Journal of Digital Contents Society Vol. 14 No. 1 Mar. 2013(pp. 7-14)
http://dx.doi.org/10.9728/dcs.2013.14.1.7

Android 플랫폼에서 구현 기술에 따른 화질 개선 연산 성능 비교

이주호*, 이구연**, 정충교***

요 약

고사양 디지털 카메라를 내장한 스마트 기기의 보급이 급격하게 증가하면서 고해상도 영상의 품질을 개선하는 기능이 중요해지고 있다. 모바일 기기의 제한된 자원과 영상의 정보량을 감안하면 지나치게 복잡하지 않은 알고리즘을 선택해야 하고 가능한 효율적인 구현 기술을 사용해야 한다. 영상의 질을 향상시키는 기법 중 간단하면서도 널리 사용되는 기술은 명암 값 분포를 조절하는 명암비신장 (contrast stretching) 기법이다. 안드로이드 스마트폰의 경우, 구현 기술로는 가상 머신 위에서 동작하는 자바 라이브러리, JNI (Java Native Interface) 기반 원시 (native) C/C++ 구현 기술, 그리고 SIMD (Single Instruction Multiple Data) 기법이 적용된 NEON 구현 기술 등이 있다. 이 연구에서는 이 세 가지 기술을 선형 명암비신장 연산, 그리고 평활화 명암비신장 연산에 적용하여 그 성능을 실증적으로 비교 분석하였다. 그 결과 원시 C/C++과 NEON 라이브러리의 실행 속도는 자바 라이브러리 실행 속도에 비해 각각 56-78배, 50-76배 빠르다는 점을 확인하였다.

키워드 : 안드로이드, JNI, 네온, 선형 명암 비신장, 평활화 명암비신장

Performance Comparison of Implementation Technologies for Image Quality Enhancement Operations on Android Platforms

Ju-Ho Lee*, Goo-Yeon Lee**, Choong-Kyo Jeong***

Abstract

As mobiles devices with high-spec camera built in are used widely, the visual quality enhancement of the high-resolution images turns out to be one of the key capabilities of the mobile devices. Due to the limited computational resources of the mobile devices and the size of the high-resolution images, we should choose an image processing algorithm not too complex and use an efficient implementation technology. One of the simple and widely used image quality enhancement algorithms is contrast stretching. Java libraries running on a virtual machine, JNI (Java Native Interface) based native C/C++, and NEONTM SIMD (Single Instruction Multiple Data) are common implementation technologies in the case of Android smartphones. Using these three implementation technologies, we have implemented two image contrast stretching algorithms - linear and equalized, and compared the computation times. The native C/C++ and the NEONTM SIMD outperformed the native C/C++ implementation by 56-78 and 50-76 time faster respectively.

Keywords : Android, JNI, NEON, Linear Contrast Stretching, Equalized Contrast Stretching

1. 서론

최근 고사양의 디지털 카메라가 스마트폰, 타

※ 책임저자(Corresponding Author): Choong-Kyo Jeong
접수일:2012년 12월 10일 수정일:2013년 01월 26일
완료일:2013년 02월 26일
* 강원대학교 컴퓨터정보통신공학과 박사과정
** 강원대학교 컴퓨터정보통신공학과 교수
*** 강원대학교 컴퓨터정보통신공학과 교수
Tel: +82.33.250.6325 , Fax: +82.33.252.6390
email: ckjeong@kangwon.ac.kr

▣ 이 논문은 2012년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임(No. 2012-0004625)

블렛 PC와 같은 안드로이드 스마트 기기에 내장되면서 고해상도 이미지의 시각적 질을 향상시키는 기능이 중요해지고 있다.

모바일 기기의 제한된 자원과 영상의 정보량을 감안하면 지나치게 복잡하지 않은 알고리즘을 선택해야 하고 가능한 효율적인 구현 기술을 사용해야 한다. 영상의 질을 향상시키는 기법 중 간단하면서도 널리 사용되어온 방식은 명암 값 분포를 조정하는 명암비신장(contrast stretching)이다. 그러나 가상 머신 위에서 동작하는 안드로이드 API(Application Programming Interface)를 사용해서 이 기능을 구현할 경우 원시 C/C++ 프로그램으로 구현한 경우에 비해서 매우 낮은 성능을 보인다. 이 연구에서는 안드로이드 플랫폼에서 효율적인 영상처리를 위해 사용되는 기술들의 유효성을 실증적으로 비교 분석하였다.

JNI(Java Native Interface) 기술은 원시 C/C++ 코드와 자바 코드가 상호 작용할 수 있도록 하는 기술이다 [1]. 안드로이드 플랫폼에서는 NDK(Native Development Kit)를 사용하여 안드로이드 응용에서 JNI 기술을 활용할 수 있도록 한다 [2]. JNI 기술을 적용할 경우 프로그램의 복잡도가 증가한다는 단점이 있지만, 기존의 원시 C/C++ 코드들을 재사용할 수 있고 성능의 향상을 기대할 수 있다. NEON 기술은 멀티미디어 및 신호 처리 응용 프로그램의 성능을 확장시키기 위해 ARM에서 개발한 64/128비트 혼합 SIMD 확장 구조로서, 한 명령어를 실행하는 동안 같은 명령이 병렬로 최대 16개 데이터 셋까지 처리할 수 있다 [3].

이 연구에서는 자바, 원시 C.C++, NEON 버전으로 각각 두 가지 명암비신장 알고리즘(선형 명암비신장, 평활화 명암비신장 - linear contrast stretching, equalized contrast stretching)을 구현하고 안드로이드 ICS(Ice Cream and Sandwich)[4] 버전의 제품들에서 그 성능을 비교분석하였다.

2. 관련 연구

JNI 기술을 사용하여 자바 응용 프로그램의 성능을 향상시키기 위한 많은 연구들이 수행되었다. Y.-H. Lee 등은 자바 객체, 변수, 메소드에 접근하기 위한 JNI 함수의 오버헤드를 줄이기 위한 연구를 수행하였고 실험을 통해 10~30% 성능이 향상됨을 보였다[5]. C.-M. Lin 등은 원시 C/C++ 코드와 자바 코드의 성능을 비교 분석하기 위해서 안드로이드 플랫폼에서 12개의 테스트 프로그램을 실험하였고, 그 결과 원시 C/C++ 코드가 대략 34.2% 더 빠르다는 것을 확인하였다[6]. Sangchul Lee 등은 안드로이드 응용에서 JNI를 사용하여 원시 C/C++ 함수와 통신하는데 걸리는 지연 시간이 응용 프로그램 성능에 거의 영향을 끼치지 못할 만큼 매우 작다는 것을 실험을 통해 확인하였다[7]. 김대윤 등은 안드로이드 플랫폼에서 응용 프로그램에 할당되는 제한적 메모리 자원의 문제점을 해결하기 위해서 반응속도를 고려한 효율적 메모리 자원 관리에 대한 연구를 수행하였다[8]. 임영규 등은 안드로이드 시스템의 메모리 성능 향상을 위한 스택 할당 기법을 제안하였고, 실험에서 가비지 컬렉션 수행 빈도의 감소를 통해 어플리케이션 동작 및 사용자 인터페이스 성능이 향상됨을 확인하였다[9]. 추가적으로 자바와 C/C++의 성능 비교에 관련한 연구가 [10-12]에서 수행되었다.

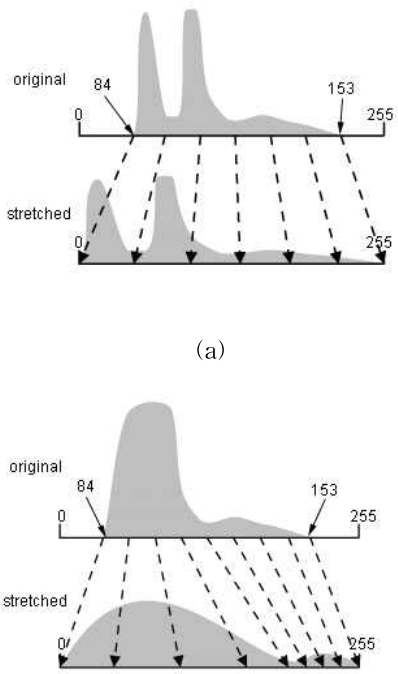
대부분의 안드로이드 스마트 기기에서 NEON 기술을 지원하는 Cortex-A8 계열의 ARM 프로세서를 사용하고 있어 NEON 코드 관련 성능 비교가 필요하다. 그러나 이전의 연구들에서는 이러한 고려가 이루어지지 않고 단지 자바 코드와 원시 C/C++ 코드의 성능 비교에만 초점을 맞추었다. 이 연구에서는 선형 명암비신장, 평활화 명암비신장 기법을 각각 자바, 원시 C/C++, 그리고 NEON 코드로 구현하였고 그 계산 시간을 비교하였다.

3. 명암비신장 연산

명암비신장은 영상의 시각적 질을 향상시키기 위한 가장 간단하고 널리 사용되는 기법이며, 영상 패턴 인식에서도 널리 사용된다. 본 본문에서는 32비트 고화질 이미지를 입력으로 하여 가장 대표적인 명암비신장 연산인 선형 명암비신장과 평활화 명암비신장의 성능을 분석한다.

명암비신장은 명암 대비를 높여줌으로써 시각적 영상 품질을 향상시키는 연산이다. 각 픽셀의 명암을 N 비트 데이터로 표현하는 경우 최대 2^N 가지 값을 표현할 수 있다. 그런데 어떤 영상이 이 중 일부 값만 사용한다면 2^N가지 값을 모두 사용하도록 각 픽셀의 명암을 조정하는 기법이다. 이 때 픽셀의 명암을 선형적으로 조정하는 방법이 선형 명암비신장이다. (그림 1)은 이미지를 구성하는 픽셀들의 명암 분포를 도수분포그래프(히스토그램)로 표현하고 이것이 명암비신장 연산을 통해 어떻게 변화되는지 보여준다. 평활화 명암비신장은 상대적으로 빈번히 발생 하는 명암값 구간은 더 많이 신장하고 드물게 발생하는 명암값 구간은 적게 신장하는 기법이다.

(그림 1) (a) 선형 명암비신장, (b) 평활화 명암비신장



(Figure 1) (a)Linear contrast stretching, (b)Equalized Contrast Stretching

4. 구현 기술

이 절에서는 3절에서 설명한 두 가지 명암비신장 알고리즘을 세 가지 기술로 구현해 그 성능을 비교해 본다. 세 가지 구현 기술은 안드로이드 자바 라이브러리, 원시 C/C++ 라이브러리, 그리고 ARM의 NEON SIMD이다.

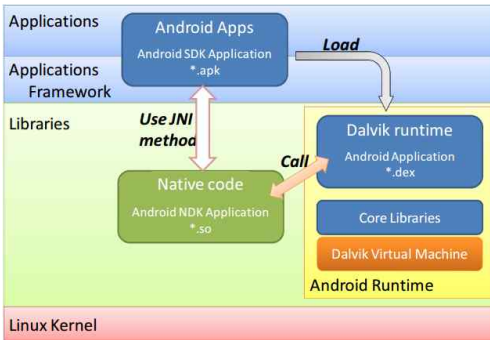
4.1 안드로이드 NDK

NDK는 원시 C/C++ 코드를 Android 플랫폼에서 활용할 수 있도록 한다. (그림 2)는 Android 프레임워크에서 원시 C/C++ 코드와 자바 코드가 JNI를 사용하여 상호 작용하는 구조를 보여준다.

4.2 ARM NEON

NEON 기술은 멀티미디어 및 신호 처리 응용 프로그램의 성능을 확장시키기 위해 ARM에서 개발한 64/128비트 혼합 SIMD 기술로서 한 명령어를 실행하는 동안 병렬로 최대 16개 데이터 셋까지 처리할 수 있다. NEON 오퍼레이션은 ARM 코어 파이프라인에서 분기가 일어나는 동안 NEON 파이프라인에서 실행된다.

(그림 2) 안드로이드 JNI 프레임워크



(Figure 2) Android JNI Framework

4.3 테스트 프로그램 구현

<표 1>의 의사코드를 세 가지 기술로 구현하여 ARGB8888 비트맵 이미지(32비트)에 적용하였다. 이미지의 R, G, B 값 각각의 히스토그램을 구하고 각각의 히스토그램에 세 가지 구현 기술을 적용하여 명암비신장 연산을 실행하고

연산시간을 측정하였다.

<표 2>는 코드 내에서 사용하는 변수들을 정리하고 있고, <표 3>과 <표 4>는 각각 NEON 명령어 셋을 적용한 히스토그램 생성과 픽셀 맵핑 단계의 코드를 보여준다.

NEON 코드는 원시 C/C++ 코드에서 히스토그램 생성과 픽셀 맵핑 부분을 NEON 명령어 셋으로 재작성하여 구현하였다. ARGB 픽셀을 R, G, B로 나누는 과정과 룩업테이블을 사용하여 변환된 R, G, B 값을 다시 ARGB 픽셀로 변환하는 과정을 NEON 명령어 셋의 vld4_u8(8x8x4-비트 데이터 로드)과 vst4_u8(8x8x4-비트 데이터 저장)를 사용함으로써 원시 C 코드의 반복문 루프 수가 1/8로 줄어들었다.

<표 1> 명암비신장 알고리즘 의사 코드

Algorithm	Linear contrast stretching	Equalized Contrast Stretching
Histogram Creation	for (i = 0; i < number_of_pixels; ++i) Histogram[Input[i]]++;	

Lookup Table Creation	highThresh = max(Histogram);
	lowThresh = min(Histogram);
	
	d = 255.0 / (highThresh-lowThresh);
	for (i = 0; i < lowThresh; ++i)
	LUT[i] = 0;
	for (i = lowThresh; i < highThresh; ++i)
	LUT[i] = (i-lowThresh) * d;
	for (i=255; i >= highThresh; --i)
	LUT[i] = 255;
		sum = 0;
		for (i = 0; i < 256; ++i)
		{
		sum += Histogram[i];
		LUT[i] = sum/number_of_pixels * 255 + 0.5;
		}

Pixel Mapping	for (i = 0; i < number_of_pixels; ++i) Output[i] = LUT[Input[i]];	

<Table 1> Pseudo-code of the contrast stretching algorithm

<표 2> 변수 선언부

Input	uint32* input // ARGB888 Bitmap (Input)

Variable declaration	uint32 Histogram_R[256];
	uint32 Histogram_G[256];
	uint32 Histogram_B[256];
	uint8 r[number_of_pixels];
	uint8 g[number_of_pixels];
	uint8 b[number_of_pixels];
	
	uint8x8_t afac = vdup_n_u8(255);
	uint8x8_t rfac = vdup_n_u8(0;
	uint8x8_t gfac = vdup_n_u8(0;
	uint8x8_t bfac = vdup_n_u8(0);
	uint8x8x4_t argb;

<Table 2> Variable Declaration

<표 3> 히스토그램 생성을 위한 원시 C/C++ 코드와 NEON 코드

Input	uint32* input // ARGB888 Bitmap (Input)

Native C/C++	for (i = 0; i < number_of_pixels; ++i) {
	r[i] = (input[i] >> 16) & 0xFF;
	g[i] = (input[i] >> 8) & 0xFF;
	b[i] = input[i] & 0xFF;
	++Histogram_R[r[i]];
	++Histogram_G[g[i]];
	++Histogram_B[b[i]];
	}

NEON	for (i = 0; i < number_of_pixels; i += 8) {
	argb = vld4_u8((uint8 *)(input+i));
	vst1_u8((uint8 *)(r+p), argb.val[2]);
	vst1_u8((uint8 *)(g+p), argb.val[1]);
	vst1_u8((uint8 *)(b+p), argb.val[0]);
	}
	for (i = 0; i < number_of_pixels; i += 8) {
	++Histogram_R[r[j]];
	++Histogram_G[g[j]];
	++Histogram_B[b[j]];
	}

<Table 3> Native C/C++ and NEON code for histogram creation

<표 4> 픽셀 맵핑을 위한 원시 C/C++ 코드와 NEON 코드

Return	uint32* output // ARGB888 Bitmap (Result)

Native C/C++	for (i = 0; i < number_of_pixels; ++i) {
	output[i] = 0xFF000000 | (r[i] << 16) | (g[i] << 8) | b[i];
	}

NEON	for (i = 0; i < number_of_pixels; i += 8) {
	argb.val[3] = vdup_n_u8(255); // alpha is fixed (255)
	argb.val[2] = vld1_u8((unit8 *)(r+i));
	argb.val[1] = vld1_u8((unit8 *)(g+i));
	argb.val[0] = vld1_u8((unit8 *)(b+i));
	vst4_u8((uint8 *)(output+i), argb);
	}

<Table 4> Native C/C++ and NEON code for pixel-mapping

5. 실 험

성능 비교를 위해 Android ICS(Ice Cream and Sandwidth) 버전을 탑재한 Galaxy S2, HTC EVO 4G, Vega S 제품들을 사용하였다. <표 5>는 각 제품의 스펙을 보여준다. 실험에서는 자바, 원시 C/C++, NEON 코드를 3000회씩 호출하여 그 평균값을 측정하였다. 코드 실행은 시스템 프로세스를 제외한 모든 응용 프로그램을 종료한 상태에서 수행하였고 실행 시간에 큰 영향을 줄 수 있는 GC(Garbage Collection)가 라이브러리 실행 중간에 실행되지 않았음을 로그 분석을 통해 확인하였다.

<표 5> 테스트 기기 스펙 정리

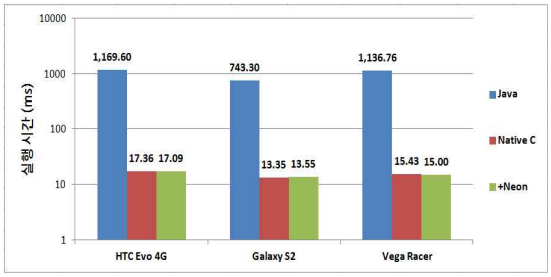
Device	Manufacturer	Processor	Memory
EVO 4G	HTC	Dual-Core 1.2GHz	1GB
Galaxy S2	Samsung	Dual-Core 1.2GHz	1GB
Vega Racer	Sky	Dual-Core 1.5GHz	1GB

<Table 5> Test device specifications

원시 C/C++로 작성한 선형 명암비신장 및 평활화 명암비신장 코드의 경우 대략 70%의 연산 시간이 히스토그램 생성과 픽셀 맵핑 단계에서 소요됨을 확인하였다.

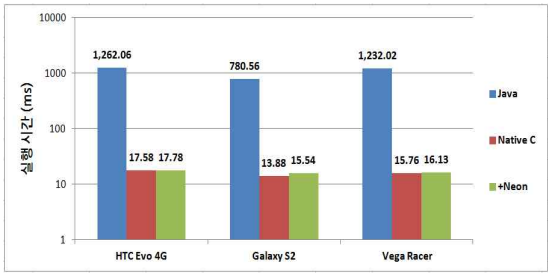
(그림 3)은 세 가지 구현 기술로 작성한 선형 명암비신장 코드가 세 가지 스마트폰에서 실행된 시간을 보여준다. (그림 4)는 평활화 명암비신장 코드의 실험 결과이다.

(그림 3) 실행 시간 비교 - 선형 명암비신장



(Figure 3) The execution time - Linear contrast stretching

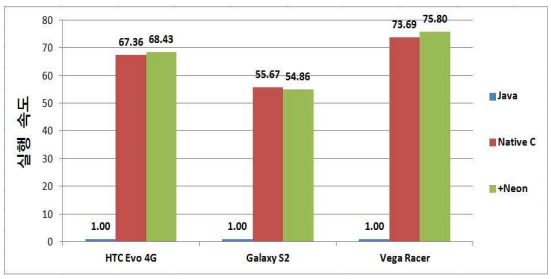
(그림 4) 실행 시간 비교 - 평활화 명암비신장



(Figure 4) The execution time - Equalized contrast stretching

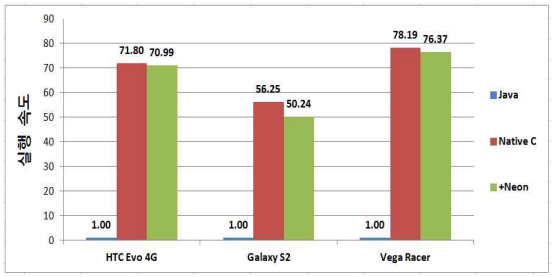
(그림 5)와 (그림 6)은 세 가지 구현 기술에 의한 코드 실행 시간을 자바 코드를 기준으로 정규화한 그래프이다. 원시 C/C++, NEON 코드는 자바 코드 비해 50배에서 78배의 속도를 보였고, 원시 C/C++ 코드와 NEON 코드는 거의 유사한 성능을 보였다. 원시 C/C++ 코드의 히스토그램 생성과 픽셀 맵핑 부분에서 루프 수를 1/8로 줄였음에도 더 높은 성능이 나오지 않은 까닭은 NEON 명령어 셋을 병렬 산술/논리 연산이 아니라 단순히 8x8x4 비트 데이터 로드 및 저장에 사용하였고, 이 때 메모리와 프로세서 간 통신 지연(latency)이 병목으로 작용하였기 때문이다.

(그림 5) 실행 속도 비교 - 선형 명암비신장



(Figure 5) Comparison of the execution times - Linear contrast stretching

(그림 6) 실행 속도 비교 - 평활화 명암비신장



(Figure 6) Comparison of the execution times - Equalized contrast stretching

멀티-태스킹 구조를 지원하는 안드로이드 모바일 플랫폼에서 응용 프로그램의 성능 향상은 배터리의 소모량을 줄이고 다른 응용 프로그램들의 응답 속도를 향상시킨다는 점에서 매우 중요하다. 그러므로 본 연구의 실험 결과는 히스토그램 연산을 사용하는 안드로이드 응용 프로그램 혹은 유사 프로그램 개발 과정에서 유용하게 활용될 수 있으며, 안드로이드 응용 프로그램의 성능 개선에 직·간접적으로 참고 될 수 있다.

6. 결론

이 연구에서는 안드로이드 플랫폼에서 영상의 시각적 질을 향상시키기 위한 명암비신장 연산을 대상으로 자바, 원시 C/C++, NEON 등 세 가지 구현 기술의 실행 속도를 비교하였다. 그 결과 Native C/C++, NEON 코드의 실행 속도가 자바 라이브러리 실행 속도에 비해 각각 55-78배, 50-76배 향상됨을 관찰하였다. 원시 C/C++ 코드와 NEON 코드는 그 속도가 유사함을 확인하였다. 이 실험 결과는 히스토그램 연산을 사용하는 안드로이드 응용 프로그램 혹은 유사 프로그램 개발 과정에서 유용하게 활용될 수 있으며, 안드로이드 응용 프로그램의 성능 개선에서 직·간접적으로 참고 될 수 있다.

References

[1] http://docs.oracle.com/javase/6/docs/technotes/guides/jni/spec/jniTOC.html

[2] http://developer.android.com/tools/sdk/ndk/index.html

[3] http://www.arm.com/products/processors/technologies/neon.php

[4] http://www.android.com/about/ice-cream-sandwich/

[5] Y.-H. Lee, P. Chandrian and B. Li, "Efficient Java Native Interface for Android Based Mobile Devices, " in Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on, Changsha, Hunan Province, P. R. China, 2011.

[6] C.-M. Lin, J.-H. Lin, C.-R. Dow, and C.-M. Wen, "Benchmark Dalvik and Native Code for Android System," in 2011 Second International Conference on Innovations in Bio-inspired Computing and Applications (IBICA), 2011, pp. 320 - 323.

[7] Sangchul Lee and Jae Wook Jeon "Evaluating Performance of Android Platform Using Native C for Embedded Systems" 2010 International Conference on Control Automation and Systems (ICCAS), pp. 1160-1163.

[8] Daeyoun Kim, Hongchul Mo, Jongho Nang, "Memory Management Method for Improving Response Time of Android Applications," Journal of KISS : Software and Applications, VOL.40, No.1, 2013.1,1-7.

[9] Yeongkyu Lim, Cheong Ghil Kim, Shin Dug Kim,

"Stack Allocation-based Memory Performance Improvement Technique on Android 2.3 Dalvik Virtual Machine", Journal of Digital Contents Society Vol.12 No.4 Dec. 2011(pp.551-557)

[10] Przemyslaw Bruski, "The Java (not really) Faster than C++ Benchmark," Available: http://bruscy.republika.pl/pages/przemek/java_not_really_faster_than_cpp.html.

[11] Christian Felde, "C++ vs Java performance; It's a tie!," Available: http://blog.cfelde.com/2010/06/c-vs-java-performance.

[12] Lasse Kärkkäinen, "Yet Another Java vs. C++ Shootout,," Available: http://zi.fi/shootout.

이 주 호

2005년 : 강원대학교 전기전자정보통신공학부 (학사)
2007년 : 강원대학교 컴퓨터정보통신공학과 (석사)
2007년~2009년 : 기산텔레콤
2009년~2011년 : 건아정보기술
2011년~ 현 재 : 강원대학교 컴퓨터정보통신공학과 (박사과정)
관심분야 : 이동통신, 빅데이터, Cloud Computing, VoIP, Embedded Linux

이 구 연

1986년 : 서울대학교 전자공학과 (학사)
1988년 : KAIST 전기및전자공학과 (석사)
1993년 : KAIST 전기및전자공학과 (박사)
1993년~1996년 : 디지콤정보통신연구소
1996년 : 삼성전자
1997년~현 재 : 강원대학교 컴퓨터학부 교수
관심분야 : 이동통신, 네트워크보안, 초고속통신망, ad-hoc 네트워크

정 충 교

1982년: 서울대학교 전기공학과 (학사)
1984년: KAIST 전기및전자공학과 (석사)
1989년: KAIST 전기및전자공학과 (박사)
1989년~1995년 : LG정보통신(주)
1995년~현 재 : 강원대학교 컴퓨터학부 교수
관심분야 : 인터넷, 통신망성능분석, 시스템프로그래밍, 소프트웨어품질