Textured Image Based Painterly Rendering Under Arbitrary Lighting

George ElKoura
University of Toronto
gelkoura@cs.toronto.edu

Abstract

Painterly rendering styles have received some attention lately as an interesting alternative to photo-realistic image processing and synthesis. State of the art painterly renderings, while producing results that superficially resemble certain artistic styles, such as impressionistic or expressionistic, fail to convey the implicit detail provided by the artist's choices of brushes and canvas.
In this paper we present techniques to simulate the effects of paint thickness and canvas texture as they interact under arbitrary lighting conditions. We also extend previous works by allowing the user to specify brush nibs of any size, shape and texture.

1. Introduction

Current techniques employed in image based painterly renderings attempt to simulate artistic styles by simulating a paint stroke's color, size, length, spacing and curvature [1]. However, physical paint artists have many more variables at their disposal. They can convey their art by their choice of paint thickness in each stroke, by their use of canvas texture, and the luckier ones even have the choice of lighting conditions under which their paintings are displayed.

We present techniques that extend previous work to enable the representation of paint thickness, canvas, and lighting. Moreover, we present a technique to generalize the shape of the brush nib.

In section 2 we discuss previous work. In section 3 we discuss the generalization of the brush nib's shape. Sections 4, 5 and 6 discuss adding paint thickness, canvas and lighting respectively. We discuss our implementation and results in sections 7 and 8. Finally we discuss some ideas for future work in section 9.

2. Previous Work

Hertzmann et al. presented techniques to generate image-based painterly renderings that make use of curved brush strokes and brushes of varying size. This work was later extended by adapting some video-coherence techniques that were originally presented by Litwinowicz in [9].

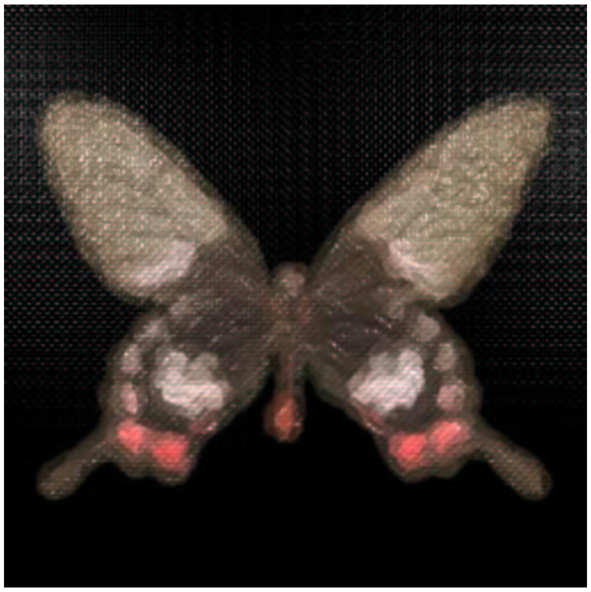
Figure 1 A painted butterfly rendered with brush depth and canvas thickness.

While this work allowed for varying brush sizes, it did not make provisions for varying the brush shape. We address the issues of using a generalized brush shape in this paper.

The idea of simulating more realistic brush strokes is not new. Lewis used signal processing approaches to give strokes a texture in 1984 [7], and Turner Whitted, in 1983, introduced the idea of sweeping an anti-aliased circle along a curve to produce anti-aliased lines [8]. Whitted also suggested that sweeping other kinds of primitives would produce other interesting effects, such as a line that looks as though it were made of glass for example.

Modern, commercial image processing software, such as Corel Corporation's PHOTO-PAINT, allows for the simulation of a variety of paint and canvas effects. The paint effects seem to be generated by processing the image with a convolution filter, and do not seem to attempt creating a realistic looking painting. Simulating the canvas in this package is done, it seems, by using the luminance of a pattern to emboss the image and cross-dissolve using user-specified parameters. Again, these techniques, though interesting, do not make any provisions for arbitrary lighting and do not attempt to achieve a 3D look.

Central to our technique is the use of displacement maps to give the appearance of depth to a flat image. Displacement maps are a generalization on the idea of texture maps and were presented as part of the Reyes rendering architecture [6], which was of great help in the development of this work.

3. Using a general brush nib

Hertzmann, in [1], uses an anti-aliased circle as the brush nib. We use a more general approach by allowing the user to use an arbitrary image as a nib. The nib template is used as a multiplicative filter on the choice of color obtained from the original picture.

The choice of a circle obviates the need to properly orient the brush nib along the direction of the stroke. However, when an arbitrary shape is used, orienting the nib becomes critical. We require that the nib template chosen by the user be given in a vertical position, in other words, it should be pointing up. Then we compute the angle between the vector (0, 1) and tangent vector to the curve at the point at which we wish to render the nib, and we rotate the nib by this angle.

The rotation is computed as

$$x' = xu_y - y\sqrt{1-u_y{}^2}$$
$$y' = x\sqrt{1-u_y{}^2} + yu_y$$

where (x, y) are the un-rotated coordinates, (x', y') are the corresponding rotated coordinates and u_y is the *y* component of the derivative of the curve at the point (x, y). This form, in general, is more efficient than explicitly computing the angle and the rotation matrix. We use NURBS for our stroke curves and computing the derivatives is fairly straightforward (see [4]).

Applying this rotation significantly improves the quality of the rendered strokes, this should be obvious in the general case. However, even for a circle with noise applied, not performing this corrective rotation will wash-out the effect of the constant noise. One way to get around this is to regenerate the noise on the brush nib for each dab, however this is unrealistic and does not represent the properties of brush thistles, which produce predictable dabs per stroke. Ideally, both would be implemented and left up to the user's discretion.

4. Paint Thickness

One approach to generating a 3D texture look that interacts with arbitrary lighting is to generate normal information for each stroke and render each stroke using simple bump-mapping techniques. This approach proved to be unsuccessful. The quality of the results is in direct proportion to the quality of the normals supplied by the user. The best way to produce these normals is to force the user to model a 3D brush nib. Needless to say, this puts too much onus on the user and provides for a very frustrating experience.

The second approach we tried, which proved to be much more successful, makes use of displacement maps [6]. Instead of generating normals, we generate a displacement map. The terms displacement maps and height maps are used interchangeably. The displacement map is like a texture map that contains the height of each pixel. When geometry is rendered with a displacement map, each point is displaced by the value indicated in the displacement map before finally being rendered on screen. This approach has many advantages over the initial "normals" technique. First, it uses one plane instead of three, thus saving memory. Secondly, it allows for self-shadowing which bump-mapping does not. In other words, under certain lighting conditions, some brush strokes will be shadowed by other strokes. Lastly, and most importantly, it is much easier for a user to supply a displacement map for a brush nib than it is to supply the normals for the nib.

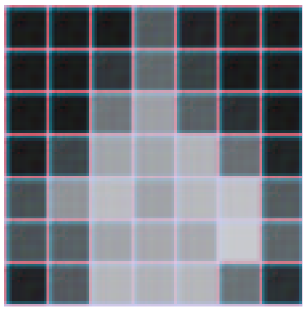

Figure 2 Example of a non-symmetric noisy brush nib.

The user only supplies one image that represents the brush nib. The luminance from the image is used for height information at each pixel, and the shape is determined by setting a threshold on luminance. That is to say, pixels with a luminance value above a certain threshold will contribute to the shape of the nib.

5. Canvas Texture

Much of a painting's highlights under certain lighting conditions result from the interaction of the paint with the canvas. It is thus important, in order to produce convincing results, to simulate the canvas as well as the paint thickness. Other attempts at simulating the paint

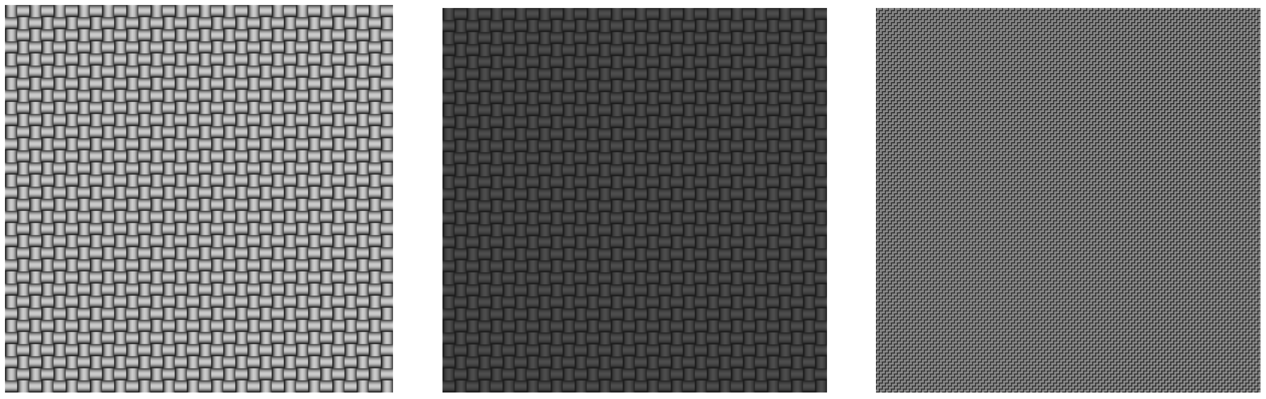
(a) (b) (c)

Figure 3 Canvas height maps generated using different parameters. (a) and (b) are 16×16 with height 0.7 and 0.2 respectively. (c) is 4×4 with height 0.5.

canvas use a patterned texture to brighten and darken certain parts of the image to produce the effect of a picture drawn on a canvas. Other luminance patterns produce pictures that have the appearance of being drawn on a brick wall for example [5].

The advantage of our technique is that the canvas only affects the height of the brush stroke, not the color of the brush stroke, as in real world paintings. The other techniques cannot be arbitrary lit and do not show the subtle specular highlights seen with our technique.

The canvas pattern in our system is only applied to the height-plane of the image and only appears during the rendering of the displacement map. Before any of the strokes are rendered, the canvas pattern is generated on the height-plane according to user settings. As paint stroke thickness is rendered into this plane, we add the stroke thickness to the height at that location (which includes the canvas and previous strokes). Our initial implementation added and capped the sum for each stroke. However, for complicated stroke patterns, we quickly reach the cap and we end up with a uniform (the maximum) height at each pixel and in essence losing all the height information. Instead we let the sum accumulate un-capped, and, after the painting is fully rendered, find the maximum height and divide all pixels by it. This normalization step produces great results.

The user can specify the size of the canvas grid and the height of the canvas. Only regular patterns are supported by the system, though there is nothing that precludes using an arbitrary, user-specified, canvas pattern.

The canvas is generated by the algorithm shown in **Algorithm 1**. The variables patch_height, patch_width and canvas_height are user supplied parameters. The variable base_height is used to make sure that the canvas has at least some height throughout. The algorithm simply breaks the image up into patches (of user-specified size) and uses quadratic roll-off to simulate the weaves.

6. Lighting the Painting

Lighting the painting, after the color plane and the height plane have been generated, is taken care of by the renderer.

We start with a single quadrilateral polygon that has the same aspect ratio as the original image, and a camera

```
numpatchx ← image_width/patch_width
numpatchy ← image_height/patch_height

for patchx = 0 to numpatchx
    for patchy = 0 to numpatchy
        for p = -patch_width/2 to patch_width/2
            for q = -patch_height/2 to patch_height/2
                if (patchx + patchy) is odd then
                    { horizontal patch }
                    $a \leftarrow$ patch_width$^2$/ 4
                    $c \leftarrow$ canvas_height
                    $v \leftarrow (-c/a)p^2 + c$;
                else
                    { vertical patch }
                    $a \leftarrow$ (patch_height$^2$/ 4
                    $c \leftarrow$ canvas_height
                    $v = (-c/a)q^2 + c$
                end

                pixel_height($p$, $q$) $\leftarrow v$ + base_height
            end
        end
    end
end
```

Algorithm 1 Routing for generating canvas height maps.

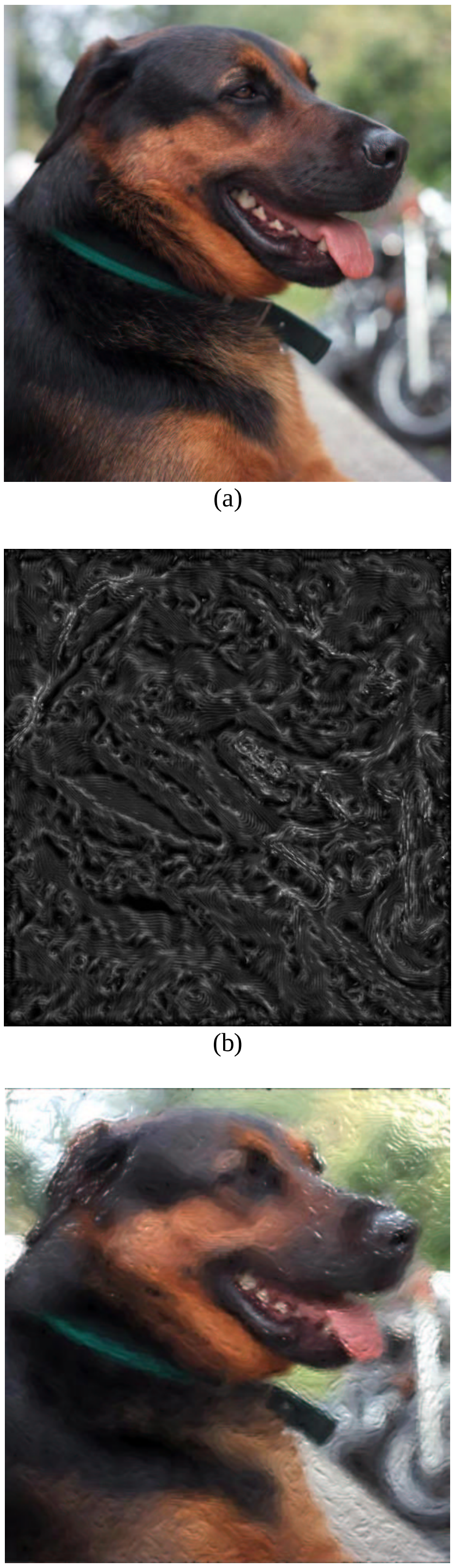

Figure 4 (a) The original dog picture. (b) The generated heigh map. (c) The final render of the painting.

with the same resolution, and apply the color plane as a texture map and the height plane as a displacement map. We can then place lights around the polygon to light our painting appropriately and then send it to a Reyes [6] based renderer.

After the displacement map and the texture map have been generated, what can be done with the renderer is limitless.

Reyes renderers work by converting all primitives to micropolygons and adaptively subdividing them as required, based on the footprint occupied by the micropolygon in the image. The rendering pipeline allows for these micropolygons to be displaced before they are rendered. We take advantage of this by letting the render do the hard work of figuring out how much our polygon needs to be subdivided and then we displace the micropolygons according to the generated displacement map. Finally the renderer takes the moved micropolygons and renderers them with accurate lighting and shadows.

7. Implementation

The techniques described in this paper were implemented as a custom Composite Operator (COP) in Side Effect Software's Houdini Animation Tools. We used COPS version 2.0, which was still in production during the development of this work. This presented its own challenges, including the necessity to implement image manipulation functions that would be readily available in other libraries designed for computer vision applications. However the power and flexibility of this architecture were well worth the extra effort spent in implementing some simple routines. The images were then rendered with Mantra, a renderer based on the Reyes architecture, similar to Pixar's RenderMan.

We used a few variations on Hertzmann's modified algorithm. Instead of using summed-area tables for blurring (and video coherence), we used the slower Gaussian blur. Summed-area tables are exceptional when performance is paramount, but are disadvantageous when memory footprints must be kept low. Also, instead of using cubic B-splines, we used cubic NRUBS curves with a uniform basis.

8. Results

In this section we discuss results obtained using our system. In general we find the results good, though we believe that it may be hard to convince someone that they are real paintings. The shortcomings of the system, and suggestions for ameliorating them, are discussed in the section following this one. More results and a more thorough discussion of the result can be found at http://www.dgp.toronto.edu/~gelkoura/proj/index.html.

8.1. The Butterfly

The butterfly results are shown on the color plates at the end of the paper. We first show what the image would be like rendered without any height information at all. This is similar to the effect that would be generated by other painterly rendering algorithms, and looks rather flat.

We then show what the image would be like rendered with only the height information obtained from the brush strokes (and not the canvas). The painting is rendered with two light sources at the top corners looking towards the painting. Without the canvas, the painting looks as though it has been painted on a perfect surface. This is because this particular picture has a constant colored background. The painting doesn't look quite right when only using the brush stroke heights.

If we don't use the brush height information at all and only use the canvas height, the painting looks more convincing than the previous case. This is because this loosely simulates thin paint on a thick canvas, which is believable.

Finally, when taking into account both the brush stroke and the canvas height, we achieve the best, and most convincing all the paintings. Note here that the specular highlights are exaggerated for illustrative purposes, and that an artist would not choose such settings.

8.2. The Dog

A picture of a dog, kindly donated, was used to illustrate the effect of lighting the painting with different light sources. An animation of lights moving across the painting can be downloaded at http://www.dgp.toronto.edu/~gelkoura/proj/index.html.

The dog picture is rich in detail and the stroke patterns produced (which are nicely visible in the height plane) are more complex than those produced by the butterfly picture. It is this complexity and fullness that made us realize quickly that capping the height after each was a bad idea.

9. Future Work

While we found the results to be exciting and rather promising, the paintings produced still have a certain plastic feeling that is so common to computer graphics. Our gaudy choice of specular highlight parameters certainly doesn't help. However what we seem to be missing mostly is the interaction of colors between each stroke. Using a constant color per stroke is unrealistic and produces a fake looking painting.

Easy additions to the algorithm may also improve the quality of the output. For example, allowing the user to specify the roughness parameter of the strokes, and not just the specular, may reduce that plastic feel.

As far as the generation of the actual paint strokes, we did all our calculations in RGB space and we expect to get better results if we worked in HSV. We rely heavily on luminance, where we really should rely on hue. This is visible in the dog painting, where the luminance difference between the dog's coat and the sidewalk are negligible, where the hue difference would have been substantial.

10. Acknowledgments

I would like to thank Professor Kiriakos Kutulakos for introducing me to the intermingling of computer graphics and computer vision. I would also like to thank the Research and Development staff at Side Effects Software for all their wonderful support and amazing tools.

11. References

[1] Aaron Hertzmann, "Painterly Rendering with Curved Brush Strokes of Multiple Sizes", *In ACM SIGGRAPH 98 Conference Proceedings*, July 1998, pp. 453-460.

[2] Aaron Hertzmann and Ken Perlin, "Painterly Rendering for Video and Interaction", *In First International Symposium on Non-Photorealistic Animation and Rendering*, June 2001.

[3] Side Effects Software, Inc. "Houdini 3D Animation Tools 5".

[4] L. Piegl and W. Tiller, *The NURBS Book*, 2nd Edition, Springer-Verlag, Berlin, 1997.

[5] Corel Corporation, Inc. "Corel PHOTO-PAINT 10".

[6] R. L. Cook, L. Carpenter and E. Catmull, "The Reyes Image Rendering Architecture", *In ACM SIGGRAPH 87 Conference Proceedings*, July 1987, pp. 95-102.

[7] John-Peter Lewis, "Texture Synthesis for Digital Painting", *In ACM SIGGRAPH 84 Conference Proceedings*, July 1984, pp. 245-252.

[8] Turner Whitted, "Anti-Aliased Line Drawing Using Brush Extrusion", *In ACM SIGGRAPH 83 Conference Proceedings*, July 1983, pp. 151-156.

[9] Peter Litwinowicz. "Processing Images and Video for an Impressionist Effect", *In Non-Photorealistic Rendering, SIGGRAPH Course Notes*, 1999.

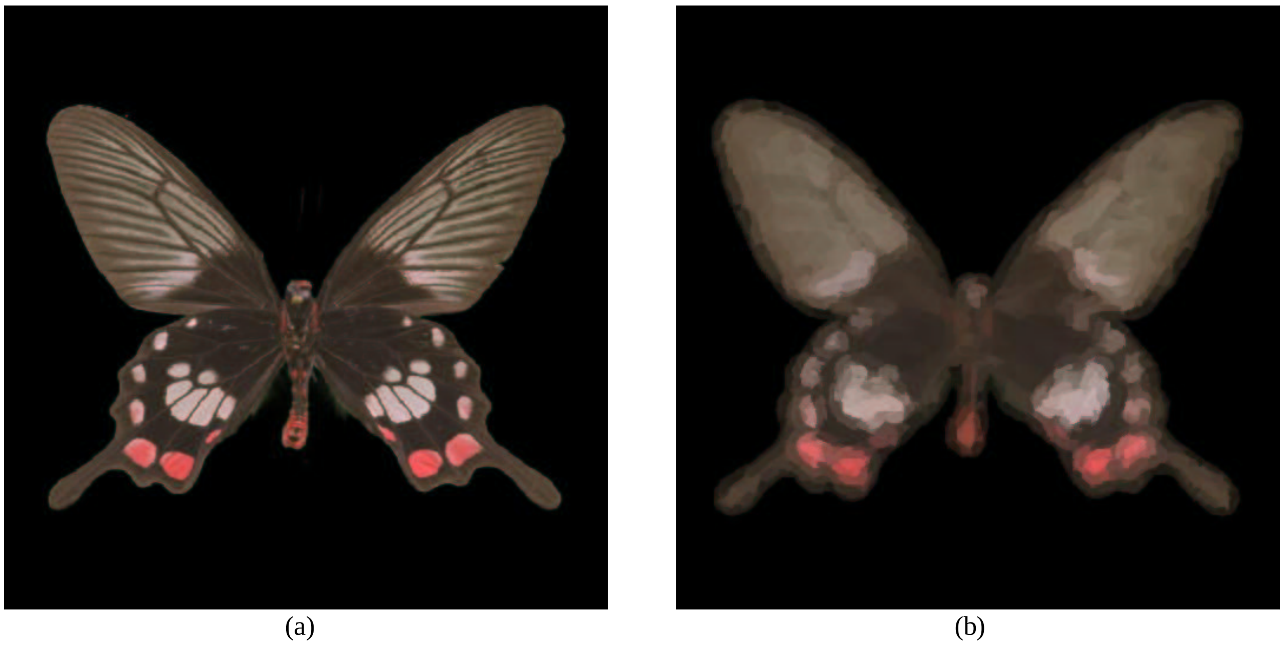

Figure 5 The original butterfly picture and the painting produced without using any height information.

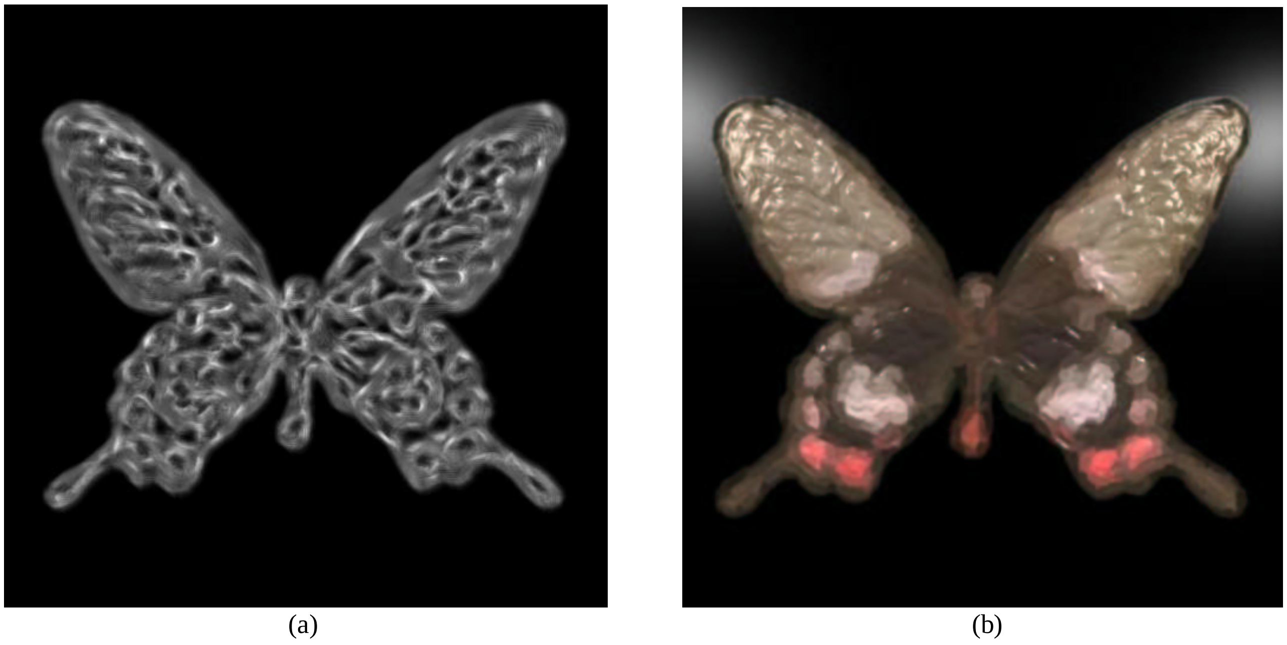

Figure 6 Displacement map generated using only the height information for the brush strokes and the resulting rendered painting.

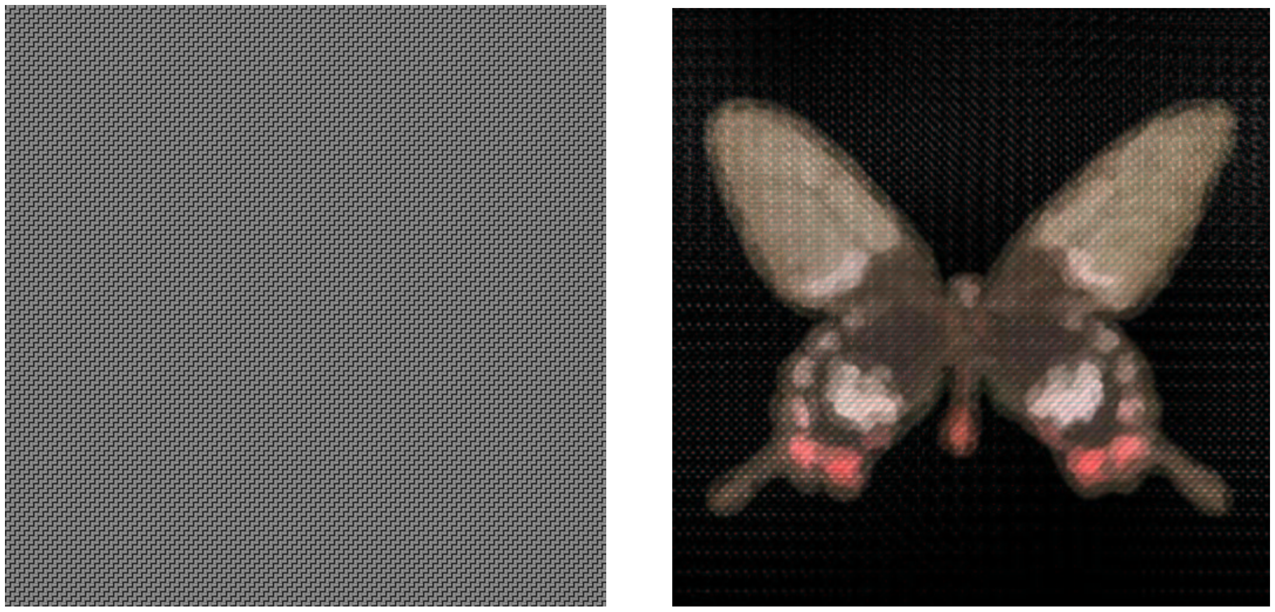

Figure 7 Displacement map and resulting image when only canvas height information is used.

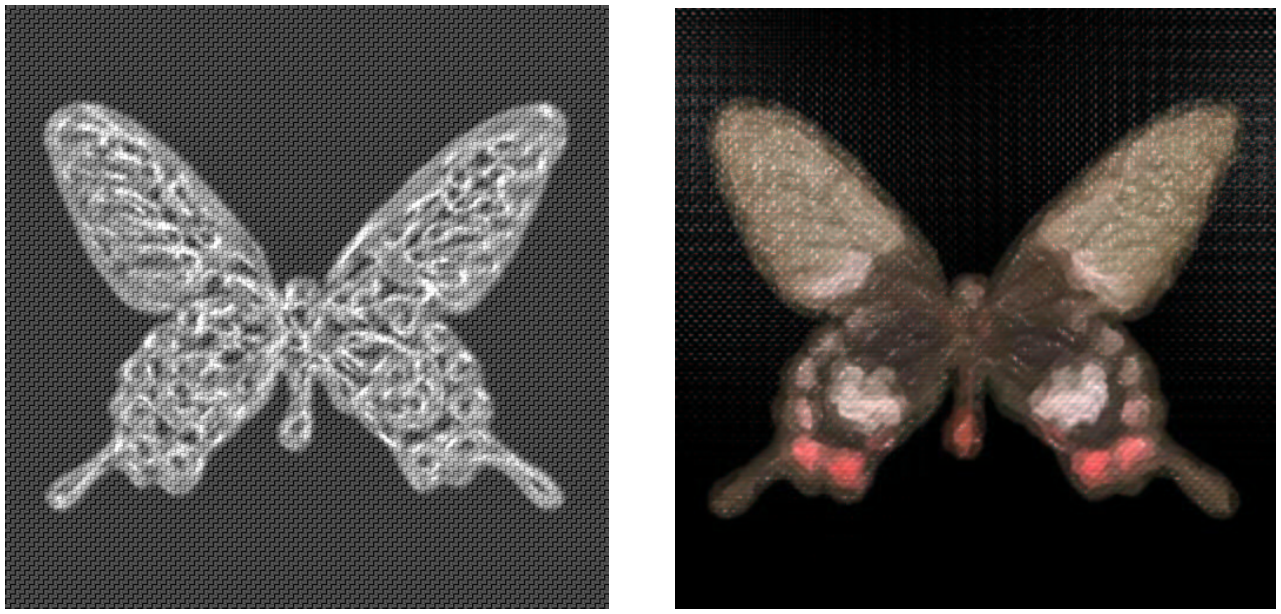

Figure 8 Displacement map and resulting image when both the canvas and the brush stroke height information is used.