Fade2.5D

v1.81

Generated by Doxygen 1.8.13

Contents

1 Main Page

1.1 C++ Constrained Delaunay Triangulation Fade2.5D

- Very fast multithreaded Delaunay triangulation library `(benchmark)`
- Well documented and with easy to use `example source codes`
- 2D Delaunay with `Polygon support` and Constraint Edges
- 2.5D Delaunay for Surfaces and Digital Elevation Models
- Earthwork Volume Computations: `Cut And Fill`
- `Segment Intersection Test Software`
- `Grid Mesher and Delaunay Mesh Generator`
- Supports Windows, MacOS, Linux on PC and Raspberry PI
- Student license. Evaluation license. Commercial licenses and support available
- Fade is actively developed and maintained. If you miss a feature please get in `contact`

1.1.1 Getting started with Delaunay triangulations:

Download. Unzip. Start to play with the included example source codes. It works without installation. `The first example is described here`.

Fade comes as two separate libraries, Fade2D and Fade2.5D. Fade2.5D can do everything that Fade2D can do. But Fade2.5D offers a z-coordinate and additional functionality for Digital Elevation Models (DEM) and surfaces. A collection of 2D and 2.5D example source codes is contained in the download. The examples go step by step over the concepts of Fade. New Fade2.5D users are advised to start with the 2D examples because the basics are described there and these apply also to Fade2.5D.

1.1.2 For Windows users:

1. Open one of the Visual Studio example projects (currently supported: VS2010, VS2012, VS2013, VS2015, VS2017, VS2019)
2. Compile the example source code. The executable is written to the Win32 or x64 folder.

When you link Fade with your own software you can use the settings from the example solutions or use the table below:

Visual Studio	IDE version	Platform Toolset
VS2010	version 10	toolset v100 or Windows7.1SDK
VS2012	version 11	toolset v110
VS2013	version 12	toolset v120
VS2015	version 14	toolset v140
VS2017	version 15	toolset v141
VS2019	version 16	toolset v142

1.1.3 For Linux and Apple users:

1. Edit the Makefile (choose Apple, your Linux distro or Raspberry PI) and type make to compile the example source code.
2. Make sure GMP is installed:
   $ sudo apt-get install libgmp10 (works on Ubuntu/Debian/Mint/Raspbian, on other systems search for libgmp or gmp)

Work through the provided examples. They are small, well documented and they visualize the results.

1.1.4 Directory Contents

- **include_fade2d** and **include_fade25d**
  Header files of the two libraries.
- **Win32** and **x64**
  This directory contains the DLL's for Windows 32-bit and 64-bit and it is the target directory for the executables of example code compiled with Visual Studio.
- **lib_${DISTRO}_${ARCHITECTURE}**
  The shared libs (*.so) for Linux/Apple developers.
- **examples_2D**
  Example source code and Visual Studio projects using Fade2D
- **examples_25D**
  Example source code and Visual Studio projects using Fade2.5D
- **doc**
  PDF Documentation

1.1.5 Troubleshooting

- Mixing multiple Visual Studio versions won't work. Use the right dll.
- Compare your settings with the ones from the example projects.
- In rare cases you might need to increase Properties->ConfigurationProperties->Linker->System->Stack↩ ReserveSize in your Visual Studio project settings.
- If your problem persists, don't hesitate to send a minimal example that reproduces it and it will be fixed asap.

1.1.6 Release notes / History

Version 1.81, May 17th, 2020:

- Memory Leak in EfficientModel fixed. EfficientModel improved: Pruning the point cloud is much faster now and the new method zSmoothing() has been implemented. It provides minimum-, maximum-, median- and average-smoothing.

Version 1.80, March 25th, 2020:

- Bug in Cut&Fill solved: A foot point was computed in 3D while it should have been computed in 2D. The difference was in most cases insignificant and thus the problem did not become apparent earlier. Sorry. Fixed.
- Improvement in Cut&Fill: The algorithm checks now if the two input zones do overlap. If not, the CutAndFill↩ ::go() method returns false and the CutAndFill object shall not further be used.
- Example source codes adapted and -std=c++98 removed from their Makefiles
- Documentation improved

Version 1.79, January 20th, 2020:
Internal version. Revision.

Version 1.78, November 15th, 2019:

- Bugfix: Multithreading did not work in Windows versions due to a CMake configuration error.
- A typo in the function name Fade_2D::measureTriangulationTime() has been corrected.

Version 1.77, October 21st, 2019

- Support for Visual Studio 2019.
- A bug has been fixed: In a rare case a self-intersecting constraint graph could generate an error.
- Improvements: The constraint-insertion-strategies CIS_CONFORMING_DELAUNAY and CIS_CONFORM↩ ING_DELAUNAY_SEGMENT_LEVEL are deprecated now.

- The fast and reliable replacement is CIS_CONSTRAINED_DELAUNAY along with the new methods ConstraintGraph::makeDelaunay() and Fade_2D::drape(). See the new example code in examples_25D/terrain.cpp.

Version 1.75 and 1.76

- Non-public test versions.

Version 1.74, March 19th, 2019:

- Cleanup: The (until now experimental) surface reconstruction module has been moved into the separate WOF Point Cloud Meshing library (https://www.geom.at/products/wof-point-cloud-mesher/). This makes the Fade binaries smaller and it improves the maintainability of the code.
- Cleanup: Support for VS2008 has been dropped (if you are a commercial user and still need VS2008 then contact the author please!).
- The build system has been migrated to CMake to reduce the manual work and to guarantee uniform flags for all builds.
- The HoleFiller class that has been developed for the removed surface reconstruction module is retained in Fade because it has already users. Its code has been revised in order to provide repeatable results for identical inputs.
- According to a user request the MeshGenParams class (used for advanced Delaunay Meshing) offers now a method to lock certain constraint segments such that they are not splitted while all others can be splitted if required.

Version 1.73, January 14th, 2019:
While all below mentioned versions after v1.63 were development versions the present v1.73 is again an official release version for all. The work of the below internal versions is included as well as a bugfix in the getProfile() method of the IsoContours class (this method was new and experimental in v1.63)

Version 1.71 and 1.72, October 24th, 2018:
(internal) Hole-Filling (Polygon-Triangulation) improved.

Version 1.70, October 17th, 2018:
(internal) Hole-Filling (Polygon-Triangulation) improved.

Version 1.69, October 15th, 2018:
(internal) Hole-Filling (Polygon-Triangulation) improved.

Version 1.68, September 14th, 2018:
(internal) Hole-Filling (Polygon-Triangulation) improved.

Version 1.67, September 4th, 2018:
(internal) Hole-Filling (Polygon-Triangulation) is now offered via. an API call. Intermediate beta release.

Version 1.66, August 25th, 2018:
(internal) Bugfix in Cut&Fill: An intersection point could be slightly off its expected range. Solved. Unofficial intermediate version.

Version 1.65, July 29th, 2018:
(internal) Another bugfix in Cut&Fill. Unofficial intermediate version.

Version 1.64, July 21st, 2018:
(internal) Bugfix in the Cut&Fill module: In rare cases Cut&Fill crashed due to unexpected numeric deviation (fixed). The importTriangles() function has been reimplemented and is considerably faster now. And there is a change that affects only 32-bit users: Binary files written with the writePointsBIN() and writeSegmentsBIN() functions on 32-bit machines were not readable on 64-bit machines. The format on 32-bit machines (read/write) has been adapted to match exactly the one of 64-bit machines. But note that old 32-bit files are not readable anymore. This should affect next to nobody, thus this solution has been chosen.

Version 1.63, June 10th, 2018:
Cookie-Cutter operation added. 3D Point Cloud Reconstruction added to the API (but is still under development, pls. take it as a preview). Raspberry PI support added again.

Version 1.62, June 3rd, 2018:
3D Point Cloud Reconstruction considerably improved. Unofficial demo.

Version 1.61, May 1st, 2018:
3D Point Cloud Reconstruction: Unofficial demo.

Version 1.60, February 26th, 2018:
Accurate computation of glancing segment intersections. Additional parameter for Advanced Meshing: bool b↩ KeepExistingSteinerPoints=true in MeshGenParams makes all Steiner points from previous refinement calls static, i.e. unremovable during subsequent refinement calls. This way Advanced Meshing can be carried out for several zones of a triangulation such that it does not destroy what has been meshed so far.

Version 1.59, January 14th, 2018:
Performance upgrade: Multithreading is available now. Large point sets reach a speedup of 4.4 on a hexacore CPU (i7 6800K)

Version 1.58, October 23th, 2017:
Mesh Generator refactored. Delaunay Meshing is +10x faster now. A function to create polygons from boundary edges has been added.

Version 1.57, October 9th, 2017:
Nonpublic test version.

Version 1.56, September 24th, 2017:
Bugfix: createConstraint() crashed in a rare case. Solved. Functions for binary file I/O added.

Version 1.55, August 12th, 2017:
Access to internal Cut&Fill datastructures revised. Example source codes revised. Support for Visual Studio 2017 added.

Version 1.54beta, August 8th, 2017:
Access to internal Cut&Fill datastructures. This is a pre-released beta version, code quality is good but final tests and documentation updates required.

Version 1.53, July 15th, 2017:
Error corrections and performance upgrades in the still quite new Cut&Fill library module.

Version 1.53 beta, June 2nd, 2017:
The new Cut&Fill library module has been added. Cut&Fill computes the volume between two surfaces.

Version 1.51 beta, May 27th, 2017:
Non-public test version

Version 1.50, April 5th, 2017:
After three internal test versions (that concetrated on refactoring and rare bugs) this is again a stable public release version: The constraint insertion subsystem has been rewrittten and is faster now. Visualization improved. Exact orientation tests provided through the API. Improved progress bar support. Mesh generator improved.

Users who upgrade from earlier Fade versions: The Zone2::getArea() and Triangle2::getArea() methods have been replaced by getArea2D() in Fade2D and by getArea2D() AND getArea25D() in Fade2.5D. The reason is that the old getArea() method was easily misunderstood in Fade2.5D (it returned the same result as getArea25D() now). We have decided to remove the old method to avoid confusion and a potential source of error. If necessary, please adapt your code.

Version 1.49, March 2nd, 2017:
Constraint insertion subsystem improved. Mesh generator revised.

Version 1.48, February 15th, 2017:
Corrections of yesterday's v1.47 version.

Version 1.47, February 14th, 2017:
The focus of this (for now) non-public version is stability: Intersecting constraint segments must be subdivided although their exact intersection is not always representable with double precision coordinates. Thus tiny rounding errors are unavoidable and these caused trouble in very unlikely cases. The constraint insertion subsystem has now been re-implemented to behave robust also in such cases.

Version 1.46a, January 14th, 2017:
+++ Raspberry PI is supported now +++ // Appart from RPI support version 1.46a is equal to v1.46. Raspberry PI users: Please give feedback, do you have everything you need for RPI development now?

Version 1.46, January 8th, 2017:
+++ MacOS is supported now +++ // A new class EfficientModel takes oversampled 2.5D point clouds and returns a subset that represents the model efficiently. The automatic pruning process runs in a controlled fashion such that a user specified maximum error is kept. // The Delaunay Mesh Generator is now supported by a Grid Mesher, thus it creates more regular meshes. // The Delaunay triangulation of specific point sets is not unique, for example when grid points are triangulated (4 points on a common circumcircle). To improve the repeatability and for reasons of visual appearance the new method Zone2::unifyGrid() has been implemented. // A problem in the point location method Fade_2D::locate() when the query point was exactly on the convex hull of the triangulation has been solved.

Version 1.43, November 20th, 2016:
Better example source code for the new SegmentChecker class. And the SegmentChecker of v1.42 returned false positives, this problem is solved now.

Version 1.42, October 19th, 2016:
The new tool SegmentChecker takes a bunch of segments and fully automatically identifies intersecting segments. The underlying data structure makes the tool incredibly fast. Intersecting segments can be visualized. Intersections can be computed in 2D and 2.5D (with heights). A new module named TestDataGenerators creates random polygons, random segments, points, random numbers and polylines for automated software stress tests. Progress bar support added.

Version 1.41, July 24th, 2016:
New constraint insertion strategy. Minor bug fixes. Performance slightly improved.

Version 1.40 beta, June 14th, 2016:
Non public intermediate test version. Bounded zones introduced: Mesh generation algorithms require that zones are bounded by constraint segments. This is certainly the case for the most usual zones with zoneLocation=Z↩
L_INSIDE. But other types of zones may be unbounded and in this case remeshing won't work well, so it was necessary to change the behavior: From now on calling refine() and refineAdvanced() is only allowed with zones whose zoneLocation is ZL_INSIDE or ZL_BOUNDED. A bounded zone can easily be gained from any other zone using Zone2::convertToBoundedZone(). Also new: Fade_2D::createConstraintGraph(..) has now a third parameter 'bool bOrientedSegments=false'. By default it is false to provide backwards compatibility. This parameter allows you to specify that the provided segments are CCW oriented. This way more complex inside- and outside-zones can be formed. Performance of Fade_2D::createConstraint(..) drastically improved.

Version 1.39, May 31st, 2016:
Non public intermediate test version.

Version 1.37, March 15th, 2016:
Small upgrade: The performance of the remove method has been improved.

Version 1.37, March 10th, 2016:
Interface change in the MeshGenParams class. The class has been introduced two weeks before, so chances are good that the change does not affect you. Previously the class had the methods getMaxTriangleArea(double x,double y) and getMaxEdgeLength(double x,double y) where x and y where the barycenter of a triangle for which the algorithm determines if it must be refined. The change is that x and y have been replaced by the triangle itself to give client code even more control (x and y can still be computed from the triangle).

Version 1.36, February 29th, 2016:
Experimental method refineExtended(..) replaced by the (now permanent) method refineAdvanced(MeshGen↩ Params∗ pParams). This method allows much more control over the mesh density.

Version 1.34, February 14th, 2016:
Vertex management subsystem revised (sometimes Vertex removement did not work as expected). Performance improvement.

Version 1.33 PreRelease, January 17th, 2016:
The previous official Fade version is Fade 1.24. It was released 6 months ago. Since then major developments have been made and now a big upgrade follows with version 1.33.14: Constraint segments may intersect now and they are automatically subdivided at their intersection points. Import of existing triangles is supported and one can cut through static triangulations. This version is well tested. It also runs at two customers sites with no known problems. But due to the large amount of new code we call this version a pre-release. Please report if you find any problems and note that it is also helpful if you report that the library works well in your setting. The DLL names have been adapted to the safer and more convenient pattern

fade[2D|25D]_$(Platform)_$(PlatformToolset)_$(Configuration).dll

If you upgrade from an earlier version it is recommended that you remove any previous Fade DLL's to avoid unintended linking to an old version.

Version 1.31 and 1.32, December 1st, 2015:
Non public intermediate release, improves the CDT.

Version 1.30, November 18th, 2015:
Non public intermediate release, improves the refineExtended method.

Version 1.29, October 17th, 2015:
Non public intermediate release. The method importTriangles() detects invalid input data now and returns NULL to avoid an assertion or even an infinite loop when the input data is not clean. The possibly invalid input elements are written to stdout and a postscript file visualizes where the problem occurs.

Version 1.28, October 10th, 2015:
Non public intermediate release. Customer specific code revised. Stress tests with random polygons and segments have been made. Heap checking to ensure proper memory handling.

Version 1.27, October 5th, 2015:
Non public release, improvements of the recently implemented functions, especially of customer specific code Fade_2D::importTriangles() and Fade2D::cutTriangles().

Version 1.26, September 8th, 2015:
New functions of the last inofficial version 1.25 have been revised. Constraint segments may intersect now.

Version 1.25, August 18th, 2015:
Intermediate pre-release with new features: importTriangles() imports arbitrary triangles into a triangulation, cut↩ Triangles() allows to insert a constraint segment as if it where a knife, getOrientation() provides an exact orientation test. Zone2 objects can now also be made from a set of triangles. Constraint segments can intersect now. These

features correspond to a large amount of new code: Please test v1.25 carefully before deploying it in a production environment.

Version 1.24, July 22nd, 2015:
Public release of version 1.23's improvements. And I'm sorry but we had a bug in Fade_2D::getVertexPointers(..). The method may have missed to return a few pointers after a call to refine() or remove(). This bug is fixed now.

Version 1.23, July 9th, 2015:
Internal test release with the new refineExtended() method for the specific needs of a certain client software.

Version 1.22, May 25th, 2015:
Code refactored, build system refactored and as a result improved Linux support: CentOS 6.4, Ubuntu 14.04, Ubuntu 15.04 and similar systems. Removement of points has been implemented, Delaunay meshing has been reworked, sqDistance() has been replaced by sqDistance2D() and sqDistance25D() because both versions are useful in 2.5D. OpenMP has been removed, it was only used under Linux and currently I work on a better way to provide multithreading.

Version 1.21, May 17th, 2015:
Unofficial intermediate release. Testing new features.

Version 1.20, April 5th, 2015:
3D scene Visualization for (up to date) web browsers added. Misleading enumeration values CIS_KEEP_DELA↩UNAY and CIS_IGNORE_DELAUNAY have been replaced by CIS_CONFORMING_DELAUNAY and CIS_CON↩STRAINED_DELAUNAY (the two deprecated names are kept for backward compatibility). Bug in the free function center(Point2&,Point2&) solved. Major revision of the documentation pages. The source codes of the examples have been reengineered and are included in the present documentation pages.

Version 1.19, October 26th, 2014:
Support for Visual Studio 2013 (VC12) has been added. Only minor code changes.

Version 1.18.3, June 9th, 2014:
Delaunay Mesh Generation has been improved: Better quality, better performance. API improved. Small bug fixes.

Version 1.16.1, February 10th, 2014:
Small update: In rare cases it was possible that subdivided ConstraintSegments caused problems in combination with zone growing. This is fixed now.

Version 1.16, February 3rd, 2014:
Constrained Delaunay triangulation improved, Delaunay meshing improved, aspect ratio meshing (experimental) added. Minor bug fixes. Added support for Visual Studio 2012.

Version 1.14, November 2013 and version 1.15, December 2013:
Non-public intermediate releases (test versions with experimental features).

Version 1.13, August 4th, 2013:
Mesh generation (Delaunay Meshing) has been improved and two bugfixes have been made in the new IsoContours class: A message can be suppressed now and a numeric problem has been fixed.

Version 1.12, June 30th, 2013:
Starting with version v1.12 the download consists of two separate libraries: The familiar full version of the 2D flavor as well as a 2.5D evaluation version. Two very fast new methods have been added to the 2.5D version: One computes iso-contours, the other computes the height of a point with arbitrary (x,y) coordinates. Delaunay mesh generation has been improved. Support for VS2008, 32-bit and 64-bit, has been added. The performance has been improved.

Version 1.11, June 14th, 2013:
Non-public intermediate release with VS2008 support and a first version of the iso-contour feature.

Version 1.10, March 30th, 2013:
Delaunay Refinement (already included as preview in the previous release) has been improved and is officially released now. Parts of the algorithm can use up to 8 CPUs under Linux if explicitly switched on using Fade2D↩
::enableMultithreading(). There is a new insert method in the API which uses arrays.

Version 1.03, Nov. 4th, 2012:
A critical bug has been fixed, please switch to the current version. Performance upgrade: A first step towards multithreading has been made in the Linux version. In order to facilitate the installation for users without administrator privileges the installers have been replaced by a simple zipped directory that contains everything. Meshing through Delaunay Refinement is scheduled for the next release but it is pre-released as an experimental feature in the current version 1.03.

Version 1.02, 9/2012:
An additional debug library version for Windows has been added and the directory structure has been reorganized.

Version 1.01, 9/2012:
This is a stable public release. Since version 0.9 we have introduced insertion of constraint edges and the zone concept. Moreover the API is under a namespace now. Boost types have been removed from the API to avoid this additional dependency. New demo software has been written and the library is now also available for 64-bit Windows.

2 Module Index

2.1 Modules

Here is a list of all modules:

3 Class Index

3.1 Class List

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

4 File Index

4.1 File List

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

5 Module Documentation

5.1 Tools

Functions

- void GEOM_FADE25D::pointsToPolyline (std::vector< Point2 > &vInPoints, bool bClose, std::vector< Segment2 > &vOutSegments)

  Points-to-Polyline.
- bool GEOM_FADE25D::isSimplePolygon (std::vector< Segment2 > &vSegments)

  isSimplePolygon
- Vector2 GEOM_FADE25D::getNormalVector (const Point2 &p0, const Point2 &p1, const Point2 &p2, bool &bOK)

  Get normal vector.
- void GEOM_FADE25D::getDirectedEdges (std::vector< Triangle2 *> &vT, std::vector< Edge2 > &vDirectedEdgesOut)

  Get directed edge The directed edges of `vT` are returned `vDirectedEdgesOut`. Directed means that each edge (a,b) with two adjacent triangles in vT is returned twice, as edge(a,b) and edge(b,a).
- void GEOM_FADE25D::getUndirectedEdges (std::vector< Triangle2 *> &vT, std::vector< Edge2 > &vUndirectedEdgesOut)

  Get undirected edges.
- bool GEOM_FADE25D::fillHole (std::vector< std::pair< Segment2, Vector2 > > vPolygonSegments, bool bWithRefine, bool bVerbose, std::vector< Point2 > &vCornersOut)

  Fill a hole in a 3D mesh with triangles (deprecated)
- bool GEOM_FADE25D::fillHole (std::vector< Point2 > &vMeshCorners, std::vector< Segment2 > &vPolygonSegments, bool bWithRefine, bool bVerbose, std::vector< Point2 > &vCornersOut)

  Fill a hole in a 3D mesh with triangles (deprecated)
- bool GEOM_FADE25D::fillHole (Mesh3 *pMesh, std::vector< Edge2 > &vPolygonEdges, bool bWithRefine, bool bVerbose, std::vector< Point2 > &vCornersOut)

  Fill a hole in a 3D mesh with triangles (deprecated)
- void GEOM_FADE25D::edgesToPolygons (std::vector< Edge2 > &vEdgesIn, std::vector< std::vector< Edge2 > > &vvPolygonsOut, std::vector< Edge2 > &vRemainingOut)

  Create polygons from a set of edges.
- void GEOM_FADE25D::getBorders (const std::vector< Triangle2 *> &vT, std::vector< Segment2 > &vBorderSegmentsOut)

  Get Borders.
- bool GEOM_FADE25D::sortRing (std::vector< Segment2 > &vRing)

  Sort a vector of Segments.
- bool GEOM_FADE25D::sortRingCCW (std::vector< Segment2 > &vRing)

  Sort a vector of Segments.
- FUNC_DECLSPEC Orientation2 GEOM_FADE25D::getOrientation2 (const Point2 *p0, const Point2 *p1, const Point2 *p2)

  Get the orientation of three points.
- FUNC_DECLSPEC Orientation2 GEOM_FADE25D::getOrientation2_mt (const Point2 *p0, const Point2 *p1, const Point2 *p2)

  Get Orientation2 (MT)

5.1.1 Detailed Description

5.1.2 Function Documentation

5.1.2.1 edgesToPolygons()

```
void GEOM_FADE25D::edgesToPolygons (
            std::vector< Edge2 > & vEdgesIn,
            std::vector< std::vector< Edge2 > > & vvPolygonsOut,
            std::vector< Edge2 > & vRemainingOut )
```

A number of methods in Fade returns an unorganized set of edges that delimit a certain area. But sometimes it is more beneficial to have these edges organized as a set of one or more polygons. This is the purpose of the present method.

Parameters

`in`	*vEdgesIn*	is a vector of oriented edges
`out`	*vvPolygonsOut*	contains one vector<Edge2> for each polygon found in the input data.
`out`	*vRemainingOut*	is used to return unusable remaining edges

The present function adds one vector<Edge2> to `vvPolygonsOut` for each polygon found in `vEdgesIn`. Each such polygon starts with the leftmost vertex (and when two or more vertices share the smallest x-coordiante then the one of them with the smallest y-coordinate is chosen). Edges that do not form a closed polygon are returned in `vRemainingOut`.

Note

An Edge2 object represents an edge of a triangle. Triangle corners are always counterclockwise (CCW) oriented. Thus outer polygons are CCW-oriented while hole-polygons are CW-oriented, see the figure.

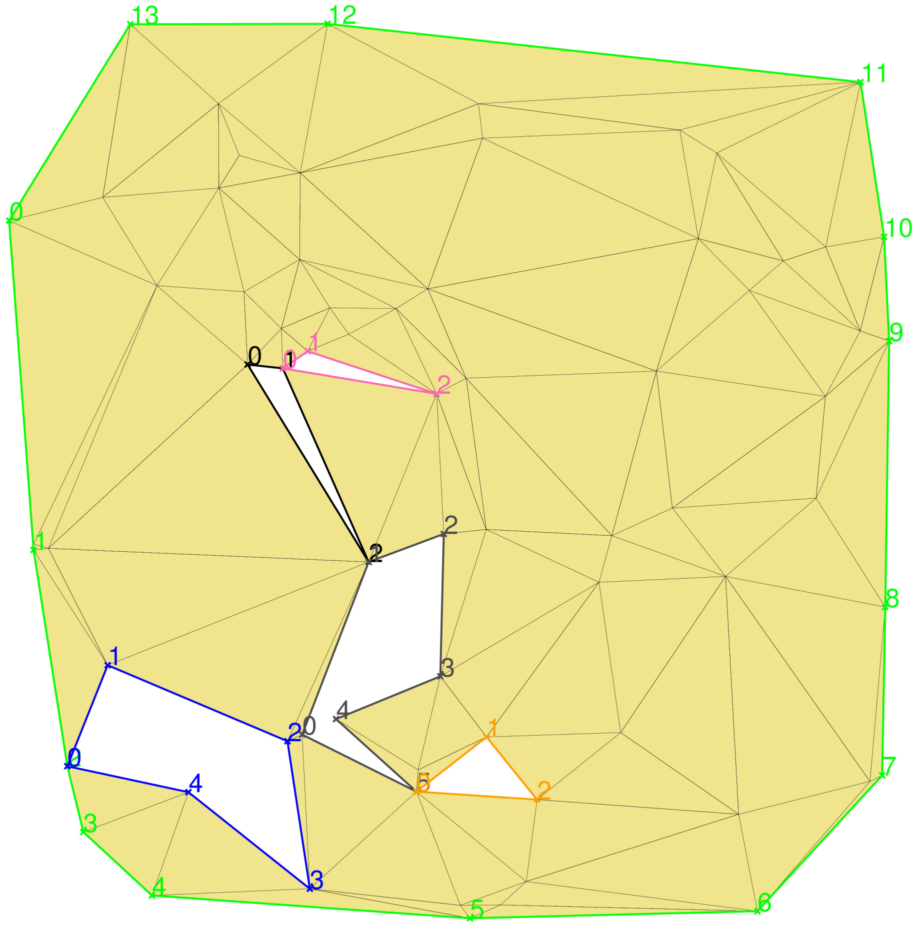


Figure 1 Polygons created by edgesToPolygons

5.1.2.2 fillHole() [1/3]

```
bool GEOM_FADE25D::fillHole (
            std::vector< std::pair< Segment2, Vector2 > > vPolygonSegments,
            bool bWithRefine,
            bool bVerbose,
            std::vector< Point2 > & vCornersOut )
```

This function was experimental and is now deprecated because 3D point cloud meshing has been moved to the WOF library.

Parameters

in	*vPolygonSegments*	contains the segments of a closed, simple input polygon along with normal vectors. The segments are counterclockwise oriented and ordered with respect to the surface to be created. Check twice, the orientation is very important. The normal vectors point in the direction of the thought surface at the segment i.e., if a hole is filled, the normal vector of an adjecent triangle is taken but if a T-joint is filled the normal vector should be the average normal of the two triangles at the edge.
in	*bWithRefine*	specifies if additional vertices shall be created. (bWithRefine=true is experimental, don't use currently)
in	*bVerbose*	specifies if warnings shall be printed to stdout
out	*vCornersOut*	contains the created fill triangles, 3 corners per triangle, counterclockwise oriented.

5.1.2.3 fillHole() [2/3]

```
bool GEOM_FADE25D::fillHole (
            std::vector< Point2 > & vMeshCorners,
            std::vector< Segment2 > & vPolygonSegments,
            bool bWithRefine,
            bool bVerbose,
            std::vector< Point2 > & vCornersOut )
```

This function was experimental and is now deprecated because 3D point cloud meshing has been moved to the WOF library.

Parameters

in	*vMeshCorners*	specifies the input mesh, 3 points per triangle in counterclockwise order.
in	*vPolygonSegments*	are the edges of the *closed* polygon to be triangulated.
in	*bWithRefine*	specifies if additional vertices shall be created (bWithRefine=true is experimental, don't use currently)
in	*bVerbose*	specifies if warnings shall be printed to stdout
out	*vCornersOut*	contains the created fill triangles, 3 corners per triangle, counterclockwise oriented.

5.1.2.4 fillHole() [3/3]

```
bool GEOM_FADE25D::fillHole (
            Mesh3 * pMesh,
            std::vector< Edge2 > & vPolygonEdges,
            bool bWithRefine,
            bool bVerbose,
            std::vector< Point2 > & vCornersOut )
```

This function was experimental and is now deprecated because 3D point cloud meshing has been moved to the WOF library.

Parameters

in	*pMesh*	
in	*vPolygonEdges*	are edges of the polygon to be triangulated. They must form a closed polygon in the mesh.
in	*bWithRefine*	specifies if additional vertices shall be created (Note: bWithRefine=true is experimental, don't use currently)
in	*bVerbose*	specifies if warnings shall be printed to stdout
out	*vCornersOut*	contains the created fill triangles, 3 corners per triangle, counterclockwise oriented.

5.1.2.5 getBorders()

```
void GEOM_FADE25D::getBorders (
```

```
const std::vector< Triangle2 *> & vT,
std::vector< Segment2 > & vBorderSegmentsOut )
```

Computes the border of the triangles in `vT`. The border consists of all edges having only one adjacent triangle in vT.

Parameters

in	*vT*	are the input triangles
out	*vBorderSegmentsOut*	is used to return all border segments

5.1.2.6 getNormalVector()

```
Vector2 GEOM_FADE25D::getNormalVector (
        const Point2 & p0,
        const Point2 & p1,
        const Point2 & p2,
        bool & bOK )
```

Returns the normalized normal vector of the triangle defined by the three input points `p0`, `p1`, `p2`.

Parameters

in	*p0,p1,p2*	When these points are counterclockwise (CCW) oriented then the resulting normal vector points towards the viewer.
out	*bOK*	returns true for valid results. When the plane defined by `p0`, `p1`, `p2` is degenerate, bOK returns false.

5.1.2.7 getOrientation2()

```
FUNC_DECLSPEC Orientation2 GEOM_FADE25D::getOrientation2 (
        const Point2 * p0,
        const Point2 * p1,
        const Point2 * p2 )
```

This function returns the *exact* orientation of the points `p0`, `p1`, `p2` Possible values are
ORIENTATION2_COLLINEAR if `p0`, `p1`, `p2` are located on a line,
ORIENTATION2_CCW if `p0`, `p1`, `p2` are counterclockwise oriented
ORIENTATION2_CW if `p0`, `p1`, `p2` are clockwise oriented
Not thread-safe but a bit faster than the thread-safe version

5.1.2.8 getOrientation2_mt()

```
FUNC_DECLSPEC Orientation2 GEOM_FADE25D::getOrientation2_mt (
        const Point2 * p0,
        const Point2 * p1,
        const Point2 * p2 )
```

See also

getOrientation2(const Point2∗ p0,const Point2∗ p1,const Point2∗ p2)

This version is thread-safe.

5.1.2.9 getUndirectedEdges()

```
void GEOM_FADE25D::getUndirectedEdges (
            std::vector< Triangle2 *> & vT,
            std::vector< Edge2 > & vUndirectedEdgesOut )
```

A unique set of edges of `vT` is returned.

5.1.2.10 isSimplePolygon()

```
bool GEOM_FADE25D::isSimplePolygon (
            std::vector< Segment2 > & vSegments )
```

Parameters

in	*vSegments*	specifies segments to be checked. Degenerate segments (0-length) are ignored.

Returns

true when `vSegments` contains a closed polygon without selfintersections. False otherwise.

5.1.2.11 pointsToPolyline()

```
void GEOM_FADE25D::pointsToPolyline (
            std::vector< Point2 > & vInPoints,
            bool bClose,
            std::vector< Segment2 > & vOutSegments )
```

Turns a vector of points (p0,p1,p2,...pm,pn) into a vector of segments ((p0,p1),(p1,p2),...,(pm,pn)). In case that bClose is true an additional segment (pn,p0) is constructed. Degenerate segments are ignored. Selfintersections of the polyline are not checked.

Parameters

in	*vInPoints*	
in	*bClose*	specifies whether a closing segment shall be constructed
out	*vOutSegments*	is where the output segments are stored

5.1.2.12 sortRing()

```
bool GEOM_FADE25D::sortRing (
            std::vector< Segment2 > & vRing )
```

The segments in vRing are reoriented and sorted such that subsequent segments join at the endpoints.

5.1.2.13 sortRingCCW()

```
bool GEOM_FADE25D::sortRingCCW (
            std::vector< Segment2 > & vRing )
```

The segments in vRing are reoriented and sorted such that the resulting polygon is counterclockwise oriented and subsequent segments join at the endpoints.

5.2 Version Information

Functions

- std::string GEOM_FADE25D::getFade2DVersion ()

    Get the Fade2D version string.
- FUNC_DECLSPEC int GEOM_FADE25D::getMajorVersionNumber ()

    Get the major version number.
- FUNC_DECLSPEC int GEOM_FADE25D::getMinorVersionNumber ()

    Get the minor version number.
- FUNC_DECLSPEC int GEOM_FADE25D::getRevisionNumber ()

    Get the revision version number.
- FUNC_DECLSPEC bool GEOM_FADE25D::isRelease ()

    Check if a RELEASE or a DEBUG version is used.

5.2.1 Detailed Description

5.3 File I/O

Functions

- FUNC_DECLSPEC bool GEOM_FADE25D::writePointsASCII (const char ∗filename, const std::vector< Point2 ∗> &vPointsIn)

    Write points to an ASCII file.
- bool GEOM_FADE25D::writePointsASCII (const char ∗filename, const std::vector< Point2 > &vPointsIn)

    Write points to an ASCII file.
- FUNC_DECLSPEC bool GEOM_FADE25D::readXY (const char ∗filename, std::vector< Point2 > &vPoints↵ Out)

    Read (x y) points.
- FUNC_DECLSPEC bool GEOM_FADE25D::readXYZ (const char ∗filename, std::vector< Point2 > &v↵ PointsOut)

    Read (x y z) points.
- bool GEOM_FADE25D::writePointsBIN (const char ∗filename, std::vector< Point2 > &vPointsIn)

    Write points to a binary file.
- bool GEOM_FADE25D::writePointsBIN (const char ∗filename, std::vector< Point2 ∗> &vPointsIn)

    Write points to a binary file.
- bool GEOM_FADE25D::readPointsBIN (const char ∗filename, std::vector< Point2 > &vPointsIn)

    Read points from a binary file.
- bool GEOM_FADE25D::writeSegmentsBIN (const char ∗filename, std::vector< Segment2 > &vSegmentsIn)

    Write segments to a binary file.
- bool GEOM_FADE25D::readSegmentsBIN (const char ∗filename, std::vector< Segment2 > &vSegments↵ Out)

    Read segments from a binary file.

5.3.1 Detailed Description

5.3.2 Function Documentation

5.3.2.1 readPointsBIN()

```
bool GEOM_FADE25D::readPointsBIN (
            const char * filename,
            std::vector< Point2 > & vPointsIn )
```

Reads points from a binary file.

See also

writePointsBIN()

5.3.2.2 readSegmentsBIN()

```
bool GEOM_FADE25D::readSegmentsBIN (
            const char * filename,
            std::vector< Segment2 > & vSegmentsOut )
```

Reads segments from a binary file of type 21 or 31

See also

writeSegmentsBIN()

5.3.2.3 readXY()

```
FUNC_DECLSPEC bool GEOM_FADE25D::readXY (
            const char * filename,
            std::vector< Point2 > & vPointsOut )
```

Reads points from an ASCII file. Expected file format: Two coordinates (x y) per line, whitespace separated.

The z coordinate is set to 0.

5.3.2.4 readXYZ()

```
FUNC_DECLSPEC bool GEOM_FADE25D::readXYZ (
            const char * filename,
            std::vector< Point2 > & vPointsOut )
```

Reads points from an ASCII file. Expected file format: Three coordinates (x y z) per line, whitespace separated.

5.3.2.5 writePointsASCII() [1/2]

```
FUNC_DECLSPEC bool GEOM_FADE25D::writePointsASCII (
            const char * filename,
            const std::vector< Point2 *> & vPointsIn )
```

Writes points to an ASCII file, three coordinates (x y z) per line, whitespace separated.

Note

Data exchange through ASCII files is easy and convenient but floating point coordinates are not necessarily exact when represented as decimal numbers. If the tiny rounding errors can't be accepted in your setting you are advised to write binary files, (use writePointsBIN())

5.3.2.6 writePointsASCII() [2/2]

```
bool GEOM_FADE25D::writePointsASCII (
            const char * filename,
            const std::vector< Point2 > & vPointsIn )
```

Write points to an ASCII file

See also

readPointsASCII()

5.3.2.7 writePointsBIN() [1/2]

```
bool GEOM_FADE25D::writePointsBIN (
            const char * filename,
            std::vector< Point2 > & vPointsIn )
```

File format:
int filetype (30)
size_t numPoints (`vPointsIn.size()`)
double x0
double y0
double z0
...
double xn
double yn
double zn

Note

Since version 1.64 the binary file format written by 32-bit machines is identical with the file format of x64 machines i.e., the numPoints value is always 8 bytes, not 4. This change affects only 32-bit programs.

5.3.2.8 writePointsBIN() [2/2]

```
bool GEOM_FADE25D::writePointsBIN (
            const char * filename,
            std::vector< Point2 *> & vPointsIn )
```

Writes points to a binary file

See also

readPointsBIN()

5.3.2.9 writeSegmentsBIN()

```
bool GEOM_FADE25D::writeSegmentsBIN (
            const char * filename,
            std::vector< Segment2 > & vSegmentsIn )
```

Binary file format:
int filetype (31)
size_t numSegments (`vSegmentsIn.size()`)
double x0_source
double y0_source
double z0_source
double x0_target
double y0_target
double z0_target
...
double xn_source
double yn_source
double zn_source
double xn_target
double yn_target
double zn_target

Note

Since version 1.64 the binary file format written by 32-bit machines is identical with the file format of x64 machines i.e., the numSegments value is always 8 bytes, not 4. This change affects only 32-bit programs.

See also

readSegmentsBIN()

5.4 Test Data Generators

Functions

- FUNC_DECLSPEC void GEOM_FADE25D::generateRandomNumbers (size_t num, double min, double max, std::vector< double > &vRandomNumbersOut, unsigned int seed=0)

  Generate random numbers.
- FUNC_DECLSPEC void GEOM_FADE25D::generateRandomPoints (size_t numRandomPoints, double min, double max, std::vector< Point2 > &vRandomPointsOut, unsigned int seed=0)

  Generate random points.
- FUNC_DECLSPEC void GEOM_FADE25D::generateRandomPoints3D (size_t numRandomPoints, double min, double max, std::vector< Point2 > &vRandomPointsOut, unsigned int seed=0)

  Generate random points with height.
- FUNC_DECLSPEC void GEOM_FADE25D::generateRandomPolygon (size_t numSegments, double min, double max, std::vector< Segment2 > &vPolygonOut, unsigned int seed=0)

  Generate a random simple polygon.
- FUNC_DECLSPEC void GEOM_FADE25D::generateRandomSegments (size_t numSegments, double min, double max, double maxLen, std::vector< Segment2 > &vSegmentsOut, unsigned int seed)

  Generate random line segments.
- FUNC_DECLSPEC void GEOM_FADE25D::generateSineSegments (int numSegments, int numPeriods, double xOffset, double yOffset, double xFactor, double yFactor, bool bSwapXY, std::vector< Segment2 > &vSineSegmentsOut)

  Generate segments from a sine function.
- FUNC_DECLSPEC void GEOM_FADE25D::generateCircle (int numPoints, double x, double y, double z, double radiusX, double radiusY, std::vector< Point2 > &vCirclePointsOut)

  Generate a circle.
- FUNC_DECLSPEC void GEOM_FADE25D::generateRandomSurfacePoints (size_t numX, size_t numY, size_t numCenters, double xmin, double ymin, double zmin, double xmax, double ymax, double zmax, std::vector< Point2 > &vSurfacePointsOut, unsigned int seed)

  Generate a random surface.
- FUNC_DECLSPEC void **GEOM_FADE25D::shear** (std::vector< Point2 > &vPointsInOut, double shearX, double shearY)

5.4.1 Detailed Description

Generate random polygons and other test objects

Theory, careful programming and automated software stress tests. Neither of them can replace the other one. Testing with random data helps to discover errors early. Fade provides random object generators for your automated software stress tests:

- Random simple polygons
- Random segments
- Random point clouds
- Random numbers.
- Polylines from sine functions

If you discover an error in your software you must be able to reproduce the input data that has triggered your bug. For this reason the random object generators take a seed value to initialize the internal random number generators. A certain seed value always leads to the same sequence of objects. Only when the special seed value 0 is used then the random number generators are initialized from the system time.

5.4.2 Function Documentation

5.4.2.1 generateCircle()

```
FUNC_DECLSPEC void GEOM_FADE25D::generateCircle (
            int numPoints,
            double x,
            double y,
            double z,
            double radiusX,
            double radiusY,
            std::vector< Point2 > & vCirclePointsOut )
```

Returns points on a circle centered at the given coordinates

5.4.2.2 generateRandomNumbers()

```
FUNC_DECLSPEC void GEOM_FADE25D::generateRandomNumbers (
            size_t num,
            double min,
            double max,
            std::vector< double > & vRandomNumbersOut,
            unsigned int seed = 0 )
```

Parameters

	num	Number of random numbers to be generated
	min	Lower bound
	max	Upper bound
out	*vRandomNumbersOut*	is the output vector
	seed	initializes the random number generator RNG (default: 0...mapped to a random seed, other values: constant initialization)

Note

Reproducable random numbers are often desirable when software is tested with random geometric constructions. Thus each seed value different from *0* leads to its own, reproducible, output sequence. In contrast the `seed` value *0* is mapped to random initialization of the RNG. In this case the RNG will produce a different output sequence each time it is called.

5.4.2.3 generateRandomPoints()

```
FUNC_DECLSPEC void GEOM_FADE25D::generateRandomPoints (
            size_t numRandomPoints,
            double min,
            double max,
            std::vector< Point2 > & vRandomPointsOut,
            unsigned int seed = 0 )
```

Parameters

	numRandomPoints	Number of points to be generated
	min	Lower bound (x,y)
	max	Upper bound (x,y)
out	*vRandomPointsOut*	is the output vector
	seed	initializes the random number generator RNG (default: 0...mapped to a random seed, other values: constant initialization)

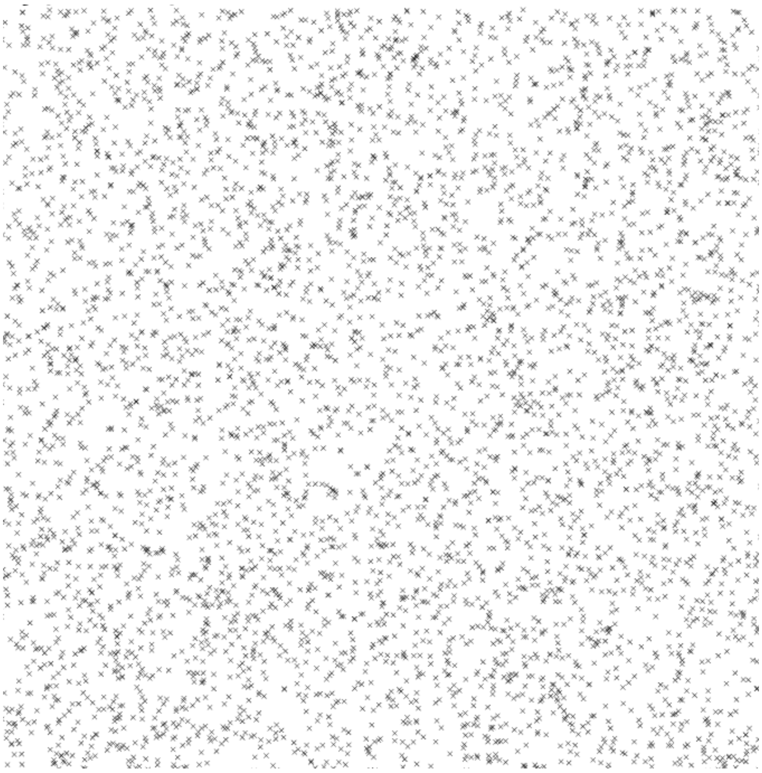


Figure 2 Point generator

5.4.2.4 generateRandomPoints3D()

```
FUNC_DECLSPEC void GEOM_FADE25D::generateRandomPoints3D (
            size_t numRandomPoints,
            double min,
            double max,
            std::vector< Point2 > & vRandomPointsOut,
            unsigned int seed = 0 )
```

Parameters

	numRandomPoints	Number of points to be generated
	min	Lower bound (x,y)
	max	Upper bound (x,y)
out	*vRandomPointsOut*	is the output vector
	seed	initializes the random number generator RNG (default: 0...mapped to a random seed, other values: constant initialization)

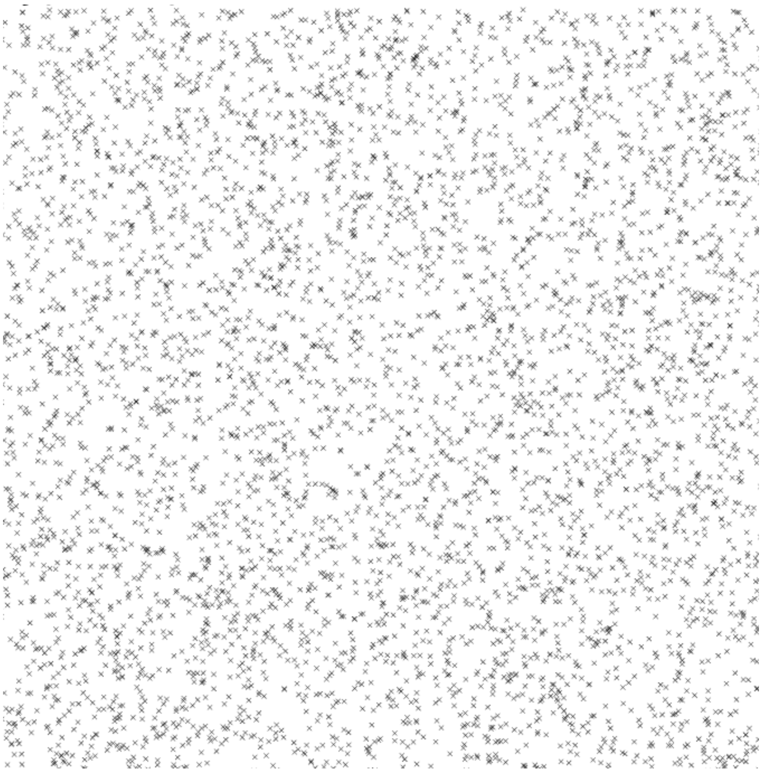


Figure 3 Point generator

5.4.2.5 generateRandomPolygon()

```
FUNC_DECLSPEC void GEOM_FADE25D::generateRandomPolygon (
            size_t numSegments,
            double min,
            double max,
            std::vector< Segment2 > & vPolygonOut,
            unsigned int seed = 0 )
```

Parameters

	numSegments	Number of segments to be generated
	min	Lower bound (x,y)
	max	Upper bound (x,y)
out	*vPolygonOut*	is the output vector
	seed	initializes the random number generator RNG (default: 0...mapped to a random seed, other values: constant initialization)

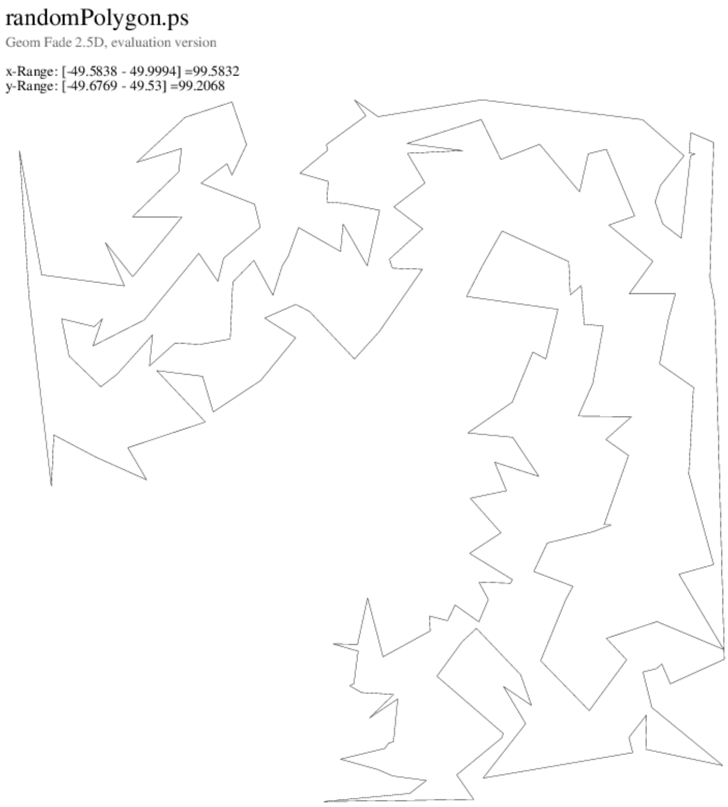


Figure 4 Polygon generator: Random simple polygon

5.4.2.6 generateRandomSegments()

```
FUNC_DECLSPEC void GEOM_FADE25D::generateRandomSegments (
            size_t numSegments,
            double min,
            double max,
            double maxLen,
            std::vector< Segment2 > & vSegmentsOut,
            unsigned int seed )
```

Parameters

	numSegments	Number of segments to be generated
	min	Lower bound (x,y)
	max	Upper bound (x,y)
	maxLen	Maximal segment length
out	*vSegmentsOut*	is the output vector
	seed	initializes the random number generator RNG (default: 0...mapped to a random seed, other values: constant initialization)

Figure 5 Segment generator: Random line segments

5.4.2.7 generateRandomSurfacePoints()

```
FUNC_DECLSPEC void GEOM_FADE25D::generateRandomSurfacePoints (
            size_t numX,
            size_t numY,
            size_t numCenters,
            double xmin,
            double ymin,
            double zmin,
            double xmax,
            double ymax,
            double zmax,
            std::vector< Point2 > & vSurfacePointsOut,
            unsigned int seed )
```

Parameters

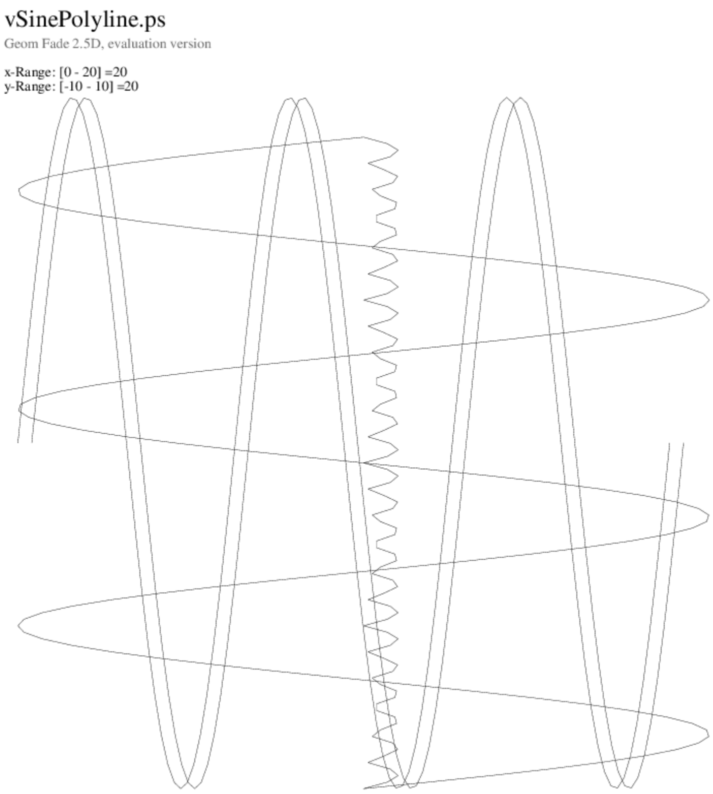
	numX,numY	specifies the grid size and must be >1. numX∗numY points are created
	numCenters	defines the number of extreme points (must be >0)
	xmin,ymin,zmin,xmax,ymax,zmax	specifies the geometric bounds
out	*vSurfacePointsOut*	is the output vector
	seed	initializes the random number generator RNG (default: 0...mapped to a random seed, other values: constant initialization)

5.4.2.8 generateSineSegments()

```
FUNC_DECLSPEC void GEOM_FADE25D::generateSineSegments (
            int numSegments,
            int numPeriods,
            double xOffset,
            double yOffset,
            double xFactor,
            double yFactor,
            bool bSwapXY,
            std::vector< Segment2 > & vSineSegmentsOut )
```

Parameters

	numSegments	Number of segments to be generated
	numPeriods	Number of periods of the sine function
	xOffset	Offset of the output x-coordinates
	yOffset	Offset of the output y-coordinates
	xFactor	Factor to scale the sine function in x direction
	yFactor	Factor to scale the sine function in y direction
	bSwapXY	Swap the x and y coordinate of the function
out	*vSineSegmentsOut*	is the output vector



•

Figure 6 Polyline generator: Polylines from sine functions

6 Class Documentation

6.1 GEOM_FADE25D::Bbox2 Class Reference

Bbox2 is an axis aligned 2D bounding box.

```
#include <Bbox2.h>
```

Public Member Functions

- Bbox2 (GeomTest ∗pGeomTest_=NULL)

  Constructor.
- Bbox2 (std::vector< Point2 >::const_iterator start_it, std::vector< Point2 >::const_iterator end_it, GeomTest ∗pGeomTest_=NULL)

  Constructor.
- bool isValid () const

  Check if the bounds are valid.
- void getCorners (std::vector< Point2 > &vBoxCorners) const

  Get corners.
- void getOffsetCorners (double offset, std::vector< Point2 > &vBoxCorners) const

  Get offset corners.
- bool doIntersect (const Bbox2 &other) const

  Check intersection.
- bool add (std::vector< Point2 ∗>::const_iterator start_it, std::vector< Point2 ∗>::const_iterator end_it)

  Add points.
- bool add (std::vector< Point2 >::const_iterator start_it, std::vector< Point2 >::const_iterator end_it)

  Add points.
- bool add (size_t numPoints, double ∗coordinates)

  Add points.
- bool add (const Point2 &p)

  Add a point.
- bool isInBox (const Point2 &p) const

  Point-in-Box Test.
- Point2 computeCenter () const

  Compute the 2D midpoint.
- Bbox2 operator+ (const Bbox2 &b)

  Add a bounding box.
- Point2 getMinPoint () const

  Get the min point.
- Point2 getMaxPoint () const

  Get the max point.
- double getMinCoord () const

  Get minimum coordinate.
- double getMaxCoord () const

  Get maximum coordinate.
- double getRangeX () const

  Get x-range.
- double getRangeY () const

  Get y-range.
- double getMaxRange () const

Get max range.
- double get_minX () const
  Get minX.
- double get_minY () const
  Get minY.
- double get_maxX () const
  Get maxX.
- double get_maxY () const
  Get maxY.
- void getBounds (double &minX_, double &maxX_, double &minY_, double &maxY_) const
  Get bounds.
- void doubleTheBox ()
  Double the box.
- void setMinX (double val)
  Set minX.
- void setMaxX (double val)
  Set maxX.
- void setMinY (double val)
  Set minY.
- void setMaxY (double val)
  Set maxY.
- void **enlargeRanges** (double factor)
- void inflateIfDegenerate (double val)
  Inflate if Degenerate.

Protected Member Functions

- void **treatPointForValidBox** (const Point2 &p)
- void **treatPointForInvalidBox** (const Point2 &p)

Protected Attributes

- double **minX**
- double **minY**
- double **maxX**
- double **maxY**
- bool **bValid**
- GeomTest ∗ **pGeomTest**

Friends

- std::ostream & **operator**<< (std::ostream &stream, Bbox2 &pC)

6.1.1 Detailed Description

6.1.2 Constructor & Destructor Documentation

6.1.2.1 Bbox2() [1/2]

```
GEOM_FADE25D::Bbox2::Bbox2 (
            GeomTest * pGeomTest_ = NULL )  [inline], [explicit]
```

Minimum bounds are initialized to DBL_MAX. Maximum bounds are initialized to -DBL_MAX. Box is not valid yet

6.1.2.2 Bbox2() [2/2]

```
GEOM_FADE25D::Bbox2::Bbox2 (
            std::vector< Point2 >::const_iterator start_it,
            std::vector< Point2 >::const_iterator end_it,
            GeomTest * pGeomTest_ = NULL )  [inline]
```

Bounds initialized to the minimal bounding box of the iterator range of points.

6.1.3 Member Function Documentation

6.1.3.1 add() [1/4]

```
bool GEOM_FADE25D::Bbox2::add (
            std::vector< Point2 *>::const_iterator start_it,
            std::vector< Point2 *>::const_iterator end_it )  [inline]
```

Extends the 2D bounding box if required.

Returns

true if the bounding box changes, false otherwise

6.1.3.2 add() [2/4]

```
bool GEOM_FADE25D::Bbox2::add (
            std::vector< Point2 >::const_iterator start_it,
            std::vector< Point2 >::const_iterator end_it )  [inline]
```

Extends the 2D bounding box if required.

Returns

true if the bounding box changes, false otherwise

6.1.3.3 add() [3/4]

```
bool GEOM_FADE25D::Bbox2::add (
            size_t numPoints,
            double * coordinates ) [inline]
```

Extends the 2D bounding box if required.

Returns

true if the bounding box changes, false otherwise

6.1.3.4 add() [4/4]

```
bool GEOM_FADE25D::Bbox2::add (
            const Point2 & p ) [inline]
```

Extends the 2D bounding box if required.

Returns

true if the bounding box changes, false otherwise

6.1.3.5 computeCenter()

```
Point2 GEOM_FADE25D::Bbox2::computeCenter ( ) const
```

6.1.3.6 doIntersect()

```
bool GEOM_FADE25D::Bbox2::doIntersect (
            const Bbox2 & other ) const
```

Two valid bounding boxes intersect if they share at least one point in the XY plane.

6.1.3.7 doubleTheBox()

```
void GEOM_FADE25D::Bbox2::doubleTheBox ( )
```

Changes the bounds such that the box grows in each direction by half the previous range

6.1.3.8 get_maxX()

```
double GEOM_FADE25D::Bbox2::get_maxX ( ) const [inline]
```

Returns

maxX

6.1.3.9 get_maxY()

```
double GEOM_FADE25D::Bbox2::get_maxY ( ) const [inline]
```

Returns

maxY

6.1.3.10 get_minX()

```
double GEOM_FADE25D::Bbox2::get_minX ( ) const [inline]
```

Returns

minX

6.1.3.11 get_minY()

```
double GEOM_FADE25D::Bbox2::get_minY ( ) const [inline]
```

Returns

minY

6.1.3.12 getBounds()

```
void GEOM_FADE25D::Bbox2::getBounds (
            double & minX_,
            double & maxX_,
            double & minY_,
            double & maxY_ ) const
```

6.1.3.13 getCorners()

```
void GEOM_FADE25D::Bbox2::getCorners (
            std::vector< Point2 > & vBoxCorners ) const
```

Convenience function: Returns the 4 corners of the bounding box

6.1.3.14 getMaxCoord()

```
double GEOM_FADE25D::Bbox2::getMaxCoord ( ) const [inline]
```

Returns

the largest coordinate value, i.e. max(maxX,maxY)

6.1.3.15 getMaxPoint()

```
Point2 GEOM_FADE25D::Bbox2::getMaxPoint ( ) const [inline]
```

Returns

the 2D corner point with the maximum coordinates, the z-coordinate is set to 0

6.1.3.16 getMaxRange()

```
double GEOM_FADE25D::Bbox2::getMaxRange ( ) const [inline]
```

Returns

the largest range, i.e. max(getRangeX(),getRangeY())

6.1.3.17 getMinCoord()

```
double GEOM_FADE25D::Bbox2::getMinCoord ( ) const [inline]
```

Returns

the smallest coordinate value, i.e. min(minX,minY)

6.1.3.18 getMinPoint()

```
Point2 GEOM_FADE25D::Bbox2::getMinPoint ( ) const [inline]
```

Returns

the corner point with the minimum coordinates, the z-coordinate is set to 0

6.1.3.19 getOffsetCorners()

```
void GEOM_FADE25D::Bbox2::getOffsetCorners (
            double offset,
            std::vector< Point2 > & vBoxCorners ) const
```

Convenience function: Returns the 4 corners of an enlarged box. The box es enlarged by `offset` in each direction

6.1.3.20 getRangeX()

```
double GEOM_FADE25D::Bbox2::getRangeX ( ) const  [inline]
```

Returns

maxX-minX

6.1.3.21 getRangeY()

```
double GEOM_FADE25D::Bbox2::getRangeY ( ) const  [inline]
```

Returns

maxY-minY

6.1.3.22 inflateIfDegenerate()

```
void GEOM_FADE25D::Bbox2::inflateIfDegenerate (
            double val )  [inline]
```

When only one point has been added to Bbox2 or when all points have the same x- and/or y- coordinates then Bbox2 is degenerate. This is a valid state but sometimes undesireable. The present method inflates the Bbox2 by adding /p val to maxX and/or maxY.

6.1.3.23 isInBox()

```
bool GEOM_FADE25D::Bbox2::isInBox (
            const Point2 & p ) const
```

Returns

true if minX <= p.x() <=maxX and minY <= p.y() <=maxY or false otherwise.

6.1.3.24 isValid()

```
bool GEOM_FADE25D::Bbox2::isValid ( ) const  [inline]
```

The bounds are valid when at least one point has been added or when set-methods have been used to set minX<=maxX and minY<=maxY

6.1.3.25 operator+()

```
Bbox2 GEOM_FADE25D::Bbox2::operator+ (
            const Bbox2 & b )
```

Extends the 2D bounding box if required.

Returns

the resulting bounding box

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

- Bbox2.h

6.2 GEOM_FADE25D::CAF_Component Class Reference

CAF_Component stands for CUT AND FILL COMPONENT. It represents a connected area of the surface.

`#include <CAF_Component.h>`

Public Member Functions

- **CAF_Component** (std::vector< Triangle2 ∗> &vT_, std::map< Point2 ∗, std::pair< double, double > > &mVtx2BeforeAfter_, int label_)
- CAFTYP getCAFType () const
  Get Cut&Fill-Type.
- double getVolume () const
  Get the volume.
- int getLabel () const
  Get label.
- void getTriangles (std::vector< Triangle2 ∗> &vTrianglesOut) const
  Get Triangles.
- void getBorder (std::vector< Segment2 > &vBorderSegments) const
  Get border.

Protected Member Functions

- void **init** (std::map< Point2 ∗, std::pair< double, double > > &mVtx2BeforeAfter)
- void **showGeomview** (const std::string &name, std::vector< Point2 > ∗pvBeforeT, std::vector< Point2 > ∗pvAfterT, std::vector< Point2 > ∗pvWallT) const
- void **setVolume** (std::vector< Point2 > ∗pvBeforeT, std::vector< Point2 > ∗pvAfterT, std::vector< Point2 > ∗pvWallT)

Protected Attributes

- std::vector< Triangle2 ∗ > ∗ **pVT**
- CAFTYP **caftype**
- double **volume**
- int **label**

6.2.1 Detailed Description

A CAF_Component object represents a connected part of the surface such that:

- the first surface is below the second one (CAFTYP=CT_FILL) or
- the first surface is above the second one (CAFTYP=CT_CUT) or
- the first surface corresponds to the second one (CAFTYP=CT_NULL)

6.2.2 Member Function Documentation

6.2.2.1 getBorder()

```
void GEOM_FADE25D::CAF_Component::getBorder (
            std::vector< Segment2 > & vBorderSegments ) const
```

Returns

border segments of the present component in no particular order

6.2.2.2 getCAFType()

```
CAFTYP GEOM_FADE25D::CAF_Component::getCAFType ( ) const
```

Returns

CT_CUT, CT_FILL or CT_NULL

- CT_CUT means that earth must be digged off to turn the first surface into the second one,
- CT_FILL means that earth must be added.
- CT_NULL is returned when the first surface corresponds to the second one.

6.2.2.3 getLabel()

```
int GEOM_FADE25D::CAF_Component::getLabel ( ) const
```

Returns

the component label

Components are consecutively numbered.

6.2.2.4 getTriangles()

```
void GEOM_FADE25D::CAF_Component::getTriangles (
            std::vector< Triangle2 *> & vTrianglesOut ) const
```

Returns

the triangles of the present component. The z-coordinates of their corners correspond to the height differences between the two input surfaces.

Parameters

out	*vTrianglesOut*	is used to return the triangles

6.2.2.5 getVolume()

```
double GEOM_FADE25D::CAF_Component::getVolume ( ) const
```

Returns

the volume of the present component.

Note

The volume is an absolute value. Use getCAFType() to determine if it is a CUT, FILL or ZERO volume.

Warning

The computations are unitless but you must make sure that the x, y, and z-coordinate are given in the same unit.

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

- CAF_Component.h

6.3 GEOM_FADE25D::Circle2 Class Reference

Circle.

```
#include <Circle2.h>
```

Public Member Functions

- Circle2 (double x, double y, double sqRadius_)

  Constructor.
- Circle2 (const Point2 &center_, double sqRadius_)

  Constructor.
- double getRadius ()

  Get the radius of the circle.
- double getSqRadius ()

  Get the squared radius of the circle.
- Point2 getCenter ()

  Get the center of the circle.

Protected Attributes

- Point2 **center**
- double **sqRadius**

Friends

- std::ostream & **operator**<< (std::ostream &stream, Circle2 b)

6.3.1 Detailed Description

6.3.2 Constructor & Destructor Documentation

6.3.2.1 Circle2() [1/2]

```
GEOM_FADE25D::Circle2::Circle2 (
            double x,
            double y,
            double sqRadius_ )
```

Parameters

x	is x-coordinate of the center
y	is y-coordinate of the center
*sq←→ Radius_*	is the squared radius of the circle

Warning

The method expects the *squared* radius

6.3.2.2 Circle2() [2/2]

```
GEOM_FADE25D::Circle2::Circle2 (
            const Point2 & center_,
            double sqRadius_ )
```

Parameters

*center_*	center of the circle
*sq←→ Radius_*	squared radius of the circle

Warning

The method expects the *squared* radius

6.3.3 Member Function Documentation

6.3.3.1 getCenter()

```
Point2 GEOM_FADE25D::Circle2::getCenter ( )
```

Returns

a Point2 which represents the center

6.3.3.2 getRadius()

```
double GEOM_FADE25D::Circle2::getRadius ( )
```

Returns

the radius

6.3.3.3 getSqRadius()

```
double GEOM_FADE25D::Circle2::getSqRadius ( )
```

Returns

the squared radius

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

- Circle2.h

6.4 GEOM_FADE25D::Color Class Reference

Color.

```
#include <Color.h>
```

Public Member Functions

- Color (double r_, double g_, double b_, double width_, bool bFill_=false)
- Color (Colorname c, float width_=0.001, bool bFill_=false)
- bool **operator**< (const Color &other) const
- bool **operator!=** (const Color &other) const
- bool **operator==** (const Color &other) const

Static Public Member Functions

- static Colorname **getNextColorName** ()

Public Attributes

- float r

  Red.
- float g

  Green.
- float b

  Blue.
- float width

  Linewidth.
- bool bFill

  Fill the shape or not.

Static Public Attributes

- static size_t **currentColorName**

Friends

- std::ostream & **operator**<< (std::ostream &stream, const Color &c)

6.4.1 Detailed Description

See also

Visualizer2

6.4.2 Constructor & Destructor Documentation

6.4.2.1 Color() [1/2]

```
GEOM_FADE25D::Color::Color (
            double r_,
            double g_,
            double b_,
            double width_,
            bool bFill_ = false )
```

Parameters

*r_*	red
*g_*	green
*b_*	blue
*width_*	linewidth
*bFill_*	fill (default: *false*)

Note

bFill_=true has two meanings: Objects that can be filled (Triangle2, Circle2) are filled with the rgb-color but line segments get x-marks at their endpoints.

6.4.2.2 Color() [2/2]

```
GEOM_FADE25D::Color::Color (
            Colorname c,
            float width_ = 0.001,
            bool bFill_ = false )
```

For convenience predefined colors can be used.

Parameters

c	is a predefined color name
*width_*	linewidth (default: *0.001*)
*bFill_*	fill (default: *false*)

Note

bFill_=true has two meanings: Objects that can be filled (Triangle2, Circle2) are filled with the rgb-color but line segments get x-marks at their endpoints.

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

- Color.h

6.5 GEOM_FADE25D::ConstraintGraph2 Class Reference

ConstraintGraph2 is a set of Constraint Edges (ConstraintSegment2)

```
#include <ConstraintGraph2.h>
```

Public Member Functions

- bool isPolygon () const

    Does the constraint graph form a closed polygon?
- bool isOriented () const

    Are the segments of the constraint graph oriented?
- void getPolygonVertices (std::vector< Point2 *> &vVertices_)

    Get the vertices of the constraint segments.
- ConstraintInsertionStrategy getInsertionStrategy () const

    Get the constraint insertion strategy.
- bool isConstraint (Point2 *p0, Point2 *p1) const

    Check if an edge is a constraint.
- bool isConstraint (ConstraintSegment2 *pCSeg) const

    Check if a ConstraintSegment2 is a member.
- void show (const std::string &name)

    Visualization.
- void show (Visualizer2 *pVis, const Color &color)

    Visualization.
- void getOriginalConstraintSegments (std::vector< ConstraintSegment2 *> &vConstraintSegments_) const

    Get the original ConstraintSegment2 objects.
- void getChildConstraintSegments (std::vector< ConstraintSegment2 *> &vConstraintSegments_) const

    Get child ConstraintSegment2 objects.
- Dt2 * getDt2 ()
- void getDirectChildren (ConstraintSegment2 *pParent, ConstraintSegment2 *&pChild0, ConstraintSegment2 *&pChild1)

    Get direct children.
- bool isReverse (ConstraintSegment2 *pCSeg) const
- bool makeDelaunay (double minLength)

Protected Attributes

- Dt2 * **pDt2**
- GeomTest * **pGeomPredicates**
- ConstraintInsertionStrategy **cis**
- std::vector< ConstraintSegment2 * > **vCSegParents**
- bool **bIsPolygon**
- std::map< ConstraintSegment2 *, bool, func_ltDerefPtr< ConstraintSegment2 * > > **mCSegReverse**
- std::map< Point2 *, size_t > **mSplitPointNum**
- bool **bIsOriented**

6.5.1 Detailed Description

See also

Fade_2D::createConstraint()

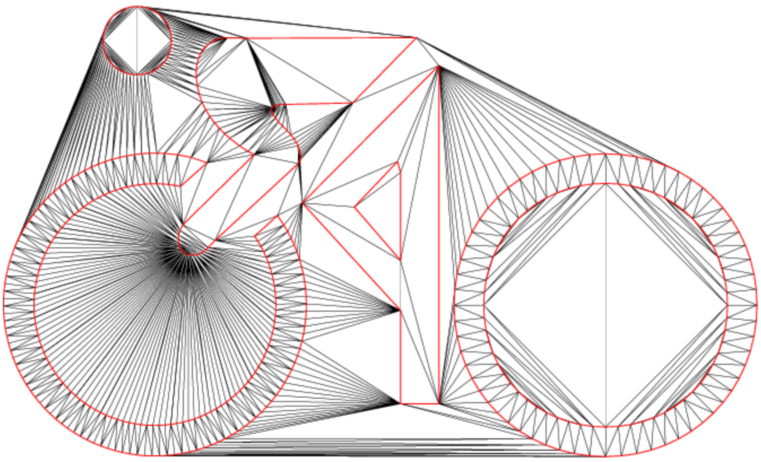


Figure 7 Constraint Delaunay triangulation

6.5.2 Member Function Documentation

6.5.2.1 getChildConstraintSegments()

```
void GEOM_FADE25D::ConstraintGraph2::getChildConstraintSegments (
            std::vector< ConstraintSegment2 *> & vConstraintSegments_ ) const
```

Returns the current constraint segments, i.e., the original ones or, if splitted, their child segments.

6.5.2.2 getDirectChildren()

```
void GEOM_FADE25D::ConstraintGraph2::getDirectChildren (
            ConstraintSegment2 * pParent,
            ConstraintSegment2 *& pChild0,
            ConstraintSegment2 *& pChild1 )
```

Parameters

in	*pParent*	is a ConstraintSegment that may have been splitted
out	*pChild0,pChild1*	are the direct child segments of `pParent`. They can be alive or dead (splitted).

The children are returned in the correct order of the present ConstraintGraph2.

6.5.2.3 getDt2()

```
Dt2* GEOM_FADE25D::ConstraintGraph2::getDt2 ( )
```

Returns

the Delaunay class it belongs to

6.5.2.4 getInsertionStrategy()

```
ConstraintInsertionStrategy GEOM_FADE25D::ConstraintGraph2::getInsertionStrategy ( ) const
```

Returns

CIS_CONFORMING_DELAUNAY, CIS_CONFORMING_DELAUNAY_SEGMENT_LEVEL or CIS_CONSTRAINED_DELAUNAY

6.5.2.5 getOriginalConstraintSegments()

```
void GEOM_FADE25D::ConstraintGraph2::getOriginalConstraintSegments (
            std::vector< ConstraintSegment2 *> & vConstraintSegments_ ) const
```

Get the original, ConstraintSegment2 objects. They are not subdivided but may be dead and have child segments (which may also be dead and have child segments...)

6.5.2.6 getPolygonVertices()

```
void GEOM_FADE25D::ConstraintGraph2::getPolygonVertices (
            std::vector< Point2 *> & vVertices_ )
```

Use this method to retrieve the vertices of the present ConstraintGraph2. If it forms ONE closed polygon, then the vertices are ordered and oriented in counterclockwise direction, e.g. (a,b,b,c,c,d,d,a). Otherwise they are returned in original order. Be aware that the order is only maintained if the ConstraintGraph2 has been created with Fade_2D::createConstraint(..,..,`bOrientedSegments=true`).

Note

The segments of the present ConstraintGraph2 may have been splitted. In this case the split points are also contained in the result. If, in the above example, the ConstraintSegment2(a,b) has been subdivided at vertex x then the result is (a,x,x,b,b,c,c,d,d,a).

See also

Do you already know Zone2::getBorderEdges() and edgesToPolygons() ?

6.5.2.7 isConstraint() [1/2]

```
bool GEOM_FADE25D::ConstraintGraph2::isConstraint (
            Point2 * p0,
            Point2 * p1 ) const
```

Checks if the edge (p0,p1) is a constraint of the present ConstraintGraph2 object.

6.5.2.8 isConstraint() [2/2]

```
bool GEOM_FADE25D::ConstraintGraph2::isConstraint (
            ConstraintSegment2 * pCSeg ) const
```

The present ConstraintGraph2 has been created using a set of edges and this method checks if the ConstraintSegment2 `pCSeg` is one of them. Original edges that have been splitted are not alive anymore and are no members. But their child segments are members.

6.5.2.9 isOriented()

```
bool GEOM_FADE25D::ConstraintGraph2::isOriented ( ) const
```

Returns

true if the constraint graph has been created with bOrientedSegments=true or if automatic reorientation was possible which is the case for simple polygons.

6.5.2.10 isPolygon()

```
bool GEOM_FADE25D::ConstraintGraph2::isPolygon ( ) const
```

Returns

true when the present ConstraintGraph forms a closed polygon.

Note

This method won't check if it is a simple polygon (one without self-intersections).

6.5.2.11 isReverse()

```
bool GEOM_FADE25D::ConstraintGraph2::isReverse (
            ConstraintSegment2 * pCSeg ) const
```

Get the orientation of a ConstraintSegment2

A ConstraintSegment2 `pCSeg` is unoriented because it may participate (with different orientations) in more than just one ConstraintGraph2 and thus the vertices returned by pCSeg->getSrc() and pCSeg->getTrg() do not carry any orientation information. However, the orientation of `pCSeg` is stored in the ConstraintGraph2 objects where `pCSeg` is a member and this method returns if the source and target vertex must be exchanged to match the present graph's direction.

6.5.2.12 makeDelaunay()

```
bool GEOM_FADE25D::ConstraintGraph2::makeDelaunay (
            double minLength )
```

Improve the triangle quality (make Delaunay)

Constraint segments can make a triangulation locally non-delaunay i.e., the empty-circumcircle property is not maintained for all triangles. `makeDelaunay()` subdivides the constraint segments so that they appear naturally as part of the Delaunay triangulation. Use this function to create visually more appealing triangles with better aspect ratios.

Note

This optimization considers the projection of the triangles to the xy plane.

Parameters

in	*minLength*	specifies a lower bound. Constraint segments smaller than `minLength` are not subdivided. This parameter avoids excessive subdivision in narrow settings.

Returns

TRUE when all required somedevisions have been carried out or FALSE when `minLength` has avoided further subdivision.

6.5.2.13 show() [1/2]

```
void GEOM_FADE25D::ConstraintGraph2::show (
            const std::string & name )
```

6.5.2.14 show() [2/2]

```
void GEOM_FADE25D::ConstraintGraph2::show (
            Visualizer2 * pVis,
            const Color & color )
```

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

- ConstraintGraph2.h

6.6 GEOM_FADE25D::ConstraintSegment2 Class Reference

A ConstraintSegment2 represents a Constraint Edge.

`#include <ConstraintSegment2.h>`

Public Member Functions

- Point2 ∗ getSrc () const

  Get the first endpoint.
- Point2 ∗ getTrg () const

  Get the second endpoint.
- bool isAlive () const

  Check if the present ConstraintSegment2 is alive.
- ConstraintInsertionStrategy getCIS () const

  Get the Constraint Insertion Strategy (CIS)
- bool operator< (const ConstraintSegment2 &pOther) const

  operator<(..) Compares the vertex pointers of the endpoints, not the length
- Point2 ∗ insertAndSplit (const Point2 &splitPoint)

  Split a constraint segment.
- bool split_combinatorialOnly (Point2 ∗pSplit)

  Split a constraint segment.
- void getChildrenRec (std::vector< ConstraintSegment2 ∗> &vChildConstraintSegments)

  Get all children Recursively retrieve all children of the current ConstraintSegment2.
- void getChildrenAndSplitPoint (ConstraintSegment2 ∗&pCSeg0, ConstraintSegment2 ∗&pCSeg1, Point2 ∗&pSplitPoint)

  Get the children and the split point Retrieve the two direct children of the current ConstraintSegment2 as well as the split point.

Public Attributes

- int **label**

Protected Attributes

- Point2 ∗ **p0**
- Point2 ∗ **p1**
- ConstraintInsertionStrategy **cis**
- bool **bAlive**
- std::vector< ConstraintSegment2 ∗ > **vChildren**

Static Protected Attributes

- static int **runningLabel**

Friends

- class **ConstraintMgr**
- class **ConstraintGraph2**
- std::ostream & **operator**<< (std::ostream &stream, const ConstraintSegment2 &cSeg)

6.6.1 Detailed Description

A ConstraintSegment2 can belong to more than one ConstraintGraph2 object, thus it is unoriented. But the ConstraintGraph knows the orientation of its ConstraintSegment2's.

6.6.2 Member Function Documentation

6.6.2.1 getCIS()

```
ConstraintInsertionStrategy GEOM_FADE25D::ConstraintSegment2::getCIS ( ) const
```

Returns

the constraint insertion strategy (CIS) of the present object

6.6.2.2 getSrc()

```
Point2* GEOM_FADE25D::ConstraintSegment2::getSrc ( ) const
```

Returns

the first vertex

6.6.2.3 getTrg()

```
Point2* GEOM_FADE25D::ConstraintSegment2::getTrg ( ) const
```

Returns

the second vertex

6.6.2.4 insertAndSplit()

```
Point2* GEOM_FADE25D::ConstraintSegment2::insertAndSplit (
            const Point2 & splitPoint )
```

Splits the ConstraintSegment2 (which must be alive) at `splitPoint`.

It may be impossible to represent a point on a certain line segment using floatingpoint arithmetic. Therefore it is highly recommended to split a ConstraintSegment2 object not just be inserting points into the triangulation but using the present method. It does not require that `splitPoint` is exactly on the segment.

Note

A splitted ConstraintSegment2 is dead and it has two child segments (which may also be dead and have children). The class is organized as a binary tree.

6.6.2.5 isAlive()

```
bool GEOM_FADE25D::ConstraintSegment2::isAlive ( ) const
```

Returns

TRUE when the object is alive, FALSE otherwise

6.6.2.6 split_combinatorialOnly()

```
bool GEOM_FADE25D::ConstraintSegment2::split_combinatorialOnly (
            Point2 * pSplit )
```

internal use only (unless you do something very unusual)

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

- ConstraintSegment2.h

6.7 GEOM_FADE25D::CutAndFill Class Reference

Cut-And-Fill.

`#include <CutAndFill.h>`

Public Member Functions

- CutAndFill (Zone2 *pZoneBefore, Zone2 *pZoneAfter, double ignoreThreshold=1e-3)

  Constructor.
- bool getDiffZone (Zone2 *&pDiffZone, std::map< Point2 *, std::pair< double, double > > &mVtx2BeforeAfter)

  Get the difference zone.
- void subscribe (MsgType msgType, MsgBase *pMsg)

  Register a progress bar object.
- void unsubscribe (MsgType msgType, MsgBase *pMsg)

  Unregister a progress bar object.
- size_t getNumberOfComponents () const

  Get the number of components.
- CAF_Component * getComponent (size_t ith) const

  Get component `ith`.
- bool go ()

  Start the computation.
- void show (Visualizer2 *pVis) const

  Draw a postscript visualization.

Protected Attributes

- CutAndFillImpl * **pCAFI**

6.7.1 Detailed Description

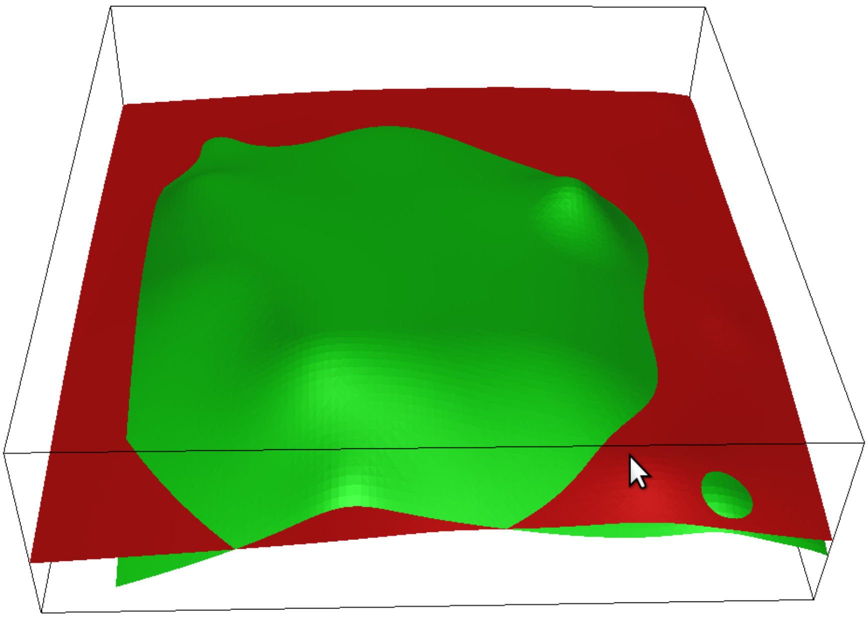


Figure 8 Overlapping input surfaces for Cut-And-Fill: RED=before, GREEN=after. The surfaces do not need to match exactly, the overlapping area is used

Given two overlapping surfaces with different elevations, CutAndFill partitions the surfaces into connected components and computes the volume that must be removed or added to turn one surface into the other.

See also

http://www.geom.at/cut-and-fill/

6.7.2 Constructor & Destructor Documentation

6.7.2.1 CutAndFill()

```
GEOM_FADE25D::CutAndFill::CutAndFill (
            Zone2 * pZoneBefore,
            Zone2 * pZoneAfter,
            double ignoreThreshold = 1e-3 )
```

Parameters

pZoneBefore	represents the surface before the earthworks
pZoneAfter	is the surface afterwards
ignoreThreshold	(default: 1e-3) can be used to ignore insignificant height differences

6.7.3 Member Function Documentation

6.7.3.1 getComponent()

```
CAF_Component* GEOM_FADE25D::CutAndFill::getComponent (
            size_t ith ) const
```

Returns

the `ith` CAF_Component.

6.7.3.2 getDiffZone()

```
bool GEOM_FADE25D::CutAndFill::getDiffZone (
            Zone2 *& pDiffZone,
            std::map< Point2 *, std::pair< double, double > > & mVtx2BeforeAfter )
```

This method gives access to the internal data structures, namely to a Zone2 object whose vertices have z-values that correspond to the height differences between the two input meshes (SurfaceBefore minus SurfaceAfter). And a map is returned that contains for each vertex the height in the first and in the second input mesh.

Returns

true in case of success, false otherwise.

Note

This method may set pDiffZone=NULL and return false when the two input surfaces do not share a common area. In this case the previous call to CutAndFill::go() has already returned false.

6.7.3.3 getNumberOfComponents()

```
size_t GEOM_FADE25D::CutAndFill::getNumberOfComponents ( ) const
```

Returns

the number of components.

A CAF_Component object represents a connected part of the surface such that

- the first surface is below the second one (Type CT_FILL)
- the first surface is above the second one (Type CT_CUT)
- the first surface corresponds to the second one (Type CT_NULL)

6.7.3.4 go()

```
bool GEOM_FADE25D::CutAndFill::go ( )
```

Returns

true in case of success, false otherwise.

Note

When an input zone is empty or when the two input zones do not overlap then there is no common area on which the algorithm could operate. In this case the present method returns false.

6.7.3.5 show()

```
void GEOM_FADE25D::CutAndFill::show (
            Visualizer2 * pVis ) const
```

For a quick overview a postscript visualization can be created.

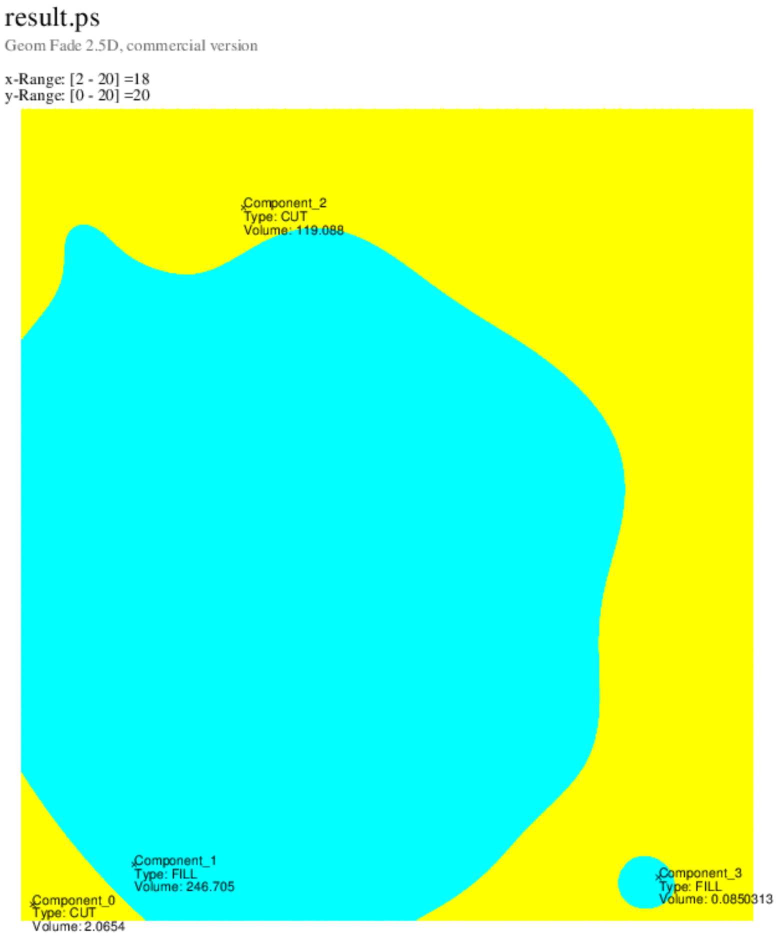


Figure 9 Cut&Fill-Result: YELLOW area CUT, CYAN area: FILL

6.7.3.6 subscribe()

```
void GEOM_FADE25D::CutAndFill::subscribe (
            MsgType msgType,
            MsgBase * pMsg )
```

A user defined message receiver object (for example your own progress-bar class) can be registered to get progress updates. This step is optional.

Parameters

msgType	is the message type. For progress information the type is always MSG_PROGRESS
pMsg	is a user defined progress bar which derives from Fade's MsgBase.

6.7.3.7 unsubscribe()

```
void GEOM_FADE25D::CutAndFill::unsubscribe (
            MsgType msgType,
            MsgBase * pMsg )
```

Parameters

msgType	is the message type. For progress information the type is always MSG_PROGRESS
pMsg	is a user defined class which derives from Fade's MsgBase

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

- CutAndFill.h

6.8 GEOM_FADE25D::Edge2 Class Reference

Directed Edge.

`#include <Edge2.h>`

Public Member Functions

- **Edge2** (const Edge2 &e_)
- Edge2 (Triangle2 ∗pT, int oppIdx_)

  Constructor.
- Edge2 & **operator=** (const Edge2 &other)
- bool operator< (const Edge2 &e) const

  operator<()
- bool operator== (const Edge2 &e) const

  operator==()
- bool operator!= (const Edge2 &e) const

  operator!=()
- Point2 ∗ getSrc () const

  Get the source point.
- Point2 ∗ getTrg () const

  Get the target point.
- void getPoints (Point2 ∗&p1, Point2 ∗&p2) const

  Get the endpoints.
- double getLength2D () const
- double getLength25D () const
- Triangle2 ∗ getTriangle () const
- void getTriangles (Triangle2 ∗&pT0, Triangle2 ∗&pT1, int &idx0, int &idx1) const

Protected Attributes

- Triangle2 ∗ **pT**
- int **oppIdx**

Friends

- std::ostream & **operator**<< (std::ostream &stream, const Edge2 &e)

6.8.1 Constructor & Destructor Documentation

6.8.1.1 Edge2()

```
GEOM_FADE25D::Edge2::Edge2 (
            Triangle2 * pT,
            int oppIdx_ )
```

Parameters

pT	is the triangle from which the edge is constructed
*opp↵ Idx_*	is intra-triangle-index of the opposite vertex (of the edge) in pT

The orientation of the constructed Edge2 is counterclockwise (CCW) with respect to `pT`. Example: Edge2(pT,0) creates an edge from pT->getCorner(1) to pT->getCorner(2).

6.8.2 Member Function Documentation

6.8.2.1 getLength25D()

```
double GEOM_FADE25D::Edge2::getLength25D ( ) const
```

Get the 2.5D length

Returns

the 2.5D length of the edge

6.8.2.2 getLength2D()

```
double GEOM_FADE25D::Edge2::getLength2D ( ) const
```

Get the 2D length

Returns

the 2D length of the edge, the z-coordinate is ignored

6.8.2.3 getPoints()

```
void GEOM_FADE25D::Edge2::getPoints (
            Point2 *& p1,
            Point2 *& p2 ) const
```

returns the source point of the edge as `p1` and the target point as `p2`

6.8.2.4 getSrc()

```
Point2* GEOM_FADE25D::Edge2::getSrc ( ) const
```

Returns

the source point of the edge, i.e. pT->getCorner((oppIdx+1)%3)

6.8.2.5 getTrg()

```
Point2* GEOM_FADE25D::Edge2::getTrg ( ) const
```

Returns

the target point of the edge, i.e. pT->getCorner((oppIdx+2)%3)

6.8.2.6 getTriangle()

```
Triangle2* GEOM_FADE25D::Edge2::getTriangle ( ) const
```

Get the triangle

Returns

the triangle whose directed edge the present edge is

6.8.2.7 getTriangles()

```
void GEOM_FADE25D::Edge2::getTriangles (
            Triangle2 *& pT0,
            Triangle2 *& pT1,
            int & idx0,
            int & idx1 ) const
```

Get the triangles

Returns

the two adjacent triangles of the present edge along with their intra-triangle-indices

Parameters

pT0	is used to return the triangle whose directed edge the present edge is
idx0	is the opposite intra-triangle-index in pT0 of the present edge
pT1	is the other adjacent triangle at the present edge (or NULL)
idx1	is the intra-triangle index of the present edge in pT1 (or -1)

6.8.2.8 operator"!=()

```
bool GEOM_FADE25D::Edge2::operator!= (
             const Edge2 & e ) const  [inline]
```

operator!=() returns true if the compared edges are different. Be aware that edges are directed and therefore two adjacent triangles do not share the same edge.

6.8.2.9 operator<()

```
bool GEOM_FADE25D::Edge2::operator< (
             const Edge2 & e ) const  [inline]
```

operator<() does NOT compare edge lengths but the associated triangle pointers and intra-triangle indices. This is useful when edges are used in STL containers.

6.8.2.10 operator==()

```
bool GEOM_FADE25D::Edge2::operator== (
             const Edge2 & e ) const  [inline]
```

operator==() compares oriented edges, i.e., it returns only true when the two edges have been made from the same triangle and the same intra-triangle-index.

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

- Edge2.h

6.9 GEOM_FADE25D::EfficientModel Class Reference

EfficientModel.

```
#include <EfficientModel.h>
```

Public Member Functions

- **EfficientModel** (const std::vector< Point2 > &vPoints)
- void zSmoothing (int numIterations, double maxDifferencePerIteration, SmoothingStrategy sms)

  Smoothing.
- void extract (double maxError, std::vector< Point2 > &vEfficientPointsOut)

  Extract a subset of points.

Protected Member Functions

- void **solveCand** (Candidate *pCand, double maxErr)
- void **go** ()
- void **part1_extractFC** ()
- void **part2_setWeights** ()
- void **sortVtx** (std::vector< Point2 *> &vVtx)
- int **insertKeepError** (double factor, double err, std::vector< Point2 *> &vIn, std::vector< Point2 *> &v↵ NeedlessBigError, std::vector< Point2 *> &vNeedlessSmallError)
- void **insertMinHull** ()
- void **show** (const std::string &name)

Protected Attributes

- EMData * **pEMData**

6.9.1 Detailed Description

Point clouds from terrains are often oversampled. The present class aims to smooth and to reduce these point clouds in a controlled way such that the resulting surface keeps a certain user specified quality.

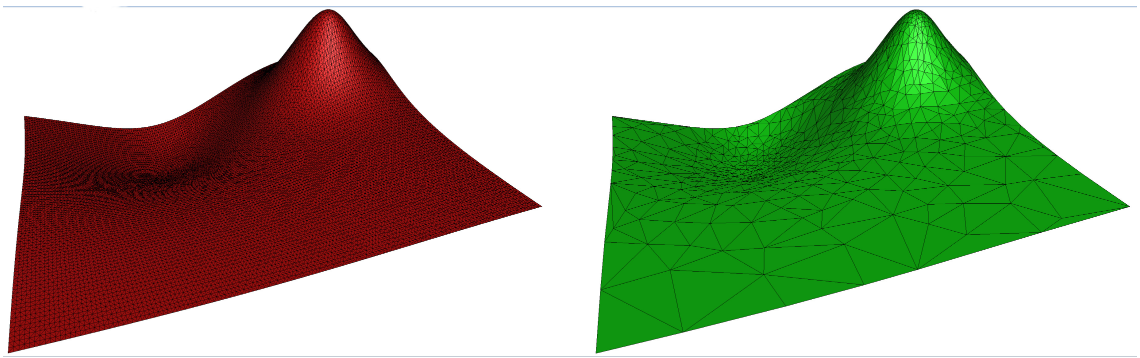


Figure 10 Point Cloud Reduction: Left original, right reduced

6.9.2 Member Function Documentation

6.9.2.1 extract()

```
void GEOM_FADE25D::EfficientModel::extract (
            double maxError,
            std::vector< Point2 > & vEfficientPointsOut )
```

This method extracts a subset of the original point cloud that represents the model more efficiently. Thereby the original and the simplified model cover the same area.

Parameters

	maxError	is the maximum height difference between the original points and the simplified model.
`out`	*vEfficientPointsOut*	is used to return a subset of the original points that represents the model more efficiently.

Note

When `maxError` is tiny i.e., below the noise level of the point cloud, then processing can take quite some time. Consider using the zSmoothing() method before.

6.9.2.2 zSmoothing()

```
void GEOM_FADE25D::EfficientModel::zSmoothing (
            int numIterations,
            double maxDifferencePerIteration,
            SmoothingStrategy sms )
```

This method should be used before extract(). It adapts the z-values according to the chosen SmoothingStrategy `sms`.

Parameters

numIterations	Number of iterations
maxDifferencePerIteration	is the maximum change of any z-value
sms	is one of SMS_MINIMUM, SMS_MAXIMUM, SMS_MEDIAN, SMS_AVERAGE

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

- EfficientModel.h

6.10 GEOM_FADE25D::Fade_2D Class Reference

Delaunay triangulation - the main class.

`#include <Fade_2D.h>`

Public Member Functions

- Fade_2D (unsigned numExpectedVertices=3)

  Constructor of the main triangulation class.
- bool checkValidity (bool bCheckEmptyCircleProperty, const std::string &msg) const

  Checks if a triangulation is valid.
- int setNumCPU (int numCPU)

  Set the number CPU cores for multithreading.
- void statistics (const std::string &s) const

Statistics.

- void show (const std::string &postscriptFilename, bool bWithConstraints=true) const

  Draws the triangulation as postscript file.

- void show (Visualizer2 ∗pVis, bool bWithConstraints=true) const

  Draws the triangulation as postscript file using an existing Visualizer2 object.

- void showGeomview (const std::string &filename)

  Draws the triangulation in 3D.

- void showGeomview (Visualizer3 ∗pVis)

  Draws the triangulation in 3D.

- void remove (Point2 ∗pVertex)

  Remove a single vertex.

- void getConvexHull (bool bAllVertices, std::vector< Point2 ∗> &vConvexHullPointsOut)

  Compute the convex hull.

- Point2 ∗ insert (const Point2 &p)

  Insert a single point.

- void insert (const std::vector< Point2 > &vInputPoints)

  Insert a vector of points.

- void insert (const std::vector< Point2 > &vInputPoints, std::vector< Point2 ∗> &vHandles)

  Insert points from a std::vector and store pointers in vHandles.

- void insert (int numPoints, double ∗aCoordinates, Point2 ∗∗aHandles)

  Insert points from an array.

- double measureTriangulationTime (std::vector< Point2 > &vPoints)

  Measure the Delaunay triangulation time.

- Triangle2 ∗ locate (const Point2 &p)

  Locate a triangle which contains p.

- bool getHeight (double x, double y, double &heightOut, Triangle2 ∗pApproxT=NULL, double tolerance=0) const

  Compute the height of a certain point.

- void refine (Zone2 ∗pZone, double minAngleDegree, double minEdgeLength, double maxEdgeLength, bool bAllowConstraintSplitting)

  Delaunay refinement.

- void refineAdvanced (MeshGenParams ∗pParameters)

  Delaunay refinement and grid meshing.

- size_t numberOfPoints () const

  Number of points.

- size_t numberOfTriangles () const

  Number of triangles.

- void getTrianglePointers (std::vector< Triangle2 ∗> &vAllTriangles) const

  Get pointers to all triangles.

- void getVertexPointers (std::vector< Point2 ∗> &vAllPoints) const

  Get pointers to all vertices.

- Triangle2 ∗ getAdjacentTriangle (Point2 ∗p0, Point2 ∗p1) const

  Get adjacent triangle.

- bool is2D () const

  Check if the triangulation contains triangles (which is the case if at least 3 non-collinear points exist in the triangulation.

- ConstraintGraph2 ∗ createConstraint (std::vector< Segment2 > &vSegments, ConstraintInsertionStrategy cis, bool bOrientedSegments=false)

  Add constraint edges (edges, polyline, polygon)

- Zone2 ∗ createZone (ConstraintGraph2 ∗pConstraintGraph, ZoneLocation zoneLoc, bool bVerbose=true)

  Create a zone.

- Zone2 ∗ createZone (const std::vector< ConstraintGraph2 ∗> &vConstraintGraphs, ZoneLocation zoneLoc, const Point2 &startPoint, bool bVerbose=true)

    Create a zone limited by multiple ConstraintGraph2 objects by growing from a start point.
- Zone2 ∗ createZone (ConstraintGraph2 ∗pConstraintGraph, ZoneLocation zoneLoc, const Point2 &startPoint, bool bVerbose=true)

    Create a zone limited by a ConstraintGraph by growing from a start point.
- Zone2 ∗ createZone (std::vector< Triangle2 ∗> &vTriangles, bool bVerbose=true)

    Create a zone defined by a vector of triangles.
- void deleteZone (Zone2 ∗pZone)

    Delete a Zone2 object.
- void applyConstraintsAndZones ()

    Apply conforming constraints and zones (deprecated!)
- Bbox2 computeBoundingBox () const

    Compute the axis-aligned bounding box of the points.
- bool isConstraint (Triangle2 ∗pT, int ith) const

    Check if an edge is a constraint edge.
- void getAliveConstraintSegments (std::vector< ConstraintSegment2 ∗> &vAliveConstraintSegments) const

    Get active (alive) constraint segments.
- void getAliveAndDeadConstraintSegments (std::vector< ConstraintSegment2 ∗> &vAllConstraintSegments) const

    Get all (alive and dead) constraint segments.
- ConstraintSegment2 ∗ getConstraintSegment (Point2 ∗p0, Point2 ∗p1) const

    Retrieve a ConstraintSegment2.
- void getIncidentTriangles (Point2 ∗pVtx, std::vector< Triangle2 ∗> &vIncidentT) const

    Get incident triangles.
- void getIncidentVertices (Point2 ∗pVtx, std::vector< Point2 ∗> &vIncidentVertices) const

    Get incident vertices.
- void writeObj (const std::string &filename) const

    Write the current triangulation to an ∗.obj file.
- void writeObj (const std::string &filename, Zone2 ∗pZone) const

    Write a zone to an ∗.obj file.
- void writeWebScene (const char ∗path) const

    Write the current triangulation to an ∗.obj file.
- void writeWebScene (const char ∗path, Zone2 ∗pZone) const

    Write a zone to an ∗.obj file.
- void subscribe (MsgType msgType, MsgBase ∗pMsg)

    Register a message receiver.
- void unsubscribe (MsgType msgType, MsgBase ∗pMsg)

    Unregister a message receiver.
- bool isConstraint (Point2 ∗p0, Point2 ∗p1) const

    Check if an edge is a constraint edge.
- bool isConstraint (Point2 ∗pVtx) const

    Check if a vertex is a constraint vertex.
- void printLicense () const
- Zone2 ∗ importTriangles (std::vector< Point2 > &vPoints, bool bReorientIfNeeded, bool bCreateExtended←BoundingBox)

    Import triangles.
- Orientation2 getOrientation (const Point2 &p0, const Point2 &p1, const Point2 &p2)

    Compute the orientation of 3 points.
- void cutTriangles (const Point2 &knifeStart, const Point2 &knifeEnd, bool bTurnEdgesIntoConstraints)

    Cut through a triangulation.

- void cutTriangles (std::vector< Segment2 > &vSegments, bool bTurnEdgesIntoConstraints)

    Cut through a triangulation.
- Zone2 ∗ createZone_cookieCutter (std::vector< Segment2 > &vSegments, bool bProtectEdges)

    Cookie Cutter The Cookie Cutter cuts out a part of a triangulation and returns it as a Zone2 object.
- bool drape (std::vector< Segment2 > &vSegmentsIn, std::vector< Segment2 > &vSegmentsOut, double zTolerance) const

    Drape segments along a surface.
- void enableMultithreading ()

    Enable multithreading (deprecated)

6.10.1 Detailed Description

Fade_2D represents a Delaunay triangulation in 2D or 2.5D (depends on the used namespace)

6.10.2 Constructor & Destructor Documentation

6.10.2.1 Fade_2D()

```
GEOM_FADE25D::Fade_2D::Fade_2D (
            unsigned numExpectedVertices = 3 )  [inline], [explicit]
```

Parameters

numExpectedVertices	specifies the number of points that will be inserted. This is a default parameter that does not need to be specified.

6.10.3 Member Function Documentation

6.10.3.1 applyConstraintsAndZones()

```
void GEOM_FADE25D::Fade_2D::applyConstraintsAndZones ( )
```

This method establishes conforming constraint segments and zones which depend on them. For technical reasons conforming constraint segments are not immediately established but inserted at the end of the triangulation process. This step must be triggered manually i.e., it is up to the user to call applyConstraintsAndZones() before the resulting triangulation is used. If afterwards the triangulation is changed in any way, applyConstraintsAndZones() must be called again.

Note

> The present function applyConstraintsAndZones() as well as the two constraint insertion strategies CIS_CO↩ NFORMING_DELAUNAY and CIS_CONFORMING_DELAUNAY_SEGMENT_LEVEL are deprecated. These are only kept for backwards compatibilty. The replacement is CIS_CONSTRAINED_DELAUNAY along with the methods Fade_2D::drape() and/or ConstraintGraph2::makeDelaunay(). See the example code in examples↩ _25D/terrain.cpp

6.10.3.2 checkValidity()

```
bool GEOM_FADE25D::Fade_2D::checkValidity (
            bool bCheckEmptyCircleProperty,
            const std::string & msg ) const
```

Checks the validity of the data structure.

Parameters

bCheckEmptyCircleProperty	specifies if (slow!) multiprecision arithmetic shall be used to recheck the empty circle property
msg	is a debug string that will be shown in terminal output so that you know which checkValidity call currently runs.

This method is thought for development purposes. Don't call it method unless you assume that something is wrong with the code.

6.10.3.3 computeBoundingBox()

```
Bbox2 GEOM_FADE25D::Fade_2D::computeBoundingBox ( ) const
```

If no points have been inserted yet, then the returned Bbox2 object is invalid and its member function *Bbox2::isValid()* returns false.

6.10.3.4 createConstraint()

```
ConstraintGraph2* GEOM_FADE25D::Fade_2D::createConstraint (
            std::vector< Segment2 > & vSegments,
            ConstraintInsertionStrategy cis,
            bool bOrientedSegments = false )
```

Parameters

vSegments	are segments which shall appear as edges of the triangulation. The segments may be automatically reordered and reoriented, see `bOrientedSegments` below.
cis	is the Constraint-Insertion-Strategy. Use always CIS_CONSTRAINED_DELAUNAY. This mode inserts the constraint segments at their original level (no projection onto the surface) and without subdivision unless existing vertices or existing constraint segments are crossed. When subdivision (e.g., to achieve better triangle shapes) is desired then use ConstraintGraph2::makeDelaunay() after insertion. When the segments shall be adapted to the elevation of the existing surface then use Fade_2D::drape(). See the example code in `examples_25D/terrain.cpp`
bOrientedSegments	specifies whether the segments in vSegments are oriented (*oriented, not ordered!*). To maintain backwards compatibility `bOrientedSegments` is a default parameter and it defaults to false. Fade will maintain the orientation of the segments only when `bOrientedSegments=true`. This regards functions like ConstraintGraph2::getPolygonVertices() when the order of the returned vertices is important. Another consequence is when later a Zone2 object shall be constructed from this ConstraintGraph2. This is only possible if either this value is true (then the algorithm will assume that all segments exist in counterclockwise orientation) or when the value is false and the segments can be automatically reoriented and reordered such that they form one closed polygon.

Returns

a pointer to the new ConstraintGraph2 object

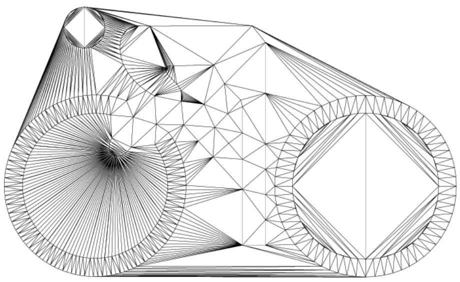


Figure 11 Delaunay triangulation without constraints

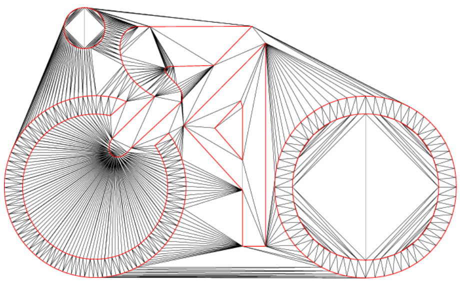


Figure 12 Constraint Delaunay triangulation

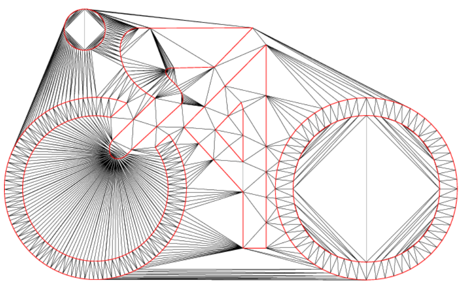


Figure 13 Conforming Delaunay triangulation through the ConstraintGraph::makeDelaunay() method

6.10.3.5 createZone() [1/4]

```
Zone2* GEOM_FADE25D::Fade_2D::createZone (
            ConstraintGraph2 * pConstraintGraph,
            ZoneLocation zoneLoc,
            bool bVerbose = true )
```

A Zone2 object is an area of a triangulation, possibly bounded by a ConstraintGraph.

Parameters

zoneLoc	is ZL_INSIDE, ZL_OUTSIDE or ZL_GLOBAL.
pConstraintGraph	points to a formerly created ConstraintGraph2 object (which must be oriented and contain a *simple* polygon) or is NULL in case of zoneLoc==ZL_GLOBAL.
bVerbose	is by default true and causes a warning if NULL is returned.

Returns

a pointer to the new Zone2 object or NULL if no triangles exist or pConstraintGraph->isOriented() returns false.

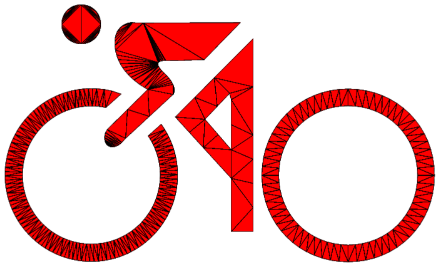


Figure 14 Zones in a triangulation

6.10.3.6 createZone() [2/4]

```
Zone2* GEOM_FADE25D::Fade_2D::createZone (
            const std::vector< ConstraintGraph2 *> & vConstraintGraphs,
            ZoneLocation zoneLoc,
            const Point2 & startPoint,
            bool bVerbose = true )
```

A Zone2 object is an area of the traingulation, see createZone

Parameters

vConstraintGraphs	is a vector of ConstraintGraph objects
zoneLoc	must be ZL_GROW
startPoint	is the point from which the area is grown until the borders specified in vConstraintGraphs are reached
bVerbose	is by default true and causes a warning if NULL is returned.

Returns

a pointer to the new Zone2 object (or NULL if zoneLoc!=ZL_GROW or no triangles exist)

6.10.3.7 createZone() [3/4]

```
Zone2* GEOM_FADE25D::Fade_2D::createZone (
            ConstraintGraph2 * pConstraintGraph,
            ZoneLocation zoneLoc,
            const Point2 & startPoint,
            bool bVerbose = true )
```

A Zone2 object is an area of the traingulation, see createZone

Parameters

pConstraintGraph	is a constraint whose edges specify the area's border
zoneLoc	must be ZL_GROW
startPoint	is the point from which the area is grown until the borders specified in pConstraint are reached
bVerbose	is by default true and causes a warning if NULL is returned.

Returns

a pointer to the new Zone2 object (or NULL if zoneLoc!=ZL_GROW or no triangles exist)

6.10.3.8 createZone() [4/4]

```
Zone2* GEOM_FADE25D::Fade_2D::createZone (
            std::vector< Triangle2 *> & vTriangles,
            bool bVerbose = true )
```

A Zone2 object is an area of the traingulation, see createZone

Parameters

vTriangles	
bVerbose	is by default true and causes a warning if NULL is returned.

Returns

a pointer to the new Zone2 object (or NULL if vTriangles is empty)

6.10.3.9 createZone_cookieCutter()

```
Zone2* GEOM_FADE25D::Fade_2D::createZone_cookieCutter (
            std::vector< Segment2 > & vSegments,
            bool bProtectEdges )
```

Parameters

in	*vSegments*	specifies a simple polygon.
in	*bProtectEdges*	specifies if existing triangles shall be protected with constraint segments.

Returns

a Zone2 object consisting of all triangles inside the polygon or NULL when the operation has failed due to wrong preconditions.

Properties: The input polygon (`vSegments`) does not need to have certain height values, the z-coordinates are computed automatically. The input polygon is automatically trimmed when it is outside the convex hull of the triangulation. Insertion of intersection points could flip existing edges in the triangulation, this can be avoided using bProtectEdges=true. The operation may create constraint segments.

6.10.3.10 cutTriangles() [1/2]

```
void GEOM_FADE25D::Fade_2D::cutTriangles (
            const Point2 & knifeStart,
            const Point2 & knifeEnd,
            bool bTurnEdgesIntoConstraints )
```

Parameters

knifeStart	is one point of the knife segment
knifeEnd	is the second point of the knife segment
bTurnEdgesIntoConstraints	turns all 3 edges of each intersected triangle into constraint segments.

This method inserts a constraint edge *knife*(*knifeStart*,*knifeEnd*). If existing edges *E* are intersected by *knife*, then *knife* is subdivided at the intersection points *P*.

In any case *knife* will exist (in a possibly subdivided form) in the result. But a consequence of the insertion of the points *P* is that the edges *E* and even edges which are not intersected by *knife* may be flipped. Use bTurnEdges↵ IntoConstraints=true to avoid that.

Note

The intersection point of two line segments may not be exactly representable in double precision floating point arithmetic and thus tiny rounding errors may occur. As a consequence two very close intersection points may be rounded to the same coordinates.

When more than one knife segment is inserted then the method void cutTriangles(std::vector<Segment2>& vSegments,bool bTurnEdgesIntoConstraints) should be used. The reason is that each individual cut operation changes the triangulation and thus iterative calls to the present version of the method can lead to a different result.

6.10.3.11 cutTriangles() [2/2]

```
void GEOM_FADE25D::Fade_2D::cutTriangles (
            std::vector< Segment2 > & vSegments,
            bool bTurnEdgesIntoConstraints )
```

Parameters

vSegments	are the knife segments
bTurnEdgesIntoConstraints	specifies if intersected edges shall automatically be turned into constraints

Same method as void cutTriangles(const Point2& knifeStart,const Point2& knifeEnd,bool bTurnEdgesInto↵ Constraints) but it takes a vector of segments instead of a single segment. This is the recommended method to cut through a triangulation when more than one knife segment exists.

6.10.3.12 deleteZone()

```
void GEOM_FADE25D::Fade_2D::deleteZone (
            Zone2 * pZone )
```

Zone2 objects are automatically destroyed with their Fade_2D objects. In addition this method provides the possibility to eliminate Zone2 objects earlier.

Note

Zones are designed transparently: When two zones Z1 and Z2 are combined to a new one Z3 (for example through a boolean operation) then Z1,Z2,Z3 form a tree such that changes in the leaf nodes Z1 and Z2 can propagate up to the root node Z3. For this reason Z1 and Z2 must be alive as long as Z3 is used.

6.10.3.13 drape()

```
bool GEOM_FADE25D::Fade_2D::drape (
            std::vector< Segment2 > & vSegmentsIn,
            std::vector< Segment2 > & vSegmentsOut,
            double zTolerance ) const
```

Projects the segments from `vSegmentsIn` onto the triangulation. Thereby the segments are subdivided where they intersect edges of the triangulation. Segment parts outside the triangulation are cut off and ignored. Degenerate input segments are also ignored.

The heights (z-values) of the result segments are adapted to the surface.

Parameters

in	*zTolerance*	is used to avoid excessive subdivision of segments. Use some positive value to define the acceptable geometric error or use `zTolerance=-1.0` to split the segments at all intersections with triangulation-edges.
in	*vSegmentsIn*	Input segments
out	*vSegmentsOut*	Output segments

Returns

TRUE when all input segments are inside the convex hull of the triangulation. Otherwise FALSE is returned and the result is still valid but it contains only the segment parts inside the convex hull.

Note

The tiny rounding errors that occur when segment intersections are computed are largely theoretical. But be aware that subdivided segments are not always perfectly collinear. This can't be avoided because the exact split point is sometimes not even representable using floating point coordinates.

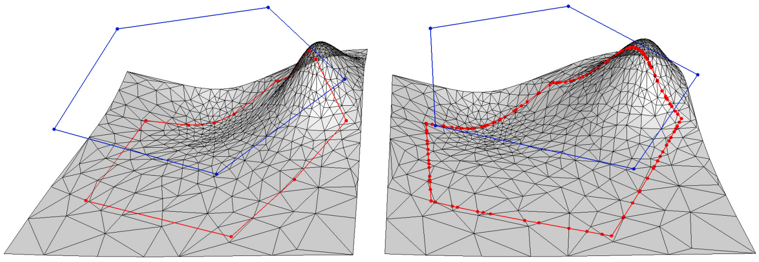
- 

Figure 15 Drape: Input segments (blue) are draped (red) onto a TIN. Left with tolerance 1.0, right without tolerance

Note

Draping segments onto a TIN does not insert them. Use Fade_2D::createConstraint() for that purpose.

6.10.3.14 enableMultithreading()

```
void GEOM_FADE25D::Fade_2D::enableMultithreading ( )
```

Deprecated: Use setNumCPU() instead. This method is kept for compatibility with existing applications. Internally it calls setNumCPU(0) to automatically determine and use the number of available CPU cores.

6.10.3.15 getAdjacentTriangle()

```
Triangle2* GEOM_FADE25D::Fade_2D::getAdjacentTriangle (
            Point2 * p0,
            Point2 * p1 ) const
```

Returns

the triangle that has the edge (p0,p1) or NULL if no such edge is present

Note

Recall the counter-clockwise enumeration of vertices in a triangle. If (p0,p1) is used, the unique triangle with the CCW oriented edge (p0,p1) is returned, using (p1,p0) one gets the other adjacent triangle.

6.10.3.16 getConstraintSegment()

```
ConstraintSegment2* GEOM_FADE25D::Fade_2D::getConstraintSegment (
            Point2 * p0,
            Point2 * p1 ) const
```

Returns

a pointer to the ConstraintSegment2 between p0 and p1 or NULL if the segment is not a constraint edge (or dead because it has been splitted)

6.10.3.17 getConvexHull()

```
void GEOM_FADE25D::Fade_2D::getConvexHull (
            bool bAllVertices,
            std::vector< Point2 *> & vConvexHullPointsOut )
```

Parameters

	bAllVertices	determines if all convex hull points are returned or if collinear ones shall be removed.
out	*vConvexHullPointsOut*	is used to return the convex hull vertices in counterclockwise order. The start vertex is the leftmost vertex. If more than one leftmost vertex exists, the bottommost of them is the start vertex.

6.10.3.18 getHeight()

```
bool GEOM_FADE25D::Fade_2D::getHeight (
            double x,
            double y,
            double & heightOut,
            Triangle2 * pApproxT = NULL,
            double tolerance = 0 ) const
```

Computes the height (z) at the coordinates `x` and `y`, assigns it to heightOut and returns true if successful.

Parameters

in	*x,y*	are the input coordinates
out	*heightOut*	is the computed height
in	*pApproxT*	can be set to a nearby triangle. If unknown, use NULL.
in	*tolerance*	is by default 0, see below

Note

pApproxT is an optional parameter to speed up the search in case that you know a nearby triangle. But point location if very fast anyway and if you are not sure, using NULL is probably faster.

Due to rounding errors your query point may lie slightly outside the convex hull of the triangulation and in such a case the present method would correctly return false. But you can use the optional `tolerance` parameter (default: 0): If your query point is not farther outside the convex hull than `tolerance` then the height of the closest point of the convex hull is returned.

6.10.3.19 getIncidentTriangles()

```
void GEOM_FADE25D::Fade_2D::getIncidentTriangles (
            Point2 * pVtx,
            std::vector< Triangle2 *> & vIncidentT ) const
```

Stores pointers to all triangles around pVtx into vIncidentT

6.10.3.20 getIncidentVertices()

```
void GEOM_FADE25D::Fade_2D::getIncidentVertices (
            Point2 * pVtx,
            std::vector< Point2 *> & vIncidentVertices ) const
```

Stores pointers to all vertices around pVtx into vIncidentVertices

6.10.3.21 getOrientation()

```
Orientation2 GEOM_FADE25D::Fade_2D::getOrientation (
            const Point2 & p0,
            const Point2 & p1,
            const Point2 & p2 )
```

Returns

ORIENTATION2_COLLINEAR, ORIENTATION2_CW (clockwise) or ORENTATION2_CCW (counterclockwise)

6.10.3.22 getTrianglePointers()

```
void GEOM_FADE25D::Fade_2D::getTrianglePointers (
            std::vector< Triangle2 *> & vAllTriangles ) const
```

This command fetches the existing triangles

Parameters

out	*vAllTriangles*	is used to return the triangles

Note

that the lifetime of data from the Fade2D datastructures does exceed the lifetime of the Fade2D object.

6.10.3.23 getVertexPointers()

```
void GEOM_FADE25D::Fade_2D::getVertexPointers (
            std::vector< Point2 *> & vAllPoints ) const
```

Parameters

vAllPoints	is an empty vector of Point2 pointers.

Stores pointers to all vertices of the triangulation in vAllPoints. The order in which the points are stored is *not* necessarily the insertion order. For geometrically identical points which have been inserted multiple times, only one pointer exists. Thus vAllPoints.size() can be smaller than the number of inserted points.

Note

that the lifetime of data from the Fade2D datastructures does exceed the lifetime of the Fade2D object.

6.10.3.24 importTriangles()

```
Zone2* GEOM_FADE25D::Fade_2D::importTriangles (
            std::vector< Point2 > & vPoints,
            bool bReorientIfNeeded,
            bool bCreateExtendedBoundingBox )
```

This method imports triangles into an empty Fade object. The triangles do not need to satisfy the empty circle property.

Parameters

vPoints	contains the input vertices (3 subsequent ones per triangle)
bReorientIfNeeded	specifies if the orientations of the point triples shall be checked and corrected. If the point triples are certainly oriented in counterclockwise order then the orientation test can be skipped.
bCreateExtendedBoundingBox	can be used to insert 4 dummy points of an extended bounding box. This is convenient in some cases. Use false if you are unsure.

Returns

a pointer to a Zone2 object or NULL if the input data is invalid

Warning

This method requires 100% correct input. A frequent source of trouble is when client software reads points from an ASCII file. The ASCII format is convenient but it can **introduce rounding errors that cause intersections and flipped triangle orientations**. Thus it is highly recommended to transfer point coordinates with binary files. See also readPointsBIN() and writePointsBIN().

6.10.3.25 insert() [1/4]

```
Point2* GEOM_FADE25D::Fade_2D::insert (
            const Point2 & p )
```

Parameters

p	is the point to be inserted.

Returns

a pointer to the point in the triangulation

The triangulation keeps a copy of *p*. The return value is a pointer to this copy. If duplicate points are inserted, the triangulation does not create new copies but returns a pointer to the copy of the very first insertion.

Note

This method offers a very good performance but it is still faster if all points are passed at once, if possible.

6.10.3.26 insert() [2/4]

```
void GEOM_FADE25D::Fade_2D::insert (
            const std::vector< Point2 > & vInputPoints )
```

Parameters

vInputPoints	contains the points to be inserted.

Note

Use Fade_2D::setNumCPU() to activate multithreading

6.10.3.27 insert() [3/4]

```
void GEOM_FADE25D::Fade_2D::insert (
            const std::vector< Point2 > & vInputPoints,
            std::vector< Point2 *> & vHandles )
```

Parameters

vInputPoints	contains the points to be inserted.
vHandles	(empty) is used by Fade to return Point2 pointers

Internally, the triangulation keeps copies of the inserted points which are returned in *vHandles* (in the same order). If duplicate points are contained in vInputPoints then only one copy will be made and a pointer to this unique copy will be stored in vHandles for every occurance.

Note

Use Fade_2D::setNumCPU() to activate multithreading

6.10.3.28 insert() [4/4]

```
void GEOM_FADE25D::Fade_2D::insert (
            int numPoints,
            double * aCoordinates,
            Point2 ** aHandles )
```

Parameters

numPoints	is the number of points to be inserted
aCoordinates	is an array of *3n* double values, e.g. {x0,y0,z0,x1,y1,z1,...,xn,yn,zn}
aHandles	is an array with size *n* where pointers to the inserted points will be stored

Note

Use Fade_2D::setNumCPU() to activate multithreading

6.10.3.29 is2D()

```
bool GEOM_FADE25D::Fade_2D::is2D ( ) const
```

As long as all inserted points are collinear the triangulation does not contain triangles. This is clearly the case as long as less than three input points are present but it may also be the case when 3 or more points have been inserted when all these points are collinear. These points are then in a pending state, i.e. they will be triangulated as soon as the first non-collinear point is inserted.

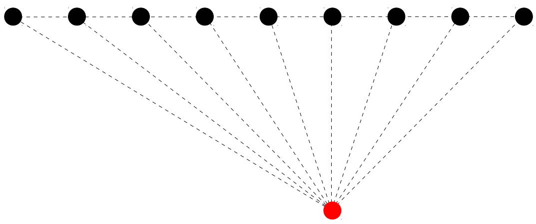


Figure 16 Triangles are generated as soon as the first non-collinear point is inserted.

Returns

true if at least one triangle exists
false otherwise

6.10.3.30 isConstraint() [1/3]

```
bool GEOM_FADE25D::Fade_2D::isConstraint (
            Triangle2 * pT,
            int ith ) const
```

Returns whether the edge in triangle *pT* which is opposite to the *ith* vertex is a constraint edge.

6.10.3.31 isConstraint() [2/3]

```
bool GEOM_FADE25D::Fade_2D::isConstraint (
            Point2 * p0,
            Point2 * p1 ) const
```

Returns whether the edge (p0,p1) is a constraint edge.

6.10.3.32 isConstraint() [3/3]

```
bool GEOM_FADE25D::Fade_2D::isConstraint (
            Point2 * pVtx ) const
```

Returns whether the vertex `pVtx` belongs to a constraint edge.

6.10.3.33 locate()

```
Triangle2* GEOM_FADE25D::Fade_2D::locate (
            const Point2 & p )
```

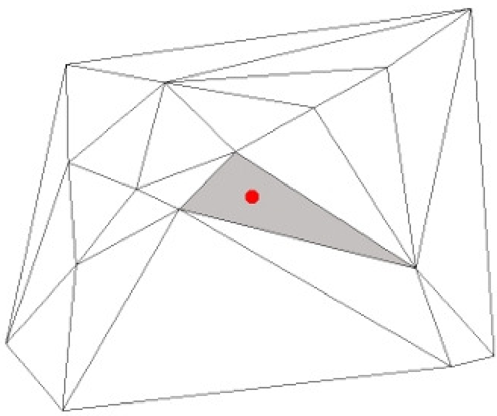


Figure 17 Point location

The Fade_2D class can be used as a data structure for point location. This method returns a pointer to a triangle which contains *p*.

Parameters

p	is the query point

Returns

a pointer to a Triangle2 object (or NULL if is2D()==false or if *p* is outside the triangulation)

6.10.3.34 measureTriangulationTime()

```
double GEOM_FADE25D::Fade_2D::measureTriangulationTime (
            std::vector< Point2 > & vPoints )
```

This method evaluates the performance of single- and multithreaded point insertion into a Delaunay triangulation.

Parameters

in	*vPoints*	are the points to be inserted

Returns

the total wall-time for point insertion in seconds

Note

The method cleans up the triangulation (objects, memory) on exit. Thus the time measured outside this method may be slightly larger than the returned time that is exactly the time needed to triangulate the input points.
Use Fade_2D::setNumCPU() to activate multithreading

6.10.3.35 numberOfPoints()

```
size_t GEOM_FADE25D::Fade_2D::numberOfPoints ( ) const
```

Returns

the number of points in the triangulation

Note

Due to possibly duplicate input points the number of points is not stored somewhere but freshly computed in O(n) time. This is fast but you are adviced to avoid calling this method over-frequently in a loop. Duplicate point insertions count only once.

6.10.3.36 numberOfTriangles()

```
size_t GEOM_FADE25D::Fade_2D::numberOfTriangles ( ) const
```

Returns

the number of triangles in the triangulation (or 0 as long as is2D() is false).

6.10.3.37 printLicense()

```
void GEOM_FADE25D::Fade_2D::printLicense ( ) const
```

Prints informations about the currently used license

6.10.3.38 refine()

```
void GEOM_FADE25D::Fade_2D::refine (
            Zone2 * pZone,
            double minAngleDegree,
            double minEdgeLength,
            double maxEdgeLength,
            bool bAllowConstraintSplitting )
```

Creates a mesh inside the area given by a Zone2 object.

Parameters

pZone	is the zone whose triangles are refined. Allowed zoneLocation values are ZL_INSIDE and ZL_BOUNDED.
minAngleDegree	(up to 30) is the minimum interior triangle angle
minEdgeLength	is a lower threshold on the edge length. Triangles with smaller edges are not refined.
maxEdgeLength	is an upper threshold on the edge length. Triangles with larger edges are always refined.
bAllowConstraintSplitting	specifies if constraint edges may be splitted

Note

The behavior of the present method had to be changed in Fade v1.39: Only ZL_INSIDE and ZL_BOUNDED zones are accepted. But you can easily convert other types of zones to ZL_BOUNDED using Zone2::convert↩ ToBoundedZone().

6.10.3.39 refineAdvanced()

```
void GEOM_FADE25D::Fade_2D::refineAdvanced (
            MeshGenParams * pParameters )
```

This method calls an advanced Delaunay mesh generator and grid mesher. The parameters are encapsulated in the MeshGenParams class. This class provides default parameters that can be used as is. Alternatively client code can derive from MeshGenParams and overwrite the methods and parameters to gain full control over the mesh generation process.

6.10.3.40 remove()

```
void GEOM_FADE25D::Fade_2D::remove (
            Point2 * pVertex )
```

Parameters

pVertex	shall be removed.

Note

`pVertex` must not be a vertex of a ConstraintGraph2 or ConstraintSegment2 object. If this is the case, the vertex is not removed and a warning is issued.

6.10.3.41 setNumCPU()

```
int GEOM_FADE25D::Fade_2D::setNumCPU (
            int numCPU )
```

Parameters

numCPU	is the number of CPU cores to be used. The special value `numCPU=0` means: auto-detect and use the number of available CPU cores.

Returns

the number of CPU cores that will be used (useful in case of auto-detection)

Characteristics:

- This setting affects Fade_2D::measureTriangulationTime() and Fade_2D::insert() which is by default single-threaded to avoid undeliberate nested multithreading (an application may run Fade in a thread).
- For technical reasons points should be inserted before any constraint segments so that the algorithm can fully benefit from multithreading.
- Fade continues support for very old compilers but multithreading is not available for VS2010 and CentOS6.4 library versions.

6.10.3.42 show() [1/2]

```
void GEOM_FADE25D::Fade_2D::show (
            const std::string & postscriptFilename,
            bool bWithConstraints = true ) const
```

show() is a convenience function for quick outputs with a default look. It is also possible to use the Visualizer2 class directly to draw arbitrary circles, line segments, vertices and labels with custom colors.

Parameters

postscriptFilename	is the output name, i.e. "myFile.ps"
bWithConstraints	specifies if constraint segments shall be shown (default: true)

6.10.3.43 show() [2/2]

```
void GEOM_FADE25D::Fade_2D::show (
            Visualizer2 * pVis,
            bool bWithConstraints = true ) const
```

This overload of the show() method allows to add further geometric primitives to the Visualizer2 object before it is finally written.

Parameters

pVis	is the pointer of a Visualizer2 object that may already contain geometric primitives or that may later be used to draw further elements
bWithConstraints	specifies if constraint segments shall be shown (default: true)

Note

The postscript file must be finalized with Visualizer2::writeFile().

6.10.3.44 showGeomview() [1/2]

```
void GEOM_FADE25D::Fade_2D::showGeomview (
            const std::string & filename )
```

Note

The free viewer Geomview can be used to view such files

6.10.3.45 showGeomview() [2/2]

```
void GEOM_FADE25D::Fade_2D::showGeomview (
            Visualizer3 * pVis )
```

Note

The free viewer Geomview can be used to view such files

6.10.3.46 statistics()

```
void GEOM_FADE25D::Fade_2D::statistics (
            const std::string & s ) const
```

Prints mesh statistics to stdout.

6.10.3.47 subscribe()

```
void GEOM_FADE25D::Fade_2D::subscribe (
            MsgType msgType,
            MsgBase * pMsg )
```

Parameters

msgType	is the type of message the subscriber shall receive, e.g. MSG_PROGRESS or MSG_WARNING
pMsg	is a pointer to a custom class derived from MsgBase

6.10.3.48 unsubscribe()

```
void GEOM_FADE25D::Fade_2D::unsubscribe (
            MsgType msgType,
            MsgBase * pMsg )
```

Parameters

msgType	is the type of message the subscriber shall not receive anymore
pMsg	is a pointer to a custom class derived from MsgBase

6.10.3.49 writeObj() [1/2]

```
void GEOM_FADE25D::Fade_2D::writeObj (
            const std::string & filename ) const
```

Visualizes the current triangulation. The *.obj format represents a 3D scene but can also be used with 2D triangles (all z-values are zero then).

6.10.3.50 writeObj() [2/2]

```
void GEOM_FADE25D::Fade_2D::writeObj (
            const std::string & filename,
            Zone2 * pZone ) const
```

Visualizes a Zone2 object. The *.obj format represents a 3D scene but can also be used with 2D triangles (all z-values are zero then).

6.10.3.51 writeWebScene() [1/2]

```
void GEOM_FADE25D::Fade_2D::writeWebScene (
            const char * path ) const
```

Made for terrain visualizations in 2.5D but will work also for 2D.

6.10.3.52 writeWebScene() [2/2]

```
void GEOM_FADE25D::Fade_2D::writeWebScene (
            const char * path,
            Zone2 * pZone ) const
```

Made for terrain visualizations in 2.5D but will work also for 2D.

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

- Fade_2D.h

6.11 GEOM_FADE25D::Func_gtEdge2D Struct Reference

Functor to sort edges by 2d length (descending)

```
#include <Edge2.h>
```

Public Member Functions

- bool **operator()** (const Edge2 &e0, const Edge2 &e1) const

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

- Edge2.h

6.12 GEOM_FADE25D::Func_ltEdge25D Struct Reference

Functor to sort edges by 2.5d length (ascending)

```
#include <Edge2.h>
```

Public Member Functions

- bool **operator()** (const Edge2 &e0, const Edge2 &e1) const

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

- Edge2.h

6.13 GEOM_FADE25D::Func_ltEdge2D Struct Reference

Functor to sort edges by 2d length (ascending)

```
#include <Edge2.h>
```

Public Member Functions

- bool **operator()** (const Edge2 &e0, const Edge2 &e1) const

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

- Edge2.h

6.14 GEOM_FADE25D::Func_ltPointXYZ Struct Reference

Public Member Functions

- bool **operator()** (const Point2 &p0, const Point2 &p1) const

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

- Point2.h

6.15 GEOM_FADE25D::Func_ltUndirected Struct Reference

Public Member Functions

- bool **operator()** (const Edge2 &eA, const Edge2 &eB) const

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

- Edge2.h

6.16 GEOM_FADE25D::IsoContours Class Reference

IsoContours uses a fast datastructure to compute intersections of horizontal planes with a given list of triangles.

`#include <IsoContours.h>`

Public Member Functions

- **IsoContours** (std::vector< Triangle2 *> &vTriangles)
- IsoContours (std::vector< Point2 > &vCorners, const Vector2 &dirVec)
- double getMinHeight ()
- double getMaxHeight ()
- bool getContours (double height, std::vector< std::vector< Segment2 > > &vvContours, bool bVerbose, bool bAutoPerturbate=true)
- void getProfile (const Point2 &p, std::vector< Segment2 > &vSegmentsOut)

Protected Attributes

- std::vector< Triangle2 * > **vTriangles**

6.16.1 Detailed Description

See also

http://www.geom.at/terrain-triangulation/

6.16.2 Constructor & Destructor Documentation

6.16.2.1 IsoContours()

```
GEOM_FADE25D::IsoContours::IsoContours (
            std::vector< Point2 > & vCorners,
            const Vector2 & dirVec )
```

Experimental feature

IsoContours can be used to create profiles (slices). This is a new and still experimental feature.

Parameters

vCorners	contains 3∗n points to specify n triangles, i.e. it is a corners-list where 3 subsequent points define a triangle.
dirVec	specifies the slice direction to compute profiles

6.16.3 Member Function Documentation

6.16.3.1 getContours()

```
bool GEOM_FADE25D::IsoContours::getContours (
            double height,
            std::vector< std::vector< Segment2 > > & vvContours,
            bool bVerbose,
            bool bAutoPerturbate = true )
```

Get Contours

Computes the intersection of a horizontal plane at a certain height with all triangles and returns a vector of assembled polygons and polylines. The method works only for `height` values that do not occur as heights of any of the vertices. It returns false in such as case except `bAutoPerturbate=true`. In this case a tiny offset is automatically added to `height`.

Note

Polylines that end in the middle of a terrain can not exist and if you see such lines in the output then these are not only individual line segments but extremely narrow polygons - enforced by your geometric setting. You can numerically inspect those lines to verify that.

6.16.3.2 getMaxHeight()

```
double GEOM_FADE25D::IsoContours::getMaxHeight ( )
```

The the maximum height

Returns the largest z-coordinate

6.16.3.3 getMinHeight()

```
double GEOM_FADE25D::IsoContours::getMinHeight ( )
```

Get the minimum height

Returns the smallest z coordinate

6.16.3.4 getProfile()

```
void GEOM_FADE25D::IsoContours::getProfile (
            const Point2 & p,
            std::vector< Segment2 > & vSegmentsOut )
```

Get Profile

This is a new method to compute profiles i.e., to produce slices orthogonal to a direction specified in the constructor IsoContours(std::vector<Point2>& vCorners,const Vector2& dirVec)

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

- IsoContours.h

6.17 GEOM_FADE25D::Label Class Reference

Text-Label.

`#include <Label.h>`

Public Member Functions

- Label (const Point2 &p_, const std::string &s_, bool bWithMark_=true, int fontSize_=8)

    Constructs a Text-Label.

Public Attributes

- Point2 **p**
- std::string **s**
- bool **bWithMark**
- int **fontSize**

6.17.1 Detailed Description

See also

Visualizer2 where Label objects are used for visualizations

6.17.2 Constructor & Destructor Documentation

6.17.2.1 Label()

```
GEOM_FADE25D::Label::Label (
            const Point2 & p_,
            const std::string & s_,
            bool bWithMark_ = true,
            int fontSize_ = 8 )
```

Parameters

*p_*	is the point where the label appears
*s_*	is the text to be shown
*bWith←Mark_*	switches between text-only and text-with-mark
*fontSize_*	

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

- Label.h

6.18 GEOM_FADE25D::MeshGenParams Class Reference

Parameters for the mesh generator.

```
#include <MeshGenParams.h>
```

Public Member Functions

- **MeshGenParams** (Zone2 ∗pZone_)
- virtual double getMaxTriangleArea (Triangle2 ∗pT)

  getMaxTriangleArea(Triangle2∗ pT)
- virtual double getMaxEdgeLength (Triangle2 ∗pT)

  getMaxEdgeLength(Triangle2∗ pT)
- void addLockedConstraint (ConstraintSegment2 ∗pConstraintSegment)

  Constraint Segments that shall not be splitted.

Public Attributes

- Fade_2D ∗ pHeightGuideTriangulation

  pHeightGuideTriangulation
- double maxHeightError

  maxHeightError
- Zone2 ∗ pZone

  Zone to be meshed.
- double minAngleDegree

  Minimum interior triangle angle.
- double minEdgeLength

  Minimum edge length.
- double maxEdgeLength

  Maximum edge length.
- double maxTriangleArea

  maxTriangleArea
- bool bAllowConstraintSplitting

  bAllowConstraintSplitting
- double growFactor

  growFactor

- double growFactorMinArea

    growFactorMinArea
- double capAspectLimit

    capAspectLimit
- Vector2 gridVector

    gridVector
- double gridLength

    gridLength
- bool bKeepExistingSteinerPoints

    Steiner points from previous refinements.
- int command

    Command.

6.18.1 Detailed Description

This class serves as container for mesh generator parameters. Client code can provide a class which derives from MeshGenParams and which provides custom implementations of the getMaxTriangleArea(Triangle∗ pT) method or the getMaxEdgeLength(Triangle∗ pT) method in order to gain control over the local density of the generated mesh. When the meshing algorithm decides if a certain triangle T must be refined, then it calls these functions.

See also

`http://www.geom.at/advanced-mesh-generation/`

6.18.2 Member Function Documentation

6.18.2.1 addLockedConstraint()

```
void GEOM_FADE25D::MeshGenParams::addLockedConstraint (
            ConstraintSegment2 * pConstraintSegment )
```

In case that some ConstraintSegment2 can be splitted and others must not be splitted use `bAllowConstraintSplitting=true` and add the ones that must not be splitted.

6.18.2.2 getMaxEdgeLength()

```
virtual double GEOM_FADE25D::MeshGenParams::getMaxEdgeLength (
            Triangle2 * pT ) [inline], [virtual]
```

Parameters

pT	is a triangle for which the meshing algorithm checks if it must be refined.

The default implementation of the present class returns the value maxEdgeLength (which is DBL_MAX if not changed by the user). This method can be overridden by the client software in order to control the local mesh density.

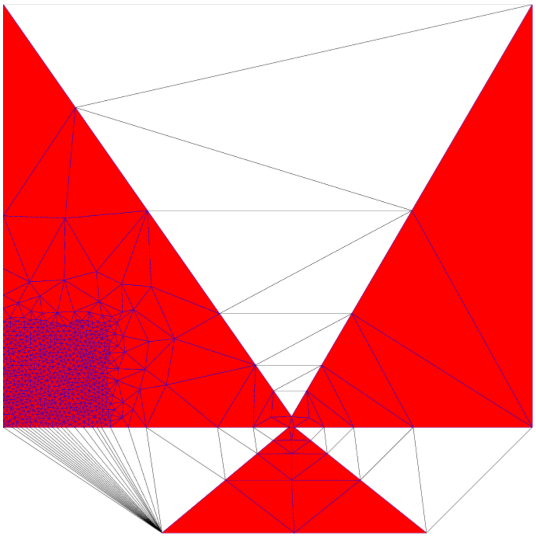


Figure 18 User Controlled Mesh Density, Edge Length

6.18.2.3 getMaxTriangleArea()

```
virtual double GEOM_FADE25D::MeshGenParams::getMaxTriangleArea (
            Triangle2 * pT )  [inline], [virtual]
```

Parameters

pT	is a triangle for which the meshing algorithm checks if it must be refined.

The default implementation of the present class returns the value maxTriangleArea (which is the default value DBL_MAX if not changed by the user). This method can be overridden by the client software in order to control the local mesh density.

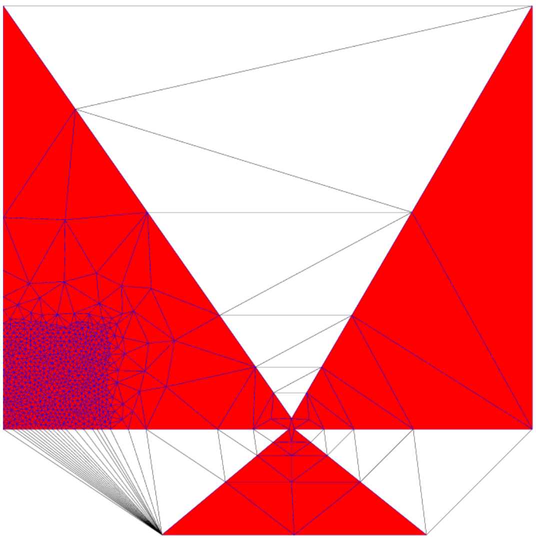


Figure 19 User Controlled Mesh Density, Area

6.18.3 Member Data Documentation

6.18.3.1 bAllowConstraintSplitting

`bool GEOM_FADE25D::MeshGenParams::bAllowConstraintSplitting`

Defines if constraint segments can be splitted. Default: yes

6.18.3.2 bKeepExistingSteinerPoints

`bool GEOM_FADE25D::MeshGenParams::bKeepExistingSteinerPoints`

A previous call to refine() or refineAdvanced() may have created Steiner points. These may be partially or entirely removed during a later refinement call, even (!) if this later refinement takes place in a different zone. It depends on your application if this behavior is desired or not. Usually you want to preserve the points, thus the default value of /p bKeepExistingSteinerPoints is true.

6.18.3.3 capAspectLimit

`double GEOM_FADE25D::MeshGenParams::capAspectLimit`

Limits the quotient edgeLength / height. Default value: 10.0

6.18.3.4 command

`int GEOM_FADE25D::MeshGenParams::command`

A command for development, not for public use. Will vanish soon.

6.18.3.5 gridLength

`double GEOM_FADE25D::MeshGenParams::gridLength`

Set gridLength $>$ 0 to mesh large enough areas with grid points. Border areas and narrow stripes where a grid does not fit are automatically meshed using classic Delaunay methods. By default gridLength=0 (off).

Note

The length of the diagonals in the grid is sqrt(2)$*$gridLength and the algorithm may automatically adapt the gridLength a bit such that the grid fits better into the shape.

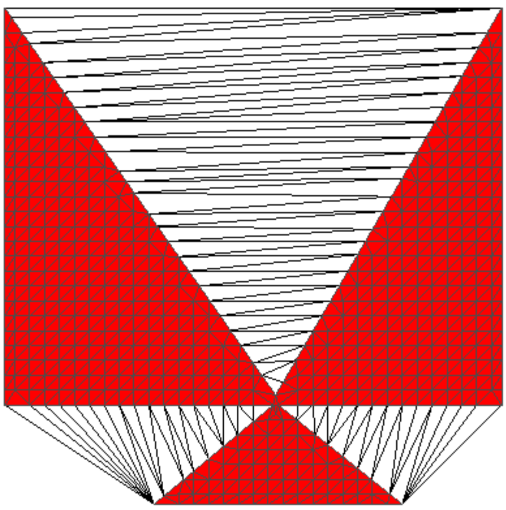


Figure 20 Grid Meshing axis aligned

6.18.3.6 gridVector

`Vector2 GEOM_FADE25D::MeshGenParams::gridVector`

When grid-meshing is used the grid is aligned to the `gridVector`. By default `gridVector` is axis aligned.

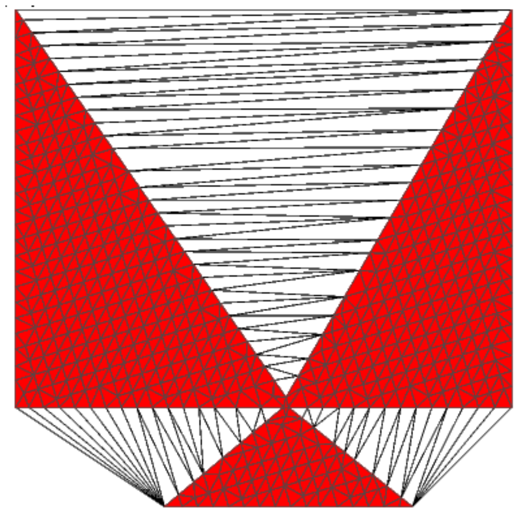


Figure 21 Grid Meshing along Vector2(1.0,0.3,0.0)

6.18.3.7 growFactor

`double GEOM_FADE25D::MeshGenParams::growFactor`

Limits the growth of adjacent triangles. The mesh is constructed such that for any two adjacent triangles t0 and t1 (where t0 is the larger one) area(t0)/area(t1) $<$ growFactor. Recommendation: growFactor$>$5.0, Default: grow←
Factor=DBL_MAX

6.18.3.8 growFactorMinArea

`double GEOM_FADE25D::MeshGenParams::growFactorMinArea`

The growFactor value is ignored for triangles with a smaller area than growFactorMinArea. This value prevents generation of hundreds of tiny triangles around one that is unusually small. Default: 0.001

6.18.3.9 maxEdgeLength

`double GEOM_FADE25D::MeshGenParams::maxEdgeLength`

This value is returned by the default implementation of getMaxEdgeLength(Triangle$*$ pT). Larger edges are automatically subdivided. If a custom implementation of getMaxEdgeLength(Triangle$*$ pT) is provided then this value is ignored. Default value: DBL_MAX.

6.18.3.10 maxHeightError

`double GEOM_FADE25D::MeshGenParams::maxHeightError`

If pHeightGuideTriangulation is set and the height error exceeds locally maxHeightError then the triangulation is further refined.

6.18.3.11 maxTriangleArea

```
double GEOM_FADE25D::MeshGenParams::maxTriangleArea
```

This value is returned by the default implementation of getMaxTriangleArea(Triangle∗ pT). Larger triangles are automatically subdivided. If a custom implementation of getMaxTriangleArea(Triangle∗ pT) is provided then this value is ignored. Default value: DBL_MAX.

6.18.3.12 minAngleDegree

```
double GEOM_FADE25D::MeshGenParams::minAngleDegree
```

Minimum interior angle: Default: 20.0, maximum: 30.0

6.18.3.13 minEdgeLength

```
double GEOM_FADE25D::MeshGenParams::minEdgeLength
```

Edges below the minimum length are not subdivided. This parameter is useful to avoid tiny triangles. Default: 0.001

6.18.3.14 pHeightGuideTriangulation

```
Fade_2D* GEOM_FADE25D::MeshGenParams::pHeightGuideTriangulation
```

When new vertices are created then their height (z-coordinate) is usually computed from the existing triangles. In a situation where an extra triangulation with more accurate heights exists this extra triangulation cn be set als height guide triangulation. In this case the z-coordinates are computed from the triangles of the height guide triangulation.

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

- MeshGenParams.h

6.19 GEOM_FADE25D::MsgBase Class Reference

MsgBase

```
#include <MsgBase.h>
```

Public Member Functions

- virtual void update (MsgType msgType, const std::string &s, double d)=0

  update

6.19.1 Detailed Description

MsgBase is a base class from which message subscriber classes (for example widgets, progress bars, ...) can be derived which then receive messages (progress, warnings, ...) from Fade.

See also

http://www.geom.at/progress-bar/

6.19.2 Member Function Documentation

6.19.2.1 update()

```
virtual void GEOM_FADE25D::MsgBase::update (
             MsgType msgType,
             const std::string & s,
             double d )  [pure virtual]
```

This method must be defined in derived classes. It is automatically called everytime Fade has a message of type `msgType`.

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

- MsgBase.h

6.20 GEOM_FADE25D::Point2 Class Reference

Point.

`#include <Point2.h>`

Public Member Functions

- Point2 (const double x_, const double y_, const double z_)

  Constructor.
- Point2 ()

  Default constructor.
- Point2 (const Point2 &p_)

  Copy constructor.
- Point2 & **operator=** (const Point2 &other)
- void print ()

  Print.
- double x () const

  Get the x-coordinate.
- double y () const

  Get the y-coordinate.
- double z () const

  Get the z-coordinate.
- void xyz (double &x_, double &y_, double &z_) const

  Get the x-, y- and z-coordinate.
- void xy (double &x_, double &y_) const

  Get the x- and y-coordinate.
- void setHeight (double z)

  Set the z-coordinate.
- double getMaxAbs () const

  Get max(abs(x),abs(y))
- bool operator< (const Point2 &p) const

Less than operator.
- bool operator> (const Point2 &p) const

  Greater than operator.
- bool operator== (const Point2 &p) const

  Equality operator.
- bool samePoint (const Point2 &p) const

  Equality operator.
- bool operator!= (const Point2 &p) const

  Inequality operator.
- Triangle2 * getIncidentTriangle () const

  Get the associated triangle.
- void set (const double x_, const double y_, const double z_, int customIndex_)

  Set the coordinates.
- void set (const Point2 &pnt)

  Set the coordiantes.
- void setCustomIndex (int customIndex_)

  Set a custom index.
- int getCustomIndex () const

  Get the custom index.
- void setIncidentTriangle (Triangle2 *pT)

  Associate a triangle with the point.
- Vector2 operator- (const Point2 &other) const

  *Returns a vector from other to *this.*
- Point2 operator+ (const Vector2 &vec) const

  Add vector and point.
- Point2 operator- (const Vector2 &vec) const

  Subtract vector from point.

Protected Attributes

- double **coordX**
- double **coordY**
- double **coordZ**
- Triangle2 * **pAssociatedTriangle**
- int **customIndex**

Friends

- class **Dt2**
- std::ostream & **operator**<< (std::ostream &stream, const Point2 &pnt)
- std::istream & **operator**>> (std::istream &stream, Point2 &pnt)

6.20.1 Detailed Description

This class represents a point in 2D with x- and y-coordinates and an additional pointer to an associated triangle.

6.20.2 Constructor & Destructor Documentation

6.20.2.1 Point2() [1/3]

```
GEOM_FADE25D::Point2::Point2 (
            const double x_,
            const double y_,
            const double z_ )  [inline]
```

Parameters

x↩ _↩	x-coordinate
y↩ _↩	y-coordinate
z↩ _↩	z-coordinate

6.20.2.2 Point2() [2/3]

```
GEOM_FADE25D::Point2::Point2 ( )  [inline]
```

6.20.2.3 Point2() [3/3]

```
GEOM_FADE25D::Point2::Point2 (
            const Point2 & p_ )  [inline]
```

Create a point as a copy of p_. The associated triangle pointer is initialized to NULL

6.20.3 Member Function Documentation

6.20.3.1 getCustomIndex()

```
int GEOM_FADE25D::Point2::getCustomIndex ( ) const  [inline]
```

Returns

the custom index.

Note

The custom index defaults to -1. It is not the index of the point in the triangulation (such an index does not exist) but an arbitrary value which can be set by the user.

See also

void setCustomIndex(int customIndex_)
A best practices example that deals with indices: http://www.geom.at/runtime/

6.20.3.2 getIncidentTriangle()

```
Triangle2* GEOM_FADE25D::Point2::getIncidentTriangle ( ) const [inline]
```

Returns

the associated triangle

6.20.3.3 getMaxAbs()

```
double GEOM_FADE25D::Point2::getMaxAbs ( ) const [inline]
```

6.20.3.4 operator"!=()

```
bool GEOM_FADE25D::Point2::operator!= (
            const Point2 & p ) const [inline]
```

Compares the x and y coordinates

Note

Although a point has a z-coordinate in the 2.5D version only x and y a compared by this method

6.20.3.5 operator<()

```
bool GEOM_FADE25D::Point2::operator< (
            const Point2 & p ) const [inline]
```

Compares the x and y coordinates

Note

Although a point has a z-coordinate in the 2.5D version only x and y a compared by this method

6.20.3.6 operator==()

```
bool GEOM_FADE25D::Point2::operator== (
            const Point2 & p ) const [inline]
```

Compares the x and y coordinates

Note

Although a point has a z-coordinate in the 2.5D version only x and y a compared by this method

6.20.3.7 operator>()

```
bool GEOM_FADE25D::Point2::operator> (
            const Point2 & p ) const  [inline]
```

Compares the x and y coordinates

Note

Although a point has a z-coordinate in the 2.5D version only x and y a compared by this method

6.20.3.8 print()

```
void GEOM_FADE25D::Point2::print ( )  [inline]
```

Internal, prints the coordinates to stdout

6.20.3.9 samePoint()

```
bool GEOM_FADE25D::Point2::samePoint (
            const Point2 & p ) const  [inline]
```

Compares the x,y,z coordinates while operator==() compares only x,y

6.20.3.10 set() [1/2]

```
void GEOM_FADE25D::Point2::set (
            const double x_,
            const double y_,
            const double z_,
            int customIndex_ )  [inline]
```

Internal method

Parameters

*x_*	x-coordinate
*y_*	y-coordinate
*z_*	z-coordinate
*custom←↩ Index_*	Arbitrary index, use -1 if not required

6.20.3.11 set() [2/2]

```
void GEOM_FADE25D::Point2::set (
            const Point2 & pnt )  [inline]
```

Parameters

pnt	is the point whose coordinates are assigned to the current point

6.20.3.12 setCustomIndex()

```
void GEOM_FADE25D::Point2::setCustomIndex (
            int customIndex_ )  [inline]
```

An arbitrary index can be assigned to a point. Use getCustomIndex() to retrieve it later.

Note

This method is provided for the users' convenience. It has nothing to do with the internal data structures of Fade 2D and using this method is optional. By default this index is -1.

See also

int getCustomIndex()
A best practices example that deals with indices: http://www.geom.at/runtime/

6.20.3.13 setHeight()

```
void GEOM_FADE25D::Point2::setHeight (
            double z )  [inline]
```

Allows to exchange the z-coordinate

6.20.3.14 setIncidentTriangle()

```
void GEOM_FADE25D::Point2::setIncidentTriangle (
            Triangle2 * pT )  [inline]
```

Parameters

pT	will be associated with the triangle

6.20.3.15 x()

```
double GEOM_FADE25D::Point2::x ( ) const  [inline]
```

Returns

the x-coordinate

6.20.3.16 xy()

```
void GEOM_FADE25D::Point2::xy (
            double & x_,
            double & y_ ) const  [inline]
```

Parameters

x_	x-coordinate
y_	y-coordinate

6.20.3.17 xyz()

```
void GEOM_FADE25D::Point2::xyz (
            double & x_,
            double & y_,
            double & z_ ) const  [inline]
```

Parameters

x_	x-coordinate
y_	y-coordinate
z_	z-coordinate

6.20.3.18 y()

```
double GEOM_FADE25D::Point2::y ( ) const  [inline]
```

Returns

the y-coordinate

6.20.3.19 z()

```
double GEOM_FADE25D::Point2::z ( ) const  [inline]
```

Returns

the z-coordinate

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

- Point2.h

6.21 GEOM_FADE25D::Segment2 Class Reference

Segment.

`#include <Segment2.h>`

Public Member Functions

- Segment2 (const Point2 &src_, const Point2 &trg_)

    Create a Segment2.
- Segment2 ()
- Point2 getSrc () const
- Point2 getTrg () const
- double getSqLen2D () const
- double getSqLen25D () const
- void swapSrcTrg ()
- bool operator== (const Segment2 &other) const

Protected Attributes

- Point2 **src**
- Point2 **trg**

Friends

- std::ostream & **operator**<< (std::ostream &stream, Segment2 seg)

6.21.1 Detailed Description

6.21.2 Constructor & Destructor Documentation

6.21.2.1 Segment2() [1/2]

```
GEOM_FADE25D::Segment2::Segment2 (
            const Point2 & src_,
            const Point2 & trg_ )
```

Parameters

*src_*	First endpoint (source)
*trg_*	Second endpoint (target)

6.21.2.2 Segment2() [2/2]

```
GEOM_FADE25D::Segment2::Segment2 ( )
```

Create a Segment2 Default constructor

6.21.3 Member Function Documentation

6.21.3.1 getSqLen25D()

```
double GEOM_FADE25D::Segment2::getSqLen25D ( ) const
```

Get the squared length (2.5D version)

6.21.3.2 getSqLen2D()

```
double GEOM_FADE25D::Segment2::getSqLen2D ( ) const
```

Get the squared length

6.21.3.3 getSrc()

```
Point2 GEOM_FADE25D::Segment2::getSrc ( ) const
```

Get the source point

Returns

the source point

6.21.3.4 getTrg()

```
Point2 GEOM_FADE25D::Segment2::getTrg ( ) const
```

Get the target point

Returns

the target point

6.21.3.5 operator==()

```
bool GEOM_FADE25D::Segment2::operator== (
            const Segment2 & other ) const
```

operator==

Undirected equality operator

6.21.3.6 swapSrcTrg()

```
void GEOM_FADE25D::Segment2::swapSrcTrg ( )
```

Internally swaps the source and target point

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

- Segment2.h

6.22 GEOM_FADE25D::SegmentChecker Class Reference

SegmentChecker identifies intersecting line segments.

```
#include <SegmentChecker.h>
```

Public Member Functions

- SegmentChecker (const std::vector< Segment2 ∗> &vSegments_)
- Segment2 ∗ getSegment (size_t i) const
- size_t getNumberOfSegments () const
- int getIndex (Segment2 ∗pSeg) const
- void subscribe (MsgType msgType, MsgBase ∗pMsg)
- void unsubscribe (MsgType msgType, MsgBase ∗pMsg)
- void showSegments (const std::string &name) const
- void showIllegalSegments (bool bAlsoEndPointIntersections, const std::string &name) const
- void getIllegalSegments (bool bAlsoEndPointIntersections, std::vector< Segment2 ∗> &vIllegalSegments↩ Out) const
- SegmentIntersectionType getIntersectionType (const Segment2 ∗pSeg1, const Segment2 ∗pSeg2) const
- void getIntersectors (Segment2 ∗pTestSegment, bool bAlsoEndPointIntersections, std::vector< std::pair< Segment2 ∗, SegmentIntersectionType > > &vIntersectorsOut) const
- void getIntersectionPoint (SegmentIntersectionType sit, const Segment2 &seg0, const Segment2 &seg1, Point2 &ispOut0, Point2 &ispOut1) const
- void getIntersectionSegment (const Segment2 &seg0, const Segment2 &seg1, Segment2 &issOut0, Segment2 &issOut1) const
- std::string getIntersectionTypeString (SegmentIntersectionType sit) const

6.22.1 Detailed Description

SegmentChecker takes a bunch of line segments and fully automatically identifies illegal segment intersections. The intersection points can be computed in 2D and in 2.5D. Further this class offers visualization methods. Due to the underlying datastructure the search algorithm scales very well to large inputs.

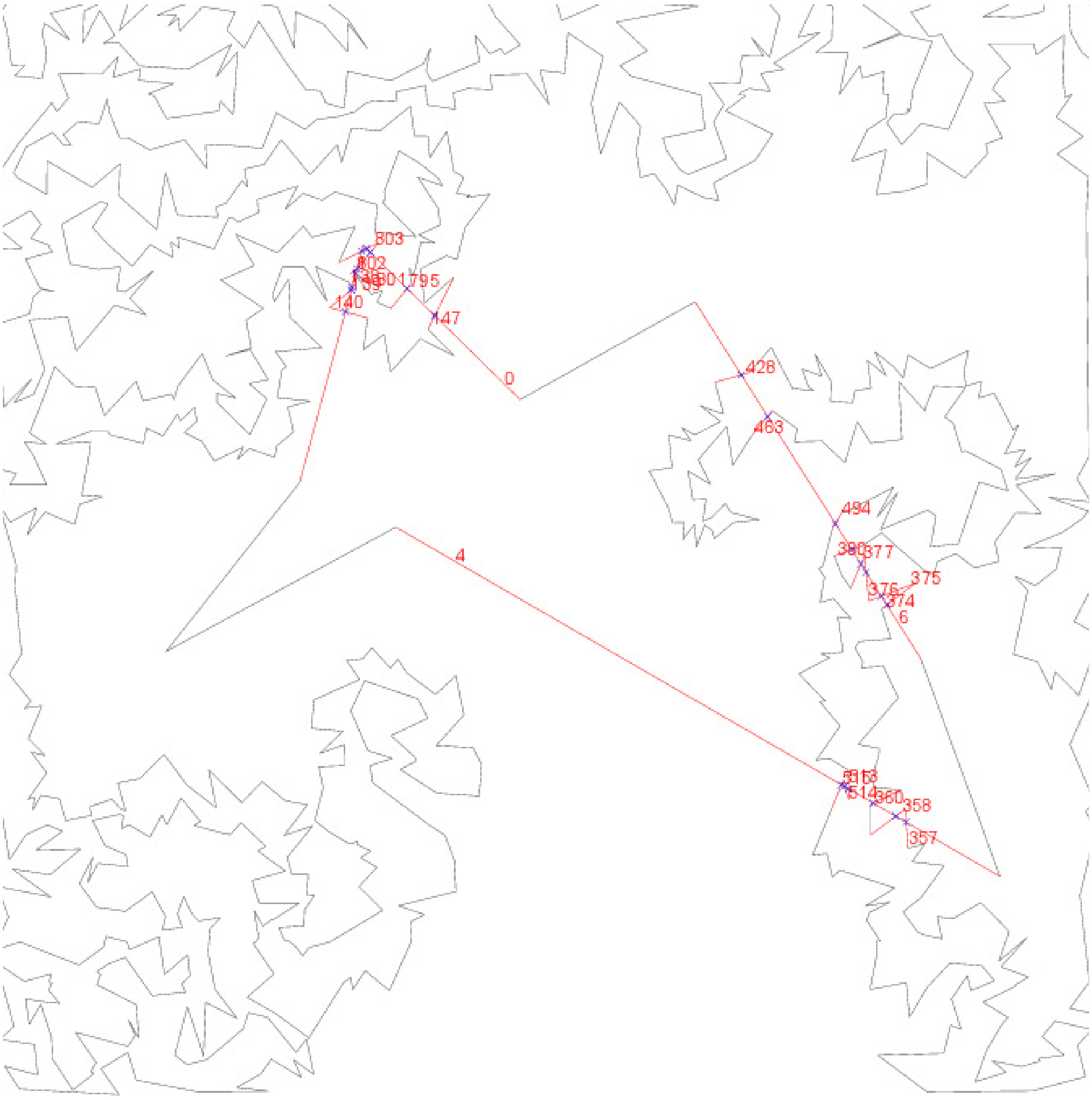


Figure 22 Polylines: Intersecting segments are automatically found

See also

http://www.geom.at/segment-checker/

6.22.2 Constructor & Destructor Documentation

6.22.2.1 SegmentChecker()

```
GEOM_FADE25D::SegmentChecker::SegmentChecker (
            const std::vector< Segment2 *> & vSegments_ )  [explicit]
```

Internally this constructor prepares a data structure from `vSegments` that enables efficient spatial searches. The time complexity is O(n∗log(n)).

Parameters

v↩ Segments↩ _	contains the segments to be checked

6.22.3 Member Function Documentation

6.22.3.1 getIllegalSegments()

```
void GEOM_FADE25D::SegmentChecker::getIllegalSegments (
            bool bAlsoEndPointIntersections,
            std::vector< Segment2 *> & vIllegalSegmentsOut ) const
```

Get illegal segments

Returns segments which are involved in intersections. Intersections at endpoints are only reported when `b↩ AlsoEndPointIntersections` is true. The asymptotic time consumption for the lookup per segment S is O(log(n)+k) where k is the number of segments that intersect the minimal bounding box of S. Thus, for n segments the method takes O(n∗(log(n)+k)) time.

Parameters

	bAlsoEndPointIntersections	specifies if intersections at endpoints shall be detected
`out`	*vIllegalSegmentsOut*	is the output vector

6.22.3.2 getIndex()

```
int GEOM_FADE25D::SegmentChecker::getIndex (
            Segment2 * pSeg ) const
```

Returns the index of a segment

Parameters

pSeg	is the segment whose index is returned

6.22.3.3 getIntersectionPoint()

```
void GEOM_FADE25D::SegmentChecker::getIntersectionPoint (
            SegmentIntersectionType sit,
            const Segment2 & seg0,
            const Segment2 & seg1,
```

```
            Point2 & ispOut0,
            Point2 & ispOut1 ) const
```

Compute the intersection point(s) of two segments.

Use getIntersectionType() to determine the segment intersection type `sit` before. Call this function only when the intersection type is SIT_POINT or SIT_ENDPOINT.

Parameters

	sit	is the segment intersection type (SIT_POINT or SIT_ENDPOINT for the present method)
	seg0,seg1	are the intersecting segments
out	*ispOut0*	output intersection point at `seg0`
out	*ispOut1*	output intersection point at `seg1`

The resulting two output intersection points `ispOut0` and `ispOut1` have always the same (x,y) coordinates but possibly different heights `z`.

Note

`pSeg1` and `pSeg2` do not need to be from the set of segments that have been used as argument for the constructor of the SegmentChecker. You can use any segments.

6.22.3.4 getIntersectionSegment()

```
void GEOM_FADE25D::SegmentChecker::getIntersectionSegment (
            const Segment2 & seg0,
            const Segment2 & seg1,
            Segment2 & issOut0,
            Segment2 & issOut1 ) const
```

Compute the intersection segment(s) of two collinear intersecting segments.

Use getIntersectionType() to determine the segment intersection type `sit` before. Call this function only when the intersection type is SIT_SEGMENT.

Parameters

	seg0,seg1	are intersecting segments such that their SegmentIntersectionType is SIT_SEGMENT
out	*issOut0*	intersection segment at `seg0`
out	*issOut1*	intersection segment at `seg1`

The two output segments have always the same (x,y) coordinates but possibly diffent heights `z`.

Note

`pSeg1` and `pSeg2` do not need to be from the set of segments that have been used as argument for the constructor of the SegmentChecker. You can use any segments.

6.22.3.5 getIntersectionType()

```
SegmentIntersectionType GEOM_FADE25D::SegmentChecker::getIntersectionType (
            const Segment2 * pSeg1,
            const Segment2 * pSeg2 ) const
```

Get the intersection type of two segments

Parameters

pSeg1,pSeg2	are the segments to be checked

Returns

SIT_NONE (no intersection),
SIT_SEGMENT (collinear intersection),
SIT_POINT (intersection somewhere between the endpoints) or
SIT_ENDPOINT (endpoint intersection)

Note

`pSeg1` and `pSeg2` do not need to be from the set that has been used to initialize the present object

6.22.3.6 getIntersectionTypeString()

```
std::string GEOM_FADE25D::SegmentChecker::getIntersectionTypeString (
            SegmentIntersectionType sit ) const
```

Return the intersection type as a human readable string. This is a convenience function

Parameters

sit	is an intersection type to be converted to a string

6.22.3.7 getIntersectors()

```
void GEOM_FADE25D::SegmentChecker::getIntersectors (
            Segment2 * pTestSegment,
            bool bAlsoEndPointIntersections,
            std::vector< std::pair< Segment2 *, SegmentIntersectionType > > & vIntersectors↩
Out ) const
```

Return segments that intersect a certain segment along with their intersection type

Parameters

	pTestSegment	is the segment to be analyzed
	bAlsoEndPointIntersections	specifies if intersections of type SIT_ENDPOINT shall also be reported.
out	*vIntersectorsOut*	is the output vector. Segments intersecting `pTestSegment` are added to `vIntersectorsOut` along with their intersection type.

Note

When vIntersectorsOut is non-empty, it is not cleared but the intersected segments are added.

The time complexity is O(log(n)+k) where n is the number of segments and k is the number of intersections for `pTestSegment`.

6.22.3.8 getNumberOfSegments()

```
size_t GEOM_FADE25D::SegmentChecker::getNumberOfSegments ( ) const
```

Returns the number of segments contained in this SegmentChecker object

6.22.3.9 getSegment()

```
Segment2* GEOM_FADE25D::SegmentChecker::getSegment (
            size_t i ) const
```

Returns the i-th segment

Parameters

i	is the index of the segment to be returned

6.22.3.10 showIllegalSegments()

```
void GEOM_FADE25D::SegmentChecker::showIllegalSegments (
            bool bAlsoEndPointIntersections,
            const std::string & name ) const
```

Write a postscript file, highlight illegal segments

Parameters

bAlsoEndPointIntersections	specifies if intersections at endpoints are also illegal
name	is the output filename

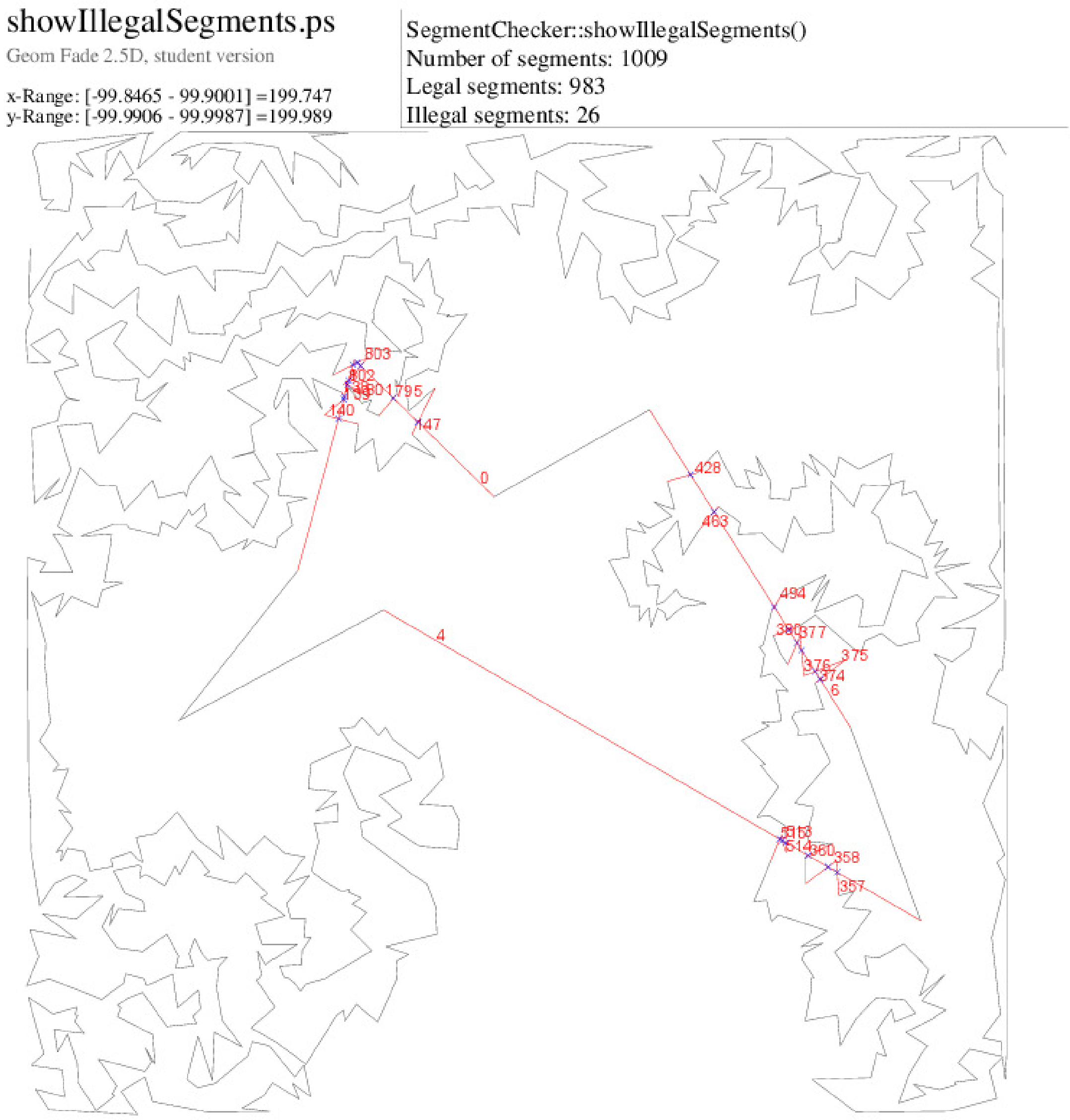


Figure 23 Visualization of polyline intersections

6.22.3.11 showSegments()

```
void GEOM_FADE25D::SegmentChecker::showSegments (
            const std::string & name ) const
```

Write all segments, with and without intersection, to a postscript file

Parameters

name	is the output filename

Figure 24 Line segments written to a postscript file

6.22.3.12 subscribe()

```
void GEOM_FADE25D::SegmentChecker::subscribe (
            MsgType msgType,
            MsgBase * pMsg )
```

Register a progress bar object

The SegmentChecker does its job typically in fractions of a second. But inputs may contain a quadratic number of intersections and such tasks take a while. Therefore a user defined message object (your own progress-bar class) can be registered in order to get progress updates. This step is optional.

Parameters

msgType	is the message type. For progress information the type is always MSG_PROGRESS
pMsg	is a user defined progress bar which derives from Fade's MsgBase.

6.22.3.13 unsubscribe()

```
void GEOM_FADE25D::SegmentChecker::unsubscribe (
            MsgType msgType,
            MsgBase * pMsg )
```

Unregister a progress bar object

Parameters

msgType	is the message type. For progress information the type is always MSG_PROGRESS
pMsg	is a user defined class which derives from Fade's MsgBase

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

- SegmentChecker.h

6.23 GEOM_FADE25D::Triangle2 Class Reference

Triangle.

`#include <Triangle2.h>`

Public Member Functions

- Triangle2 ()
  Constructor.
- Point2 * getCorner (const int ith) const
  Get the i-th corner of the triangle.
- std::pair< Point2, bool > getDual () const
  Get the dual Voronoi vertex.
- Point2 getBarycenter () const
  Get the barycenter of a triangle.
- Vector2 getNormalVector () const
  Get the normal vector of a triangle.
- double getInteriorAngle2D (int ith) const
  Get interior 2D angle.
- double getInteriorAngle25D (int ith) const
  Get interior 2.5D angle.
- double getArea2D () const
  Get 2D Area.

- double getArea25D () const

  Get 2.5D Area.
- Triangle2 * getOppositeTriangle (const int ith) const

  Get the i-th neighbor triangle.
- int getIntraTriangleIndex (const Point2 *p) const

  Get the index of p in the triangle.
- int getIntraTriangleIndex (const Triangle2 *pTriangle) const

  Get the neighbor index of pTriangle.
- int getIntraTriangleIndex (const Point2 *p0, const Point2 *p1) const

  Get the index of (p0,p1)
- double getSquaredEdgeLength2D (int ith) const

  Method for internal use.
- double getSquaredEdgeLength25D (int ith) const

  Squared edge length.
- void setOppTriangle (const int ith, Triangle2 *pTriangle)

  Set the i-th neighbor triangle.
- void setProperties (Point2 *pI, Point2 *pJ, Point2 *pK)

  Set all corners.
- void clearProperties ()

  Clear all corners and neighbor pointers.
- void setPropertiesAndOppT (Point2 *pI, Point2 *pJ, Point2 *pK, Triangle2 *pNeig0, Triangle2 *pNeig1, Triangle2 *pNeig2)

  Set all corners and neighbor triangles.
- void setVertexPointer (const int ith, Point2 *pp)

  Set the i-th corner.
- bool hasVertex (Point2 *pVtx) const

  Has vertex.
- bool hasVertex (const Point2 &vtx) const

  Has vertex.
- bool hasOnEdge (int i, const Point2 &q) const

  Has point on edge.
- int getMaxIndex () const

  Get the index of the largest edge.
- int getMinIndex () const

  Get the index of the smallest edge.
- double getMaxSqEdgeLen2D () const

  Get the maximum squared 2D edge length.

Protected Member Functions

- double **computeArea** (double l0, double l1, double l2) const
- bool **isAccurateCC** (int maxIdx, const Point2 &cc) const
- bool **getCC_strategy1** (double avgOffX, double avgOffY, Point2 &cc) const
- void **getCC_strategy2** (int maxIdx, double avgOffX, double avgOffY, Point2 &cc) const
- void **getCommonOffset** (double &x, double &y) const

Protected Attributes

- Point2 * **aVertexPointer** [3]
- Triangle2 * **aOppTriangles** [3]

Friends

- std::ostream & **operator**<< (std::ostream &stream, const Triangle2 &c)
- void **registerTriangles** (Triangle2 *fromTriangle, int ith, Triangle2 *toTriangle, int jth)

6.23.1 Detailed Description

Triangle2 is a triangle in the Fade_2D triangulation. It holds three Point2 pointers to its corners. The corners are numbered in counterclockwise order. We refer to these indices as intra-triangle-indices.

Each triangle has three neighbors which can be accessed through intra-triangle-indices: The i-th neighbor triangle of a certain triangle T is the one which shares an edge with T such that this edge does not include the i-th corner of T.

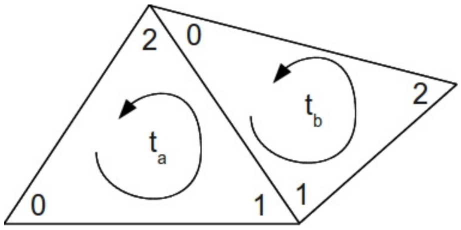


Figure 25 Indices and neighborships, *tb* is the 0-th neighbor of *ta* and *ta* is the 2nd neighbor of *tb*.

See also

TriangleAroundVertexIterator to find out how to access all triangles incident to a certain vertex.

6.23.2 Constructor & Destructor Documentation

6.23.2.1 Triangle2()

```
GEOM_FADE25D::Triangle2::Triangle2 ( )  [inline]
```

6.23.3 Member Function Documentation

6.23.3.1 getArea25D()

```
double GEOM_FADE25D::Triangle2::getArea25D ( ) const
```

Returns the 2.5D area of the triangle.

Note: The getArea() method is deprecated and replaced by getArea2D() and getArea25D()

6.23.3.2 getArea2D()

```
double GEOM_FADE25D::Triangle2::getArea2D ( ) const
```

Returns the 2D area of the triangle.

Note: The getArea() method is deprecated and replaced by getArea2D() and getArea25D()

6.23.3.3 getBarycenter()

```
Point2 GEOM_FADE25D::Triangle2::getBarycenter ( ) const
```

Returns

the barycenter of the triangle.

6.23.3.4 getCorner()

```
Point2 * GEOM_FADE25D::Triangle2::getCorner (
            const int ith ) const  [inline]
```

Returns

a pointer to the i-th corner point of the triangle.

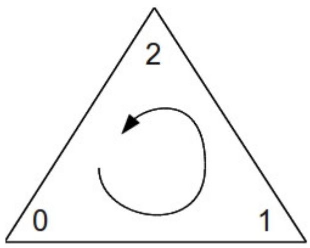


Figure 26 Intra triangle indices are ordered counterclockwise

Parameters

ith	is the intra-triangle-index, ith={0,1,2}.

6.23.3.5 getDual()

```
std::pair<Point2,bool> GEOM_FADE25D::Triangle2::getDual ( ) const
```

Returns

a std::pair<Point2,bool>, where the first component is the dual Voronoi vertex of the triangle and the second component is a boolean value which is true if the vertex is accurate. The z-coordinate of the returned point is always 0. Use Fade_2D::getHeight(..) to determine the height value.

Note

The true dual Voronoi vertex of an almost collinear Delaunay triangle can be outside the bounds of floating point arithmetic. In such cases this method returns a point with very large coordinates but still inside the range of double precision floating point arithmetic, and it will inform the user by setting the boolean return value to false.
Such cases can easily be avoided by insertion of four dummy vertices around the triangulation, e.g., at coordinates ten times larger than the domain of the data points. This will automatically restrict the Voronoi diagram of the data points to this range.

6.23.3.6 getInteriorAngle25D()

```
double GEOM_FADE25D::Triangle2::getInteriorAngle25D (
             int ith ) const
```

Returns

the interior 2.5D angle at the ith vertex

6.23.3.7 getInteriorAngle2D()

```
double GEOM_FADE25D::Triangle2::getInteriorAngle2D (
             int ith ) const
```

Note: The getInteriorAngle() method is deprecated and replaced by getInteriorAngle2D() and getInteriorAngle25D()

Returns

the interior 2D angle at the ith vertex

6.23.3.8 getIntraTriangleIndex() [1/3]

```
int GEOM_FADE25D::Triangle2::getIntraTriangleIndex (
             const Point2 * p ) const [inline]
```

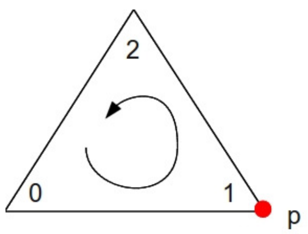


Figure 27 Intra triangle index of a vertex pointer

Parameters

p	is a pointer to a vertex in ∗*this*

Returns

the intra-triangle-index 0,1 or 2 of *p* in ∗*this*

6.23.3.9 getIntraTriangleIndex() [2/3]

```
int GEOM_FADE25D::Triangle2::getIntraTriangleIndex (
            const Triangle2 * pTriangle ) const  [inline]
```

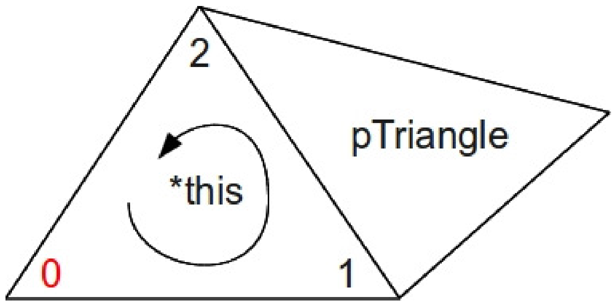


Figure 28 pTriangle is the 0-th neighbor of ∗this

Parameters

pTriangle	is a neighbor triangle of ∗this.

Returns

the intra-triangle-index of the vertex in ∗*this* which is opposite (i.e., does not touch the neighbor) *pTriangle*.

6.23.3.10 getIntraTriangleIndex() [3/3]

```
int GEOM_FADE25D::Triangle2::getIntraTriangleIndex (
            const Point2 * p0,
            const Point2 * p1 ) const  [inline]
```

Returns

the index of the edge (p0,p1) in the triangle

6.23.3.11 getNormalVector()

```
Vector2 GEOM_FADE25D::Triangle2::getNormalVector ( ) const
```

Returns

the normalized normal vector

6.23.3.12 getOppositeTriangle()

```
Triangle2 * GEOM_FADE25D::Triangle2::getOppositeTriangle (
            const int ith ) const [inline]
```

Returns the *i-th* neighbor triangle, i.e. the one opposite to the *i-th* corner.

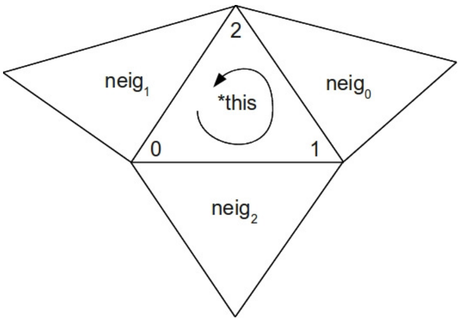


Figure 29 Neighbors of a triangle

Parameters

ith	is the intra-triangle-index of the opposite corner of ∗*this*

Returns

the i-th neighbor triangle, i.e. the one opposite to the i-th vertex or NULL if no neighbor triangle exists which is the case at the convex hull edges of the triangulation.

6.23.3.13 getSquaredEdgeLength25D()

```
double GEOM_FADE25D::Triangle2::getSquaredEdgeLength25D (
            int ith ) const
```

Returns the squared length of the *ith* edge.

6.23.3.14 getSquaredEdgeLength2D()

```
double GEOM_FADE25D::Triangle2::getSquaredEdgeLength2D (
            int ith ) const
```

Internal useSquared edge length

Returns the squared length of the *ith* edge. This method ignores the z-coordinate.

6.23.3.15 hasOnEdge()

```
bool GEOM_FADE25D::Triangle2::hasOnEdge (
            int i,
            const Point2 & q ) const
```

Returns

if q is exactly on the i-th edge

6.23.3.16 hasVertex() [1/2]

```
bool GEOM_FADE25D::Triangle2::hasVertex (
            Point2 * pVtx ) const
```

Returns

if pVtx is a corner of the triangle

6.23.3.17 hasVertex() [2/2]

```
bool GEOM_FADE25D::Triangle2::hasVertex (
            const Point2 & vtx ) const
```

Returns

if vtx is a corner of the triangle

6.23.3.18 setOppTriangle()

```
void GEOM_FADE25D::Triangle2::setOppTriangle (
            const int ith,
            Triangle2 * pTriangle ) [inline]
```

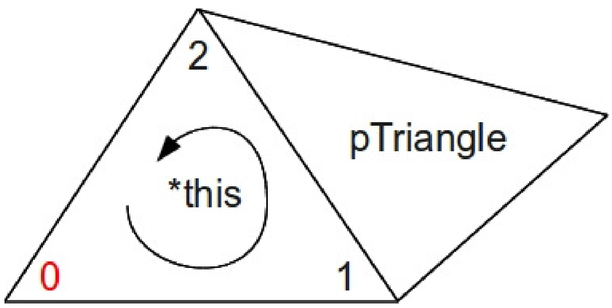


Figure 30 Make pTriangle the 0-th neighbor of ∗this

Parameters

ith	is the index of the corner of ∗*this* which does not touch *pTriangle*
pTriangle	is a pointer to the triangle which shares two corners with ∗*this*

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

- Triangle2.h

6.24 GEOM_FADE25D::TriangleAroundVertexIterator Class Reference

Iterator for all triangles around a given vertex.

```
#include <TriangleAroundVertexIterator.h>
```

Public Member Functions

- TriangleAroundVertexIterator (const Point2 ∗pPnt_)
  Constructor.
- TriangleAroundVertexIterator (Point2 ∗pPnt_, Triangle2 ∗pTr_)
  Constructor.
- TriangleAroundVertexIterator (const TriangleAroundVertexIterator &it)
  Copy constructor.
- TriangleAroundVertexIterator & **operator=** (const TriangleAroundVertexIterator &other)
- TriangleAroundVertexIterator & operator++ ()
  Proceed to the next triangle (the one in counterclockwise order)
- TriangleAroundVertexIterator & operator-- ()
  Proceed to the previous triangle (the one in clockwise order)
- bool operator== (const TriangleAroundVertexIterator &rhs)
  operator==()
- bool operator!= (const TriangleAroundVertexIterator &rhs)
  operator!=()
- Triangle2 ∗ operator∗ ()
  Returns a pointer to the current triangle (or NULL)
- Triangle2 ∗ previewNextTriangle ()
  Preview next triangle (CCW direction)
- Triangle2 ∗ previewPrevTriangle ()
  Preview previous triangle (CW direction)

Protected Member Functions

- void **loop** ()

Protected Attributes

- const Point2 ∗ **pPnt**
- Triangle2 ∗ **pTr**
- Triangle2 ∗ **pSavedTr**

6.24.1 Detailed Description

The TriangleAroundVertexIterator iterates over all triangles incident to a given vertex of a Fade_2D instance. The advantage is that the incident triangles can be visited in a certain order, namely counterclockwise with operator++() or clockwise using operator--(). If the order is not important you can use Fade_2D::getIncidentTriangles() instead.

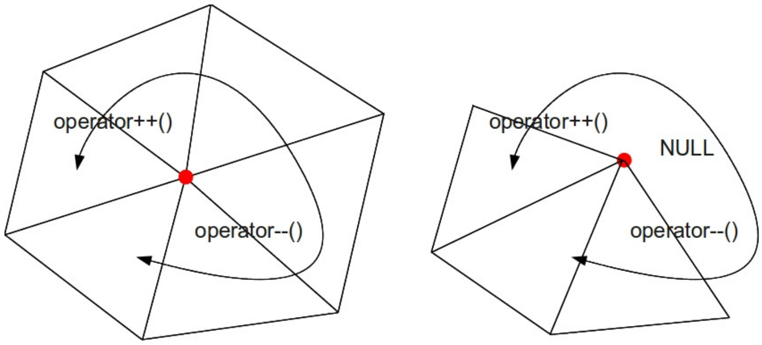


Figure 31 Left: the iterator visits the triangles around a vertex. Right: The iterator 'jumps' over the border edges of the triangulation

6.24.2 Constructor & Destructor Documentation

6.24.2.1 TriangleAroundVertexIterator() [1/3]

```
GEOM_FADE25D::TriangleAroundVertexIterator::TriangleAroundVertexIterator (
            const Point2 * pPnt_ )  [inline], [explicit]
```

Parameters

*pPnt_*	is the vertex whose incident triangles can be visited with the iterator

Initially the iterator points to an arbitrary triangle (not NULL)

6.24.2.2 TriangleAroundVertexIterator() [2/3]

```
GEOM_FADE25D::TriangleAroundVertexIterator::TriangleAroundVertexIterator (
            Point2 * pPnt_,
            Triangle2 * pTr_ )  [inline]
```

Parameters

*pPnt_*	is the vertex whose incident triangles can be visited with the iterator
*pTr_*	is the triangle the iterator initially points to

6.24.2.3 TriangleAroundVertexIterator() [3/3]

```
GEOM_FADE25D::TriangleAroundVertexIterator::TriangleAroundVertexIterator (
            const TriangleAroundVertexIterator & it )  [inline]
```

Copies the iterator `it`

6.24.3 Member Function Documentation

6.24.3.1 operator"!=()

```
bool GEOM_FADE25D::TriangleAroundVertexIterator::operator!= (
            const TriangleAroundVertexIterator & rhs )  [inline]
```

Compares the center and the current triangle of ∗this and `rhs`

Returns

true when they are different, false otherwise

6.24.3.2 operator∗()

```
Triangle2* GEOM_FADE25D::TriangleAroundVertexIterator::operator* ( )  [inline]
```

Dereferencing the iterator yields a pointer to the triangle to which the iterator points.

Warning

This method might yield NULL at the border of a triangulation.

6.24.3.3 operator++()

```
TriangleAroundVertexIterator& GEOM_FADE25D::TriangleAroundVertexIterator::operator++ ( )  [inline]
```

Moves the iterator to the next triangle in counterclockwise order.

Warning

At the border of a triangulation, two border edges exist which are incident to the center vertex. Consequently, the neighbor triangles are NULL there. If operator++() leads the iterator off the triangulation then the iterator will point to NULL. Another call to operator++() will set the iterator to the next triangle in counterclockwise order.

6.24.3.4 operator--()

`TriangleAroundVertexIterator& GEOM_FADE25D::TriangleAroundVertexIterator::operator-- ( ) [inline]`

Moves the iterator to the next triangle in clockwise order.

Warning

At the border of a triangulation, two border edges are incident to the center vertex. Consequently, the neighbor triangles are NULL there. If operator--() leads the iterator off the triangulation then the iterator will point to N↩ULL. Another call to operator--() will set the iterator to the next triangle in clockwise order.

6.24.3.5 operator==()

```
bool GEOM_FADE25D::TriangleAroundVertexIterator::operator== (
            const TriangleAroundVertexIterator & rhs ) [inline]
```

Compares the center and the current triangle of ∗this and `rhs`

Returns

true when they are identically or false otherwise

6.24.3.6 previewNextTriangle()

`Triangle2* GEOM_FADE25D::TriangleAroundVertexIterator::previewNextTriangle ( ) [inline]`

Returns

the next triangle (the one in CCW direction) without changing the current position.

Warning

This method might yield NULL at the border of a triangulation.

6.24.3.7 previewPrevTriangle()

`Triangle2* GEOM_FADE25D::TriangleAroundVertexIterator::previewPrevTriangle ( ) [inline]`

Returns

the previous triangle (the one in CW direction) without changing the current position.

Warning

This method might yield NULL at the border of a triangulation.

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

- TriangleAroundVertexIterator.h

6.25 GEOM_FADE25D::UserPredicateT Class Reference

User defined predicate.

`#include <UserPredicates.h>`

Public Member Functions

- virtual bool **operator()** (const Triangle2 ∗)=0

6.25.1 Detailed Description

See also

http://www.geom.at/remove-border-triangles/

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

- UserPredicates.h

6.26 GEOM_FADE25D::Vector2 Class Reference

Vector.

`#include <Vector2.h>`

Public Member Functions

- Vector2 (const double x_, const double y_, const double z_)
  Constructor.
- Vector2 ()
  Default constructor.
- Vector2 (const Vector2 &v_)
  Copy constructor.
- Vector2 orthogonalVector () const
  Get an orthogonal vector (CCW direction)
- bool isDegenerate () const
  isDegenerate
- double x () const
  Get the x-value.
- double y () const
  Get the y-value.
- double z () const
  Get the z-value.
- void set (const double x_, const double y_, const double z_)
  Set the values.
- double sqLength () const
  Get the squared length of the vector.

- int getMaxIndex () const
  Get max index.
- double length () const
  Get the length of the vector.
- double operator* (const Vector2 &other) const
  Scalar product.
- Vector2 operator* (double val) const
  Multiply by a scalar value.
- Vector2 operator/ (double val) const
  Divide by a scalar value.

Protected Attributes

- double **valX**
- double **valY**
- double **valZ**

6.26.1 Detailed Description

This class represents a vector in 2D

6.26.2 Constructor & Destructor Documentation

6.26.2.1 Vector2() [1/3]

```
GEOM_FADE25D::Vector2::Vector2 (
            const double x_,
            const double y_,
            const double z_ )
```

6.26.2.2 Vector2() [2/3]

```
GEOM_FADE25D::Vector2::Vector2 ( )
```

The vector is initialized to (0,0,0)

6.26.2.3 Vector2() [3/3]

```
GEOM_FADE25D::Vector2::Vector2 (
            const Vector2 & v_ )
```

Create a copy of vector v_

6.26.3 Member Function Documentation

6.26.3.1 getMaxIndex()

```
int GEOM_FADE25D::Vector2::getMaxIndex ( ) const
```

Returns

the index of the largest component (0,1 or 2)

6.26.3.2 isDegenerate()

```
bool GEOM_FADE25D::Vector2::isDegenerate ( ) const
```

Returns

true if the vector length is 0, false otherwise.

6.26.3.3 orthogonalVector()

```
Vector2 GEOM_FADE25D::Vector2::orthogonalVector ( ) const
```

Note

: Only (x,y) coordinates are computed, z=0

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

- Vector2.h

6.27 GEOM_FADE25D::Visualizer2 Class Reference

Visualizer2 is a general Postscript writer. It draws the objects Point2, Segment2, Triangle2, Circle2 and Label.

```
#include <Visualizer2.h>
```

Public Member Functions

- Visualizer2 (const std::string &filename_)

  Constructor.
- void addObject (const Segment2 &seg, const Color &c)

  Add a Segment2 object to the visualization.
- void addObject (const Edge2 &edge, const Color &c)

  Add an Edge2 object to the visualization.
- void addObject (const std::vector< Point2 > &vPoints, const Color &c)

  Add a vector of Point2 objects to the visualization.
- void addObject (const std::vector< Point2 *> &vPoints, const Color &c)

  Add a vector of Point2 to the visualization.*
- void addObject (const std::vector< Segment2 > &vSegments, const Color &c)

  Add a vector of Segment2 objects to the visualization.
- void addObject (const std::vector< ConstraintSegment2 *> &vConstraintSegments, const Color &c)

  Add a vector of ConstraintSegment2 objects to the visualization.
- void addObject (const std::vector< Edge2 > &vSegments, const Color &c)

  Add a vector of Edge2 objects to the visualization.
- void addObject (const std::vector< Triangle2 > &vT, const Color &c)

  Add a vector of Triangle2 objects to the visualization.
- void addObject (const Circle2 &circ, const Color &c)

  Add a Circle2 object to the visualization.
- void addObject (const Point2 &pnt, const Color &c)

  Add a Point2 object to the visualization.
- void addObject (const Triangle2 &tri, const Color &c)

  Add a Triangle2 object to the visualization.
- void addObject (const std::vector< Triangle2 *> &vT, const Color &c)

  Add a Triangle2 vector to the visualization.*
- void addObject (const Label &lab, const Color &c)

  Add a Label object to the visualization.
- void addHeaderLine (const std::string &s)

  Add a header line to the visualization.
- void writeFile ()

  Finish and write the postscript file.

Protected Member Functions

- void **writeHeaderLines** ()
- Point2 **scaledPoint** (const Point2 &p)
- double **scaledDouble** (const double &d)
- void **changeColor** (float r, float g, float b, float linewidth, bool bFill)
- void **changeColor** (const Color &c)
- void **writeHeader** (const std::string &title)
- void **writeFooter** ()
- void **writeLabel** (Label l)
- void **writeLine** (const Point2 &pSource, const Point2 &pTarget)
- void **writeTriangle** (const Point2 &p0_, const Point2 &p1_, const Point2 &p2_, bool bFill, double width)
- void **writeTriangle** (const Triangle2 *pT, bool bFill_, double width)
- void **writePoint** (Point2 &p1_, float size)
- void **writeCircle** (const Point2 &p1_, double radius, bool bFill)
- void **periodicStroke** ()
- void **setRange** ()

Protected Attributes

- std::ofstream **outFile**
- std::vector< std::pair< Segment2, Color > > **vSegments**
- std::vector< std::pair< Circle2, Color > > **vCircles**
- std::vector< std::pair< Point2, Color > > **vPoints**
- std::vector< std::pair< Triangle2, Color > > **vTriangles**
- std::vector< std::pair< Label, Color > > **vLabels**
- int **updateCtr**
- Bbox2 **bbox**
- bool **bFill**
- Color **lastColor**
- std::string **filename**
- std::vector< std::string > **vHeaderLines**
- bool **bFileClosed**

6.27.1 Detailed Description

See also

http://www.geom.at/example2-traversing/

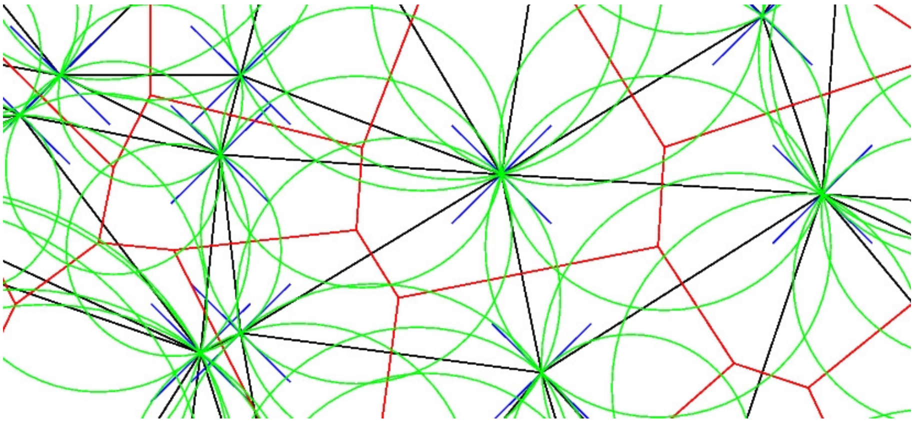


Figure 32 Example output of the Visualizer

6.27.2 Constructor & Destructor Documentation

6.27.2.1 Visualizer2()

```
GEOM_FADE25D::Visualizer2::Visualizer2 (
            const std::string & filename_ )  [explicit]
```

Parameters

*filename_*	is the name of the postscript file to be written

6.27.3 Member Function Documentation

6.27.3.1 writeFile()

```
void GEOM_FADE25D::Visualizer2::writeFile ( )
```

Note

This method *must* be called at the end when all the objects have been added.

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

- Visualizer2.h

6.28 GEOM_FADE25D::Zone2 Class Reference

Zone2 is an exactly defined area of a triangulation.

```
#include <Zone2.h>
```

Public Member Functions

- ZoneLocation getZoneLocation () const

  Get the zone location.
- Zone2 ∗ convertToBoundedZone ()

  Convert a zone to a bounded zone.
- void show (const std::string &postscriptFilename, bool bShowFull, bool bWithConstraints) const

  Postscript visualization.
- void show (Visualizer2 ∗pVisualizer, bool bShowFull, bool bWithConstraints) const

  Postscript visualization.
- void showGeomview (const std::string &filename, const std::string &color) const

  Geomview visualization.
- void showGeomview (Visualizer3 ∗pVis, const std::string &color) const

  Geomview visualization.
- void **analyzeAngles** ()
- void slopeValleyRidgeOptimization ()
- void optimizeValleysAndRidges (double tolerance2D, double lowerThreshold25D)
- void unifyGrid (double tolerance)
- void getTriangles (std::vector< Triangle2 ∗> &vTriangles_) const

  Get the triangles of the zone.
- void getVertices (std::vector< Point2 ∗> &vVertices_) const

  Get the vertices of the zone.
- void statistics (const std::string &s) const
- ConstraintGraph2 ∗ getConstraintGraph () const

  Get the associated constraint.
- size_t getNumberOfTriangles () const

  Get the number of triangles.

- void getConstraintGraphs (std::vector< ConstraintGraph2 *> &vConstraintGraphs_) const

  Get the associated constraint graphs.
- size_t numberOfConstraintGraphs () const

  Get a the number of ConstraintGraph2 objects.
- void debug (std::string name="")

  Development function.
- Bbox2 getBoundingBox () const

  Compute the bounding box.
- void getBoundaryEdges (std::vector< Edge2 > &vEdges) const

  Compute the boundary edges of the zone.
- void getBoundarySegments (std::vector< Segment2 > &vSegments) const

  Compute the boundary segments of the zone.
- double getArea2D () const

  Get 2D Area.
- double getArea25D () const

  Get 2.5D Area.
- void getBorderEdges (std::vector< Edge2 > &vBorderEdgesOut) const

  Get border edges.

Protected Attributes

- Dt2 * **pDt**
- ZoneLocation **zoneLoc**

Friends

- Zone2 * zoneUnion (Zone2 *pZone0, Zone2 *pZone1)

  Compute the union of two zones.
- Zone2 * zoneIntersection (Zone2 *pZone0, Zone2 *pZone1)

  Compute the intersection of two zones.
- Zone2 * zoneDifference (Zone2 *pZone0, Zone2 *pZone1)

  Compute the difference of two zones.
- Zone2 * zoneSymmetricDifference (Zone2 *pZone0, Zone2 *pZone1)

  Compute the symmetric difference of two zones.
- Zone2 * **peelOffIf** (Zone2 *pZone, UserPredicateT *pPredicate, bool bVerbose)

6.28.1 Detailed Description

See also

`http://www.geom.at/example4-zones-defined-areas-in-triangulations/`
`http://www.geom.at/boolean-operations/`
createZone in the Fade2D class

6.28.2 Member Function Documentation

6.28.2.1 convertToBoundedZone()

```
Zone2* GEOM_FADE25D::Zone2::convertToBoundedZone ( )
```

The mesh generation algorithms refine() and refineAdvanced() require a zone object that is bounded by constraint segments. This is always the case for zones with zoneLocation ZL_INSIDE but other types of zones may be unbounded. For convenience this method is provided to create a bounded zone from a possibly unbounded one.

Returns

a pointer to a new Zone2 object with zoneLocation ZL_RESULT_BOUNDED or *this* if this->getZoneLocation() is ZL_INSIDE.

6.28.2.2 getArea25D()

```
double GEOM_FADE25D::Zone2::getArea25D ( ) const
```

Returns the 2.5D area of the zone.

Note: The getArea() method is deprecated and replaced by getArea2D() and getArea25D()

6.28.2.3 getArea2D()

```
double GEOM_FADE25D::Zone2::getArea2D ( ) const
```

Returns the 2D area of the zone.

Note: The getArea() method is deprecated and replaced by getArea2D() and getArea25D()

6.28.2.4 getBorderEdges()

```
void GEOM_FADE25D::Zone2::getBorderEdges (
            std::vector< Edge2 > & vBorderEdgesOut ) const
```

Returns

: the CCW oriented border edges of the zone

6.28.2.5 getConstraintGraph()

```
ConstraintGraph2* GEOM_FADE25D::Zone2::getConstraintGraph ( ) const
```

Returns

a pointer to the ConstraintGraph2 object which defines the zone.
or NULL for ZL_RESULT-, ZL_GROW and ZL_GLOBAL_-zones.

6.28.2.6 getConstraintGraphs()

```
void GEOM_FADE25D::Zone2::getConstraintGraphs (
            std::vector< ConstraintGraph2 *> & vConstraintGraphs_ ) const
```

6.28.2.7 getNumberOfTriangles()

```
size_t GEOM_FADE25D::Zone2::getNumberOfTriangles ( ) const
```

Warning

This method is fast but O(n), so don't call it frequently in a loop.

6.28.2.8 getTriangles()

```
void GEOM_FADE25D::Zone2::getTriangles (
            std::vector< Triangle2 *> & vTriangles_ ) const
```

This command fetches the existing triangles of the zone.

Note

Fade_2D::void applyConstraintsAndZones() must be called after the last insertion of points and constraints. that the lifetime of data from the Fade2D datastructures does exceed the lifetime of the Fade2D object.

6.28.2.9 getVertices()

```
void GEOM_FADE25D::Zone2::getVertices (
            std::vector< Point2 *> & vVertices_ ) const
```

6.28.2.10 getZoneLocation()

```
ZoneLocation GEOM_FADE25D::Zone2::getZoneLocation ( ) const
```

Returns

ZL_INSIDE if the zone applies to the triangles inside one or more ConstraintGraph2 objects
ZL_OUTSIDE if the zone applies to the outside triangles
ZL_GLOBAL if the zone applies (dynamically) to all triangles
ZL_RESULT if the zone is the result of a set operation
ZL_GROW if the zone is specified by a set of constraint graphs and an inner point

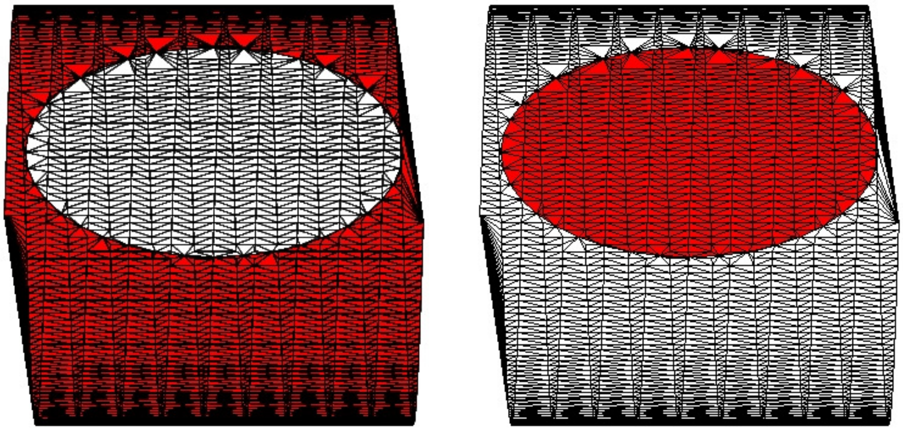


Figure 33 An ouside zone and in inside zone

6.28.2.11 numberOfConstraintGraphs()

```
size_t GEOM_FADE25D::Zone2::numberOfConstraintGraphs ( ) const
```

A Zone2 object might be defined by zero, one or more ConstraintGraph2 objects.

6.28.2.12 optimizeValleysAndRidges()

```
void GEOM_FADE25D::Zone2::optimizeValleysAndRidges (
            double tolerance2D,
            double lowerThreshold25D )
```

Optimize Valleys and Ridges

A Delaunay triangulation not unique when when 2 or more triangles share a common circumcircle. As a consequence the four corners of a rectangle can be triangulated in two different ways: Either the diagonal proceeds from the lower left to the upper right corner or it connects the other two corners. Both solutions are valid and an arbitrary one is applied when points are triangulated. To improve the repeatability and for reasons of visual appearance this method unifies such diagonals such that they point from the lower left to the upper right corner (or in horizontal direction).
Moreover a Delaunay triangulation does not take the z-value into account and thus valleys and ridges may be disturbed. The present method flips diagonals such that they point from the lower left to the upper right corner of a quad. And if the 2.5D lengths of the diagonals are significantly different, then the shorter one is applied.

Parameters

tolerance2D	is 0 when only exact cases of more than 3 points on a common circumcircle shall be changed. But in practice input data can be disturbed by noise and tiny rounding errors such that grid points are not exactly on a grid. The numeric error is computed as $error = \frac{abs(diagonalA - diagonalB)}{max(diagonalA, diagonalB)}$. and `tolerance2D` is an upper threshold to allow modification despite such tiny inaccuracies.
lowerThreshold25D	is used to take also the heights of the involved points into account. For example, the points Point_2 a(0,0,0); Point_2 b(10,0,0); Point_2 c(10,10,0); Point_2 d(0,10,1000); can form the triangles (a,b,c) and (a,c,d) or the triangles (a,b,d) and (d,b,c) but (a,c) is obviousy the better diagonal because the points a,b,c share the same elevation while d is at z=1000. Technically spoken, the diagonal with the smaller 2.5D-length is applied if the both, the 2D error is below `tolerance2D` and the 2.5D error is above `lowerThreshold25D`. The 2.5D criterion has priority over the 2D criterion.

6.28.2.13 show() [1/2]

```
void GEOM_FADE25D::Zone2::show (
            const std::string & postscriptFilename,
            bool bShowFull,
            bool bWithConstraints ) const
```

Parameters

postscriptFilename	is the name of the output file.
bShowFull	specifies if only the zone or the full triangulation shall be drawn
bWithConstraints	specifies if constraint edges shall be drawn

6.28.2.14 show() [2/2]

```
void GEOM_FADE25D::Zone2::show (
            Visualizer2 * pVisualizer,
            bool bShowFull,
            bool bWithConstraints ) const
```

Parameters

pVisualizer	is a pointer to an existing Visualizer2 object.

Note

You must call pVisualizer->writeFile() before program end

Parameters

bShowFull	specifies if only the zone or the full triangulation shall be drawn
bWithConstraints	specifies if constraint edges shall be drawn

6.28.2.15 showGeomview() [1/2]

```
void GEOM_FADE25D::Zone2::showGeomview (
            const std::string & filename,
            const std::string & color ) const
```

Parameters

filename	is the name of the output file.
color	is a string("red green blue alpha"), e.g., "1.0 0.0 0.0 1.0"*

6.28.2.16 showGeomview() [2/2]

```
void GEOM_FADE25D::Zone2::showGeomview (
            Visualizer3 * pVis,
            const std::string & color ) const
```

Parameters

pVis	points to a Visualizer3 object
color	is a string("red green blue alpha"), e.g., "1.0 0.0 0.0 1.0"*

6.28.2.17 slopeValleyRidgeOptimization()

```
void GEOM_FADE25D::Zone2::slopeValleyRidgeOptimization ( )
```

Optimize Slopes, Valleys and Ridges

6.28.2.18 statistics()

```
void GEOM_FADE25D::Zone2::statistics (
            const std::string & s ) const
```

Statistics

Prints statistics to stdout.

6.28.2.19 unifyGrid()

```
void GEOM_FADE25D::Zone2::unifyGrid (
            double tolerance )
```

Unify Grid

A Delaunay triangulation not unique when when 2 or more triangles share a common circumcircle. As a consequence the four corners of a rectangle can be triangulated in two different ways: Either the diagonal proceeds from the lower left to the upper right corner or it connects the other two corners. Both solutions are valid and an arbitrary one is applied when points are triangulated. To improve the repeatability and for reasons of visual appearance this method unifies such diagonals to point from the lower left to the upper right corner (or in horizontal direction).

Parameters

tolerance	is 0 when only exact cases of more than 3 points on a common circumcircle shall be changed. But in practice input data can be disturbed by noise and tiny rounding errors such that grid points are not exactly on a grid. The numeric error is computed as $error = \frac{abs(diagonalA - diagonalB)}{max(diagonalA, diagonalB)}$. and `tolerance` is an upper threshold to allow modification despite such tiny inaccuracies. Use with caution, such flips break the empty circle property and this may or may not fit your setting.

6.28.3 Friends And Related Function Documentation

6.28.3.1 zoneDifference

```
Zone2* zoneDifference (
            Zone2 * pZone0,
            Zone2 * pZone1 )   [friend]
```

Returns

a new zone containing the triangles of $*$pZone0 minus the ones of $*$pZone1

Note

`pZone0` and `pZone1` must belong to the same Fade_2D object.

6.28.3.2 zoneIntersection

```
Zone2* zoneIntersection (
            Zone2 * pZone0,
            Zone2 * pZone1 )   [friend]
```

Returns

a new zone containing the intersection of $*$pZone0 and $*$pZone1

Note

`pZone0` and `pZone1` must belong to the same Fade_2D object.

6.28.3.3 zoneSymmetricDifference

```
Zone2* zoneSymmetricDifference (
            Zone2 * pZone0,
            Zone2 * pZone1 )  [friend]
```

Returns

a new zone containing the triangles that are present in one of the zones but not in the other one.

Note

`pZone0` and `pZone1` must belong to the same Fade_2D object.

6.28.3.4 zoneUnion

```
Zone2* zoneUnion (
            Zone2 * pZone0,
            Zone2 * pZone1 )  [friend]
```

Returns

a new zone containing the union of the triangles of *pZone0 and *pZone1

Note

`pZone0` and `pZone1` must belong to the same Fade_2D object.

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

- Zone2.h

7 File Documentation

7.1 CAF_Component.h File Reference

```
#include <map>
#include "common.h"
#include "Segment2.h"
```

Classes

- class GEOM_FADE25D::CAF_Component
  CAF_Component stands for CUT AND FILL COMPONENT. It represents a connected area of the surface.

Enumerations

- enum GEOM_FADE25D::CAFTYP { GEOM_FADE25D::CT_NULL, GEOM_FADE25D::CT_CUT, GEOM_FADE25D::CT_FILL }

Functions

- std::ostream & GEOM_FADE25D::operator<< (std::ostream &stream, const CAF_Component &c)

7.1.1 Enumeration Type Documentation

7.1.1.1 CAFTYP

```
enum GEOM_FADE25D::CAFTYP
```

enumerates the three possible Cut-And-Fill types

Enumerator

CT_NULL	the first surface corresponds to the second one
CT_CUT	the first surface is above the second one
CT_FILL	the first surface is below the second one

7.1.2 Function Documentation

7.1.2.1 operator<<()

```
std::ostream& GEOM_FADE25D::operator<< (
            std::ostream & stream,
            const CAF_Component & c ) [inline]
```

Report

7.2 Color.h File Reference

```
#include "common.h"
```

Classes

- class GEOM_FADE25D::Color
  Color.

Enumerations

- enum GEOM_FADE25D::Colorname {
  CRED, **CGREEN**, **CBLUE**, **CBLACK**,
  CPINK, **CGRAY**, **CORANGE**, **CLIGHTBLUE**,
  CLIGHTBROWN, **CDARKBROWN**, **CPURPLE**, **COLIVE**,
  CLAWNGREEN, **CPALEGREEN**, **CCYAN**, **CYELLOW**,
  CWHITE }

  Predefined colors for convenience.

Functions

- std::ostream & **GEOM_FADE25D::operator**<< (std::ostream &stream, const Color &c)

7.3 ConstraintSegment2.h File Reference

```
#include <set>
#include "common.h"
```

Classes

- class GEOM_FADE25D::ConstraintSegment2

  A ConstraintSegment2 represents a Constraint Edge.

Enumerations

- enum GEOM_FADE25D::ConstraintInsertionStrategy {
  CIS_CONFORMING_DELAUNAY =0, GEOM_FADE25D::CIS_CONSTRAINED_DELAUNAY =1, GEOM_FADE25D::CIS_CONFORMING_DELAUNAY_SEGMENT_LEVEL =2, GEOM_FADE25D::CIS_KEEP_DELAUNAY =0,
  GEOM_FADE25D::CIS_IGNORE_DELAUNAY =1 }

  Constraint Insertion Strategy determines how a constraint edge shall be inserted:

7.3.1 Enumeration Type Documentation

7.3.1.1 ConstraintInsertionStrategy

`enum GEOM_FADE25D::ConstraintInsertionStrategy`

- CIS_CONSTRAINED_DELAUNAY inserts a segment without subdivision unless required (which is the case if existing vertices or constraint segments are crossed).

All other constraint insertion strategies are deprecated and only kept for backwards compatibility. Their behavior can be achieved using ConstraintGraph2::makeDelaunay() and/or Fade_2D::drape(). See also `examples_25D/terrain.cpp`.

Note

In former library versions the terms CIS_IGNORE_DELAUNAY and CIS_KEEP_DELAUNAY were used but these were misleading and are now deprecated. For backwards compatibility they are kept.

Enumerator

CIS_CONSTRAINED_DELAUNAY	Deprecated.
CIS_CONFORMING_DELAUNAY_SEGMENT_LEVEL	Deprecated.
CIS_KEEP_DELAUNAY	Deprecated name.
CIS_IGNORE_DELAUNAY	Deprecated.

7.4 EfficientModel.h File Reference

```
#include "common.h"
#include "Point2.h"
```

Classes

- class GEOM_FADE25D::EfficientModel

  EfficientModel.

Enumerations

- enum GEOM_FADE25D::SmoothingStrategy { GEOM_FADE25D::SMS_MINIMUM, GEOM_FADE25D::SMS_MAXIMUM, GEOM_FADE25D::SMS_MEDIAN, GEOM_FADE25D::SMS_AVERAGE }

  SmoothingStrategy for EfficientModel::zSmoothing()

7.4.1 Enumeration Type Documentation

7.4.1.1 SmoothingStrategy

`enum GEOM_FADE25D::SmoothingStrategy`

Enumerator

SMS_MINIMUM	Assign the minimum height.
SMS_MAXIMUM	Assign the maximum height.
SMS_MEDIAN	Assign the median height.
SMS_AVERAGE	Assign the average height.

7.5 SegmentChecker.h File Reference

```
#include <map>
#include "common.h"
#include "Segment2.h"
```

```
#include "MsgBase.h"
```

Classes

- class GEOM_FADE25D::SegmentChecker

  SegmentChecker identifies intersecting line segments.

Enumerations

- enum SegmentIntersectionType {
  SIT_UNINITIALIZED, SIT_NONE, SIT_SEGMENT, SIT_POINT,
  SIT_ENDPOINT }

7.5.1 Enumeration Type Documentation

7.5.1.1 SegmentIntersectionType

`enum SegmentIntersectionType`

The Segment intersection type enumerates the way two line segments intersect each other

Enumerator

SIT_UNINITIALIZED	Invalid value
SIT_NONE	No intersection
SIT_SEGMENT	The intersection is a non-degenerate segment (collinear intersection)
SIT_POINT	The intersection is a single point differnt from the endpoints
SIT_ENDPOINT	The two segments share a common endpoint which is the only intersection

7.6 TriangleAroundVertexIterator.h File Reference

```
#include "common.h"
#include "Point2.h"
#include "Triangle2.h"
```

Classes

- class GEOM_FADE25D::TriangleAroundVertexIterator

  Iterator for all triangles around a given vertex.

Functions

- int **GEOM_FADE25D::inc1** (int num)
- int **GEOM_FADE25D::inc2** (int num)

Index