규칙성을 가진 벡터 텍스처의 합성에 관한 연구

Synthesis of Near-Regular Vector Texture Patterns

서재우, JaeWoo Seo, 프레데릭 코디어, Frederic Cordier
한국과학기술원 문화기술대학원, 비주얼미디어 연구실

요약 컴퓨터에서 사용되는 2D 이미지는 크게 비트맵과 벡터의 두 가지 표현 방식이 존재한다. 일반적으로 사용되는 이미지와 텍스처는 대부분 비트맵을 기반으로 하고 있으며, 이에 따라 많은 텍스처 합성에 관한 연구 또한 비트맵 기반으로만 이루어져 왔다. 그러나 일부 분야들에서는 몇 가지 단점에도 불구하고 벡터 형식의 이미지를 선호하고 있으며, 비트맵이 가지지 못한 장점들과 현재의 충분한 컴퓨터 연산 능력을 고려해 볼 때 벡터 이미지의 필요성과 활용분야는 앞으로도 늘어날 것이라 생각된다. 이에 따라 본 논문에서는 벡터 형식으로 주어진 텍스처 패턴을 분석, 합성하는 새로운 방법을 제안한다. 입력 받는 벡터 이미지는 몇 가지의 속성을 지닌 스트로크(Stroke)들의 집합으로서, 각각의 스트로크는 비트맵에서의 픽셀과 같이 기본적인 분석과 합성의 단위가 된다.

핵심어: CG, Vector, Texture, Stroke, Pattern, Synthesis

1. 서론

텍스처는 일반적으로 어떠한 객체가 지닌 특유한 재질감을 말한다. 컴퓨터 그래픽스 분야에서는 여러 개의 다각형으로 이루어진 모델(Model) 위에 텍스처를 입히는'텍스처 맵핑(Texture Mapping)'과정을 통해 가상의 객체(Object)에 재질감을 표현하게 된다. 2D 이미지로 정의된 텍스처 패턴은 컴퓨터 그래픽스를 비롯한 벽지나 의복 등의 분야에서도 폭넓게 사용된다.

텍스처 합성은 텍스처를 만들기 위한 몇 가지 방법 중 하나이다. 일반적으로 사용되는 비트맵 이미지 기반의 텍스처는 면적에 비례하여 용량이 커지게 되므로, 넓은 영역에 텍스처를 사용하는 경우에는 보통 반복적으로 텍스처를 이어 붙이는 방법을 사용하게 된다. 이는 가장 쉽고 유용한 방법이지만 약간의 문제점이 있는데, 먼저 완전한 규칙성(Regularness)을 가진 텍스처가 아니라면, 해당 텍스처에서의 불규칙한 패턴 또한 반복적인 규칙성을 보이게 된다. 또한 반복되는 텍스처의 경계가 부드럽게 이어질 수 있도록 잘 가공되어 있어야 하는데, 이 역시 능숙한 아티스트가 아니라면 쉽게 해결할 수 있는 문제는 아니다.

이럴 때 유용하게 사용될 수 있는 방법이 텍스처 합성이다. 지금까지 연구된 텍스처 합성 방법을 사용하면 비교적 적은 노력으로 넓은 면적의 텍스처를 만들어 낼 수 있으며, 타일링을 할 수 있도록 부드럽게 이어지는 경계선을 자동적으로 생성할 수도 있다[10].

텍스처의 종류를 나누는 데에는 여러 가지 방법이 있겠으나, 본 논문에서는 텍스처가 가진 규칙성을 기준으로 하여 <그림1>의 다섯 가지 범주로 구분하도록 한다.

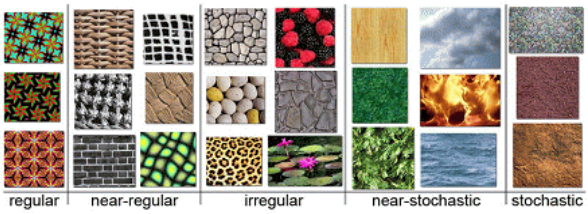

그림 1. 규칙성에 따른 텍스처의 분류

2. 관련 연구

지금까지 이루어진 텍스처 합성에 관한 연구는 크게 매개변수를 이용한 방법(Parametric Approach)과 예제에 기반한 방법(Example-based Approach)으로 나눌 수 있다.

매개변수를 기반으로 한 방법은 주어진 몇 가지의 변수를 미리 정해진 함수에 입력하여 나온 결과를 바탕으로 텍스처를 합성하는 방법으로서, 합성에 필요한 입력의 정보량이 적어 용량에서의 이득을 얻을 수 있으며, 어떠한 재질에 대해 잘 만들어진 합성 함수가 존재한다면 그에 대해 꽤 자연스러운 결과를 비교적 빠른 시간에 얻어낼 수 있다는 장점이 있다. 그러나 각 생성 함수는 특정 종류의 재질에 맞추어져 만들어져 있기 때문에 범용적인 합성에는 사용하기 어려운 문제점이 있다. 매개변수 기반의 방법은 주로 구조성을 가지지 않은, 통계적 불규칙성을 가진 텍스처의 합성에 이용된다.

예제를 기반으로 한 텍스처 합성은 입력으로 하나의 이미지를 사용하는데, 매개변수 기반보다 좀 더 일반적인 경우에서 사용이 가능하며 결과의 품질도 뛰어나다. 이 분야의 연구의 시초에 해당하는 [1]과 [10]에서는 픽셀 기반(Pixel-based)의 접근을 시도하고 있다. 이 방법은 주로 새로이 합성될 픽셀에 대해, 주위에 이웃한 픽셀들을 비교하여 원본 텍스처에서 가장 비슷한 부분의 픽셀을 가져오는 방식으로 합성을 수행한다. 이는 텍스처 합성뿐만 아니라 일부 영역이 손상된 이미지를 복구하는 데에도 효과적으로 쓰일 수 있다. 그러나 픽셀 단위의 합성과 비교 과정으로 인해 처리속도가 느린 단점이 있는데, 이러한 문제점을 보완하기 위하여 픽셀보다 큰 이미지 조각을 이어 붙이는 방법으로 합성을 수행하는 시도들도 있었다([2], [4], [15]). 패치 기반 합성으로 불릴 수 있는 이러한 방법들은 픽셀 기반 방법에 비해 일반적으로 더 빠르고 효율적이다.

텍스처 합성에 관해 위와 같이 다양한 시도들이 있어 왔지만, 이들의 거의 비트맵 기반의 텍스처를 위한 방법이었다는 공통점을 가지고 있다. 벡터 기반 텍스처의 합성에 관한 연구가 시작된 것은 매우 최근의 일이다. [12]에서는 입력된 벡터 스트로크(Stroke)들의 특성을 시각적 인식 구조에 기반한 몇 가지 방법으로 분석하고, 이를 기반으로 새로운 스트로크를 합성하는 방법을 처음으로 제시하고 있다. 이는 예제 기반 (Example-based) 비트맵 텍스처 합성으로 유명한 [1, 10]의 영향을 직접적으로 받아, 픽셀 기반의 비트맵 텍스처 합성을 스트로크 기반의 벡터 텍스처 합성으로 응용한 경우이다. 이는 비트맵에서와 유사한 접근을 통해 비교적 처리과정이 간단한 장점이 있으나, 항상 균일한 분포(Uniform Distribution)를 가진 패턴만을 합성할 수 있다는 단점을 가지고 있다. 이는 구조적 규칙성은 있으나 스트로크의 분포가 균일하지 않은 패턴은 합성할 수 없음을 의미한다(그림 2).

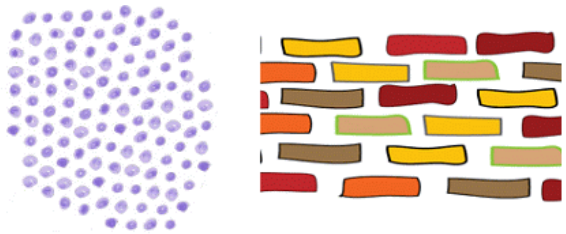

그림 2.
규칙적이고 균일한 분포(좌)와
규칙적이지만 균일하지 않은 분포(우)의 예

본 연구에서는 [12]에서의 문제점을 개선하여, 주어진 패턴의 규칙적인 분포 구조를 유지할 수 있는 새로운 벡터 기반 텍스처 합성 방법을 제시한다. 이는 컴퓨터 그래픽스는 물론 벡터 기반의 패턴을 필요로 하는 의복이나 벽지 등에서의 시각 디자인 분야에서도 유용하게 사용될 수 있다.

3. 알고리즘

전체 알고리즘은 크게 준비 단계 (Preparing Stage)와 성장 단계 (Growing Stage)로 나뉜다. 준비 단계에서는 합성을 시작하기 전에 필요한 기초적인 작업들을 거치고, 성장 단계에서는 앞에서 만들어진 자료를 토대로 실질적인 합성을 수행한다.

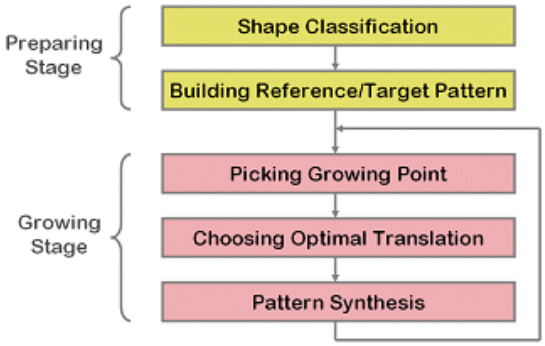

그림 3. 전체 파이프라인(Pipeline) 개요

3.1 Shape Classification

입력된 모든 스트로크들은 먼저 주어진 기준에 따라 유사한 종류의 개체들끼리 분류되어야 한다. 예를 들어 다음 패턴은 세 가지 종류의 모양으로 구성되어 있음을 알 수 있다. 본 단계에서는 이처럼 비슷한 모양의 개체를 판단하여 몇 가지 종류(Class)로 스트로크들을 정리하게 된다.

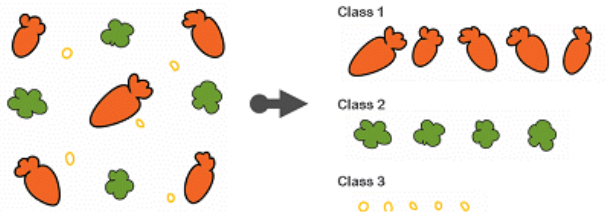

그림 4. 스트로크 분류 예

전체적인 처리 과정은 [12]의 “Element Clustering”단계와 유사한 접근으로서, 처리의 효율성을 위해 Greedy Algorithm을 사용하였다. 먼저 모든 스트로크들은 하나의 큐(Queue) Q_s에 저장된다. 각 단계마다, Q_s에서 처음으로 나오는 스트로크 Si는 또 다른 큐 C_i에 넣어지고, Q_s에서 S_i의 뒤에 위치했던 스트로크 S_j와 Match라는 함수를 통해 비교된다. S_i와 S_j에 대한 Match 함수의 결과가 참이면 S_j는 C_i에 추가되면서 Q_s에서 삭제되며, 거짓이라면 다시 Q_s로 들어간다. 이렇게 Q_s와 C_i의 스트로크에 대해 한 번씩 Match함수를 수행하여 더 이상 참이 나오지 않을 때까지 이 작업을 반복하면 하나의 유사 스트로크 집합(Cluster) C_i가 만들어 진다. 이 작업을 Q_s가 빈(Empty) 상태가 될 때까지 계속 수행하여 모든 스트로크가 몇 개의 집합으로 나누어지도록 한다.

하나의 스트로크 S_i는 네 가지 종류의 속성을 가지며, 이들은 두 스트로크의 유사성을 판단하기 위한 기반이 된다.

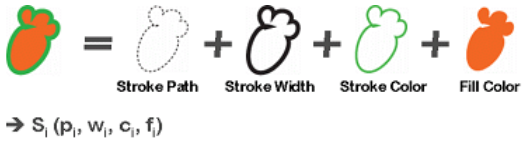

그림 5. 스트로크의 속성

- Stroke Path (p_i): 형태를 나타내는 선의 경로로서 직선으로 연결된 여러 2차원 점들의 배열(Polyline)로 나타내어 진다.
- Stroke Width (w_i): 실수 값으로 정의된 선의 두께이다.
- Stroke Color (c_i): 선의 색상으로 채널 당 0~255의 범위를 가지는 RGB값이다.
- Fill Color (f_i): 스트로크의 내부를 채우는 색상으로서 Stroke Color와 마찬가지로 RGB로 나타내어진다.

두 스트로크 S_i와 S_j를 비교하는 함수 Match()는 다음과 같다.

$$Match(S_i, S_j) = (Dist(p_i, p_j)<\varepsilon_1)\wedge (Dist(p_i, p_j)<\varepsilon_2)\wedge (Dist(p_i, p_j)<\varepsilon_3)\wedge (Dist(p_i, p_j)<\varepsilon_3) \quad (1)$$

각각의 속성들은 Dist()함수를 통해 개별적으로 평가되는데, 이 결과 값들은 유저에 의해 주어진 세 종류의 경계 값(Threshold) ε_1~ε_3 과 비교되어 참 혹은 거짓의 결과가 되면서 ∧(and)연산을 통해 조합된다. 즉, 각 속성의 유사도가 모두 정해진 일정 범위 안에 들어오게 되면 유사한 스트로크로 판단한다.

Dist()는 스트로크가 가진 각각의 속성을 비교하는 함수이다. RGB색상인 c_i와 f_i를 제외하면 각 속성의 성질이 다르므로 서로 다른 방법과 경계 값이 사용되었다.

Stroke Width 속성은 두 값 차이의 절대값을, Stroke Color와 Fill Color는 RGB색상공간에서의 유클리드 거리(Euclidean Distance)를 사용한다. Stroke Path 속성에 대해서는 형태 비교를 위한 거리 측정 방법 (Shape Matching Distance Metric)중 하나인 Fréchet Distance를 사용하였으며, 이동(Translation)을 제외한 회전(Rotation)이나 크기(Scale)변환에 의한 비교는 고려하지 않았다. 즉, 비슷한 경로를 가졌다 하더라도 크기나 회전 상태가 다른 경우에는 다른 형태로 취급된다. 주어진 두 도형의 Fréchet Distance가 주어진 값 ε_1보다 작은 지를 결정하는 문제는 일반적으로 의적지 않은 시간이 소요되는데, Dynamic Programming을 적용하여 가능한 효율적으로 계산할 수 있도록 구현하였다.

Dist()함수의 결과 값을 비교하기 위한 경계 값 상수(Threshold Constant) ε_1~ε_3는 사용자에 의해 주어지도록 함으로써 입력되는 텍스처의 특성에 따라 유동적으로 적용되도록 하였다. 경계 값들의 조절에 의해, 본 분류 과정에서 도형들의 형태와 색상, 두께 중 비중 있게 고려되어야 할 부분을 결정할 수 있으며, 이는 이후 합성되는 패턴의 결과에도 영향을 미치게 된다. 원하는 패턴으로 합성되도록 하기 위해서는 이 경계 값들의 신중한 설정이 필요하다.

3.2 Building Reference/Target Pattern

합성의 진행을 위해서는 참조 패턴(Reference Pattern)과 대상 패턴(Target Pattern)이라는 두 종류의 패턴이 필요하다. 참조 패턴은 합성이 진행되면서 참조하는 원본과 같은 패턴으로서, 입력된 패턴과 동일한 형태로 초기화 되어 합성 작업에서 계속적으로 참조만 될 뿐 새로운 정보가 쓰여 지지 않는다. 대상 패턴은 앞으로 실제 패턴 합성이 행해져 새로운 정보들이 계속적으로 쓰여 질 패턴으로서 합성될 만큼의 면적을 가진 백지와 비슷하다.

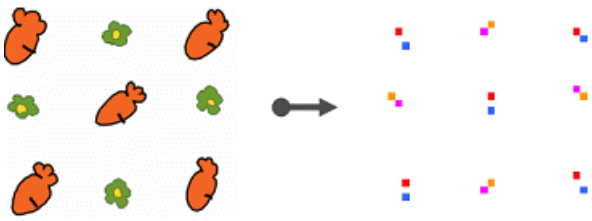

그림 6. 참조 패턴의 생성

위에서 분류된 각 스트로크를 하나의 점 요소(Element)로 치환시키는 과정을 통해 참조 패턴 P_{ref}를 만든다(그림 6). 각 요소에는 원래의 스트로크 정보와 앞 단계에서 분류된 클래스의 번호가 함께 기록된다. 또한 벡터 이미지에서는 스트로크들이 그려지는 순서가 중요하므로, 마지막의 그리기 단계를 위해 원래 이미지에서 스트로크들이 그려진 순서 또한 각 요소에 저장하여 둔다.

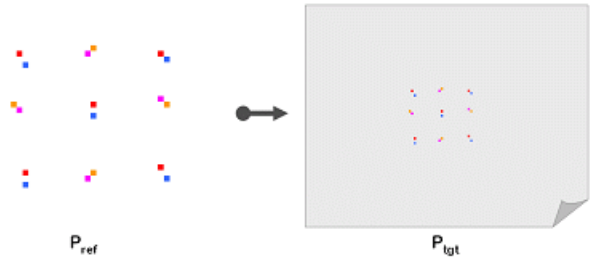

그림 7. 참조 패턴과 대상 패턴

대상 패턴 P_{tgt}는 먼저 만들어진 빈 공간의 중심에 참조 패턴을 그대로 한 번 복사함으로써 초기화 된다(그림 7). 이후 합성 단계는 중앙에서부터 남은 빈 공간을 채워나가는 형태로 진행될 것이다. 성장 단계를 위한 준비는 이렇게 모두 끝나게 된다.

3.3 Picking Growing Point

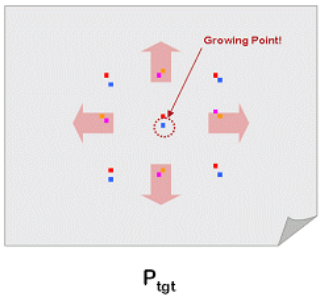

그림 8. 성장 방향과 지점의 선택

성장 단계에서의 합성의 기본 아이디어는 대상 패턴에 존재하는 요소들이 가진 규칙성에서 오차를 최소한으로 유지하면서 참조 패턴의 요소들을 스탬프를 찍어 나가듯 조금씩 추가하여 나가는 것이다.

새로운 패턴은 대상 패턴의 중앙에서 시작하여 주변을 향해 조금씩 자라 나가는(Growing) 순서로 합성되는데, 본 단계에서는 이를 수행할 기준 위치인 성장 지점(Growing Point)를 고르게 된다. 성장 지점을 고르는 데에는 여러 가지 효율적인 아이디어가 고려될 수 있을 수 있으나, 여기서는 가장 간단한 방법으로 대상 패턴의 중심에서 가장 가까운 성분부터 차례대로 선택하여 나가는 방법을 사용하였다. 한 번 성장 지점으로 선택된 성분은 다시 성장 지점으로 지정되지 않는다. 이는 성장 지점이 패턴의 중심에서 조금씩 바깥 방향으로 뻗어나가는 형태로 선택될 것임을 보장한다.

3.4 Choosing Optimal Translation

이 단계에서는 참조 패턴을 대상 패턴 내에서 최소한의 오차로 이동시키는 변환(Translation)을 구하여 새로이 합성될 요소와 그 위치를 결정한다.

먼저 앞의 단계에서 선택된 성장 지점을 중심으로 참조 패턴을 이동시키는 후보 집합(Candidate Set)을 만든다. 성장 지점과 같은 클래스의 모든 요소들을 참조 패턴에서 찾아내어, 선택된 각 참조 패턴의 요소를 대상 패턴의 성장 지점으로 이동시키는 변환을 만든다.

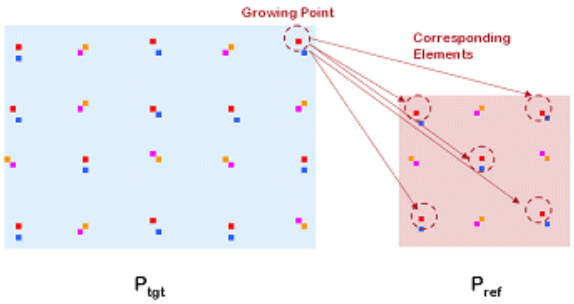

그림 9. 대응 요소의 탐색

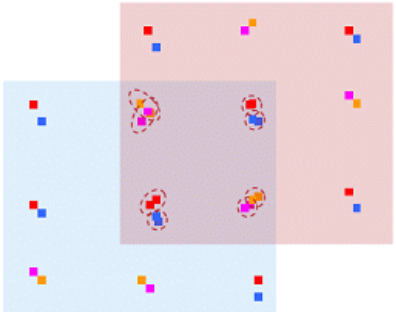

그림 10. 참조 패턴과 대상 패턴의 대응 요소

이렇게 만들어진 이동 변환 집합 T의 모든 변환 t_i에 대하여, t_i에 의해 옮겨진 참조 패턴 P_{ref}와 대상 패턴 P_{tgt} 사이에서 겹쳐지는, 서로 대응되는 요소(Corresponding Element)를 두 패턴에서 찾아 짝지어 주게 된다(그림 10).

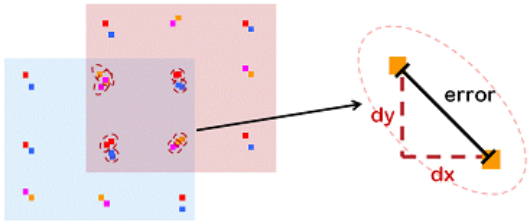

그림 11. 하나의 대응 요소 쌍에서 나타나는 오차

완벽히 규칙적인 패턴이 아니라면, 대응된 성분으로 서로 짝지어진 요소들 사이에는 오차(error)들이 존재한다(그림 11). 이들의 합이 최소한이 되도록 참조 패턴을 조금 이동시킴으로써 각 이동 변환들을 최적화 시킨다. 오차 합(error sum)의 최소화에는 최소 제곱법 (LSM, Least Square Minimization)을 이용하였다. 일반적인 경우 최소 제곱법에서는 선형 시스템을 풀어내는 과정이 필요하지만, 여기서는 패턴들의 이동 변환만을 고려하고 있다는 점을 이용하여 간단히 할 수 있으며, 그 결과는 각 대응점들이 가지는 dx와 dy의 산술평균과 같다.[1] 이에 따라 각 이동 변환에서 대응점들의 dx와 dy의 평균을 구하고, 그만큼씩 참조 패턴을 이동시킴으로써 오차 최소화를 수행한다.

이렇게 최적화가 끝난 변환들의 집합 가운데 가장 작은 오차 합을 가지는 변환을 최종적으로 선택하게 된다. 여기서 다음의 두 가지 사항을 고려해야 한다.

(1) 각 변환들이 가지는 오차 합은 대응점들의 수가 늘어남에 따라 커지는 경향이 있으므로 정규화 과정을 필요로 한다.

(2) 대응점이 하나만 존재하는 변환의 경우 오차 합은 언제나 0이므로, 변환 셋에 이 경우가 존재한다면 항상 이 변환이 선택될 것이다. 그러나 이 변환을 수행하는 경우 이후에 합성되는 성분들은 전적으로 이 대응점을 중심으로 배열, 결정되므로 이는 바람직하지 않다.

이러한 이유로 인해, 모든 오차 합은 대응점의 수만큼 나누어지는 방법으로 정규화를 수행한 후 비교하게 된다. 또한 대응점이 하나만 존재하는 변환은 그 이상의 대응점들을 가진 변환이 존재하는 한 선택하지 않도록 하였다.

3.5 Pattern Synthesis

선택된 변환을 참조 패턴에 적용하여 실제 성분의 합성을 수행한다. 앞서 짝짓기 단계에서 짝이 지어진 참조 패턴의 성분들은 이미 대상 패턴에 해당하는 요소가 존재한다는 뜻이므로 그렇지 않은 요소들만을 대상 패턴에 추가하게 된다.

[1] '첨부 1' 참조

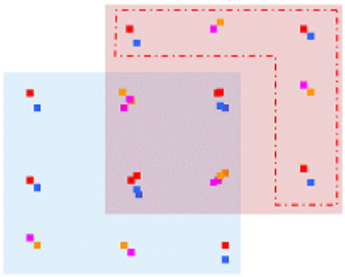

그림 12. 짝지어지지 않은 참조 패턴 요소들

최종 합성 단계에서는 각 스트로크의 속성을 해당 클래스에 있는 성분에서 무작위로 가져오는 방법을 이용해 원래의 입력보다 더 많은 변화(Variation)들을 자연스럽게 이끌어 낼 수 있다.

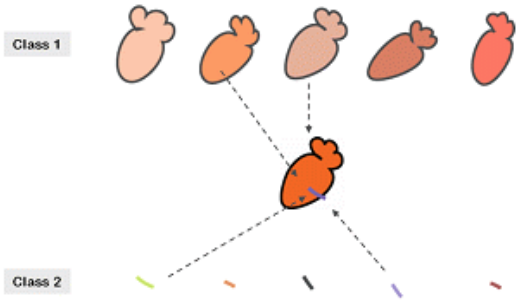

그림 13. 새로운 개체의 합성

4. 결과

알고리즘은 닷넷(.net) 2.0 기반의 윈도우즈용 어플리케이션으로 구현되었으며, 기본적으로 SVG 파일을 통한 입출력을 사용한다. 이는 W3C에서 지정된 XML기반의 벡터 그래픽스 표준으로, 일반적으로 사용되는 대부분의 벡터 그래픽스 프로그램에서 지원하는 높은 호환성을 가진 포맷이다 [20]. 따라서 사용자는 기존에 존재하던 익숙한 프로그램을 이용하여 원하는 패턴 입력을 쉽게 만들어 낼 수 있으며 그 출력도 다시 해당 프로그램으로 가져와 원하는 대로 가공할 수 있다. 또한 스케칭 인터페이스를 통한 프로그램 자체에서의 기본적인 드로잉도 가능하기 때문에, 간단한 패턴은 별도의 전문 프로그램 없이도 직접 그릴 수 있도록 하였다.

합성된 결과는 입력된 패턴이 가진 규칙성을 유지하면서 기존의 속성을 활용한 새로운 성분을 무작위로 합성해내고 있다. 특히 [12]에서 사용된 일부 예제와 유사한 입력을 사용하였을 때 그보다 나은 규칙성을 보이는 결과를 얻을 수 있었다.

그림 14. 1차원 배열 스트로크 합성

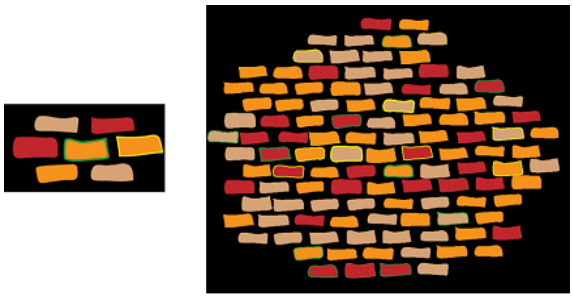

그림 15. 한 종류의 성분으로 이루어진 2차원 배열 텍스처 합성

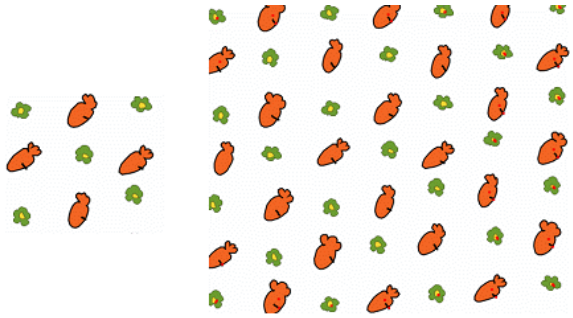

그림 16. 여러 종류의 성분으로 이루어진 텍스처 합성
(형태 중심 스트로크 분류)

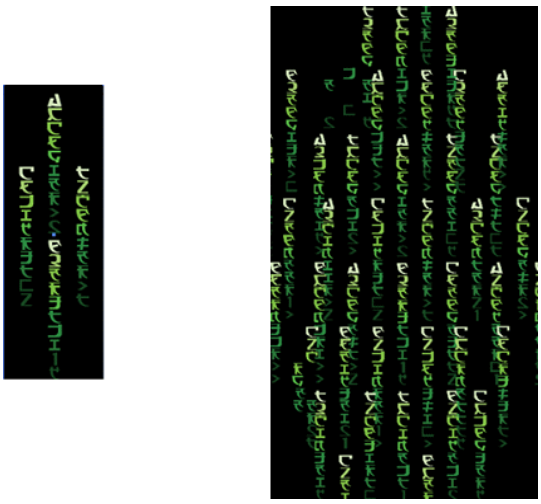

그림 17. 여러 종류의 성분으로 이루어진 텍스처 합성
(색상 중심 스트로크 분류)

그림 18. 새로운 성분들의 무작위 합성

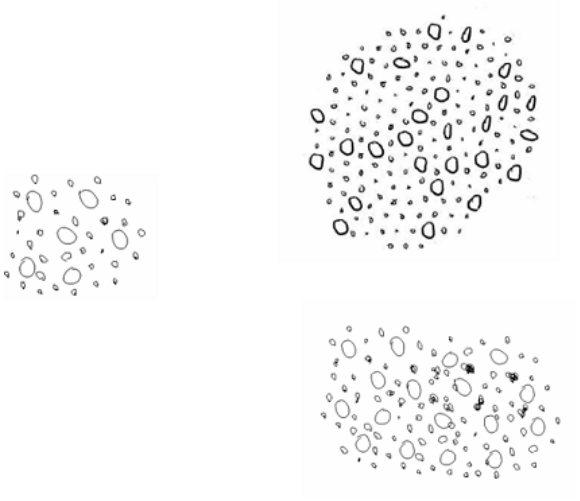

그림 19. 기존 연구 결과와의 비교
(좌) 입력된 예제 (우상) [12]의 결과 (우하) 제안된 방법의 결과

5. 검토 및 결론

5.1 검토

이 방법은 이전의 방법에서 해결할 수 없었던, 균등하지 않은 분포의 규칙성을 가진 텍스처의 합성을 위해 제안되었다. 이는 규칙성을 가진 패턴에 대해서는 잘 동작하지만, 입력된 패턴에서 규칙성이 발견되지 않는(irregular) 경우에는 제대로 된 결과를 얻기 어렵다. 이러한 경우, 보통 대응 성분을 찾아 짝짓는 과정이 제대로 수행되지 못하면서 불필요한 성분들이 노이즈처럼 들어가 합성되는 결과를 가져온다. 이는 입력된 패턴에서 발견되는 반복의 규칙에 전적으로 의지하기 때문으로서, 이러한 경우에도 납득할 만한 결과를 만들 수 있도록 스스로 분포를 결정할 수 있는 방법이 필요하다.

첫 단계인 Shape Classification 부분은 세 종류의 경계값을 올바르게 설정하기 위하여 사용자의 노력이 필요하다는 단점을 가지고 있다. 따라서 이 부분에서 좀 더 자동적으로 분류가 될 수 있도록 하는 Clustering 알고리즘의 보강이 필요하다고 생각된다.

마지막으로 성장 지점을 고르는 방법에 있어서도 개량을 시도할 수 있다. 여기서 사용된 방법은 매우 간단하지만, 많은 경우 이미 성장된 패턴의 내부의 지점을 고르게 됨으로써 불필요한 성장 지점들이 비효율적으로 선택되는 문제가 있다. 합성되는 패턴의 경계에서 우선적으로 합성되어야 부분을 자동적으로 찾아낼 수 있도록 한다면 전체적인 수행 효율의 증가를 기대할 수 있을 것이다.

5.2 결론

지금까지 새로운 벡터 기반 텍스처의 합성 방법을 제안하였다. 이 방법은 기존의 방법에 비하여 입력된 요소간의 구조를 유지하면서 자연스러운 변화도 함께 나타낼 수 있다는 장점을 가지고 있다. 만들어진 프로그램은 표준적으로 널리 사용되는 파일을 지원하여 쉽게 사용될 수 있도록 하였으며, 규칙성을 가지는 입력에 대하여 좋은 합성 결과를 보여준다. 이 프로그램은 벡터 텍스처 패턴이 필요한 여러 분야에서 사용될 수 있다.

참고문헌

[1] Alexei A. Efros, Thomas K. Leung, Texture Synthesis by Non-parametric Sampling, IEEE International Conference on Computer Vision, Corfu, Greece, September 1999.

[2] Alexei A. Efros, William T. Freeman, `Image Quilting for Texture Synthesis and Transfer, Proceedings of SIGGRAPH 2001, Los Angeles, California, August, 2001.

[3] Craig S. Kaplan, David H. Salesin, Escherization, Proceedings of the 27th annual conference on Computer graphics and interactive techniques, 2000, pp. 499-510.

[4] Feng Dong, Hai Lin, Gordon Clapworthy, Cutting and Pasting Irregularly Shaped Patches for Texture Synthesis, Computer Graphics forum Vol. 24 (2005), number 1 pp.17-26.

[5] H. Alt and M. Godau, Computing the Fréchet distance between two polygonal curves, Internat. J. Comput. Geom. Appl., 5:75-91, 1995.

[6] H. Alt, C. Knauer and C. Wenk, Matching polygonal curves with respect to the Fréchet distance, Proc. 18th Int. Symp. Theoretical Aspect of Computer Science (STACS):63-74, 2001, Dresden, Germany.

[7] H. Alt, C. Knauer and C. Wenk, Bounding the Fréchet distance by the Hausdorff distance, Proc. 17th European Workshop on Computational Geometry, 166-169, 2001, Berlin, Germany.

[8] Jean-Michel Dischler, Karl Maritaud, Bruno Lévy, Djamchid Ghazanfarpour, Texture Particles, Computer Graphics Forum (Proc. of Eurographics 2002), Volume 21 - 2002.

[9] Jonathan Richard Shewchuk, Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator, vol. 1148 of Lecture Notes in Computer Science, pages 203-222, Springer-Verlag, Berlin, May 1996.

[10] Li-Yi Wei, Marc Levoy, Fast Texture Synthesis using Tree-structured Vector Quantization, Proceedings of SIGGRAPH 2000.

[11] Nicolas Ray, Thibaut Neiger, Xavier Cavin, Bruno Levy, Vector Texture Maps on the GPU, Tech Report, INRIA, 2005.

[12] Pascal Barla, Simon Breslav, Jo□lle Thollot, François Sillion, Lee Markosian, Stroke Pattern Analysis and Synthesis, Computer Graphics Forum (Proc. of Eurographics 2006), Volume 25 - 2006.

[13] Remco C. Veltkamp, Shape Matching: Similarity Measures and Algorithms, Proceedings of the International Conference on Shape Modeling & Applications, 2001.

[14] Sylvain Lefebvre, Samuel Hornus, Fabrice Neyret, Texture Sprites: Texture Elements Splatted on Surfaces, ACM-SIGGRAPH Symposium on Interactive 3D Graphics (I3D) - April 2005.

[15] Vivek Kwatra , Arno Schődl , Irfan Essa , Greg Turk, Aaron Bobick, Graphcut Textures: Image and Video Synthesis Using Graph Cuts, Proc. ACM Transactions on Graphics, SIGGRAPH 2003.

첨부 1. LSM의 단순화

한 이동 변환으로부터 만들어진 대응점들의 집합 $C=(c_1, c_2, \cdots, c_n)$ 의 성분을

$$\begin{aligned} c_i &= (E_i^{ref}, E_i^{tgt}) \\ E_i^{ref} &= (x_i^{ref}, y_i^{ref}) \\ E_i^{tgt} &= (x_i^{tgt}, y_i^{tgt}) \end{aligned} \qquad (2)$$

로 정의하고, E_i^{ref} 가 이동된 결과를 $\square E_i^{ref} = (\square x_i^{ref}, \square y_i^{ref})$ 라 한다. 두 대응점 사이의 오차 e_i는 E_i^{ref} 와 E_i^{tgt} 사이의 유클리드 거리이므로, 오차의 총 합을 E라고 하면 다음이 성립한다.

$$E^2 = \sum_{i=1}^{n}((\square x_i^{ref} - x_i^{tgt})^2 + (\square y_i^{ref} - y_i^{tgt})^2) \qquad (3)$$

$\square x_i^{ref} - x_i^{tgt} = dx_i, \square y_i^{ref} - y_i^{tgt} = dy_i$ 라고 할 때, dx_i와 dy_i는 실수공간에서 정의되어 항상 0 이상의 값을 가지므로 서로 독립적인 관계임을 알 수 있다.

일반적인 선형 변환 $f(a, b)$는 $a+bx$ 의 형태로 나타낼 수 있는데, 이동 변환만을 고려할 경우 b의 값은 항상 1로 고정되므로 이 식은 $f(a)=a+x$ 로 단순화 될 수 있다.

위의 결과들을 x_i^{ref} 에 대하여 적용하면

$$\square x_i^{ref} = f(a) = a + x_i^{ref} \qquad (4)$$

$$E_x^2(a) = \sum_{i=1}^{n}(a + x_i^{ref} - x_i^{tgt})^2 \qquad (5)$$

여기서 $E_x^2(a)$를 최소로 하는 a의 값을 구하면

$$\frac{\partial E_x^2}{\partial a} = 2\sum_{i=1}^{n}(a + x_i^{ref} - x_i^{tgt}) = 0 \qquad (6)$$

$$na + \sum_{i=1}^{n}(x_i^{ref} - x_i^{tgt}) = 0 \qquad (7)$$

$$a = \frac{\sum_{i=1}^{n}(x_i^{tgt} - x_i^{ref})}{n} \qquad (8)$$

이므로, a의 값은 각 대응점이 가지는 dx들의 산술 평균임을 알 수 있다.