한국컴퓨터그래픽스학회
Korea Computer Graphics Society Vol. 28, No. 2, P. 21~28

PSNR 값 기반의 자동화된 ASTC 블록 크기 결정 방법

나재호°

상명대학교

jaeho.nah@smu.ac.kr

ASTC Block-Size Determination Method based on PSNR Values

Jae-Ho Nah°

Sangmyung University

요 약

ASTC는 OpenGL ES 3.2 및 Vulkan 1.0 이상의 버전에서 지원하는 표준 텍스쳐 포맷 중 하나로, 모바일 플랫폼(Android 및 iOS)에서 지속적으로 사용이 증가해 왔다. ASTC의 가장 큰 특징은 블록 크기 설정으로, 이를 통해 품질과 압축률 간의 트레이드 오프를 조절할 수 있다. 하지만 텍스쳐의 개수가 많을 경우 텍스쳐별 최적의 블록 크기를 일일히 수작업으로 설정하는 것은 많은 시간과 노고를 야기하게 된다. 이러한 문제점을 해결하기 위해 본 논문은 PSNR 값을 기반으로 자동으로 ASTC 블록 크기를 결정하는 새로운 방법을 제안한다. 모든 블록 크기에 대해 압축을 수행한 후 PSNR값을 비교하는 brute-force 방식은 최고 14배까지 압축 시간을 증가시킬 수 있는 반면, 본 논문의 방법은 압축 과정을 3단계로 나누어 이러한 압축 시간 증가를 최소화한다. 다양한 형태의 64개 이미지로 구성된 텍스쳐 셋을 통해 실험한 결과, 제안하는 방법은 텍스쳐별로 4x4에서 12x12까지 다양한 블록 크기를 결정하였으며, 블록 크기를 6x6으로 일괄적으로 정한 경우에 비해 압축된 파일들의 총 크기가 68% 감소하였다.

Abstract

ASTC is one of the standard texture formats supported in OpenGL ES 3.2 and Vulkan 1.0 (and later versions), and it has been increasingly used on mobile platforms (Android and iOS). ASTC's most important feature is the block size configuration, thereby providing a trade-off between compression quality and rates. With the higher number of textures, however, it is difficult to manually determine the optimal block sizes of each texture. To solve the problem, we present a new approach based on PSNR values to automatically determine the ASTC block size. A brute-force approach, which compresses a texture on all block sizes and compares the PSNR values of the compressed textures, can increase the compression time by up to 14 times. In contrast, our three-step approach minimizes the compression-time overhead. According to our experiments on a texture set including 64 various textures, our method determined the block sizes from 4x4 to 12x12 and reduced the size of compressed files by 68%.

키워드: 텍스쳐 압축, 텍스쳐 인코딩, ASTC, GPU
Keywords: Texture compression, Texture encoding, ASTC, GPU

1. 서론

텍스쳐 매핑은 컴퓨터 그래픽스에서 현실감 넘치는 렌더링 결과를 위해 필수불가결하게 사용되는 기법 중 하나이다. 최근의 어플리케이션들은 더욱 더 현실에 근접한 렌더링 결과를 만들어내기 위해 점점 더 다양한 형태의 고해상도 텍스쳐 데이터를 사용하는 추세에 있으며, 텍스쳐 압축 기법들은 이러한 대용량 고해상도 텍스쳐로 인해 증가된 요구 메모리 크기와 대역폭, 전력 소모량을 모두 감소시키는 데 도움을 준다[1]. 이와 같이 압축된 텍스쳐 데이터는 디스크 및 GPU 메모리 상에 저장되고, 필요시 실시간으로 GPU 내의 디코더를 통해 압축을 풀어 사용하게 된다.

*corresponding author: Jae-Ho Nah/Sangmyung University(jaeho.nah@smu.ac.kr)

Received : 2022.01.24./ Review completed : 1st 2022.03.11. / Accepted : 2022.05.24.
DOI : 10.15701/kcgs.2022.28.2.21
ISSN : 1975-7883(Print)/2383-529X(Online)

데스크탑 및 모바일 플랫폼에서 사용되는 표준 압축 형식은 크게 4가지로, BC[2], ETC1/2[3, 4], PVRTC[5], ASTC[6]이다. 주로 데스크탑 플랫폼에서는 BC가, 안드로이드에서는 ETC1/2가, iOS에서는 PVRTC가 쓰여 왔으며, ARM사와 AMD사에서 제안한 ASTC는 OpenGL ES 3.2 또는 Vulkan 1.0 이상을 지원하는 안드로이드와 iOS 플랫폼 모두에서 사용이 되는 가장 최신의 표준 압축 형식이다.

ASTC가 다른 포맷과 가장 차별화되는 점은 압축할 블록의 크기를 설정하여 품질 및 압축률을 조절할 수 있다는 점이다. 하지만 수 천, 수 만개의 텍스쳐가 존재하는 앱의 경우 모든 개별 텍스쳐에 대해서 최적의 블록 크기를 일일히 지정하는 것은 불가능에 가깝기 때문에, 보통 카테고리 또는 품질을 기준으로 일괄적인 블록 크기를 설정하는 방법이 추천되고 있다[7, 8, 9]. 하지만 이 경우 텍스쳐의 특성에 따라 압축 아티팩트(artifact)를 발생시키거나 불필요한 용량 증가의 결과를 나타낼 수 있다.

본 논문에서는 위에 서술한 문제점을 해결하는 방안으로, 타겟 PSNR값에 따라 각 텍스쳐별로 자동으로 ASTC 블록 크기를 결정하는 방법을 제안한다. 이 방법은 전처리를 통해 카테고리별 타겟 PSNR값을 정한 후, 텍스쳐별로 고속 압축을 수행하여 이 타겟 PSNR값을 만족하는 최적 블록 크기를 검색하며, 이 결과값을 이용하여 최종 압축을 수행한다. 이러한 3단계 방법은 압축 시간 증가는 최소화하면서도 텍스쳐별로 각기 다른 블록 크기를 사용하는 것을 가능하게 하였다. QuickETC2[10, 11] 논문에서 사용된 64개의 이미지로 이루어진 텍스쳐 셋(Figure 1)을 이용하여 실험한 결과, 제안하는 방법은 텍스쳐별 PSNR 편차를 크게 줄였고, 압축된 전체 파일들의 용량을 68% 감소시켰으며, 반면 압축 시간의 증가는 1.3배 수준으로 나타나 실용적으로 사용하기에 크게 문제 없는 방법으로 나타났다.

2. 관련 연구

2.1 Adaptive Scalable Texture Compression (ASTC)

ASTC[6]는 기존 압축 포맷들에 비해 여러가지 측면에서 이점을 가지는 포맷이다. 이 포맷의 가장 두드러지는 특징은 블록 크기 조정을 통해 2차원 텍스쳐는 픽셀당 8비트(bits per pixel, bpp)부터 0.89비트까지, 3차원 텍스쳐는 픽셀당 4.74 비트부터 0.59 비트까지 압축률을 조절할 수 있다는 점이다. 이를 통해, 개발자에게 압축률과 압축 품질간 트레이드 오프(trade-off)를 제공한다. 또한, 다양한 컬러 채널 수(1-4채널)와 형식(LDR & HDR, 2D & 3D)을 지원하므로, 이 포맷 하나로 매우 광범위한 범위의 텍스쳐들을 압축할 수 있다. 또한 블록당 최고 4개의 파티션 및 independent, base+offset, base+scale과 같은 다양한 압축 모드를 지원하고 이와 관련된 값들을 bounding integer sequence encoding(BISE)으로 표현한다. 이를 통해, 4x4 블록 크기로 압축시 기존 고화질 압축 포맷으로 유명한 BC7과 유사한 품질을, 6x6 블록 크기 사용시 ETC2보다 좀 더 나은 품질을 보여준다[6].

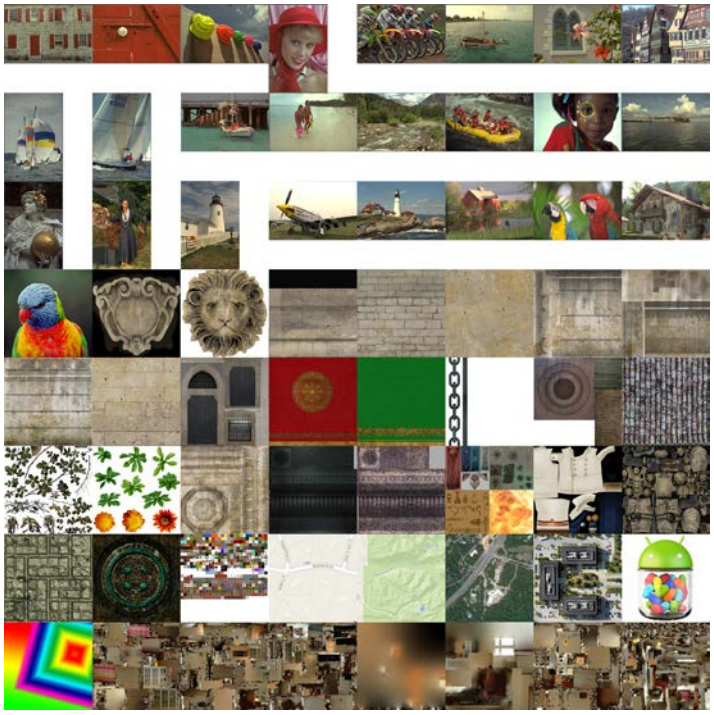


Figure 1: The texture set of 64 test images used in our experiments. If you are interested in a detailed description of this texture set, please refer to the supplemental document of the QuickETC2 paper [11].

ASTC는 OpenGL ES 3.2 이상과 Vulkan 1.0 이상을 지원하는 GPU에서 사용 가능하며, 현재 출시되는 대부분의 안드로이드 및 iOS 기기들은 ASTC 디코더를 내장한 GPU를 탑재하고 있다. 또한 구글의 Android app bundle[12]은 타겟 기기별로 각기 다른 텍스쳐 압축 포맷을 사용 가능하게 만들어 주었고, 그래서 이제는 구형 안드로이드 기기에서의 호환성을 염려하던 개발자들도 ASTC 사용을 고려해 볼 수 있게 되었다. 이러한 이유로, ASTC로 압축된 텍스쳐를 포함하는 모바일 앱의 비율은 점점 높아지는 추세이다.

2.2 최근 ASTC 관련 연구

ASTC는 포맷 자체가 복잡한 만큼 인코딩 시간도 다소 긴 단점이 있다. 이를 보완하기 위해 최근 몇 년동안 ASTC 인코딩 시간을 줄이기 위한 여러가지 연구가 진행되었다. 여러 GPU 제조사들은 개발자들을 위해 ASTC 인코더를 개발하여 공개하고 있는데, 대표적으로 ARM사에서는 레퍼런스 인코더인 astcenc[13]를 지속적으로 업데이트 하고 있고, 가장 최근 버전은 2021년 11월에 릴리즈된 버전 3.3이다. 인텔사와 엔비디아사는 각각 CPU와 GPU의 이점을 최대로 이용하는 ISPC texture compressor[14]와 NVIDIA Texture Tools 3[15]를 발표하였으며, 두 인코더 모두 ASTC LDR 형식을 지원한다. 학계에서 진행된 고속 ASTC 인코딩에 관한 연구도 존재한다. astcrt[16]은 웹 상에서의 실시간

인코딩을 목적으로 다소 품질이 저하되더라도 성능을 향상시킬 수 있는 여러가지 휴리스틱을 적용한 인코더이고, TexNN[17]은 고품질 인코딩을 가속하기 위해 신경망을 이용하는 기법이다.

인코딩 가속이 아닌 다른 측면에서 진행된 연구들도 존재한다. Griffin과 Olano[18] 및 Lavoué 등[19]은 멀티 텍스쳐링을 이용하여 렌더링을 수행한 후의 masking 효과를 ASTC의 각 블록 크기별로 조사하였다. Binomial LLC사는 Basis Universal이라는 supercompressed 코덱[20]을 발표했는데, 이를 이용하면 텍스쳐가 기존 텍스쳐 압축 코덱들보다 더 높은 압축률을 가지는 UASTC 또는 ETC1S로 인코딩된 상태로 앱에 저장이 되고, 실제 GPU에서 사용시에는 다양한 표준 압축 코덱들 중 하나로 트랜스코딩(transcoding)된다. 이 코덱은 최근 Khronos에서 발표한 KTX 2.0 컨테이너 포맷[21]에서도 지원된다.

3. 제안하는 방법

이 장에서는 본 논문에서 제안하는 ASTC 블록 크기 결정 방법을 서술한다. 이 방법의 가장 큰 특징은 모든 텍스쳐들에 대해 일괄적인 블록 사이즈를 설정하는 대신, Figure 2에서 보여지는 것과 같이 카테고리별로 다른 타겟 PSNR값을 설정한 후 각 텍스쳐별로 이 타겟 값을 만족하는 블록 사이즈를 선택한다는 점이다. 본 논문에서 제안하는 방법은 크게 세 단계로 나뉘는데, 전처리 단계에서 카테고리별 수동 분석을 수행한 후, 최적 블록 크기 검색 단계에서 자동으로 텍스쳐별 블록 크기를 결정한 다음, 최종적으로 이 블록 크기를 이용하여 압축이 수행된다.

3.1 전처리 단계

이 단계에서는 Chait의 image quality (IQ) 등급 산정 방식[7]과 유사하게, 텍스쳐별로 각 블록 크기에 대해서 압축을 수행한 후 이상적인 블록 크기를 찾는다. 압축률과 압축 품질은 반비례하나, 압축 품질이 일정 수준 이상일 경우에 도달하게 되면 압축 품질의 차이는 눈에 잘 띄지 않기 때문에, 결국 이상적인 블록 크기는 압축 아티팩트가 눈에 잘 눈에 띄지 않는 가장 큰 블록 크기를 의미한다. Chait의 분류 기준으로는, IQ값이 양수인 경우가 이에 해당한다. 그리고 이 블록 크기에 대해 fastest 모드로 압축을 수행했을 때의 PSNR값을 토대로, 다음 단계에서 사용될 카테고리별 타겟 PSNR값과 검색 시작 블록 크기를 구한다. 이와 같이 이상적인 블록 크기를 찾는 과정은 수작업으로 이루어지지만, 한 번 구해진 카테고리별 결과값은 해당 카테고리에 속하는 다른 텍스쳐들에 대해서도 공통적으로 적용할 수 있으므로, 이러한 일련의 과정은 전처리라고 볼 수 있다.

상세한 과정은 다음과 같다. 본 논문에서는 Figure 1의 64개 텍스쳐들을 이용하여 전처리를 수행하였는데, 이 텍스쳐들은 크게 사진 텍스쳐들, 게임 텍스쳐들, GIS 데이터, 합성 이미지들, 카메라를 통해 현실에서 캡쳐된 이미지들, 이렇게 다섯 카테고리로 나뉘어져 있다. 이 단계에서 가장 먼저 수행하는 작업은 각각의 텍스쳐에 대해 astcenc의 thorough모드를 이용하여 블록 크기별로 압축을 수행한 후 이상적인 블록 크기를 찾는 것이다. 즉, 이 블록 크기는 블록 아티팩트(block artifact)나 블러링(blurring)이 크게 보이지 않으면서도 압축률을 극대화할 수 있는 블록 크기가 된다. 이 때 thorough 모드를 사용하는 이유는 최종 결과물을 산출하는 세번째 단계와 압축 모드를 일치시키기 위해서이다.

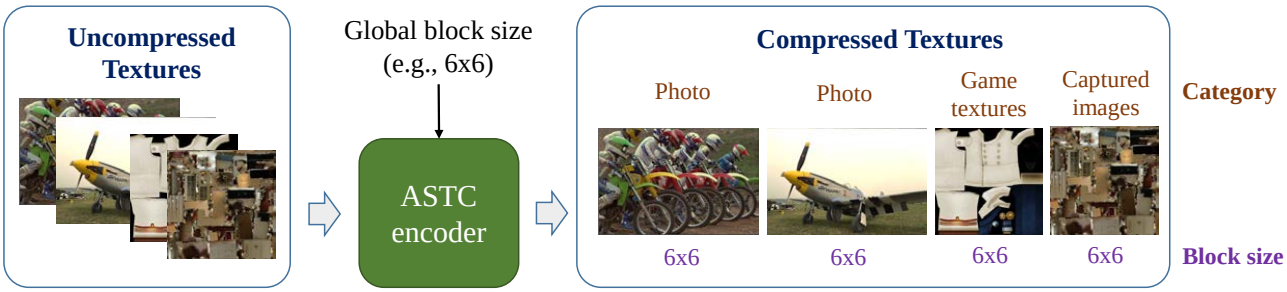
각 텍스쳐별로 분석이 완료되었으면, 다음으로 텍스쳐별로 구해진 이상적인 블록 크기를 이용해 두번째 단계에서 사용할 fastest 모드로 다시 압축을 수행한 후 PSNR값을 산출한다. astcenc는 내부적으로 압축시 PSNR값을 이용하고 PSNR값 출력 옵션도 제공하기 때문에, PSNR값 계산을 위해 별도의 프로그램을 실행할 필요는 없다. 각 텍스쳐별로 PSNR값이 구해졌으면 이를 토대로 카테고리별 타겟 PSNR값을 구한다. 본 논문에서 사용하는 타겟 PSNR값은 아래 식과 같은데, 이는 PSNR값으로 동일 카테코리 내의 텍스쳐들을 정렬했을 때 상위 25% 수준에 해당하는 PSNR값을 타겟 값으로 설정함을 의미한다. 만약 이 타겟값을 평균 PSNR값으로 설정할 경우 몇몇 텍스쳐들이 눈에 띄는 압축 아티팩트를 보이게 되고, 타겟값을 최고 PSNR값으로 설정할 경우에는 동일 카테고리 내 PSNR값이 높은 하나의 텍스쳐에 의해서 압축률이 너무 낮게 설정되기 때문에, 본 논문은 카테고리별로 PSNR값과 평균 PSNR값의 중앙값을 타겟 PSNR로 설정하였다.

$$PSNR_{TARGET} = \frac{PSNR_{MAX} + PSNR_{AVG}}{2} \qquad (1)$$

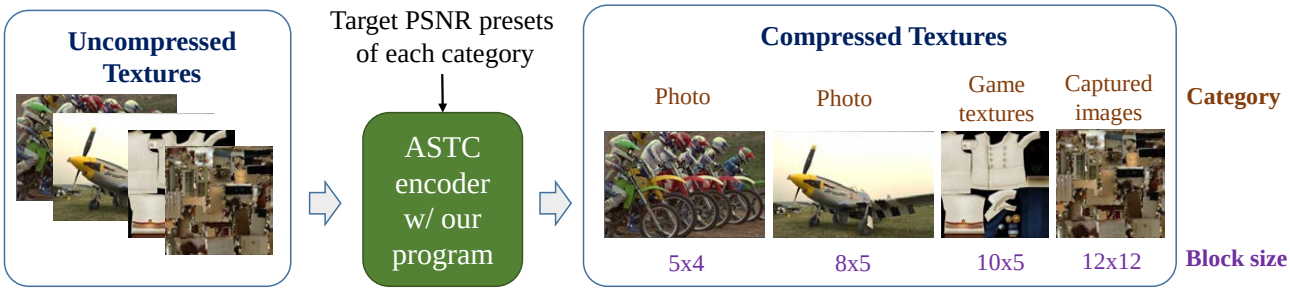
다만 위 식을 적용하지 않는 예외도 존재한다. 어떠한 카테코리의 모든 이미지가 가장 큰 압축 블록(12x12)에서도 충분한 품질을 보여줬을 경우, 위 식을 사용하면 다음 단계에서 설정할 최적 블록 크기가 12x12보다 작아지는 결과가 초래된다. 즉, 불필요하게 압축률을 낮추는 결과를 초래할 수 있는 것이다. Figure 1에서는 8K 해상도의 캡처된 이미지가 이에 해당되는데, 이 카테고리의 텍스쳐들은 가장 큰 12x12 블록 크기에서도 44-55 dB의 높은 PSNR값을 나타냈다. 따라서 이 카테고리에 대해서는 타겟 PSNR값을 40 dB로 설정하였는데, 이는 일반적으로 PSNR 40 dB 이상의 결과물은 높은 품질로 여겨지기 때문이다 [22].

이 단계에서는 카테고리별 타겟 PSNR값과 더불어, 2번째 단계에서 검색을 시작할 때 카테고리별로 다르게 사용할 블록 크기를 설정한다. 이 시작 블록 크기의 적절한 설정은 최종 결과물에는 영향을 끼치지 않지만, 2번째 단계에서의 검색 오버헤드를 줄이는 데 도움이 된다. 검색 시작 블록 크기도 타겟 PSNR값과 유사한 수준으로 설정해야 검색 횟수를 최소화할 수 있기 때문에, 블록 크기별로 동일 카테고리 내의 텍스쳐들을 정렬한 후 상위 25% 수준의 블록 크기를 시작 블록 크기로 설정하였다.

본 논문에서 전처리 단계에 사용한 텍스쳐의 개수는 64개이기 때문에 모든 텍스쳐들에 대해서 위에서 서술한 분석 작업을 수행하였다. 하지만, 전체 텍스쳐의 개수가 이보다 수십 수백배 많다면 이러한 수동 분석 작업은 불가능할 것이다. 따라서 이러한



(a) Traditional ASTC compression approach with a global block size



(b) Our approach based on PSNR values

Figure 2: This figure illustrates the core difference between (a) a traditional compression approach with a global block size and (b) our approach. In contrast to the traditional approach, our approach selects a proper block size of each texture on the basis of target PSNR presets of each category.

경우에는 동일 카테고리에서 일부 샘플만 추출함으로써 필요한 시간과 노력을 줄일 수 있다. 또한 다수의 고해상도 텍스쳐가 존재할 경우 이 해상도를 낮춘 후 전처리를 수행하는 다운샘플링도 고려할 수 있지만, 이 경우 블록당 저장되는 정보의 양이 달라지기 때문에 그보다는 샘플 추출이 적합할 것으로 보인다. 또는 앞에서 언급한 것과 같이, 별도로 전처리를 수행하지 않고 본 논문에서 카테고리별로 산출된 값들(4.2절에 소개)을 그대로 또는 일부 수정하여 사용해도 될 것이다. 만약 좀 더 품질을 높이고 싶을 경우에는 타겟 PSNR값을 높이고 시작 블록 크기를 작게 하면 되고, 좀 더 압축률을 높이고자 할 경우에는 타겟 PSNR값을 낮추고 시작 블록 크기를 좀 더 크게 설정하면 된다.

3.2 최적 블록 크기 검색 단계

이 단계에서는 앞 단계에서 구해진 타겟 PSNR값과 시작 블록 크기를 이용하여, 타겟 PSNR값을 만족시키는 가장 큰 크기의 블록을 검색하고, 이 결과를 다음 단계로 넘긴다. 직관적인 brute-force 방식으로 이 과정을 수행한다면, 모든 블록 크기에 대해 압축을 한 후 조건을 만족하는 블록 크기를 선택하면 되지만, 2D LDR 텍스쳐의 경우 14배 긴 압축 시간을 필요로 한다. 순차 검색은 조건을 만족하는 블록 크기가 발견되었을 때 검색 중단을 할 수 있기 때문에 조금 더 성능이 나을 수 있다. 이 경우 먼저 가장 작거나 큰 블록 크기부터 압축을 시작하여, 압축된 결과물의 PSNR값이 타겟 PSNR값을 만족하는지 비교한 후, 이를 만족하지 않을 경우 다음 블록으로 넘어가고, 이를 만족하는 블록 크기가 나오면 더 이상의 검색을 중단하고 이 결과물을 최종 압축 결과로 사용하면 된다. 하지만 이러한 순차 검색 방법을 이용하더라도 최악의 경우에는 brute-force 방식과 동일하게 14배 긴 압축 시간을 요하게 되고, 평균적으로도 평균 검색 횟수의 배수만큼 압축 시간을 요하게 된다.

이러한 압축 오버헤드를 최소화하기 위해, 본 논문에서는 두 가지 방안을 함께 사용한다. 첫째, 최적 블록 크기 검색을 수행할 때에는 astcenc의 fastest 모드를 사용하고, 최종 압축시에만 thorough 모드를 사용한다. 이 두 모드의 압축 속도는 대략 10배 이상 차이 난다. 둘째, 가장 작거나 큰 블록 크기로부터 정해진 방향으로 순차 검색을 하는 대신, 앞단계에서 산출된 시작 블록 크기부터 압축을 시작하여, 이 블록 크기에서의 PSNR값을 토대로 검색 방향을 달리한다. 이러한 방법을 통해 검색 횟수를 감소시킬 수 있다.

Table 1은 이 단계의 의사코드를 기술한다. 먼저 블록 크기를 담은 배열과 현재 블록 크기, 그리고 검색 방향을 설정한다(1-3번째 줄). 현재 블록 크기의 초기값은 이전 단계에서 설정한 시작 블록 크기 값이고, 검색 방향은 아직 결정되지 않았으므로 초기값이 0이다. 다음으로 do-while 루프를 돌면서 검색을 수행하는데

Table 1: Pseudocode of the optimal block-size search stage.

```
int blockSizeSearch (fileName, targetPSNR, startBlockSizeIdx)
01.   blockSizes[14] = { 4x4, 5x4, ... , 12x12 }
02.   currentBlockSizeIdx = startBlockSizeIdx
03.   direction = 0
04.   do
05.     execute astcenc with
        blockSizes[currentBlockSizeIdx] and fileName
06.     obtain PSNR from astcenc
07.     if (direction == 0)
08.       direction = (PSNR≥targetPSNR)? 1:-1
09.     if (direction == 1)
10.       if (PSNR<targetPSNR)
11.         currentBlockSizeIdx -= 1
12.         break
13.       if (currentBlockSizeIdx == 13)
14.         break
15.     if (direction == -1)
16.       if (PSNR≥targetPSNR ‖ currentBlockSizeIdx == 0)
17.         break
18.     currentBlockSizeIdx += direction
19.   while (true)
20.   return currentBlockSizeIdx
```

(4-19번째 줄), 루프 안에서는 가장 먼저 현재 블록 크기와 파일 이름을 astcenc의 인자로 넘겨 압축을 수행한 후 압축된 결과물의 PSNR값을 얻는다 (5-6번째 줄). 이후 검색 방향이 초기값일 경우에는 향후 검색 방향을 설정해야 하는데(7-8번째 줄), 압축된 결과물의 PSNR값이 타겟 PSNR값보다 크거나 같을 경우에는 검색 방향을 오른쪽(1)으로 설정하여 계속 블록 크기를 증가시키게 되고, 그렇지 않을 경우에는 검색 방향을 왼쪽(-1)으로 설정하여 계속 블록 크기를 감소시킨다. 이후 9-17번째 줄에서는 검색 종료 조건을 검사한다. 검색 방향이 오른쪽일 때에는, 타겟 PSNR보다 작은 PSNR값을 보이는 블록 크기가 최초로 발견되었을 경우 이보다 한 단계 큰 블록 크기를 현재 블록 크기로 설정하면서 루프를 종료하고 (10-12번째줄), 가장 큰 블록 크기인 12x12에 도달했을 때에도 루프를 종료한다 (13-14번째 줄). 검색 방향이 왼쪽일 경우, 산출된 PSNR값이 타겟 PSNR값보다 크거나 같은 경우, 또는 가장 작은 블록 크기인 4x4에 도달한 경우에 loop를 종료한다(16-17번째 줄). 이러한 종료 조건을 만족하지 않을 경우에는 현재 블록 크기를 검색 방향에 맞게 증가 또는 감소시키고 (18번째 줄) 검색을 계속한다. 이 단계의 최종 반환값은 위 루프가 끝난 후의 현재 블록 크기가 된다(20번째 줄).

3.3 최종 압축 단계

이 단계에서는 앞 단계에서 구해진 블록 크기를 이용하여 최종적으로 텍스쳐의 압축을 수행한다. 최종 결과물의 품질은 가능한 한 높아야 하므로, 본 논문에서는 astcenc [13]의 thorough모드를 이용한다.

4. 실험 및 결과

4.1 실험 환경

실험 환경은 다음과 같다. 먼저 실험에 사용된 하드웨어는 AMD Ryzen 5 3600 XT (6코어 12쓰레드) CPU, 16GB RAM, 1TB M.2 SSD를 가진 데스크탑이다. 실험에 사용된 소프트웨어는 윈도우10과 astcenc 3.3[13]이며, 제안하는 방법은 JDK 16을 이용하여 구현하였다. 비교군으로는 6x6 블록 크기(3.56 bpp)를 사용했는데, 그 이유는 Unity에서 기본값으로 추천하는 일반 품질(normal quality)에 해당되기도 하고, 모바일 기기에서 널리 쓰이고 있는 다른 4 bpp의 압축 포맷들(ETC2, PVRTC)에 대응되는 크기이기도 하기 때문이다. 본 실험을 위해서 사용된 텍스쳐들은 전처리 단계와 동일하게 Figure 1의 64개 텍스쳐들이기 때문에, 실제로 다른 텍스쳐를 이용하여 압축을 수행할 때에는 본 장에서 기술한 결과와 상이한 결과를 얻을 수도 있다.

4.2 품질 비교

이 절에서는 6x6 블록 크기를 사용했을 때와 본 논문의 방법을 사용하였을 때 압축 품질이 어떻게 다른지 정량적 및 정성적으로 비교한다. 먼저 Figure 3는 전체 테스트 셋의 PSNR 수치를 보여준다. 6x6 블록 크기를 사용했을 때와 비교해 보면, 제안하는 방법이 압축된 텍스쳐간 PSNR 편차를 크게 감소시킴을 볼 수 있으며, 이는 제안하는 방법이 타겟 PSNR 값을 이용하여 블록 크기를 결정하기 때문이다. 참고로, 카테고리별로 산출된 타겟 PSNR 값은, 사진(1번-25번)은 37.1351 dB, 게임(26번-51번)은 37.3873 dB, GIS 데이터(52번-55번)는 40.9436 dB, 합성 이미지(56번-57번)는 42.4310 dB, 캡처된 이미지(58번-64번)는 40.0000 dB이다.

이와 같이 텍스쳐간 감소된 PSNR 편차는 압축률과 압축 품질간 밸런스가 개선되었음을 의미한다. 즉, 본 논문의 방식을 사용하면 압축 후 PSNR값이 낮고 아티팩트가 눈에 띄는 이미지의 경우 보다 압축률을 낮춰 품질을 높인다(Figure 3의 Kodim05가 이에 해당). 반면, 압축 후에도 별다른 압축 아티팩트가 눈에 띄지 않는 이미지의 경우에는 보다 압축률을 높일 수 있도록 큰 블록 크기를 사용한다 (Figure 3의 ISCV2_u2_v1이 이에 해당). 압축률에 대한 실험 결과는 다음 절에서 소개한다.

4.3 압축률 비교

Table 2은 각각 6x6 블록 크기 및 본 논문에서 제안하는 방법을 사용하여 압축된 파일들의 크기를 보여준다. 제안하는 방법을 사용할 경우 카테고리별로 압축률이 크게 달라지는데, 8K 해상도의 캡쳐 이미지들에 대해서는 큰 블록 크기를 사용하여 파일들의 크기가 6x6 블록 크기 사용 대비 1/4 수준으로 줄어든 반면, 좀 더 고화질이 요구되거나 저해상도인 GIS 데이터 및 합성 이미지의 경우에는 더 작은 블록 크기를 사용하여 크기가 1.72-1.97배 수준으로 늘어났다. 사진 및 게임 텍스쳐의 경우에는 1.11-1.17배

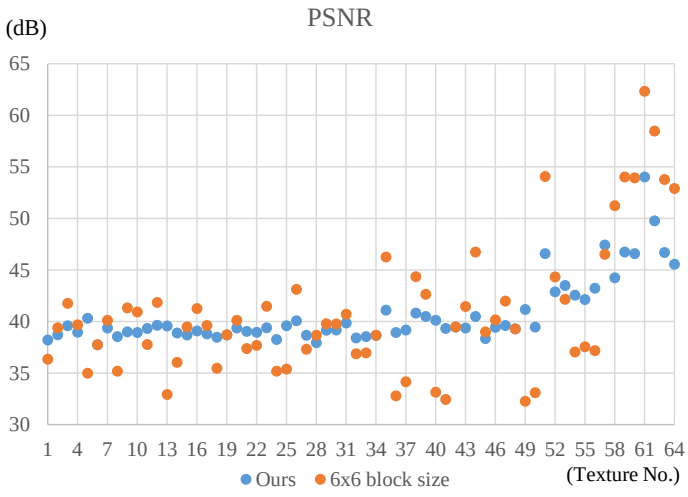


Figure 3: PSNR comparison between images compressed with the 6x6 block size and our approach. This graph shows that the use of our approach results in less deviation than that of a global block size.

수준이였다. 고해상도 텍스쳐에서 많은 용량 절감을 보여준 덕에, 본 논문의 방법은 압축된 파일들의 크기를 전체적으로 0.32배 수준으로 감소시켰다. 이는 해상도가 커지면 커질수록 Unity의 normal mode에 해당하는 6x6 크기를 일괄적으로 선택하는 것보다, 적응적으로 블록 크기를 설정하는 것이 압축률 측면에서 유리함을 보여준다.

Table 2: Comparison of the size of compressed files.

Category (num of tex)	6x6 block size	Ours
Photos (25)	4.1 MB	4.6 MB
Game textures (26)	12.2 MB	14.2 MB
GIS data (4)	531 KB	911 KB
Synthesized iamges (2)	141 KB	278 KB
Captured images (7)	199.3 MB	49.8 MB
Total (64)	216.3 MB	69.8 MB

압축된 파일들의 전체 크기는 디스크 사용량 및 GPU 메모리 사용량과 관련 있는 반면, 요구 메모리 대역폭은 bpp와 관련이 있다. 제안하는 방법 사용시 카테고리별 평균 bpp는 다음과 같다. 사진 텍스쳐는 4.01 bpp, 게임 텍스쳐는 4.15 bpp, GIS 데이터는 5.07 bpp, 합성 이미지는 5.78 bpp, 캡쳐 이미지는 0.89 bpp이다. 평균적으로는 3.85 bpp로, 6x6 블록 크기(3.56bpp)와 비교하여 8% 정도 상승한 수준이다. 평균 bpp 산출은 해상도와 관계없이 각 텍스쳐에 똑같은 비중을 두기 때문에, 앞에서 서술한 압축된 파일들의 용량 비교 결과와는 차이가 있다.

4.4 압축 시간 비교

제안하는 방법은 최적 블록 크기를 검색하기 위한 추가 작업 시간을 필요로 한다. Table 3은 각각 6x6 블록 크기 및 본 논문에서

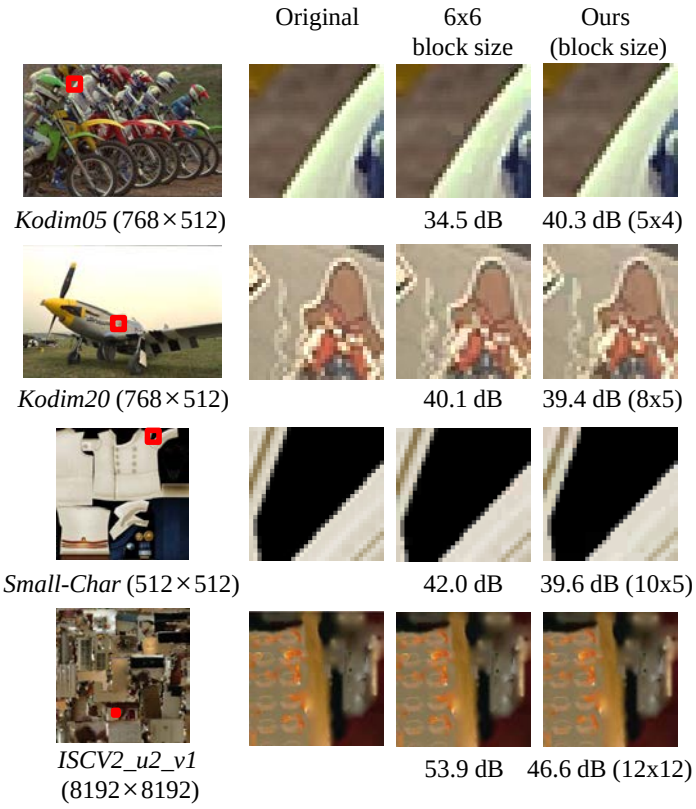


Figure 4: Quality comparison based on PSNR values between the images compressed with the 6x6 block size and our approach. Our approach selects a smaller block size for higher quality if the PSNR value is lower than a target value. In contrast, our approach selects a larger block size for a higher compression rate if the PSNR value is higher than a target value. As a result, our approach can provide more balanced results between compression quality and rates.

제안하는 방법을 사용하여 파일을 압축할 때에 걸린 시간을 보여준다. 본 논문의 방법은 카테고리별로 1.1배-1.9배 더 긴 시간을 소요하였으며, 전체적으로는 1.3배 더 긴 시간을 소요하였다. Brute-force 방식이나 순차 검색 방식을 사용하여 thorough 모드 압축을 수행했을 때 최고 14배까지 압축 시간이 늘어날 수 있음을 생각해 본다면, 본 논문에서 제안하는 방법이 블록 크기 선택을 위한 오버헤드를 최소화하였음을 알 수 있다. 참고로 이 수치는 전처리 과정을 포함하지 않은 것으로, 전처리 과정에서 생성되는 카테고리별 타겟 PSNR값은 한 번 정해지면 여러 앱 개발시 재사용 가능하기 때문이다.

Table 3: Comparison of the encoding time (unit: second).

Category (num of tex)	6x6 block size	Ours
Photos (25)	17.6	26.9
Game textures (26)	44.6	74.0
GIS data (4)	2.0	3.3
Synthesized iamges (2)	0.4	0.8
Captured images (7)	104.3	115.0
Total (64)	169.0	220.0

5. 결론 및 한계

본 논문에서는 텍스쳐별로 PSNR 값을 산출하여 이를 통해 ASTC의 블록 크기를 설정하는 방법을 제안하였다. 이 방법을 사용하면 텍스쳐별 특성에 따라 각기 다른 블록 크기가 지정되기 때문에, 다수의 텍스쳐들에 대해 일괄적으로 블록 크기를 지정하는 방법에 비해 품질 또는 압축률이 적응적으로 향상될 수 있다. 실험 결과, 제안하는 방법은 더 적은 PSNR 편차와 전체적인 압축률 향상을 이루어냈으며, 압축 오버헤드는 30% 가량으로 나타나 실용적으로 사용하기에 크게 무리가 없는 수준으로 나타났다.

본 논문의 가장 큰 한계는 텍스쳐의 압축 품질을 판단하는데 PSNR값만 이용하였다는 점이다. 압축된 결과들 간에 PSNR값 차이가 크게 없더라도 실제로 압축 품질은 크게 차이나는 경우가 있는데, 특정 부위에 나타나는 블록 아티팩트가 대표적인 예이다[11]. 또한 SSIM[23]과 같은 타 품질 측정 메트릭이 PSNR과 다른 결과를 보여주는 경우도 존재하고[24], Lavoué 등[19]의 정신물리학적 실험은 PSNR만으로는 masking 효과를 제대로 측정하기 어려움을 보여주었다. 그러므로, PSNR, SSIM, ꟻLIP[25] 등과 같은 여러 개의 품질 측정 메트릭들을 함께 사용하면 더 적합한 블록 크기를 선정할 수 있을 것으로 보인다. 하지만, PSNR 같은 경우 astcenc에서 내부에서 계산하여 사용하는 데 반해 다른 메트릭들은 그렇지 않다. 그러므로 이와 같은 다중 메트릭을 사용할 시에는 별도의 계산으로 인해 필연적으로 압축 오버헤드가 증가하게 된다. 또한 여러가지 메트릭들로부터 얻어진 결과값들을 어떠한 식으로 조합해서 블록 크기 결정에 적용해야 하는지에 대해서도 명확한 답이 존재하지 않는다. 이와 같이 예상되는 문제점들에 대한 해결 방법은 추후 연구 주제로 남겨 두고자 한다.

아울러 본 논문은 RGB, RGBA 형식의 2D LDR 텍스쳐들에 대해서만 실험을 수행하였다는 한계를 가지고 있다. 향후 좀 더 다양한 형식의 텍스쳐들(컬러 채널 1-2개, 3차원 텍스쳐 등)에 대해서도 최적의 블록 크기를 어떻게 설정할 것인지에 대해 연구를 수행할 수 있을 것이다.

감사의 글

본 연구는 2021학년도 상명대학교 교내연구비를 지원받아 수행하였음. 실험 텍스쳐들은 Kodak, Simon Fenny, Crytek, UNC GAMMA Lab, Spiral Graphics, Vokselia Spawn, Cesium, Google, Sketchfab의 fhernand가 공개한 이미지를 포함함.

References

[1] T. Paltashev and I. Perminov, “Texture compression techniques,” *Scientific Visualization*, vol. 6, no. 1, pp. 106–146, 2014.

[2] Microsoft. (2018) Texture block compression in Direct3D 11. [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/direct3d11/texture-block-compression-in-direct3d-11

[3] J. Ström and T. Akenine-Möller, “iPACKMAN: High-quality, low-complexity texture compression for mobile phones,” in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics Hardware*, 2005, pp. 63–70.

[4] J. Ström and M. Pettersson, “ETC 2: texture compression using invalid combinations,” in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics Hardware*, 2007, pp. 49–54.

[5] S. Fenney, “Texture compression using low-frequency signal modulation,” in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics Hardware*, 2003, pp. 84–91.

[6] J. Nystad, A. Lassen, A. Pomianowski, S. Ellis, and T. Olson, “Adaptive scalable texture compression,” in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on High-Performance Graphics (HPG)*, 2012, pp. 105–114.

[7] D. Chait. (2015) Using ASTC texture compression for game assets. [Online]. Available: https://developer.nvidia.com/astc-texture-compression-for-game-assets

[8] Arm Limited. (2020) Adaptive scalable texture compressioson user guide issue 02 - using ASTC for game assets. [Online]. Available: https://developer.arm.com/documentation/102162/0002/Using-ASTC-for-game-assets

[9] Unity Technologies. (2022) Unity user manual (2022.1) - recommended, default, and supported texture compression formats, by platform. [Online]. Available: https://docs.unity3d.com/2022.1/Documentation/Manual/class-TextureImporterOverride.html

[10] J.-H. Nah, “QuickETC2: How to finish ETC2 compression within 1 ms,” in *ACM SIGGRAPH 2020 Talks*, 2020.

[11] J.-H. Nah, “QuickETC2: Fast ETC2 texture compression using luma differences,” *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH Asia 2020)*, vol. 39, no. 6, 2020.

[12] Google Developers. (2021) About android app bundles. [Online]. Available: https://developer.android.com/guide/app-bundle

[13] Arm Limited. (2022) astc-encoder. [Online]. Available: https://github.com/ARM-software/astc-encoder/

[14] Intel Corp., "Fast ISPC texture compressor," 2021. [Online]. Available: https://github.com/GameTechDev/ISPCTextureCompressor

[15] NVIDIA. (2021) NVIDIA texture tools 3. [Online]. Available: https://developer.nvidia.com/gpu-accelerated-texture-compression

[16] D. Oom, "Real-time adaptive scalable texture compression for the web," Master's thesis, Chalmers University of Technology., 2016.

[17] S. Pratapa, T. Olson, A. Chalfin, and D. Manocha, "TexNN: Fast texture encoding using neural networks," *Computer Graphics Forum*, vol. 38, no. 1, pp. 328–339, 2019.

[18] W. Griffin and M. Olano, "Evaluating texture compression masking effects using objective image quality assessment metrics," *IEEE Transactions on Visualization and Computer Graphics*, vol. 21, no. 8, pp. 970–979, 2015.

[19] G. Lavoué, M. Langer, A. Peytavie, and P. Poulin, "A psychophysical evaluation of texture compression masking effects," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 2, pp. 1336–1346, 2019.

[20] Binomial LLC. (2022) Basis Universal Supercompressed GPU Texture Codec. [Online]. Available: https://github.com/BinomialLLC/basis_universal

[21] M. Callow. (2021) KTX file format specification - version 2.0. [Online]. Available: https://www.khronos.org/registry/KTX/specs/2.0/ktxspec_v2.html

[22] D. Bull and F. Zhang, *Intelligent image and video compression: communicating pictures*, 2nd ed. Elsevier, 2021.

[23] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[24] P. Krajcevski and D. Manocha, "Fast PVRTC texture compression using intensity dilation," *Journal of Computer Graphics Techniques*, vol. 3, no. 4, pp. 132–145, 2014.

[25] P. Andersson, J. Nilsson, T. Akenine-Möller, M. Oskarsson, and K. Åströmand Mark D. Fairchild, "ꟻLIP: A difference evaluator for alternating images," *Proceedings of the ACM on Computer Graphics and Interactive Techniques (HPG 2020)*, vol. 3, no. 2, 2020.

〈저자소개〉

나재호

- 2005 연세대학교 컴퓨터산업공학부 학사
- 2007 연세대학교 컴퓨터과학과 석사
- 2012 연세대학교 컴퓨터과학과 박사
- 2012 세종대학교 컴퓨터공학과 방문연구원
- 2012~2013 University of North Carolina at Chapel Hill Visiting Researcher
- 2014~2021 LG전자 선임연구원, 책임연구원
- 2021~현재 상명대학교 컴퓨터과학과 조교수
- 관심분야: 광선 추적법, 렌더링 알고리즘, GPU 구조 등
- https://orcid.org/ 0000-0001-7805-5333