OverCoat: An Implicit Canvas for 3D Painting

Johannes Schmid
Disney Research Zurich
ETH Zurich

Martin Sebastian Senn
Disney Research Zurich
ETH Zurich

Markus Gross
Disney Research Zurich
ETH Zurich

Robert W. Sumner
Disney Research Zurich

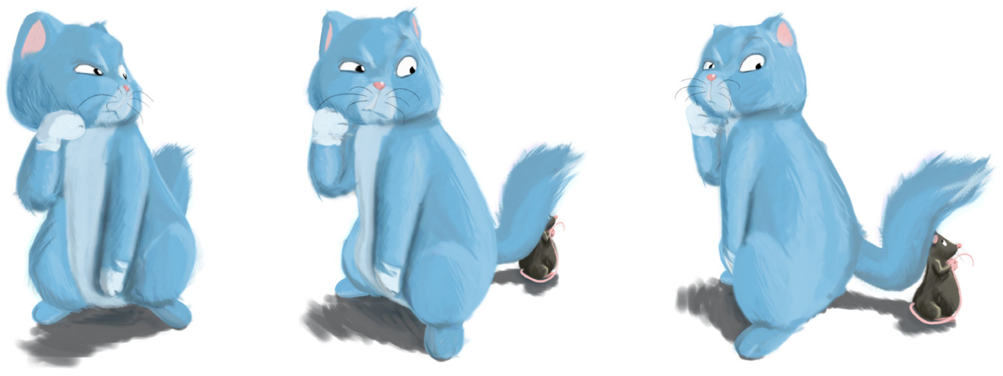

Figure 1: *Our implicit canvas generalizes the traditional 2D painting metaphor to 3D, enabling a new class of expressive 3D painting. Strokes on the cat's tail, for example, do not conform to any precise 3D surface, but are painted in space to give the tail its rough, stylized look.*

Abstract

We present a technique to generalize the 2D painting metaphor to 3D that allows the artist to treat the full 3D space as a canvas. Strokes painted in the 2D viewport window must be embedded in 3D space in a way that gives creative freedom to the artist while maintaining an acceptable level of controllability. We address this challenge by proposing a canvas concept defined implicitly by a 3D scalar field. The artist shapes the implicit canvas by creating approximate 3D proxy geometry. An optimization procedure is then used to embed painted strokes in space by satisfying different objective criteria defined on the scalar field. This functionality allows us to implement tools for painting along level set surfaces or across different level sets. Our method gives the power of fine-tuning the implicit canvas to the artist using a unified painting/sculpting metaphor. A sculpting tool can be used to paint into the implicit canvas. Rather than adding color, this tool creates a local change in the scalar field that results in outward or inward protrusions along the field's gradient direction. We address a visibility ambiguity inherent in 3D stroke rendering with a depth offsetting method that is well suited for hardware acceleration. We demonstrate results with a number of 3D paintings that exhibit effects difficult to realize with existing systems.

Keywords: digital painting, 3D painting, stroke based rendering

1 Introduction

An empty canvas represents the work space in which a painter realizes his or her creative vision. Working directly with brushes and paint to fill the canvas gives the artist full creative freedom of expression, evidenced by the huge variety of styles that have been explored through art's rich history. Modern digital painting software emulates the traditional painting metaphor while further empowering the user with control over layering, compositing, filtering, and other effects. As a result, digital artists have an extremely powerful, flexible, and expressive tool set for creating 2D digital paintings.

The same is not true for 3D digital painting. Most attempts to bring digital painting into the third dimension focus on texture painting or methods that project stroke centerlines onto an object's surface. The strokes must precisely conform to the object's surface, and the mathematical nature of these algorithms can betray the underlying 3D structure of the scene, leading to a "gift-wrapped" appearance. Stylistic effects that require off-surface brush strokes cannot easily be realized. Indistinct structures such as fur, hair, or smoke must be addressed using special-purpose modeling software without the direct control afforded by painting. These limitations ultimately restrict the variety of styles possible with 3D digital painting and may hinder the artist's ability to realize their creative vision.

In our research, we experiment with an alternate way to define the 3D painter's workspace that targets existing limitations. We elevate the 2D painting metaphor to 3D with a generalization that allows the artist to treat the full 3D space as a canvas. With this new 3D canvas, painting no longer focuses on how to paint *on* an object, but rather how to paint *in* space. Figure 1 demonstrates a 3D painting created by our prototype system called "OverCoat." The implementation of the generalized canvas concept in OverCoat, which makes this painting possible, poses several challenges.

Strokes painted in the 2D viewport window must be embedded in 3D space in a way that gives creative freedom to the artist while maintaining an acceptable level of control. We address this challenge by proposing a canvas concept defined implicitly by a 3D scalar function. The artist shapes the implicit canvas by creating approximate 3D proxy geometry that defines a scalar distance field.

An optimization procedure is used to embed painted strokes into 3D space by satisfying criteria defined on the scalar field and implemented as different objective terms. For example, one objective term ensures that strokes are embedded on a particular level set of the scalar field. Since any level set value can be chosen, the artist is not restricted to painting on any particular surface. Other objective criteria allow the artist to paint across level sets, allowing fur, hair, whiskers, or other effects to be created. By formulating the optimization problem on the strokes themselves, the full scalar field need never be created and stored explicitly, leading to an efficient stroke embedding algorithm. The need for fine-scale control over the implicit canvas presents a second challenge, which we address by a unified painting/sculpting metaphor. A sculpting brush uses the same optimization procedure discussed above but creates a local change in the scalar field, resulting in outward or inward protrusions along the field's gradient. Using this sculpting tool, artists can shape the canvas before painting into it, or move strokes that have already been embedded to fine-tune the result. Finally, efficient rendering is a key consideration in any interactive painting system. In our case, the problem is non-trivial because merging 2D painting and 3D concepts creates an ambiguity for the visibility determination of paint strokes. We resolve this issue with a simple pre-sorting step that generates an unambiguous rendering order and allows our brush model to be implemented with interactive performance.

Our primary contribution is a 3D painting system based on the concept of the implicit canvas that allows artists to create a new class of expressive 3D paintings. We also make the technical contributions of an efficient optimization procedure for stroke embedding that supports a unified interface for both painting and sculpting. Finally, we contribute a real-time rendering algorithm to efficiently display the resulting 3D paintings.

2 Background

In his invitation to discuss computer depiction, Durand [2002] highlights the difference between primary space (the 3D world in which objects live) and secondary space (the 2D canvas on which depictions of those objects are created), illustrated in Figure 2. This distinction, originally introduced to analyze the historical use of representation systems in engineering drawings [Booker 1963] and fine art [Willats 1997], provides a lens through which to explore the development of expressive depiction in computer graphics.

The field of non-photorealistic rendering (NPR) has developed a rich collection of expressive depiction methods. Although traditional photorealistic rendering research focuses on the primary space (e.g., scene representation, visibility determination, global illumination), the NPR community first approached the problem from the opposite direction by focusing entirely on the secondary space of the 2D canvas. Haeberli's interactive "Paint By Numbers" system [1990] fills a 2D canvas with brush strokes whose attributes are controlled by information contained in a photograph. This concept led to an entire sub-field of research on stroke-based rendering [Hertzmann 2003]. Lu and colleagues' impressive work on the interactive stylization of images, video, and animations [Lu et al. 2010] includes an overview of recent work. These methods share a common focus on computation performed in the secondary space of the 2D canvas without an explicit representation of the 3D world.

Inspiring secondary-space results naturally led researchers to extend expressive depiction to the primary space of 3D objects, leading to a diverse collection of non-photorealistic rendering algorithms. Representative works in this area include watercolor synthesis [Bousseau et al. 2006], stylized depiction of fur, grass, and trees [Kowalski et al. 1999], apparent ridges [Judd et al. 2007], and realtime non-photorealistic rendering [Markosian et al. 1997; Praun et al. 2001]. The common focus of these techniques is an algorithmic mapping from primary space to secondary space that implements different expressive styles.

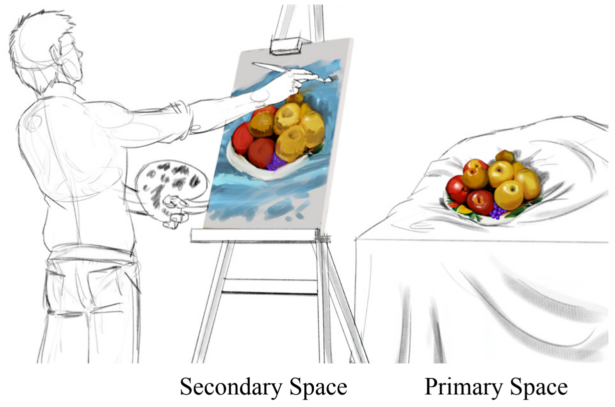

Figure 2: *Primary space refers to the 3D world in which objects live, while secondary space denotes the 2D canvas on which depictions of those objects are created [Durand 2002]. Our work blurs the distinction between these two spaces by upgrading strokes to the primary space and downgrading traditional 3D objects to serve only as helpers in defining an implicit canvas.*

Among these methods, Meier's painterly rendering system [1996] marks the inception of a line of research closely related to our own. Particles attached to an object's primary-space surface are used to render secondary-space brush strokes so that the strokes stick to the object as the camera moves. This simple, though groundbreaking, concept ultimately led to the development of Disney's Deep Canvas technology [Katanics and Lappas 2003], which replaces Meier's procedurally generated particles with an artist-driven painting system. Painted strokes are projected on the object's surface and stored along with all data associated with the painting system. A new view is rendered by "repainting," or playing back all recorded painting operations, using the camera's new view transformation. Concurrent with the development of Deep Canvas, Teece [2000] proposed a related painting concept with a focus on interactivity. The WYSIWYG NPR system of Kalnins and colleagues [2002] expands upon this line of work by showing how algorithmic rendering techniques such as silhouette stylization or hatching can be controlled directly by the artist via a painting interface. In the commercial world, Maya Paint Effects [Paint Effects 2011] projects painted strokes onto the surface of scene geometry and uses them as seed points to create new geometric primitives such as grass or flowers.

Our work shares many similarities with these methods: We have built an artist-driven 3D painting system that employs a stroke-based renderer to generate temporally coherent expressive imagery. The distinguishing characteristic of our method and one of our key contributions can be seen in terms of primary and secondary space. Existing methods such as Meier's painterly rendering system, Deep Canvas, Teece's technique, WYSIWYG NPR, and Paint Effects all embed strokes directly on the surface of an object or along object features such as silhouettes. Although the artist has a great deal of freedom when painting input strokes in secondary space, the strokes must conform to the surfaces of their associated primary-space objects. In this way, the functionality of these systems is intimately tied to and restricted by traditional surface representations such as polygon meshes and NURBS surfaces. The exacting nature of these surface representations may be at odds with an artist's particular style, hindering their ability to realize their artistic vision.

Our work targets precisely this limitation by proposing a generalized 3D canvas concept that allows strokes to be embedded anywhere in space. This change, in essence, blurs the distinction between the primary and secondary spaces by upgrading painted strokes to the primary space while at the same time downgrading traditional 3D objects to play a subservient helper role in shaping the implicit canvas. This new freedom to paint anywhere in space comes at the cost of additional challenges related to usability and control not present in previous work. Our implicit canvas concept and optimization formulation answer these challenges.

Other researchers have approached the problem of placing color directly in 3D with the use of specialized input devices. The Cave-Painting system of Keefe and colleagues [Keefe et al. 2001] uses motion capture in a virtual-reality cave to allow the artist to directly author scenes composed of ribbons, tubes, and other primitives using hand gestures. Schkolne and colleagues' Surface Drawing work [2001] enables organic shapes to be modelled using hand gestures in a semi-immersive VR setup. Because controllability is an issue with gesture-based systems, Keefe and colleagues [2007] propose a method that uses a haptic device and 6-DOF trackers to draw lines in space in a more precise fashion. This body of work makes important advancements in human-computer interaction for ab initio design using advanced hardware devices. Our contribution is distinguished from these direct 3D painting systems in two primary ways: OverCoat extends the tablet-based 2D digital painting metaphor to 3D without the need of special hardware, thus making it more accessible and familiar to work with for artists. In addition, OverCoat's embedding procedure, brush model, and rendering algorithm merge 2D and 3D concepts to enable paintings that retain their 2D expressiveness when viewed from any angle. Since existing direct 3D painting systems create scenes composed of 3D primitives such as ribbons, tubes, and surfaces, they cannot easily accommodate the expressive aesthetic of traditional digital painting.

Commercial modeling packages [Maya; Mudbox; ZBrush] complement our work by providing tools to create 3D proxy geometry. Sketch-based modeling tools [Igarashi et al. 1999; Nealen et al. 2007], especially those that use an implicit surface representation [Karpenko et al. 2002; Schmidt et al. 2005; Bernhardt et al. 2008], are well suited since the proxy geometry need only provide a rough guide to control the implicit canvas, but is never rendered directly. Other related work [Cohen et al. 2000; Bourguignon et al. 2001; Tolba et al. 2001; Rivers et al. 2010] explores interesting concepts concerning 3D drawing, but does not address detailed 3D painting and cannot easily accommodate the expressive aesthetic we target with OverCoat.

3 Concepts and Implementation

In order to provide a controllable and effective system for 3D painting, OverCoat employs a 3D canvas based on scalar fields that are used to embed paint strokes in space. The artist sculpts proxy objects with any modeling package and imports them into the scene as triangle meshes. The objects define the overall 3D layout of the scene but need not exhibit fine geometric details since they only serve as a guide for stroke embedding and are not rendered in the final painting. Each proxy object implicitly defines a signed distance field that, conceptually, represents the object's 3D canvas.

OverCoat provides the user with a set of tools to embed paint strokes into the 3D canvas. The user first selects a proxy object and then paints into the canvas using a familiar 2D painting interface. The way in which paint strokes are embedded is determined by the semantics of the embedding tools. To obtain a maximum of flexibility and extensibility in the creation of these tools, we formulate the embedding process as a mathematical optimization problem that assigns a depth value to each point of the input paint stroke. Objective terms and constraints are used in varying combinations to obtain different embedding behaviors. These combinations are encapsulated and presented to the user as a set of different embedding tools, such as a tool to paint at a certain distance to the object, or to paint strokes that are perpendicular to the proxy surface. The embedding tools can use the scalar field magnitude, sign, and gradient in their objective terms to establish criteria that relate any position in space to the proxy object.

Another advantage of embedding paint strokes using an optimization instead of specialized heuristics (such as direct projection) is that it allows our system to gracefully handle cases where a tool's primary goal cannot be met. For example, by incorporating a regularizing smoothness term, our level set painting tool can easily handle the case where painted strokes extend beyond the level set's silhouette. A method based on direct projection would require special heuristics since, in this case, there is no surface on which to project.

An additional sculpting tool allows the user to make localized modifications to the scalar field using the same embedding procedure. In this way, the painting interface can be used both for coloring the canvas and for manipulating the shapes.

3.1 Canvas representation

OverCoat represents a 3D canvas as a scalar field $f : \mathbb{R}^3 \to \mathbb{R}$. A point $\mathbf{x}$ with $f(\mathbf{x}) = l$ is said to lie on level l. The corresponding implicit surface at level l, also called an isosurface or level set, is the set of all points $\mathbf{x} \in \mathbb{R}^3$ such that $f(\mathbf{x}) = l$. The scalar field relates the points in space to the surface of the corresponding proxy object. In OverCoat, a proxy object is defined by a triangle mesh that forms a closed manifold solid. The scalar field is initially defined by the signed Euclidean distance to the proxy geometry's surface. However, the sculpting tool can inflict direct, localized changes to the field values so that they no longer represent distances. In either case, the scalar field is only C^0 continuous. This property has no negative influence on the stroke embedding with the objective terms and tools presented in this paper. Problems could arise if objective terms were introduced which are more sensitive to the scalar field smoothness. In this case, a scalar field formulation with higher order continuity, such as the one described by Peng and colleagues [2004], might be preferable.

We define all operations on the scalar field in a form that does not require explicit storage of the field values. Thus, we avoid the memory and computational costs of a voxel decomposition or the algorithmic complexity of more sophisticated distance-field representation methods [Frisken et al. 2000].

3.2 Painting representation

Inspired by Deep Canvas, a 3D painting is represented with a data structure that stores all available input information associated with a painted stroke in OverCoat. This information includes the stroke's centerline (embedded in 3D space), the brush shape and color, and additional information, such as pen pressure. The centerline of a stroke is stored as a sequence of points that we will refer to as stroke points. Position and variable parameters are interpolated linearly between the stroke points, thus forming a polyline. With this information, the painting can be reproduced faithfully from the original view point by playing back all painting operations. This repainting procedure corresponds to rendering in our context. Due to the 3D nature of our canvas, OverCoat can render the painting from any other vantage point.

In order to match the feel of traditional 2D digital painting, paint strokes are always rendered as 2D brush strokes on the view plane,

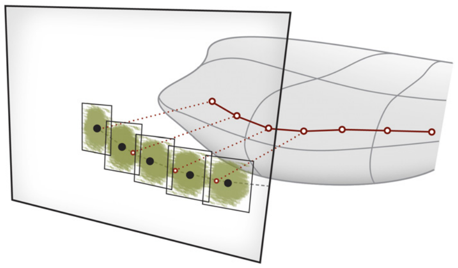

Figure 3: *Paint strokes are stored in 3D in a parametric representation. For rendering, their centerlines are projected to the view plane and sampled with brush splats. Red circles in this figure represent stroke points, while black dots are sampled splat locations.*

much like a traditional 2D raster painting software might draw them. Stroke centerlines are first projected from their embedded 3D locations onto the current view plane. OverCoat then renders the strokes by repeatedly compositing a brush splat texture along the stroke into the image (Figure 3). If the distance between splats is small enough, they will appear as one smooth stroke. At paint time, the width of a stroke is defined in screen space. When rendered from different view points, this width is perspectively transformed to maintain its size relative to the location in space at which it was embedded.

3.3 Stroke embedding

The artist paints in a particular 2D view of the 3D canvas, generating an ordered sequence of n stroke points $\mathbf{s}_i \in \mathbb{R}^2$. The goal of stroke embedding is to find 3D positions $\mathbf{p}_i \in \mathbb{R}^3$ for these points in a way that is meaningful and useful to the artist. To target an embedding algorithm that meets these workflow considerations in a flexible and extendable way, we cast the embedding of the stroke points as an optimization problem. This framework allows us to implement objective function terms that accomplish different embedding behaviors, such as painting on a level set of the 3D canvas's scalar field, or across the scalar field between two chosen level sets. Combinations of these terms are exposed to the user as different embedding tools.

To ensure that the embedded strokes match the artist's intent, it is crucial that the stroke points $\mathbf{p}_i$ project back to their original screen space locations $\mathbf{s}_i$ in the view from which they were painted. We enforce this property strictly by parameterizing the stroke points by their view ray: $\mathbf{p}_i = \mathbf{o} + t_i\mathbf{d}_i$, where $\mathbf{o}$ is the camera position, $\mathbf{d}_i$ the view vector that passes through $\mathbf{s}_i$ on the screen plane, and t_i the ray parameter. The t_i are thus the unknown variables of the optimization.

3.3.1 Objective terms

We propose three objective terms that provide the ingredients necessary to build OverCoat's embedding tools. The *level distance* term is minimized when all stroke points are at a particular distance from the proxy geometry. The *angle* term minimizes the curvature of the stroke and thus smoothes its embedding. The *arc length* term favors straight embeddings by minimizing the total length of a stroke.

Level distance OverCoat allows the user to select a specific level l, and hence a specific isosurface $f(\mathbf{x}) = l$, on which to apply strokes. The corresponding objective term should ensure that all stroke points are embedded as closely as possible to the selected isosurface. The *level distance* objective term sums the difference between the actual field value $f(\mathbf{x})$ evaluated at all point locations $\mathbf{p}_i$ and the desired level l:

$$E_{\mathit{level}} = \sum_{i=1}^{n} \left(f(\mathbf{p}_i) - l\right)^2 . \quad (1)$$

Angle The *angle* objective term aims to minimize the directional deviation of consecutive line segments along a stroke. This deviation is measured by the dot product between the normalized line segments, which equals 1 when the segments are co-linear:

$$E_{\mathit{angle}} = \sum_{i=1}^{n-2} \left(1 - \frac{\mathbf{p}_{i+2} - \mathbf{p}_{i+1}}{\|\mathbf{p}_{i+2} - \mathbf{p}_{i+1}\|} \cdot \frac{\mathbf{p}_{i+1} - \mathbf{p}_i}{\|\mathbf{p}_{i+1} - \mathbf{p}_i\|}\right)^2 \quad (2)$$

Arc length The *arc length* objective term penalizes the collective length of all segments:

$$E_{\mathit{length}} = \sum_{i=1}^{n-1} \|\mathbf{p}_{i+1} - \mathbf{p}_i\|^2 \quad (3)$$

3.3.2 Optimization

The goal for an embedding tool is to find ray parameter values t_i and thus 3D locations for all stroke points that minimize the weighted sum of all objective terms:

$$E = w_{\mathit{level}}E_{\mathit{level}} + w_{\mathit{angle}}E_{\mathit{angle}} + w_{\mathit{length}}E_{\mathit{length}} \quad (4)$$

Individual embedding tools, described in the next section, achieve different behaviors by setting different values for the weights w_{level}, w_{angle}, and w_{length}. Our system uses the quasi-Newton L-BFGS method to solve this non-linear optimization problem. Since the only unknowns to the optimization are the depth values t_i, the optimization does not change the shape of a stroke in the view in which the stroke was painted.

3.3.3 Embedding tools

The objective terms presented in the previous section provide the ingredients necessary to implement three powerful embedding tools (Figure 4). All examples were painted using these three tools.

Level set tool The level set tool embeds all stroke points as closely as possible onto a selected level set surface. This goal is achieved by giving a dominant weight to the level distance term E_{level}. By itself, this term has the same effect as direct projection for paint strokes within the silhouette boundaries of the level set. When a stroke extends outside the silhouette, the closest distance solution will be roughly perpendicular to the surface in the region where the silhouette is crossed, thus creating a sharp corner along the embedded stroke. We incorporate the angle term E_{angle} to achieve a smoother transition in this case. The weights $w_{\mathit{level}} = 1$, $w_{\mathit{angle}} = 0.1$, and $w_{\mathit{length}} = 0$ were used for level set tool in all of our examples. If a fuzzy embedding is desired, the target level can be displaced by a random amount for each stroke, or even for each individual stroke point.

Hair and feather tool Another set of tools allows the user to paint across level sets. The user selects a target level for both the start and the end of the stroke. The first and last points of a painted stroke are constrained to lie on these prescribed levels using $w_{\mathit{level}} = 1$. The remaining stroke points, however, are optimized with $w_{\mathit{level}} = 0$. In the absence of a target surface, the angle objective term ensures a smooth transition between the two ends with $w_{\mathit{angle}} = 1$. With this

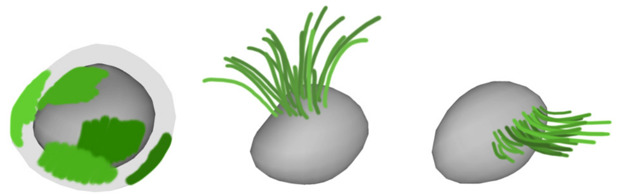

Figure 4: *The three embedding tools implemented by OverCoat.*

term alone, the resulting embedding will be smooth, but may be extended undesirably in order to meet the angle criteria optimally, resulting in strokes that overshoot the prescribed target level set. We used the arc length term with $w_{\mathit{length}} = 0.05$ to regularize this behaviour and cause a straighter embedding in space.

The resulting cross-level embedding can be controlled more explicitly with additional constraints. For example, the initial direction of the stroke can be prescribed by temporarily pre-pending an artificial stroke point. This point stays fixed during the optimization of the stroke, but affects the embedding solution through the angle objective term. Depending on its relative position to the first actual stroke point $\mathbf{p}_1$, it will cause the embedded stroke to leave the surface in a particular direction. For the "hair" tool, the temporary point is placed along the negative gradient direction at $\mathbf{p}_1$, causing the initial direction of the stroke to be perpendicular to the level set. The "feather" tool was realized by placing the temporary point in the direction that is tangential to the scalar field at $\mathbf{p}_1$ and has the largest angle to the straight line connecting $\mathbf{p}_1$ and $\mathbf{p}_n$.

3.3.4 Distance, derivative, and gradient computations

In the process of embedding strokes in space as described above, the optimization procedure must repeatedly evaluate a canvas's scalar field to calculate the field's magnitude $f(\mathbf{x})$, the gradient ∇f, and the derivative of the scalar field with respect to the ray parameter $\partial f(\mathbf{p}_i)/\partial t_i = \nabla f \cdot \mathbf{d}_i$. In the absence of sculpting operations, $f(\mathbf{x})$ is defined to be the smallest distance to the proxy geometry, which we designate as $f_{\mathit{proxy}}(\mathbf{x})$. This distance is computed by finding the closest point within any primitive of the proxy geometry mesh. The sign of the distance is found using the angle-weighted normals of the mesh primitives [Baerentzen and Aanaes 2005]. The gradient is generally defined by the normalized vector between the query point $\mathbf{x}$ and its closest point on the surface. If $\mathbf{x}$ lies very close to or exactly on the surface, this definition becomes unstable, and the angle-weighted normal of the closest primitive is used instead. All scalar field evaluations are computed on the fly, so that OverCoat never needs to store the field in a discretized form. An oriented bounding box tree [Gottschalk et al. 1996] is used to accelerate the closest primitive look-ups, which allows the embedding to be performed interactively.

3.3.5 Initialization and refinement

To accelerate convergence and avoid inappropriate local minima, our system initializes the unknowns to lie on the frontmost target level set of the scalar field using Sphere Tracing, a ray marching technique described by Hart [1994]. For the hair and feather tools, only the first and the last stroke points are initialized to their respective target levels, while the remaining unknowns are initialized with a linear interpolation between the two end points.

If a target surface or parts of it are at a considerable angle to the screen plane, the sampling of points along the stroke from the input device may not be sufficient for the stroke to be embedded nicely in the scalar field. For example, the level objective term can only be faithful to the chosen isosurface if the stroke sampling is fine enough for the level of detail of the surface. Likewise, the angle term can only provide an effective smoothing if the sampling is appropriate. Our system therefore refines input strokes painted with the level set tool during their initialization. If the Euclidean distance between two consecutive stroke points after initialization is larger than a given threshold, a new stroke point is inserted halfway between the existing stroke points in 2D and immediately projected according to the initialization method described above. This step is repeated until all stroke segments are at most twice as long as the shortest segment in the stroke, thus guaranteeing a roughly uniform sampling along the stroke.

The refinement process has the added benefit of detecting strokes that cross occluding contours. Painting across occluding contours can result in stroke segments bridging two parts of an isosurface, which are potentially far apart in 3D. Attempting to refine such a segment causes an infinite recursion within a segment that cannot become any shorter. The end points of this segment eventually converge to two distinct 3D points that both project to the same point in 2D (Figure 5). OverCoat detects this case by comparing the original segment length with the lengths of the two new segments. If the ratio is below a given threshold, the paint stroke is split into two at the location of convergence. We found that a value of 0.1 works well for the threshold ratio.

3.4 Sculpting

In the same way that paint strokes are embedded to add color to the canvas, our sculpting tool embeds sculpting strokes that alter the shape of the canvas itself. Sculpting strokes act as direct modifiers to the canvas's scalar field and thus have an influence on the embedding of subsequent strokes. A sculpting stroke defines a contribution function $C(r, R)$, where r is the smallest distance from $\mathbf{x}$ to any of the line segments of the sculpting stroke, and R is a user-defined radius of influence. OverCoat uses a cubic polynomial with local support [Wyvill et al. 1986] for $C(r, R)$:

$$C(r,R) = \begin{cases} 2\frac{r^3}{R^3} - 3\frac{r^2}{R^2} + 1, & \text{if } r < R \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

The scalar field is modified by adding the contributions of all sculpting strokes to the field function:

$$f(\mathbf{x}) = f_{\mathit{proxy}}(\mathbf{x}) - \sum_j K_j C(r_j, R_j), \tag{6}$$

where j enumerates all sculpting strokes with a non-zero contribution at $\mathbf{x}$, and K_j determines the magnitude and direction of each sculpting operation. The contribution functions locally change the magnitude of the scalar field gradient. The amount by which a surface is shifted by a sculpting operation therefore depends both on K_j and previous sculpting operations in the region. OverCoat keeps the magnitude of the surface deformation approximately constant by setting K_j to the scalar field at a user-chosen distance in gradient direction from the stroke centerline.

Once a sculpting stroke has been embedded, it is incorporated into the evaluation of the canvas's scalar field. As a consequence, the scalar field values no longer represent the distance to the zero level set. The gradient of the scalar field is augmented by the sculpting contribution functions:

$$\nabla f(\mathbf{x}) = \nabla f_{\mathit{proxy}}(\mathbf{x}) - \sum_j K_j \nabla C(r_j, R_j), \tag{7}$$

$$\nabla C(r,R) = \begin{cases} 6\frac{r^2}{R^3}\frac{\partial r}{\partial \mathbf{x}} - 6\frac{r}{R^2}\frac{\partial r}{\partial \mathbf{x}}, & \text{if } r < R \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

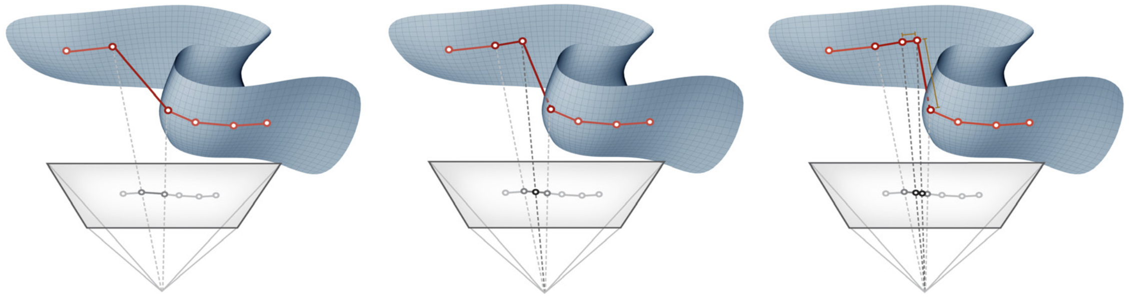

Figure 5: *This illustration shows three consecutive refinement steps of a paint stroke crossing an occluding contour of a target isosurface. In every step, the ratio between the original segment and the smaller sub-segment gets larger.*

The ray marching procedure used to find an initial embedding solution (Section 3.3.5) requires a lower bound of the Euclidean distance to the target level set l to determine its step size. If the query point $\mathbf{x}$ is not within the influence region of any sculpting strokes, a practical lower distance bound is the minimum between the distance to the closest sculpting stroke and $f_{proxy}(\mathbf{x}) - l$. Otherwise, OverCoat uses a distance bound derived according to Hart [1994]:

$$d(\mathbf{x}, S) >= \frac{f(\mathbf{x}) - l}{1 + \sum_j |K_j| \frac{3}{2R_j}}, \tag{9}$$

where the lower part of the fraction is the Lipschitz constant of the signed distance field of the proxy geometry (which is equal to 1) plus the Lipschitz constant of the sum of all contributions of the sculpting strokes.

To provide immediate feedback of the sculpting operations to the user, OverCoat deforms a copy of the proxy geometry by moving affected vertices along their normal to the new zero level set. If necessary, the mesh is refined to account for geometric complexity added by the sculpting tool. This copy is used for display only, while future scalar field computations use the original proxy geometry together with the sculpting stroke influence functions directly (Equation 6).

3.5 Rendering

As described in Section 3.2, paint strokes are rendered after projecting their centerlines onto the view plane. A small textured quad representing the paint brush is moved along the stroke and repeatedly composited into the image with the *over* operator. For each view change, all strokes must be redrawn. Our method is able to perform rendering at interactive refresh rates for scenes with hundreds of thousands of splats (Table 1).

A complication is introduced by conflicting rules for the overlap order of paint strokes. The classic 2D painting metaphor requires paint strokes to be placed on top of each other in the order that they were originally painted. The semantics of the underlying 3D object, on the other hand, prescribe that strokes from different surfaces should be rendered in depth order. Locally, we expect strokes to be rendered in the sequence that they were painted, while globally they must adhere to depth order.

We address this conflict in OverCoat by sorting all splats according to a modified depth value before rendering. This modified depth value is taken from a shifted centerpoint position:

$$\mathbf{q_k} = \mathbf{p_k} + C \cdot n \cdot \mathbf{d_k}. \tag{10}$$

Position $\mathbf{p_k}$ and direction $\mathbf{d_k}$ are interpolated from the stroke points. The interpolated direction represents the view rays along which the stroke points were initially embedded. The integer n is an identifier of the stroke sequence, which is increased by one for each stroke that is drawn. The stroke order is therefore translated into a depth offset, but this modified depth value is only used for splat sorting. A constant rendering parameter C scales the magnitude of the depth offsetting.

4 Results

We present five complete 3D paintings created by three different artists using our prototype OverCoat software. For these paintings, the artists modeled approximate proxy geometry in either Maya [Maya] or ZBrush [ZBrush] and imported it into OverCoat for painting. The proxy geometry does not include fine details. Instead, the artists achieved the detailed result by painting strokes with the level set, hair, and feather tools, or sculpting additional details with the sculpting tool. The accompanying video contains an overview with live screen captures that show the different tools in action, a video of an artist using the system, and turntable animations of all five paintings.

The "Cat and Mouse" painting is shown in Figure 1 from three different viewpoints. The cat's tail is depicted with strokes that do not conform to the proxy geometry's surface. By painting off surface, the artist gave the tail its rough, comic look. The whiskers on the cat and mouse demonstrate strokes painted in space using the hair tool. Figure 6 depicts an "Autumn Tree" from front and top views. In the bottom row of the figure, a rendering of the stroke centerlines is blended with the proxy geometry. It shows that the leaves are painted in the space surrounding the rough canopy geometry. "Captain Mattis" is shown in Figure 8. The sculpting tool was used to sculpt the Captain's beard and eyebrows. The bottom row of Figure 8 visualizes the original, unsculpted head, the sculpting strokes, and the final painted result. The "Angry Bumble Bee" in Figure 7 shows how the hair and feather tools can be used to create a fluffy appearance. The "Wizard vs. Genie" painting shown in Figure 11 is our most complex example. Facial features and cloth wrinkles were sculpted using OverCoat's sculpting tool, and the smoke was given a fuzzy appearance by using the random offset feature of the level set tool.

Figure 9 demonstrates the advantage of OverCoat over more traditional methods that restrict 3D paintings to conform tightly to the surface of scene objects. The left column in this figure shows the 3D painting as created by the artist. To create the right column, we reprojected all paint strokes onto the zero level set so that they lie exactly on the proxy geometry. In the reprojection, the silhouette of the captain's arm becomes a precise line without stylization, revealing the smooth nature of the underlying 3D geometry. Likewise, the bee's fuzzy body and hairstyle lose their expressive quality.

Statistics about the examples are included in Table 1. The paintings range in complexity from 5,000 to 24,000 strokes. Our renderer maintained interactive frame rates for all examples.

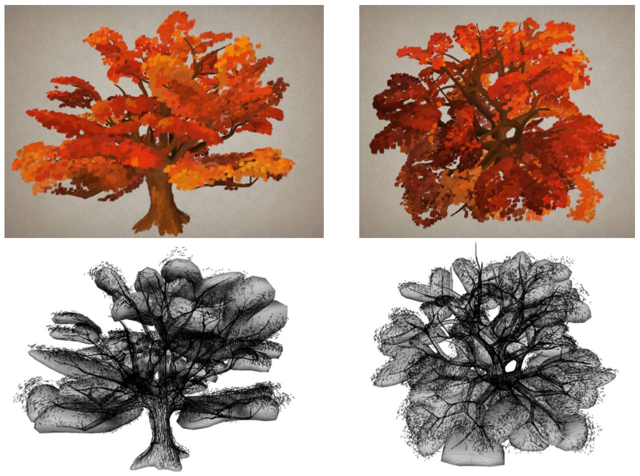

Figure 6: *"Autumn Tree": Leaves are individually painted strokes on offset levels of rough geometry representing canopies.*

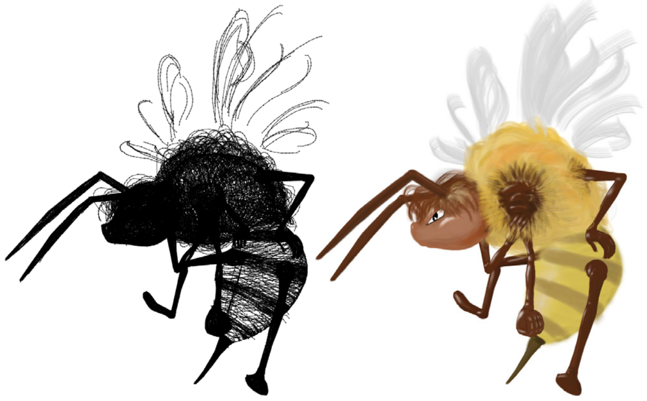

Figure 7: *"Angry Bumble Bee": OverCoat allows painting hair and fur, as well as other structure that may not be easily representable using textured meshes.*

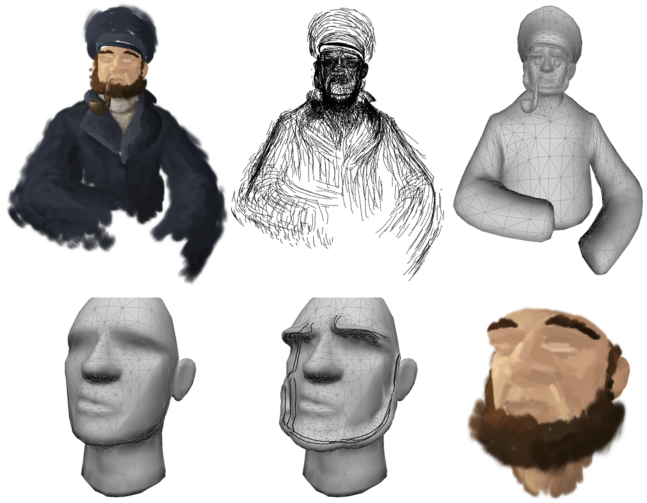

Figure 8: *"Captain Mattis": Top row: Finished painting, paint strokes, and proxy geometry. Bottom row: original proxy geometry, geometry with sculpting strokes, and final painting.*

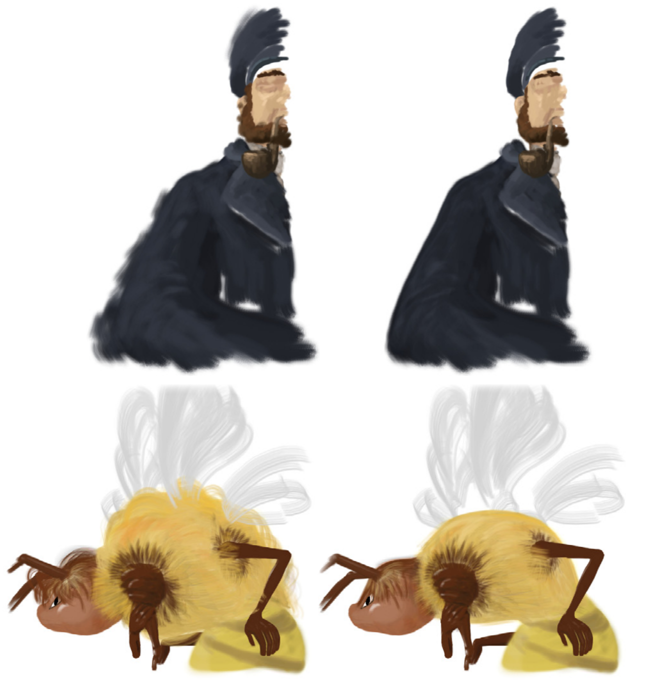

Figure 9: *The left column shows excerpts of the original paintings. In the right column, all strokes were projected onto the zero level set in order to highlight the benefit of our embedding methods.*

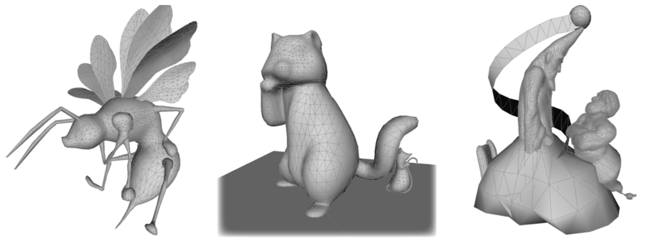

Figure 10: *This figure shows the proxy geometry used for the "Angry Bumble Bee," "Cat and Mouse" and "Wizard vs. Genie" examples.*

5 Conclusion

We have presented a system that bridges the worlds of 2D raster painting and 3D rendering by generalizing the 2D painting metaphor to the third dimension in a way that empowers the artist to create a new class of expressive 3D paintings. We contribute the concept of an implicit canvas shaped by the artist through the creation of approximate proxy geometry, as well as an optimization procedure for stroke embedding. Taken together, these contributions enable new workflows for 3D painting, including the painting and sculpting tools implemented in our prototype software.

Our work represents the first experiment in realizing a new concept for 3D painting. As such, there are many limitations in our prototype system and, correspondingly, ample room for future work. With our current system, the artist must paint all lighting and texture information by hand. While this gives the artist the utmost level of freedom of expression, it also means more work. Future work could incorporate ideas from Meier's painterly rendering system [1996] or WYSIWYG NPR [Kalnins et al. 2002] to transfer scene lighting and shading information to painted strokes. The fidelity of our rendered brushes is limited by the simplicity of our splat-based paint model. Future work could incorporate more re-

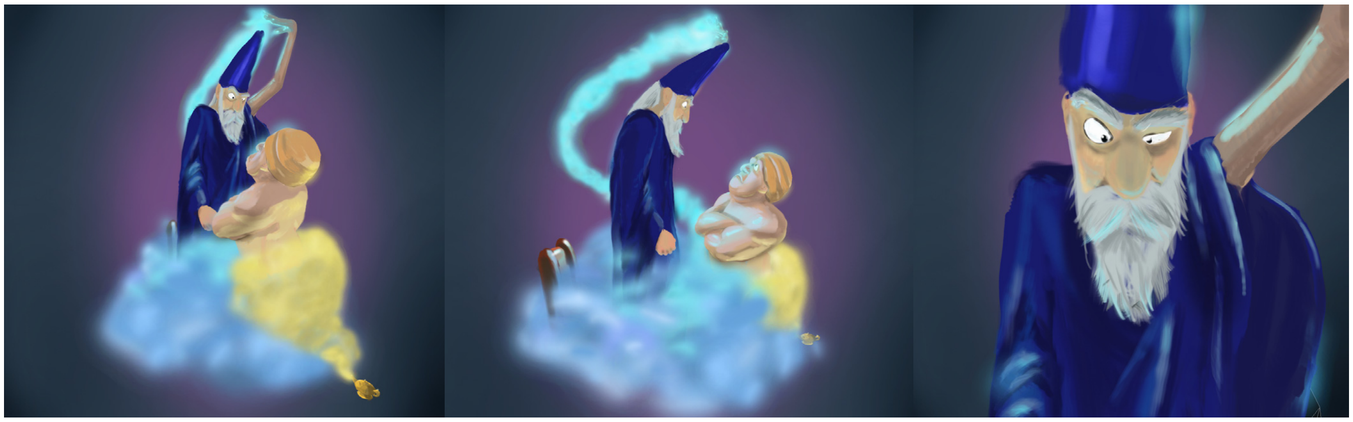

Figure 11: *"Wizard vs. Genie": Since OverCoat has a unifying representation for both surface and space, it is easy to paint clouds and other volumetric effects, by painting on offset surfaces. Effects such as the clouds in these images would be difficult to achieve with texture painting techniques. The beard in the rightmost image shows an exemplary use of the feather tool.*

Example	Triangles	Strokes	Splats	Time
Autumn Tree	29k	21k	138k	270 ms
Captain Mattis	6.6k	5k	40k	70 ms
Cat and Mouse	7.5k	5k	130k	200 ms
Angry Bumble Bee	6.3k	20k	304k	370 ms
Wizard vs. Genie	30k	24k	452k	610 ms

Table 1: *Results statistics: The second and third columns list the triangle count of the proxy geometry and the total number of paint strokes of each example scene. The last two columns show the number of splats and the rendering time for a representative view rendered with 720p resolution.*

alistic media simulation, such as the impressive paint model used by Project Gustav [Baxter et al. 2010]. Stylized 2D paintings often exhibit view-dependent shape changes. Our system cannot support such changes, since the 3D canvas's structure is independent of camera view. Future would could incorporate ideas from view-dependent geometry [Rademacher 1999] into the 3D canvas authoring process. Our renderer implements a fixed-function compositing algorithm. In contrast, 2D digital painting software packages give the artist a great deal of control over layering, masking, and compositing. Incorporating these concepts into OverCoat represents an additional avenue of future work. Ambiguities in stroke order are known to present problems for stroke-based renderers [Meier 1996; Katanics and Lappas 2003]. Our solution, based on splat ordering, can lead to occasional popping artifacts due to order changes. More thorough investigation of stroke-order ambiguities is a topic for future work. Our artists were successful in using Maya and ZBrush for modeling in conjunction with OverCoat. However, the system could be improved by integrating sketch-based modeling and deformation concepts [Igarashi et al. 1999; Nealen et al. 2007] directly into OverCoat. Level-of-detail, which has played a prominent role in non-photorealistic rendering [Markosian et al. 2000], is only loosely handled by our system by scaling brush sizes according to view distance. Future work could target more direct incorporation of level-of-detail into the implicit canvas.

Our current system accommodates only static paintings. This enables many interesting applications, such as interactive comic books and children's stories, or 3D concept art. An exciting direction for future research is the animation of paintings created in OverCoat. Paint strokes could be transformed according to animated proxy objects using space deformation techniques or by exploiting the point-to-surface correspondence given by the distance field. Ideally, the user could bring paintings to life directly within OverCoat by using the painting interface to author animations.

Acknowledgements

We would like to thank Daniel Teece for his support of our research, and Maurizio Nitti and Alessia Marra for their artistic input and the "Cat and Mouse", "Angry Bumble Bee", and "Wizard vs. Genie" example artwork.

References

BAERENTZEN, J. A., AND AANAES, H. 2005. Signed distance computation using the angle weighted pseudonormal. *IEEE Transactions on Visualization and Computer Graphics 11* (May), 243–253.

BAXTER, W., CHU, N., AND GOVINDARAJU, N. 2010. Project gustav: immersive digital painting. In *ACM SIGGRAPH 2010 Talks*, ACM, 41:1–41:1.

BERNHARDT, A., PIHUIT, A., CANI, M.-P., AND BARTHE, L. 2008. Matisse: Painting 2D regions for modeling free-form shapes. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling*, 57–64.

BOOKER, P. 1963. *A history of engineering drawing*. Chatto & Windus.

BOURGUIGNON, D., CANI, M.-P., AND DRETTAKIS, G. 2001. Drawing for illustration and annotation in 3d. *Computer Graphics Forum 20*, 3, 114–122.

BOUSSEAU, A., KAPLAN, M., THOLLOT, J., AND SILLION, F. X. 2006. Interactive watercolor rendering with temporal coherence and abstraction. In *Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*, 141–149.

COHEN, J. M., HUGHES, J. F., AND ZELEZNIK, R. C. 2000. Harold: A world made of drawings. In *Proceedings of the 1st international symposium on Non-photorealistic Animation and Rendering*, 83–90.

DURAND, F. 2002. An invitation to discuss computer depiction. In *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, ACM, 111–124.

FRISKEN, S. F., PERRY, R. N., ROCKWOOD, A. P., AND JONES, T. R. 2000. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, 249–254.

GOTTSCHALK, S., LIN, M., AND MANOCHA, D. 1996. Obbtree: A hierarchical structure for rapid interference detection. In *Proceedings of SIGGRAPH 96*, Computer Graphics Proceedings, Annual Conference Series, 171–180.

HAEBERLI, P. E. 1990. Paint by numbers: Abstract image representations. In *Computer Graphics (Proceedings of SIGGRAPH 90)*, 207–214.

HART, J. C. 1994. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer 12*, 527–545.

HERTZMANN, A. 2003. A survey of stroke-based rendering. *Computer Graphics and Applications, IEEE 23*, 4 (july-aug.), 70 – 81.

IGARASHI, T., MATSUOKA, S., AND TANAKA, H. 1999. Teddy: A sketching interface for 3d freeform design. In *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, 409–416.

JUDD, T., DURAND, F., AND ADELSON, E. 2007. Apparent ridges for line drawing. *ACM Transactions on Graphics 26*, 3 (July), 19:1–19:7.

KALNINS, R. D., MARKOSIAN, L., MEIER, B. J., KOWALSKI, M. A., LEE, J. C., DAVIDSON, P. L., WEBB, M., HUGHES, J. F., AND FINKELSTEIN, A. 2002. Wysiwyg npr: Drawing strokes directly on 3d models. *ACM Transactions on Graphics 21*, 3 (July), 755–762.

KARPENKO, O., HUGHES, J. F., AND RASKAR, R. 2002. Free-form sketching with variational implicit surfaces. *Computer Graphics Forum 21*, 3, 585–594.

KATANICS, G., AND LAPPAS, T. 2003. Deep Canvas: Integrating 3D Painting and Painterly Rendering. In *Theory and Practice of Non-Photorealistic Graphics: Algorithms, Methods, and Production Systems*, ACM SIGGRAPH 2003 Course Notes.

KEEFE, D. F., FELIZ, D. A., MOSCOVICH, T., LAIDLAW, D. H., AND LAVIOLA, JR., J. J. 2001. Cavepainting: a fully immersive 3d artistic medium and interactive experience. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, ACM, 85–93.

KEEFE, D., ZELEZNIK, R., AND LAIDLAW, D. 2007. Drawing on air: Input techniques for controlled 3d line illustration. *IEEE Transactions on Visualization and Computer Graphics 13*, 1067–1081.

KOWALSKI, M. A., MARKOSIAN, L., NORTHRUP, J. D., BOURDEV, L., BARZEL, R., HOLDEN, L. S., AND HUGHES, J. F. 1999. Art-based rendering of fur, grass, and trees. In *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, 433–438.

LU, J., SANDER, P. V., AND FINKELSTEIN, A. 2010. Interactive painterly stylization of images, videos and 3d animations. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, 127–134.

MARKOSIAN, L., KOWALSKI, M. A., TRYCHIN, S. J., BOURDEV, L. D., GOLDSTEIN, D., AND HUGHES, J. F. 1997. Real-time nonphotorealistic rendering. In *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, 415–420.

MARKOSIAN, L., MEIER, B. J., KOWALSKI, M. A., HOLDEN, L. S., NORTHRUP, J. D., AND HUGHES, J. F. 2000. Art-based rendering with continuous levels of detail. In *Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, 59–66.

MAYA. http://www.autodesk.com/maya.

MEIER, B. J. 1996. Painterly rendering for animation. In *Proceedings of SIGGRAPH 96*, Computer Graphics Proceedings, Annual Conference Series, 477–484.

MUDBOX. http://www.autodesk.com/mudbox.

NEALEN, A., IGARASHI, T., SORKINE, O., AND ALEXA, M. 2007. Fibermesh: Designing freeform surfaces with 3d curves. *ACM Transactions on Graphics 26*, 3 (July), 41:1–41:9.

PAINT EFFECTS. 2011. Painting in 3d using paint effects. In *Autodesk Maya Learning Resources*. http://download.autodesk.com/us/maya/2011help.

PENG, J., KRISTJANSSON, D., AND ZORIN, D. 2004. Interactive modeling of topologically complex geometric detail. *ACM Transactions on Graphics 23*, 3 (Aug.), 635–643.

PRAUN, E., HOPPE, H., WEBB, M., AND FINKELSTEIN, A. 2001. Real-time hatching. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, 579–584.

RADEMACHER, P. 1999. View-dependent geometry. In *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, 439–446.

RIVERS, A., IGARASHI, T., AND DURAND, F. 2010. 2.5d cartoon models. *ACM Transactions on Graphics 29*, 4 (July), 59:1–59:7.

SCHKOLNE, S., PRUETT, M., AND SCHRÖDER, P. 2001. Surface drawing: creating organic 3d shapes with the hand and tangible tools. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM, 261–268.

SCHMIDT, R., WYVILL, B., SOUSA, M., AND JORGE, J. 2005. Shapeshop: Sketch-based solid modeling with blobtrees. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling*, 53–62.

TEECE, D. 2000. Animating with expressive 3d brush strokes (animation abstract). In *Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, ACM.

TOLBA, O., DORSEY, J., AND MCMILLAN, L. 2001. A projective drawing system. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, ACM, 25–34.

WILLATS, J. 1997. *Art and representation: new principles in the analysis of pictures*. Princeton University Press.

WYVILL, G., MCPHEETERS, C., AND WYVILL, B. 1986. Data structure for soft objects. *The Visual Computer 2*, 4, 227–234.

ZBRUSH. http://www.pixologic.com/zbrush.