Manipulating Surfaces Differentially

William Welch [1]
Michael Gleicher[2]
Andrew Witkin

September 10, 1991
CMU-CS-91-175

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

To appear in *Proceedings Compugraphics '91,* Springer Verlag.

Abstract

We want to create interactive surface design systems which provide intuitive interfaces to parametric surface representations. In this paper, we show how the technique of *Differential Manipulation* can be used in constructing such interfaces. It allows surface manipulation issues to be treated separately from surface representation issues. Arbitrary differentiable functions of representation parameters can be used to control the surface. Constraint and optimization techniques can be used to enhance interaction and control many surface degrees of freedom at once. We provide examples of the technique's use in our interactive surface modeling program.

[1]electronic mail: claude@cs.cmu.edu

[2]electronic mail: gleicher@cs.cmu.edu

This research was sponsored in part by Apple Computer, Siemens Corporation and Silicon Graphics Incorporated. The second author is supported in part by a fellowship from the Schlumberger Foundation, Inc. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of Schlumberger, Siemens, SGI or Apple.

Keywords: Interaction Techniques, Computational Geometry and Object Modeling, Computer Aided Design, Surface Modeling

We are interested in the interactive design of free-form surfaces. Such systems must be able to represent the diversity of objects which designers require, and should still be easy to use. Unfortunately, the need for sufficiently expressive representations is at odds with the desire for intuitive interaction methods. Much of the prior work addressing this issue has been in defining surface representations which are expressive yet easy to control. Interpolating splines, B-spline[GR74] and NURB[Til83] control nets, and transfinite curve skinning[Gor69] are all attempts to specify surfaces in terms of parameter sets which are both extensible to more degrees of freedom in a regular way, and are easy to control because of the straightforward geometric meaning of their parameter sets. Indeed, the interfaces provided to such representations often directly couple parameters to user controls (a pervasive example being control-point dragging for B-spline nets).

Such direct manipulation interfaces – in which the explicit degrees of freedom of the representation are presented to the user as a set of independent controls – have several shortcomings. First, a fixed representation cannot offer parameters which correspond directly to the range of effects desired by designers. Second, most representations achieve general expressiveness by offering numerous degrees of freedom, each of which must be specified independently in creating a surface. Making even incremental changes to such a surface can require adjusting large numbers of parameters. Third, interactions between qualitatively different parameters (for example, control-point positions, weights, and knot spacings in a NURBS curve) can leave a designer confused as to exactly how a given shape should be achieved.

In this paper, we pursue an alternative approach to providing interfaces for surface manipulation. Rather than seeking a representation which is easier to control, we present a way to uniformly manipulate surfaces defined in any parametric representation. We thus separate the problem of manipulation from the problem of representation. The technique of *Differential Manipulation*[GW91a] permits us to provide controls which are convenient for the user and lets us map operations on these controls to changes in the representation parameters.

A previous paper[GW91a] discussed how the technique can be used to construct direct manipulation interfaces for a wide variety of geometric objects. In this paper, we will use it to build interfaces which abstract away from rich underlying representations and provide controls which can modify many degrees of freedom at once. This is particularly effective in surface sculpting applications. The resulting interface retains the expressivity of the underlying representation while giving the user direct control over meaningful higher-level shape parameters expressed as functions of an object's basic parameters. The technique also suggests very natural ways to control potentially unintuitive degrees of freedom in the underlying representation.

We first present an overview of Differential Manipulation. We give a geometric specification for "intuitive" behavior of objects in a direct-manipulation system. We then show that this same behavior can be described in terms of the equations of motion for a simple mechanical system. We show how to implement general controls within this framework, and discuss applications to surface sculpting, including a number of examples from our B-spline surface editor.

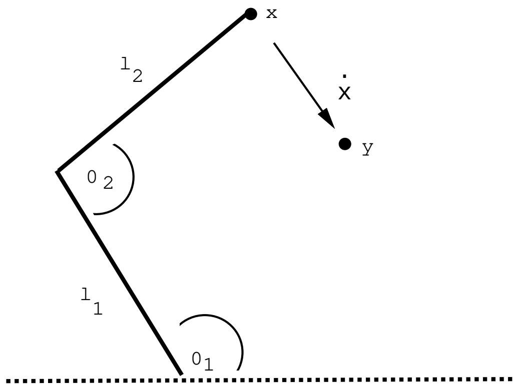

Figure 1: A two-joint linkage, with joint-angles $\{\theta_1, \theta_2\}$. The velocity of point x is related to $\{\theta_1, \theta_2\}$ by the Jacobian of its position function.

1. Differential Manipulation

We would like to allow the user to manipulate a surface using controls which are computed from the representation parameters. For example, the user might control positions of surface points, tangents and normals of various points on the surface, or area and curvature properties. For some of these controls it may be possible to compute an inverse which maps directly from a specified function value or object configuration to the corresponding parameter settings. But in general, such a direct mapping will not be possible because nonlinear or underconstrained systems of equations preclude inverses.

If, however, it is possible to compute derivatives of the control functions, the underlying parameters can be altered differentially. Rather than specifying the target value for a control, the user specifies, from moment to moment, how the controls should move. These motion requests can be mapped into small changes in the parameters. By iterating this process, the object moves with continuous motion in response to the users control, permitting the user to "push and pull" on the object to move it into a desired configuration. Because we are only specifying motion of the outputs (as opposed to positions), we only require derivatives (as opposed to inverses) of the control functions.

Differential manipulation uses numerical techniques to allow users to manipulate geometric objects. In this section, we will motivate the use of techniques from optimization through a series of simple examples. In the final part of this section, we will show how the same results can be obtained using techniques from mechanics.

1.1. Generalized Coordinates

Consider the two-joint linkage in Figure 1. For fixed l_1, l_2, the linkage's configuration is determined by the joint angles θ_1 and θ_2. We concatenate these to form the object's *state vector*, $\mathbf{q} = \{\theta_1, \theta_2\}$, which represents the generalized coordinates of the current configuration in the space of possible configurations. The world-space coordinates of any point on the object

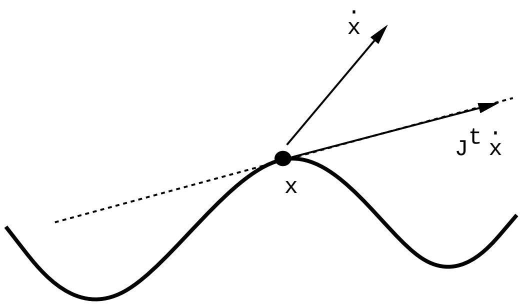

Figure 2: A point constrained to a curve. The user-specified velocity is projected onto the space of allowable velocities, the tangent at the point.

can then be expressed as a function of $\mathbf{q}$.

We now consider how to allow the user to manipulate the point $\mathbf{x}$ on this object. Allowing a user to specify a new position for the point directly requires solving a nonlinear system of equations. Instead, we allow the user to control the motion of the point. For example, instead of moving $\mathbf{x}$ to a target point, the user will specify the instantaneous velocity $\dot{\mathbf{x}}$ (the dot indicates the time derivative of the function) so that $\mathbf{x}$ moves *toward* the target. The Jacobian of the position function, $\mathbf{J} = \mathbf{x}_q$ (the 2 x 2 matrix of partial derivatives of $\mathbf{x}$ with respect to $\mathbf{q}$) linearizes $\mathbf{x}$ about the current $\mathbf{q}$, and solving

$$\mathbf{J}\dot{\mathbf{q}} = \dot{\mathbf{x}} \tag{1}$$

for $\dot{\mathbf{q}}$ transforms the world-space velocity to the corresponding generalized velocity.

If we specify that the velocity $\dot{\mathbf{x}}$ be some multiple of the displacement from a target point $\mathbf{y}$, $\mathbf{x}$ will continually move towards $\mathbf{y}$ as $\dot{\mathbf{q}}$ is integrated. By making the target point be the position of the mouse, the linkage will track the pointer as the user moves it about in world-space as if the two were connected by a damped spring. The user can manipulate the linkage without regard to the underlying parameterization in $\{\theta_1, \theta_2\}$.

1.2. Least Squares Projection

Consider a point which is constrained to slide along a fixed curve (Figure 2). Clearly, if the user pulls the point in any direction which is not tangent to the curve at the point, the point cannot move in that direction. But the user would be reasonable in expecting the point to slide along the curve so long as the specified velocity was not perpendicular to the curve at the point. We get this behavior by projecting the given velocity onto the (linearized) space of allowable displacements for the point. In this example, this space is the line tangent to the curve at that point; in general, it is the space spanned by the column vectors of the Jacobian. Thus, while $\mathbf{J}\dot{\mathbf{q}} = \dot{\mathbf{x}}$ is in this instance overdetermined (two equations in one unknown), the projected equation

$$\mathbf{J}^T\mathbf{J}\dot{\mathbf{q}} = \mathbf{J}^T\dot{\mathbf{x}} \tag{2}$$

does have a solution. Further, it is the *least squares solution* to $\mathbf{J}\dot{\mathbf{q}} = \dot{\mathbf{x}}$, in that the solution $\dot{\mathbf{q}} = (\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T\dot{\mathbf{x}}$ minimizes the error $(\mathbf{J}\dot{\mathbf{q}} - \dot{\mathbf{x}})^2$. Under this model, one might think of user-

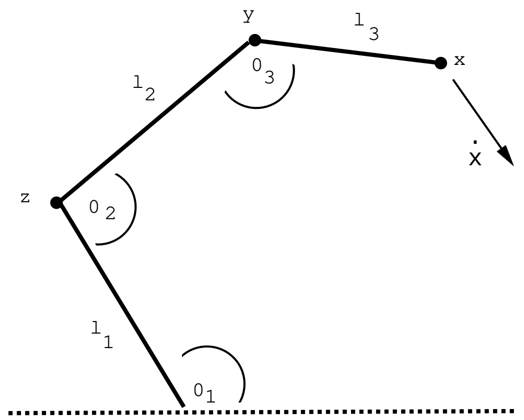

Figure 3: A three-joint linkage. $\dot{\mathbf{x}}$ does not uniquely determine $\dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3$, so additional velocities must be specified.

specified velocities as "suggestions" rather than hard constraints. The system moves so as to minimize the error between the requested and actual velocities.

1.3. Underdetermined Motions

Consider the three-joint linkage in Figure 3. A user-specified velocity for $\mathbf{x}$ leads to an underdetermined system: two equations in three unknowns. One way to resolve this indeterminancy is to define a reasonable behavior for the objects which sufficiently constrains the problem.

We first note that if the user had specified velocities for $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ simultaneously, we could generalize Equation 2, the system would be overdetermined, and a unique least-squares solution could be found. The generalization is straightforward: we concatenate each of the world-space points into the vector $\mathbf{P}(\mathbf{q})$, which we call the configuration vector (the world-space analog of the state-vector). The user-specified velocities are similarly concatenated to form $\dot{\mathbf{P}}$. Now $\mathbf{J}$ will contain a row each for each point in the configuration vector, and the system of equations becomes $\mathbf{J}\dot{\mathbf{q}} = \dot{\mathbf{P}}$. If there are at least as many independent rows in $\mathbf{J}$ as there are components in $\mathbf{q}$, the matrix $\mathbf{J}^T\mathbf{J}$ will be positive-definite, and there will be a unique least-squares solution.

Given this generalization, we are able to sufficiently constrain the problem by making the following crucial assumption: in addition to explicitly specifying velocities for some of the points in $\mathbf{P}$, the user has implicitly specified *zero-velocities* for the remaining points – that is, these points should not move if the user is not pulling on them. This is reasonable behavior for the model, and guarantees that for a sufficiently populated $\mathbf{P}$, there is a unique least-squares solution to any set of user-specified velocities interpreted as $\dot{\mathbf{P}}$. When a nonzero velocity has been specified for a point which is coupled to a zero velocity point by the underlying parameterization, this solution minimizes the distance the zero velocity point actually moves.

1.4. Generalized Configuration Vectors

In the previous example, $\mathbf{P}$ was a collection of world-space points. It is worth emphasizing that the components of $\mathbf{P}$ can be any differentiable functions of $\mathbf{q}$. A good example arises in the context of manipulating a curve.

Consider a parametric curve $\mathbf{C}(\mathbf{q}, s)$ whose shape is determined by the parameter vector $\mathbf{q}$. To manipulate the curve differentially, we must somehow construct a configuration vector $\mathbf{P}(\mathbf{q})$ which will uniquely determine a corresponding value for $\mathbf{q}$. One could of course construct $\mathbf{P}$ by sampling points along the curve. The movement of the sample points would be minimized as the curve changed in response to velocity signals from the user, and given a sufficient number of sample points this would in turn minimize the magnitude of world-space change along the entire curve.

Consider instead the limit of this sampling process, $\int \mathbf{C} ds$. Under a least squares projection, specifying a zero velocity for the value of the integral would result in minimizing the magnitude of the corresponding change in the curve's world-space configuration,

$$\|\Delta \mathcal{C}\|^2 = \int (\mathbf{C}(\mathbf{q} + \mathbf{\Delta q}, s) - \mathbf{C}(\mathbf{q}, s))^2 ds.$$

For a differential change $\mathbf{dq}$, the derivative $\mathbf{C}_q(\mathbf{q})$ linearizes this expression, yielding

$$\begin{aligned} \int (\mathbf{C}(\mathbf{q} + \mathbf{dq}, s) - \mathbf{C}(\mathbf{q}, s))^2 ds &= \int (\mathbf{C}_q \mathbf{dq})^2 ds \\ &= \mathbf{dq}^T (\int \mathbf{C}_q^T \mathbf{C}_q ds) \mathbf{dq} \\ &= \mathbf{dq}^T \mathcal{S} \mathbf{dq}, \end{aligned}$$

where $\mathcal{S}$ represents the integral over the curve parameter s, and is a symmetric, positive-definite matrix. The displacements which minimize the magnitude of the change in the curve are those which minimize the gradient of this function at the current value of $\mathbf{q}$. That is, we want

$$\mathcal{S} \mathbf{dq} = 0.$$

This yields $\dim(\mathbf{q})$ independent equations in $\mathbf{q}$, and we may take each of these as a row in the Jacobian of a generalized configuration vector, for which a zero velocity will be specified. The actual components of the configuration vector would be given by the solutions of these differential equations – but note that we do not have to solve them since we are only interested in generating the necessary independent rows for the Jacobian. Minimizing each of these velocities results in minimizing the world-space change in the entire curve.

As a simple example of this technique, consider a parametric curve represented as a linear blend of a set of control points $\mathbf{G}$:

$$\mathbf{C}(\mathbf{G}, u) = \mathbf{N}(\mathbf{u}) \mathbf{G}.$$

Taking $\mathbf{G}$ as the state vector, we see that $\mathcal{S}$ is independent of $\mathbf{G}$, and the shape projections are just $\mathcal{S}\mathbf{G}$. Further, if the basis function $N(u)$ has local support, as with a B-spline, $\mathcal{S}$ will be sparse and dominated by its diagonal entries, and may be reasonably approximated by a diagonal scaling matrix, reflecting the independence of the underlying control-points.

This leads to the result that control points for a linearly blended curve may be pressed into service as the components of a generalized configuration vector. This should come as no surprise, since the original B-spline representation was created with the intention that the control-points be direct-manipulation parameters.

The use of control points in the configuration vector leads in turn to a very simple formula for directly manipulating a linear blend curve. If the user is pulling on a single point $\mathbf{x}(u)$ of the curve, $\mathbf{J}$ and $\dot{\mathbf{P}}$ become:

$$\mathbf{J} = \begin{vmatrix} N(u)^T & & \\ \alpha & & \\ & \ddots & \\ & & \alpha \end{vmatrix} \quad \dot{\mathbf{P}} = \begin{vmatrix} \dot{\mathbf{x}} \\ 0 \\ \vdots \\ 0 \end{vmatrix} .$$

For sufficiently large α, the matrix $\mathbf{J}^T\mathbf{J}$ is essentially the identity scaled by α^2, and we thus choose to disregard the $\mathrm{N}(u)$ terms in $\mathbf{J}^T\mathbf{J}$ (that is, the controlled point has already been accounted for by the integration along the curve. The $\mathrm{N}(u)$ terms merely provide additional weighting at that point). With $\alpha = 1$, Equation 2 then becomes:

$$\dot{\mathbf{q}} = \mathrm{N}(u)^T\dot{\mathbf{x}}.$$

Further, since $\mathbf{J}$ is constant, we may directly solve for the total change in $\mathbf{q}$: it is simply a multiple of $\mathrm{N}(u)$. Because of the least squares weighting, the $\Delta\mathbf{q}$ which displaces a world-space point $\mathbf{x}(u)$ by $\Delta\mathbf{x}$ is

$$\Delta\mathbf{q} = \frac{\mathrm{N}(u)^T\Delta\mathbf{x}}{\|\mathrm{N}(u)\|^2}$$

Figure 4 shows a B-spline curve being manipulated in this way. The resulting displacements are those which minimize the magnitude of the change in the curve while placing the controlled point at the desired position.

1.5. First-Order Equations of Motion

That the previous techniques lead to animated models which respond to user interaction in uniform, intuitive ways is not accidental. The equations – and motion – arising from this purely geometric specification correspond exactly to a those of a physical simulation of a mechanical system.

The basic equation of motion for a mechanical system is

$$\mathbf{f} = \mathbf{M}\ddot{\mathbf{p}} + \mathbf{C}\dot{\mathbf{p}} + \mathbf{K}\mathbf{p}, \tag{3}$$

where $\mathbf{p}$ is a vector of particles in the system and $\mathbf{f}$ is a corresponding vector of applied forces. $\mathbf{M}$ and $\mathbf{C}$ are the mass and damping matrices – linear maps from accelerations and velocities to forces. $\mathbf{K}$ is the stiffness matrix, which relates the current configuration to the amount of potential energy in the system.

The dynamic behavior specified in the previous section corresponds to a physical situation in which drag dominates inertia, so that a particle only moves while a force is being applied

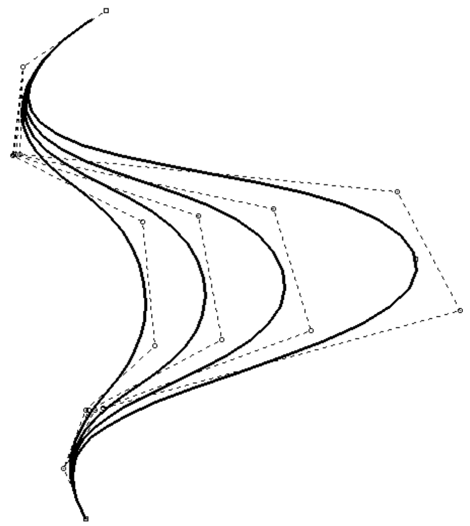

Figure 4: Pulling on a point on a B-spline curve. Dashed lines connect the spline control points. Displacements for the point on the curve are mapped to corresponding displacements in the control points.

to it (ie, $\mathbf{M} = 0$). The particles in $\mathbf{p}$ correspond to the components of the configuration-image $\mathbf{P}$, and the applied forces $\mathbf{f}$ to the user-supplied velocity signals.

For now, we assume there is no potential energy in the system, so $\mathbf{K} = 0$. Because we have not coupled any of the components of $\mathbf{P}$ (ie, $\mathbf{C}$ is the identity), the equation of motion is reduced to

$$\dot{\mathbf{p}} = \mathbf{f}.$$

Finally, we view the parameters in $\mathbf{q}$ as an alternate coordinate representation for the configuration of the particles $\mathbf{p}$, and the standard change of coordinates from mechanics yields

$$\mathbf{J}^T\mathbf{J}\dot{\mathbf{q}} = -\mathbf{J}^T\mathbf{f},$$

where, as before, $\mathbf{J}$ is the Jacobian of the map from $\mathbf{q}$ to $\mathbf{p}$.

This is simply Equation 2, but with a physical interpretation. It is a valuable correspondence, because it motivates the use of techniques from mechanics in constructing intuitive user interfaces[WGW90].

2. Manipulating Surfaces

We now consider Differential Manipulation as it applies to interactive surface modeling. In particular, we show how the technique addresses the problems mentioned earlier: fixed parameterization, unintuitive parameter interactions, and specifying many independent degrees of freedom. In the previous section we showed how to differentially control functions of the state vector. We will now look at some control functions useful for free-form surface design. These controls will be formulated independently of any particular surface representation. For concreteness, we give some examples of controls from our B-spline surface editor.

2.1. Control Functions

Differential Manipulation hides the underlying representation of a model by letting us operate on control functions of the state vector. Such a control may have a simple geometric interpretation, while the corresponding effect on the state vector will reflect the complexity of the mapping from state-space to world-space quantities.

For example, in our surface modeler we allow the user to control the position of any point on the surface, even if that point is not a control point, by generalizing the curve manipulation results from section 1.4. Similarly, we allow the user to pull directly on surface tangents and normals. One interface we have used for this is to attach a control rod (which represents the second parametric derivative) to the surface at the specified point, and let the user pull on the end of the rod. Changing the rod's direction changes the tangent plane at the control point, and stretching the rod increases the curvature of the surface at that point. More interestingly, we can ask designers what they want to control, and write functions to do it.

Such a control scheme requires the dynamic creation and evaluation of control functions and their derivatives[GW91b]. Note that because of the local support property of B-splines, functions of $\mathbf{q}$ generally depend only on a subset of the components, and the resulting Jacobians will be very sparse vectors. If a model with many degrees of freedom is to run at interactive speeds, it is crucial that an implementation take advantage of this sparsity.

2.2. Hierarchical Parameterizations

It is common to structure surface design parameters hierarchically. For example, one might embed a surface in a deformation[Bar84], then apply a rigid-body transformation to the resulting deformed points. This is an appealing design strategy because parameters at each level may be treated as independent sets of controls, with well-defined scopes of effect. But they are certainly not independent in terms of the parameters' effect on the world-space location of a particular surface point. Placing a surface point at a particular world-space position by manipulating deformation parameters is rarely straightforward given such a structure.

Traditional direct manipulation of a surface embedded in such a hierarchy would be possible only if we could compute each deformation's inverse. But because the deformation hierarchy is simply a composition of functions, the chain rule lets us formulate differential controls which map forces from one level of composition to the next, and thus apply formces not only in world-space or state-space, but in each of the intermediate spaces in the hierarchy.

The parameters at a specified level in the hierarchy can be isolated for manipulation simply by freezing parameters associated with other levels. World-space forces are then transmitted only to the active level. Several levels may be active at once, with the damping matrix providing a way of apportioning the applied force between them. Thus, the user can manipulate any kind of parameter at any level of the hierarchy in the same way: by applying forces directly to the world-space surface.

For example, in our modeling system one can specify a rigid-body rotation by grabbing an object and pulling it around to the desired orientation (the surface degrees of freedom are frozen during this operation). This interface is markedly different from many commonly encountered rotation controls[CMS88], and is very easy to use for quick, non-precision positioning of an object.

2.3. Constraints for Manipulation

Sometimes it is useful to have the modeling system itself control a function's value. For instance, if we have the modeler generate forces which hold a function at a fixed value during manipulation, this enforces a constraint relationship between the underlying parameters and thereby reduces the number of degrees of freedom of the model. Because we may constrain arbitrary functions of the modeler state, the explicit degrees of freedom in the underlying representation have only an indirect effect on what kinds of constraints we can effectively enforce.

There are a number of ways to differentially enforce a constraint. Penalty methods, which use springs to pull output functions to desired values[Pla89] are simple to implement but can lead to very stiff differential equations. This in turn requires that smaller integrator steps be taken, degrading the interactivity of the system. Alternatively, we formulate a set of constrained differential equations to restrict allowable motions of the model. Rather than try to solve these constrained equations, we use a variation of the Lagrange Multiplier technique to convert them to unconstrained equations by computing forces of constraint and projecting out the portion of the applied force which would violate the constraints [Bau72] (the application of this technique to interactive systems is further discussed in [GW91a]). Useful modeling constraints can be highly nonlinear functions of state, but because we maintain them differentially we need only solve linear systems of equations at each timestep to compute the needed constraint forces.

We have experimented with a number of different constraints in our system. The simplest of these nails a fixed material point $\mathbf{x}$ to a fixed point $\mathbf{p}$ in space. The constraint function is $\mathbf{C} = \mathbf{x}(\mathbf{q}) - \mathbf{p}$, and by holding its three components at 0 the system forces points to remain coincident (Figures 5,6). A constraint which connects two surface points is formulated similarly. In the case of a B-spline surface representation, this is a linear constraint, and its constant Jacobian makes it inexpensive to maintain.

If we nail a surface point whose material coordinates are allowed to vary – that is, whose u, v coordinates are part of the state vector – we get a (nonlinear) sliding point of contact constraint. The surface freely slides over the point in world space, always maintaining contact.

We may also constrain a surface normal to lie in a particular direction. This may be used in conjunction with a connection constraint to force two surfaces to lie tangent at a point. If the second parametric derivative is used instead of the unit normal, the constraint will be linear in the control points. This scheme has some drawbacks in regions of low curvature, arising from its dependence on the underlying parameterization.

Note that the world-space nail-point need not be fixed. The time derivative of the point may be included in the constraint formulation to cause a material point on the object to track the nail as it moves from one location to another[WW90]. We have used such a scheme in a

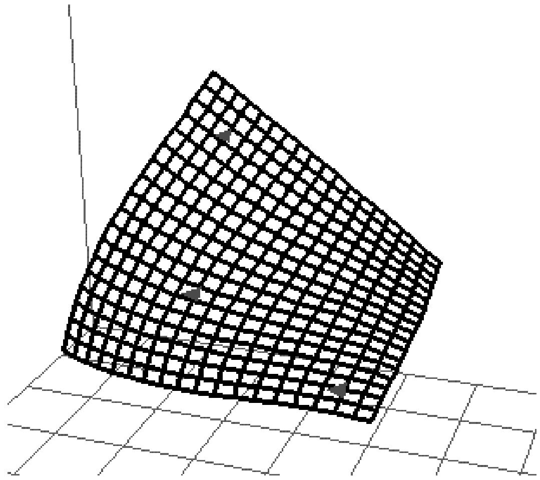

Figure 5: A B-spline surface with three point constraints.

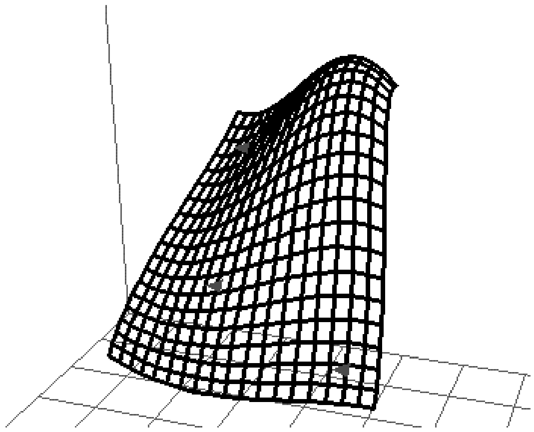

Figure 6: Manipulating the constrained B-spline surface in Figure 5.

user-interface for assembling parts of a model via constraints. The resulting motion brings the model into a configuration which will satisfy a connection constraint between the parts while preserving existing constraints. If we simply relied on the constraint stabilizing forces to pull the model into a satisfying configuration these forces would tend to momentarily disrupt other constraints, or worse, cause the integration to diverge.

2.4. Optimization for Manipulation

Instead of having the system constrain a function's output at a particular value, we can have it generate forces which tend to drive the function to 0, and thus optimize (rather than constrain) the control quantity. Because it is possible to formulate control functions which involve many surface degrees of freedom, nontrivial surface features can be expressed. We thus are able to optimize local or global surface attributes subject to user-supplied forces and constraints.

As an example, we consider a simplified energy of deformation used in [CG91] to create fair-seeking patches for interactive surface design. The integrals of the first and second parametric derivatives are taken as linearized approximations to the metric and curvature tensors for a surface patch. Minimizing the local surface metric causes the surface to behave like a soap-film, contracting in world-space until a minimal area is achieved (balanced against other constraints and forces on the surface). Minimizing the local curvature across the patch has the effect of distributing curvature evenly across the patch, causing it to seek a fairer shape.

From the user's point of view these forces reparameterize the model in terms of a higher-level surface characteristic (fairness). Other possibilities for optimization forces include the shape attractors discussed in[TWK87] and [WFB87]. In optimizing such function outputs, the modeler continuously adjusts numerous surface parameters in nontrivial ways in response to simple manipulation by the user.

2.5. Other Controls and Representations

With Differential Manipulation, we can use the same manipulation techniques for different surface modeling representations. We can build systems which integrate many, such as tensor product surfaces, free-form deformations[SP86] and parameterized transforms.

Differential Manipulation allows us to use many different types of controls with a given surface type. Since we can easily define new types of controls, we can provide ones based on what designers want, or even provide tools which allow users to define their own.

The use of Differential Manipulation allows us to address many of the difficult issues of surface design in our modeling programs. Because we can define controls independently of the representation parameters, it is not a problem that the representation parameters are unintuitive or hard to control. The large number of parameters required for expressive surface representations become easier to manage through the use of controls which operate on groups of parameters, and constraint and optimization techniques.

References

[Bar84] A. H. Barr. Global and local deformations of solid primitives. *Computer Graphics*, 18(3), July 1984. Proceedings SIGGRAPH '84.

[Bau72] J. Baumgarte. Stabilization of constraints and integrals of motion in dynamical systems. *Computer Methods in Applied Mechanics*, 1972.

[CG91] George Celniker and Dave Gossard. Deformable curve and surface finite-elements for free-form shape design. *Computer Graphics*, 25(4), July 1991. Proceedings SIGGRAPH '91.

[CMS88] Michael Chen, S. Joy Mountford, and Abigail Sellen. A study in interactive 3d rotation using 2d input devices. *Computer Graphics*, 22(4):121–130, August 1988. Proceedings SigGraph '88.

[Gor69] W. Gordon. Spline-blended surface interpolation through curve networks. *Journal of Math and Mechanics*, 18(10), 1969.

[GR74] W. Gordon and R. Reisenfeld. B-spline curves and surfaces. In R.E Barnhill and Risenfeld R., editors, *Computer Aided Geometric Design*. Academic Press, 1974.

[GW91a] Michael Gleicher and Andrew Witkin. Differential manipulation. to appear in *Graphics Interface '91*, June 1991.

[GW91b] Michael Gleicher and Andrew Witkin. Snap together mathematics. In Edwin Blake and Peter Weisskirchen, editors, *Advances in Object Oriented Graphics 1: Proceedings of the 1990 Eurographics Workshop on Object Oriented Graphics*. Springer Verlag, 1991. Also appears as CMU School of Computer Science Technical Report CMU-CS-90-164.

[Pla89] John Platt. *Constraint Methods for Neural Networks and Computer Graphics*. PhD thesis, California Institute of Technology, 1989.

[SP86] Thomas Sederberg and Scott Parry. Free-form deformation of solid geometric models. *Computer Graphics*, 20(4), August 1986. Proceedings SIGGRAPH '86.

[Til83] W. Tiller. Rational b-splines for curve and surface representation. *IEEE Computer Graphics and Applications*, 3(6), 1983.

[TWK87] Demetri Terzopoulos, Andrew Witkin, and Michael Kass. Symmetry seeking models and 3d object reconstruction. *International Journal Computer Vision*, 1(3), 1987.

[WFB87] Andrew Witkin, Kurt Fleischer, and Alan Barr. Energy constraints on parameterized models. *Computer Graphics*, 21(4), July 1987. Proceedings of SIGGRAPH '87.

[WGW90] Andrew Witkin, Michael Gleicher, and William Welch. Interactive dynamics. *Computer Graphics*, 24(2):11–21, March 1990. Proceedings 1990 Symposium on Interactive 3d Graphics.

[WW90] Andrew Witkin and William Welch. Fast animation and control of non-rigid structures. *Computer Graphics*, 24, August 1990. Proceedings SIGGRAPH '90.