A Proposed Glass-Painting Filter

Priti Sehgal
Keshav Mahavidyalaya,
University of Delhi,
Delhi – 35
psehgal25@rediffmail.com

P. S. Grover
Department of Computer Science,
University of Delhi,
Delhi – 7
groverps@hotmail.com

Abstract

In this paper we present an algorithm for a glass-painting filter. The filter inputs any arbitrary 2D image and converts it into its equivalent glass painted form. The initial image is processed through various non-photorealistic rendering stages. The major steps involved in the process include – edge detection and refinement, conversion of image into painted form, changing the appearance of the paints into glass paints and enhancing the image with black colored outlines and crushed silver foil background. The specular effect is also achieved by reflection mapping on per-pixel basis.

Keywords : non-photorealistic rendering, filter, glass painting, edge detection, Perlin noise, reflection mapping, OpenGL lighting and blending functions.

1. Introduction

Non-photorealism is a technique and abstraction of some visual outcome, content or style. Various non-photorealistic styles using pen and ink [1], graphite pencils [2], watercolors [3], pastel colors [4] etc have been studied and automated by many researchers. In recent years there has been considerable attention put on mosaics [5] and stained glass windows [6]. The "stained glass" is a colored and/or painted glass used in a decorative or architectural setting. The paint used is mixed with certain chemicals. These glasses come in variety of textures such as smooth, rough, wavy etc. Traditional artists, to create windows known as stained glass windows, used these glass pieces. The pieces are fused with the help of lead cames, or copper foils. The glass pieces are cut, by following a plan called a "cartoon", the lead cames are cut to size and the glass slotted in to create windows. The small glass pieces are known as *tiles* and the arrangement of tiles is referred to as *cartoon*. The art of glass painting is a unique and pleasing form of art. It is a method of decorating the surface of plain glass objects without cutting and piecing dines with stained glass. Decorative glass painting is a great way to create a beautiful look on stained glass windows and doors. Stained glass creations are difficult and fairly expensive to produce. One can imitate the same aesthetic quality as other stained products with considerable less cost. Nowadays Glass Paintings are widely used for decoration purposes in the form of wall hangings. In a glass painting, an image is painted with bright, shiny and transparent colors and is outlined with black color.

Mould [6] presented an automated method for transforming any arbitrary image into its stained glass version. The key issues used by him in designing were the tile boundaries and the tile colors. He gave an algorithm for constructing a cartoon from initial segmentation and suggested a palette close to the limited palette available in medieval times. The images produced by him gave very good results of stained glass windows. But these images could not be used for glass paintings due to several reasons. Firstly, the segmentation carried out does not produce natural looking boundaries required in glass paintings. Secondly, the glass paints are not that bright and shiny as desired in glass paintings. We had simulated a glass painting using a multi-pass non-photorealistic rendering technique [7]. The simulation involved the creation of wrinkled aluminium foil texture as background, simulation of glass surface, creation of procedural textures for generating glass colors and creation of raised, thick black outlines. The above-mentioned algorithm was an interactive method to generate a glass painting. The algorithm suffered from certain drawbacks. It was limited to geometrical images only. Also large amount of memory was required to store the glass color palette. We have proposed a new algorithm to transform any arbitrary image into its glass paint version in this paper.

2. Overview of the Algorithm

In constructing a glass painted image, we process the initial image through various stages. Firstly, we find out the outlines of the image using the most optimal algorithm. Then we convert the image into its painted form. This painted image is then rendered non-photorealistically so as to change the appearance of the paints into glass paints, which are thin, bright and shiny. The painted image is then mapped on the surface of glass; outlines are created in black color. The transparency in the image is achieved through the transparency in glass. To enhance the look of the painting, the painting is framed with a crushed silver foil background. Also, in the end the painting is specular

reflection mapped to get the desired effect through silver foil background. The steps followed in the algorithm are described in the following sections.

2.1. Remove Noise from the image

Noise is the data without meaning. This data does not transmit any signal and is produced as a by-product of some activities. For example, in images taken from digital cameras or film cameras may pick up noise from variety of sources. Since this as an unwanted noise, it may affect the image processing operations. Hence the noise must be removed fully or partially from the image for aesthetic purposes. We have implemented median filter to remove noise from the input image. This filter is very good at preserving image detail. It removes the salt and pepper noise from the image very nicely and causes very slight blurring of the edge hence it is often used for computer vision applications. The pseudocode for this filter is:

```
//pseudocode
begin
 for every pixel in the image
    sort the neighbouring pixels (8-connectivity) in
   increasing order of intensity
    Replace the value of the intensity of the pixel with
   the intensity of the median pixel from the list
End.
```

Figure 1 (a) shows the original image. Figure 1 (b) shows the noise-removed image. This noise filter reduces the image sharpness, which in turn helps to remove unwanted edges in edge detection process (Section 2.2).

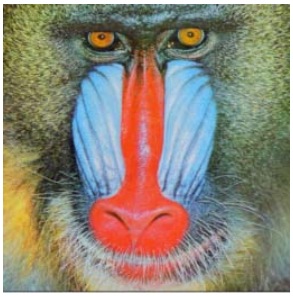

Figure 1(a):Original image

Figure 1(b): Filtered Image

Figure 1: Removing noise from image

2.2. Detect Edges

Edge detection is one of the most commonly used operations in image analysis. Although there are several edge detectors, we have tried to implement only those that would result in strong and connected edges in optimal time.

Marc et al. [8] had proposed an algorithm for enhancing of real-time non-photorealistic renderings. It was based on the edge map, a 2D texture that encoded visually important edges of 3D scene objects, and includes silhouette edges, border edges, and crease edges. The multi-pass rendering algorithm extracted geometrical properties of 3D scene objects to generate image-space data similar to G-buffers. If the input image to a glass-painting filter is a 3D image, then this edge detection technique proves to be quite efficient by using accelerated graphics hardware. But, if the input image is a 2D image as in our implementation, then we are able to get edges extracted from the buffer of normals but the edges due to varying depths are difficult to achieve. This is because of the non-availability of the depth of image. Although there are various algorithms, with the help of which we can get the depth of 2D image but all those require the sequence of images as an input, also it involves a lot of overhead time to calculate the depth image. Hence we have tried but not used this algorithm for edge detection in our implementation.

We had also tried to use the concept of watersheds to detect edges [9]. The advantage of using this concept for edge detection is that it yields continuous edges of single pixel width. In a watershed approach an image is divided into catchment basins separated by counters. Wright et al. [9] had adopted a watershed pyramid algorithm to create an edge detector. We had implemented the watershed algorithm on a Fendi image [Figure 2 (a)] and the result was an over segmented edge map [Figure 2(b)]. The image then needs to be post processed by several methods to merge the resulting segments. Although Wright's algorithm achieves good results but we have not adopted this approach since it is a very time consuming process and in a glass-painting filter, edge detection is an essential part of the algorithm. The speed of our algorithm very much depends on the edge detection part. Hence we have compensated for the speed at the cost of look to some extent.

In our implementation we have used Roberts filter on noise-removed image to detect edges since it is much faster than the other edge detection filters and it helps to produce significant results. If we apply Roberts filter on Figure 2 (a) after removing noise, we get the image as shown in Figure 2 (c). It is clearly visible that the intensity of the pixels is more at the edges of an image. We have converted this RGB image into binary image by comparing the intensity of each pixel with the threshold intensity of an image. The resultant binary image is shown in Figure 2 (d). Although this filter is designed for grayscale images but works fine for RGB images as well in our algorithm. The edge map resulting in Figure 2(d) shows some unwanted pixels in the image. These pixels are marked with red circles [Figure 2(d)]. To remove these unwanted pixels, we have used the operator similar to erosion and dilation morphological operators. While using an erosion operator, for every foreground pixel we check the every pixel corresponding to the structural element in

Figure 2(a):Original Fendi Image

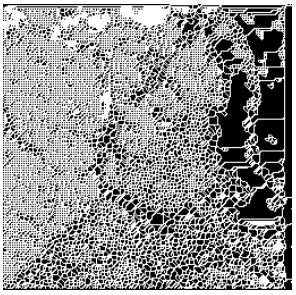

Figure 2(b):Over-segmented image obtained through watershed algorithm

Figure 2 (c) : Image obtained after applying Roberts Operator

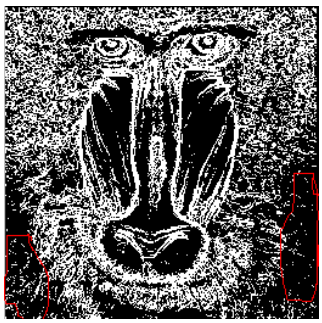

Figure 2(d) : Images obtained after thresholding

Figure 2(e) :Images obtained after eroding white pixels

Figure 2 : Edge Maps

the image, if *any* of these is black then we set the pixel to black.

In our implementation we have slightly modified the erosion operator; we check for every foreground pixel, if *every* pixel corresponding to the structural element in the image is black, then we set the pixel to black. We applied this operator on Figure 2 (d). The resulting image is shown in Figure 2 (e). The unwanted pixels encircled in Figure 2(d) are reduced in Figure 2(e). The most important aspect in this operator is the choice of structural element; a wrong choice may lead to blank image. In our implementation, we have considered the following structural element.

0	1	0
1	x	1
0	1	0

Structural Element

The pixel placed under x is checked with the pixels placed at locations marked 1. Basically this is a cross shaped structural element and gives sufficiently good results.

2.3. Convert the image into hand-painted image

The digitized RGB image can contain all color shades. There is a smooth distribution of colors in an image. While when we look at a hand painted image, the appearance of colors is solid rather than smooth. There has been a lot of research in painterly rendering techniques. Many algorithms have been devised by the researchers to create a hand-painted image or to convert any arbitrary image into the hand-painted one [10,11,12,13]. According to these researchers, an image can be painted by approximating the brush strokes or by applying textured paints. Another method to convert any arbitrary image into hand-painted image is passing it through a filter. Waltmann [14] has developed several image filters such as: point filters, area filters, geometric filters and artistic filters. Artistic filters are those, which filter an image to convert it in some artistic form. Some of the filters that come under this category are - Oil Paint, Frosted Glass, Random Blur, Raindrops. An oil paint filter converts an image into corresponding oil painted image. We have implemented a filter similar to oil paint filter and then post processed the image generated from it to make it resemble to a image painted with glass paints. In an oil paint filter, the intensity that appears most frequently around the pixel replaces the intensity of that pixel.

The proposed algorithm is summarized in the pseudocode below:

// pseudocode

```
Begin
  for every pixel (i, j) in the noise removed image
   { Let r be the radius that defines the number of pixels to
     be checked in the neighborhood.
      Let I be the number of intensities to be considered.
       Let avgr be an array with I number of elements to
     hold R components, avgg be an array with I number
     of elements to hold G components and avgb be an
     array to with I number of elements to hold B
     components.
    for every pixel (x,y) in the radius r of pixel( i, j)
     {
       Also keep a count of distinct intensity of every
       RGB pixel.
       Calculate the intensity (p) of a pixel (x,y).
       Increment the particular intensity count (p^th intensity
       count) .
       Add the value of R,G,and B components of pixel
       (x,y) to p^th element of respective arrays (avgr, avgg,
       avgb).
     }
    Find the maximum intensity count (maxi) and the
     value of intensity appearing most (mi).
   Replace the value of RGB components of pixel (i, j)
     with the RGB components of the pixel appearing most
     (mi^th element) of avgr, avgg and avgb arrays
     respectively divided by maxi.
   }
End;
```

We have implemented the above algorithm with various parameters using Figure 1 (a) as the input. The resulting images are shown in Figure 3 (a-c). The radius is 2 and number of intensities is 10 in Figure 3 (a). The radius is 4 and number of intensities is 10 in Figure 3 (b) and the radius is 2 and number of intensities is 100 in Figure 3(c). It can be seen that Figure 3 (a) is most appropriate as increasing the radius gives the image a more distorted look [Figure 3 (b)] and increasing the number of intensities smoothens the image hence loosing the hand painted appearance [Figure 3 (c)]. Thus the choice of parameters (radius and intensities) is very crucial in getting an appropriate hand-painted image.

2.4. Converting the hand-painted image into glass-paint colored image

The colors used for glass paintings are bright, shiny, watery and emissive. The image produced in Section 2.3 [Figure 3] gives a hand-painted appearance but the medium of colors used does not reflect glass colors. The paint seems to be thick in Figure 3. To convert the image into an image painted with thin watery colors, the image needs to be post processed. We have tried to achieve the effect of required image by applying a coat of water on the hand painted image in such a manner that water mixes with the paint. We had created a color palette of glass colors using Perlin Noise function [7]. There we had created textures of colors using Perlin function by considering the light and dark shade of the color to be produced as background and noise colors. We have used the same concept in this algorithm with the difference that the background color is the same as that of the hand-painted image and the noise color is considered to be that of water approximately light blue color. The result of applying Perlin function with the above-mentioned colors on Figure 3 (a) is shown in Figure 4 (a).

The result of Figure 4 (a) satisfies the requirement of thin paints but as we have already said, glass paints are shiny, emissive and bright. These properties are missing in Figure 4 (a). To make the image in Figure 4 (a) bright and shiny, we have used OpenGL lighting and blending functions. The image is captured in a texture map and the texture map is blended with the specular highlight map [15]. Using OpenGL, proves to be the most convenient way to convert an image into its shiny counterpart without any need to physically change the pixel values. The result is shown in Figure 4 (b). The outlined Fendi image using the edge map created in Section 2.2 is shown in Figure 5.

2.5. Map the resulting image on glass

The images created in the last section exhibit the properties of bright shiny colors and the outlines, but fail to show the transparency effect. The transparency effect is achieved by mapping the image on the glass surface. Drawing a surface of glass color with an alpha value (A in RGBA) less than 1 simulates the glass being transparent. The value of alpha affects the transparency of glass surface. As the alpha value increases, the surface becomes less transparent. The glass surface is also mapped with the specular highlight map to achieve the desired appearance of glass.

2.6. Enhancing the image with silver foil background

Framing the painting with a shiny background enhances the glass paintings and they can be used as wall hangings. Generally this shiny background is either crushed aluminium or gold colored foil. We have enhanced our paintings by framing with crushed aluminium foil background. The wrinkled aluminium foil is simulated using bump-mapping technique [7,16]. Again the foil generated is not bright and shiny, hence we have converted it into a shiny background using OpenGL lighting and blending functions [15].The similar concept of mapping the foil texture with specular highlight map is used, as used while simulating glass [7]. The effect of the glass painting with wrinkled aluminium foil is shown in

Figure 3(a):Painted image with radius=2 and I = 10

Figure 3(b) : Painted image with radius=4 and I = 10

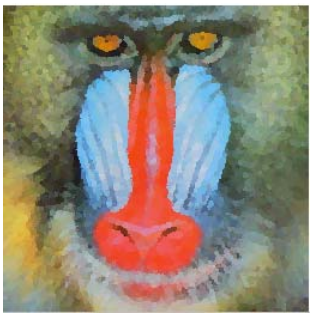

Figure 3 (c) : Painted image with radius=2 and I = 100

Figure 3: Images passed through Paint-Filter (Section 2.3)

Figure 4 (a):Hand Painted image passed through Perlin Function

Figure 4(b): Image painted with bright shiny colors

Figure 4 : Hand-Painted Image converted into painting with glass colors

Figure 5 : Outlined Fendi Image

Figure 6 (a).

The background is visible through the glass [Figure 6(a)], but the shine of the aluminium foil that is often reflected through the glass is missing. To incorporate the same, we have performed reflection mapping [7]. For every point on the glass surface (where the image is mapped), reflected vector is calculated by using the Eqn 1.

$$r = a - 2*m \qquad (1)$$

Here, r is the reflected vector,
a is the incident vector and
m is in the direction of the normal

For this purpose, the normal at any point on the surface is obtained by parameterizing the surface using the function O(u, v) [Eqn 2]. Then the normal is given by:

$$n = O_u \times O_v \qquad (2)$$

Here O_u and O_v are partial derivatives of the surface at point p and are obtained by central difference method. The dot product of the reflected vector is then calculated with the vector between the surface point and the light source. The value thus obtained is added to the RGBA values of the image on the glass surface. The result is shown in Figure 6(b).

3. Results and Conclusions

This paper proposes a filtering technique that gives the freedom to the user to convert any arbitrary image into a glass painted version. In our implementation, we have used SGI's .RGB image format but we can easily use the images in any other format provided we know how to load it and read it. The resulting images are shown in Figure 6 (b-d). Figure 6(c) shows a Fendi image with reflection mapping and Figure 6 (d) shows a glass painted version of a leaf. The image generated by our algorithm is lacking bumped outlines. This feature can be easily incorporated to further enhance the image. The algorithm to bump the outlines is provided in our previous work [7]. It takes approximately 2800 millisecs to generate the glass painted images shown in Figure 6. This algorithm proves to be optimal since the time taken to generate only the outlines using watershed algorithm as shown in Figure is approximately 2500 millisecs. The work presented in this paper provides a fast, efficient and simple technique to produce beautiful art work.

4. Bibliography

[1] G. Winkenbach and D. H. Salesin, "Computer-Generated Pen and Ink Illustration", *In Proceedings of ACM SIGGRAPH '94*, pp-91-100, 1994.

[2] C. M. Sousa And J. W. Buchanan, "Observational models of graphite Pencil materials*", proceedings of Eurographics 1999*.

Figure 6 (a) : Glass painting with crushed aluminium foil background

Figure 6 (b) : Result of glass painting after reflection mapping

Figure 6(c): Glass Painted Fendi image with reflection mapping

Figure 6(d): Glass Painted Leaf image with reflection mapping

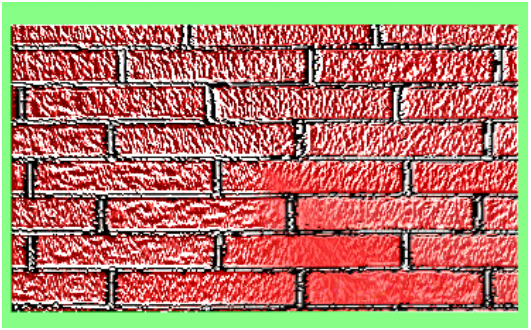

Figure 6(e): Glass Painted brick pattern with reflection mapping

Figure 6: Examples of Glass Paintings

[3] C. J. Curtis, S. E. Anderson, J. E. Seims, K. W. Fleischer and D. Salesin, "Computer-generated Watercolor", *In SIGGRAPH '97 Conference Proceedings*, pages 421-430, August 1997.

[4] K. Murakami, and R. Tsuruno, 2002 "Pastel-Like Rendering Considering the Properties of Pigments and the Support Medium" In *SIGGRAPH 02* pages.cpsc.ucalgary.ca/~mario/npr/bib/refs/00s/2002/murakami-02.pdf

[5] G. Elber, and G. Wolberg, "Rendering traditional mosaics", *The Visual Computer 19*, 1 (2003), 67ñ78.

[6] D. Mould, " A stained glass image filter". *Proceedings of the Eurographics Symposium on Rendering 2003*, 16(1):20–25,June 2003.

[7] P. Sehgal and P.S. Grover "Simulation of 2D glass Painting using multi-pass non-photorealistic rendering technique", In the *proceedings of The International Conference on imaging science, systems and technology'04(CISST'04),* pages 279-285.

[8] Marc Nienhaus and Jürgen Döllner. Edge-Enhancement - An Algorithm or Real-Time Non-Photorealistic Rendering. *Journal of WSCG*, 11(2):346-353, 2003.

[9] A. Wright and S. Acton, Watershed Pyramids for Edge Detection, Proceedings of IEEE ICIP'97, 2(1997) 578-581.

[10] P. Haeberli. "Paint by numbers: abstract image representations*", In ACM Siggraph*, pages 207--214, 1990.

[11] A. Hertzmann, "Painterly Rendering with Curved Brush Strokes of Multiple Sizes", *SIGGRAPH '98 Conference Proceedings*, pages 453-460, July 1998.

[12] Michio Shiraishi and Yasushi Yamaguchi. "An algorithm for automatic painterly rendering based on local source image approximation." *Non-Photorealistic Animation and Rendering*, 53-58, June 2000.

[13] D. Robert, Kalnins, Lee Markosian, J. Barbara Meier, Michael A. Kowalski, Joseph C. Lee, Philip L. Davidson, Matthew Webb, John F. Hughes and Adam Finkelstein. "WYSIWYG NPR: Drawing strokes directly on 3d

models.” In *Computer Graphics (SIGGRAPH 2002*. pp. 755-762.
[14] J. Waltman, “Blurring the Line : algorithms and aesthetics of image filtering”, April 2001. www.jasonwaltman.com
[15] D Blythe, B Grantham, S Nelson, and T McReynolds. “Advanced Graphics Programming Techniques using OpenGL.” http://www.sgi.com/Technology/OpenGL/advanced_sig98.html
[16] J Blinn, "Simulation of Wrinkled Surfaces", *Computer Graphics*, *(Proc. Siggraph),* Vol. 12, No. 3, August 1978, pp. 286-292.