CGLA Reference

J. Andreas Bærentzen

June 20, 2003

Contents

1 Introduction

CGLA is a set of numerical C++ vector and matrix classes and class templates designed with computer graphics in mind. CGLA stands for "Computer Graphics Linear Algebra".

Let us get right down to the obvious question: Why create another linear algebra package? Well, CGLA evolved from a few matrix and vector classes because I didn't have anything better. Also, I created CGLA to experiment with some template programming techniques. This led to the most important feature of CGLA, namely the fact that all vector types are derived from the same template.

This makes it easy to ensure identical semantics: Since all vectors have inherited, say, the * operator from a common ancestor, it works the same for all of them.

It is important to note that CGLA was designed for Computer Graphics (not numerical computations) and this had a number of implications. Since, in computer graphics we mainly need small vectors of dimension 2,3, or 4 CGLA was designed for vectors of low dimensionality. Moreover, the amount of memory allocated for a vector is decided by its type at compile time. CGLA does not use dynamic memory. CGLA also does not use virtual functions, and most functions are inline. These features all help making CGLA relatively fast.

Of course, other libraries of vector templates for computer graphics exist, but to my knowledge none where the fundamental templates are parametrized w.r.t. dimension as well as type. In other words, we have a template (ArithVec)that gets both type (e.g. float) and dimension (e.g. 3) as arguments. the intended use of this template is as ancestor of concrete types such as Vec3f - a 3D floating point type.

The use of just one template as basis is very important, I believe, since it makes it extremely simple to add new types of vectors. Another very generic template is ArithMat which is a template for matrix classes. (and not necessarily NxN matrices).

From a users perspective CGLA contains a number of vector and matrix classes, a quaternion and some utility classes. In summary, the most important features are

- A number of 2, 3 and 4 d vector classes.
- A number of Matrix classes.
- A Quaternion class.
- Some test programs.
- Works well with OpenGL.

1.1 Naming conventions

Vectors in CGLA are named VecDT where D stands for dimension at T for type. For instance a 3D floating point vector is named Vec3f. Other types are d (double), s (short), i (int), uc (unsigned char), and usi (unsigned shourt int).

Matrices are similiarly named MatDxDT. For instance a 4D double matrix is called Mat4x4d.

1.2 How to use CGLA

If you need a given CGLA class you can find the header file that contains it in this document. Simply include the header file and use the class. Remember also that all CGLA functions and classes live in the CGLA namespace! Lastly, look at the example programs that came with the code.

An important point is that you should never use the Arith... classes directly. Classes whose names begin with Arith are templates used for deriving concrete types. It is simpler, cleaner, and the intended thing to do to only use the derived types.

In some cases, you may find that you need a vector or matrix class that I haven't defined. If so, it is fortunate that CGLA is easy to extend. Just look at, say, Vec4f if you need a Vec5d class.

1.3 How to use this document

This document is mostly autogenerated from Doxygen tags in the source code. While this is the only way of creating documentation that stands a reasonable chance of being updated every time the code is, the method does have some drawbacks.

If you want to know whether a given class contains a given function, first look at the class. If you don't find the function, look at its ancestors. For instance, the class Vec3f certainly has a += operator function but it is defined in the template class ArithVec.

Another problem is that since templates are used extensively in CGLA, template syntax clutters this document. Unfortunately, that cannot be helped.

1.4 Help, bugs, contributions

CGLA was written (mostly) by Andreas Bærentzen (jab@imm.dtu.dk), and any bug fixes, contributions, or questions should be addressed to me.

2 CGLA Class Documentation

2.1 CGLA::ArithMat< VVT, HVT, MT, ROWS > Class Template Reference

Public Types

- typedef HVT::ScalarType **ScalarType**

  The type of a matrix element.

Public Methods

- **ArithMat** ()

  Construct 0 matrix.

- **ArithMat** (**ScalarType** x)

  Construct a matrix where all entries are the same.

- **ArithMat** (HVT _a)

Construct a matrix where all rows are the same.

- **ArithMat** (HVT _a, HVT _b)

  Construct a matrix with two rows.

- **ArithMat** (HVT _a, HVT _b, HVT _c)

  Construct a matrix with three rows.

- **ArithMat** (HVT _a, HVT _b, HVT _c, HVT _d)

  Construct a matrix with four rows.

- const **ScalarType** * **get** () const
- **ScalarType** * **get** ()
- void **set** (const **ScalarType** *sa)
- **ArithMat** (const **ScalarType** *sa)

  Construct a matrix from an array of scalar values.

- void **set** (HVT _a, HVT _b)

  Assign the rows of a 2D matrix.

- void **set** (HVT _a, HVT _b, HVT _c)

  Assign the rows of a 3D matrix.

- void **set** (HVT _a, HVT _b, HVT _c, HVT _d)

  Assign the rows of a 4D matrix.

- const HVT & **operator[ ]** (int i) const

  Const index operator. Returns i'th row of matrix.

- HVT & **operator[ ]** (int i)

  Non-const index operator. Returns i'th row of matrix.

- bool **operator==** (const MT &v) const

  Equality operator.

- bool **operator!=** (const MT &v) const

  Inequality operator.

- const MT **operator** * (**ScalarType** k) const

  Multiply scalar onto matrix. All entries are multiplied by scalar.

- const MT **operator/** (**ScalarType** k) const

  Divide all entries in matrix by scalar.

- void **operator** *= (**ScalarType** k)

  Assignment multiplication of matrix by scalar.

- void **operator/=** (**ScalarType** k)

  Assignment division of matrix by scalar.

- const MT **operator+** (const MT &m1) const

  Add two matrices.

- const MT **operator-** (const MT &m1) const

Subtract two matrices.

- void **operator+=** (const MT &v)
  Assigment addition of matrices.
- void **operator-=** (const MT &v)
  Assigment subtraction of matrices.
- const MT **operator-** () const
  Negate matrix.

Static Public Methods

- int **get_v_dim** ()
  Get vertical dimension of matrix.
- int **get_h_dim** ()
  Get horizontal dimension of matrix.

2.1.1 Detailed Description

template<class VVT, class HVT, class MT, int ROWS> class CGLA::ArithMat< VVT, HVT, MT, ROWS >

Basic class template for matrices.

In this template a matrix is defined as an array of vectors. This may not in all cases be the most efficient but it has the advantage that it is possible to use the double subscripting notation:

T x = m[i][j]

This template should be used through inheritance just like the vector template

2.1.2 Member Function Documentation

2.1.2.1 template<class VVT, class HVT, class MT, int ROWS> ScalarType* CGLA::ArithMat< VVT, HVT, MT, ROWS >::get () [inline]

Get pointer to data array. This function may be useful when interfacing with some other API such as OpenGL (TM).

2.1.2.2 template<class VVT, class HVT, class MT, int ROWS> const ScalarType* CGLA::ArithMat< VVT, HVT, MT, ROWS >::get () const [inline]

Get const pointer to data array. This function may be useful when interfacing with some other API such as OpenGL (TM).

2.1.2.3 template<class VVT, class HVT, class MT, int ROWS> void CGLA::ArithMat< VVT, HVT, MT, ROWS >::set (const ScalarType * *sa*) [inline]

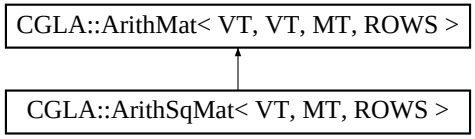
Set values by passing an array to the matrix. The values should be ordered like [[row][row]...[row]]

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

- ArithMat.h

2.2 CGLA::ArithSqMat< VT, MT, ROWS > Class Template Reference

Inheritance diagram for CGLA::ArithSqMat< VT, MT, ROWS >::



Public Types

- typedef VT::ScalarType **ScalarType**

  The type of a matrix element.

Public Methods

- **ArithSqMat** ()

  Construct 0 matrix.

- **ArithSqMat** (**ScalarType** _a)

  Construct matrix where all values are equal to constructor argument.

- **ArithSqMat** (VT _a, VT _b)

  Construct 2x2 Matrix from two vectors.

- **ArithSqMat** (VT _a, VT _b, VT _c)

  Construct 3x3 Matrix from three vectors.

- **ArithSqMat** (VT _a, VT _b, VT _c, VT _d)

  Construct 4x4 Matrix from four vectors.

- **ArithSqMat** (const **ScalarType** *sa)

  Construct matrix from array of values.

- void **operator** *= (const MT &m2)

2.2.1 Detailed Description

template<class VT, class MT, int ROWS> class CGLA::ArithSqMat< VT, MT, ROWS >

Template for square matrices. Some functions like trace and determinant work only on square matrices. To express this in the class hierarchy, **ArithSqMat** (p. 6) was created. **ArithSqMat** (p. 6) is derived from **ArithMat** (p. 3) and contains a few extra facilities applicable only to square matrices.

2.2.2 Member Function Documentation

2.2.2.1 template<class VT, class MT, int ROWS> void CGLA::ArithSqMat< VT, MT, ROWS >::operator *= (const MT & *m2*) [inline]

Assignment multiplication of matrices. This function is not very efficient. This because we need a temporary matrix anyway, so it can't really be made efficient.

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

- ArithSqMat.h

2.3 CGLA::ArithVec< T, V, N > Class Template Reference

Public Types

- typedef T **ScalarType**
  For convenience we define a more meaningful name for the scalar type.
- typedef V **VectorType**
  A more meaningful name for vector type.

Public Methods

- **ArithVec** ()
  Construct 0 vector.
- **ArithVec** (T _a)
  Construct a vector where all coordinates are identical.
- **ArithVec** (T _a, T _b)
  Construct a 2D vector.
- **ArithVec** (T _a, T _b, T _c)
  Construct a 3D vector.
- **ArithVec** (T _a, T _b, T _c, T _d)
  Construct a 4D vector.
- void **set** (T _a, T _b)
  Set all coordinates of a 2D vector.
- void **set** (T _a, T _b, T _c)
  Set all coordinates of a 3D vector.
- void **set** (T _a, T _b, T _c, T _d)
  Set all coordinates of a 4D vector.
- const T & **operator[ ]** (int i) const
  Const index operator.
- T & **operator[ ]** (int i)
  Non-const index operator.

- T * **get** ()
- const T * **get** () const
- bool **operator==** (const V &v) const

  Equality operator.

- bool **operator==** (T k) const

  Equality wrt scalar. True if all coords are equal to scalar.

- bool **operator!=** (const V &v) const

  Inequality operator.

- bool **operator!=** (T k) const

  Inequality wrt scalar. True if any coord not equal to scalar.

- bool **all_l** (const V &v) const
- bool **all_le** (const V &v) const
- bool **all_g** (const V &v) const
- bool **all_ge** (const V &v) const
- void **operator** *= (T k)

  Assigment multiplication with scalar.

- void **operator/=** (T k)

  Assignment division with scalar.

- void **operator+=** (T k)

  Assignment addition with scalar. Adds scalar to each coordinate.

- void **operator-=** (T k)

  Assignment subtraction with scalar. Subtracts scalar from each coord.

- void **operator** *= (const V &v)

  Assignment multiplication with vector. Multiply each coord independently.

- void **operator/=** (const V &v)

  Assigment division with vector. Each coord divided independently.

- void **operator+=** (const V &v)

  Assignmment addition with vector.

- void **operator-=** (const V &v)

  Assignment subtraction with vector.

- const V **operator-** () const

  Negate vector.

- const V **operator** * (const V &v1) const
- const V **operator+** (const V &v1) const

  Add two vectors.

- const V **operator-** (const V &v1) const

  Subtract two vectors.

- const V **operator/** (const V &v1) const

  Divide two vectors. Each coord separately.

- const V **operator** * (T k) const

  Multiply scalar onto vector.

- const V **operator/** (T k) const

  Divide vector by scalar.

- const T **min** () const

  Return the smallest coordinate of the vector.

- const T **max** () const

  Return the largest coordinate of the vector.

Static Public Methods

- int **get_dim** ()

  Return dimension of vector.

Protected Attributes

- T **data** [N]

  The actual contents of the vector.

2.3.1 Detailed Description

template<class T, class V, int N> class CGLA::ArithVec< T, V, N >

The **ArithVec** (p. 7) class template represents a generic arithmetic vector. The three parameters to the template are

T - the scalar type (i.e. float, int, double etc.)

V - the name of the vector type. This template is always (and only) used as ancestor of concrete types, and the name of the class _inheriting_ _from_ this class is used as the V argument.

N - The final argument is the dimension N. For instance, N=3 for a 3D vector.

This class template contains all functions that are assumed to be the same for any arithmetic vector - regardless of dimension or the type of scalars used for coordinates.

The template contains no virtual functions which is important since they add overhead.

2.3.2 Member Function Documentation

2.3.2.1 template<class T, class V, int N> bool CGLA::ArithVec< T, V, N >::all_g (const V & *v*) const [inline]

Compare all coordinates against other vector. (>) Similar to testing whether we are on one side of three planes.

2.3.2.2 template<class T, class V, int N> bool CGLA::ArithVec< T, V, N >::all_ge (const V & *v*) const [inline]

Compare all coordinates against other vector. (>=) Similar to testing whether we are on one side of three planes.

2.3.2.3 template<class T, class V, int N> bool CGLA::ArithVec< T, V, N >::all_l (const V & *v*) const [inline]

Compare all coordinates against other vector. (<) Similar to testing whether we are on one side of three planes.

2.3.2.4 template<class T, class V, int N> bool CGLA::ArithVec< T, V, N >::all_le (const V & *v*) const [inline]

Compare all coordinates against other vector. (<=) Similar to testing whether we are on one side of three planes.

2.3.2.5 template<class T, class V, int N> const T* CGLA::ArithVec< T, V, N >::get () const [inline]

Get a const pointer to first element in data array. This function may be useful when interfacing with some other API such as OpenGL (TM).

2.3.2.6 template<class T, class V, int N> T* CGLA::ArithVec< T, V, N >::get () [inline]

Get a pointer to first element in data array. This function may be useful when interfacing with some other API such as OpenGL (TM)

2.3.2.7 template<class T, class V, int N> const V CGLA::ArithVec< T, V, N >::operator * (const V & *v1*) const [inline]

Multiply vector with vector. Each coord multiplied independently Do not confuse this operation with dot product.

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

- ArithVec.h

2.4 CGLA::BitMask Class Reference

Public Methods

- **BitMask** (int _fb, int _lb)
- **BitMask** (int num)

  first bit is 0 mask num bits.

- **BitMask** ()

  Mask everything.

- int **first_bit** () const

  get number of first bit in mask

- int **last_bit** () const

  get number of last bit in mask

- int **no_bits** () const

Return number of masked bits.

- int **mask** (int var) const

  Mask a number.

- int **mask_shift** (int var) const
- **Vec3i mask** (const **Vec3i** &v) const
- **Vec3i maskshift** (const **Vec3i** &v) const

2.4.1 Detailed Description

The **BitMask** (p. 10) class is mostly a utility class. The main purpose is to be able to extract a set of bits from an integer. For instance this can be useful if we traverse some tree structure and the integer is the index.

2.4.2 Constructor & Destructor Documentation

2.4.2.1 CGLA::BitMask::BitMask (int *_fb*, int *_lb*) [inline]

Mask _fb-_lb+1 bits beginning from _fb. First bit is 0. Say _fb=_lb=0. In this case, mask 1 bit namely 0.

2.4.3 Member Function Documentation

2.4.3.1 Vec3i CGLA::BitMask::mask (const Vec3i & *v*) const [inline]

Mask a vector by masking each coordinate.

2.4.3.2 int CGLA::BitMask::mask_shift (int *var*) const [inline]

Mask a number and shift back so the first bit inside the mask becomes bit 0.

2.4.3.3 Vec3i CGLA::BitMask::maskshift (const Vec3i & *v*) const [inline]

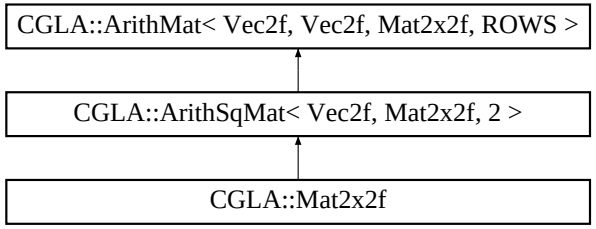
Mask each coord of a vector and shift

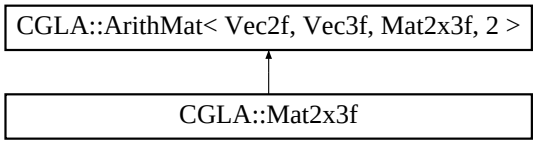
The documentation for this class was generated from the following file:

- BitMask.h

2.5 CGLA::Mat2x2f Class Reference

Inheritance diagram for CGLA::Mat2x2f::





Public Methods

- **Mat2x3f** (const **Vec3f** &_a, const **Vec3f** &_b)

  Construct **Mat2x3f** (p. 12) *from two* **Vec3f** (p. 19) *vectors (vectors become rows).*

- **Mat2x3f** ()

  Construct 0 matrix.

- **Mat2x3f** (const float *sa)

  Construct matrix from array of values.

2.6.1 Detailed Description

2x3 float matrix class. This class is useful for projecting a vector from 3D space to 2D.

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

- Mat2x3f.h

Public Methods

- **Mat2x2f** (**Vec2f** _a, **Vec2f** _b)

  Construct a **Mat2x2f** (p. 11) *from two* **Vec2f** (p. 17) *vectors.*

- **Mat2x2f** (float _a, float _b, float _c, float _d)

  Construct a **Mat2x2f** (p. 11) *from four scalars.*

- **Mat2x2f** ()

  Construct the 0 matrix.

2.5.1 Detailed Description

Two by two float matrix. This class is useful for various vector transformations in the plane.

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

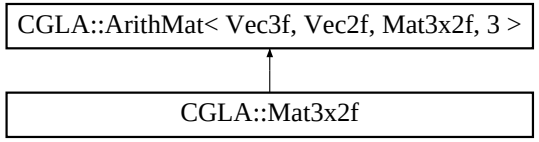
- Mat2x2f.h

2.6 CGLA::Mat2x3f Class Reference

Inheritance diagram for CGLA::Mat2x3f::

2.7 CGLA::Mat3x2f Class Reference

Inheritance diagram for CGLA::Mat3x2f::



Public Methods

- **Mat3x2f** (const **Vec2f** &_a, const **Vec2f** &_b, const **Vec2f** &_c)
- **Mat3x2f** ()

  Construct 0 matrix.

- **Mat3x2f** (const float *sa)

  Construct matrix from array of values.

2.7.1 Detailed Description

3x2 float matrix class. This class is useful for going from plane to 3D coordinates.

2.7.2 Constructor & Destructor Documentation

2.7.2.1 CGLA::Mat3x2f::Mat3x2f (const Vec2f & *_a*, const Vec2f & *_b*, const Vec2f & *_c*) [inline]

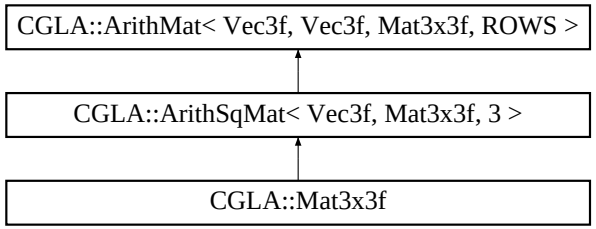
Construct matrix from three **Vec2f** (p. 17) vectors which become the rows of the matrix.

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

- Mat2x3f.h

2.8 CGLA::Mat3x3f Class Reference

Inheritance diagram for CGLA::Mat3x3f::



Public Methods

- **Mat3x3f** (**Vec3f** _a, **Vec3f** _b, **Vec3f** _c)

Construct matrix from 3 **Vec3f** (p. 19) *vectors.*

- **Mat3x3f** ()

  Construct the 0 matrix.

- **Mat3x3f** (float a)

  Construct a matrix from a single scalar value.

2.8.1 Detailed Description

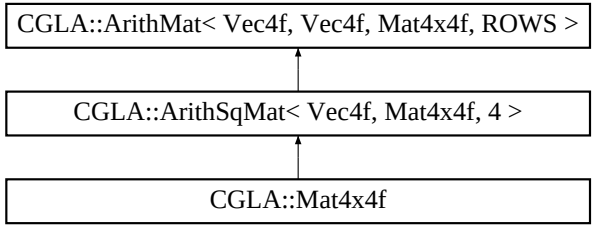
3 by 3 float matrix. This class will typically be used for rotation or scaling matrices for 3D vectors.

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

- Mat3x3f.h

2.9 CGLA::Mat4x4f Class Reference

Inheritance diagram for CGLA::Mat4x4f::



Public Methods

- **Mat4x4f** (**Vec4f** _a, **Vec4f** _b, **Vec4f** _c, **Vec4f** _d)

  Construct a **Mat4x4f** (p. 14) *from four* **Vec4f** (p. 23) *vectors.*

- **Mat4x4f** ()

  Construct the 0 matrix.

- **Mat4x4f** (const float *sa)

  Construct from a pointed to array of 16 floats.

- const **Vec3f mul_3D_vector** (const **Vec3f** &v) const
- const **Vec3f mul_3D_point** (const **Vec3f** &v) const
- const **Vec3f project_3D_point** (const **Vec3f** &v) const

2.9.1 Detailed Description

Four by four float matrix. This class is useful for transformations such as perspective projections or translation where 3x3 matrices do not suffice.

2.9.2 Member Function Documentation

2.9.2.1 const Vec3f CGLA::Mat4x4f::mul_3D_point (const Vec3f & *v*) const [inline]

Multiply 3D point onto matrix. Here the fourth coordinate becomes 1 to ensure that the point is translated. Note that the vector is converted back into a **Vec3f** (p. 19) without any division by w. This is deliberate: Typically, w=1 except for projections. If we are doing projection, we can use project_3D_point instead

2.9.2.2 const Vec3f CGLA::Mat4x4f::mul_3D_vector (const Vec3f & *v*) const [inline]

Multiply vector onto matrix. Here the fourth coordinate is se to 0. This removes any translation from the matrix. Useful if one wants to transform a vector which does not represent a point but a direction. Note that this is not correct for transforming normal vectors if the matric contains anisotropic scaling.

2.9.2.3 const Vec3f CGLA::Mat4x4f::project_3D_point (const Vec3f & *v*) const [inline]

Multiply 3D point onto matrix. We set w=1 before multiplication and divide by w after multiplication.

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

- Mat4x4f.h

2.10 CGLA::Quaternion Class Reference

Public Methods

- **Quaternion** ()

  Construct 0 quaternion.

- **Quaternion** (const **Vec3f** _qv, float _qw=1)

  Construct quaternion from vector and scalar.

- **Quaternion** (float x, float y, float z, float _qw)

  Construct quaternion from four scalars.

- void **set** (float x, float y, float z, float _qw)

  Assign values to a quaternion.

- void **get** (float &x, float &y, float &z, float &_qw) const

  Get values from a quaternion.

- **Mat3x3f get_mat3x3f** () const

  Get a 3x3 rotation matrix from a quaternion.

- **Mat4x4f get_mat4x4f** () const
  Get a 4x4 rotation matrix from a quaternion.
- void **make_rot** (float angle, const **Vec3f** &)
  Construct a **Quaternion** (p. 15) *from an angle and axis of rotation.*
- void **make_rot** (const **Vec3f** &, const **Vec3f** &)
- void **get_rot** (float &angle, **Vec3f** &)
  Obtain angle of rotation and axis.
- Quaternion **operator** ∗ (Quaternion quat) const
  Multiply two quaternions. (Combine their rotation).
- Quaternion **operator** ∗ (float scalar) const
  Multiply scalar onto quaternion.
- Quaternion **operator+** (Quaternion quat) const
  Add two quaternions.
- Quaternion **inverse** () const
  Invert quaternion.
- Quaternion **conjugate** () const
  Return conjugate quaternion.
- float **norm** () const
  Compute norm of quaternion.
- Quaternion **normalize** ()
  Normalize quaternion.
- **Vec3f apply** (const **Vec3f** &vec) const
  Rotate vector according to quaternion.

Public Attributes

- **Vec3f qv**
  Vector part of quaternion.
- float **qw**
  Scalar part of quaternion.

2.10.1 Detailed Description

A Quaterinion class. Quaternions are algebraic entities useful for rotation.

2.10.2 Member Function Documentation

2.10.2.1 void CGLA::Quaternion::make_rot (const Vec3f &, const Vec3f &)

Construct a **Quaternion** (p. 15) rotating from the direction given by the first argument to the direction given by the second.

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

- Quaternion.h

2.11 CGLA::UnitVector Class Reference

Public Methods

- **UnitVector** (const **Vec3f** &v)
  Construct unitvector from normal vector.
- **UnitVector** ()
  Construct default unit vector.
- float **t** () const
  Get theta angle.
- float **f** () const
  Get phi angle.
- **operator Vec3f** () const
  Reconstruct **Vec3f** (p. 19) *from unit vector.*
- bool **operator==** (const UnitVector &u) const
  Test for equality.

2.11.1 Detailed Description

The **UnitVector** (p. 17) stores a unit length vector as two angles.

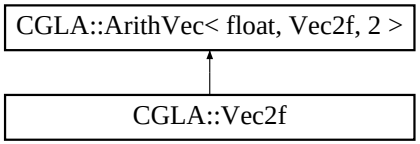
A vector stored as two (fix point) angles is much smaller than vector stored in the usual way. On a 32 bit architecture this class should take up four bytes. not too bad.

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

- UnitVector.h

2.12 CGLA::Vec2f Class Reference

Inheritance diagram for CGLA::Vec2f::



Public Methods

- float **length** () const
  Return Euclidean length.
- void **normalize** ()

Normalize vector.

2.12.1 Detailed Description

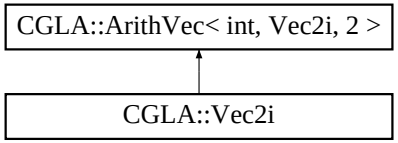
2D floating point vector

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

- Vec2f.h

2.13 CGLA::Vec2i Class Reference

Inheritance diagram for CGLA::Vec2i::



Public Methods

- **Vec2i** ()

  Construct 0 vector.

- **Vec2i** (int _a, int _b)

  Construct 2D int vector.

- **Vec2i** (const **Vec2f** &v)

  Convert from 2D float vector.

2.13.1 Detailed Description

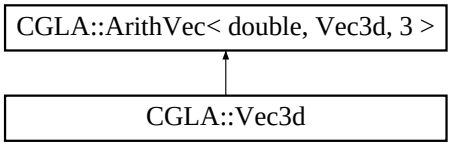
2D Integer vector.

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

- Vec2i.h

2.14 CGLA::Vec3d Class Reference

Inheritance diagram for CGLA::Vec3d::



Public Methods

- **Vec3d** ()

  Construct 0 vector.

- **Vec3d** (double a, double b, double c)

  Construct vector.

- **Vec3d** (double a)

  Construct vector where all coords = a.

- **Vec3d** (const **Vec3i** &v)

  Convert from int vector.

- **Vec3d** (const **Vec3f** &v)

  Convert from float vector.

- double **length** () const

  Returns euclidean length.

- void **normalize** ()

  Normalize vector.

- void **get_spherical** (double &, double &, double &) const
- bool **set_spherical** (double, double, double)

2.14.1 Detailed Description

A 3D double vector. Useful for high precision arithmetic.

2.14.2 Member Function Documentation

2.14.2.1 void CGLA::Vec3d::get_spherical (double &, double &, double &) const

Get the vector in spherical coordinates. The first argument (theta) is inclination from the vertical axis. The second argument (phi) is the angle of rotation about the vertical axis. The third argument (r) is the length of the vector.

2.14.2.2 bool CGLA::Vec3d::set_spherical (double, double, double)

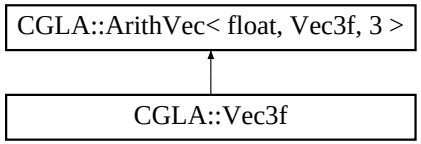
Assign the vector in spherical coordinates. The first argument (theta) is inclination from the vertical axis. The second argument (phi) is the angle of rotation about the vertical axis. The third argument (r) is the length of the vector.

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

- Vec3d.h

2.15 CGLA::Vec3f Class Reference

Inheritance diagram for CGLA::Vec3f::



Public Methods

- **Vec3f** ()

  Construct 0 vector.

- **Vec3f** (float a, float b, float c)

  Construct a 3D float vector.

- **Vec3f** (float a)

  Construct a vector with 3 identical coordinates.

- **Vec3f** (const **Vec3i** &v)

  Construct from a 3D int vector.

- **Vec3f** (const **Vec3usi** &v)

  Construct from a 3D unsigned int vector.

- **Vec3f** (const **Vec3d** &)

  Construct from a 3D double vector.

- **Vec3f** (const **Quaternion** &)

  Construct from a **Quaternion** (p. 15). *((NOTE: more explanation needed)).*

- float **length** () const

  Compute Euclidean length.

- void **normalize** ()

  Normalize vector.

- void **get_spherical** (float &, float &, float &) const
- void **set_spherical** (float, float, float)

2.15.1 Detailed Description

3D float vector. Class **Vec3f** (p. 19) is the vector typically used in 3D computer graphics. The class has many constructors since we may need to convert from other vector types. Most of these are explicit to avoid automatic conversion.

2.15.2 Member Function Documentation

2.15.2.1 void CGLA::Vec3f::get_spherical (float &, float &, float &) const

Get the vector in spherical coordinates. The first argument (theta) is inclination from the vertical axis. The second argument (phi) is the angle of rotation about the vertical axis. The third argument (r) is the length of the vector.

2.15.2.2 void CGLA::Vec3f::set_spherical (float, float, float)

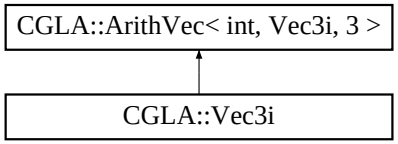
Assign the vector in spherical coordinates. The first argument (theta) is inclination from the vertical axis. The second argument (phi) is the angle of rotation about the vertical axis. The third argument (r) is the length of the vector.

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

- Vec3f.h

2.16 CGLA::Vec3i Class Reference

Inheritance diagram for CGLA::Vec3i::



Public Methods

- **Vec3i** ()

  Construct 0 vector.

- **Vec3i** (int _a, int _b, int _c)

  Construct a 3D integer vector.

- **Vec3i** (int a)

  Construct a 3D integer vector with 3 identical coordinates.

- **Vec3i** (const **Vec3f** &v)

  Construct from a **Vec3f** (p. 19).

- **Vec3i** (const **Vec3uc** &v)

  Construct from a **Vec3uc** (p. 21).

- **Vec3i** (const **Vec3usi** &v)

  Construct from a **Vec3usi** (p. 22).

2.16.1 Detailed Description

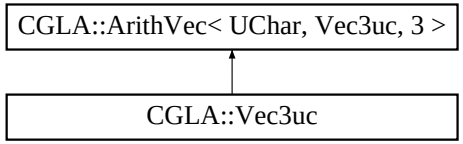
3D integer vector. This class does not really extend the template and hence provides only the basic facilities of an **ArithVec** (p. 7). The class is typically used for indices to 3D voxel grids.

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

- Vec3i.h

2.17 CGLA::Vec3uc Class Reference

Inheritance diagram for CGLA::Vec3uc::



Public Methods

- **Vec3uc** ()

  Construct 0 vector.

- **Vec3uc** (UChar _a, UChar _b, UChar _c)

  Construct 3D uchar vector.

- **Vec3uc** (const **Vec3i** &v)

  Convert from int vector.

2.17.1 Detailed Description

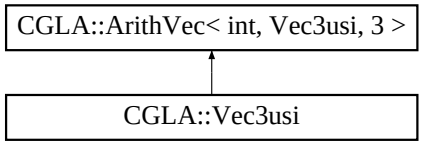
3D unsigned char vector.

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

- Vec3uc.h

2.18 CGLA::Vec3usi Class Reference

Inheritance diagram for CGLA::Vec3usi::



Public Methods

- **Vec3usi** ()

  Construct 0 vector.

- **Vec3usi** (USInt _a, USInt _b, USInt _c)

  Construct a **Vec3usi** (p. 22).

- **Vec3usi** (const **Vec3i** &v)

  Construct a **Vec3usi** (p. 22) *from a* **Vec3i** (p. 21).

2.18.1 Detailed Description

Unsigned short int 3D vector class. This class is mainly useful if we need a 3D int vector that takes up less room than a **Vec3i** (p. 21) but holds larger numbers than a

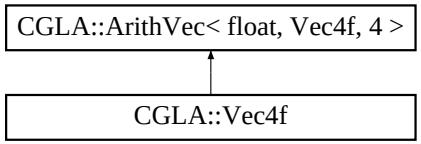
Vec3c.

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

- Vec3usi.h

2.19 CGLA::Vec4f Class Reference

Inheritance diagram for CGLA::Vec4f::



Public Methods

- **Vec4f** ()
  Construct a (0,0,0,0) homogenous Vector.
- **Vec4f** (float _a)
  Construct a (0,0,0,0) homogenous Vector.
- **Vec4f** (float _a, float _b, float _c, float _d)
  Construct a 4D vector.
- **Vec4f** (float _a, float _b, float _c)
  Construct a homogenous vector (a,b,c,1).
- **Vec4f** (const **Vec3f** &v)
  Construct a homogenous vector from a non-homogenous.
- **Vec4f** (const **Vec3f** &v, float _d)
  Construct a homogenous vector from a non-homogenous.
- void **de_homogenize** ()
  Divide all coordinates by the fourth coordinate.

2.19.1 Detailed Description

A four dimensional floating point vector. This class is also used (via typedef) for homogeneous vectors.

2.19.2 Member Function Documentation

2.19.2.1 void CGLA::Vec4f::de_homogenize () [inline]

This function divides a vector (x,y,z,w) by w to obtain a new 4D vector where w=1.

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

- Vec4f.h