DiffusionMesh

Ming-Yee Iu*
Wobastic Software Inc

ABSTRACT

The DiffusionMesh system incorporates an advanced gradient system that is inspired by diffusion curves with a traditional vector graphics tool. Shapes in DiffusionMesh are defined using closed paths, like in traditional vector drawing tools, but DiffusionMesh extends this model in several ways: different colors can be assigned to the nodes of closed paths, closed paths can be joined together into arbitrary meshes, a C^1 continuity of colors is maintained across seams between the different shapes of a mesh, and each shape can constrain which colors are allowed inside the shape. Since shapes are defined using closed paths, they can be easily composited into more complex drawings. DiffusionMesh renders these shapes by decomposing the shapes into triangle meshes and diffusing colors across the mesh to create a smooth gradient. During changes to the drawing, DiffusionMesh only needs to recompute the gradient for the affected shapes instead of solving a global optimization for every change. The use of triangle meshes for rendering means that drawings can be scrolled and zoomed without needing to recompute the gradient. Although rendering is too slow for real-time animation, it is still fast enough for interactive use.

Keywords: Vector graphics, gradient meshes, diffusion curves.

Index Terms: Computing methodologies—Computer graphics—Rendering—Rasterization;

1 INTRODUCTION

Vector graphics drawing tools are popular because they create art that can be manipulated, resized, and animated without a loss of quality. These tools have traditionally used a vector graphics model where shapes are represented as closed paths, which can be filled with colors and have strokes applied to their outlines [7]. In the last few years, there has been considerable excitement around new vector graphics models derived from diffusion curves [10]. With these models, drawings are produced by drawing curves on a canvas. These curves have color constraints defined at points along each side of each curve. These colors are then diffused throughout the canvas to create a rendered image. The diffusion curve approach to vector graphics is particularly well-suited for creating complex, subtle shading.

DiffusionMesh[1] is a vector drawing system that demonstrates how advanced gradients with capabilities similar to diffusion curves can be integrated into a vector drawing tool based on a traditional vector graphics model. DiffusionMesh supports the traditional closed path representation of shapes but extends it to give artists more control over how colors are blended across shapes. Artists can use a familiar vector graphics workflow with layers, groups, paths, and fills, but they also have enough control over gradients that they can draw subtle shading and soft shadows.

The main contributions of DiffusionMesh are

- a gradient model that adapts diffusion curve gradients for use with shapes defined by closed paths that are typical of traditional vector graphics tools. Notably, the model defines extra constraints on the generated gradient that are specific to closed path shapes
- a method for calculating a gradient that satisfies the above gradient model
- a functioning vector drawing tool based on these gradients that offers insights into system architecture issues and into the practical experience of using such gradients

This paper describes a model for representing advanced gradients using closed path shapes, presents a method for rendering a gradient from this representation, and evaluates the feasibility of using this model in interactive vector drawing tools.

2 GRADIENT MODEL

To support advanced gradients over closed path shapes, DiffusionMesh requires its own model for vector objects and gradients. The gradient model of DiffusionMesh is defined in terms of

1. how the underlying geometry and gradient parameters are specified
2. the properties of the gradient that is generated to fit the geometry and parameters

2.1 How Gradients Are Specified

Similar to traditional vector drawing tools, DiffusionMesh uses closed paths as the primary primitive for representing objects in drawings. These closed paths are made up of a series of nodes that are joined together by lines and curves, forming an outline for a shape. This shape can be filled by solid colors or patterns. Since objects are self-contained, they can be rendered independently and stacked in layers.

DiffusionMesh extends this model by allowing colors to be assigned to the nodes of closed paths. When a shape has different colors assigned to its nodes, DiffusionMesh will generate a gradient over the surface of the shape using those colors. When a shape has the same color assigned to all of its nodes, DiffusionMesh will fill the shape entirely with that color as a solid fill.

Allowing artists to assign colors to the nodes of a closed path only gives them the ability to directly control the colors along that path. To assign specific colors to areas on the inside of an object, artists must use multiple shapes to model an object, so that a path exists in the area where they want to assign colors. To facilitate this, DiffusionMesh allows closed paths to be joined together into complex meshes (Fig. 1). Any closed path can share any of its edges with any other closed path, allowing for meshes made up of complex shapes that are joined together in arbitrary ways.

For example, an artist cannot assign a color on the inside of a simple circle shape because there are no paths or nodes in the center of the circle where a color can be assigned. Instead, the circle can be modeled using two half-circles that are joined together into a circle mesh. Then, a node can be added to the middle of the half-circles and a color assigned to it, resulting in DiffusionMesh rendering an approximation of a radial gradient for that circle (Fig. 2).

*e-mail: diffusionmesh@user00.com

[1] A patent application has been filed for a system incorporating some of the methods described in this paper

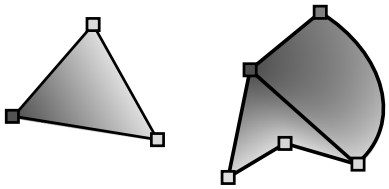


Figure 1: Different colors can be assigned to the nodes of a closed path, and multiple closed paths can be joined together into meshes.

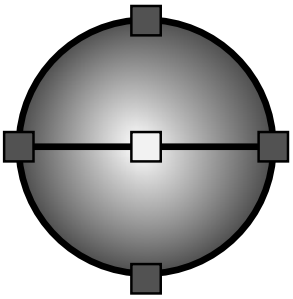
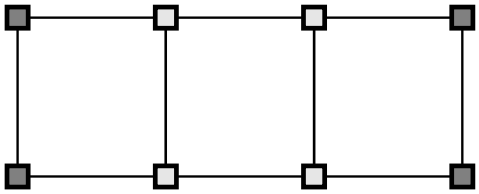
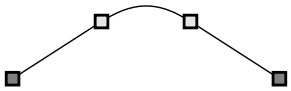


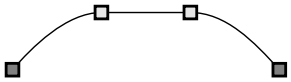
Figure 2: Radial-style gradients can be defined by splitting a shape in half and assigning a different color at the center.



(a) All the nodes of the center rectangle in the mesh have the same color, so the center rectangle should be a single color



(b) Naively trying to maintain a C^1 continuity of colors on the sides of the center rectangle will cause the color in the center to fall outside that range of colors



(c) DiffusionMesh requires a solution similar to monotone spline interpolation used for 1d functions to force colors within the center rectangle to be a single color

Figure 3: DiffusionMesh enforces the restriction that the colors inside a closed path shape must be within the range of colors defined at the nodes of the path making up the shape. Fig. 3a shows three closed path shapes forming a mesh. Fig. 3b and Fig. 3c show a horizontal cross-section of possible color values for the mesh.

2.2 Gradient Properties

Once the mesh geometry and node colors have been specified, DiffusionMesh can generate a gradient that satisfies that input. DiffusionMesh generates gradients that satisfy certain properties, allowing artists to anticipate what the gradient will look like and allowing them to control the resulting gradient.

The first property is that the gradient is constrained so that at a node, the gradient will have the same color as that node. Along the edge of a path, the gradient is constrained to be a linear interpolation of the colors assigned to the nodes on either end of the edge. With these constraints, artists have full control over the colors along the outside boundary of a closed path. They can assign any color to nodes, constraining the color of the gradient in that portion of the path. They can add more nodes to a path if they need more control over colors.

Within a closed path, DiffusionMesh uses a diffusion process to blend the colors of the nodes across the surface of the shape. To ensure that the gradient is smooth, DiffusionMesh attempts to minimize the change in color values across the surface.

In fact, DiffusionMesh imposes a much more restrictive constraint on the smoothness of the gradient. When shapes are joined together into a mesh, it is desirable for the generated gradient to appear smooth across the whole surface of the mesh. To ensure this, DiffusionMesh imposes a C^1 continuity of colors for gradients across the whole mesh. In particular, the color derivatives at shared edges between shapes must match. Although each shape already imposes the constraint that gradients near an edge must be a linear interpolation of the colors assigned to nodes on either end of the edge, that constraint only guarantees that the colors at shared edges of adjacent shapes are the same. Without the color derivatives also being the same, seams or creases will appear in the mesh wherever two shapes join.

The above properties define a PDE with fixed values at various points along its surface. DiffusionMesh defines an additional property for the gradient though. In DiffusionMesh, the gradient inside any shape of a mesh will only be an interpolation of the colors defined on the nodes of that shape. This provides artists some direct control over which colors will appear in the individual shapes of their drawings. In particular, if all the colors defined on the nodes of a shape are the same color, then the gradient within that shape will just be that single color. As a result of this property, there is a tension between mesh-wide smoothness constraints and the local-to-each-shape restrictions on color values (Fig. 3).

DiffusionMesh does not provide any explicit control over color derivatives. Although allowing control over color derivative values could potentially allow artists to fine-tune their gradients, it is difficult to expose such controls in a user interface.

3 Rendering

Given meshes of closed path shapes with colors assigned to their nodes, DiffusionMesh can generate a gradient for the shapes and render the shapes to the screen. DiffusionMesh renders the shapes by converting them to Gouraud shaded triangles that can be rasterized efficiently using modern graphics hardware. Converting shapes into triangles also has the advantage that the triangles can be scaled and translated for display entirely in graphics hardware. DiffusionMesh uses an ad hoc method for rendering that yields visually satisfying results. Rendering is divided into two stages: triangulation, where the mesh of shapes is broken down into a triangle mesh, and diffusion, where a gradient is constructed over this mesh.

3.1 Triangulation

DiffusionMesh tries to convert shapes into a triangle mesh of roughly equal-sized, uniformly spaced triangles. This allows DiffusionMesh to use simpler, discrete methods for calculating the diffusion of colors across these triangles. A standard constrained Delaunay triangulation algorithm can be used, but, for convenience, DiffusionMesh uses its own approach for generating a triangulation (Fig. 4).

Initially, the shapes are presented to the DiffusionMesh renderer in a mesh data structure. Each shape is made up of a closed path of nodes connected by straight or curved edges. Each node has an assigned color, and edges can be shared between two shapes. DiffusionMesh first converts all curves into a set of straight lines using standard curve subdivision techniques. The new nodes in-between these lines are assigned colors by linearly interpolating the colors of the nodes of the original curve. At this point, each shape has been converted into a polygon.

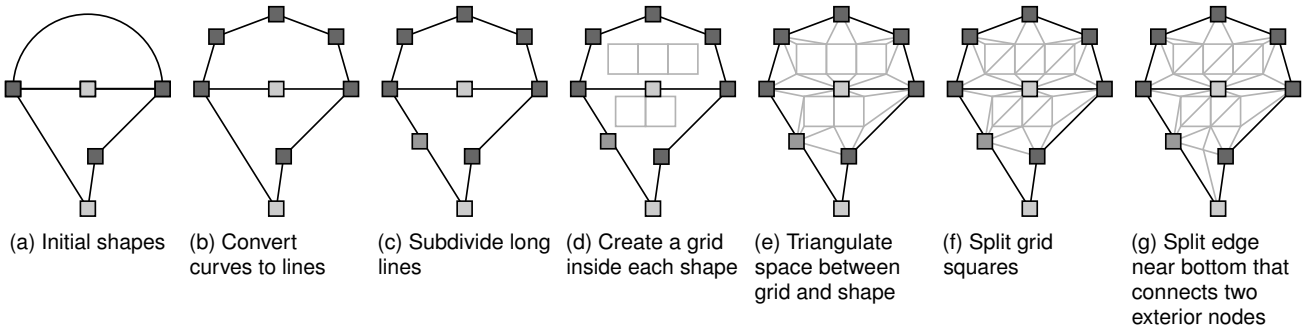


Figure 4: Steps for creating a triangle mesh

Then for each polygon, DiffusionMesh chooses a desired triangle size for the triangle mesh that the polygon will be broken into. This desired triangle size is defined in terms of an approximate length for the edges of the triangles. If any edges of a polygon are longer than the desired triangle edge length, then the edges are subdivided until they are shorter than the desired triangle edge length. The new nodes created by the subdivision are assigned colors based on the linear interpolation of the colors from the original nodes for the edge.

DiffusionMesh uses a plane-sweep-style algorithm to generate a grid of squares inside each polygon. The length of sides of the squares in the grid is the same as the desired triangle edge length. A standard sweep-line algorithm for polygon triangulation is used to triangulate the space between the grid and the boundary of the polygon. Successive rounds of Delaunay edge flipping are applied to this triangulation until all poorly proportioned triangles are removed. Then, each square in the grid is divided in half into two triangles. Each polygon will then have been subdivided into a mesh of roughly uniform triangles.

Finally, DiffusionMesh alters the mesh by identifying all edges interior to a polygon that join nodes from the border of the polygon. These interior edges are split, adding two new triangles to the mesh. This splitting ensures that nodes on the outline of a polygon are never directly connected to each other, so there will always be an interior node with a color that DiffusionMesh can adjust to optimize the generated gradient for the polygon.

3.2 Diffusion

With the triangle mesh generated, DiffusionMesh can now assign colors to the interior nodes of each polygon using a diffusion process. Since each polygon has been triangulated into a mesh of triangles that are roughly uniformly spaced and equal-sized, DiffusionMesh is able to use discrete methods for its diffusion. A smooth gradient should have a C^1 continuity of colors across its surface. To solve these constraints over a mesh of discrete triangles, DiffusionMesh assumes that colors are linearly interpolated across the surface of each triangle, meaning that the x and y color derivatives are uniform over the surface of each triangle. Then, DiffusionMesh only has to minimize the differences in the color derivatives of all adjacent triangles.

For each interior node, DiffusionMesh looks at the triangles in the neighborhood of the node. It wants to set the color of the node so that the differences between the colors derivatives of all the neighboring triangles are minimized (Fig. 5). It does this by looking at all edges adjacent to the node and all edges opposite to the node. For those edges, it wants to minimize the square of the differences in the x and y color derivatives of the triangles on either side of the edges. Differentiating this value by the color of the node and setting the result to zero yields a linear equation.

After generating equations for every interior node, DiffusionMesh will have a sparse system of linear equations. Solving this system using standard techniques like successive over-relaxation will yield a smooth gradient over the entire surface of the mesh that fails to satisfy the constraint that the gradient for each shape must only use the colors of the nodes from that shape (Fig. 3).

In order to satisfy these local constraints, DiffusionMesh modifies its successive over-relaxation so that every time it computes a new color value for a node, it will also clamp that color value to be within the range of colors defined by its neighboring nodes. As a result, the successive over-relaxation will no longer converge to a true solution to the system of linear equations but will instead converge to a solution where no node can have extreme colors that are outside the range of its neighbors. DiffusionMesh then renders this triangle mesh and its colors to the screen.

Since the color diffusion process can be slow to converge, DiffusionMesh uses hierarchical refinement to acclerate the calculations. Diffusion is initially performed over a very coarse triangle mesh. DiffusionMesh then creates successively more detailed triangle meshes and computes gradients for them by using the previous gradient solution as a starting point in the successive over-relaxation.

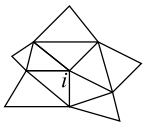
4 Performance

The performance of DiffusionMesh has been evaluated by taking a Java implementation of the system and measuring the time it needs to generate gradients for a simple drawing. The drawing is made up of three squares arranged in a row, with the edges shared between them (Fig. 6). The nodes of the squares are all assigned random colors between 0 and 1. The squares have a side length of 1000 units, and the time needed to create gradients using a triangle mesh at different levels of coarseness is evaluated. For each drawing, DiffusionMesh first generates a gradient using a triangle mesh with an approximate triangle edge length of 200, then iteratively generates more refined meshes by halving this triangle edge length until it reaches the desired refinement level. The diffusion stage is run until no colors at any of the nodes changes by more than 0.001.

The time needed to generate a gradient for 20 drawings was measured. The benchmark was repeated 70 times, with the first 20 values discarded in order to remove just-in-time compilation times from the results. The code was executed using Java8u11 on a Haswell Core i5-3470s CPU that runs at 2.9GHz-3.6GHz.

Table 1 summarizes the benchmark results. The Setup Time shows the average time needed to run the benchmark with all the color diffusion code being skipped. The Setup Time primarily measures the time needed to convert drawings into triangle meshes. The Full Time shows the average time needed to run the full benchmark including the diffusion of colors across the triangle mesh.

As can be seen from the performance results, the current gradient diffusion code is too slow for real-time animation. The generation of high quality gradients is also too slow for interactive use, but coarse



$$\text{minimize} \sum_{\text{adjacent edges}} \text{square of x and y gradient differences} + \sum_{\text{opposite edges}} \text{square of x and y gradient differences}$$

$$\therefore 0 = \sum_{\text{adjacent edges}} \frac{\partial(\text{square of x and y gradient differences})}{\partial(\text{color})} + \sum_{\text{opposite edges}} \frac{\partial(\text{square of x and y gradient differences})}{\partial(\text{color})}$$

Expression for edge *i*-*h* adjacent to a point *i*:

$$\left(\frac{y_g-y_h}{A_1}-\frac{y_h-y_m}{A_3}\right)\left(\frac{C_g}{2A_1}(y_h-y_i)+\frac{C_h}{2A_1}(y_i-y_g)+\frac{C_i}{2A_1}(y_g-y_h)-\frac{C_m}{2A_3}(y_i-y_h)-\frac{C_i}{2A_3}(y_h-y_m)-\frac{C_h}{2A_3}(y_m-y_i)\right)$$
$$+\left(\frac{x_h-x_g}{A_1}-\frac{x_m-x_h}{A_3}\right)\left(\frac{C_g}{2A_1}(x_i-x_h)+\frac{C_h}{2A_1}(x_h-x_g)+\frac{C_i}{2A_1}(x_h-x_g)-\frac{C_m}{2A_3}(x_h-x_i)-\frac{C_i}{2A_3}(x_m-x_h)-\frac{C_h}{2A_3}(x_i-x_m)\right)$$

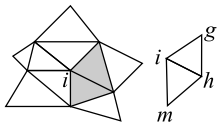
where $A_1 = \frac{1}{2}\left(x_i(y_g-y_h)+x_g(y_h-y_i)+x_h(y_i-y_g)\right)$

$A_3 = \frac{1}{2}\left(x_g(y_h-y_m)+x_h(y_m-y_i)+x_m(y_i-y_h)\right)$

$C_p =$ color at point p

$x_p =$ x-coordinate of point p

$y_p =$ y-coordinate of point p



Expression for edge *g*-*h* opposite to a point *i*:

$$\frac{y_g-y_h}{A_1}\left(\frac{C_i}{2A_1}(y_g-y_h)+\frac{C_g}{2A_1}(y_h-y_i)+\frac{C_h}{2A_1}(y_i-y_g)-\frac{C_k}{2A_2}(y_h-y_g)-\frac{C_h}{2A_2}(y_g-y_k)-\frac{C_g}{2A_2}(y_k-y_h)\right)$$
$$+\frac{x_h-x_g}{A_1}\left(\frac{C_i}{2A_1}(x_h-x_g)+\frac{C_g}{2A_1}(x_i-x_h)+\frac{C_h}{2A_1}(x_g-x_i)-\frac{C_k}{2A_2}(x_g-x_h)-\frac{C_h}{2A_2}(x_k-x_g)-\frac{C_g}{2A_2}(x_h-x_k)\right)$$

where $A_1 = \frac{1}{2}\left(x_i(y_g-y_h)+x_g(y_h-y_i)+x_h(y_i-y_g)\right)$

$A_2 = \frac{1}{2}\left(x_g(y_k-y_h)+x_k(y_h-y_g)+x_h(y_g-y_k)\right)$

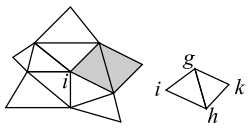


Figure 5: The color of each point *i* in the triangle mesh should minimize the differences in the derivative gradients of neighboring triangles

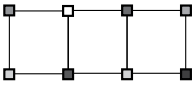


Figure 6: Simple drawing used to benchmark DiffusionMesh

Table 1: Benchmark results

Triangle Edge Length	Number of Triangles	Setup Time	Full Time
200	162	17.6 ms	43.4 ms
200 then 100	612	48.7 ms	107.5 ms
200 then 100 then 50	2412	218.8 ms	380.5 ms

gradients can be generated at adequate speeds. Approximately half the execution time is used for creating the triangle mesh while the other half is used for diffusion.

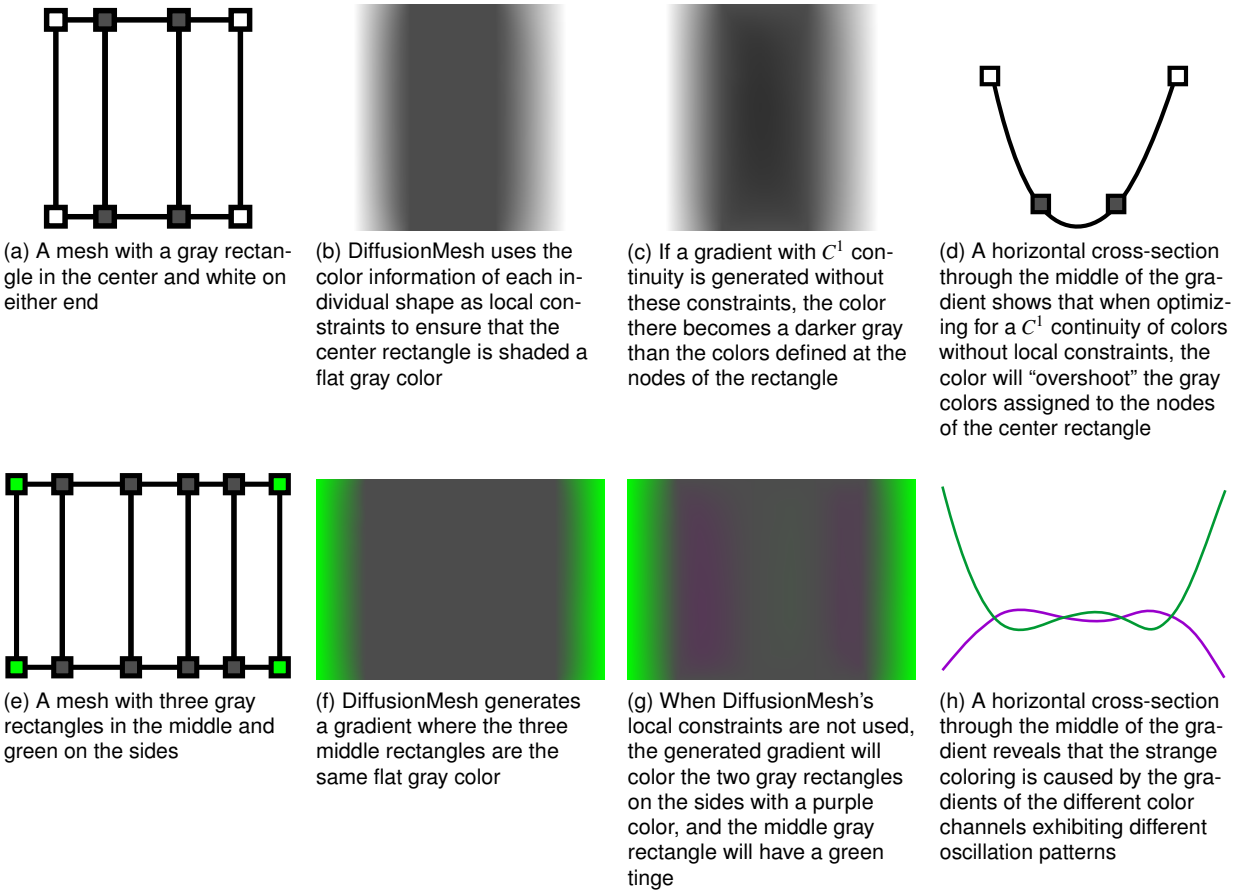
Further optimization is possible, especially by taking advantage of parallelism. For example, each shape can be triangulated independently in parallel, and the diffusion code for each point can also be run in parallel.

Nevertheless, DiffusionMesh is able to build an interactive drawing system without these optimizations. One technique it uses is to hide the latency of gradient diffusion through the use of progressive refinement. When an artist is creating gradients interactively, DiffusionMesh will initially render shapes with no gradient diffusion. It will then compute a gradient for the shapes in a separate background thread and update the screen as new gradients are calculated. The DiffusionMesh algorithm already generates a gradient over a coarse triangle mesh initially and then progressively derives finer triangle meshes to generate more detailed gradients. DiffusionMesh displays these intermediate triangle meshes as they are generated, so as to provide artists with a rough visualization of the gradient that they can work with interactively while a higher quality gradient is computed in the background.

Once DiffusionMesh computes a gradient, it does not need to recompute it if an artist is scrolling or zooming the drawing. If an artist makes a change to a shape, DiffusionMesh does not need to recalculate the gradients for the whole drawing but only for the affected meshes.

5 Results and Discussion

This section examines the characteristics of gradients generated by DiffusionMesh and discusses the experience of using DiffusionMesh in vector drawings.



(a) A mesh with a gray rectangle in the center and white on either end

(b) DiffusionMesh uses the color information of each individual shape as local constraints to ensure that the center rectangle is shaded a flat gray color

(c) If a gradient with C^1 continuity is generated without these constraints, the color there becomes a darker gray than the colors defined at the nodes of the rectangle

(d) A horizontal cross-section through the middle of the gradient shows that when optimizing for a C^1 continuity of colors without local constraints, the color will "overshoot" the gray colors assigned to the nodes of the center rectangle

(e) A mesh with three gray rectangles in the middle and green on the sides

(f) DiffusionMesh generates a gradient where the three middle rectangles are the same flat gray color

(g) When DiffusionMesh's local constraints are not used, the generated gradient will color the two gray rectangles on the sides with a purple color, and the middle gray rectangle will have a green tinge

(h) A horizontal cross-section through the middle of the gradient reveals that the strange coloring is caused by the gradients of the different color channels exhibiting different oscillation patterns

Figure 7: The colors defined on the nodes of each shape act as local constraints on what colors are allowed within the shape. These constraints give artists some control over the colors inside the individual shapes of a mesh.

5.1 Effect of Local Color Constraints for Mesh Shapes

DiffusionMesh constrains the colors of gradients within each shape of a mesh to be an interpolation of the colors assigned to the nodes of the shape. Fig. 7 illustrates the benefits of incorporating these local constraints into the diffusion process. The local constraints provide artists with some control over the colors that will appear in their meshes. Without those constraints, artists can only control the colors on the boundaries of shapes. The gradient algorithm may generate unexpected colors for the interior of shapes while trying to satisfy the global smoothness constraints of the mesh. In extreme cases, the gradient solution might even exhibit oscillation artifacts. And in color drawings, where the gradient for each color channel is computed separately, the lack of local color constraints might result in spurious colors being introduced into a drawing.

5.2 Drawing Experience

DiffusionMesh is included in a vector drawing app called Omber under the name Wrinkle-Free Gradient Engine. The app provides a user interface for manipulating meshes that resembles that of 3D polygonal modeling tools. Shapes can be drawn, welded together, or split to create meshes of different topologies. The app is a JavaScript program. The Java code of DiffusionMesh is compiled to JavaScript using the GWT Java-to-JavaScript transpiler. Due to the slowness of JavaScript when performing heavy calculations, portions of DiffusionMesh are replaced with WebAssembly code written in C++ in order to achieve performance similar to the original Java code.

As implemented in Omber, DiffusionMesh uses heuristics to determine when to stop progressively refining a gradient. A shape in a mesh will be retriangulated with a more dense triangulation if the color value of any triangle point changes by more than 0.0117 from the same location in the previous triangulation (for color values ranging from 0–1), if more than 1000 grid points are generated for the shape during the triangulation, or if the grid spacing used in the triangulation drops below 1/72 inch in size.

Various bitmap images were gathered, and vector drawings of them were made by hand. Those drawings were rendered using DiffusionMesh running on the web browser Chrome v73 using a Haswell Core i5-3740s machine that runs at 2.9GHz-3.6GHz. For each drawing, the time needed to render an initial image that can be interacted with and the time needed to fully calculate the gradient were measured. Twelve measurements were taken, and the average of the last five measurements are shown in Table 2.

Although it can take a fair amount of time for a drawing to render in full, DiffusionMesh is still suitable for interactive use due to its use of progressive refinement and because it only needs to recalculate gradients for shapes when they are changed.

There is a rare visual artifact that is occasionally noticeable when working with DiffusionMesh. Currently, the colors along an edge of a shape are a linear interpolation of the colors of nodes on either end of the edge; whereas, the colors inside a shape are interpolated in order to maximize smoothness. This difference in interpolation sometimes results in the edges of shapes within a mesh being noticeable (Fig. 14). The use of linear interpolation along shape edges is intended to give artists full control over colors at the boundaries of

Table 2: Rendering times for some drawings

Drawing	Shape Count	Initial Render Time	Final Triangle Count	Full Gradient Time
Box (Fig. 8)	8	6ms	199	240ms
Hand (Fig. 9)	22	84ms	28816	2682ms
Butterfly (Fig. 10)	40	112ms	40545	3202ms
Apple (Fig. 11)	15	42ms	12189	1440ms
Dog (Fig. 12)	48	90ms	31589	3381ms
Face (Fig. 13)	227	270ms	173559	11236ms

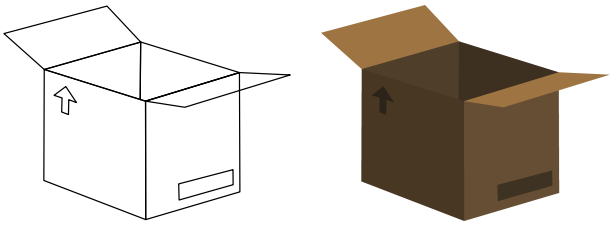


Figure 8: DiffusionMesh's vector graphics model can handle traditional flat-shaded vector graphics

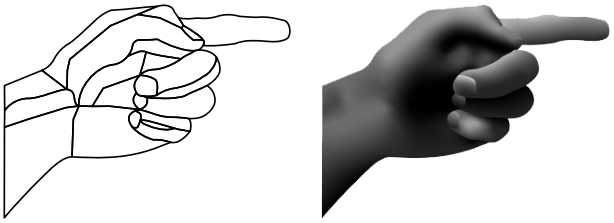


Figure 9: A small number of shapes is sufficient to specify the shading of a hand

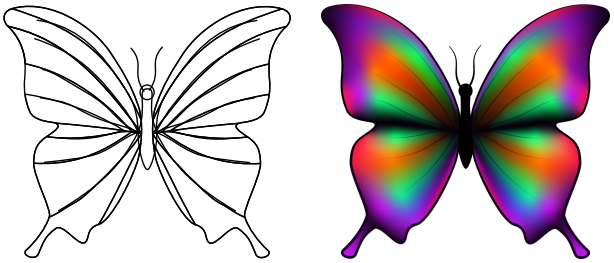


Figure 10: A drawing of a butterfly with a gradient involving many different colors

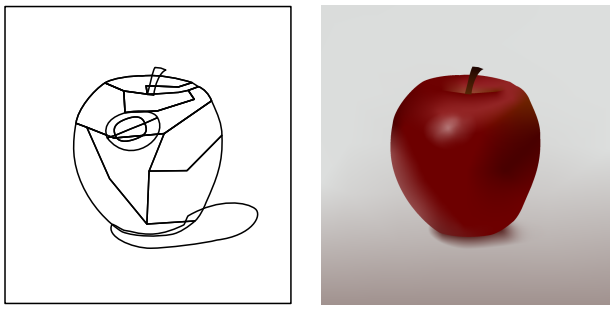


Figure 11: A drawing of an apple with gradients showing some subtle shading

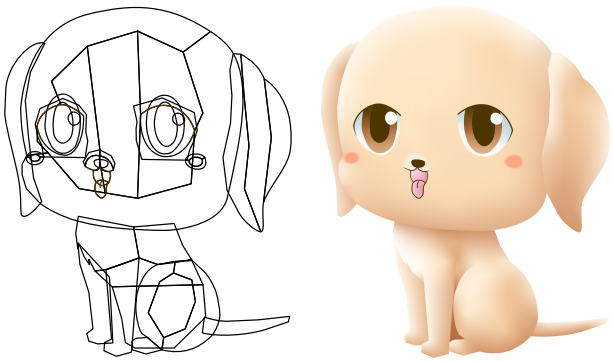


Figure 12: Drawings can be made with multiple overlapping meshes. A mesh can even overlap itself.

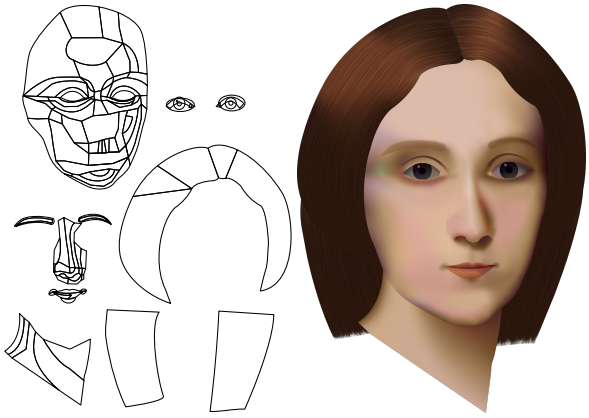
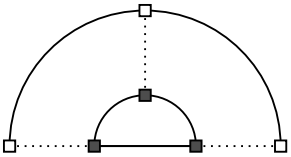
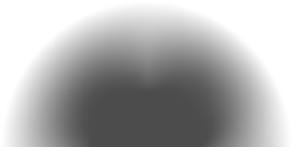


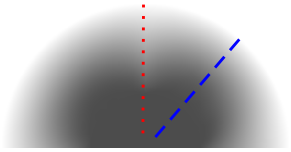
Figure 13: In this drawing, different parts of the body are modeled on separate layers and are alpha composited together based on a gradient of transparency values. The hair is made up of a hair texture that is mapped using a gradient of UV coordinates.



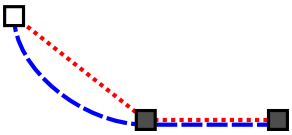
(a) In this mesh, DiffusionMesh will impose a linear interpolation of colors over the dotted edges



(b) The linear interpolation of colors at those edges might not match the smooth interpolation that is calculated for the gradients, potentially producing a visual "seam" at those edges

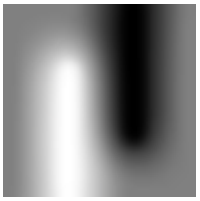


(c) The reason for this visual artifact can be more easily observed by examining two cross-sections of the colors of the mesh: the red dotted line over an edge and the blue dashed line through the center of a gradient

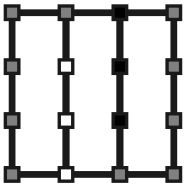


(d) Along the blue dashed line, DiffusionMesh will produce a curved interpolation of colors in order to maintain a C^1 continuity of colors. This differs from the linear interpolation along the edge, which makes the edge appear as a seam.

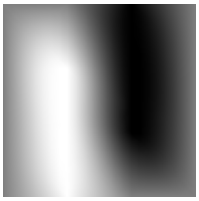
Figure 14: Visual artifacts can appear if the linear interpolation of colors on edges does not match the smooth interpolation of adjacent gradients



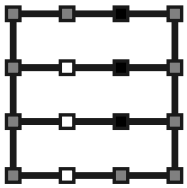
(a) A bitmap image of a white blurry line and a black blurry line



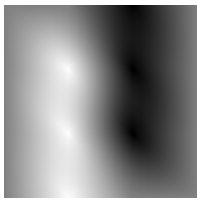
(b) Modeling the image as a mesh of vertical strips matches the original topology of colors



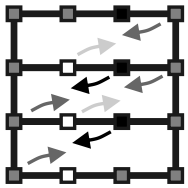
(c) The gradient produced from a mesh of vertical strips resembles the original image



(d) Modeling the image as a mesh of horizontal strips does not match the original topology of colors



(e) The gradient produced from a mesh of horizontal strips appears splotchy, and the lines of the original image are not obvious



(f) The gradient for horizontal strips will blend black, gray, and white colors together, resulting in gray regions in the middle of each strip

Figure 15: Artists must use care to ensure that mesh topologies correctly model the intended shapes

shapes, but this amount of control might be unnecessary. Instead, relaxing the constraints on edges so as to permit any type of interpolation of colors would allow DiffusionMesh to incorporate more of the diffusion process into the assignment of colors there, allowing mesh edges to be less noticeable. Even without this change, this artifact can be mitigated by adding additional nodes along an edge and using those nodes to manually adjust the edge's coloring to closely match the coloring of neighboring gradients.

In addition, during the drawing process, it was observed that although DiffusionMesh makes it much easier to create complex gradients, it cannot be used blindly. A naive choice of shapes and colors can produce poor gradients (Fig. 15). Because all the colors specified on the nodes of the shape end up blending across the whole shape and because the gradient of a shape depends on the gradients of neighboring shapes as well, inexperienced artists can create meshes with unexpected colors or uneven gradients, which can only be fixed by retopologizing the mesh. Unsurprisingly, the best gradients tend to result when the input mesh matches the topology of colors from the image being modeled.

6 Related Work

Vector drawing tools have traditionally offered simple procedural gradients such as linear gradients or radial gradients that provide limited artist control over colors, making it difficult for artists to combine these simple gradients together into more complicated gradients.

Some tools allow gradients to be defined using gradient meshes [12], which are gradients represented as 3D surfaces such as Coons patches. Although gradient meshes do provide artists with powerful controls over colors, the mesh itself is limited to being topologically a grid shape, which might not be suitable for objects with holes or multiple appendages. A gradient mesh can also provide too many control points to an artist for objects where fine control of colors is only needed for a small section of the objects.

To circumvent this problem, gradient meshes have been extended to support meshes of arbitrary topology [9]. Subdivision surfaces can be used to calculate a gradient for the mesh, but this solution technique requires color derivatives at nodes to have fixed values. In practice, these derivative values are set to zero, resulting in "flat spots" of color around control nodes, making it difficult to specify more subtle gradients. The use of subdivision surfaces can also encounter problems when dealing with concave shapes in the topology.

Generalizations of barycentric coordinates extend the concept of barycentric coordinates of triangles to arbitrary polygons [6]. This allows for the filling of complex shapes with a smooth gradient of colors based on colors defined on the exterior. Due to the lack of control over gradients at the edges of shapes, it is difficult to create more complex gradients that span multiple shapes.

Diffusion curves allow users to draw curves of colors on a canvas [10]. Those colors are then diffused across the canvas to create a smooth blend of those colors across the canvas. In the diffusion curve model, artists may experience difficulties when making localized changes to a drawing since the colors of any curve can diffuse across the entire canvas. Small changes in one part of a drawing also require solving a global optimization problem to recompute the gradient for the whole canvas, which is inefficient. Since the diffusion curve primitive spans an entire canvas, it does not support the compositing of multiple layers by default. Overall, because diffusion curves define a completely new graphics primitive that exhibits poor compositing behavior, they don't integrate well with traditional vector graphics tools which use a model based on layers of path primitives.

The traditional diffusion curve model is only able to generate gradients with a C^0 continuity of colors on either side of a curve. Since this can result in visual artifacts at these seams, an additional

blur operation is allowed in the area surrounding curves to hide these effects. One approach to avoiding these visual artifacts involves providing more control over the diffusion process so as to reduce the number of curves required for a drawing [1, 8]. Having only a C^0 continuity at curves results in a gradient calculation [4] that is amenable to various performance optimizations [2, 11]. Unfortunately, the use of blur operations makes it difficult for artists to create drawings with specific colors at specific locations, and it requires artists to specify extra blur parameters for each of the nodes in their drawings.

The concept of diffusion curves has been extended to allow for the specification of not only constraints on the colors of curves but constraints on color derivatives as well [5]. In particular, a C^1 continuity of colors on both sides of a curve can be achieved, allowing for a smooth blending of colors spanning the curve. Generating diffusion curve gradients with derivative constraints results in a biharmonic system. Finite element methods have been used to solve this biharmonic system on a triangle mesh [3]. DiffusionMesh achieves its C^1 continuity of colors by using a simpler approach of directly calculating its gradient on a set of Gouraud-shaded triangles. Although it is possible to arrange diffusion curves so that they form meshes, just as in DiffusionMesh, DiffusionMesh is specifically adapted for the use of meshes: it has a user interface that supports the creation of meshes; it takes advantage of the performance benefits of meshes; it allows meshes to be composited; and it generates gradients that do not exceed or "overshoot" the color values of individual shapes in a mesh.

7 Future Work

Although DiffusionMesh is fast enough for interactive use, the ability to use these techniques for animation would require much faster methods for computing gradients.

In addition, although the gradients produced by DiffusionMesh are visually satisfying, the mathematical foundations for the diffusion process could be strengthened. Solution techniques derived from finite element methods would be ideal since they would provide for more flexibility in allowing varying triangle sizes in the rendered triangle mesh. It is unclear whether finite element methods can be adapted to incorporate the constraint that the colors within a shape must be within the range of colors specified on the nodes defining the shape though.

DiffusionMesh uses a linear interpolation of colors across triangles in its rendered triangle meshes. This can lead to visual artifacts at the seams between triangles because they lack a C^1 continuity of colors between them. In practice, DiffusionMesh refines its triangle meshes to a level where this lack of C^1 continuity is not noticeable, but this often requires a large number of triangles. Since triangulation makes up a large part of the rendering time of a gradient, both performance and visual quality could be improved by using nonlinear interpolation of colors across triangles and coarser triangle meshes.

8 Conclusion

DiffusionMesh successfully demonstrates how diffusion-style gradients can be integrated into a traditional vector drawing system. Closed path shapes with different colors assigned to their nodes can be assembled into large meshes, allowing complex gradients to be defined. DiffusionMesh uses simple discrete methods to generate a mesh of Gouraud-shaded triangles that can be displayed easily by basic graphics hardware. Although the performance of the gradient diffusion code is too slow for real-time use, it is fast enough that its latency can be hidden by executing it in a background thread, thus providing enough performance to allow artists to interactively create gradients.

Acknowledgments

The box drawing (Fig. 8) was made by Salvador Clariana. The butterfly drawing (Fig. 10) was made by Camilo Torres. The chibi dog drawing (Fig. 12) is based off of art made by kirara-cecilvenes.

References

[1] H. Bezerra, E. Eisemann, D. DeCarlo, and J. Thollot. Diffusion constraints for vector graphics. In *Proc. NPAR*, pp. 35–42. ACM, New York, USA, 2010.

[2] J. C. Bowers, J. Leahey, and R. Wang. A ray tracing approach to diffusion curves. *Computer Graphics Forum*, 30(4):1345–1352, June 2011.

[3] S. Boyé, P. Barla, and G. Guennebaud. A vectorial solver for free-form vector gradients. *ACM Transactions on Graphics*, 31(6):173:1–173:9, Nov. 2012.

[4] S. Carlsson. Sketch based coding of grey level images. *Signal Processing*, 15(1):57–83, July 1988.

[5] M. Finch, J. Snyder, and H. Hoppe. Freeform vector graphics with controlled thin-plate splines. *ACM Transactions on Graphics*, 30(6):166:1–166:10, Dec. 2011.

[6] M. S. Floater. Mean value coordinates. *Computer Aided Geometric Design*, 20(1):19–27, Mar. 2003.

[7] D. Jackson, J. Fujisawa, J. Ferraiolo, J. Watt, D. Schepers, C. McCormack, C. Lilley, A. Grasso, P. Dengler, and E. Dahlström. Scalable vector graphics (SVG) 1.1 (second edition). W3C recommendation, W3C, Aug. 2011.

[8] S. Jeschke. Generalized diffusion curves: An improved vector representation for smooth-shaded images. *Computer Graphics Forum*, 35(2):71–79, May 2016.

[9] H. Lieng, J. Kosinka, J. Shen, and N. Dodgson. A colour interpolation scheme for topologically unrestricted gradient meshes. *Computer Graphics Forum*, 36(6):112–121, Sept. 2017.

[10] A. Orzan, A. Bousseau, H. Winnemöller, P. Barla, J. Thollot, and D. Salesin. Diffusion curves: A vector representation for smooth-shaded images. *ACM Transactions on Graphics*, 27(3):92:1–92:8, Aug. 2008.

[11] R. Prévost, W. Jarosz, and O. Sorkine-Hornung. A vectorial framework for ray traced diffusion curves. *Computer Graphics Forum*, 34(1):253–264, Feb. 2015.

[12] J. Sun, L. Liang, F. Wen, and H.-Y. Shum. Image vectorization using optimized gradient meshes. *ACM Transactions on Graphics*, 26(3), July 2007.