Combinatorial Pyramids Reference Manual

0.1

Generated by Doxygen 1.2.18

Tue Jun 24 14:58:23 2003

Contents

Chapter 1

Combinatorial Pyramids Hierarchical Index

1.1 Combinatorial Pyramids Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Combinatorial Pyramids Compound Index

2.1 Combinatorial Pyramids Compound List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

Combinatorial Pyramids File Index

3.1 Combinatorial Pyramids File List

Here is a list of all documented files with brief descriptions:

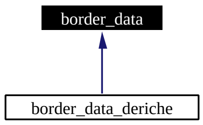
Chapter 4

Combinatorial Pyramids Class Documentation

4.1 border_data Class Reference

`#include <border_data.hh>`

Inheritance diagram for border_data:



Public Methods

- virtual void **init** (**combi_map_base** &, dart)=0

  A pur virtual function to initialize the border_value of the dart.

- virtual void **merge** (int, dart, dart)=0

  A pur virtual function to manage the border_value during the merge of two dart.

4.1.1 Detailed Description

model class to store a value associated to each dart. Generally associated to the corresponding boundary

4.1.2 Member Function Documentation

4.1.2.1 virtual void border_data::init (combi_map_base &, dart) `[pure virtual]`

A pur virtual function to initialize the border_value of the dart.

Implemented in **border_data_deriche** (p. 10).

4.1.2.2 virtual void border_data::merge (int, dart, dart) `[pure virtual]`

A pur virtual function to manage the border_value during the merge of two dart.

Implemented in **border_data_deriche** (p. 10).

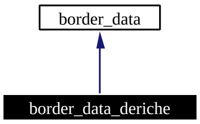
The documentation for this class was generated from the following file:

- **border_data.hh**

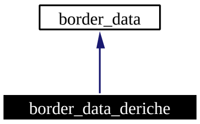
4.2 border_data_deriche Class Reference

`#include <border_data_deriche.hh>`

Inheritance diagram for border_data_deriche:



Collaboration diagram for border_data_deriche:



Public Methods

- **border_data_deriche** (image &I, double &d)

  Constructor with parameters, the number of darts.

- **border_data_deriche** (const border_data_deriche &)

  Copy constructor.

- virtual ~**border_data_deriche** ()

  Destructor.

- border_data_deriche & **operator=** (const border_data_deriche &)

  Reaffectation operator.

- **border_value** & **operator()** (long) const

  Access an element of dataset.

- void **del** (dart)

  Remove an element from the dataset.

- void **init** (**combi_map_base** &, dart)

  Function to initialize the **border_value** (p. 12) *of the dart.*

- void **merge** (int, dart, dart)

  Function to manage the **border_value** (p. 12) *during the merge of two dart.*

- float **give_max_gradient** ()

  Access to the maximum value of the gradient.

4.2.1 Detailed Description

One class which store the gradient of each vertex

4.2.2 Constructor & Destructor Documentation

4.2.2.1 border_data_deriche::border_data_deriche (image & *I*, double & *d*)

Constructor with parameters, the number of darts.

4.2.2.2 border_data_deriche::border_data_deriche (const border_data_deriche &)

Copy constructor.

4.2.2.3 border_data_deriche::~border_data_deriche () `[virtual]`

Destructor.

4.2.3 Member Function Documentation

4.2.3.1 void border_data_deriche::del (dart)

Remove an element from the dataset.

4.2.3.2 float border_data_deriche::give_max_gradient ()

Access to the maximum value of the gradient.

4.2.3.3 void border_data_deriche::init (combi_map_base &, dart) `[virtual]`

Function to initialize the **border_value** (p. 12) of the dart.

Implements **border_data** (p. 7).

4.2.3.4 void border_data_deriche::merge (int, dart, dart) `[virtual]`

Function to manage the **border_value** (p. 12) during the merge of two dart.

Implements **border_data** (p. 8).

4.2.3.5 border_data_deriche::border_value & border_data_deriche::operator() (long) const

Access an element of dataset.

4.2.3.6 border_data_deriche & border_data_deriche::operator= (const border_data_deriche &)

Reaffectation operator.

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

- **border_data_deriche.hh**
- border_data_deriche.cc

4.3 border_data_deriche::border_value Class Reference

#include <border_data_deriche.hh>

Public Methods

- **border_value** ()
  Constructor.
- **border_value** (const border_value &)
  Copy constructor.
- **border_value** (float)
  Constructor with parameters, the gradient value of the border.
- ~**border_value** ()
  Destructor.
- border_value & **operator=** (const border_value &)
  Reaffectation operator.
- void **reinit** (float, int)
  Reinitialiation function.
- float **give_gradient** ()
  Access to the sum of the gradient value of all the borders which were merged.
- int **give_cardinal** ()
  Access to the cardinal of the border (number of border merged with it).

4.3.1 Detailed Description

Class that hold all the information of one border (associated to one dart)

4.3.2 Constructor & Destructor Documentation

4.3.2.1 border_data_deriche::border_value::border_value ()

Constructor.

4.3.2.2 border_data_deriche::border_value::border_value (const border_value &)

Copy constructor.

4.3.2.3 border_data_deriche::border_value::border_value (float)

Constructor with parameters, the gradient value of the border.

4.3.2.4 border_data_deriche::border_value::~border_value () [inline]

Destructor.

4.3.3 Member Function Documentation

4.3.3.1 int border_data_deriche::border_value::give_cardinal ()

Access to the cardinal of the border (number of border merged with it).

4.3.3.2 float border_data_deriche::border_value::give_gradient ()

Access to the sum of the gradient value of all the borders which were merged.

4.3.3.3 border_data_deriche::border_value & border_data_deriche::border_-value::operator= (const border_value &)

Reaffectation operator.

4.3.3.4 void border_data_deriche::border_value::reinit (float, int)

Reinitialiation function.

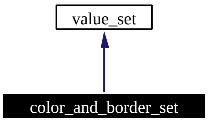
The documentation for this class was generated from the following files:

- **border_data_deriche.hh**
- border_data_deriche.cc

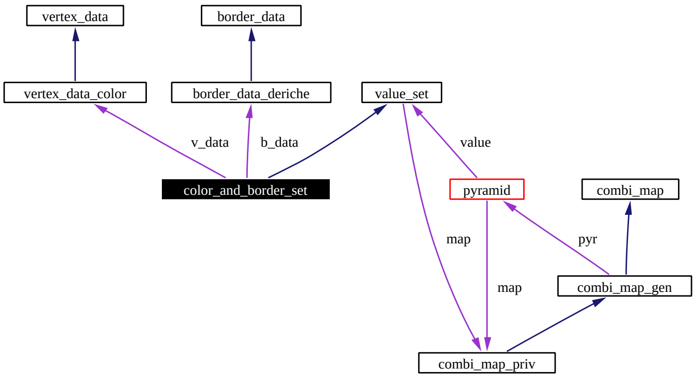
4.4 color_and_border_set Class Reference

`#include <color_and_border_set.hh>`

Inheritance diagram for color_and_border_set:



Collaboration diagram for color_and_border_set:



Public Methods

- void **init** (**combi_map_base** &)

  Function to initialize the data_set during the construction of the pyramid.

- void **contract_edge** (int, dart, dart, vertex, vertex)

  Function to manage the data when an edge will be contracted.

- void **remove_empty_self_loops** (dart, dart, vertex)

  Function to manage the data when an empty self-loop will be remove.

- void **remove_double_edges** (int, dart, dart)

  Function to manage the data when a double edge will be remove.

- void **update_vertex_set** (int, vertex, vertex)

  Function to manage the data when vertex set is updated.

4.4.1 Detailed Description

Class to manage color and border data of the pyramid

4.4.2 Member Function Documentation

4.4.2.1 void color_and_border_set::contract_edge (int, dart, dart, vertex, vertex) [virtual]

Function to manage the data when an edge will be contracted.

Reimplemented from **value_set** (p. 67).

4.4.2.2 void color_and_border_set::init (combi_map_base &) [virtual]

Function to initialize the data_set during the construction of the pyramid.

Implements **value_set** (p. 67).

4.4.2.3 void color_and_border_set::remove_double_edges (int, dart, dart) [virtual]

Function to manage the data when a double edge will be remove.

Reimplemented from **value_set** (p. 67).

4.4.2.4 void color_and_border_set::remove_empty_self_loops (dart, dart, vertex) [virtual]

Function to manage the data when an empty self-loop will be remove.

Reimplemented from **value_set** (p. 67).

4.4.2.5 void color_and_border_set::update_vertex_set (int, vertex, vertex) [virtual]

Function to manage the data when vertex set is updated.

Reimplemented from **value_set** (p. 67).

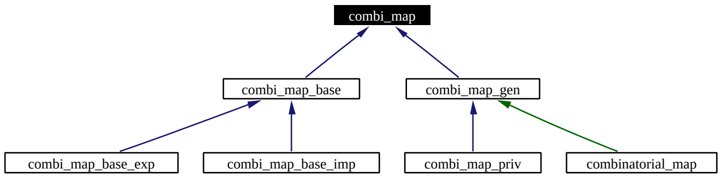
The documentation for this class was generated from the following files:

- **color_and_border_set.hh**
- color_and_border_set.cc

4.5 combi_map Class Reference

#include <combi_map.hh>

Inheritance diagram for combi_map:



Public Methods

- virtual dart **sigma** (dart) const=0

  A pur virtual function to access to the sigma of one dart of the combinatorial map.

- dart **sigma_1** (dart b)

  A pur virtual function which returns the dart d=sigma_1(b) such that sigma(d)=b.

- dart **phi** (dart b)
- virtual dart **alpha** (dart) const=0

  A pur virtual function to access to the alpha of one dart of the combinatorial map.

- virtual int **give_nb_edge** () const=0

  A pur virtual function that give the number of edge.

- virtual unsigned long **give_background** () const=0

  A pur virtual function that give the backround value.

4.5.1 Detailed Description

Model class for the combinatorial maps

4.5.2 Member Function Documentation

4.5.2.1 virtual dart combi_map::alpha (dart) const [pure virtual]

A pur virtual function to access to the alpha of one dart of the combinatorial map.

Implemented in **combi_map_base** (p. 19), **combi_map_gen** (p. 29), and **combi_map_priv** (p. 32).

4.5.2.2 virtual unsigned long combi_map::give_background () [pure virtual]

A pur virtual function that give the backround value.

Implemented in **combi_map_base_exp** (p. 22), **combi_map_base_imp** (p. 25), **combi_map_gen** (p. 29), and **combi_map_priv** (p. 32).

4.5.2.3 virtual int combi_map::give_nb_edge () `[pure virtual]`

A pur virtual function that give the number of edge.

Implemented in **combi_map_base_exp** (p. 23), **combi_map_base_imp** (p. 26), **combi_map_-gen** (p. 29), and **combi_map_priv** (p. 32).

4.5.2.4 dart combi_map::phi (dart *b*) `[inline]`

A pure virtual function which returns the phi successor of a dart sigma o alpha

Reimplemented in **combi_map_base** (p. 20), and **combi_map_base_exp** (p. 23).

4.5.2.5 virtual dart combi_map::sigma (dart) const `[pure virtual]`

A pur virtual function to access to the sigma of one dart of the combinatorial map.

Implemented in **combi_map_base** (p. 20), **combi_map_base_exp** (p. 23), **combi_map_base_-imp** (p. 26), **combi_map_gen** (p. 29), and **combi_map_priv** (p. 32).

4.5.2.6 dart combi_map::sigma_1 (dart *b*) `[inline]`

A pur virtual function which returns the dart d=sigma_1(b) such that sigma(d)=b.

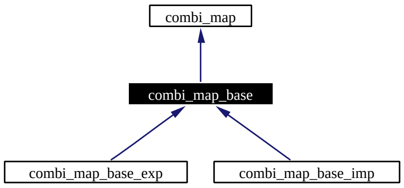
The documentation for this class was generated from the following file:

- **combi_map.hh**

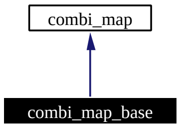
4.6 combi_map_base Class Reference

`#include <combi_map_base.hh>`

Inheritance diagram for combi_map_base:



Collaboration diagram for combi_map_base:



Public Methods

- virtual dart **sigma** (dart) const=0

  A pur virtual function to access to the sigma of one dart of the combinatorial map.

- dart **phi** (dart b)

  optimization of the phi function of the mother class.

- virtual dart **alpha** (dart b) const

  A pure virtual function to access to the alpha of one dart of the combinatorial map.

- virtual int **compute_index_orientation** (dart, dart)=0

  Function that compute the variation of orientation between two darts.

- virtual void **give_pixel** (dart, point &p) const=0

  A pure virtual function that give the coordinates of the pixel defined by the dart.

- virtual void **give_pointel** (dart, point &p) const=0

  A pure virtual function that give the coordinates of the vertex of the map defined by the dart.

- virtual **move_type** **give_move** (dart) const=0

  A pure virtual function that give the orientation of the dart.

- virtual void **init_combi_map_priv** (**combi_map_priv** &c_map) const=0

  A pure virtual function that initialize a **combinatorial_map** (p. 33) *from a combi_map_base.*

- virtual int **give_width** () const=0

A pure virtual function that give the width of the base map.

- virtual int **give_height** () const=0

  A pure virtual function that give the height of the base map.

4.6.1 Detailed Description

Model class for the combinatorial maps of the base of the pyramid

4.6.2 Member Function Documentation

4.6.2.1 virtual dart combi_map_base::alpha (dart *b*) const `[inline, virtual]`

A pure virtual function to access to the alpha of one dart of the combinatorial map.

Implements **combi_map** (p. 16).

4.6.2.2 virtual int combi_map_base::compute_index_orientation (dart, dart) `[pure virtual]`

Function that compute the variation of orientation between two darts.

Implemented in **combi_map_base_exp** (p. 22), and **combi_map_base_imp** (p. 25).

4.6.2.3 virtual int combi_map_base::give_height () `[pure virtual]`

A pure virtual function that give the height of the base map.

Implemented in **combi_map_base_exp** (p. 22), and **combi_map_base_imp** (p. 25).

4.6.2.4 virtual move_type combi_map_base::give_move (dart) const `[pure virtual]`

A pure virtual function that give the orientation of the dart.

Implemented in **combi_map_base_exp** (p. 23), and **combi_map_base_imp** (p. 26).

4.6.2.5 virtual void combi_map_base::give_pixel (dart, point & *p*) const `[pure virtual]`

A pure virtual function that give the coordinates of the pixel defined by the dart.

Implemented in **combi_map_base_exp** (p. 23), and **combi_map_base_imp** (p. 26).

4.6.2.6 virtual void combi_map_base::give_pointel (dart, point & *p*) const `[pure virtual]`

A pure virtual function that give the coordinates of the vertex of the map defined by the dart.

Implemented in **combi_map_base_exp** (p. 23), and **combi_map_base_imp** (p. 26).

4.6.2.7 virtual int combi_map_base::give_width () [pure virtual]

A pure virtual function that give the width of the base map.

Implemented in **combi_map_base_exp** (p. 23), and **combi_map_base_imp** (p. 26).

4.6.2.8 virtual void combi_map_base::init_combi_map_priv (combi_map_priv & *c_map*) const [pure virtual]

A pure virtual function that initialize a **combinatorial_map** (p. 33) from a combi_map_base.

Implemented in **combi_map_base_exp** (p. 23), and **combi_map_base_imp** (p. 26).

4.6.2.9 dart combi_map_base::phi (dart *b*) [inline]

optimization of the phi function of the mother class.

Reimplemented from **combi_map** (p. 17).

Reimplemented in **combi_map_base_exp** (p. 23).

4.6.2.10 virtual dart combi_map_base::sigma (dart) const [pure virtual]

A pur virtual function to access to the sigma of one dart of the combinatorial map.

Implements **combi_map** (p. 17).

Implemented in **combi_map_base_exp** (p. 23), and **combi_map_base_imp** (p. 26).

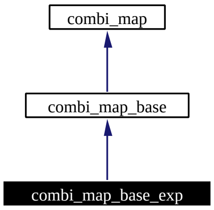
The documentation for this class was generated from the following file:

- **combi_map_base.hh**

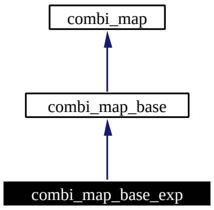
4.7 combi_map_base_exp Class Reference

`#include <combi_map_base_exp.hh>`

Inheritance diagram for combi_map_base_exp:



Collaboration diagram for combi_map_base_exp:



Public Methods

- **combi_map_base_exp** (int dimx, int dimy)

  Constructor.

- virtual **~combi_map_base_exp** ()

  Virtual destructor.

- dart **sigma** (dart) const

  Function to access to the sigma of one dart of the combinatorial map.

- dart **phi** (dart b)

  Function that compute sigma precedent of the alpha of the dart.

- int **give_nb_edge** () const

  Function that give the number of edge.

- unsigned long **give_background** () const

  Function that give the backround value.

- int **compute_index_orientation** (dart, dart)

  Function that compute the variation of orientation between two darts.

- void **give_pixel** (dart, point &p) const

Function that give the coordinates of the pixel defined by the dart.

- void **give_pointel** (dart, point &p) const
  Function that give the coordinates of the vertex of the map defined by the dart.
- **move_type give_move** (dart) const
  Function that give the orientation of the dart.
- void **init_combi_map_priv** (**combi_map_priv** &c_map) const
  Function that initialize a **combinatorial_map** (p. 33) *from a combi_map_base_exp.*
- int **give_width** () const
  Function that give the width of the base map.
- int **give_height** () const
  Function that give the height of the base map.

4.7.1 Detailed Description

Class that manage the combinatorial maps of the base of the pyramid by storing sigma permutation

4.7.2 Constructor & Destructor Documentation

4.7.2.1 combi_map_base_exp::combi_map_base_exp (int *dimx*, int *dimy*)

Constructor.

4.7.2.2 virtual combi_map_base_exp::~combi_map_base_exp () `[inline, virtual]`

Virtual destructor.

4.7.3 Member Function Documentation

4.7.3.1 int combi_map_base_exp::compute_index_orientation (dart, dart) `[virtual]`

Function that compute the variation of orientation between two darts.

Implements **combi_map_base** (p. 19).

4.7.3.2 unsigned long combi_map_base_exp::give_background () `[virtual]`

Function that give the backround value.

Implements **combi_map** (p. 16).

4.7.3.3 int combi_map_base_exp::give_height () `[virtual]`

Function that give the height of the base map.

Implements **combi_map_base** (p. 19).

4.7.3.4 move_type combi_map_base_exp::give_move (dart) const [virtual]

Function that give the orientation of the dart.

Implements **combi_map_base** (p. 19).

4.7.3.5 int combi_map_base_exp::give_nb_edge () [virtual]

Function that give the number of edge.

Implements **combi_map** (p. 17).

4.7.3.6 void combi_map_base_exp::give_pixel (dart, point & *p*) const [virtual]

Function that give the coordinates of the pixel defined by the dart.

Implements **combi_map_base** (p. 19).

4.7.3.7 void combi_map_base_exp::give_pointel (dart, point & *p*) const [virtual]

Function that give the coordinates of the vertex of the map defined by the dart.

Implements **combi_map_base** (p. 19).

4.7.3.8 int combi_map_base_exp::give_width () [virtual]

Function that give the width of the base map.

Implements **combi_map_base** (p. 20).

4.7.3.9 void combi_map_base_exp::init_combi_map_priv (combi_map_priv & *c_map*) const [virtual]

Function that initialize a **combinatorial_map** (p. 33) from a combi_map_base_exp.

Implements **combi_map_base** (p. 20).

4.7.3.10 dart combi_map_base_exp::phi (dart *b*)

Function that compute sigma precedent of the alpha of the dart.

Reimplemented from **combi_map_base** (p. 20).

4.7.3.11 dart combi_map_base_exp::sigma (dart) const [virtual]

Function to access to the sigma of one dart of the combinatorial map.

Implements **combi_map_base** (p. 20).

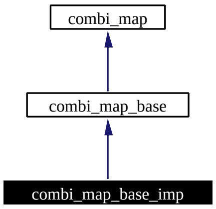
The documentation for this class was generated from the following files:

- **combi_map_base_exp.hh**
- combi_map_base_exp.cc

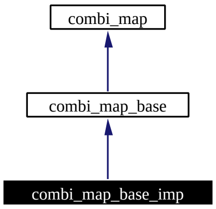
4.8 combi_map_base_imp Class Reference

`#include <combi_map_base_imp.hh>`

Inheritance diagram for combi_map_base_imp:



Collaboration diagram for combi_map_base_imp:



Public Methods

- **combi_map_base_imp** (int dimx, int dimy)

  Constructor.

- virtual **~combi_map_base_imp** ()

  Virtual destructor.

- dart **sigma** (dart) const

  Function to compute the sigma of one dart of the combinatorial map.

- int **give_nb_edge** () const

  Function that give the number of edge.

- unsigned long **give_background** () const

  Function that give the backround value.

- int **compute_index_orientation** (dart, dart)

  Function that compute the variation of orientation between two darts.

- void **give_pixel** (dart, point &p) const

  Function that give the coordinates of the pixel defined by the dart.

- void **give_pointel** (dart, point &p) const

Function that give the coordinates of the vertex of the map defined by the dart.

- **move_type give_move** (dart) const

  Function that give the orientation of the dart.

- void **init_combi_map_priv** (**combi_map_priv** &c_map) const

  Function that initialize a **combinatorial_map** (p. 33) *from a combi_map_base_imp.*

- int **give_width** () const

  Function that give the width of the base map.

- int **give_height** () const

  Function that give the height of the base map.

4.8.1 Detailed Description

Class that manage the combinatorial maps of the base of the pyramid without storing sigma permutation

4.8.2 Constructor & Destructor Documentation

4.8.2.1 combi_map_base_imp::combi_map_base_imp (int *dimx*, int *dimy*)

Constructor.

4.8.2.2 virtual combi_map_base_imp::~combi_map_base_imp () `[inline, virtual]`

Virtual destructor.

4.8.3 Member Function Documentation

4.8.3.1 int combi_map_base_imp::compute_index_orientation (dart, dart) `[virtual]`

Function that compute the variation of orientation between two darts.

Implements **combi_map_base** (p. 19).

4.8.3.2 unsigned long combi_map_base_imp::give_background () `[virtual]`

Function that give the backround value.

Implements **combi_map** (p. 16).

4.8.3.3 int combi_map_base_imp::give_height () `[virtual]`

Function that give the height of the base map.

Implements **combi_map_base** (p. 19).

4.8.3.4 move_type combi_map_base_imp::give_move (dart) const [virtual]

Function that give the orientation of the dart.

Implements **combi_map_base** (p. 19).

4.8.3.5 int combi_map_base_imp::give_nb_edge () [virtual]

Function that give the number of edge.

Implements **combi_map** (p. 17).

4.8.3.6 void combi_map_base_imp::give_pixel (dart, point & *p*) const [virtual]

Function that give the coordinates of the pixel defined by the dart.

Implements **combi_map_base** (p. 19).

4.8.3.7 void combi_map_base_imp::give_pointel (dart, point & *p*) const [virtual]

Function that give the coordinates of the vertex of the map defined by the dart.

Implements **combi_map_base** (p. 19).

4.8.3.8 int combi_map_base_imp::give_width () [virtual]

Function that give the width of the base map.

Implements **combi_map_base** (p. 20).

4.8.3.9 void combi_map_base_imp::init_combi_map_priv (combi_map_priv & *c_map*) const [virtual]

Function that initialize a **combinatorial_map** (p. 33) from a combi_map_base_imp.

Implements **combi_map_base** (p. 20).

4.8.3.10 dart combi_map_base_imp::sigma (dart) const [virtual]

Function to compute the sigma of one dart of the combinatorial map.

Implements **combi_map_base** (p. 20).

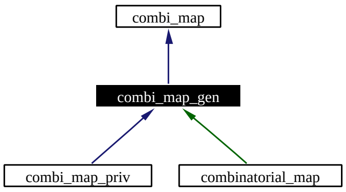
The documentation for this class was generated from the following files:

- **combi_map_base_imp.hh**
- combi_map_base_imp.cc

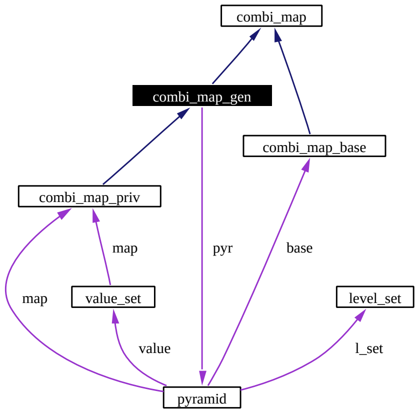
4.9 combi_map_gen Class Reference

`#include <combi_map_gen.hh>`

Inheritance diagram for combi_map_gen:



Collaboration diagram for combi_map_gen:



Public Methods

- **combi_map_gen** (**pyramid** ∗P)

  Constructor.

- virtual **~combi_map_gen** ()

  Virtual destructor.

- virtual dart **sigma** (dart d) const

  A pur virtual function to access to the sigma of one dart of the combinatorial map.

- virtual dart **alpha** (dart d) const

  Function to access to the alpha of one dart of the combinatorial map.

- virtual int **give_nb_edge** () const

  Function that give the number of edge.

- unsigned long **give_background** () const

  Function that give the backround value.

- int **give_nb_vertices** () const

  Function that give the number of edge.

Public Attributes

- **pyramid ∗ pyr**

  pointor to the pyramid from which the map was extracted

- tab_dart **sig**

  Tab of dart that hold the sigma of the map.

- tab_dart **alph**

  Tab of dart that hold the alpha of each dart in the map.

- tab_dart **m**

  Tab of dart that hold the corresponding vertex for each dart of the map.

- tab_label **m_1**

  Tab of vertex that hold on dart for each vertex of the map.

- list_label **vertex_set**

  List of the vertex of the map.

- int **nb_edge**

  Number of edge in the map.

- unsigned long **background**

  The backgroud value.

4.9.1 Detailed Description

Class for the generalised combinatorial maps

4.9.2 Constructor & Destructor Documentation

4.9.2.1 combi_map_gen::combi_map_gen (pyramid ∗ *P*) `[inline]`

Constructor.

4.9.2.2 virtual combi_map_gen::∼combi_map_gen () `[inline, virtual]`

Virtual destructor.

4.9.3 Member Function Documentation

4.9.3.1 virtual dart combi_map_gen::alpha (dart *d*) const [inline, virtual]

Function to access to the alpha of one dart of the combinatorial map.

Implements **combi_map** (p. 16).

Reimplemented in **combi_map_priv** (p. 32).

4.9.3.2 unsigned long combi_map_gen::give_background () const [inline, virtual]

Function that give the backround value.

Implements **combi_map** (p. 16).

Reimplemented in **combi_map_priv** (p. 32).

4.9.3.3 virtual int combi_map_gen::give_nb_edge () const [inline, virtual]

Function that give the number of edge.

Implements **combi_map** (p. 17).

Reimplemented in **combi_map_priv** (p. 32).

4.9.3.4 int combi_map_gen::give_nb_vertices () const [inline]

Function that give the number of edge.

4.9.3.5 virtual dart combi_map_gen::sigma (dart *d*) const [inline, virtual]

A pur virtual function to access to the sigma of one dart of the combinatorial map.

Implements **combi_map** (p. 17).

Reimplemented in **combi_map_priv** (p. 32).

4.9.4 Member Data Documentation

4.9.4.1 tab_dart combi_map_gen::alph

Tab of dart that hold the alpha of each dart in the map.

4.9.4.2 unsigned long combi_map_gen::background

The backgroud value.

4.9.4.3 tab_dart combi_map_gen::m

Tab of dart that hold the corresponding vertex for each dart of the map.

4.9.4.4 tab_label combi_map_gen::m_1

Tab of vertex that hold on dart for each vertex of the map.

4.9.4.5 int combi_map_gen::nb_edge

Number of edge in the map.

4.9.4.6 pyramid* combi_map_gen::pyr

pointor to the pyramid from which the map was extracted

4.9.4.7 tab_dart combi_map_gen::sig

Tab of dart that hold the sigma of the map.

4.9.4.8 list_label combi_map_gen::vertex_set

List of the vertex of the map.

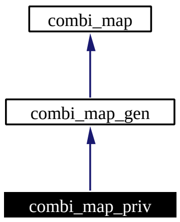
The documentation for this class was generated from the following file:

- **combi_map_gen.hh**

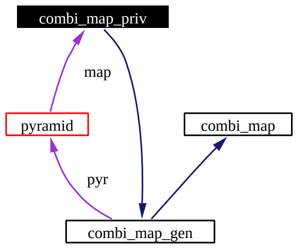
4.10 combi_map_priv Class Reference

`#include <combi_map_priv.hh>`

Inheritance diagram for combi_map_priv:



Collaboration diagram for combi_map_priv:



Public Methods

- **combi_map_priv** (**pyramid** &P)

    Constructor.

- virtual **~combi_map_priv** ()

    Virtual destructor.

- dart **sigma** (dart d) const

    Function to read the sigma of one dart of the combinatorial map.

- dart **alpha** (dart d) const

    Function that compute sigma precedent.

- int **give_nb_edge** () const

    Function that give the number of edge.

- unsigned long **give_background** () const

    Function that give the backround value.

4.10.1 Detailed Description

Class for the combinatorial maps used in the pyramid

4.10.2 Constructor & Destructor Documentation

4.10.2.1 combi_map_priv::combi_map_priv (pyramid & *P*) `[inline]`

Constructor.

4.10.2.2 virtual combi_map_priv::~combi_map_priv () `[inline, virtual]`

Virtual destructor.

4.10.3 Member Function Documentation

4.10.3.1 dart combi_map_priv::alpha (dart *d*) const `[inline, virtual]`

Function that compute sigma precedent.

Reimplemented from **combi_map_gen** (p. 29).

4.10.3.2 unsigned long combi_map_priv::give_background () const `[inline, virtual]`

Function that give the backround value.

Reimplemented from **combi_map_gen** (p. 29).

4.10.3.3 int combi_map_priv::give_nb_edge () const `[inline, virtual]`

Function that give the number of edge.

Reimplemented from **combi_map_gen** (p. 29).

4.10.3.4 dart combi_map_priv::sigma (dart *d*) const `[inline, virtual]`

Function to read the sigma of one dart of the combinatorial map.

Reimplemented from **combi_map_gen** (p. 29).

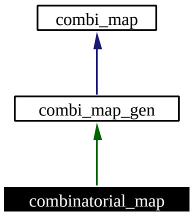
The documentation for this class was generated from the following file:

- **combi_map_priv.hh**

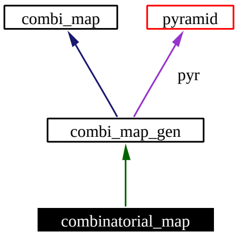
4.11 combinatorial_map Class Reference

`#include <combinatorial_map.hh>`

Inheritance diagram for combinatorial_map:



Collaboration diagram for combinatorial_map:



Public Methods

- **combinatorial_map** ()

  Constructor.

- **combinatorial_map** (**pyramid** &P)

  Constructor.

- virtual **~combinatorial_map** ()

  Virtual destructor.

- dart & **sigma** (dart d)

  Function to access to the sigma of one dart of the combinatorial map.

- dart & **alpha** (dart d)

  Function to access to the alpha of one dart of the combinatorial map.

- vertex **mu** (dart d) const

  Function to read the vertex of one dart of the combinatorial map.

- dart **mu_1** (vertex v) const

  Function to read one dart of one vertex of the combinatorial map.

- bool **is_open_parenthesis** (dart d) const

  Function that return true if the dart is an openparenthesis for an inclused area.

- dart **give_level** ()

  give the level of the combinatorial map

- unsigned int **give_background** ()

  give the background of the combinatorial map

- int **give_level** (dart d)

  Give the max level of the dart.

- int **give_width** ()

  Give the width of the base map.

- int **give_height** ()

  Give the height of the base map.

- dart **give_alpha_base** (dart d)

  Give the alpha of the dart in the base map.

- dart **give_sigma_base** (dart d)

  Give the sigma of the dart in the base map.

- void **give_point** (dart d, point &p)

  Give the departure point of the dart.

- void **give_pixel** (dart d, point &p)

  Give the departure pixel of the dart.

- **move_type give_move** (dart d)

  Give the orientation of the dart.

- vertex_iterator & **give_vertex_first** ()

  Give an iterator positioned on the first element of the vertex list.

- vertex_iterator & **give_vertex_last** ()

  Give an iterator positioned on the last element of the vertex list.

- vertex_iterator & **give_vertex_end** ()

  Give an iterator positioned on the element after the last of the vertex list.

- dart_iterator & **give_dart_first** ()

  Give an iterator positionned on the first dart of the **level_set** (p. 58) *for the given level.*

- dart_iterator & **give_dart_begin** ()

  Give an iterator positionned on the first dart of the given level.

- dart_iterator & **give_dart_last** ()

Give an iterator positionned after the last dart of the given level (loop : for(i=level_-set.first();i<level.end();i++).

- dart_iterator & **give_dart_end** ()

  Give an iterator positionned on the last dart of the given level (loop : for(i=level_-set.first();i<=level.last();i++).

- void **give_all_neighbors** (vertex, list_label &, list_label &)

  Function that put in the two lists the neighbors for the first list and the inclused area in the second.

- void **give_neighbors** (vertex, list_label &)

  Function that put in the list the neighbors.

- void **give_included** (vertex, list_label &)

  Function that put in the list the included neighbors.

- void **display** ()

  Function that print on the standard output the list of the vertex and the sigma permutation.

- void **draw_border** (image &, unsigned char ∗, draw_type=cm_draw_single)

  Function to draw the border on the picture.

- void **draw_vertex_border** (vertex, image &, unsigned char ∗, draw_type=cm_draw_-single)

  Function to draw the border of one area on the picture.

- dart **give_neighbor_dart** (vertex)

  Return a dart that is connected to a non included neighbor.

- **segment give_segment** (dart d)

  Function that give a segment corresponding to the dart.

4.11.1 Detailed Description

Class for the combinatorial maps

4.11.2 Constructor & Destructor Documentation

4.11.2.1 combinatorial_map::combinatorial_map () `[inline]`

Constructor.

4.11.2.2 combinatorial_map::combinatorial_map (pyramid & *P*) `[inline]`

Constructor.

4.11.2.3 virtual combinatorial_map::~combinatorial_map () [inline, virtual]

Virtual destructor.

4.11.3 Member Function Documentation

4.11.3.1 dart& combinatorial_map::alpha (dart *d*) [inline]

Function to access to the alpha of one dart of the combinatorial map.

4.11.3.2 void combinatorial_map::display ()

Function that print on the standard output the list of the vertex and the sigma permutation.

4.11.3.3 void combinatorial_map::draw_border (image &, unsigned char *, draw_type = cm_draw_single)

Function to draw the border on the picture.

4.11.3.4 void combinatorial_map::draw_vertex_border (vertex, image &, unsigned char *, draw_type = cm_draw_single)

Function to draw the border of one area on the picture.

4.11.3.5 void combinatorial_map::give_all_neighbors (vertex, list_label &, list_label &)

Function that put in the two lists the neighbors for the first list and the inclused area in the second.

4.11.3.6 dart combinatorial_map::give_alpha_base (dart *d*)

Give the alpha of the dart in the base map.

4.11.3.7 unsigned int combinatorial_map::give_background () [inline]

give the background of the combinatorial map

4.11.3.8 dart_iterator & combinatorial_map::give_dart_begin ()

Give an iterator positionned on the first dart of the given level.

4.11.3.9 dart_iterator & combinatorial_map::give_dart_end ()

Give an iterator positionned on the last dart of the given level (loop : for(i=level_set.first();i<=level.last();i++).

4.11.3.10 dart_iterator & combinatorial_map::give_dart_first ()

Give an iterator positionned on the first dart of the **level_set** (p. 58) for the given level.

4.11.3.11 dart_iterator & combinatorial_map::give_dart_last ()

Give an iterator positionned after the last dart of the given level (loop : for(i=level_set.first();i<level.end();i++).

4.11.3.12 int combinatorial_map::give_height ()

Give the height of the base map.

4.11.3.13 void combinatorial_map::give_included (vertex, list_label &)

Function that put in the list the included neighbors.

4.11.3.14 int combinatorial_map::give_level (dart *d*)

Give the max level of the dart.

4.11.3.15 dart combinatorial_map::give_level () `[inline]`

give the level of the combinatorial map

4.11.3.16 move_type combinatorial_map::give_move (dart *d*)

Give the orientation of the dart.

4.11.3.17 int combinatorial_map::give_neighbor_dart (vertex)

Return a dart that is connected to a non included neighbor.

4.11.3.18 void combinatorial_map::give_neighbors (vertex, list_label &)

Function that put in the list the neighbors.

4.11.3.19 void combinatorial_map::give_pixel (dart *d*, point & *p*)

Give the departure pixel of the dart.

4.11.3.20 void combinatorial_map::give_point (dart *d*, point & *p*)

Give the departure point of the dart.

4.11.3.21 segment combinatorial_map::give_segment (dart *d*) `[inline]`

Function that give a segment corresponding to the dart.

4.11.3.22 dart combinatorial_map::give_sigma_base (dart *d*)

Give the sigma of the dart in the base map.

4.11.3.23 vertex_iterator & combinatorial_map::give_vertex_end ()

Give an iterator positioned on the element after the last of the vertex list.

4.11.3.24 vertex_iterator & combinatorial_map::give_vertex_first ()

Give an iterator positioned on the first element of the vertex list.

4.11.3.25 vertex_iterator & combinatorial_map::give_vertex_last ()

Give an iterator positioned on the last element of the vertex list.

4.11.3.26 int combinatorial_map::give_width ()

Give the width of the base map.

4.11.3.27 bool combinatorial_map::is_open_parenthesis (dart *d*) const `[inline]`

Function that return true if the dart is an openparenthesis for an inclused area.

4.11.3.28 vertex combinatorial_map::mu (dart *d*) const `[inline]`

Function to read the vertex of one dart of the combinatorial map.

4.11.3.29 dart combinatorial_map::mu_1 (vertex *v*) const `[inline]`

Function to read one dart of one vertex of the combinatorial map.

4.11.3.30 dart& combinatorial_map::sigma (dart *d*) `[inline]`

Function to access to the sigma of one dart of the combinatorial map.

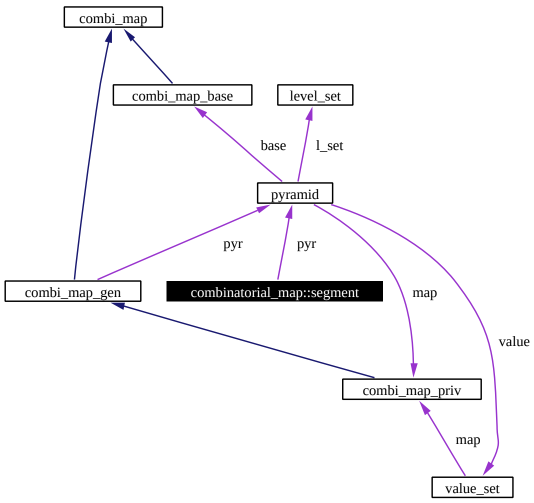
The documentation for this class was generated from the following files:

- **combinatorial_map.hh**
- combinatorial_map.cc

4.12 combinatorial_map::segment Class Reference

`#include <combinatorial_map.hh>`

Collaboration diagram for combinatorial_map::segment:



Public Methods

- **segment** ()

  Constructor.

- **segment** (dart, dart, int, **pyramid** ∗)

  Constructor.

- iterator & **first** ()

  first dart of the border

- iterator & **end** ()

  after the last dart of the border (loop : for(i=segment.first();i<segment.end();i++)

- iterator & **last** ()

  last dart of the border (loop : for(i=segment.first();i<=segment.last();i++)

4.12.1 Detailed Description

Class that manage the border of a region on the base map

4.12.2 Constructor & Destructor Documentation

4.12.2.1 combinatorial_map::segment::segment ()

Constructor.

4.12.2.2 combinatorial_map::segment::segment (dart, dart, int, pyramid *)

Constructor.

4.12.3 Member Function Documentation

4.12.3.1 combinatorial_map::segment::iterator & combinatorial_map::segment::end ()

after the last dart of the border (loop : for(i=segment.first();i<segment.end();i++)

4.12.3.2 combinatorial_map::segment::iterator & combinatorial_map::segment::first ()

first dart of the border

4.12.3.3 combinatorial_map::segment::iterator & combinatorial_map::segment::last ()

last dart of the border (loop : for(i=segment.first();i<=segment.last();i++)

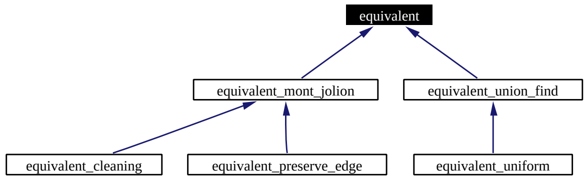
The documentation for this class was generated from the following files:

- **combinatorial_map.hh**
- combinatorial_map.cc

4.13 equivalent Class Reference

`#include <equivalent.hh>`

Inheritance diagram for equivalent:



Public Methods

- virtual bool **lambda** (int, vertex, vertex, dart, dart)=0

    Pure virtual function to know if the two vertices must be merged.

4.13.1 Detailed Description

Model class used to determine if two vertices are equivalent

4.13.2 Member Function Documentation

4.13.2.1 virtual bool equivalent::lambda (int, vertex, vertex, dart, dart) `[pure virtual]`

Pure virtual function to know if the two vertices must be merged.

Implemented in **equivalent_cleaning** (p. 43), **equivalent_mont_jolion** (p. 45), **equivalent_preserve_edge** (p. 47), **equivalent_uniform** (p. 49), and **equivalent_union_find** (p. 51).

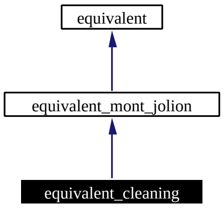
The documentation for this class was generated from the following file:

- **equivalent.hh**

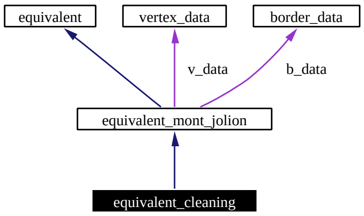
4.14 equivalent_cleaning Class Reference

`#include <equivalent_cleaning.hh>`

Inheritance diagram for equivalent_cleaning:



Collaboration diagram for equivalent_cleaning:



Public Methods

- **equivalent_cleaning** (const **vertex_data_color** &v, const **border_data_deriche** &b)

  Constructor.

- bool **lambda** (int, vertex, vertex, dart, dart)

  Function to know if the two vertices must be merged.

- double **value** (vertex)
- double **interest** (int, vertex, vertex, dart, dart)

4.14.1 Detailed Description

Model class used to determine and remove all the small vertices

4.14.2 Constructor & Destructor Documentation

4.14.2.1 equivalent_cleaning::equivalent_cleaning (const vertex_data_color & *v*, const border_data_deriche & *b*) `[inline]`

Constructor.

4.14.3 Member Function Documentation

4.14.3.1 double equivalent_cleaning::interest (int, vertex, vertex, dart, dart) [virtual]

Function to compute the merging interset of between two vertices That will be minimized during the reduction process

Implements **equivalent_mont_jolion** (p. 45).

4.14.3.2 bool equivalent_cleaning::lambda (int, vertex, vertex, dart, dart) [virtual]

Function to know if the two vertices must be merged.

Implements **equivalent_mont_jolion** (p. 45).

4.14.3.3 double equivalent_cleaning::value (vertex) [virtual]

Function to compute the surviving criterion of a vertex That will be minimized during the reduction process

Implements **equivalent_mont_jolion** (p. 45).

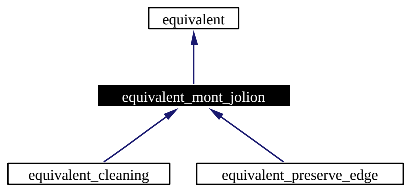
The documentation for this class was generated from the following files:

- **equivalent_cleaning.hh**
- equivalent_cleaning.cc

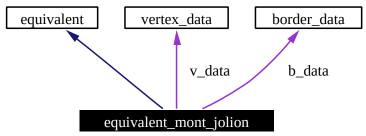
4.15 equivalent_mont_jolion Class Reference

`#include <equivalent_mont_jolion.hh>`

Inheritance diagram for equivalent_mont_jolion:



Collaboration diagram for equivalent_mont_jolion:



Public Methods

- **equivalent_mont_jolion** (const **vertex_data** &v, const **border_data** &b)

  Constructor.

- virtual ∼**equivalent_mont_jolion** ()

  Virtual destructor.

- virtual bool **lambda** (int, vertex, vertex, dart, dart)=0

  Pure virtual function to know if the two vertices must be merged.

- virtual double **value** (vertex)=0
- virtual double **interest** (int, vertex, vertex, dart, dart)=0

4.15.1 Detailed Description

Model class used to determine if two vertices are equivalent

4.15.2 Constructor & Destructor Documentation

4.15.2.1 equivalent_mont_jolion::equivalent_mont_jolion (const vertex_data & *v*, const border_data & *b*) `[inline]`

Constructor.

4.15.2.2 virtual equivalent_mont_jolion::~equivalent_mont_jolion () [inline, virtual]

Virtual destructor.

4.15.3 Member Function Documentation

4.15.3.1 virtual double equivalent_mont_jolion::interest (int, vertex, vertex, dart, dart) [pure virtual]

Pure virtual function to compute the merging interset of between two vertices That will be minimized during the reduction process

Implemented in **equivalent_cleaning** (p. 43), and **equivalent_preserve_edge** (p. 47).

4.15.3.2 virtual bool equivalent_mont_jolion::lambda (int, vertex, vertex, dart, dart) [pure virtual]

Pure virtual function to know if the two vertices must be merged.

Implements **equivalent** (p. 41).

Implemented in **equivalent_cleaning** (p. 43), and **equivalent_preserve_edge** (p. 47).

4.15.3.3 virtual double equivalent_mont_jolion::value (vertex) [pure virtual]

Pure virtual function to compute the surviving criterion of a vertex That will be minimized during the reduction process

Implemented in **equivalent_cleaning** (p. 43), and **equivalent_preserve_edge** (p. 47).

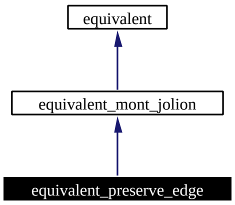
The documentation for this class was generated from the following file:

- **equivalent_mont_jolion.hh**

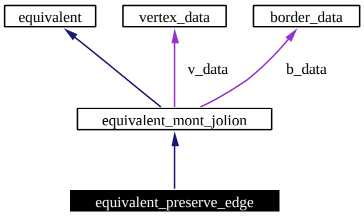
4.16 equivalent_preserve_edge Class Reference

`#include <equivalent_preserve_edge.hh>`

Inheritance diagram for equivalent_preserve_edge:



Collaboration diagram for equivalent_preserve_edge:



Public Methods

- **equivalent_preserve_edge** (const **vertex_data_color** &v, const **border_data_deriche** &b)

  Constructor.

- bool **lambda** (int, vertex, vertex, dart, dart)

  Function to know if the two vertices must be merged.

- double **value** (vertex)
- double **interest** (int, vertex, vertex, dart, dart)

4.16.1 Detailed Description

Model class used to determine if two vertices are equivalent

4.16.2 Constructor & Destructor Documentation

4.16.2.1 equivalent_preserve_edge::equivalent_preserve_edge (const vertex_data_color & *v*, const border_data_deriche & *b*) `[inline]`

Constructor.

4.16.3 Member Function Documentation

4.16.3.1 double equivalent_preserve_edge::interest (int, vertex, vertex, dart, dart) [virtual]

Function to compute the merging interset of between two vertices That will be minimized during the reduction process

Implements **equivalent_mont_jolion** (p. 45).

4.16.3.2 bool equivalent_preserve_edge::lambda (int, vertex, vertex, dart, dart) [virtual]

Function to know if the two vertices must be merged.

Implements **equivalent_mont_jolion** (p. 45).

4.16.3.3 double equivalent_preserve_edge::value (vertex) [virtual]

Function to compute the surviving criterion of a vertex That will be minimized during the reduction process

Implements **equivalent_mont_jolion** (p. 45).

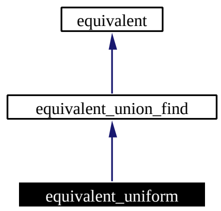
The documentation for this class was generated from the following files:

- **equivalent_preserve_edge.hh**
- equivalent_preserve_edge.cc

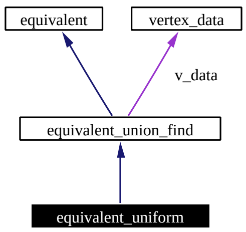
4.17 equivalent_uniform Class Reference

`#include <equivalent_uniform.hh>`

Inheritance diagram for equivalent_uniform:



Collaboration diagram for equivalent_uniform:



Public Methods

- **equivalent_uniform** (const **vertex_data_color** &v)

  Constructor.

- bool **lambda** (int, vertex, vertex, dart, dart)

  Function to know if the two vertices must be merged.

4.17.1 Detailed Description

Model class used to determine if two vertices are equivalent

4.17.2 Constructor & Destructor Documentation

4.17.2.1 equivalent_uniform::equivalent_uniform (const vertex_data_color & *v*) `[inline]`

Constructor.

4.17.3 Member Function Documentation

4.17.3.1 bool equivalent_uniform::lambda (int, vertex, vertex, dart, dart) [virtual]

Function to know if the two vertices must be merged.

Implements **equivalent_union_find** (p. 51).

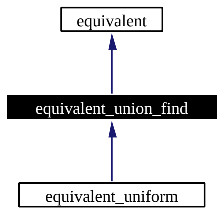
The documentation for this class was generated from the following files:

- **equivalent_uniform.hh**
- equivalent_uniform.cc

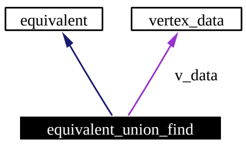
4.18 equivalent_union_find Class Reference

`#include <equivalent_union_find.hh>`

Inheritance diagram for equivalent_union_find:



Collaboration diagram for equivalent_union_find:



Public Methods

- **equivalent_union_find** (const **vertex_data** &v)
  Constructor.
- virtual **~equivalent_union_find** ()
  Virtual destructor.
- virtual bool **lambda** (int, vertex, vertex, dart, dart)=0
  Pure virtual function to know if the two vertices must be merged.

4.18.1 Detailed Description

Model class used to determine if two vertices are equivalent

4.18.2 Constructor & Destructor Documentation

4.18.2.1 equivalent_union_find::equivalent_union_find (const vertex_data & *v*) `[inline]`

Constructor.

4.18.2.2 virtual equivalent_union_find::~equivalent_union_find () `[inline, virtual]`

Virtual destructor.

4.18.3 Member Function Documentation

4.18.3.1 virtual bool equivalent_union_find::lambda (int, vertex, vertex, dart, dart) [pure virtual]

Pure virtual function to know if the two vertices must be merged.

Implements **equivalent** (p. 41).

Implemented in **equivalent_uniform** (p. 49).

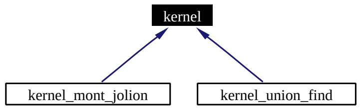
The documentation for this class was generated from the following file:

- **equivalent_union_find.hh**

4.19 kernel Class Reference

`#include <kernel.hh>`

Inheritance diagram for kernel:



Public Methods

- **kernel** ()

  Constructor.

- virtual **~kernel** ()

  Destructor.

- virtual void **init** (int lev=1)

  Virtual function to initialize the data befor the computation of the kernel.

- virtual void **compute_kernel** (**combi_map_priv** &)=0
- virtual bool **continue_computation** ()=0
- bool **is_marked** (dart)

  Function to know if a dart is a contracted one.

- int **cardinal** ()

  Give the number of survivor.

- virtual vertex **father** (vertex)

  Access to the father of one vertex.

- virtual vertex **find_compress** (vertex)

  Give the father of one vertex.

4.19.1 Detailed Description

Model class to compute and hold the survivor kernel of the combinatorial map

4.19.2 Constructor & Destructor Documentation

4.19.2.1 kernel::kernel () `[inline]`

Constructor.

4.19.2.2 virtual kernel::~kernel () `[inline, virtual]`

Destructor.

4.19.3 Member Function Documentation

4.19.3.1 int kernel::cardinal ()

Give the number of survivor.

4.19.3.2 virtual void kernel::compute_kernel (combi_map_priv &) `[pure virtual]`

Function to compute the kernel of the combinatorial map used in a do-compute_kernel while(**continue_computation()** (p. 53))

Implemented in **kernel_mont_jolion** (p. 55), and **kernel_union_find** (p. 57).

4.19.3.3 virtual bool kernel::continue_computation () `[pure virtual]`

Function to know if the compute_kernel function must be iterate an other time used in a do-compute_kernel while(**continue_computation()** (p. 53))

Implemented in **kernel_mont_jolion** (p. 55), and **kernel_union_find** (p. 57).

4.19.3.4 vertex kernel::father (vertex) `[virtual]`

Access to the father of one vertex.

4.19.3.5 vertex kernel::find_compress (vertex) `[virtual]`

Give the father of one vertex.

4.19.3.6 void kernel::init (int *lev* = 1) `[virtual]`

Virtual function to initialize the data befor the computation of the kernel.

4.19.3.7 bool kernel::is_marked (dart)

Function to know if a dart is a contracted one.

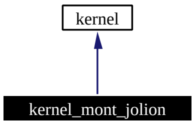
The documentation for this class was generated from the following files:

- **kernel.hh**
- kernel.cc

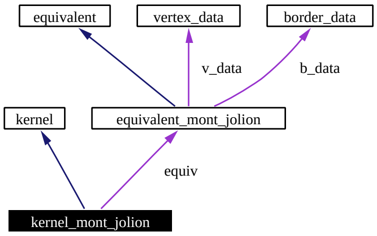
4.20 kernel_mont_jolion Class Reference

`#include <kernel_mont_jolion.hh>`

Inheritance diagram for kernel_mont_jolion:



Collaboration diagram for kernel_mont_jolion:



Public Methods

- **kernel_mont_jolion** (**equivalent_mont_jolion** &equi)

  Constructor.

- virtual ~**kernel_mont_jolion** ()

  Destructor.

- void **compute_kernel** (**combi_map_priv** &)

  Function to compute the kernel of the combinatorial map.

- bool **continue_computation** ()

  Function to know if the compute_kernel function must be iterate an other time.

4.20.1 Detailed Description

Model class to compute and hold the survivor kernel of the combinatorial map

4.20.2 Constructor & Destructor Documentation

4.20.2.1 kernel_mont_jolion::kernel_mont_jolion (equivalent_mont_jolion & *equi*) `[inline]`

Constructor.

4.20.2.2 virtual kernel_mont_jolion::~kernel_mont_jolion () [inline, virtual]

Destructor.

4.20.3 Member Function Documentation

4.20.3.1 void kernel_mont_jolion::compute_kernel (combi_map_priv &) [virtual]

Function to compute the kernel of the combinatorial map.

Implements **kernel** (p. 53).

4.20.3.2 bool kernel_mont_jolion::continue_computation () [virtual]

Function to know if the compute_kernel function must be iterate an other time.

Implements **kernel** (p. 53).

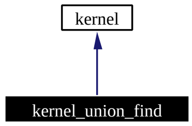
The documentation for this class was generated from the following files:

- **kernel_mont_jolion.hh**
- kernel_mont_jolion.cc

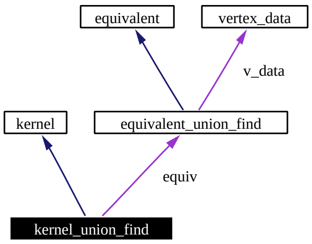
4.21 kernel_union_find Class Reference

`#include <kernel_union_find.hh>`

Inheritance diagram for kernel_union_find:



Collaboration diagram for kernel_union_find:



Public Methods

- **kernel_union_find** (**equivalent_union_find** &equi)

  Constructor.

- virtual **~kernel_union_find** ()

  Destructor.

- void **compute_kernel** (**combi_map_priv** &)
- bool **continue_computation** ()

4.21.1 Detailed Description

Model class to compute and hold the survivor kernel of the combinatorial map

4.21.2 Constructor & Destructor Documentation

4.21.2.1 kernel_union_find::kernel_union_find (equivalent_union_find & *equi*) `[inline]`

Constructor.

4.21.2.2 virtual kernel_union_find::~kernel_union_find () `[inline, virtual]`

Destructor.

4.21.3 Member Function Documentation

4.21.3.1 void kernel_union_find::compute_kernel (combi_map_priv &) [virtual]

Function to compute the kernel of the combinatorial map used in a do-compute_kernel while(**continue_computation()** (p. 57))

Implements **kernel** (p. 53).

4.21.3.2 bool kernel_union_find::continue_computation () [virtual]

Function to know if the compute_kernel function must be iterate an other time used in a do-compute_kernel while(**continue_computation()** (p. 57))

Implements **kernel** (p. 53).

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

- **kernel_union_find.hh**
- kernel_union_find.cc

4.22 level_set Class Reference

`#include <level_set.hh>`

Public Methods

- **level_set** (const level_set &)

  Copy constructor.

- **level_set** (int)

  Constructor with parameters, the number of darts.

- **level_set** ()

  Constructor.

- ∼**level_set** ()

  Destructor.

- level_set & **operator=** (const level_set &)

  Reaffectation operator.

- int **load** (ifstream &in)

  Set the level_set from a flow.

- int **save** (ofstream &out)

  Save the level_set into a flow.

- int **give_max_level** ()

  Give the maximum level.

- void **new_level** ()

  Add a new level to the level_set.

- int **give_level** (dart)

  Give the level of the dart.

- iterator & **first** (int)

  first dart of the level_set

- iterator & **begin** (int)

  first dart of the level

- iterator & **end** (int)

  after the last dart of the level (loop : for(i=level_set.first();i<level.end();i++)

- iterator & **last** (int)

  last dart of the level (loop : for(i=level_set.first();i<=level.last();i++)

4.22.1 Detailed Description

Encoding of the level of each dart in the pyramid *

4.22.2 Constructor & Destructor Documentation

4.22.2.1 level_set::level_set (const level_set &)

Copy constructor.

4.22.2.2 level_set::level_set (int)

Constructor with parameters, the number of darts.

4.22.2.3 level_set::level_set () `[inline]`

Constructor.

4.22.2.4 level_set::~level_set ()

Destructor.

4.22.3 Member Function Documentation

4.22.3.1 level_set::iterator & level_set::begin (int)

first dart of the level

4.22.3.2 level_set::iterator & level_set::end (int)

after the last dart of the level (loop : for(i=level_set.first();i<level.end();i++)

4.22.3.3 level_set::iterator & level_set::first (int)

first dart of the level_set

4.22.3.4 int level_set::give_level (dart)

Give the level of the dart.

4.22.3.5 int level_set::give_max_level ()

Give the maximum level.

4.22.3.6 level_set::iterator & level_set::last (int)

last dart of the level (loop : for(i=level_set.first();i<=level.last();i++)

4.22.3.7 int level_set::load (ifstream & *in*)

Set the level_set from a flow.

4.22.3.8 void level_set::new_level ()

Add a new level to the level_set.

4.22.3.9 level_set & level_set::operator= (const level_set &)

Reaffectation operator.

4.22.3.10 int level_set::save (ofstream & *out*)

Save the level_set into a flow.

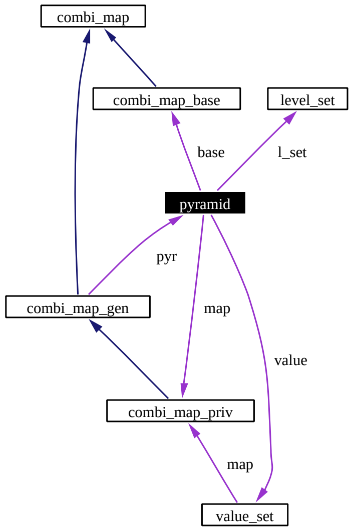
The documentation for this class was generated from the following files:

- **level_set.hh**
- level_set.cc

4.23 pyramid Class Reference

`#include <pyramid.hh>`

Collaboration diagram for pyramid:



Public Types

- enum **construction**

  Enum used to determine if the map of the base must be implicit or explicit.

Public Methods

- **pyramid** (int dimx, int dimy, **value_set** &val, **construction** c=explicit_base)

  Constructor.

- **pyramid** (char ∗, **construction** c=explicit_base)

  Constructor.

- **pyramid** (ifstream &, **construction** c=explicit_base)

  Constructor.

- **~pyramid** ()

  Destructor.

- void **compute_level** (**kernel** &)

  Fonction to generate one level of the pyramid.

- void **compute_pyramid** (**kernel** &)

  Fonction to generate all the level of the pyramid.

- int **save** (const char ∗)

  Save pyramid into a flow.

- void **give_combinatorial_map** (int, **combinatorial_map** &)

  Compute the combinatorial map of one level of the pyramid.

- int **give_max_level** ()

  Give the maximum level of the pyramid.

- dart **give_alpha_base** (dart d)

  Give the alpha of the dart on the base map.

- dart **give_sigma_base** (dart d)

  Give the sigma of the dart on the base map.

- **move_type give_move_base** (dart d)

  Give the move of the dart on the base map.

- int **give_level** (dart d)

  Give the max level of the dart d.

- void **give_pointel** (dart d, point &p)

  Give the departure point of the dart.

- void **give_pixel** (dart d, point &p)

  Give the initial pixel of the vertex describe by the dart in the base map.

- int **compute_index_orientation** (dart d1, dart d2)

  Function that compute the variation of orientation between two darts orientation.

- dart_iterator & **first_dart** (int l)

  Give an iterator positionned on the first dart of the **level_set** (p. 58) *for the given level.*

- dart_iterator & **begin_dart** (int l)

  Give an iterator positionned on the first dart of the given level.

- dart_iterator & **end_dart** (int l)

  Give an iterator positionned after the last dart of the given level (loop : for(i=level_-set.first();i<level.end();i++).

- dart_iterator & **last_dart** (int l)

  Give an iterator positionned on the last dart of the given level (loop : for(i=level_-set.first();i<=level.last();i++).

- int **give_width** ()

  Give the width of the base of the pyramid.

- int **give_height** ()

  Give the height of the base of the pyramid.

- dart **give_next_border_dart** (dart, int)

  Fonction to compute the next dart of the border for the given dart of the given level;.

4.23.1 Detailed Description

Class that will manage the combinatorial pyramid

4.23.2 Member Enumeration Documentation

4.23.2.1 enum pyramid::construction

Enum used to determine if the map of the base must be implicit or explicit.

4.23.3 Constructor & Destructor Documentation

4.23.3.1 pyramid::pyramid (int *dimx*, int *dimy*, value_set & *val*, construction *c* = explicit_base)

Constructor.

4.23.3.2 pyramid::pyramid (char *, construction *c* = explicit_base)

Constructor.

4.23.3.3 pyramid::pyramid (ifstream &, construction *c* = explicit_base)

Constructor.

4.23.3.4 pyramid::~pyramid ()

Destructor.

4.23.4 Member Function Documentation

4.23.4.1 dart_iterator& pyramid::begin_dart (int *l*) `[inline]`

Give an iterator positionned on the first dart of the given level.

4.23.4.2 int pyramid::compute_index_orientation (dart *d1*, dart *d2*) `[inline]`

Function that compute the variation of orientation between two darts orientation.

4.23.4.3 void pyramid::compute_level (kernel &)

Fonction to generate one level of the pyramid.

4.23.4.4 void pyramid::compute_pyramid (kernel &)

Fonction to generate all the level of the pyramid.

4.23.4.5 dart_iterator& pyramid::end_dart (int *l*) [inline]

Give an iterator positionned after the last dart of the given level (loop : for(i=level_set.first();i<level.end();i++).

4.23.4.6 dart_iterator& pyramid::first_dart (int *l*) [inline]

Give an iterator positionned on the first dart of the **level_set** (p. 58) for the given level.

4.23.4.7 dart pyramid::give_alpha_base (dart *d*) [inline]

Give the alpha of the dart on the base map.

4.23.4.8 void pyramid::give_combinatorial_map (int, combinatorial_map &)

Compute the combinatorial map of one level of the pyramid.

4.23.4.9 int pyramid::give_height () [inline]

Give the height of the base of the pyramid.

4.23.4.10 int pyramid::give_level (dart *d*) [inline]

Give the max level of the dart d.

4.23.4.11 int pyramid::give_max_level ()

Give the maximum level of the pyramid.

4.23.4.12 move_type pyramid::give_move_base (dart *d*) [inline]

Give the move of the dart on the base map.

4.23.4.13 dart pyramid::give_next_border_dart (dart, int)

Fonction to compute the next dart of the border for the given dart of the given level;.

4.23.4.14 void pyramid::give_pixel (dart *d*, point & *p*) [inline]

Give the initial pixel of the vertex describe by the dart in the base map.

4.23.4.15 void pyramid::give_pointel (dart *d*, point & *p*) [inline]

Give the departure point of the dart.

4.23.4.16 dart pyramid::give_sigma_base (dart *d*) [inline]

Give the sigma of the dart on the base map.

4.23.4.17 int pyramid::give_width () [inline]

Give the width of the base of the pyramid.

4.23.4.18 dart_iterator& pyramid::last_dart (int *l*) [inline]

Give an iterator positionned on the last dart of the given level (loop : for(i=level_set.first();i<=level.last();i++).

4.23.4.19 int pyramid::save (const char *)

Save pyramid into a flow.

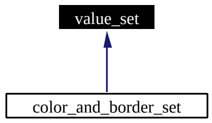
The documentation for this class was generated from the following files:

- **pyramid.hh**
- pyramid.cc

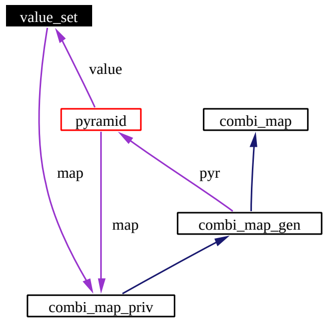
4.24 value_set Class Reference

`#include <value_set.hh>`

Inheritance diagram for value_set:



Collaboration diagram for value_set:



Public Methods

- void **set_combinatorial_map** (**combi_map_priv** &c_map)

    Function that will be use to update the pointor on the **combinatorial_map** (p. 33).

- virtual void **init** (**combi_map_base** &)=0

    A pure virtual function to initialize the data_set during the construction of the pyramid.

- virtual void **contract_edge** (int level, dart d1, dart d2, vertex v1, vertex v2)

    A virtual function to manage the data when an edge will be contracted.

- virtual void **remove_empty_self_loops** (dart d1, dart d2, vertex v)

    A virtual function to manage the data when an empty self-loop will be remove.

- virtual void **remove_double_edges** (int level, dart d1, dart d2)

    A virtual function to manage the data when a double edge will be remove.

- virtual void **update_vertex_set** (int, vertex, vertex)

    A virtual function to manage the data when vertex set is updated.

4.24.1 Detailed Description

Model class to manage the value data of the pyramid

4.24.2 Member Function Documentation

4.24.2.1 virtual void value_set::contract_edge (int *level*, dart *d1*, dart *d2*, vertex *v1*, vertex *v2*) `[inline, virtual]`

A virtual function to manage the data when an edge will be contracted.

Reimplemented in **color_and_border_set** (p. 15).

4.24.2.2 virtual void value_set::init (combi_map_base &) `[pure virtual]`

A pure virtual function to initialize the data_set during the construction of the pyramid.

Implemented in **color_and_border_set** (p. 15).

4.24.2.3 virtual void value_set::remove_double_edges (int *level*, dart *d1*, dart *d2*) `[inline, virtual]`

A virtual function to manage the data when a double edge will be remove.

Reimplemented in **color_and_border_set** (p. 15).

4.24.2.4 virtual void value_set::remove_empty_self_loops (dart *d1*, dart *d2*, vertex *v*) `[inline, virtual]`

A virtual function to manage the data when an empty self-loop will be remove.

Reimplemented in **color_and_border_set** (p. 15).

4.24.2.5 void value_set::set_combinatorial_map (combi_map_priv & *c_map*) `[inline]`

Function that will be use to update the pointor on the **combinatorial_map** (p. 33).

4.24.2.6 virtual void value_set::update_vertex_set (int, vertex, vertex) `[inline, virtual]`

A virtual function to manage the data when vertex set is updated.

Reimplemented in **color_and_border_set** (p. 15).

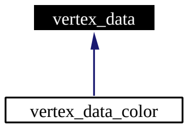
The documentation for this class was generated from the following file:

- **value_set.hh**

4.25 vertex_data Class Reference

`#include <vertex_data.hh>`

Inheritance diagram for vertex_data:



Public Methods

- virtual void **init** (**combi_map_base** &, vertex)=0

  Function to initialize the vertex_value of the vertex.

- virtual void **merge** (int, vertex, vertex)=0

  Function to manage the vertex_value during the merge of two vertices.

4.25.1 Detailed Description

Model class to store data into vertices *

4.25.2 Member Function Documentation

4.25.2.1 virtual void vertex_data::init (combi_map_base &, vertex) `[pure virtual]`

Function to initialize the vertex_value of the vertex.

Implemented in **vertex_data_color** (p. 70).

4.25.2.2 virtual void vertex_data::merge (int, vertex, vertex) `[pure virtual]`

Function to manage the vertex_value during the merge of two vertices.

Implemented in **vertex_data_color** (p. 70).

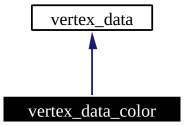
The documentation for this class was generated from the following file:

- **vertex_data.hh**

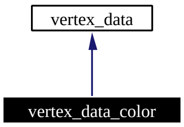
4.26 vertex_data_color Class Reference

`#include <vertex_data_color.hh>`

Inheritance diagram for vertex_data_color:



Collaboration diagram for vertex_data_color:



Public Methods

- **vertex_data_color** (image &I, quantize &Q)
  Constructor with parameters, the number of darts.
- virtual ~**vertex_data_color** ()
  Destructor.
- **vertex_value** & **operator()** (long) const
  Access an element of dataset.
- void **del** (vertex)
  Remove an element from the dataset.
- void **init** (**combi_map_base** &, vertex)
  Function to initialize the **vertex_value** (p. 71) *of the vertex.*
- void **merge** (int, vertex, vertex)
  Function to manage the **vertex_value** (p. 71) *during the merge of two vertices.*
- int **give_label** (vertex)
  Give the label of the color of one vertex.
- int **give_dim_color** ()
  Give the depth of the color.

4.26.1 Detailed Description

One class which store the perimeter and mean color of each vertex

4.26.2 Constructor & Destructor Documentation

4.26.2.1 vertex_data_color::vertex_data_color (image & *I*, quantize & *Q*)

Constructor with parameters, the number of darts.

4.26.2.2 vertex_data_color::~vertex_data_color () [virtual]

Destructor.

4.26.3 Member Function Documentation

4.26.3.1 void vertex_data_color::del (vertex)

Remove an element from the dataset.

4.26.3.2 int vertex_data_color::give_dim_color ()

Give the depth of the color.

4.26.3.3 int vertex_data_color::give_label (vertex)

Give the label of the color of one vertex.

4.26.3.4 void vertex_data_color::init (combi_map_base &, vertex) [virtual]

Function to initialize the **vertex_value** (p. 71) of the vertex.

Implements **vertex_data** (p. 68).

4.26.3.5 void vertex_data_color::merge (int, vertex, vertex) [virtual]

Function to manage the **vertex_value** (p. 71) during the merge of two vertices.

Implements **vertex_data** (p. 68).

4.26.3.6 vertex_data_color::vertex_value & vertex_data_color::operator() (long) const

Access an element of dataset.

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

- **vertex_data_color.hh**
- vertex_data_color.cc

4.27 vertex_data_color::vertex_value Class Reference

`#include <vertex_data_color.hh>`

Public Methods

- **vertex_value** ()

  Constructor.

- **vertex_value** (const vertex_value &)

  Copy constructor.

- **vertex_value** (int *, int)

  Constructor with parameters, a pointor on the vertex color value and the dim of the color array.

- ~**vertex_value** ()

  Destructor.

- vertex_value & **operator=** (const vertex_value &)

  Reaffectation operator.

- vertex_value **reinit** (double *, double *, int, int, int)
- void **give_color** (double *)

  Access to the color sum of all the vertices merged with the vertex.

- void **give_squared_color** (double *)

  Access to the squared color sum of all the vertices which were merged.

- int **give_cardinal** ()

  Access to the cardinal of the vertex (number of vertices merged with it).

- void **give_mean** (double *)

  Give the color mean of the vertex.

- double **give_variance** ()

  Give the color varience inside the vertex.

- double **give_se** ()

  Give the quadratic error inside the vertex.

- int **give_perimetre** ()

  Give the perimeter of the vertex with a pixel side as unit.

- int **add_perimetre** (int)

  Add a value to the perimeter of the vertex.

4.27.1 Detailed Description

Class that hold all the information of one vertex

4.27.2 Constructor & Destructor Documentation

4.27.2.1 vertex_data_color::vertex_value::vertex_value ()

Constructor.

4.27.2.2 vertex_data_color::vertex_value::vertex_value (const vertex_value &)

Copy constructor.

4.27.2.3 vertex_data_color::vertex_value::vertex_value (int *, int)

Constructor with parameters, a pointor on the vertex color value and the dim of the color array.

4.27.2.4 vertex_data_color::vertex_value::~vertex_value () `[inline]`

Destructor.

4.27.3 Member Function Documentation

4.27.3.1 int vertex_data_color::vertex_value::add_perimetre (int)

Add a value to the perimeter of the vertex.

4.27.3.2 int vertex_data_color::vertex_value::give_cardinal ()

Access to the cardinal of the vertex (number of vertices merged with it).

4.27.3.3 void vertex_data_color::vertex_value::give_color (double *)

Access to the color sum of all the vertices merged with the vertex.

4.27.3.4 void vertex_data_color::vertex_value::give_mean (double *)

Give the color mean of the vertex.

4.27.3.5 int vertex_data_color::vertex_value::give_perimetre ()

Give the perimeter of the vertex with a pixel side as unit.

4.27.3.6 double vertex_data_color::vertex_value::give_se ()

Give the quadratic error inside the vertex.

4.27.3.7 void vertex_data_color::vertex_value::give_squared_color (double *)

Access to the squared color sum of all the vertices which were merged.

4.27.3.8 double vertex_data_color::vertex_value::give_variance ()

Give the color varience inside the vertex.

4.27.3.9 vertex_data_color::vertex_value & vertex_data_color::vertex_-value::operator= (const vertex_value &)

Reaffectation operator.

4.27.3.10 vertex_data_color::vertex_value vertex_data_color::vertex_value::reinit (double *, double *, int, int, int)

Reinitialiation function M1, M2, Cardinal, img_dim, perimetre

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

- **vertex_data_color.hh**
- vertex_data_color.cc

Chapter 5

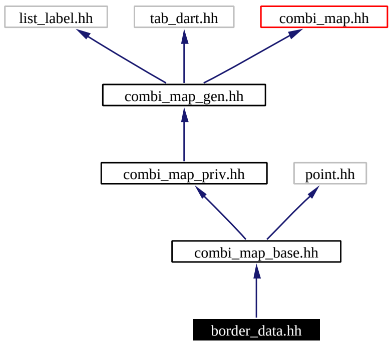
Combinatorial Pyramids File Documentation

5.1 border_data.hh File Reference

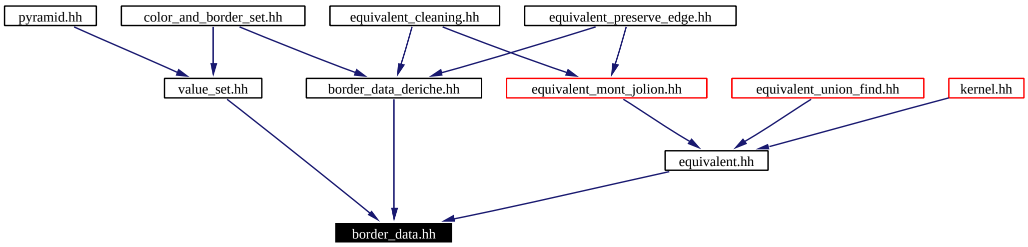
File to manage value associated to each dart. Generally associated to the corresponding boundary.

`#include "combi_map_base.hh"`

Include dependency graph for border_data.hh:



This graph shows which files directly or indirectly include this file:



Compounds

- class **border_data**

5.1.1 Detailed Description

File to manage value associated to each dart. Generally associated to the corresponding boundary.

5.2 border_data_deriche.hh File Reference

File to manage the gradient of each vertex.

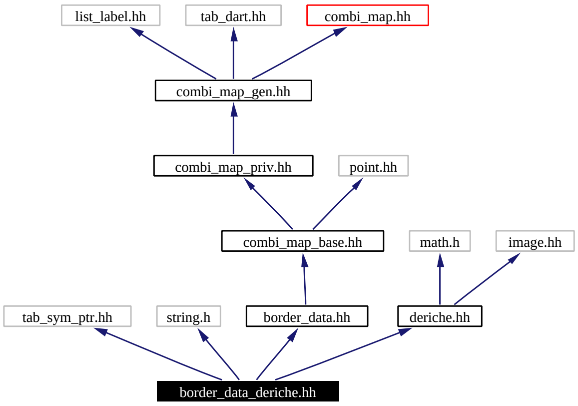
`#include <tab_sym_ptr.hh>`

`#include <string.h>`

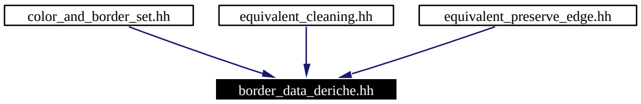
`#include "border_data.hh"`

`#include "deriche.hh"`

Include dependency graph for border_data_deriche.hh:



This graph shows which files directly or indirectly include this file:



Compounds

- class **border_data_deriche**
- class **border_value**

5.2.1 Detailed Description

File to manage the gradient of each vertex.

5.3 color_and_border_set.hh File Reference

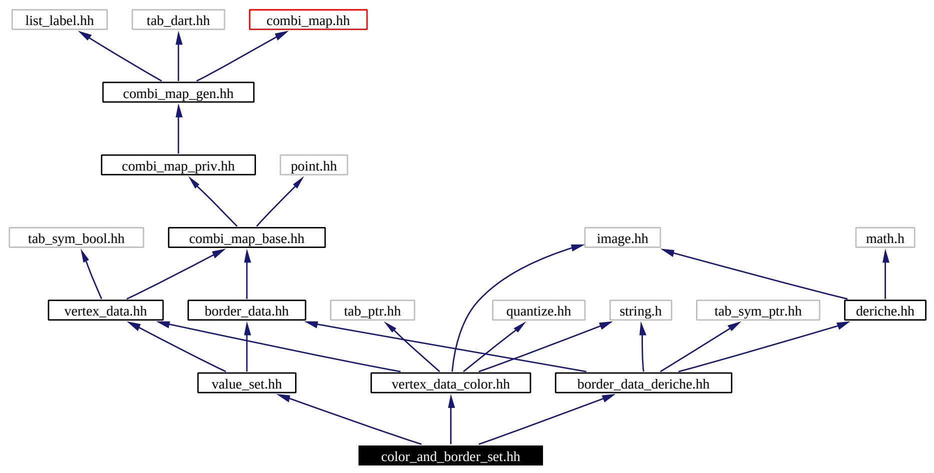
File to manage color and border data of the pyramid.

`#include "value_set.hh"`

`#include "vertex_data_color.hh"`

`#include "border_data_deriche.hh"`

Include dependency graph for color_and_border_set.hh:



Compounds

- class **color_and_border_set**

5.3.1 Detailed Description

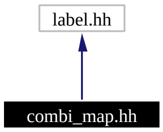
File to manage color and border data of the pyramid.

5.4 combi_map.hh File Reference

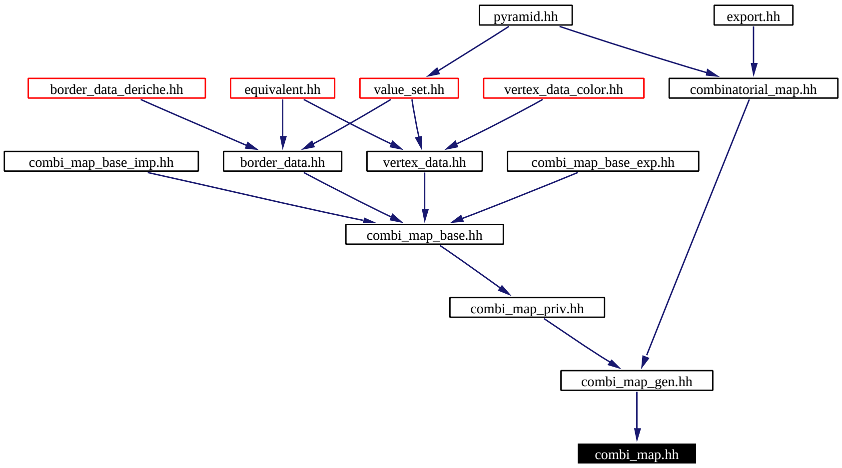
File for the combinatorial maps.

```
#include <label.hh>
```

Include dependency graph for combi_map.hh:



This graph shows which files directly or indirectly include this file:



Compounds

- class **combi_map**

Enumerations

- enum **move_type**

5.4.1 Detailed Description

File for the combinatorial maps.

5.4.2 Enumeration Type Documentation

5.4.2.1 enum move_type

An enum for the orientation of the dart

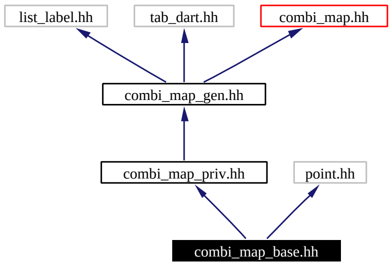
5.5 combi_map_base.hh File Reference

File for the combinatorial maps of the base of the pyramid.

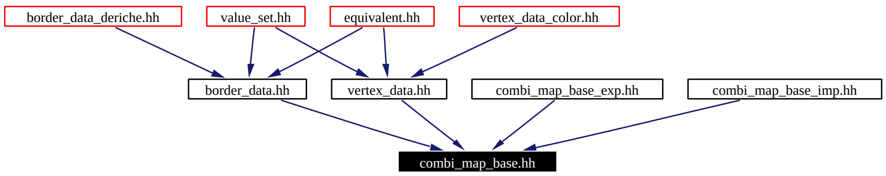
`#include "combi_map_priv.hh"`

`#include "point.hh"`

Include dependency graph for combi_map_base.hh:



This graph shows which files directly or indirectly include this file:



Compounds

- class **combi_map_base**

5.5.1 Detailed Description

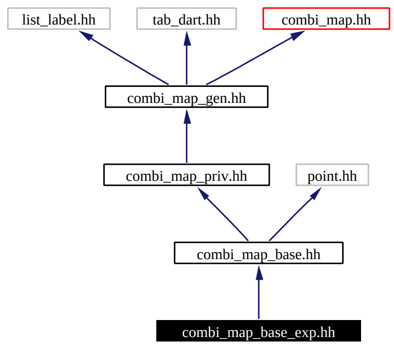
File for the combinatorial maps of the base of the pyramid.

5.6 combi_map_base_exp.hh File Reference

File that manage the combinatorial maps of the base of the pyramid by storing sigma permutation.

`#include "combi_map_base.hh"`

Include dependency graph for combi_map_base_exp.hh:



Compounds

- class **combi_map_base_exp**

5.6.1 Detailed Description

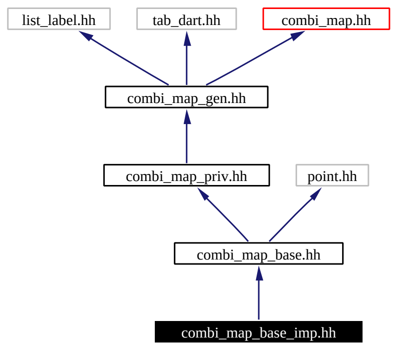
File that manage the combinatorial maps of the base of the pyramid by storing sigma permutation.

5.7 combi_map_base_imp.hh File Reference

File to manage the combinatorial maps of the base of the pyramid without storing sigma permutation.

`#include "combi_map_base.hh"`

Include dependency graph for combi_map_base_imp.hh:



Compounds

- class **combi_map_base_imp**

5.7.1 Detailed Description

File to manage the combinatorial maps of the base of the pyramid without storing sigma permutation.

5.8 combi_map_gen.hh File Reference

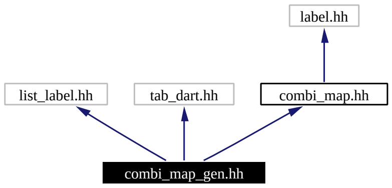
File for the generalised combinatorial maps.

`#include <list_label.hh>`

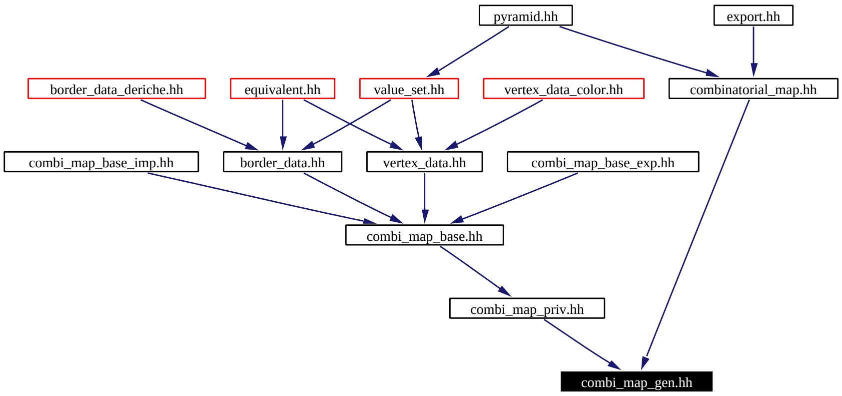
`#include <tab_dart.hh>`

`#include "combi_map.hh"`

Include dependency graph for combi_map_gen.hh:



This graph shows which files directly or indirectly include this file:



Compounds

- class **combi_map_gen**

5.8.1 Detailed Description

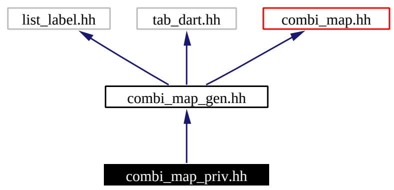
File for the generalised combinatorial maps.

5.9 combi_map_priv.hh File Reference

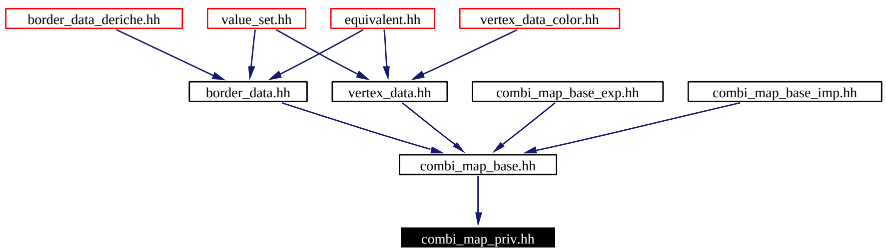
File for the combinatorial maps used in the pyramid.

`#include "combi_map_gen.hh"`

Include dependency graph for combi_map_priv.hh:



This graph shows which files directly or indirectly include this file:



Compounds

- class **combi_map_priv**

5.9.1 Detailed Description

File for the combinatorial maps used in the pyramid.

5.10 combinatorial_map.hh File Reference

File for the combinatorial maps.

`#include <point.hh>`

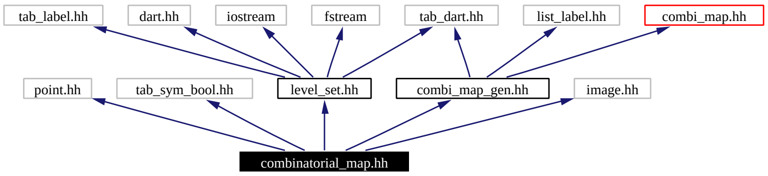
`#include <tab_sym_bool.hh>`

`#include "level_set.hh"`

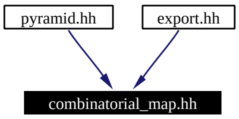
`#include "image.hh"`

`#include "combi_map_gen.hh"`

Include dependency graph for combinatorial_map.hh:



This graph shows which files directly or indirectly include this file:



Compounds

- class **combinatorial_map**
- class **segment**
- class **iterator**

5.10.1 Detailed Description

File for the combinatorial maps.

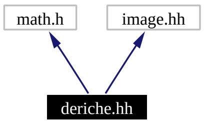
5.11 deriche.hh File Reference

File to compute the deriche of a picture.

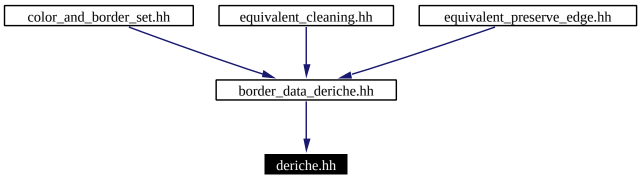
`#include <math.h>`

`#include <image.hh>`

Include dependency graph for deriche.hh:



This graph shows which files directly or indirectly include this file:



5.11.1 Detailed Description

File to compute the deriche of a picture.

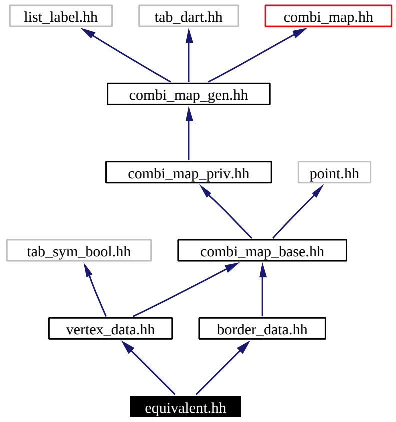
5.12 equivalent.hh File Reference

File used to determine if two vertices are equivalent.

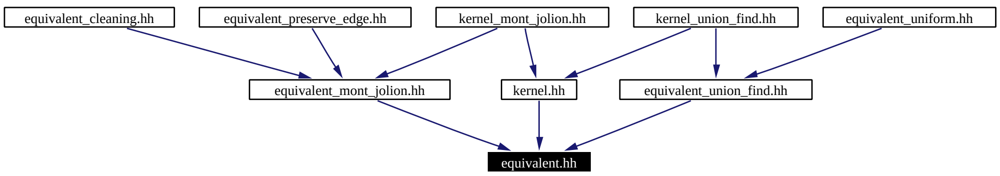
`#include "vertex_data.hh"`

`#include "border_data.hh"`

Include dependency graph for equivalent.hh:



This graph shows which files directly or indirectly include this file:



Compounds

- class **equivalent**

5.12.1 Detailed Description

File used to determine if two vertices are equivalent.

5.13 equivalent_cleaning.hh File Reference

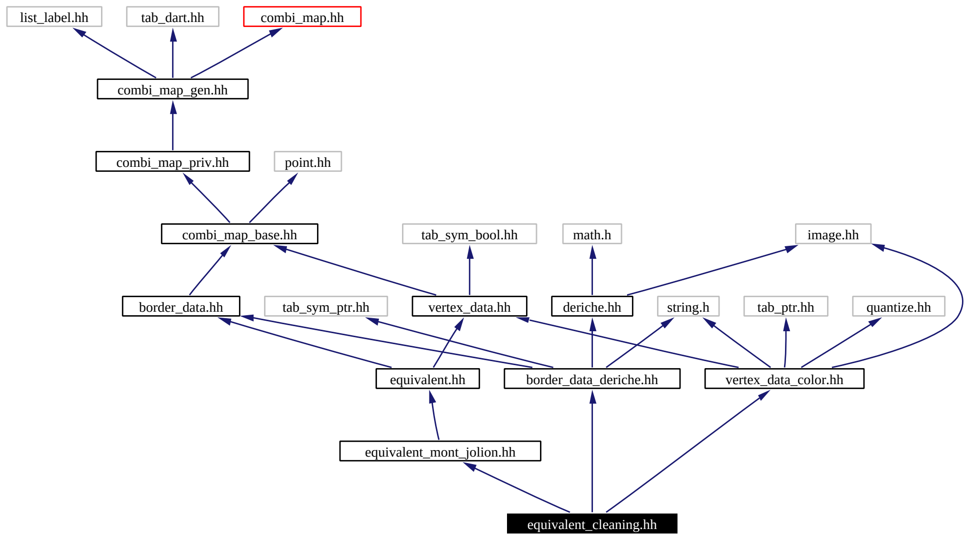
File used to determine and remove all the small vertices.

`#include "equivalent_mont_jolion.hh"`

`#include "border_data_deriche.hh"`

`#include "vertex_data_color.hh"`

Include dependency graph for equivalent_cleaning.hh:



Compounds

- class **equivalent_cleaning**

5.13.1 Detailed Description

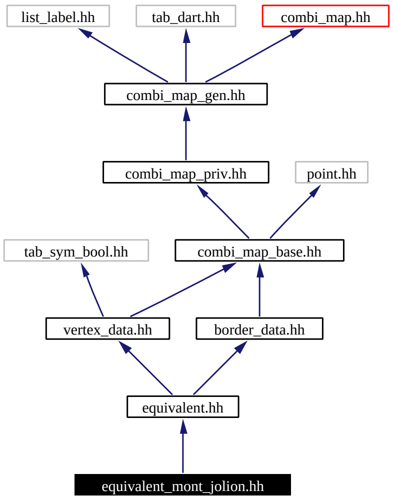
File used to determine and remove all the small vertices.

5.14 equivalent_mont_jolion.hh File Reference

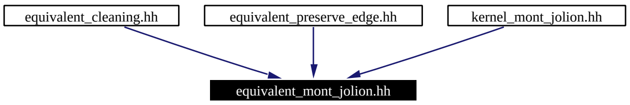
File used to determine if two vertices are equivalent.

`#include "equivalent.hh"`

Include dependency graph for equivalent_mont_jolion.hh:



This graph shows which files directly or indirectly include this file:



Compounds

- class **equivalent_mont_jolion**

5.14.1 Detailed Description

File used to determine if two vertices are equivalent.

5.15 equivalent_preserve_edge.hh File Reference

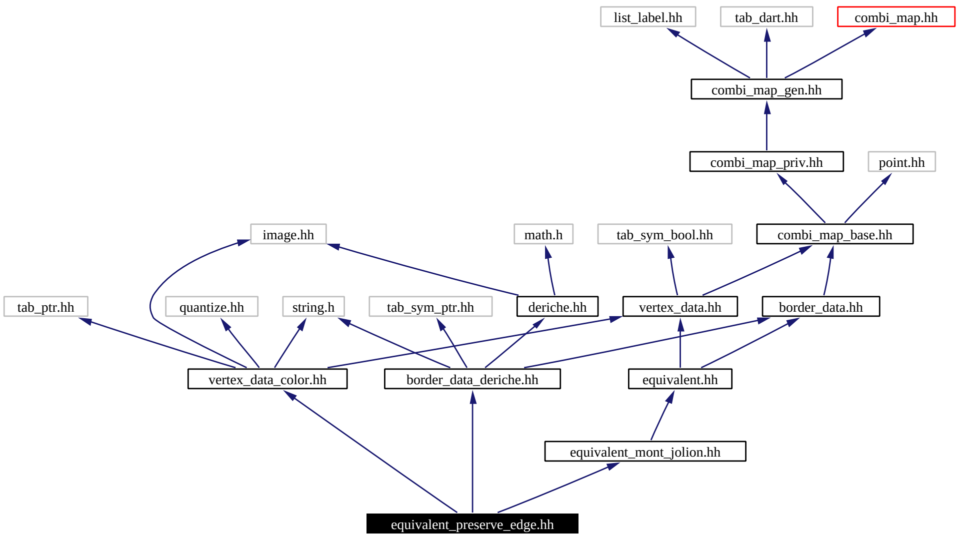
File used to determine if two vertices are equivalent.

`#include "equivalent_mont_jolion.hh"`

`#include "vertex_data_color.hh"`

`#include "border_data_deriche.hh"`

Include dependency graph for equivalent_preserve_edge.hh:



Compounds

- class **equivalent_preserve_edge**

5.15.1 Detailed Description

File used to determine if two vertices are equivalent.

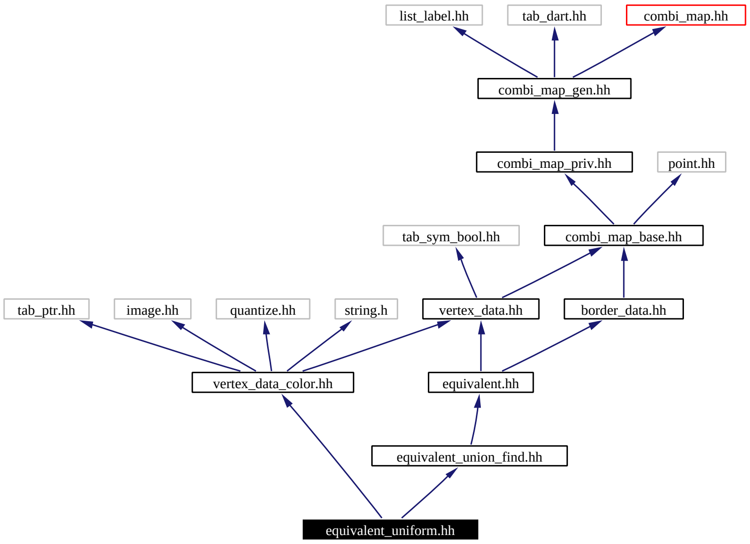
5.16 equivalent_uniform.hh File Reference

File used to determine if two vertices are equivalent.

`#include "equivalent_union_find.hh"`

`#include "vertex_data_color.hh"`

Include dependency graph for equivalent_uniform.hh:



Compounds

- class **equivalent_uniform**

5.16.1 Detailed Description

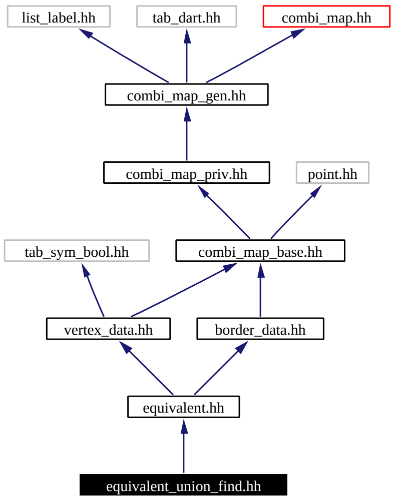
File used to determine if two vertices are equivalent.

5.17 equivalent_union_find.hh File Reference

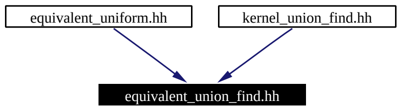
File used to determine if two vertices are equivalent.

`#include "equivalent.hh"`

Include dependency graph for equivalent_union_find.hh:



This graph shows which files directly or indirectly include this file:



Compounds

- class **equivalent_union_find**

5.17.1 Detailed Description

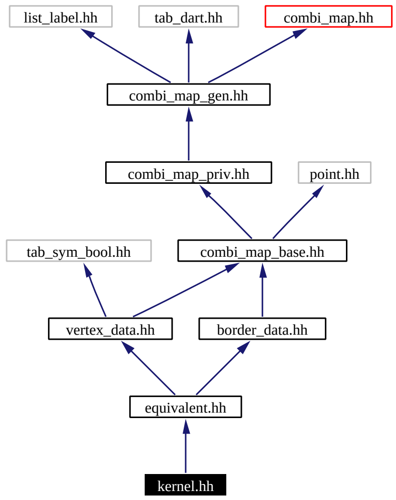
File used to determine if two vertices are equivalent.

5.18 kernel.hh File Reference

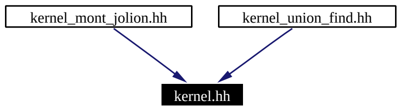
File to compute and hold the survivor kernel of the combinatorial map.

`#include "equivalent.hh"`

Include dependency graph for kernel.hh:



This graph shows which files directly or indirectly include this file:



Compounds

- class **kernel**

5.18.1 Detailed Description

File to compute and hold the survivor kernel of the combinatorial map.

5.19 kernel_mont_jolion.hh File Reference

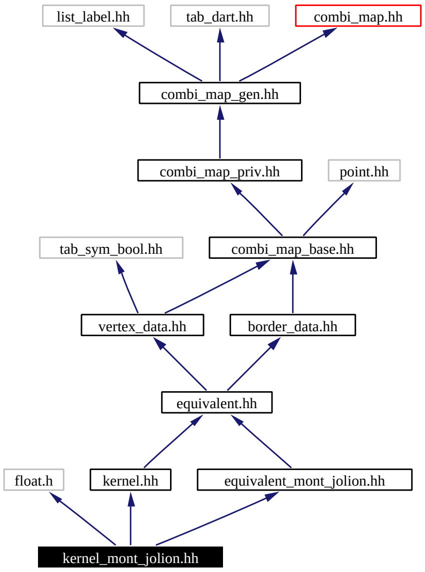
File to compute and hold the survivor kernel of the combinatorial map.

`#include <float.h>`

`#include "kernel.hh"`

`#include "equivalent_mont_jolion.hh"`

Include dependency graph for kernel_mont_jolion.hh:



Compounds

- class **kernel_mont_jolion**

5.19.1 Detailed Description

File to compute and hold the survivor kernel of the combinatorial map.

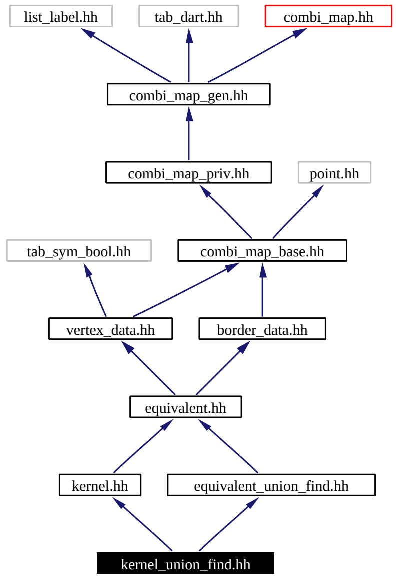
5.20 kernel_union_find.hh File Reference

File to compute and hold the survivor kernel of the combinatorial map.

`#include "kernel.hh"`

`#include "equivalent_union_find.hh"`

Include dependency graph for kernel_union_find.hh:



Compounds

- class **kernel_union_find**

5.20.1 Detailed Description

File to compute and hold the survivor kernel of the combinatorial map.

5.21 level_set.hh File Reference

File to encoding of the level of each dart in the pyramid.

`#include <tab_label.hh>`

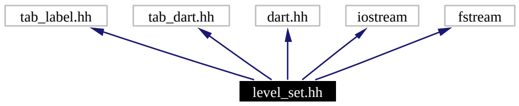
`#include <tab_dart.hh>`

`#include <dart.hh>`

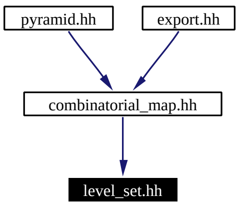
`#include <iostream>`

`#include <fstream>`

Include dependency graph for level_set.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **std**

Compounds

- class **iterator**
- class **l_set_iteratorator_error**
- class **level_set**

5.21.1 Detailed Description

File to encoding of the level of each dart in the pyramid.

5.22 pyramid.hh File Reference

File that will manage the combinatorial pyramid.

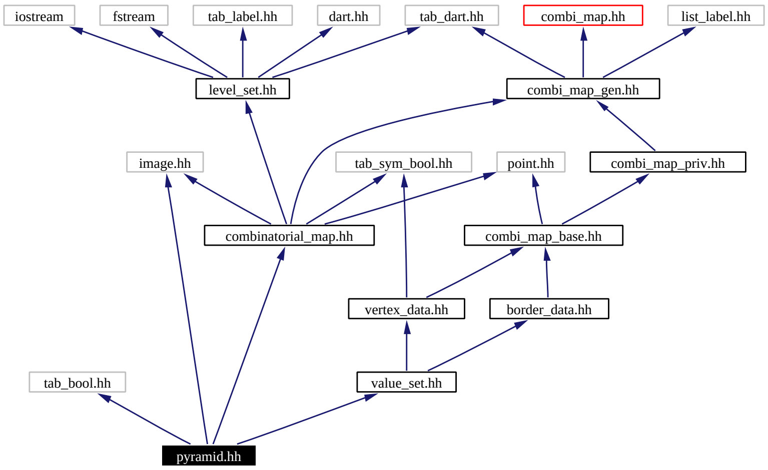
`#include <tab_bool.hh>`

`#include <image.hh>`

`#include "value_set.hh"`

`#include "combinatorial_map.hh"`

Include dependency graph for pyramid.hh:



Compounds

- class **pyramid**

5.22.1 Detailed Description

File that will manage the combinatorial pyramid.

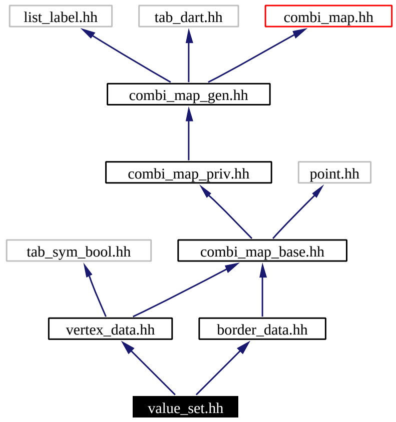
5.23 value_set.hh File Reference

File to manage the value data of the pyramid.

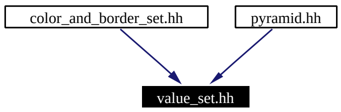
`#include "border_data.hh"`

`#include "vertex_data.hh"`

Include dependency graph for value_set.hh:



This graph shows which files directly or indirectly include this file:



Compounds

- class **value_set**

5.23.1 Detailed Description

File to manage the value data of the pyramid.

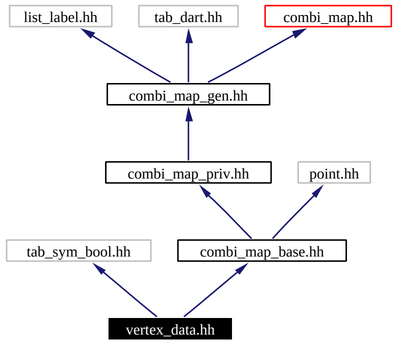
5.24 vertex_data.hh File Reference

File to store data into vertices.

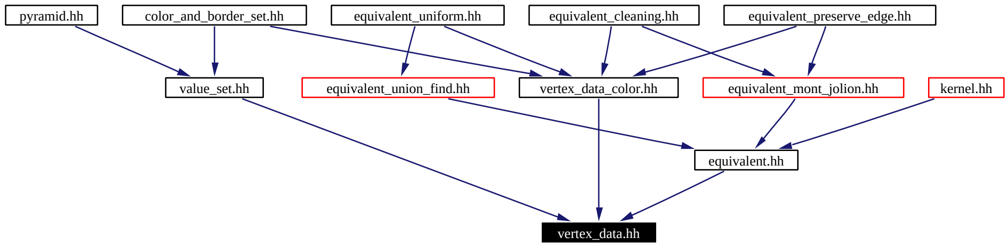
`#include <tab_sym_bool.hh>`

`#include "combi_map_base.hh"`

Include dependency graph for vertex_data.hh:



This graph shows which files directly or indirectly include this file:



Compounds

- class **vertex_data**

5.24.1 Detailed Description

File to store data into vertices.

5.25 vertex_data_color.hh File Reference

File that manage the class which store the perimeter and mean color of each vertex.

`#include <tab_ptr.hh>`

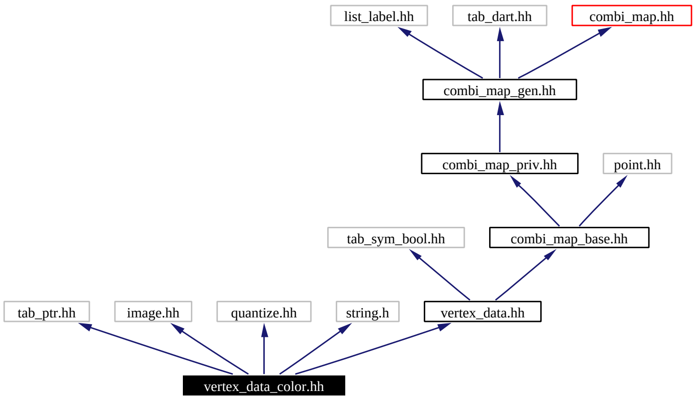
`#include <image.hh>`

`#include <quantize.hh>`

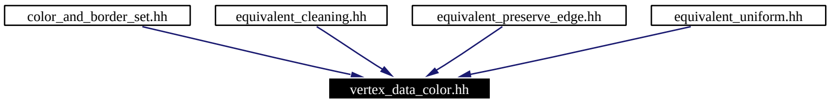
`#include <string.h>`

`#include "vertex_data.hh"`

Include dependency graph for vertex_data_color.hh:



This graph shows which files directly or indirectly include this file:



Compounds

- class **vertex_data_color**
- class **vertex_value**

5.25.1 Detailed Description

File that manage the class which store the perimeter and mean color of each vertex.