# 1 Main Page

1.1 C++ Constrained Delaunay Triangulation Fade2D ........................................... 1  
  1.1.1 Using the C++ Constrained Delaunay triangulation library: .......................... 2  
  1.1.2 Compiling and Linking the Library under Windows: .................................. 2  
  1.1.3 Compiling and Linking the Library under Linux and Mac: ........................... 2  
  1.1.4 Directory Contents ......................................................................................... 3  
  1.1.5 Troubleshooting ............................................................................................. 3  
  1.1.6 Release notes / History .................................................................................. 3  

# 2 Module Index

  2.1 Modules ............................................................................................................. 12  

# 3 Class Index

  3.1 Class List .......................................................................................................... 12  

# 4 File Index

  4.1 File List ............................................................................................................. 13  

# 5 Module Documentation

  5.1 Tools .................................................................................................................. 14  
    5.1.1 Detailed Description ...................................................................................... 15  
    5.1.2 Function Documentation .............................................................................. 15  
  5.2 Version Information ........................................................................................... 19  
    5.2.1 Detailed Description ...................................................................................... 19  
  5.3 File I/O ............................................................................................................... 20  
    5.3.1 Detailed Description ...................................................................................... 20  
    5.3.2 Function Documentation .............................................................................. 20  
  5.4 Test Data Generators ......................................................................................... 23  
    5.4.1 Detailed Description ...................................................................................... 23  
    5.4.2 Generate random polygons and other test objects ..................................... 23  
    5.4.3 Function Documentation .............................................................................. 23  

# 6 Class Documentation

  6.1 GEOM_FADE2D::Bbox2 Class Reference ............................................................. 29  
    6.1.1 Detailed Description ...................................................................................... 30  
    6.1.2 Constructor & Destructor Documentation .................................................. 30  
    6.1.3 Member Function Documentation ............................................................... 30  
    6.1.4 Friends And Related Function Documentation .......................................... 34  
  6.2 GEOM_FADE2D::Circle2 Class Reference ......................................................... 34  
    6.2.1 Detailed Description ...................................................................................... 34  
    6.2.2 Constructor & Destructor Documentation .................................................. 34  
    6.2.3 Member Function Documentation ............................................................... 35  
  6.3 GEOM_FADE2D::Color Class Reference ........................................................... 35  
    6.3.1 Detailed Description ...................................................................................... 36  

---

Generated by Doxygen
1 Main Page

1.1 C++ Constrained Delaunay Triangulation Fade2D

- Fast C++ Delaunay triangulation library, see the benchmark.
- C++ examples for 2D Delaunay triangulations and 2.5D triangulations.
- Free Student license. Commercial licenses and support are available.
- Support for Windows (Visual Studio), MacOS (Clang), Linux (GCC) on PC and Raspberry PI
1.1.1 Using the C++ Constrained Delaunay triangulation library:

Download. Unzip. Start to play with the included C++ example source codes: The first example is described here.

Fade comes as two separate libraries:

- Fade2D is a 2D Constrained Delaunay triangulation library with
  - Polygon support
  - Constraint edges
  - Grid Mesher and Delaunay Mesh Generator
  - Segment Intersection Test Software

- Fade2.5D is a C++ Constrained Delaunay triangulation library for 2.5D and it can do anything that the Fade2D library can do. But it has an additional z-coordinate and a rich selection of additional algorithms made for Digital Elevation Models (DEM) and surfaces like
  - Cut-and-Fill
  - Cookie Cutter
  - Valley-/Ridge-triangulations
  - Mesh smoothing
  - Point cloud simplification.

A collection of 2D and 2.5D example source codes (*.cpp files) is contained in the download. The C++ examples go step by step over the concepts of the library. New Fade2.5D users are advised to check also the 2D examples because the basics are described there and these apply also to 2.5D.

1.1.2 Compiling and Linking the Library under Windows:

2. Compile the example source code. The executable is written to the Win32 or x64 folder.

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

<table>
<thead>
<tr>
<th>Visual Studio</th>
<th>IDE</th>
<th>Platform Toolset</th>
</tr>
</thead>
<tbody>
<tr>
<td>VS2010</td>
<td>v10</td>
<td>toolset v100 or Windows7.1SDK</td>
</tr>
<tr>
<td>VS2012</td>
<td>v11</td>
<td>toolset v110</td>
</tr>
<tr>
<td>VS2013</td>
<td>v12</td>
<td>toolset v120</td>
</tr>
<tr>
<td>VS2015</td>
<td>v14</td>
<td>toolset v140</td>
</tr>
<tr>
<td>VS2017</td>
<td>v15</td>
<td>toolset v141</td>
</tr>
<tr>
<td>VS2019</td>
<td>v16</td>
<td>toolset v142</td>
</tr>
</tbody>
</table>

1.1.3 Compiling and Linking the Library under Linux and Mac:

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 library (*.so) for Linux/Apple developers.

- **examples_2D**
  2D Example source code (*.cpp files) and Visual Studio projects

- **examples_25D**
  2.5D Example source code (*.cpp files) and Visual Studio projects

- **doc**
  Library Documentation in *.pdf format

### 1.1.5 Troubleshooting

- Check if the examples work on your computer. Then compare their settings with your project settings.

- When updating from an earlier version: UPDATE ALSO THE HEADER FILES.

- Mixing multiple Visual Studio versions won’t work. Use the right dll.

- 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 1.88, June 4th, 2021:

- Voronoi diagram. Accuracy of Triangle::getDual() improved with multiple-precision arithmetic. Version 1.87, May 5th., 2021:

  - Additional versions of the load() and save() commands to also accept std::ostream and std::istream.

  - There is a new version of the peelOffIf() function that removes unwanted border-triangles. It can prevent a zone from breaking apart by deleting triangles. It takes the new predicate “PeelPredicateTS”, which allows more precise decisions. The version of peelOffIf() that takes the old “UserPredicateT” remains valid for backwards compatibility.
• When Fade_2D::createConstraint() inserts a constraint segment that intersects an existing one or an existing point, then it needs to be subdivided. By default, the intersection point is then assigned the height of the existing element. But now this function has an additional parameter 'bool bUseHeightOfLatest=false' which can be used to enforce the height of the last inserted segment.

Version 1.86, April 28th., 2021:

• New commands Fade_2D::saveTriangulation(), Fade_2D::saveZones(), Zone2::save() and Fade_2D::load() to save and load triangulation data. The new example ex11_save_and_load.cpp demonstrates it.

• New dry-mode parameter for CloudPrepare::uniformSimplifyGrid() and for CloudPrepare::adaptiveSimplify() so that the size of the point cloud that would result from the reduction can be determined.

Version 1.85, March 8th., 2021:

• Bugfixes: A multithreading-bug has been solved and strings are now correctly passed to the Visualizer2 class.

• New method Fade_2D::setFastMode(true) to avoid expensive computations. This accelerates triangulation of raster data i.e., points on a regular grid.

v1.84, Jan. 7th., 2021:

• IMPORTANT IF YOU UPGRADE FROM A PREVIOUS VERSION: To avoid passing std::strings over the DLL boundary, some function parameters have been changed from std::string to const char *. You will often not even notice this, but if your code should not compile anymore, then this is the reason. Instead of passing "yourString" please pass it as "yourString.c_str()". This change was unavoidable. Thank you for your understanding!

• New CloudPrepare class to simplify point clouds and also to avoid memory-usage-peaks. Have a look at the examples!

• New function Fade_2D::exportTriangulation() allows convenient transfer of triangulation data to your own data structures. The function was created with memory consumption in mind, i.e. while the data is exported, it frees memory from the library gradually.

• New function Zone2::smoothing() applies weighted Laplacian smoothing to the vertices of a zone.

• New Valley/Ridge optimization: With Zone2::slopeValleyRidgeOptimization() one can choose between 3 algorithms now to adapt the triangulation better to valleys and ridges. Have a look at the new examples.

• Example codes completely rewritten.

v1.83, Dec. 30th, 2020:

• Internal test release. Significant changes, thus it has not been released.

v1.82, Nov. 15th, 2020:

• Intermediate release to support CentOS/RedHat7.8. Minor improvements here and there.

v1.81, 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.

v1.80, 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\textasciitilde 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


v1.78, November 15th, 2019:

• Bugfix: Multithreading did not work in Windows due to a CMake configuration error.

• A typo in the function name Fade_2D::measureTriangulationTime() has been corrected.

v1.77, October 21st, 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\_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.

v1.75 and 1.76

• Non-public tests.

v1.74, 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 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 the library 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.

v1.73, January 14th, 2019:

• While all below mentioned releases after v1.63 were development versions the present v1.73 is again an official release for all.

• The work of the below betas is included

• as well as a bugfix in the getProfile() method of the IsoContours class (this method was new and experimental in v1.63)

v1.71 and 1.72, October 24th, 2018:

• (internal) Hole-Filling (Polygon-Triangulation) improved.
v1.70, October 17th, 2018:
• (internal) Hole-Filling (Polygon-Triangulation) improved.

v1.69, October 15th, 2018:
• (internal) Hole-Filling (Polygon-Triangulation) improved.

v1.68, September 14th, 2018:
• (internal) Hole-Filling (Polygon-Triangulation) improved.

v1.67, September 4th, 2018:
• (internal) Hole-Filling (Polygon-Triangulation) is now offered via. an API call. Intermediate beta release.

v1.66, August 25th, 2018:
• (internal) Bugfix in Cut&Fill: An intersection point could be slightly off its expected range. Solved. Unofficial intermediate code.

v1.65, July 29th, 2018:
• (internal) Another bugfix in Cut&Fill. Unofficial intermediate binary.

v1.64, 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.

v1.63, 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.

v1.62, June 3rd, 2018:
• 3D Point Cloud Reconstruction considerably improved. Unofficial demo.

v1.61, May 1st, 2018:
• 3D Point Cloud Reconstruction: Unofficial demo.

v1.60, February 26th, 2018:
• Accurate computation of glancing segment intersections.
• Additional parameter for Advanced Meshing: bool bKeepExistingSteinerPoints=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.

v1.59, January 14th, 2018:
• Performance upgrade: Multithreading is available now. Large point sets reach a speedup of 4.4 on a hexacore CPU (i7 6800K)
v1.58, October 23th, 2017:

- Mesh Generator refactored. Delaunay Meshing is +10x faster now.
- A function to create polygons from boundary edges has been added.

v1.57, October 9th, 2017:

- Non-public test code.

v1.56, September 24th, 2017:

- Bugfix: createConstraint() crashed in a rare case. Solved.
- Functions for binary file I/O added.

v1.55, August 12th, 2017:

- Access to internal Cut&Fill datastructures revised.

v1.54beta, August 8th, 2017:

- Access to internal Cut&Fill datastructures. This is a pre-released beta, code quality is good but final tests and documentation updates required.

v1.53, July 15th, 2017:

- Error corrections and performance upgrades in the still quite new Cut&Fill library module.

v1.53 beta, June 2nd, 2017:

- The new Cut&Fill library module has been added. Cut&Fill computes the volume between two surfaces.

v1.51 beta, May 27th, 2017:

- Non-public test binary

v1.50, April 5th, 2017: After three internal betas (that concetrated on refactoring and rare bugs) this is again a stable public release:

- The constraint insertion subsystem has been rewritten 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 releases: 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.

v1.49, March 2nd, 2017:

- Constraint insertion subsystem improved.
- Mesh generator revised.

v1.48, February 15th, 2017:

- Corrections of yesterday's v1.47.

v1.47, 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.

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

v1.46, 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.

v1.43, 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.

v1.42, 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.

v1.41, July 24th, 2016:
• New constraint insertion strategy.
• Minor bug fixes.
• Performance slightly improved.

v1.40 beta, June 14th, 2016:
• Non public intermediate test code.
• 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=ZL_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.
v1.39, May 31st, 2016:
  • Non public intermediate beta.

v1.37a, March 15th, 2016:
  • Small upgrade: The performance of the remove method has been improved.

v1.37, 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).

v1.36, February 29th, 2016:
  • Experimental method refineExtended(..) replaced by the (now permanent) method refineAdvanced(MeshGenParams* pParams). This method allows much more control over the mesh density.

v1.34, February 14th, 2016:
  • Vertex management subsystem revised (sometimes Vertex removement did not work as expected). Performance improvement.

v1.33 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 v1.33.14:
  • 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 one 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 release it is recommended that you remove any previous Fade DLL’s to avoid unintended linking to an old binary.

v1.31 and 1.32, December 1st, 2015:
  • Non public intermediate release, improves the CDT.

v1.30, November 18th, 2015:
  • Non public intermediate release, improves the refineExtended method.

v1.29, 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.

v1.28, 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.

v1.27, October 5th, 2015:
  • Non public release, improvements of the recently implemented functions, especially of customer specific code Fade_2D::importTriangles() and Fade2D::cutTriangles().
v1.26, September 8th, 2015:
- New functions of the last unofficial v1.25 have been revised. Constraint segments may intersect now.

v1.25, August 18th, 2015:
- Intermediate pre-release with new features: importTriangles() imports arbitrary triangles into a triangulation, cutTriangles() allows to insert a constraint segment as if it were 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.

v1.24, July 22nd, 2015:
- Public release of v1.23’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.

v1.23, July 9th, 2015:
- Internal test release with the new refineExtended() method for the specific needs of a certain client software.

v1.22, 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.
- Removal 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.

v1.21, May 17th, 2015:
- Unofficial intermediate release. Testing new features.

v1.20, April 5th, 2015:
- 3D scene Visualization for (up to date) web browsers added. Misleading enumeration values CIS_KEEP_DELAUNAY and CIS_IGNORE_DELAUNAY have been replaced by CIS_CONFORMING_DELAUNAY and CIS_CONSTRAINED_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 has been reengineered and is included in the present documentation pages.

v1.19, October 26th, 2014:
- Support for Visual Studio 2013 (VC12) has been added.
- Only minor code changes.

v1.18.3, June 9th, 2014:
- Delaunay Mesh Generation has been improved: Better quality, better performance.
- API improved.
- Small bug fixes.

v1.16.1, February 10th, 2014:
• Small update: In rare cases it was possible that subdivided ConstraintSegm ents caused problems in combi-
• Small update: In rare cases it was possible that subdivided ConstraintSegments caused problems in combi-
nation with zone growing. This is fixed now.

v1.16, February 3rd, 2014:
• Constrained Delaunay triangulation improved,
• Delaunay meshing improved,
• aspect ratio meshing (experimental) added.
• Minor bug fixes.
• Added support for Visual Studio 2012.

v1.14, November 2013 and v1.15, December 2013:
• Non-public intermediate releases (betas with experimental features).

v1.13, 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.

v1.12, June 30th, 2013:
• Starting with 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 edition: 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.

v1.11, June 14th, 2013:
• Non-public intermediate release with VS2008 support and a first version of the iso-contour feature.

v1.10, 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::
D::enableMultithreading().
• There is a new insert method in the API which uses arrays.

v1.03, Nov. 4th, 2012:
• A critical bug has been fixed, please switch to v1.03.
• 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 experi-
mental feature in the current v1.03.

v1.02, 9/2012:
• An additional debug library for Windows has been added and
• the directory structure has been reorganized.

v1.01, 9/2012:
• This is a stable public release. Since v0.9 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:

- Tools 14
- Version Information 19
- File I/O 20
- Test Data Generators 23

3 Class Index

3.1 Class List

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

- **GEOM_FADE2D::Bbox2**
  - Bbox2 is an axis aligned 2D bounding box 29

- **GEOM_FADE2D::Circle2**
  - Circle for visualization 34

- **GEOM_FADE2D::Color**
  - Color for visualization 35

- **GEOM_FADE2D::ConstraintGraph2**
  - ConstraintGraph2 is a set of Constraint Edges (ConstraintSegment2) 37

- **GEOM_FADE2D::ConstraintSegment2**
  - A ConstraintSegment2 represents a Constraint Edge 40

- **GEOM_FADE2D::Edge2**
  - Edge2 is a directed edge 42

- **GEOM_FADE2D::Fade_2D**
  - Fade_2D is the Delaunay triangulation main class 45

- **GEOM_FADE2D::FadeExport**
  - FadeExport is a simple struct to export triangulation data 66

- **GEOM_FADE2D::Func_gtEdge2D**
  - Functor to sort edges by 2d length (descending) 67

- **GEOM_FADE2D::Func_ltEdge2D**
  - Functor to sort edges by 2d length (ascending) 67

- **GEOM_FADE2D::Label**
  - Label is a Text-Label for Visualization 67

- **GEOM_FADE2D::MeshGenParams**
  - Parameters for the mesh generator 68

- **GEOM_FADE2D::MsgBase**
  - MsgBase, a base class for message subscriber classes 72

- **GEOM_FADE2D::PeelPredicateTS**
  - User-defined peel predicate 73
GEOM_FADE2D::Point2
    Point 73
GEOM_FADE2D::Segment2
    Segment 78
GEOM_FADE2D::SegmentChecker
    SegmentChecker identifies intersecting line segments 79
GEOM_FADE2D::Triangle2
    Triangle 87
GEOM_FADE2D::TriangleAroundVertexIterator
    Iterator for all triangles around a given vertex 93
GEOM_FADE2D::UserPredicateT
    User-defined predicate (deprecated) 96
GEOM_FADE2D::Vector2
    Vector 96
GEOM_FADE2D::Visualizer2
    Visualizer2 is a general Postscript writer. It draws the objects Point2, Segment2, Triangle2, Circle2 and Label 98
GEOM_FADE2D::VoroCell2
    Voronoi cell 100
GEOM_FADE2D::Voronoi2
    Voronoi diagram 103
GEOM_FADE2D::VoroVertex2
    Voronoi vertex 106
GEOM_FADE2D::Zone2
    Zone2 is a certain defined area of a triangulation 107

4 File Index

4.1 File List

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

Bbox2.h 114
Circle2.h 115
Color.h 115
ConstraintGraph2.h 115
ConstraintSegment2.h 116
Edge2.h 116
Fade_2D.h 117
FadeExport.h 117
freeFunctions.h 117
5 Module Documentation

5.1 Tools

Functions

- void GEOM_FADE2D::edgesToPolygons (std::vector<Edge2> &vEdgesIn, std::vector<std::vector<Edge2>> &vvPolygonsOut, std::vector<Edge2> &vRemainingOut)

  Create polygons from a set of edges.

- bool GEOM_FADE2D::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)

- double GEOM_FADE2D::getArea2D (Point2 *p0, Point2 *p1, Point2 *p2)

  Get 2D area of a triangle.

- void GEOM_FADE2D::getBorders (const std::vector<Triangle2> *vT, std::vector<Segment2> &vBorderSegmentsOut)

  Get Borders.

- void GEOM_FADE2D::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).

- FUNC_DECLSPEC Orientation2 GEOM_FADE2D::getOrientation2 (const Point2 *p0, const Point2 *p1, const Point2 *p2)

  Get the orientation of three points.
• FUNC_DECLSPEC Orientation2 GEOM_FADE2D::getOrientation2_mt (const Point2 *p0, const Point2 *p1, const Point2 *p2)
  Get Orientation2 (MT)

• void GEOM_FADE2D::getUndirectedEdges (std::vector Triangle2 *, std::vector Edge2 &vUndirectedEdgesOut)
  Get undirected edges.

• bool GEOM_FADE2D::isSimplePolygon (std::vector Segment2 &vSegments)
  isSimplePolygon

• void GEOM_FADE2D::pointsToPolyline (std::vector Point2 &vInPoints, bool bClose, std::vector Segment2 &vOutSegments)
  Points-to-Polyline.

• bool GEOM_FADE2D::sortRing (std::vector Segment2 &vRing)
  Sort a vector of Segments.

• bool GEOM_FADE2D::sortRingCCW (std::vector Segment2 &vRing)
  Sort a vector of Segments.

5.1.1 Detailed Description

5.1.2 Function Documentation

5.1.2.1 edgesToPolygons()

void GEOM_FADE2D::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

<table>
<thead>
<tr>
<th>in</th>
<th>vEdgesIn</th>
<th>is a vector of oriented edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>vvPolygonsOut</td>
<td>contains one vector&lt;Edge2&gt; for each polygon found in the input data.</td>
</tr>
<tr>
<td>out</td>
<td>vRemainingOut</td>
<td>is used to return unusable remaining edges</td>
</tr>
</tbody>
</table>

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-coordinate 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](image)

5.1.2.2 fillHole()

```cpp
bool GEOM_FADE2D::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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vPolygonSegments</code></td>
<td>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 adjacent 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.</td>
</tr>
<tr>
<td><code>bWithRefine</code></td>
<td>specifies if additional vertices shall be created. (bWithRefine=true is experimental, don’t use currently)</td>
</tr>
</tbody>
</table>
5.1 Tools

### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>bVerbose</th>
<th>specifies if warnings shall be printed to stdout</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>vCornersOut</td>
<td>contains the created fill triangles, 3 corners per triangle, counterclockwise oriented.</td>
</tr>
</tbody>
</table>

#### 5.1.2.3 getArea2D()  
```cpp
double GEOM_FADE2D::getArea2D (  
    Point2 * p0,  
    Point2 * p1,  
    Point2 * p2 )
```

Returns the 2D area of the triangle defined by the three input points p0, p1, p2.

**Parameters**

| in   | p0, p1, p2 | are the corners of the triangle |

#### 5.1.2.4 getBorders()  
```cpp
void GEOM_FADE2D::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**

<table>
<thead>
<tr>
<th>in</th>
<th>vT</th>
<th>are the input triangles</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>vBorderSegmentsOut</td>
<td>is used to return all border segments</td>
</tr>
</tbody>
</table>

#### 5.1.2.5 getOrientation2()  
```cpp
FUNC_DECLSPEC Orientation2 GEOM_FADE2D::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

#### 5.1.2.6 getOrientation2_mt()  
```cpp
FUNC_DECLSPEC Orientation2 GEOM_FADE2D::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.

#### 5.1.2.7 getUndirectedEdges()  
```cpp
void GEOM_FADE2D::getUndirectedEdges (  
    std::vector< Triangle2 * > & vT,  
    std::vector< Edge2 > & vUndirectedEdgesOut )
```

A unique set of edges of vT is returned.
5.1.2.8 isSimplePolygon()  
bool GEOM_FADE2D::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.

5.1.2.9 pointsToPolyline()  
void GEOM_FADE2D::pointsToPolyline ( 
std::vector< Point2 > & vInPoints, 
bool bClose, 
std::vector< Segment2 > & vOutSegments )

Turns a vector of points (p0,p1,p2,...,pn,pm) 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

5.1.2.10 sortRing()  
bool GEOM_FADE2D::sortRing ( 
std::vector< Segment2 > & vRing )

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

5.1.2.11 sortRingCCW()  
bool GEOM_FADE2D::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

- `const char * GEOM_FADE2D::getFade2DVersion ()`
  Get the Fade2D version string.

- `FUNC_DECLSPEC int GEOM_FADE2D::getMajorVersionNumber ()`
  Get the major version number.

- `FUNC_DECLSPEC int GEOM_FADE2D::getMinorVersionNumber ()`
  Get the minor version number.

- `FUNC_DECLSPEC int GEOM_FADE2D::getRevisionNumber ()`
  Get the revision version number.

- `FUNC_DECLSPEC bool GEOM_FADE2D::isRelease ()`
  Check if a RELEASE or a DEBUG version is used.

5.2.1 Detailed Description
5.3 File I/O

Functions

- bool GEOM_FADE2D::readPointsBIN (const char ∗filename, std::vector<Point2> &vPointsIn)
  
  Reads points from a binary file.

- bool GEOM_FADE2D::readSegmentsBIN (const char ∗filename, std::vector<Segment2> &vSegmentsOut)
  
  Reads segments from a binary file.

- FUNC_DECLSPEC bool GEOM_FADE2D::readXY (const char ∗filename, std::vector<Point2> &vPointsOut)
  
  Reads (x y) points.

- FUNC_DECLSPEC bool GEOM_FADE2D::writePointsASCII (const char ∗filename, const std::vector<Point2> * &vPointsIn)
  
  Writes points to an ASCII file.

- bool GEOM_FADE2D::writePointsASCII (const char ∗filename, const std::vector<Point2> &vPointsIn)
  
  Write points to an ASCII file.

- bool GEOM_FADE2D::writePointsBIN (const char ∗filename, std::vector<Point2> &vPointsIn)
  
  Write points to a binary file.

- bool GEOM_FADE2D::writePointsBIN (const char ∗filename, std::vector<Point2> &vPointsIn)
  
  Write points to a binary file.

- bool GEOM_FADE2D::writeSegmentsBIN (const char ∗filename, std::vector<Segment2> &vSegmentsIn)
  
  Write segments to a binary file.

5.3.1 Detailed Description

5.3.2 Function Documentation

5.3.2.1 readPointsBIN()

bool GEOM_FADE2D::readPointsBIN ( 
    const char ∗ filename,
    std::vector<Point2> & vPointsIn )

reads points from a binary file.

See also

writePointsBIN()

5.3.2.2 readSegmentsBIN()

bool GEOM_FADE2D::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_FADE2D::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.
5.3.2.4 writePointsASCII() [1/2]  
```cpp
FUNC_DECLSPEC bool GEOM_FADE2D::writePointsASCII (  
    const char * filename,  
    const std::vector< Point2 * > & vPointsIn )
```
Writes points to an ASCII file, two coordinates (x y) 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()`)  

5.3.2.5 writePointsASCII() [2/2]  
```cpp
bool GEOM_FADE2D::writePointsASCII (  
    const char * filename,  
    const std::vector< Point2 > & vPointsIn )
```
Write points to an ASCII file  
See also  
`readPointsASCII()`  

5.3.2.6 writePointsBIN() [1/2]  
```cpp
bool GEOM_FADE2D::writePointsBIN (  
    const char * filename,  
    std::vector< Point2 * > & vPointsIn )
```
Writes points to a binary file  
See also  
`readPointsBIN()`  

5.3.2.7 writePointsBIN() [2/2]  
```cpp
bool GEOM_FADE2D::writePointsBIN (  
    const char * filename,  
    std::vector< Point2 > & vPointsIn )
```
File format:  
- `int filetype (20)`  
- `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 writeSegmentsBIN()  
```cpp
bool GEOM_FADE2D::writeSegmentsBIN (  
    const char * filename,  
    std::vector< Segment2 > & vSegmentsIn )
```
Binary file format:  
- `int filetype (21)`
size_t numSegments (vSegmentsIn.size())
double x0_source
double y0_source
double x0_target
double y0_target
...
double xn_source
double yn_source
double xn_target
double yn_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_FADE2D::generateCircle (int numPoints, double x, double y, double radiusX, double radiusY, std::vector < Point2 > &vCirclePointsOut)`
  
  Generate a circle.

- **FUNC_DECLSPEC** `void GEOM_FADE2D::generateRandomNumbers (size_t num, double min, double max, std::vector < double > &vRandomNumbersOut, unsigned int seed=0)`
  
  Generate random numbers.

- **FUNC_DECLSPEC** `void GEOM_FADE2D::generateRandomPoints (size_t numRandomPoints, double min, double max, std::vector < Point2 > &vRandomPointsOut, unsigned int seed=0)`
  
  Generate random points.

- **FUNC_DECLSPEC** `void GEOM_FADE2D::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_FADE2D::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_FADE2D::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_FADE2D::shear (std::vector < Point2 > &vPointsInOut, double shearX, double shearY)`

5.4.1 Detailed Description

5.4.2 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.

5.4.3 Function Documentation

5.4.3.1 `generateCircle()`  **FUNC_DECLSPEC** `void GEOM_FADE2D::generateCircle ( int numPoints, double x, double y, double radiusX, double radiusY, std::vector < Point2 > & vCirclePointsOut )`

Returns points on a circle centered at the given coordinates.
### 5.4.3.2 *generateRandomNumbers*()  

**FUNCTION**

```c++
DECLSPEC void GEOM_FADE2D::generateRandomNumbers(
    size_t num,
    double min,
    double max,
    std::vector< double > & vRandomNumbersOut,
    unsigned int seed = 0 );
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num</td>
<td>Number of random numbers to be generated</td>
</tr>
<tr>
<td>min</td>
<td>Lower bound</td>
</tr>
<tr>
<td>max</td>
<td>Upper bound</td>
</tr>
<tr>
<td>vRandomNumbersOut</td>
<td>is the output vector</td>
</tr>
<tr>
<td>seed</td>
<td>Initializes the random number generator RNG (default: 0...mapped to a random seed, other values: constant initialization)</td>
</tr>
</tbody>
</table>

**Note**

Reproducible 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.

### 5.4.3.3 *generateRandomPoints*()  

**FUNCTION**

```c++
DECLSPEC void GEOM_FADE2D::generateRandomPoints(
    size_t numRandomPoints,
    double min,
    double max,
    std::vector< Point2 > & vRandomPointsOut,
    unsigned int seed = 0 );
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>numRandomPoints</td>
<td>Number of points to be generated</td>
</tr>
<tr>
<td>min</td>
<td>Lower bound (x,y)</td>
</tr>
<tr>
<td>max</td>
<td>Upper bound (x,y)</td>
</tr>
<tr>
<td>vRandomPointsOut</td>
<td>is the output vector</td>
</tr>
<tr>
<td>seed</td>
<td>Initializes the random number generator RNG (default: 0...mapped to a random seed, other values: constant initialization)</td>
</tr>
</tbody>
</table>

Generated by Doxygen
5.4.3.4 generateRandomPolygon()  

```
FUNC_DECLSPEC void GEOM_FADE2D::generateRandomPolygon (
    size_t numSegments,
    double min,
    double max,
    std::vector< Segment2 > & vPolygonOut,
    unsigned int seed = 0 )
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>numSegments</td>
<td>Number of segments to be generated</td>
</tr>
<tr>
<td>min</td>
<td>Lower bound (x,y)</td>
</tr>
<tr>
<td>max</td>
<td>Upper bound (x,y)</td>
</tr>
<tr>
<td>out vPolygonOut</td>
<td>is the output vector</td>
</tr>
<tr>
<td>seed</td>
<td>initializes the random number generator RNG (default: 0...mapped to a random seed, other values: constant initialization)</td>
</tr>
</tbody>
</table>
5.4.3.5 generateRandomSegments()  

```cpp
FUNC_DECLSPEC void GEOM_FADE2D::generateRandomSegments(
    size_t numSegments,
    double min,
    double max,
    double maxLen,
    std::vector< Segment2 > & vSegmentsOut,
    unsigned int seed)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>numSegments</td>
<td>Number of segments to be generated</td>
</tr>
<tr>
<td>min</td>
<td>Lower bound (x,y)</td>
</tr>
<tr>
<td>max</td>
<td>Upper bound (x,y)</td>
</tr>
<tr>
<td>maxLen</td>
<td>Maximal segment length</td>
</tr>
<tr>
<td>vSegmentsOut</td>
<td>is the output vector</td>
</tr>
<tr>
<td>seed</td>
<td>initializes the random number generator RNG (default: 0...mapped to a random seed, other values: constant initialization)</td>
</tr>
</tbody>
</table>
5.4 Test Data Generators

5.4.3.6 generateSineSegments()  

\begin{verbatim}
void GEOM_FADE2D::generateSineSegments (int numSegments, int numPeriods, double xOffset, double yOffset, double xFactor, double yFactor, bool bSwapXY, std::vector< Segment2 > & vSineSegmentsOut )
\end{verbatim}

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>numSegments</td>
<td>Number of segments to be generated</td>
</tr>
<tr>
<td>numPeriods</td>
<td>Number of periods of the sine function</td>
</tr>
<tr>
<td>xOffset</td>
<td>Offset of the output x-coordinates</td>
</tr>
<tr>
<td>yOffset</td>
<td>Offset of the output y-coordinates</td>
</tr>
<tr>
<td>xFactor</td>
<td>Factor to scale the sine function in x direction</td>
</tr>
<tr>
<td>yFactor</td>
<td>Factor to scale the sine function in y direction</td>
</tr>
<tr>
<td>bSwapXY</td>
<td>Swap the x and y coordinate of the function</td>
</tr>
<tr>
<td>vSineSegmentsOut</td>
<td>is the output vector</td>
</tr>
</tbody>
</table>
Figure 5 Polyline generator: Polylines from sine functions
6 Class Documentation

6.1 GEOM_FADE2D::Bbox2 Class Reference

Bbox2 is an axis aligned 2D bounding box.
#include <Bbox2.h>

Public Member Functions

- **Bbox2 (GeomTest *pGeomTest_=NULL)**
  Constructor.
- **bool add (const Point2 &p)**
  Add a point.
- **bool add (size_t numPoints, double *coordinates)**
  Add points.
- **bool add (std::vector<Point2 >::const_iterator start_it, std::vector<Point2 >::const_iterator end_it)**
  Add points.
- **Point2 computeCenter () const**
  Compute the 2D midpoint.
- **bool doIntersect (const Bbox2 &other) const**
  Check intersection.
- **void doubleTheBox ()**
  Double the box.
- **void enlargeRanges (double factor)**
- **double get_maxX () const**
  Get maxX.
- **double get_maxY () const**
  Get maxY.
- **double get_minX () const**
  Get minX.
- **double get_minY () const**
  Get minY.
- **void getBoundary (std::vector<Segment2 > &vBoundary) const**
  Get boundary.
- **void getBounds (double &minX_, double &maxX_, double &minY_, double &maxY_) const**
  Get bounds.
- **void get_corners (std::vector<Point2 > &vBoxCorners) const**
  Get corners.
- **double getMaxCoord () const**
  Get maximum coordinate.
- **Point2 getMaxPoint () const**
  Get the max point.
- **double getMaxRange () const**
  Get max range.
- **double getMinCoord () const**
  Get minimum coordinate.
- **Point2 getMinPoint () const**
  Get the min point.
- **void getOffsetCorners (double offset, std::vector<Point2 > &vBoxCorners) const**
  Get offset corners.
• double getRangeX () const
  Get x-range.
• double getRangeY () const
  Get y-range.
• void inflateIfDegenerate (double val)
  Inflate if Degenerate.
• bool isInBox (const Point2 &p) const
  Point-in-Box Test.
• bool isValid () const
  Check if the bounds are valid.
• Bbox2 operator+ (const Bbox2 &b)
  Add a bounding box.
• void setMaxX (double val)
  Set maxX.
• void setMaxY (double val)
  Set maxY.
• void setMinX (double val)
  Set minX.
• void setMinY (double val)
  Set minY.

Protected Member Functions
• void treatPointForInvalidBox (const Point2 &p)
• void treatPointForValidBox (const Point2 &p)

Protected Attributes
• bool bValid
• double maxX
• double maxY
• double minX
• double minY
• GeomTest * pGeomTest

Friends
• std::ostream & operator<< (std::ostream &stream, const Bbox2 &pC)
  Print the box.

6.1.1 Detailed Description

6.1.2 Constructor & Destructor Documentation

6.1.2.1 Bbox2() GEOM_FADE2D::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

6.1.3 Member Function Documentation
6.1.3.1 add() [1/4] bool GEOM_FADE2D::Bbox2::add {
    const Point2 & p ) [inline]
Extends the 2D bounding box if required.
Returns
    true if the bounding box changes, false otherwise

6.1.3.2 add() [2/4] bool GEOM_FADE2D::Bbox2::add {
    size_t numPoints,
    double * coordinates ) [inline]
Extends the 2D bounding box if required.
Returns
    true if the bounding box changes, false otherwise

6.1.3.3 add() [3/4] bool GEOM_FADE2D::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

6.1.3.4 add() [4/4] bool GEOM_FADE2D::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

6.1.3.5 computeCenter() Point2 GEOM_FADE2D::Bbox2::computeCenter ( ) const

6.1.3.6 doIntersect() bool GEOM_FADE2D::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_FADE2D::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_FADE2D::Bbox2::get_maxX ( ) const [inline]
Returns
    maxX
6.1.3.9 get maxY() double GEOM_FADE2D::Bbox2::get maxY ( ) const [inline] Thanks
Returns
maxY

6.1.3.10 get minX() double GEOM_FADE2D::Bbox2::get minX ( ) const [inline] Returns
minX

6.1.3.11 get minY() double GEOM_FADE2D::Bbox2::get minY ( ) const [inline] Returns
minY

6.1.3.12 getBoundary() void GEOM_FADE2D::Bbox2::getBoundary ( std::vector< Segment2 > & vBoundary ) const Convenience function: Returns 4 border segments

6.1.3.13 getBounds() void GEOM_FADE2D::Bbox2::getBounds ( double & minX_,
double & maxX_,
double & minY_,
double & maxY_ ) const

6.1.3.14 getcorners() void GEOM_FADE2D::Bbox2::getcorners ( std::vector< Point2 > & vBoxCorners ) const Convenience function: Returns the 4 corners of the box

6.1.3.15 getMaxCoord() double GEOM_FADE2D::Bbox2::getMaxCoord ( ) const [inline] Returns
the largest coordinate value, i.e. max(maxX,maxY)

6.1.3.16 getMaxPoint() Point2 GEOM_FADE2D::Bbox2::getMaxPoint ( ) const [inline] Returns
the 2D corner point with the maximum coordinates

6.1.3.17 getMaxRange() double GEOM_FADE2D::Bbox2::getMaxRange ( ) const [inline] Returns
the largest range, i.e. max(getRangeX(),getRangeY())

Generated by Doxygen
6.1.3.18 getMinCoord() double GEOM_FADE2D::Bbox2::getMinCoord ( ) const [inline]
Returns
the smallest coordinate value, i.e. min(minX,minY)

6.1.3.19 getMinPoint() Point2 GEOM_FADE2D::Bbox2::getMinPoint ( ) const [inline]
Returns
the 2D corner point with the minimum coordinates

6.1.3.20 getOffsetCorners() void GEOM_FADE2D::Bbox2::getOffsetCorners (  
   double offset,
   std::vector< Point2 > & vBoxCorners ) const
Convenience function: Returns the 4 corners of an enlarged box. The box is enlarged by offset in each direction

6.1.3.21 getRangeX() double GEOM_FADE2D::Bbox2::getRangeX ( ) const [inline]
Returns
maxX-minX

6.1.3.22 getRangeY() double GEOM_FADE2D::Bbox2::getRangeY ( ) const [inline]
Returns
maxY-minY

6.1.3.23 inflateIfDegenerate() void GEOM_FADE2D::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 undesirable. The present method inflates the Bbox2 by adding /p val to maxX and/or maxY.

6.1.3.24 isInBox() bool GEOM_FADE2D::Bbox2::isInBox (  
   const Point2 & p ) const
Returns
true if minX <= p.x() <= maxX and minY <= p.y() <= maxY or false otherwise.

6.1.3.25 isValid() bool GEOM_FADE2D::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

6.1.3.26 operator+() Bbox2 GEOM_FADE2D::Bbox2::operator+ (  
   const Bbox2 & b )
Extends the 2D bounding box if required.
Returns
the resulting bounding box
6.1.4 Friends And Related Function Documentation

6.1.4.1 operator\texttt{\textless \textgreater} std::ostream\& operator\texttt{\textless \textgreater} ( std::ostream \& stream, const Bbox2 \& pC ) [friend]
Prints the box coordinates to \texttt{stream}

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

- \texttt{Bbox2.h}

6.2 GEOM_FADE2D::Circle2 Class Reference

Circle for visualization.

\#include \texttt{<Circle2.h>}

Public Member Functions

- \texttt{Circle2 (const Point2 \&center\_., double sqRadius\_)}  
  Constructor.
- \texttt{Circle2 (double x, double y, double sqRadius\_)}  
  Constructor.
- \texttt{Point2 getCenter ()}  
  \textit{Get the center of the circle.}
- \texttt{double getRadius ()}  
  \textit{Get the radius of the circle.}
- \texttt{double getSqRadius ()}  
  \textit{Get the squared radius of the circle.}

Protected Attributes

- \texttt{Point2 center}
- \texttt{double sqRadius}

Friends

- \texttt{std::ostream \& operator\texttt{\textless \textgreater} (std::ostream \&stream, Circle2 b)}

6.2.1 Detailed Description

See also

- Visualizer2

6.2.2 Constructor & Destructor Documentation

6.2.2.1 \texttt{Circle2() [1/2]}  
GEOM_FADE2D::Circle2::Circle2 (  
double x,  
double y,  
double sqRadius\_)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{x}</td>
<td>is x-coordinate of the center</td>
</tr>
<tr>
<td>\texttt{y}</td>
<td>is y-coordinate of the center</td>
</tr>
<tr>
<td>\texttt{sqRadius_}</td>
<td>is the squared radius of the circle</td>
</tr>
</tbody>
</table>
Warning

The method expects the *squared* radius

### 6.2.2.2 Circle2() [2/2]

```cpp
GEOM_FADE2D::Circle2::Circle2 (const Point2 & center_,
    double sqRadius_ )
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>center_</td>
<td>center of the circle</td>
</tr>
<tr>
<td>sqRadius_</td>
<td>squared radius of the circle</td>
</tr>
</tbody>
</table>

Warning

The method expects the *squared* radius

### 6.2.3 Member Function Documentation

#### 6.2.3.1 getCenter()

```cpp
Point2 GEOM_FADE2D::Circle2::getCenter ( )
```

**Returns**

a `Point2` which represents the center

#### 6.2.3.2 getRadius()

```cpp
double GEOM_FADE2D::Circle2::getRadius ( )
```

**Returns**

the radius

#### 6.2.3.3 getSqRadius()

```cpp
double GEOM_FADE2D::Circle2::getSqRadius ( )
```

**Returns**

the squared radius

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

- Circle2.h

### 6.3 GEOM_FADE2D::Color Class Reference

**Color** for visualization.

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

**Public Member Functions**

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

Generated by Doxygen
Static Public Member Functions

- static Color name getNextColorName()

Public Attributes

- float b
  Blue.
- bool bFill
  Fill the shape or not.
- float g
  Green.
- float r
  Red.
- float width
  Linewidth.

Static Public Attributes

- static size_t currentColorName

Friends

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

6.3.1 Detailed Description

See also
Visualizer2

6.3.2 Constructor & Destructor Documentation

6.3.2.1 Color() [1/2]

GEOM_FADE2D::Color::Color(
  double r_,
  double g_,
  double b_,
  double width_,
  bool bFill_ = false)

Parameters

<table>
<thead>
<tr>
<th>r_</th>
<th>red</th>
</tr>
</thead>
<tbody>
<tr>
<td>g_</td>
<td>green</td>
</tr>
<tr>
<td>b_</td>
<td>blue</td>
</tr>
<tr>
<td>width</td>
<td>linewidth</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>bFill</td>
<td>fill (default: false)</td>
</tr>
</tbody>
</table>

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.
6.3.2.2 Color() [2/2] GEOM_FADE2D::Color::Color {
    Colorname c,
    float width_ = 0.001,
    bool bFill_ = false 
}
For convenience predefined colors can be used.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>is a predefined color name</td>
</tr>
<tr>
<td>width</td>
<td>linewidth (default: 0.001)</td>
</tr>
<tr>
<td>bFill</td>
<td>fill (default: false)</td>
</tr>
</tbody>
</table>

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

6.4 GEOM_FADE2D::ConstraintGraph2 Class Reference

ConstraintGraph2 is a set of Constraint Edges (ConstraintSegment2)
#include <ConstraintGraph2.h>

Public Member Functions

- void getChildConstraintSegments (std::vector< ConstraintSegment2 * > &vConstraintSegments_) const
  Get child ConstraintSegment2 objects.
- void getDirectChildren (ConstraintSegment2 *pParent, ConstraintSegment2 *&pChild0, ConstraintSegment2 *&pChild1)
  Get direct children.
- Dt2 *getDt2 ()
- ConstraintInsertionStrategy getInsertionStrategy () const
  Get the constraint insertion strategy.
- void getOriginalConstraintSegments (std::vector< ConstraintSegment2 * > &vConstraintSegments_) const
  Get the original ConstraintSegment2 objects.
- void getPolygonVertices (std::vector< Point2 * > &vVertices_)
  Get the vertices of the constraint segments.
- bool isConstraint (ConstraintSegment2 *pCSeg) const
  Check if a ConstraintSegment2 is a member.
- bool isConstraint (Point2 *p0, Point2 *p1) const
  Check if an edge is a constraint.
- bool isOriented () const
  Are the segments of the constraint graph oriented?
- bool isPolygon () const
  Does the constraint graph form a closed polygon?
- bool isReverse (ConstraintSegment2 *pCSeg) const
- bool makeDelaunay (double minLength)
- void show (const char *name)
  Visualization.
- void show (Visualizer2 *pVis, const Color &color)
  Visualization.
Protected Attributes

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

6.4.1 Detailed Description

See also

Fade_2D::createConstraint()

Figure 6 Constraint Delaunay triangulation

6.4.2 Member Function Documentation

6.4.2.1 getChildConstraintSegments()

```cpp
void GEOM_FADE2D::ConstraintGraph2::getChildConstraintSegments(std::vector<ConstraintSegment2*>& vConstraintSegments_) const
```

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

6.4.2.2 getDirectChildren()

```cpp
void GEOM_FADE2D::ConstraintGraph2::getDirectChildren(ConstraintSegment2* pParent,
                                                      ConstraintSegment2** pChild0,
                                                      ConstraintSegment2** pChild1)
```

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>pParent</th>
<th>is a ConstraintSegment that may have been splitted</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>pChild0,pChild1</td>
<td>are the direct child segments of pParent. They can be alive or dead ( splitted ).</td>
</tr>
</tbody>
</table>

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

6.4.2.3 getDt2()

```cpp
Dt2* GEOM_FADE2D::ConstraintGraph2::getDt2()
```
Returns
the Delaunay class it belongs to

6.4.2.4 getInsertionStrategy()

ConstraintInsertionStrategy GEOM_FADE2D::ConstraintGraph2::getInsertionStrategy () const

Returns
CIS_CONFORMING_DELAUNAY or CISCONSTRAINED_DELAUNAY

6.4.2.5 getOriginalConstraintSegments()

void GEOM_FADE2D::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...)

6.4.2.6 getPolygonVertices()

void GEOM_FADE2D::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,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()?

6.4.2.7 isConstraint()[1/2]

bool GEOM_FADE2D::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.

6.4.2.8 isConstraint()[2/2]

bool GEOM_FADE2D::ConstraintGraph2::isConstraint ( Point2 * p0,
Point2 * p1 ) const

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

6.4.2.9 isOriented()

bool GEOM_FADE2D::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.
6.4.10 **isPolygon()**  
bool GEOM_FADE2D::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).

6.4.11 **isReverse()**  
bool GEOM_FADE2D::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.

6.4.12 **makeDelaunay()**  
bool GEOM_FADE2D::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.

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 subdivisions have been carried out or FALSE when minLength has avoided further subdivision.

6.4.13 **show() [1/2]**  
void GEOM_FADE2D::ConstraintGraph2::show (  
const char * name )

6.4.14 **show() [2/2]**  
void GEOM_FADE2D::ConstraintGraph2::show (  
Visualizer2 * pVis,  
const Color & color )

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

- ConstraintGraph2.h

6.5 **GEOM_FADE2D::ConstraintSegment2 Class Reference**

A ConstraintSegment2 represents a Constraint Edge.

# include <ConstraintSegment2.h>
Public Member Functions

- `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.

- `void getChildrenRec (std::vector<ConstraintSegment2 *> &vChildConstraintSegments)`
  Get all children. Recursively retrieve all children of the current `ConstraintSegment2`.

- `ConstraintInsertionStrategy getCIS () const`
  Get the Constraint Insertion Strategy (CIS)

- `Point2 * getSrc () const`
  Get the first endpoint.

- `Point2 * getTrg () const`
  Get the second endpoint.

- `Point2 * insertAndSplit (const Point2 &splitPoint)`
  Split a constraint segment.

- `bool isAlive () const`
  Check if the present `ConstraintSegment2` is alive.

- `bool operator< (const ConstraintSegment2 &pOther) const`
  `operator< (...)` Compares the vertex pointers of the endpoints, not the length

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

Public Attributes

- `int label`

Protected Attributes

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

Static Protected Attributes

- `static int runningLabel`

Friends

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

6.5.1 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.

6.5.2 Member Function Documentation
6.5.2.1 getCIS()  

ConstraintInsertionStrategy GEOM_FADE2D::ConstraintSegment2::getCIS ( ) const

Returns

the constraint insertion strategy (CIS) of the present object

6.5.2.2 getSrc()  

Point2* GEOM_FADE2D::ConstraintSegment2::getSrc ( ) const

Returns

the first vertex

6.5.2.3 getTrg()  

Point2* GEOM_FADE2D::ConstraintSegment2::getTrg ( ) const

Returns

the second vertex

6.5.2.4 insertAndSplit()  

Point2* GEOM_FADE2D::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 floating point 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.

6.5.2.5 isAlive()  

bool GEOM_FADE2D::ConstraintSegment2::isAlive ( ) const

Returns

TRUE when the object is alive, FALSE otherwise

6.5.2.6 split_combinatorialOnly()  

bool GEOM_FADE2D::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

6.6 GEOM_FADE2D::Edge2 Class Reference

Edge2 is a directed edge.
#include <Edge2.h>
Public Member Functions

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

- int getIndex () const
- double getLength2D () const
- void getPoints (Point2 *&p1, Point2 *&p2) const
  
  Get the endpoints.

- Point2 *getSrc () const
  
  Get the source point.

- Point2 *getTrg () const
  
  Get the target point.

- Triangle2 *getTriangle () const
- void getTriangles (Triangle2 *&pT0, Triangle2 *&pT1, int &idx0, int &idx1) const
- bool operator!= (const Edge2 &e) const
  
  operator!=()

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

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

Protected Attributes

- int oppIdx
- Triangle2 *pT

Friends

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

6.6.1 Constructor & Destructor Documentation

6.6.1.1 Edge2()  

**GEOM_FADE2D::Edge2::Edge2** ( 

Triangle2 *pT, 

int oppIdx_, )

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( pT )</td>
<td>is the triangle from which the edge is constructed</td>
</tr>
<tr>
<td>oppIdx_</td>
<td>is intra-triangle-index of the opposite vertex (of the edge) in ( pT )</td>
</tr>
</tbody>
</table>

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).

6.6.2 Member Function Documentation

6.6.2.1 getIndex()  

**int GEOM_FADE2D::Edge2::getIndex () const**

Get the opposite index

Generated by Doxygen
Returns

the intra-triangle-index of the opposite vertex

6.6.2.2 getLength2D()

double GEOM_FADE2D::Edge2::getLength2D ( ) const

Get the length

Returns

the length of the edge

6.6.2.3 getPoints()

void GEOM_FADE2D::Edge2::getPoints ( 
    Point2 * & p1,
    Point2 * & p2 ) const

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

6.6.2.4 getSrc()

Point2* GEOM_FADE2D::Edge2::getSrc ( ) const

Returns

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

6.6.2.5 getTrg()

Point2* GEOM_FADE2D::Edge2::getTrg ( ) const

Returns

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

6.6.2.6 getTriangle()

Triangle2* GEOM_FADE2D::Edge2::getTriangle ( ) const

Get the triangle

Returns

the triangle whose directed edge the present edge is

6.6.2.7 getTriangles()

void GEOM_FADE2D::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

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pT0</td>
<td>is used to return the triangle whose directed edge the present edge is</td>
</tr>
<tr>
<td>idx0</td>
<td>is the opposite intra-triangle-index in pT0 of the present edge</td>
</tr>
<tr>
<td>pT1</td>
<td>is the other adjacent triangle at the present edge (or NULL)</td>
</tr>
<tr>
<td>idx1</td>
<td>is the intra-triangle index of the present edge in pT1 (or -1)</td>
</tr>
</tbody>
</table>
6.7 GEOM_FADE2D::Fade_2D Class Reference

6.7.1 GEOM_FADE2D::Fade_2D Class Reference

Fade_2D is the Delaunay triangulation main class.

#include <Fade_2D.h>

Public Member Functions

- **Fade_2D (unsigned numExpectedVertices=3)**
  Constructor of the main triangulation class.

- ~Fade_2D ()
  Destructor.

- void applyConstraintsAndZones ()
  Apply conforming constraints and zones (deprecated!)

- bool checkValidity (bool bCheckEmptyCircleProperty, const char ∗msg) const
  Checks if a triangulation is valid.

- Bbox2 computeBoundingBox () const
  Compute the axis-aligned bounding box of the points.

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

- 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, bool bVerbose=true)
  Create a zone.

- 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.

- 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.

- void cutTriangles (const Point2 &knifeStart, const Point2 &knifeEnd, bool bTurnEdgesIntoConstraints)

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

- Edge2.h
Cut through a triangulation.

- void **cutTriangles** (std::vector< Segment2 > &vSegments, bool bTurnEdgesIntoConstraints)
  Cut through a triangulation.

- void deleteZone (Zone2 *pZone)
  Delete a Zone2 object.

- bool drape (std::vector< Segment2 > &vSegmentsIn, std::vector< Segment2 > &vSegmentsOut) const
  Drape segments along a surface.

- void exportTriangulation (FadeExport &fadeExport, bool bWithCustomIndices, bool bClear)
  Export triangulation data from Fade.

- Triangle2 *getAdjacentTriangle (Point2 *p0, Point2 *p1) const
  Get adjacent triangle.

- void getAliveAndDeadConstraintSegments (std::vector< ConstraintSegment2 * > &vAllConstraintSegments) const
  Get all (alive and dead) constraint segments.

- void getAliveConstraintSegments (std::vector< ConstraintSegment2 * > &vAliveConstraintSegments) const
  Get active (alive) constraint segments.

- ConstraintSegment2 *getConstraintSegment (Point2 *p0, Point2 *p1) const
  Retrieve a ConstraintSegment2.

- void getConvexHull (bool bAllVertices, std::vector< Point2 * > &vConvexHullPointsOut)
  Compute the convex hull.

- void getIncidentTriangles (Point2 *pVtx, std::vector< Triangle2 * > &vIncidentT) const
  Get incident triangles.

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

- Orientation2 getOrientation (const Point2 &p0, const Point2 &p1, const Point2 &p2)
  Compute the orientation of 3 points.

- void getTrianglePointers (std::vector< Triangle2 * > &vAllTriangles) const
  Get pointers to all triangles.

- void getVertexPointers (std::vector< Point2 * > &vAllPoints) const
  Get pointers to all vertices.

- Voronoi2 *getVoronoiDiagram ()
  Get the Voronoi diagram.

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

- Zone2 *importTriangles (std::vector< Point2 > &vPoints, bool bReorientIfNeeded, bool bCreateExtendedBoundingBox)
  Import triangles.

- 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.

- bool isConstraint (Point2 *p0, Point2 *p1)
  Check if an edge is a constraint edge.

- bool isConstraint (Point2 *pVtx)
  Check if a vertex is a constraint vertex.

- bool isConstraint (Triangle2 *pT, int ith)
  Check if an edge is a constraint edge.
• bool load (const char *filename, std::vector< Zone2 * > &vZones)
  Load a triangulation.
• bool load (std::istream &stream, std::vector< Zone2 * > &vZones)
  Load a triangulation.
• Triangle2 * locate (const Point2 &p)
  Locate a triangle which contains p.
• double measureTriangulationTime (std::vector< Point2 > &vPoints)
  Measure the Delaunay triangulation time.
• size_t numberOfPoints () const
  Number of points.
• size_t numberOfTriangles () const
  Number of triangles.
• void printLicense () const
  Prints license information.
• void refine (Zone2 *pZone, double minAngleDegree, double minEdgeLength, double maxEdgeLength, bool bAllowConstraintSplitting)
  Delaunay refinement.
• void refineAdvanced (MeshGenParams *pParameters)
  Delaunay refinement and grid meshing.
• void remove (Point2 *pVertex)
  Remove a single vertex.
• bool saveTriangulation (const char *filename, std::vector< Zone2 * > &vSaveZones)
  Save a triangulation.
• bool saveTriangulation (std::ostream &stream, std::vector< Zone2 * > &vSaveZones)
  Save a triangulation.
• bool saveZones (const char *filename, std::vector< Zone2 * > &vSaveZones)
  Save zones.
• bool saveZones (std::ostream &stream, std::vector< Zone2 * > &vSaveZones)
  Save zones.
• void setFastMode (bool bFast)
  Set fast mode.
• int setNumCPU (int numCPU)
  Set the number CPU cores for multithreading.
• void show (const char *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 statistics (const char *s) const
  Statistics.
• void subscribe (MsgType msgType, MsgBase *pMsg)
  Register a message receiver.
• void unsubscribe (MsgType msgType, MsgBase *pMsg)
  Unregister a message receiver.
• void writeObj (const char *filename) const
  Write the current triangulation to an *.obj file.
• void writeObj (const char *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.
6.7.1 Detailed Description

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

6.7.2 Constructor & Destructor Documentation

6.7.2.1 Fade_2D()

```
GEOM_FADE2D::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.

6.7.3 Member Function Documentation

6.7.3.1 applyConstraintsAndZones()

```
void GEOM_FADE2D::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_CONFORMING_DELAUNAY` and `CIS_CONFORMING_DELAUNAY_SEGMENT_LEVEL` are deprecated. These are only kept for backwards compatibility. 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

6.7.3.2 checkValidity()

```
bool GEOM_FADE2D::Fade_2D::checkValidity ( 
    bool bCheckEmptyCircleProperty,
    const char * 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.

6.7.3.3 computeBoundingBox()

```
Bbox2 GEOM_FADE2D::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.

6.7.3.4 createConstraint()

```
ConstraintGraph2* GEOM_FADE2D::Fade_2D::createConstraint ( 
```

Generated by Doxygen
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 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. |
| bOrientedSegments | specifies whether the segments in vSegments are oriented (oriented, not ordered!). If later a zone is to be made with the returned ConstraintGraph2 object this is only possible if the value is true (then it is assumed that all segments are counterclockwise oriented) or if the ConstraintGraph2 represents exactly one closed polygon. The value affects also the order of the returned vertices when later ConstraintGraph2::getPolygonVertices() is called. This is a default parameter and it defaults to false. |

Returns

a pointer to the new ConstraintGraph2 object

Figure 7 Delaunay triangulation without constraints

Figure 8 Constraint Delaunay triangulation
6.7.3.5 createZone() [1/4]

Zone2* GEOM_FADE2D::Fade_2D::createZone ( const std::vector< ConstraintGraph2* > & vConstraintGraphs, ZoneLocation zoneLoc, const Point2 & startPoint, bool bVerbose = true )

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vConstraintGraphs</td>
<td>is a vector of ConstraintGraph objects</td>
</tr>
<tr>
<td>zoneLoc</td>
<td>must be ZL_GROW</td>
</tr>
<tr>
<td>startPoint</td>
<td>is the point from which the area is grown until the borders specified in vConstraintGraphs are reached</td>
</tr>
<tr>
<td>bVerbose</td>
<td>is by default true and causes a warning if NULL is returned.</td>
</tr>
</tbody>
</table>

Returns

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

6.7.3.6 createZone() [2/4]

Zone2* GEOM_FADE2D::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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>zoneLoc</td>
<td>is ZL_INSIDE, ZL_OUTSIDE or ZL_GLOBAL.</td>
</tr>
<tr>
<td>pConstraintGraph</td>
<td>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.</td>
</tr>
<tr>
<td>bVerbose</td>
<td>is by default true and causes a warning if NULL is returned.</td>
</tr>
</tbody>
</table>
Returns

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

Figure 10 Zones in a triangulation

6.7.3.7 createZone() [3/4] Zone2* GEOM_FADE2D::Fade_2D::createZone (  
    ConstraintGraph2 * pConstraintGraph,  
    ZoneLocation zoneLoc,  
    const Point2 & startPoint,  
    bool bVerbose = true )  
A Zone2 object is an area of the triangulation, see createZone

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pConstraintGraph</td>
<td>is a constraint whose edges specify the area's border</td>
</tr>
<tr>
<td>zoneLoc</td>
<td>must be ZL_GROW</td>
</tr>
<tr>
<td>startPoint</td>
<td>is the point from which the area is grown until the borders specified in pConstraint are reached</td>
</tr>
<tr>
<td>bVerbose</td>
<td>is by default true and causes a warning if NULL is returned.</td>
</tr>
</tbody>
</table>

Returns

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

6.7.3.8 createZone() [4/4] Zone2* GEOM_FADE2D::Fade_2D::createZone (  
    std::vector< Triangle2 * > & vTriangles,  
    bool bVerbose = true )  
A Zone2 object is an area of the triangulation, see createZone

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vTriangles</td>
<td></td>
</tr>
<tr>
<td>bVerbose</td>
<td>is by default true and causes a warning if NULL is returned.</td>
</tr>
</tbody>
</table>

Returns

a pointer to the new Zone2 object (or NULL if vTriangles is empty)
6.7.3.9 createZone_cookieCutter() Zone2* GEOM_FADE2D::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 may flip existing edges in the triangulation but this can be avoided using bProtectEdges=true. In this case new constraint edges may be created.

6.7.3.10 cutTriangles()[1/2] void GEOM_FADE2D::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 bTurnEdgesIntoConstraints=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.

6.7.3.11 cutTriangles()[2/2] void GEOM_FADE2D::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 bTurnEdgesIntoConstraints)`
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.

### 6.7.3.12 deleteZone()

```cpp
void GEOM_FADE2D::Fade_2D::deleteZone ( Zone2 * pZone )
```

**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.

### 6.7.3.13 drape()

```cpp
bool GEOM_FADE2D::Fade_2D::drape ( std::vector< Segment2 > & vSegmentsIn,
                                   std::vector< Segment2 > & vSegmentsOut ) 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.

#### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>vSegmentsIn</th>
<th>Input segments</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>vSegmentsOut</td>
<td>Output segments</td>
</tr>
</tbody>
</table>

#### 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 11 Drape: Input segments are draped (red) onto a TIN. They are subdivided (blue) at intersections with triangulation edges
Draping segments onto a TIN does not insert them. Use `Fade_2D::createConstraint()` for that purpose.

### 6.7.3.14 exportTriangulation()

```cpp
void GEOM_FADE2D::Fade_2D::exportTriangulation (
    FadeExport & fadeExport,
    bool bWithCustomIndices,
    bool bClear )
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fadeExport</td>
<td>is a struct that will hold the requested triangulation data</td>
</tr>
<tr>
<td>bWithCustomIndices</td>
<td>determines whether the custom indices of the points are also stored</td>
</tr>
<tr>
<td>bClear</td>
<td>determines whether the Fade instance is cleared during the export operation to save memory</td>
</tr>
</tbody>
</table>

**Note**

When `bClear` is true then all memory of the Fade object is deleted i.e., all existing pointers to its objects become invalid.

### 6.7.3.15 getAdjacentTriangle()

```cpp
Triangle2* GEOM_FADE2D::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.

### 6.7.3.16 getConstraintSegment()

```cpp
ConstraintSegment2* GEOM_FADE2D::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)

### 6.7.3.17 getConvexHull()

```cpp
void GEOM_FADE2D::Fade_2D::getConvexHull ( 
    bool bAllVertices,
    std::vector<Point2*> & vConvexHullPointsOut )
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bAllVertices</td>
<td>determines if all convex hull points are returned or if collinear ones shall be removed.</td>
</tr>
<tr>
<td>vConvexHullPointsOut</td>
<td>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.</td>
</tr>
</tbody>
</table>
6.7.3.18 **getIncidentTriangles()**  
```cpp
void GEOM_FADE2D::Fade_2D::getIncidentTriangles (  
    Point2 * pVtx,  
    std::vector<Triangle2 *> & vIncidentT ) const
```
Stores pointers to all triangles around pVtx into vIncidentT.

6.7.3.19 **getIncidentVertices()**  
```cpp
void GEOM_FADE2D::Fade_2D::getIncidentVertices (  
    Point2 * pVtx,  
    std::vector<Point2 *> & vIncidentVertices ) const
```
Stores pointers to all vertices around pVtx into vIncidentVertices.

6.7.3.20 **getOrientation()**  
```cpp
Orientation2 GEOM_FADE2D::Fade_2D::getOrientation (  
    const Point2 & p0,  
    const Point2 & p1,  
    const Point2 & p2  
)
```
Returns  
ORIENTATION2_COLLINEAR, ORIENTATION2_CW (clockwise) or ORIENTATION2_CCW (counterclockwise).

6.7.3.21 **getTrianglePointers()**  
```cpp
void GEOM_FADE2D::Fade_2D::getTrianglePointers (  
    std::vector<Triangle2 *> & vAllTriangles ) const
```
This command fetches the existing triangles.

Parameters

- `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.

6.7.3.22 **getVertexPointers()**  
```cpp
void GEOM_FADE2D::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.

6.7.3.23 **getVoronoiDiagram()**  
```cpp
Voronoi2* GEOM_FADE2D::Fade_2D::getVoronoiDiagram ( )
```
Returns

a dual Voronoi diagram that changes dynamically when the triangulation changes.

6.7.3.24 hasArea() bool GEOM_FADE2D::Fade_2D::hasArea() 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 12 Triangles are generated as soon as the first non-collinear point is inserted.](image)

Returns

true if at least one triangle exists
false otherwise

6.7.3.25 importTriangles() Zone2* GEOM_FADE2D::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

<table>
<thead>
<tr>
<th>vPoints</th>
<th>contains the input vertices (3 subsequent ones per triangle)</th>
</tr>
</thead>
<tbody>
<tr>
<td>bReorientIfNeeded</td>
<td>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.</td>
</tr>
<tr>
<td>bCreateExtendedBoundingBox</td>
<td>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.</td>
</tr>
</tbody>
</table>

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().
6.7.3.26 insert() [1/4]  

Point2* GEOM_FADE2D::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.

6.7.3.27 insert() [2/4] void GEOM_FADE2D::Fade_2D::insert (const std::vector<Point2> & vInputPoints)

Parameters

\( vInputPoints \) contains the points to be inserted.

Note

Use Fade_2D::setNumCPU() to activate multithreading

6.7.3.28 insert() [3/4] void GEOM_FADE2D::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 occurrence.

Note

Use Fade_2D::setNumCPU() to activate multithreading

6.7.3.29 insert() [4/4] void GEOM_FADE2D::Fade_2D::insert (int numPoints, double * aCoordinates, Point2 ** aHandles)

Parameters

\( numPoints \) is the number of points to be inserted

\( aCoordinates \) is an array of \( 2n \) double values, e.g. \{x0,y0,x1,y1,...,xn,yn\}
Parameters

**aHandles** is an empty array with size n where pointers to the inserted points will be stored by Fade.

Note

Use `Fade_2D::setNumCPU()` to activate multithreading.

### 6.7.3.30 isConstraint [1/3]

```cpp
bool GEOM_FADE2D::Fade_2D::isConstraint ( Point2 * p0,
                                           Point2 * p1 ) const
```

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

### 6.7.3.31 isConstraint [2/3]

```cpp
bool GEOM_FADE2D::Fade_2D::isConstraint ( Point2 * pVtx ) const
```

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

### 6.7.3.32 isConstraint [3/3]

```cpp
bool GEOM_FADE2D::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.

### 6.7.3.33 load [1/2]

```cpp
bool GEOM_FADE2D::Fade_2D::load ( const char * filename,
                                 std::vector< Zone2 * > & vZones )
```

Loads a triangulation together with any custom indices, constraint-edges and zones from a binary file

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>filename</th>
<th>is the name of the input file</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>vZones</td>
<td>is used to return Zone2+ pointers if any. The order of the pointers is the same as at the time of storage</td>
</tr>
</tbody>
</table>

Returns

whether the operation was successful

### 6.7.3.34 load [2/2]

```cpp
bool GEOM_FADE2D::Fade_2D::load ( std::istream & stream,
                                 std::vector< Zone2 * > & vZones )
```

Loads a triangulation together with any custom indices, constraint-edges and zones from a stream

Parameters

<table>
<thead>
<tr>
<th>stream</th>
<th>is an input stream</th>
</tr>
</thead>
<tbody>
<tr>
<td>out vZones</td>
<td>is used to return Zone2+ pointers if any. The order of the pointers is the same as at the time of storage</td>
</tr>
</tbody>
</table>

Returns

whether the operation was successful
6.7.3.35 locate()  

```cpp
Triangle2* GEOM_FADE2D::Fade_2D::locate (const Point2 & p )
```

Figure 13 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 hasArea()==false or if \( p \) is outside the triangulation)

6.7.3.36 measureTriangulationTime()  

```cpp
double GEOM_FADE2D::Fade_2D::measureTriangulationTime (std::vector<Point2> & vPoints )
```

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

Parameters

\( \text{in vPoints} \) 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

6.7.3.37 numberOfPoints()  

```cpp
size_t GEOM_FADE2D::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 advised to avoid calling this method over-frequently in a loop. Duplicate point insertions count only once.

6.7.3.38 numberOfTriangles()  
size_t GEOM_FADE2D::Fade_2D::numberOfTriangles() const

Returns

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

6.7.3.39 refine()  
void GEOM_FADE2D::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

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

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::convertToBoundedZone().

6.7.3.40 refineAdvanced()  
void GEOM_FADE2D::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.

6.7.3.41 remove()  
void GEOM_FADE2D::Fade_2D::remove(
    Point2 * pVertex)

Generated by Doxygen
Parameters

\texttt{pVertex} shall be removed.

Note

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

6.7.3.42 \texttt{saveTriangulation()} [1/2]

\begin{verbatim}
bool GEOM_FADE2D::Fade_2D::saveTriangulation {
    const char * filename,
    std::vector< Zone2 * > & vSaveZones
}
\end{verbatim}

The \texttt{saveTriangulation()} command saves all triangles of the present triangulation to a binary file. Thereby it retains constraint edges and custom vertex indices, if any. If Zone2\* pointers are specified, these zones will be saved also and their order will be retained.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>\texttt{filename}</th>
<th>is the name of the output file</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>\texttt{vSaveZones}</td>
<td>is used specify zones that shall additionally be saved</td>
</tr>
</tbody>
</table>

See also

If you just want to store zones, use \texttt{Zone2::save()} or \texttt{Fade_2D::saveTriangulation()}. Use \texttt{Fade_2D::load()} to reload data from such files.

Returns

whether the operation was successful

6.7.3.43 \texttt{saveTriangulation()} [2/2]

\begin{verbatim}
bool GEOM_FADE2D::Fade_2D::saveTriangulation {
    std::ostream & stream,
    std::vector< Zone2 * > & vSaveZones
}
\end{verbatim}

The \texttt{saveTriangulation()} command saves all triangles of the present triangulation to a stream. Thereby it retains constraint edges and custom vertex indices, if any. If Zone2\* pointers are specified, these zones will be saved also and their order will be retained.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>\texttt{stream}</th>
<th>is the output stream</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>\texttt{vSaveZones}</td>
<td>is used specify zones that shall additionally be saved</td>
</tr>
</tbody>
</table>

See also

If you just want to store zones, use \texttt{Zone2::save()} or \texttt{Fade_2D::saveTriangulation()}. Use \texttt{Fade_2D::load()} to reload data from such files.

Returns

whether the operation was successful
### saveZones() \([1/2]\) bool GEOM_FADE2D::Fade_2D::saveZones (const char ∗ filename, std::vector< Zone2 ∗ > & vSaveZones )

The `saveZones()` command saves the triangles of the zones in `vSaveZones` to a binary file. Thereby it keeps the order of the zones and it retains any constraint edges and custom indices in the domain.

**Note**

A Delaunay triangulation is convex without holes and this may not hold for the zones to be saved. Thus extra triangles may be saved to fill concavities. These extra-triangles will belong to the Fade_2D instance but not to any Zone2 when reloaded later.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>filename</th>
<th>is the name of the output file</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>vSaveZones</td>
<td>(non-empty) specifies the zones to be saved</td>
</tr>
</tbody>
</table>

**Returns**

whether the operation was successful

**See also**

The `saveTriangulation()` command can be used to store all triangles of a triangulation plus any specified zones. The `Zone2::save()` command is used to store just one zone. Use `Fade_2D::load()` to reload data from such files.

### saveZones() \([2/2]\) bool GEOM_FADE2D::Fade_2D::saveZones (std::ostream & stream, std::vector< Zone2 ∗ > & vSaveZones )

The `saveZones()` command saves the triangles of the zones in `vSaveZones` to stream. Thereby it keeps the order of the zones and it retains any constraint edges and custom indices in the domain.

**Note**

A Delaunay triangulation is convex without holes and this may not hold for the zones to be saved. Thus extra triangles may be saved to fill concavities. These extra-triangles will belong to the Fade_2D instance but not to any Zone2 when reloaded later.

**Parameters**

| out    | vSaveZones | (non-empty) specifies the zones to be saved |

**Returns**

whether the operation was successful

**See also**

The `saveTriangulation()` command can be used to store all triangles of a triangulation plus any specified zones. The `Zone2::save()` command is used to store just one zone. Use `Fade_2D::load()` to reload data from such files.
6.7.3.46 **setFastMode()** void GEOM_FADE2D::Fade_2D::setFastMode (bool bFast)

By default, numerically perfect calculations are performed to compute a 100% perfect Delaunay triangulation. However, the difference to using double precision arithmetic is hardly noticeable. It is rather relevant in scientific applications while practical applications may want to skip the computationally expensive calculations. Depending on the position of the input points, the effect of FastMode can be zero or a quite considerable acceleration.

**Parameters**

| bFast | use true to avoid using multiple precision arithmetic in favor of better performance. |

6.7.3.47 **setNumCPU()** int GEOM_FADE2D::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.

6.7.3.48 **show() [1/2]** void GEOM_FADE2D::Fade_2D::show (const char * 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) |

6.7.3.49 **show() [2/2]** void GEOM_FADE2D::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

<table>
<thead>
<tr>
<th>pVis</th>
<th>is the pointer of a Visualizer2 object that may already contain geometric primitives or that may later be used to draw further elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>bWithConstraints</td>
<td>specifies if constraint segments shall be shown (default: true)</td>
</tr>
</tbody>
</table>

Note

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

6.7.3.50 statistics()  

void GEOM_FADE2D::Fade_2D::statistics (  
    const char * s ) const  

Prints mesh statistics to stdout.

6.7.3.51 subscribe()  

void GEOM_FADE2D::Fade_2D::subscribe (  
    MsgType msgType,  
    MsgBase * pMsg )  

Parameters

<table>
<thead>
<tr>
<th>msgType</th>
<th>is the type of message the subscriber shall receive, e.g. MSG_PROGRESS or MSG_WARNING</th>
</tr>
</thead>
<tbody>
<tr>
<td>pMsg</td>
<td>is a pointer to a custom class derived from MsgBase</td>
</tr>
</tbody>
</table>

6.7.3.52 unsubscribe()  

void GEOM_FADE2D::Fade_2D::unsubscribe (  
    MsgType msgType,  
    MsgBase * pMsg )  

Parameters

<table>
<thead>
<tr>
<th>msgType</th>
<th>is the type of message the subscriber shall not receive anymore</th>
</tr>
</thead>
<tbody>
<tr>
<td>pMsg</td>
<td>is a pointer to a custom class derived from MsgBase</td>
</tr>
</tbody>
</table>

6.7.3.53 writeObj() [1/2]  

void GEOM_FADE2D::Fade_2D::writeObj (  
    const char * filename ) const  

Visualizes the current triangulation. The *.obj format represents a 3D scene.

6.7.3.54 writeObj() [2/2]  

void GEOM_FADE2D::Fade_2D::writeObj (  
    const char * filename,  
    Zone2 * pZone ) const  

Visualizes a certain Zone2 object of the present triangulation. The *.obj format represents a 3D scene.

6.7.3.55 writeWebScene() [1/2]  

void GEOM_FADE2D::Fade_2D::writeWebScene (  
    const char * path ) const  

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

6.7.3.56 writeWebScene() [2/2]  

void GEOM_FADE2D::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`

### 6.8 GEOM_FADE2D::FadeExport Struct Reference

`FadeExport` is a simple struct to export triangulation data.

```cpp
#include <FadeExport.h>
```

**Public Member Functions**

- void `extractTriangleNeighborhoods(std::vector<std::pair<int, int>> &vNeigs) const`  
  Determine index-pairs of adjacent triangles.
- void `getCoordinates(int vtxIdx, double &x, double &y) const`  
  Get the coordinates for a certain vertex index.
- void `getCornerIndices(int triIdx, int &vtxIdx0, int &vtxIdx1, int &vtxIdx2) const`  
  Get the corner indices of a certain triangle.
- void `print() const`  
  Print data for demonstration purposes.
- bool `writeObj(const char *filename) const`  
  Write an `.obj` file (supported by virtually any 3D viewer)

**Public Attributes**

- double * `aCoords`  
  Cartesian coordinates (dim=numPoints)
- int * `aCustomIndices`  
  Custom indices of the points (only when exported)
- int * `aTriangles`  
  3 counterclockwise oriented vertex-indices per triangle (3*numTriangles)
- int `dim`  
  Dimension.
- int `numCustomIndices`  
  number of custom indices (same as numPoints when exported, otherwise 0)
- int `numPoints`  
  number of points
- int `numTriangles`  
  number of triangles

#### 6.8.1 Detailed Description

This data structure is there to get data out of Fade easily and memory efficiently. **The source code of this class is deliberately included in the header file** so that users can take over the code to their individual project. Have a look at the [Examples](#).

#### 6.8.2 Member Function Documentation

##### 6.8.2.1 `getCoordinates()`

```cpp
void GEOM_FADE2D::FadeExport::getCoordinates ( 
  int vtxIdx, 
  double & x, 
  double & y ) const [inline]
```
Parameters

<table>
<thead>
<tr>
<th>vtxIdx</th>
<th>[in] vertex index</th>
</tr>
</thead>
<tbody>
<tr>
<td>x,y</td>
<td>[out] coordinates</td>
</tr>
</tbody>
</table>

6.8.2.2 getCornerIndices()  
void GEOM_FADE2D::FadeExport::getCornerIndices (  
    int triIdx,  
    int & vtxIdx0,  
    int & vtxIdx1,  
    int & vtxIdx2 ) const [inline]

Parameters

<table>
<thead>
<tr>
<th>triIdx</th>
<th>[in] triangle index</th>
</tr>
</thead>
<tbody>
<tr>
<td>vtxIdx0,vtxIdx1,vtxIdx2</td>
<td>[out] corner indices</td>
</tr>
</tbody>
</table>

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

- FadeExport.h

6.9 GEOM_FADE2D::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

6.10 GEOM_FADE2D::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

6.11 GEOM_FADE2D::Label Class Reference

Label is a Text-Label for Visualization.
#include <Label.h>

Public Member Functions

- Label (const Label &other)
- Label (const Point2 &p_, const char *s_, bool bWithMark_=true, int fontSize_=8)

Constructs a Text-Label.
- const char * getCS () const
- Label & operator= (const Label &other)
Public Attributes

- bool bWithMark
- int fontSize
- Point2 p
- LDat * pDat

6.11.1 Detailed Description

See also

Visualizer2 where Label objects are used for visualizations

6.11.2 Constructor & Destructor Documentation

6.11.2.1 Label() GEOM_FADE2D::Label::Label {
    const Point2 & p_,
    const char * s_,
    bool bWithMark_ = true,
    int fontSize_ = 8
}

Parameters

<table>
<thead>
<tr>
<th>p_</th>
<th>is the point where the label appears</th>
</tr>
</thead>
<tbody>
<tr>
<td>s_</td>
<td>is the text to be shown</td>
</tr>
<tr>
<td>bWithMark_</td>
<td>switches between text-only and text-with-mark</td>
</tr>
<tr>
<td>fontSize_</td>
<td></td>
</tr>
</tbody>
</table>

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

- Label.h

6.12 GEOM_FADE2D::MeshGenParams Class Reference

Parameters for the mesh generator.

#include <MeshGenParams.h>

Public Member Functions

- MeshGenParams (Zone2 *pZone_)
- void addLockedConstraint (ConstraintSegment2 *pConstraintSegment)
  Constraint Segments that shall not be splitted.
- virtual double getMaxEdgeLength (Triangle2 *pT)
  getMaxEdgeLength(Triangle2* pT)
- virtual double getMaxTriangleArea (Triangle2 *pT)
  getMaxTriangleArea(Triangle2* pT)

Public Attributes

- bool bAllowConstraintSplitting
  bAllowConstraintSplitting
- bool bKeepExistingSteinerPoints
  Steiner points from previous refinements.
- double capAspectLimit
  capAspectLimit
6.12 GEOM_FADE2D::MeshGenParams Class Reference

- int command
  Command.
- double gridLength
  gridLength
- Vector2 gridVector
  gridVector
- double growFactor
  growFactor
- double growFactorMinArea
  growFactorMinArea
- double maxEdgeLength
  Maximum edge length.
- double maxTriangleArea
  maxTriangleArea
- double minAngleDegree
  Minimum interior triangle angle.
- double minEdgeLength
  Minimum edge length.
- Zone2 * pZone
  Zone to be meshed.

6.12.1 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/

6.12.2 Member Function Documentation

6.12.2.1 addLockedConstraint()  

void GEOM_FADE2D::MeshGenParams::addLockedConstraint ( 
  ConstraintSegment2 * pConstraintSegment )

In case that some ConstraintSegment2 can be splitted and others must not be splitted use bAllow<- 
  ConstraintSplitting=true and add the ones that must not be splitted.

6.12.2.2 getMaxEdgeLength()  

virtual double GEOM_FADE2D::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.
6.12.2.3 getMaxTriangleArea() virtual double GEOM_FADE2D::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 $D_{\text{BL_MAX}}$ if not changed by the user). This method can be overridden by the client software in order to control the local mesh density.
 Defines if constraint segments can be splitted. Default: yes

### 6.12.3.2 bKeepExistingSteinerPoints

`bool GEOM_FADE2D::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 `bKeepExistingSteinerPoints` is true.

### 6.12.3.3 capAspectLimit

`double GEOM_FADE2D::MeshGenParams::capAspectLimit`

Limits the quotient `edgeLength / height`. Default value: `10.0`

### 6.12.3.4 command

`int GEOM_FADE2D::MeshGenParams::command`

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

### 6.12.3.5 gridLength

`double GEOM_FADE2D::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 16 Grid Meshing axis aligned](image)

### 6.12.3.6 gridVector

`Vector2 GEOM_FADE2D::MeshGenParams::gridVector`

When grid-meshing is used the grid is aligned to the `gridVector`. By default `gridVector` is axis aligned.
6.12.3.7 **growFactor** double GEOM_FADE2D::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) \[ \frac{\text{area}(t0)}{\text{area}(t1)} < \text{growFactor} \].
Recommendation: \( \text{growFactor} > 5.0 \), Default: \( \text{growFactor} = \text{DBL_MAX} \).

6.12.3.8 **growFactorMinArea** double GEOM_FADE2D::MeshGenParams::growFactorMinArea

The \( \text{growFactor} \) value is ignored for triangles with an area smaller than \( \text{growFactorMinArea} \). This value prevents generation of hundreds of tiny triangles around one that is unusually small. Default: 0.001.

6.12.3.9 **maxEdgeLength** double GEOM_FADE2D::MeshGenParams::maxEdgeLength

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

6.12.3.10 **maxTriangleArea** double GEOM_FADE2D::MeshGenParams::maxTriangleArea

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

6.12.3.11 **minAngleDegree** double GEOM_FADE2D::MeshGenParams::minAngleDegree

Minimum interior angle: Default: 20.0, maximum: 30.0.

6.12.3.12 **minEdgeLength** double GEOM_FADE2D::MeshGenParams::minEdgeLength

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

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

- **MeshGenParams.h**

6.13 **GEOM_FADE2D::MsgBase Class Reference**

**MsgBase**, a base class for message subscriber classes.

#include **<MsgBase.h>**
6.14 GEOM_FADE2D::PeelPredicateTS Class Reference

User-defined peel predicate.
#include <UserPredicates.h>

Public Member Functions

- virtual bool operator() (const Triangle2 *, std::set<Triangle2 *> * pCurrentSet)=0

6.14.1 Detailed Description

This class is the successor of the deprecated (but still valid) UserPredicateT. In contrast to UserPredicateT the operator() receives also a set of current triangles to enable border-edge tests.

See also
https://www.geom.at/mesh-improvements/

The documentation for this class was generated from the following file:
- UserPredicates.h

---

6.13 GEOM_FADE2D::MsgBase Class Reference

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/

6.13.2 Member Function Documentation

6.13.2.1 update()

virtual void GEOM_FADE2D::MsgBase::update (MsgType msgType, const char * 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

---

6.15 GEOM_FADE2D::Point2 Class Reference

Point.
#include <Point2.h>

Public Member Functions

- Point2 ()
  Default constructor.
- Point2 (const double x__, const double y__)
  Constructor.
• **Point2** (const **Point2** &p_)
  Copy constructor.
• **void change** (const double x_, const double y_)
• int **getCustomIndex** () const
  Get the custom index.
• **Triangle2** ∗ **getIncidentTriangle** () const
  Get the associated triangle.
• double **getMaxAbs** () const
  Get max(abs(x),abs(y))
• **bool operator!=** (const **Point2** &p) const
  Inequality operator.
• **Point2 operator+** (const **Vector2** &vec) const
  Add vector and point.
• **Vector2 operator-** (const **Point2** &other) const
  Returns a vector from other to +this.
• **Point2 operator-** (const **Vector2** &vec) const
  Subtract vector from point.
• **bool operator<** (const **Point2** &p) const
  Less than operator.
• **Point2 & operator=** (const **Point2** &other)
• **bool operator==** (const **Point2** &p) const
  Equality operator.
• **bool operator>** (const **Point2** &p) const
  Greater than operator.
• **void set** (const double x_, const double y_, int customIndex_)
  Set the coordinates and customIndex.
• **void set** (const **Point2** &pnt)
  Set the coordiantes.
• **void setCustomIndex** (int customIndex_)
  Set a custom index.
• **void setIncidentTriangle** (**Triangle2** ∗pT)
  Associate a triangle with the point.
• double **x** () const
  Get the x-coordinate.
• **void xy** (double &x_, double &y_) const
  Get the x- and y-coordinate.
• double **y** () const
  Get the y-coordinate.

### Protected Attributes

• double **coordX**
• double **coordY**
• int **customIndex**
• **Triangle2** ∗ **pAssociatedTriangle**

### Friends

• class **Dt2**
• **std::ostream & operator<<** (**std::ostream &stream, const **Point2** &pnt)
  Print to stream.
• **std::istream & operator>>** (**std::istream &stream, **Point2** &pnt)
  Stream-to-Point.
6.15 GEOM_FADE2D::Point2 Class Reference

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

6.15.2 Constructor & Destructor Documentation

6.15.2.1 Point2() [1/3] GEOM_FADE2D::Point2::Point2 {
    const double x_,
    const double y_ } [inline]

Parameters

<table>
<thead>
<tr>
<th>x coordinate</th>
<th>y coordinate</th>
</tr>
</thead>
</table>

6.15.2.2 Point2() [2/3] GEOM_FADE2D::Point2::Point2 () [inline]
The coordinates are initialized to -DBL_MAX

6.15.2.3 Point2() [3/3] GEOM_FADE2D::Point2::Point2 {
    const Point2 & p_ } [inline]
Create a point as a copy of p_. The associated triangle pointer is initialized to NULL

6.15.3 Member Function Documentation

6.15.3.1 getCustomIndex() int GEOM_FADE2D::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/

6.15.3.2 getIncidentTriangle() Triangle2* GEOM_FADE2D::Point2::getIncidentTriangle () const [inline]
Returns
the associated triangle

Generated by Doxygen
### getMaxAbs()
```cpp
double GEOM_FADE2D::Point2::getMaxAbs() const [inline]
```

### operator"!=(
```cpp
bool GEOM_FADE2D::Point2::operator!=(const Point2 & p) const [inline]
```
Note
Although a point has a z-coordinate in the 2.5D version only x and y are compared by this method.

### operator<()
```cpp
bool GEOM_FADE2D::Point2::operator<(const Point2 & p) const [inline]
```
Note
Although a point has a z-coordinate in the 2.5D version only x and y are compared by this method.

### operator==(]
```cpp
bool GEOM_FADE2D::Point2::operator==(const Point2 & p) const [inline]
```
Note
Although a point has a z-coordinate in the 2.5D version only x and y are compared by this method.

### operator>()
```cpp
bool GEOM_FADE2D::Point2::operator>(const Point2 & p) const [inline]
```
Note
Although a point has a z-coordinate in the 2.5D version only x and y are compared by this method.

### set() [1/2]
```cpp
void GEOM_FADE2D::Point2::set(const double x_, const double y_, int customIndex_ = ) [inline]
```
Internal method

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_</td>
<td>x-coordinate</td>
</tr>
<tr>
<td>y_</td>
<td>y-coordinate</td>
</tr>
<tr>
<td>customIndex_</td>
<td>Arbitrary index, use -1 if not required</td>
</tr>
</tbody>
</table>

### set() [2/2]
```cpp
void GEOM_FADE2D::Point2::set(const Point2 & pnt) [inline]
```
Parameters

\textbf{pnt} is the point whose coordinates are assigned to the current point

6.15.3.10 **setCustomIndex()**

```cpp
void GEOM_FADE2D::Point2::setCustomIndex ( int customIndex_ ) [inline]
```

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

\textbf{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

- \texttt{int getCustomIndex()}
- A best practices example that deals with indices: \url{http://www.geom.at/runtime/}

6.15.3.11 **setIncidentTriangle()**

```cpp
void GEOM_FADE2D::Point2::setIncidentTriangle ( Triangle2 * pT ) [inline]
```

Parameters

\textbf{pT} will be associated with the triangle

6.15.3.12 **x()**

```cpp
double GEOM_FADE2D::Point2::x ( ) const [inline]
```

Returns

the x-coordinate

6.15.3.13 **xy()**

```cpp
void GEOM_FADE2D::Point2::xy ( double & x_,
                             double & y_ ) const [inline]
```

Parameters

\textbf{x} ← \textbf{x-coordinate}

\textbf{y} ← \textbf{y-coordinate}

6.15.3.14 **y()**

```cpp
double GEOM_FADE2D::Point2::y ( ) const [inline]
```
Returns
the y-coordinate

The documentation for this class was generated from the following file:
• Point2.h

6.16 GEOM_FADE2D::Segment2 Class Reference

Segment.
#include <Segment2.h>

Public Member Functions
• Segment2 ()
• Segment2 (const Point2 &src_, const Point2 &trg_)
  Create a Segment2.
• double getSqLen2D () const
• Point2 getSrc () const
• Point2 getTrg () const
• bool operator== (const Segment2 &other) const
• void swapSrcTrg ()

Protected Attributes
• Point2 src
• Point2 trg

Friends
• std::ostream & operator<< (std::ostream &stream, Segment2 seg)

6.16.1 Detailed Description

6.16.2 Constructor & Destructor Documentation

6.16.2.1 Segment2() [1/2] GEOM_FADE2D::Segment2::Segment2 (const Point2 & src_,
const Point2 & trg_ )
Parameters
| src←_ | First endpoint (source) |
| trg←_ | Second endpoint (target) |

6.16.2.2 Segment2() [2/2] GEOM_FADE2D::Segment2::Segment2 ( )
Create a Segment2 Default constructor

6.16.3 Member Function Documentation
6.16.3.1 getSqLen2D() double GEOM_FADE2D::Segment2::getSqLen2D ( ) const
Get the squared length

6.16.3.2 getSrc() Point2 GEOM_FADE2D::Segment2::getSrc ( ) const
Get the source point
Returns
  the source point

6.16.3.3 getTrg() Point2 GEOM_FADE2D::Segment2::getTrg ( ) const
Get the target point
Returns
  the target point

6.16.3.4 operator==( ) bool GEOM_FADE2D::Segment2::operator== ( const Segment2 & other ) const
operator==
Undirected equality operator

6.16.3.5 swapSrcTrg() void GEOM_FADE2D::Segment2::swapSrcTrg ( )
Internally swaps the source and target point

The documentation for this class was generated from the following file:
  • Segment2.h

6.17 GEOM_FADE2D::SegmentChecker Class Reference

SegmentChecker identifies intersecting line segments.
#include <SegmentChecker.h>

Public Member Functions

• SegmentChecker (const std::vector< Segment2 * > &vSegments_)
• ClipResult clipSegment (Segment2 &seg)
• void getIllegalSegments (bool bAlsoEndPointIntersections, std::vector< Segment2 * > &vIllegalSegments← Out) const
• int getIndex (Segment2 *pSeg) const
• void getIntersectionPoint (SegmentIntersectionType typ, const Segment2 &seg0, const Segment2 &seg1, Point2 &ispOut) const
• SegmentIntersectionType getIntersectionType (const Segment2 *pSeg1, const Segment2 *pSeg2) const
• const char *getIntersectionTypeString (SegmentIntersectionType sit) const
• void getIntersectors (Segment2 *pTestSegment, bool bAlsoEndPointIntersections, std::vector< std::pair< Segment2 *, SegmentIntersectionType > > &vIntersectorsOut) const
• Bbox2 getLimit ( ) const
• size_t getNumberOfSegments ( ) const
• Segment2 *getSegment (size_t i) const
• void setLimit (const Bbox2 &bbx)
• void showIllegalSegments (bool bAlsoEndPointIntersections, const char *name) const
• void showSegments (const char *name) const
• void subscribe (MsgType msgType, MsgBase *pMsg)
• void unsubscribe (MsgType msgType, MsgBase *pMsg)
6.17.1 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.

See also

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

6.17.2 Constructor & Destructor Documentation

6.17.2.1 SegmentChecker()

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

\[
\begin{array}{|c|c|}
\hline
\text{Segments} & \text{contains the segments to be checked} \\
\hline
\end{array}
\]

6.17.3 Member Function Documentation

6.17.3.1 clipSegment()  
Clip Result GEOM_FADE2D::SegmentChecker::clipSegment ( 
\text{Segment2} \ & \ \text{seg} 
)

Clip Segments

Use this method to limit the length of a line-segment to its intersection with a box. The result can be the whole segment, a subsegment, a degenerate segment or the result can be empty. In the last case the segment is not changed but the method returns CR_EMPTY.

Parameters

\[
\begin{array}{|c|}
\hline
\text{in, out} \ \text{seg} \ \text{is the line segment to be clipped} \\
\hline
\end{array}
\]

Returns

CR_INVALID, CR_EMPTY, CR_CLIPPED_DEGENERATE, CR_CLIPPED_NONDEGENERATE, CR_COMPLETE_DEGENERATE, CR_COMPLETE_NONDEGENERATE or CR_INVALID.

Note

In case that you missed to call setLimit() with a valid bounding box before, the method returns CR_INVALID.;

6.17.3.2 getIllegalSegments()  
void GEOM_FADE2D::SegmentChecker::getIllegalSegments ( 
\text{bool} \ bAlsoEndPointIntersections, 
\text{std::vector< Segment2 * >} \ & \ vIllegalSegmentsOut \ ) \ \text{const}

Get illegal segments

Returns segments which are involved in intersections. Intersections at endpoints are only reported when \text{bAlsoEndPointIntersections} 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

\[
\begin{array}{|c|}
\hline
\text{bAlsoEndPointIntersections} & \text{specifies if intersections at endpoints shall be detected} \\
\hline
\text{out} \ vIllegalSegmentsOut & \text{is the output vector} \\
\hline
\end{array}
\]

6.17.3.3 getIndex()  
int GEOM_FADE2D::SegmentChecker::getIndex ( 
\text{Segment2} \ * \ pSeg \ ) \ \text{const}

Returns the index of a segment

Parameters

\[
\begin{array}{|c|}
\hline
\text{pSeg} & \text{is the segment whose index is returned} \\
\hline
\end{array}
\]

Generated by Doxygen
6.17.3.4 getIntersectionPoint()  
void GEOM_FADE2D::SegmentChecker::getIntersectionPoint (  
    SegmentIntersectionType typ,  
    const Segment2 & seg0,  
    const Segment2 & seg1,  
    Point2 & ispOut ) const

Compute the intersection point of two segments  
Use getIntersectionType() to determine the segment intersection type sit.

Parameters

<table>
<thead>
<tr>
<th>typ</th>
<th>is the intersection type (SIT_POINT or SIT_ENDPOINT for the present method)</th>
</tr>
</thead>
<tbody>
<tr>
<td>seg0, seg1</td>
<td>are the intersecting segments</td>
</tr>
<tr>
<td>out</td>
<td>ispOut</td>
</tr>
<tr>
<td></td>
<td>is the output intersection point.</td>
</tr>
</tbody>
</table>

Note

pSeg1 and pSeg2 do not need to be from the set that has been used to initialize the SegmentChecker.

6.17.3.5 getIntersectionSegment()  
void GEOM_FADE2D::SegmentChecker::getIntersectionSegment (  
    const Segment2 & seg0,  
    const Segment2 & seg1,  
    Segment2 & issOut ) const

Computes the intersection segment of two collinear intersecting segments

Parameters

| seg0, seg1 | are intersecting segments such that their SegmentIntersectionType is SIT_SEGMENT |
| out   | issOut                                                                  |
|       | is the computed intersection of seg0 and seg1                           |

Note

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

6.17.3.6 getIntersectionType()  
SegmentIntersectionType GEOM_FADE2D::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

\( p\text{Seg1} \) and \( p\text{Seg2} \) do not need to be from the set that has been used to initialize the present object.

### 6.17.3.7 `getIntersectionTypeString()`

```cpp
const char* GEOM_FADE2D::SegmentChecker::getIntersectionTypeString(const 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.

### 6.17.3.8 `getIntersectors()`

```cpp
void GEOM_FADE2D::SegmentChecker::getIntersectors(Segment2* pTestSegment, bool bAlsoEndPointIntersections, std::vector<std::pair<Segment2*, SegmentIntersectionType>> & vIntersectors)
```

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pTestSegment</code></td>
<td>is the segment to be analyzed</td>
</tr>
<tr>
<td><code>bAlsoEndPointIntersections</code></td>
<td>specifies if intersections of type SIT_ENDPOINT shall also be reported.</td>
</tr>
<tr>
<td><code>out</code></td>
<td><code>vIntersectorsOut</code> is the output vector. Segments intersecting <code>pTestSegment</code> are added to <code>vIntersectorsOut</code> along with their intersection type.</td>
</tr>
</tbody>
</table>

**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`.

### 6.17.3.9 `getLimit()`

```cpp
Bbox2 GEOM_FADE2D::SegmentChecker::getLimit() const
```

Get Limit

Returns

the bounding box that has been set before using `setLimit()`.

### 6.17.3.10 `getNumberOfSegments()`

```cpp
size_t GEOM_FADE2D::SegmentChecker::getNumberOfSegments() const
```

Returns the number of segments contained in this `SegmentChecker` object.

### 6.17.3.11 `getSegment()`

```cpp
Segment2* GEOM_FADE2D::SegmentChecker::getSegment(size_t i) const
```

Returns the \(i\)-th segment.

**Parameters**

- `i` is the index of the segment to be returned.
6.17.3.12  **setLimit()**  void GEOM_FADE2D::SegmentChecker::setLimit (  
   const Bbox2 & bbx )

Set Limit
Sets the bounding box bbx that is used by clipSegment() to limit the length of a line segment

6.17.3.13  **showIllegalSegments()**  void GEOM_FADE2D::SegmentChecker::showIllegalSegments (  
   bool bAlsoEndPointIntersections,  
   const char * name ) const

Write a postscript file, highlight illegal segments

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bAlsoEndPointIntersections</td>
<td>specifies if intersections at endpoints are also illegal</td>
</tr>
<tr>
<td>name</td>
<td>is the output filename</td>
</tr>
</tbody>
</table>
Figure 19 Visualization of polyline intersections

6.17.3.14 showSegments()  void GEOM_FADE2D::SegmentChecker::showSegments (  
  const char * name ) const  
Write all segments, with and without intersection, to a postscript file

Parameters

| name | is the output filename |

SegmentChecker::showIllegalSegments()
Number of segments: 1009
Legal segments: 983
Illegal segments: 26
6.17.3.15 subscribe()  
```cpp
void GEOM_FADE2D::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

<table>
<thead>
<tr>
<th>msgType</th>
<th>is the message type. For progress information the type is always MSG_PROGRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>pMsg</td>
<td>is a user defined progress bar which derives from Fade's MsgBase.</td>
</tr>
</tbody>
</table>

6.17.3.16 unsubscribe()  
```cpp
void GEOM_FADE2D::SegmentChecker::unsubscribe {
```
Unregister a progress bar object

Parameters

<table>
<thead>
<tr>
<th>msgType</th>
<th>is the message type. For progress information the type is always MSG_PROGRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>pMsg</td>
<td>is a user defined class which derives from Fade's MsgBase</td>
</tr>
</tbody>
</table>

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

- SegmentChecker.h

6.18 GEOM_FADE2D::Triangle2 Class Reference

Triangle.

#include <Triangle2.h>

Public Member Functions

- Triangle2 ()
  Constructor.
- void clearProperties ()
  Clear all corners and neighbor pointers.
- double getArea2D () const
  Get 2D Area.
- Point2 getBarycenter () const
  Get the barycenter of a triangle.
- Point2 * getCorner (const int ith) const
  Get the i-th corner of the triangle.
- std::pair<Point2, bool> getDual (bool bForceExact=false) const
  Get the dual Voronoi vertex.
- double getInteriorAngle2D (int ith) const
  Get interior 2D angle.
- int getIntraTriangleIndex (const Point2 *p) const
  Get the index of p in the triangle.
- int getIntraTriangleIndex (const Point2 *p0, const Point2 *p1) const
  Get the index of (p0,p1)
- int getIntraTriangleIndex (const Triangle2 *pTriangle) const
  Get the neighbor index of pTriangle.
- int getMaxIndex () const
  Get the index of the largest edge.
- double getMaxSqEdgeLen2D () const
  Get the maximum squared 2D edge length.
- int getMinIndex () const
  Get the index of the smallest edge.
- Triangle2 * getOppositeTriangle (const int ith) const
  Get the i-th neighbor triangle.
- double getSquaredEdgeLength2D (int ith) const
  **
- bool hasOnEdge (int i, const Point2 &q) const
  Has point on edge.
- bool hasVertex (const Point2 &vtx) const

Generated by Doxygen
Has vertex.
• bool hasVertex (Point2 *pVtx) const

Has vertex.
• 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 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.

Protected Member Functions

• double computeArea (double l0, double l1, double l2) const
• bool getCC_strategy1 (double avgOffX, double avgOffY, Point2 &cc) const

Protected Attributes

• Triangle2 * aOppTriangles [3]
• Point2 * aVertexPointer [3]

Friends

• std::ostream & operator<< (std::ostream &stream, const Triangle2 &c)
• void registerTriangles (Triangle2 *fromTriangle, int ith, Triangle2 *toTriangle, int jth)

6.18.1 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 21 Indices and neighborships, tb is the 0-th neighbor of ta and ta is the 2nd neighbor of tb.](image)

See also

TriangleAroundVertexIterator to find out how to access all triangles incident to a certain vertex.

6.18.2 Constructor & Destructor Documentation

6.18.2.1 Triangle2()

GEOM_FADE2D::Triangle2::Triangle2 {} [inline]
6.18.3 Member Function Documentation

6.18.3.1 getArea2D()  

double GEOM_FADE2D::Triangle2::getArea2D ( ) const  

Returns the 2D area of the triangle.  
Note: The getArea() method is deprecated and replaced by getArea2D() to keep the names consistent.

6.18.3.2 getBarycenter()  

Point2 GEOM_FADE2D::Triangle2::getBarycenter ( ) const  

Returns  
the barycenter of the triangle.

6.18.3.3 getCorner()  

Point2* GEOM_FADE2D::Triangle2::getCorner (  
const int ith ) const [inline]  

Returns  
a pointer to the i-th corner point of the triangle.

Figure 22 Intra triangle indices are ordered counterclockwise

Parameters  
ith is the intra-triangle-index, ith={0,1,2}.

6.18.3.4 getDual()  

std::pair<Point2,bool> GEOM_FADE2D::Triangle2::getDual (  
bool bForceExact = false ) const  

Returns  
a std::pair<Point2,bool>, where the first component is the circumcenter of the triangle (i.e., the dual Voronoi vertex) and the second component is a boolean value which is normally true to indicate the computation is reliable.

Parameters  
bForceExact can be used to enforce exact computation with multiple-precision arithmetic. By default the computationally more expensive multiple-precision arithmetic is only used for bad-shaped triangles.
Attention

Attention: The circumcenter of a nearly collinear triangle can have coordinates beyond the bounds of floating point arithmetic. Fade could compute the circumcenter with multiple-precision arithmetic but it could not represent the result as a point with floatingpoint coordinates. Thus the involved coordinates are DBL_MAX in such a case and the boolean return value is then false.

Note

Such extreme numeric cases can easily be avoided by insertion of four dummy vertices around the triangulation, e.g., at coordinates 100 times larger than the domain of the data points. This will automatically restrict the Voronoi diagram of the data points to this range.

6.18.3.5 getInteriorAngle2D() double GEOM_FADE2D::Triangle2::getInteriorAngle2D ( 
    int ith ) const

Note: The getArea() method is deprecated and replaced by getInteriorAngle2D() to keep the names consistent.

Returns

the interior 2D angle at the ith vertex

6.18.3.6 getIntraTriangleIndex() [1/3] int GEOM_FADE2D::Triangle2::getIntraTriangleIndex ( 
    const Point2 * p ) const [inline]

Figure 23 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 \(*this\)

6.18.3.7 getIntraTriangleIndex() [2/3] int GEOM_FADE2D::Triangle2::getIntraTriangleIndex ( 
    const Point2 * p0, 
    const Point2 * p1 ) const [inline]

Returns

the index of the edge \((p0,p1)\) in the triangle

Generated by Doxygen
6.18.3.8 *getIntraTriangleIndex()* [3/3]  

```cpp
int GEOM_FADE2D::Triangle2::getIntraTriangleIndex (const Triangle2 *pTriangle) const [inline]
```

Figure 24 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*.

6.18.3.9 *getOppositeTriangle()*

```cpp
Triangle2 * GEOM_FADE2D::Triangle2::getOppositeTriangle (const int ith) const [inline]
```

Returns the *i*-th neighbor triangle, i.e. the one opposite to the *i*-th corner.

Figure 25 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.

6.18.3.10 *getSquaredEdgeLength2D()*

```cpp
double GEOM_FADE2D::Triangle2::getSquaredEdgeLength2D (int ith) const
```

---

Generated by Doxygen
Method for internal use
Squared edge length
Returns the squared length of the \( i \)th edge.

6.18.3.11 hasOnEdge()

```cpp
bool GEOM_FADE2D::Triangle2::hasOnEdge ( 
    int i,
    const Point2 & q ) const
```

Returns

- if \( q \) is exactly on the \( i \)th edge

6.18.3.12 hasVertex() [1/2]

```cpp
bool GEOM_FADE2D::Triangle2::hasVertex ( 
    const Point2 & vtx ) const
```

Returns

- if \( \text{vtx} \) is a corner of the triangle

6.18.3.13 hasVertex() [2/2]

```cpp
bool GEOM_FADE2D::Triangle2::hasVertex ( 
    Point2 * pVtx ) const
```

Returns

- if \( \text{pVtx} \) is a corner of the triangle

6.18.3.14 setOppTriangle()

```cpp
void GEOM_FADE2D::Triangle2::setOppTriangle ( 
    const int ith,
    Triangle2 * pTriangle ) [inline]
```

Parameters

- \( \text{ith} \): is the index of the corner of \( \ast this \) which does not touch \( \text{pTriangle} \)
- \( \text{pTriangle} \): is a pointer to the triangle which shares two corners with \( \ast this \)

Figure 26 Make pTriangle the 0-th neighbor of \( \ast this \)

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

- Triangle2.h
Public Member Functions

- `TriangleAroundVertexIterator (const Point2 *pPnt_)` Constructor.
- `TriangleAroundVertexIterator (const TriangleAroundVertexIterator &it)` Copy constructor.
- `TriangleAroundVertexIterator (Point2 *pPnt_, Triangle2 *pTr_)` Constructor.
- `bool operator!= (const TriangleAroundVertexIterator &rhs)`
  \[ \text{operator\textbar!=()} \]
- `Triangle2 * operator* ()`
  Returns a pointer to the current triangle (or NULL)
- `TriangleAroundVertexIterator & operator++ ()`
  Proceed to the next triangle (the one in counterclockwise order)
- `TriangleAroundVertexIterator & operator-- ()`
  Proceed to the previous triangle (the one in clockwise order)
- `TriangleAroundVertexIterator & operator= (const TriangleAroundVertexIterator &other)`
- `bool operator== (const TriangleAroundVertexIterator &rhs)`
  \[ \text{operator\textbar==()} \]
- `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 * pSavedTr`
- `Triangle2 * pTr`

6.19.1 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.
6.19.2 Constructor & Destructor Documentation

6.19.2.1 TriangleAroundVertexIterator() [1/3]

```cpp
GEOM_FADE2D::TriangleAroundVertexIterator::TriangleAroundVertexIterator ( const Point2 * pPnt_ ) [inline], [explicit]
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pPnt_</code></td>
<td>is the vertex whose incident triangles can be visited with the iterator</td>
</tr>
</tbody>
</table>

Initially the iterator points to an arbitrary triangle (not NULL)

6.19.2.2 TriangleAroundVertexIterator() [2/3]

```cpp
GEOM_FADE2D::TriangleAroundVertexIterator::TriangleAroundVertexIterator ( Point2 * pPnt_, Triangle2 * pTr_ ) [inline]
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pPnt_</code></td>
<td>is the vertex whose incident triangles can be visited with the iterator</td>
</tr>
<tr>
<td><code>pTr_</code></td>
<td>is the triangle the iterator initially points to</td>
</tr>
</tbody>
</table>

6.19.2.3 TriangleAroundVertexIterator() [3/3]

```cpp
GEOM_FADE2D::TriangleAroundVertexIterator::TriangleAroundVertexIterator ( const TriangleAroundVertexIterator & it ) [inline]
```

Copies the iterator `it`

6.19.3 Member Function Documentation

6.19.3.1 operator"!="

```cpp
bool GEOM_FADE2D::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

6.19.3.2 operator*() Triangle2* GEOM_FADE2D::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.

6.19.3.3 operator++() TriangleAroundVertexIterator& GEOM_FADE2D::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.

6.19.3.4 operator--() TriangleAroundVertexIterator& GEOM_FADE2D::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 NULL. Another call to operator--() will set the iterator to the next triangle in clockwise order.

6.19.3.5 operator==() bool GEOM_FADE2D::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

6.19.3.6 previewNextTriangle() Triangle2* GEOM_FADE2D::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.
6.19.3.7 previewPrevTriangle()  

Triangle2* GEOM_FADE2D::TriangleAroundVertexIterator::prevPrevTriangle () [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

6.20 GEOM_FADE2D::UserPredicateT Class Reference

User-defined predicate (deprecated)
#include <UserPredicates.h>

Public Member Functions

- virtual bool operator() (const Triangle2 *)=0

6.20.1 Detailed Description

This class is deprecated in favor of PeelPredicateTS. It is kept for backwards compatibility.
The documentation for this class was generated from the following file:

- UserPredicates.h

6.21 GEOM_FADE2D::Vector2 Class Reference

Vector.
#include <Vector2.h>

Public Member Functions

- Vector2 ()
  Default constructor.
- Vector2 (const double x_, const double y_)
  Constructor.
- Vector2 (const Vector2 &v_)
  Copy constructor.
- bool isDegenerate () const
  isDegenerate
- double length () const
  Get the length of the vector.
- double operator+ (const Vector2 &other) const
  Scalar product.
- Vector2 operator+ (double val) const
  Multiplication.
- Vector2 operator/ (double val) const
  Division.
- Vector2 & operator= (const Vector2 &other)
  Assignment operator.
- void set (const double x_, const double y_)
  null

Generated by Doxygen
Set the values.

- double sqLength () const
  Get the squared length of the vector.
- double x () const
  Get the x-value.
- double y () const
  Get the y-value.

Protected Attributes

- double valX
- double valY

6.21.1 Detailed Description

This class represents a vector in 2D

6.21.2 Constructor & Destructor Documentation

6.21.2.1 Vector2() [1/3]  
GEOM_FADE2D::Vector2::Vector2 ( 
    const double x_,
    const double y_ )

6.21.2.2 Vector2() [2/3]  
GEOM_FADE2D::Vector2::Vector2 ( )
The vector is initialized to (0,0)

6.21.2.3 Vector2() [3/3]  
GEOM_FADE2D::Vector2::Vector2 ( 
    const Vector2 & v_ )
Create a copy of vector v_

6.21.3 Member Function Documentation

6.21.3.1 isDegenerate()  
bool GEOM_FADE2D::Vector2::isDegenerate ( ) const
Returns
true if the vector length is 0, false otherwise.

6.21.3.2 operator*() [1/2]  
double GEOM_FADE2D::Vector2::operator* ( 
    const Vector2 & other ) const
Scalar product

6.21.3.3 operator*() [2/2]  
Vector2 GEOM_FADE2D::Vector2::operator* ( 
    double val ) const
Multiply by a scalar value

6.21.3.4 operator/()  
Vector2 GEOM_FADE2D::Vector2::operator/ ( 
    double val ) const
Divide by a scalar value

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

- Vector2.h

Generated by Doxygen
6.22 GEOM_FADE2D::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 char *filename_)**
  Constructor.
- **void addHeaderLine (const char *s)**
  Add a header line to the visualization.
- **void addObject (const Circle2 &circ, const Color &c)**
  Add a Circle2 object to the visualization.
- **void addObject (const Edge2 &edge, const Color &c)**
  Add an Edge2 object to the visualization.
- **void addObject (const Label &lab, const Color &c)**
  Add a Label object to the visualization.
- **void addObject (const Point2 &pnt, const Color &c)**
  Add a Point2 object to the visualization.
- **void addObject (const Segment2 &seg, const Color &c)**
  Add a Segment2 object 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<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 objects 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<Triangle2*> &vT, const Color &c)**
  Add a vector of Triangle2 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 std::vector<VoroCell2*> &vT, const Color &c)**
  Add a vector of Voronoi Cells to the visualization.
- **void addObject (const VoroCell2 &tri, const Color &c)**
  Add a VoroCell2 object to the visualization.
- **void addObject (VoroCell2 *pVoroCell, const Color &c)**
  Add a Voronoi cell to the visualization.
- **Bbox2 computeRange (bool bWithVoronoi)**
  Compute the range.
- **void writeFile ()**
  Finish and write the postscript file.
Protected Member Functions

- void `changeColor` (const `Color` &c)
- void `changeColor` (float r, float g, float b, float linewidth, bool bFill)
- void `periodicStroke` ()
- double `scaledDouble` (const double &d)
- `Point2` `scaledPoint` (const `Point2` &p)
- void `writeCircle` (const `Point2` &p1_, double radius, bool bFill)
- void `writeFooter` ()
- void `writeHeader` (const char *title)
- void `writeHeaderLines` ()
- void `writeLabel` (Label l)
- void `writeLine` (const `Point2` &pSource, const `Point2` &pTarget)
- void `writePoint` (const `Point2` &p1_, float size)
- 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 `writeVoroCell` (VoroCell2 *pVoroCell, bool bFill, double width)

Protected Attributes

- `Bbox2` `bbox`
- bool `bFileClosed`
- bool `bFill`
- `Color` `lastColor`
- std::ofstream `outFile`
- Dat * `pDat`
- int `updateCtr`
- std::vector<std::pair<Circle2, Color>> `vCircles`
- std::vector<std::pair<Label, Color>> `vLabels`
- std::vector<std::pair<Point2, Color>> `vPoints`
- std::vector<std::pair<Segment2, Color>> `vSegments`
- std::vector<std::pair<Triangle2, Color>> `vTriangles`
- std::vector<std::pair<VoroCell2 *, Color>> `vVoroCells`

6.22.1 Detailed Description

See also

http://www.geom.at/example2-traversing/

![Figure 28 Example output of the Visualizer](image-url)
6.22.2 Constructor & Destructor Documentation

6.22.2.1 Visualizer2()

GEOM_FADE2D::Visualizer2::Visualizer2 (const char * filename_ ) [explicit]

Parameters

filename_ ← _ is the name of the postscript file to be written

6.22.3 Member Function Documentation

6.22.3.1 computeRange()

Bbox2 GEOM_FADE2D::Visualizer2::computeRange (bool bWithVoronoi )

Parameters

bWithVoronoi specifies if the Voronoi cells shall be incorporated.

Returns

a bounding box of currently contained objects

6.22.3.2 writeFile()

void GEOM_FADE2D::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

6.23 GEOM_FADE2D::VoroCell2 Class Reference

Voronoi cell.

#include <VoroCell2.h>

Public Member Functions

• bool getAdjacentVCells (std::vector< VoroCell2 * > &vAdjacentCells) const
• double getArea () const
• bool getBoundaryPoints (std::vector< Point2 > &vPoints, std::vector< Vector2 > *pvInfiniteDirections=NULL) const
• double getCentroid (Point2 &centroid) const
• int getCustomCellIndex () const
• bool getIncidentTriangles (std::vector< Triangle2 * > &vIncTriangles) const
• bool getNeighborSites (std::vector< Point2 > * &vSites) const
• Point2 * getSite () const
• bool getVoronoiVertices (std::vector< VoroVertex2 * > &vVoroVertices) const
• bool isFinite () const
• void setCustomCellIndex (int customCellIndex_) const
6.23.1 Detailed Description

This class represents a Voronoi cell. A Voronoi cell corresponds to a certain site which is also a vertex of the dual Delaunay triangulation.

6.23.2 Member Function Documentation

6.23.2.1 getAdjacentVCells()  

```cpp
bool GEOM_FADE2D::VoroCell2::getAdjacentVCells ( 
    std::vector< VoroCell2 * > & vAdjacentCells ) const
```

Get adjacent Voronoi cells

Parameters

- `vAdjacentCells` is used to return the neighbor cells in counterclockwise order.

Returns

- whether the cell is finite.

6.23.2.2 getArea()  

```cpp
double GEOM_FADE2D::VoroCell2::getArea ( ) const
```

Get the area

Returns

- the area of the cell if it is finite or -1.0 for an infinite cell.

6.23.2.3 getBoundaryPoints()  

```cpp
bool GEOM_FADE2D::VoroCell2::getBoundaryPoints ( 
    std::vector< Point2 > & vPoints, 
    std::vector< Vector2 > * pvInfiniteDirections = NULL ) const
```

Get boundary points

Use this method to retrieve the Voronoi vertices of the present cell as `Point2`.

Parameters

- `out vPoints` contains the boundary points in counterclockwise order.
- `out pvInfiniteDirections` can optionally be used to retrieve the directions of the infinite Voronoi edges: When a pointer to a vector is provided for `pvInfiniteDirections` and the cell is finite, then the first stored `Vector2` describes the direction of the infinite edge at `vPoints[0]` and the second one the direction of the infinite edge at `vPoints.back()`.

Returns

- whether the cell is finite

See also

- `getVoronoiVertices()`

6.23.2.4 getCentroid()  

```cpp
double GEOM_FADE2D::VoroCell2::getCentroid ( 
    Point2 & centroid ) const
```

Get the centroid and area
Parameters

| out | centroid |

Returns

the area of the cell if it is finite OR -1.0 for an infinite cell.

**6.23.2.5** `getCustomCellIndex()`  
```cpp
int GEOM_FADE2D::VoroCell2::getCustomCellIndex ( ) const
```
Get custom cell-index

Returns

the custom cell index that has been set before or -1 when no custom cell index has been set.

See also

`setCustomCellIndex()`

**6.23.2.6** `getIncidentTriangles()`  
```cpp
bool GEOM_FADE2D::VoroCell2::getIncidentTriangles ( 
    std::vector<Triangle2 * > & vIncTriangles ) const
```
Get incident triangles
The site of the present Voronoi cell is also a vertex of the dual Delaunay triangulation.

Parameters

| vIncTriangles | is used to return the the Delaunay triangles incident to the site of the cell in counterclockwise order. |

Returns

whether the cell is finite.

**6.23.2.7** `getNeighborSites()`  
```cpp
bool GEOM_FADE2D::VoroCell2::getNeighborSites ( 
    std::vector<Point2 * > & vSites ) const
```
Get neighbor sites

Parameters

| vSites | is used to return the sites of the adjacent cells in counterclockwise order. |

Returns

whether the cell is finite.

**6.23.2.8** `getSite()`  
```cpp
Point2* GEOM_FADE2D::VoroCell2::getSite ( ) const
```
Get site

Returns

the site of the cell, which is also a vertex of the dual Delaunay triangulation.
6.23.9  getVoronoiVertices() bool GEOM_FADE2D::VoroCell2::getVoronoiVertices (std::vector<VoroVertex2 *> &vVoroVertices) const

Get Voronoi vertices
Used to retrieve the Voronoi vertices of the cell.

Parameters

| out  | vVoroVertices | contains VoroVertex2 objects in counterclockwise order. |

Returns

whether the cell is finite

See also

ggetBoundaryPoints()

6.23.10  isFinite() bool GEOM_FADE2D::VoroCell2::isFinite ( ) const

Is finite cell

Returns

whether the cell is finite

6.23.11  setCustomCellIndex() void GEOM_FADE2D::VoroCell2::setCustomCellIndex (int customCellIndex_)

Set custom cell-index
Use this method to associate Voronoi cells with your own data structures or to assign labels for a visualization.

Parameters

| customCellIndex_ | is an arbitrary integer |

See also

ggetCustomCellIndex()

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

- Voronoi2.h

6.24  GEOM_FADE2D::Voronoi2 Class Reference

Voronoi diagram.
#include <Voronoi2.h>

Public Member Functions

- Voronoi2 * getVoronoiCell (Point2 *pSite)
  Get Voronoi cell.
- void getVoronoiCells (std::vector<VoroCell2 *> &vVoronoiCells)
  Get all Voronoi cells.
- VoroVertex2 * getVoronoiVertex (Triangle2 *pT)
  Get the Voronoi vertex of a triangle.
• bool isValid () const
  Is valid.
• VoroCell2 ∗ locateVoronoiCell (const Point2 &queryPoint)
  Locate a Voronoi Cell.
• void show (const char ∗ filename, bool bVoronoi=true, bool bCellColors=true, bool bSites=true, bool b← Delaunay=true, bool bCellLabels=false)
  Draw the Voronoi diagram.
• void show (Visualizer2 ∗ pVisualizer, bool bVoronoi=true, bool bCellColors=true, bool bSites=true, bool b← Delaunay=true, bool bCellLabels=false)
  Draw the Voronoi diagram.

Protected Attributes
  • Voronoi2Impl ∗ pImpl

6.24.1 Detailed Description
This class represents a Voronoi diagram. A Voronoi diagram is the dual graph of a Delaunay triangulation i.e.,
  • Each Voronoi cell contains the area closest to its site which is a Delaunay vertex
  • Each Voronoi edge has a dual edge in the Delaunay triangulation at an angle of 90 degrees to it.
  • Each Voronoi vertex is the circumcenter of a Delaunay triangle

See also
  https://en.wikipedia.org/wiki/Voronoi_diagram

6.24.2 Member Function Documentation

6.24.2.1 getVoronoiCell() VoroCell2 ∗ GEOM_FADE2D::Voronoi2::getVoronoiCell ( 
          Point2 ∗ pSite )
Use this method to retrieve the Voronoi cell of a specific site.

Parameters
  in  pSite

Returns
  the VoroCell2 of pSite.

6.24.2.2 getVoronoiCells() void GEOM_FADE2D::Voronoi2::getVoronoiCells ( 
           std::vector< VoroCell2 ∗ > & vVoronoiCells )
Use this method to retrieve all finite and infinite Voronoi cells.

Parameters
  out  vVoronoiCells

6.24.2.3 getVoronoiVertex() VoroVertex2 ∗ GEOM_FADE2D::Voronoi2::getVoronoiVertex ( 

Get the Voronoi vertex of a certain dual Delaunay triangle \( p_T \)

**Parameters**

| in | \( p_T \) |

**Returns**

the Voronoi vertex that corresponds to \( p_T \)

### 6.24.4 isValid()

```cpp
bool GEOM_FADE2D::Voronoi2::isValid ( ) const
```

**Returns**

whether the Voronoi diagram is ready for use. This is the case as soon 3 sites exist which are not collinear.

### 6.24.5 locateVoronoiCell()

```cpp
VoroCell2* GEOM_FADE2D::Voronoi2::locateVoronoiCell ( const Point2 & queryPoint )
```

This is a high performance method to locate the Voronoi cell of an arbitrary \( queryPoint \)

**Parameters**

| in | \( queryPoint \) |

**Returns**

the Voronoi cell that contains \( queryPoint \) or NULL if the Voronoi diagram is invalid.

### 6.24.6 show() [1/2]

```cpp
void GEOM_FADE2D::Voronoi2::show ( const char * filename, bool bVoronoi = true, bool bCellColors = true, bool bSites = true, bool bDelaunay = true, bool bCellLabels = false )
```

**Parameters**

| filename | is the output *.ps filename |
| bVoronoi | draw the edges of the Voronoi diagram (default: true) |
| bCellColors | use background colors for the Voronoi cells (default: true) |
| bSites | draw the sites (default: true) |
| bDelaunay | draw the Delaunay triangles (default: true) |
| bCellLabels | show cell labels (or -1 if not assigned) (default: false) |

This method does automatically crop the viewport to twice the range of the sites. Thus very large and infinite cells appear clipped.

### 6.24.7 show() [2/2]

```cpp
void GEOM_FADE2D::Voronoi2::show ( Visualizer2 * pVisualizer,
```

Generated by Doxygen
bool bVoronoi = true,
bool bCellColors = true,
bool bSites = true,
bool bDelaunay = true,
bool bCellLabels = false)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pVisualizer</td>
<td>is the Visualizer2 object to be used</td>
</tr>
<tr>
<td>bVoronoi</td>
<td>draw the edges of the Voronoi diagram (default: true)</td>
</tr>
<tr>
<td>bCellColors</td>
<td>use background colors for the Voronoi cells (default: true)</td>
</tr>
<tr>
<td>bSites</td>
<td>draw the sites (default: true)</td>
</tr>
<tr>
<td>bDelaunay</td>
<td>draw the Delaunay triangles (default: true)</td>
</tr>
<tr>
<td>bCellLabels</td>
<td>show cell labels (or -1 if not assigned) (default: false)</td>
</tr>
</tbody>
</table>

Note

This method only clips infinite cells. But finite cells can also be very large. Call Visualizer2::setLimit() to specify
the range of interest.

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

- Voronoi2.h

6.25 GEOM_FADE2D::VoroVertex2 Class Reference

Voronoi vertex.
#include <VoroVertex2.h>

Public Member Functions

- Triangle2* getDualTriangle() const
  Get dual triangle.
- Point2 getPoint() const
  Get Point.
- bool isAlive() const
  Is alive.

6.25.1 Detailed Description

This class represents a vertex of the Voronoi diagram. A Voronoi vertex is the circumcenter of a certain triangle of
the dual Delaunay triangulation.

6.25.2 Member Function Documentation

6.25.2.1 getDualTriangle() Triangle2* GEOM_FADE2D::VoroVertex2::getDualTriangle ( ) const
A Voronoi vertex is the circumcenter of a certain triangle in the dual Delaunay triangulation.

Returns
the corresponding Delaunay triangle
6.26.2.2 **getPoint()**  

`Point2 GEOM_FADE2D::VoroVertex2::getPoint () const`  

Returns  

the Voronoi vertex as a `Point2`  

6.25.2.3 **isAlive()**  

`bool GEOM_FADE2D::VoroVertex2::isAlive () const`  

The Voronoi diagram changes dynamically when points are inserted or removed from the dual Delaunay triangulation.  

Returns  

whether the present Voronoi vertex is still valid  

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

- `VoroVertex2.h`  

6.26 **GEOM_FADE2D::Zone2 Class Reference**  

`Zone2` is a certain defined area of a triangulation.  

`#include <Zone2.h>`  

**Public Member Functions**  

- `Zone2 * convertToBoundedZone ()`  
  
  Convert a zone to a bounded zone.  
- `void debug (const char *name="")`  
  
  Development function.  
- `double getArea2D () const`  
  
  Get 2D Area.  
- `void getBorderEdges (std::vector< Edge2 > &vBorderEdgesOut) const`  
  
  Get border edges.  
- `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.  
- `Bbox2 getBoundingBox () const`  
  
  Compute the bounding box.  
- `ConstraintGraph2 * getConstraintGraph () const`  
  
  Get the associated constraint.  
- `void getConstraintGraphs (std::vector< ConstraintGraph2 * > &vConstraintGraphs_) const`  
  
  Get the associated constraint graphs.  
- `size_t getNumberOfTriangles () const`  
  
  Get the number of triangles.  
- `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.  
- `ZoneLocation getZoneLocation () const`  
  
  Get the zone location.  
- `size_t numberOfConstraintGraphs () const`  
  
  Get a the number of `ConstraintGraph2` objects.  
- `bool save (const char *filename)`  
  
  Save the zone.
• bool save (std::ostream &stream)
  Save the zone.

• void show (const char *postscriptFilename, bool bShowFull, bool bWithConstraints) const
  Postscript visualization.

• void show (Visualizer2 *pVisualizer, bool bShowFull, bool bWithConstraints) const
  Postscript visualization.

• void statistics (const char *s) const
  Register a message receiver.

• void unsubscribe (MsgType msgType, MsgBase *pMsg)
  Unregister a message receiver.

• void writeObj (const char *outFilename) const
  Write the zone to *.obj Writes the triangles of the present Zone2 to an *.obj file (The *.obj format represents a 3D scene).

Friends

• Zone2 * peelOffIf (Zone2 *pZone, bool bAvoidSplit, PeelPredicateTS *pPredicate)
  Peel off border triangles.

• Zone2 * peelOffIf (Zone2 *pZone, UserPredicateT *pPredicate, bool bVerbose)
  Peel off border triangles (deprecated)

• Zone2 * zoneDifference (Zone2 *pZone0, Zone2 *pZone1)
  Compute the difference of two zones.

• Zone2 * zoneIntersection (Zone2 *pZone0, Zone2 *pZone1)
  Compute the intersection of two zones.

• Zone2 * zoneSymmetricDifference (Zone2 *pZone0, Zone2 *pZone1)
  Compute the symmetric difference of two zones.

• Zone2 * zoneUnion (Zone2 *pZone0, Zone2 *pZone1)
  Compute the union of two zones.

6.26.1 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

6.26.2 Member Function Documentation

6.26.2.1 convertToBoundedZone() Zone2* GEOM_FADE2D::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 a pointer to the present zone if this->getZoneLocation() is ZL_INSIDE.
6.26.2.2 **getArea2D()**

```cpp
double GEOM_FADE2D::Zone2::getArea2D ( ) const
```

Returns the 2D area of the zone.

Note: The getArea() method is deprecated and replaced by `getArea2D()` to keep the names consistent.

6.26.2.3 **getBorderEdges()**

```cpp
void GEOM_FADE2D::Zone2::getBorderEdges ( 
    std::vectorEDGE2 > & vBorderEdgesOut ) const
```

Returns:
- the CCW oriented border edges of the zone

6.26.2.4 **getConstraintGraph()**

```cpp
ConstraintGraph2* GEOM_FADE2D::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.

6.26.2.5 **getConstraintGraphs()**

```cpp
void GEOM_FADE2D::Zone2::getConstraintGraphs ( 
    std::vectorConstraintGraph2 * > & vConstraintGraphs_ ) const
```

6.26.2.6 **getNumberOfTriangles()**

```cpp
size_t GEOM_FADE2D::Zone2::getNumberOfTriangles ( ) const
```

Warning:
- This method is fast but O(n), so don't call it frequently in a loop.

6.26.2.7 **getTriangles()**

```cpp
void GEOM_FADE2D::Zone2::getTriangles ( 
    std::vectorTriangle2 * > & 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.

6.26.2.8 **getVertices()**

```cpp
void GEOM_FADE2D::Zone2::getVertices ( 
    std::vectorPoint2 * > & vVertices_ ) const
```

6.26.2.9 **getZoneLocation()**

```cpp
ZoneLocation GEOM_FADE2D::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
6.26.2.10  numberOfConstraintGraphs()  

```cpp
size_t GEOM_FADE2D::Zone2::numberOfConstraintGraphs() const
```

A Zone2 object might be defined by zero, one or more ConstraintGraph2 objects.

6.26.2.11  save()  [1/2]  

```cpp
bool GEOM_FADE2D::Zone2::save(
    const char* filename)
```

This command saves the present Zone2 to a binary file. Any constraint edges and custom indices in the domain are retained.

**Parameters**

- `filename` is the output filename

**Returns**

- whether the operation was successful

**Note**

A Delaunay triangulation is convex without holes but this may not hold for the zone to be saved. Thus extra triangles may be saved to fill concavities. These extra-triangles will belong to the Fade_2D instance but not to the Zone2 object when reloaded.

**See also**

`save(std::ostream& stream)`. Use the similar command `Fade_2D::saveZones(const char* file, std::vector<Zone2*>& vZones)` to store more than just one zone. Use `Fade_2D::saveTriangulation()` to store all triangles of the triangulation plus any specified zones. Use `Fade_2D::load()` to reload the data from such files.

6.26.2.12  save()  [2/2]  

```cpp
bool GEOM_FADE2D::Zone2::save(
    std::ostream& stream)
```

This command saves the present Zone2 to an ostream. Any constraint edges and custom indices in the domain are retained.
Parameters

| stream | is the output stream |

Returns

whether the operation was successful

Note

A Delaunay triangulation is convex without holes but this may not hold for the zone to be saved. Thus extra triangles may be saved to fill concavities. These extra-triangles will belong to the Fade_2D instance but not to the Zone2 object when reloaded.

See also

Use the similar command Fade_2D::saveZones(const char* file, std::vector<Zone2*>& vZones) to store more than just one zone. Use Fade_2D::saveTriangulation() to store all triangles of the triangulation plus any specified zones. Use Fade_2D::load() to reload the data from such files.

6.26.2.13 show() [1/2] void GEOM_FADE2D::Zone2::show (const char* postscriptFilename, bool bShowFull, bool bWithConstraints) const

Parameters

<table>
<thead>
<tr>
<th>postscriptFilename</th>
<th>is the name of the output file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>bShowFull</td>
<td>specifies if only the zone or the full triangulation shall be drawn</td>
</tr>
<tr>
<td>bWithConstraints</td>
<td>specifies if constraint edges shall be drawn</td>
</tr>
</tbody>
</table>

6.26.2.14 show() [2/2] void GEOM_FADE2D::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 |
6.26.2.15 **statistics()** void GEOM_FADE2D::Zone2::statistics

    const char * s ) const

Statistics
Prints statistics to stdout.

6.26.2.16 **subscribe()** void GEOM_FADE2D::Zone2::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 |

6.26.2.17 **unifyGrid()** void GEOM_FADE2D::Zone2::unifyGrid

    ( double tolerance )

Unify Grid
A Delaunay triangulation not unique when when 2 or more triangles share a common circumcircle. As a conse-
quence 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 \( \text{error} = \frac{|\text{diagonalA} - \text{diagonalB}|}{\max(\text{diagonalA},\text{diagonalB})} \)
          | 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. |

6.26.2.18 **unsubscribe()** void GEOM_FADE2D::Zone2::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 |

6.26.2.19 **writeObj()** void GEOM_FADE2D::Zone2::writeObj

    ( const char * outFilename ) const

Parameters

| outFilename | is the output filename |
6.26.3 Friends And Related Function Documentation

6.26.3.1 peelOffIf [1/2]

```cpp
Zone2* peelOffIf ( 
    Zone2 * pZone, 
    bool bAvoidSplit, 
    PeelPredicateTS * pPredicate ) [friend]
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pZone</code></td>
<td>is the input zone</td>
</tr>
<tr>
<td><code>bAvoidSplit</code></td>
<td>if true, then the algorithm removes a triangle only if it does not break the zone into independent components.</td>
</tr>
<tr>
<td><code>pPredicate</code></td>
<td>is a user-defined predicate that decides if a triangle shall be removed.</td>
</tr>
</tbody>
</table>

### Returns

A new zone containing a subset of the triangles of `pZone` or NULL when no triangles remain.

### Attention

Check whether NULL is returned!

6.26.3.2 peelOffIf [2/2]

```cpp
Zone2* peelOffIf ( 
    Zone2 * pZone, 
    UserPredicateT * pPredicate, 
    bool bVerbose ) [friend]
```

This function is DEPRECATED but kept for backwards compatibility. The new and better function is:

```cpp
peelOffIf(Zone2* pZone, bool bAvoidSplit,PeelPredicateTS* pPredicate)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pZone</code></td>
<td>is the input zone</td>
</tr>
<tr>
<td><code>pPredicate</code></td>
<td>is a user-defined predicate that decides if a triangle shall be removed.</td>
</tr>
<tr>
<td><code>bVerbose</code></td>
<td></td>
</tr>
</tbody>
</table>

### Returns

A new zone containing a subset of the triangles of `pZone` or NULL when no triangles remain.

6.26.3.3 zoneDifference

```cpp
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.
6.26.3.4  zoneIntersection  

```cpp
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.

6.26.3.5  zoneSymmetricDifference  

```cpp
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.

6.26.3.6  zoneUnion  

```cpp
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  Bbox2.h File Reference

```cpp
#include "Segment2.h"
#include "common.h"
```

Classes

- class GEOM_FADE2D::Bbox2

  `Bbox2` is an axis aligned 2D bounding box.

Functions

- Bbox2 GEOM_FADE2D::getBox (std::vector<Point2>& &vP)  
  Compute the bounding box.

- Bbox2 GEOM_FADE2D::getBox (std::vector<Point2>& &vP)  
  Compute the bounding box.

- std::ostream & GEOM_FADE2D::operator<< (std::ostream &stream, const Bbox2 &pC)  
  Print the box.
7.2 Circle2.h File Reference

7.1 Function Documentation

7.1.1 getBox() [1/2]

```cpp
Bbox2 GEOM_FADE2D::getBox {
    std::vector<Point2*> & vP
}
```

Computes the bounding box of points

7.1.1.2 getBox() [2/2]

```cpp
Bbox2 GEOM_FADE2D::getBox {
    std::vector<Point2> & vP
}
```

Computes the bounding box of points

7.1.1.3 operator<<()

```cpp
std::ostream & GEOM_FADE2D::operator<< {
    std::ostream & stream,
    const Bbox2 & pC ) [inline]
```

Prints the box coordinates to `stream`

7.2 Circle2.h File Reference

```cpp
#include "Point2.h"
#include "common.h"
```

Classes

- class GEOM_FADE2D::Circle2
  
  `Circle for visualization`

7.3 Color.h File Reference

```cpp
#include "common.h"
```

Classes

- class GEOM_FADE2D::Color
  
  `Color for visualization`

Enumerations

- enum GEOM_FADE2D::Colorname {
    CRED, CGREEN, CBLUE, CBLACK,
    CPINK, CGRAY, CORANGE, CLIGHTBLUE,
    CLIGHTBROWN, CDARKBROWN, CPUPLRE, COLIVE,
    CLAWNGREEN, CPALEGREEN, CCYAN, CYELLOW,
    CWHITE
}

  `Predefined colors for convenience`

Functions

- std::ostream & GEOM_FADE2D::operator<< (std::ostream &stream, const Color &c)

7.4 ConstraintGraph2.h File Reference

```cpp
#include "Segment2.h"
#include "ConstraintSegment2.h"
#include "Edge2.h"
#include <map>
```

Generated by Doxygen
#include "common.h"

**Classes**

- class GEOM_FADE2D::ConstraintGraph2
  
  *ConstraintGraph2 is a set of Constraint Edges (ConstraintSegment2)*

### 7.5 ConstraintSegment2.h File Reference

```cpp
#include <set>
#include "common.h"
```

**Classes**

- class GEOM_FADE2D::ConstraintSegment2
  
  A *ConstraintSegment2* represents a Constraint Edge.

**Enumerations**

- enum GEOM_FADE2D::ConstraintInsertionStrategy { CIS_CONFORMING_DELAUNAY =0, GEOM_FADE2D::CIS_CONSTRAINED_DELAUNAY =1, GEOM_FADE2D::CIS_KEEP_DELAUNAY =0, GEOM_FADE2D::CIS_IGNORE_DELAUNAY =1 }

*Constraint Insertion Strategy determines how a constraint edge shall be inserted:*

#### 7.5.1 Enumeration Type Documentation

**7.5.1.1 ConstraintInsertionStrategy** enum GEOM_FADE2D::ConstraintInsertionStrategy

- CIS_CONFORMING_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_25←D/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**

<table>
<thead>
<tr>
<th>CIS_CONFORMING_DELAUNAY</th>
<th>Deprecated.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIS_KEEP_DELAUNAY</td>
<td>Deprecated name.</td>
</tr>
<tr>
<td>CIS_IGNORE_DELAUNAY</td>
<td>Deprecated.</td>
</tr>
</tbody>
</table>

### 7.6 Edge2.h File Reference

```cpp
#include "common.h"
```

**Classes**

- class GEOM_FADE2D::Edge2
7.7 Edge2.h File Reference

Edge2 is a directed edge.

- struct GEOM_FADE2D::Func_gtEdge2D
  Functor to sort edges by 2d length (descending)
- struct GEOM_FADE2D::Func_ltEdge2D
  Functor to sort edges by 2d length (ascending)

7.7 Fade_2D.h File Reference

```c
#include "common.h"
#include "Point2.h"
#include "Triangle2.h"
#include "TriangleAroundVertexIterator.h"
#include "Visualizer2.h"
#include "Zone2.h"
#include "ConstraintGraph2.h"
#include "Performance.h"
#include "MeshGenParams.h"
#include "MsgBase.h"
#include "SegmentChecker.h"
#include "testDataGenerators.h"
#include "freeFunctions.h"
#include "FadeExport.h"
#include "Voronoi2.h"
#include "License.h"
```

Classes

- class GEOM_FADE2D::Fade_2D
  Fade_2D is the Delaunay triangulation main class.

7.7.1 Detailed Description

Fade_2D.h is the main API of the Fade library

7.8 FadeExport.h File Reference

```c
#include <vector>
#include <algorithm>
#include "common.h"
#include "License.h"
```

Classes

- struct GEOM_FADE2D::FadeExport
  FadeExport is a simple struct to export triangulation data.

7.9 freeFunctions.h File Reference

```c
#include "Point2.h"
#include "Segment2.h"
#include "Edge2.h"
#include <vector>
```
Functions

- void GEOM_FADE2D::edgesToPolygons (std::vector<Edge2> &vEdgesIn, std::vector<Edge2> &vvPolygonsOut, std::vector<Edge2> &vRemainingOut)
  
  Create polygons from a set of edges.

- bool GEOM_FADE2D::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)

- double GEOM_FADE2D::getArea2D (Point2 *p0, Point2 *p1, Point2 *p2)
  
  Get 2D area of a triangle.

- void GEOM_FADE2D::getBorders (const std::vector<Triangle2 *> &vT, std::vector<Segment2> &vBorderSegmentsOut)
  
  Get Borders.

- void GEOM_FADE2D::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).

- const char * GEOM_FADE2D::getFade2DVersion ()
  
  Get the Fade2D version string.

- FUNC_DECLSPEC int GEOM_FADE2D::getMajorVersionNumber ()
  
  Get the major version number.

- FUNC_DECLSPEC int GEOM_FADE2D::getMinorVersionNumber ()
  
  Get the minor version number.

- FUNC_DECLSPEC Orientation2 GEOM_FADE2D::getOrientation2 (const Point2 *p0, const Point2 *p1, const Point2 *p2)
  
  Get the orientation of three points.

- FUNC_DECLSPEC Orientation2 GEOM_FADE2D::getOrientation2_mt (const Point2 *p0, const Point2 *p1, const Point2 *p2)
  
  Get Orientation2 (MT)

- FUNC_DECLSPEC int GEOM_FADE2D::getRevisionNumber ()
  
  Get the revision version number.

- void GEOM_FADE2D::getUndirectedEdges (std::vector<Triangle2 *> &vT, std::vector<Edge2> &vUndirectedEdgesOut)
  
  Get undirected edges.

- FUNC_DECLSPEC bool GEOM_FADE2D::isRelease ()
  
  Check if a RELEASE or a DEBUG version is used.

- bool GEOM_FADE2D::isSimplePolygon (std::vector<Segment2> &vSegments)
  
  isSimplePolygon

- void GEOM_FADE2D::pointsToPolyline (std::vector<Point2> &vInPoints, bool bClose, std::vector<Segment2> &vOutSegments)
  
  Points-to-Polyline.

- bool GEOM_FADE2D::readPointsBIN (const char *filename, std::vector<Point2> &vPointsIn)
  
  Read points from a binary file.

- bool GEOM_FADE2D::readSegmentsBIN (const char *filename, std::vector<Segment2> &vSegmentsOut)
  
  Read segments from a binary file.

- FUNC_DECLSPEC bool GEOM_FADE2D::readXY (const char *filename, std::vector<Point2> &vPointsOut)
  
  Read (x y) points.

- bool GEOM_FADE2D::sortRing (std::vector<Segment2> &vRing)
  
  Sort a vector of Segments.

- bool GEOM_FADE2D::sortRingCCW (std::vector<Segment2> &vRing)
  
  Sort a vector of Segments.
7.10 Label.h File Reference

```cpp
#include "Point2.h"
#include "common.h"
```

**Classes**

- class GEOM_FADE2D::Label

  Label is a Text-Label for Visualization.

7.11 MeshGenParams.h File Reference

```cpp
#include "common.h"
#include "Zone2.h"
```

**Classes**

- class GEOM_FADE2D::MeshGenParams

  Parameters for the mesh generator.

**Functions**

- template<typename T>
  void GEOM_FADE2D::unusedParameter (const T &)

  Unused parameter.

7.11.1 Function Documentation

7.11.1.1 unusedParameter()  template<typename T>

```cpp
void GEOM_FADE2D::unusedParameter (const T & ) [inline]
```

Empty template to avoid compiler warnings about unused function parameters

7.12 MsgBase.h File Reference

```cpp
#include "common.h"
```
Classes

- class GEOM_FADE2D::MsgBase

  *MsgBase*, a base class for message subscriber classes.

Enumerations

- enum MsgType { MSG_PROGRESS, MSG_WARNING }

### 7.13 Performance.h File Reference

```c
#include "common.h"
```

Functions

- double GEOM_FADE2D::timer (const char *cstr)
  
  *Timer*.

#### 7.13.1 Function Documentation

#### 7.13.1.1 timer()

```c
double GEOM_FADE2D::timer (const char *cstr )
```

Call the timer function with a certain string to start time measurement. Call it a second time with the same string to finish the time measurement.

Returns

-1 when the timer is started or the elapsed time in seconds when the timer is stopped.

### 7.14 Point2.h File Reference

```c
#include "common.h"
#include "Vector2.h"
```

Classes

- class GEOM_FADE2D::Point2

  *Point*.

Functions

- Point2 GEOM_FADE2D::center (const Point2 &p0, const Point2 &p1)

  Compute the midpoint of p0 and p1.

- Point2 GEOM_FADE2D::centerWithShift (const Point2 &p0, const Point2 &p1)

  Compute the midpoint of p0 and p1 and adapt it.

- std::ostream & GEOM_FADE2D::operator<< (std::ostream &stream, const Point2 &pnt)

  Print to stream.

- std::istream & GEOM_FADE2D::operator>> (std::istream &stream, Point2 &pnt)

  Stream-to-Point.

- double GEOM_FADE2D::sqDistance2D (const double x0, const double y0, const Point2 &p1)

  Get the squared distance between two points in 2D.

- double GEOM_FADE2D::sqDistance2D (const Point2 &p0, const Point2 &p1)

  Get the squared distance between two points in 2D.
7.14.1 Function Documentation

7.14.1.1 center() Point2 GEOM_FADE2D::center (const Point2 & p0, const Point2 & p1) [inline]

Note: The exact midpoint of p0 and p1 may not exist in floating point numbers. Thus the returned point may not be collinear with p0 and p1.

7.14.1.2 centerWithShift() Point2 GEOM_FADE2D::centerWithShift (const Point2 & p0, const Point2 & p1)

Experimental new function that may change in the future. Thought for specific applications. This function works like center() but additionally it adapts the midpoint to the segment (p0,p1) such that it is 'as collinear as possible' with p0 and p1 in the x/y plane. Bounds for the shift are 0.01 and 1 % of the range in x- and y-direction.

7.14.1.3 sqDistance2D() [1/2] double GEOM_FADE2D::sqDistance2D (const double x0, const double y0, const Point2 & p1) [inline]

7.14.1.4 sqDistance2D() [2/2] double GEOM_FADE2D::sqDistance2D (const Point2 & p0, const Point2 & p1) [inline]

7.15 Segment2.h File Reference

#include "Point2.h"
#include "common.h"

Classes

• class GEOM_FADE2D::Segment2
  Segment.

7.16 SegmentChecker.h File Reference

#include <map>
#include "common.h"
#include "Segment2.h"
#include "MsgBase.h"
#include "Bbox2.h"

Classes

• class GEOM_FADE2D::SegmentChecker
  SegmentChecker identifies intersecting line segments.
Enumerations

- enum **ClipResult** {
  CR_INVALID, CR_EMPTY, CR_CLIPPED_DEGENERATE, CR_CLIPPED_NONDEGENERATE, 
  CR_COMPLETE_DEGENERATE, CR_COMPLETE_NONDEGENERATE }

- enum **SegmentIntersectionType** {
  SIT_UNINITIALIZED, SIT_NONE, SIT_SEGMENT, SIT_POINT, 
  SIT_ENDPOINT }

7.16.1 Enumeration Type Documentation

7.16.1.1 ClipResult  enum **ClipResult**

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR_INVALID</td>
<td>Can't compute a result, call setLimit() with a valid Bbox2 before!</td>
</tr>
<tr>
<td>CR_EMPTY</td>
<td>The result is empty (input completely outside the box)</td>
</tr>
<tr>
<td>CR_CLIPPED_DEGENERATE</td>
<td>The result has been clipped and is degenerate</td>
</tr>
<tr>
<td>CR_CLIPPED_NONDEGENERATE</td>
<td>The result has been clipped and is non-degenerate</td>
</tr>
<tr>
<td>CR_COMPLETE_DEGENERATE</td>
<td>The result is unclipped and degenerate (because the segment was already degenerate)</td>
</tr>
<tr>
<td>CR_COMPLETE_NONDEGENERATE</td>
<td>The result is unclipped and non-degenerate</td>
</tr>
</tbody>
</table>

7.16.1.2 SegmentIntersectionType  enum **SegmentIntersectionType**

The Segment intersection type enumerates the way two line segments intersect each other

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIT_UNINITIALIZED</td>
<td>Invalid value</td>
</tr>
<tr>
<td>SIT_NONE</td>
<td>No intersection</td>
</tr>
<tr>
<td>SIT_SEGMENT</td>
<td>The intersection is a non-degenerate segment (collinear intersection)</td>
</tr>
<tr>
<td>SIT_POINT</td>
<td>The intersection is a single point different from the endpoints</td>
</tr>
<tr>
<td>SIT_ENDPOINT</td>
<td>The two segments share a common endpoint which is the only intersection</td>
</tr>
</tbody>
</table>

7.17 testDataGenerators.h File Reference

```cpp
#include "Point2.h"
#include "Segment2.h"
#include <vector>
```

Functions

- FUNC DECLSPEC void **GEOM_FADE2D::generateCircle** (int numPoints, double x, double y, double radiusX, double radiusY, std::vector<Point2> &vCirclePointsOut)  
  Generate a circle.

- FUNC DECLSPEC void **GEOM_FADE2D::generateRandomNumbers** (size_t num, double min, double max, std::vector<double> &vRandomNumbersOut, unsigned int seed=0)  
  Generate random numbers.

- FUNC DECLSPEC void **GEOM_FADE2D::generateRandomPoints** (size_t numRandomPoints, double min, double max, std::vector<Point2> &vRandomPointsOut, unsigned int seed=0)
Generate random points.

- FUNC_DECLSPEC void GEOM_FADE2D::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_FADE2D::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_FADE2D::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_FADE2D::shear (std::vector< Point2 > &vPointsInOut, double shearX, double shearY)

7.18 Triangle2.h File Reference

```cpp
#include "Point2.h"
#include "common.h"
```

Classes

- class GEOM_FADE2D::Triangle2

  Triangle.

7.19 TriangleAroundVertexIterator.h File Reference

```cpp
#include "common.h"
#include "Point2.h"
#include "Triangle2.h"
```

Classes

- class GEOM_FADE2D::TriangleAroundVertexIterator

  Iterator for all triangles around a given vertex.

7.20 UserPredicates.h File Reference

```cpp
#include "common.h"
#include "Triangle2.h"
```

Classes

- class GEOM_FADE2D::PeelPredicateTS

  User-defined peel predicate.

- class GEOM_FADE2D::UserPredicateT

  User-defined predicate (deprecated)

7.21 Vector2.h File Reference

```cpp
#include "common.h"
```
Classes

- class GEOM_FADE2D::Vector2
  
  Vector.

Functions

- Vector2 GEOM_FADE2D::normalize (const Vector2 &other)
  
  Normalize a vector.

- Vector2 GEOM_FADE2D::operator* (double d, const Vector2 &vec)
  
  Multiplication with a scalar.

- Vector2 GEOM_FADE2D::operator+ (const Vector2 &vec0, const Vector2 &vec1)
  
  Addition.

- Vector2 GEOM_FADE2D::operator- (const Vector2 &in)
  
  Opposite vector.

- Vector2 GEOM_FADE2D::operator- (const Vector2 &vec0, const Vector2 &vec1)
  
  Subtraction.

- std::ostream & GEOM_FADE2D::operator<< (std::ostream &stream, const Vector2 &vec)
  
  Print to stream.

7.21.1 Function Documentation

7.21.1.1 operator*() Vector2 GEOM_FADE2D::operator* (double d, const Vector2 &vec) [inline]

Multiplication with a scalar

7.21.1.2 operator+() Vector2 GEOM_FADE2D::operator+ (const Vector2 &vec0, const Vector2 &vec1) [inline]

Addition

7.21.1.3 operator-() [1/2] Vector2 GEOM_FADE2D::operator- (const Vector2 &in) [inline]

Returns

  a vector that points in the opposite direction

7.21.1.4 operator-() [2/2] Vector2 GEOM_FADE2D::operator- (const Vector2 &vec0, const Vector2 &vec1) [inline]

Subtraction

7.21.1.5 operator<<() std::ostream & GEOM_FADE2D::operator<< (std::ostream &stream, const Vector2 &vec) [inline]

Print to stream

Generated by Doxygen
# Visualizer2.h File Reference

#include "Point2.h"
#include "Circle2.h"
#include "Segment2.h"
#include "Color.h"
#include "Label.h"
#include "Bbox2.h"
#include "Edge2.h"
#include "common.h"

**Classes**

- class **GEOM_FADE2D::Visualizer2**  
  
  Visualizer2 is a general Postscript writer. It draws the objects Point2, Segment2, Triangle2, Circle2 and Label.

## VoroCell2.h File Reference

#include "common.h"
#include "Point2.h"
#include "Bbox2.h"

**Classes**

- class **GEOM_FADE2D::VoroCell2**  
  
  Voronoi cell.

### Detailed Description

Voronoi cell

## Voronoi2.h File Reference

#include "common.h"
#include "VoroVertex2.h"
#include "VoroCell2.h"

**Classes**

- class **GEOM_FADE2D::Voronoi2**  
  
  Voronoi diagram.

### Detailed Description

Voronoi diagram

## VoroVertex2.h File Reference

#include "common.h"
#include "Point2.h"

**Classes**

- class **GEOM_FADE2D::VoroVertex2**  
  
  Voronoi vertex.
7.25.1 Detailed Description

Voronoi vertex

7.26 Zone2.h File Reference

```cpp
#include "common.h"
#include "Bbox2.h"
#include "Edge2.h"
#include "Segment2.h"
#include "UserPredicates.h"
#include "MsgBase.h"
```

Classes

- class GEOM_FADE2D::Zone2
  
  Zone2 is a certain defined area of a triangulation.

Enumerations

- enum GEOM_FADE2D::OptimizationMode { GEOM_FADE2D::OPTMODE_STANDARD, GEOM_FADE2D::OPTMODE_BETTER, GEOM_FADE2D::OPTMODE_BEST }

Functions

- Zone2 * GEOM_FADE2D::zoneDifference (Zone2 *pZone0, Zone2 *pZone1)
  
  Compute the difference of two zones.

- Zone2 * GEOM_FADE2D::zoneIntersection (Zone2 *pZone0, Zone2 *pZone1)
  
  Compute the intersection of two zones.

- Zone2 * GEOM_FADE2D::zoneSymmetricDifference (Zone2 *pZone0, Zone2 *pZone1)
  
  Compute the symmetric difference of two zones.

- Zone2 * GEOM_FADE2D::zoneUnion (Zone2 *pZone0, Zone2 *pZone1)
  
  Compute the union of two zones.

7.26.1 Enumeration Type Documentation

7.26.1.1 OptimizationMode

Enumerates the possible modes for Valley/Ridge optimization through Zone2::slopeValleyRidgeOptimization().

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTMODE_STANDARD</td>
<td>Fastest optimization mode.</td>
</tr>
<tr>
<td>OPTMODE_BETTER</td>
<td>Considerably better quality and still fast.</td>
</tr>
<tr>
<td>OPTMODE_BEST</td>
<td>Best quality but quite time consuming.</td>
</tr>
</tbody>
</table>

7.26.2 Function Documentation

7.26.2.1 zoneDifference() Zone2* GEOM_FADE2D::zoneDifference ( Zone2 * pZone0, Zone2 * pZone1 )
Returns

a new zone containing the triangles of \texttt{*pZone0} minus the ones of \texttt{*pZone1}

Note

\texttt{pZone0} and \texttt{pZone1} must belong to the same \texttt{Fade_2D} object.

\subsection{7.26.2.2 zoneIntersection()}

\begin{verbatim}
Zone2* GEOM_FADE2D::zoneIntersection (  
  Zone2 * pZone0,  
  Zone2 * pZone1 )
\end{verbatim}

Returns

a new zone containing the intersection of \texttt{*pZone0} and \texttt{*pZone1}

Note

\texttt{pZone0} and \texttt{pZone1} must belong to the same \texttt{Fade_2D} object.

\subsection{7.26.2.3 zoneSymmetricDifference()}

\begin{verbatim}
Zone2* GEOM_FADE2D::zoneSymmetricDifference (  
  Zone2 * pZone0,  
  Zone2 * pZone1 )
\end{verbatim}

Returns

a new zone containing the triangles that are present in one of the zones but not in the other one.

Note

\texttt{pZone0} and \texttt{pZone1} must belong to the same \texttt{Fade_2D} object.

\subsection{7.26.2.4 zoneUnion()}

\begin{verbatim}
Zone2* GEOM_FADE2D::zoneUnion (  
  Zone2 * pZone0,  
  Zone2 * pZone1 )
\end{verbatim}

Returns

a new zone containing the union of the triangles of \texttt{*pZone0} and \texttt{*pZone1}

Note

\texttt{pZone0} and \texttt{pZone1} must belong to the same \texttt{Fade_2D} object.
Index

add
GEOM_FADE2D::Bbox2, 30, 31
addLockedConstraint
GEOM_FADE2D::MeshGenParams, 69
applyConstraintsAndZones
GEOM_FADE2D::Fade_2D, 48
bAllowConstraintSplitting
GEOM_FADE2D::MeshGenParams, 70
Bbox2
GEOM_FADE2D::Bbox2, 30
Bbox2.h, 114
gBox, 115
operator<<, 115
bKeepExistingSteinerPoints
GEOM_FADE2D::MeshGenParams, 71
capAspectLimit
GEOM_FADE2D::MeshGenParams, 71
center
Point2.h, 121
centerWithShift
Point2.h, 121
checkValidity
GEOM_FADE2D::Fade_2D, 48
Circle2
GEOM_FADE2D::Circle2, 34, 35
Circle2.h, 115
CIS_CONSTRAINED_DELAUNAY
ConstraintSegment2.h, 116
CIS_IGNORE_DELAUNAY
ConstraintSegment2.h, 116
CIS_KEEP_DELAUNAY
ConstraintSegment2.h, 116
ClipResult
SegmentChecker.h, 122
clipSegment
GEOM_FADE2D::SegmentChecker, 81
Color
GEOM_FADE2D::Color, 36
Color.h, 115
command
GEOM_FADE2D::MeshGenParams, 71
computeBoundingBox
GEOM_FADE2D::Fade_2D, 48
computeCenter
GEOM_FADE2D::Bbox2, 31
computeRange
GEOM_FADE2D::Visualizer2, 100
ConstraintGraph2.h, 115
ConstraintInsertionStrategy
ConstraintSegment2.h, 116
ConstraintSegment2.h, 116
CIS_CONSTRAINED_DELAUNAY, 116
CIS_IGNORE_DELAUNAY, 116
CIS_KEEP_DELAUNAY, 116
ConstraintInsertionStrategy, 116
convertToBoundedZone
GEOM_FADE2D::Zone2, 108
CR_CLIPPED_DEGENERATE
SegmentChecker.h, 122
CR_CLIPPED_NONDEGENERATE
SegmentChecker.h, 122
CR_COMPLETE_DEGENERATE
SegmentChecker.h, 122
CR_COMPLETE_NONDEGENERATE
SegmentChecker.h, 122
CR_EMPTY
SegmentChecker.h, 122
CR_INVALID
SegmentChecker.h, 122
createConstraint
GEOM_FADE2D::Fade_2D, 48
createZone
GEOM_FADE2D::Fade_2D, 50, 51
createZone_cookieCutter
GEOM_FADE2D::Fade_2D, 51
cutTriangles
GEOM_FADE2D::Fade_2D, 52
deleteZone
GEOM_FADE2D::Fade_2D, 53
doIntersect
GEOM_FADE2D::Bbox2, 31
doubleTheBox
GEOM_FADE2D::Bbox2, 31
drape
GEOM_FADE2D::Fade_2D, 53
Edge2
GEOM_FADE2D::Edge2, 43
Edge2.h, 116
dgesToPolygons
Tools, 15
exportTriangulation
GEOM_FADE2D::Fade_2D, 54
Fade_2D
GEOM_FADE2D::Fade_2D, 48
Fade_2D.h, 117
FadeExport.h, 117
File I/O, 20
readPointsBIN, 20
readSegmentsBIN, 20
readXY, 20
writePointsASCII, 20, 21
writePointsBIN, 21
writeSegmentsBIN, 21
fillHole
Tools, 16
freeFunctions.h, 117
getOrientation2, 17
getOrientation2_mt, 17
getUndirectedEdges, 17
isSimplePolygon, 17
pointsToPolyline, 18
sortRing, 18
sortRingCCW, 18
Triangle2
  GEOM_FADE2D::Triangle2, 88
Triangle2.h, 123
TriangleAroundVertexIterator
  GEOM_FADE2D::TriangleAroundVertexIterator, 94
TriangleAroundVertexIterator.h, 123
unifyGrid
  GEOM_FADE2D::Zone2, 112
unsubscribe
  GEOM_FADE2D::Fade_2D, 65
  GEOM_FADE2D::SegmentChecker, 86
  GEOM_FADE2D::Zone2, 112
unusedParameter
  MeshGenParams.h, 119
update
  GEOM_FADE2D::MsgBase, 73
UserPredicates.h, 123
Vector2
  GEOM_FADE2D::Vector2, 97
Vector2.h, 123
    operator<<, 124
    operator+, 124
    operator-, 124
Version Information, 19
Visualizer2
  GEOM_FADE2D::Visualizer2, 100
Visualizer2.h, 125
VoroCell2.h, 125
Vorono2.h, 125
VoroVertex2.h, 125
writeFile
  GEOM_FADE2D::Visualizer2, 100
writeObj
  GEOM_FADE2D::Fade_2D, 65
  GEOM_FADE2D::Zone2, 112
writePointsASCII
  File I/O, 20, 21
writePointsBIN
  File I/O, 21
writeSegmentsBIN
  File I/O, 21
writeWebScene
  GEOM_FADE2D::Fade_2D, 65
x
  GEOM_FADE2D::Point2, 77
xy
  GEOM_FADE2D::Point2, 77
y
  GEOM_FADE2D::Point2, 77
Zone2.h, 126
OptimizationMode, 126
OPTMODE_BEST, 126
OPTMODE_BETTER, 126
OPTMODE_STANDARD, 126
zoneDifference, 126
zoneIntersection, 127
zoneSymmetricDifference, 127
zoneUnion, 127
zoneDifference
  GEOM_FADE2D::Zone2, 113
Zone2.h, 126
zoneIntersection
  GEOM_FADE2D::Zone2, 113
Zone2.h, 127
zoneSymmetricDifference
  GEOM_FADE2D::Zone2, 114
Zone2.h, 127
zoneUnion
  GEOM_FADE2D::Zone2, 114
Zone2.h, 127
Generated by Doxygen