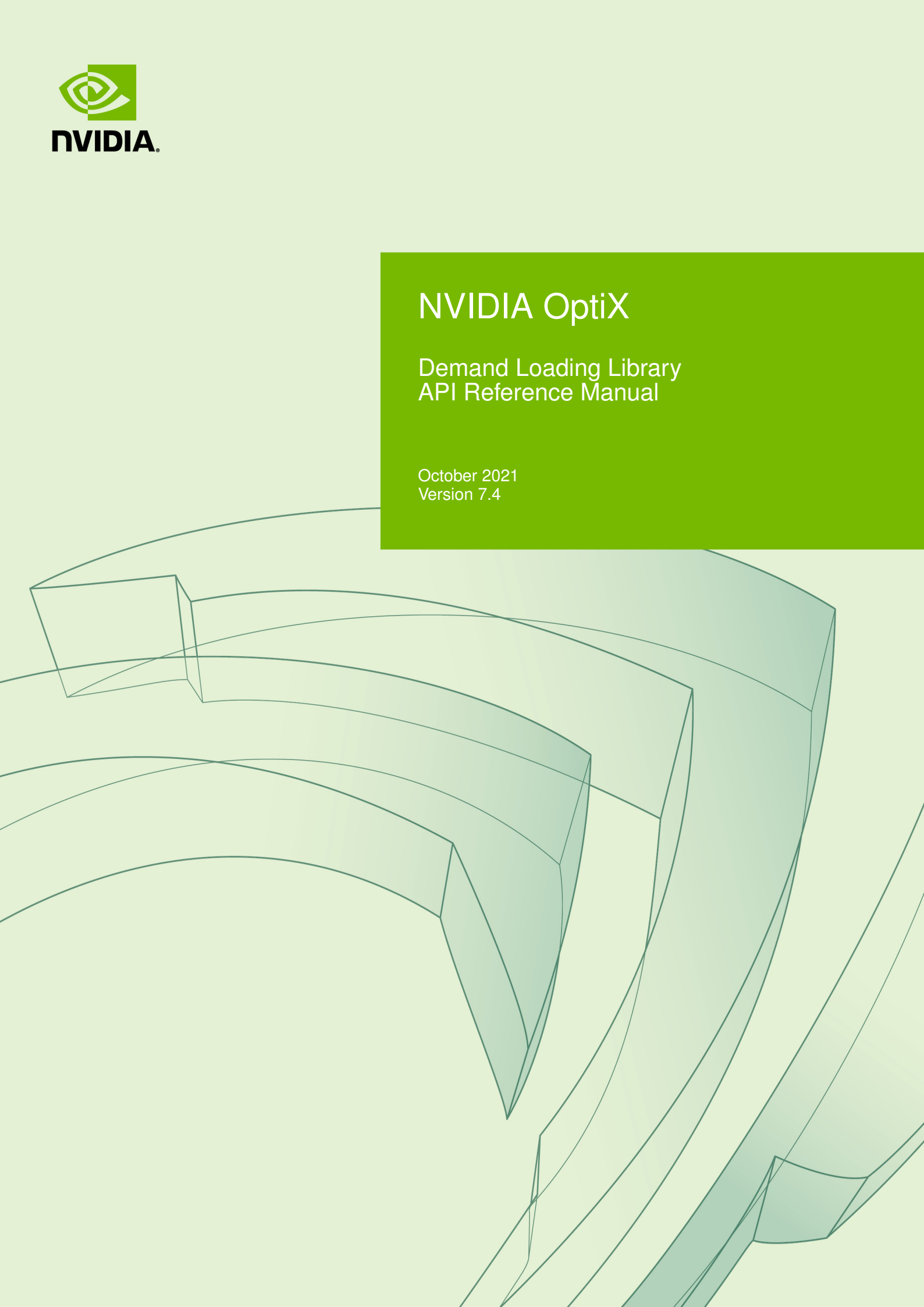


# NVIDIA OptiX

## Demand Loading Library API Reference Manual

October 2021
Version 7.4

# 1 Overview

The OptiX Demand Loading library allows hardware-accelerated sparse textures to be loaded on demand, which greatly reduces memory requirements, bandwidth, and disk I/O compared to preloading textures.

It works by maintaining a page table that tracks which texture tiles have been loaded into GPU memory. An OptiX closest-hit program fetches from a texture by calling library device code (e.g. tex2DGrad) that checks the page table to see if the required tiles are present. If not, the library records a page request, which is processed by library host code after the kernel has terminated. Kernel launches are repeated until no tiles are missing, typically using adaptive sampling to avoid redundant work. Although it is currently focused on texturing, much of the library is generic and can be adapted to load arbitrary data on demand, such as per-vertex data like colors or normals.

This document describes the Demand Loading API. For a higher-level discussion, please see the Optix Programming Guide

The DemandLoader class is the primary interface of the Demand Loading library. Here is some sample code (adapted from samples/optixDemandLoadSimple) that demonstrates how to launch a kernel that requests a demand-loaded resource:

```
00
// Create DemandLoader and a demand-loaded resource.
DemandLoader* loader = createDemandLoader( Options() );
unsigned int startPage = loader->createResource( numPages, callback );
// Prepare for launch, obtaining DeviceContext.
DeviceContext context;
loader->launchPrepare( deviceIndex, stream, context );
// Launch the kernel, using the DeviceContext to make page requests.
myKernel<<<numBlocks, threadsPerBlock, 0U, stream>>>( context );
// Initiate request processing, which returns a Ticket.
Ticket ticket = loader->processRequests( deviceIndex, stream, context );
// Wait for any page requests to be processed.
ticket.wait();
// Launch the kernel again, using the DeviceContext to locate the requested pages.
myKernel<<<numBlocks, threadsPerBlock, 0U, stream>>>( context );
// Clean up.
destroyDemandLoader( loader );
```

# 2 Namespace Index

## 2.1 Namespace List

Here is a list of all namespaces with brief descriptions:



# 3 Hierarchical Index

## 3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

# 4 Class Index

## 4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

# 5 File Index

## 5.1 File List

Here is a list of all files with brief descriptions:



# 6 Namespace Documentation

## 6.1 demandLoading Namespace Reference

### Classes

- class DemandLoader
- class DemandTexture
- struct Options
- struct Statistics
- struct Texture2DFootprint
- struct TextureDescriptor
- class Ticket

### Typedefs

- using ResourceCallback = std::function< void *(unsigned int deviceIndex, CUstream stream, unsigned int pageIndex)>

### Functions

- DemandLoader * createDemandLoader (const Options &options)
- void destroyDemandLoader (DemandLoader *manager)
- template<class TYPE >
  static __device__ __forceinline__ TYPE tex2DGrad (const DeviceContext &context, unsigned int textureId, float x, float y, float2 ddx, float2 ddy, bool *isResident, bool requestIfResident)
- template<class TYPE >
  static __device__ __forceinline__ TYPE tex2DLod (const DeviceContext &context, unsigned int textureId, float x, float y, float lod, bool *isResident, bool requestIfResident)
- template<class TYPE >
  static __device__ __forceinline__ TYPE tex2D (const DeviceContext &context, unsigned int textureId, float x, float y, bool *isResident, bool requestIfResident)
- static __device__ __forceinline__ void wrapAndSeparateUdimCoord (float x, CUaddress_mode wrapMode, unsigned int udim, float &newx, unsigned int &xidx)
- template<class TYPE >
  static __device__ __forceinline__ TYPE tex2DGradUdim (const DeviceContext &context, unsigned int textureId, float x, float y, float2 ddx, float2 ddy, bool *isResident, bool requestIfResident)
- template<class TYPE >
  static __device__ __forceinline__ TYPE tex2DGradUdimBlend (const DeviceContext &context, unsigned int textureId, float x, float y, float2 ddx, float2 ddy, bool *isResident, bool requestIfResident)

## 6.1.1 Typedef Documentation

### 6.1.1.1 ResourceCallback

```
using demandLoading::ResourceCallback = typedef std::function<void*(
unsigned int deviceIndex, CUstream stream, unsigned int pageIndex )>
```

ResourceCallback is a user-provided function that fills requests for pages in arbitrary demand-loaded buffers. It takes three arguments: an integer device index, a stream, and an integer page index. It returns the new page table entry for the requested page, which is typically a device pointer (but it can be an arbitrary 64-bit value).

## 6.1.2 Function Documentation

### 6.1.2.1 createDemandLoader()

```
DemandLoader* demandLoading::createDemandLoader (
        const Options & options )
```

Create a DemandLoader with the given options.

### 6.1.2.2 destroyDemandLoader()

```
void demandLoading::destroyDemandLoader (
        DemandLoader * manager )
```

Function to destroy a DemandLoader.

#### 6.1.2.3 tex2D() template<class TYPE >

```
static __device__ __forceinline__ TYPE demandLoading::tex2D (
            const DeviceContext & context,
            unsigned int textureId,
            float x,
            float y,
            bool * isResident,
            bool requestIfResident )  [static]
```

Fetch from a demand-loaded texture with the specified identifer, obtained via DemandLoader::createTexture. The given DeviceContext is typically a launch parameter, obtained via DemandLoader::launchPrepare, that has been copied to device memory.

#### 6.1.2.4 tex2DGrad() template<class TYPE >

```
static __device__ __forceinline__ TYPE demandLoading::tex2DGrad (
            const DeviceContext & context,
            unsigned int textureId,
            float x,
            float y,
            float2 ddx,
            float2 ddy,
            bool * isResident,
            bool requestIfResident )  [static]
```

Fetch from a demand-loaded texture with the specified identifer, obtained via DemandLoader::createTexture. The given DeviceContext is typically a launch parameter, obtained via DemandLoader::launchPrepare, that has been copied to device memory.

#### 6.1.2.5 tex2DGradUdim() template<class TYPE >

```
static __device__ __forceinline__ TYPE demandLoading::tex2DGradUdim (
            const DeviceContext & context,
            unsigned int textureId,
            float x,
            float y,
            float2 ddx,
            float2 ddy,
            bool * isResident,
            bool requestIfResident )  [static]
```

Fetch from a demand-loaded udim texture. A "udim" texture is an array of texture images that are treated as a single texture object (with an optional base texture). This entry point is fast, but assumes that texture samples will not cross subtexture boundaries. When using this entry point, use CU_TR_ADDRESS_MODE_CLAMP when defining all subtextures to prevent dark lines between textures.

#### 6.1.2.6 tex2DGradUdimBlend() template<class TYPE >

```
static __device__ __forceinline__ TYPE demandLoading::tex2DGradUdimBlend (
            const DeviceContext & context,
            unsigned int textureId,
            float x,
            float y,
            float2 ddx,
            float2 ddy,
            bool * isResident,
            bool requestIfResident )  [static]
```

Fetch from demand-loaded udim texture. A "udim" texture is an array of texture images that are treated as a single texture object (with an optional base texture). This entry point will combine multiple samples to blend across sub-texture boundaries.
For proper blending, use CU_TR_ADDRESS_MODE_BORDER when defining all subtextures. Other blending modes will show lines between subtextures.

#### 6.1.2.7 tex2DLod() template<class TYPE >

```
static __device__ __forceinline__ TYPE demandLoading::tex2DLod (
            const DeviceContext & context,
            unsigned int textureId,
            float x,
            float y,
            float lod,
            bool * isResident,
            bool requestIfResident )  [static]
```

Fetch from a demand-loaded texture with the specified identifer, obtained via DemandLoader::createTexture. The given DeviceContext is typically a launch parameter, obtained via DemandLoader::launchPrepare, that has been copied to device memory.

#### 6.1.2.8 wrapAndSeparateUdimCoord() static __device__ __forceinline__ void demandLoading↩ ::wrapAndSeparateUdimCoord (

```
            float x,
            CUaddress_mode wrapMode,
            unsigned int udim,
            float & newx,
            unsigned int & xidx )  [static]
```

## 6.2 imageReader Namespace Reference

### Classes

- class EXRReader
- class ImageReader
- class MipTailImageReader

# 7 Class Documentation

## 7.1 demandLoading::DemandLoader Class Reference

### Public Member Functions

- virtual ~DemandLoader ()=default
- virtual const DemandTexture & createTexture (std::shared_ptr< imageReader::ImageReader > image, const TextureDescriptor &textureDesc)=0
- virtual const DemandTexture & createUdimTexture (std::vector< std::shared_ptr< imageReader::ImageReader >> &imageReaders, std::vector< TextureDescriptor > &textureDescs, unsigned int udim, unsigned int vdim, int baseTextureId)=0
- virtual unsigned int createResource (unsigned int numPages, ResourceCallback callback)=0
- virtual bool launchPrepare (unsigned int deviceIndex, CUstream stream, DeviceContext &context)=0
- virtual Ticket processRequests (unsigned int deviceIndex, CUstream stream, const DeviceContext &deviceContext)=0
- virtual Statistics getStatistics () const =0
- virtual const std::vector< unsigned int > getDevices () const =0

#### 7.1.1 Detailed Description

DemandLoader loads sparse textures on demand.

#### 7.1.2 Constructor & Destructor Documentation

##### 7.1.2.1 ~DemandLoader() `virtual demandLoading::DemandLoader::~DemandLoader ( ) [virtual], [default]`

Base class destructor.

#### 7.1.3 Member Function Documentation

##### 7.1.3.1 createResource() `virtual unsigned int demandLoading::DemandLoader::createResource ( unsigned int numPages, ResourceCallback callback ) [pure virtual]`

Create an arbitrary resource with the specified number of pages.

**See also**

ResourceCallback. Returns the starting index of the resource in the page table.

#### 7.1.3.2 createTexture()

```
virtual const DemandTexture& demandLoading::DemandLoader::createTexture
(
            std::shared_ptr< imageReader::ImageReader > image,
            const TextureDescriptor & textureDesc )  [pure virtual]
```

Create a demand-loaded texture for the given image. The texture initially has no backing storage. The readTile() method is invoked on the image to fill each required tile. The ImageReader pointer is retained for the lifetime of the DemandLoader.

#### 7.1.3.3 createUdimTexture()

```
virtual const DemandTexture& demandLoading::DemandLoader::createUdimTexture (
            std::vector< std::shared_ptr< imageReader::ImageReader >> & imageReaders,
            std::vector< TextureDescriptor > & textureDescs,
            unsigned int udim,
            unsigned int vdim,
            int baseTextureId )  [pure virtual]
```

Create a demand-loaded UDIM texture for a given set of images. If a baseTexture is used, it should be created first by calling createTexture. The id of the returned texture should be used when calling tex2DGradUdim. All of the image readers are retained for the lifetime of the DemandLoader.

#### 7.1.3.4 getDevices()

```
virtual const std::vector<unsigned int> demandLoading::DemandLoader::getDevices ( ) const  [pure virtual]
```

Get indices of the devices that can be employed by the DemandLoader (i.e. those that support sparse textures).

#### 7.1.3.5 getStatistics()

```
virtual Statistics demandLoading::DemandLoader::getStatistics ( ) const [pure virtual]
```

Get current statistics.

#### 7.1.3.6 launchPrepare()

```
virtual bool demandLoading::DemandLoader::launchPrepare (
            unsigned int deviceIndex,
            CUstream stream,
            DeviceContext & context )  [pure virtual]
```

Prepare for launch. Returns false if the specified device does not support sparse textures. If successful, returns a DeviceContext via result parameter, which should be copied to device memory (typically along with OptiX kernel launch parameters), so that it can be passed to Tex2D().

#### 7.1.3.7 processRequests()

```
virtual Ticket demandLoading::DemandLoader::processRequests (
        unsigned int deviceIndex,
        CUstream stream,
        const DeviceContext & deviceContext )  [pure virtual]
```

Fetch page requests from the given device context and enqueue them for background processing. The given DeviceContext must reside in host memory. The given stream is used when copying tile data to the device. Returns a ticket that is notified when the requests have been filled on the host side.

## 7.2 demandLoading::DemandTexture Class Reference

### Public Member Functions

- virtual ~DemandTexture ()=default
- virtual unsigned int getId () const =0

### 7.2.1 Detailed Description

Demand-loaded textures are created and owned by the DemandLoader. The methods may be called from multiple threads; the implementation must be threadsafe.

### 7.2.2 Constructor & Destructor Documentation

#### 7.2.2.1 ~DemandTexture()

```
virtual demandLoading::DemandTexture::~DemandTexture ( )  [virtual], [default]
```

Default destructor.

### 7.2.3 Member Function Documentation

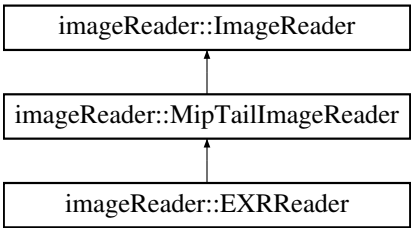
#### 7.2.3.1 getId()

```
virtual unsigned int demandLoading::DemandTexture::getId ( ) const  [pure virtual]
```

Get the texture id, which is used as an index into the device-side sampler array.

## 7.3 imageReader::EXRReader Class Reference

Inheritance diagram for imageReader::EXRReader:



### Public Member Functions

- EXRReader (const char *filename, bool readBaseColor=true)
- ~EXRReader () override
- bool open (TextureInfo *info) override
- void close () override
- const TextureInfo & getInfo () override
- bool readTile (char *dest, unsigned int mipLevel, unsigned int tileX, unsigned int tileY, unsigned int tileWidth, unsigned int tileHeight) override
- bool readMipLevel (char *dest, unsigned int mipLevel, unsigned int expectedWidth, unsigned int expectedHeight) override
- virtual bool readBaseColor (float4 &dest) override
- unsigned int getTileWidth () const
- unsigned int getTileHeight () const
- unsigned long long getNumTilesRead () const override
- unsigned long long getNumBytesRead () const override
- double getTotalReadTime () const override
- void serialize (std::ostream &stream) const

### Static Public Member Functions

- static std::shared_ptr< ImageReader > deserialize (std::istream &stream)

#### 7.3.1 Detailed Description

OpenEXR image reader.

#### 7.3.2 Constructor & Destructor Documentation

##### 7.3.2.1 EXRReader() `imageReader::EXRReader::EXRReader ( const char * filename, bool readBaseColor = true ) [inline], [explicit]`

The constructor copies the given filename. The file is not opened until open() is called.

##### 7.3.2.2 ~EXRReader() `imageReader::EXRReader::~EXRReader ( ) [inline], [override]`

Destructor.

#### 7.3.3 Member Function Documentation

#### 7.3.3.1 close() `void imageReader::EXRReader::close ( ) [override], [virtual]`

Close the image.

Implements imageReader::ImageReader.

#### 7.3.3.2 deserialize() `static std::shared_ptr<ImageReader> imageReader::EXRReader::deserialize ( std::istream & stream ) [static]`

Deserialize an EXRReader. Called from ImageReader::deserialize.

#### 7.3.3.3 getInfo() `const TextureInfo& imageReader::EXRReader::getInfo ( ) [inline], [override], [virtual]`

Get the image info. Valid only after calling open().

Implements imageReader::ImageReader.

#### 7.3.3.4 getNumBytesRead() `unsigned long long imageReader::EXRReader::getNumBytesRead ( ) const [inline], [override], [virtual]`

Returns the number of bytes that have been read.

Reimplemented from imageReader::ImageReader.

#### 7.3.3.5 getNumTilesRead() `unsigned long long imageReader::EXRReader::getNumTilesRead ( ) const [inline], [override], [virtual]`

Returns the number of tiles that have been read.

Reimplemented from imageReader::ImageReader.

#### 7.3.3.6 getTileHeight() `unsigned int imageReader::EXRReader::getTileHeight ( ) const [inline]`

Get tile height (used only for testing).

#### 7.3.3.7 getTileWidth() `unsigned int imageReader::EXRReader::getTileWidth ( ) const [inline]`

Get tile width (used only for testing).

#### 7.3.3.8 getTotalReadTime() `double imageReader::EXRReader::getTotalReadTime ( ) const [inline], [override], [virtual]`

Returns the time in seconds spent reading image tiles.

Reimplemented from imageReader::ImageReader.

#### 7.3.3.9 open() `bool imageReader::EXRReader::open ( TextureInfo * info ) [override], [virtual]`

Open the image and read header info, including dimensions and format. Returns false on error.

Implements imageReader::ImageReader.

#### 7.3.3.10 readBaseColor() `virtual bool imageReader::EXRReader::readBaseColor ( float4 & dest ) [override], [virtual]`

Read the base color of the image (1x1 mip level) as an array of floats. Returns true on success.

Implements imageReader::ImageReader.

#### 7.3.3.11 readMipLevel() `bool imageReader::EXRReader::readMipLevel ( char * dest, unsigned int mipLevel, unsigned int expectedWidth, unsigned int expectedHeight ) [override], [virtual]`

Read the specified mipLevel. Returns true for success.

Implements imageReader::ImageReader.

#### 7.3.3.12 readTile() `bool imageReader::EXRReader::readTile ( char * dest, unsigned int mipLevel, unsigned int tileX, unsigned int tileY, unsigned int tileWidth, unsigned int tileHeight ) [override], [virtual]`

Read the specified tile or mip level, returning the data in dest. dest must be large enough to hold the tile. Pixels outside the bounds of the mip level will be filled in with black.

Implements imageReader::ImageReader.

#### 7.3.3.13 serialize() `void imageReader::EXRReader::serialize ( std::ostream & stream ) const`

Serialize the image filename (etc.) to the give stream.

## 7.4 half4 Struct Reference

### Public Attributes

- half x
- half y
- half z
- half w

### 7.4.1 Member Data Documentation

#### 7.4.1.1 w `half half4::w`

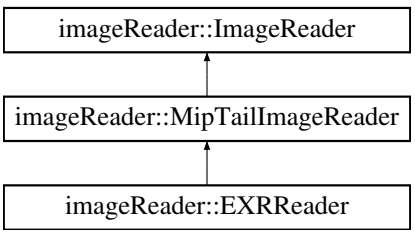
#### 7.4.1.2 x `half half4::x`

#### 7.4.1.3 y `half half4::y`

#### 7.4.1.4 z `half half4::z`

## 7.5 imageReader::ImageReader Class Reference

Inheritance diagram for imageReader::ImageReader:



### Public Member Functions

- virtual ~ImageReader ()=default
- virtual bool open (TextureInfo *info)=0
- virtual void close ()=0
- virtual const TextureInfo & getInfo ()=0
- virtual bool readTile (char *dest, unsigned int mipLevel, unsigned int tileX, unsigned int tileY, unsigned int tileWidth, unsigned int tileHeight)=0
- virtual bool readMipLevel (char *dest, unsigned int mipLevel, unsigned int expectedWidth, unsigned int expectedHeight)=0
- virtual bool readMipTail (char *dest, unsigned int mipTailFirstLevel, unsigned int numMipLevels, const uint2 *mipLevelDims, unsigned int pixelSizeInBytes)=0
- virtual bool readBaseColor (float4 &dest)=0
- virtual unsigned long long getNumTilesRead () const
- virtual unsigned long long getNumBytesRead () const
- virtual double getTotalReadTime () const

#### 7.5.1 Detailed Description

Interface for a mipmapped image.

Any method may be called from multiple threads; the implementation must be threadsafe.

#### 7.5.2 Constructor & Destructor Documentation

##### 7.5.2.1 ~ImageReader() `virtual imageReader::ImageReader::~ImageReader ( ) [virtual], [default]`

The destructor is virtual to ensure that instances of derived classes are properly destroyed.

#### 7.5.3 Member Function Documentation

##### 7.5.3.1 close() `virtual void imageReader::ImageReader::close ( ) [pure virtual]`

Close the image.

Implemented in imageReader::EXRReader.

##### 7.5.3.2 getInfo() `virtual const TextureInfo& imageReader::ImageReader::getInfo ( ) [pure virtual]`

Get the image info. Valid only after calling open().

Implemented in imageReader::EXRReader.

#### 7.5.3.3 getNumBytesRead() `virtual unsigned long long imageReader::ImageReader::getNumBytesRead ( ) const [inline], [virtual]`

Returns the number of bytes that have been read. This number may be zero if the reader does not load tiles from disk, e.g. for procedural textures.

Reimplemented in imageReader::EXRReader.

#### 7.5.3.4 getNumTilesRead() `virtual unsigned long long imageReader::ImageReader::getNumTilesRead ( ) const [inline], [virtual]`

Returns the number of tiles that have been read.

Reimplemented in imageReader::EXRReader.

#### 7.5.3.5 getTotalReadTime() `virtual double imageReader::ImageReader::getTotalReadTime ( ) const [inline], [virtual]`

Returns the time in seconds spent reading image data (tiles or mip levels). This number may be zero if the reader does not load tiles from disk, e.g. for procedural textures.

Reimplemented in imageReader::EXRReader.

#### 7.5.3.6 open() `virtual bool imageReader::ImageReader::open ( TextureInfo * info ) [pure virtual]`

Open the image and read header info, including dimensions and format. Returns false on error.

Implemented in imageReader::EXRReader.

#### 7.5.3.7 readBaseColor() `virtual bool imageReader::ImageReader::readBaseColor ( float4 & dest ) [pure virtual]`

Read the base color of the image (1x1 mip level) as a float4. Returns true on success.

Implemented in imageReader::EXRReader.

#### 7.5.3.8 readMipLevel()

```
virtual bool imageReader::ImageReader::readMipLevel (
        char * dest,
        unsigned int mipLevel,
        unsigned int expectedWidth,
        unsigned int expectedHeight )  [pure virtual]
```

Read the specified mipLevel. Returns true for success.

Implemented in imageReader::EXRReader.

#### 7.5.3.9 readMipTail()

```
virtual bool imageReader::ImageReader::readMipTail (
        char * dest,
        unsigned int mipTailFirstLevel,
        unsigned int numMipLevels,
        const uint2 * mipLevelDims,
        unsigned int pixelSizeInBytes )  [pure virtual]
```

Read the mip tail into the given buffer, starting with the specified level. An array containing the expected dimensions of all the miplevels is provided (starting from miplevel zero), along with the pixel size. Returns true for success.

Implemented in imageReader::MipTailImageReader.

#### 7.5.3.10 readTile()

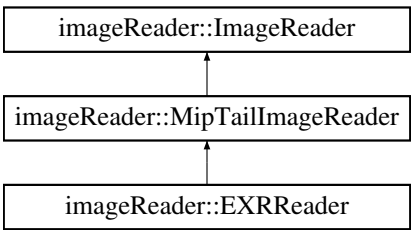
```
virtual bool imageReader::ImageReader::readTile (
        char * dest,
        unsigned int mipLevel,
        unsigned int tileX,
        unsigned int tileY,
        unsigned int tileWidth,
        unsigned int tileHeight )  [pure virtual]
```

Read the specified tile or mip level, returning the data in dest. dest must be large enough to hold the tile. Pixels outside the bounds of the mip level will be filled in with black.

Implemented in imageReader::EXRReader.

## 7.6 imageReader::MipTailImageReader Class Reference

Inheritance diagram for imageReader::MipTailImageReader:



### Public Member Functions

- virtual ~MipTailImageReader ()=default
- bool readMipTail (char *dest, unsigned int mipTailFirstLevel, unsigned int numMipLevels, const uint2 *mip↩ LevelDims, unsigned int pixelSizeInBytes) override

#### 7.6.1 Detailed Description

Abstract base class for ImageReaders that use a common implementation of readMipTail.

#### 7.6.2 Constructor & Destructor Documentation

##### 7.6.2.1 ~MipTailImageReader() `virtual imageReader::MipTailImageReader::~MipTailImageReader ( ) [virtual], [default]`

#### 7.6.3 Member Function Documentation

##### 7.6.3.1 readMipTail() `bool imageReader::MipTailImageReader::readMipTail ( char * dest, unsigned int mipTailFirstLevel, unsigned int numMipLevels, const uint2 * mipLevelDims, unsigned int pixelSizeInBytes ) [override], [virtual]`

Read the mip tail into the given buffer, starting with the specified level. An array containing the expected dimensions of all the miplevels is provided (starting from miplevel zero), along with the pixel size. Returns true for success.

Implements imageReader::ImageReader.

## 7.7 demandLoading::Options Struct Reference

### Public Attributes

- unsigned int numPages = 1 << 26
- unsigned int numPageTableEntries = 256 * 1024
- unsigned int maxRequestedPages = 8192
- unsigned int maxFilledPages = 8192
- unsigned int maxStalePages = 8192
- unsigned int maxEvictablePages = 8192
- unsigned int maxInvalidatedPages = 8192
- unsigned int maxStagedPages = 8192
- bool useLruTable = true
- size_t maxTexMemPerDevice = 0
- size_t maxPinnedMemory = 640 * 1024 * 1024
- unsigned int maxThreads = 0
- unsigned int maxActiveStreams = 4
- std::string traceFile = ""

### 7.7.1 Detailed Description

Demand loading configuration options.

**See also**

createDemandLoader

### 7.7.2 Member Data Documentation

#### 7.7.2.1 maxActiveStreams `unsigned int demandLoading::Options::maxActiveStreams = 4`

number of active streams across all devices.

#### 7.7.2.2 maxEvictablePages `unsigned int demandLoading::Options::maxEvictablePages = 8192`

max evictable pages to pull from device

#### 7.7.2.3 maxFilledPages `unsigned int demandLoading::Options::maxFilledPages = 8192`

num slots to push mappings back to device

#### 7.7.2.4 maxInvalidatedPages `unsigned int demandLoading::Options::maxInvalidatedPages = 8192`

max slots to push invalidated pages back to device

#### 7.7.2.5 maxPinnedMemory `size_t demandLoading::Options::maxPinnedMemory = 640 * 1024 * 1024`

max pinned memory.

#### 7.7.2.6 maxRequestedPages `unsigned int demandLoading::Options::maxRequestedPages = 8192`

max requests to pull from device

#### 7.7.2.7 maxStagedPages `unsigned int demandLoading::Options::maxStagedPages = 8192`

max staged pages (pages ready to be evicted)

#### 7.7.2.8 maxStalePages `unsigned int demandLoading::Options::maxStalePages = 8192`

max stale pages to pull from device

#### 7.7.2.9 maxTexMemPerDevice `size_t demandLoading::Options::maxTexMemPerDevice = 0`

max texture data to be allocated per device (0 is unlimited)

#### 7.7.2.10 maxThreads `unsigned int demandLoading::Options::maxThreads = 0`

max number of threads to use when processing requests; zero means use std::thread::hardware_concurrency.

#### 7.7.2.11 numPages `unsigned int demandLoading::Options::numPages = 1 << 26`

max virtual pages (approx. 4 TB of texture tiles)

#### 7.7.2.12 numPageTableEntries `unsigned int demandLoading::Options::numPageTableEntries = 256 * 1024`

page table entries are needed for samplers, but not tiles.

#### 7.7.2.13 traceFile `std::string demandLoading::Options::traceFile = ""`

trace filename (empty if disabled).

#### 7.7.2.14 useLruTable `bool demandLoading::Options::useLruTable = true`

use LRU table for eviction

## 7.8 demandLoading::Statistics Struct Reference

### Public Attributes

- double requestProcessingTime
- size_t numTilesRead
- size_t numBytesRead
- double readTime
- size_t memoryUsedPerDevice [16]

### 7.8.1 Detailed Description

Demand loading statistics.

**See also**

DemandLoader::getStatistics

### 7.8.2 Member Data Documentation

#### 7.8.2.1 memoryUsedPerDevice `size_t demandLoading::Statistics::memoryUsedPerDevice[16]`

Amount of device memory allocated per device.

#### 7.8.2.2 numBytesRead `size_t demandLoading::Statistics::numBytesRead`

Number of bytes read from disk by all ImageReaders.

#### 7.8.2.3 numTilesRead `size_t demandLoading::Statistics::numTilesRead`

Total number of tiles read by all ImageReaders.

#### 7.8.2.4 readTime `double demandLoading::Statistics::readTime`

Total time in seconds spent reading image data by all ImageReaders. This is the cumulative time and does not take into account simultaneous reads, e.g. by multiple threads.

#### 7.8.2.5 requestProcessingTime `double demandLoading::Statistics::requestProcessingTime`

Time in seconds spent processing page requests.

## 7.9 demandLoading::Texture2DFootprint Struct Reference

### Public Attributes

- unsigned long long mask
- unsigned int tileY: 12
- unsigned int reserved1: 4
- unsigned int dx: 3
- unsigned int dy: 3
- unsigned int reserved2: 2
- unsigned int granularity: 4
- unsigned int reserved3: 4
- unsigned int tileX: 12
- unsigned int level: 4
- unsigned int reserved4: 16

### 7.9.1 Detailed Description

Texture2DFootprint is binary compatible with the uint4 returned by the texture footprint intrinsics.

See optixTexFootprint2DGrad (etc.) in the OptiX API documentation. ( https://raytracing-docs.↩nvidia.com/optix7/api/html/index.html)

### 7.9.2 Member Data Documentation

#### 7.9.2.1 dx `unsigned int demandLoading::Texture2DFootprint::dx`

X rotation of mask relative to anchor tile. Mask starts at 8∗tileX-dx in texel group coordinates.

#### 7.9.2.2 dy `unsigned int demandLoading::Texture2DFootprint::dy`

Y rotation of mask relative to anchor tile. Mask starts at 8∗tileY-dy in texel group coordinates.

#### 7.9.2.3 granularity `unsigned int demandLoading::Texture2DFootprint::granularity`

enum giving texel group size. 0 indicates "same size as requested"

#### 7.9.2.4 level `unsigned int demandLoading::Texture2DFootprint::level`

mip level

#### 7.9.2.5 mask `unsigned long long demandLoading::Texture2DFootprint::mask`

Toroidally rotated 8x8 texel group mask to store footprint coverage.

#### 7.9.2.6 reserved1 `unsigned int demandLoading::Texture2DFootprint::reserved1`

not used

#### 7.9.2.7 reserved2 `unsigned int demandLoading::Texture2DFootprint::reserved2`

not used

#### 7.9.2.8 reserved3 `unsigned int demandLoading::Texture2DFootprint::reserved3`

not used

#### 7.9.2.9 reserved4 `unsigned int demandLoading::Texture2DFootprint::reserved4`

not used

#### 7.9.2.10 tileX `unsigned int demandLoading::Texture2DFootprint::tileX`

X position of anchor tile.

#### 7.9.2.11 tileY `unsigned int demandLoading::Texture2DFootprint::tileY`

Y position of anchor tile. Tiles are 8x8 blocks of texel groups.

## 7.10 demandLoading::TextureDescriptor Struct Reference

### Public Attributes

- CUaddress_mode addressMode [2] = {CU_TR_ADDRESS_MODE_WRAP, CU_TR_ADDRESS_MODE_↩ WRAP}
- CUfilter_mode filterMode = CU_TR_FILTER_MODE_LINEAR
- CUfilter_mode mipmapFilterMode = CU_TR_FILTER_MODE_LINEAR
- unsigned int maxAnisotropy = 16
- unsigned int flags = CU_TRSF_DISABLE_TRILINEAR_OPTIMIZATION

### 7.10.1 Detailed Description

TextureDescriptor specifies the address mode (e.g. wrap vs. clamp), filter mode (point vs. linear), etc.

### 7.10.2 Member Data Documentation

#### 7.10.2.1 addressMode `CUaddress_mode demandLoading::TextureDescriptor::addressMode[2] = {CU_↩ TR_ADDRESS_MODE_WRAP, CU_TR_ADDRESS_MODE_WRAP}`

Address mode (e.g. wrap)

#### 7.10.2.2 filterMode `CUfilter_mode demandLoading::TextureDescriptor::filterMode = CU_TR_FILTER↩ _MODE_LINEAR`

Filter mode (e.g. linear vs. point)

#### 7.10.2.3 flags `unsigned int demandLoading::TextureDescriptor::flags = CU_TRSF_DISABLE_TRILINE↩ AR_OPTIMIZATION`

CUDA texture flags. Use 0 to enable trilinear optimization (off by default).

#### 7.10.2.4 maxAnisotropy `unsigned int demandLoading::TextureDescriptor::maxAnisotropy = 16`

Maximum anisotropy. A value of 1 disables anisotropic filtering.

#### 7.10.2.5 mipmapFilterMode `CUfilter_mode demandLoading::TextureDescriptor::mipmapFilterMode = CU_TR_FILTER_MODE_LINEAR`

Filter mode between miplevels (e.g. linear vs. point)

## 7.11 demandLoading::Ticket Class Reference

### Public Member Functions

- Ticket ()
- int numTasksTotal () const
- int numTasksRemaining () const
- void wait (CUevent *event=nullptr)

### Friends

- class TicketImpl

### 7.11.1 Detailed Description

A Ticket tracks the progress of a number of tasks.

### 7.11.2 Constructor & Destructor Documentation

#### 7.11.2.1 Ticket() `demandLoading::Ticket::Ticket ( ) [inline]`

A default-constructed ticket has no tasks.

### 7.11.3 Member Function Documentation

#### 7.11.3.1 numTasksRemaining() `int demandLoading::Ticket::numTasksRemaining ( ) const`

Get the number of tasks remaining, if known. Returns -1 if the number of tasks is unknown, which indicates that task processing has not yet started.

#### 7.11.3.2 numTasksTotal() `int demandLoading::Ticket::numTasksTotal ( ) const`

Get the total number of tasks tracked by this ticket, if known. Returns -1 if the number of tasks is unknown, which indicates that task processing has not yet started.

#### 7.11.3.3 wait() `void demandLoading::Ticket::wait (`

`CUevent * event = nullptr )`

Wait for the host-side execution of the tasks to finish. Optionally, if a CUDA event is provided, it is recorded when the last task is finished, allowing the caller to wait for device-side execution to finish (e.g. via cuEventSynchronize or cuStreamWaitEvent).

### 7.11.4 Friends And Related Function Documentation

#### 7.11.4.1 TicketImpl `friend class TicketImpl [friend]`

# 8 File Documentation

## 8.1 DemandLoader.h File Reference

### Classes

- class demandLoading::DemandLoader

### Namespaces

- imageReader
- demandLoading

### Functions

- DemandLoader * demandLoading::createDemandLoader (const Options &options)
- void demandLoading::destroyDemandLoader (DemandLoader *manager)

### 8.1.1 Detailed Description

Primary interface of the Demand Loading library.

## 8.2 DemandTexture.h File Reference

### Classes

- class demandLoading::DemandTexture

### Namespaces

- demandLoading

### 8.2.1 Detailed Description

Opaque handle for demand-loaded sparse texture.

## 8.3 EXRReader.h File Reference

#### Classes

- class imageReader::EXRReader

#### Namespaces

- imageReader

### 8.3.1 Detailed Description

OpenEXR image reader.

## 8.4 ImageReader.h File Reference

#### Classes

- class imageReader::ImageReader
- class imageReader::MipTailImageReader

#### Namespaces

- imageReader

### 8.4.1 Detailed Description

Interface for a mipmapped image.

## 8.5 Options.h File Reference

#### Classes

- struct demandLoading::Options

#### Namespaces

- demandLoading

### 8.5.1 Detailed Description

Demand loading configuration options.

## 8.6 overview.txt File Reference

### Namespaces

- demandLoading

## 8.7 Resource.h File Reference

### Namespaces

- demandLoading

### Typedefs

- using demandLoading::ResourceCallback = std::function< void *(unsigned int deviceIndex, CUstream stream, unsigned int pageIndex)>

### 8.7.1 Detailed Description

Definitions for demand-loaded resources.

## 8.8 Statistics.h File Reference

### Classes

- struct demandLoading::Statistics

### Namespaces

- demandLoading

### 8.8.1 Detailed Description

Demand loading statistics.

## 8.9 Texture2D.h File Reference

### Classes

- struct half4
- struct demandLoading::Texture2DFootprint

### Namespaces

- demandLoading

### Functions

- template<class TYPE >
  static __device__ __forceinline__ TYPE demandLoading::tex2DGrad (const DeviceContext &context, unsigned int textureId, float x, float y, float2 ddx, float2 ddy, bool *isResident, bool requestIfResident)
- template<class TYPE >
  static __device__ __forceinline__ TYPE demandLoading::tex2DLod (const DeviceContext &context, unsigned int textureId, float x, float y, float lod, bool *isResident, bool requestIfResident)
- template<class TYPE >
  static __device__ __forceinline__ TYPE demandLoading::tex2D (const DeviceContext &context, unsigned int textureId, float x, float y, bool *isResident, bool requestIfResident)

#### 8.9.1 Detailed Description

Device code for fetching from demand-loaded sparse textures.

## 8.10 Texture2DExtended.h File Reference

### Namespaces

- demandLoading

### Functions

- static __device__ __forceinline__ void demandLoading::wrapAndSeparateUdimCoord (float x, CUaddress↩
  _mode wrapMode, unsigned int udim, float &newx, unsigned int &xidx)
- template<class TYPE >
  static __device__ __forceinline__ TYPE demandLoading::tex2DGradUdim (const DeviceContext &context, unsigned int textureId, float x, float y, float2 ddx, float2 ddy, bool *isResident, bool requestIfResident)
- template<class TYPE >
  static __device__ __forceinline__ TYPE demandLoading::tex2DGradUdimBlend (const DeviceContext &context, unsigned int textureId, float x, float y, float2 ddx, float2 ddy, bool *isResident, bool requestIfResident)

#### 8.10.1 Detailed Description

Extended device-side entry points for fetching from demand-loaded sparse textures.

## 8.11 Texture2DFootprint.h File Reference

### Classes

- struct demandLoading::Texture2DFootprint

#### Namespaces

- demandLoading

## 8.12 TextureDescriptor.h File Reference

#### Classes

- struct demandLoading::TextureDescriptor

#### Namespaces

- demandLoading

### 8.12.1 Detailed Description

TextureDescriptor specifies address mode, filter mode, etc.

## 8.13 Ticket.h File Reference

#### Classes

- class demandLoading::Ticket

#### Namespaces

- demandLoading

### 8.13.1 Detailed Description

A Ticket tracks the progress of a number of tasks.