"코어 이미지(Core Image)"를 이용한 실시간 이미지 프로세싱에 대한 연구

A study of image processing by using "Core Image"

서준석, June Seok Seo*, 노승석, Seung Seok Noh*, 박진완, Jin Wan Park*,
서명석, Myeong Seok Seo**
*중앙대학교 첨단영상대학원 예술공학과, **홍익대학교 컴퓨터공학과

요약 컴퓨터 하드웨어 및 소프트웨어의 발전으로 그래픽 처리 기술은 날로 발전을 거듭하고 있으며 하드웨어의 기능을 최대한 이끌어내기 위한 소프트웨어 기술 또한 발전하고 있다. 이러한 발전으로 인하여 다양한 영상처리 분야에서 빠른 이미지 프로세싱이 가능하게 되었지만 빠른 이미지 프로세싱 능력에도 불구하고 프로그래밍 기술은 기존 것을 고수하고 있다. 일반적으로 사용해오던 기술은 높은 프로그래밍 지식을 필요로 하는 분야였고 이러한 이유로 이미지 프로세싱 기법을 전문적인 프로그래밍 지식이 부족한 예술계의 사용자가 이용하는 데에는 어려움이 있었다.
이러한 문제점을 해결하고 이미지 하드웨어 및 소프트웨어의 효율적인 사용 환경을 위하여 Apple사에서 OpenGL과 Objective-C를 이용한 좀더 간단한 이미지 프로세싱 기법인 코어 이미지(Core Image)를 개발하였다. OS와 어플리케이션 상에서 빠른 이미지 프로세싱을 위해 개발된 코어 이미지는 기존의 이미지 프로세싱 기법에서 복잡한 형태의 프로그래밍을 요했던 것과는 달리, 여러 이미지 프로세싱 기법을 간단한 플러그인 형태로 지원한다. 예술적인 측면에서 다양한 이미지 프로세싱 기법을 보다 손쉽게 사용할 수 있으며 사용 방법 또한 간단하여 전문적인 프로그램 지식이 부족한 일반 사용자도 예술적인 측면에서 이미지 프로세싱 기술을 쉽게 접목할 수 있도록 구성되어 있다.
하지만 코어 이미지가 국내에서는 소수의 사용자만이 이용하는 Mac OS에서만 사용 가능한 프로세싱 기술이라는 문제점 또한 지니고 있다.
본 논문에서는 코어 이미지의 개념과 구동 원리 및 실제 예술 작품에 적용된 사례를 분석하고, 이를 통하여 다양한 분야에서의 코어 이미지의 가능성에 대하여 전망해 본다.

핵심어: Core Image, Mac OS X, OpenGL, Image processing

1. 서론

지난 20년간 컴퓨터를 구성하는 하드웨어 및 소프트웨어의 발전으로 컴퓨터의 처리 성능은 빠른 발전을 거듭해 왔다. 그중 그래픽을 처리하는 CPU와 그에 관련된 그래픽 가속 하드웨어는 가장 급격한 발전을 한 부분 중 하나이다. Black & White의 단색 표현에서부터 시작한 색상 표현은 최근에는 인간의 눈으로 구별해내기 힘든 True Color까지 수많은 색상이 구현 가능하게 되었으며, 2D의 영역에서 벗어나 3D 이미지를 실시간으로 처리하는 기술까지 GPU와 그래픽 가속 하드웨어의 데이터 처리 능력은 비약적으로 발전해왔다.

그래픽 가속 하드웨어는 빠르게 발전하였고 이러한 그래픽 가속 하드웨어의 성능을 최대환 이끌어 내기 위한 소프트웨어의 기술 또한 발전을 거듭하고 있다. 하지만 하드웨어와 소프트웨어의 발전을 비교해 보았을 때 날로 증가하는 그래픽 처리 능력을 컨트롤하기 위한 그래픽 소프트웨어 개발 환경은 GPU와 그래픽 가속 하드웨어가 도입되었던 그 시기와 크게 달라지지 않은 기존의 개발 환경에서 크게 벗어나지 못하고 있고 날로 증가하는 하드웨어적 이미지 프로세싱 능력에 불구하고 단순한 수학적 수치의 하드웨어적 처리 능력의 발전만이 그래픽의 처리 능력 향상이라 볼 수 없다는 견해에서 GPU 및 그래픽 가속 하드웨어만의 발전으로는 한계에 근접했다는 것이 일반적인 평가다.

또한 하드웨어의 성능을 이용하기 위한 프로그래밍 방법으로 일반적으로 사용해 오던 기술은 높은 프로그래밍 지식을 필요로 하는 분야였고, 전문 교육을 받은 프로그래머를 제외한 일반 사용자가 이러한 프로그래밍을 하는 데에는 한계가 있기에 이미지 프로세싱 기법을 다양한 분야로 접목하는데 있어 큰 걸림돌로 존재하고 있다.

일반적으로 C계열의 언어인 C, C++, C#등의 개발 언어가 많이 사용되어져 왔고, 최근에 들어 그래픽 기능을 강화한 언어인 JAVA또한 많이 사용되어지고 있지만, 예술의 목적으로 이미지 프로세싱 기술을 사용하는데 있어 기존의 언어들을 통하여 표현하는 데는 한계가 있었고, 좀 더 다루기 쉬운 이미지 프로세싱 언어의 필요성이 일반 사용자들 사이에서

요구되어 왔다.

“코어 이미지(Core Image)” 는 이러한 사용자의 요구와 다양한 분야에 좀더 편리한 이미지 프로세싱의 접목을 목적으로 만들어진 프로그래밍 언어로서, 본 논문에서는 코어 이미지에 대한 이해와 코어 이미지를 이용한 작품의 예를 통하여 코어 이미지의 활용 방안을 전망해 보고자 한다.

2. 코어 이미지(Core Image)

2.1 코어 이미지의 정의

코어 이미지는 애플(Apple Computer, Inc.)에서 개발한 이미지 프로세싱 언어로서, GPU의 기능을 중심으로 구동되는 그래픽 하드웨어의 이미지 처리 성능을 원활하게 OS 및 어플리케이션 프로그램에 적용시키기 위하여 개발된 이미지 프로세싱 언어이다. UNIX기반의 Mac OS X 10.4에서부터 탑재되기 시작한 이 기능은, 코어 그래픽스(Core Graphics), 코어 비디오(Core Video)등의 기능과 연동성을 가지고 있다.

기존에 많이 사용되던 이미지 프로세싱 툴은 위에서 언급한 C계열 혹은 JAVA등이 있었지만 이는 대부분 DOS나 Windows의 OS를 기반으로 구동되는 프로그래밍 언어였고, 이를 제어하기 위해서는 높은 수준의 프로그래밍 지식을 필요로 해왔기 때문에 점차 높아지는 사용자의 프로그래밍 욕구를 자가 충족시키기 위해서는 프로그램에 익숙하지 못한 사용자가 자신의 목적에 접합한 프로그래밍을 하는 데에 한계가 있었다. 따라서 일반 사용자가 필요한 기능만을 좀더 간단한 방법으로 제어할 수 있는 해결책과 같은 맥락에서 전문 프로그래머 외에 일반 사용자 그룹을 위한 간단한 프로그래밍 언어로 조작할 수 있는 코어이미지를 개발하기에 이르렀다

코어 이미지의 주요 기능은 이미지 잘라내기, 색상보정, 회전, 합성 등 일반적인 이미지 프로세싱 툴과 그 맥락을 같이 하고 있으나, 코어 이미지의 주 목적은 이러한 필터링 작업을 실시간 화상과 그 외의 동영상의 리얼타임 이미지 프로세싱을 기준으로 염두에 두고 제작 되었다는데 있다. 즉, 실시간으로 캠코더 혹은 웹캠을 통해 입력받은 영상, 혹은 사전 제작되어져 있는 2D, 3D그래픽을 코어 이미지를 통한 필터링 된 이미지로 실시간 1:1 대비 출력이 가능하다는 것이 큰 장점이라 볼 수 있다.

그림 1은 코어 이미지의 구동 방식을 나타내고 있다. 그림에 나타나있는 순서대로 GPU를 기반으로 하는 그래픽 하드웨어를 기본으로, OpenGL을 기반으로 하는 코어 그래픽스, 코어 이미지, 코어 비디오가 실시간으로 이미지에 대한 프로세싱을 처리하며 그 출력은 QuickTime을 통해 이루어진다. 이는 Mac OS를 구성하는 기본 기능들의 순차적인 배치와 같은 구성으로 코어 이미지가 OS의 기본 기능으로 구성되어 있다는 것을 알 수 있다. 이는 외부에서 입력되는 신호를 필터링 하는 과정에서 OS의 기본 기능 이상을 요구로 하는 2차적인 어플리케이션 프로그램을 구동하는 과정에서 발생할 수 있는 하드웨어 및 소프트웨어적인 과부하 문제를 최소화하기 위한 방법이라 볼 수 있다.

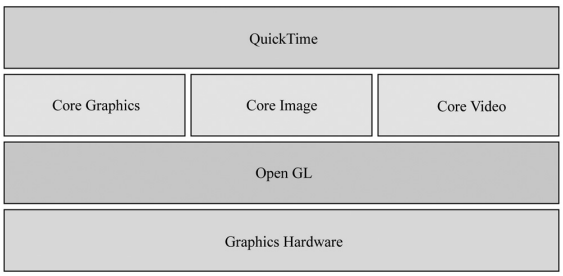

그림 1. 코어 이미지의 구동 방식

코어 이미지의 구동 기반을 이루고 있는 그래픽 처리 기술인 OpenGL은 2D와 3D 그래픽 표현에 있어 작도나 특수효과에 많은 기능들을 가지고 있으며, 그래픽 가속 하드웨어와 연동 측면에서 기술적으로 안정성이 높아 많은 어플리케이션에서 채택하여 사용 중인 기술이다. 기존의 프로그래밍 방식을 이용하여 사용자가 OpenGL 기능을 사용하고자 한다면 제작 단계에서 높은 수준의 프로그래밍에 관련된 지식을 필요로 했었지만, 공학적인 지식이 부족한 일반 사용자의 활용 측면을 고려하여 개발된 코어 이미지의 경우 Mac OS의 프로그래밍 도구인 Xcode의 Objective-C를 이용한 플러그인 형태로 구성된 몇 개의 간단한 프로그래밍 언어의 조합을 통하여 기초적인 수준의 프로그래밍으로 Quartz 2D 알고리즘을 통한 OpenGL의 기능들을 이용 가능하다.

2.2 코어 이미지의 기능

표 1. 코어 이미지 프로세싱 코어 분류

Target	Objects
Strings	NSString
Floating-point numbers	NSNumber
Vectors	CIVector
Colors	CIColor
Images	CIImage
Transforms	NSAffineTransform

코어 이미지에서 가장 주목할만한 이미지 프로세싱 기능으로 실시간 이미지 필터링을 들 수 있다. 코어 이미지에서 지원하는 이미지 프로세싱의 기본이 되는 이미지 프로세싱 코어의 분류는 표 1과 같다.

코어 이미지는 Quartz Core라는 이미지 프로세싱 코어를 중심으로 구동된다. Quartz Core는 Objective-C의 프로세싱 구동 명령을 통하여 표 1의 기능들을 구현하는 역할을 담당하고 있다. Objects의 형태로 구분되어있는 코어 이미지의 구동 명령어 중에서 실시간 화상 및 이미지에 대한 플러그인 형태의 필터링에 관련된 기능은 CI(CI = Core Image)계열의 오브젝트가 담당하게 된다. CI계열의 이미지 필터링 기능에 세분화 되어있는 각 기능은 기본적인 구동 명령어와 조합된 변수 값의 조정을 통하여 필터링의 효과 및 강약을 조절할 수 있다. 프로그래밍의 단점인 실시간으로 작업의 결과를 확인할 수 없다는 단점을 극복하기 위해 필터를 적용하는 과정에서 Core Image fun house를 이용하여 이미지를 이용한 필터 적용 시 필터의 형태 및 각 필터에 적용되는 변수 값의 예상치 적용 결과물을 시각적으로 확인할 수 있는 기능을 제공함으로서 프로그램 구동 전 결과를 예상할 수 있어 프로그래밍에 소요되는 시간을 줄여줄 수 있다.

코어 이미지의 오브젝트를 이용한 프로세싱 구동 방식은 필터 클라이언트와 필터 크리에이터로 구분할 수 있다. 필터 클라이언트는 단순하게 기존에 존재하는 표 1의 CI계열의 오브젝트 기능들에 대하여 세분화 되어있는 필터의 기능들을 플러그인 형태로 불러들여 이미지 및 영상에 적용하여 사용하는 방식이며, 필터 크리에이터는 두개 이상의 필터를 조합하여 새로운 형태의 필터링 효과를 내는 방식이다. 필터 크리에이터는 각각의 다른 OpenGL의 Shading Language를 사용하여 순차적으로 각각의 픽셀에 대한 필터의 적용 값을 계산해내어 픽셀 단위로 결과 값을 화면상에 뿌려주는 방식으로 표현된다.

이미지 프로세싱을 처리하는 과정에서 가상의 코어 이미지 연산 장치는 최종 단계의 이미지 출력 명령이 내려지기 전까지 어플리케이션의 결과를 예상하기 위해 계속적인 사전 계산 단계를 거치게 된다. 대표적 기능으로 Lazy evaluation과 같은 알고리즘은 필터가 적용된 이미지의 최종 결과 값이 출력되는 연산 과정동안 필터 크리에이터 내에 한 가지 이상의 필터가 적용 되는지를 판단하여 복수의 필터가 적용될 때 연관성이 있는 기능들의 재조합을 통하여 각각의 픽셀에 대한 필터 적용 값을 계산해내고 적절한 기능의 재조합 및 적용단계를 거쳐 최종 이미지를 얻어내는 자체 알고리즘을 가지고 있다.

그림 2는 Lazy evaluation에서 두개 이상의 복수의 필터가 하나의 이미지에 순차적으로 적용될 때 어떠한 과정을 거치게 되는지 나타내고 있다. Image1의 Color adjust 필터를 적용하여 그 결과 값이 출력되는 과정까지는 입력 당시 원본 이미지의 형태가 변형되지 않지만, Image2에서 Scale down 필터를 적용시키게 되면 초기 형태를 유지하던 이미지의 형태에 변형이 일어난다. 결국 최종 이미지 출력 단계에서는 변형된 이미지의 원활한 출력을 위해서 Thumbnail image의 형태로 최종 결과물을 재조합하여 출력하는 형태의 새로운 프로세싱 과정을 추가하게 된다.

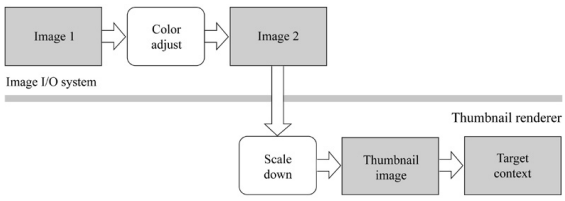

그림 2. Lazy evaluation

2.3 코어 이미지의 구동 원리

코어 이미지를 이용한 이미지 필터링의 프로그래밍을 위해서는 2.2장 표 1에서 언급한 기본적인 필터 및 원본이 되는 이미지 및 영상 소스, 그리고 그에 따라 부가적으로 추가되는 오브젝트 기능이 요구된다.

코어 이미지에서 구동되는 필터링 기능은 플러그인 형태로 제공된다. Xcode의 Objective-C상에서 load의 명령어를 통하여 불러들여진 CI계열의 필터링 플러그인 기능은 각 플러그인 object기능에 따라 설정되어있는 추가 변수 값 조정으로 미세한 필터링 효과의 조절이 가능하다. 이는 Mac OS X의 개발자 도구 상에서 Xcode의 좀더 간편한 사용을 위해 개발된 Quartz composer에서도 동일하게 적용 가능한 값으로 세밀한 필터링의 조절을 가능하게 한다.

코어 이미지를 이용한 영상 및 이미지 필터링이 진행되는 동안 행해지는 프로세싱 과정의 단편적인 예로 이미지 합성에 대한 과정을 도식화 하여 표현하면 그림 3과 같이 표현할 수 있다.

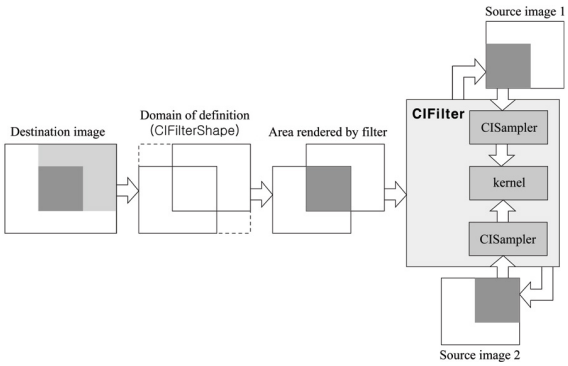

그림 3. 이미지 합성의 순서

그림 3에서 보이는 것과 같이 두개의 소스 이미지가 합성되는 과정에서 주목할 점은 이미지 영역이다. 코어 이미지는 이미지 프로세싱이 행해지는 영역에 해당하는 이미지 영역을 가지고 있다. 이미지가 프로세싱 되는 과정에서 스케일의 변형이 발생하여 사용자가 보게 되는 화면상의 이미지 영역을 100%로 보았을 때 그 이하 또는 이상의 영역에 대한 부분에 확장, 축소, 왜곡이 일어나 화면 비율 100%에 대한 이미지 영역을 벗어나거나 영역 내부로 축소되는 부분에 대해서는 그림 4와 같은 background 영역이 노출되는 현상을 보이게 된다. 이러한 현상은 background를 기준으로 프로세싱 된 이미지를 픽셀 단위로 레이어의 형태로 뿌려주는 방식을 취하는 코어 이미지의 특성이라 볼 수 있다. 그림 4는 코어 이미지의 필터링 작업의 최종 결과물에서 나타나는 이미지 왜곡 현상이 이미지 영역에 대하여 화면상에 어떻게 표현될 수 있는지를 보여주고 있다.

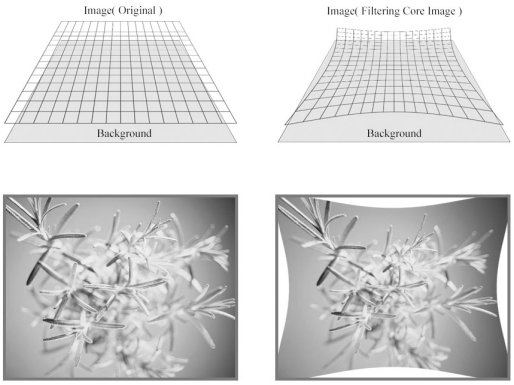

그림 4. Background에 대한 이미지 영역

2.4 코어 이미지 프로그래밍

코어 이미지의 장점은 간단한 구동 명령 체계를 이용하여 일반 사용자도 쉽게 프로그래밍이 가능하다는데 있다. 일반적으로 프로그래밍에 사용되는 선언문이나 사전 구동 명령어 없이 Objective-C상에서 플러그인 명령어의 배열을 통하여 코어 이미지의 기능들을 구동 시키는 것이 가능하다.

아래는 그림 5는 코어 이미지에서 이미지에 대한 모자이크 픽셀 모자이크를 적용하는 과정을 단편적으로 비교한 예제 으로서, 코어 이미지의 CIPixellate 필터와 C#의 Pixelate함수를 이용한 작업이다. 코어 이미지의 프로그래밍에서는 C#의 A부분이 생략되어 있는 단순한 구조의 프로그래밍 언어로 구성되어 있다.

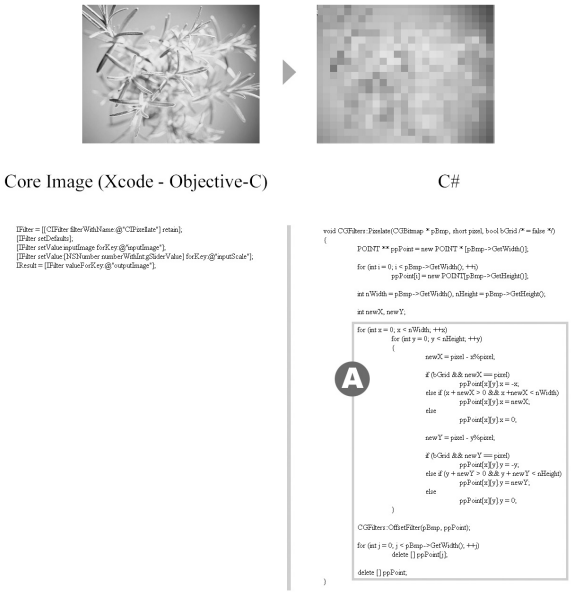

그림 5. 코어 이미지와 C#의 비교

A의 영역은 이미지상에서 순차적으로 이미지의 데이터 값을 읽어 들여 각 데이터를 이미지 프로세싱 된 값으로 치환하는 과정으로 코어 이미지 상에서는 일반 사용자의 편의를 위해 기본 명령으로 A의 기능을 내장하여 코어 이미지 필터에 대한 특별한 지식 없이도 필터 값을 불러오는 과정과 변수 값을 조정하는 두 가지의 과정만으로 원하는 기능의 구동이 가능하다. 코어 이미지의 구동 명령어는 전문적인 프로그래밍 지식이 많지 않은 일반인이 상식적으로 사용 가능한 범위 내에서 기본값을 지정한 기능이 내장되어 있는 경우가 많다는 특징을 가지고 있다.

3. 코어 이미지의 활용

3.1 Touch mirror

Touch mirror는 코어 이미지를 이용하여 제작한 작품으로, 하드웨어의 구성은 Mac mini Intel Core Duo 1.66GHz, 20.1" LCD모니터, 터치패널, 웹캠으로 구성되어 있다.

Touch mirror의 컨셉은 현실과 가상의 공간의 구별의 모호한 거울이라는 공간의 영상에 대하여 왜곡을 줌으로서 현

실과 구별되어있는 공간에 대한 인식을 유발하는 데에 있다.

구동 원리는 웹캠 으로부터 전송받은 실시간 영상을 화면상에 출력하는 동시에 화면에 부착된 터치패널을 이용하여 화면에 대하여 1:1로 매치되는 좌표에 대한 압력 신호의 입력 값을 받아 압력이 감지되는 좌표 점에 대한 화상 부분을 왜곡시킨다. 이 왜곡에 코어 이미지의 CI Object계열의 필터를 이용한다.

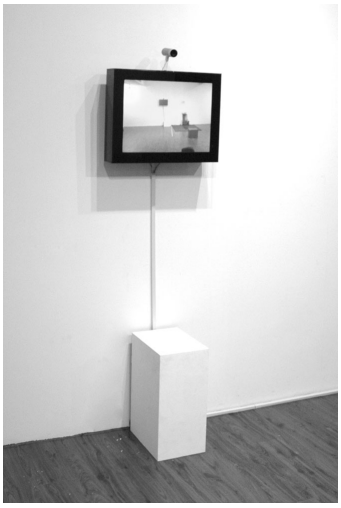

그림 6. Touch mirror' 덕원갤러리, 상외상, 2006.12, 서준석

Touch mirror에서는 CI계열의 외곡 필터 2종이 사용되었다. 첫 번째로 CIBumpDistortion 필터로 화면에 접촉하는 순간 접촉점을 중심으로 물결 모양으로 퍼져 나가는 왜곡 현상을 표현하였다. CIBumpDistortion 필터의 가능은 사용자 지정 좌표 점을 기준으로 화면의 축소/확장을 나타내는 필터로서, 좌표 점을 터치패널의 압력 입력 점으로 대체하여 압력 입력시 좌표를 중심으로 시간이 지남에 따라 순차적인 확장 영역의 범위 확대를 통해 물결무늬로 보이게 하는 착시 현상을 이용하였다.

두 번째로 이용된 필터는 CIPinchDistortion필터로, 이미지의 한 점을 중심으로 함몰된 효과를 나타내는데 적합한 필터라 할 수 있다. 압력 점을 기준으로 필터링 효과가 발생하기 때문에 사용자가 화면에 압력을 가한 상태에서 압력 좌표를 움직이게 되면 움직임에 따라 필터링 영역이 이동하게 된다.

이 두 필터의 상호작용은 최초 사용자가 화면에 압력을 가했을 때, 1회의 CIBumpDistortion 필터의 확장이 일어나게 되며, 압력을 유지하게 되면 CIPinchDistortion 필터가 압력 점을 중심으로 지속적으로 유지된다.

이는 두개 이상의 필터를 사용한 필터 크리에이터 기능의 표현으로 CIBumpDistortion 필터의 경우 화면의 왜곡이, CIPinchDistortion 필터는 화면의 축소현상이 발생하게 되어 2.3장의 그림 4와 같은 이미지 영역에 대한 왜곡이 발생하게 된다. 이를 해결하게 위해서 Lazy evaluation기능을 이용한 이미지 영역에 대하여 각 필터 적용 단계상에서 화면 비율의 재조정으로 실제 사용자가 보게 되는 화면 영역에서의 왜곡 문제를 해결하였다. 그림 7은 Touch mirror상에서 하드웨어와 코어 이미지의 데이터 처리과정을 보여주고 있다.

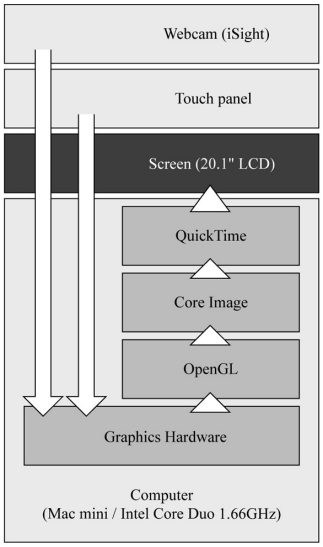

그림 7. Touch mirror의 구동 순서

Touch mirror의 프로그래밍은 간단하게 세 개의 부분으로 이루어져 있다.

① 웹캠을 통하여 입력받은 이미지에 대한 좌우 반전

② 마우스 입력 값에 대한 반응 설정

③ 코어 이미지 필터 적용

위 3개의 기능은 Xcode에서 기본으로 제공하는 기능들과 OpenGL을 이용한 코어 이미지의 필터링 기능을 불러들이는 간단한 구동 명령어를 통하여 구현 가능하다.

기본적으로 언급한 위의 3개의 기능 외에 전체화면 디스플레이, 영상 및 사운드 레코딩 등의 부가 기능이 추가되었지만, 이러한 기능 또한 Xcode에서 기본적으로 제공 되는 라이브러리를 통하여 적용 가능하다.

3.2 코어 이미지의 응용

코어 이미지의 이용 범위는 이 밖에도 다양하다.

이미지 보정 필터링 기능을 이용하여 선명도, 색상 등의 보정 기능을 이용하여 각 분야에서 다양하게 사용되고 있는 영상물의 실시간 이미지 보정에 사용될 수 있다. 코어 이미지는 최초 제작 단계부터 실시간 영상의 필터링을 지원하고자 설계된 프로세싱 툴이라는 이점을 가지고 있으므로 영상 분야의 이미지 프로세싱에 강점을 가지고 있다고 볼 수 있다. 이미지 자르기, 색상보정, 합성과 같은 이미지 프로세싱 기능 외에도 OpenGL이 가지고 있는 약 250여개의 특징적인 기능들의 응용 및 조합을 통하여 상업용, 군사용, 학술용 등의 용도로 다양한 접목이 가능할 것으로 예상된다.

또한 코어 이미지를 통해 재 가공된 영상물을 이용하여 코어 비디오의 제작이 가능하다. 코어 비디오를 이용하여 제작할 수 있는 코어 애니메이션을 통하여 영상물의 재조합 또는 새로운 영상물의 창조가 가능하며, OS 및 기타 어플리케이션에서 지원 가능한 코덱, OpenGL, DirectX, H.264등의 신기술과 접목하여 새로운 어플리케이션 개발에도 응용될 수 있다. 실제 Mac OS X 10.4버전 이후에 적용된 OS상에서 구현되는 대부분의 기본 동작을 구성하는 애니메이션 기능들은 코어 이미지를 이용하여 생성되고 코어 비디오를 통하여 구성된 결과물이다.

3.3 코어 이미지의 기능 및 제약

코어 이미지의 장점은 위에서 언급한 사항 외에 Objective-C에서는 JAVA와 유사하게 메모리를 관리하는데 있어 시스템에 큰 무리가 따르지 않는다는 점, 이미지 프로세싱 단계에서 OS와의 연동을 통하여 하드웨어의 성능을 체크하여 특정 하드웨어의 존재 유무에 따라 하드웨어 가속 기능 지원이 선택적으로 처리되기 때문에 하드웨어의 종류에 따라 코어 이미지의 기능에 제약을 두어 시스템 상에 무리가 가지 않는 선에서 이미지 프로세싱 작업을 처리하는 점, 그리고 여러 시스템 요소들을 직접적으로 다룰 수 있다는 점 등을 들 수 있다.

하지만 코어 이미지의 가장 큰 문제점은 Mac OS에서만 구동 가능하다는 점이다. 이는 대다수의 컴퓨터 사용자가 윈도우즈 계열의 OS를 사용하고 있다는데 그 문제가 발생한다.

단순히 코어 이미지만을 사용하기 위하여 하드웨어에 대한 지출 및 새로운 OS와 개발 언어에 적응해야 한다는 간단하면서도 치명적인 약점을 가지고 있다.

또한 이미지 프로세싱 전 단계에서 하드웨어의 성능을 체크하여 그 수준에 맞추어 코어 이미지 구동의 유무를 결정하기 때문에 시스템을 구성하는 하드웨어의 성능에 따라 일부 기능에 대한 제약이 발생할 수 있으며, Mac OS X 10.4버전 이전의 Mac OS에서는 코어 이미지를 구동이 불가능하다는 문제점을 가지고 있다. 최근 Apple사에서 CPU를 PPC(PowerPC)에서 Intel계열의 듀얼 코어 프로세서로 교체하는 과정에서 PPC상에서 구동되는 Mac OS X 10.4와 Intel계열의 CPU에서 구동되는 Mac OS X 10.4상의 코어 이미지의 구동 성능 간에도 큰 차이를 보이고 있어 코어 이미지가 어느 수준 이상의 하드웨어 성능을 가진 시스템 상에서는 이미지 프로세싱을 하는데 적합한 도구라 할 수 있지만 그 이하의 성능을 지닌 시스템에서는 만족할만한 결과물을 얻는 데는 무리가 있다고 볼 수 있다.

예술의 목적으로 사용되는 데는 간단한 기능의 접목으로도 만족할만한 결과물을 얻을 수 있지만 좀더 전문적인 이미지 프로세싱 기능을 접목하기 위해서는 어느 수준 이상의 프로그래밍 지식을 필요로 한다. 이를 해결하기 위해 Quartz composer, Interface builder와 같은 프로그램 개발 툴이 포함되어 있지만 이 또한 기존 개발자들이 사용하던 툴과는 다른 새로운 개발 환경을 가지고 있어 새로운 언어에 대한 습득이 문제가 된다.

3. 결론

코어 이미지를 통한 예술 분야에서 응용 사례는 3장에서 소개되었던 Touch mirror와 유사한 미디어, 인터렉티브 작품 등에 응용될 수 있다. 코어 이미지뿐만 아니라 Quartz Core를 이용한 코어 이미지, 코어 비디오 등의 기술이 다른 다양한 분야에 접목 가능하다.

코어 시리즈의 최대 장점은 '단순함' 에 있다. 프로그래머만을 위한 프로그래머의 언어가 아닌, 일반 사용자를 염두에 두고 만들어 졌지만 제작 단계에서부터 높은 안정성을 지닌 OpenGL, DirectX등의 기술을 쉽게 사용 가능하도록 염두에 두고 만들어진 이미지 프로세싱 언어이다.

최근 예술의 경향중 하나로, 공학 기술과 예술의 접목을 목적으로 하는 움직임이 일어나고 있다. 이러한 움직임에서 예술가의 입장에서 공학 기술의 접목을 막는 가장 큰 부분은 예술가가 공학 기술을 이용하는 데에 요구로 하는 전문 지식 습득이 어렵다는데 있다. 이러한 문제에 대한 어느 정도의 해결책으로 코어 이미지의 사용을 언급할 수 있을 것이다.

일반적으로 지식의 습득 방법과 경로가 다른 예술과 공학이라는 학문의 특성상 예술가의 눈으로 바라보는 공학 기술은 알 수 없는 기호들의 집합체와 같을 것이라는 전제 하에서 코어 이미지의 사용 방법은 플러그인 형태의 간단한 구동 방식과 세부적인 변수 값에 대한 손쉬운 조정방법, 프로그래밍 결과에 대하여 실시간으로 예상할 수 있는 각종 어플리케이션을 통하여 좀 더 나은 개발 환경을 제공할 수 있을 것이다.

참고문헌

[1] Core Image Programming Guide, Apple Computer, Inc. Developer connection, California, USA, pp. 7~28, 2006

[2] Core Image Reference Collection, Apple Computer, Inc. Developer connection, California, USA, pp. 15~83, 103~205, 2006

[3] Quartz 2D Programming Guide, Apple Computer, Inc. Developer connection, California, USA, pp.19~49, 2006

[4] "그래픽스 카드의 새로운 용도" http://challenge.ewha.ac.kr/main/no8/challenge8_71.asp

[5] "Core Image" http://www.apple.co.kr/macosx/features/coreimage/

[6] "OpenGL" http://www.apple.co.kr/macosx/features/opengl/

[7] "OpenGL" http://en.wikipedia.org/wiki/OpenGL