Jet_fitting_3_ref

Separate Build

December 20, 2007

Contents

Estimation of Local Differential Properties of Sampled Surfaces via Polynomial Fitting
Reference Manual

Marc Pouget and Frédéric Cazals

Concepts

Classes

Global Functions

The insert operator (*operator<<*) is overloaded for the class *Monge_form*.

0.1 Alphabetical List of Reference Pages

DataKernel

Definition

The concept DataKernel describes the set of requirements to be fulfilled by any class used to instantiate first template parameter of the class *Monge_via_jet_fitting<DataKernel,LocalKernel,KernelConverters,SvdTraits>*.

Types

DataKernel:: FT	The scalar type.
DataKernel:: Point_3	The point type.
DataKernel:: Vector_3	The vector type.

Operations

Only constructors (from 3 scalars and copy constructors) and access methods to coordinates *x()*, *y()*, *z()* are needed.

See Also

The *LocalKernel* concept.

KernelConverters<DataKernel, LocalKernel>

Definition

The concept KernelConverters<DataKernel, LocalKernel> describes the set of requirements to be fulfilled by any class used to instantiate third template parameter of the class *Monge_via_jet_fitting< DataKernel,LocalKernel,KernelConverters,SvdTraits>*. It enables convertions forth and back between number types, points and vectors of the DataKernel and LocalKernel.

Operations

LocalKernel::Point_3

Kc.D2L_converter(DataKernel::Point_3 p)

returns a *LocalKernel::Point_3* which coordinates are those of p converted to the *LocalKernel::FT*.

The same function is also defined for the number types and the vector types.

DataKernel::Point_3 *Kc.L2D_converter(LocalKernel::Point_3 p)*

returns a *DatalKernel::Point_3* which coordinates are those of p converted to the *DatalKernel::FT*.

The same function is also defined for the number types and the vector types.

See Also

The *LocalKernel* and the *DataKernel* concepts.

LocalKernel

Definition

The concept LocalKernel describes the set of requirements to be fulfilled by any class used to instantiate the second template parameter of the class *Monge_via_jet_fitting< DataKernel,LocalKernel,KernelConverters,SvdTraits>*.

This concept provides the geometric primitives used for the computations in the class *Monge_via_jet_fitting*.

Requirements

In the class *Monge_via_jet_fitting* the scalar type, *LocalKernel::FT*, must be the same as that of the *SvdTraits* concept : *SvdTraits::FT*.

Types

LocalKernel:: FT	The scalar type.
LocalKernel:: Point_3	The point type.
LocalKernel:: Vector_3	The vector type.
LocalKernel:: LKMatrix	For dimension 2 and 3 square matrices.
LocalKernel:: Aff_transformation	For 3d affine tranformation.

Operations

The scalar type *LocalKernel::FT* must be a field type with a square root.

FT *LK.Lsqrt(FT x)*

Only constructors (from 3 scalars and copy constructors) and access methods to coordinates *x()*, *y()*, *z()* are needed for the point and vector types.

MATRICES

Definition

An instance of data type *LKMatrix* is a matrix of variables of number type *FT*.

Types

LKMatrix::iterator : bidirectional iterator for accessing all components row-wise.

Creation

LKMatrix M(int n): creates an instance *M* of type *LKMatrix* of dimension $n \times n$ initialized to the zero matrix.

Operations

LKMatrix *LK.inverse(LKMatrix M, FT& D)*

returns the inverse matrix of M. More precisely, 1/D times the matrix returned is the inverse of M. Precondition: *determinant(M) != 0*. Precondition: M is square.

int *LK.sign_of_determinant(LKMatrix M)*

returns the sign of the determinant of M. Precondition: M is square.

Affine Transformations

Definition

The class *Aff_transformation* represents three-dimensional affine transformations.

Creation

Aff_transformation t(const FT &m00, const FT &m01, const FT &m02, const FT &m03, const FT &m10, const FT &m11, const FT &m12, const FT &m13, const FT &m20, const FT &m21, const FT &m22, const FT &m23);

introduces a general affine transformation; the matrix mij for i and j from 0 to 2 defines the scaling and rotational part of the transformation, while the vector $(m03, m13, m23)$ contains the translational part.

Aff_transformation t(const FT &m00, const FT &m01, const FT &m02, const FT &m10, const FT &m11, const FT &m12, const FT &m20, const FT &m21, const FT &m22);

introduces a general affine transformation without translational part.

Operations

Aff_transformation_3 t.operator (s);* composes two affine transformations.

Aff_transformation_3 t.inverse (); gives the inverse transformation.

Eigen Decomposition of a Symmetric Matrix

void *LK.eigen_symmetric(const FT *mat,*
const int n,
*FT *eigen_vectors,*
*FT *eigen_values)*

Computes eigenvalues and eigenvectors of a dimension n symmetric matrix. The matrix is given by the coefficients of its lower part row-wise. Eigenvalues are sorted in descending order, eigenvectors are sorted in accordance.

See Also

The *DataKernel* and *SvdTraits* concepts.

Class

CGAL::Monge_via_jet_fitting< DataKernel, LocalKernel, KernelConverters, SvdTraits>::Monge_form

Definition

The class *Monge_form* stores the Monge representation, i.e., the Monge coordinate system and the coefficients of the Monge form in this system.

Types

typedef typename DataKernel::FT

FT;

typedef typename DataKernel::Point_3

Point_3;

typedef typename DataKernel::Vector_3

Vector_3;

Creation

Monge_form monge_form; — default constructor.

Access Functions

Point_3 *monge_form.origin()* — Point on the fitted surface where differential quantities are computed.

The Monge basis is given by:

Vector_3 *monge_form.maximal_principal_direction()*
Vector_3 *monge_form.minimal_principal_direction()*
Vector_3 *monge_form.normal_direction()*

The Monge coefficients are given by:

FT *monge_form.principal_curvatures(size_t i)*

$i = 0$ for the maximum and $i = 1$ for the minimum.

FT *monge_form.third_order_coefficients(size_t i)*

$0 \leq i \leq 3$

FT *monge_form.fourth_order_coefficients(size_t i)*

$0 \leq i \leq 4$

Operations

void *monge_form.comply_wrt_given_normal(const Vector_3 given_normal)*

change principal basis and Monge coefficients so that the given_normal and the Monge normal make an acute angle. If given_normal.monge_normal < 0 then change the orientation: if $z = g(x, y)$ in the basis (d1,d2,n) then in the basis (d2,d1,-n) $z = h(x, y) = -g(y, x)$.

The usual output operator (*operator<<*) is overloaded for *Monge_form*, it gives the Monge coordinate system (the origin and an orthonormal basis) and the coefficients of the Monge form in this system.

See Also

Monge_via_jet_fitting

Class

CGAL::Monge_via_jet_fitting<DataKernel, LocalKernel, KernelConverters, SvdTraits>

Definition

The class *Monge_via_jet_fitting<DataKernel, LocalKernel, KernelConverters, SvdTraits>* is designed to perform the estimation of the local differential quantities at a given point. The point range is given by a pair of input iterators, and it is assumed that the point where the calculation is carried out is the point that the begin iterator refers to. The results are stored in an instance of the nested class *Monge_form*, the particular information returned depending on the degrees specified for the polynomial fitting and for the Monge form.

Parameters

The class *Monge_via_jet_fitting<DataKernel, LocalKernel, KernelConverters, SvdTraits>* has four template parameters. Parameter *DataKernel* provides the geometric classes and tools corresponding to the input points, and also members of the *Monge_form* class. Parameter *LocalKernel* provides the geometric classes and tools required by local computations. Parameter *KernelConverters* enables conversions of kernel geometric classes. Parameter *SvdTraits* features the linear algebra algorithm required by the fitting method.

Types

typedef DataKernel *Data_kernel;*
typedef LocalKernel *Local_kernel;*
typedef typename Local_kernel::FT

FT;

typedef typename Local_kernel::Vector_3

Vector_3;

typedef typename DataKernel::Vector_3

DVector_3;

Monge_via_jet_fitting<DataKernel, LocalKernel, KernelConverters, SvdTraits>:: Monge_form

see its specific the section.

Creation

Monge_via_jet_fitting<DataKernel, LocalKernel, KernelConverters, SvdTraits> monge_fitting;

default constructor

Operations

template <class InputIterator>
Monge_form *monge_fitting(InputIterator begin, InputIterator end, size_t d, size_t d')*

This operator performs all the computations. The N input points are given by the *InputIterator* parameters which value-type are *Data_kernel::Point_3*, d is the degree of the fitted polynomial, d' is the degree of the expected Monge coefficients.
Precondition: $N \geq N_d := (d+1)(d+2)/2, \ 1 \leq d' \leq \min(d,4)$.

template <class InputIterator>
Monge_form *monge_fitting.operator()(InputIterator begin,*
InputIterator end,
size_t d,
size_t d',
DVector_3 vx,
DVector_3 vy,
DVector_3 vz)

This operator performs the same computations as the former. The difference is that the coordinate system in which the fitting is performed is given by the orthonormal basis (vx, vy, vz).

FT *monge_fitting.condition_number()*

condition number of the linear fitting system.

std::pair<FT, Vector_3>
monge_fitting.pca_basis(size_t i)

pca eigenvalues and eigenvectors, the pca_basis has always 3 such pairs. Precondition : i ranges from 0 to 2.

See Also

Monge_form

Concept

SvdTraits

Definition

The concept SvdTraits describes the set of requirements to be fulfilled by any class used to instantiate the fourth template parameter of the class *Monge_via_jet_fitting<DataKernel,LocalKernel,KernelConverters,SvdTraits>*.

It describes the linear algebra types and algorithms needed by the class *Monge_via_jet_fitting*.

Requirements

The scalar type, *SvdTraits::FT*, must be the same as that of the *LocalKernel* concept : *LocalKernel::FT*.

Types

SvdTraits:: FT	The scalar type.
SvdTraits:: Vector	The vector type.
SvdTraits:: Matrix	The matrix type.

Operations

SvdTraits vector(size_t n); initialize all the elements of the vector to zero.

The type *Vector* has the access methods

size_t	*vector.size()*	
FT	*vector(size_t i)*	return the i^{th} entry, i from 0 to $size() - 1$.
void	*vector.set(size_t i, const FT value)*	
		set the i^{th} entry to *value*.

The type *Matrix* has the access methods

SvdTraits matrix(size_t n1, size_t n2); initialize all the entries of the matrix to zero.

size_t	*matrix.number_of_rows()*	
size_t	*matrix.number_of_columns()*	
FT	*matrix(size_t i, size_t j)*	
		return the entry at row i and column j, i from 0 to *number_of_rows - 1*, j from 0 to *number_of_columns - 1*.
void	*matrix.set(size_t i, size_t j, const FT value)*	
		set the entry at row i and column j to *value*.

The concept *SvdTraits* has a linear solver using a singular value decomposition algorithm.

FT *traits.solve(Matrix& M, Vector& B)*

Solves the system $MX = B$ (in the least square sense if M is not square) using a singular value decomposition and returns the condition number of M. The solution is stored in B.

See Also

LocalKernel

Index

Pages on which definitions are given are presented in **boldface**.