과제4: Painterly Rendering 구현하기

22011831 김형규

목차

1. 논문 요약

1-1. 개요

이 논문에서는 사진을 사람이 그린 그림처럼 표현할 수 있는 기술인 Painterly Rendering에 대해서 소개하고 있다. Painterly Rendering에서는 화가가 실제로 그림을 그릴 때 따르는 절차를 구현해 놓은 것이 특징이다.

화가들은 보통 큰 붓으로는 그림의 대략적인 스케치를 하고, 작은 붓으로는 그림의 세세한 디테일을 추가한다. Painterly Rendering에서도 먼저 큰 붓을 이용하여 캔버스에 대략적인 스케치를 하고, 붓의 크기를 줄여가면서 그림을 그린다. 단, 그림을 그릴 때는 캔버스와 원본 이미지의 차이가 큰 곳에 붓의 시작점을 두고, 이미지의 변화율(Image Gradient)의 수직방향으로 붓을 이동시키면서 그린다. 만약, 굳이 그리지 않고도 원본 이미지와 캔버스가 비슷하다면, 그리지 않는다. 그림의 모든 부분에서 위의 작업을 수행하고, 붓의 크기를 줄인다. 이 작업을 반복하다 보면, 최종적인 결과물은 여러 개의 층(Layer)으로 이루어져 있다고 할 수 있다.

1-2. Paint()함수

결과 이미지는 여러 개의 Layer로 구성되어 있다. Paint(sourceImage, R) 함수는, 인자로 원본 이미지와 시작하는 붓의 크기를 받아 결과물을 반환하는 함수이다.
이 함수에서는 원본 이미지와 같은 크기의 canvas를 생성한 뒤에, 각 R마다 paintLayer() 함수를 수행한다. paintLayer()함수의 인자로는 canvas, 레퍼런스 이미지, R을 넘기는 데, 여기서 '레퍼런스 이미지'란, 원본 이미지를 가우시안 블러링한 이미지를 말한다. 즉, canvas를 그릴 때는 원본 이미지가 아니라 레퍼런스 이미지를 보고 그리게 된다. 가우시안 블러링에 사용하는 가우시안 필터의 크기는 R에 비례한다. 즉, 붓이 클 때는 그림을 많이 뭉개고, 붓이 작을 때는 그림을 조금만 뭉갠다. 이 작업은 붓의 크기에 따라 그림의 디테일에 집중할지, 아니면 대략적인 스케치를 할지 목적이 달라지기 때문에 그 목적에 따라 원본 이미지를 조금씩 변형시켜 주기 위해서 수행한다.

1-3. paintLayer() 함수

paintLayer()함수는 결과물을 구성하는 각 층을 그리는 함수이다. 인자로는 canvas, 레퍼런스 이미지, R을 넘겨받는다. 이 함수에서는 그릴 선의 위치와 색깔 등을 정한 뒤, 그 선이 배열 S에 저장된다. 나중에 배열 S에 있는 선 들을 무작위 순서로 그리면, 하나의 층을 생성할 수 있다.
선의 위치와 색깔을 정할 때는, 그림을 격자 무늬로 나누어야 한다. 이 격자를 Jittered Grid라고 한다. Jittered Grid의 각 격자는 정사각형 모양이며, 한 변의 길이는 R에 비례한다. 즉, R이 클수록 격자 하나는 커지고, 전체 격자의 개수는 적어진다.
Jittered Grid의 각 격자마다, 최대 오차와 평균 오차를 계산해야 한다. 최대 오차는 그릴 선의 시작 위치와 색깔을 정할 때 사용되고, 평균 오차는 선을 그릴지 말지 결정할 때 사용된다. 여기서 말하는 오차는 레퍼런스 이미지와 캔버스의 오차를 말하며, 오차는 하나의 픽셀마다 RGB성분의 각 차이를 제곱한 뒤 모두 더한 값에 루트를 씌워서 구할 수 있다. 만약 평균 오차가 미리 정해 놓은 파라미터 값보다 크다면, 최대 오차가 생기는 좌표가 선의 시작 위치가 되고, 선의 색깔은 그 위치에서의 레퍼런스 이미지의 컬러 값이 된다. 이 지점에서 makeStroke()함수를 호출하면 단순히 반지름이 R이고, 원의 중심이 시작 위치이고, 색깔이 시작 위치에서의 컬러 값인 원을 그릴 수 있다.
또는, makeSplineStroke()함수를 호출하여 시작 좌표, R, 레퍼런스 이미지를 인자로 넘겨주면 그 위치에서 시작하는 SplineStroke를 그릴 수 있다. 이 선을 배열 S에 추가한 뒤, 모든 격자에서 이 작업을 반복한다. S에 있는 선 들을 무작위 순서로 그리면 하나의 층을 생성하는 작업이 끝난다.

1-4. makeSplineStroke() 함수

SplineStroke란, 곡선을 표현하기 위해 등장한 개념이다. 직선을 그릴 때 각도를 조금씩 틀면서 선을 그리면 곡선처럼 보이게 된다. 이렇게, 직선을 여러 번 꺾어서 곡선처럼 표현한 선을

SplineStroke라고 하고, SplineStroke의 꺾이는 각 지점을 Control Point라고 한다.
makeSplineStroke()함수는 인자로 넘겨받은 시작 위치에서 시작하는 SplineStroke를 반환하는 역할을 한다. Spline Stroke를 그리려면 여러 개의 Control Point들을 찾는 것이 중요하다. Control Point를 계산할 때는 현재 좌표에서의 Image Gradient를 먼저 구해야 한다. Image Gradient란, 이미지의 색깔이 변하는 방향을 나타내는 일종의 벡터이다.
x방향으로의 Image Gradient는 단순히 현재 좌표의 오른쪽 좌표에서의 컬러 값에서 왼쪽 좌표에서의 컬러 값을 빼 주면 된다.
y방향으로의 Image Gradient 역시, 아래쪽 좌표의 컬러 값에서 위쪽 좌표의 컬러 값을 빼 주면 된다.
여기서 주의해야 할 점은, 실제로 붓은 Image Gradient의 수직 방향으로 움직인다는 점이다. 즉, 어떤 한 지점에서의 Image Gradient를 구했으면, 그 좌표에서 Image Gradient에 수직인 방향으로 R만큼 간 곳에 새로운 Control Point가 있다. 새로운 Control Point에서의 Image Gradient를 다시 계산한 뒤, 다시 새로운 Control Point를 찾는 과정을 반복한다.
이 과정은 다음 세가지 조건을 만족할 때까지 반복한다.

1. SplineStroke가 최대 길이에 도달했을 때

2. Image Gradient가 0일 때

3. Spline Stroke가 최소 길이보다 길고, 마지막 Control Point에서 레퍼런스 이미지의 컬러 값과 선의 색깔 사이의 오차가 같은 위치에서 레퍼런스 이미지와 canvas 사이의 오차보다 커졌을 때

Control Point를 찾는 과정이 종료되면, 맨 처음 좌표에서 시작하여 각 Control Point를 이은 선을 반환한다.

2. 논문을 코드로 구현하기

먼저, 직접 설정해줘야 하는 파라미터들을 정의해준다.

```
#define R 32     //원의 반지름 (또는 Stroke의 크기)
#define G 4      //가우시안 블러링할 때 반지름에 곱해지는 변수
#define fg 1     //jittered Grid를 생성할 때 반지름에 곱해지는 변수
#define T 20     //threshold값 (그림의 거친 정도를 결정하는 변수)
#define fc 1.0f //Curved Brush Stroke의 곡률을 결정하는 변수
#define MAX_STROKE_LENGTH 10    //Curved Brush Stroke를 이루는 점들의 최대 개수
#define MIN_STROKE_LENGTH 2     //Curved Brush Stroke를 이루는 점들의 최소 개수
```

- R: 첫 번째 층을 그릴 때 사용하는 붓의 크기 (=32)
- G: 가우시안 블러링을 할 때 R에 곱해지는 변수 (=4)
- fg: Jittered Grid를 생성할 때 R에 곱해지는 변수. 즉, 격자의 한 변의 길이는 fg*R이 된

다. (=1)

- T: 각 격자마다 선을 그릴지 말지 결정할 때 사용하는 변수. 이 값이 작을수록 원본과 닮은 이미지가 만들어진다. (=20)
- fc: Spline Stroke의 곡률을 결정하는 변수. (=1.0)
- MAX_STROKE_LENGTH: Spline Stroke를 이루는 점들의 최대 개수 (=10)
- MIN_STROKE_LENGTH: Spline Stroke를 이루는 점들의 최소 개수 (=2)

그 다음은, 논문 구현에 필요한 구조체들을 정의한다.

- Grid 구조체
  - 격자의 한 변의 길이
  - 격자판의 열 개수
  - 격자판의 행 개수
- Circle 구조체
  - 원의 반지름
  - 원의 색깔
  - 원의 중심의 좌표
- Stroke 구조체
  - Stroke의 시작좌표
  - Control Point의 배열(선이 움직이는 경로)
  - Control Point의 개수
  - Stroke의 크기
  - Stroke의 색깔

그림을 그릴 때 사용하는 붓의 크기(r)는 R로 초기화한다. 각 층을 그리고 나면 r을 2로 나눈 뒤, 작아진 붓으로 층을 다시 그린다. 이 작업을 5번 반복하여, 총 5개의 층을 그릴 것이다. 각 층에서 수행해야 하는 작업은 다음 절차를 따라서 진행된다.

1. 레퍼런스 이미지 생성하기

원본 이미지에 cvSmooth()함수를 이용해 가우시안 블러링을 적용하여 레퍼런스 이미지를 생성한다. cvSmooth()함수에 인자로 G*r을 넘기면 그림을 뭉개는 정도를 r에 비례하게 할 수 있다.

2. Jittered Grid 생성하기

Grid 구조체를 이용하여 Jittered Grid를 생성한다. 각 격자의 한 변의 길이는 fg*r이다. 레퍼런스 이미지의 가로 길이를 w, 세로 길이를 h라고 하면,

격자의 열의 개수: (w/(격자의 한 변의 길이)) + 1
격자의 행의 개수: (h/(격자의 한 변의 길이)) + 1

이다. 이렇게 Grid를 생성하고 나면 각 격자마다 선을 그릴 것인지 말 것인지 정해야 한다.

2-1. Jittered Grid 안에서 평균 오차와 최대 오차 찾기

오차는 getDifference(CvScalar f, CvScalar g); 함수를 통해 계산한다. getDifference()함수에서는 인자로 넘겨 받은 두 CvScalar형 변수의 각 RGB성분의 차이를 제곱하여 모두 더한다. 그 값에 루트를 씌워 반환한다. 격자 안의 모든 점에서 오차를 계산하여 평균 오차와, 최대 오차가 생긴 점의 좌표를 구한다. 만약 평균 오차가 T보다 크다면, 최대 오차가 생긴 좌표에서 시작하는 Spline Stroke를 구한다.

3. Spline Stroke 구하기

Control Point를 구하기 전에, Stroke구조체의 멤버변수를 초기화 해준다. 시작 좌표는 최대 오차가 생긴 곳의 좌표, Stroke의 크기는 r, Stroke의 색깔은 최대 오차가 생긴 곳의 레퍼런스 이미지의 CvScalar 값으로 초기화 한다. Control Point들은 CvPoint형 배열로 표현할 수 있다. CvPoint형 배열을 MAX_STROKE_LENGTH만큼 동적할당 받는다. Control Point의 개수는 0으로 초기화 한다.

Control Point를 구하려면 Image Gradient를 먼저 구해야 한다. (x, y)에서의 Image Gradient를 (gx, gy)라고 하고, (x, y)에서의 CvScalar값을 L(x, y)라고 하자.
Image Gradient를 계산할 때는, 현재 좌표를 기준으로 왼쪽, 오른쪽, 위, 아래의 컬러 값을 가져와야 한다. 즉,

$$gx = L(x+1, y) - L(x-1, y)$$
$$gy = L(x, y+1) - L(x, y-1)$$

이다. 여기서 좌표 p1=(x-1, y-1), p2=(x+1, y+1)라고 하면,

$$gx = L(p2.x, y) - L(p1.x, y)$$
$$gy = L(x, p2.y) - L(x, p1.y)$$

이다. 만약 p1이나 p2가 이미지 밖이라면, p1과 p2를 이미지 안쪽으로 이동시킨다. 예를 들어, 현재 좌표가 이미지의 왼쪽 모서리에 붙어 있다면 p1.x가 정의되지 않는다. 이럴 때는 gx를 L(x+1, y)–L(x-1, y)가 아닌 L(x+1, y)–L(x, y)로 정의한다.

이렇게 gx와 gy를 계산하고 나면, <gx, gy>를 Image Gradient를 표현하는 벡터처럼 생각할 수 있다. 실제 그림은 Image Gradient의 수직방향으로 그려지므로, <gx, gy>에 수직인 벡터 <dx, dy>를

구해야 한다. 어떤 벡터의 수직인 벡터를 구하는 방법은 x성분과 y성분의 위치를 바꾼 뒤, 둘 중 하나의 부호를 바꾸면 된다. 즉,

$$< dx, dy > = < -gy, gx > or < gy, -gx >$$

이다. 위의 식에서 알 수 있듯이, <gx, gy>에 수직인 벡터는 두 개가 있다. 어느 벡터를 선택할지는 마지막으로 구했던 <dx, dy>에 따라서 결정되는데, 마지막으로 구한 <dx, dy>를 <lastdx, lastdy>라고 하면, <dx, dy>의 방향은 <lastdx, lastdy>의 방향과 크게 달라지지 않아야 한다.

$$lastdx * dx + lastdy * dy \geq 0$$

위의 식을 만족한다면, 두 벡터의 방향은 크게 다르지 않다고 할 수 있다. 만약 식을 만족하지 않는다면, <dx,dy> = <-dx,-dy>로 초기화 해준다.

벡터 <dx, dy>의 곡률 또한 파라미터를 이용해서 컨트롤할 수 있다. 미리 정의된 파라미터인 fc를 이용하면, <lastdx, lastdy>가 <dx, dy>에 영향을 끼치는 정도를 정할 수 있다. 즉,

$$< dx, dy > = f_c * < dx, dy > + (1 - f_c) * < lastdx, lastdy >$$

로 표현할 수 있다. fc는 0이상, 1이하의 값을 갖는데, 0에 가까울수록 <lastdx, lastdy>가 <dx, dy>에 끼치는 영향이 많아진다.

벡터 <dx, dy>는 방향만 표현하는 단위벡터처럼 사용되어야 하므로 벡터의 크기를 1로 바꿔주어야 한다. 벡터의 각 성분을 벡터의 크기로 나눠주면 벡터를 단위벡터로 바꿀 수 있다. 이를 수식으로 표현하면 다음과 같다.

$$< dx, dy > = < dx, dy > / \sqrt{dx^2 + dy^2}$$

즉, 현재 위치 <x, y>에서 <dx, dy>방향으로 r만큼 떨어진 곳에 새로운 Control Point가 있는 것이다. 새로운 Control Point의 x좌표를 nx, y좌표를 ny라고 하면,

$$< nx, ny > = < x + dx * r, y + dy * r >$$

이다.

이렇게 하면 새로운 Control Point를 구할 수 있다. 이제 <lastdx, lastdy>에 <dx, dy>를 대입하고, <x, y>에 <nx, ny>를 대입한 뒤 위의 작업을 반복한다. 이 작업이 종료되는 조건은 세가지이다.

(1). Spline Stroke가 최대 길이에 도달했을 때
Control Point의 개수가 MAX_STROKE_LENGTH일 때 반복문을 탈출한다.

(2). Image Gradient가 0일 때
벡터 <gx, gy>의 크기가 0일 때 반복문을 탈출한다.

(3). Spline Stroke가 최소 길이보다 길고, 마지막 Control Point에서 레퍼런스 이미지의 컬러 값과 선의 색깔 사이의 오차가 같은 위치에서 레퍼런스 이미지와 canvas 사이의 오차보다 커졌을 때
Control Point의 개수가 MIN_STROKE_LENGTH보다 크고, 레퍼런스 이미지의 <x, y>에서의 색깔과

Stroke의 색깔의 차이가 canvas의 <x, y>에서의 색깔의 차이보다 커지면 반복문을 탈출한다.

모든 격자에서 연산을 진행하고 나면, Stroke배열의 원소들을 무작위로 섞은 뒤, canvas에 그리면 된다. shuffleStrokeArr(Stroke* arr, int size); 함수를 이용하면 Stroke형 배열 arr에 있는 원소들을 무작위 순서로 섞는다. I를 0부터 size까지 돌려 각 i마다 0~size사이의 무작위 난수를 생성한 뒤, 생성한 난수 위치에 있는 원소와 i번째 원소의 위치를 바꾸면, 배열을 무작위로 섞을 수 있다.

무작위로 섞인 배열의 각 원소에서 drawSplineStroke(IplImage* img, Stroke stroke) 함수를 호출하여 해당 원소에 대항하는 Spline Stroke를 그린다. cvLine()함수를 이용하여 시작지점과 도착지점을 바꿔가면서 그리면 Spline Stroke를 그릴 수 있다.

배열에 있는 모든 Spline Stroke를 그리고 나면, 하나의 층을 그리는 작업이 종료된다. 붓의 크기인 r을 반으로 줄인 뒤, 1~3의 과정을 4번 더 반복하면 총 5개의 층으로 이루어진 결과 이미지를 생성할 수 있다.

그리기 모드가 원일때는, Stroke구조체의 멤버변수를 초기화 했던 것과 같은 방식으로 Circle구조체의 멤버변수를 초기화하면 된다. Control Point를 구하는 과정을 생략하고, 배열을 무작위로 섞은 뒤 각 원소마다 drawCircle(IplImage* img, Circle circle); 함수를 호출하여 원을 그린다. drawCircle() 함수는 cvCircle()함수를 이용하여 원을 그리는 함수이다. 원을 그린 뒤에는 같은 방식으로 r을 반으로 줄인 뒤, 위의 과정을 반복하면 그리기 모드가 원일 때 결과 이미지를 생성할 수 있다.