SharpMedia Math Design

© 2007 Žiga Osolin – SharpMedia project, document version: DRAFT

Table Of Contents

[About 3](#_Toc205777382)

[Goals 3](#_Toc205777383)

[Deployment Structure 4](#_Toc205777384)

[Overview 4](#_Toc205777385)

[Namespace Structure 4](#_Toc205777386)

[*Namespace* SharpMedia.Math 5](#_Toc205777387)

[*Class* Expression 6](#_Toc205777388)

[*Class* Sparse Vector 7](#_Toc205777389)

[*Namespace* SharpMedia.Math.Functions 7](#_Toc205777390)

[Distribution functions 8](#_Toc205777391)

[Sampled functions 8](#_Toc205777392)

[*Numerical namespaces* 8](#_Toc205777393)

[Root finders 8](#_Toc205777394)

[Derivates, Integrals 8](#_Toc205777395)

[Differential Equations 9](#_Toc205777396)

[Fast Furier Transformation (FFT) 9](#_Toc205777397)

[*Namespace* SharpMedia.Math.Matrix 9](#_Toc205777398)

[*Interface* IBaseMatrix 9](#_Toc205777399)

[Generic matrices 9](#_Toc205777400)

[Specialized matrices 9](#_Toc205777401)

[Sparse matrices 9](#_Toc205777402)

[*Namespace* SharpMedia.Math.Misc 10](#_Toc205777403)

[*Namespace* SharpMedia.Math.Functions.Discrete 10](#_Toc205777404)

[*Interface* IHistrogram 10](#_Toc205777405)

[*Interface* ISeries 10](#_Toc205777406)

[*Namespace* SharpMedia.Math.Shapes 11](#_Toc205777407)

[Usage scenarios 12](#_Toc205777408)

[Expression reusage 12](#_Toc205777409)

[Operator overloads 12](#_Toc205777410)

[Implementation Notes 13](#_Toc205777411)

[ToString method 13](#_Toc205777412)

[Unstability issues 13](#_Toc205777413)

[Performance vs. Usability 13](#_Toc205777414)

# About

The SharpMedia math is a SharpMedia library (actually a part of core library) that can be utilized by writers of any component that in some way requires math.

# Goals

The goal of mathematic classes is to introduce mathematical concepts in engine. Using these concepts, we can solve problems. The main aim is to provide clean, object-orientated mathematical library that is able to solve any mathematical problem (that can theoretically be solved) that may arise at engine/game development.

We will try to stick to mathematical correctness so mathematicians won't have problems working with classes because they implement based on mathematical theory. When we do not stick to mathematical notations or we don't allow certain patterns, this should be specifically stated.

One of the goals is also simplicity of mathematical engine. We should not overcomplicate things and assume users are good at maths, we should provide certain shortcuts. For example, if the user does not know the concepts of matrices, we can abstract them as a transformation and we simply apply those transformations using some sort of transformation delegate.

Performance is also one of the crucial elements of math. The code written must be clear and roboust. We should rely on compiler and JIT to optimize code because it actually does quite a good job. At the same time, we shouldn't use too many virtual methods because we want as much code as possible to be inlined.

In the relation to performance is also multithreading. Many mathematical entities can be evaluated in paralell. Multithreading should be exposed for functions that may require it. Dividing a polynomial is clearly not a good example of that, but Monte-Carlo integration (especially with a lot of samples and complex functions) is.

With multithreading comes the thread safety. In general, most of the classes in math won't be thread safe because we want full performance. Sometimes, however, ensuring thread safety is required and should be done (it is better to compute something a little bit slower that crash the application).

Last but not least, the reusability of code must be maximum. Many times, people think that they will optimize the code if they manually »inline« the code. This may be true for debug mode, but certainly not for release mode. Trust compiler to optimize code and never manually inline code. Math library will also depend heavily on post-compile inlining plugins for better performance.

# Deployment Structure

Math library is something so common and essential that is is distributed together with SharpMedia core library.

# Overview

The math system is implementation of mathematic theory useful to SharpMedia and it's components. As a result of that, the library is preaty algorithm intense.

Mathematic system is written for double and single precission floating point number. Because it is impossible to make all classes generics (C# is to restrictive for this), two (or more) classes are make, one using double and one single precision floating point number (and possibly for integer if resonable). We use our own templating engine to address this issue. All double versions are postfixed by 'd' and all single precission versions are postfixed by 'f' (e.g. Vector3d and Vector3f).

Static classes do not have seperate single and double precission floating point versions but they have methods for both versions. This means that methods are overloaded. If all arguments are the same for methods, 'd' or 'f' is put at the end of the method's name.

# Namespace Structure

The namespaces represent the basic subdivision into categories of mathematics. Some may interfere with each other. The hierarchy is as follows:

## *Namespace* SharpMedia.Math

SharpMedia.Math implements common and most generic functionality of maths and is the one the user most frequently needs. The concepts of vector, complex numbers, quaternion and coordinate systems are implemented here. Such mathematical object are then used through other mathematical namespace.

Note that some classes won't be discussed into details. Basically they just represent the mathematical implementation.

Some classes may seem duplicated. Vector2, Vector3, Vector4 and generic Vector may be put into just one generic Vector class but this is not done for several reasons:

* Performance (array manipulation is pointer indirection);
* Vector2, Vector3 and Vector4 can be structs and thus do not reside in heap memory;
* Operations are type safe at any stage at compile time when using specific vectors.

Conversions between vectors (representing points in cartesian system) to other systems (polar, cylindical, spherical) and vice versa must be provided, usually as properties.

### *Class* Expression

Expression class is somewhat special because it is actually a dynamic delegate generator. This means it emits code at runtime, given a string description of function (in mathematical formulation).

The expression supports all basic mathematical operations. It also exposes parameters and variables and allows user to bind them to specific values or to make them variables. The expression can create any number of delegates with different bindings. It is immutable.

The class should also automatically optimize expressions in case of constrants. All optimisations are done at code generation, this means at the stage when parameters and variables are known and we can precompute as many as we want.

### *Class* Sparse Vector

Sparse vector are vectors with a lot of zeros. In SharpMedia, they should be represented as linked list of elements.

This way, we can efficiently store and transverse the vector. The implementation is efficient only if a lot of elements are sparse. The dot product of vector is very fast.

## *Namespace* SharpMedia.Math.Functions

Function namespace contains all functions that are special in that way that they are defined by classes. Such functions usually have some special properties that allow faster mathematics, such as special properties when they are integrated, when derivates are seeked or when we want to multiply them with others.

Each function must extend ***IFunction*** interface. This implies that each function can represent itself in a form of delegate, that is in generic form where no special properties are known.

### Distribution functions

All probability functions extend from ***IDistributionyFunction*** interface. This interface requires comulative probability distribution and some other probability functions specific properties, such as mean value.

It is desirable that probability function includes static construct that creates function from N points on graph.

### Sampled functions

Samples functions are those that are either too complex to compute at realtime or they simply cannot be computed. Factorial is one of them. Factorial is defined only for 170 double precission floats because 171 already overflows. This means that factorial can be sampled (in range 0-170) and evaluating a function is just performing a lookup in table.

## *Numerical namespaces*

Numerics implements some of the numerical analysis methods. This includes numerical derivates, integrals, minimums, maximums, roots, differential equation solutions …

### Root finders

Many root finding algorithms exist. Some have slow convengance but are stable and vice versa. The root finding algorithms should expose all methods and than also provide the middle point – stable and still quite fast. This middle point usually includes using stable method until faster method can be used. Root finders can also use derivate information (if available) to find roots.

Multidimension function roots should also be implemented.

### Derivates, Integrals

Derivates and integrals are numerically unstable for all types of functions. You should always prefer non-numerical methods because they are more accurate and they know how function behaves. All numerical methods know is the values of function. This makes them hard to analyse.

Various orders of derivations should be supported, using normal first order up to at least forth order derivation. Higher order implyies higher complexity of algorithm and not necessary higher precission, but in most cases, higher precission is also achieved.

Numerical integration is probably one of the most useful numerical algorithms. Integrals should be available for all types of functions (1D, 2D, 3D or 4D) and multithread evaluations should also be possible. Spherical functions should not be missed since they are important for game engines.

Importance sampling is also one of the important properties of integration, where more samples are taken where the value of the function is higher. This is especially useful for integration over hemisphere. It should be done using Monte-Carlo integration.

### Differential Equations

We should allow differential equations to be computed in numeric way. Euler and Runge-Kutta (order 4) methods are the ones that we should support.

### Fast Furier Transformation (FFT)

Fast Furier Transform is mapping of periodic function one series of sinuses. The fast algorithm should be implemented in order to allow fast periodic function analysis. This can be useful in sound systems and elsewhere when such functions arise.

## *Namespace* SharpMedia.Math.Matrix

Matrix namespace is implementation of systems of N equations with M unknowns. Such systems can be numerically solved and they have special properties (such as determinants, eigen vectors and values, similarities …). The basic goal of this library is to allow operations on such systems and to provide some specialized matrix systems that can be treated in special way (sparse matrices).

### *Interface* IBaseMatrix

The ***IBaseMatrix*** interface defines the needed set of every matrix type. Besides this interface, the matrix should also implement operator overloads (\*, /, +, -) and other shortcuts to algorithms (that can be further optimized for specific matrix later on).

### Generic matrices

Generic matrix implementation allows any number of rows and columns. It implements all operations, also operations that assume square or any other form of matrix. Such operations should throw an exception if called on wrong matrix type.

Generic matrices should implement multiplication, substraction, addition, inverse (at least Gauss-Jordan method), determinant calculation, eigen vectors and values, transpose and LU decomposition.

### Specialized matrices

Generic matrices are not the fastest. This is the reason why we hardcoded most common matrices (that are also used for transformations in 3D graphics). Those matrices are:

* 2x2 matrix – determinant, inverse and all others are very simple calculations, also storage wise[[1]](#footnote-2)
* 3x3 matrix – determinant, inverse and all others are simple operations, also storage wise
* 4x4 matrix – there are some speed benefits because we use compact storage. If necessary, we should hardcode matrix inverse, multiplications and other operations are hardcoded.

### Sparse matrices

Sparse matrices are special type of matrices that are known to have some zeros. Such matrices are treated specifically because we can optimize their operations if we known the zeros layout.

#### Generic Sparse

The generic version of sparse matrix is based on sparse vector implementation. It allows fast swapping operations, as well as fast multiplication. It also has an efficient storage, since only non zero elements are indexed.

Generic sparse matrix should only be used when the specialized sparse matrix does not exist and when there are really a lot of zeros in matrix.

#### Band matrix

Band matrix is matrix, that has non-zero elements at main diagonal and a few adjectory diagonals. The storage is more compact because we index per diagonal and not per row/column. Special version (most commonly used) is triagonal band matrix, which has 3 diagonals with non-zero elements.

## *Namespace* SharpMedia.Math.Misc

Misc is namespace that contains extended features that are not used often. For example, a solver of system of two quadratic equations is someting that is placed in this namespace.

## *Namespace* SharpMedia.Math.Functions.Discrete

Discrete mathematics is mathematics dealing with discrete values. Discrete functions are those that map an unsigned integer to real number. We deal with discrete series in this namespace, as well as other concepts categorized in this section by mathematical theory.

### *Interface* IHistrogram

The histrogram is a discrete structure. It is actually a special kind of function that represents the distribution of data on some discrete values. It can be useful for statistics in conjuction with image analysis (and optimizations), geometry analysis, …

The histrogram allows obtaining the procentage of data on some discrete value, the size of interval, conversion to discrete function and other statistical quantities (mean, average …). Classes that represent historgram should extend this interface. Interface is not limited to Math classes only.

### *Interface* ISeries

Series are elements that have some relation to each other. Series are actually transformation specified as:. We are interested in values and also sums (finite and infinite) of elements. We support the following series:

* Geometric series ();
* Arithmetic series (;
* Recursive series ());
* Other (non-special) series.

## *Namespace* SharpMedia.Math.Shapes

This namespace is probably one of the most important namespaces for game engine. It is also the namespace that needs careful design to address issues of speed, format compatibility, simplicity to use and so on. After many experiments, we have decided to split namespace into following subnamespaces:

* **Shapes –** implements 2D and 3D structures (point, triangle, rectangle, quad, curves, ...).
* **Shapes.Storage** – implements plain data formats for shape compact storage (compatible with GPU)
* **Shapes.Volumes** – implements volumes (AAB, box, sphere, voxel grid ...), closely related to **3D** shapes
* **Shapes.Spatial** – a spatial organisation of shapes using volumes (octree, KD-trees etc.), mostly specialised for triangle/quad soups.
* **Shapes.Algorithms** – contains some algorithms, usually available as specialized versions in seperate 2D/3D namespaces.

The most important aspect of this namespace is that it is easily extendable by GPU and fully compatible with it. The most important namespace addressing compatibility issues is **Storage**.

### Storage

**Storage** defines buffers that are able to store data, topology sets and the actual composition of everything, described in geometry. It is designed in such way that it may share data with graphics representation. While its representation is totally independant of graphics' and in many ways more genric, it can link to the same buffers.

We use the concept of views in graphics. **ControlPointView** is another view that can be directed to any **IMappable<byte[]>** interface. Both **TypelessBuffer** and **TypelessTexture** implement this interface, making use of those buffer easy by **Shapes API**. You can also use *raw* **MemoryBuffer** provided by **Storage** itself if no graphics is desired. The **ControlPointView** is bound to **ControlPointFormat** which specifies the data organisation of specific control point. The view can also be assigned an offset and number of control points for maximum flexibility.

**ControlPointFormat** is similiar to **VertexFormat** except that:

* The names of components are strings, not precreated flags. This means that you can have any number of components (64 is not the limit) and names can be more representative;
* Each component can have an optional description string;
* All fixed-length types are can be written, including strings (8-character strings etc.);

**IShapeView** is a base interface for all shape views. A shape view usually(but not necessarily) links a buffer that is used for indexing (for control point based resources). The only requirement is that the buffer is mappable. A shape view is then specialised for specific shape types and organisations.

Many views also provide **IShapeBuilder** interface that allows removing, inserting, adding and replacing shapes.

# Usage scenarios

A set of usage scenarios (especially harder ones) are given in this section.

## Expression reusage

We can write something like this to create expression:

Expression e = Expression.FromString(“k\*x+n”);  
Expression.FunctionParams p = e.Params;  
// Bind parameters.  
p[“k”] = 1.0;  
p[“n”]=-3.0;  
  
// We bind ‘x’ to first parameter.  
p.SetBinding(“x”,e.Variable(0));   
  
// We create function.  
Functiond delegate = e.GetFunctiond(p, FunctionSet.Default);  
  
// Simply alter the parameters.  
p[“k”]=2.0;  
Functiond delegate2 = e.GetFunctiond(p, FunctionSet.Default);

Note that delegate is independent of expression and parameter set after it is constructed. Parameters are not thread safe.

## Operator overloads

We can use overloaded operators to achieve goals. The code is clearer this way.

Matrix4x4d m1 = GetSomeMatrix1();  
Matrix4x4d m2 = GetSomeMatrix2();  
Matrix4x4d r = m1\*m2;  
  
// We now transform using resulting matrix.  
Vector4d v = r \* Vector4d.AxisX;

# Implementation Notes

## ToString method

Each non-static class should implement this method and provide full information about the class in it. This is very useful when debugging.

## Unstability issues

The math system is algorithm-dense system with a lot of numerical analysis. At some stages, it can be unstable. When such numerical unstable algorithms are introduced, the algorithms must state in it's remarks when and why it is unstable.

## Performance vs. Usability

Sometimes, there are performance versus usability conflicts. For example, generic root finding algorithms is slow but useful, while specific polynomial root finders (quadratic equation) are fast but not so widely usable. In such cases, you should code both; the fast (and in this case also exact solution) and the generic (and slower solution).

1. No arrays needed, arrays are objects in C# and this implies an indirect link when accessing elements in array. [↑](#footnote-ref-2)