Abstract

In this paper, we describe a novel technique that uses Moving Least Squares (MLS) method to interpolate sparse constraints over mesh surfaces. Given a set of constraints, the proposed technique evaluates directly on the surface to generate a smooth scalar field that interpolates or approximates the constraints. To provide better control of the design of the scalar field, three types of constraints are introduced: point-value, point-gradient and iso-contour constraints. Furthermore, it is also provided to the user control over the region of influence and rate of influence decrease of the constraints through parameter adjustment. At the end of the paper, we demonstrate that the scalar fields resulted from this framework can be used in several mesh applications such as deformation, region selection and curve drawing.


Keywords: scattered data interpolation, moving least squares, scalar field design, geodesic path, mesh processing

1 Introduction

The ability to construct smooth scalar fields over surfaces from a set of sparse constraints is a useful tool in a variety of geometric algorithms. Naturally, the requirements of the application might lead to fairly different scalar field constructions. Techniques suitable for constructing coordinate functions of parameterizations are not necessarily suitable for interpolating color values being painted by an artist, for example. In this paper, we focus primarily on the use of scalar fields to define families of smooth isocurves on 2-manifold surfaces. Such fields and their isocurves are particularly useful for describing smooth curves on the surface, defining the boundaries of smooth regions, and similar constructions. Implicit functions have proven to be particularly well-suited to this kind of construction in 2- and 3-dimensional Euclidean space and it is this type of method that we choose to generalize to the manifold setting.

We present a new generalization of the well-known Moving Least Squares (MLS) technique which supports the construction of smooth scalar fields on 2-manifold surfaces. Given a set of constraints, we demonstrate how to solve for a piecewise linear field that interpolates the constraints directly over the surface. We call this the field generation problem. In addition, we also seek to address the field design problem, where a user is provided sufficient flexibility to interactively construct a suitable field by adding and manipulating constraints. The method we propose provides three types of constraints to achieve the goal: (1) scalar value at a point, (2) gradient vector at a point, and (3) isocontour on the surface.

The key to our construction is to use geodesic curves to provide both the point-to-point distances required for the MLS basis functions as well as for parallel transport of gradient vector constraints. In the process of doing this, we also develop a modified geodesic construction algorithm that supports 1-dimensional sources without extra computation.

The result is a system that enables the construction of smooth scalar fields on manifolds from a sparse set of constraints. Just as similarly constructed MLS fields in Euclidean space have numerous applications, so to do our MLS fields on manifolds. We demonstrate that they can be particularly useful for defining smooth curves on the surface and for other local mesh modification operations.

2 Related Work

MLS method has been extensively used in graphics applications. Examples of that include surface reconstruction [Fleishman et al. 2005], [Sorkine and Cohen-Or 2004], implicitization [Shen et al. 2005], image deformation [Schaef er et al. 2006], etc. However, to our knowledge, no work has explored using MLS to interpolate a scalar field directly over 2-manifolds.

Generating scalar fields over the surface has been approached in [Dong et al. 2006], in the context of quadrilateral surface remeshing. Although their method produces field suitable for remeshing application, it provides little freedom to the user in the field design. Another work by [Hua and Qin 2003] uses scalar field in the Euclidian domain as guidance for free-form deformation. In a divergent, but yet similar research direction, several vector field design techniques has been developed recently - [Zhang et al. 2006] [Fisher et al. 2007].

In [Lodha and Franke 1997], the authors present a good survey of general scattered data interpolation methods. The survey includes a few interpolation techniques over surfaces. However, these techniques have the capability to interpolate scalar value, but not other types of controls which enhances the field design. Yet, these techniques make too rough approximations on the geodesic distance evaluation, by approximating the surface with spherical models. Another radial basis interpolation in the context of shape reconstruction in Euclidian space is presented in [Nielson 2004].

Many geodesic algorithms have been proposed in the literature [Mitchell et al. 1987], [Goldberg and Harrelson 2005], [Kapoor 1999], [Kimmel and Sethian 1998]. Fast exact and approximate geodesic algorithm [Surazhsky et al. 2005] has gained significant attention in the geodesic evaluation over triangle meshes. However, its computational and storage requirement makes it too bulky for applications that do not require high accuracy. On the other hand,
Novotni et. al [Marcin Novotni 2002] developed an approximate geodesic algorithm that is fast. The algorithm was later improved by fast approximate geodesic [Jie Tang and Zhang January, 2007].

3 Interpolation method

Given a set of sparse constraints over a surface, we would like to build a piecewise linear function \( f(x) \) over the surface, that interpolates or approximates the constraint values at those points. The primary tool that we use to build the function is Moving Least Squares (MLS). In section 3.1, we review the formulation of standard MLS in the Euclidian setting. In section 3.2, we extend the formulation to 2D manifolds embedded in 3D space.

3.1 MLS for value constraints in Euclidean space

As formulated in [Shen et al. 2005], given \( N \) points positioned at \( p_i \), where \( i = 1, \ldots, N \), to build a function \( f(x) \) in Euclidean that interpolates or approximates the values \( \phi_i \) at those points, one solves the standard least squares fit

\[
\begin{bmatrix}
  b^T(p_1) \\
  \vdots \\
  b^T(p_N)
\end{bmatrix}
\begin{bmatrix}
  \phi_1 \\
  \vdots \\
  \phi_N
\end{bmatrix}
= \begin{bmatrix}
  w(x, p_1) \\
  \vdots \\
  w(x, p_N)
\end{bmatrix},
\]

(1)

where \( b(x) \) is the vector of basis functions used for the fit, and \( c \) is the unknown vector of coefficients.

In the above equation, \( b(x) \) can vary according to the fit of choice, for example, \( b(x) = [1, x, y, z] \) if one wishes to fit a plane and \( b(x) = [1] \) if a constant. Having solved \( c \), the resulting function is

\[
f(x) = b^T(x)c.
\]

(2)

For moving least squares, the fit changes depending on where the function is evaluated, thus \( c \) varies with \( x \). To incorporate this change, a distance based weighting function \( w(x, p_i) \) is multiplied to each side of Equation 1, resulting

\[
\begin{bmatrix}
  w(x, p_1) \\
  \vdots \\
  w(x, p_N)
\end{bmatrix}
\begin{bmatrix}
  b^T(p_1) \\
  \vdots \\
  b^T(p_N)
\end{bmatrix}
\begin{bmatrix}
  \phi_1 \\
  \vdots \\
  \phi_N
\end{bmatrix}
= \begin{bmatrix}
  w(x, p_1) \\
  \vdots \\
  w(x, p_N)
\end{bmatrix},
\]

(3)

\( w(x, p_i) \) is a weighting function chosen to characterize the behavior of the interpolation. In [Shen et al. 2005], it is defined that \( w(x, p_i) = w(||x - p_i||) = w(r) = 1/(r^2 + \epsilon^2) \), where \( \epsilon \) is a parameter that controls the interpolating or approximating nature of the function.

If we rename the matrices, then Equation 3 becomes

\[
W(x)B(x)c = W(x)\phi,
\]

(4)

then \( f(x) \) can be solved as

\[
f(x) = b^T H^{-1} B^T (W(x))^2 \phi,
\]

(5)

where

\[
H = B^T (W(x))^2 B.
\]

3.2 MLS on 2-manifolds

Extending the MLS formulation to 2-manifold, and so that it interpolates for different kinds of constraints, changes must be made. We will see in the sessions below, how the formulation varies for each of the controllers that we provide to the user for field design.

3.2.1 Point value constraints

The problem of interpolating values at point constraints on the surface can be formulated similarly to what is presented in Section 3.1: for a surface \( S \), and \( N \) points positioned at \( p_i \), where \( p_i \in S \), build a function \( f(x) \), defined \( \forall x \in S \), that approximates or interpolates the constraint values \( \phi_i \) at \( p_i \). Note that Equation 3 is still valid, if we define \( w(x, p_i) \) differently, so that \( w \) becomes free from Euclidian distance evaluation. Moreover, if a constant basis function \( b(x) = [1] \) is chosen, which in practice produces faster yet similar quality results, Equation 3 is now reduced to

\[
\begin{bmatrix}
  w(x, p_1) \\
  \vdots \\
  w(x, p_N)
\end{bmatrix} c_1 = \begin{bmatrix}
  w(x, p_1) \\
  \vdots \\
  w(x, p_N)
\end{bmatrix} \begin{bmatrix}
  \phi_1 \\
  \vdots \\
  \phi_N
\end{bmatrix},
\]

(6)

which can be rewritten as

\[
f(x) = c_1 = \sum_{i=1}^{N} w(x, p_i) \phi_i / \sum_{i=1}^{N} w(x, p_i).
\]

(7)

The choice of \( w(x, p_i) \) is made such that it accommodates the surface setting. The original Euclidean distance \( ||x - p_i|| \) is now replaced by the geodesic distance between the points, which we denote by \( g(x, p_i) \). Using the most trivial \( w(x, p_i) = w(r) = 1/r \), it is clear that it does not lead to useful results, because it causes a cusp, or \( C^2 \) discontinuity, at the constraint site. This can be alleviated with the use of \( w(r) = (1/r)^\mu_i \), setting \( 1 < \mu_i \). This use of radial function in the context of data interpolation of form in Equation 7 is called Shepard’s method [Shepard 1968]. Although the radial function used in Shepard’s method is an improvement over the simplest function, it still has one major drawback - the constraints has global influence over the function. The resulting \( f(x) \) is affected by all constraints changes independent of distance that separates them. This is an undesirable characteristic from both computational and field design aspect. Besides that, Shepard’s radial function does not provide ability to control the proximity of the interpolated field to the constraints. To incorporate these two features: local influence and control on interpolation proximity, we define our weighting function as the follows, by combining the ideas from Hermite Radial function in [Nielson 2004], and weighting function in [Shen et al. 2005]:

\[
w_i(r) = \left[ \frac{\epsilon + (R_i - r)}{R_i} \right]^\mu_i, \text{where } e^+ = \begin{cases} 
  z, & \text{if } z \geq 0 \\
  0, & \text{if } z < 0 
\end{cases},
\]

and \( 0 \leq \epsilon \).

(8)

Figure 1: The resulting scalar field by fixing six value constraints on the cow model. The parameters used to generate the field were: \( \epsilon = 0; \mu_i = 2; R_i = \infty \).

From Equation 8, it is observed that \( \epsilon \) controls the interpolating or approximating choice of \( f(x) \). If \( \epsilon = 0 \), \( f(x) \) interpolates the
constraints, as
\[
\frac{w(p_i, p_j)}{\sum_j w(p_i, p_j)} = \delta_{i,j} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j. \end{cases}
\] (9)

If \( \epsilon \neq 0 \), the function approximates the constraints. \( R_i \) determines the radius of influence at the constraint point \( p_i \). When \( R_i = \infty \), the constraint at \( p_i \) affects the whole model. The parameter \( \mu_i \) controls the rate of influence decrease.

\[<Figure 2: Pipeline to evaluate the scalar field using point value constraints.>\]

Fig 1 shows an example of generated scalar field over the cow model using six constraints (one on each of its legs, one on its tale, and one on its nose). Picture on the left shows the resulting field in color scheme, while picture on the right shows the field encoded by iso-contours. Computationalwise, the process of evaluating the field works as a pipeline. Fig 2 shows the field evaluation over a 30x15 grid mesh, with two constraints (red and blue). The first stage of the pipeline is to compute the geodesic distance for each constraint. Once the distance fields are computed, they are used to scalar field evaluation as in Equation 7. The bottom row shows the result in color scheme and iso-bar. As the design is an interactive process, the user can prune the field by updating the constraints. If the position of a constraint is modified, the system recomputes its geodesic distance field and reevaluates the scalar field. If the constraint value or interpolation parameters (\( \epsilon, R_i, \mu_i \)) are changed, only the scalar field is reevaluated.

### 3.2.2 Point with gradient constraint

By setting value constraints, we are able to control the value of the field at a specific point, but not how the field flows around that point. To allow higher controlling flexibility, we introduce another type of constraint, which is the gradient of the field, defined at the tangent plane of the point.

To incorporate this additional constraint, we first reformulate the problem. Given a surface \( S \) and \( N \) points positioned at \( p_i \) (where \( p_i \in S \)), build a function \( f(x) \), defined \( \forall x \in S \), such that it approximates or interpolates the constraint values \( \phi_i \) at \( p_i \), and the gradient of \( f(x) \) is equals to or approximates (depending on \( \epsilon \)) to the constraint vector \( g_i \) (defined at the tangent plane of \( p_i \)). Following the idea presented in [Shen et al. 2005], we choose to solve this problem by adding the projection of the gradient \( g_i \) with the path that connects \( x \) and \( p_i \) to the interpolated value \( \phi_i \). This way, instead of interpolating a value, we interpolate a function \( s(x) \), which is defined as
\[
s_i(x) = \phi_i + g(x, p_i) \odot g_i.
\] (10)

The term \( g(x, p_i) \) defines a path on the surface from \( x \) to \( p_i \) and \( g(x, p_i) \odot g_i \) measures the projection of this path to \( g_i \).

Now equation 6 is expressed as
\[
\begin{bmatrix}
w(x, p_1) \\
\vdots \\
w(x, p_N)
\end{bmatrix}
= \begin{bmatrix}
w(x, p_1) \\
\vdots \\
w(x, p_N)
\end{bmatrix}
+ \begin{bmatrix}s_1(x) \\
\vdots \\
s_N(x)
\end{bmatrix},
\] (11)

which leads to
\[
f(x) = s_1 = \sum_{i=1}^{N} w(x, p_i) s_i(x)/\sum_{i=1}^{N} w(x, p_i).
\] (12)

We prove in Appendix A that the formulation above gives the desired properties that we required in the problem statement.

\[<Figure 3: (a) Parallel transport of a gradient vector at \( p_{source} \) to \( p_{target} \). (b) and (c) shows that the length of piecewise projection of the a gradient field with a path is independent of the path configuration.>\]

\[<Figure 4: We can generate very different scalar field over the mesh by varying the constraint gradient direction. The first two pictures show the result when we use a vertical gradient; while the last two pictures show the result using horizontal gradient.>\]
Figure 3.(a) illustrates how the parallel transport method works. Given two points \( p_{source} \) and \( p_{target} \), the goal is to transport the vector \( g_{source} \) at the tangent place of \( p_{source} \) to \( p_{target} \). It can be done by first finding a path that connects them, and then calculate \( g_{target} \) such that it forms a complementary angle \( \pi - \theta \) with the path in the tangent plan, as \( g_{source} \) forms angle \( \theta \) with the path. Theoretically, the choice of the path does not affect the result of projection, as shown in Figure 3. However, to minimize computation inaccuracies, we opt the shortest path or the geodesic path between the points.

Figure 5: Pipeline to evaluate the scalar field using point gradient constraints.

Figure 4 shows an example of scalar field generated over a hand model. Only one gradient constraint is used in this case. On the left, the gradient constraint is set to point upwards, thus the field flows vertically. On the right, the constraint is set to point to the right, the field flows from left to right.

In terms of implementation, the field evaluation can still been seen as a pipeline of processes, shown in Figure 5. We show the example again in a planar grid mesh. For given two gradient constraints over the mesh, we compute the geodesic distance and geodesic path for each of the constraints (the first two rows). Next, geodesic pathes are used to transport the gradient from the source all vertices of the mesh. Once the gradient field is generated, it is computed a piecewise projection of the path with the gradient field. At last, the projected distance value is used to evaluate the interpolation. As the field design is an interactive process, if the scalar value of any of the constraints is modified, only the interpolation part needs to be recomputed. If the gradient vector has changed, it is required to recompute from the geodesic distance and path for the modified constraint.

3.2.3 Iso-Contour constraints

Although useful most of the time, point-like constraints are often not expressive enough to support feature-sensitive field design. We introduce another type of constraint that provides such capability: iso-contour constraints.

Figure 6: Generating field over the Twirl model using six iso-contour constraints.

An iso-contour controller \( s_i \) is a curve on the surface of which the field value is the value \( \phi_i \). The curve can be specified by user drawing (projecting screen points on the surface), sharp edge detection algorithm, or use the iso-contour of the generated scalar field, as seen in Section 4.3. The contour can be used to enforce sharp edges or boundaries in the field generation, or simply to create motives on the surface. We show an example of that in Figure 6. Six iso-contours were used to protect the sharp edges of this twirl model. The generated field obeys the climbing of the spirals.

Figure 7: Pipeline to evaluate the scalar field from two iso-contour constraints.

As we will show in session 3.3, processing iso-contour constraint requires only a small extra amount of computation compared to point constraints. The pipeline to generate a field from those controllers is shown in Figure 7. Similarly, given two iso-contour constraint, shown in red and blue, the system computes geodesic distance fields for each constraint first. Next, the field information is used in interpolation to generate the final field. Likewise, if the constraint value \( \phi_i \) is modified, the field is interpolated again. If the shape or location of the iso-contour modifies, the geodesic fields need to be recomputed.
3.3 Approximate geodesic algorithm

As seen in the previous sections, the evaluation of the scalar field relies on the geodesic distances and geodesic paths over the mesh. Although the best choice is have exact geodesic evaluation, it is somehow impractical as the exact evaluation is overly slow. That consequently degrades the interactive experience of the field design. To assure interactivity, we choose to trade off moderate accuracy with speed. In other words, we opt to adopt an approximate geodesic algorithm. We take ideas from the algorithm approached in [Jie Tang and Zhang January, 2007], which we refer as Fast Approximate Geodesic (FAG) algorithm. We modifies the algorithm implementation, and found big gains of computational time. Besides, we modified the algorithm to support iso-contour constraints.

As it is proposed in [Mitchell et al. 1987][Surazhsky et al. 2005][Marcin Novotni 2002], the main idea of FAG algorithm is to treat the geodesic computation as a wave propagation process. From a seed source point over the polygonal model, wave structures are propagated to the neighboring geometric primitives. Surazhsky et al [Surazhsky et al. 2005] chose to represent the wave structure explicitly in a data structure which they called interval. This explicit storage and management of wave structure gives accuracy, nevertheless requires extra computational time and space. In FAG [Jie Tang and Zhang January, 2007] and [Marcin Novotni 2002], the authors choose to simplify the structures so that they are stored implicitly and approximated as distance values at vertices.

![Figure 8: Three possible configurations in a propagation step.](image)

The FAG algorithm performs as Dijkstra’s algorithm. It starts with an initial set of active-front vertices, then always choose to propagate the unvisited adjacent edge of the vertex $v$, where $v$ has the least distance value among the active-front vertices.

The propagation rule is such that, if we wish to propagate the edge $(v_i, v_j)$, we first map the triangle $(v_i, v_j, v_k)$, where $v_k$ is the target vertex to where we wish to propagate, isometrically to a 2D coordinate system in which $v_i$ lies in the origin, and $v_j$ lies in the $x$-axis. From the converted coordinates of $v_i$, $v_j$, and the geodesic distance values at them, we are able to retrieve the approximate location of the pseudo propagation source $v_s$ by:

$$
v_s = \left[ \frac{d_j^2 - d_i^2}{2 d_j} \right] \pm \sqrt{d_i^2 - v_{s_2}} \quad (13)
$$

where $v_{s_2}$ is the $x$ component of coordinate of $v_j$, $d_i$ and $d_j$ are the distance values at $v_i$ and $v_j$, respectively. The relationship of $v_s$ and $v_k$ can be in three situations as illustrated in Figure 8. Depending on the configuration, the distance value at $v_k$ can be calculated:

- In the first case (a), $v_s$ and $v_k$ are visible to each other, thus, the distance value at $v_k$ is $||v_s - v_k||$.
- In the second case (b), $v_j$ is obscured from the left, thus the distance at $v_k$ is $d_i + ||v_i - v_j||$.
- In the last case, the distance at $v_k$ is $d_j + ||v_1 - v_j||$.

![Figure 9: (a) shows the virtual segment source, and the distances used to retrieve its end points. (b) shows that once this segment is retrieved, the geodesic distance at $v_k$ can easily computed.](image)

While the FAG authors chose to implement the algorithm based on vertex-based operations, we chose to implement in edge-based primitives. That is, we chose to use half-edge data structure, and instead of keeping track of active-front vertices, we keep track of active-front edges. This saves times at each step of the propagation, as our approach can pop up directly the edge, instead of query the adjacent edges in the case of FAG. With this slight modification, we notice big computational gains. The FAG algorithm runs 5:00s over a mesh of 20K vertices, in a 2.4GHz, with 512MB of RAM machine. Our program runs 1:18s over a mesh with the same number of vertices, under the same machine configuration.

To support iso-contour constraint (section 3.2.3), we need to compute the geodesic distance field of any point over the surface to a constraint iso-contour. This makes the source of the geodesic propagation no longer a point, but a curve over the surface. We approximate the curve with edges of the polygonal model. If the input curve consists of polylines that crosses faces of the model, the polylines are snapped to the closest edges.

The algorithm operates in a similar way as the point source algorithm that we described above. The edge $(v_i, v_j)$ that contains the smallest distance value is popped off from a priority structure. $(v_i, v_j)$ is mapped to a 2D domain together with the target vertex $v_k$ as illustrated in Figure 8. As the propagation source is now a curve broken into segments, at the beginning, the propagation source of all primitives is a single segment. Thus, we approximate the source with a virtual segment, of which the coordinates of its end-vertices can be retrieved by using the distances $d_1, d_{i2}, d_{j1}, d_{j2}$ store previously in an interval structure at $(v_i, v_j)$. This is illustrated in Figure 9.(a). Once the source segment is found, the distance at $v_k$ is the closest distance of $v_k$ to the the segment, as depicted in Figure 9.(b). Besides updating the distance value, we create/update intervals at edges $(v_i, v_k)$ and $(v_k, v_j)$ so that they store their respective distances to the end-vertices of the source, for the next propagation step.

For efficiency reason, we collapse the virtual segment source when its length becomes $1/10$ of the minimum distance value at the propagation edge. From there on, the algorithm will perform the same way that point source propagations, described previously.

4 Scalar field properties and applications

In Euclidian setting, where the space is ever expanding, fields generated by distance based interpolation are $C^1$ continuous. Exception may occur only at the constraint point. For a 2D surface embedded in 3D, if the surface is ever expanding, this property is observed likewise. Nevertheless, in most cases the waves of the geodesic propagation meet themselves at some frontier. This phenomenon leads to $C^1$ discontinuity in the resulting field, which can
be observed in Figure 10 (a). Furthermore, recall that the projected distance field relies on geodesic path to transport the gradients. It is noticed that the paths (thus the transported gradient) diverge in where the distance field meet itself (shown in Figure 10.b). This phenomena leads to $C^0$ discontinuities when the projected distance of the path to the gradient field is calculated, as shown in Figure 10.c.

Although this characteristic invalidates the field from being used in application such as surface remeshing, where global smoothness is desired, it does not prevent the field from being used for more localized mesh operators. In the next sections, we demonstrate the usability of the field in three mesh applications.

4.1 Skin deformation

By generating a scalar field over the mesh, we are essentially parametrizing the model into a 1D domain. Borrowing ideas from generalized cylinder construction, we can build profiles to guide the deformation of the mesh skin. Figure 11 shows three examples of scalar fields generated over the torus model. The top row of the picture shows the scalar field and the controllers used to generate it. The bottom line shows the resulting shapes after being deformed according to specified profiles.

4.2 Region selection

In many circumstances, mesh deformation is only intended to affect the model partially. In these cases, smooth transition must be made between the active region and inactive part of the mesh. Our scalar field framework is very convenient to define such regions. One can first design a field and then set iso-value thresholds that divide the surfaces into different regions. We demonstrate this capability with the example in Figure 13. We use six value constraints to build a scalar field over the Nicolo model. Then we chose iso-values 0.31 and 0.65 to be the region delimiters. The resulting transition region, color-encoded in green, is shown in Figure 13.(b). Finally, Figure 13.(c) shows the effect after deformation.

4.3 Curve drawing

Another natural application of our method is to draw curves on the surface. Iso-contours are natural curves defined on the surface. We provide a tool so that the user can trim the curve, to make it an open curve. Figure 14 shows three examples of drawing curves on surfaces. The vase lion model has two sets of curves on it. Its glasses are drawn using two close curves and three open curves. The bow tie is drawing using a single close curve.

5 Results

Table on Figure 15 lists the computational time needed to generate the examples scattered across the paper. They are all generated on a Intel Core2 Duo T7200, 2.0GHz, with 1.5GB of RAM laptop. The columns of the table are listed in the following order: 1) the model name, 2) mesh information (number of vertices, number of faces), 3) number of value constraints, 4) number of gradient constraints, 5) the coverage type (whether the constraints influence the entire mesh, or only partly), 6) average time spent on evaluating geodesic distance for each constraint, 7) time spent on evaluate projected distance for each gradient constraint, 8) total time spent.

6 Conclusion

In this paper, we have presented and analyzed an innovative framework to directly generate scalar fields over 2-manifolds. Our method takes constraints in various forms, which provides the user
the ability to control 1) the value of the interpolating field at points; 2) the gradient flow at points; and finally 3) the iso-contours of the field. Furthermore, it is provided to the user system parameters that control the range of influence, and the rate of decrease of the constraints.

In the aspect of generating field from the constraints, we have contributed with the idea of using geodesic evaluation for weighting the basic functions in the MLS formulation, and using the geodesic paths for parallel transport, required to support point gradient constraint. In order to support iso-contour constraint, we also propose a geodesic algorithm that evaluates distance field from 1-dimensional sources without much extra computation.

At the end of the paper, we demonstrated with several examples that uses the resulting field in localized meshing editing operations, such as skin deformation, selection and curve drawing.

As the future extension, we seek improvements in computational efficiency. The bottleneck of the current system is in tracing geodesic paths, which is used in the gradient constraint part. We believe that because the paths are independent of each other, with the use of GPU, parallel acceleration techniques, the program can be made much faster. Another research direction to take is to further investigate the possible applications of the scalar field, and to introduce other intuitive controllers for the field design.

Appendix A

Proof: First we set \( \epsilon = 0 \), so that \( f(x) \) interpolates the constraints. Then we rewrite Equation 12 as

\[ f(x) = \sum_{i=1}^{N} \rho_i(x) s_i(x) \]  

(14)

where

\[ \rho_i(x) = \frac{w(x, p_i)}{\sum_j w(x, p_j)}. \]

Since \( gp(p_i, p_j) = 0 \), which consequently makes \( s_i(p_i) = \phi_i \), and \( \rho_i(p_i) = \delta_{i,j} \) (as shown in Equation 9), it can be verified that \( f(p_i) = \phi_i \).

In order to verify that the function obeys the constraint gradient, \( \nabla f(x) = g_i \) needs to be met. We apply product rule in Equation 14 to obtain

\[ \nabla f(x) = \sum_{i=1}^{N} \rho_i(x) \nabla s_i(x) + \nabla \rho_i(x) s_i(x). \]

Since \( \rho_i(p_i) = \delta_{i,j} \), we get \( \nabla \rho_i(p_i) = 0 \). Also, it is verified that \( \nabla (\phi_i + gp(x, p_i) \odot g_i) = g_i \), as \( gp(x, p_i) = x - p_i \) and \( (x - p_i) \odot g_i = (x - p_i)^T g_i \), in planar meshes. Combining the above pieces and the fact that \( \nabla ((x - p_i)^T g_i) = g_i \), we conclude that

\[ \nabla f(x) = g_i. \]

This completes the proof. ♦

References


<table>
<thead>
<tr>
<th>Model</th>
<th>$#v$, $#f$</th>
<th>$#vc$</th>
<th>$#ge$</th>
<th>$#ic$</th>
<th>Cover</th>
<th>Dist/ctrl</th>
<th>Grad/ctrl</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>cow (Fig 1)</td>
<td>3k, 6k</td>
<td>6</td>
<td></td>
<td></td>
<td>full</td>
<td>0.09</td>
<td>0</td>
<td>0.6</td>
</tr>
<tr>
<td>hand (Fig 5)</td>
<td>27k, 53k</td>
<td>1</td>
<td></td>
<td></td>
<td>full</td>
<td>15.8</td>
<td>15.9</td>
<td></td>
</tr>
<tr>
<td>twirl (Fig 7)</td>
<td>122k, 244k</td>
<td>6</td>
<td></td>
<td></td>
<td>full</td>
<td>6.50</td>
<td>0</td>
<td>41.18</td>
</tr>
<tr>
<td>nico (Fig 13)</td>
<td>25k, 50k</td>
<td>6</td>
<td></td>
<td></td>
<td>partial</td>
<td>1.15</td>
<td>0</td>
<td>7.06</td>
</tr>
</tbody>
</table>

Figure 15: Computational cost of examples that are scattered across the paper. Timing is measured in units of seconds.