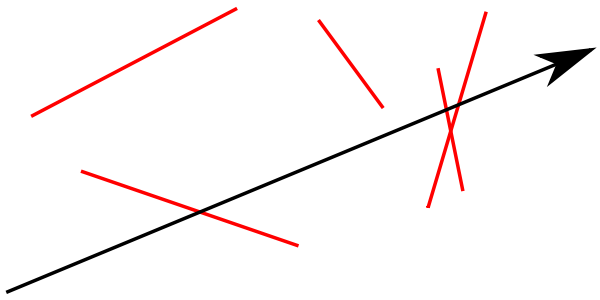
Algolab 2012
CGAL - Exercise Sheet 1

November 21, 2012

Hit – Problem



Hit – Solution

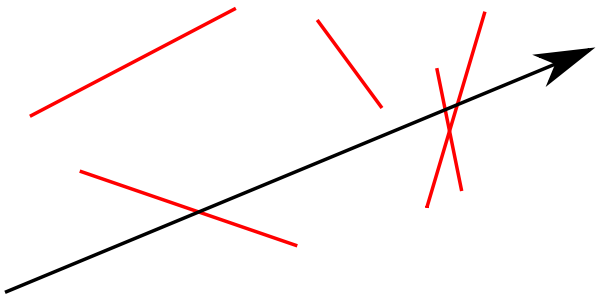
- `Ray_2 r`: stored as 2 points, source and one point of the ray
- `Segment_2 s`: stored as 2 points: source and target
- `do_intersect(r,s)`: predicate

⇒ We only need predicates and trivial constructions.

Hit – Code

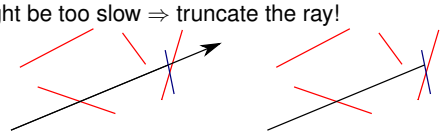
```
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>

typedef CGAL::Exact_predicates_inexact_constructions_kernel K;

int main()
{
  for (std::size_t n; std::cin >> n && n > 0;) {
    K::Ray_2 r;
    std::cin >> r;
    bool found = false;
    do {
      K::Segment_2 s;
      std::cin >> s;
      if (!found && CGAL::do_intersect(s,r)) found = true;
    } while (--n > 0);
    std::cout << (found ? "yes" : "no") << std::endl;
  }
}
```


First Hit – Problem

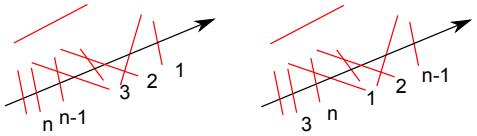


First Hit – Solution

- We need the exact coordinates of the intersection ⇒ exact constructions
- invoking `CGAL::intersection(r,s)` for *every* intersection might be too slow ⇒ truncate the ray!



- why should this be better?



- process segments in random order!

How many intersections do we need to compute?

Let $\{b_i\}_{i=1}^n$ denote a (uniform) random permutation of $[n]$ and let $c_i := \min\{b_j\}_{j=1}^i$. Let X be the number of different elements in $\{c_i\}_{i=1}^n$. Then

$$\mathrm{E}[X] = O(\log n)$$

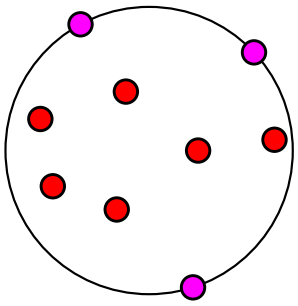
First Hit – Code

```
#include <CGAL/Exact_predicates_exact_constructions_kernel.h>
#include <vector>
#include <algorithm>

typedef CGAL::Exact_predicates_exact_constructions_kernel K;

// round down to next double
double floor_to_double(const K::FT& x)
{
  double a = std::floor(CGAL::to_double(x));
  while (a > x) a -= 1;
  while (a+1 <= x) a += 1;
  return a;
}

// clip/set target of s to o
void shorten_segment(K::Segment_2& s, CGAL::Object o)
{
  if (const K::Point_2* p = CGAL::object_cast<K::Point_2>(&o))
    s = K::Segment_2(s.source(), *p);
  else if (const K::Segment_2* t = CGAL::object_cast<K::Segment_2>(&o))
    // select endpoint of *t closer to s.source()
    if (CGAL::collinear_are_ordered_along_line
        (s.source(), t->source(), t->target()))
      s = K::Segment_2(s.source(), t->source());
    else
      s = K::Segment_2(s.source(), t->target());
  else
    throw std::runtime_error("Strange segment intersection.");
}
```


First Hit – Code

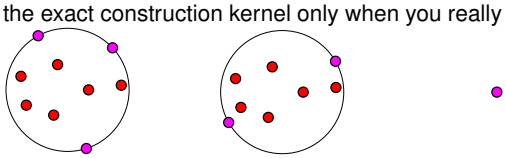
```
void find_hit(std::size_t n) {
  // read input
  K::Ray_2 r;
  std::cin >> r;
  std::vector<K::Segment_2> segs;
  segs.reserve(n);
  double ix,iy,jx,jy;
  for (std::size_t i = 0; i < n; ++i) {
    // as each coordinate can be represented as double, this is
    // significantly faster than K::Segment_2 s; std::cin >> s;
    std::cin >> ix >> iy >> jx >> jy;
    segs.push_back(K::Segment_2(K::Point_2(ix,iy),K::Point_2(jx,jy)));
  }
  std::random_shuffle(segs.begin(), segs.end());
  K::Segment_2 rc(r.source(), r.source());

  // find some segment hit by r
  std::size_t i = 0;
  for (; i < n; ++i)
    if (CGAL::do_intersect(segs[i], r)) {
      shorten_segment(rc, CGAL::intersection(segs[i], r));
      break;
    }
  if (i == n) { std::cout << "no\n"; return; }
  // check remaining segments against rc
  while (++i < n)
    if (CGAL::do_intersect(segs[i], rc))
      shorten_segment(rc, CGAL::intersection(segs[i], r)); // not rc!

  std::cout << floor_to_double(rc.target().x()) << " "
            << floor_to_double(rc.target().y()) << "\n";
}
```


Antenna – Problem



Antenna – Solution

- minimum enclosing circle ⇒ example from tutorial
- need to output the squared radius (requires construction of the center)
- use the exact construction kernel only when you really need it!



- (the purple points are called *support points*)

Antenna – Code

```
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
#include <CGAL/Exact_predicates_exact_constructions_kernel_with_sqrt.h>
#include <CGAL/Min_circle_2.h>
#include <CGAL/Min_circle_2_traits_2.h>
#include <vector>
#include <iostream>
#include <cmath>

typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
typedef CGAL::Min_circle_2<CGAL::Min_circle_2_traits_2<K> > MC;
typedef CGAL::Exact_predicates_exact_constructions_kernel_with_sqrt EK;
typedef CGAL::Min_circle_2<CGAL::Min_circle_2_traits_2<EK> > EMC;

// round up to next integer double
double ceil_to_double(const EK::FT& x)
{
  double a = std::ceil(CGAL::to_double(x));
  while (a < x) a += 1;
  while (a-1 >= x) a -= 1;
  return a;
}
```

Antenna – Code

```
22
   int main()
24 {
     std::cout << std::setiosflags(std::ios::fixed) << std::setprecision(0);
26   std::cin.sync_with_stdio(false);

28   for (std::size_t n; std::cin >> n && n > 0;) {
       std::vector<K::Point_2> pts;
30     pts.reserve(n);
       for (std::size_t i = 0; i < n; ++i) {
32       K::Point_2 p;
         std::cin >> p;
34       pts.push_back(p);
       }
36     MC mc(pts.begin(), pts.end(), true);
       // now we construct the circle using the exact kernel...
38     EK::Point_2 spt[3];
       for (std::size_t i = 0; i < mc.number_of_support_points(); ++i)
40       spt[i] = EK::Point_2(mc.support_point(i).x(), mc.support_point(i).y());
     EMC emc(spt, spt+mc.number_of_support_points());
42   // compute the exact squareroot and then an upper bound in double
     std::cout << ceil_to_double(sqrt(emc.circle().squared_radius())) << "\n";
44 }
     return 0;
46 }
```