실시간 렌더링을 위한 신경망 기반 텍스처 압축의 성능 및 실용성 분석

김도형, 나재호O†
상명대학교 컴퓨터과학과

Performance and Practicality Analysis of Neural Texture Compression in Real-Time Rendering

Dohyung Kim, Jae-Ho NahO†
Department of Computer Science, Sangmyung University
202110907@sangmyung.kr jaeho.nah@smu.ac.kr

요 약

현대 실시간 렌더링 파이프라인에서 4K 이상의 초고해상도 텍스처 수요 증가와 물리 기반 렌더링(PBR)의 보편화는 GPU 메모리 용량 및 대역폭에 상당한 부담을 초래하고 있다. 이에 대한 대안으로 신경망 기반 텍스처 압축(Neural Texture Compression, 이하 NTC) 기술이 제안되었으나, 실제 상용 개발 환경에서의 성능 특성을 반영한 실무 운용 가이드라인은 아직 충분히 제시되지 않고 있다. 이에 본 연구는 NVIDIA NTC SDK 를 활용하여 NTC 기술의 실무적 효용성을 검증하였다. 먼저 네트워크 모델 복잡도와 텍스처 해상도에 따른 성능 및 화질 변화를 분석하고, 이어서 저채널 환경에서의 화질과 BC 포맷으로의 트랜스코딩 효율을 PSNR 및 FLIP 지표를 통해 비교·분석하였다. 분석 결과, NTC 는 복잡한 머티리얼 표현이 요구되는 차세대 그래픽스 환경에서 산업 표준인 BC 포맷의 유력한 대안이 될 수 있는 잠재력을 보였다. 그러나 실시간 고해상도 텍스처 매핑 환경에 적용하기 위해서는 디코딩 오버헤드 감소를 위한 추가적인 알고리즘 개선과 하드웨어 가속 기술이 요구된다.

Abstract

The increasing demand for ultra-high-resolution textures (4K and above), together with the widespread adoption of Physically Based Rendering (PBR), has created significant bottlenecks in GPU memory capacity and bandwidth within modern real-time rendering pipelines. As a potential solution, Neural Texture Compression (NTC) has been proposed; however, practical guidelines that characterize its performance in real-world production environments remain limited. This study evaluates the practical effectiveness of NTC using the NVIDIA NTC SDK. First, we investigate the impact of network model complexity and texture resolution on performance and visual quality. We then analyze image quality in low-channel scenarios and transcoding efficiency into BC formats using PSNR and FLIP metrics. The results indicate that NTC has strong potential as an alternative to industry-standard BC formats in next-generation graphics applications that require complex material representation. However, reducing decoding overhead through further algorithmic improvements and hardware acceleration remains essential for real-time high-resolution texture mapping.

키워드: 신경 텍스처 압축, 실시간 렌더링, 트랜스코딩,

Keywords: Neural Texture Compression, Real-time Rendering, Transcoding,

† 교신저자(corresponding author)

* 학부생 주저자 논문임

1. 서론

현대의 실시간 렌더링 환경에서는 4K 이상의 초고해상도 텍스처와 물리 기반 렌더링(Physically Based Rendering, PBR)을 위한 복잡한 머티리얼(material) 표현에 대한 수요가 지속적으로 증가하고 있다. 이러한 텍스처 데이터 용량의 급증은 GPU 메모리 용량에 큰 부담을 줄 뿐만 아니라, 데이터를 처리하는 메모리 대역폭에 심각한 부하를 주어 전체 렌더링 파이프라인의 주요 병목 현상으로 작용할 가능성이 있다.

위 문제를 해결하기 위해 지난 수십여 년간 블록 기반의 텍스처 압축 방식이 널리 활용되어 왔다. 이는 텍스처 이미지를 4x4 픽셀과 같은 고정된 크기의 블록 단위로 압축하는 기법들로, 데스크탑에서는 마이크로소프트의 BC(Block Compression) 포맷[33]이, 모바일에서는 ETC1/2[34,35], ASTC[36], PVRTC[37]와 같은 포맷이 표준으로 자리잡아 왔다. 텍스처 압축은 일반적인 JPEG 과 같은 이미지 압축과 유사한 손실 압축 기법이지만, 임의의 텍스처 좌표에 즉시 접근 가능한 랜덤 액세스(random access)를 지원해야 하고, GPU 내 하드웨어 디코더를 통해 실시간으로 압축 해제가 가능하다는 특징을 지닌다. 또한 ASTC 를 제외한 대부분의 표준 텍스처 압축 포맷은 제한된 수의 고정 비트레이트만 제공한다.

이러한 고정 압축률은 화질과 압축률 간 트레이드오프를 적절히 조절하지 못한다는 단점을 지닌다. 화질 측면에서는, 고주파 디테일이 집중된 영역에서 정보가 손실되거나 색상 변화가 급격한 영역에서 블록 간의 경계면이 시각적으로 부각되는 '블록 아티팩트 (block artifacts)' 현상이 유발될 수 있다. 다른 한편으로, 유사한 색상이 반복되는 영역이 많은 이미지에서는 JPEG 과 같은 표준 이미지 압축에 비해 불필요하게 많은 정보를 유지하여 낮은 압축 효율로 이어질 수 있다. 또한, 앞서 언급한 표준 텍스처 압축 포맷들은 모두 텍스처 당 채널 수를 1-4 개로 제한하기 때문에, 알베도(albedo), 노멀(normal), 스페큘러(specular), 거칠기(roughness) 등 여러 채널로 구성된 PBR 머티리얼을 압축할 경우 텍스처 맵들 간의 유사성을 적절히 활용하지 못하는 한계를 지니며, 이는 낮은 압축 효율 또는 화질 저하로 이어질 수 있다.

최근 딥러닝 기술이 발전하면서 위에서 언급한 한계를 극복할 새로운 대안으로 신경망 기반 텍스처 압축 기법들이 새롭게 제안되고 있다. 그 중 가장 대표적인 기법은 엔비디아의 NTC (Neural Texture Compression) [1]로, 텍스처 데이터를 픽셀 배열이 아닌 MLP(multi-layer perceptron) 신경망의 가중치(weight)로 표현하는 기법이다. 이 방법은 밉맵(mipmap)과 멀티 채널 텍스처들을 함께 압축 가능하고, 가변 압축률을 지원하며, 기존 신경망 기반 이미지 압축 기법들과 달리 랜덤 액세스를 지원한다. 실험 결과, PBR 머티리얼을 NTC 로 압축 시 낮은 비트레이트(bit rate)에서도 원본에 가까운 디테일을 복원할 수 있는 가능성을 제시하였다. 또한 엔비디아는 NTC 를 개발자들이 프로젝트에 바로 적용할 수 있도록 SDK[3] 또한 함께 공개하고 있다.

이러한 기술적 진보와 SDK 의 편의성 개선에도 불구하고, 실제 상용 게임 개발 환경에 NTC 를 도입하기 위해서는 여전히 해소되지 않은 실무적 의문점들이 존재한다. 관련 논문[1], 발표 자료[2], SDK 문서[3]는 주로 기술의 구조적 우수성과 최신 하드웨어에서의 성능 검증에 집중하고 있어, 다양한 GPU 환경과 레거시 시스템이 공존하는 실제 개발 환경에서 활용할 수 있는 구체적인 운용 가이드라인은 충분히 제시하지 못하고 있기 때문이다. 이에 본 연구는 NTC SDK 를 기반으로, 실무 도입 시 발생하는 불확실성을 해소하기 위해 다음과 같은 세 가지 핵심 요소를 검증한다.

첫째, MLP 모델의 복잡도에 따른 화질과 디코딩 성능의 트레이드 오프(trade-off)를 정량적으로 규명한다. 기존 표준 텍스처 코덱과 달리 NTC 는 하드웨어 디코딩이 불가하므로, NTC 로 압축된 텍스처를 사용하면 추론을 위한 디코딩 오버헤드가 런타임 시 추가될 수밖에 없다. NTC SDK 는 MLP 레이어 구성 및 파라미터 규모를 조절할 수 있는 기능을 제공하지만, 이것이 저사양 GPU 에서 어느 정도의 성능 이득을 주는지, 혹은 화질 저하의 임계점이 어디인지에 대한 정량적 기준은 부재하다. 본 연구는 신경망의 레이어 깊이와 파라미터 수가 압축 품질과 추론 속도에 미치는 영향을 분석하여, 하드웨어 가속 지원 유무 및 시스템 사양에 따른 최적의 네트워크 구성 가이드라인을 제시한다.

둘째, 텍스처 채널 복잡도에 따른 압축 효율 변화를 분석한다. 선행 NTC 연구[1]는 다중 채널 PBR 머티리얼의 채널 간 상관관계 학습이 높은 압축 효율로 이어질 수 있음을 입증하였으나, 소수의 채널만을 사용하는 일반적인 환경에서의 효율성은 자세히 검증하지 않았다. 이에 따라 본 연구에서는 이러한 저채널 텍스처 환경에서, 즉 학습할 수 있는 상관관계 정보가 제한된 상황에서도 NTC 가 여전히 기존 압축 방식 대비 유효한 화질 우위를 점하는지 실험적으로 검증한다.

셋째, 하드웨어 호환성 확보를 위한 'NTC-to-BCn 트랜스코딩'의 화질 효율성과 실무적 타당성을 검증한다. 앞서 언급한 런타임 단계에서 수행되는 추론은 프레임 레이트 저하를 유발할 수 있으며, 이러한 특성은 연산 성능이 떨어지는 레거시 시스템에서의 NTC 사용을 특히 어렵게 만든다. 이에 따라 NTC SDK [3]는 런타임이 아닌 애플리케이션

로딩 시 사용할 수 있는, NTC 를 BCn 과 같은 표준 코덱으로 트랜스코딩하는 기능을 제공한다. 이 기법을 적용하면 VRAM 사용량 증가를 감수하는 대신 런타임 오버헤드가 제거되므로 호환성 문제 해결에 도움이 된다. 하지만 이와 같이 트랜스코딩 단계를 추가하게 되면 이중 압축 손실(generation loss)로 인한 추가적인 화질 저하가 발생할 수 있으며, 앱 로딩 시 트랜스코딩 연산 시간이 추가된다. 선행 연구[2]에서는 이러한 성능 및 품질과 관련하여 실험 평균 결과 수치를 제시했으나, 본 연구에서는 PSNR 및 FLIP[31]의 두 가지 메트릭을 통해 트랜스코딩 결과물의 화질을 각 텍스처별로 보다 정밀하게 측정한다. 또한 이를 통해 트랜스코딩으로 인한 이중 압축 손실과, NTC 의 압축률 설정으로 인한 화질 저하 두 가지를 함께 분석한다. 이러한 분석 결과는 향후 단순한 호환성 확보를 넘어 실제 상용 엔진에서의 NTC 적용 가능성을 판단하는 데 도움이 될 것이다.

2. 관련 연구

2.1 전통적인 텍스처 압축 기술 및 연구 동향

실시간 렌더링 파이프라인에서 텍스처 메모리 대역폭은 성능을 결정짓는 주요 병목 구간 중 하나이다. 이를 해결하기 위해 지난 수십 년간 블록 기반 텍스처 압축이 표준으로 자리잡아 왔다. 그 중 마이크로소프트의 BC, 에릭슨의 ETC, ARM 의 ASTC, 이매지네이션 테크놀러지의 PVRTC 와 같은 코덱들은 이미지를 일정 크기의 작은 블록으로 나누고, 각 블록을 고정된 비트레이트로 압축하여 GPU 하드웨어에서 랜덤 액세스(random access)가 가능하도록 설계되었다. 초기의 BC1(DXT1, S3TC[4])은 블록당 두 개의 대표 색상을 저장하고 이를 보간하여 픽셀을 표현하는 단순한 방식을 사용했으나, 현대의 렌더링은 더 높은 품질을 요구함에 따라 이후 BC7, ETC2, ASTC 와 같은 고품질 포맷이 등장했다. 이러한 포맷들은 다양한 모드와 블록 내 다중 파티션을 지원하여 색상 손실을 최소화하고 높은 압축 품질을 제공하는 데 초점을 두고 있다.

각 GPU 내에는 위 표준 코덱들 중 적어도 하나 이상에 대한 디코더가 존재하여 텍스처 매핑 시 실시간 디코딩을 지원한다. 디코더가 이와 같이 하드웨어상에 이미 구현되어 있는 것과 달리, 인코딩은 소프트웨어로 수행된다. 따라서 어떤 인코딩 알고리즘을 사용하느냐에 따라 압축된 결과의 품질과 인코딩 속도 간에 매우 큰 차이가 발생할 수 있다. 이러한 측면에서 텍스처 압축 분야의 많은 기존 연구들은 품질을 최대한 유지하면서도 인코딩 속도를 증가시켜 소프트웨어 개발성을 향상시키는 데 집중해 왔다. 그 예로는 BC7 인코딩을 위한 FasTC[5], SegTC[6], bc7enc[7], bc7e[8], QuickBC7[39], ETC 인코딩을 위한 etcpak[9], etc2comp[10], QuickETC2[11,38], QuickETC2-HQ[12], H-ETC[13], PVRTC 인코딩에 사용되는 Intensity Dilation 기법[14], ASTC 인코딩을 위한 astcenc[15], TexNN[16] 등이 있다. 이러한 고속 인코더는 대용량 텍스처를 포함하는 소프트웨어 개발 시간 절감에 도움이 될 뿐 아니라, 일부 실시간 텍스처 압축이 필요한 시나리오(3D 재구축, 텍스처 스트리밍, 실시간 절차적 텍스처 생성 등)에서도 유용하게 사용 가능하다.

ASTC 는 128-bit 블록 구조를 기반으로 다양한 블록 footprint(예: 4×4~12×12)를 선택할 수 있어, 여러 수준의 가변 비트레이트 압축을 지원한다. 이에 따라 ASTC 에서는 최적의 블록 크기 설정을 위한 몇 가지 연구들이 추가적으로 수행되었다. Griffin 과 Olano[17], Lavoué 등[18]은 PBR 머티리얼에서 각 텍스처별 블록 크기를 마스킹(masking) 효과를 고려하여 다르게 지정해야 함을 주장하였고, Nah[19][20]는 타겟 PSNR 수치에 따라 블록 크기를 다르게 설정 가능한 기법을 제안하였다.

마지막 연구 주제는 이미 압축된 텍스처를 재압축하여 압축률을 향상시키는 초압축(supercompression)이다. Strom 등[21]은 ETC2 로 이미 압축된 텍스처를 손실 없이 재압축하여 압축률을 더욱 향상시켰다. 이후 더 높은 압축률을 위해 별도의 코덱으로 사전 압축한 뒤, CPU (Crunch[22], Basis Universal[23]) 또는 GPU (GST[24]) 기반 방식을 통해 표준 코덱으로 트랜스코딩하는 손실 초압축 기법들이 제안되었다. 이와 같은 초압축 기법은 VRAM 사용량에는 영향을 주지 않지만, 스토리지 사용량과 다운로드 시 네트워크 대역폭을 줄이는 데 효과적이다. 다만 앱 로딩 시 디코딩 또는 트랜스코딩을 위한 추가 오버헤드가 발생한다.

2.2 신경망 기반 텍스처 압축의 등장 및 발전

위와 같이 기존 텍스처 압축 연구들이 주로 표준 코덱들(BCn, ETC, PVRTC, ASTC)의 사용성 향상에 초점을 맞춘 반면, 엔비디아의 NTC 는 신경망 기반의 완전히 새로운 코덱을 제안하는 연구이다. 최근 딥러닝 기술의 발전은 JPEG 나 AVIF 를 뛰어넘는 신경망 기반 이미지 압축 기술들[25][26]의 등장으로 이어져 왔다. 하지만 이러한 기법들은 텍스처 압축에 직접 적용하기는 어려웠는데, 그 이유는 전체 이미지를 한 번에 복원하거나 엔트로피 코딩을 사용함으로써 텍스처 매핑에 필수적인 랜덤 액세스가 불가능하였기 때문이다. 이와 달리 NTC 는 기존 신경망 이미지 압축 기법들과 달리 실시간 렌더링에 적합한 구조로 제안되어, 기존 텍스처 압축 기법들의 압축률 한계를 극복하는 데 초점을 두고 있다.

NTC 의 핵심 개념 중 하나는 머티리얼별 과적합으로, 특정 머티리얼의 전체 텍스처 맵들을 하나의 작은 신경망(MLP)에 학습시켜, 신경망 자체가 압축된 데이터 역할을 수행하도록 한다. 데이터는 특징 피라미드(feature pyramid) 형태로 저장되며, 렌더링 시 셰이더 내부에서 필요한 픽셀 위치의 특징(feature)을 읽어와 MLP 를 통해 실시간으로 압축을 해제한다. 이 방식은 신경망이 텍스처 채널 간의 잠재적 상관관계를 학습하여 데이터의 중복을 근본적으로 제거함으로써, 기존 블록 압축 방식 대비 메모리 점유율을 대폭 절감하면서도 시각적으로 훨씬 정교하고 고밀도의 디테일을 표현하는 탁월한 압축 효율을 보여준다. 즉, NTC 는 단순한 압축을 넘어, 고해상도 디테일을 저용량으로 실시간 렌더링할 수 있는 새로운 패러다임을 제시하였다.

NVIDIA 에서 NTC 를 발표한 이후, NVIDIA 를 포함한 주요 하드웨어 제조사 및 게임 개발사에서는 신경망 기반 압축 기술을 기존 렌더링 파이프라인에 통합하거나 하드웨어 가속을 통해 실용성을 높이는 방향으로 연구를 확장하고 있다. 우선, NVIDIA 는 SDK[3] 발표와 함께 SDK 상에 구현된 NTC 의 개선사항을 Vulkanised[2]에서 발표하였다. 이는 고해상도 특징 맵에서의 선형 가중치 사용, 텍스처 채널의 정규화, 활성화 함수의 범위 제한 등을 통해 품질 향상이 이루어졌고, [19][20]과 유사하게 타겟 PSNR 수치 입력이 가능해졌으며, 협동 벡터(cooperative vectors)를 통한 추론 속도의 개선과 CUDA 최적화를 통한 압축 속도 향상, 타 표준 코덱으로의 트랜스코딩 지원 등 다양한 기능 추가 및 성능 향상이 이루어졌다.

AMD 에서는 기존 하드웨어와의 호환성을 극대화하려는 연구인 Neural Texture Block Compression (NTBC)[27]을 발표하였다. 이는 셰이더나 하드웨어 구조를 변경하지 않고도 압축 효율을 높이는 데 초점을 맞추고 있다. 이 연구는 신경망을 이용해 압축된 데이터를 표준 블록 압축(BC) 포맷으로 변환하는 최적의 매핑을 학습하며, 이를 통해 NTC 트랜스코딩과 마찬가지로 런타임 오버헤드 없이 저장 공간을 절약할 수 있다.

Ubisoft 는 신경망의 특징 벡터를 기존 GPU 가 지원하는 BC6H 와 같은 압축 포맷에 저장하는 방식을 도입하였다[28]. 이를 통해 하드웨어의 텍스처 페치(texture fetch) 및 필터링 기능을 그대로 활용하면서, 셰이더 내부의 경량 디코더를 통해 실시간으로 고해상도 머티리얼을 복원하는 기법을 입증하였다.

하드웨어 레벨에서의 가속을 통한 렌더링 성능 최적화 연구도 활발하다. Intel 은 Intel Xe Matrix Extensions(XMX)를 활용하여 신경망 압축의 디코딩 비용을 줄이는 TSNC[29](Texture Set Neural Compression)를 발표하였다. 이 논문의 저자들은 Ubisoft 와 달리 BC1 포맷을 사용하고, 협동 벡터를 이용한 하드웨어 가속 기법을 통해 고품질의 신경망 텍스처 압축을 비등방성 필터링(anisotropic filtering)과 함께 실시간 성능으로 구현할 수 있음을 보였다.

마지막으로, Qualcomm 은 모바일 및 엣지 디바이스 환경을 고려하여 랜덤 액세스가 가능한 비대칭 오토인코더(asymmetric auto-encoder) 프레임워크를 제안하였다[30]. 이들은 인코더에 컨볼루션 계층을 사용하여 압축 효율을 높이는 한편, 디코더는 병렬 렌더링에 적합하도록 설계하여 기존 방식 대비 우수한 압축률과 화질을 달성하였다.

이와 같이 산업계에서는 단순히 신경망을 적용하는 것을 넘어 여러 가지 측면에서 기술을 고도화하고 있다. 이는 기존 하드웨어 포맷과의 호환성 확보 및 전용 연산 유닛을 통한 디코딩 가속, 새로운 네트워크 구조를 통한 품질 향상 등을 포함한다.

3. 실험 환경 및 방법

3.1. NTC 의 개요

NTC 는 BCn 과 같은 기존의 블록 압축 방식들과 달리, 텍스처 데이터를 신경망의 가중치와 학습된 특징 피라미드 형태로 저장하는 혁신적인 방식을 취한다. 본 연구의 실험에 앞서, NTC 의 핵심 작동 원리를 Figure 1 을 통해 살펴보면 다음과 같다.

NTC 구조는 기능적으로 크게 데이터 표현을 담당하는 '특징 피라미드(feature pyramid)'와 복원을 담당하는 '신경망 디코더(MLP decoder)' 두 부분으로 구성된다. 첫째, 텍스처 데이터를 효율적으로 저장하기 위한 특징 피라미드 구조를 살펴보면 다음과 같다. Figure 1 의 a)와 같이 원본 텍스처는 밉맵과 유사한 계층적 구조를 가진 특징 격자(feature grid)로 변환된다. 일반적인 텍스처가 픽셀마다 RGB 값을 저장하는 것과 달리, NTC 는 각 격자점(grid cell)에 압축된 고차원 특징 벡터(feature vector)를 저장한다. 이 과정에서 b) 단계인 '양자화 시뮬레이션(simulated quantization)'이 수행되는데, 이는 추후 데이터를 압축(양자화)하여 저장할 때 발생하는 정보 손실에 대해 신경망이 강건함을 갖게 하여, 화질 저하를 최소화하면서도 원본을 효과적으로 복원할 수 있도록 훈련하는 핵심 과정이다. 이후 렌더링 시 c) 단계에서 보듯이, 텍스처 좌표에 인접한 특징 벡터들을 샘플링하고 이를 셰이더 내부에서 선형 보간(linear interpolation)하여 해당 위치의 대표 특징값을 추출한다. 이러한 소프트웨어적 필터링 방식은 기존 하드웨어 필터링을 모사하여 병렬 연산에 최적화된 랜덤 액세스를 지원한다.

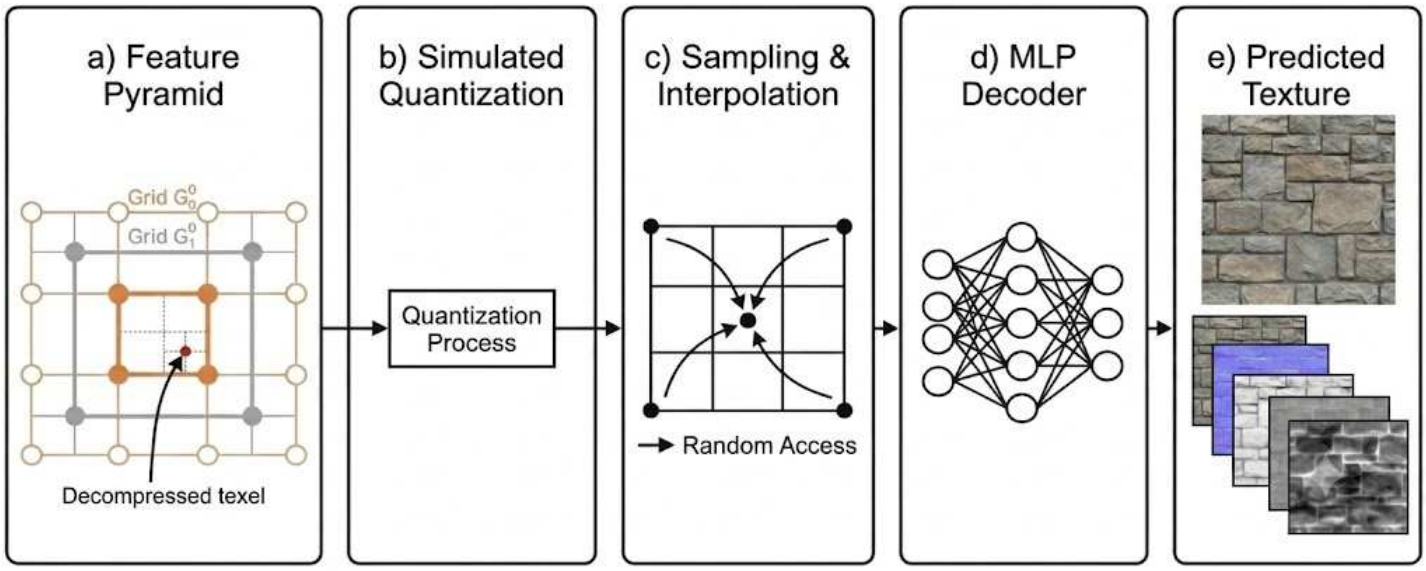


Figure 1. The Overall Architecture and Processing Pipeline of Neural Texture Compression

둘째, 추출된 특징을 시각 정보로 변환하는 MLP 디코더 구조이다. 상기 과정에서 추출된 특징 벡터는 d) 단계의 MLP 로 전달된다. 이 MLP 는 입력받은 특징 정보를 연산하여 최종적으로 e)와 같은 고품질 텍스처 값을 실시간으로 추론한다. 즉, NTC 에서 압축은 시각적 정보를 특징 피라미드 형태로 변환하여 저장하는 과정이며, 학습(training)을 통해 이를 복원하는 MLP 가 최적화된다.

특히 본 연구에서 주목하는 부분은 d) network 단계의 MLP 구조이다. 이 신경망의 은닉층(hidden layer) 깊이는 텍스처의 복원 품질과 연산 속도를 결정짓는 가장 중요한 하이퍼파라미터(hyperparameter)이다. 레이어가 깊어질수록 더 정교한 복원이 가능하지만 연산 비용이 증가하는 트레이드오프가 발생하며, 본 연구는 이를 실험적으로 검증하여 최적의 구성을 찾고자 한다.

3.2. 실험 설계 및 평가 전략

본 연구는 NTC 의 실무적 효용성을 검증하기 위해 네트워크 구조 최적화, 압축 품질 비교 우위 검증, 그리고 하드웨어 호환성을 위한 트랜스코딩 효율성의 세 가지 차원에서 체계적인 실험을 수행한다.

3.2.1. MLP 모델 복잡도에 따른 최적화 및 성능 트레이드 오프 분석

NTC 디코더의 핵심인 MLP 의 구조적 복잡도는 추론 연산 비용과 직결되는 핵심 변수이다. 본 실험에서는 먼저, 실시간 렌더링 환경에서의 실무적 효율성을 검증하기 위해 NTC SDK 에서 제공하는 세 가지 아키텍처 프리셋(Small, Medium, Large)에서 화질과 성능 간의 균형점을 규명한다. 기본값인 Medium 과 비교하여 Small 은 더 적은 레이어 수를, Large 는 더 많은 레이어 수를 의미한다.

이 실험의 목표는 모델의 복잡도가 Small 에서 Large 로 증대됨에 따라 발생하는 학습 시간 및 추론 지연의 증가폭과 그에 상응하는 화질 개선폭의 상관관계를 정량적으로 분석하는 데 있다. 이를 통해 제한된 연산 자원내에서 가장 높은 효율을 보이는 '최적 성능 구간'을 식별하고, 하드웨어 사양에 따른 적정 모델 선정 가이드라인을 도출한다.

비교의 공정성을 위해 모든 모델의 학습 스텝 수는 100k steps 로 고정하였다. 또한 표면 거칠기, 패턴의 규칙성, 그리고 주파수 특성이 서로 다른 6 종의 머티리얼 세트(Table 1)를 벤치마크로 사용하여 평가의 객관성을 확보하고자 하였다. 앞의 5 개 세트는 NTC 논문의 실험에 사용된 머티리얼 중 일부로, 전체 실험 머티리얼 가운데 온라인에 공개되어 활용 가능한 5 종을 선정하였다. 아울러 8K 해상도에서의 실험을 보강하기 위해 ambientCG 의 Rock064 를 추가로 사용하였다. 화질 비교를 위해서는 NTC SDK 에서 제공하는 PSNR(Peak Signal-to-Noise Ratio)을 사용하였으며, 압축률 설정에는 기본값인 2.06 BPP 를 적용하였다.

3.2.2. 저채널 머티리얼에서의 압축 화질 분석

NTC 논문[1] 실험 결과에 따르면 채널 수가 높은 머티리얼에서 더 높은 압축 효율이 관찰되었다. 이는 고채널 PBR 텍스처 압축에 유리한 특성으로 해석된다. 하지만 2D 환경이나 캐주얼 게임, 웹 등에서는 PBR 텍스처가 아닌 컬러 텍스처 맵만 사용하거나, 부가적으로 노멀 맵 정도만 사용하는 경우도 빈번하다. 이와 같은 일반적인 저채널 머티리얼 압축 시에도 NTC 가 산업 표준 중 하나인 BC7 대비 제공하는 이점에 대해서는 선행 연구에서 명확한 결과를 제시하지 않았다.

이에 따라, 본 실험에서는 BC7 과 동일하게 NTC 의 단일 텍스처 맵 당 픽셀당 비트 수(bits per pixel, bpp)를 8bpp 로 설정하였다. 또한 일반적인 diffuse map 만 사용한 경우(8bpp)와 normal map 을 함께 사용하는 경우(전체 16bpp)에 대해서, 화질이 어떻게 달라지는지 확인한다. 이 실험에서는 서로 다른 코덱을 비교하는 것이기 때문에, 전통적인 PSNR 에 더해 지각적 화질 평가를 위한 FLIP 알고리즘[31]을 도입하였다. 이를 통해 수치적으로 드러나지 않는 경계면의 아티팩트나 고주파 영역의 블러링 현상을 시각화하여, 인간 시각 시스템 관점에서의 실질적 우위를 평가하고자 한다.

3.2.3. 하드웨어 호환성을 위한 트랜스코딩 화질 검증

하드웨어 호환성 또는 런타임 오버헤드 감소를 위해 NTC 데이터를 BC7 으로 변환할 경우, 이에 따라 필연적으로 발생하는 화질 저하를 실험한다. 이는 NTC 에서 높은 압축률을 설정할시에 생기는 화질 저하와, 트랜스코딩으로 생기는 2 차 화질 저하로 구분된다. 이 실험을 위해서는 첫번째 실험과 마찬가지로 원본 텍스처를 기본 압축률(머티리얼당 2.06 BPP)로 NTC 형태로 압축한 후, 이를 컬러/노멀은 BC7(텍스처당 8 BPP)으로, 단일채널 텍스처는 BC4(텍스처당 4 BPP)로 변환하였다. 결과 분석시에는 NTC 와 BC7 트랜스코딩 후의 PSNR 및 FLIP 변화값을 살펴봄으로써 이중 압축 손실량이 얼마나 되는지 확인하고, 또한 컬러 및 노멀의 경우 앞서 BC7 형태로 인코딩했을 때와 화질 차이가 얼마나 나는지 살펴보고자 한다.

4. 실험 결과 및 분석

본 장에서는 앞서 설계한 세 가지 실험 시나리오를 바탕으로 NTC SDK 의 성능을 정량적으로 분석하고, 실무 파이프라인 적용 시의 실효성을 검증한다. 본 연구의 모든 성능 평가는 NTC 의 협동 벡터 하드웨어 가속을 지원하는 GPU 환경(NVIDIA GeForce RTX 3080)에서 수행되었다. 텍스처의 인코딩 및 디코딩 속도(ms) 측정은 Windows 환경에서 NVIDIA NTC SDK 0.8 을 활용하여 진행하였고, 디코딩 시에는 Vulkan 백엔드를 사용하여 협동 벡터 사용 여부에 따른 결과를 각각 측정하였다. 비교 대상으로 사용한 BC7 압축을 위해서 Rock064 에서는 bc7e[8]의 최고 품질 옵션(u6)을, 나머지 머티리얼 세트에서는 NVTT3[32]의 Production 모드를 이용하였다.

4.1. 모델 크기 및 해상도에 따른 디코딩 성능 분석

NTC SDK 는 사용자의 하드웨어 환경과 요구 품질에 맞춰 네트워크의 용량을 조절할 수 있도록 다양한 모델 프리셋(preset)을 제공한다. 본 절에서는 세 가지 다른 모델 크기(Small, Medium, Large)와 서로 다른 해상도(2K 대 8K)의 6 개 텍스처를 이용하여 이러한 요인들이 실시간 추론 속도와 화질에 미치는 영향을 확인하였으며, 그 결과는 Table 2 와 같다. 이 결과가 의미하는 바는 다음과 같다.

첫째, 모델 크기에 따른 확연한 성능 트레이드 오프가 확인되었다. 2K 해상도 텍스처 그룹(PavingStones, MetalPlates, Tiles)의 경우, Small 모델은 협동 벡터 미사용 시 3ms 내외, 협동 벡터 사용 시 1.2ms 내외의 매우 빠른 디코딩 속도를 기록하여 실시간 렌더링이 가능한 성능을 보였다. 이를 Medium 모델로 교체할 경우 PSNR 은 평균 약 +1.1dB 상승하여 화질 개선 효과가 있었으나, 디코딩 및 인코딩 비용 또한 증가하는 경향을 보였다. 반면, 2K 와 8K 해상도 모두, Medium 모델과 Large 모델 간에는 유의미한 화질 차이는 관찰되지 않았으나 인코딩과 디코딩 시간은 늘어났다. 다만 협동 벡터 사용 시 디코딩 오버헤드 차이는 미사용 시 대비 다소 감소하였다.

위 실험 결과에 따르면, 일반적인 환경에서는 Medium 모델을 통해 화질과 성능 간의 균형을 확보하고, 성능 제약이 있거나 초고해상도 텍스처를 사용하는 환경에서는 Small 모델을 사용하는 전략이 유효하다고 판단된다. Large 모델과 같이 단순히 레이어 수와 파라미터 수를 증가시키는 것만으로는 화질 향상에 제약이 있으므로, 전용 하드웨어 디코더 설계시에도 이와 같은 단순한 접근 방식은 배제할 필요가 있다.

텍스처 해상도 증가에 따른 오버헤드 분석 결과, 8K 급 초고해상도 텍스처들의 디코딩 소요 시간은 협동 벡터 미사용 시 50 ms 내외, 협동 벡터 사용 시 18 ms 수준으로 측정되었다. 이는 2K 텍스처(4~6 MB) 대비 8K 텍스처(46 MB)의 데이터량이 약 10 배 이상 증가함에 따라 디코딩 연산 부하 역시 크게 증가했음을 의미한다. 일반적인 실시간 렌더링 환경에서 60 FPS 에 도달하기 위한 프레임 시간이 16.6 ms 이하임을 감안할 때, 텍스처 디코딩에만 50 ms 이상을 소모하는 것은 치명적인 성능 저하를 야기한다. 협동 벡터를 사용하면 디코딩 오버헤드가 약 1/2.5~1/3 수준으로 감소하므로, 신경망 기반 텍스처 압축을 실시간 렌더링에 활용하기 위해서는 AI 가속 유닛과 협동 벡터를 지원하는 GPU 및 API 환경이 사실상 필수적이다. 하지만 여전히 단일 머티리얼에 18 ms 수준의 디코딩 시간이 필요하다는 것은, 다수의 초고해상도 NTC 텍스처 사용이 심각한 프레임 레이트 저하로 이어질 수 있다는 것을 의미한다. 따라서 현재의 NTC 알고리즘을 실무 애플리케이션에 적용하기 위해서는 여전히 디코딩 오버헤드를 줄이기 위한 추가 최적화가 필요하다.

Table 1: Material sets used in our experiments

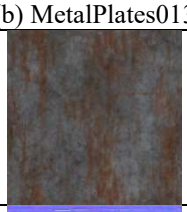
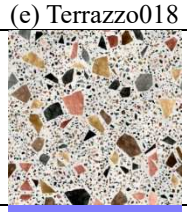
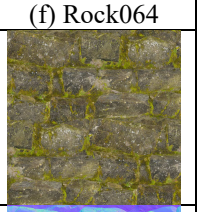
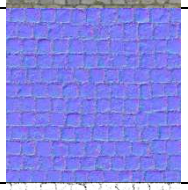
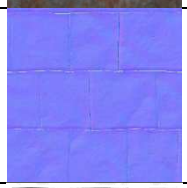
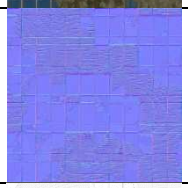
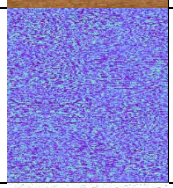
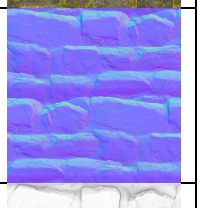
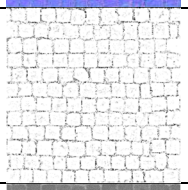
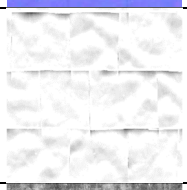
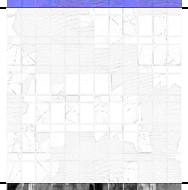
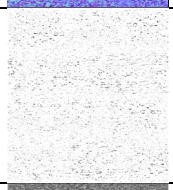
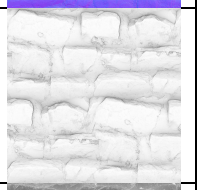
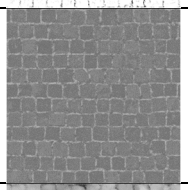
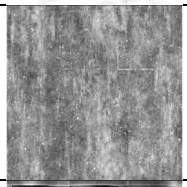
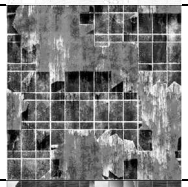
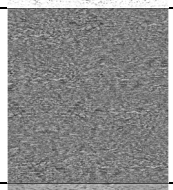
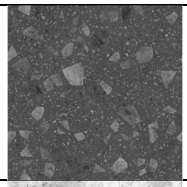
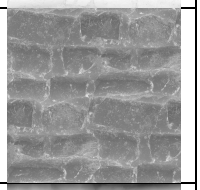
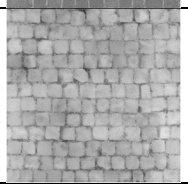
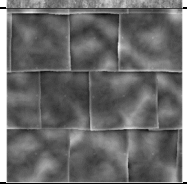
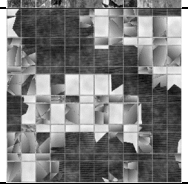
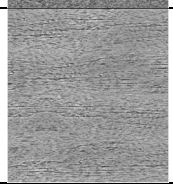
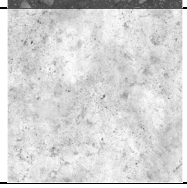
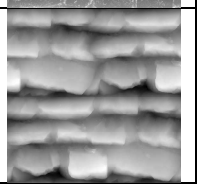
Material	(a) PavingStones	(b) MetalPlates013	(c) Tiles135D	(d) Wood063	(e) Terrazzo018	(f) Rock064
Color						
Normal						
Ambient Occlusion						
Roughness						
Displacement						

Table 2: Quality and performance analysis by changing the model size and texture resolution. S, M, and L in the model size column denote small, medium, and large, respectively. CV stands for cooperative vectors.

Material (Res.)	Model Size	PSNR (dB)	Encoding (ms/step)	Decoding(ms)	
				w/o/ CV	w/ CV
Paving Stones (2K)	S	27.17	0.5251	2.99	1.17
	M	28.08	0.5688	3.18	1.18
	L	28.35	0.6352	3.45	1.22
Metal Plates (2K)	S	36.23	0.5157	3.00	1.20
	M	37.23	0.5677	3.20	1.21
	L	37.34	0.6277	3.47	1.24
Tiles 135D (2K)	S	33.99	0.5317	2.99	1.17
	M	35.27	0.5604	3.18	1.17
	L	35.28	0.5952	3.45	1.22
Wood 063 (8K)	S	31.83	0.5328	48.16	18.54
	M	33.93	0.5592	51.54	18.68
	L	33.89	0.5979	56.44	19.02
Terrazzo 018 (8K)	S	42.59	0.5222	48.12	18.16
	M	43.48	0.5467	51.73	18.02
	L	43.09	0.5850	56.31	18.72
Rock 064 (8K)	S	30.35	0.5340	48.38	18.25
	M	31.87	0.5466	51.88	18.32
	L	31.86	0.5917	56.13	18.92

4.2. 텍스처 채널 복잡도에 따른 압축 효율 변화 분석

3.2.2 절에서 언급한 것과 같이, 본 절에서는 일반적인 환경에서 컬러 및 노멀 맵만 압축할 때에도 NTC 가 기존 BC7 대비 이점이 있는지 확인한다. 이는 신경망이 학습할 수 있는 상관관계 정보가 제한된 상황에서 기존 블록 압축 방식과 같은 압축률을 설정했을 때에도 NTC 가 화질 우위를 점한다면 비 PBR 텍스처에서도 유용함을 입증하기 때문이다.

Table 3 에 기재된 단일 맵 실험 결과는 컬러 맵만을 단독으로 8 BPP 로 압축한 것으로, 6 종 중 5 종의 텍스처에서 NTC 가 BC7 대비 높은 PSNR 을 기록하여 수치적 우위를 점하였다. 하지만 지각적 오차(FLIP) 측면에서는 MetalPlates 를 제외하고 대등하거나 다소 높은 수치를 보였다. 참고로 FLIP 은 수치가 낮을수록 원본과의 차이가 적음을 나타낸다. NTC 가 BC7 대비 열위에 있는 Terrazzo018 는 압축 아티팩트 자체가 적어 높은 PSNR 과 낮은 FLIP 수치를 나타내는 경우로, 수치상의 차이와 달리 시각적인 차이는 거의 없다.

다음으로, Table 4 에 컬러 맵에 기하학적 정보인 노멀 맵을 추가하여 채널 복잡도를 증가시킨 환경에서의 실험 결과를 기재하였다. 데이터의 차원이 단일 채널에서 복합 채널로 확장되고 할당되는 용량도 8 BPP 에서 16 BPP 로 늘어나자, 앞선 결과와는 달리 NTC 의 화질적 우위가 도드라지는 경우가 노멀 맵을 중심으로 나타났다. NTC 는 표면 굴곡이 복잡한 자연물 텍스처인 Wood063, PavingStones, Rock064 의 노멀 맵에서 7~8dB 더 높은 PSNR 수치를 기록하였고, PavingStones 와 MetalPlates, Rock064 의 노멀 맵에서 FLIP 값이 각각 0.0082, 0.0029, 0.0027 만큼 감소하였다. 이는 NTC 가 BC7 대비 표면의 미세한 요철과 질감을 시각적으로 보다 정교하게 보존하였음을 의미한다. Figure 2 는 PavingStones 의 노멀 맵을 동일 압축률에서 BC7 및 NTC 로 각각 압축했을 때의 결과를 비교하는데, 이를 보면 NTC 가 원본 이미지의 방향 벡터값을 좀 더 잘 보존한다. Figure 3 의 색반전된 FLIP 맵을 보면 특히 움푹 들어간 부분에서 NTC 의 FLIP 오차값이 더 작은 것을 확인 가능하다. 또한 컬러 맵에서도 기존 8 BPP 환경과 비교하여 Tiles135D 를 제외한 모든 머티리얼에서 PSNR 및 FLIP 수치 모두 향상된 결과를 나타냈다. 이는 컬러와 노멀 정보를 독립적인 데이터로 처리하는 기존 BC7 방식과 달리, NTC 모델은 두 채널 간의 구조적 상관관계를 효과적으로 학습하기 때문에 색상 변화와 표면 굴곡이 일치하는 영역의 디테일을 성공적으로 복원 가능함을 보여준다.

위 결과는 PBR 환경이 아닌 컬러와 노멀 맵만 사용하는 일반적인 렌더링 환경에서도 NTC 가 기존 블록 압축 방식 대비 품질 측면에서 일부 이점을 가질 수 있음을 추가적으로 의미한다. 아울러 이 결과는 텍스처 데이터의 채널이 늘어나고 복잡도가 높아질수록 신경망 기반 압축(NTC)의 효율성이 증대된다는 선행 연구[1]의 결과와도 일치한다.

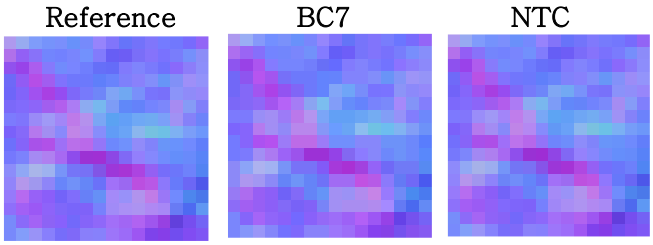


Figure 2. Zoom-in comparison of normal compression in PavingStones, highlighting loss of detail in BC7 and detail preservation in NTC in the lower-right light purple region.

Table 3: Compression Quality Comparison of Single Color Map Configuration (8 BPP)

Texture	Type	PSNR (dB)			FLIP		
		BC7	NTC	Delta	BC7	NTC	Detla
PavingStones	Color	40.36	40.95	+0.59	0.0271	0.0335	+0.0064
MetalPlates	Color	47.38	49.09	+1.71	0.0153	0.0148	-0.0005
Tiles135D	Color	49.83	50.12	+0.29	0.0128	0.0137	+0.0009
Wood063	Color	43.41	44.14	+0.73	0.0159	0.0164	+0.0005
Terrazzo018	Color	52.93	45.85	-7.08	0.0082	0.0144	+0.0062
Rock064	Color	39.48	40.30	+0.82	0.0155	0.0199	+0.0044

Table 4: Compression Quality in Color and Normal Map Configuration (16 BPP)

Material	Type	PSNR (dB)			FLIP		
		BC7	NTC	Delta	BC7	NTC	Detla
PavingStones	Color	40.36	41.77	+1.41	0.0271	0.0317	+0.0046
	Normal	34.88	42.14	+7.26	0.0376	0.0294	-0.0082
MetalPlates	Color	47.38	49.60	+2.22	0.0153	0.0144	-0.0009
	Normal	44.89	49.13	+4.24	0.0181	0.0152	-0.0029
Tiles135D	Color	49.83	49.34	-0.49	0.0128	0.0155	+0.0027
	Normal	41.60	47.89	+6.29	0.0205	0.0185	-0.0020
Wood063	Color	43.41	44.54	+1.13	0.0159	0.0159	0.0000
	Normal	37.06	45.63	+8.57	0.0160	0.0160	0.0000
Terrazzo018	Color	52.93	47.37	-5.56	0.0082	0.0122	+0.0040
	Normal	52.58	48.66	-3.92	0.0080	0.0104	+0.0024
Rock064	Color	39.48	43.07	+3.59	0.0155	0.0183	+0.0028
	Normal	36.92	44.89	+7.97	0.0187	0.0160	-0.0027

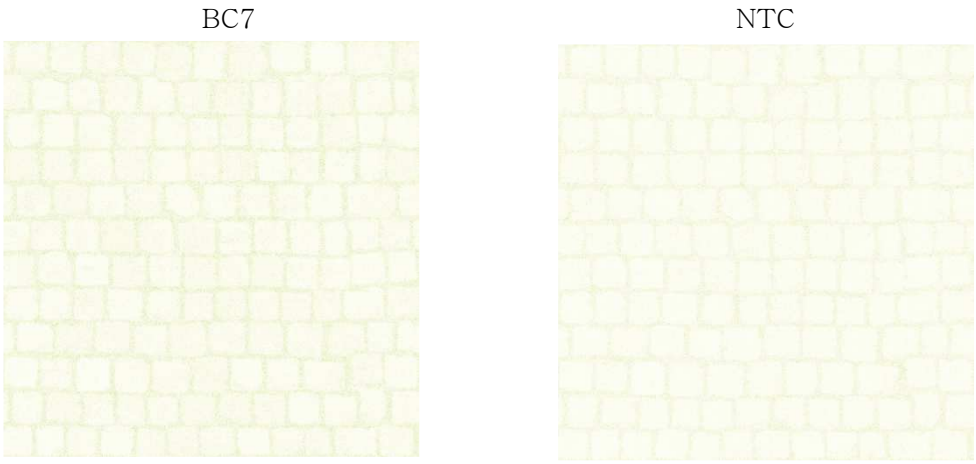


Figure 3. Color-inverted FLIP maps of normal compression in PavingStones, where lighter colors indicate smaller errors.

Table 5: Quantitative quality analysis of NTC-to-BCn transcoding

Material	Type	PSNR (dB)			FLIP		
		NTC only	BC4/7 Transcoding	Delta	NTC only	BC4/7 Transcoding	Delta
PavingStones	Color	26.66	26.58	-0.08	0.1041	0.1060	+0.0019
	Normal	27.32	26.84	-0.48	0.1115	0.1151	+0.0036
	Occlusion	31.54	31.67	+0.13	0.0465	0.0477	+0.0012
	Displacement	38.95	38.99	+0.04	0.0387	0.0387	+0.0000
	Roughness	34.60	34.59	-0.01	0.0476	0.0477	+0.0001
MetalPlates	Color	38.31	38.09	-0.22	0.0392	0.0410	+0.0018
	Normal	36.98	36.72	-0.26	0.0435	0.0449	+0.0014
	Occlusion	47.47	47.51	+0.04	0.0181	0.0181	0.0000
	Displacement	44.76	44.85	+0.09	0.0275	0.0275	0.0000
	Roughness	35.27	35.33	+0.06	0.0474	0.0476	+0.0002
Tiles135D	Color	38.19	38.06	-0.13	0.0443	0.0455	+0.0012
	Normal	34.84	34.34	-0.50	0.0566	0.0598	+0.0032
	Occlusion	41.69	41.82	+0.13	0.0235	0.0234	-0.0001
	Displacement	38.42	38.60	+0.18	0.0451	0.0452	+0.0001
	Roughness	30.43	30.56	+0.13	0.0608	0.0611	+0.0003
Wood063	Color	31.34	31.18	-0.16	0.0638	0.0639	+0.0001
	Normal	33.29	32.13	-1.16	0.0627	0.0629	+0.0002
	Occlusion	34.32	34.41	+0.09	0.0329	0.0332	+0.0003
	Displacement	41.52	41.60	+0.08	0.0304	0.0306	+0.0002
	Roughness	36.88	37.01	+0.13	0.0377	0.0377	+0.0000
Terrazzo018	Color	41.41	41.19	-0.22	0.0265	0.0249	-0.0016
	Normal	43.86	43.75	-0.11	0.0159	0.0164	+0.0005
	Displacement	41.16	41.18	+0.02	0.0204	0.0204	+0.0000
	Roughness	45.31	45.28	-0.03	0.0137	0.0138	+0.0001
Rock064	Color	36.99	36.80	-0.19	0.0321	0.0322	+0.0001
	Normal	28.51	28.36	-0.15	0.0626	0.0627	+0.0001
	Occlusion	43.12	43.28	+0.16	0.0256	0.0265	+0.0009
	Displacement	30.93	30.50	-0.43	0.0629	0.0633	+0.0004
	Roughness	35.34	35.07	-0.27	0.0335	0.0338	+0.0003

4.3. 트랜스코딩의 효율성 및 이중 압축 손실 분석

GPU 내 하드웨어 NTC 디코더가 탑재되기 전까지는 NTC 를 고효율 배포 포맷으로 사용하고 런타임에 기존 GPU 가속 포맷(BC7 등)으로 변환하는 트랜스코딩 방식을 사용할 수 있다. 이 과정은 손실 압축된 데이터를 다시 손실 압축하는 이중 압축 과정을 거치므로, 필연적으로 이중 압축 손실이 발생한다. 선행 연구[2]에서는 트랜스코딩 후 PSNR 차이가 BC1 은 -0.3dB 인 반면 BC5 와 BC7 은 각각 +0.055dB 와 -0.055dB 의 미미한 수준으로 보고하였다.

본 절에서는 6 개 실험 데이터세트에 대하여 NTC 압축 후 BC7(컬러 맵 및 노멀 맵) 또는 BC4(나머지 1 채널 텍스처)로 트랜스코딩한 결과를 서술한다. Table 5 에 따르면, 대체로 트랜스코딩 전후 PSNR 차이는 선행 연구와 유사하게 큰 차이가 없었으나, FLIP 은 소폭이지만 대부분 증가하는 경향을 나타냈다. 또한 일부 노멀 맵에서는 상대적으로 큰 차이로 PSNR 값이 감소(Wood063 1.16dB)하거나 FLIP 값이 유의미하게 증가(PavingStones 0.036, Tiles135D 0.032)한 것으로 나타나, 노멀 맵의 경우 트랜스코딩 이후 화질 확인이 필요할 것으로 보인다. 참고로 이는 SDK 상에서 트랜스코딩 속도를 높이기 위해 제공되는 --optimizeBC 옵션을 사용하지 않은 결과로, 해당 옵션을 사용하게 되면 화질 저하가 더욱 많이 일어날 수 있다.

또 한가지 중요한 점은, 트랜스코딩 사용 시 스토리지 오버헤드는 NTC 와 같지만 GPU 의 VRAM 사용량은 타겟 코덱과 동일해진다는 점이다. BC7 은 BC1 대비 압축률이 낮지만 (RGB 기준 BC7 은 BC1 대비 2 배의 용량 필요) 알파 채널 압축과 화질 측면에서 큰 이점이 있어서 사용되는데, 트랜스코딩 후의 BC7 화질은 NTC 압축 시의 화질에 따라 크게 좌우되므로 BC7 으로 압축되었음에도 낮은 화질이 관측될 수 있다. 실제로 Table 4 의 처음부터 BC7 로 압축한 결과와 Table 5 의 NTC 에서 BC7 으로 트랜스코딩한 결과를 비교하면, PSNR 수치는 최고 13.7dB (PavingStones), FLIP 수치는 3.8 배(동일 머티리얼)까지 나는 것을 확인할 수 있다. 이는 BC7 에서 고화질 압축이라는 이점을 크게 퇴색시킨 결과이다. 다만 단일 채널의 변위(displacement), 차폐(occlusion), 거칠기(roughness) 맵의 결과는 PSNR 40dB 전후인 경우가 종종 나타나 다중채널 맵들과 비교시 대체로 화질 저하가 적은 것으로 나타났다.

결론적으로 BC4 나 BC7 과 같은 고품질 압축 코덱으로 NTC 로 압축된 데이터를 트랜스코딩하고자 한다면, NTC 의 압축률을 적당한 수준으로 설정하여 스토리지 오버헤드와 압축 품질간에 적절한 트레이드 오프를 결정해야 한다. 특히 단일 채널 맵보다는 다중 채널 맵에서 예상하지 못했던 압축 아티팩트가 발생하지 않았는지 꼼꼼히 확인할 필요가 있다.

5. 결론

본 연구에서는 NVIDIA NTC SDK 를 활용하여 신경망 기반 텍스처 압축 기술의 실무적 효용성을 검증하였다. 구체적으로, 먼저 모델 복잡도에 따른 화질 및 성능 차이를 비교하였고, 둘째로 저채널 텍스처 환경에서의 화질을 산업 표준인 BC7 과 비교하였으며, 마지막으로 트랜스코딩 시 텍스처별 압축 품질을 측정하였다.

위 실험을 통해 도출된 핵심 결론과 향후 연구 방향은 다음과 같다. 먼저, 현재의 소프트웨어 디코딩 방식은 해상도가 증가할수록 협동 벡터 사용이 필수적이다. 그러나 8K 초고해상도 환경에서는 이를 사용하더라도 여전히 디코딩 병목이 발생할 수 있으므로, 전용 하드웨어의 도입 또는 알고리즘 개선을 통한 추가 디코딩 성능 향상이 필요하다.

둘째, NTC 논문[1]에서 채널 수가 적은 경우 품질 향상이 제한적일 수 있다는 한계가 제시된 바 있으나, 본 연구에서는 컬러 맵과 노멀 맵과 같은 일반적인 텍스처 환경에서도 NTC 가 BC7 대비 노멀 맵 압축에서 화질 이점을 보이는 것을 확인하였다. 따라서 디코딩 성능 문제가 해결된다면, 일반적인 그래픽 파이프라인에서도 화질 개선 또는 압축률 향상을 위해 BC7 대신 NTC 를 사용하는 것이 실용적일 수 있다.

셋째, 디코딩 오버헤드를 제거하기 위해 트랜스코딩을 사용할 경우 노멀 맵에서 일부 추가적인 화질 저하가 발생할 수 있으므로 주의가 필요하다. 또한 이 경우 NTC 의 압축률 설정에 따라 타겟 코덱과 동일한 VRAM 을 사용하면서도 화질 저하가 발생할 수 있으므로, 머티리얼 특성에 맞는 적절한 압축률 설정이 요구된다.

향후 연구로는 앞서 언급한 디코딩 성능 개선이 과제로 제시된다. 또한 일부 화질 저하의 원인 분석을 통해 기존 네트워크 구조의 개선 방안을 모색할 필요가 있다. 아울러 이러한 분석 과정에서 비등방성 필터링을 지원하는 타 신경망 기반 코덱들(예: TSNC)[29]과 실제 렌더링 화면에서의 화질을 비교한다면, 본 NTC 의 장단점을 보다 면밀히 분석할 수 있을 것이다.

감사의 글

본 연구는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행되었습니다(No. RS-2025-00521436). 논문의 수정에 도움을 주신 익명의 심사위원분들과, 추가 실험을 맡아준 이현기 박사과정, NTC SDK 및 소스코드를 공개한 NTC 논문의 저자들 및 개발자들에게도 감사의 뜻을 전합니다. 실험에 사용된 머티리얼 세트는 Poly Haven, ambientCG, Texxary 에서 다운로드 받았습니다. Figure 1 도해시에는 Gemini 를 사용하였습니다.

References

[1] K. Vaidyanathan, M. Salvi, B. Wronski, T. Akenine-Möller, P. Ebelin, and A. Lefohn, "Random-Access Neural Compression of Material Textures," ACM Transactions on Graphics (SIGGRAPH 2023), vol. 42, no. 4, Art. no. 1, 2023.

[2] A. Panteleev, "Practical Neural Texture Compression," in Vulkanised 2025, 2025.

[3] NVIDIA, NVIDIA RTX Neural Texture Compression SDK, GitHub, 2026. https://github.com/NVIDIA-RTX/RTXNTC

[4] K. I. Iourcha, K. S. Nayak, and Z. Hong, "System and Method for Fixed-Rate Block-Based Image Compression with Inferred Pixel Values," US Patent 5,956,431, Sep. 1999.

[5] P. Krajcevski, A. Lake, and D. Manocha, "FasTC: Accelerated Fixed-Rate Texture Encoding," in Proc. ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (i3D), pp. 137–144, 2013.

[6] P. Krajcevski and D. Manocha, "SegTC: Fast Texture Compression using Image Segmentation," in Proc. High-Performance Graphics (HPG), pp. 1–8, 2014.

[7] R. Geldreich, bc7enc: BC1-5 and BC7 Encoders and BC1-5/7 Decoders, GitHub, 2020. https://github.com/richgel999/bc7enc

[8] R. Geldreich, bc7e: Binomial's Fast High Quality BC7 Encoder, Binomial LLC, GitHub, 2021. https://github.com/BinomialLLC/bc7e,

[9] B. Taudul, "etcpak: The Fastest ETC Compressor on the Planet," GitHub,2022. https://github.com/wolfpld/etcpak

[10] Google Inc. and Blue Shift Inc., "Etc2Comp: Texture to ETC2 Compressor," GitHub, 2017. https://github.com/google/etc2comp

[11] J.-H. Nah, "QuickETC2: Fast ETC2 texture compression using luma differences," ACM Transactions on Graphics (SIGGRAPH Asia 2020), vol. 39, no. 6, Art. no. 270, 2020.

[12] J.-H. Nah, "QuickETC2-HQ: Improved ETC2 EEncoding Techniques for Real-time, High-quality Texture Compression," Computers & Graphics, vol. 116, pp. 308–316, 2023.

[13] H.-K. Lee and J.-H. Nah, "H-ETC2: Design of a CPU-GPU hybrid ETC2 encoder," Computer Graphics Forum (Pacific Graphics 2023), vol. 42, iss. 7, 2023.

[14] P. Krajcevski and D. Manocha, "Real-time PVRTC texture compression using intensity dilation," Journal of Computer Graphics Techniques (JCGT), vol. 3, no. 4, pp. 132–142, 2014.

[15] Arm Limited, astcenc: The Arm ASTC encoder, GitHub, 2025. https://github.com/ARM-software/astc-encoder,

[16] S. Pratapa, T. Olson, A. Chalfin, and D. Manocha, "TexNN: Fast Texture Encoding using Neural Networks," Computer Graphics Forum, vol. 38, iss. 1, pp. 328–339, 2019.

[17] 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.

[18] 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.

[19] J.-H. Nah, "ASTC Block-size Determination Method based on PSNR Values," Journal of the Korea Computer Graphics Society, vol. 28, no. 2, pp. 21–28, 2022.

[20] J.-H. Nah, "Addition of An Adaptive BBlock-size Determination Feature to astcenc, the reference ASTC encoder," Software Impacts, vol. 17, Art. no. 100569, 2023.

[21] J. Ström and P. Wennersten, "Lossless Compression of Already Compressed Textures," in Proc. High-Performance Graphics (HPG), pp. 177–182, 2011.

[22] Binomial LLC, Crunch: Advanced DXTc Texture Compression and Transcoding Library, GitHub, 2025. https://github.com/BinomialLLC/crunch

[23] Binomial LLC, Basis Universal supercompressed GPU texture codec, GitHub, 2026. https://github.com/BinomialLLC/basis_universal

[24] P. Krajcevski, S. Pratapa, and D. Manocha, "GST: GPU-Decodable Supercompressed Textures," ACM Transactions on Graphics (ACM SIGGRAPH Asia 2016), vol. 35, no. 6, Art. no. 230, 2016.

[25] J. Ballé, V. Laparra, and E. P. Simoncelli, "End-to-end Optimized Image Compression," in Proc. International Conference on Learning Representations (ICLR), 2017.

[26] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston, "Variational Image Compression with a Scale Hyperprior," in Proc. International Conference on Learning Representations (ICLR), 2018.

[27] S. Fujieda and T. Harada, "Neural Texture Block Compression," in Proc. MAM-MANER Workshop, 2024.

[28] C. Weinreich, L. De Oliverira, A. Houdard, and G. Nader, "Real-Time Neural Materials using Block-Compressed Features," Computer Graphics Forum (EUROGRAPHICS 2024), vol. 43, iss. 2, Art. no. e15013, 2024.

[29] L. Belcour and A. Benyoub, "Hardware Accelerated Neural Block Texture Compression with Cooperative Vectors," in Proc. High-Performance Graphics (HPG), 2025.

[30] F. Farhadzadeh et al., "Neural Graphics Texture Compression Supporting Random Access," in Proc. European Conference on Computer Vision (ECCV) 2024, pp. 412-429, 2024.

[31] P. Andersson, J. Nilsson, T. Akenine-Möller, M. Oskarsson, K. Åström, and M. D. Fairchild, “FLIP: A Difference Evaluator for Alternating Images,” Proceedings of

the ACM on Computer Graphics and Interactive Techniques (HPG 2020), vol. 3, iss. 2, Art. no. 15, pp.1-23, 2020.

[32] NVIDIA, NVIDIA Texture Tools 3, 2026. https://developer.nvidia.com/gpu-accelerated-texture-compression

[33] Microsoft, Texture Block Compression in Direct3D 11, 2020. https://docs.microsoft.com/en-us/windows/win32/direct3d11/texture-block-compression-in-direct3d-11

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

[35] J. Ström and M. Pettersson, “ETC 2: Texture Compression using Invalid Combinations,” in Proc. ACM SIGGRAPH/EUROGRAPHICS Symp. on Graphics Hardware, 2007, pp. 49–54.

[36] J. Nystad, A. Lassen, A. Pomianowski, S. Ellis, and T. Olson, "Adaptive Scalable Texture Compression," in Proc. High-Performance Graphics (HPG), pp. 105–114, 2012.

[37] S. Fenney, “Texture Compression using Low-frequency Signal Modulation,” in Proc. ACM SIGGRAPH/EUROGRAPHICS Symp. on Graphics Hardware, 2003, pp. 84–91.

[38] J.-H. Nah, "QuickETC2: How to Finish ETC2 Compression within 1 ms," ACM SIGGRAPH 2020 Talks, Art. no. 4, 2020.

[39] H.-K. Lee and J.-H. Nah, "QuickBC7: Fast BC7 Texture Compression Heuristics," Computers and Graphics, vol. 137, Art. no. 104631, 2026.