Virtual Reality Term Project, Fall 2011

Painterly Rendering with Curved Brush Stroke of Multiple Sizes

Advisor: Prof. Ming Ouhyoung

Team: B96902004 Hsin-Cheng Chao
      B96902033 Hsuan-Yueh Peng

Computer Science
National Taiwan University
In this term project of virtual reality course, we chose "Painterly Rendering with Curved Brush Strokes of Multiple Sizes" by Aaron Hertzmann as our referred implementation paper which is to switch an input image into an output image with oil-painted property.

The method of hand-painting is used in the paper in order to increase the real nature of curves by drawing with spline brush stroke. Besides, a canvas consists of multiple layers in the implementation. Therefore, we can make sure the drawing goes from large scale to small one, enhancing the expression of details.

**Approaches:**

1. **Multiple-layered changeable brush size**

   We provide adjustable brush size, namely that users are using multiple brush in drawing. In this way, the drawing process goes from rough sketch scale to subtle details. The algorithm is as the following figure:

   ```plaintext
   function paint(sourceImage, R1, ... Rn) {
     canvas := a new constant color image
     // paint the canvas
     for each brush radius Rj, from largest to smallest do
     {
       // apply Gaussian blur
       referenceImage = sourceImage * G(to Rj)
       // paint a layer
       paintLayer(canvas, referenceImage, Rj)
     }
     return canvas
   }
   ```

   paint(); is the main function doing drawing, and R1...Rn are the brushes with radius R1, R2,... Rn.

   We are given a plain canvas at the beginning, and take each brush according to its radius from large to small to draw on the canvas.

   In each drawing content with respect to every brush, we apply Gaussian blur to the input(source) image and draw the content onto the canvas(paintLayer());.

   Each time after a brush finishes its drawing, the canvas will be used at next time (cumulative).

   Next, we discuss the function paintLayer();
In `paintLayer();`, the parameters are: 1. the canvas, 2. the Gaussian blurred image from the previous step, and 3. the brush with radius $R$.

We want to create the difference image from the canvas and the blurred image for future uses. The Difference distance is the Euler distance of colors. The variable `grid` is our reference distance for how much we draw.

The following two for loops use the grid distance to draw/run the whole canvas. Supposed we are now at the point $(x, y)$, we need to calculate the area color difference of the pixel according to the information on the difference image.

With the `areaError`, if this error is greater than the $T$ (threshold) we defined, we must then have the maximum error point in this error area so as to draw at this point as the center. However, in this step(`makeStroke();`), we do not really draw onto the canvas right away, but store what we need to draw first and draw it on randomly after considering all pixels. Further, the `makeStroke();` function is a simple line-drawing concept. There is another function called `makeSplineStroke();` for us to perform the better curve brush stroke.
The `makeSplineStroke()` is mainly used to describe how and what we should do to create a proper curved stroke.

At first, we need to store the color we want into `strokeColor`. Then, make the decision of the curve after we store the very beginning position. (Here is a `maxStrokeLength`, the maximum length of the curve. We will explain it in the style section later.)

There are two considered facts in making the curve decision:
1. If the length of the curve is greater than `maxStrokeLength` and the color difference of at `(x, y)` on the canvas and the reference image is less than that of the stroke color and the reference image at `(x, y)`, then we return the stored curve immediately.
2. If the gradients in both x and y directions are equal to zero, we return the curve as well.

If the considered facts are all passed, we use the following figure and explanation to express the procedure of spline curve:
Finally, we now have a complete curve to return.

2. Rendering Styles

Before we head into rendering styles, let us explain the meaning of each parameter:

a. T—decides the roughness of painting
b. Ri—brush sizes
c. fc—used to limit or enhance stroke curve
d. fs—the parameter of Gaussian function
e. maxStrokeLength, minStrokeLength are the maximum and the minimum length of a stroke
f. a(opacity) is the transparency or the the opacity of the image.
g. fg—the size of the grid,

Here we provided with three styles of oil-paintings:

1. Impressionist:
   \[ T = 100, \ R = (8, 4, 2), \ fc = 1, \ fs = 0.5, \ fg = 1, \ a = 1, \]
   \[ \text{minStrokeLength} = 4, \text{maxStrokeLength} = 16 \]

2. Expressionist:
   \[ T = 50, \ R = (8, 4, 2), \ fc = 0.25, \ fs = 0.5, \ fg = 1, \ a = 0.7, \]
   \[ \text{minStrokeLength} = 10, \text{maxStrokeLength} = 16 \]

3. water-wash:
   \[ T = 200, \ R = (8, 4, 2), \ fc = 1, \ fs = 0.5, \ fg = 1, \ a = 1, \]

\textbf{Painting a brush stroke.} (a) A brush stroke begins at a control point \((x_0, y_0)\) and continues in direction \(D_p\) normal to the gradient \(G_p\) (b) From the second point \((x_1, y_1)\), there are two normal directions to choose from: \(\theta_1 + \pi/2\) and \(\theta_1 - \pi/2\). We choose \(D_p\) in order to reduce the stroke curvature. (c) This procedure is repeated to draw the rest of the stroke. The stroke will be rendered as a cubic B-spline, with the \((x_i, y_i)\) as control points. The distance between control points is equal to the brush radius.
minStrokeLength = 4, maxStrokeLength = 16

3. **Results:**
The followings are the results by our program

![input image](image1)

![after 1st layered](image2)
after 2\textsuperscript{nd} layered

after 3\textsuperscript{rd} layered
input image (left: Hsin-Cheng Chao, right: Hsuan-Yueh Peng)

after 1st layered
after 2\textsuperscript{nd} layered

after 3\textsuperscript{rd} layered
The followings are different styles of drawing:

input image

impressionist
expressionist

water-wash
As well as still images, we also applied this effect onto video. However, as video requires greater frame-to-frame consistency, we particularly chose another type of video, namely Stop Motion, to help lower the importance and the dependency of consistencies across frames.

For proper consistency of oil painting across videos, please refer "Painterly Rendering for Video and Interaction" by Aaron Hertzmann & Ken Perlin(2000).

The disk we handed in also contains the C++ source code, mainly implemented with OpenCV 2.0. There are two version of the program: One is for still image, users can type in image file as argv[1] and follow the instruction on screen to select different painting styles. Users are also allowed to have customized, special effects by configuring their own parameter in rendering. The other version is for Stop Motion video. Same as above, users type in video name as argv[1], video width as argv[2], and height as argv[3]. It takes about five to ten seconds for each data set(five frames) on average PC.

4. Discussion and Q&A:
In this paper implementation, we spent lots of time on paper surveying. Actually, we wanted to make consistent video oil painting at the very beginning. But due to the limit of time and the depth of paper, we turned to focus more on still images. The programming part of the project is pretty straight forward. The whole program can be completed smoothly as long as each step and details is fathomed well. As to the final stop motion video, it took five hours to render, and we were surprised and satisfied with the end result, which has a character of its own.

Thank you Prof. Ming Ouhyoung for advising us in this semester. :D