

VECTORS MATRICES AND C++ CODE

Professionally typeset

Basic level.

Hundreds of live crossreferences

Live Index

Live Table of Contents

Live Bibliography

Live Internet links

Live C++ code documentation

Professional C++ code included

New to C++? C++ Course included

Sergio Pissanetzky

Vectors, Matrices, and C++ Code

Sergio Pissanetzky

2004

Copyright © 2004 by Sergio Pissanetzky. All rights reserved. No part of the contents of this book can be reproduced without the written permission of the publisher.

Professionally typeset by L^AT_EX

Dr. Pissanetzky retired after a rewarding career as an Entrepreneur, Professor, Research Scientist and Consultant. He was the founder of Magnus Software Corporation, where he focused on development of specialized applications for the Magnetic Resonance Imaging (MRI) and the High Energy Particle Accelerator industries. He has served as Member of the International Editorial Board of the “International Journal for Computation in Electrical and Electronic Engineering”, as a Member of the International Advisory Committee of the International Journal “Métodos Numéricos para Cálculo y Diseño en Ingeniería”, and as a member of the International Committee for Nuclear Resonance Spectroscopy, Tokyo, Japan. Dr. Pissanetzky has held professorships in Physics at Texas A&M University and the Universities of Buenos Aires, Córdoba and Cuyo, Argentina. He has also held positions as a Research Scientist with the Houston Advanced Research Center, as Chairman of the Computer Center of the Atomic Energy Commission, San Carlos de Bariloche, Argentina, and as a Scientific Consultant at Brookhaven National Laboratory. Dr. Pissanetzky is currently a member of the Advisory Board of [Meedio, LLC](#). Dr. Pissanetzky holds several US and European patents and is the author of two books and numerous peer reviewed technical papers. Dr. Pissanetzky earned his Ph.D. in Physics at the Balseiro Institute, University of Cuyo, in 1965. Dr. Pissanetzky has 35 years of teaching experience and 30 years of programming experience in languages such as Fortran, Basic, C and C++. Dr. Pissanetzky now lives in a quite suburban neighborhood in Texas.

Website: <http://www.SciControls.com>

Trademark Notices

Microsoft[®], Windows[®] and Visual C++[®] are registered trademarks of Microsoft Corporation.

Java[™] and Sun[™] are trademarks of Sun Microsystems, Inc.

UNIX[®] is a registered trade mark licensed through X/Open Company, Ltd.

PostScript[®], PDF[®] and Acrobat Reader[®] are registered trademarks of Adobe Systems, Inc.

Merriam-Webster[™] is a trademark of Merriam-Webster, Inc.

VAX[™] is a trademark of Digital Equipment Corporation.

Other product and company names mentioned herein may be the trademarks of their respective owners.

ISBN 0-00-000000-0

Preface

Presented here is an integrated approach - perhaps the first in its class - of the basics of vector and matrix Algebra with the object-oriented C++ code that implements the vectors and matrix objects and brings them to life. This is not the traditional road map followed by textbooks, and a rationale is needed.

The concept of object has existed in Science for centuries. An object associates properties and behavior in a single, inseparable entity, an abstraction consisting of properties and a description of their behavior. Although the term *object* is not used in Science, scientists always did all their thinking in terms of objects. In Physics, a rigid body, an atom, an electromagnetic wave, are all objects, because they all have properties and behavior. The laws of Physics describe their behavior in terms of their properties. Similarly, in Mathematics, a vector, a graph, even a number, all have properties and behavior and are therefore objects.

As a matter of fact, thinking in terms of objects is the natural way of thinking, perhaps the only way of thinking, and we all practice it every day. A piece of paper is an object, it has color, size, weight, shape. It can be printed upon, it can absorb water, it can burn, it can be folded or torn and hold its new shape. A bank account is an object, it has a balance and rules that govern how funds can be deposited, withdrawn or transferred, and how interest is earned. Man knew objects for as long as humanity existed, primitive objects such as food or shelter, and more technological objects such as fire or the wheel.

More recently, the concept of object was introduced in Computation, and object-oriented languages were created. A computational object is a model of the real object, an abstraction. So also are our thoughts. In either case, creating the object is difficult because the concept of modeling or abstracting the real world is involved. Among the new languages, C++ became very popular, and is our choice because it offers full and efficient support for all the mathematical calculations we need, and it is entirely built around object-oriented concepts.

Yet, the concept of object is not routinely used when teaching Science, frequently not even mentioned. And the idea that objects can come alive in a computer remains relegated to specialized areas of Computer Science and has not been fully exploited.

As a result, the world of disciplines is divided into the world of science and the world of code. Books on science are written by scientists who may not be very used to coding, software documentation and books on coding are written by developers and computer scientists who may not be very interested in Mathematics or Physics. And books on Computational Physics tend to concentrate on the specialized mathematical methods used to solve the problems but offer at best a background of the mathematical and physical concepts and little or nothing about the code. Many simply refer the reader to a commercial implementation of the methods, thus neatly separating the two worlds.

Where is the scientist who has never felt the need to calculate some numbers in support of his/her research? Where is the developer who has never felt the need for science when writing code? Where is the user who has never been confused with code documentation written by someone not

very familiar with a right-handed coordinate system or a 0-based array index?

This book is intended to provide an integrated approach to basic vector and matrix Algebra with object-oriented concepts and the actual code implementing them. The source code is included and readers are free to use it for their own work provided proper credit is always given. Coding is about empowerment, and we want to empower the reader with the ability to create his/her own live objects and cause them to act their parts. This is the first volume of what we expect to be a series covering, approximately, the following topics of Mathematics and Physics: Coordinate Transformations, Graphs, Sparse Matrices [3], Linear Equations, the Dynamics of Multibody Systems, and the intelligent control of Articulated Multibody Robotic Structures. Other titles may be added as needed. All of them, of course, with the corresponding source code included.

This product is both a textbook and a software release. The book can be regarded as very complete software documentation, consisting not only of the description of classes, attributes and methods, but also of the mathematical background that supports the code. Just as business code is best understood by those with a background in that business, scientific code is best understood by those with a background in mathematics.

The code presented here is in no way new. Its roots date back more than 30 years. It evolved from its original non-object-oriented Fortran versions to its current fully object-oriented versions in C++. It grew as it evolved, becoming part of several well known professional scientific programs such as Kubik [5], Magnus [6], Epilog [7], PhysicSolver [9], and others, and it served engineering applications such as the design of Particle Accelerators, Magnetic Resonance Imaging systems, thermal systems for nuclear reactors, and many others, and the teaching of Science.

To understand this book you will need a basic knowledge of Mathematical notation, Algebra and Trigonometry. If you are not familiar with C++ or object-oriented methodology, you can learn it right here, because a basic course on C++ and computational objects is included. It is not a regular course, because as you learn you can refer directly to the included professional code, something not usually available in a traditional course. If you are not interested in learning vectors and matrices but only in using the code, you can still do so. The code documentation has links to the underlying mathematical concepts, or it can be understood even if you ignore them. We do encourage you to check the concepts, however, you can learn some Mathematics in the process. You should read this book if:

- You are a developer and you need a background in vector or matrix algebra.
- You are a science student and you need to learn C++.
- You are a scientist or a science student and you need to write advanced code but you don't want to waste time developing the basics.
- You need ready-to-use C++ source code for your science project.

We could have released our material as html web pages to the Internet, but we have chosen to present it in pdf format and in the form of an electronic book, an eBook. Web pages are sometimes

disconcerting, and the navigational adventure may be chaotic because the emphasis in light, color and motion creates confusion. One can write a whole encyclopedia in web pages and nobody will ever realize the magnitude or comprehend the extent or content of the work. Concepts are scattered and difficult to relate to each other, or even to find.

EBooks, instead, offer unity and integrity of content. The reader travels between boundaries with a clear notion of content, quality and organization of the information. Modern eBooks have all the live cross references and internal and external links that one finds in web pages. eBooks are free from advertising, and they can talk. eBooks are sturdy, pages never wear out or get loose. Our eBook has a live table of contents, a live index, live bibliography, live references, live comments, and external links to the Internet.

There are currently thousands of eBooks covering many areas of human knowledge or entertainment. In a sense, eBooks are changing the very art of technical book writing. Using links and cross references, it has now become possible to include advanced and even encyclopedic content without disturbing the natural flow of the subject, something not found in paper textbooks, and something authors are not yet accustomed to. eBooks are even ideal to be used as a reference for the subject matter because of the many live internal links.

If you are reading this, you are probably using the Adobe Reader. Just a couple of suggestions you may find useful. Run the reader by itself, not inside a browser, to maximize the amount of screen area available to you. Magnify as much as you can to improve readability. But try to see a whole page on the screen, even if a piece of the Reader is outside the screen. You can use the left and right arrows in your keyboard to change pages. Internal links will also bring you to another page, and you can return to where you were using alt-left arrow, even repetitively. Ctrl-2 sets fit-to-page and lets you adjust the magnification by dragging the edge of the window until the height of the page is as tall as fits in the window. Also ctrl-m is very useful, it lets you type-in the magnification you want. There are other good pdf readers as well, for example GSview from [Ghostsceipt](#).

Please drop us a line when you feel like it, just make sure your e-mail is clearly identified so it doesn't get confused with spam. We will try to respond via frequently asked questions in our web site, so please read that material before you write. Please consider that we are a few, you are many. Enjoy!

Sergio Pissanetzky
October 2004

Contents

1	Introduction	1
1.1	Objectives and Road Map	1
1.2	Code Design Considerations	1
1.2.1	Object Oriented Programming	1
1.2.2	Code	2
1.2.3	Style	3
1.2.4	Efficiency	4
1.2.5	Operators	4
2	Scalars and Vectors	7
2.1	Basics	7
2.2	The graphic representation of vectors	8
2.3	Operations with vectors	9
2.4	Unit vectors	11
2.5	Components	12
2.6	Vector operations using components	14
2.7	Dot product	15
2.8	Cross product	17
2.9	Multidimensional vectors	19
2.10	Row vectors and column vectors	19
3	Matrices	21
3.1	Basics	21
3.2	Operations with matrices	22
3.3	Specialized matrices	24
3.4	Vectors as matrices	26
4	Basic C++ Programming	29
4.1	Introduction	29
4.2	C++ Essentials	31

4.2.1	Writing Classes	31
4.2.2	Names and Types	32
4.2.3	Operators	33
4.2.4	Constructors and Destructors	35
4.2.5	Pointers and References	36
4.2.6	Functions	38
4.2.7	Visibility	40
4.2.8	Arrays	40
4.2.9	Pointer Arithmetic	42
4.2.10	Compound Statements	43
4.3	Polymorphism in C++	48
4.3.1	Families of classes	48
4.3.2	Virtual Functions	50
4.3.3	Constant Objects, Functions and Pointers	52
4.3.4	Overloading Functions	53
4.3.5	Creating and Using Objects	53
4.3.6	Operators new and delete	54
4.3.7	Separating Declarations and Definitions	55
4.3.8	Parameterized Classes	59
4.4	Conclusion	62
5	Vector and Matrix Code	63
5.1	The Families of Classes	63
5.2	Equations, Algorithms and Programs	63
5.3	Method Design and Naming Style	64
5.4	All Families	66
5.5	All Classes	67
6	The Vector Family of Classes	69
6.1	All Vector Classes	69
6.2	Class PVector	69
6.2.1	PVector Attribute Detail	73
6.2.2	PVector Constructor Detail	74
6.2.3	PVector Method Detail	75
6.3	Class Vector2	82
6.3.1	Vector2 Constructor Detail	87
6.3.2	Vector2 Method Detail	88
6.4	Class Vector3	98
6.4.1	Vector3 Constructor Detail	103
6.4.2	Vector3 Method Detail	104

6.5	Class <code>UnitVector3</code>	118
6.5.1	<code>UnitVector3</code> Constructor Detail	122
6.5.2	<code>UnitVector3</code> Method Detail	123
6.6	Class <code>VectorN</code>	124
6.6.1	<code>VectorN</code> Constructor Detail	127
6.6.2	<code>VectorN</code> Method Detail	128
7	The Matrix Family of Classes	131
7.1	All Matrix Classes	131
7.2	Class <code>PMatrix</code>	131
7.2.1	<code>PMatrix</code> Attribute Detail	135
7.2.2	<code>PMatrix</code> Constructor Detail	135
7.2.3	<code>PMatrix</code> Method Detail	136
7.3	Class <code>Matrix</code>	145
7.3.1	<code>Matrix</code> Constructor Detail	150
7.3.2	<code>Matrix</code> Method Detail	151
7.4	Class <code>Matrix3</code>	165
7.4.1	<code>Matrix3</code> Constructor Detail	171
7.4.2	<code>Matrix3</code> Method Detail	172
7.5	Class <code>Matrix3X4</code>	190
7.5.1	<code>Matrix3X4</code> Constructor Detail	193
7.5.2	<code>Matrix3X4</code> Method Detail	193
8	The Array Family of Classes	197
8.1	All Array Classes	197
8.2	Class <code>PArray</code>	197
8.2.1	<code>PArray</code> Attribute Detail	200
8.2.2	<code>PArray</code> Constructor Detail	201
8.2.3	<code>PArray</code> Method Detail	202
8.3	Class <code>ArrayOfCStr</code>	210
8.3.1	<code>ArrayOfCStr</code> Constructor Detail	211
8.3.2	<code>ArrayOfCStr</code> Method Detail	212
8.4	Class <code>ArrayOfDoubles</code>	213
8.4.1	<code>ArrayOfDoubles</code> Constructor Detail	217
8.4.2	<code>ArrayOfDoubles</code> Method Detail	219
8.5	Class <code>ArrayOfIntegers</code>	226
8.5.1	<code>ArrayOfIntegers</code> Constructor Detail	230
8.5.2	<code>ArrayOfIntegers</code> Method Detail	232
8.6	Class <code>ArrayOfStr</code>	238
8.6.1	<code>ArrayOfStr</code> Constructor Detail	240

8.6.2	ArrayOfStr Method Detail	241
9	The String Family of Classes	243
9.1	All String Classes	243
9.2	Class CStr	243
9.2.1	CStr Attribute Detail	250
9.2.2	CStr Constructor Detail	250
9.2.3	CStr Method Detail	252
9.3	Class Str	279
9.3.1	Str Attribute Detail	288
9.3.2	Str Constructor Detail	288
9.3.3	Str Method Detail	292
9.4	Class Formatter	330
9.4.1	Formatter Attribute Detail	332
9.4.2	Formatter Constructor Detail	333
9.4.3	Formatter Method Detail	333
A	Terms, Conventions and Definitions	335
A.1	Terminology	335
A.2	Cabinet Drawing	336
A.3	Coordinate Systems	337
A.4	Right-handed and left-handed systems	338
A.5	Coordinate transformations	339

Now let \mathbf{a} and \mathbf{b} be two given vectors, which we can express in components as was done in equation 2.7. The dot product is given by:

$$\mathbf{a} \cdot \mathbf{b} = (a_x \mathbf{i} + a_y \mathbf{j} + a_z \mathbf{k}) \cdot (b_x \mathbf{i} + b_y \mathbf{j} + b_z \mathbf{k}) \quad (2.20)$$

Using the associative property to expand this, and using equations 2.19 to calculate the values of several resulting dot products, we obtain:

$$\mathbf{a} \cdot \mathbf{b} = a_x b_x + a_y b_y + a_z b_z \quad (2.21)$$

Equation 2.21 tells us how to calculate the dot product between two vectors directly from their components. The dot product of a vector by itself is the square of its magnitude.

2.8 Cross product

The *cross product*, also known as the *vector product*, of two given vectors \mathbf{a} and \mathbf{b} , is represented as $\mathbf{a} \times \mathbf{b}$, and is defined as another vector \mathbf{c} :

$$\mathbf{c} = \mathbf{a} \times \mathbf{b} \quad (2.22)$$

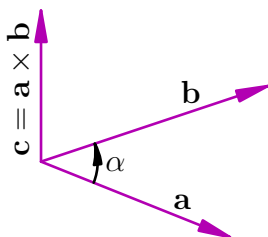


Figure 2.10: The cross product.

Vector \mathbf{c} is perpendicular to the plane determined by vectors \mathbf{a} and \mathbf{b} , its direction is determined in such a way that the three vectors \mathbf{a} , \mathbf{b} and \mathbf{c} form a [right-handed](#) system, and its magnitude is the product of the magnitude of \mathbf{a} , the magnitude of \mathbf{b} , and the sine of the angle between the positive directions of \mathbf{a} and \mathbf{b} :

$$c = a b \sin(\alpha) \quad (2.23)$$

where α is the angle between \mathbf{a} and \mathbf{b} . Figure 2.10 illustrates an example. If \mathbf{a} and \mathbf{b} are parallel or antiparallel, the result of the vector product is the [zero vector](#). This is because $\alpha = 0$ or $\alpha = \pi$, respectively, and $\sin \alpha = 0$ in both cases. If \mathbf{a} and \mathbf{b} are not parallel, then their directions do in

and the elements of \mathbf{C} are those of \mathbf{A} multiplied by s :

$$C_{ij} = sA_{ij} \quad (3.8)$$

for all i and j . The matrix and the number commute:

$$s\mathbf{A} = \mathbf{A}s \quad (3.9)$$

The operation of multiplication can be defined for two matrices of any size or shape, provided they *conform* to each other in the order they are given. Two matrices \mathbf{A} and \mathbf{B} are said to conform in that order if the number of columns of \mathbf{A} is equal to the number of rows of \mathbf{B} . Let \mathbf{A} be a $p \times q$ matrix, and let \mathbf{B} be a $q \times r$ matrix. Then matrix \mathbf{C} is the product of \mathbf{A} and \mathbf{B}

$$\begin{array}{c} \mathbf{C} \\ p \times r \end{array} = \begin{array}{cc} \mathbf{A} & \mathbf{B} \\ p \times q & q \times r \end{array} \quad (3.10)$$

if \mathbf{C} is of order $p \times r$ and its elements are calculated as follows:

$$C_{ik} = \sum_{j=1}^q A_{ij}B_{jk} \quad (3.11)$$

for all i and all k . At this point it is important to explain the mechanics of these equations. In 3.10, we have indicated the sizes of the matrices. On the right-hand side, the adjacent size parameters of the two pairs $(p \times q)$, $(q \times r)$ both have the same value, q . Upon effecting the product, the repeated parameter q disappears, leaving only the pair $p \times r$ on the left-hand side. This mnemonic is very useful.

We now examine equation 3.11, which is actually a set of pr equations, one for each value of i and one for each value of k . Consider one of the equations in the set, where i has a certain value and k has a certain value. The summation index on the right-hand side is j , while i and k remain constant and with the same values they have on the left-hand side. This means that the entire row i of \mathbf{A} , and the entire column k of \mathbf{B} are involved in the summation, and all this just for the single element i, j of \mathbf{C} . If we think of row i of \mathbf{A} as a vector with q components, and the column k of \mathbf{B} as another vector also with q components, then the calculations implied in 3.11 are the same as in equation 2.30, which describes the dot product of two multidimensional vectors. In summary: element C_{ik} of matrix \mathbf{C} is calculated as the dot product of row i of \mathbf{A} and column k of matrix \mathbf{B} . This is, again, a very useful mnemonic.

Matrix multiplication can be extended to the case of multiple factors. For example:

$$\begin{array}{c} \mathbf{D} \\ p \times s \end{array} = \begin{array}{ccc} \mathbf{A} & \mathbf{B} & \mathbf{C} \\ p \times q & q \times r & r \times s \end{array} \quad (3.12)$$

where the sizes of the matrices are indicated underneath their names, the product is done by first multiplying \mathbf{A} and \mathbf{B} , and then multiplying the result by \mathbf{C} . Note that all adjacent pairs of factors

Chapter 4

Basic C++ Programming

Only experience can make you a good C++ developer. This course will give you the knowledge you need to understand the basics of C++ programming and to begin working with C++ code. C++ is a language, a language that compilers understand, and it requires practice just like any other language.

This course is exceptional in at least one respect. Normally, a C++ course explains the concepts, provides some examples and perhaps some exercises, and leaves the reader right there, without much access to professional code. In this case, however, we have taken advantage of the fact that the professional [Vector and Matrix](#) code is part of the eBook. The student is taken seamlessly from the basic concepts explained in the course, to the actual code. The code is extensively documented and cross-references between course material and code are provided. Students are given a unique chance to practice and develop their skills quickly to a level that only developers with years of experience can achieve.

Still, none of this is a substitute for the actual experience of developing code.

If you are an experienced C++ developer, you can skip this entire chapter and go directly to the professional code presented in the following chapters. But if you are not familiar with C++, you must peruse this material first.

4.1 Introduction

C++ programming is done in terms of *objects*. An object associates properties and behavior in a single, inseparable entity, a computational abstraction consisting of data and a description of the behavior of that data. Similar objects are grouped into classes, and a particular object is said to be an instance of its class.

Objects have been used in Science for centuries. In Physics, a solid body, an atom, an electromagnetic wave, are all objects, because they all have properties and behavior. The laws of Physics describe their behavior in terms of their properties. Similarly, in Mathematics, a vector, a matrix,

5.5 All Classes

Class name	Description
ArrayOfCStr	An array of variable length that can hold pointers to objects of type CStr . The array of pointers is owned by the class and is deleted in the destructor. The objects, however, are not.
ArrayOfDoubles	An array of variable length that can hold doubles.
ArrayOfIntegers	An array of variable length that can hold integers.
ArrayOfStr	An array of variable length that can hold pointers to objects of type Str . The array of pointers is owned by the class and is deleted in the destructor. The objects, however, are not.
CStr	A class defining a substring that is part of a larger NULL-terminated string or an Str object.
Formatter	A class that supports <code>printf</code> -style formatting for Str and CStr objects.
Matrix	A general rectangular matrix.
Matrix3	A square matrix of numbers of order 3 x 3.
Matrix3X4	A rectangular matrix of numbers of order 3 x 4.
PArray	A parameterized class used as base for all Array classes. It has an array of variable length that can hold pointers to objects of any type. The array of pointers is owned by the class and is deleted in the destructor. The objects, however, are not.
PMatrix	The abstract class used as base for all Matrix classes. It describes a general rectangular matrix of numbers.
PVector	The base class for all Vector classes. It describes a vector in a space with any number of dimensions.
Str	A NULL-terminated array of characters. It is owned by the class and deleted by the destructor when the object is destructed.
UnitVector3	A unit vector in a three-dimensional space.
Vector2	A vector in a two-dimensional space.
Vector3	A vector in a three-dimensional space.
VectorN	A vector in a space with any number of dimensions.

Method Summary	
public: virtual void	<code>Create(int nComps)</code> Creates a <code>PVector</code> object with the specified number of components.
public: virtual void	<code>Create(int nComps, const double * comps)</code> Creates a <code>PVector</code> object with the specified components.
public: virtual double	<code>DotProduct(const PVector & other)</code> Calculates the dot product of this vector and another given vector.
public: virtual int	<code>FindLargestComponent()</code> Returns the index of the component with the largest absolute value.
public: virtual int	<code>FindSmallestComponent()</code> Returns the index of the component with the smallest absolute value.
public: inline const double &	<code>GetComponent(int index)</code> Returns the value of a component of this vector.
public: inline const double *	<code>GetComponents()</code> Returns a pointer to the the <code>components</code> array of constant doubles.
public: void	<code>GetCopyOfComponents(double * copy)</code> Returns by argument copies of the components of this vector in an array.
public: virtual double	<code>GetMyLength()</code> Returns the length or magnitude of this vector.
public: virtual double	<code>GetMyLengthSquared()</code> Returns the square of the length or magnitude of this vector.
public: inline double *	<code>GetNonconstantComponents()</code> Returns a pointer to the <code>components</code> array.
public: inline int	<code>GetNumberComponents()</code> Returns the number of components of this <code>PVector</code> object.
public: virtual bool	<code>IsZero()</code> Returns <code>true</code> if this vector is zero, <code>false</code> otherwise.
public: virtual void	<code>MakeMeZero()</code> Sets this vector to the zero vector.
public: virtual void	<code>MeEqualsDifferenceBetween(const PVector & a, const PVector & b)</code> Sets this vector equal to the difference between two given vectors.
public: virtual void	<code>MeEqualsMeMinus(const PVector & other)</code> Subtracts a given vector from this vector.
public: virtual void	<code>MeEqualsMePlus(const PVector & other)</code> Adds a given vector to this vector.
public: virtual void	<code>MeEqualsMeTimesConstant(double f)</code> Multiplies this vector by a constant.
public: virtual void	<code>MeEqualsMeTimesDiagonalMatrix(const double * diag)</code> Left-multiplies this vector by a given diagonal matrix.
public: virtual void	<code>MeEqualsMinusMe()</code> Changes the sign of this vector.

6.2.2 PVector Constructor Detail

public: PVector::PVector(int nComps, const double * comps)

Components constructor. It sets `m_nComponents` to `nComps` , invokes operator `new` to create array `components` of size `m_nComponents` , and copies `m_nComponents` values from the given array `comps` into array `components` .

Arguments

- `nComps` Number of components.
- `comps` A pointer to a constant array of doubles that contains the values for the components.

public: PVector::PVector(const PVector & other)

Copy constructor. It sets this vector identical to the given vector by copying the value of `m_nComponents` and all element values from `other` to this object.

Arguments

- `other` The given `PVector` object.

public: PVector::PVector(int nComps)

Size constructor. It sets `m_nComponents` to `nComps` and invokes operator `new` to create array `components` of size `m_nComponents` , but leaves the elements of the array uninitialized.

Arguments

- `nComps` Number of components.

public: PVector::PVector()

Default constructor. It sets `components` to `NULL` .

public: virtual PVector::~~PVector()

The virtual destructor. It deletes the array pointed at by `components` if this pointer is not `NULL` .

Inherited from Vector3

```
protected: int m_nComponents;
protected: double * components;
public: double Angle(const Vector3 & q);
public: double AngleDirected(const Vector3 & q, const Vector3 & m);
public: bool Create(const CStr & text);
public: bool Create(const Str & text);
public: bool Create(double x, double y, double z);
public: virtual void Create(int nComps);
public: virtual void Create(int nComps, const double * comps);
public: Vector3 CrossVector(const Vector3 & uu);
public: virtual double DotProduct(const PVector & other)const;
public: double DotVector(const Vector3 & uu);
public: Vector3 DoubleCrossVector(const Vector3 & uu);
public: int FindLargestComponent();
public: int FindSmallestComponent();
public: Vector3 GenerateArbitraryNormal();
public: double GetComponent(int index);
public: void GetComponents(double &ux, double &uy, double &uz);
public: void GetComponents(double * array);
public: inline const double * GetComponents();
public: void GetCopyOfComponents(double * copy)const;
public: double GetMyLength();
public: double GetMyLengthSquared();
public: Vector3 GetMyPNormal(const Vector3 & w);
public: Vector3 GetMyUnit();
public: bool GetMyUnit(Vector3 & unit);
public: inline double * GetNonconstantComponents();
public: inline int GetNumberComponents();
public: Matrix3 GetWMatrix();
public: bool IsZero();
public: double MakeMeUnit();
public: void MakeMeZero();
public: void MeEqualsDifferenceBetween(const Vector3 & a, const Vector3 & b);
public: virtual void MeEqualsDifferenceBetween(const PVector & a, const PVector & b);
public: virtual void MeEqualsMeMinus(const PVector & other);
public: virtual void MeEqualsMePlus(const PVector & other);
public: virtual void MeEqualsMeTimesConstant(double f);
public: virtual void MeEqualsMinusMeMinus(const PVector & other);
```

Chapter 8

The Array Family of Classes

8.1 All Array Classes

Class name	Description
ArrayOfCStr	An array of variable length that can hold pointers to objects of type CStr . The array of pointers is owned by the class and is deleted in the destructor. The objects, however, are not.
ArrayOfDoubles	An array of variable length that can hold doubles.
ArrayOfIntegers	An array of variable length that can hold integers.
ArrayOfStr	An array of variable length that can hold pointers to objects of type Str . The array of pointers is owned by the class and is deleted in the destructor. The objects, however, are not.
PArray	A parameterized class used as base for all Array classes. It has an array of variable length that can hold pointers to objects of any type. The array of pointers is owned by the class and is deleted in the destructor. The objects, however, are not.

8.2 Class PArray

A parameterized (template) container class representing an array of variable length containing objects or pointers to objects of any type. For types other than primitive types, the default constructor is used to initialize the items in the array, and therefore the existence of a default constructor is required for such types. The class offers support for all standard array operations, including inserting, appending or prepending items or other arrays, sorting, and removing and extracting items or arrays. The single type parameter is called `TYPE`. The class can be used to declare arrays directly, for example:

Index

- Algorithm**, [63](#)
- Antisymmetric matrix**, [24](#)
- Argument**
 - return, [39](#)
- Arguments**
 - formal, [38](#)
- Array**
 - classes, [197](#)
 - family of classes, [197](#)
 - of characters, [41](#), [238](#)
 - of doubles, [213](#)
 - of integers, [226](#)
- ArrayOfCStr**
 - class, [210](#)
 - constructor details, [211](#)
 - method details, [212](#)
- ArrayOfDoubles**
 - class, [213](#)
 - constructor details, [217](#)
 - method details, [219](#)
- ArrayOfIntegers**
 - class, [226](#)
 - constructor details, [230](#)
 - method details, [232](#)
- ArrayOfStr**
 - class, [238](#)
 - constructor details, [240](#)
 - method details, [241](#)
- Arrays**, [40](#)
- ASCII**
 - character codes, [41](#)
 - character set, [41](#)
- Attributes**, [31](#)
- Band matrix**, [25](#)
- Behavior**, [31](#)
- Block matrix**, [25](#)
- BSTR**, [283](#), [285](#), [286](#), [309](#), [315](#), [317](#), [320](#)
- C++**
 - essentials, [31](#)
 - programming, [29](#)
- Cabinet**
 - drawing, [336](#)
 - representation, [336](#)
- Cartesian coordinate system, [337](#)
- Case label**, [47](#)
- char**, [41](#)
- Character array**, [41](#)
 - NULL terminated, [41](#)
- Character codes**, [41](#)
- Circular inclusions**
 - preventing, [58](#)
- Class**, [31](#)
 - derived, [31](#)
 - instance, [31](#)
 - members, [31](#)
 - prototype, [56](#)
 - root, [50](#)
 - templates, [59](#)
- Class ArrayOfCStr**
 - AppendCStrIfNotThere, [212](#)
 - ArrayOfCStr, [211](#)
 - ~ArrayOfCStr, [212](#)
 - FindCStr, [212](#)

Class ArrayOfCStr (cntd)

- constructor details, 211

- method details, 212

Class ArrayOfDoubles

- AddArray, 219

- AppendElement, 219

- AppendElementIfNotThere, 220

- ArrayOfDoubles, 217, 218

- ~ArrayOfDoubles, 218

- ChangeSign, 220

- CopyArray, 220

- CopyMinusArray, 220

- DotProduct, 221

- FillWithConstant, 221

- FindElement, 221

- GetLengthSquared, 221

- IsZero, 221

- MakeMeZero, 222

- MultiplyByConstant, 222

- operator*, 222

- operator*=: 222

- operator+, 223

- operator+=, 223

- operator-, 223

- operator-=, 224

- operator/, 224

- operator/=, 224

- operator=, 225

- operator const double *, 219

- ReportHorizontal, 225

- ReportVertical, 226

- SubtractArray, 226

Class ArrayOfIntegers

- AddArray, 232

- AppendElement, 232

- AppendElementIfNotThere, 233

- ArrayOfIntegers, 230, 231

- ~ArrayOfIntegers, 231

- ChangeSign, 233

- CopyArray, 233

- CopyMinusArray, 233

- FindElement, 234

- InitializeWithConstant, 234

- InitializeWithSequence, 234

- IsZero, 234

- MakeMeZero, 234

- operator+, 235

- operator+=, 235

- operator-, 235

- operator-=, 236

- operator=, 236

- operator const int *, 232

- ReportHorizontal, 236

- ReportVertical, 237

- SortMe, 237

- SubtractArray, 237

Class ArrayOfStr

- AppendStr, 241

- AppendStrIfNotThere, 241

- ArrayOfStr, 240

- ~ArrayOfStr, 240

- FindStr, 241

- operator=, 242

Class CStr

- AtoF, 252

- AtoI, 252

- Compare, 253

- ConvertMeToNumber, 254

- CountBracketedGroups, 254

- CountLines, 255

- CountNonemptyLines, 255

- CountNonemptyWords, 255

- CountOccurrences, 255

- Create, 256

- CStr, 250, 251

- ~CStr, 251

- Find, 256, 257

- FindChar, 257

- FindExclusive, 258

- FindMatchingBracket, 258

Class CStr (cntd)

FindMatchingBracketExclusive, 259
 FindNextCommand, 260
 FindNextCommandExclusive, 261
 FindNextNonalnumDelimitedWord, 261
 FindNextSpaceDelimitedWord, 262
 FindNthCommandWithTitle, 262
 FindNthCommandWithTitleExclusive, 263
 FindNthLineStartingWithWord, 263
 FindPreviousNonalnumDelimitedWord, 264
 FindPreviousSpaceDelimitedWord, 264
 FindTextBetween, 265
 GetAt, 266
 GetMainString, 266
 GetNumberCharacters, 266
 GetOffset, 266
 GetOffsetPastEnd, 266
 GetRangeString, 267
 GetTextBetweenBrackets, 267
 GetTextBetweenBracketsExclusive, 268
 isAlnum, 268
 isAlpha, 268
 isDigit, 269
 IsEmpty, 269
 isGraph, 269
 isLower, 269
 isPrint, 269
 isPunct, 270
 isSpace, 270
 isUpper, 270
 Left, 270
 MainString, 250
 Mid, 271
 NumberCharacters, 250
 Offset, 250
 operator!=, 271, 272
 operator=, 274
 operator==, 274
 operator[], 277
 operator>, 275

operator>=, 276
 operator<, 272
 operator<=, 273
 Right, 277
 SeparateIntoLines, 277
 SeparateIntoNonemptyLines, 278
 SeparateIntoNonemptyWords, 278
 Trim, 278
 TrimLeft, 278
 TrimRight, 278
 TSQ, 279
 Verify, 279

Classes

all, 67
 ArrayOfCStr, 210
 ArrayOfDoubles, 213
 constructor details, 217
 method details, 219
 ArrayOfIntegers, 226
 constructor details, 230
 method details, 232
 ArrayOfStr, 238
 constructor details, 240
 method details, 241
 CStr, 243
 Attribute details, 250
 constructor details, 250
 method details, 252
 Formatter, 330
 Attribute details, 332
 constructor details, 333
 method details, 333
 Matrix, 145
 constructor details, 150
 method details, 151
 Matrix3, 165
 constructor details, 171
 method details, 172
 Matrix3X4, 190
 constructor details, 193

- method details, 193
- PArray, 197
 - Attribute details, 200
 - constructor details, 201
 - method details, 202
- PMatrix, 131
 - Attribute details, 135
 - constructor details, 135
 - method details, 136
- PVector, 69
 - Attribute details, 73
 - constructor details, 74
 - method details, 75
- Str, 279
 - Attribute details, 288
 - constructor details, 288
 - method details, 292
- UnitVector3, 118
 - constructor details, 122
 - method details, 123
- Vector2, 82
 - constructor details, 87
 - method details, 88
- Vector3, 98
 - constructor details, 103
 - method details, 104
- VectorN, 124
 - constructor details, 127
 - method details, 128
- Class families**, 63, 66
- Class Formatter**
 - charLen, 333
 - ConversionSpec, 332
 - EstimateLengthOfFormattedString, 333
 - Formatter, 333
 - ~Formatter, 333
 - intLen, 332
 - isConversionChar, 334
 - longLen, 332
 - shrtLen, 332
 - uintLen, 332
 - ulongLen, 332
 - ushrtLen, 332
- Class Matrix**
 - AddToElement, 151
 - ChangeSignOfElement, 151
 - CopySparseMatrix, 152
 - CreateArbitraryPositiveDefiniteMatrix, 152
 - DepermuteMe, 153
 - DivideElement, 153
 - GetElement, 153
 - GetListOfNonzeroColumns, 154
 - GetMyTranspose, 154
 - GetNumberCols, 154
 - GetNumberElements, 154
 - GetNumberNonzeroColumns, 155
 - GetNumberNonzeroRows, 155
 - GetNumberRows, 155
 - IsPositiveDefinite, 156
 - IsProperDiagonalDominant, 156
 - IsRowZero, 156
 - IsSquare, 157
 - IsSymmetric, 157
 - IsUpperTriangular, 157
 - Matrix, 150, 151
 - ~Matrix, 151
 - MeCrossVector, 157
 - MeTimesColumnVector, 158
 - MeTimesDiagonalTimesMeTransposed, 158
 - MeTimesMatrix, 159
 - MeTimesMatrixTimesMeTransposed, 159
 - MeTimesMatrixTransposed, 159
 - MeTransposedTimesDiagonalTimesMeAccumulate, 160
 - MeTransposedTimesDiagonalTimesMeFill, 160, 161
 - MeTransposedTimesMatrix, 161
 - MeTransposedTimesMatrixTimesMe, 161
 - MultiplyElement, 162
 - operator=, 162

Class Matrix (cntd)

PermuteMe, 162
 PermuteMeSymmetric, 163
 RowVectorTimesMe, 163
 SetElement, 163
 SetSubmatrix, 164
 SetSubmatrixAsIdentity, 164
 VectorCrossMe, 164

Class Matrix3

FindLargest2X2Minor, 172
 GetColumn, 173
 GetComponents, 173
 GetDeterminant, 173
 GetElement, 174
 GetInverse, 174
 GetMyTranspose, 174
 GetNumberCols, 174
 GetNumberElements, 175
 GetNumberRows, 175
 GetReport, 175
 GetRow, 175
 InsertMeIntoMatrix, 176
 MakeMeZero, 176
 Matrix3, 171, 172
 ~Matrix3, 172
 MeCrossVector, 176
 MeEqualsMatrix3TimesMe, 176
 MeEqualsMatrix3TimesMeTransposed, 177
 MeEqualsMatrix3TransposedTimesMe, 177
 MeEqualsMeMinus, 177
 MeEqualsMePlus, 177
 MeEqualsMeTimesMatrix3, 178
 MeEqualsMeTimesMatrix3Transposed, 178
 MeEqualsMeTransposed, 178
 MeEqualsMeTransposedTimesMatrix3, 178
 MeEqualsMinusOther, 178
 MeEqualsOther, 179
 MeInnerProductMatrix3, 179
 MeTimesColumnVector, 179
 MeTimesMatrix, 180

MeTimesMatrix3, 180
 MeTimesMatrix3TimesMeTransposed, 180
 MeTimesMatrix3Transposed, 181
 MeTimesMatrixTransposed, 181
 MeTimesVector, 181, 182
 MeTimesVector3, 182
 MeTransposedInnerProductMatrix3, 183
 MeTransposedTimesDiagonalTimesMatrix3,
 183
 MeTransposedTimesDiagonalTimesMe, 183
 MeTransposedTimesMatrix, 184
 MeTransposedTimesMatrix3, 184
 MeTransposedTimesMatrix3TimesMe, 184
 MeTransposedTimesVector, 185
 MeTransposedTimesVector3, 185
 operator*, 185, 186
 operator+, 186
 operator+=", 186
 operator-, 186, 187
 operator-=", 187
 operator/, 187
 operator=, 187
 ReadFromFile, 187
 ReportOrthogonality, 188
 RowVectorTimesMe, 188
 SaveToFile, 188
 SetComponents, 188
 SetComponentsFromThreeColumnVectors,
 189
 SolveHomogeneousSystem, 189
 Vector3TimesMeTimesVector3, 189
 VectorCrossMe, 190

Class Matrix3X4

GetElement, 193
 GetNumberCols, 194
 GetNumberElements, 194
 GetNumberRows, 194
 Matrix3X4, 193
 ~Matrix3X4, 193
 MeTimesColumnVector, 194

Class Matrix3X4 (cntd)

MeTimesMatrix, 195
 MeTimesMatrixTransposed, 195
 MeTransposedTimesMatrix, 195
 RowVectorTimesMe, 196

Class PArray

AppendArray, 203
 AppendItem, 203
 Copy, 203
 Create, 203
 GetArray, 204
 GetAt, 204
 GetFirst, 204
 GetLast, 204
 GetNonconstantArray, 204
 GetSize, 205
 Insert, 205
 InsertInOrder, 205
 IsEmpty, 206
 Left, 206
 m_pType, 200
 m_Size, 200
 Mid, 206, 207
 operator[], 207
 operator const TYPE *, 202
 PArray, 201, 202
 ~PArray, 202
 Prepend, 207, 208
 RemoveItem, 208
 ReverseMe, 208
 Right, 208
 RotateMe, 209
 SetAt, 209
 SetAtGrow, 209
 SetSize, 210

Class PMatrix

CompareMatrix, 136
 element, 135
 GetCopyOfElements, 136
 GetElement, 136

GetElements, 136
 GetNonconstantElements, 137
 GetNonconstantRow, 137
 GetNumberCols, 137
 GetNumberElements, 137
 GetNumberRows, 137
 GetRow, 138
 HasLargeElements, 138
 InsertDiagonalSubmatrix, 138
 IsZero, 138
 m_nCols, 135
 m_nElements, 135
 m_nRows, 135
 MakeMeZero, 138
 MeEqualsDifferenceBetween, 139
 MeEqualsMeMinus, 139
 MeEqualsMePlus, 139
 MeEqualsMeTransposed, 139
 MeEqualsMinusMe, 139
 MeEqualsMinusMeMinus, 140
 MeEqualsMinusMePlus, 140
 MeEqualsMinusOther, 140
 MeEqualsNegativeSumOf, 140
 MeEqualsOther, 141
 MeEqualsSumOf, 141
 MeInnerProductMatrix, 141
 MeTimesColumnVector, 142
 MeTimesMatrix, 142
 MeTimesMatrixTransposed, 143
 MeTransposedTimesMatrix, 143
 operator*=, 143
 operator+=, 144
 operator-=, 144
 operator/=: 144
 PMatrix, 135
 ~PMatrix, 135
 ReportMatrix, 144
 RowVectorTimesMe, 145

Class PVector

components, 73

Class PVector (cntd)

Create, 75
 DotProduct, 75
 FindLargestComponent, 75
 FindSmallestComponent, 76
 GetComponent, 76
 GetComponents, 76
 GetCopyOfComponents, 76
 GetMyLength, 76
 GetMyLengthSquared, 77
 GetNonconstantComponents, 77
 GetNumberComponents, 77
 IsZero, 77
 m_nComponents, 73
 MakeMeZero, 77
 MeEqualsDifferenceBetween, 77
 MeEqualsMeMinus, 78
 MeEqualsMePlus, 78
 MeEqualsMeTimesConstant, 78
 MeEqualsMeTimesDiagonalMatrix, 78
 MeEqualsMinusMe, 79
 MeEqualsMinusMeMinus, 79
 MeEqualsMinusMePlus, 79
 MeEqualsMinusOther, 79
 MeEqualsNegativeSumOf, 80
 MeEqualsOther, 80
 MeEqualsSumOf, 80
 MeTimesDiagonalMatrix, 80
 operator!=, 81
 operator*=: 81
 operator+=, 81
 operator-=, 81
 operator=, 82
 operator==, 82
 PVector, 74
 ~PVector, 74

Class Str

AtoF, 292
 AtoI, 292
 CLeft, 293

CMid, 293
 CompactSpaces, 294
 Compare, 294, 295
 ConvertMeToNumber, 295
 CountLines, 295
 CountNonemptyLines, 296
 CountNonemptyWords, 296
 CountOccurrences, 296
 Create, 297, 298
 CRight, 298
 Empty, 298
 FillSpaces, 298
 Find, 299, 300
 FindExclusive, 300
 FindMatchingBracket, 301
 FindMatchingBracketExclusive, 301
 FindMeInFile, 302
 FindNextCommand, 302
 FindNextCommandExclusive, 303
 FindNextLine, 303
 FindNextNonalnumDelimitedWord, 304
 FindNextSpaceDelimitedWord, 304
 FindNthCommandWithTitle, 305
 FindNthCommandWithTitleExclusive, 305
 FindNthLineStartingWithWord, 306
 FindPreviousLine, 306
 FindPreviousNonalnumDelimitedWord, 306
 FindPreviousSpaceDelimitedWord, 307
 FindTextBetween, 307, 308
 Format, 308
 GetAt, 308
 GetBSTR, 309
 GetLength, 309
 GetString, 309
 GetTextBetweenBrackets, 309
 GetTextBetweenBracketsExclusive, 310
 Insert, 310
 isAlnum, 310
 isAlpha, 311
 isDigit, 311

Class Str (cntd)

IsEmpty, 311
 isGraph, 311
 isLower, 311
 isPrint, 312
 isPunct, 312
 isSpace, 312
 isUpper, 312
 Justify, 312
 Left, 313
 LeftJustify, 313
 m_pC, 288
 m_Size, 288
 Mid, 313, 314
 operator!=, 314
 operator+, 315, 316
 operator+=, 316, 317
 operator=, 319, 320
 operator==, 320, 321
 operator[, 323
 operator>, 321, 322
 operator>=, 322, 323
 operator<, 317, 318
 operator<=, 318, 319
 operator const char *, 292
 ReadFileIntoMe, 323
 ReadFileUntilMeFound, 324
 ReadMeFromFile, 324
 Remove, 324
 RemoveBetweenMarks, 325
 RemoveComments, 325
 ReverseFind, 325, 326
 Right, 326
 SeparateIntoLines, 326
 SeparateIntoNonemptyLines, 327
 SeparateIntoNonemptyWords, 327
 SetAt, 327
 SetSize, 328
 Str, 288–291
 ~Str, 291

ToLower, 328
 ToUpper, 328
 Trim, 328
 TrimLeft, 328
 TrimRight, 328
 TSQ, 329
 vCat, 329
 Verify, 329
 vLength, 329
 WriteMeToFile, 330
 WriteMyStringToFile, 330

Class UnitVector3

GetVectorAlongMyDirection, 123
 operator=, 123
 RotateMeAroundAxis, 124
 UnitVector3, 122
 ~UnitVector3, 123

Class Vector2

Angle, 88
 Create, 88
 DotVector, 88
 FindLargestComponent, 88
 FindSmallestComponent, 89
 GetComponent, 89
 GetComponents, 89, 90
 GetMyLength, 90
 GetMyLengthSquared, 90
 GetMyUnit, 90, 91
 GetNonconstantComponents, 91
 GetNumberComponents, 91
 IsZero, 91
 MakeMeUnit, 91
 MakeMeZero, 91
 MeEqualsDifferenceBetween, 92
 MeEqualsMeMinus, 92
 MeEqualsMePlus, 92
 MeEqualsMinusMe, 92
 MeEqualsMinusMeMinus, 92
 MeEqualsMinusMePlus, 93
 MeEqualsMinusOther, 93

Class Vector2 (cntd)

MeEqualsNegativeSumOf, 93
 MeEqualsOther, 93
 MeEqualsSumOf, 94
 MeTimesConstant, 94
 operator!=, 94
 operator*, 94
 operator*=, 95
 operator+, 95
 operator+=, 95
 operator-, 95, 96
 operator-=, 96
 operator/, 96
 operator=, 96
 operator==, 97
 operator[], 97
 operator const double *, 88
 ReadFromFile, 97
 ReportVector, 97
 RotateMe, 98
 SaveToFile, 98
 Vector2, 87
 ~Vector2, 87

Class Vector3

Angle, 104
 AngleDirected, 104
 Create, 105
 CrossVector, 105
 DotVector, 106
 DoubleCrossVector, 106
 FindLargestComponent, 106
 FindSmallestComponent, 106
 GenerateArbitraryNormal, 107
 GetComponent, 107
 GetComponents, 107, 108
 GetMyLength, 108
 GetMyLengthSquared, 108
 GetMyPNormal, 108
 GetMyUnit, 109
 GetNonconstantComponents, 109

GetNumberComponents, 109
 GetWMatrix, 109
 IsZero, 109
 MakeMeUnit, 110
 MakeMeZero, 110
 MeEqualsDifferenceBetween, 110
 MeEqualsMeMinus, 110
 MeEqualsMePlus, 110
 MeEqualsMeTimesDiagonalMatrix3, 111
 MeEqualsMinusMe, 111
 MeEqualsMinusMeMinus, 111
 MeEqualsMinusMePlus, 111
 MeEqualsMinusOther, 112
 MeEqualsNegativeSumOf, 112
 MeEqualsOther, 112
 MeEqualsSumOf, 112
 MeTimesConstant, 113
 MeTimesDiagonalMatrix3, 113
 operator!=, 113
 operator*, 113
 operator*=, 114
 operator+, 114
 operator+=, 114
 operator-, 114, 115
 operator-=, 115
 operator/, 115
 operator=, 115
 operator==, 116
 operator[], 116
 operator const double *, 104
 ReadFromFile, 116
 ReportVector, 116
 RotateMeAroundAxis, 117
 SaveToFile, 117
 SetComponents, 117, 118
 Vector3, 103
 ~Vector3, 103

Class VectorN

GetMyLength, 128
 GetMyLengthSquared, 128

Class VectorN (cntd)

- operator*, 128
- operator*=, 128
- operator+, 129
- operator+=, 129
- operator-, 129
- operator=, 130
- operator/, 130
- operator=, 130
- VectorN, 127
- ~VectorN, 127

Clock test, 339**Code**, 2

- design considerations, 1
- efficiency, 4
- encapsulated, 38
- style, 3

Column vector, 26

COM, 309

Commutative law, 10**Compiler**, 30

- directives, 31, 56

Compound statements, 43

- break, 48
- case label, 47
- do - while, 47
- for, 45
- for loop, 45
- if - else, 43
- switch - case - default, 47
- while, 46

Constant

- functions, 52
- keyword, 52
- objects, 52
- pointers, 52

Constructors, 35**Container classes**, 59**Continuation**, 31**Contravariance**, 27**Conventions**, 335**Coordinate**, 337

- axis, 337
- plane, 337
- system, 8, 337
 - right-handed, 337
- transformations, 339

Corkscrew rule, 339**Covariance**, 27**Cross product**, 17**C Standard Library**

- printf, 144

CStr

- Attribute details, 250
- class, 243
- constructor details, 250
- method details, 252

C string, 42**Data encapsulation**, 30**Declarations**, 55

- forward, 58

Definitions, 55, 335**Dereferencing**, 37**Destructors**, 35**Dimension**, 7**Dimensionless**, 7**Direction**, 7**Direction cosines**, 16**Dot product**, 15**Encapsulated code**, 38**Equation**, 63**Equations, algorithms and programs**, 63**Escape sequence**, 41**Executable**, 30**Families**, 66**Families of classes**, 48**Family of Classes**

- Array, 197

- Matrix, 131
- String, 243
- Vector, 69
- File**
 - header, 55
 - implementation, 55
 - include, 55
 - source, 55
- Files**
 - circular inclusions, 57
 - multiple inclusions, 57
- Formatter**
 - Attribute details, 332
 - constructor details, 333
 - method details, 333
- Formatter class, 330**
- Fortran, 38**
- Forward declarations, 58**
- Function**
 - arguments, 38
 - body, 40
 - call, 38
 - constant, 52
 - fully qualified name, 57
 - inline, 59
 - interface, 38
 - method, 39
 - multiple-return, 39
 - name, 38
 - overloaded, 53
 - prototype, 56
 - return, 38
 - return by argument, 39
 - see also Arguments, 38
 - single-return, 39
 - virtual, 50
- Functions, 38**
- Graphic representation of vectors, 8**
- has a, 30
- Header file, 55**
- Implementation file, 55**
- Include file, 55**
- Indirection, 37**
- Inheritance, 30, 31**
 - multiple, 50
- Inline functions, 59**
- Instruction, 64**
- Inverse matrix, 25**
- is a, 30
- Java, 38, 42**
- Kronecker delta, 22**
- Left-handed system, 338**
- Length, 7**
- Line continuation, 31**
- Line splicing, 31**
- Linker, 30**
- Lower triangular matrix, 25**
- lvalue, 35**
- Machine independence, 30**
- Magnitude, 7**
- Matrices, 21**
- Matrix**
 - antisymmetric, 24
 - band, 25
 - basics, 21
 - block, 25
 - class, 145
 - classes, 131
 - column, 21
 - conformable, 23
 - constructor details, 150
 - diagonal, 21
 - family of classes, 131
 - identity, 22
 - inverse, 25
 - line, 21

Matrix (cntd)

- lower triangle, [22](#)
- lower triangular, [25](#)
- method details, [151](#)
- operations, [22](#)
- order, [21](#)
- orthogonal, [25](#)
- partition, [25](#)
- permutation, [25](#)
- rank, [25](#)
- row, [21](#)
- size, [21](#)
- skew-symmetric, [24](#)
- sparse, [25](#)
- square, [21](#)
- submatrix, [25](#)
- symmetric, [24](#)
- transpose, [24](#)
- unit, [22](#)
- upper triangle, [21](#)
- upper triangular, [25](#)
- zero, [22](#)

Matrix3

- class, [165](#)
- constructor details, [171](#)
- method details, [172](#)

Matrix3X4

- class, [190](#)
- constructor details, [193](#)
- method details, [193](#)

Method, [39](#)

- static, [39](#)

Method design, [64](#)**Methods**, [31](#)**Multidimensional vectors**, [19](#)**Multiple inheritance**, [50](#)**Names and types**, [32](#)**Naming style**, [64](#)**Null vector**, [7](#)**Number of dimensions**, [8](#)**Object-oriented language**, [30](#)**Objectives and roadmap**, [1](#)**Object oriented programming**, [1](#)**Objects**, [29](#)

- attributes, [31](#)
- behavior, [30](#), [31](#)
- constant, [52](#)
- creating, [53](#)
- methods, [31](#)
- properties, [30](#)
- type, [31](#)
- using, [53](#)

Operating system independence, [30](#)**Operations with matrices**, [22](#)**Operator**

- address, [37](#)
- dereferencing, [37](#)
- indirection, [37](#)
- scope resolution, [57](#)

Operators, [4](#), [33](#)

- delete, [54](#)
- new, [54](#)
- precedence, [34](#)

Orthogonal matrix, [25](#)**Orthonormal basis**, [12](#)**Overloading functions**, [53](#)**p-normal**, [108](#)**Parallelogram law**, [10](#)**Parameterized classes**, [59](#)

- container classes, [59](#)
- example, [60](#)
- inlining functions, [59](#)

Parametric polymorphism, [59](#)**PArray**

- Attribute details, [200](#)
- class, [197](#)
- constructor details, [201](#)
- method details, [202](#)

- Parsing**, 280
- Partition**, 25
- Permissions**, 40
- Permutation matrix**, 25
- Physical unit**, 7
- PMatrix**
 - Attribute details, 135
 - class, 131
 - constructor details, 135
 - method details, 136
- Pointer**, 36
 - proper type, 36
 - target type, 36
 - to a pointer, 42
 - to function, 40
- Pointer arithmetic**, 42
- Pointers**, 36
- Polymorphism**, 30, 48
 - parametric, 59
- Precompiler**, 31
- Primitive types**, 33
- printf**, 144
- Program**, 63
- Programming in C++**, 29
 - conclusion, 62
 - parameterized classes, 59
- Proper type**, 36
- Prototype**
 - class, 56
 - function, 56
- PVector**
 - Attribute details, 73
 - class, 69
 - constructor details, 74
 - method details, 75
- Reference**, 37
 - proper type, 37
 - target type, 37
- References**, 36
- Reference system**, 337
- Right-handed system**, 338
- Root class**, 50
- Row vector**, 26
- rvalue**, 35
- Scalar**, 7
 - magnitude, 7
 - value, 7
- Scalars and Vectors**, 7
 - basics, 7
- Scope resolution operator**, 57
- Separating declarations and definitions**, 55
- Services**, 30
 - user, 30
- Skew-symmetric matrix**, 24
- Source file**, 55
- Space**, 8
- Sparse matrix**, 25
- Specialized matrices**, 24
- Square matrix**, 21
- Standard C Library**, 38
- Statement**
 - break, 48
 - case, 47
 - compound, 43
 - continue, 48
 - default, 47
 - switch, 47
- Str**
 - Attribute details, 288
 - class, 279
 - constructor details, 288
 - method details, 292
- String**
 - classes, 243
 - family of classes, 243
- Submatrix**, 25
- Substring index**, 244

Symmetric matrix, 24

Target type, 36

Template, 59

Terminology, 335

Theorem, 64

Transpose matrix, 24

Type

primitive, 33

user defined, 33

Unit matrix, 22

Unit vector, 11

UnitVector3

class, 118

constructor details, 122

method details, 123

Upper triangular matrix, 25

Value, 7

Vector, 7

addition, 10

application point, 8

as a matrix, 26

classes, 69

column, 26

commutative law, 10

components, 12

coordinate system, 8

cross product, 17

direction, 7

dot product, 15

family of classes, 69

graphic representation, 8

head, 8

left-handed system, 338

length, 7

magnitude, 7

multidimensional, 19

null, 7

operations, 9

origin, 8

parallelogram law, 10

point, 8

projection, 12

reference system, 8

right-handed system, 338

row, 26

scalar product, 15

tail, 8

unit, 11

vector product, 17

zero, 7

Vector2

class, 82

constructor details, 87

method details, 88

Vector3

class, 98

constructor details, 103

method details, 104

Vector and matrix code, 63

VectorN

class, 124

constructor details, 127

method details, 128

Vectors as matrices, 26

Virtual functions, 50

Visibility, 40

Writing classes, 31

Zero vector, 7