A Non-Photorealistic Rendering Framework with Temporal Coherence for Augmented Reality

Jiajian Chen* Greg Turk† Blair MacIntyre‡

School of Interactive Computing
Georgia Institute of Technology

ABSTRACT

Many augmented reality (AR) applications require a seamless blending of real and virtual content as key to increased immersion and improved user experiences. Photorealistic and non-photorealistic rendering (NPR) are two ways to achieve this goal. Compared with photorealistic rendering, NPR stylizes both the real and virtual content and makes them indistinguishable. Maintaining temporal coherence is a key challenge in NPR. We propose a NPR framework with support for temporal coherence by leveraging model-space information. Our systems targets painterly rendering styles of NPR. There are three major steps in this rendering framework for creating coherent results: tensor field creation, brush anchor placement, and brush stroke reshaping. To achieve temporal coherence for the final rendered results, we propose a new projection-based surface sampling algorithm which generates anchor points on model surfaces. The 2D projections of these samples are uniformly distributed in image space for optimal brush stroke placement. We also propose a general method for averaging various properties of brush stroke textures, such as their skeletons and colors, to further improve the temporal coherence. We apply these methods to both static and animated models to create a painterly rendering style for AR. Compared with existing image space algorithms our method renders AR with NPR effects with a high degree of coherence.

Index Terms: H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems—Artificial, Augmented, and Virtual Realities; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual Reality

1 INTRODUCTION

A seamless blending of real and virtual worlds is often seen key to increased immersion and improved user experiences for AR. Photorealistic and non-photorealistic rendering (NPR) are two ways to achieve this goal. Non-photorealistic rendering creates an abstract version of both the real and virtual world by stylization, with the goal of making them indistinguishable. NPR hides unnecessary and overwhelming details of the world, and reveals and emphasizes important information of the scene. This can be particularly useful in some applications such as technical illustrations (e.g., AR/VR aided machine repair, or virtual medical surgery training), and certain AR games with artistic stylization.

A major challenge for NPR is maintaining temporal coherence. Rendered results are temporally coherent when each frame smoothly and seamlessly transitions to the next one without visual flickering and artifacts that distract the eye from smoothness. For example, temporal coherence is maintained in brush based NPR if the brush stroke located at a certain point in the image frame smoothly transitions to a corresponding point in the next one, without any jumping or flickering.

NPR algorithms with coherence are useful for both general computer graphics and AR/VR. A large number of NPR algorithms have been proposed to improve temporal coherence. While some of these algorithms use the geometric information of the models to maintain coherence, others analyze the whole video sequences with computer vision techniques to obtain better coherence.

Early NPR techniques for AR/VR simply rendered each AR frame independently and did not consider temporal coherence at all. The resulting videos usually appear to flicker and are visually distracting. Although many NPR algorithms with coherence have been proposed in the general graphics area, the characteristics of AR make it difficult to directly apply these existing algorithms in the AR domain. AR applications must run at interactive frame rates, cannot look ahead to future video frames, and have only limited information about the scene. The framework presented here satisfies the second and third conditions, but does not currently run at interactive frame frames. We discuss how this could be solved at the end of the paper.

The remainder of this paper is organized as follows. First, we review NPR algorithms with coherence in both the general graphics and AR/VR areas. Second, we propose an NPR framework with support for coherence using model-space information and describe its important components. Finally we show results and discuss the pros and cons of our rendering framework.

2 RELATED WORK

Image and video NPR has been attracting researchers' attention in the computer graphics community for decades. Gooch et al.'s book gives a comprehensive overview of NPR algorithms [10]. There are a large number of NPR algorithms for both still images and videos in the general graphics and the AR/VR literature. The most applicable approaches related to our work are video NPR and interactive NPR, in which coherence is a key problem. Based on the type of input data and the solutions for maintaining temporal coherence of the rendered results, we divide the existing algorithms in the video and interactive NPR literature into two main categories: algorithms in image space, and algorithms in model space.

In image space approaches the input is a sequence of camera images (i.e., video) or computer-synthesized images. We have only limited knowledge of the geometric structure of the scene. In contrast, we have the full 3D geometry of the scene as input in model space approaches. Various methods for maintaining temporal coherence have been studied in these two categories. Researchers also have proposed methods that use both the information from the images and the models to improve temporal coherence. Because the main contribution of our work is for rendering the 3D models, we will concentrate on model space algorithms in this review.

*e-mail:johnchen@cc.gatech.edu
†e-mail:turk@cc.gatech.edu
‡e-mail:blair@cc.gatech.edu

IEEE International Symposium on Mixed and Augmented Reality 2012
Science and Technology Proceedings
5 - 8 November 2012, Atlanta, Georgia
978-1-4673-4662-7/12/$31.00 ©2012 IEEE

2.1 NPR with Coherence in Model Space

We are focused on the NPR algorithms that can maintain temporal coherence for rendered animated 3D scenes. In the model space category, there are several major types of algorithms for maintaining coherence: particle based algorithms, pre-built non-photorealistic textures, and geometry analysis and brush parameterization based algorithms.

Barbara Meier first presented an influential particle based NPR system that achieves high quality temporal coherence for animated 3D scenes [28]. She formally identified the "shower door" effect (i.e., the rendered scene appears as if it were being viewed through textured glass) as a major challenge for achieving frame-to-frame coherence in animations. The undesired shower door effect is present in NPR algorithms because the brush textures are fixed to the view plane not to the animated surfaces at the 3D models. Her algorithm overcomes this problem by distributing brush anchors on surfaces and placing brush stroke textures at these anchors in the screen.

Meier's inspiring algorithm has been adapted and extended by many researchers to create various NPR effects with coherence for rendering 3D models. Kowalski et al. extended Meier's work and used stroke based procedural textures, called "graftals", to render fur, grass and trees without full geometry [24]. The key for maintaining temporal coherence in their approach is placing graftals with controlled screen-space density that matches the aesthetic requirements of stylization, and sticking them to surfaces in the scene. Markosian et al. proposed to place static graftals on surfaces during the modeling phase and did not redistribute the graftals at each frame [27]. Combing the work of Meier and Kowalski, Kaplan et al. proposed an interactive NPR system for model rendering with better coherence support [22]. All of these approaches are applied to complete video sequences and leverage the ability to look ahead in time past the current frame.

Pre-built non-photorealistic textures can also be used in a pure model space approach when the 3D geometry is given. These approaches create stylized NPR textures offline and map the textures to object surfaces. Some researchers such as Horry et al. [19], Wood et al. [36], and Buck et al. [3] have built hybrid NPR/IBR (i.e., image based rendering) systems where hand-drawn art was re-rendered for different views. These IBR based approaches directly use images created by artists to stylize the scenes for producing various rendering effects. Following the spirit of these IBR systems, Klein et al. used a variant of the mip-map texture called an "Art Map" to render a virtual environment with coherence [23]. In a preprocessing stage, they capture photos of a real or synthetic environment, map the photos to a coarse model of the environment, and run a series of NPR filters to generate textures. At runtime, the system re-renders the NPR textures over the geometry of the coarse model, and it adds dark lines that emphasize creases and silhouettes. Praun et al. presented a finer "Tonal Art Map" to extend Klein's method [30]. Their algorithm renders hatching strokes over surfaces.

In addition to the particle based algorithms and pre-built textures, there are still a large volume of papers that are dedicated to rendering 3D models with NPR styles with coherence. The algorithms for maintaining frame to frame coherence in some of these approaches are mainly based on analysis and parameterization of brushes and object geometry. For example, Kalnins et al. implemented the WYSIWYG NPR system [21]. Their system can produce a sketch and hatching style for models with the user input. Their algorithm parameterizes silhouette and crease lines to maintain coherence. Recently, Lu et al. proposed an algorithm to stylize 3D models together with background videos. They introduced the idea of geometry reprojection to generate geometric flow to help guide the movement of brush strokes [26].

2.2 Point Sampling on Surfaces

Generating 3D anchors on model surfaces is a key step for the particle based NPR algorithms, since their 2D projections are the locations for brush placement in the rendering stage. A more general version of this problem, point sampling on arbitrary surfaces that satisfies a certain distribution, is an important research area in computer graphics. It can benefit many graphics applications, such as texture mapping [32], non-photorealistic rendering [28], remeshing [31, 1], and point-based graphics rendering [12].

One popular basic solution for generating 3D samples on surfaces is to randomly generate 3D samples on triangles using barycentric coordinates in a manifold mesh. The probability of generating a new 3D sample point on a triangle is proportional to the ratio of the triangle area and the total mesh area. This solution can be applied to arbitrary surfaces.

Although the samples usually do not satisfy the exact spatial uniformity, this method provides a good initial sample set that can be refined by more complex sampling algorithms [2].

The quality of many sampling algorithms is often measured by the blue noise distribution. A sample set with the blue noise distribution means it has a uniform and unbiased distribution in the spatial domain, and it does not have low frequency noise and structured bias in the frequency domain. Blue noise sampling on surfaces are particularly interesting in graphics graphics. Pastor et al. proposed a point hierarchy structure for blue noise surface sampling [29]. Alliez et al. [1] and Li et al. [25] presented sampling methods with parameterizations. However these algorithms need to pre-generate data sets offline before the sampling process. Wei proposed a parallel dart throwing technique for blue noise sampling [34]. Wei's approaches inspired Bowers et al., who presented a parallel Poisson disk sampling scheme that uses the GPU to sample surfaces at interactive frame rates [2].

The algorithms for generating 3D samples, such as the basic sampling by triangle areas, and more advanced blue noise sampling as we discussed above, can be used for creating 3D anchors in brush based NPR. These existing algorithms produce samples on model surfaces that are evenly distributed in the spatial domain. However, we often need brush anchors that are evenly distributed in *image space* for the purpose of brush stroke placement in the rendering stage. Also, most of these blue noise sampling algorithms are slow and unsuitable for real-time applications. In other words, 3D anchors generated by these existing algorithms are usable, but not very suitable for brush based NPR in AR. To solve this problem, we propose an algorithm for generating 3D samples on surfaces whose projections are uniformly distributed in the screen space for a optimal brush stroke placement.

2.3 Challenge of Coherence for AR NRP

NPR is an important approach to blend video with computer generated graphics content. It can be used in many kinds of AR applications, and therefore has attracted researchers' attention in the AR/VR community. Quite a few NPR algorithms have been proposed and studied.

Perhaps the earliest work in AR NPR was done by Fischer et al. in 2005 [8, 9]. They argued that stylizing both the virtual and real content is an important alternative to photo-realism for creating user immersion in AR. They presented several NPR styles, such as a cartoon style, for AR applications. They leveraged the power of the GPU to detect strong edges in AR frames, and blended colors to create a cartoon-like stylization for AR. They used GLSL to achieve interactive frame rates.

Haller et al. compared photorealistic rendering and NPR for AR [13, 15]. He also presented a sketch style rendering for AR [14]. Their algorithms directly apply the NPR methods designed for still

images to image sequences in video. As a result the final rendered videos appear to flicker.

In our previous work, we presented a watercolor-like rendering style for AR [5]. Our algorithm uses the Voronoi diagram to tile AR frames in GLSL shaders to achieve a watercolor effect in interactive frame rates. Our algorithm also re-tiles Voronoi cells along strong edges at each frame to provide a certain degree of coherence for the rendered AR video. We also presented a painterly rendering algorithm in image space [6]. Our algorithm keeps coherence by detecting matched feature points between frames and warping the brush anchors with the feature triangles.

There are several differences between NPR for AR and NPR for video and 3D models. First, AR applications usually require near real-time performance. Second, we do not have any information beyond the current frame in AR. Third, we usually do not have detailed information about the scene geometry in the video. Fourth, we do have complete information about the virtual content, and partial spatial information of the scene from tracking. These unique characteristics of AR imply that existing video NPR techniques cannot be easily applied in the AR domain.

In this paper we propose a general NPR framework with support for coherence in model space for AR. We demonstrate this framework by creating a painterly rendering style for AR. Our algorithm generates 3D brush stroke anchors on surfaces, and dynamically changes the anchor density from frame to frame to maintain temporal coherence. We also present a general method of reshaping brush stroke textures between two frames. We choose the painterly rendering style in our system for demonstration, but the framework of maintaining temporal coherence can be extended to create other NPR styles (e.g., a sketch style AR for technical illustration by using hatching textures).

3 Overview of NPR Framework

The flowchart of our NPR framework in model space is shown in Figure 1 below.

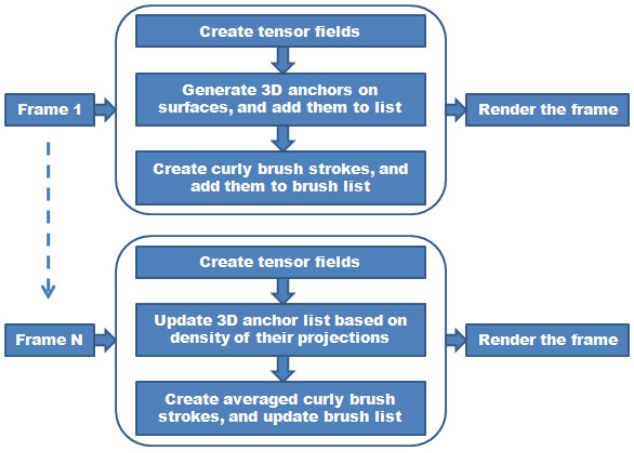

Figure 1: Flowchart of our NPR framework for AR.

We create a painterly rendering style for AR in our NPR framework. In brush-based NPR, the key for maintaining temporal coherence is to keep brush stroke textures smoothly transitioning from frame to frame. There are three major steps in our framework. First we create tensor fields on AR frames to guide the orientation of the brush strokes. The tensor values are interpolated by a tensor pyramid structure.

Second, to achieve a high degree of temporal coherence for the final rendered results, we distribute 3D brush anchors on the model surfaces. We update the 3D anchor list to maintain their density in screen from frame to frame. The 3D anchors are uniformly distributed in image space with our sampling algorithm, which is suitable for brush texture placement in NPR.

Third, we also keep the appearance of brush strokes coherent by averaging their properties between frames. The major properties of a single brush stroke include its skeleton and color. The algorithm in our method can smoothly blend curly brush strokes with a cubic B-spline representation. The second and third steps are the key for maintaining temporal coherence in our framework.

Finally we stylize the AR frame by placing brush strokes at the anchors on the screen. We will discuss the details of tensor field creation, brush anchor generation on surfaces, and property averaging of curly brush strokes in the following sections.

4 Tensor Field Creation

4.1 Background

We need the orientation of each pixel to guide our brush stroke directions. We create a tensor field instead of a vector field to obtain this pixel orientation. A major advantage of a tensor field over a vector field is that it allows sign ambiguity by a half turn [37]. We use the image gradient operator to compute the magnitude P and orientation θ of a pixel [6]. Once the edge magnitude P and orientation θ of a pixel are computed, this pixel's tensor field M is defined as follows:

$$M = P \begin{bmatrix} \cos 2\theta & \sin 2\theta \\ \sin 2\theta & -\cos 2\theta \end{bmatrix} \quad (1)$$

The eigenvector of this matrix M's major eigenvalue gives the orientation θ for this pixel. As mentioned above, this representation allows for sign ambiguity. Tensor fields are commonly used in scientific visualization [37] and video processing [20]. Some researchers also use tensor fields for different applications, such as street modeling [4].

As an alternative, we could define the following 2×1 vector:

$$v = \begin{bmatrix} P & \theta \end{bmatrix}^T \quad (2)$$

at each edge pixel. Furthermore, we could omit the magnitude component P and use only the angle θ at each edge pixel to create a scalar field, since the direction information is all that is needed by our algorithm. However, both the vector field and the scalar field of angles are directional, while the tensor field representation supports sign ambiguity, a major advantage of the tensor field. As a result, the linear interpolation of the vector field and the scalar field may result in singularities as intermediate values pass through zero. It is possible to avoid this problem by always capping the angle θ to the range $[-\pi, \pi]$ when we interpolate the values in the vector field. However this is not an elegant solution, with the resulting algorithm needing conditional statements to compute the new angle. On the other hand, the full tensor field representation allows us to interpolate new tensor values simply and elegantly, as *a linear combination of tensor values*. We can also apply filters (such as a Gaussian filter) to the tensor values by convoluting the filter kernel and the tensor values directly, without worrying about the sign ambiguity of the angle θ. This benefit becomes more important as we construct and use the tensor pyramid later in this section.

In the next section we will discuss how to treat those areas with weak orientations (far from edges).

4.2 Tensor Field Pyramid

Global radial basis interpolation can give the tensor values for non-edge pixels, but this interpolation is computationally expensive because it needs to loop through all strong edge pixels. Instead, we adopted our previous algorithm and use a tensor pyramid structure to compute the tensor values for an AR frame [6]. Our idea is

inspired by the method of interpolating sparse lumigraph data from Gortler et al. [11]. First we divide the original video frame to grid-based regions, and compute the weighted average tensor values in the regions that contain strong edge pixels. Second, we apply a low-pass Gaussian filter to the tensor values, and then sub-sample the grid to create a new tensor field at a higher level. We repeat the second step until we have reached the top level of the tensor pyramid.

Once the pyramid is created, we can look up a tensor value for any pixel in a frame. During the painting, if we need a tensor field of a pixel, first we check to see if it already has a tensor value assigned at the bottom level of the tensor pyramid (i.e., it is an edge pixel). If it does, we just use its value, otherwise we go up in the tensor field pyramid, and check if the corresponding region containing this pixel has a tensor value in this level, continuing until a value is found.

Since many NPR algorithms need direction information at the pixels, our tensor pyramid method can be easily extended to these algorithms. Tensor field creation in the CPU is computationally expensive, but it can be effectively accelerated by GPU based parallel computing techniques, such as CUDA or Stream.

5 Brush Anchor Generation on Surfaces

A key for keeping coherence in brush stroke based NPR is finding the correspondence of the brush stroke anchors and placing the brush strokes at these anchor positions. Placing brush anchors on a model's surface is a special case of the point sampling problem. Point sampling of a 3D model is an important topic in computer graphics. A widely-used algorithm is to generate samples on the triangles of a manifold surface using barycentric coordinates, as we described in the related work section. This approach randomly picks a triangle each time to create a new sample in it, with the probability proportional to the ratio of the triangle area and the total mesh area. This algorithm can produce random samples on the model surfaces but the projection of these samples are not evenly distributed in image space. Hence these samples are not ideal for brush based NPR algorithms. Also, the number of surface samples given by this algorithm is fixed. In NPR algorithms we often need to maintain the density of anchors in each region for placing brush strokes. For example, we need to add or remove anchors when the camera is moving close to or far away from the models we are painting.

To solve the problem we create an anchor list and update the list based on the density of their 2D projections on screen. Our new algorithm for brush anchor placement makes use of an image-space grid and back-projection onto the 3D model's surface. The size of grid cells in our paper is 8×8 pixels. For the first frame we pick jittered centers of grid cells in screen space and back-project them onto the model surface to obtain the initial 3D anchor list.

In the following frames, we dynamically add and remove 3D anchors in each frame to maintain the density of 2D projections of the 3D anchors in our list. If certain areas have too many anchors, we simply remove 3D anchors whose projections are inside the area. We keep updating the current anchor list from frame to frame.

After these 3D anchors are projected to the screen for a frame, it is still possible that certain areas are not covered by any anchors. Hence no brush strokes will be placed in these area and it will cause "holes" as a result. To solve this problem, we propose an algorithm to dynamically add new 3D anchors to our anchor list. These new 3D anchors are generated at each frame by back-projecting 2D points from image space into the model surface. The pseudo code of this algorithm is listed as follows.

```
Compute the 2D projection of each 3D anchor in the
    current anchor list
Update the number of anchors in each grid
for every grid in the screen
{
    if (the number of anchors in this grid <=
        lower_threshold)
    {
        Randomly pick a point P inside this grid;
        Back-project P to 3D model surfaces, and
            find the corresponding 3D location Q
            on mesh for P;
        Add Q to current anchor list;
    }
}
```

Please note that the back-projection of a point from 2D screen to a 3D mesh is a non-trivial problem, given the fact that the projection matrix returned by the AR tracker contains components for camera calibration. It is difficult to use the regular OpenGL routine to compute the intersection of the eye ray with mesh with this non-standard perspective matrix. To avoid this problem, we propose a much simpler and clearer way for the back-projection. The pseudo code of our back-projection is shown below.

```
Render the mesh with only color. The color value
    of a triangle is the index of it in the mesh.
// return a 3D anchor from mesh for a given grid
Back_Projection(grid index, mesh)
{
    Randomly pick P inside grid, get the pixel
        color at P;
    Decode the color to index i;
    Generate a random 3D point Q in i-th triangle
        in mesh;
    while (projection of Q is not inside the grid)
    {
        Generate a random 3D point Q in i-th
            triangle in mesh;
    }
    return Q;
}
```

In this back-projection routine we generate the random 3D point Q with barycentric coordinates (x,y,z) uniformly distributed in the i-th triangle as follows:

$$u,v = Unif[0,1] \quad (3)$$
$$x = 1-\sqrt{u} \quad (4)$$
$$y = \sqrt{u}(1-v) \quad (5)$$
$$z = \sqrt{u}v \quad (6)$$

Although our trial-and-error method has a loop, it is very efficient in most cases. Models we are working with usually have a large number of triangles. The projection area of a single triangle is usually very small in most camera scenarios. Hence we usually get a new anchor Q in only one trial in our testing. The back-projection method also by-passes the problem of computing intersections between the eye ray and triangles with a complex non-regular OpenGL perspective matrix.

This algorithm can be easily generalized for both static models and animated models. For static models, we store 3D positions of each anchor in the 3D anchor list. We use the color polygon ID to find the triangle index in the mesh, and then generate a new 3D anchor using barycentric coordinates. We update the 3D anchor list from frame to frame. For models with rigid body animation, the process is similar. We just need to apply the corresponding world view matrix to each anchor when we compute the 2D projection at each frame.

For models that are animated using mesh skinning, we store the triangle index and the barycentric coordinates of each anchor in

the 3D anchor list. To compute the projection of a 3D anchor at a certain frame, we need to compute the position of the three vertices in the triangle based on the corresponding bone transformation and bone weight matrix. We then compute the new 3D location of this anchor in the current frame using its barycentric coordinates inside this triangle, and finally compute its 2D projection on the screen. In the back-projection process, we first find the triangle index using color polygon ID, then generate a new anchor in the triangle and store its index and barycentric coordinates to the anchor list.

Figure 2 shows an alien spaceship, which is a static model, with over 7,000 anchors. The purple dots are the new anchors added for the current frame. The blue dots are existing anchors that are visible for the current frame in the current anchor list. Note that there are only a small number of purple anchors that are newly added to each frame, and most of anchors are blue ones after only a few "warm-up" frames. Also, newly added anchors are usually at the boundary of the model. This means that we achieve a satisfactory anchor density quickly in this algorithm. This is exactly what we want for obtaining a stable anchor list for coherent painting.

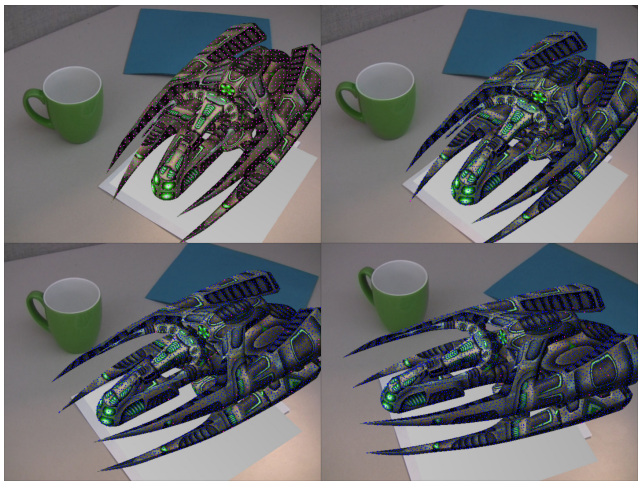

Figure 2: Brush anchor visualization for a static alien spaceship model. Four screens are created at frame 0, 20, 40, 60 from a video at 30 fps.

Figure 3 shows a running super hero model with skinning animation with over 1,000 anchors. Note that anchors stick to the model surface during the animation and that they are updated from frame to frame to maintain proper density on the screen.

This algorithm can also be applied to generate an arbitrary number of point samplings on any 3D models. Compared to the original point sampling algorithm based on triangle areas, and other more complex point sampling algorithms (e.g., blue noise sampling), our algorithm has several advantages. First, it generates 3D samples whose projections are evenly distributed in 2D image space. Other sampling algorithms produce samples that are uniformly distributed over a model's surface. As a result, their 2D projections on the screen can be sparse or overly dense at different regions. In our sampling algorithm, the 2D projections are immediately ready for later use in brush stroke placement. This is a desired feature for many NPR algorithms. Second, it is easier to add or remove anchors to maintain a proper density of brush stroke anchors on the 2D screen, since we adjust the 3D anchor list based on the density of their 2D projections from frame to frame. Third, it is possible to implement and speed up this algorithm with GPU based parallelization techniques such as CUDA. Our sampling algorithm is straightforward to parallelize, since the generation of an anchor in each region is independent of other anchors.

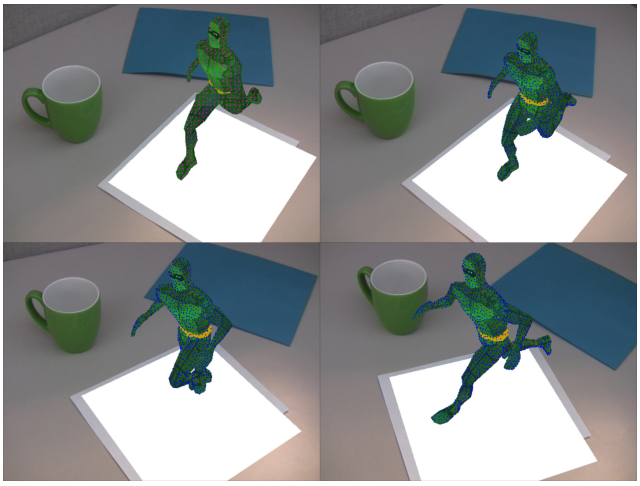

Figure 3: Brush anchor visualization for the running super hero model with skinning animation. Four screens are created at frame 0, 20, 40, 60 from a video at 30 fps.

Given the complete geometric information of the scene, our anchor generation algorithm generates *perfectly coherent* brush anchors from frame to frame that are also uniformly distributed in image space. As a result, this method provides a higher degree of coherence for brush strokes, compared to the anchor warping methods that are based on optical flow [16] or geometric flow [26].

6 Coherent Curly Brush Strokes

In this section, we describe how we render long, curly brush strokes. Moreover, we introduce a new method of maintaining the appearance of a stroke across frames as its path and color changes.

6.1 Brush Rendering

We create long curly brush strokes by elongating the brush skeleton in the tensor field direction. We also use bump mapping to simulate the lighting of textured brush strokes that is present in real world paintings. The algorithm is adapted and improved from our previous work [6]. The basic brush elongation algorithm has also been used in the work of Hertzmann et al. and Hays et al. [17, 18, 16]. The difference is that we choose cut-off angles as the stop condition for creating curly brush strokes with multiple segments. Hertzmann et al. use only the color difference as the stop condition for creating brush strokes. Hays et al. allow only straight brush strokes with a single segment.

6.2 Brush Shape Coherence

In order to keep temporal coherence we move the anchor points of brush strokes from the previous frame to the current frame. However, repainting a brush stroke at the corresponding anchor points at the current frame sometimes produces bad coherent results. The shape of a textured brush stroke in the final painting is given by its skeleton, which is a sequence of control points. The computation of brush stroke control points is decided by the local tensor field and the color difference of regions around the anchor. The shape of a curly brush stroke in two frames can vary significantly, even if its anchor points are coherent. Hence it is also necessary to change the shape of a brush stroke in the current frame to make it similar to the previous one. In additional to the shape of a brush stroke skeleton, we also need to add constraints to cap the change of a brush stroke color between two consecutive frames.

Our assumption is that a coherent brush stroke in two consecutive frames should have similar shapes and colors that

do not vary too much. We propose an algorithm to average the properties of two corresponding brush strokes, including their shapes and colors, between the current frame and the previous frame. We maintain a history list of properties of brush strokes at each 3D anchor in our algorithm. At each new frame, we first compute the new properties of a brush stroke (e.g., averaged color and skeleton control points) based on the information on the current frame. We then interpolate the new properties with the previous properties in the history list of this brush stroke. We use the interpolated results to paint the brush, and update the history list.

The interpolation of colors of two corresponding brush strokes can be computed in many ways. We choose the following weighted average function to produce satisfactory results:

```
mycolor_3d new_color = (1 - alpha) * current_color
    + alpha * previous_color;
```

We ran several experiments to pick the optimal value for the coefficient alpha. Choosing alpha as 0.95 gives coherent results for the brush stroke skeleton and color in general.

The interpolation of brush stroke control points is more complex. Given two sequences of control points, we first solve the basic case, in which the two sequences have an equal number of control points. We start from the first control point in the current frame. For each new segment, we average the direction from two brush strokes, and then elongate the brush stroke skeleton to the new ending point. Since the length of each segment is equal in our algorithm, we also need to normalize the direction at each step. We continue doing this until we reach the end of both sequences. In this paper we choose an alpha value of 0.95 to average the direction for each segment to create the new control point sequence.

Based on this basic case, we then need to consider the scenarios in which the two sequences have different numbers of control points. If the size of the brush in the current frame is smaller than it in the previous frame, we can still use the routine above to compute the new average brush skeleton by using only part of the control points at the beginning of the previous one. If the size of the brush in the current frame is large than it in the previous frame, we do not have the information (e.g., control points) beyond the last segment from the previous frame to match it in the current frame. One possible solution to solve this problem is to use only the information of the control points in the current frame to continue the brush skeleton elongation. However this solution is not ideal.

We propose a general solution to deal with the problem of averaging two sequences of control points in all types of scenarios. We force the elongation of each brush to the same number of control points at each frame for the purpose of interpolation, but we use only the valid part of the averaged brush control points for painting textures in each frame. There are several advantages to this solution. First, it avoids the problem of lack of information during interpolation. Second, it allows to handle several complex scenarios in a single universal routine as shown above. Third, it produces more clear and elegant code that has less condition branches and runs faster. The algorithm is illustrated in Figure 4.

Figure 5 visualizes the brush skeletons of AR frames using our algorithm. In these results the brush skeletons stick to the object surfaces, which gives a very high degree of coherence for brush skeletons in the painting. Figure 6 shows another example of coherent brush skeletons with an animated model.

7 Results and Discussion

We compare the coherence of rendered AR video produced by an image space algorithm adopted from our previous work [6], and by our new model space algorithm. We have tested our results on both static and animated 3D models. The models we choose to render are common in many AR applications, such as AR games, including humans, animals and vehicles.

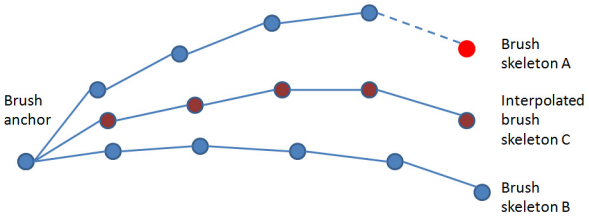

Figure 4: A general solution for averaging brush stroke skeletons with different sizes at two consecutive frames. The maximum size of a brush stroke is 5 segments and 6 control points in this case. Brush stroke skeleton A is from the previous frame. We force it to extend to 5 segments, and use only the first 4 segments in painting. Brush stroke skeleton B is initially created by local tensor fields from the current frame, and then averaged with A to interpolate C. C is the brush stroke skeleton we use for the final painting at this anchor point in the current frame. We also record C into the brush history list for this anchor in this frame.

Figure 5: Brush stroke skeleton visualization for the alien spaceship model. Four screens are created at frame 0, 20, 40, 60 from a video at 30 fps.

Figure 7 shows the rendered results of a static model produced by our model space algorithm with coherence.

Figure 8, 9 compares the rendered results produced by the image space algorithm and our new model space algorithm. Note the extremely stable and coherent brush textures in the model space images.

Figure 10, 11 gives another example for an animated model. Note the extremely stable and coherent brush textures in the model space result.

In our NPR framework we can obtain other NPR effect, such as sketch and hatching style, by using cross-hatching textures instead of brush stroke textures. Brush stroke anchors located on surfaces smoothly transition from frame to frame to provide temporal coherence. The final rendering is still composed in 2D since we place curly brush strokes at AR frames. Compared with painterly rendering algorithms in image space [6], our new model space framework provides better coherent brush strokes from frame to frame.

This algorithm can be used to render 3D graphics content in AR. A problem of using this method in model space is that in AR we usually do not have the accurate information of the scene

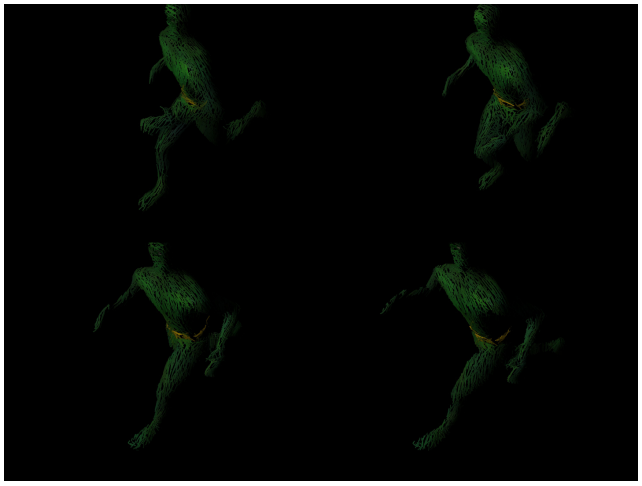

Figure 6: Brush stroke skeleton visualization for the running superhero model. Four screens are created at frame 0, 20, 40, 60 from a video at 30 fps.

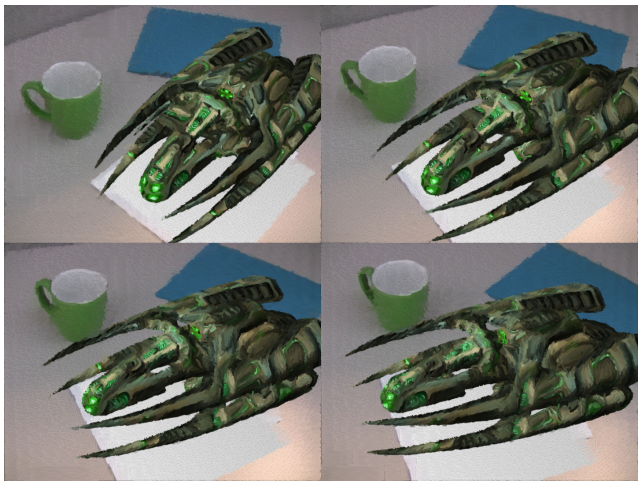

Figure 7: An alien spaceship painted with coherence by our new model space algorithm.

geometry except the 3D graphical models. If we can reconstruct the 3D scene from the video then this algorithm can be directly used to render the video content with coherence. However if the precise reconstruction cannot be done in interactive frame rates then we can use image space algorithms to process the video content with coherence, which leads to a hybrid combination for rendering AR. In this paper we rendered the 3D graphics model with our new model space method, and rendered video content with an image space method. Both of them fit into our general rendering framework with coherence support.

7.1 Performance Analysis

Our algorithm creates NPR effects for AR with a high degree of coherence. Although our algorithm is currently running offline, it could be further optimized to achieve interactive frame rates. We divide the rendering pipeline in our NPR framework into the following four stages to analyze the bottleneck of the performance: the tensor field creation, the brush anchor generation on the surface, the brush stroke averaging and the final rendering. The time cost of each stage for two scenes with the static models (the dragon

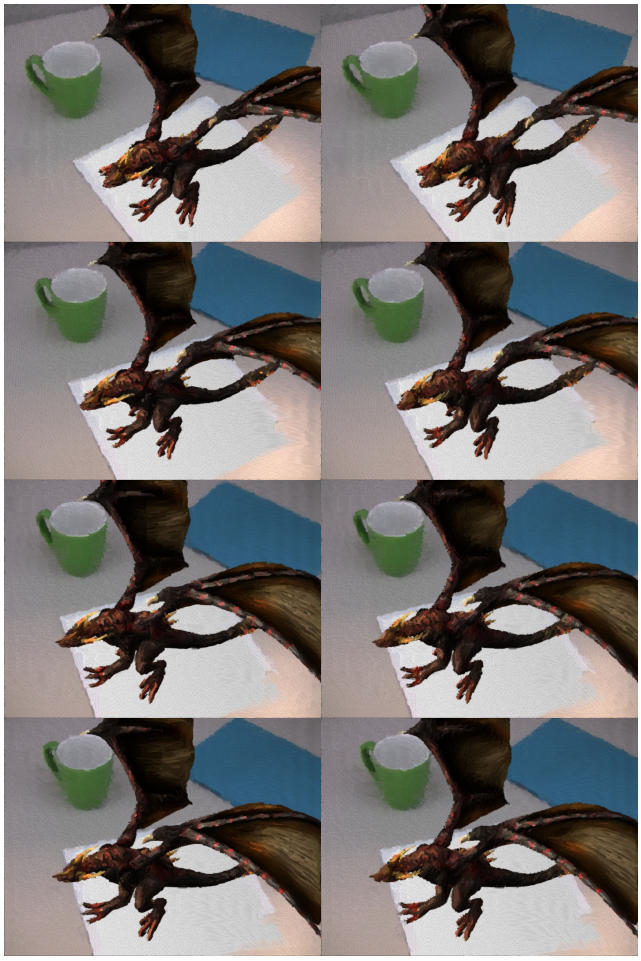

Figure 8: A painted dragon model. While each individual frame looks acceptable in both image space method (left) and our new model space method (right), careful examination reveals significantly greater brush stroke coherence between the model space images.

and the alien spaceship), and one scene with the animated model (the running superhero) is shown in Table 1. The resolution of AR frames is 640×480 pixels in these test scenes.

Table 1: Average Processing Time per AR frame (Unit: ms)

Test scene	Stage 1	Stage 2	Stage 3	Stage 4	Total
dragon	50	195	200	20	465
alien spaceship	50	210	200	20	480
running superhero	48	195	190	20	453

The final rendering is not the bottleneck of our system. Although models shown here have different number of triangles, in our model space algorithm the total number of brush stroke textures placed on screen is capped by the video frame size (640×480 pixels), the grid size (8×8 pixels), and brush density per grid (2 brush strokes per grid). This is another advantage of our anchor generation method. The total number of brush strokes is about $9,600$ per frame. Each brush stroke can have up to 20 quads, so the total number of polygons rendered per frame is about $192,000$.

The bottleneck of the performance is the tensor field creation,

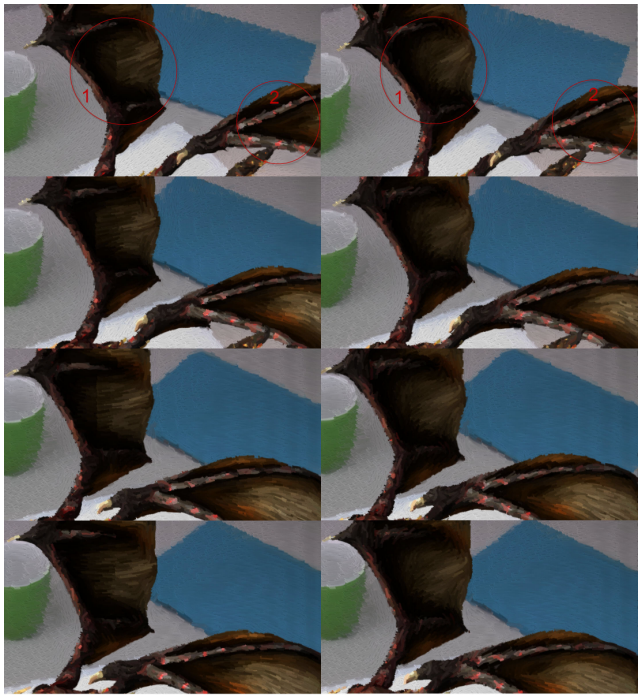

Figure 9: A zoomed in view of the dragon wing area. (Left: image space method. Right: our new model space method.) Although both results look good (the brush strokes in Circle 2 look coherent in the both methods), the brush strokes in Circle 1 are significantly more stable in the model space method than in the image space method.

the brush anchor generation and the brush stroke averaging, as these three stages are currently running on the CPU. The tensor field creation could be parallelized by CUDA. Similarly the brush anchor generation and brush stroke averaging methods can also be parallelized in a straightforward way. The generation of a new anchor is independent of other anchors. The averaging method for a single brush stroke on a certain anchor is entirely decided by its own brush stroke history list, which is also independent of other brush strokes. The use of parallelization may allow our algorithm to achieve interactive frame rates in the future.

7.2 Evaluation of Video Quality

In Section 7 we render the same video sequences using our original image space algorithm and our new model space algorithm. We mainly use subjective measurements to judge the quality of coherence in the renderings. Objective measurements are possible for the anchor distribution: our anchor generation algorithm gives the ground truth for anchors between frames, and it can be proven that their projections are evenly distributed in image space, compared with the basic point sampling algorithm. Given the 3D anchors on the model's surface, we can precisely compute the movement of these anchors from frame to frame in our model space algorithm. On the other hand, we use feature matching to warp anchors in our original image space algorithm. We used the result of anchor movement in our model space algorithm as the ground truth and tested the error rate in anchor warping for three scenes (the dragon, the alien spaceship and the running superhero). The comparison is shown in Table 2.

However, it is difficult to create objective measurements to compare the quality of the final rendering. We visualize the brush skeletons to show the coherence improvement, but the effect on human visual perception of coherence in the final painted

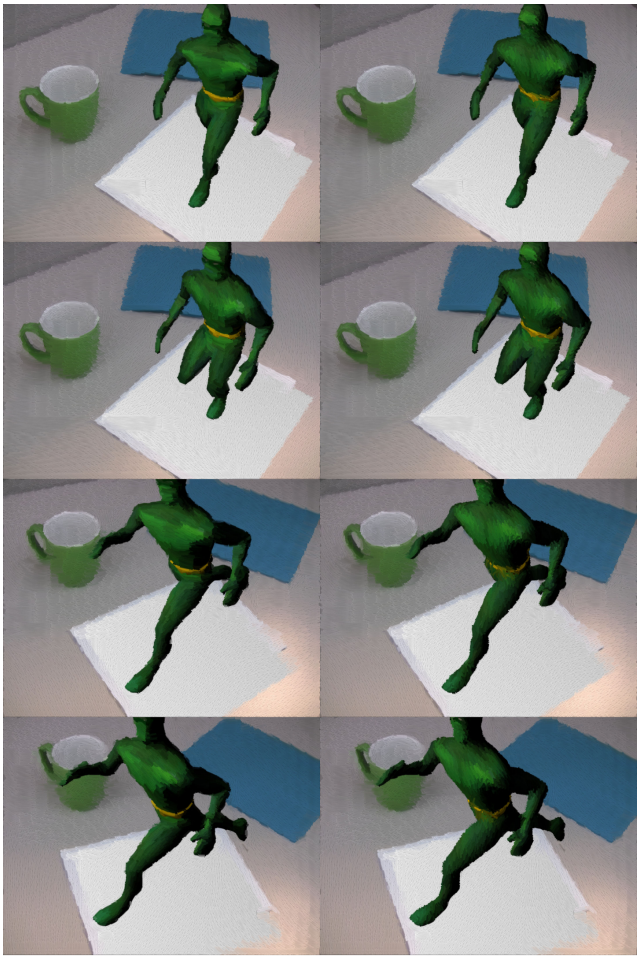

Figure 10: A painted running superhero model. (Left: image space method. Right: our new model space method.)

results is more difficult to quantify. Researchers have been used various subjective and objective metrics to help evaluate the perceptual quality of images and video. Traditional error summation methods have been used as an objective measurement for image quality by many researchers. Wang and Bovik presented a universal objective image quality index to assess the various image processing applications without employing the human visual system [33]. Claypool and Tanner asked test subjects to rate a *quality opinion score* ranging from 1 to 1000 after watching a video clip to measure the effects of jitter and packet loss on perceptual quality of streaming video [7]. Winkler and Mohandas gave a comprehensive review of subjective experiments and objective metrics of video quality and their uses [35].

We are focused on the perceptual quality and coherence of videos that are rendered by NPR algorithms. Many objective metrics such as the error summation cannot be used in NPR, as we usually do not have a corresponding stylized image or video to use as a ground truth to compare with our rendered results. Subjective experiments may be more useful to evaluate the quality of stylized videos in NPR. We used informal methods to compare the video quality in this comparison section, but a formal user study would be useful to better quantify the quality and coherence of our results. For example, a questionnaire or survey could be used to collect scores for perceived video quality and level of coherence of videos that are rendered by different algorithms with different settings. Interviews could be conducted after the study to collect user's feedback and

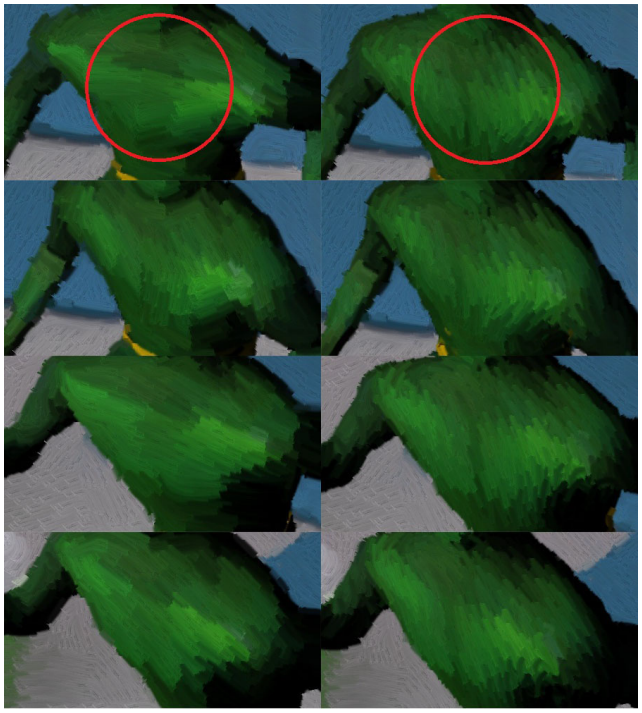

Figure 11: A zoomed in view of the chest area of the animated model. (Left: image space method. Right: our new model space method.)

Table 2: Comparison of Anchor Warping

Tested algorithms (scenes: dragon, alien spaceship and superhero)	Average number of anchors	Incorrectly warped anchors	Error rate in warping (%)
Image Space Algorithm	4300	1105	25.7
Model Space Algorithm	4300	0	0

comments. In addition to the subjective scores and comments, objective measurements such as the movement of a subject's eye gaze when they are watching rendered videos may be recorded to help evaluate which parts of the video sequence are attracting the user's attention (and thus may exhibit distracting artifacts). Although most of the collected data would still be subjective, this information may be useful to help better understand and evaluate the coherence problem in NPR for AR.

8 Conclusion and Future Work

Our new model model space NPR algorithm provides better coherent results than image space methods. Brush strokes located on polygons smoothly transition from frame to frame. However, the final rendering is still performed in 2D since we create a long curly brush strokes at AR frames. A problem of using this method as well as any other model space NPR algorithm is that in AR we usually do not have the accurate information of the scene geometry, except for the 3D graphical models. This algorithm will achieve the best coherence if it could be integrated with a scene reconstruction algorithm running at interactive frame rates.

We have proposed a general NPR framework with support of coherence for AR applications. Our method improves the rendering result by providing visual coherence. The final painting result is created by placing bump-mapping curly brush strokes on each AR frame. The placement of brush strokes is guided by tensor fields for each frame. Temporal coherence is maintained by moving the brush anchors with the model and reshaping the brush strokes from the previous frame to the current frame. The contribution of our algorithm has two parts. The first part is the anchor sampling algorithm. This algorithm maintains proper density of brush anchors on the screen, which is a desired feature for many NPR algorithms. It is also particularly suitable for AR, as we do not know the camera motion in advance. By controlling the brush anchor density from 2D and then back-projecting the 3D anchors onto surfaces, our algorithm achieves better coherence for the brush stroke placement. The second contribution is our method of averaging brush properties, including their skeletons and colors, to achieve better coherence in the final rendering. Compared with existing methods, our method allows us to smoothly blend curly brush strokes with a cubic B-spline representation. Our method can be extended to average other high dimension data as well.

Currently our algorithm creates a painterly rendering effect for AR with a high degree of coherence, but it renders the sequences offline. As we discussed in the previous section the rendering is not the bottleneck of our system. Our algorithm can be further optimized to achieve interactive frame rates. We analyzed the performance of our current algorithm. There are several improvements that can be done to speed up the rendering to achieve interactive frame rates in the future. For example, we can accelerate the creation of the initial tensor field and final tensor pyramid by using GPU-based parallelization such as CUDA. Another direction to explore is to study how to make better use of the information extracted from the video and graphics content. In this paper we use an image space algorithm to render the background video. The information extracted from video can be potentially useful to refine coherence for the graphics content, and vice versa. We would like to explore these two directions in the future.

References

[1] P. Alliez, M. Meyer, and M. Desbrun. Interactive geometry remeshing. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '02, pages 347–354, New York, NY, USA, 2002. ACM.

[2] J. Bowers, R. Wang, L.-Y. Wei, and D. Maletz. Parallel poisson disk sampling with spectrum analysis on surfaces. *ACM Trans. Graph.*, 29(6):166:1–166:10, Dec. 2010.

[3] I. Buck, A. Finkelstein, C. Jacobs, A. Klein, D. H. Salesin, J. Seims, R. Szeliski, and K. Toyama. Performance-driven hand-drawn animation. In *Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, NPAR '00, pages 101–108, New York, NY, USA, 2000. ACM.

[4] G. Chen, G. Esch, P. Wonka, P. Müller, and E. Zhang. Interactive procedural street modeling. In *ACM SIGGRAPH 2008 papers*, SIGGRAPH '08, pages 103:1–103:10, New York, NY, USA, 2008. ACM.

[5] J. Chen, G. Turk, and B. MacIntyre. Watercolor inspired non-photorealistic rendering for augmented reality. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology, VRST 2008, Bordeaux, France, October 27-29, 2008*, pages 231–234, 2008.

[6] J. Chen, G. Turk, and B. MacIntyre. Painterly rendering with coherence for augmented reality. In *IEEE ISVRI*, 2011.

[7] M. Claypool and J. Tanner. The effects of jitter on the peceptual quality of video. In *Proceedings of the seventh ACM international conference on Multimedia (Part 2)*, MULTIMEDIA '99, pages 115–118, New York, NY, USA, 1999. ACM.

[8] J. Fischer and D. Bartz. Real-time cartoon-like stylization of AR video streams on the GPU. In *Technical Report WSI-2005-18, University of Tubingen*, 2005.

[9] J. Fischer, D. Bartz, and W. Straßer. Artistic reality: fast brush stroke stylization for augmented reality. In *VRST*, pages 155–158, 2005.

[10] B. Gooch and A. Gooch. *Non-Photorealistic Rendering*. AK Peters Ltd, july 2001. ISBN: 1-56881-133-0.

[11] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The lumigraph. In *SIGGRAPH*, pages 43–54, 1996.

[12] J. P. Grossman and W. J. Dally. Point sample rendering. In *Rendering Techniques 1998*, pages 181–192. Springer, 1998.

[13] M. Haller. Photorealism or/and non-photorealism in augmented reality. In *ACM International Conference on Virtual Reality Continuum and its Applications in Industry*, pages 189–196, 2004.

[14] M. Haller and F. Landerl. A mediated reality environment using a loose and sketchy rendering technique. In *ISMAR*, pages 184–185, Washington, DC, USA, 2005. IEEE Computer Society.

[15] M. Haller and D. Sperl. Real-time painterly rendering for mr applications. In *International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia*, 2004.

[16] J. Hays and I. A. Essa. Image and video based painterly animation. In *NPAR*, pages 113–120. ACM, 2004.

[17] A. Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. In *SIGGRAPH*, pages 453–460, 1998.

[18] A. Hertzmann. Fast paint texture. In *NPAR*, page 91, 2002.

[19] Y. Horry, K.-I. Anjyo, and K. Arai. Tour into the picture: using a spidery mesh interface to make animation from a single image. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, pages 225–232, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[20] M. Kagaya, W. Brendel, Q. Deng, T. Kesterson, S. Todorovic, P. J. Neill, and E. Zhang. Video painting with space-time-varying style parameters. *IEEE Transactions on Visualization and Computer Graphics*, 17(1):74–87, Jan. 2011.

[21] R. D. Kalnins, L. Markosian, B. J. Meier, M. A. Kowalski, J. C. Lee, P. L. Davidson, M. Webb, J. F. Hughes, and A. Finkelstein. Wysiwyg npr: drawing strokes directly on 3d models. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '02, pages 755–762, New York, NY, USA, 2002. ACM.

[22] M. Kaplan, B. Gooch, and E. Cohen. Interactive artistic rendering. In *NPAR*, pages 67–74, 2000.

[23] A. Klein, W. Li, M. M. Kazhdan, W. T. Corrêa, A. Finkelstein, and T. A. Funkhouser. Non-photorealistic virtual environments. In *SIGGRAPH*, pages 527–534, 2000.

[24] M. A. Kowalski, L. Markosian, J. D. Northrup, L. Bourdev, R. Barzel, L. S. Holden, and J. Hughes. Art-based rendering of fur, grass, and trees. In *SIGGRAPH*, pages 433–438, 1999.

[25] H. Li, K.-Y. Lo, M.-K. Leung, and C.-W. Fu. Dual poisson-disk tiling: An efficient method for distributing features on arbitrary surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 14(5):982–998, Sept. 2008.

[26] J. Lu, P. V. Sander, and A. Finkelstein. Interactive painterly stylization of images, videos and 3D animations. In *Proceedings of I3D 2010*, feb 2010.

[27] L. Markosian, B. J. Meier, M. A. Kowalski, L. Holden, J. D. Northrup, and J. F. Hughes. Art-based rendering with continuous levels of detail. In *NPAR*, pages 59–66, 2000.

[28] B. J. Meier. Painterly rendering for animation. In *SIGGRAPH*, pages 477–484, 1996.

[29] O. M. Pastor, B. Freudenberg, and T. Strothotte. Real-time animated stippling. *IEEE Comput. Graph. Appl.*, 23(4):62–68, July 2003.

[30] E. Praun, H. Hoppe, M. Webb, and A. Finkelstein. Real-time hatching. In *SIGGRAPH*, page 581, 2001.

[31] G. Turk. Re-tiling polygonal surfaces. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '92, pages 55–64, New York, NY, USA, 1992. ACM.

[32] G. Turk. Texture synthesis on surfaces. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 347–354, New York, NY, USA, 2001. ACM.

[33] Z. Wang and A. Bovik. A universal image quality index. *IEEE Signal Processing Letters*, 9:81 – 84, 2002.

[34] L.-Y. Wei. Parallel poisson disk sampling. In *ACM SIGGRAPH 2008 papers*, SIGGRAPH '08, pages 20:1–20:9, New York, NY, USA, 2008. ACM.

[35] S. Winkler and P. Mohandas. The evolution of video quality measurement: From PSNR to hybrid metrics. *IEEE Transactions on Broadcasting*, 54(3):660–668, Sept. 2008.

[36] D. N. Wood, A. Finkelstein, J. F. Hughes, C. E. Thayer, and D. H. Salesin. Multiperspective panoramas for cel animation. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, pages 243–250, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[37] E. Zhang, J. Hays, and G. Turk. Interactive tensor field design and visualization on surfaces. *IEEE TVCG*, 13(1):94–107, 2007.