, June 1999. 11, 13

[6] B. Eberhardt, A. Weber, and W. Strasser. A fast, flexible, particle-system model for cloth draping. *IEEE Computer Graphics and Applications*, 16(5):52–60, September 1996. 1, 10

[7] Berndhard Eberhardt, Olaf Etzmuß, and Michael Hauth. Implicit-explicit schemes for fast animation with particle systems. In *Eurgraphics Computer Animation and Simulation Workshop 2000*, pages 137–151, 2000. 11

[8] Jeffrey W. Eischen, Shigan Deng, and Timothy G. Clapp. Finite-element modeling and

control of flexible fabric parts. *IEEE Computer Graphics and Applications*, 16(5):71–80, September 1996. ISSN 0272-1716. 2

[9] O. Etzmuss, J. Gross, and W. Straßer. Deriving a particle system from continuum mechanics. *Transactions on Visualization and Computer Graphics*, 2001. Accepted for publication. 1

[10] Roland W. Freund and Marlis Hochbruck. On the use of two QMR algorithms for solving singular systems and applications in Markov chain modeling. *Numer. Linear Algebra Appl.*, 1(4):403–420, 1994. 15

[11] A. Greenbaum. *Iterative Methods for Solving Linear Systems*, volume 17 of *Frontiers in Applied Mathematics*. SIAM, 1997. 13

[12] E. Hairer and G. Wanner. Solving Ordinary Differential Equations II. Springer-Verlag, Berlin, 1996. 7, 8

[13] Ernst Hairer, Syvert P. Nørsett, and Gerhard Wanner. *Solving ordinary differential equations. I: Nonstiff problems. 2. rev. ed.* Springer-Verlag. , Berlin, 1993. 4

[14] Ernst Hairer and Gerhard Wanner. Stiff differential equations solved by Radau methods. *J. Comput. Appl. Math.*, 111(1-2):93–111, 1999. 13

[15] M. Hauth and O. Etzmuß. A High Performance Solver for the Animation of Deformable Objects using Advanced Numerical Methods. In *Proc. Eurographics*, pages 137–151, Manchester, UK, 2001. 11, 12, 13

[16] M. Hauth, O. Etzmuss, and W. Strasser. Analysis of numerical methods for the simulation of deformable models. *The Visual Computer*, 2002. Accepted for publication. 11

[17] Young-Min Kang, Jjeong-Hyeon Choi, Hwan-Gue Cho, Do-Hoon Lee, and Chan-Jong Park. Real-time animation technique for flexible and thin objects. In *WSCG*, pages 322–329, February 2000. 11

[18] M. Meyer, M. Desbrun, P. Schröder, and A. H. Barr. Discrete differential-geometry operators for triangulated 2-manifolds. Berlin, Germany, 2002. 1

[19] K. Polthier. Computational aspects of discrete minimal surfaces. In J. Hass, D. Hoffman, A. Jaffe, H. Rosenberg, R. Schoen, and M. Wolf, editors, *Proc. of the Clay Summer School on Global Theory of Minimal Surfaces*, 2002. 1

[20] William H. Press, Saul A. Teukolski, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, first edition, 1988. 10

[21] Xavier Provot. Deformation constraints in a mass-spring model to describe rigid cloth behavior. In Wayne A. Davis and Przemyslaw Prusinkiewicz, editors, *Graphics Interface '95*, pages 147–154, May 1995. 1, 11

[22] W. C. Rheinboldt. *Methods for Solving Systems of Nonlinear Equations*, volume 70 of *CBMS-NSF regional conference series in applied mathematics*. SIAM, second edition, 1998. 12, 13

[23] Y. Saad. *Iterative Methods for Sparse Linear Systems*. PWS Publishing., Boston, 1996. 13

[24] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. volume 21, pages 205–214, July 1987. 2, 10

[25] Lloyd N. Trefethen and David Bau. *Numerical linear algebra.* SIAM, 2000. 13

[26] T. Vassilev, B. Spanlang, and Y. Chrysanthou. Fast cloth animation on walking avatars. In A. Chalmers and T.-M. Rhyne, editors, *EG 2001 Proceedings*, volume 20(3) of *Computer Graphics Forum*, pages 260–267. Blackwell Publishing, 2001. 1

[27] P. Volino and N. Magnenat-Thalmann. Implementing fast cloth simulation with collision response. In *Computer Graphics International Proceedings*, pages 257–268, 2000. 12, 14

[28] P. Volino and N. Magnenat-Thalmann. Comparing Efficiency of Integration Methods for Cloth Animation. In *Computer Graphics International Proceedings*, pages 265–274, 2001. 11