REPORT

제목 : Painterly Rendering 기술의 이해

제출일	2022.05.14	전 공	소프트웨어
과 목	멀티미디어프로그래밍	학 번	18011688
담당교수	박상일	이 름	차태관

목 차

서론

1. 기술의 개발 동기

'Painterly Rendering with Curved Brush Strokes of Multiple Sizes' by Aaron Hertzmann 이 논문은 1998년 뉴욕 대학교의 컴퓨터 과학 분야의 교수님이 발표한 논문이다. Painterly Rendering 기술의 개발 이전에는 전통적인 방식으로의 그림 및 일러스트레이션은 상당한 시간과 자원을 투자해야 하는 숙련자를 위한 영역이었다. 하지만 Painterly Rendering 기술이 그림을 그리는데 드는 시간과 자원을 줄이고 알고리즘을 통한 다양한 방식으로 이미지를 표현하게 만들기 위해 개발되었다.

2. 기술의 필요성 및 아이디어

1998년까지 개발되었던 컴퓨터 회화적 알고리즘은 크기와 모양이 모두 동일한 매우 간단한 브러시를 사용을 하여 구현되었다. 따라서 이미지가 손으로 그린 이미지에 비해서 기계적으로 보이는 경향이 있었는데, 이 논문에서는 간단한 브러시만을 사용하는 것이 아닌 여러 브러시 크기와 다양한 브러시 형태(구부러지거나, 길거나)를 이용해 이미지를 회화화 하는 기술을 제시한다. 그러므로 이전의 알고리즘으로 생성된 그림보다 더 자연스럽다.

따라서 기술을 사용할 때 개인의 창의성을 배제하지 않고 여러 방식을 조합해 여러 회화 그림을 표현할 수 있다.

Painterly Rendering 알고리즘의 아이디어는 보통의 이미지 혹은 사진을 사람이 그린 회화처럼 보이도록 바꾸어주는 것이다. 간단히 소개하자면, 여러 가지의 브러시 크기를 이용해 그림을 뭉개고, 뭉갠 그림에서 디테일을 점점 살린 후, 마지막으로 세부 정보들을 표현해주어 회화를 만드는 것이라고 할 수 있다.

본론

1. 기술의 구체적 방법론

본래 예술가는 본인이 강조하고 싶은 요소에는 다른 요소들과 구별되는 스트로크 혹은 붓 크기를 사용했다. 예시로 아래 그림을 보자.

<해변의 풍경 - Edgar Degar>

이것은 Edgar Degar 의 그림 '해변의 풍경' 의 일부분이다. 그림을 자세하게 살펴보면 화가가 강조하고 싶었던 요소를 알 수 있다. 바로 그림에 나타나는 여인인데, 여인은 뒷 배경을 차지하고 있는 배, 어린 아이 혹은 다른 가족들, 바다, 등 다른 요소들과는 확실히 차이나는 질감으로 그려져 있다. 얼굴의 홍조, 입술, 머리색 등이 특히 그렇다. 따라서 자연스레 우리는 여인에게 집중하게 되는 것이다.

논문에서는 이처럼 화가의 의도를 본따기 위해, 원본 이미지에 대한 대략적인 묘사부터 시작하여 브러시의 크기를 줄여나가 디테일을 추가하는 알고리즘을 사용했다. 즉 실제 화가가 그린 것처럼 회화를 만들기 위함이다. 이것을 방법에 대한 자세한 사항은 후술한다.

제시된 의사 코드를 살펴보자.

```
function paint(sourceImage,R1 ... Rn)
{
   canvas := a new constant color image

   // paint the canvas
   for each brush radius Ri,
         from largest to smallest do
   {
      // apply Gaussian blur
      referenceImage = sourceImage * G(fσ Ri)
      // paint a layer
      paintLayer(canvas, referenceImage, Ri)
   }

   return canvas
}
```

paint 알고리즘에서는 인자로 원본 이미지, 그리고 브러시의 크기 목록을 받는다. 브러시의 크기를 R1부터 Rn까지, 다시 말해 가장 큰 브러시부터 가장 작은 브러시까지 사용하여 이미지를 다시 그리는데, 이때 가우시안 블러링 후, 레이어를 그림에 칠한다. 가우시안 블러링을 사용하는 자명한 이유는 원래 이미지는 사진 혹은 그림이므로, 회화만큼의 질감 표현이 되어있지 않기 때문에, 블러링을 통해 그림의 디테일을 줄여주는 것이라고 생각할 수 있다.

알고리즘 진행 과정을 다시 보면, 각 레이어에 대해 원본 이미지를 먼저 블러링을 하여 참조할 이미지를 만든다. 큰 이미지로부터 참조 이미지를 만들고, 점점 더 작은 이미지로의 참조 이미지를 만들어 나간다. 그러면 각각의 브러시 크기만큼의 세부 정보를 담은 이미지 레이어들이 만들어 지는데, 이것들을 기반으로 paintLayer 함수를 통해 원본 이미지와 다른 영역을 찾아서 새 브러시 스트로크(색상 차이를 확인하고 효과를 낼 위치)로 덮는다. 각 레이어는 간단한 루프를 이용하여 색상이 채워지는데, 그림을 칠할 점의 주변을 탐색하여 원본 이미지와 가장 차이가 큰 점을 찾아 그림을 칠해준다. 이때 원본 이미지 색상이 임계값과 일치하는 영역은 변경되지 않은 채 남는다.임계값 매개변수를 증가시키면 대략적인 이미지가 만들어지고, 임계값을 줄이면 줄일수록 원본과 유사한 이미지가 만들어지게 된다.

다시 말해 큰 브러시로 원본 이미지를 뭉개 대략적인 평평한(혹은 배경이 될 수 있는) 이미지를 만든 후, 작은 브러시로 뭉개 디테일이 잘 드러나 있는 이미지를 참조해 디테일을 살리는 것이다.
PaintLayer 의 의사코드는 아래와 같다.

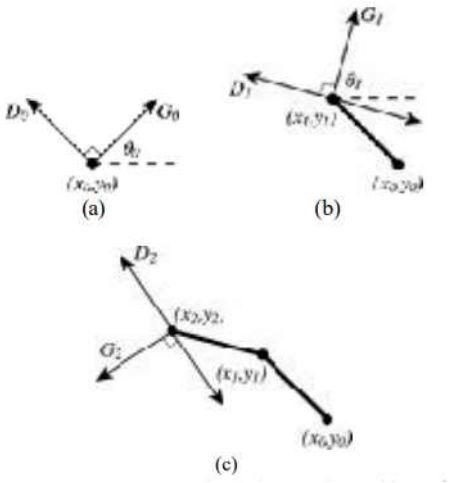
```
procedure paintLayer(canvas,referenceImage, R)
{
   S := a new set of strokes, initially empty

   // create a pointwise difference image
   D := difference(canvas,referenceImage)

   grid := f_g R

   for x=0 to imageWidth stepsize grid do
      for y=0 to imageHeight stepsize grid do
      {
        // sum the error near (x,y)
        M := the region (x-grid/2..x+grid/2,
                         y-grid/2..y+grid/2)
        areaError := Σ_{i,j∈M} D_{i,j} / grid^2
        if (areaError > T) then
        {
           // find the largest error point
           (x_1, y_1) := arg max_{i,j∈M} D_{i,j}
           s :=makeStroke(R, x_1, y_1, referenceImage)
           add s to S
        }
      }

   paint all strokes in S on the canvas,
      in random order
}
```

색상 차이 공식은 4|(r1,g1,b1)-(r2,g2,b2)| = ((r1-r2)2+ (g1-g2)2+ (b1-b2)1/2 이다. make Stroke() 함수는 참조 이미지와 브러시 반경이 주어졌을 때 (x1,y1) 에서 시작하는 이미지에 스트로크를 배치하는 절차이다. fg는 그리드 크기 변수이다.

다음으로 살펴볼 것은 가까운 길고 구부러진 붓글씨를 배치하기 위한 알고리즘이다. 방금까지 살펴본 알고리즘은 이미지를 회화 그림처럼 바꾸어주는 역할만 했다. 따라서 예시를 들어 설명한 것처럼 세부 디테일을 표현해줄 알고리즘이 따로 필요한데, 그것이 지금 설명할 알고리즘이다.

논문에서는 본인의 알고리즘을 이전의 동일한 작은 스트로크들을 사용한 자동 색칠 알고리즘과는 다르게, 길고 연속적인 곡선을 그리는 방식으로 행한다고 말한다. 이 알고리즘에서는 기울기의 법선을 따라 곡선에 대한 제어점을 배치하는 것으로 스트로크를 그린다. 또한 스트로크의 색상 값이 표기된(혹은 사용자에 의해 설정된) 임계점 이상을 벗어나면 해당 제어점에서 스트로크가 종료된다.

```
function makeSplineStroke(x0,y0,R,refImage)
{
   strokeColor = refImage.color(x0,y0)
   K = a new stroke with radius R
         and color strokeColor
   add point (x0,y0) to K
   (x,y) := (x0,y0)
   (lastDx,lastDy) := (0,0)

   for i=1 to maxStrokeLength do
   {
      if (i > minStrokeLength and
        |refImage.color(x,y)-canvas.color(x,y)|<
        |refImage.color(x,y)-strokeColor|) then
          return K

      // detect vanishing gradient
      if (refImage.gradientMag(x,y) == 0) then
        return K

      // get unit vector of gradient
      (gx,gy) := refImage.gradientDirection(x,y)
      // compute a normal direction
      (dx,dy) := (-gy, gx)

      // if necessary, reverse direction
      if (lastDx * dx + lastDy * dy < 0) then
        (dx,dy) := (-dx, -dy)

      // filter the stroke direction
      (dx,dy) :=fc*(dx,dy)+(1-fc)*(lastDx,lastDy)
      (dx,dy) := (dx,dy)/(dx2 + dy2)1/2
      (x,y) := (x+R*dx, y+R*dy)
      (lastDx,lastDy) := (dx,dy)

      add the point (x,y) to K
   }
   return K
}
```



makeSplineStroke 알고리즘은 이렇게 작동한다.

참조 이미지의 지정된 지점(x0,y0)에서 반지름 R을 가진 브러시로 시작한다. 제어점(x0,y0)이 스플라인(점을 이어 만든 곡선)에 추가되고 (x0,y0)에서 원본 이미지의 색상이 스플라인 색상으로 사용된다. 그런 다음 곡선을 따라 다음 점을 계산한다. 다음 점에 대한 기울기(θ)는 원본 이미지의 Sobel Filtering(윤곽선 검출 필터) 의 휘도를 통해 구할 수 있다. 다음 점(x1,y1)은 (x0,y0)에서 거리 R로 (θi+pi/2) 방향으로 배치된다. R은 이 브러시 크기로 캡쳐할 세부 정보 수준을 나타내기 때문에 브러시 반지름 R을 제어점 사이의 거리로 사용한다. 즉, 매우 큰 브러시는 이미지의 넓은 스케치를 만들고 나중에 더 작은 브러시로 다듬어진다.

나머지 제어점은 이미지 그라데이션(변화하는 정도)에 수직인 영상을 따라 이동하고 제어 지점을 배치하는 이 과정을 반복함으로써 계산된다. 스트로크는 미리 결정된 최대 스트로크 길이에 도달하거나 스트로크의 색상이 해당 지점의 현재 그림과 다른 것보다 마지막 제어 지점 아래의 색과 더 많이 다를 때 종료된다. 이때 최대 스트로크 길이는 무한 루프 발생을 방지하는 역할을 해준

다. 점(xi, yi)에 대해, 우리는 그 점의 기울기 방향 q를 계산하는데, 실제로는 두 개의 본래 방향이 있으므로 다음 방향에 대한 두 가지 후보인 θi+pi/2와 θi-pi/2가 있다. 이때 스트로크 곡률을 최소화하기 위해 Di와 Di-1 사이의 각도가 p/2 이하가 되도록 방향 Di를 선택한다. 또한 스트로크 방향에 IIR 필터(입력신호와 출력신호의 값이 재귀적으로 적용되는 필터) 를 적용하여 브러시 스트로크 곡률을 과장하거나 줄일 수 있다. 이 필터는 미리 결정된 단일 필터 상수 fc에 의해 제어된다.

(a) 브러시 스트로크는 제어점(x0,y0)에서 시작하여 그레이디언트 G0에 수직인 방향 D0으로 이어진다. (b) 두 번째 점(x1,y1)부터는 θ1+p/2, θ1-p/2에서 선택할 수 있는 두 가지 방향이 있다. 스트로크 곡률을 줄이기 위해 D1을 선택합니다. (c) 이 절차를 반복하여 나머지 스트로크를 그린다. 스트로크는 (xi,yi)를 제어점으로 하는 입방체인 B 곡선으로 렌더링된다. 이러한 곡선들을 따라서 스트로크를 하면 원하는 화풍으로 이미지를 필터링 할 수 있다.

2. 실험결과에 대한 평가

그림 2의 (a), (b), (c), (d) 는 각각 원본, 브러시 크기 8로 만든 참조 이미지. (b)를 브러시 크기 4 로 다시 만든 참조 이미지. (c)는 (b)를 브러시 크기 2로 만든 참조 이미지이다. 모두 곡선 순차적으로 브러시 크기를 줄여가며 원본 이미의 디테일을 살려간다. 브러시의 크기가 작을수록 이미지의 디테일이 살아나는 것이 느껴지는데, 최종 이미지인 (d)를 보면 인상주의파 화가가 그린 그린처럼 사진이 회화화 된 걸 볼 수 있다. 또한 이전 브러시의 스트로크가 그대로 남아 있는 것도 볼 수 있다. 사람이 직접 터치한 것 과 비슷하게 그려진 것을 볼 수 있다.

그림 3의 (a), (b) 는 각각 8, 4, 2 브러시 크기 스트로크로 만들어졌는데, 차이점은 (a) 는 원 모양의 터치, (b) 는 짧은 터치와 더불어 테두리를 다듬은 곡선으로 만들었다. 그림3-(a) 는 원 모양으로 스트로크를 하자 이미지가 매우 기계적으로 필터링 된 것을 알 수 있다. 사과 및 손 등을 보면 원의 브러시 모양이 그대로 드러나서 부자연스럽다. 반면에 그림3-(b) 는 성공적으로 필터링 된 것을 볼 수 있다.

그림6 에서의 6가지 그림들은 각각 인상파, 표현파, 컬러리스트 워싱을 적용한 결과이다. 화풍에 맞게 필터링 되었으며, 모자이크화, 윤곽선 강조, 색 왜곡 등이 되는 것을 확인할 수 있다.

3. 단점 및 한계점

사람이 직접 그린 것과 비슷하게 그려졌지만, 그림2-(d)의 사과를 들고 있는 손을 자세히 보면, 알고리즘 실행 후 손가락에 사과의 색인 빨간색이 약간 석여있고, 오른쪽 맨 끝의 사과가 번진 것처럼 표현된 부분을 볼 수 있다. 이는 그림의 어두운 부분에서 도드라진다. 그리고 오른쪽 아래의 어두운 부분들은 디테일이 사라져 브러시의 모양이 그대로 드러난다.

추가로 그림6의 여섯 가지 그림들을 보면, 각각 화풍에 맞게 필터링 되었지만, 모두 정보의 왜곡을 피할 수는 없었다. 물론 회화로 만드는 알고리즘이니 만큼 왜곡은 있을 수 있지만, 그림6 - (a)에서 동물의 손이 아예 사라지거나, 잡고 있는 나뭇가지의 형태가 불분명해지는 등 실제 사람이 표현할 때와는 차이가 났다. 심지어는 원본에는 없을 것이라 판단되는 색상이 추가되기까지 한다. 이런 디테일의 소실은 실제 화가가 그린 것처럼 특정 부분에 강조를 두고 싶지만 그럴 수 없는 알고리즘의 한계를 나타낸다.

따라서 이 논문에 나온 알고리즘으로는 특정 화풍을 성공적으로 모방해 낼 수는 있지만 실제 그림처럼 특정 디테일들을 전부 살릴 수는 없다는 것을 알 수 있다. 이 점에 대한 보완 가능성은 결론에서 얘기해보도록 한다.

결론

1. 보완 가능성

논문에 기재된 실험 결과 이미지들을 살펴보며, 한계점을 알아보았는데, 그것은 바로 실제 화가가 그렸을 때처럼 강조하고 싶은 부분에 대해서 왜곡이 일어날 수 있다는 점이다. (동물을 그렸는데 발의 윤곽이 드러나지 않는 등)

이 점을 보완할 수 있는 방법을 생각해 보았다. 논문에서는 여러 가지 매개변수들을 이용해 이미지 필터링을 한다. 하지만 이것은 항상 이미지 전체에 해당되는 필터링이 되게 된다. 따라서 알고리즘을 그대로 차용하되, 필터링 수행 전에 디테일을 살리고 싶은 영역을 지정하는 방법이다. 이는 그림을 오려내는 것과는 다르게, 이미지의 특정 좌표를 지정하면, 그 좌표를 기점으로 사용자가 설정한 범위만큼의 디테일을 살리고 정보 소실을 방지하는 것이다.

이 방법은 먼저 그림 전체를 필터링 한 후, 설정한 영역들을 원본에서 다시 참조해 필터링을 하는 방법으로 구현할 수 있을 것이다.

2. 총평

1998년 뉴욕 대학교에서 발표한 이미지를 회화 화풍으로 필터링하는 것에 대한 기술 논문을 읽어보았다. 논문을 읽고 초기에는 알고리즘이 이미지를 실제 화가가 그린 그림처럼 잘 필터링 하는 것을 보고 놀랐다. 허나 실험 결과를 자세히 살펴보니 단점이 드러나 보였고, 그럼에도 보완한다면 지금 쓰이고 있는 필터들처럼 잘 활용이 될 기술처럼 느껴졌다.

앞서 멀티미디어 프로그래밍 중간고사 이전에 했던 강의에서 이미지의 색 변형, 대비 변화, 윤곽선 강조, 평균 필터링, 가우시안 필터링 등의 기법들을 배웠는데, 이 논문에서도 강의에서 배운 부분들을 일부 사용하는 것을 볼 수 있었다(대표적으로는 가우시안 필터링). 덕분에 논문이 훨씬 더 쉽게 읽혔다.