A Case Study on the Portability of old CGAL Code

Stefan Schirra[*] Christian Schulz[*]

May 7, 2004

1 Introduction

In this abstract we report on our efforts to port "a case study on the cost of geometric computing" [5], which was based on release 1.2 of CGAL, to the latest release 3.0.1.

Right form the beginning of CGAL, algorithms in CGAL were parameterized by a geometry kernel and CGAL provided geometry kernels parameterized by number types [1, 2]. The resulting adaptability made CGAL an excellent platform for experimentation regarding robustness of geometric computing and the cost of geometric computing. An extensive case study on the performance of various geometry kernels in a number of algorithms for computing the convex hull of a planar point set was performed by the first author in 1998 [5]. The geometry kernels compared in the case study include the kernels resp. convex hull traits classes of CGAL-1.2. In addition, a few other kernels as well as specializations and variations of the CGAL kernels had been implemented and tested in the case study. Supplementary planar convex hull algorithms had been implemented and compared with CGAL's convex hull algorithms existent in release 1.2. Besides the case study framework, the code to be ported comprises these additional convex hull traits classes and supplementary convex hull algorithms.

Since release 1.2, CGAL evolved a lot. Many modifications have been made to make the library more uniform (e.g. with respect to naming conventions) or more flexible (e.g. geometry kernels with enhanced extensibility [3]), accompanied by realizations of the standardization of C++ (e.g. use of namespaces). The progress made in CGAL reflects in many problems that arose while attempting to port the old code [4] to the most recent release CGAL-3.0.1. In the sequel we discuss (some of) these problems.

2 Namespace CGAL

A major change in release 2.0 of CGAL was the introduction of namespace CGAL in order to get rid of the `CGAL_` prefix used in release 1.2 to avoid name clashes with other libraries. Release 2.0 and later releases come with a script `use_cgal_namespace` to replace the prefix by `CGAL::`. The script contains a list of reserved words where `CGAL_` is not replaced.

Since many functions and classes in the case study package had been considered as candidates for later inclusion in CGAL at the time of writing the code, these functions and classes had a `CGAL_` prefix as well, although they were user code and not part of CGAL-1.2. For all these functions and

[*]Otto-von-Guericke-University Magdeburg, Department of Computer Science, Institute of Simulation and Graphics, Magdeburg, Germany. Email: {stschirr, chrschul}@isg.cs.uni-magdeburg.de

classes, the script replaces prefix `CGAL_` by `CGAL::` when applied to the case study source files. The definitions of all these functions and classes had to be put into namespace `CGAL` by enclosing them by `namespace CGAL {` and `}`.

Not surprisingly, the case study package used a few own macro names starting with `CGAL_` as well. But of course, these names are not listed as reserved words in `use_cgal_namespace` and hence the script replaces the `CGAL_` prefix by `CGAL::` in unwanted places, too, for example, in include protection mechanisms and `#ifdef` directives. These errors had to be corrected manually. Note, that this is not a bug of the script provided by CGAL. Looking at the case study code retrospectively, it was not a good idea to mimic CGAL code and CGAL programming style that closely.

3 Convex Hull Traits

Convex hull algorithms are parameterized by convex hull traits, i.e., a collection of types for the geometric primitives used in the code. The case study package provided own convex hull traits according to the specification documented with CGAL-1.2 and own convex hull algorithm implementations parameterized by such traits classes. The types contained in the convex hull traits in CGAL-1.2 were:

```
typedef CGAL_Point_2<R>       Point_2;
CGAL_convex_hull_traits_2<R>::Less_xy
CGAL_convex_hull_traits_2<R>::Less_yx
CGAL_convex_hull_traits_2<R>::Leftturn
CGAL_convex_hull_traits_2<R>::Rightturn
CGAL_convex_hull_traits_2<R>::Right_of_line
CGAL_convex_hull_traits_2<R>::Less_dist_to_line
CGAL_convex_hull_traits_2<R>::Less_rotate_ccw
```

In the process of unification of naming schemes and traits classes, in particular in order to make a kernel usable as a traits class in the algorithms of CGAL's basic library, some names of function object classes have been changed in the convex hull traits. Now all the names have a `_2` suffix and abbreviations are now largely avoided. The concept of a convex hull traits class now contains the following types:

```
typedef R::Point_2                             Point_2;
typedef R::Less_xy                             Less_xy_2;
typedef R::Less_yx                             Less_yx_2;
typedef R::Less_signed_distance_to_line_2      Less_signed_distance_to_line_2;
typedef R::Less_rotate_ccw_2                   Less_rotate_ccw_2;
typedef R::Left_turn_2                         Left_turn_2;
typedef R::Equal_2                             Equal_2;
```

The `Rightturn` function object has been dropped, as it provides redundant functionality only. Wherever `Rightturn` was used, a left turn predicate can be used as well with rearranged parameters. Consequently corresponding modification had to be made in the convex hull code of the case study package. CGAL-3.0.1 provides an adaptor called `Turn_reverser` to turn a left turn predicate into a right turn predicate. This is a very useful, but apparently undocumented tool.

Next, there is now an `Equal_2` function object in the new traits class and this function object is actually used in CGAL's planar convex hull code. Therefore, all the additional traits classes of the case study package had to provide this function object as well. On the other hand we decided not to use this function object in our own convex hull code, but to stay with overloaded `operator==()`s.

We think, that this increases readability of the source code and hence eases maintenance work and ranked this higher than the gain in adaptability reached by the use of `Equal_2`.

The `Leftturn` predicate got the new name `Left_turn_2`. Since CGAL's planar convex hull code uses the function object with the new name, we had to rename the corresponding function in our own convex hull traits classes.

Worst of all, the function objects `Less_dist_to_line` and `Less_rotate_ccw` not only got a new name, but they got a new interface as well. Previously, the function object `Less_dist_to_line` had a constructor with two arguments of corresponding point type, the two points defining the line, and a function call operator with two arguments, namely the points whose distance is compared. Apparently in order to have only default constructible function objects in the traits classes (resp. kernel), the constructor of the substitute `Less_signed_distance_to_line_2` now has no arguments and the function call operator now has four arguments of corresponding point type, the first two are defining the line and the latter two are the points whose distance is compared. Now and then function object constructors are called through member functions of the convex hull traits class. In the old code we had

```
less_dist = ch_traits.get_less_dist_to_line_object(*a_it,*b_it);
c_it = max_element( f_it, b_it, less_dist );
```

And now we have

```
less_dist = ch_traits.less_signed_distance_to_line_2_object();
c_it = std::max_element( f_it, b_it, bind_1(bind_1(less_dist, *a_it), *b_it) );
```

So a binder must be applied twice to fix the points defining the line. Of course, this calls for major adjustments in both the traits classes and the convex hull code of the case study package. In the code fragment above, the function object is always used with the first two arguments fixed to the points defining the line. However, the function object can not make use of this fact, e.g. by doing some precomputation internally. The use of the old-style function object would reveal an avenue for optimization by precomputing the line data. However, to my knowledge, the old CGAL code in release 1.2 did not make use of such an optimization either. But at least, it did render this possible. So we consider the new interface a change for the worse.

Analogously, the interface of `Less_rotate_ccw` changed. Previously, this was a binary function object predicate with an unary constructor and construction member function in the traits class. Now, it is a ternary predicate, which can be trimmed to the old usage by binding its first argument using `bind_1`.

In order to enable the use of `bind_1` in CGAL's convex hull code, the function object predicates now must be models of the concept `AdaptableFunctor`, in particular, they must provide an `CGAL::Arity_tag<>` tag. So we had to provide this functionality in our traits class models as well. Concerning the construction member functions, the `get_` prefix present in CGAL-1.2 was removed and the `_2` suffix reflecting the change in the function object names has been added. Again, a change requiring adjustments in both own traits classes and own planar convex hull algorithms.

4 Kernels

The implementation of the kernel models provided by CGAL has gone through extensive revision. Since some of the case study code was mere specializations of kernel code existent in CGAL-1.2, these specializations had to be adjusted according to internal changes in CGAL. For example, some internal code had been dropped, some files had been renamed, ...

An early goal in the development of CGAL was to make a model of a kernel usable as a traits class for those algorithms that have no needs for special predicates. This is the main justification for the changes in the convex hull traits class. However, there is an incongruity in the current release. The requirements on the convex hull traits list function object `Left_turn_2`, whereas the requirements on the kernel list `Leftturn_2`. Hence, a model for CGAL's concept of a kernel can not be used as a convex hull traits model, unfortunately. At present, the kernels delivered by CGAL provide both function objects. Moreover, the documentation of the `..._object()` member functions for constructing predicate instances is somewhat hidden in the kernel documentation.

5 Further Porting Issues

We also had a number of minor problems with the proceeding realization of standard C++ in recent compilers, for example, delete `.h` from `<vector.h>` and others, add `std::` to `cout` and so on. Furthermore, there was a design problem in a few case study algorithms, where it was assumed implicitly that the iterator type of a vector of `Point`s is `Point*`. By the way, nowadays we use gcc-3.3.3, while egcs-2.91 was used in [5].

6 Conclusions

Porting the old code is a challenge, especially, because the old code made extensive use of features available in CGAL-1.2, documented and undocumented ones. The library further evolved and is now much more uniform and consistent. The evolution was accompanied with many changes destroying backwards compatibility and causes a very time-consuming and demanding revision of the case study code. Nevertheless, the improvements in the design of the CGAL library are highly welcome and, with the exception of discarding function object constructors with arguments, we consider all the relevant changes made between 1.2 and 3.0.1 an improvement. Moreover, we expect the kernel and traits class concept to be stable now, at least for some time ☺. The porting load is just the price a user has to pay for the progress.

References

[1] A. Fabri, G.-J. Giezeman, L. Kettner, S. Schirra, and S. Schönherr. On the design of CGAL a computational geometry algorithms library. *Softw. – Pract. Exp.*, 30(11):1167–1202, 2000.

[2] A. Fabri, G.-J. Giezeman, L. Kettner, S. Schirra, and S. Schönherr. The CGAL kernel: A basis for geometric computation. In M. C. Lin and D. Manocha, editors, *Proc. 1st ACM Workshop on Appl. Comput. Geom.*, volume 1148 of *Lecture Notes Computer Science*, pages 191–202. Springer-Verlag, 1996.

[3] S. Hert, M. Hoffmann, L. kettner, S. Pion, and M. Seel. An adaptable and extensible geometry kernel. In *Proc. Workshop on Algorithm Engineering*, volume 2141 of *Lecture Notes Computer Science*, pages 79–90. Springer-Verlag, 2001.

[4] S. Schirra. Companion pages to "A case study on the cost of geometric computing". `http://www.mpi-sb.mpg.de/~stschirr/exact/cost_of_geometric_computing/`

[5] S. Schirra. A case study on the cost of geometric computing. In M. T. Goodrich and C. C. McGeoch, editors, *Algorithm Engineering and Experimentation (Proc. ALENEX '99)*, volume 1619 of *Lecture Notes Computer Science*, pages 156–176. Springer-Verlag, 1999.