CGLA Reference

J. Andreas Bærentzen

September 1, 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.

For some more specific help look at the next section where some of the commonly used operations are shown.

1.3 How to get CGLA

If you have this document but not the code, look at
http://www.imm.dtu.dk/ jab/software.html#cgla

1.4 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.

1.5 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: Examples of Usage

While this is mostly a reference manual some examples are probably helpful. The examples below are by no means complete. Many things are possible but not covered below. However, most of the common usage is shown, so this should be enough to get you started. Note that in the following we assume that you are `using namespace CGLA` and hence don't prefix with `CGLA::`.

In short, to compile the examples below you would need the following in the top of your file

```
#include <iostream> // For input output
#include "CGLA/Vec3f.h"
#include "CGLA/Quaternion.h"
#include "CGLA/Mat4x4f.h"

using namespace std; // For input output
using namespace CGLA;
```

To begin with let us create 3 3D vectors. This is done as follows:

```
Vec3f p0(10,10,10);
Vec3f p1(20,10,10);
Vec3f p2(10,20,10);
```

A very common operation is to compute the normal of a triangle from the position of its vertices. Assuming the three vectors represent the vertices of a triangle, we can compute the normal by finding the vector from the first vertex to the two other vertices, taking the cross product and normalizing the result. This is a one-liner:

```
Vec3f n = normalize(cross(p1-p0,p2-p0));
```

Quite a lot goes on in the snippet above. Observe that the - operator also works for vectors. In fact almost all the arithmetic operators work for vectors. You can also use assignment operators (i.e `+=`) which is often faster. Then there is the function `cross` which simply computes the cross product of its arguments. Another frequently used function is `dot` which takes the dot product. Finally the vector is normalized using the function normalize.

Of course, we can print all or at least most CGLA entities. For example

```
cout << n << endl;
```

will print the normal vector just computed. We can also treat a vector as an array as shown below

```
float x = n[0];
```

here, of course, we just extracted the first coordinate of the vector.

CGLA contains a number of features that are not used very frequently, but which are used frequently enough to warrent inclusion. A good example is assigning to a vector using spherical coordinates:

```
Vec3f p;
p.set_spherical(0.955317, 3.1415926f/4.0f , 1);
```

CGLA also includes a quaternion class. Here it is used to construct a quaternion which will rotate the x axis into the y axis.

```
Quaternion q;
q.make_rot(Vec3f(1,0,0), Vec3f(0,1,0));
```

Next, we construct a 4×4 matrix `m` and assign a translation matrix to the newly constructed matrix. After that we ask the quaternion to return a 4×4 matrix corresponding to its rotation. This rotation matrix is then multiplied onto `m`.

```
Mat4x4f m = translation_Mat4x4f(Vec3f(1,2,3));
m *= q.get_mat4x4f();
```

Just like for vectors, the subscript operator works on matrices. However, in this case

there are two indices. Just using one index will return the ith row as a vector as shown on the first line below. On the second line we see that using two indices will get us an element in the matrix.

```
Vec4f v4 = m[0];
float c = m[0][3];
```

There is a number of constructors for vectors. The default constructor will create a null vector as we have already seen. We can also specify all the coordinates. Finally, we can pass just a single number a. This will create the $[a\ a\ a]^T$ vector. For instance, below we create the $[1\ 1\ 1]^T$ vector. Subsequently, this vector is multiplied onto `m`.

```
Vec3f p(1);
Vec3f p2 = m.mul_3D_point(p);
```

Note though that `m` is a 4×4 matrix so ... how is that possible? Well, we use the function `mul_3D_point` which, effectively, adds a $w = 1$ coordinate to p making it a 4D vector. This w coordinate is stripped afterwards. In practice, this means that the translation part of the matrix is also applied. There is a similar function `mul_3D_vector` if we want to transform vectors without having the translation. This function, effectively, sets $w = 0$.

Finally, CGLA is often used together with OpenGL although there is no explicit tie to the OpenGL library. However, we can call the `get` function of most CGLA entities to get a pointer to the contents. E.g. `p.get()` will return a pointer to the first float in the 3D vector `p`. This can be used with OpenGL's "v" functions as shown below.

```
glVertex3fv(p.get());
```

3 CGLA Class Documentation

3.1 CGLA Namespace Reference

Compounds

- class ArithMat
- class ArithSqMat
- class ArithVec
- class BitMask
- class Mat2x2f
- class Mat2x3f
- class Mat3x2f
- class Mat3x3f
- class Mat4x4f
- class Quaternion
- class UnitVector
- class Vec2f
- class Vec2i
- class Vec3d
- class Vec3f
- class Vec3i
- class Vec3uc
- class Vec3usi
- class Vec4f
- class Vec4uc

Typedefs

- typedef Vec4f Vec3Hf

Enumerations

- enum Axis

  Useful enum that represents coordiante axes.

Functions

- template<class VVT, class HVT, class MT, int ROWS> const MT operator * (double k, const ArithMat< VVT, HVT, MT, ROWS > &v)

  Multiply scalar onto matrix.

- template<class VVT, class HVT, class MT, int ROWS> const MT operator * (float k, const ArithMat< VVT, HVT, MT, ROWS > &v)

  Multiply scalar onto matrix.

- template<class VVT, class HVT, class MT, int ROWS> const MT operator * (int k, const ArithMat< VVT, HVT, MT, ROWS > &v)

  Multiply scalar onto matrix.

- template<class VVT, class HVT, class MT, int ROWS> VVT operator * (const ArithMat< VVT, HVT, MT, ROWS > &m, const HVT &v)

  Multiply vector onto matrix.

- template<class VVT, class HVT, class HV1T, class VV2T, class MT1, class MT2, class MT, int ROWS1, int ROWS2> void mul (const ArithMat< VVT, HV1T, MT1, ROWS1 > &m1, const ArithMat< VV2T, HVT, MT2, ROWS2 > &m2, ArithMat< VVT, HVT, MT, ROWS1 > &m)
- template<class VVT, class HVT, class M1T, class M2T, int ROWS, int COLS> void transpose (const ArithMat< VVT, HVT, M1T, ROWS > &m, ArithMat< HVT, VVT, M2T, COLS > &m_new)
- template<class VVT, class HVT, class MT, int ROWS> std::ostream & operator<< (std::ostream &os, const ArithMat< VVT, HVT, MT, ROWS > &m)
- template<class VVT, class HVT, class MT, int ROWS> std::istream & operator>> (std::istream &is, const ArithMat< VVT, HVT, MT, ROWS > &m)
- template<class VT, class MT, int ROWS> MT operator * (const ArithSqMat< VT, MT, ROWS > &m1, const ArithSqMat< VT, MT, ROWS > &m2)
- template<class VT, class MT, int ROWS> MT transpose (const ArithSqMat< VT, MT, ROWS > &m)
- template<class VT, class MT, int ROWS> MT::ScalarType trace (const ArithSqMat< VT, MT, ROWS > &M)

  Compute trace. Works only for sq. matrices.

- template<class T, class V, int N> std::ostream & operator<< (std::ostream &os, const ArithVec< T, V, N > &v)

  Put to operator for ArithVec descendants.

- template<class T, class V, int N> std::istream & operator>> (std::istream &is, ArithVec< T, V, N > &v)

  Get from operator for ArithVec descendants.

- template<class T, class V, int N> T dot (const ArithVec< T, V, N > &v0, const ArithVec< T, V, N > &v1)
- template<class T, class V, int N> T sqr_length (const ArithVec< T, V, N > &v)
- template<class T, class V, int N> const V operator * (double k, const ArithVec< T, V, N > &v)
- template<class T, class V, int N> const V operator * (float k, const ArithVec< T, V, N > &v)
- template<class T, class V, int N> const V operator * (int k, const ArithVec< T, V, N > &v)
- template<class T, class V, int N> V v_min (const ArithVec< T, V, N > &v0, const ArithVec< T, V, N > &v1)
- template<class T, class V, int N> V v_max (const ArithVec< T, V, N > &v0, const ArithVec< T, V, N > &v1)
- float determinant (const Mat2x2f &m)
- bool invert (const Mat2x2f &m, Mat2x2f &)
- Mat3x3f invert (const Mat3x3f &)

  Invert 3x3 matrix.

- Mat3x3f rotation_Mat3x3f (CGLA::Axis axis, float angle)

  Create a rotation _matrix. Rotates about one of the major axes.

- Mat3x3f scaling_Mat3x3f (const Vec3f &)

  Create a scaling matrix.

- Mat3x3f identity_Mat3x3f ()

  Create an identity matrix.

- float determinant (const Mat3x3f &m)
- Mat4x4f rotation_Mat4x4f (CGLA::Axis axis, float angle)

  Create a rotation _matrix. Rotates about one of the major axes.

- Mat4x4f translation_Mat4x4f (const Vec3f &)

  Create a translation matrix.

- Mat4x4f scaling_Mat4x4f (const Vec3f &)

  Create a scaling matrix.

- Mat4x4f identity_Mat4x4f ()

  Create an identity matrix.

- Mat4x4f adjoint (const Mat4x4f &in)
- float determinant (const Mat4x4f &)
- Mat4x4f invert (const Mat4x4f &)

  Compute the inverse matrix of a Mat4x4f.

- Mat4x4f invert_affine (const Mat4x4f &)

  Compute the inverse matrix of a Mat4x4f that is affine.

- Mat4x4f perspective_Mat4x4f (float d)
- bool operator== (const Quaternion &q0, const Quaternion &q1)

  Compare for equality.

- std::ostream & operator<< (std::ostream &os, const Quaternion v)

Print quaternion to stream.

- Quaternion operator ∗ (float scalar, Quaternion quat)

  Multiply scalar onto quaternion.

- Quaternion slerp (Quaternion q0, Quaternion q1, float t)
- std::ostream & operator<< (std::ostream &os, const UnitVector &u)

  Inline output operator.

- Vec2f normalize (const Vec2f &v)

  Returns normalized vector.

- Vec2f orthogonal (const Vec2f &v)

  Rotates vector 90 degrees to obtain orthogonal vector.

- bool linear_combine (const Vec2f &, const Vec2f &, const Vec2f &, float &, float &)
- double dot (const Vec3d &x, const Vec3d &y)

  Compute dot product.

- Vec3d cross (const Vec3d &x, const Vec3d &y)

  Compute cross product.

- Vec3f normalize (const Vec3f &v)

  Returns normalized vector.

- Vec3f cross (const Vec3f &x, const Vec3f &y)

  Returns cross product of arguments.

- void orthogonal (const Vec3f &, Vec3f &, Vec3f &)

3.1.1 Detailed Description

The H in Vec3Hf stands for homogenous.

3.1.2 Typedef Documentation

3.1.2.1 typedef Vec4f CGLA::Vec3Hf

A 3D homogeneous vector is simply a four D vector. I find this simpler than a special class for homogeneous vectors.

3.1.3 Function Documentation

3.1.3.1 Mat4x4f adjoint (const Mat4x4f & *in*)

Compute the adjoint of a matrix. This is the matrix where each entry is the subdeterminant of 'in' where the row and column of the element is removed. Use mostly to compute the inverse

3.1.3.2 float determinant (const Mat4x4f &)

Compute the determinant of a 4x4f matrix.

3.1.3.3 float determinant (const Mat3x3f & *m*) [inline]

Compute determinant. There is a more generic function for computing determinants of square matrices (ArithSqMat). This one is faster but works only on Mat3x3f

3.1.3.4 float determinant (const Mat2x2f & *m*) [inline]

Compute the determinant of a Mat2x2f. This function is faster than the generic determinant function for ArithSqMat

3.1.3.5 template<class T, class V, int N> T dot (const ArithVec< T, V, N > & *v0*, const ArithVec< T, V, N > & *v1*) [inline]

Dot product for two vectors. The '∗' operator is reserved for coordinatewise multiplication of vectors.

3.1.3.6 bool invert (const Mat2x2f & *m*, Mat2x2f &)

Invert a two by two matrix. ((NOTE: Perhaps this function should be changed to return the inverse)).

3.1.3.7 bool linear_combine (const Vec2f &, const Vec2f &, const Vec2f &, float &, float &)

The two last (scalar) arguments are the linear combination of the two first arguments (vectors) which produces the third argument.

3.1.3.8 template<class VVT, class HVT, class HV1T, class VV2T, class MT1, class MT2, class MT, int ROWS1, int ROWS2> void mul (const ArithMat< VVT, HV1T, MT1, ROWS1 > & *m1*, const ArithMat< VV2T, HVT, MT2, ROWS2 > & *m2*, ArithMat< VVT, HVT, MT, ROWS1 > & *m*) [inline]

Multiply two arbitrary matrices. In principle, this function could return a matrix, but in general the new matrix will be of a type that is different from either of the two matrices that are multiplied together. We do not want to return an ArithMat - so it seems best to let the return value be a reference arg.

This template can only be instantiated if the dimensions of the matrices match – i.e. if the multiplication can actually be carried out. This is more type safe than the win32 version below.

3.1.3.9 template<class T, class V, int N> const V operator ∗ (int *k*, const ArithVec< T, V, N > & *v*) [inline]

Multiply int onto vector. See the note in the documentation regarding multiplication of a double onto a vector.

3.1.3.10 template<class T, class V, int N> const V operator ∗ (float *k*, const ArithVec< T, V, N > & *v*) [inline]

Multiply float onto vector. See the note in the documentation regarding multiplication of a double onto a vector.

3.1.3.11 template<class T, class V, int N> const V operator ∗ (double *k*, const ArithVec< T, V, N > & *v*) [inline]

Multiply double onto vector. This operator handles the case where the vector is on the righ side of the '∗'.

Note:

It seems to be optimal to put the binary operators inside the ArithVec class template, but the operator functions whose left operand is _not_ a vector cannot be inside, hence they are here. We need three operators for scalar ∗ vector although they are identical, because, if we use a separate template argument for the left operand, it will match any type. If we use just T as type for the left operand hoping that other built-in types will be automatically converted, we will be disappointed. It seems that a float ∗ ArithVec<float,Vec3f,3> function is not found if the left operand is really a double.

3.1.3.12 template<class VT, class MT, int ROWS> MT operator ∗ (const ArithSqMat< VT, MT, ROWS > & *m1*, const ArithSqMat< VT, MT, ROWS > & *m2*) [inline]

Multiply two matrices derived from same type, producing a new of same type

3.1.3.13 template<class VVT, class HVT, class MT, int ROWS> std::ostream& operator<< (std::ostream & *os*, const ArithMat< VVT, HVT, MT, ROWS > & *m*) [inline]

Put to operator

3.1.3.14 template<class VVT, class HVT, class MT, int ROWS> std::istream& operator>> (std::istream & *is*, const ArithMat< VVT, HVT, MT, ROWS > & *m*) [inline]

Get from operator

3.1.3.15 void orthogonal (const Vec3f &, Vec3f &, Vec3f &)

Compute basis of orthogonal plane. Given a vector Compute two vectors that are orothogonal to it and to each other.

3.1.3.16 Mat4x4f perspective_Mat4x4f (float *d*)

Create a perspective matrix. Assumes the eye is at the origin and that we are looking down the negative z axis.

ACTUALLY THE EYE IS NOT AT THE ORIGIN BUT BEHIND IT. CHECK UP ON THIS ONE

3.1.3.17 Quaternion slerp (Quaternion *q0*, Quaternion *q1*, float *t*) [inline]

Perform linear interpolation of two quaternions. The last argument is the parameter used to interpolate between the two first. SLERP - invented by Shoemake - is a good way to interpolate because the interpolation is performed on the unit sphere.

3.1.3.18 template<class T, class V, int N> T sqr_length (const ArithVec< T, V, N > & *v*) [inline]

Compute the sqr length by taking dot product of vector with itself.

3.1.3.19 template<class VT, class MT, int ROWS> MT transpose (const ArithSqMat< VT, MT, ROWS > & *m*) [inline]

Multiply two matrices derived from same type, producing a new of same type

3.1.3.20 template<class VVT, class HVT, class M1T, class M2T, int ROWS, int COLS> void transpose (const ArithMat< VVT, HVT, M1T, ROWS > & *m*, ArithMat< HVT, VVT, M2T, COLS > & *m_new*) [inline]

Transpose. See the discussion on mul if you are curious as to why I don't simply return the transpose.

3.1.3.21 template<class T, class V, int N> V v_max (const ArithVec< T, V, N > & *v0*, const ArithVec< T, V, N > & *v1*) [inline]

Returns the vector containing for each coordinate the largest value from two vectors.

3.1.3.22 template<class T, class V, int N> V v_min (const ArithVec< T, V, N > & *v0*, const ArithVec< T, V, N > & *v1*) [inline]

Returns the vector containing for each coordinate the smallest value from two vectors.

3.2 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.

3.2.1 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

3.2.2 Member Function Documentation

3.2.2.1 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).

3.2.2.2 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).

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

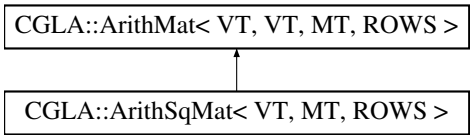
Set values by passing an array to the matrix. The values should be ordered like [[row][row]...[row]]

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

- ArithMat.h

3.3 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)

3.3.1 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 was created. ArithSqMat is derived from ArithMat and contains a few extra facilities applicable only to square matrices.

3.3.2 Member Function Documentation

3.3.2.1 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

3.4 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_coord () const

  Return the smallest coordinate of the vector.

- const T max_coord () 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.

3.4.1 Detailed Description

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

The ArithVec 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.

3.4.2 Member Function Documentation

3.4.2.1 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.

3.4.2.2 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.

3.4.2.3 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.

3.4.2.4 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.

3.4.2.5 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).

3.4.2.6 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)

3.4.2.7 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

3.5 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

3.5.1 Detailed Description

The BitMask 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.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 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.

3.5.3 Member Function Documentation

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

Mask a vector by masking each coordinate.

3.5.3.2 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.

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

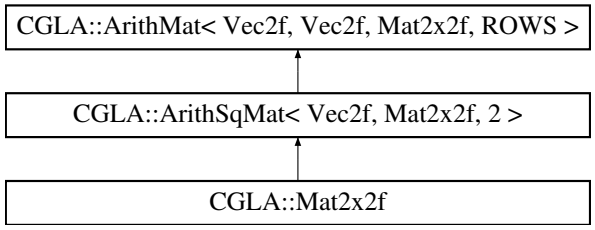
Mask each coord of a vector and shift

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

- BitMask.h

3.6 CGLA::Mat2x2f Class Reference

Inheritance diagram for CGLA::Mat2x2f::



Public Methods

- Mat2x2f (Vec2f _a, Vec2f _b)

  Construct a Mat2x2f from two Vec2f vectors.

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

  Construct a Mat2x2f from four scalars.

- Mat2x2f ()

  Construct the 0 matrix.

3.6.1 Detailed Description

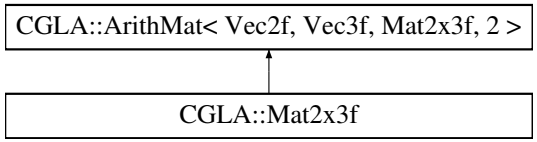
Two by two float matrix. This class is useful for various vector transformations in the plane.

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

- Mat2x2f.h

3.7 CGLA::Mat2x3f Class Reference

Inheritance diagram for CGLA::Mat2x3f::



Public Methods

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

  Construct Mat2x3f from two Vec3f vectors (vectors become rows).

- Mat2x3f ()

  Construct 0 matrix.

- Mat2x3f (const float *sa)

  Construct matrix from array of values.

3.7.1 Detailed Description

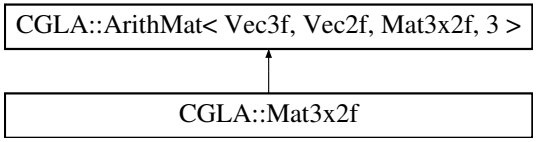
2x3 float matrix class. This class is useful for projecting a vector from 3D space to 2D.

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

- Mat2x3f.h

3.8 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.

3.8.1 Detailed Description

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

3.8.2 Constructor & Destructor Documentation

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

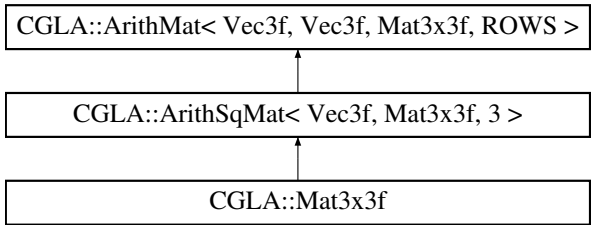
Construct matrix from three Vec2f vectors which become the rows of the matrix.

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

- Mat2x3f.h

3.9 CGLA::Mat3x3f Class Reference

Inheritance diagram for CGLA::Mat3x3f::



Public Methods

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

  Construct matrix from 3 Vec3f vectors.

- Mat3x3f ()

  Construct the 0 matrix.

- Mat3x3f (float a)

  Construct a matrix from a single scalar value.

3.9.1 Detailed Description

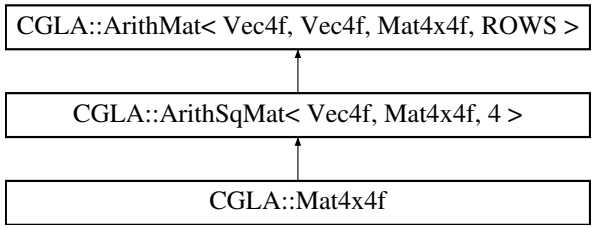
3 by 3 float matrix. This class will typically be used for rotation or scaling matrices for 3D vectors.

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

- Mat3x3f.h

3.10 CGLA::Mat4x4f Class Reference

Inheritance diagram for CGLA::Mat4x4f::



Public Methods

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

  Construct a Mat4x4f from four Vec4f 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

3.10.1 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.

3.10.2 Member Function Documentation

3.10.2.1 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 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

3.10.2.2 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.

3.10.2.3 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

3.11 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 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.

3.11.1 Detailed Description

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

3.11.2 Member Function Documentation

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

Construct a Quaternion 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

3.12 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 from unit vector.

- bool operator== (const UnitVector &u) const

  Test for equality.

3.12.1 Detailed Description

The UnitVector stores a unit length vector as two angles.

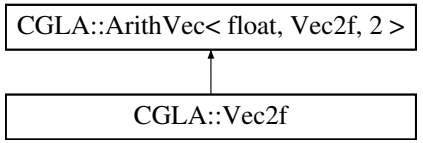
A vector stored as two (fix point) angles is much smaller than vector stored in the usual way. On a 32 bit architecture this class should take up four bytes. not too bad.

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

- UnitVector.h

3.13 CGLA::Vec2f Class Reference

Inheritance diagram for CGLA::Vec2f::



Public Methods

- float length () const

  Return Euclidean length.

- void normalize ()

Normalize vector.

3.13.1 Detailed Description

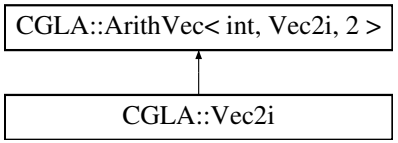
2D floating point vector

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

- Vec2f.h

3.14 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.

3.14.1 Detailed Description

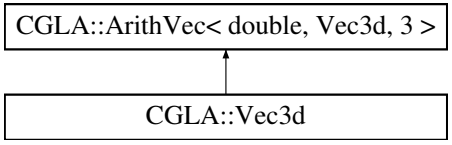
2D Integer vector.

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

- Vec2i.h

3.15 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)

3.15.1 Detailed Description

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

3.15.2 Member Function Documentation

3.15.2.1 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.

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

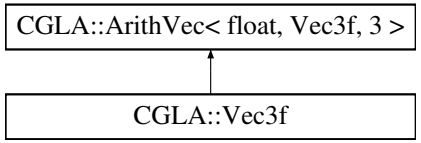
Assign the vector in spherical coordinates. The first argument (theta) is inclination from the vertical axis. The second argument (phi) is the angle of rotation about the vertical axis. The third argument (r) is the length of the vector.

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

- Vec3d.h

3.16 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. ((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)

3.16.1 Detailed Description

3D float vector. Class Vec3f 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.

3.16.2 Member Function Documentation

3.16.2.1 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.

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

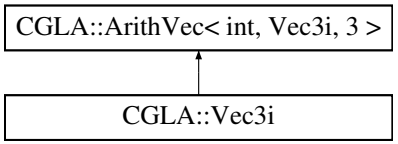
Assign the vector in spherical coordinates. The first argument (theta) is inclination from the vertical axis. The second argument (phi) is the angle of rotation about the vertical axis. The third argument (r) is the length of the vector.

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

- Vec3f.h

3.17 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.
- Vec3i (const Vec3uc &v)
  Construct from a Vec3uc.
- Vec3i (const Vec3usi &v)
  Construct from a Vec3usi.

3.17.1 Detailed Description

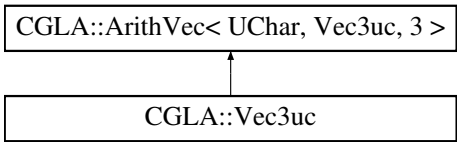
3D integer vector. This class does not really extend the template and hence provides only the basic facilities of an ArithVec. The class is typically used for indices to 3D voxel grids.

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

- Vec3i.h

3.18 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.

3.18.1 Detailed Description

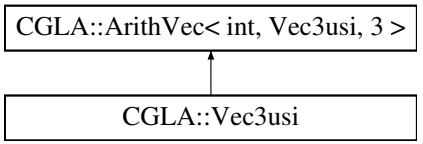
3D unsigned char vector.

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

- Vec3uc.h

3.19 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.

- Vec3usi (const Vec3i &v)

  Construct a Vec3usi from a Vec3i.

3.19.1 Detailed Description

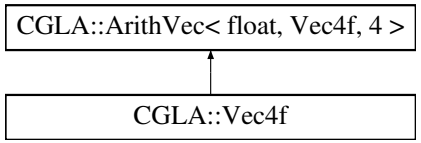
Unsigned short int 3D vector class. This class is mainly useful if we need a 3D int vector that takes up less room than a Vec3i but holds larger numbers than a Vec3c.

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

- Vec3usi.h

3.20 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.
- float length () const
  Compute Euclidean length.

3.20.1 Detailed Description

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

3.20.2 Member Function Documentation

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

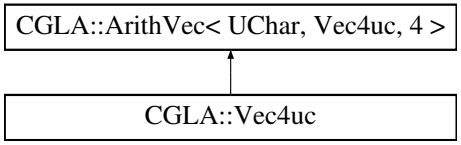
This function divides a vector (x,y,z,w) by w to obtain a new 4D vector where w=1.

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

- Vec4f.h

3.21 CGLA::Vec4uc Class Reference

Inheritance diagram for CGLA::Vec4uc::



Public Methods

- Vec4uc ()
  Construct 0 vector.
- Vec4uc (unsigned char a)
  Construct 0 vector.
- Vec4uc (UChar _a, UChar _b, UChar _c, UChar _d)
  Construct 4D uchar vector.
- Vec4uc (const Vec4f &v)
  Convert from float vector.

3.21.1 Detailed Description

4D unsigned char vector.

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

- Vec4uc.h