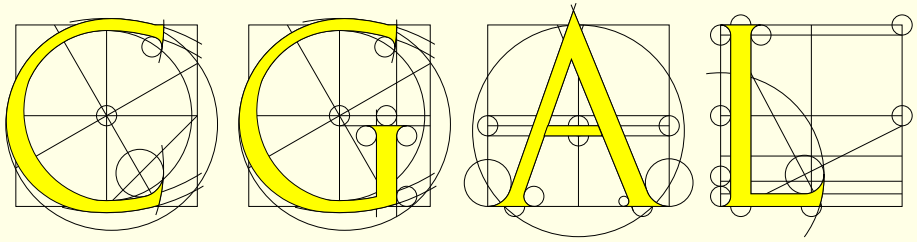
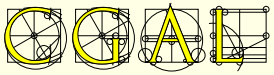


Curved Kernel

Monique Teillaud



www.cgal.org

EGC paradigm: Exact (and Efficient) Geometric Computation

www.cgal.org

EGC paradigm: Exact (and Efficient) Geometric Computation

Basic Library

Algorithms and Data Structures

Kernel

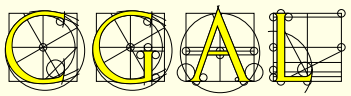
Geometric objects
Geometric operations

core library

configurations, assertions, ...

Support Library

Visualization
File I/O
NumberTypes
Generators
...

The  Kernel

In the kernel

Elementary geometric objects

Elementary computations on them

Primitives
2D, 3D, dD
- Point
- Vector
- Triangle
- Iso_rectangle
- Circle

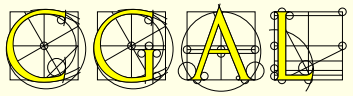
. . .

Predicates
- comparison
- Orientation
- InSphere

. . .

Constructions
- intersection
- squared distance

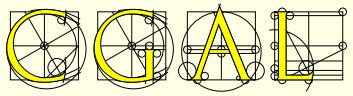
. . .

Curves in 

- "nothing" in the kernel

- Packages of the basic library
  - primitives for minimum enclosing ellipsis
  - primitives for arrangement of conic arcs
  - primitives for Apollonius diagram
  - primitives for segment Voronoi diagram
  - ...

need for a kernel for curves (and surfaces)



kernel for curves and surfaces

An old dream...

[Devillers-Fronville-Mourrain-T. SoCG'00]
[T.] [Pion-T.] *(ECG)*
[Emiris-Kakargias-Pion-T.-Tsigaridas SoCG'04]
first design ideas,
prototype implementation for arrangements of arcs of ellipses

Design and implementation in progress

collaborations in ACS (and out of ACS)

Several goals:

- more functionality for CGAL
- common platform for comparing/combining (algebraic) methods

Design overview

Constraints for a CGAL kernel:

- Interface must not be restricted to any particular implementation
- Interface must not be restricted to any particular application/algorithm

Arrangements of conic arcs: first example.

Design overview

Objectives :

- ability to reuse the CGAL kernel for points, lines,. . .
- possibility to use other implementations
- possibility to use several algebraic implementations

Design overview

Objectives :

- ability to reuse the CGAL kernel for points, lines,. . .
- possibility to use other implementations
- possibility to use several algebraic implementations

[Hert-Hoffmann-Kettner-Pion-Seel WAE'01]

$\Longrightarrow$ `Curved_kernel` parameterized by `BasicKernel`
and `Curved_kernel` derives from `BasicKernel`

$\Longrightarrow$ `Curved_kernel` parameterized by `AlgebraicKernel`

```
 template < BasicKernel, AlgebraicKernel >
class Curved_kernel
```


Concepts

```
 template < BasicKernel, AlgebraicKernel >
class Curved_kernel
```

concept (C++ / STL) = set of requirements that a type must provide in order to be usable by some template function or class.

AlgebraicKernel concept to be defined.

Types

- Inherited from `Basic_kernel`
  number type `RT`, points. . .

- Inherited from `Algebraic_kernel`
  algebraic numbers, polynomials

- Defined by `Curved_kernel`
  `Circle_2` ? `Conic_2`

  `Circular_arc_2, Circular_arc_point_2`
  `Conic_arc_2, Conic_arc_point_2`

  `Sphere_3,..., Quadric_3`

Predicates and Constructions

Geometric objects

`Conic_2`

equation = bivariate polynomial of degree 2

`Polynomial_2_2` concept

coefficients of type `RT`

`Conic_arc_point_2`

= intersection point or endpoint

coordinates = solution of system 2 equations, degree 2, 2 variables

`RootOfSys_2_2` concept

Geometric primitives

Geometric predicates and constructions
call primitives of `AlgebraicKernel` on `RootOfSys_2_2` and `Polynomial_2_2`

Geometric primitives

Geometric predicates and constructions
call primitives of `AlgebraicKernel` on `RootOfSys_2_2` and `Polynomial_2_2`

`Curved_Kernel::Construct_intersection_2(c1,c2)`

```
template < class OutputIterator >
OutputIterator
operator()(ConicKernel::Conic_2 c1, ConicKernel::Conic_2 c2,
           OutputIterator pts);
```

`pts` iterates on elements of type
`std::pair<ConicKernel::Conic_arc_point_2, int>`,
integer: multiplicity of the intersection point.

Geometric primitives

Geometric predicates and constructions
call primitives of `AlgebraicKernel` on `RootOfSys_2_2` and `Polynomial_2_2`

`Curved_Kernel::Construct_intersection_2(c1,c2)`

```
template < class OutputIterator >
OutputIterator
operator()(ConicKernel::Conic_2 c1, ConicKernel::Conic_2 c2,
           OutputIterator pts);
```

`pts` iterates on elements of type
`std::pair<ConicKernel::Conic_arc_point_2, int>`,
integer: multiplicity of the intersection point.

calls `Curved_Kernel::Get_equation(c_i)`

and then `Algebraic_kernel::Solve` on the equations.

Algebraic primitives

`Algebraic_Kernel::Solve(p1,p2)`

```
template < class OutputIterator >
OutputIterator
operator()(AlgebraicKernel::Polynomial_2_2 p1,
           AlgebraicKernel::Polynomial_2_2 p2,
           OutputIterator sols);
```

`sols` iterates on elements of type
`std::pair<AlgebraicKernel::``RootOfSys_2_2`*`, int>`,*
integer: multiplicity of the solution of {*p1*, *p2*}.

Algebraic kernel concepts

`AlgebraicKernel` must provide

- bivariate polynomials of degree 2 — `Polynomial_2_2` concept
- type for solutions of systems — `RootOfSys_2_2` concept
- algebraic numbers — `RootOf_d` concept

Algebraic kernel concepts

`AlgebraicKernel` must provide

- bivariate polynomials of degree 2 — `Polynomial_2_2` concept
- type for solutions of systems — `RootOfSys_2_2` concept
- algebraic numbers — `RootOf_d` concept

Concepts must be able to accept several models (= implementations)

- Athens
- MPI
- Core-based
- ...

$\Longrightarrow$ high level operations only

Algebraic numbers

`RootOf_d` concepts

The concepts must support:

- approximate handling, e.g. `C++ double`
- approximate certified handling, e.g. `CGAL::Interval_nt`
- exact number types, e.g.
  - for degree 2: `LEDA::real` and `CORE::Expr` with `sqrt()`
  - for degrees > 2:
    `LEDA::real` with diamond operator
    or `CORE::Expr` with `CORE::rootOf`
- polynomial representation
  `Root_of` [Emiris-Tsigaridas]
  or specialized version `Root_of_2` CGAL implementation [Pion]
  or `ROOT` [Karavelas]
  ...

What is left for the Curved kernel?

`Curved_Kernel::Construct_intersection_2(c1,c2)`
CK looks only like a wrapper, translating geometric words into algebraic words

But:

- other predicates/constructions give more work to CK
- choice of representation of geometric objects
- caching / storing history of construction
- geometric filtering (bounding boxes, bounding polygons...)
- . . .

Conclusion

collaborative work in progress...

- interfaces, specifications
- implementations
- benchmarking
- ...