ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Προγραμματισμός υπολογιστών με C++

Τεκμηρίωση HTML 3ης προγραμματιστικής εργασίας όπως παράχθηκε με το εργαλείο Doxy Wizard

Η ομάδα των εκπονητών απαρτίζεται από τα ακόλουθα μέλη:

Ονοματεπώνυμο	Αριθμός Μητρώου	E-mail επικοινωνίας
Γεώργιος Μοσχόβης	3150113	p3150113@dias.aueb.gr
Μαρκόπουλος Σταύρος	3150098	markopoulos.steve@outlook.com

Αθήνα, 22 Φεβρουαρίου 2017

math::Array< T > Class Template Reference

`#include <Array.h>`

Public Member Functions

unsigned int	**getWidth** () const
unsigned int	**getHeight** () const
void *const	**getRawDataPtr** ()
T &	**operator()** (int x, int y)
const T &	**operator()** (int x, int y) const
	Array (unsigned int w, unsigned int h)
	Array (const **Array**< T > &source)
Array &	**operator=** (const **Array**< T > &source)
bool	**operator==** (const **Array**< T > &right) const
void	**resize** (unsigned int new_width, unsigned int new_height)
virtual	**~Array** ()

Protected Attributes

T *	**buffer** Flat storage of the elements of the array of type T.
unsigned int	**width** The width of the array (number of columns)
unsigned int	**height** The height of the array (number of rows)

Detailed Description

template<typename T>
class math::Array< T >

The **Array** class implements a generic two-dimensional array of elements of type T.

Constructor & Destructor Documentation

◆ Array() [1/2]

template<typename T >

math::Array< T >::**Array** (unsigned int w,
unsigned int h
)

Constructor with array size.

No default constructor is provided as it makes no sense.

Parameters

w is the width (columns) of the array

h is the height (rows) of the array

◆ Array() [2/2]

template<typename T>

math::Array< T >::**Array**< T > (const **Array**< T > & source)

Copy constructor.

No default constructor is provided as it makes no sense.

Parameters

source is the array to replicate.

◆ ~Array()

template<typename T >

math::Array< T >::~**Array** ( ) virtual

Virtual destructor.

Member Function Documentation

◆ getHeight()

template<typename T>

unsigned int **math::Array**< T >::getHeight ( ) const inline

Reports the height (rows) of the array

Returns

the height.

◆ getRawDataPtr()

template<typename T >

void *const **math::Array**< T >::getRawDataPtr ( )

Obtains a constant pointer to the internal data.

This is NOT a copy of the internal array data, but rather a pointer to the internally allocated space.

◆ getWidth()

template<typename T>

unsigned int **math::Array**< T >::getWidth ( ) const inline

Reports the width (columns) of the array

Returns

the width.

◆ operator()() [1/2]

```
template<typename T >
T & math::Array< T >::operator() ( int  x,
                                   int  y
                                 )
```

Returns a reference to the element at the zero-based position (column x, row y).

Parameters

x is the zero-based column index of the array.

y is the zero-based row index of the array.

Returns

a reference to the element at position (x,y)

◆ operator()() [2/2]

```
template<typename T >
const T & math::Array< T >::operator() ( int  x,
                                         int  y
                                       )     const
```

Returns a constant reference to the element at the zero-based position (column x, row y).

Parameters

x is the zero-based column index of the array.

y is the zero-based row index of the array.

Returns

a reference to the element at position (x,y)

◆ operator=()

```
template<typename T>
Array< T > & math::Array< T >::operator= ( const Array< T > &  source )
```

Copy assignment operator

Parameters

source is the array to replicate.

◆ operator==()

template<typename T>

bool **math::Array**< T >::operator== (const **Array**< T > & right) const

Equality operator.

Parameters

right is the array to compare the current object to.

Returns

true if the current array and the source have the same dimensions AND one by one their elements of type T are the same.

◆ resize()

template<typename T >

void **math::Array**< T >::resize (unsigned int new_width,
unsigned int new_height
)

Changes the internal array data storage size.

If the one or both of the given dimensions are smaller, the array should be clipped by discarding the remaining elements in the rows and/or columns outside the margins. If the new dimensions are larger, pad the old elements with default values of type T. If the array is not yet allocated (zero width and height), allocate the internal buffer and set the array size according to the given dimensions.

Parameters

new_width is the user-provided width to resize the array to.

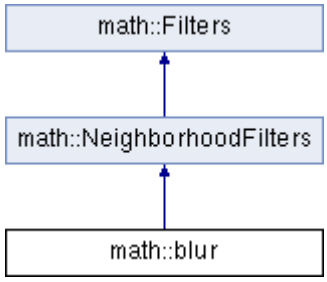
new_height is the user-provided height to resize the array to.

The documentation for this class was generated from the following files:

- C:/Users/George/Documents/Visual Studio 2015/Projects/TeamProject3/**Array.h**
- C:/Users/George/Documents/Visual Studio 2015/Projects/TeamProject3/**Array.hpp**

math::blur Class Reference

Inheritance diagram for math::blur:



Public Member Functions

virtual void **filterate** (**Image** &sampleimage)

Public Member Functions inherited from math::NeighborhoodFilters

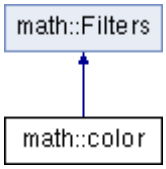
The documentation for this class was generated from the following file:

- C:/Users/George/Documents/Visual Studio 2015/Projects/TeamProject3/**blur.h**

Generated by doxygen 1.8.13

math::color Class Reference

Inheritance diagram for math::color:



Public Member Functions

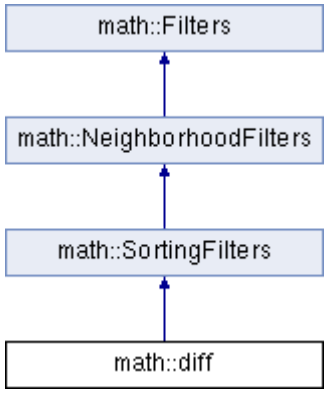
	color::color (float arg1, float arg2, float arg3)
virtual void	**filterate** (**Image** &sampleimage)

The documentation for this class was generated from the following file:

- C:/Users/George/Documents/Visual Studio 2015/Projects/TeamProject3/**color.h**

math::diff Class Reference

Inheritance diagram for math::diff:



Public Member Functions

virtual void **filterate** (**Image** &sampleimage)

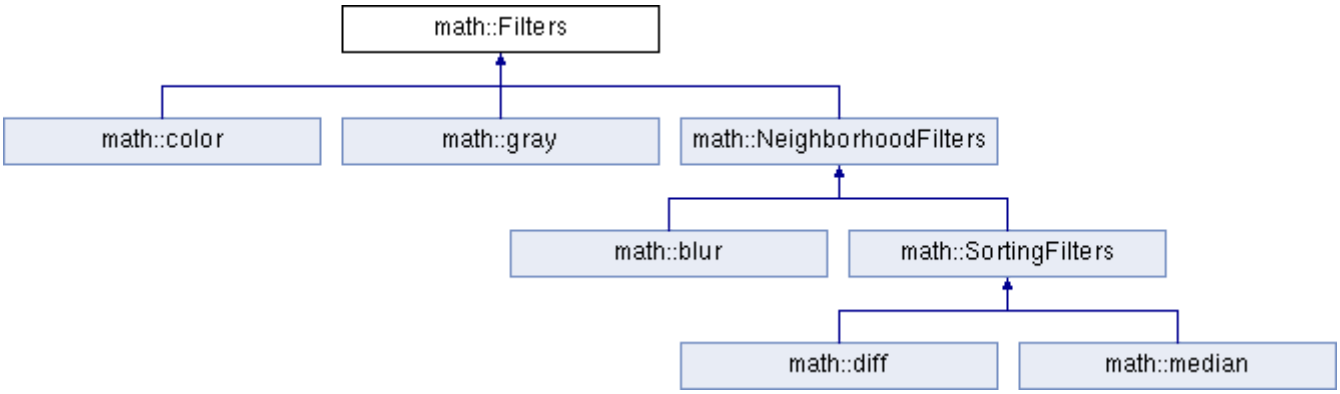
- **Public Member Functions inherited from math::SortingFilters**
- **Public Member Functions inherited from math::NeighborhoodFilters**

The documentation for this class was generated from the following file:

- C:/Users/George/Documents/Visual Studio 2015/Projects/TeamProject3/**diff.h**

math::Filters Class Reference abstract

Inheritance diagram for math::Filters:



Public Member Functions

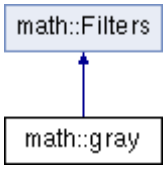
virtual void **filterate** (**Image** &sampleimage)=0

The documentation for this class was generated from the following file:

- C:/Users/George/Documents/Visual Studio 2015/Projects/TeamProject3/**Filters.h**

math::gray Class Reference

Inheritance diagram for math::gray:



Public Member Functions

virtual void **filterate** (**Image** &sampleimage)

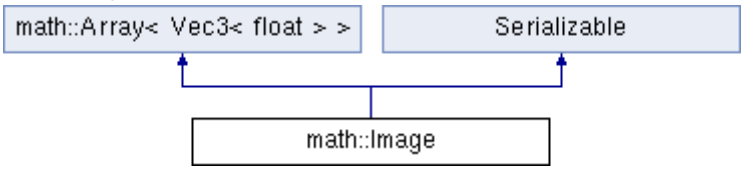
The documentation for this class was generated from the following file:

- C:/Users/George/Documents/Visual Studio 2015/Projects/TeamProject3/**gray.h**

math::Image Class Reference

`#include <Image.h>`

Inheritance diagram for math::Image:



Public Member Functions

Vec3< float > *	**Image::getRawDataPtr** ()
Vec3< float >	**Image::getPixel** (unsigned int x, unsigned int y) const
void	**Image::setPixel** (unsigned int x, unsigned int y, **Vec3**< float > &value)
void	**Image::setData** (const **Vec3**< float > *&data_ptr)
void	**Image::resize** (unsigned int new_width, unsigned int new_height)
bool	**operator<<** (std::string filename)
bool	**operator>>** (std::string filename)
	Image::Image ()
	Image::Image (unsigned int **width**, unsigned int **height**)
	Image::Image (unsigned int **width**, unsigned int **height**, const **Vec3**< float > *data_ptr)
	Image::Image (const **Image** &src)
	Image::~Image ()
Image &	**Image::operator=** (const **Image** &right)

Public Member Functions inherited from math::Array< Vec3< float > >

Additional Inherited Members

Protected Attributes inherited from math::Array< Vec3< float > >

Detailed Description

It is the class that represents a generic data container for an image.

It holds the actual buffer of the pixel values and provides methods for accessing them, either as individual pixels or as a memory block. The **Image** class alone does not provide any functionality for loading and storing an image, as it is the result or input to such a procedure.

The internal buffer of an image object stores the actual bytes (data) of the color image as a contiguous sequence of RGB triplets. All values stored in the internal memory buffer are floating point values and for typical (normalized) intensity ranges, each color component is within the range [0.0, 1.0].

Constructor & Destructor Documentation

◆ Image::~Image()

math::Image::Image::~Image ( )

The **Image** destructor.

Member Function Documentation

◆ Image::getPixel()

Vec3<float> math::Image::Image::getPixel (unsigned int x,
unsigned int y
) const

Obtains the color of the image at location (x,y).

The method should do any necessary bounds checking.

Parameters

x is the (zero-based) horizontal index of the pixel to get.

y is the (zero-based) vertical index of the pixel to get.

Returns

The color of the (x,y) pixel as a Color object. Returns a black (0,0,0) color in case of an out-of-bounds x,y pair.

◆ Image::getRawDataPtr()

Vec3<float>* math::Image::Image::getRawDataPtr ( )

Obtains a pointer to the internal data.

This is NOT a copy of the internal image data, but rather a pointer to the internally allocated space, so DO NOT attempt to delete the pointer.

◆ Image::Image() [1/4]

math::Image::Image::Image ( )

Default constructor.

By default, the dimensions of the image should be zero and the buffer must be set to nullptr.

◆ Image::Image() [2/4]

```
math::Image::Image::Image ( unsigned int  width,
                            unsigned int  height
                          )
```

Constructor with width and height specification.

Parameters

width is the desired width of the new image.

height is the desired height of the new image.

◆ Image::Image() [3/4]

```
math::Image::Image::Image ( unsigned int             width,
                            unsigned int             height,
                            const Vec3< float > *    data_ptr
                          )
```

Constructor with data initialization.

Parameters

width is the desired width of the new image.

height is the desired height of the new image.

data_ptr is the source of the data to copy to the internal image buffer.

See also

setData

◆ Image::Image() [4/4]

math::Image::Image::Image (const **Image** & src)

Copy constructor.

Parameters

src is the source image to replicate in this object.

◆ Image::operator=()

Image& math::Image::Image::operator= (const **Image** & right)

Copy assignment operator.

Parameters

right is the source image.

◆ Image::resize()

void math::Image::Image::resize (unsigned int new_width,
unsigned int new_height
)

Changes the internal image data storage size.

If the one or both of the given dimensions are smaller, the image should be clipped by discarding the remaining pixels in the rows and/or columns outside the margins. If the new dimensions are larger, pad the old pixels with zero values (black color). If the image is not yet allocated (zero width and height), allocate the internal buffer and set the image size according to the given dimensions.

Parameters

new_width is the user-provided width to resize the image storage to.

new_height is the user-provided height to resize the image storage to.

◆ Image::setData()

void math::Image::Image::setData (const **Vec3**< float > *& data_ptr)

Copies the image data from an external raw buffer to the internal image buffer.

The member function ASSUMES that the input buffer is of a size compatible with the internal storage of the **Image** object and that the data buffer has been alreeady allocated. If the image buffer is not allocated or the width or height of the image are 0, the method should exit immediately.

Parameters

data_ptr is the reference to the preallocated buffer from where to copy the data to the **Image** object.

◆ Image::setPixel()

void math::Image::Image::setPixel (unsigned int x,
unsigned int y,
Vec3< float > & value
)

Sets the RGB values for an (x,y) pixel.

The method should perform any necessary bounds checking.

Parameters

x is the (zero-based) horizontal index of the pixel to set.

y is the (zero-based) vertical index of the pixel to set.

value is the new color for the (x,y) pixel.

◆ operator<<()

bool math::Image::operator<< (std::string filename) virtual

Reads the contents of an object from the specified file.

The operator can be used for the implementation of "de-serialization".

Parameters

filename is the name of the file to use for loading the data.

Returns

true if the operation was successful, false otherwise.

Implements **Serializable**.

◆ operator>>()

bool math::Image::operator>> (std::string filename) virtual

Writes the contents of an object to the specified file.

The operator can be used for the implementation of "serialization".

Parameters

filename is the name of the file to use for saving the data.

Returns

true if the operation was successful, false otherwise.

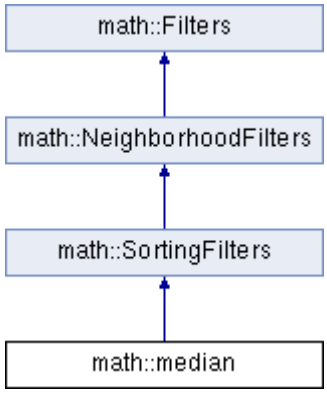
Implements **Serializable**.

The documentation for this class was generated from the following files:

- C:/Users/George/Documents/Visual Studio 2015/Projects/TeamProject3/**Image.h**
- C:/Users/George/Documents/Visual Studio 2015/Projects/TeamProject3/Image.cpp

math::median Class Reference

Inheritance diagram for math::median:



Public Member Functions

virtual void **filterate** (**Image** &sampleimage)

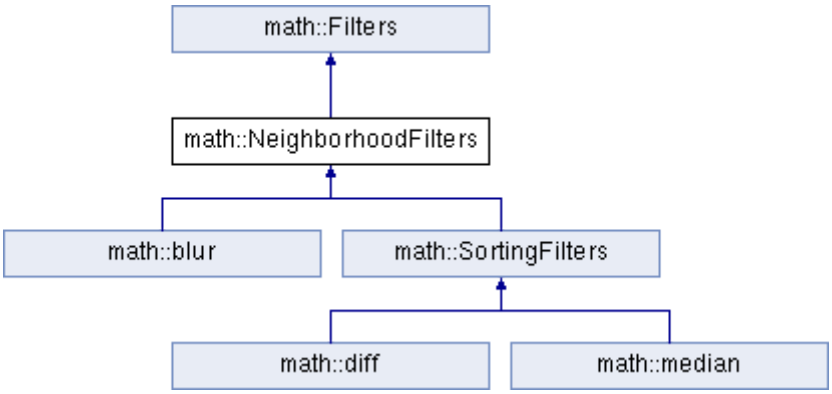
- **Public Member Functions inherited from math::SortingFilters**
- **Public Member Functions inherited from math::NeighborhoodFilters**

The documentation for this class was generated from the following file:

- C:/Users/George/Documents/Visual Studio 2015/Projects/TeamProject3/**median.h**

math::NeighborhoodFilters Class Reference abstract

Inheritance diagram for math::NeighborhoodFilters:



Public Member Functions

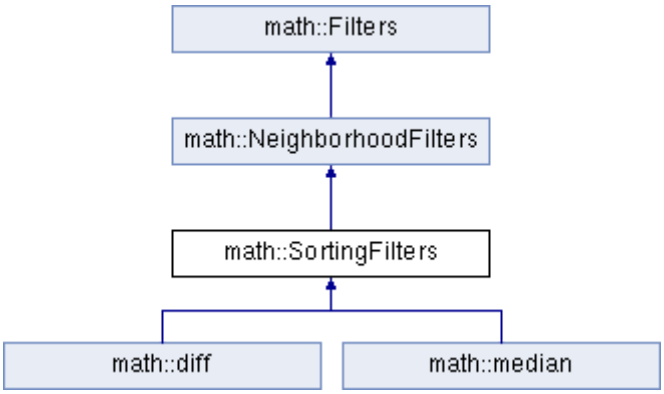
virtual void	**filterate** (**Image** &sampleimage)=0
virtual bool	**checkIfValid** (**Image** &sampleimage, size_t x, size_t y)

The documentation for this class was generated from the following file:

- C:/Users/George/Documents/Visual Studio 2015/Projects/TeamProject3/**NeighborhoodFilters.h**

math::SortingFilters Class Reference abstract

Inheritance diagram for math::SortingFilters:



Public Member Functions

virtual void	**filterate** (**Image** &sampleimage)=0
virtual void	**bubbleSort** (float *pixelarray, size_t arraysize)

Public Member Functions inherited from math::NeighborhoodFilters

The documentation for this class was generated from the following file:

- C:/Users/George/Documents/Visual Studio 2015/Projects/TeamProject3/**SortingFilters.h**

math::Vec3< S > Class Template Reference

`#include <Vec3.h>`

Public Member Functions

S &	**operator[]** (size_t index)
Vec3< S >	**operator+** (const **Vec3**< S > &right)
Vec3< S >	**operator-** (const **Vec3**< S > &right)
Vec3< S >	**operator*** (const **Vec3**< S > &right)
Vec3< S >	**operator*** (S right)
Vec3< S >	**operator/** (S right)
Vec3< S >	**operator/** (const **Vec3**< S > &right)
Vec3< S > &	**operator+=** (const **Vec3**< S > &right)
Vec3< S > &	**operator-=** (const **Vec3**< S > &right)
Vec3< S > &	**operator/=** (const **Vec3**< S > &right)
Vec3< S > &	**operator*=** (const **Vec3**< S > &right)
Vec3< S > &	**operator*=** (S right)
Vec3< S > &	**operator/=** (S right)
	Vec3 (S x, S y, S z)
	Vec3 (S val)
	Vec3 ()
	Vec3 (const **Vec3**< S > &right)
Vec3< S > &	**operator=** (const **Vec3**< S > &right)
bool	**operator==** (const **Vec3**< S > &right) const
bool	**operator!=** (const **Vec3**< S > &right) const

Public Attributes

union {
  S **x**
  S **r**
};

The first coordinate of the vector.

union {
  S **y**
  S **g**
};

The second coordinate of the vector.

union {
  S **z**
  S **b**

};

The third coordinate of the vector.

Detailed Description

template<typename S>
class math::Vec3< S >

Represents a triplet of values of the same type S.

The **Vec3** class is used as a generic three-dimensional vector and thus it defines several numerical operators that can be used on Vec3<S> and S data.

Constructor & Destructor Documentation

◆ Vec3() [1/4]

template<typename S>

math::Vec3< S >::**Vec3** (S x,
S y,
S z
)

inline

Constructor with three-element initialization.

Parameters

x is the value of th first element.

y is the value of the second element.

z is the value of the third element.

◆ Vec3() [2/4]

template<typename S>

math::Vec3< S >::**Vec3** (S val)

inline

Constructor with single-element initialization.

Parameters

val is the value that is replicated to all elements of the vector.

◆ Vec3() [3/4]

template<typename S>

math::Vec3< S >::**Vec3** ( ) inline

Default constructor.

Initializes all elements to their default numerical value.

◆ Vec3() [4/4]

template<typename S>

math::Vec3< S >::**Vec3** (const **Vec3**< S > & right) inline

Copy constructor constructor.

Member Function Documentation

◆ operator!=()

template<typename S>

bool **math::Vec3**< S >::operator!= (const **Vec3**< S > & right) const inline

Inequality operator

Parameters

right is the vector to compare the current with.

Returns

true if at least one element of the current vector differs from the right vector, false otherwise.

◆ operator*() [1/2]

template<typename S>

Vec3<S> **math::Vec3**< S >::operator* (const **Vec3**< S > & right) inline

Component-wise vector multiplication.

Parameters

right is the right-hand vector operand of the multiplication

Returns

a new vector whose elements are the component-wise multiplied elements of the current and the right vectors.

◆ operator*() [2/2]

template<typename S>

Vec3<S> **math::Vec3**< S >::operator* (S right) inline

Vector-scalar multiplication.

Parameters

right is the right-hand scalar operand of the multiplication

Returns

a new vector whose elements are the elements of the current vector multiplied with the right operand.

◆ operator*=() [1/2]

template<typename S>

Vec3<S>& **math::Vec3**< S >::operator*= (const **Vec3**< S > & right) inline

Multiplication assignment using a vector multiplier.

Parameters

right is the vector multiplier.

Returns

a reference to the current vector after the change.

◆ operator*=() [2/2]

template<typename S>

Vec3<S>& **math::Vec3**< S >::operator*= (S right) inline

Multiplication assignment using a scalar multiplier.

Parameters

right is the scalar multiplier.

Returns

a reference to the current vector after the change.

◆ operator+()

template<typename S>

Vec3<S> **math::Vec3**< S >::operator+ (const **Vec3**< S > & right) inline

Vector addition.

Parameters

right is the right-hand vector operand of the addition.

Returns

a new vector that is the component-wise sum of the current and the right vectors.

◆ operator+=()

template<typename S>

Vec3<S>& **math::Vec3**< S >::operator+= (const **Vec3**< S > & right) inline

Addition assignment.

Parameters

right is the vector to add to the current one.

Returns

a reference to the current vector after the change.

◆ operator-()

template<typename S>

Vec3<S> **math::Vec3**< S >::operator- (const **Vec3**< S > & right) inline

Vector subtraction.

Parameters

right is the right-hand vector operand of the subtraction

Returns

a new vector that is the component-wise subtraction of the current and the right vectors.

◆ operator-=()

template<typename S>

Vec3<S>& **math::Vec3**< S >::operator-= (const **Vec3**< S > & right) inline

Subtraction assignment

Parameters

right is the vector to subtract from the current one.

Returns

a reference to the current vector after the change.

◆ operator/() [1/2]

template<typename S>

Vec3<S> **math::Vec3**< S >::operator/ (S right) inline

Scalar division.

No checks are made for zero divisor.

Parameters

right is the scalar divisor.

Returns

a new vector whose elements are the elements of the current vector divided by the right operand.

◆ operator/() [2/2]

template<typename S>

Vec3<S> **math::Vec3**< S >::operator/ (const **Vec3**< S > & right) inline

Component-wise vector division.

No checks are made for zero divisor elements.

Parameters

right is the vector divisor.

Returns

a new vector whose elements are the elements of the current vector divided by the corresponding elements of the right operand.

◆ operator/=() [1/2]

template<typename S>

Vec3<S>& **math::Vec3**< S >::operator/= (const **Vec3**< S > & right) inline

Division assignment using a vector divisor.

Parameters

right is the vector divisor.

Returns

a reference to the current vector after the change.

◆ operator/=() [2/2]

template<typename S>

Vec3<S>& **math::Vec3**< S >::operator/= (S right) inline

Division assignment using a scalar divisor.

Parameters

right is the scalar divisor.

Returns

a reference to the current vector after the change.

◆ operator=()

template<typename S>

Vec3<S>& **math::Vec3**< S >::operator= (const **Vec3**< S > & right) inline

Copy assignment operator.

Parameters

right is the vector to copy.

Returns

a reference to the current vector after the assignment.

◆ operator==()

template<typename S>

bool **math::Vec3**< S >::operator== (const **Vec3**< S > & right) const inline

Equality operator

Parameters

right is the vector to compare the current with.

Returns

true if the vectors are equal element by element, false otherwise.

◆ operator[]()

template<typename S>

S& **math::Vec3**< S >::operator[] (size_t index) inline

Data access operator.

Parameters

index is the zero-based index to the elements of the vector. No bounds checking is performed for performance reasons.

Returns

the index-th element of the vector.

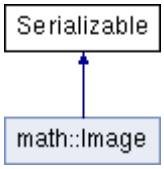
The documentation for this class was generated from the following file:

- C:/Users/George/Documents/Visual Studio 2015/Projects/TeamProject3/**Vec3.h**

Serializable Class Reference abstract

`#include <Serializable.h>`

Inheritance diagram for Serializable:



Public Member Functions

virtual bool **operator<<** (std::string filename)=0

virtual bool **operator>>** (std::string filename)=0

Detailed Description

Interface for object serialization to a file.

Provides two operators that objects derived from this class can use to store and load their data from a file, Using a custom implementation for them.

Member Function Documentation

◆ operator<<()

virtual bool Serializable::operator<< (std::string filename) pure virtual

Reads the contents of an object from the specified file.

The operator can be used for the implementation of "de-serialization".

Parameters

filename is the name of the file to use for loading the data.

Returns

true if the operation was successful, false otherwise.

Implemented in **math::Image**.

◆ operator>>()

virtual bool Serializable::operator>> (std::string filename) pure virtual

Writes the contents of an object to the specified file.

The operator can be used for the implementation of "serialization".

Parameters

filename is the name of the file to use for saving the data.

Returns

true if the operation was successful, false otherwise.

Implemented in **math::Image**.

The documentation for this class was generated from the following file:

- C:/Users/George/Documents/Visual Studio 2015/Projects/TeamProject3/**Serializable.h**