Subscribe to DeepL Pro to translate larger documents.
Visit www.DeepL.com/pro for more information.

NVIDIA 이미지 스케일링 SDK(버전 1.0.3)

프로그래밍 가이드

문서 개정: 1.0.3

출시되었습니다: 출시일: 2022년 8월 15일

저작권 NVIDIA Corporation. © 2022.

목차

초록

NVIDIA 이미지 스케일링 SDK 프로그래밍 가이드에서는 게임 또는 3D 애플리케이션에 NVIDIA 스케일링 및 선명화 알고리즘을 통합하는 방법에 대한 자세한 내용을 제공합니다. 이 가이드는 또한 몇 가지 코드 스니펫과 GitHub의 전체 샘플 구현 링크를 제공합니다. 또한 모범 사례와 알고리즘에 대한 자세한 설명이 설명되어 있습니다.

개정 내역

1.0.3	통합 샘플 NV12 지원 간소화 경고 수정 속어 컴파일러 지원 추가	2022년 8월 15일
1.0.2	성능 최적화 최소 선명도 값 조정 경고 수정	2022년 2월 2일
1.0.1	성능 최적화 fp16 계수 지원 GLSL 지원 DX12 및 벌칸 샘플	2021년 12월 13일
1.0.0	초기 릴리스	2021년 11월 16일

1 소개

NVIDIA 이미지 스케일링 SDK는 크로스 플랫폼 지원을 위한 단일 공간 스케일링 및 선명화 알고리즘을 제공합니다. 스케일링 알고리즘은 6탭 스케일링 필터와 4개의 방향성 스케일링 및 적응형 선명화 필터를 결합하여 부드러운 이미지와 선명한 가장자리를 생성합니다. 또한 SDK는 스케일링이 필요하지 않은 애플리케이션에서 사용할 수 있는 최첨단 적응형 방향 샤프닝 알고리즘을 제공합니다. 개발자는 NVIDIA 이미지 스케일링과 NVIDIA DLSS를 모두 통합하여 최고의 이미지 품질을 위한 NVIDIA DLSS와 크로스 플랫폼 지원을 위한 NVIDIA 이미지 스케일링의 두 가지 장점을 모두 얻을 수 있습니다.

방향성 스케일링과 선명화 알고리즘은 NVScaler에서 함께 결합된 반면 NVSharpen은 적응형 방향성 선명화 알고리즘만 구현합니다. 두 알고리즘은 모두 컴퓨팅 셰이더로 제공되며 개발자는 애플리케이션에 자유롭게 통합할 수 있습니다. **NV스케일러를** 통합하는 경우 **NV스케일러에는 이미 샤프닝 패스가 포함되어** 있으므로 NV샤프닝도 통합해서는 안 됩니다.

2 시작하기

2.1 시스템 요구 사항

NV스케일러와 NV샤프닝을 로드하고 실행하려면 다음이 필요합니다:

- Windows 10 v1709 이상 버전이 설치된 PC
- DX11, DX12 또는 Vulkan 호환 GPU

SDK에 포함된 샘플 앱은 Windows 또는 Linux용으로 컴파일할 수 있습니다.

2.2 렌더링 엔진 요구 사항

컴퓨팅 셰이더는 Direct3D 또는 벌칸과 통합할 수 있으며, 애플리케이션 또는 렌더링 엔진이 있어야 합니다:

- 하이레벨 셰이더 언어(HLSL) 모델 5.0 이상 또는 OpenGL 셰이더 언어(GLSL) 4.50.7(버전 450)을 지원합니다.
- HLSL 셰이더를 통합하려면 DirectX11, DirectX12 또는 Vulkan을 지원해야 합니다.
- TAA와 같은 고품질 앤티 앨리어싱 기술 지원
- 텍스처와 지오메트리에 대한 LOD를 네거티브 바이어스할 수 있습니다.
- 각 셰이더 호출(즉, 각 프레임)에서 다음을 제공합니다:
    - 프레임의 원시 색상 텍스처(디스플레이 참조 색 공간의 HDR 또는 SDR)
    - 올바른 치수의 출력 텍스처

- NVScaler의 경우: 스케일 및 선명도 값과 구성 및 계수를 제공합니다.
- NVSharpen의 경우: 선명도 값 및 구성 값

향후 호환성을 확보하고 NVIDIA의 지속적인 연구를 용이하게 하기 위해 애플리케이션은 수정 없이 NVIDIA 이미지 스케일링 SDK 컴퓨팅 셰이더를 통합하는 것을 고려해야 합니다.

2.3 샘플 요구 사항

NVIDIA 이미지 스케일링 SDK에 포함된 샘플에는 다음 도구가 필요합니다:

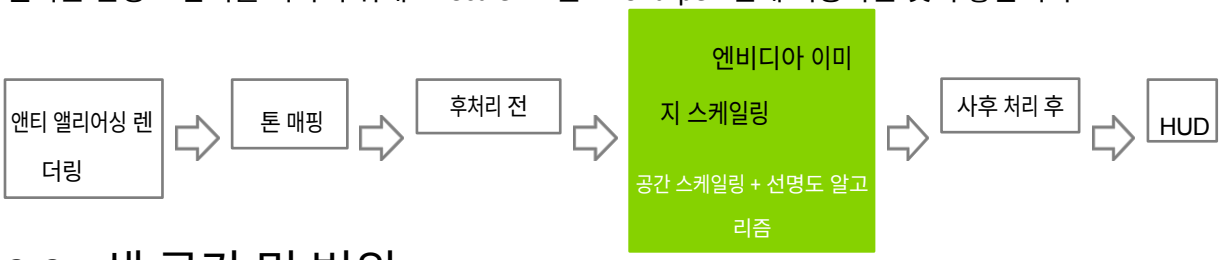
- CMake 3.12 이상
- Visual Studio 2019
- Windows SDK
- 벌칸(벌칸 샘플이 활성화된 경우)

3 NVIDIA 이미지 스케일링 셰이더 통합

3.1 파이프라인 배치

NVIDIA 이미지 스케일링 셰이더에 대한 호출은 톤 매핑 후 포스트 프로세싱 단계에서 이루어져야 합니다. 선형 HDR 인게임 색 공간에서 스케일링을 적용하면 선명도 효과가 보이지 않거나 너무 강해질 수 있습니다.

선명화 알고리즘은 노이즈나 입자가 거친 영역을 향상시킬 수 있으므로 필름 그레인과 같은 특정 효과는 NVScaler 또는 NVSharpen 이후에 적용하는 것이 좋습니다. 모션 블러 또는 라이트 블룸과 같은 저역 통과 필터는 선명도 감쇠를 피하기 위해 NVScaler 또는 NVSharpen 전에 적용하는 것이 좋습니다.



3.2 색 공간 및 범위

NVIDIA 이미지 스케일링 셰이더는 다음과 같은 제한 사항과 함께 LDR 또는 HDR로 저장된 컬러 텍스처를 처리할 수 있습니다:

1) LDR

- 색상 값의 범위는 [0, 1] 범위여야 합니다.
- 입력 색상 텍스처는 톤 매핑 및 OETF(감마 보정)가 적용된 후 디스플레이 참조 색 공간에 있어야 합니다.

2) HDR PQ

- 색상 값의 범위는 [0, 1] 범위여야 합니다.
- 입력 색상 텍스처는 Rec로 톤 매핑한 후 디스플레이 참조 색 공간에 있어야 합니다.
  2020 PQ OETF 적용

3) HDR 리니어

- 권장되는 색상 값 범위는 [0, 12.5]이며, 여기서 휘도 값(BT. 709에 따른)은 다음과 같습니다.
  1.0 맵에서 밝기 값 80니트(sRGB 피크), 12.5 맵에서 1000니트까지 지원
- 입력 컬러 텍스처는 선형 및 장면 참조 또는 선형 및 디스플레이 참조(톤 매핑 후) 휘도 값을 가질 수 있습니다.

NIS로 전송된 입력 컬러 텍스처가 HDR 형식인 경우 NIS_HDR_MODE 정의를 NIS_HDR_MODE_LINEAR(1) 또

는 NIS_HDR_MODE_PQ(2)로 설정합니다.

3.3 지원되는 텍스처 포맷

3.3.1 입력 및 출력 형식

입력 및 출력 형식은 이전 섹션에서 정의된 범위여야 하며, DXGI_FORMAT_R8G8B8A8_UNORM 또는 DXGI_FORMAT_NV12와 같은 정수가 아닌 데이터 유형을 사용하여 지정해야 합니다.

3.3.2 계수 형식

스케일러 계수 및 USM 계수 형식은 DXGI_FORMAT_R32G32B32A32_FLOAT 또는 DXGI_FORMAT_R16G16B16A16_FLOAT와 같은 float4 타입을 사용하여 지정해야 합니다.

3.4 리소스 상태, 버퍼 및 샘플러

NVIDIA 이미지 스케일링 SDK 셰이더를 호출하는 게임 또는 애플리케이션은 텍스처가 올바른 상태인지 확인해야 합니다.

- 입력 컬러 텍스처는 픽셀 셰이더 읽기 상태여야 합니다.
    - DirectX의 셰이더 리소스 뷰(SRV)
    - 벌칸의 샘플링된 이미지
- 출력 텍스처는 읽기/쓰기 상태여야 합니다.
    - DirectX의 주문되지 않은 액세스 보기(UAV)
    - 벌칸의 스토리지 이미지
- NVScaler의 계수 텍스처는 읽기 상태여야 합니다.
    - DirectX의 셰이더 리소스 뷰(SRV)
    - 벌칸의 샘플링된 이미지
- 구성 변수는 상수 버퍼로 전달되어야 합니다.
    - DirectX의 상수 버퍼 뷰(CBV)
    - 벌칸의 균일한 버퍼
- 텍스처 픽셀 샘플링을 위한 샘플러는 선형 필터 보간과 클램프 투 에지 어드레싱 모드를 사용해야 합니다.

4 프로젝트에 NVIDIA 이미지 스케일링 SDK 추가하기

4.1 프레임워크와 통합

이 섹션의 통합 지침은 HLSL 또는 GLSL을 사용하여 자체 DX11, DX12 또는 Vulkan 애플리케이션에 최소한의 변경만으로 적용할 수 있습니다.

4.1.1 SDK 패키지 파일, 구성 도구 및 상수 정의

장치

- NIS_Scaler.h: 셰이더 파일, NVScaler 및 NVSharpen 구현이 포함되어 있습니다.
- NIS_Main.hlsl: 기본 HLSL 셰이더 예제(자체 셰이더로 대체 가능)
- NIS_Main.glsl: 기본 GLSL 셰이더 예제(자체 셰이더로 대체 가능)

호스트 구성

- NIS_Config.h: fp32 및 fp16 형식의 계수 및 구성 선언

정의

- **NIS_SCALER**: 1은 NvScaler 활성화, 0은 빠른 NvSharpen만 수행합니다(업스케일링 없음).
- **NIS_HDR_MODE**: 0 비활성화, 1 선형, 2 PQ
- **NIS_BLOCK_WIDTH**: 블록 너비당 픽셀 수입니다. 플랫폼에 맞는 GetOptimalBlockWidth 쿼리 사용
- **NIS_BLOCK_HEIGHT**: 블록 높이당 픽셀 수입니다. 플랫폼에 맞는 GetOptimalBlockHeight 쿼리 사용
- **NIS_THREAD_GROUP_SIZE**: 그룹당 스레드 수입니다. 플랫폼에 맞는 GetOptimalThreadGroupSize 쿼리 사용
- **NIS_VIEWPORT_SUPPORT**: 0 비활성화, 1 입력/출력 뷰포트 지원 활성화
- **NIS_USE_HALF_PRECISION**: 0 비활성화, 1 반정밀도 계산 활성화
- **NIS_HLSL**: 0 비활성화, 1 HLSL 지원 활성화
- **NIS_HLSL_6_2**: 0 HLSL v5, 1 HLSL v6.2는 NIS_HSLS=1 강제 적용
- **NIS_GLSL**: 0 비활성화, 1 GLSL 지원 활성화
- **NIS_NV12_SUPPORT**: 0 비활성화, 1 NV12 입력 활성화
- **NIS_CLAMP_OUTPUT**: 0 비활성화, 1 출력 클램프 활성화

DXC 바인딩이 있는 HLSL에 대한 정의

- **NIS_DXC**: 0 비활성화, 1 HLSL DXC 벌칸 지원 활성화

최적의 셰이더 설정

현재 및 향후 하드웨어에 대한 NVScaler 및 NVSharpen의 최적의 성능을 얻으려면 다음 API를 사용하여 **NIS_BLOCK_WIDTH**, **NIS_BLOCK_HEIGHT** 및 **NIS_THREAD_GROUP_SIZE** 값을 구하는 것이 좋습니다. 이 값은 오프라인에서 NVScaler 및 NVSharpen의 순열을 컴파일하는 데 사용할 수 있습니다.

```
열거형 클래스 NISGPUArchitecture : uint32_t
구조체 NISOptimizer
{
    bool isUpscaling; NISGPU아키텍처
    gpuArch;

    NISOptimizer(bool isUpscaling = true,
                 NISGPUArchitecture gpuArch = NISGPUArchitecture::NVIDIA_Generic); uint32_t
    GetOptimalBlockWidth();
    uint32_t GetOptimalBlockHeight();
    uint32_t GetOptimalThreadGroupSize();
```

```
};
```

HDR 셰이더 설정

다음 열거형 값을 사용하여 **NIS_HDR_MODE**를 설정합니다.

```
열거형 클래스 NISHDRMode : uint32_t
{
    없음 = 0,
    선형 = 1,
    PQ = 2
};
```

4.1.2 NVScaler 통합

NIS_Main.hlsl 셰이더 컴파일

NIS_SCALER를 **1로** 설정하고 isUpscaling 인수를 **true로** 설정해야 합니다.

```
bool isUpscaling = true;
NISOptimizer opt(isUpscaling, NISGPUArchitecture::NVIDIA_Generic);
uint32_t blockWidth = opt.GetOptimalBlockWidth();
uint32_t blockHeight = opt.GetOptimalBlockHeight(); uint32_t
threadGroupSize = opt.GetOptimalThreadGroupSize();

defines; defines.add("NIS_SCALER",
isUpscaling); defines.add("NIS_HDR_MODE",
hdrMode);
defines.add("NIS_BLOCK_WIDTH", blockWidth);
defines.add("NIS_BLOCK_HEIGHT", blockHeight);
defines.add("NIS_THREAD_GROUP_SIZE", threadGroupSize);
NVScalerCS = CompileComputeShader(device, "NIS_Main.hlsl", &defines);
```

참고: 셰이더 퍼뮤테이션 컴파일은 오프라인에서 수행할 수 있습니다.

NVIDIA 이미지 스케일링 SDK 구성 상수 버퍼 만들기

```
구조체 NISConfig
{
    float kDetectRatio;
    float kDetectThres;
    float kMinContrastRatio;
    float kRatioNorm;
    ...
};

NISConfig config;
createConstBuffer(&config, &csBuffer);
```

스케일러 및 USM 위상 계수에 대한 SRV 텍스처 생성

```
const int rowPitch = kFilterSize * sizeof(float); // 사용 fp32: float, fp16: uint16_t
const int coefSize = rowPitch * kPhaseCount;
// RGBA 포맷을 사용하므로 텍스처 너비 = kFilterSize / 4 createTexture2D(kFilterSize / 4,
kPhaseCount, DXGI_FORMAT_R32G32B32A32_FLOAT, D3D11_USAGE_DEFAULT, coef_scaler,
rowPitch, coefSize, &scalerTex) // 이 값은 다음과 같습니다;
```

```
createTexture2D(kFilterSize / 4, kPhaseCount, DXGI_FORMAT_R32G32B32A32_FLOAT,
D3D11_USAGE_DEFAULT, coef_usm, rowPitch, coefSize, &usmTex)를 호출합니다;

createSRV(scalerTex.Get(), DXGI_FORMAT_R32G32B32A32_FLOAT, &scalerSRV);
createSRV(usmTex.Get(), DXGI_FORMAT_R32G32B32A32_FLOAT, &usmSRV);
```

참고: 계수에 DXGI_FORMAT_R16G16B16A16_FLOAT와 같은 fp16 형식을 지정할 수도 있습니다. 이 경우 fp16 계수 coef_scaler_fp16 및 coef_usm_fp16을 사용합니다.

샘플러 만들기

```
createLinearClampSampler(&linearClampSampler);
```

NVIDIA 이미지 스케일링 SDK 구성 및 상수 버퍼 업데이트

다음 API 호출을 사용하여 NVIDIA 이미지 스케일링 SDK 구성을 업데이트합니다.

```
bool NVScalerUpdateConfig(NISConfig& config,
    float 선명도,
    uint32_t inputViewportOriginX, uint32_t inputViewportOriginY,
    uint32_t inputViewportWidth, uint32_t inputViewportHeight,
    uint32_t inputTextureWidth, uint32_t inputTextureHeight, uint32_t
    outputViewportOriginX, uint32_t 출력 뷰포트 오리진 Y, uint32_t 출력
    뷰포트 폭, uint32_t 출력 뷰포트 높이, uint32_t 출력 텍스처 폭,
    uint32_t 출력 텍스처 높이, NISHDRMode hdrMode = NISHDRMode::None
);
```

NVScalerUpdateConfig는 구성에 성공하면 참을 반환하고, 구성할 수 없으면 거짓을 반환합니다. 입력 텍스처 크기는 출력 텍스처 크기보다 작거나 같아야 합니다. 이 알고리즘은 뷰포트 불일치를 확인하지 않습니다.

뷰포트가 필요한 경우 NIS_VIEWPORT_SUPPORT = 1로 셰이더를 컴파일합니다. 출력 뷰포트 크기가 출력 텍스처 크기에 가까울 때 뷰포트를 사용하면 성능에 영향을 줄 수 있습니다. 성능을 향상시키려면 뷰포트 크기에 맞게 디스패치 치수를 조정하는 것이 좋습니다.

입력 크기, 선명도 또는 배율이 변경될 때마다 상수 버퍼를 업데이트합니다.

```
NISUpdateConfig(m_config, 선명도,
                    0, 0, 입력폭, 입력높이, 입력폭, 입력높이,
                    0, 0, 출력폭, 출력높이, 출력폭, 출력높이, NISHDRMode::None);

updateConstBuffer(&config, csBuffer.Get());
```

간단한 DX11 NVScaler 디스패치 예제

```
context->CSSetShaderResources(0, 1, input); // SRV
context->CSSetShaderResource (1, 1, scalerSRV.GetAddressOf());
context->CSSetShaderResource (2, 1, usmSRV.GetAddressOf());
```

```
context->CSSetUnorderedAccessViews(0, 1, output, nullptr);
context->CSSetSamplers(0, 1, linearClampSampler.GetAddressOf());
context->CSSetConstantBuffers(0, 1, csBuffer.GetAddressOf());
context->CSSetShader(NVScalerCS.Get(), nullptr, 0);

context->Dispatch(UINT(std::ceil(outputWidth / float(blockWidth))),
                  UINT(std::ceil(outputHeight / float(blockHeight))), 1);
```

4.1.3 NVSharpen 통합

애플리케이션에 업스케일링 및 선명화가 필요한 경우 **NVSharpen을 사용하지 마십시오.** 대신 NVScaler를 사용하십시오. NVScaler는 업스케일링과 샤프닝 두 가지 작업을 한 번에 수행하므로 더 빠른 성능과 더 나은 이미지 품질을 생성합니다.

NIS_Main.hlsl 셰이더 컴파일

NIS_SCALER를 0으로 설정하고 옵티마이저 isUscaling 인수를 **false로** 설정해야 합니다.

```
bool isUpscaling = false;
NISOptimizer opt(isUpscaling, NISGPUArchitecture::NVIDIA_Generic);
uint32_t blockWidth = opt.GetOptimalBlockWidth();
uint32_t blockHeight = opt.GetOptimalBlockHeight(); uint32_t
threadGroupSize = opt.GetOptimalThreadGroupSize();

defines.add("NIS_DIRSCALER", isUpscaling);
defines.add("NIS_HDR_MODE", hdrMode);
defines.add("NIS_BLOCK_WIDTH", blockWidth)를
정의합니다;
defines.add("NIS_BLOCK_HEIGHT", blockHeight);
defines.add("NIS_THREAD_GROUP_SIZE", threadGroupSize);
NVSharpenCS = CompileComputeShader(device, "NIS_Main.hlsl", &defines);
```

참고: 셰이더 퍼뮤테이션 컴파일은 오프라인에서 수행할 수 있습니다.

NVIDIA 이미지 스케일링 SDK 구성 상수 버퍼 만들기

```
구조체 NISConfig
{
    float kDetectRatio;
    float kDetectThres;
    float kMinContrastRatio;
    float kRatioNorm;
    ...
};

NISConfig config;
createConstBuffer(&config, &csBuffer);
```

샘플러 만들기

```
createLinearClampSampler(&linearClampSampler);
```

NVIDIA 이미지 스케일링 SDK 구성 및 상수 버퍼 업데이트

다음 API 호출을 사용하여 NVIDIA 이미지 스케일링 SDK 구성을 업데이트합니다. NVSharpen은 선명도 알고리즘이므로 선명도와 입력 크기만 필요합니다. **샤프닝을 통한 업스케일링의 경우** 두 작업을 동시에 수행하므로 **NVScaler를 사용합니다.**

```
void NVSharpenUpdateConfig(NISConfig&&) config, float 선명도,
    uint32_t inputViewportOriginX, uint32_t inputViewportOriginY,
    uint32_t inputViewportWidth, uint32_t inputViewportHeight,
    uint32_t inputTextureWidth, uint32_t inputTextureHeight, uint32_t
    outputViewportOriginX, uint32_t outputViewportOriginY, NISHDRMode
    hdrMode = NISHDRMode::None
);
```

입력 크기나 선명도가 변경될 때마다 상수 버퍼를 업데이트합니다.

```
NVSharpenUpdateConfig(m_config, 선명도,
                              0, 0, 입력폭, 입력높이, 입력폭, 입력높이,
                              0, 0, NISHDRMode::None);

updateConstBuffer(&config, csBuffer.Get());
```

간단한 DX11 NVSharpen 디스패치 예제

```
context->CSSetShaderResources(0, 1, input);
context->CSSetUnorderedAccessViews(0, 1, output, nullptr);
context->CSSetSamplers(0, 1, linearClampSampler.GetAddressOf());
context->CSSetConstantBuffers(0, 1, csBuffer.GetAddressOf());
context->CSSetShader(NVSharpenCS.Get(), nullptr, 0);

context->Dispatch(UINT(std::ceil(outputWidth / float(blockWidth))),
                  UINT(std::ceil(outputHeight / float(blockHeight))), 1);
```

4.1.4 Nvidia 이미지 스케일링 플러그인 통합 간소화

Streamline는 최신 NVIDIA 및 기타 독립 하드웨어 공급업체의 초고해상도 기술을 애플리케이션과 게임에 쉽게 통합할 수 있는 오픈 소스 솔루션입니다.

NIVIDA Streamline에 대한 자세한 개요는 NVIDIA 개발자 Streamline 페이지에서 확인할 수 있습니다.

Streamline SDK는 여기에서 찾을 수 있습니다. NVIDIA Streamline Github 페이지.

Streamline NIS 플러그인 통합 지침 및 설명서는 Streamline NIS 프로그래밍 가이드를 참조하세요.

4.2 샘플 코드

샘플 코드 예제는 NVIDIA 이미지 스케일링 SDK와 함께 제공됩니다. 샘플 앱은 애플리케이션에 NVScaler 또

는 NVSharpen을 통합하는 방법을 보여주는 매우 간단한 예제입니다.

샘플을 컴파일합니다:

```
$> cd samples
$> mkdir build
```

```
$> cd build
$> cmake ..
```

Visual Studio 2019로 솔루션을 엽니다. 샘플 프로젝트를 마우스 오른쪽 버튼으로 클릭하고 '시작 프로젝트로 설정'을 선택한 다음 프로젝트를 빌드합니다.

벌칸 샘플을 빌드하려면 다음 명령줄 옵션을 추가하세요.

```
$> cmake .. -DNIS_VK_SAMPLE=ON
```

NV12 샘플을 빌드하려면 다음 명령줄 옵션을 추가합니다.

```
$> cmake .. -DNIS_NV12_SAMPLE=ON
```

Streamline 샘플을 빌드하려면 먼저 NVIDA Streamline SDK를 체크아웃하고 빌드합니다. Visual Studio 버전을 지정합니다(기본값 vs2017).

```
$> cd samples\third_party
$> setup_streamline.bat vs2019
```

Streamline 빌드 프로세스가 완료된 후 cmake 명령에 다음 명령줄 옵션을 추가합니다. 명령줄 옵션을 연결하여 다른 샘플을 빌드할 수도 있습니다.

```
$> cd samples
$> cmake .. -DNIS_SL_SAMPLE=ON
```

5 모범 사례

5.1 밉맵 바이어스

애플리케이션에서 밉맵 바이어스(텍스처 LOD 바이어스라고도 함)를 0보다 낮은 값으로 설정해야 합니다. 이렇게 하면 NVScaler에서 사용하는 낮은 렌더링 해상도가 아닌 디스플레이 해상도에서 텍스처가 샘플링되므로 전반적인 이미지 품질이 향상됩니다. NVIDIA 권장 사항

```
밉레벨 바이어스 = 네이티브 바이어스 + 로그2(렌더 X 해상도/디스플레이 X 해상도) + 엡실론
```

참고: NV스케일러가 활성화된 경우 텍스처 선명도를 주의 깊게 확인하고 기본 AA 방식으로 기본 해상도에서 렌더링할 때 텍스처 선명도와 일치하는지 확인하세요. 텍스트나 기타 미세한 디테일이 있는 텍스처

(예: 벽에 붙은 포스터, 번호판, 신문 등)에 주의하세요.

기본 해상도 렌더링 중에 네거티브 바이어스가 적용된 경우 일부 아트 에셋이 기본 바이어스에 맞게 조정되었을 수 있습니다. NVScaler를 활성화하면 바이어스가 기본값에 비해 너무 크거나 작아 이미지 품질이 저하될 수 있습니다. 이러한 경우 MipLevelBias 계산의 "엡실론" 값을 조정하세요.

참고: 일부 렌더링 엔진에는 밉맵 바이어스에 대한 전역 클램프가 있습니다. 이러한 클램프가 있는 경우 NVScaler를 활성화할 때 비활성화하세요.

6 부록

6.1 공지사항

6.1.1 상표

NVIDIA 및 NVIDIA 로고는 다음 국가에서 NVIDIA Corporation의 상표 및/또는 등록 상표입니다.
미국 및 기타 국가. 기타 회사 및 제품 이름은 해당 회사 및 관련 회사의 상표일 수 있습니다.

MIT 라이선스(MIT)
Copyright(c) 2022 NVIDIA Corporation

본 소프트웨어 및 관련 문서 파일(이하 "소프트웨어")의 사본을 취득하는 모든 사람에게 다음과 같은 권리를 포함하되 이에 국한되지 않는 제한 없이 소프트웨어를 취급할 수 있는 권한이 무료로 부여됩니다.
소프트웨어의 사본을 사용, 복사, 수정, 병합, 게시, 배포, 재라이선스 및/또는 판매하고, 소프트웨어를 제공받은 자에게 다음 조건에 따라 그렇게 하도록 허용할 수 있습니다:

위의 저작권 고지 및 본 사용권 고지는 소프트웨어의 모든 사본 또는 상당 부분에 포함되어야 합니다.

소프트웨어는 상품성, 특정 목적에의 적합성 및 비침해에 대한 보증을 포함하되 이에 국한되지 않는 어떠한 종류의 명시적 또는 묵시적 보증 없이 "있는 그대로" 제공됩니다.어떠한 경우에도 저작자 또는 저작권 소유자는 소프트웨어 또는 소프트웨어의 사용 또는 기타 거래로 인해, 그로부터 또는 이와 관련하여 발생하는 계약, 불법행위 또는 기타 소송에서 어떠한 청구, 손해 또는 기타 책임에 대해서도 책임이 면제됩니다.

6.1.2 라이선스

6.2 타사 소프트웨어

MIT 라이선스(MIT)

저작권 (c) 2014-2021 오마르 코넛

이에 따라 본 소프트웨어 및 관련 문서 파일("소프트웨어")의 사본을 취득하는 모든 사람에게 소프트웨어의 사용, 복사, 수정, 병합, 게시, 배포, 재라이선스 및/또는 판매 권한을 포함하되 이에 국한되지 않고 제한 없이 소프트웨어를 취급할 수 있는 권한이 무료로 부여되며, 다음 조건에 따라 소프트웨어를 제공받는 사람에게도 그렇게 할 수 있는 권한이 허용됩니다:

위의 저작권 고지 및 본 사용권 고지는 소프트웨어의 모든 사본 또는 상당 부분에 포함되어야 합니다.

소프트웨어는 상품성, 특정 목적에의 적합성 및 비침해에 대한 보증을 포함하되 이에 국한되지 않는 어떠한 종류의 명시적 또는 묵시적 보증 없이 "있는 그대로" 제공됩니다. 어떠한 경우에도 저작자 또는 저작권 소유자는 소프트웨어 또는 소프트웨어의 사용 또는 기타 거래로 인해, 그로부터 또는 이와 관련하여 발생하는 계약, 불법행위 또는 기타 소송에서 어떠한 청구, 손해배상 또는 기타 책임에 대해서도 책임을 지지 않습니다.

6.2.1 임구이님께

6.2.2 GLFW

저작권 (c) 2002-2006 Marcus Geelnard 저작권 (c)
2006-2019 Camilla Löwy

이 소프트웨어는 어떠한 명시적 또는 묵시적 보증 없이 '있는 그대로' 제공됩니다. 어떠한 경우에도 작성자는 이 소프트웨어의 사용으로 인해 발생하는 어떠한 손해에 대해서도 책임을 지지 않습니다.

상업적 용도를 포함하여 어떤 목적으로든 이 소프트웨어를 사용하고, 변경하고, 자유롭게 재배포할 수 있는 권한은 누구에게나 부여되지만, 다음과 같은 제한 사항이 적용됩니다:

이 소프트웨어의 출처를 허위로 표시해서는 안 되며, 자신이 원본 소프트웨어를 작성했다고 주장해서는 안 됩니다. 제품에서 이 소프트웨어를 사용하는 경우 제품 설명서에 이 소프트웨어의 출처를 명시하면 감사하겠지만 필수는 아닙니다.

변경된 소스 버전은 그 사실을 명확하게 표시해야 하며, 원본 소프트웨어인 것처럼 잘못 표시해서는 안 됩니다.

이 공지는 모든 소스 배포에서 제거하거나 변경할 수 없습니다.

```
==============================================================================
LLVM 릴리스 라이선스
==============================================================================
일리노이 대학교/NCSA 오픈 소
스 라이선스

저작권 (c) 2003-2015 일리노이 대학교 어바나-샴페인 캠퍼스. 모든 권
리 보유.

개발사:

    LLVM 팀
```

6.2.3 DirectX 셰이더 컴파일러

일리노이 대학교 어바나-샴페인 캠퍼스

http://llvm.org

이에 따라 본 소프트웨어 및 관련 문서 파일("소프트웨어")의 사본을 취득하는 모든 사람에게 소프트웨어의 사용, 복사, 수정, 병합, 게시, 배포, 재라이선스 및/또는 판매 권한을 포함하되 이에 국한되지 않고 제한 없이 소프트웨어를 취급할 수 있는 권한이 무료로 부여되며, 소프트웨어를 제공받는 사람에게도 다음 조건에 따라 그렇게 할 수 있도록 허용합니다:

* 소스 코드 재배포 시에는 위의 저작권 고지, 이 조건 목록 및 다음 면책 조항을 유지해야 합니다.

* 바이너리 형식의 재배포는 배포와 함께 제공되는 문서 및/또는 기타 자료에 위의 저작권 고지, 이 조건 목록 및 다음 면책 조항을 복제해야 합니다.

* 일리노이대학교 어바나-샴페인 캠퍼스 LLVM 팀의 이름이나 기여자의 이름은 다음과 같은 용도로 사용할 수 없습니다.
구체적인 사전 서면 허가 없이 본 소프트웨어에서 파생된 제품을 보증하거나 홍보할 수 없습니다.

소프트웨어는 상품성, 특정 목적에의 적합성 및 비침해에 대한 보증을 포함하되 이에 국한되지 않는 어떠한 종류의 명시적 또는 묵시적 보증 없이 "있는 그대로" 제공됩니다. 어떠한 경우에도 기여자 또는 저작권 소유자는 소프트웨어 또는 소프트웨어의 사용 또는 기타 거래로 인해, 그로부터 또는 이와 관련하여 발생하는 계약, 불법행위 또는 기타 소송에서 어떠한 청구, 손해 또는 기타 책임에 대해서도 책임을 지지 않습니다.

==
LLVM과 함께 배포된 타사 소프트웨어의 저작권 및 라이선스:
==
LLVM 소프트웨어에는 타사에서 작성한 코드가 포함되어 있습니다. 이러한 소프트웨어는 해당 소프트웨어가 나타나는 디렉토리에 개별 LICENSE.TXT 파일을 갖게 됩니다. 이 파일에는 해당 코드에 적용되는 저작권, 라이선스 및 제한 사항이 설명되어 있습니다.

일리노이 대학교 오픈 소스 라이선스의 보증 면책 조항은 LLVM 배포판의 모든 코드에 적용되며, 다른 라이선스의 어떤 조항도 이 소프트웨어에서 파생된 제품을 보증하거나 홍보하기 위해 LLVM 팀 또는 일리노이 대학교의 이름을 사용할 수 있는 권한을 부여하지 않습니다.

다음 소프트웨어에는 추가 또는 대체 저작권, 라이선스 및/또는 제한 사항이 적용됩니다:

프로그램	디렉토리
Autoconf	llvm/autoconf

	llvm/projects/ModuleMaker/autoconf
Google 테스트	llvm/utils/unitest/googletest
OpenBSD 정규식	llvm/lib/Support/{reg*, COPYRIGHT.regex}
pyyaml 테스트	llvm/test/YAMLParser/{*.data, LICENSE.TXT}
ARM 기여	llvm/lib/Target/ARM/LICENSE.TXT
MD5 기여	llvm/lib/Support/MD5.cpp llvm/include/llvm/Support/MD5.h
miniz	llvm/lib/Miniz/miniz.c llvm/include/miniz/miniz.h llvm/lib/Miniz/LICENSE.txt

6.2.4 tinyEXR

라이선스
3절 BSD

tinyexr는 리치 겔드라이히(richgel99@gmail.com)가 개발하고 공개 도메인에 따라 라이선스가 부여된 미니즈를 사용합니다.

tinyexr 도구는 퍼블릭 도메인으로 라이선스가 부여된 stb를 사용합니다:
https://github.com/nothings/stb tinyexr는 3절 BSD 라이선스에 따라 라이선스가 부여된 OpenEXR의 일부 코드를 사용합니다.

MIT 라이선스
저작권 (c) 2017 Sean Barrett
이에 따라 본 소프트웨어 및 관련 문서 파일("소프트웨어")의 사본을 취득하는 모든 사람에게 소프트웨어의 사용, 복사, 수정, 병합, 게시, 배포, 재라이선스 및/또는 판매 권한을 포함하되 이에 국한되지 않고 제한 없이 소프트웨어를 취급할 수 있는 권한이 무료로 부여되며, 다음 조건에 따라 소프트웨어를 제공받는 사람에게도 그렇게 할 수 있는 권한이 허용됩니다:
위의 저작권 고지 및 본 사용권 고지는 소프트웨어의 모든 사본 또는 상당 부분에 포함되어야 합니다.
소프트웨어는 상품성, 특정 목적에의 적합성 및 비침해에 대한 보증을 포함하되 이에 국한되지 않는 어떠한 종류의 명시적 또는 묵시적 보증 없이 "있는 그대로" 제공됩니다. 어떠한 경우에도 저작자 또는 저작권 소유자는 소프트웨어 또는 소프트웨어의 사용 또는 기타 거래로 인해, 그로부터 또는 이와 관련하여 발생하는 계약, 불법행위 또는 기타 소송에서 어떠한 청구, 손해배상 또는 기타 책임에 대해서도 책임을 지지 않습니다.

6.2.5 stb 이미지

6.2.6 d3dx12.h

저작권 (c) Microsoft. 모든 권리 보유.
이 코드는 MIT 라이선스(MIT)에 따라 라이선스가 부여됩니다. 이 코드는 특정 목적에의 적합성, 상업성 또는 비침해성에 대한 묵시적 보증을 포함하여 명시적이든 묵시적이든 어떠한 종류의 보증도 없이 *있는 그대로* 제공됩니다.

6.2.7 간소화

저작권 (c) 2022 NVIDIA CORPORATION. 모든 권리 보유

이에 따라 본 소프트웨어 및 관련 문서 파일("소프트웨어")의 사본을 취득하는 모든 사람에게 소프트웨어의 사용, 복사, 수정, 병합, 게시, 배포, 재라이선스 및/또는 판매 권한을 포함하되 이에 국한되지 않고 제한 없이 소프트웨어를 취급할 수 있는 권한이 무료로 부여되며, 다음 조건에 따라 소프트웨어를 제공받는 사람에게도 그렇게 할 수 있는 권한이 허용됩니다:

위의 저작권 고지 및 본 사용권 고지는 소프트웨어의 모든 사본 또는 상당 부분에 포함되어야 합니다.

소프트웨어는 상품성, 특정 목적에의 적합성 및 비침해에 대한 보증을 포함하되 이에 국한되지 않는 어떠한 종류의 명시적 또는 묵시적 보증 없이 "있는 그대로" 제공됩니다. 어떠한 경우에도 저작자 또는 저작권 소유자는 소프트웨어 또는 소프트웨어의 사용 또는 기타 거래로 인해, 그로부터 또는 이와 관련하여 발생하는 계약, 불법행위 또는 기타 소송에서 어떠한 청구, 손해배상 또는 기타 책임에 대해서도 책임을 지지 않습니다.