Eurographics Symposium on Point-Based Graphics (2004)
M. Alexa and S. Rusinkiewicz (Editors)

Interactive 3D Painting on Point-Sampled Objects

Bart Adams[1] Martin Wicke[2] Philip Dutré[1] Markus Gross[2] Mark Pauly[3] Matthias Teschner[2]

[1]Katholieke Universiteit Leuven† [2]ETH Zürich‡ [3]Stanford University§

Abstract

We present a novel painting system for 3D objects. In order to overcome parameterization problems of existing applications, we propose a unified sample-based approach to represent geometry and appearance of the 3D object as well as the brush surface. The generalization of 2D pixel-based paint models to point samples allows us to elegantly simulate paint transfer for 3D objects. In contrast to mesh-based painting systems, an efficient dynamic resampling scheme permits arbitrary levels of painted detail.

Our system provides intuitive user interaction with a six degree-of-freedom (DOF) input device. As opposed to other 3D painting systems, real brushes are simulated including their dynamics and collision handling.

Categories and Subject Descriptors (according to ACM CCS): I.3.4 [Computer Graphics]: Graphics Utilities, I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling

1. Introduction

For many years, digital painting has been of major interest in computer graphics. Numerous approaches were proposed to realistically represent brushes and to model their behavior and interaction with canvas. Recently, such models for painting on 2D canvases have been extended to 3D objects. Very often, 3D painting systems employ polygonal meshes or spline patches to represent the underlying 3D geometry. By establishing some mapping between a 2D parameter domain and the 3D surface, appearance attributes, resulting from paint operations, can be stored separately in texture maps. Once created, these texture maps can be reprojected onto the object surface.

This separation of geometry and appearance entails various inherent drawbacks: the surface parameterization required to connect the two domains unavoidably leads to distortions degrading the visual quality of the 3D painting. In addition, the uniform resolution of the texture map makes it difficult to handle spatially varying levels of painted detail. Most often, a brute force global upsampling is applied to accommodate high resolution strokes. While local parameterizations and surface patching offer potential solutions, optimal patch layout and texture packing can be cumbersome. Furthermore, discontinuities arising at the patch boundaries are difficult to cope with.

In this paper, we provide a solution to the aforementioned limitations which is entirely based on point-sampled geometry. By representing both the object and the brush surface as collections of point samples, we remove the separation between appearance and geometry. All relevant attributes and parameters, such as paint pigments, color, spatial position, and normals, are stored along with the sample.

This conceptual generalization of 2D pixel-based paint models to 3D geometry allows us to elegantly simulate paint transfer by immediate access of pigment properties stored in the samples. In addition, our point-based model can be resampled dynamically and adaptively to store appearance detail across a wide range of scales. Since the paint transfer is handled locally between brush and surface samples, texture parameterization and patching become obsolete. Our approach permits painting onto irregularly sampled object surfaces without distortions or visual artifacts.

Based on this sampled representation we built a prototype framework for interactive 3D painting. Our system supports a variety of paint effects, including paint diffusion, gold, chrome, and mosaic paint, and renders the objects in high quality. For intuitive 3D user interaction we added a haptic feedback model and a six DOF input device.

The remainder of this paper is devoted to the description of the technical details of the approach. After discussing

† email: {barta,phil}@cs.kuleuven.ac.be
‡ email: {wicke,grossm,teschner}@inf.ethz.ch
§ email: mapauly@graphics.stanford.edu

related work, we give an overview in Section 3. Next, we present our brush model, including dynamics, collision handling, and haptic feedback in Section 4. Paint transfer is described in Section 5 and the dynamic resampling is discussed in Section 6. Section 7 addresses implementation details. Finally, we demonstrate the performance of our method and present results in Section 8.

2. Related Work

Point-Sampled Geometry. Points have shown to be very effective as a display primitive for high quality rendering of complex objects. Rusinkiewicz and Levoy [RL00] use a bounding sphere hierarchy for progressive rendering of large objects. Alexa et al. [ABCO*01] use a dynamic resampling strategy to obtain high display quality. Zwicker et al. [ZPvG01] further increase the rendering quality by introducing the Elliptical Weighted Average (EWA) filter in the point-based rendering pipeline.

Point-sampled surfaces are also used for modeling and editing. Pauly et al. [PKKG03] are able to perform large free-form deformations on point-sampled geometry. Both Pauly et al. [PKKG03] and Adams and Dutré [AD03] present algorithms to perform boolean operations on point-sampled objects. Central in both approaches is a dynamic resampling strategy.

In the context of appearance modeling, Pointshop 3D [ZPKG02] extends 2D photo editing functions to 3D point clouds. Zwicker et al. propose a set of tools to paint, filter and sculpt point-sampled objects. Painting is performed by projecting a brush footprint bitmap. Recently, Reuter et al. [RSPS04] developed a Pointshop 3D plugin to interactively texture an object using a point-sampled multiresolution surface representation. Photometric attributes are assigned to the point samples which result from sampling the 3D object in a preprocess. There is no resampling during interactive texturing and therefore texture detail is limited by the sampling resolution at the finest level. There is no brush or paint metaphor.

Virtual Painting. There are several 2D painting systems of which the work of Baxter et al. [BSLM01] is most related to this paper. They present a haptic painting system using deformable 3D brushes to paint on a 2D canvas. Thanks to the virtual brushes and a bidirectional paint transfer model the artist can achieve an expressive power similar to painting on real canvases. They also introduce various more advanced paint models [BLL03, BWL04]. An alternative brush and paint model is presented by Xu et al. [XLTP03]. They simulate clusters of hairs and use a diffusion process to transfer paint from brush to canvas. Several researchers [WI00, XTLP02, YLO02, CT02] propose other virtual brush models in the context of Chinese calligraphy. However, none of these brush models is employed in a 3D painting system.

Figure 1: *The user interface. The brush is controlled via a PHANToM Desktop haptic device. The object can be rotated using a SpaceMouse.*

Hanrahan and Haeberli [HH90] first suggested a 3D painting system, using a mouse to position the brush. Agrawala et al. [ABL95] color the vertices of a scanned object using a spherical brush volume. There is no remeshing and therefore the painted detail is limited by the original resolution. More recent painting systems [JTK*99, GEL00, KSD03] provide a haptic interface, but still use a sphere-shaped brush to apply paint to the object. In all these systems, color and material information is stored in fixed-sized textures, limiting the level of detail that the artist can apply.

To overcome these limitations, Berman et al. [BBS94] propose the use of images with different resolutions in different places, called *multiresolution images*, to represent 2D paintings with regions of varying levels of detail. In 3D, the *Chameleon* painting system [IC01] overcomes the limitiations of fixed-sized textures and predefined uv-mappings by automatically building a texture map and corresponding parameterization during interactive painting. By using different patches for different regions they allow for adaptively varying the painted level of detail. However, even these elaborate techniques cannot fully solve the parameterization problems inherent in texture mapping. DeBry et al. [DGPR02] as well as Benson and Davis [BD02] solve the parameterization problems by storing paint properties in adaptive octrees, thus only creating texture detail when necessary. Painting is limited to a 2D plane which is then projected onto the surface. The resulting color attributes are stored in the octree.

3. System Overview

User Interface. We developed a user interface which enables the artist to manipulate the brush, mix paint, move the object and apply paint in an intuitive manner (see Figure 1). In our painting system, the virtual brush is positioned using a six DOF input device, such as the PHANToM

Desktop (Sensable Technologies) which also provides haptic feedback to the user. The object can be translated and rotated using a mouse or a six DOF input device such as the SpaceMouse (3DConnexion). The artist can choose different paint types, such as aquarelle and oil paint. A virtual palette is used to mix paint. We vary the reflectivity of the paint and use environment mapping to enhance realism. The brush casts shadows on the palette and the object, giving visual depth cues. An advanced point renderer [ZPvG01] is used to obtain high quality images of the painted object.

Object Representation and Dynamic Resampling. We avoid problems, such as texture distortion and patch discontinuities, which are often apparent in polygon-based painting systems, by using a point-sampled surface representation and a dynamic sampling strategy. The fundamental idea is to upsample the object locally if necessary or downsample the object locally if possible. For example, if the artist paints a thin line, our system locally upsamples the surface. If later the artist overpaints this small stroke with a large brush, the system locally downsamples the affected areas without losing any geometric information.

Our system handles regularly or irregularly sampled objects, given that the samples adequately capture the object geometry. Each surface sample carries geometric attributes such as position $\mathbf{x}$, normal $\mathbf{n}$ and radius r, as well as a set of appearance attributes which represent the paint pigments: dry paint attributes $\mathbf{A}_d$, wet paint attributes $\mathbf{A}_w$ and wet paint volume per unit area V_w. The point samples are stored in a kd-tree which is used for efficient collision detection and neighbor collection during painting.

Virtual Brushes. We model virtual brushes using a point-sampled surface, wrapped around a mass-spring skeleton. The skeleton is used to model the dynamics of the brush, the surface samples store paint information. This flexible brush model enables us to define different brush types of various sizes and resolutions. Collision detection between the brush and complex 3D objects is possible at high rates. Although more accurate, simulating individual brush hairs or clusters is too expensive to compute for haptic feedback. Our brush model gives us all the flexibility we need for a plausible painting simulation.

When zooming in on the object surface, the brush is scaled down. As a result, the brush sampling density increases relative to the object sampling density. This enables the artist to apply fine detail to the object. Since both the object and the brush are represented with point samples, an elegant implementation of bidirectional paint transfer can be realized.

Paint Model. Based on [BSLM01], our system handles different paint types such as aquarelle, oil paint, metallic or otherwise reflective paint as well as other surface types such as mosaic or beaten gold. In order to model this broad variety of paint types, the paint model stores color as well as other attributes, such as diffusion coefficients, reflectivity, shininess,

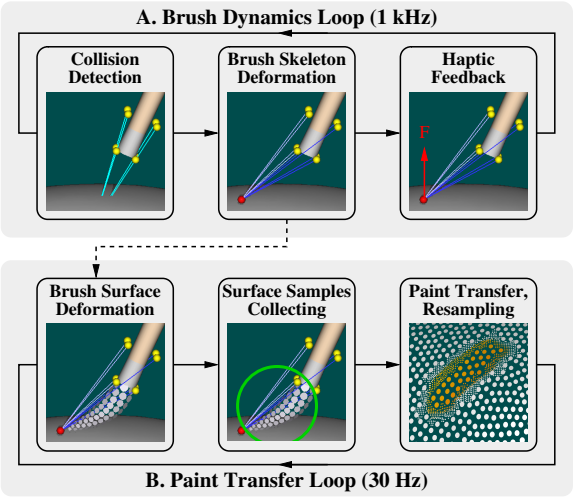

Figure 2: *Top row: brush dynamics loop. When a collision occurs, the brush tip is projected onto the object surface and the brush skeleton is deformed accordingly. The resulting force acting on the handle is sent to the haptic device. Bottom row: paint transfer loop. The brush point samples are deformed according to the skeleton deformation. After collecting the relevant surface samples paint is transfered between the brush and the surface samples.*

and procedural small-scale geometry manipulations. The latter are used to simulate the geometric detail often found in oil or acrylic paint, or planar patches typically found in mosaics. We chose the paint model from [BSLM01] as a basis for ours as it allows for bidirectional paint transfer while being computationally cheap. Any other paint transfer model that is defined on pixels can be used as well.

Decoupled Haptics. To guarantee the required 1 kHz update frequency of the haptic device we decouple the force computation from the rest of the application. Only operations that are necessary to simulate the dynamic behavior of the brush, such as collision detection and skeleton deformation, are performed in the *brush dynamics loop* (Figure 2, A). All other (more costly) operations, such as brush surface deformation, paint transfer and dynamic resampling, are performed in the *paint transfer loop* (Figure 2, B) which runs at the 30 Hz display frequency.

4. Brush Model and Haptic Display

To model a virtual brush, we have to devise a geometric representation for the brush surface as well as a physics-based model to simulate the dynamic behavior. We use point samples storing paint information to represent the surface of the brush. These samples are defined relative to a mass-spring skeleton which is used to simulate the dynamic behavior. The force resulting from the dynamics is directly used for haptic feedback.

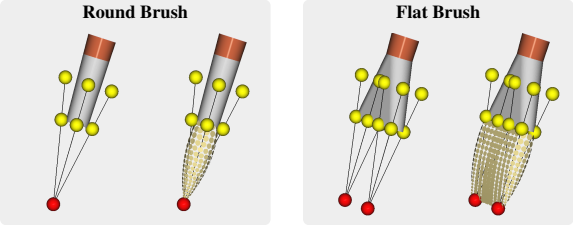

Figure 3: *The brush is represented as a point-sampled surface wrapped around a mass-spring skeleton. Left: round brush consisting of one basic skeleton. Right: flat brush modeled using two tips.*

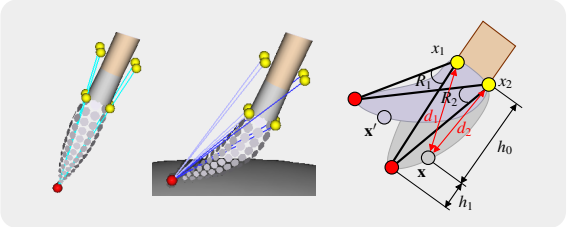

Figure 4: *Left: undeformed brush. Middle and right: when the brush is deformed, the new positions* $\mathbf{x}'$ *of the brush samples are computed from the original positions and the rotation of the springs.*

4.1. Brush Dynamics

Mass-Spring Skeleton. To simulate the dynamic behavior of a brush, we use a mass-spring skeleton similar to [BSLM01]. The basic skeleton block is a *tip skeleton* consisting of a single mass (the *tip*) and eight springs attached to *handle points*. We model round brushes using one tip. Flat brushes have skeletons consisting of several tips (see Figure 3). More exotic brushes can be modeled using an arbitrary mass-spring skeleton. To simulate viscosity, the brush simulation is heavily damped. Leap-Frog integration [Hoc70] is used to solve the differential equations governing the brush behavior. Even with larger skeletons, this method is fast enough to run in the brush dynamics loop. With an update frequency of 1 kHz, the simulation proved robust for all user manipulations.

Collision Handling. The brush skeleton should never penetrate the object. Therefore, we perform a collision detection query for each skeleton mass point. In order to implement a variant of force shading as proposed by [RKK97], we compute smoothed surface normals and penetration depths during collision detection. Given the position $\mathbf{x}$ of the mass point, we search for the N (typically $N = 10$) closest object samples with positions $\mathbf{x}_i$ and normals $\mathbf{n}_i$ and compute a weighted average penetration depth d and local surface normal $\mathbf{n}$ as follows:

$$d = \sum_{i=1}^{N} w_i \cdot \mathbf{n}_i \cdot (\mathbf{x}_i - \mathbf{x}), \tag{1}$$

$$\mathbf{n} = \sum_{i=1}^{N} w_i \cdot \mathbf{n}_i, \tag{2}$$

$$w_i = \frac{d_{max} - d_i}{\sum_{j=1}^{N} d_{max} - d_j}, \tag{3}$$

where $d_i = \|\mathbf{x}_i - \mathbf{x}\|$ and $d_{max} = \max d_i$. This weighting scheme provides a smooth interpolation of normals over the surface. When a collision is detected, i.e. $d > 0$, the mass point is constrained to the surface of the object:

$$\mathbf{x} \leftarrow \mathbf{x} + d \cdot \mathbf{n}/\|\mathbf{n}\|. \tag{4}$$

Depending on its tangential speed, we also apply static or dynamic friction.

Haptic Display. The resulting force $\mathbf{F}$ acting on the handle can be computed by adding up the forces exerted by all springs in the brush skeleton that are attached to the handle. The torque resulting from the simulation can be used for haptic feedback, if supported. When the user zooms in on some part of the object, the transformations returned by the haptic device are scaled down to allow for controlled movements even in a very small field of view. The forces sent to the haptic device are scaled up proportionately in order to maintain the illusion of a hard surface.

Thin sheets are a problem for the haptic rendering algorithm, since the springs cannot generate sufficient force to keep the user from pushing the brush through a thin part of the object. This problem disappears when zooming in, mainly due to the scaling of the haptic device movement.

4.2. Brush Surface Representation

The samples representing the surface of the brush carry attributes similar to the object samples, namely position $\mathbf{x}$, orientation $\mathbf{n}$, radius r, paint volume per unit area V_b and paint attributes $\mathbf{A}_b$. The undeformed brush surface is manually modeled to resemble a real brush.

To deform the brush surface, we apply a combination of linear blend skinning [LCF00] and the free-form deformation method for point-sampled geometry presented in [PKKG03] (see Figure 4). Given the position $\mathbf{x}$ of the point sample on the undeformed brush, we compute the distances d_i from the point $\mathbf{x}$ to the $N = 4$ closest handle points. When the brush is deformed, the spring attached to each of those handle points defines a rotation R_i. Applying the rotations R_i to the point $\mathbf{x}$ yields new point positions $R_i(\mathbf{x})$. The final position $\mathbf{x}'$ of the deformed sample is obtained as a convex combination of the original position $\mathbf{x}$ and the rotated positions $R_i(\mathbf{x})$:

$$\mathbf{x}' = (1 - \alpha) \cdot \mathbf{x} + \alpha \cdot \sum_{i=1}^{N} w_i \cdot R_i(\mathbf{x}), \tag{5}$$

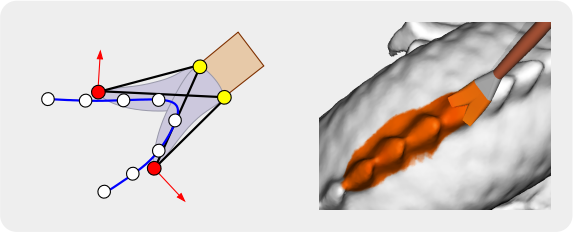

Figure 5: *Left: if two or more skeletons are constrained to differently oriented surface parts, the brush splits. Right: brush splitting on the back of the Stanford Dragon.*

$$\alpha = h_0/(h_0+h_1), \tag{6}$$

$$w_i = \frac{1}{N-1}\cdot(1-\frac{d_i}{\sum_{j=1}^{N} d_j}), \tag{7}$$

where h_0 and h_1 denote the distance of the brush sample to the handle and the tip respectively. Thus, a brush sample close to the skeleton tip will deform more than a sample close to the handle. We apply the same transformation to compute the deformed normal $\mathbf{n}'$ of the brush sample.

4.3. Brush Splitting

When a brush with several tips moves over a highly curved surface, two tips may be constrained to differently oriented surfaces (see Figure 5). We detect this by comparing the local surface normals computed for each of the tips. If the local surface orientation differs significantly, i.e. when the angle between the two normals is more than 60 degrees, the brush is split and interior brush samples are activated to represent the two brush head volumes. This way we can paint on highly curved surfaces such as the back of the Stanford Dragon (see Figure 5). As will be explained in the next section, we compute paint transfer separately for each of the brush parts.

5. Paint Transfer

When a collision between the brush and the object surface is detected, paint is transferred from the brush to the surface and vice versa. Inspired by the orthogonal projection mapping presented in [ZPKG02] we construct a local planar approximation of the object surface, the *paint buffer*. We splat both the object samples and the brush samples into the paint buffer, which serves as a common projection plane. Reprojecting the paint buffer results in new object samples storing the painted detail. The different steps performed during a paint event are explained below (see Figure 6).

A. Collecting Surface Samples. In a first step, we collect all object samples that might be affected by the brush. As the points are stored in a kd-tree, this can be efficiently implemented by performing a range query corresponding to the bounding sphere of the brush samples (see Figure 6, A). The bounding sphere is computed from the current positions of the brush samples using the smallest enclosing ball algorithm presented in [Gar99].

B. Paint Buffer Construction. After collecting the relevant samples, we construct the paint buffer in a plane defined by the average normal of the collected surface samples (see Figure 6, B). The position of the plane is arbitrary. Its dimensions are chosen so that the bounding sphere of the brush head projects completely inside the buffer. If the sampling density of the brush is higher than the sampling density of the surface, one brush sample should project to approximately one pixel in the paint buffer. The paint buffer resolution is chosen accordingly. Typical paint buffer resolutions range from 30 by 30 to 50 by 50 pixels. If however the local sampling density of the object is higher than the brush sampling density, the paint buffer pixel size is adjusted to the object sample size. This guarantees that texture detail and geometric features are preserved during painting.

Note that the paint buffer plane is usually a good approximation to the area of the surface that is touched by the brush. If the curvature of this region is very high, the brush is very likely to split. In this case we use multiple paint buffers, one for each skeleton tip. The same holds when the brush enters a crease as the surface normals computed for the tips will differ significantly. This way, we minimize distortion when painting on highly curved surfaces.

C. Surface Sample Projection. We use a software implementation of the EWA splatting algorithm to rasterize the front-facing samples into the paint buffer (see Figure 6, C). By using the EWA splatting algorithm we avoid aliasing artifacts in all attributes. Note that the orthogonal projection simplifies the splatting algorithm. The following point attributes are written to the paint buffer:

- depth d (distance to the projection plane),
- normal $\mathbf{n}$,
- paint attributes $\mathbf{A}_w$ and $\mathbf{A}_d$,
- wet paint volume per unit area V_w.

D. Brush Sample Projection. Similar to the object samples, the back-facing brush samples are projected into the paint buffer (see Figure 6, D). Only fragments with a depth greater than the depth already stored in the paint buffer are written. These fragments represent parts of the brush that penetrate the surface of the object. The following brush sample attributes are written to the paint buffer:

- penetration depth d_p,
- paint attributes $\mathbf{A}_b$,
- paint volume per unit area V_b.

Here the penetration depth d_p denotes the signed distance between the surface of the brush and the surface of the object at the relevant pixel.

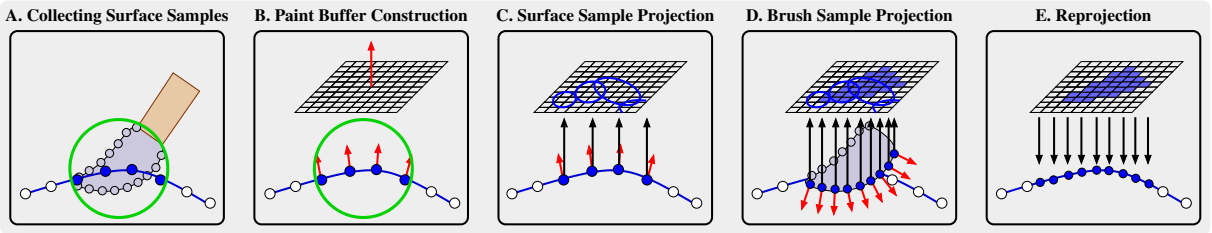

Figure 6: *Different steps performed during a paint event. A. Collecting the surface samples. B. Constructing the paint buffer orthogonal to the average surface normal. C. EWA splatting of the collected surface samples. D. EWA splatting of the back-facing brush samples. E. Reprojection of pixels in the paint buffer to surface samples.*

E. Reprojection. We compute bidirectional paint transfer using the transfer model proposed in [BSLM01] to determine the resulting color in the paint buffer. After computing paint transfer, we reproject the newly painted pixels onto the object surface (see Figure 6, E). If the brush sampling rate is higher than the local object sampling rate, the object surface is locally upsampled using our dynamic upsampling operator which is described in detail in Section 6. In order to add geometric effects to the paint type, the normal and position values of the new samples can be modified by the paint model.

6. Dynamic Sampling

To preserve the detail that is potentially created with a high resolution brush on a less densely sampled object, the object surface has to be upsampled in order to accomodate the texture detail. Conversely, if there is no texture detail to justify the high sampling density, the object surface needs to be downsampled to remove redundant information. These dynamic resampling operators are based on the assumption that the surface of the original object is adequately sampled.

Up- and downsampling is facilitated by a two-level data structure (see Figure 7). The original object samples are stored in a static kd-tree. They may never be deleted in order to retain the original geometric information. However, during resampling, they can be marked as *dead*, meaning they will not be rendered. These samples can have *children*, i.e. new samples replacing or complementing the *parent*. Children are uniquely assigned to one parent, which is in general the closest sample in the kd-tree. Children are stored in a list belonging to the parent, and are instantly deleted when marked as dead. This way we avoid updating the kd-tree, which is too costly during interactive painting.

Upsampling. The upsampling operator needs to be invoked whenever a brush paints a less densely sampled object. It locally upsamples the area in which more texture detail needs to be stored (see Figure 8, B). In order to find the affected area, the paint buffer storing the paint information is analyzed.

Since the brush can have a very uneven color distribution, the complete brush footprint, i.e. the area where the

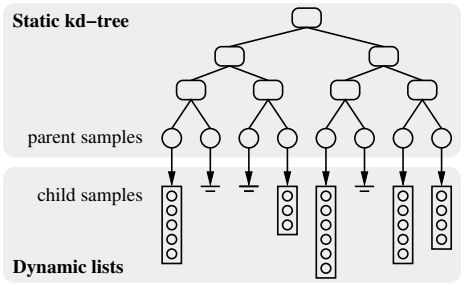

Figure 7: *Two-level data structure. Top: the original surface samples are stored in a static kd-tree. Bottom: when new samples are added, they are stored in a list belonging to the closest object sample in the kd-tree.*

brush touches the surface, needs to be resampled. In the paint buffer, this area consists of all pixels that have a penetration depth $d_p > 0$. Each of these pixels is reprojected onto the object and becomes a child of the closest sample in the kd-tree. The position of the new sample can be computed using the paint buffer plane position and orientation as well as the pixel's depth value. Its normal and the paint attributes can be directly read from the paint buffer. The new sample's radius equals the diagonal of one paint buffer pixel. All object samples fully or partly covered by these new samples are marked as dead. If some of the killed samples were only partly covered by the reprojected pixels, we additionally reproject pixels that are only touched by the projection of killed samples (see Figure 9).

Note that child samples never become the parent of new samples: they are instantly deleted and the new sample is added to the list of the closest sample in the kd-tree. This way our system can handle multiple overpaints without the need to reorganize a hierarchical data structure.

Downsampling. Reprojection of the paint buffer may result in neighboring samples having the same appearance attributes. This usually happens when overpainting fine detail with a large brush. To remove this unnecessary detail, we apply the following simple downsampling operator. For each parent sample, we compute the deviation of paint attributes of its child samples. When this deviation is below a

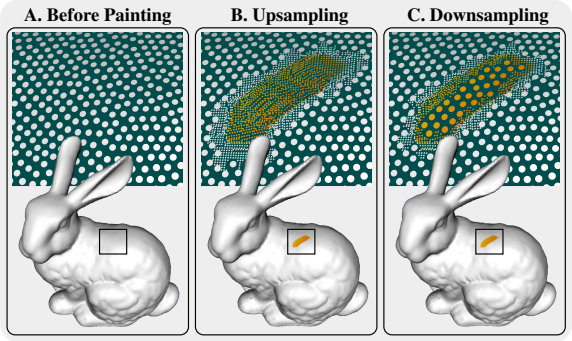

Figure 8: *The sampling density is locally adapted to accurately represent the texture detail. Left: sampling density of the Stanford Bunny. Middle: upsampling to represent the painted detail. Right: downsampling of child samples.*

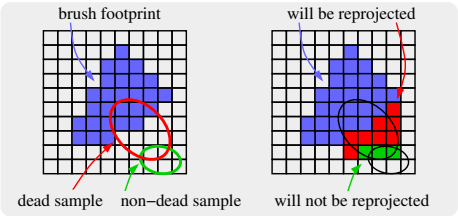

Figure 9: *Analyzing the paint buffer. The footprint of the brush is shaded blue. Left: samples projected onto a pixel that is touched by the brush are marked as dead. Right: pixels affected by the brush (blue pixels) will result in new samples. Additionally, in order to avoid holes, we reproject pixels touched by a dead sample and not touched by any non-dead sample (red pixels).*

threshold, we remove the child samples, resurrect the parent if necessary and set the parent's attributes to the average of all its children's attributes. Reasonable values of the deviation threshold are between 0.95 and 0.99, depending on the amount of smoothing the downsampling operator is allowed to perform. To ensure that no geometric detail is lost, we remove only child samples. Thus, we maintain an adequately sampled surface at all times. An example of downsampling is illustrated in Figure 8, C.

7. Implementation

Rendering. High quality renderings of the painted objects are generated with a software implementation of the EWA splatting algorithm [ZPvG01]. Each paint event only affects a local part of the surface. Thus, we can achieve high frame rates by only locally updating the rendered image, unsplatting samples that have been killed and splatting newly added or resurrected samples.

When rotating or translating the object, the system switches to a hardware implementation of the EWA splatting algorithm similar to [BK03] for performance reasons. We use the software renderer during painting because the hardware implementation suffers from quantization artifacts ocurring when locally updating the rendered image.

The brush casts a shadow on the object and the palette. Shadow mapping can be integrated into the hardware renderer without requiring an additional rendering pass. To render shadows using the software renderer, we save the rendered image to a texture and add the shadow using an additional hardware rendering pass performing the shadow test in a fragment program. Environment mapping enhances realism for reflective paint types.

Paint Effects. In order to give the artists a variety of paint types, we modeled various paint effects. We do not limit the paint attributes to color information. Reflectivity makes chrome or gold paint possible and shininess can be used to model matt paint or glossy polished surfaces.

The paint transfer model is allowed to modify the small-scale geometry of painted surfaces. A mosaic-like effect is achieved by setting the normal of newly created child samples to their parent's normal instead of blending it. When using gold paint combined with the mosaic effect, we obtain the appearance of beaten gold.

When painting with highly viscous paint, such as oil or acryl, the brush hairs leave an imprint in the paint. Although we are not able to model the complete geometric effect of adding layers of paint, we can model the surface structure. If the brush skeleton is aligned with the brush velocity, we slightly manipulate the surface normals of the new samples as to create the illusion of a hair imprint.

Diffusion is the most important feature of aquarelle. Our paint model stores diffusion coefficients and supports isotropic surface diffusion. Each surface sample $\mathbf{x}_i$ exchanges wet paint volume ΔV_w with other surface samples $\mathbf{x}_j$:

$$\Delta V_w = (V_w^i - V_w^j) \cdot e^{-\frac{d^2}{\bar{\mathbf{v}}^2 \cdot T^2}}, \tag{8}$$

where $d = \|\mathbf{x}_i - \mathbf{x}_j\|$ is an approximation of the geodesic distance between the two sample points, $\bar{\mathbf{v}}^2$ denotes the average squared particle speed and T is the elapsed time period. The wet paint attributes $\mathbf{A}_w$ are adjusted using the paint transfer rules. Because of the exponential decay of ΔV_w, we can restrict the number of samples $\mathbf{x}_j$ by only considering neighbor samples within a threshold distance to $\mathbf{x}_i$.

8. Results

Inspired by the *Art on Cows* project, we set up our own virtual *Art on Bunnies* project and asked a number of artists to paint the Stanford Bunny using our painting system. The artists all used the same irregularly sampled bunny model consisting of 97k point samples. We provided them with a set of 12 round and flat brushes. The painting system runs on a 3 GHz PC with a GForce FX 5900 graphics board.

A selection of the resulting bunnies is displayed in Figures 10, 11, 12 and 13 and in the accompanying video. Figure 10 shows the sampling density of the painted Day-and-Night Bunny. The sampling density is increased locally to preserve sharp painted edges. The dynamic resampling strategy also allows for fine painted detail such as the flowers and the bee in Figure 12. The entire bee covers an area about the size of a single point sample of the original model. Reflective paint was used for the Caesar Bunny (Figure 11). You can see diffusion effects on the Savannah Bunny (Figure 11). Note the imprints left by the virtual brush hairs in the painted water on the Beach Bunny (see Figure 13 and the video). Depending on the amount of detail, the resulting bunnies consist of 300k to 800k point samples.

One of the artists painted the Stanford Dragon (Figure 14). Environment mapping is used for the reflecting dragon ball. The eyes of the dragon are laid in mosaic.

9. Conclusion and Future Work

We presented a novel painting system for 3D objects. Our system provides virtual brushes, various paint types, and an intuitive user interface. In order to overcome parameterization problems of existing painting applications we employ a unified sample-based approach to represent geometry and appearance of the 3D object surface as well as the brush surface. Our paint transfer model locally approximates the object surface with one or more planes, also handling highly curved surfaces without distortions. Dynamic resampling of the point-sampled object surface allows the artist to apply arbitrarily fine painted detail.

In our current implementation, collision handling of the brush is performed with respect to the original object geometry. However, user feedback suggests that the actual thickness of applied paint should be considered in order to be felt by the user. Therefore, we intend to integrate a height field to represent paint thickness. This height field would also support the incorporation of more advanced paint transfer models [BLL03]. User feedback has also shown that more intuitive depth cues should be provided. Although our system gives depth information such as the brush shadow, it might be useful to add stereo vision.

Acknowledgments. We thank Michael Waschbüsch for his work on the renderer and the artists Silke Lang and Christian Ratti for their enthusiasm and feedback. The first author is funded as a Research Assistant by the Fund for Scientific Research - Flanders, Belgium (Aspirant F.W.O.-Vlaanderen).

References

[ABCO*01] ALEXA M., BEHR J., COHEN-OR D., FLEISHMAN S., LEVIN D., SILVA C. T.: Point set surfaces. In *Proceedings of IEEE Visualization 2001* (2001), pp. 21–28. 2

[ABL95] AGRAWALA M., BEERS A. C., LEVOY M.: 3d painting on scanned surfaces. In *1995 Symposium on Interactive 3D Graphics* (Apr. 1995), pp. 145–150. 2

[AD03] ADAMS B., DUTRÉ P.: Interactive boolean operations on surfel-bounded solids. *ACM Transactions on Graphics 22*, 3 (July 2003), 651–656. 2

[BBS94] BERMAN D. F., BARTELL J. T., SALESIN D. H.: Multiresolution painting and compositing. In *Proceedings of SIGGRAPH 94* (July 1994), Computer Graphics Proceedings, Annual Conference Series, pp. 85–90. 2

[BD02] BENSON D., DAVIS J.: Octree textures. In *Proceedings of ACM Siggraph 2002* (2002), pp. 785–790. 2

[BK03] BOTSCH M., KOBBELT L.: High-quality point-based rendering on modern gpus. In *Proceedings of Pacific Graphics 2003* (2003), pp. 335–343. 7

[BLL03] BAXTER W., LIU Y., LIN M. C.: *A Viscous Paint Model for Interactive Applications*. Tech. rep., University of North Carolina at Chapel Hill, 2003. 2, 8

[BSLM01] BAXTER W., SCHEIB V., LIN M. C., MANOCHA D.: Dab: Interactive haptic painting with 3d virtual brushes. In *Proceedings of ACM Siggraph 2001* (2001), pp. 461–468. 2, 3, 4, 6

[BWL04] BAXTER W., WENDT J., LIN M. C.: IMPaSTo, a realistic, interactive model for paint. In *Proceedings of the Third International Symposium on Non-Photorealistic Animation and Rendering (NPAR) for Art and Entertainment* (2004). to appear. 2

[CT02] CHU N. S.-H., TAI C.-L.: An efficient brush model for physically-based 3d painting. In *Proceedings of Pacific Graphics 2002* (2002), pp. 413–422. 2

[DGPR02] DEBRY D., GIBBS J., PETTY D. D., ROBINS N.: Painting and rendering textures on unparameterized models. In *Proceedings of ACM Siggraph 2002* (2002), pp. 763–768. 2

[Gar99] GARTNER B.: Fast and robust smallest enclosing balls. In *Proceedings of the European Symposium on Algorithms 1999* (1999), pp. 325–338. 5

[GEL00] GREGORY A. D., EHMANN S. A., LIN M. C.: inTouch: Interactive multiresolution modeling and 3d painting with a haptic inter-

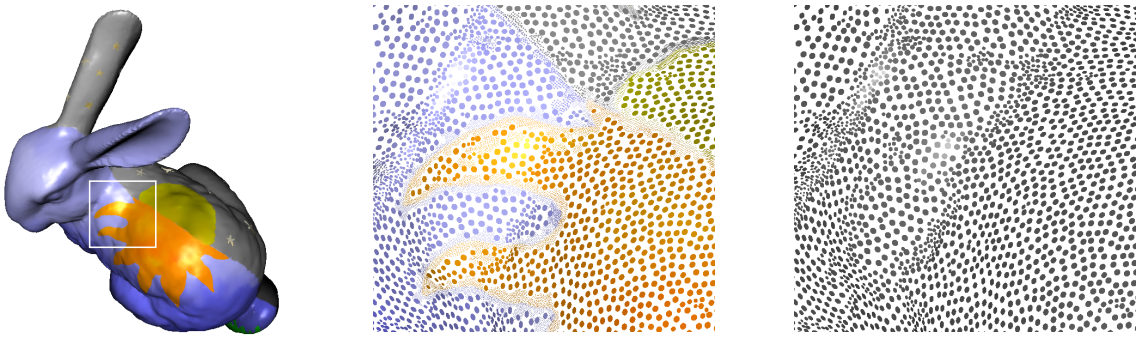

Figure 10: *Left: Day-and-Night Bunny. Middle: sampling density on the back of the painted bunny. The sampling density is higher where necessary to represent fine detail. You can see higher sampling rates around the sun's boundary. Right: original sampling density in the same region.*

face. In *Proceedings of IEEE Conference on Virtual Reality 2002* (2000), pp. 45–52. 2

[HH90] HANRAHAN P., HAEBERLI P.: Direct wysiwyg painting and texturing on 3d shapes. In *Proceedings of ACM Siggraph 1990* (1990), pp. 215–223. 2

[Hoc70] HOCKNEY R.: The potential calculation and some applications. In *Methods in Computational Physics* (1970), Alder B., Fernbach S.,, Rotenberg M., (Eds.), vol. 9, Academic Press, New York, pp. 136–211. 4

[IC01] IGARASHI T., COSGROVE D.: Adaptive unwrapping for interactive texture painting. In *2001 ACM Symposium on Interactive 3D Graphics* (Mar. 2001), pp. 209–216. 2

[JTK*99] JOHNSON D., THOMPSON II T. V., KAPLAN M., NELSON D. D., COHEN E.: Painting textures with a haptic interface. In *Proceedings of the IEEE Conference on Virtual Reality 1999* (1999), pp. 282–285. 2

[KSD03] KIM L., SUKHATME G. S., DESBRUN M.: Haptic editing of decoration and material properties. In *Proceedings of the Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems 2003* (2003), pp. 213–220. 2

[LCF00] LEWIS J. P., CORDNER M., FONG N.: Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of ACM Siggraph 2000* (2000), pp. 165–172. 4

[PKKG03] PAULY M., KEISER R., KOBBELT L. P., GROSS M.: Shape modeling with point-sampled geometry. In *Proceedings of ACM Siggraph 2003* (2003), pp. 641–650. 2, 4

[RKK97] RUSPINI D. C., KOLAROV K., KHATIB O.: The haptic display of complex graphical environments. In *Proceedings of ACM Siggraph 1997* (1997), pp. 345–352. 4

[RL00] RUSINKIEWICZ S., LEVOY M.: Qsplat: A multiresolution point rendering system for large meshes. In *Proceedings of ACM Siggraph 2000* (2000), pp. 343–352. 2

[RSPS04] REUTER P., SCHMITT B., PASKO A., SCHLICK C.: Interactive solid texturing using point-based multiresolution representations. In *Journal of WSCG 2004* (2004), vol. 12, pp. 363–370. 2

[WI00] WONG H. T., IP H. H.: Virtual brush: a model-based synthesis of chinese calligraphy. *Computers & Graphics 24*, 1 (2000), 99–113. 2

[XLTP03] XU S., LAU F. C., TANG F., PAN Y.: Advanced design for a realistic virtual brush. In *Proceedings of Eurographics 2003* (2003), pp. 533–542. 2

[XTLP02] XU S., TANG M., LAU F., PAN Y.: A solid model based virtual hairy brush. *Computer Graphics Forum 21*, 3 (2002), 299–308. 2

[YLO02] YEH J., LIEN T., OUHYOUNG M.: On the effects of haptic display in brush and ink simulation for chinese painting and calligraphy. In *Proceedings of IEEE Pacific Graphics 2002* (2002), pp. 439–441. 2

[ZPKG02] ZWICKER M., PAULY M., KNOLL O., GROSS M.: Pointshop 3d: an interactive system for point-based surface editing. In *Proceedings of ACM Siggraph 2002* (2002), pp. 322–329. 2, 5

[ZPvG01] ZWICKER M., PFISTER H., VAN BAAR J., GROSS M.: Surface splatting. In *Proceedings of ACM Siggraph 2001* (2001), pp. 371–378. 2, 3, 7

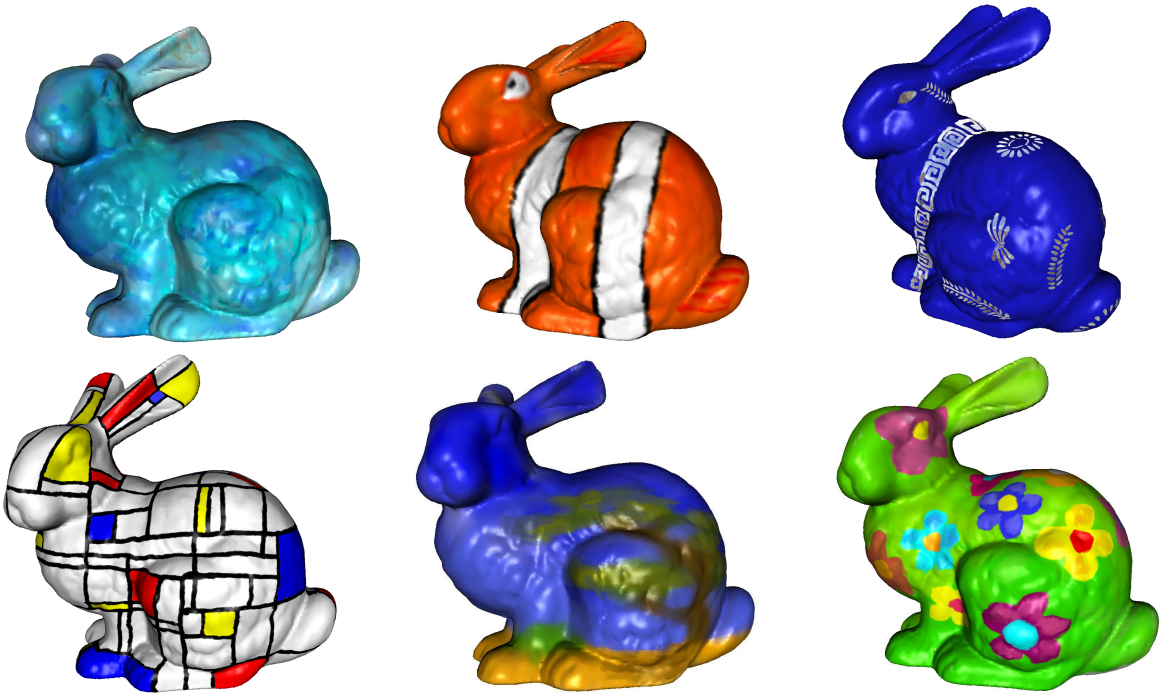

Figure 11: *Several bunnies painted using our painting system. From left to right and top to bottom: Cloud Bunny, Nemo Bunny, Caesar Bunny, Mondriaan Bunny, Savannah Bunny and Flower-Power Bunny.*

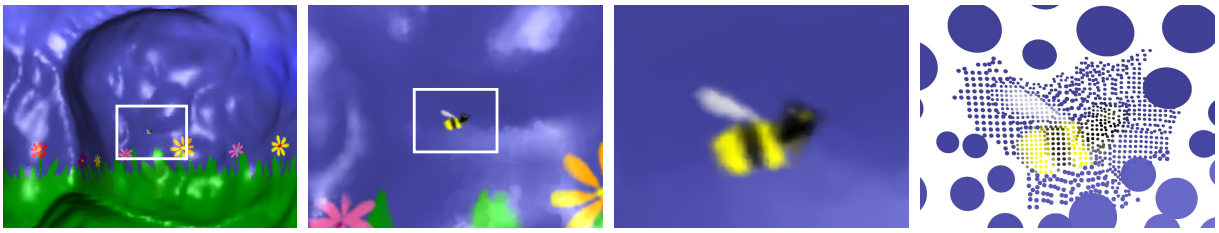

Figure 12: *Close-ups of the Day-and-Night Bunny. Note the very fine detail. Right: sampling density around the bee.*

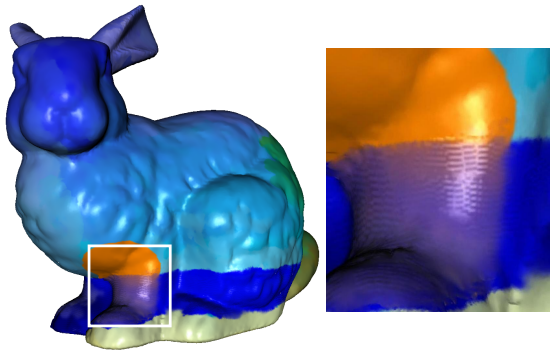

Figure 13: *The Beach Bunny. Right: geometric detail on the water.*

Figure 14: *Left: the Fire Dragon. Top right: close-up of the reflecting dragon ball. Bottom right: close-up of one of the eyes painted with the mosaic effect.*