Applying Constrained Optimization to Computer Graphics

Jeff Goldsmith and Alan H. Barr

October 28, 1998

Abstract

For many years, objects in computer-generated animation moved only as an animator moved them. In the last few years, physically-based modeling techniques have demonstrated that more attractive and realistic seeming motion can be created by subjecting objects to forces and constraints and making the objects move as they would in a physical environment. Sometimes the environment under- or over-constrained. In these cases, numerical optimization is used to find the best movement path the objects, where "best" is evaluated according to user-specified criteria. This article discusses some of the previous results using this technique; identifies some future uses, including some that are likely to become commonplace in a few years; and describes numerical methods to solve these problems.

Introduction

In creating computer animation, a great deal of the animator's time is spent getting objects to move as he or she wants. Usually, the animator has a good idea of where the objects should be at a few different times, but creating motion paths for them between these times requires a fair amount of tinkering. Most currently available systems use splines for this purpose. A spline is (usually) a piecewise cubic polynomial that interpolates a set of given points. The "natural" cubic spline is the set of cubics that minimizes the magnitude of the second derivative of any interpolating curve given zeroth, first, and second derivative continuity. In a sense, it is the curve that would be created by forcing a flexible but stiff rod through the required points. Experience shows that animators are often dissatisfied with the results achieved by spline paths; to work around this problem, they add many more specified positions to force the spline to represent the motion they had in mind. In some cases, they need to select a position for each frame of video. Throughout this process, the animator knew what he wanted; the tinkering was needed to force the computer to do it. A more efficient computer graphics system ought to have a set of choices that might correspond to a subset of a human animator's vocabulary, so that some of the time he or she can solve this problem immediately. Still better would be a system that can build more complicated motions out of chosen ideas or constraints, while still meeting some original intent.

An important tool currently being used to solve this and related problems is numerical constrained optimization. A good problem for constrained optimization is one that has a single measure of the quality of the answer plus some requirements upon the solution that must not be violated. The quality measure is called the objective function; the requirements are called constraints. A constrained optimization solver minimizes (or maximizes) the objective function subject to meeting the constraints. The objective function is usually a piece of software, but we are using it to represent ideas about motion that the animator had in mind when envisioning the scene, for example, making the motion as smooth as possible. The constraints translate more straightforwardly into animation requirements; they are conditions that must be met for

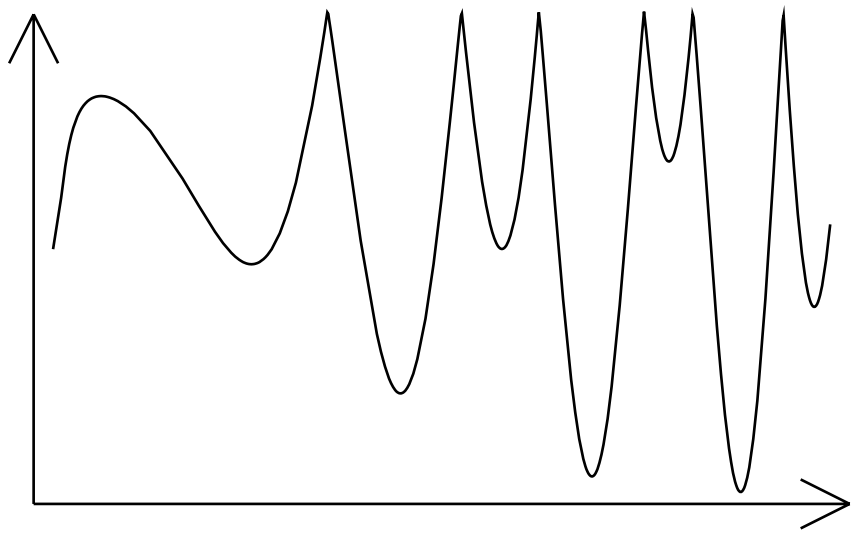

Figure 1: An imagined energy landscape. The global minimum is near the right; many local minima are present.

an answer to be acceptable, for example, for objects not to interpenetrate. Identifying a good set of effects and constraints such that a nontechnical animator can use this tool effectively is what is currently standing between constrained optimization and commercial software packages.

Constrained optimization turns out to be a very flexible tool. Many surfaces in computer graphics are parametric surfaces. Sometimes spline curves are inadequate or inconvenient, for creating free-form parametric surfaces. Constrained optimization may be the solution if the surface wanted meets a definition that requires minimizing some aspect of it. If this is desired to be the smoothest surface meeting certain criteria, those criteria can (hopefully) be described by constraints, and the smoothness of the surface can be maximized by minimizing the square of the magnitude of the second derivatives of the surface or other curvature measures. Many other problems in computer graphics require finding a good solution to an underconstrained or overconstrained problem; constrained optimization is ideal for this purpose.

Constrained Optimization

Optimization problems, in general, require finding the best solution to some underconstrained problem. Usually, the problem contains a single function of a set of free variables that returns one real number for any position in its domain. This function is called the objective function. The solution is defined as either the minimum or maximum value possible of the objective function, depending on the problem. (Minimum and maximum are equivalent, of course, to within sign of the objective function.) Constraints are requirements on the values of the variables in the objective function, and reduce the size of the domain.

The value of the objective function graphed as a function of the free variables is loosely called the "energy landscape" (See Fig. 1.) because an optimization solver tries to find local minima (or maxima) of the objective function by moving around locally in the function's domain. This looks a little like a ball rolling around on a hilly landscape; it finally comes to rest at the bottom of a valley.

One of the simplest forms of constrained optimization, in terms of formulation of the problem, is linear programming. (See Fig. 2.) Linear programming is just constrained optimization

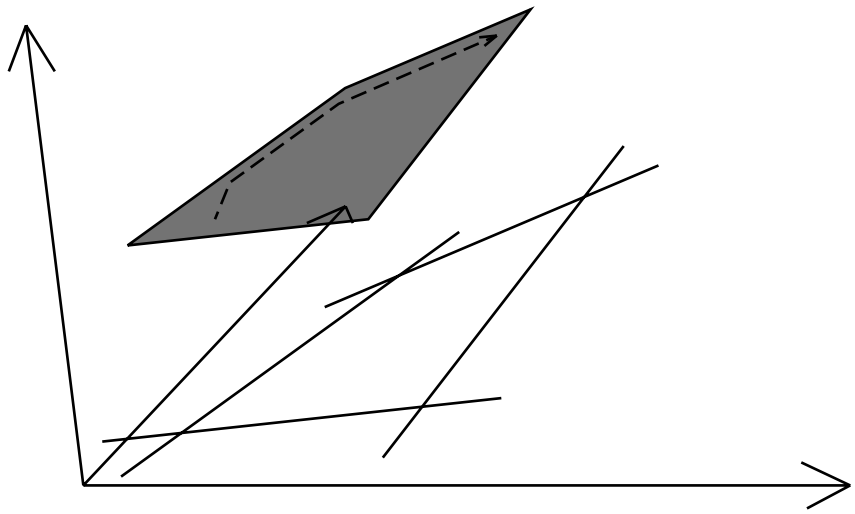

Figure 2: Linear programming. The dashed line is a solver's converging to the solution.

with a linear objective function and linear constraints. Most computer graphics problems are not linear, but linear programming is easier to visualize than general constrained optimization. Because all the constraints are linear in linear programming problems, the space of interest is limited to some convex region of the objective function's domain. The objective function can be imagined as a hyperplane over the space of its variables with some convex area defined on it as the area of possible solutions.

If the objective function is linear, then it turns out that the optimal solution must occur at a corner point of the domain. A commonly used solution method for problems of this sort is the "simplex method," which just climbs uphill (or downhill) around the perimeter of the domain until it cannot go higher.

If the objective function is nonlinear, then the problem is not a linear programming one. For simply defined objective functions, about which much is known, further mathematical programming techniques can help solve or at least prove some information about the solution. In the general case, for a nonlinear objective function about which not much is known *a priori,* searching methods must be used. These methods compute gradients of the objective function and use that information to try to move downhill (or uphill) in the solution. A globally optimal solution, therefore, is not always guaranteed, but a locally optimal one will be found. Boundedness constraints upon the variables and a good initial search point can help ensure that the local maximum found is the global maximum.

Problems with nonlinear constraints require solution validity checking and are thus very time consuming. The Jacobian of the constraint matrix is computed to create a pseudoforce to push the solver away from bad areas of the domain, but the constraints still have to be checked, and since convexity of the domain is not assured, solvers must just try to avoid out-of-bounds areas, rather than follow edges. Additional linear constraints including bounding of variables, are usually included in very nonlinear problems so that the search space is at least bounded.

A few numerical techniques roughly follow the simple non-linear optimization solution paradigm. The simplest method, gradient descent (or ascent,) just follows the gradient of the objective function until it hits a local minimum or the edge of the space and then stops. (See Fig. 3.) This is similar to Euler's method for solution to differential equations and is not often a satisfactory solution method for complicated objective functions. (By the way, it

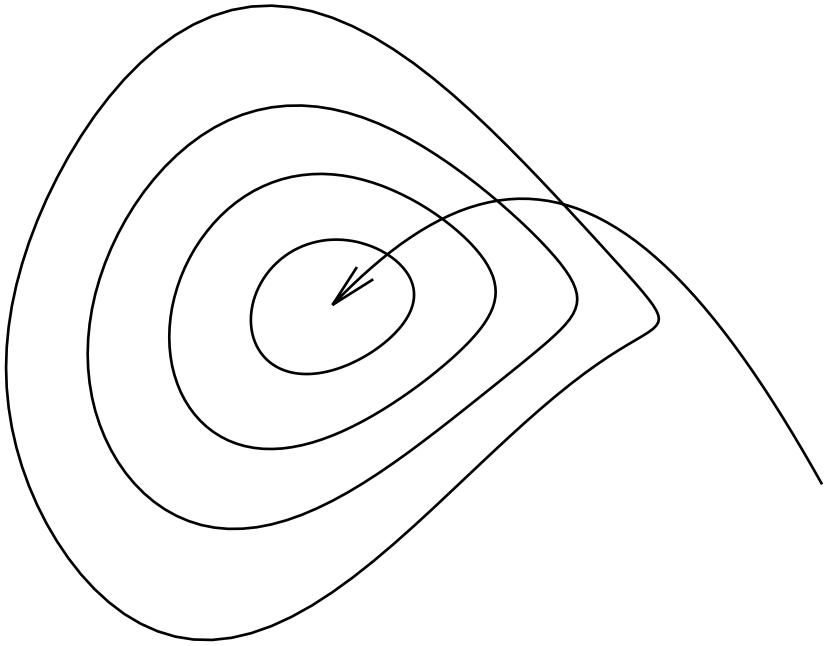

Figure 3: Gradient descent. The solid line is a solver's converging to the solution.

is attributed to Cauchy[1].) Conjugate gradient methods use more complicated functions of the objective function's gradient to search and are therefore less susceptible to getting stuck in undesired local minima.

Some more recent methods for solving optimization problems try to estimate or compute second derivatives of the objective functions. With this information, local areas of the objective can be approximated quadratically for far better convergence properties. In large-dimensioned problems, this requires saving more information about every variable, but usually if the objective function is more than trivially nonlinear, it seems to be worth it.

Several commercially available packages implement fairly modern optimization solvers. We use NAG [2] E04UCF, an implementation of sequential quadratic programming, for dense problems, that is, very nonlinear problems, and MINOS [3], a projected Lagrangian method for sparse problems. Each is commercially available and fairly inexpensive.

For very non-linear optimization problems, the absolute best solution is not always guaranteed unless we start close to it. Fortunately, for computer graphics applications, we often do not need the best solution, just one that meets our criteria. Since local maxima are often very close to the best solution, this drawback of constrained optimization is not usually a problem.

Applications

A fair number of applications of constrained optimization have already been found within the realm of computer animation, typically either starting or finishing problems. Finding an improved set of initial conditions to a problem is a good use of optimization; also, polishing approximate solutions or rough solutions generated either by hand or by computational techniques works well with these methods. Finding initial conditions, having objects assemble themselves, smoothing or finishing motion paths, interpolating orientations, animating flexible bodies, and finding boundaries have all been performed using this technique.

A typical problem of model construction is to find exact coordinates for objects fitting together. This normally requires many hours of tedious manual tinkering. It would seem as if a computer were more suited to the job, and it is. Barzel and Barr [4] showed an example of a self-assembling model that was created by specifying only object sizes and shapes and interobject connectivity. Witken et al. [5] showed an example of a self-assembling structure in which even some of the shape parameters of the objects were unknown before assembly.

Some surfaces that might be very difficult to express parametrically can be created by constrained optimization. Imagine a sphere surrounding a complicated object. If the sphere is allowed to contract so that it does not penetrate the interior object, yet is required to maintain maximal smoothness while diminishing its surface's distance from its center, a shrink-wrap [6] effect can be generated. The constraints are interpenetration constraints, and the objective function is to minimize either surface area or radius up to some given minimum, while maintaining smoothness.

Another form of surface that can be generated by optimization is the class of minimal surfaces. These are surfaces that have zero mean curvature everywhere, and are the minimal area surfaces that interpolate a given boundary. Soap bubbles formed by dipping wires of different shapes into soapy water are minimal surfaces. Constraining the surfaces to maintain contact with their frames and minimizing surface area is a way to generate these, often beautiful, surfaces using constrained optimization.

Interpolating orientations has long been known to be less simple than is immediately apparent. Shoemake [7] first suggested for computer graphics that orientations be represented by quaternions and interpolated in unit quaternion space, perhaps via Bezier curves. Barr et al. [8] actually solved the spline equation in that space, but found that numerical solution was necessary. In order to maintain that the quaternions maintain unit length, a constrained solver was used.

In order to simulate flexible bodies, one method is to use finite element meshes [9] connected by damped "rubber bands." Adding constraints is done by adding an augmented Lagrangian term to the differential equations. Solving these equations becomes an optimization problem, for which constrained optimization solvers are ideal.

Many numerical solutions or manually entered data sets retain noise from the data creation, either numerical error or manual precision limitations. This data can be filtered to remove the high-frequency noise; however, if the data is intended to be smooth, that is, to have small second derivative throughout, then inserting the data into a constrained optimizer with objective function set to minimize both the deviation from the initial data and to minimize the sum of the squares of the second derivative will produce smoothed data that deviates very little from the initial data set.

Applications to Television

Most of the applications described above are useful for creating computer animation, which is frequently shown on commercial television. Still, many of these effects have not yet reached many production facilities due to the complexity of specifying constraints for these problems.

Objects that are supposed to behave in a seemingly physical manner, for example an object sliding down a curved ramp such as a playground slide, are desirable when attempting psuedo-realism in computer animation. These motions are most easily computed using constraints and optimization techniques. Alternatively, in surrealistic settings, unreal laws of physics can be created and simulated.

Moving cameras in three-dimensional space is a difficult modeling problem. An animator-cinematographer has to be careful to avoid huge swings, combined pan-tilts, and backwards motion, for example, in order to prevent airsickness in viewers. Camera tilt angles are orientations and can be interpolated using the quaternion splining scheme while being constrained to single-axis motion whenever possible. A camera's motion can also constrained to avoid other annoyances such as flipping upside-down. This formulation of camera angles avoids some of the other problems real cameras might have, such as gimbal lock.

Smooth motion, surfaces, and painted images are usually made smoother via digital filtering. For NTSC display, removal of high-frequency components in any of the above is critical to avoid aliasing artifacts, either spatial or temporal. Image filtering has been around for awhile; no one solution is best for the myriad filtering problems. A technique that will find the smoothest result given express stated goals may be more intuitive and may adapt better to varied needs, so optimization might well be a good choice for these applications.

The applications of a tool as general as constrained optimization are, of course, endless. Most of the uses described above have been used successfully in at least example animations. As these ideas disseminate through the television community, we shall see many more motions, surfaces, and pictures created by constrained optimization.

References

[1] W. Press et al., *Numerical Recipes,* p. 302, Cambridge University Press, 1986.

[2] Numerical Applications Group, *The NAG Fortran Library Manual, Mark 14,* 1990.

[3] B. Murtagh, and M. Saunders, *A Projected Lagrangian Algorithm and its Implementation for Sparse Nonlinear Constraints,* Mathematical Programming Study, No. 16, North-Holland Publishing Co., 1982.

[4] R. Barzel, and A. Barr, *A Modeling System Based on Dynamic Constraints,* Computer Graphics, Vol. 22, No. 4, Aug. 1988, pp. 179-188.

[5] A. Witkin, K. Fleischer, and A. Barr, *Energy Constraints On Parameterized Models,* Computer Graphics, Vol. 21, No. 4, July 1987, pp. 225-232.

[6] D. Terzopolous et al., *Elastically Deformable Models,* Computer Graphics, Vol. 21, No.4, July 1987, pp. 205-214.

[7] K. Shoemake, *Animating Rotations with Quaternion Curves,* Computer Graphics, Vol. 19, No. 3, July 1985, pp. 245-254.

[8] A. Barr et al., *Smooth Interpolation of Orientations with Angular Velocity Constraints using Quaternions,* Computer Graphics, Vol. 26, No. 2, July 1992, pp. 313-320.

[9] J. Platt, and A. Barr, *Constraint Methods for Flexible Bodies,* Computer Graphics, Vol. 22, No. 4, Aug. 1988, pp. 279-288.