Choreographing Goal-Oriented Motion Using Cost Functions

DAVID E. BRENN

ABSTRACT

This paper describes a technique employing cost functions to produce complex motions. Cost functions can be used to define goal-oriented motions and actions. A cost function can be defined whose variables are the animated parameters of a scene. The parameters are modified in such a way to minimize the cost function. The minimum cost configuration can be viewed as a "key goal" configuration. The values of the parameters are stored at certain intervals during the minimization process. This produces a path through the parameter space of the model being animated. By incrementally moving along the parameter space curve and updating the model defined by the parameters, an animation of the model performing a goal-oriented action may be produced.

Keywords: computer animation, cost function minimization, object-oriented computer graphics, goal-oriented choreography

1. INTRODUCTION

The days when a slick piece of computer animation only consists of a few flying logos with glass spheres are behind us. The entertainment and computer graphics industries expect much more complexity and therefore realism in the computer-generated films of today. As the complexity of animations grows to meet this demand, the limits of traditional computer animation techniques become apparent. A majority of computer animation is still produced using keyframing. In this approach an animator defines all the modeling parameters of a scene for a sequence of selected moments, or "key frames", in the animated sequence. After this tedious process is completed for a sequence, the modeling parameters are interpolated between the keyframes to produce a complete animation [1].

This approach is acceptable for animations with a low number of modeling parameters and rather simple geometric structures, and is especially well suited for non-hierarchical structures which follow a specific path. Once the number of parameters increases or the geometric modeling hierarchy becomes complex, keyframing becomes tedious to use at best, and impossible to use at worst. This implies that there are whole classes of actions and motions which cannot be satisfactorily produced using keyframing. Modeling structures like people, animals, or chains is extremely difficult with keyframing techniques. It has been done, but with only limited success. The motions of these keyframed structures can be jerky, uneven and "unnatural". Modeling flexible and particle-based materials and phenomena is nearly impossible. Animations which contain flags waving, people walking, waves rolling, Jell-O bouncing cannot be accurately produced using keyframing techniques.

Faced with the inadequacies of keyframing, research groups around the world have begun to explore mathematical-, algorithmic- and artificial intelligence-based techniques which will assist animators in the creation of complex computer animation through simulation. The research spans a broad range of topics from articulated structures research [2,3,4], to the study of dynamics and mechanics for animation
There has been significant study of the models of flexible materials [8,9,10]. The concepts of artificial intelligence are also being applied to computer animation to produce actor and group behavior [11,12,13,14]. Another interesting approach to generating animation is the utilization of mathematical models and techniques to implement constraint-based motions [16,17]. In particular the energy constraint work of Witkin, et. al. [15] motivates the work with cost functions described in this paper.

Witkin, et. al. present a simple but general approach to imposing and solving geometric constraints on parameterized models. They define geometric constraints in terms of an "energy" function. The variables of the function are the geometric parameters being constrained. The function is formulated such that the constraints are met when the ensemble of the parameter values minimizes the function. Finding the parameter values which minimize the "energy" function imposes the geometric constraint. The minimum is found by numerically calculating the gradient of the "energy" function and following it through the parameter space of the model to a minimum. The functions defined by Witkin do not necessarily model the energy of actual physical systems. Thus, one might prefer to refer to these functions as cost functions.

2. COST FUNCTIONS FOR ANIMATION

Building on the work of Witkin, et. al., the creation of goal-oriented motion through the definition of cost functions has been explored, allowing for the specification of "key goals" and "key situations". The cost function minimization process "interpolates" these key goals in order to produce the complete animation. The term "cost function" implies a more general view and usage of the technique of function minimization for computer animation. Cost functions cannot only be used to define geometric constraints, but can also be used to encapsulate high-order activities. They can be used to define goal-oriented motions and actions. This alternate application of cost functions centers around the motion that occurs during the minimization process. A cost function is defined whose variables are the animated parameters of a scene. The parameters are modified in such a way as to minimize the cost function. The minimum cost configuration can be viewed as a "key goal" configuration. The values of the parameters are stored at certain intervals during the cost function minimization. This produces a path through the parameter space of the model being animated. By incrementally moving along the parameter space curve and updating the model defined by the parameters, an animation of the model performing a goal-oriented action may be produced. A good example is path planning. A function can be written which encapsulates the idea "move from current position to goal position." Constraints may also be built into that idea. The idea then becomes "Move from here to there without intersecting obstacles, while taking into consideration geometric limitations." This idea is explored in more detail in Section 4.

Viewing cost functions as tools for defining goal-oriented motions makes the process of finding the minimum more important than actually maintaining the minimum. It is the changing of the animated parameters during the minimum-finding process which produces the goal-oriented action. Defining actions in this fashion simplifies the action specification task. It becomes simply a matter of defining a function. The disadvantage is that the animator surrenders a significant amount of control over the details of the motion. In return the animator is provided with a powerful high-level tool for specifying complicated actions. The animator specifies what must happen, not how it happens. The technique determines the details of the "how", by finding the minimum of the function.

A high-level tool like cost functions supports the concept of data amplification for computer animation. In geometric modeling, data amplification can be demonstrated by a fractal algorithm. The algorithm accepts a few polygons and by stochastically subdividing the polygons, a more complex geometric structure is created [18]. The graftal algorithms of Smith [19] also provide a large degree of data amplification when modeling plants. The user need not specify the thousands of data points which define the final fractal surface. Only a few initial points and algorithmic parameters are specified in order to create an extremely complex structure.
The same idea can be applied to computer animation. An animator may define only a small amount of
data in the form of a cost function. An algorithm is applied to the data and a complex motion is then
produced. The "data amplification" factor of the cost function approach is significantly higher than the
factor associated with standard keyframing techniques. A crowded, busy environment may be required
for a particular animated scene. A collection of moving objects may be needed to provide the backdrop
for an action in the foreground. The specifics of the background motion are unimportant, but its density,
complexity and character may be very important. Keyframing the motions of all the background charac-
ters would certainly be tedious and unnecessary. With cost functions the idea of moving through an en-
vironment and avoiding collisions with other objects is easily implemented, freeing the animator from
significant amounts of tedious work. Using cost functions, a small amount of initial information is
enhanced by the minimum-finding algorithm to create complex motions.

3. THE COST ANALYSIS OBJECT

The cost function technique for defining goal-oriented motion has been encapsulated in the cost_analysis
object [20] of The Clockworks, an object-oriented computer animation system [21,22,23]. It relies heavily
on the message passing facilities of The Clockworks, and makes full use of the other object-oriented
tools and capabilities of the system, including data structure, modeling, rendering and device interfacing
objects [24]. The cost_analysis object has been used to explore both our alternate approach to cost func-
tions and new cost functions themselves. Motion data for numerous animated situations has been generat-
ed. The types of activities which will be produced with cost functions include path planning with and
without collision avoidance, articulated motion with and without joint limits and collision avoidance, and
flocking and herding behavior. Experiments with these types of actions are detailed in Section 4.

The cost_analysis object which implements and supports choreography with cost functions is just one of
61 objects which constitute the The Clockworks computer animation system. The Clockworks is more
than a system for creating computer generated images and animations. It is a testbed environment where
research in the areas of advanced rendering, geometric modeling, particle-based systems, interaction
modeling, and user interfaces is currently being conducted. The cost_analysis object builds on the
myriad of features provided by The Clockworks. It is a part of an ever-expanding environment which is
supporting the computer graphics/computer animation research of the RDRC.

The implementation of the cost_analysis object is described in detail by Breen [20]. It has several impor-
tant features which should be highlighted. The most important of these features is its message-based
implementation. In order to utilize the object, the user specifies a collection of messages which defines the
cost function as a function of the instance variables of the objects being animated. The user also specifies
the instance variables that will be modified during the minimum-finding process as "object-name
instance-variable-name" pairs. These pairs are used to construct the messages which update the animated
instance variables. In turn, the process of updating the instance variables produces the goal-oriented
motion. The results of the minimum-finding process may be stored for use at a later time or may be seen
immediately on a graphics device.

The minimum of the cost function is determined by first numerically approximating the differential of
the cost function/instance-variable for each instance variable being animated. These differentials are collected
to produce the gradient. The gradient is the vector in parameter space which points in the direction of the
cost function's steepest descent. The cost_analysis object then uses an adaptive algorithm to move
through parameter space toward a minimum using the gradient. The size of the step taken in the direc-
tion of the gradient is adjusted based on the history of previous steps. If the cost function value increases
after taking a step, the step size is divided in half. On the other hand, if the cost function value continu-
ously decreases for a specified number of iterations, the step size is adjusted upwards by a factor of 2.
The minimum and maximum step size are parameters which are specified by the user.
The process continues until a maximum number of iterations has been performed, all the differentials effectively go to zero (i.e. a minimum has been found), or the cost function effectively goes to zero. The user may also specify three groups of messages, one at the start of the minimum-finding process, one for each iteration, and one at the end of the process. These message groups may be used to create, set, and modify temporary objects which are used to store the outcome of the minimum-finding process.

The implementation of the cost_analysis object provides a general, powerful, and versatile tool for defining cost functions for computer animation. Since all actions are defined as message strings, the object itself is flexible and not hardcoded into a particular application. The object may be used to explore a wide variety of cost functions and applications. Since the cost_analysis object has been integrated into The Clockworks’ interpretative environment, there is easy interactive access to the object and other data structuring and visualization objects.

4. EXPERIMENTS WITH GOAL-DIRECTED ANIMATION

The cost_analysis object was used to experiment with several different cost functions. The types of goal-oriented motions produced by these functions can be categorized as path planning, path planning with collision avoidance, articulated motion, articulated motion with collision avoidance and joint limits, and flocking behavior.

4.1 Path Planning

A very simple cost function that will direct an object to move from its current position to some goal position is given by

\[ cost = |P_{goal} - P_{obj}| \]

(1)

\( P_{obj} \) is the current position of the object being moved. \( P_{goal} \) is the goal position which the object is moving towards. The cost function clearly goes to zero when the moving object reaches the goal position. If the object is moving in three dimensions, the variables being modified by the algorithm would be the three components of \( P_{obj}, P_x, P_y, P_z \). This example produces the very uninteresting result of moving the object along a straight line from its current position to the goal position.

4.2 Path Planning with Collision Avoidance

A more interesting example is where an object must move from its current position to some goal position and also avoid colliding with obstacles which may be in the way. The concept of collision avoidance simply can be introduced by adding extra terms to the cost function. These terms represent cost fields around the obstacles. As the moving object enters the cost field of an obstacle, the field contributes to the cost, making it “expensive” to approach the obstacle, therefore “directing” the moving obstacle away from it. An appropriate cost field function for an obstacle based on the natural logarithm is

\[ field(d) = \ln \left( \frac{R}{d} \right) \quad 0 \leq d \leq R \]

\[ field(d) = 0 \quad d > R . \]

(2)

Here, \( d \) is the distance from the surface of the obstacle. The function goes to infinity at the surface of the obstacle and falls to zero at some specified distance from the surface, \( R \).

Collision avoidance is accomplished by adding the field functions of all the obstacles to equation (1). Given \( n \) obstacles (\( obst_i \)) in the moving object’s environment, we have
\[ \text{cost} = |P_{\text{goal}} - P_{\text{obj}}| + \sum_{i=1}^{n} \text{field} \left( \text{dist} \left( \text{obj}, \text{obst}_i \right) \right). \quad (3) \]

\text{dist} is a function which determines the distance between the surface of the object \text{obj} and the obstacle \text{obst}_i. Again, the variables which are being modified are the \( P_x, P_y, P_z \) components of the object's position.

The result of an experiment using this function is shown in Figure 1. Here, a sphere moved through an environment of 14 cylinders. Its path was automatically generated by finding the zero of equation (3). Once the path is generated, the path information can be used to generate an animation of the sphere moving through the cluttered environment.

4.3 Articulated Motion

In the previous two examples the position of a simple object was modified to move the object through space. However, the cost function approach is not limited to dealing with such straightforward applications. There does not need to be such a direct connection between the variables being modified and the components of the cost functions. Another application of cost functions is in the area of planning articulated motion.

In this application, a robot arm consisting of four revolute joints is directed to reach for a particular point in space. A cost function for this application is very similar to the one given in equation (3). The function is simply the distance between two points. The difference is that one of the points is at the end of an articulated arm. The variables which are modified to drive the cost function to zero are the joint angles of the arm. This function is given by

\[ \text{cost} = |P_{\text{goal}} - P_{\text{tip}}|, \quad (4) \]

where \( P_{\text{tip}} \) is the position of the tip of the arm, and \( P_{\text{goal}} \) is the position in space to which the arm is reaching. The cost function is deceptively simple. In fact, \( P_{\text{tip}} \) is actually a complex function of the joint angles and link dimensions of the arm. Though apparently simple, the function encapsulates a powerful concept. Varying the joint angles of the arm to drive the cost function to zero effectively directs the arm to reach for an arbitrary point in space.

4.4 Articulated Motion With Collision Avoidance and Joint Limits

In a more realistic scene a moving articulated arm must consider other factors when moving from one point to another. There are usually obstacles to avoid. It is very unusual for motion to be completely unimpeded. To account for this, collision avoidance may be added by associating a cost function field with each obstacle, and adding the field terms to the original cost function. With this addition, equation (4) becomes

\[ \text{cost} = |P_{\text{goal}} - P_{\text{tip}}| + \sum_{i=1}^{n} \text{field} \left( \text{dist} \left( \text{tip}, \text{obst}_i \right) \right). \quad (5) \]

Here, the function \text{field} has been defined as in equation (2). \text{dist} returns the distance between the tip of the robot and the surface of the obstacle \text{obst}_i. Generating a path that drives equation (5) to zero will produce an animation of an articulated arm moving its tip from its current position to some goal position while avoiding obstacles.
Another robot arm constraint is its joint limits. A revolute joint of an arm cannot generally rotate the full 360 degrees. There is some angular range in which each joint may operate. These joint limits need not be treated as a special case. Joint limits may also be incorporated into the cost function. A term can be added to the cost function which increases as a particular joint approaches its limit and is zero otherwise. An example is

\[
\text{limit} = \begin{align*}
\ln(\text{delta}/(\alpha - \text{min})) & \quad \text{min} < \alpha \leq \text{min} + \text{delta} \\
0 & \quad \text{min} + \text{delta} < \alpha < \text{max} - \text{delta} \\
\ln(\text{delta}/(\text{max} - \alpha)) & \quad \text{max} - \text{delta} \leq \alpha < \text{max},
\end{align*}
\]

where \( \text{min} \) and \( \text{max} \) are the minimum and maximum allowable angles for a particular joint. \( \text{delta} \) is the angular distance from the limits at which the limit functions become non-zero, and \( \alpha \) is the angle of a particular joint. Using equation (6), additional terms may be added to equation (5) to produce a goal-directed motion of an articulated arm which avoids obstacles and is constrained by joint limits. This gives the extended cost function

\[
\text{cost} = |\overline{\text{P}}_{\text{goal}} - \overline{\text{P}}_{\text{tip}}| + \sum_{i=1}^{n} \text{field} \left( \text{dist} \left( \text{tip, obst}_i \right) \right) + \sum_{j=1}^{m} \text{limit} \left( \alpha_j \right).
\]

Here, \( \overline{\text{P}}_{\text{tip}}, \overline{\text{P}}_{\text{goal}}, \text{field}, \text{dist}, \text{tip, obst}, \) and \( \text{limit} \) are as described in the previous examples. Angles \( \alpha_j \) are the angles of the joints of an \( m \)-jointed arm. As in the previous example, the angles \( \alpha_j \) are the variables which are modified in order to drive the cost function to zero. The result of an experiment with a 4-jointed arm is exhibited in Figure 2. In this example the arm reaches for 9 goal points. The goal points are represented by the yellow spheres. The arm avoids the obstacles in the room and is constrained by its joint limits. On two occasions, it is unable to reach a goal point because of the obstacles and its own constraints. In this situation the arm will come as close as it possibly can. The path swept out by the tip of the arm is also shown.

### 4.5 Flocking Behavior

The simulation of flocking and herding behavior has been investigated by Reynolds [14]. His approach involved the implementation of a distributed behavioral model in LISP. A complex set of rules based on conditions local to each "boid" (flock member) is used to determine its actions and motions. Each boid examines the actions of the boids around it and determines what it should be doing itself. The individual actions of each boid combine to produce the macroscopic phenomenon of flocking. The boids collect into a group and the group can be directed to follow a path. Each boid will try to avoid colliding with the other boids and any obstacles in the environment. Each boid tries to stay close to its neighbors and attempts to match their velocities. These local rules for behavior produce a simulated flock.

Flocking behavior may also be simulated with cost functions. Though the results will not be exactly like Reynolds', a type of flocking can be achieved by having each member of the flock attracted to the center of the flock and repelled by each other member and obstacles. As the center of the flock is moved, the members of the flock will move with it. The members of the flock may be directed to maintain a certain distance from each other and to avoid obstacles. A random term may also be added to the cost function to provide minor perturbations in the individual motions of each flock member. The cost function for a flock of \( n \) objects flying through an environment with \( m \) obstacles is

\[
\text{cost} = \sum_{i=1}^{n} |\overline{\text{P}} - \overline{\text{P}}_{\text{center}}| + \sum_{i=1}^{n} \sum_{j=1}^{m} \text{field} \left( \text{dist} \left( \text{mem}_i, \text{mem}_j \right) \right) + \sum_{i=1}^{n} \sum_{j=1}^{m} \text{field} \left( \text{dist} \left( \text{mem}_i, \text{obst}_j \right) \right)
\]
$P_{center}$ is the position of the center of the flock. The $P_i$'s are the positions of the members of the flock. $field$, $dist$, $obst$, have been defined in previous examples. The position $P$ of each flock member is the variable which is modified to minimize the cost function. Results of an experiment with such a flocking behavior cost function are exhibited in Figure 3 and Figure 4. Figure 3 illustrates how the flock members will move toward the center of the flock while avoiding obstacles. Figure 4 is a close-up of the center of the flock and illustrates how the members will cluster around the center while avoiding each other.

5. CONCLUSION

A new application for cost functions in computer animation has been presented. Cost functions may be used in a more general way than to simply enforce geometric constraints; they may also be used to define goal-oriented actions. A function can be defined which has animation parameters as its variables. A goal-oriented action can be produced by driving the cost function to a minimum. Experiments with this approach have been shown to generate collision-free paths around obstacles, reaching motions of constrained articulated arms and flocking behavior.

Cost functions are another data-enhancing tool in the animator’s toolbox. A complex motion or activity may be produced by simply defining a single function. Cost functions provide a uniform method for defining a variety of activities. These activities may build on each other by adding additional terms to the function. There are problems with the approach. The zero-finding algorithm may become stuck in a local minimum which is not the goal configuration. Often a simple readjustment of the animation parameters alleviates this problem.

An object has been created and integrated into the object-oriented system, The Clockworks, which encapsulates the algorithm and data structures necessary to implement the cost function approach. It is a message-based object which utilizes the message passing, data structuring, geometric modeling, and rendering capabilities of the system. The cost analysis object significantly enhances the choreography capabilities of the system. It will be used to further investigate the applications of cost functions in computer animation.

Acknowledgement

This work was conducted while the author was visiting the Zentrum für Graphische Datenverarbeitung, Darmstadt, Federal Republic of Germany. I would like to thank Dr. Michael Wozny, Dr. Jose Encarnação, Ms. Mary Johnson, and Mr. Detlef Krömker for organizing and supporting the cooperative exchange which brought me to Darmstadt. I would also like to thank Dr. Donald House for his helpful and voluminous editorial comments. This work was supported by the Industrial Associates Program of the Rensselaer Design Research Center. Any opinions, findings, conclusions, or recommendations are those of the author and do not necessarily reflect the views of the IAP or its members.
Figure 1. Path Planning Example

Figure 2. Constrained Articulated Motion Example
Figure 3. A Flocking Example

Figure 4. Behavior Around The Center Of The Flock
References


For David E. Breen's biography, see page 82.