NAME

Texture Object Management −

Functions

cudaError_t cudaCreateTextureObject (**cudaTextureObject_t** *pTexObject, const struct **cudaResourceDesc** *pResDesc, const struct **cudaTextureDesc** *pTexDesc, const struct **cudaResourceViewDesc** *pResViewDesc)
Creates a texture object.

cudaError_t cudaDestroyTextureObject (**cudaTextureObject_t** texObject)
Destroys a texture object.

cudaError_t cudaGetTextureObjectResourceDesc (struct **cudaResourceDesc** *pResDesc, **cudaTextureObject_t** texObject)
Returns a texture object's resource descriptor.

cudaError_t cudaGetTextureObjectResourceViewDesc (struct **cudaResourceViewDesc** *pResViewDesc, **cudaTextureObject_t** texObject)
Returns a texture object's resource view descriptor.

cudaError_t cudaGetTextureObjectTextureDesc (struct **cudaTextureDesc** *pTexDesc, **cudaTextureObject_t** texObject)
Returns a texture object's texture descriptor.

Detailed Description

\brief texture object management functions of the CUDA runtime API (cuda_runtime_api.h)

This section describes the low level texture object management functions of the CUDA runtime application programming interface. The texture object API is only supported on devices of compute capability 3.0 or higher.

Function Documentation

cudaError_t cudaCreateTextureObject (cudaTextureObject_t * pTexObject, const struct cudaResourceDesc * pResDesc, const struct cudaTextureDesc * pTexDesc, const struct cudaResourceViewDesc * pResViewDesc)

Creates a texture object and returns it in `pTexObject`. `pResDesc` describes the data to texture from. `pTexDesc` describes how the data should be sampled. `pResViewDesc` is an optional argument that specifies an alternate format for the data described by `pResDesc`, and also describes the subresource region to restrict access to when texturing. `pResViewDesc` can only be specified if the type of resource is a CUDA array or a CUDA mipmapped array.

Texture objects are only supported on devices of compute capability 3.0 or higher.

The **cudaResourceDesc** structure is defined as:

```
struct cudaResourceDesc {
  enum cudaResourceType resType;

  union {
    struct {
      cudaArray_t array;
    } array;
    struct {
      cudaMipmappedArray_t mipmap;
    } mipmap;
    struct {
      void *devPtr;
      struct cudaChannelFormatDesc desc;
      size_t sizeInBytes;
    } linear;
    struct {
      void *devPtr;
      struct cudaChannelFormatDesc desc;
      size_t width;
      size_t height;
      size_t pitchInBytes;
    } pitch2D;
```

```
    } res;
};
```

where:

- **cudaResourceDesc::resType** specifies the type of resource to texture from. CUresourceType is defined as:

```
enum cudaResourceType {
    cudaResourceTypeArray          = 0x00,
    cudaResourceTypeMipmappedArray = 0x01,
    cudaResourceTypeLinear         = 0x02,
    cudaResourceTypePitch2D        = 0x03
};
```

.RS 4 If **cudaResourceDesc::resType** is set to **cudaResourceTypeArray**, cudaResourceDesc::res::array::array must be set to a valid CUDA array handle.

.RS 4 If **cudaResourceDesc::resType** is set to **cudaResourceTypeMipmappedArray**, cudaResourceDesc::res::mipmap::mipmap must be set to a valid CUDA mipmapped array handle.

.RS 4 If **cudaResourceDesc::resType** is set to **cudaResourceTypeLinear**, cudaResourceDesc::res::linear::devPtr must be set to a valid device pointer, that is aligned to **cudaDeviceProp::textureAlignment**. cudaResourceDesc::res::linear::desc describes the format and the number of components per array element. cudaResourceDesc::res::linear::sizeInBytes specifies the size of the array in bytes. The total number of elements in the linear address range cannot exceed **cudaDeviceProp::maxTexture1DLinear**. The number of elements is computed as (sizeInBytes / sizeof(desc)).

.RS 4 If **cudaResourceDesc::resType** is set to **cudaResourceTypePitch2D**, cudaResourceDesc::res::pitch2D::devPtr must be set to a valid device pointer, that is aligned to **cudaDeviceProp::textureAlignment**. cudaResourceDesc::res::pitch2D::desc describes the format and the number of components per array element. cudaResourceDesc::res::pitch2D::width and cudaResourceDesc::res::pitch2D::height specify the width and height of the array in elements, and cannot exceed **cudaDeviceProp::maxTexture2DLinear**[0] and **cudaDeviceProp::maxTexture2DLinear**[1] respectively. cudaResourceDesc::res::pitch2D::pitchInBytes specifies the pitch between two rows in bytes and has to be aligned to **cudaDeviceProp::texturePitchAlignment**. Pitch cannot exceed **cudaDeviceProp::maxTexture2DLinear**[2].

The **cudaTextureDesc** struct is defined as

```
struct cudaTextureDesc {
    enum cudaTextureAddressMode addressMode[3];
    enum cudaTextureFilterMode  filterMode;
    enum cudaTextureReadMode    readMode;
    int                         sRGB;
    int                         normalizedCoords;
    unsigned int                maxAnisotropy;
    enum cudaTextureFilterMode  mipmapFilterMode;
    float                       mipmapLevelBias;
    float                       minMipmapLevelClamp;
    float                       maxMipmapLevelClamp;
};
```

where

- **cudaTextureDesc::addressMode** specifies the addressing mode for each dimension of the texture data. **cudaTextureAddressMode** is defined as:

```
enum cudaTextureAddressMode {
    cudaAddressModeWrap   = 0,
    cudaAddressModeClamp  = 1,
    cudaAddressModeMirror = 2,
    cudaAddressModeBorder = 3
```

```
};
```

This is ignored if **cudaResourceDesc::resType** is **cudaResourceTypeLinear**. Also, if **cudaTextureDesc::normalizedCoords** is set to zero, **cudaAddressModeWrap** and **cudaAddressModeMirror** won't be supported and will be switched to **cudaAddressModeClamp**.

- **cudaTextureDesc::filterMode** specifies the filtering mode to be used when fetching from the texture. **cudaTextureFilterMode** is defined as:

```
enum cudaTextureFilterMode {
  cudaFilterModePoint  = 0,
  cudaFilterModeLinear = 1
};
```

This is ignored if **cudaResourceDesc::resType** is **cudaResourceTypeLinear**.

- **cudaTextureDesc::readMode** specifies whether integer data should be converted to floating point or not. **cudaTextureReadMode** is defined as:

```
enum cudaTextureReadMode {
  cudaReadModeElementType     = 0,
  cudaReadModeNormalizedFloat = 1
};
```

Note that this applies only to 8-bit and 16-bit integer formats. 32-bit integer format would not be promoted, regardless of whether or not this **cudaTextureDesc::readMode** is set **cudaReadModeNormalizedFloat** is specified.

- cudaTextureDesc::sRGB specifies whether sRGB to linear conversion should be performed during texture fetch.
- **cudaTextureDesc::normalizedCoords** specifies whether the texture coordinates will be normalized or not.
- **cudaTextureDesc::maxAnisotropy** specifies the maximum anistropy ratio to be used when doing anisotropic filtering. This value will be clamped to the range [1,16].
- **cudaTextureDesc::mipmapFilterMode** specifies the filter mode when the calculated mipmap level lies between two defined mipmap levels.
- **cudaTextureDesc::mipmapLevelBias** specifies the offset to be applied to the calculated mipmap level.
- **cudaTextureDesc::minMipmapLevelClamp** specifies the lower end of the mipmap level range to clamp access to.
- **cudaTextureDesc::maxMipmapLevelClamp** specifies the upper end of the mipmap level range to clamp access to.

The **cudaResourceViewDesc** struct is defined as

```
struct cudaResourceViewDesc {
  enum cudaResourceViewFormat format;
  size_t          width;
  size_t          height;
  size_t          depth;
  unsigned int       firstMipmapLevel;
  unsigned int       lastMipmapLevel;
  unsigned int       firstLayer;
  unsigned int       lastLayer;
};
```

where:

- cudaResourceViewDesc::format specifies how the data contained in the CUDA array or CUDA mipmapped array should be interpreted. Note that this can incur a change in size of the texture data. If the resource view format is a block compressed format, then the underlying CUDA array or CUDA mipmapped array has to

have a 32-bit unsigned integer format with 2 or 4 channels, depending on the block compressed format. For ex., BC1 and BC4 require the underlying CUDA array to have a 32-bit unsigned int with 2 channels. The other BC formats require the underlying resource to have the same 32-bit unsigned int format but with 4 channels.

- cudaResourceViewDesc::width specifies the new width of the texture data. If the resource view format is a block compressed format, this value has to be 4 times the original width of the resource. For non block compressed formats, this value has to be equal to that of the original resource.
- cudaResourceViewDesc::height specifies the new height of the texture data. If the resource view format is a block compressed format, this value has to be 4 times the original height of the resource. For non block compressed formats, this value has to be equal to that of the original resource.
- cudaResourceViewDesc::depth specifies the new depth of the texture data. This value has to be equal to that of the original resource.
- **cudaResourceViewDesc::firstMipmapLevel** specifies the most detailed mipmap level. This will be the new mipmap level zero. For non-mipmapped resources, this value has to be zero.**cudaTextureDesc::minMipmapLevelClamp** and **cudaTextureDesc::maxMipmapLevelClamp** will be relative to this value. For ex., if the firstMipmapLevel is set to 2, and a minMipmapLevelClamp of 1.2 is specified, then the actual minimum mipmap level clamp will be 3.2.
- **cudaResourceViewDesc::lastMipmapLevel** specifies the least detailed mipmap level. For non-mipmapped resources, this value has to be zero.
- **cudaResourceViewDesc::firstLayer** specifies the first layer index for layered textures. This will be the new layer zero. For non-layered resources, this value has to be zero.
- **cudaResourceViewDesc::lastLayer** specifies the last layer index for layered textures. For non-layered resources, this value has to be zero.

Parameters:
pTexObject - Texture object to create
pResDesc - Resource descriptor
pTexDesc - Texture descriptor
pResViewDesc - Resource view descriptor

Returns:
cudaSuccess, **cudaErrorInvalidValue**

See also:
cudaDestroyTextureObject

cudaError_t cudaDestroyTextureObject (cudaTextureObject_t texObject)
Destroys the texture object specified by `texObject`.

Parameters:
texObject - Texture object to destroy

Returns:
cudaSuccess, **cudaErrorInvalidValue**

See also:
cudaCreateTextureObject

cudaError_t cudaGetTextureObjectResourceDesc (struct cudaResourceDesc * pResDesc, cudaTextureObject_t texObject)
Returns the resource descriptor for the texture object specified by `texObject`.

Parameters:
pResDesc - Resource descriptor
texObject - Texture object

Returns:
cudaSuccess, **cudaErrorInvalidValue**

See also:
cudaCreateTextureObject

cudaError_t cudaGetTextureObjectResourceViewDesc (struct cudaResourceViewDesc * pResViewDesc, cudaTextureObject_t texObject)

Returns the resource view descriptor for the texture object specified by `texObject`. If no resource view was specified, **cudaErrorInvalidValue** is returned.

Parameters:

pResViewDesc - Resource view descriptor
texObject - Texture object

Returns:

cudaSuccess, **cudaErrorInvalidValue**

See also:

cudaCreateTextureObject

cudaError_t cudaGetTextureObjectTextureDesc (struct cudaTextureDesc * pTexDesc, cudaTextureObject_t texObject)

Returns the texture descriptor for the texture object specified by `texObject`.

Parameters:

pTexDesc - Texture descriptor
texObject - Texture object

Returns:

cudaSuccess, **cudaErrorInvalidValue**

See also:

cudaCreateTextureObject

Author

Generated automatically by Doxygen from the source code.