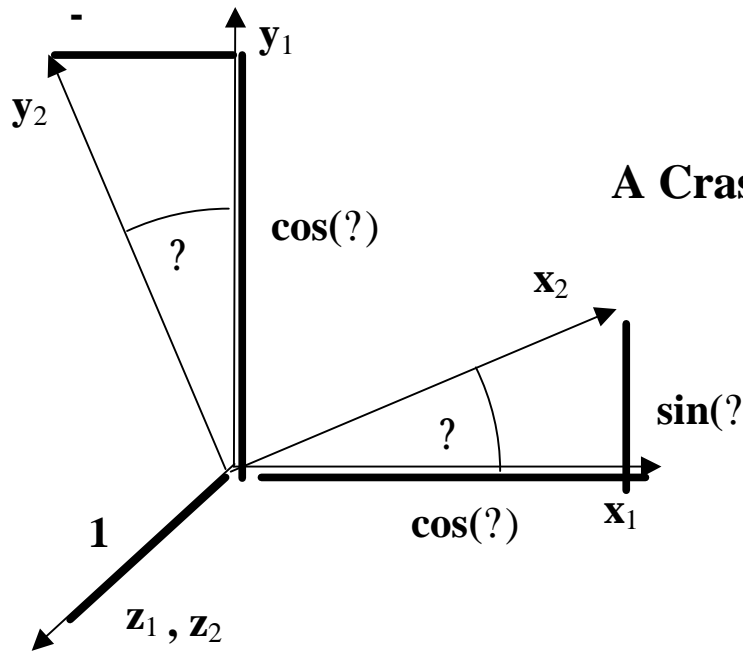


Computer Integrated Surgery Lab
NSF Engineering Research Center for Computer Integrated
Surgical Systems and Technology

The Johns Hopkins University; Baltimore, Maryland, USA
Massachusetts Institute of Technology; Cambridge, Mass.
Brigham & Women's Hospital; Boston, Mass.
Carnegie-Mellon University; Pittsburgh, Pennsylvania
Shadyside Hospital; Pittsburgh, Pennsylvania



A Crash Course for the cisVecs Library

A Crash Course for the cisVecs Library

Ver 1.2

Document CIS0.1

Date 9/4/01

Dr. Russell H Taylor

Director, Center for Computer-Integrated Surgical Systems and Technology

Room 315, New Engineering Building

3400, North Charles Street

Baltimore MD 21218

USA

Web: <http://cisstweb.cs.jhu.edu/resources/software>

Phone: +01 (410) 516 0740

Fax: +01 (410) 516 5553

Email: anton@cs.jhu.edu

Developmental Software—not certified for medical use. The CISST ERC makes no warranty that the software or other information described in this document is fit for any particular purpose. Users assume full responsibility for any consequences arising from its use.

© 1998, 2001 CISST ERC

This document contains proprietary and confidential information of the Engineering Research Center for Computer-Integrated Surgical Systems and Technology (CISST ERC). The contents of this document may not be disclosed to third parties, translated, copied, or duplicated in any form without the express written permission of the CISST ERC Director or other duly authorized representative of the CISST ERC. Queries should be directed to the CISST ERC Director.

1 Introduction

A Crash Course for the cisVecs Library:

This document is intended to be a short reference for the cisVecs library. It is not by any means a comprehensive guide to every nuance of the library's functionality.

The cisVecs library was designed to allow easy and efficient representations of fixed-length vectors and homogeneous frame matrices, as well as performing basic operations on the objects, such as cross product, dot product, scaling, addition, subtraction, comparisons, and linear transformations. The library is for use with C++ programs, and is used by simply including the file cisVecs.h.

The Data types used in the cisVecs library are as follows: cisVec2, cisVec3, cisVec4, cisVec3H, cisIntVec2, cisIntVec3, cisIntVec4, cisRotation, cisFrame, cisMat4x4, cisQuat, and cisUnitQuat.

cisVec2, cisVec3, and cisVec4 represent 2-, 3-, and 4- dimensional vectors, respectively, and store their Data as cisScalar values (floating point, double by default). cisVec3H represents a 3-dimensional vector with a fourth cisScalar value that represents the scaling of the vector. cisIntVec2, cisIntVec3, and cisIntVec4 are the same as cisVec2, cisVec3, and cisVec4, except that they store their Data as integer values (cisInt). These are used mostly in image processing applications.

The cisRotation class is used to store and perform operations on rotation matrices. The cisRotation class stores three cisVec3 objects.

The cisFrame class, consisting of a cisRotation object and a cisVec3 translation vector, is used to represent coordinate frames and to allow linear transformations to those frames.

The cisMat4x4 class is a storage class for general 4x4 matrices. Note however that while both Frames and Rotations may be constructed from the cisMat4x4 class, the class does no self-checking to insure that it's Data is properly formatted and represents a homogeneous Matrix.

A cisQuat is a 4x1 Matrix representing a quaternion. A cisUnitQuat is simply a cisQuat with a unit value.

The classes support a wide variety of functions, including allocation, overlay on top of other structures, subvectors and submatrices, etc. The matrices are stored in column order, consistent with Fortran conventions.

2 Class/Function Lookup Table

Global Functions	
<code>cisInner (const Type1& a, const Type2& b);</code>	Takes the inner product between Type1 and Type2.
<code>cisInvInner (const Type1& a, const Type2& b);</code>	Takes the inverse inner product between Type1 and Type2 where $\text{cisInvInner}(a, b) = \text{Inverse}(a) \cdot b$.
<code>cisInnerInPlace (const Type1& a, Type2& b);</code>	Takes the inner product between Type1 and Type2 and assigns the result to b.
<code>cisInvInnerInPlace (const Type1& a, Type2& b);</code>	Takes the inverse inner product between Type1 and Type2 where $\text{cisInvInner}(a, b) = \text{Inverse}(a) \cdot b$ and assigns the result to b.

Vector Classes	Description
Constructors-cisVec2	
<code>cisVec2 (cisScalar a, cisScalar b);</code>	Constructs a cisVec2 from two cisScalar components.
<code>cisVec2 (const cisVec2& V2);</code>	Constructs a cisVec2 from another cisVec2.
<code>cisVec2 (const cisIntVec2& V2);</code>	Constructs a cisVec2 from a cisIntVec2.
<code>cisVec2 (const cisVec3& V2);</code>	Constructs a cisVec2 from a cisVec3.
<code>cisVec2 (cisScalar* a);</code>	Constructs a cisVec2 from a Pointer of 2 Scalars.
<code>cisVec2 (cisScalar a = 0);</code>	Constructs a cisVec2 with x and y set to a.
Constructors-cisVec3	
<code>cisVec3 (cisScalar a, cisScalar b, cisScalar c);</code>	Constructs a cisVec3 from three cisScalar components.
<code>cisVec3 (const cisVec3& V2);</code>	Constructs a cisVec3 from another cisVec3.
<code>cisVec3 (const cisVec2& V2, cisScalar newZ = 0);</code>	Constructs a cisVec3 from a cisVec2 and a cisScalar.
<code>cisVec3 (const cisVec3H& V2);</code>	Constructs a cisVec3 from a cisVec3H.
<code>cisVec3 (const cisVec4& V2);</code>	Constructs a cisVec3 from a cisVec4.
<code>cisVec3 (cisScalar* a);</code>	Constructs a cisVec3 from a Pointer of 3 Scalars.
<code>cisVec3 (cisScalar a = 0);</code>	Constructs a cisVec3 with x, y and z set to a.
Constructors-cisVec4	
<code>cisVec4 (cisScalar a, cisScalar b, cisScalar c, cisScalar d);</code>	Constructs a cisVec4 from four cisScalar components.
<code>cisVec4 (const cisVec4& V2);</code>	Constructs a cisVec4 from another cisVec4.
<code>cisVec4 (const cisIntVec4& V2);</code>	Constructs a cisVec4 from a cisIntVec4.
<code>cisVec4 (const cisVec3& v, cisScalar newW = 0);</code>	Constructs a cisVec4 from a cisVec3 and a cisScalar.
<code>cisVec4 (const cisVec3H& v);</code>	Constructs a cisVec4 from a cisVec3H.
<code>cisVec4 (cisScalar* a);</code>	Constructs a cisVec4 from a Pointer of 4 Scalars.
<code>cisVec4 (cisScalar a = 0);</code>	Constructs a cisVec4 with x, y, z and w set to a.
Constructors-cisVec3H	
<code>cisVec3H (cisScalar a, cisScalar b, cisScalar c, cisScalar d);</code>	Constructs a cisVec3H from four cisScalar components.
<code>cisVec3H (const cisVec4& V2);</code>	Constructs a cisVec3H from a cisVec4.
<code>cisVec3H (const cisVec3H& V2);</code>	Constructs a cisVec3H from another cisVec3H.
<code>cisVec3H (const cisVec3& v);</code>	Constructs a cisVec3H from a cisVec3.
<code>cisVec3H (const cisVec3& v, cisScalar ww);</code>	Constructs a cisVec3H from a cisVec3 and a cisScalar.
<code>cisVec3H (cisScalar* a);</code>	Constructs a cisVec3H from a Pointer of 4 Scalars.

<code>cisVec3H (cisScalar a = 0);</code>	Constructs a <code>cisVec3H</code> with <code>x</code> , <code>y</code> , <code>z</code> and <code>w</code> set to <code>a</code> .
Constructors-cisIntVec2	
<code>cisIntVec2 (int a, int b);</code>	Constructs a <code>cisIntVec2</code> from two <code>int</code> components.
<code>cisIntVec2 (const cisIntVec2& V2);</code>	Constructs a <code>cisIntVec2</code> from another <code>cisIntVec2</code> .
<code>cisIntVec2 (const cisIntVec3& V2);</code>	Constructs a <code>cisIntVec2</code> from a <code>cisIntVec3</code> .
<code>cisIntVec2 (const cisVec2& V2);</code>	Constructs a <code>cisIntVec2</code> from a <code>cisVec2</code> .
<code>cisIntVec2 (int* a);</code>	Constructs a <code>cisIntVec2</code> from a Pointer of 2 ints.
<code>cisIntVec2 (int a = 0);</code>	Constructs a <code>cisIntVec2</code> with <code>x</code> and <code>y</code> set to <code>a</code> .
Constructors-cisIntVec3	
<code>cisIntVec3 (int a, int b, int c);</code>	Constructs a <code>cisIntVec3</code> from three <code>int</code> components.
<code>cisIntVec3 (const cisIntVec3& V2);</code>	Constructs a <code>cisIntVec3</code> from another <code>cisIntVec3</code> .
<code>cisIntVec3 (const cisIntVec2& V2, int newZ = 0);</code>	Constructs a <code>cisIntVec3</code> from a <code>cisIntVec2</code> and an <code>int</code> .
<code>cisIntVec3 (const cisIntVec4& V2);</code>	Constructs a <code>cisIntVec3</code> from a <code>cisIntVec4</code> .
<code>cisIntVec3 (const cisVec3& V2);</code>	Constructs a <code>cisIntVec3</code> from a <code>cisVec3</code> .
<code>cisIntVec3 (int* a);</code>	Constructs a <code>cisIntVec3</code> from a Pointer of 3 ints.
<code>cisIntVec3 (int a = 0);</code>	Constructs a <code>cisIntVec3</code> with <code>x</code> , <code>y</code> , and <code>z</code> set to <code>a</code> .
Constructors-cisIntVec4	
<code>cisIntVec4 (int a, int b, int c, int d);</code>	Constructs a <code>cisIntVec4</code> from four <code>int</code> components.
<code>cisIntVec4 (const cisIntVec4& V2);</code>	Constructs a <code>cisIntVec4</code> from another <code>cisIntVec4</code> .
<code>cisIntVec4 (const cisVec3H& v);</code>	Constructs a <code>cisIntVec4</code> from a <code>cisVec3H</code> .
<code>cisIntVec4 (const cisIntVec3& v, int ww = 0);</code>	Constructs a <code>cisIntVec4</code> from a <code>cisIntVec3</code> and an <code>int</code> .
<code>cisIntVec4 (int* a);</code>	Constructs a <code>cisIntVec4</code> from a Pointer of 4 ints.
<code>cisIntVec4 (int a = 0);</code>	Constructs a <code>cisIntVec4</code> with <code>x</code> , <code>y</code> , <code>z</code> , and <code>w</code> set to <code>a</code> .
Overloaded Operators	
<code>-</code>	Unary minus, element-wise subtraction, or scalar subtraction.
<code>+</code>	Element-wise addition or scalar addition.
<code>/</code>	scalar division.
<code>*</code>	<code>cisInner</code> product or scalar multiplication.
<code>% [cisVec2, cisVec3, cisVec3H, cisIntVec2, cisIntVec3]</code>	Cross product.
<code>--</code>	Element-wise subtraction with update or scalar subtraction with update.
<code>+=</code>	Element-wise addition with update or scalar addition with update.
<code>/= [cisVec2, cisVec3, cisVec4, cisVec3H, cisIntVec3, cisIntVec4]</code>	scalar division with update.
<code>*= [cisVec2, cisVec3, cisVec4, cisVec3H, cisIntVec3, cisIntVec4]</code>	scalar multiplication with update.
<code>()</code>	Indexing operator to retrieve the i^{th} element.
<code>=</code>	Element-wise assignment, assignment from a Pointer, or assignment from Scalars/ints.
<code>==</code>	Element-wise comparison or scalar comparison.
<code>!=</code>	Element-wise comparison or scalar comparison.
<code>></code>	Each component greater than supplied argument.
<code><</code>	Each component less than supplied argument.
<code>>=</code>	Each component greater than or equal to supplied argument.
<code><=</code>	Each component less than or equal to supplied argument.
Functions	
<code>Set</code>	Sets the Data in the vector object to the supplied

Pointer	arguments.
Length [cisVec2, cisVec3, cisIntVec2, cisIntVec3]	C/C++-Pointer access to the components of the vector.
Perp [cisVec2, cisVec3, cisIntVec2, cisIntVec3]	Returns the length of the vector.
Normalize [cisVec2, cisVec3, cisIntVec3]	Returns a vector perpendicular to the vector.
ScaleMult [cisVec2, cisVec3, cisIntVec3]	Returns the normalized vector.
ScaleDiv [cisVec2, cisVec3, cisIntVec3]	Element-wise multiplication.
ScaleMultToSelf [cisVec2, cisVec3, cisIntVec3]	Element-wise division.
ScaleDivToSelf [cisVec2, cisVec3]	Element-wise multiplication with update.
XYZ [cisVec3H, cisVec4]	Element-wise division with update.
NormW [cisVec3H]	Returns the x, y, and z components in a cisVec3 object.
	Divides all components by the w component.

cisRotation Class	Description
Constructors	
<code>cisRotation();</code>	Constructs a unit rotation.
<code>cisRotation(const cisVec3& rx, const cisVec3& ry, const cisVec3& rz);</code>	Constructs a cisRotation from three cisVec3 objects.
<code>cisRotation(const cisVec3* new_cols);</code>	Assembles a cisRotation from 3 mutually orthogonal column vectors.
<code>cisRotation(const cisRotation& R2);</code>	Constructs a cisRotation from another cisRotation.
<code>cisRotation(const cisScalar angle, const cisVec3 axis);</code>	Constructs a cisRotation from a cisScalar angle and a cisVec3 axis.
<code>cisRotation(cisScalar angle, cisAxisIndex axis);</code>	Constructs a cisRotation from a cisScalar angle about a canonical axis.
<code>cisRotation(cisScalar a1, cisScalar a2, cisScalar a3, cisRotationOrder order);</code>	Returns cisRotation (a1, AXIS1)*cisRotation (a2, AXIS2)*cisRotation (a3, AXIS3) where order determines which canonical axes are AXIS1, AXIS2, AXIS3.
<code>cisRotation(const cisVec3& angles, cisRotationOrder order);</code>	Same as Above, except angles is a cisVec3 object of [a1, a2, a3].
<code>cisRotation(const cisScalar* a);</code>	
<code>cisRotation(const cisScalar** a);</code>	
<code>cisRotation(cisScalar Rxx, cisScalar Ryy, cisScalar Rzz, cisScalar Rxy, cisScalar Ryz, cisScalar Rzx, cisScalar Rzy, cisScalar Rxz);</code>	Constructs a cisRotation from the 9 individual cisScalar components.
Overloaded Operators	
<code>()</code>	Indexing operator. Returns the i^{th} or i^{th} and j^{th} element in the Matrix.
<code>=</code>	Assignment operator.
<code>~</code>	Transpose operator (also inverse).
<code>==</code>	Compares one cisRotation to another.
<code>!=</code>	Compares one cisRotation to another.
<code>*</code>	Returns one of the following: The inner product of a cisRotation and one of the following: a cisVec3, cisVec3H, another cisRotation, each column of a cisMat4x4, or a cisFrame.
Functions	

Set	Assembles a rotation Matrix from cisScalar elements.
Inverse	Transpose function (also inverse).
SetFromAngleAndAxis	Sets the cisRotation Matrix from a specified angle and axis.
ExtractAngleAndAxis	Inverse of Above.
SetFromAngles	Sets the cisRotation from angles and the order in which those rotations should be applied.
ExtractAngles	Returns the angles and the order that represent the cisRotation object.
Angles	Accepts an order and returns a cisVec3 of the angles that represent the rotation object.
Spin	Rotates the rotation object by a specified angle about a canonical axis.
Pointer	C/C++-Pointer access to the components of the cisRotation.

cisFrame Class

Constructors

<code>cisFrame();</code>	Default constructor --makes a Unit cisFrame.
<code>cisFrame(const cisRotation& RR, const cisVec3& PP);</code>	Constructs a cisFrame from a cisRotation object and a cisVec3 displacement.
<code>cisFrame(const cisRotation& RR);</code>	Constructs a cisFrame from a cisRotation object and a unit cisVec3 displacement.
<code>cisFrame(const cisVec3& PP);</code>	Constructs a cisFrame from a unit cisRotation and a Displacement.
<code>cisFrame(const cisFrame& him);</code>	Copy constructor. Makes a copy of existing cisFrame "him".
<code>cisFrame(const cisMat4x4& M);</code>	Constructs a cisFrame from a homogeneous cisMat4x4. Does only limited renormalizing.

Overloaded Operators

<code>=</code>	Assignment operator.
<code>==</code>	Compares one cisFrame to another.
<code>!=</code>	Compares one cisFrame to another.
<code>()</code>	Returns the i^{th} and j^{th} element of the cisFrame.
<code>~</code>	Transpose function (also inverse)
<code>*</code>	Returns the inner product of a cisFrame and one of the following: a cisVec3, a cisVec3H, another cisFrame, or a cisMat4x4.

Functions

Set	Assembles a cisFrame Matrix from scalar elements.
Inverse	Inverse function.
Pointer	C/C++-Pointer access to the components of the cisFrame.
Matrix	A two-dimensional C/C++-Pointer access to the components of the cisFrame.

cisMat4x4 Class

Description

Constructors

<code>cisMat4x4();</code>	Default constructor, constructs a cisMat4x4 with a unit rotation and a zero vector.
<code>cisMat4x4(cisScalar x1, cisScalar y1, cisScalar z1, cisScalar w1, cisScalar x2, cisScalar y2, cisScalar z2, cisScalar w2, cisScalar x3, cisScalar y3,</code>	Constructs a cisMat4x4 from 16 individual cisScalar components.

<code>cisScalar z3, cisScalar w3, cisScalar x4, cisScalar y4, cisScalar z4, cisScalar w4); cisMat4x4(const cisVec3H& col1, const cisVec3H& col2, const cisVec3H& col3, const cisVec3H& col4); cisMat4x4(const cisVec4& col1, const cisVec4& col2, const cisVec4& col3, const cisVec4& col4);</code>	Constructs a cisMat4x4 from 4 cisVec3H components.
<code>cisMat4x4(cisVec3H* new_cols);</code>	Constructs a cisMat4x4 from an Pointer of four cisVec3H objects.
<code>cisMat4x4(cisVec4* new_cols);</code>	Constructs a cisMat4x4 from an Pointer of four cisVec4 objects.
<code>cisMat4x4(const cisMat4x4& M2);</code>	Constructs a cisMat4x4 from another cisMat4x4.
<code>cisMat4x4(const cisFrame& F);</code>	Constructs a cisMat4x4 from a cisFrame object.
<code>cisMat4x4(const cisRotation& R);</code>	Constructs a cisMat4x4 from a cisRotation object.
<code>cisMat4x4(const cisRotation& R, const cisVec3& P);</code>	Constructs a cisMat4x4 from a cisRotation object and a cisVec3 object.
<code>cisMat4x4(const cisVec3& P);</code>	Construction from a unit cisRotation and a cisVec3 object.
<code>cisMat4x4(const cisScalar** A);</code>	Constructs a cisMat4x4 from a 4x4 Pointer.
<code>cisMat4x4(const cisScalar* A);</code>	Constructs a cisMat4x4 from a 1x16 Pointer.

Overloaded Operators

<code>=</code>	Compares one cisMat4x4 to another.
<code>()</code>	Compares one cisMat4x4 to another. Returns the inner product of a cisMat4x4 and one of the following: a cisVec3, a cisVec3H, or a cisMat4x4.
<code>*</code>	

Functions

Set	Assembles a cisMat4x4 from cisScalar components. Used internally.
Pointer	C/C++-Pointer access to the components of the cisMat4x4.

cisQuat Classes

Constructors-cisQuat

<code>cisQuat(cisScalar a = 0, cisScalar b = 0, cisScalar c = 0, cisScalar d = 0);</code>	Constructs a cisQuat from four cisScalar components.
<code>cisQuat(cisScalar a, const cisVec3& b);</code>	Constructs a cisQuat from a cisScalar and a cisVec3 object.
<code>cisQuat(const cisVec3& a);</code>	Constructs a cisQuat from a cisVec3 object.
<code>cisQuat(const cisVec4& a);</code>	Constructs a cisQuat from a cisVec4 object.
<code>cisQuat(const cisQuat& a);</code>	Constructs a cisQuat from another cisQuat.
<code>cisQuat(const cisUnitQuat& a);</code>	Constructs a cisQuat from a cisUnitQuat.
<code>cisQuat(cisScalar *a);</code>	Constructs a cisQuat from a Pointer of 16 objects.

Constructors-cisUnitQuat

<code>cisUnitQuat(cisScalar a = 0, cisScalar b = 0, cisScalar c = 0, cisScalar d = 0);</code>	Constructs a cisUnitQuat from four cisScalar components.
<code>cisUnitQuat(cisScalar a, const cisVec3& b);</code>	Constructs a cisUnitQuat from a cisScalar and a cisVec3 object.
<code>cisUnitQuat(const cisVec3& a);</code>	Constructs a cisUnitQuat from a cisVec3 object.
<code>cisUnitQuat(const cisVec4& a);</code>	Constructs a cisUnitQuat from a cisVec4 object.
<code>cisUnitQuat(const cisUnitQuat& a);</code>	Constructs a cisUnitQuat from another cisUnitQuat.

<code>cisUnitQuat(const cisQuat& a);</code>	Constructs a <code>cisUnitQuat</code> from a <code>cisQuat</code> .
<code>cisUnitQuat(cisScalar *a);</code>	Constructs a <code>cisUnitQuat</code> from a Pointer of 16 objects.
<code>cisUnitQuat(const cisRotation& r);</code>	Constructs a <code>cisUnitQuat</code> from a <code>cisRotation</code> object.

Overloaded Operators

<code>-</code>	Unary minus, element-wise subtraction, or <code>cisScalar</code> subtraction.
<code>+</code>	Element-wise addition or <code>cisScalar</code> addition.
<code>/</code>	<code>cisScalar</code> division.
<code>*</code>	<code>cisInner</code> product or <code>cisScalar</code> multiplication.
<code>% [cisQuat]</code>	Cross product.
<code>--</code>	Element-wise subtraction with update or <code>cisScalar</code> subtraction with update.
<code>+=</code>	Element-wise addition with update or <code>cisScalar</code> addition with update.
<code>/=</code>	<code>cisScalar</code> division with update.
<code>*=</code>	<code>cisScalar</code> multiplication with update.
<code>()</code>	Indexing operator to retrieve the i^{th} element.
<code>[]</code>	Indexing operator to retrieve the i^{th} element.
<code>=</code>	Element-wise assignment, assignment from a Pointer, or assignment from Scalars/ints.
<code>==</code>	Element-wise comparison or <code>cisScalar</code> comparison.
<code>!=</code>	Element-wise comparison or <code>cisScalar</code> comparison.
<code>></code>	Each component greater than supplied argument.
<code><</code>	Each component less than supplied argument.
<code>>=</code>	Each component greater than or equal to supplied argument.
<code><=</code>	Each component less than or equal to supplied argument.

Functions

<code>Set</code>	Assembles a <code>cisQuat</code> from <code>cisScalar</code> components.
<code>Normalize</code>	Returns the normalized <code>cisQuat</code> .
<code>Length</code>	Returns the length of the <code>cisQuat</code> .
<code>Conjugate</code>	Returns the Conjugate of a <code>cisQuat</code> .
<code>ApplyNorm [cisUnitQuat]</code>	Enforces the unit norm invariant for a <code>cisUnitQuat</code> .
<code>Inverse</code>	The Inverse function.
<code>Mult</code>	Element-wise multiplication.
<code>Div</code>	Element-wise division.
<code>MultToSelf</code>	Element-wise multiplication with update.
<code>DivToSelf</code>	Element-wise division with update.
<code>ExtractRotation [cisUnitQuat]</code>	Extracts the <code>cisRotation</code> from a <code>cisQuat</code> .
<code>Pointer [unimplemented]</code>	C/C++-Pointer access to the components of the <code>cisQuat</code> and <code>cisUnitQuat</code> .

3 Coding Examples

```
#include <cisVecs.h>

int main(void)
{
    cisVec3 v1(1); // [1,1,1]
    cisVec3 origin(cisZeroVec3); // [0,0,0]
    cisVec3 v2(1,0,0);

    double radians45 = cisDEG_2_RAD(45);
    double degreesPI = cisRAD_2_DEG(cisPI);

    // Construct a rotation from Euler Angles -- takes radians
    cisRotation
R1(cisDEG_2_RAD(30),cisDEG_2_RAD(30),cisDEG_2_RAD(40),cisX_Y_Z);

    // X_Y_Z is a rotation order

    cisAxisIndex RevXYZ = REVERSE_AXIS_ORDER(cisX_Y_Z); // == Z_Y_X

    R1.Set(cisDEG_2_RAD(30),cisDEG_2_RAD(30),cisDEG_2_RAD(40),cisX_Y_Z);

    // Construct a rotation from and angle and an axis
    cisRotation R2(cisDEG_2_RAD(45),v2); // rotation of 45deg around vector
v2

    // Construct a Frame from a Rotation and a Vec3
    cisFrame F1(R1,v1);

    cisRotation RI; // defaults to I
    RI = cisUnitRot; // does not change the value

    cisFrame F2(v1); // uses I for Rotation
    cisFrame F3(R1); // uses ZeroVec3 for translation

    cisVec3 angles;
    R1.ExtractAngles(angles,cisX_Y_Z); // angles will get the Euler angles
    R1.ExtractAngles(angles,cisZ_Y_X);
    // angles will get the Euler angles
    // _but_ angles.x will be the first angle (i.e. around the Z axis )
    //         angles.y will get the second angle
    //         angles.z will get the third angle (around the X axis)

    cisVec3 vTemp;
    cisFrame fTemp; // defaults to [I,0] == UnitFrame
    cisFrame fTemp2,fTemp3;

    fTemp = F1.Inverse(); // fTemp will be the inverse of F1
    fTemp2 = F1 * fTemp; // fTemp2 will be the product of F1 and fTemp
                        // ~= UnitFrame

    fTemp3 = R1 * fTemp;

    cout << v1;
}
```


4 Contact Information

Document Maintainer Anton Deguet (anton@cs.jhu.edu)

Software Maintainer Anton Deguet (anton@cs.jhu.edu)

Repository CIS Lab, JHU

Authors Kathryn Hayes (hayes@jhu.edu) and Andrew Bzostek (bzostek@cs.jhu.edu)

Acknowledgments Rajesh Kumar (rajesh@cs.jhu.edu), Jianhua Yao (jhyao@cs.jhu.edu) and Randy Goldberg (pegasus@jhu.edu) .