Journal of Korea Multimedia Society Vol. 16, No. 10, October 2013(pp. 1189-1195)
http://dx.doi.org/10.9717/kmms.2013.16.10.1189

WebCL 기반 초고해상도 이미지 처리 기술

조명진[†], 한영선[††]

요 약

본 논문에서는 웹 환경에서 WebCL을 이용한 초고해상도 이미지 처리 기법의 성능을 분석하고자 한다. WebCL로 인한 성능의 변화를 측정하고 평가하기 위해 자바스크립트로 작성된 대표적인 이미지 처리 라이브러리인 Pixastic 라이브러리를 WebCL 기반의 코드로 수정하였다. WebCL 기반 라이브러리는 8K Ultra HD의 이미지에서 기존 라이브러리 대비 최대 4.2배의 성능 향상을 얻을 수 있었으며 평균적으로 2.8배의 성능 향상을 얻을 수 있었다.

WebCL-based Very High Resolution Image Processing Technology

Myeongjin Cho[†], Youngsun Han[††]

ABSTRACT

In this paper, we are going to describe the performance characteristic of very high resolution image processing with WebCL on the web environment. In order to evaluate the variance of the execution time by WebCL, we modified the Pixastic library, one of the most representative image processing libraries written in JavaScript, by using WebCL. We achieved a speedup of up to 4.2 times and 2.8 times on average against the original one for the image of 8K Ultra HD with the WebCL-based library.

Key words: WebCL, JavaScript(자바스크립트), Image Processing(이미지 처리), GPU

1. 서 론

스마트폰, 태블릿 PC, 스마트 TV 등 다양한 하드웨어 장치들이 새롭게 보급되고 이를 지원하기 위한 여러 종류의 소프트웨어 플랫폼(Software Platform)이 개발되면서 사용자 어플리케이션(User Application)의 플랫폼 간 호환성 문제가 발생하였다. 최근에는 이와 같은 호환성 문제 해결을 위해 Java, Ada 등의 플랫폼 독립적인 프로그래밍 환경을 이용한 해결책과 더불어 HTML5 표준이 새롭게 대두되고 있다[1-3]. HTML5 표준은 기존 HTML 표준에 캔버스(Canvas), 웹 워커(Web Worker), 웹 소켓(Web Socket) 등의 다양한 기능이 추가된 형태로 하나의 HTML5 기반 어플리케이션을 HTML5 표준을 지원하는 모든 웹 브라우저(Web Browser) 상에서 동일하게 동작시킬 수 있다는 점에서 최근 크게 각광받고 있다. 그러나 이와 같은 장점에도 불구하고 HTML5 기반 어플리케이션의 동적 처리를 담당하는 자바스크립트(JavaScript)의 낮은 성능으로 인해 HTML5의 활성화가 제한되고 있다[4,5]. 이와 더불어 최근 웹 어플리케이션(Web Application)은 대용량 미디어 및 그래픽스(Graphics) 등 고성능의 처리 능력을 요구하기 때문에 자바스크립트를 가속하여 성능 문제를 해결하려는 다양한 연구가 진행되고 있다

※ 교신저자(Corresponding Author) : 한영선, 주소 : 경상북도 경산시 하양읍 가마실길 50 경일대학교 7호관 209-1호(712-701), 전화 : 053) 600-5549, FAX : 053) 600-5559, E-mail : youngsun@kiu.ac.kr
접수일 : 2013년 8월 7일, 수정일 : 2013년 9월 27일
완료일 : 2013년 9월 28일

[†] 정회원, 고려대학교 전기전자공학연구소
(E-mail : linux@korea.ac.kr)
[††] 정회원, 경일대학교 전자공학과
※ 본 논문은 2012년도 경일대학교 교내일반연구비의 지원에 의하여 수행되었음

[6,7,8,9]. 특히 Ultra HD 급의 초고화질 TV의 등장으로 Full HD 혹은 Ultra HD급의 초고해상도 이미지 처리 능력이 요구됨에 따라 자바스크립트 성능의 중요성이 점점 커지고 있다.

자바스크립트는 기본적으로 병렬처리가 불가능하며 순차적으로 수행되는 특징을 갖고 있다. 최근 이런 문제점을 해결하고자 웹 어플리케이션을 병렬화하는 다양한 연구들이 진행되고 있다. 웹 워커는 HTML5 표준에서 제공하는 기술로 자바스크립트 코드를 병렬적으로 처리할 수 있게 해준다[10]. 그러나 웹 워커는 비교적으로 무겁기 때문에 동시에 많은 작업을 수행하기에는 부적합하다. 특히 데이터를 작은 단위로 잘게 나누어 처리해야하는 이미지 처리와 같은 용도로는 부적합한 특성을 갖는다. 병렬 자바스크립트로 불리는 인텔의 River Trail은 멀티코어 프로세서(Mutli-core Processor) 및 GPU 등 이기종 환경에서 웹 어플리케이션을 병렬처리하는 Parallel-Array를 제공한다[11,12]. ParallelArray는 자바스크립트의 기존 배열과 달리 Map, Reduce, Scatter 등의 함수를 지원하며 OpenCL을 이용해 자바스크립트를 병렬처리할 수 있게 한다. WebCL은 River Trail과 같이 웹 환경에서 자바스크립트를 통해 OpenCL을 사용하도록 지원하는 기술이다[13]. 하지만, WebCL은 River Trail과 달리 직접적으로 OpenCL 커널을 사용하기 때문에 별도로 작성된 코드를 OpenCL 코드로 변환하는 과정이 불필요하다. 또한 OpenCL API와 유사한 WebCL API를 제공하기 때문에 직접적인 제어가 가능한 효율적인 코드를 작성할 수 있다.

본 논문에서는 자바스크립트로 작성된 대표적인 이미지 처리 라이브러리(Image Processing Library)인 Pixastic를 확장하여 이기종 멀티코어 환경에서 WebCL을 통한 가속이 가능한 이미지 처리 라이브러리를 구현하였다. 또, WebCL을 이용한 이미지 처리의 유용성을 확인하기 위해 다양한 해상도의 이미지를 사용하여 원본 Pixastic 라이브러리와 WebCL 기반 Pixastic 라이브러리의 성능을 분석하였다. 그 결과 고해상도 이미지에서 WebCL 기반 이미지 처리 기법이 기존 자바스크립트 이미지 처리 기법에 비해 큰 성능 향상이 있음을 확인할 수 있었다. 특히 8K Ultra HD 해상도 이미지에서는 WebCL 기반 이미지 처리 라이브러리를 통해 최대 4.2배의 성능 향상을 얻을 수 있었다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 가장 대표적인 자바스크립트 이미지처리 라이브러리인 Pixastic에 대해 소개하고 이를 확장한 WebCL 기반의 이미지 처리 라이브러리의 구조에 대해 설명한다. 3장에서는 WebCL 기반 이미지 처리 라이브러리의 효용성을 검증하기 위한 실험과 그 결과를 설명하며 마지막으로, 4장에서는 본 논문의 결론을 맺고자 한다.

2. WebCL을 이용한 웹기반 이미지 처리

2.1 Pixastic 라이브러리

Pixastic은 가장 대표적인 이미지 처리 라이브러리 중 하나이다[14]. Pixastic 라이브러리는 자바스크립트로 작성되어 있으며 HTML5 캔버스를 사용하여 다양한 이미지 효과를 줄 수 있도록고안되어 있다. 대표적으로 흑백 효과(Desaturate), 선명 효과(Sharpen), 흐림 효과(Blur), 테두리 검출(Edge Detection)과 같은 총 28가지 영상 처리 필터를 제공하고 있다. 또한 Pixastic 라이브러리는 학술 연구에서 자바스크립트 엔진의 성능을 평가하기 위한 벤치마크로도 사용되고 있다[15]. 본 연구에서는 WebCL의 성능 특성을 분석하기 위해 알고리즘의 복잡도를 고려하지 않고 가장 일반적이고 대표적인 8가지의 알고리즘을 선택하였고 선택된 알고리즘은 표 1에 상세히 설명하고 있다.

표 1. Pixastic 라이브러리의 대표적인 알고리즘

알고리즘	설 명
Desaturate	이미지를 회색으로 변환하는 필터.
Emboss	이미지를 양각화하는 필터.
Bright	이미지의 밝기 및 대비를 조정하는 필터.
Blur	이미지를 흐리게 변환하는 필터.
Sharp	이미지를 선명하게 변환하는 필터.
Edge	이미지의 테두리를 검출하는 간단한 필터.
Laplace	이미지의 테두리를 검출하는 필터.
Posterize	이미지의 색상의 수를 줄이는 필터.

2.2 WebCL 기반 Pixastic 라이브러리

일반적으로 WebCL 프로그램은 OpenCL과 호환이 가능한 커널(Kernel) 코드와 플랫폼 및 WebCL을

실행할 수 있는 호환 장치를 탐색하고 컨텍스트(Context)를 생성하며 호환 장치에 데이터를 쓰거나 그로부터 처리된 결과를 읽어오는 등 WebCL API를 사용할 수 있는 호스트(Host) 코드로 구분되어 있다. 커널 코드는 OpenCL C 표준으로 작성되며 호스트 코드는 자바스크립트로 작성된다.

그림 1은 WebCL을 지원하도록 수정된 Pixastic 라이브러리의 기본적인 알고리즘 구조를 표현하고 있다. WebCL 기반 라이브러리의 각 알고리즘 처리 과정은 크게 시작(Start), 초기화(Initialization), 이미지 로딩(Image Loading), 커널 실행(Kernel Execution), 마무리(Finish)와 같이 총 5 단계로 구분 지을 수 있다. 여기서 시작, 이미지 로딩 및 마무리 작업은 기존 Pixastic 라이브러리와 완벽히 동일하게 구현하였다. 그 외의 초기화 및 커널 실행 단계는 WebCL 지원을 위해 새롭게 구현하였다. 우선 시작 단계가 완료된 후 초기화 단계에서 커널 코드 실행을 위한 컨텍스트, 프로그램, 커널, 명령어큐(Command Queue) 등과 같은 WebCL 객체가 생성된다. 두 번째로 이미지의 미가공 픽셀 데이터(Raw Pixel Data)를 HTML5 캔버스 위에 로드한다. 이미지 로드 단계는 앞서 설명한바와 같이 기존 Pixastic 라이브러리와 동일하게 구현되었다. 세 번째로 커널 실행 단계에서는 GPU로 데이터를 쓰고 또한 데이터를 읽기 위해 버퍼 객체를 생성한다. 또한 GPU에 로딩된 픽셀 데이터를 전송한 후에 커널 코드는 실행된다. 마지막으로 커널 코드가 실행 완료되고 그 처리 결과를 GPU로부터 읽고 난 후 모든 처리 과정은 종료된다.

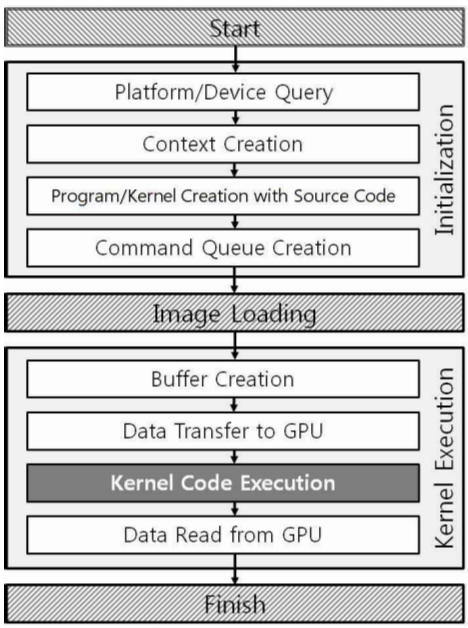


그림 1. WebCL 기반 Pixastic 라이브러리의 구조

각 필터는 수작업을 통해 원래의 알고리즘을 그대로 유지하도록 OpenCL 커널 코드로 재작성 되었다. 이는 재작성된 커널 코드가 기존의 알고리즘과 동일한 기능을 제공하는 것을 의미한다. 그림 2(a)는 Posterize 알고리즘의 자바스크립트로 구현한 기존 Pixastic 라이브러리의 코드를 보여주며 그림 2(b)는 해당 코드를 OpenCL 커널 코드로 변환한 결과이다. 이 예제는 WebCL 기반 Pixastic 라이브러리의 커널 코드가 어떻게 작성되었는지를 보여준다. 그림 2(a)에서 점선으로 표현된 자바스크립트 코드는 이중 루프로 구성되어 있으며 로딩된 이미지의 각 픽셀을 순차적으로 하나씩 처리하고 있다. 그림 2(b)의 커널 코드는 그림 2(a)의 안쪽 루프(Inner Loop)를 바탕으로 작성되었다. 커널 코드는 일반적으로 수백 또는 수천개의 프로세싱 유닛이 병렬적으로 각 픽셀을 처리하기 때문에 자바스크립트 코드처럼 이중 루프 구조를 가질 필요가 없다. get_global_id 함수를 사용하여 x, y 좌표를 획득하고 해당 좌표의 픽셀을 각 알고리즘으로 수행하는 구조로 이루어져있다. 이처럼 Posterize 알고리즘 외의 나머지 7개의 알고리즘도 원래의 알고리즘을 그대로 유지하면서 이기종 멀티코어 환경에서 병렬처리를 할 수 있도록 커널 코드를 생성하였다.

3. 성능 분석

3.1 실험 환경

자바스크립트로 작성된 기존의 Pixastic 라이브러리와 WebCL 기반 Pixastic 라이브러리의 성능을 비교하기 위하여 총 10가지의 다른 해상도를 가진 이미지를 실험에서 사용하였다. 그림 3은 이미지 해상도에 따라 픽셀의 수가 얼마나 늘어나는지를 보여주고 있으며 최근 Ultra HD TV의 등장에 따라 저해상도 이미지를 비롯하여 4K Ultra HD 및 8K Ultra HD까지 실험을 수행하였다. 또한 8가지의 대표적인 이미지 필터를 Pixastic 라이브러리에서 선택하여 그 성능을 분석하였다.

그림 4는 WebCL 실행 환경의 전체 구조를 보여준다. 웹브라우저에서 WebCL을 사용하기 위해서는

```
var rect = params.options.rect;
var w = rect.width;
var h = rect.height;
var w4 = w*4;
var y = h;

do {
    var offsetY = (y-1)*w4;
    var x = w;

    do {
        var offset = offsetY + (x-1)*4;

        var r = numValues * ((data[offset] / numAreas)>>0);
        var g = numValues * ((data[offset+1] / numAreas)>>0);
        var b = numValues * ((data[offset+2] / numAreas)>>0);

        if (r > 255) r = 255;
        if (g > 255) g = 255;
        if (b > 255) b = 255;

        data[offset] = r;
        data[offset+1] = g;
        data[offset+2] = b;

    } while (--x);
} while (--y);
```

(a) JavaScript Code

```
__kernel void clPosterize(
    __global const uchar4* inPix,
    __global uchar4* outPix,
    uint width,
    uint height,
    uint numValues,
    uint numAreas) {

    int x = get_global_id(0);
    int y = get_global_id(1);

    if(x >= width || y >= height) return;

    int i = y * width + x;
    uchar4 pix = inPix[i];

    uint r = numValues * (pix.x / numAreas);
    uint g = numValues * (pix.y / numAreas);
    uint b = numValues * (pix.z / numAreas);

    if (r > 255) r = 255;
    if (g > 255) g = 255;
    if (b > 255) b = 255;

    outPix[i].x = convert_uchar(r);
    outPix[i].y = convert_uchar(g);
    outPix[i].z = convert_uchar(b);
    outPix[i].w = inPix[i].w;
}
```

(b) Kernel Code

그림 2. Posterize 알고리즘의 자바스크립트 및 대응하는 WebCL 커널 코드

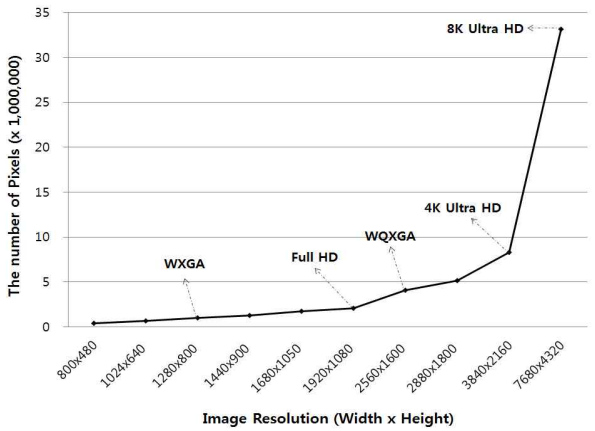


그림 3. 이미지 해상도에 따른 픽셀 수

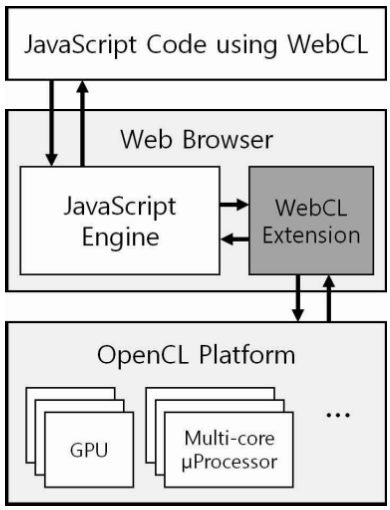


그림 4. WebCL 실행 구조

두 가지 환경이 필요하다. 우선 OpenCL을 지원하는 플랫폼이 있어야 하며 웹브라우저에서 WebCL을 지원하여야 한다. 그러나 현재 WebCL은 표준화가 진행중이며 삼성과 노키아를 비롯한 소수의 회사를 중심으로 프로토타입(Prototype)이 개발되고 있다[16, 17]. 삼성 프로토타입의 경우 Webkit의 특별 버전으로 제공하고 있으며 노키아 프로토타입의 경우 파이어폭스의 플러그인(Plug-in) 형태로 구현되어 있다.

본 연구에서는 노키아의 WebCL 프로토타입을 사용하여 구현하였으며 이에 파이어폭스 15.0.1 버전과 WebCL 1.0.3-r66 플러그인이 실험에 사용되었다. 또한 인텔 2.3GHz i7-i3615QM CPU와 16GB 메모리 그리고 NVIDIA GeForce GT 650M 그래픽 칩셋이 사용되었다[18].

3.2 성능 평가

그림 5는 10가지의 다른 해상도 이미지에 대해 기존 Pixastic 라이브러리의 실행 시간을 보여주고 있다. 그림에서 볼 수 있듯이 각 이미지 필터 알고리즘의 복잡도와 해상도에 따라 변화하는 실행 시간 결과를 보여주고 있다. Laplace의 경우 Desaturate에 비해 대부분의 해상도에서 약 4배정도의 시간을 소요하고 있다. 또한 모든 이미지 필터에서 이미지 해상도의 증가에 따라 실행 시간이 크게 늘어나고 있음을 알 수 있다. 뿐만 아니라 이 그래프의 추세는 앞선 그림 3의 기울기와 유사한 것을 알 수 있다. 이는 픽셀 수만큼 순차적으로 알고리즘을 수행해야하는 자바스크립트의 특성 때문이다.

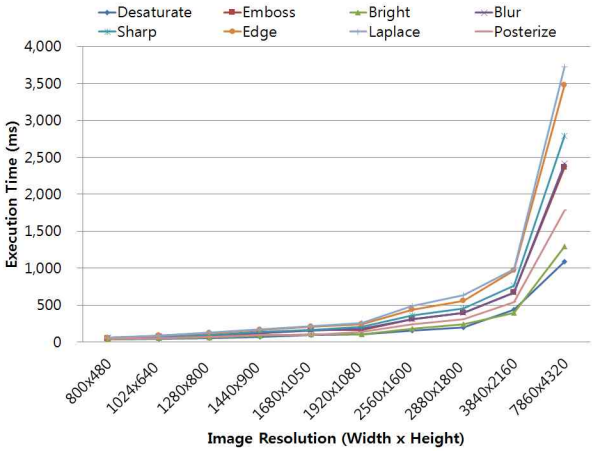


그림 5. Pixastic 라이브러리의 실행 시간

기존의 Pixastic 라이브러리와 달리 WebCL을 지원하는 Pixastic 라이브러리의 경우 이미지 해상도에 따라 실행 시간이 늘어나지만 모든 알고리즘이 거의 동일한 실행 시간을 갖는 것을 볼 수 있다. 비록 약간의 차이가 있으나 무시할 수 있을 만큼 그래프가 마치 하나의 선을 그리고 있다. 이는 기존에 순차적으로 수행되던 것을 GPU를 기반으로 병렬적으로 수행하고 있기 때문이다. 또한 커널이 수백 개의 쓰레드에서 수행되어 커널 실행 시간이 전체 실행시간에 비해서 극히 적으므로 초기화 및 이미지 로딩과 같은 시간이 전체 실행 시간을 결정짓기 때문이다. 커널 수행 시간을 제외한 다른 부분의 경우 알고리즘이 달라도 거의 동일한 시간을 소모하기 때문에 그림 5와 같은 그래프가 나타나게 된다.

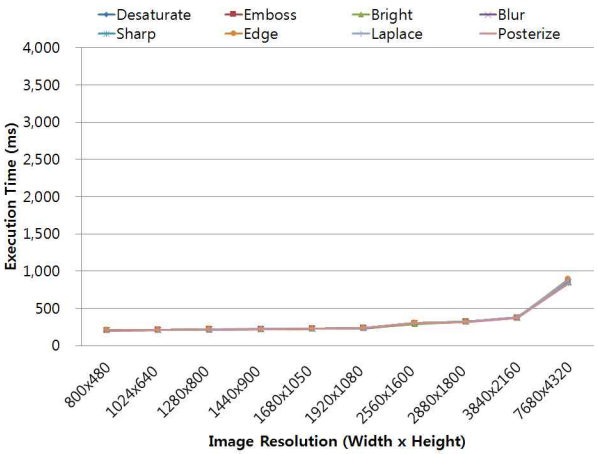


그림 6. WebCL 지원 Pixastic 라이브러리의 실행 시간

그림 5와 6을 비교하였을 때 7680x4320과 같은 8K UHD 이미지에서는 기존 Pixastic 라이브러리에 비해 WebCL 지원 라이브러리가 획기적인 성능 향상을 보이고 있다. 수정된 라이브러리의 경우 최대 약 4.2배의 성능향상을 이루었으며 평균적으로 2.8배의 성능 향상을 얻을 수 있었다. 그러나 저해상도 이미지에서는 WebCL을 사용한 버전이 오히려 성능 저하가 있음을 알 수 있다. 그 원인을 분석하기 위해서 상세한 실행 시간 분포를 그림 7과 같이 측정하였다. 그림 7은 WebCL 지원 라이브러리에서 Laplace의 실행시간의 분포를 보여준다. 실행시간을 초기화, 이미지 로딩, 커널 실행, 그리고 나머지 작업으로 분류하였다. 초기화에서는 커널 코드 실행을 위한 컨텍스트, 커널, 명령어큐 등을 생성하는데 필요한 시간이다. 이미지 로딩은 getImageData 함수를 이용하여

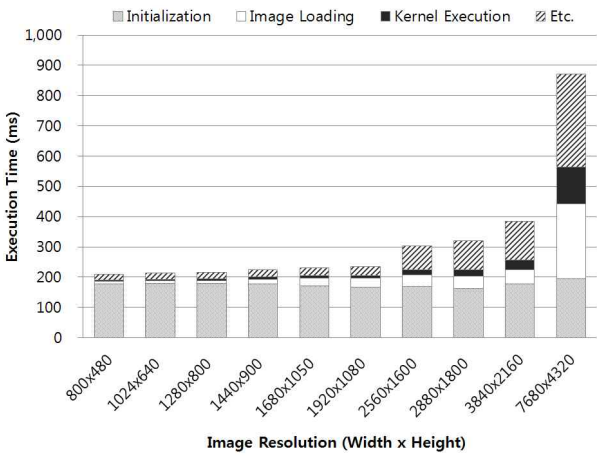


그림 7. WebCL 지원 Laplace 알고리즘의 실행 시간 분포도

미가공 픽셀 데이터를 HTML5 캔버스에 로드하는데 소요되며 각 픽셀은 빨강(R), 녹색(G), 파랑(B) 그리고 투명도(Alpha) 값으로 구성되어 있다. 기본

적으로 커널 실행은 WebCL 커널을 수행하는 시간을 의미한다. 그러나 여기서는 이미지 데이터의 GPU 전송 및 그 결과를 읽어오는 시간을 커널 실행시간에 포함하였다. 마지막으로 전체 실행시간에서 언급한 3가지 시간을 제외한 시간을 나머지 시간으로 표현하였다. 나머지 시간은 2.2장에서 언급한 시작, 마무리 단계 등이 포함된다. 그림 7에서 보이듯이 모든 이미지 해상도에서 기본적으로 초기화작업을 위해 최소 160ms의 시간을 소모함을 알 수 있다. WebCL 커널을 실행하기 위한 초기화작업 및 커널의 컴파일 등의 작업으로 인해서 기본적인 시간을 많이 소모하여 낮은 해상도의 이미지에서는 오히려 성능 저하가 발생하고 있다. 그러나 WebCL의 경우 반복적으로 수행되기 때문에 컴파일된 커널 및 WebCL 객체를 재사용할 경우 보다 큰 성능 향상을 기대해 볼 수 있다.

4. 결 론

본 논문에서는 웹 환경에서 초고해상도 이미지 처리에 대한 문제점을 분석하고 이를 극복하기 위해 자바스크립트로 작성된 대표적인 이미지 처리 라이브러리인 Pixastic 라이브러리를 확장하여 이기종 멀티코어 환경에서 병렬처리가 가능한 WebCL 기반 이미지 처리 라이브러리를 소개하였다. WebCL 지원 라이브러리는 7680×4320과 같은 초고해상도 이미지의 경우 모든 이미지 필터에 대해서 기존 Pixastic 라이브러리 대비 최대 4.2배, 평균 2.8배의 보다 나은 성능을 보이고 있다. WebCL 기반 이미지 처리 라이브러리는 플랫폼에 독립적이기 때문에 앞으로 HTML5의 표준화와 더불어 PC 환경에서 뿐만 아니라 Ultra HD TV와 같은 초고해상도 TV 및 Google TV와 같은 스마트TV, 고해상도 모바일 장치 등 다양한 플랫폼에서 유용하게 사용 가능할 것이다. 추후 WebCL 객체를 재사용하고 성능 제약으로 작용하는 초기화 시간을 크게 줄이는 등 보다 큰 성능 향상을 위한 후속 연구를 지속적으로 진행할 예정이다.

참 고 문 헌

[1] Hickson, Ian, and David Hyatt. *HTML5: A Vocabulary and Associated APIs for HTML and XHTML,* World Wide Web Consortium, W3C Working Draft Edition, 2011.

[2] 한성민, 손윤식, 이양선. “스마트 크로스 플랫폼을 위한 스마트 가상기계의 설계 및 구현,” 멀티미디어학회논문지, 제16권, 제2호, pp. 190-197, 2013.

[3] Kessin and Zachary. *Programming HTML5 Applications: Building Powerful Cross-Platform Environments in JavaScript*, O'Reilly Media Inc., Sebastopol, CA, 2011.

[4] G. Richards, S. Lebresne, B. Burg, and J. Vitek, “An Analysis of The Dynamic Behavior of JavaScript Programs,” *Proc. of 2010 ACM SIGPLAN Conf. Programming Language Design and Impl.*, pp. 1-12, 2010.

[5] E. Fortuna, O. Anderson, L. Ceze, and S. Eggers, “A Limit Study of JavaScript Parallelism,” *IEEE Int. Symp. on Workload Characterization*, pp. 1-10, 2010.

[6] P. Joshi, M. Bourges-Sevenier, K. Russell, and Z. Mo, “Graphics Programming for the Web,” *ACM SIGGRAPH 2012 Courses*, pp. 8:1-8:75, 2012.

[7] M.D. Benedetto, F. Ponchio, F. Ganovelli, and R. Scopigno, “Spidergl: A JavaScript 3D Graphics Library for Next-Generation WWW,” *Proc. of the 15th Int. Conf. on Web 3D Tech.*, pp. 165-174, 2010.

[8] J.K. Martinsen and H. Grahn, “An Alternative Optimization Technique for JavaScript Engines,” *Proc. of the Third Swedish Workshop on Multicore Computing*, pp. 155-160, 2010.

[9] M. Mehrara, H. Po-Chun, M. Samadi, and S. Mahlke, “Dynamic Parallelization of JavaScript Applications using an Ultra-Lightweight Speculation Mechanism,” *IEEE 17th Int. Symp. on High Performance Computer Architecture*, pp. 87-98, 2011.

[10] I. Hickson, *Web Workers*, W3C Working Draft 01, http://dev.w3.org/html5/workers/, 2011.

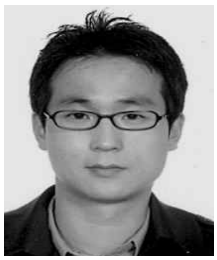
[11] S. Herhut, R.L. Hudson, T. Shpeisman, and J. Sreeram, “Parallel Programming for the

Web," *Proc. of 4th USENIX Conf. Hot Topics in Parallelism*, pp. 1-6, 2012.
[12] J. Sreeram, S. Herhut, R.L. Hudson, and T. Shpeisman, "Teaching Parallelism with River trail," *Proc. of 2012 Workshop on Developing Competency in Parallelism: Techniques for Education and Training*, pp. 1-8, 2012.
[13] Khronous Group, WebCL-Heterogeneous Parallel Computing in HTML5 Web Browsers, http://www.khronos.org/webcl/, 2011.
[14] Pixastic JavaScript Image Processing Library, http://www.pixastic.com/, 2011.
[15] M. Mehrara and S. Mahlke, "Dynamically Accelerating Client-side Web Applications through Decoupled Execution," *Proc. of 9th IEEE/ACM Int. Symp. Code Generation and Optimization*, pp. 74-84, 2011.
[16] Nokia Research, Nokia WebCL Extension for Firefox, http://webcl.nokiaresearch.com/, 2011.
[17] W. Jeon, T. Brutch, and S. Gibbs, "Webcl for Hardware-Accelerated Web Applications," *Proc. of 2012 World Wide Web*, 2012.
[18] Nvidia Corporation, Geforce GT 650m, http://www.geforce.com/hardware/notebook-gpus/geforce-gt-650m/, 2012.

조 명 진

2006년 2월 고려대학교 전기전자전파공학부 공학사
2008년 2월 고려대학교 전자전기공학과 공학석사
2013년 2월 고려대학교 전자전기공학과 공학박사

2013년~현재 고려대학교 전기전자공학연구소 연구교수
관심분야 : 성능 평가, 안드로이드 최적화, 컴퓨터 구조, 병렬처리



한 영 선

2003년 2월 고려대학교 전기전자전파공학부 공학사
2009년 2월 고려대학교 전자컴퓨터공학과 공학박사
2009년~2010년 삼성전자 시스템 LSI 책임 연구원

2011년~현재 경일대학교 전자공학과 조교수
관심분야 : 컴파일러, 임베디드 시스템, 컴퓨터 구조