Numbers and Points

Overview

Julian Pfeifle

Dept. Matemàtica Aplicada II
Universitat Politècnica de Catalunya

Computational Geometry 2011/12

Computational Geometry is about . . . Computing

. . . and *computing* means: Algorithms work on numbers.

What is a number?

$\mathbb{N}, \mathbb{Z}$	`int, long int, long long int` or work with arbitrary precision (slower)	$-9.2 \cdot 10^{18} \dots 9.2 \cdot 10^{18}$ `gmplib.org`
$\mathbb{Q}$	$\{(n, d) : n \in \mathbb{Z}, d \in \mathbb{N}_{>0}\}/\sim$, where $(n, d) \sim (n', d')$ iff $nd' = n'd$	`gmplib.org`
$\mathbb{R}$	algebraic numbers ok ($\sqrt{2}$, $\sqrt[4]{3}$), transcendental ones not (π, e)	`cgal.org`, `mpfr.org`
$\mathbb{C}, \mathbb{H}$	pairs or quadruples of reals	`#include <complex>`

Computational Geometry is about . . . Computing

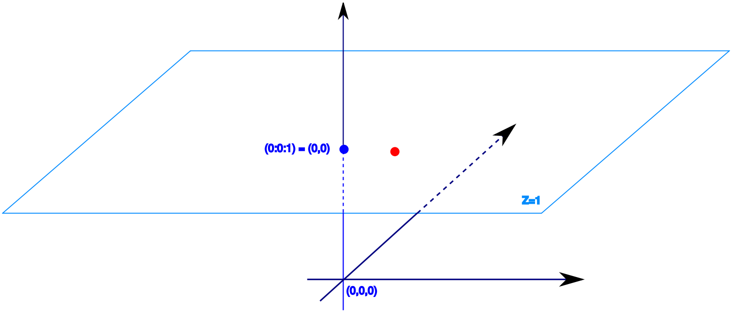
. . . and *computing* means: Algorithms work on numbers.

What is a number?

$\mathbb{N}, \mathbb{Z}$	`int, long int, long long int` or work with arbitrary precision (slower)	$-9.2 \cdot 10^{18} \dots 9.2 \cdot 10^{18}$ `gmplib.org`
$\mathbb{Q}$	$\{(n, d) : n \in \mathbb{Z}, d \in \mathbb{N}_{>0}\} / \sim$, where $(n, d) \sim (n', d')$ iff $nd' = n'd$	`gmplib.org`
$\mathbb{R}$	algebraic numbers ok ($\sqrt{2}$, $\sqrt[4]{3}$), transcendental ones not (π, e)	`cgal.org`, `mpfr.org`
$\mathbb{C}, \mathbb{H}$	pairs or quadruples of reals	`#include <complex>`

How much space does a number occupy on your hard disk?

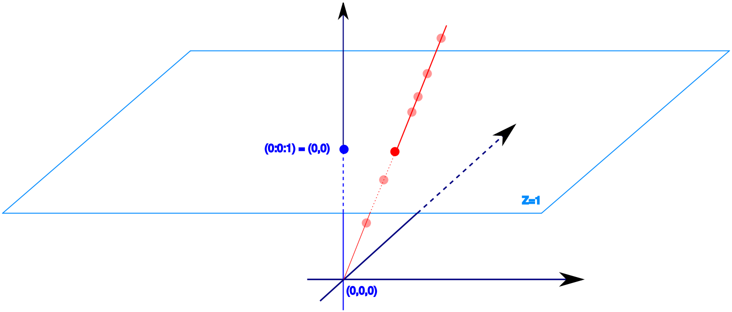
Roughly $\log_2 n$ bits.
($5 \cdot 10^{12}$ digits of π occupy 8.32 TB as `.txt`, 3.8 TB compressed)



Homogeneous coordinates for points in the plane

- Embed $\mathbb{R}^2$ at height 1 into $\mathbb{R}^3$: $(x, y) \mapsto (x : y : 1)$

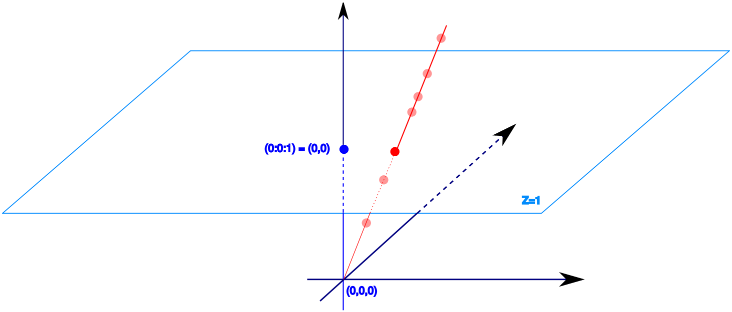
Computational Geometry is about . . . Geometry



Homogeneous coordinates for points in the plane

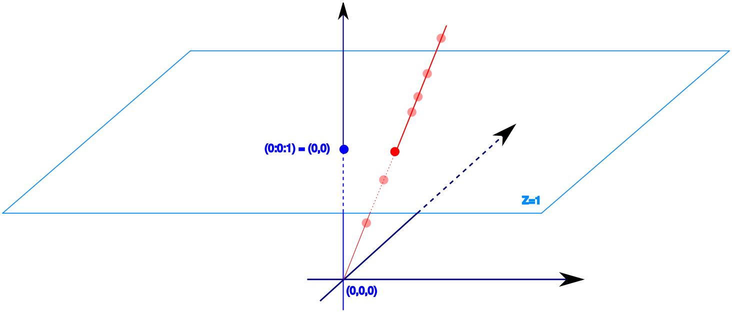
- Embed $\mathbb{R}^2$ at height 1 into $\mathbb{R}^3$: $(x, y) \mapsto (x : y : 1)$
- Identify points on a ray through the origin:
  $(x : y : 1) \sim (\lambda x : \lambda y : \lambda)$, for any $\lambda > 0$

Computational Geometry is about . . . Geometry



Homogeneous coordinates for points in the plane

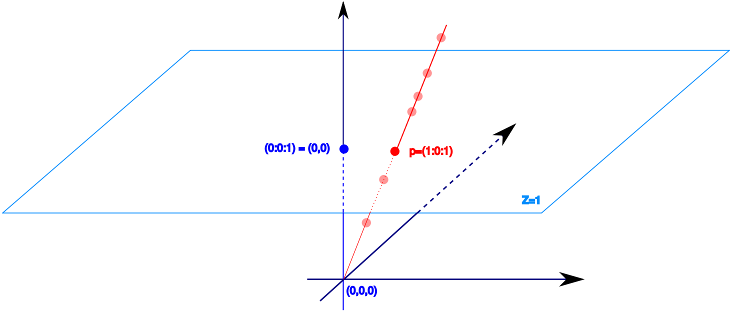
- Embed $\mathbb{R}^2$ at height 1 into $\mathbb{R}^3$: $(x, y) \mapsto (x : y : 1)$
- Identify points on a ray through the origin: $(x : y : 1) \sim (\lambda x : \lambda y : \lambda)$, for any $\lambda > 0$
- Subtlety: $\lambda > 0$ makes oriented projective geometry (we use this)



Homogeneous coordinates for points in the plane

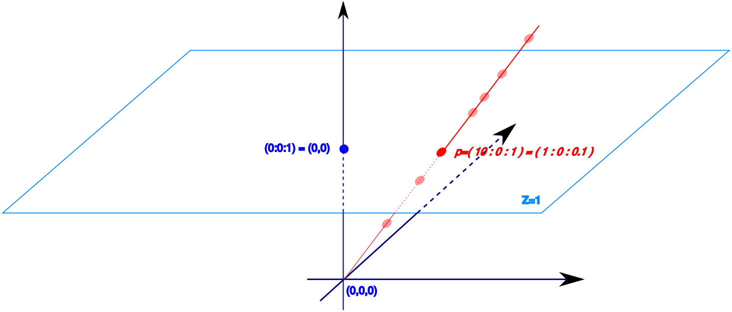
- Embed $\mathbb{R}^2$ at height 1 into $\mathbb{R}^3$: $(x, y) \mapsto (x : y : 1)$
- Identify points on a ray through the origin: $(x : y : 1) \sim (\lambda x : \lambda y : \lambda)$, for any $\lambda > 0$
- Subtlety: $\lambda \neq 0$ makes projective geometry

Computational Geometry is about . . . Geometry

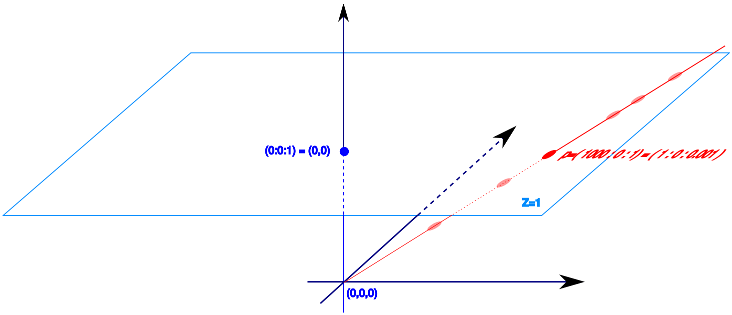


What happens if the last coordinate is zero?

Computational Geometry is about . . . Geometry

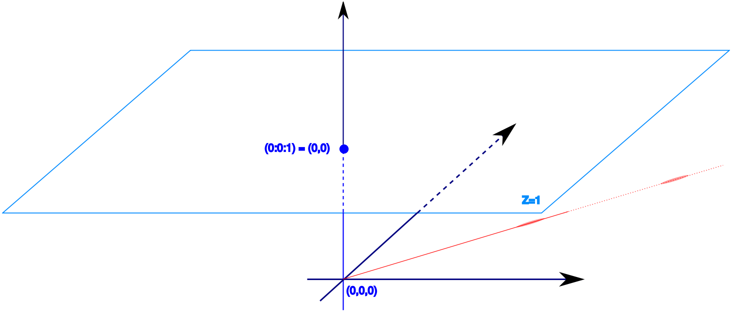


What happens if the last coordinate is zero?



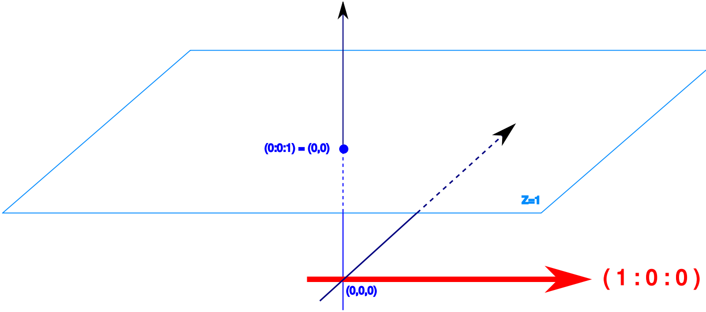
What happens if the last coordinate is zero?

Computational Geometry is about . . . Geometry



What happens if the last coordinate is zero?

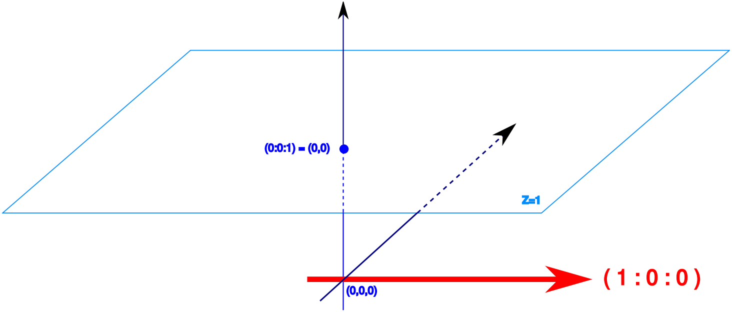
Computational Geometry is about . . . Geometry



What happens if the last coordinate is zero?

These vectors correspond to points at infinity.

Computational Geometry is about . . . Geometry



What happens if the last coordinate is zero?

These vectors correspond to points at infinity.
The only meaningless vector is $(0, 0, 0)$.

Operating with homogeneous coordinates

Homogeneous coordinates for points:

$$(x, y) \quad \longleftrightarrow \quad (x : y : 1)$$

Homogeneous coordinates for lines:

$$ax + by + c = 0 \quad \longleftrightarrow \quad (a : b : c)$$

Operating with homogeneous coordinates

Homogeneous coordinates for points:

$$(x, y) \quad \longleftrightarrow \quad (x : y : 1)$$

Homogeneous coordinates for lines:

$$ax + by + c = 0 \quad \longleftrightarrow \quad (a : b : c)$$

Incidence relationships

When does $p = (x : y : 1)$ lie on $\ell = (a : b : c)$?

Answer: When $ax + by + c = \langle (a, b, c), (x, y, 1) \rangle = 0$.

Operating with homogeneous coordinates

Homogeneous coordinates for points:

$$(x, y) \quad \longleftrightarrow \quad (x : y : 1)$$

Homogeneous coordinates for lines:

$$ax + by + c = 0 \quad \longleftrightarrow \quad (a : b : c)$$

Incidence relationships

When does $p = (x : y : 1)$ lie on $\ell = (a : b : c)$?

Answer: When $ax + by + c = \langle (a, b, c), (x, y, 1) \rangle = 0$. $\qquad \boldsymbol{p} \perp \ell$

Operating with homogeneous coordinates: lines

The line ℓ through points p_1 and p_2

- p_1 lies on ℓ: $p_1 \perp \ell$
- p_2 lies on ℓ: $p_2 \perp \ell$

Operating with homogeneous coordinates: lines

The line ℓ through points p_1 and p_2

- p_1 lies on ℓ: $\quad p_1 \perp \ell$
- p_2 lies on ℓ: $\quad p_2 \perp \ell$
- Therefore: $\ell = \lambda \cdot \boldsymbol{p_1} \times \boldsymbol{p_2}$ (the cross-product of vectors in $\mathbb{R}^3$)

Operating with homogeneous coordinates: lines

The line ℓ through points p_1 and p_2

- p_1 lies on ℓ: $\quad p_1 \perp \ell$
- p_2 lies on ℓ: $\quad p_2 \perp \ell$
- Therefore: $\ell = \lambda \cdot \mathbf{p_1} \times \mathbf{p_2}$ (the cross-product of vectors in $\mathbb{R}^3$)

Example (The line ℓ through $(1,2) = (1:2:1)$ and $(3,-4) = (3:-4:1)$)

$$\ell = (1,2,1) \times (3,-4,1) = (6:2:-10) \sim (3:1:-5).$$

This is correct, because both points satisfy $3x + y - 5 = 0$.

Operating with homogeneous coordinates: lines

The line ℓ through points p_1 and p_2

- p_1 lies on ℓ: $\quad p_1 \perp \ell$
- p_2 lies on ℓ: $\quad p_2 \perp \ell$
- Therefore: $\ell = \lambda \cdot \boldsymbol{p_1} \times \boldsymbol{p_2}$ (the cross-product of vectors in $\mathbb{R}^3$)

Example (The line ℓ through $(1,2) = (1:2:1)$ and $(3,-4) = (3:-4:1)$)

$$\ell = (1,2,1) \times (3,-4,1) = (6:2:-10) \sim (3:1:-5).$$

This is correct, because both points satisfy $3x + y - 5 = 0$.

Example (The x-axis)

$y = 0 \quad \longleftrightarrow \quad 0 \cdot x + 1 \cdot y + 0 = 0 \quad \longleftrightarrow \quad (0:1:0)$

Operating with homogeneous coordinates: points

The point q on lines ℓ_1 and ℓ_2

- q lies on ℓ_1: $q \perp \ell_1$
- q lies on ℓ_2: $q \perp \ell_2$

Operating with homogeneous coordinates: points

The point q on lines ℓ_1 and ℓ_2

- q lies on ℓ_1: $q \perp \ell_1$
- q lies on ℓ_2: $q \perp \ell_2$
- Therefore: $q = \lambda \cdot \boldsymbol{\ell_1} \times \boldsymbol{\ell_2}$ (the cross-product of vectors in $\mathbb{R}^3$)

Operating with homogeneous coordinates: points

The point q on lines ℓ_1 and ℓ_2

- q lies on ℓ_1: $q \perp \ell_1$
- q lies on ℓ_2: $q \perp \ell_2$
- Therefore: $q = \lambda \cdot \boldsymbol{\ell_1} \times \boldsymbol{\ell_2}$ (the cross-product of vectors in $\mathbb{R}^3$)

Example (The point q on $\ell_1 = (3 : 1 : -5)$ and $\ell_2 = (0 : 1 : 0)$)

$$q = (3, 1, -5) \times (0, 1, 0) = (5 : 0 : 3) \sim (\tfrac{5}{3} : 0 : 1).$$

This is correct, because this point lies on both lines.

.

Operating with homogeneous coordinates: points

The point q on lines ℓ_1 and ℓ_2

- q lies on ℓ_1: $q \perp \ell_1$
- q lies on ℓ_2: $q \perp \ell_2$
- Therefore: $q = \lambda \cdot \boldsymbol{\ell_1} \times \boldsymbol{\ell_2}$ (the cross-product of vectors in $\mathbb{R}^3$)

Example (The point q on $\ell_1 = (3:1:-5)$ and $\ell_2 = (0:1:0)$)

$$q = (3, 1, -5) \times (0, 1, 0) = (5 : 0 : 3) \sim (\tfrac{5}{3} : 0 : 1).$$

This is correct, because this point lies on both lines.

Watch out: Choose the order of the factors in $p \times q$ so that the last coordinate is nonnegative.

Operating with homogeneous coordinates: special cases

Where do parallel lines intersect?

We do an example with $y = 0$ and $y = -2$:

Operating with homogeneous coordinates: special cases

Where do parallel lines intersect?

We do an example with $y = 0$ and $y = -2$:
$(0 : 1 : 0) \times (0 : 1 : 2) = (2 : 0 : 0)$: this is a point at infinity.

Operating with homogeneous coordinates: special cases

Where do parallel lines intersect?

We do an example with $y = 0$ and $y = -2$:
$(0 : 1 : 0) \times (0 : 1 : 2) = (2 : 0 : 0)$: this is a point at infinity.

What happens if $\ell_1 = \ell_2$?

Operating with homogeneous coordinates: special cases

Where do parallel lines intersect?

We do an example with $y = 0$ and $y = -2$:
$(0:1:0) \times (0:1:2) = (2:0:0)$: this is a point at infinity.

What happens if $\ell_1 = \ell_2$?

$\ell_1 \times \ell_2 = \ell_1 \times \ell_1 = (0,0,0)$, the only vector that has no meaning.

Operating with homogeneous coordinates: special cases

Where do parallel lines intersect?

We do an example with $y = 0$ and $y = -2$:
$(0 : 1 : 0) \times (0 : 1 : 2) = (2 : 0 : 0)$: this is a point at infinity.

What happens if $\ell_1 = \ell_2$?

$\ell_1 \times \ell_2 = \ell_1 \times \ell_1 = (0, 0, 0)$, the only vector that has no meaning.

The only way to get $(0, 0, 0)$ is to intersect two equal lines,
or to calculate the line through two equal points.

Operating with homogeneous coordinates: special cases

Where do parallel lines intersect?

We do an example with $y = 0$ and $y = -2$:
$(0 : 1 : 0) \times (0 : 1 : 2) = (2 : 0 : 0)$: this is a point at infinity.

What happens if $\ell_1 = \ell_2$?

$\ell_1 \times \ell_2 = \ell_1 \times \ell_1 = (0, 0, 0)$, the only vector that has no meaning.

The only way to get $(0, 0, 0)$ is to intersect two equal lines, *or to calculate the line through two equal points.*

What is the line through two points at infinity?

Operating with homogeneous coordinates: special cases

Where do parallel lines intersect?

We do an example with $y = 0$ and $y = -2$:
$(0 : 1 : 0) \times (0 : 1 : 2) = (2 : 0 : 0)$: this is a point at infinity.

What happens if $\ell_1 = \ell_2$?

$\ell_1 \times \ell_2 = \ell_1 \times \ell_1 = (0, 0, 0)$, the only vector that has no meaning.

The only way to get $(0, 0, 0)$ is to intersect two equal lines,
or to calculate the line through two equal points.

What is the line through two points at infinity?

$(x_1 : y_1 : 0) \times (x_2 : y_2 : 0) = (0 : 0 : x_1 y_2 - x_2 y_1) \sim (0 : 0 : 1)$,
the line at infinity.

Operating with homogeneous coordinates: special cases

Where do parallel lines intersect?

We do an example with $y = 0$ and $y = -2$:
$(0 : 1 : 0) \times (0 : 1 : 2) = (2 : 0 : 0)$: this is a point at infinity.

What happens if $\ell_1 = \ell_2$?

$\ell_1 \times \ell_2 = \ell_1 \times \ell_1 = (0, 0, 0)$, the only vector that has no meaning.

The only way to get $(0, 0, 0)$ is to intersect two equal lines,
or to calculate the line through two equal points.

What is the line through two points at infinity?

$(x_1 : y_1 : 0) \times (x_2 : y_2 : 0) = (0 : 0 : x_1 y_2 - x_2 y_1) \sim (0 : 0 : 1)$,
the line at infinity.
(Unless the points coincide (are proportional); then you get $(0, 0, 0)$.)

Computational Geometry is about
. . . algorithms & coding

Let's intersect lines and join points, first in homogeneous coordinates.

```
void intersect_lines(const vec_t& l1, const vec_t& l2, vec_t& p)
{
   cross_product(l1, l2, p);
}
void join_points(const vec_t& p1, const vec_t& p2, vec_t& l)
{
   cross_product(p1, p2, l);
}
void cross_product(const vec_t& l1, const vec_t& l2, vec_t& p)
{
   p[0] =  l1[1]*l2[2] - l1[2]*l2[1];
   p[1] = -l1[0]*l2[2] + l1[2]*l2[0];
   p[2] =  l1[0]*l2[1] - l1[1]*l2[0];
}
```

- This code is correct, efficient, and robust.

Computational Geometry is about
. . . algorithms & coding

Let's intersect lines and join points, first in homogeneous coordinates.

```
void intersect_lines(const vec_t& l1, const vec_t& l2, vec_t& p)
{
   cross_product(l1, l2, p);
}
void join_points(const vec_t& p1, const vec_t& p2, vec_t& l)
{
   cross_product(p1, p2, l);
}
void cross_product(const vec_t& l1, const vec_t& l2, vec_t& p)
{
   p[0] =  l1[1]*l2[2] - l1[2]*l2[1];
   p[1] = -l1[0]*l2[2] + l1[2]*l2[0];
   p[2] =  l1[0]*l2[1] - l1[1]*l2[0];
}
```

- This code is correct, efficient, and robust.

  It calculates exactly what it should.

Computational Geometry is about
. . . algorithms & coding

Let's intersect lines and join points, first in homogeneous coordinates.

```
void intersect_lines(const vec_t& l1, const vec_t& l2, vec_t& p)
{
   cross_product(l1, l2, p);
}
void join_points(const vec_t& p1, const vec_t& p2, vec_t& l)
{
   cross_product(p1, p2, l);
}
void cross_product(const vec_t& l1, const vec_t& l2, vec_t& p)
{
   p[0] =  l1[1]*l2[2] - l1[2]*l2[1];
   p[1] = -l1[0]*l2[2] + l1[2]*l2[0];
   p[2] =  l1[0]*l2[1] - l1[1]*l2[0];
}
```

- This code is correct, efficient, and robust.

  No extraneous copying (&); reuse of code

Computational Geometry is about
. . . algorithms & coding

Let's intersect lines and join points, first in homogeneous coordinates.

```
void intersect_lines(const vec_t& l1, const vec_t& l2, vec_t& p)
{
   cross_product(l1, l2, p);
}
void join_points(const vec_t& p1, const vec_t& p2, vec_t& l)
{
   cross_product(p1, p2, l);
}
void cross_product(const vec_t& l1, const vec_t& l2, vec_t& p)
{
   p[0] =  l1[1]*l2[2] - l1[2]*l2[1];
   p[1] = -l1[0]*l2[2] + l1[2]*l2[0];
   p[2] =  l1[0]*l2[1] - l1[1]*l2[0];
}
```

- This code is correct, efficient, and robust.

  No influence of rounding errors

Computational Geometry is about
. . . algorithms & coding

Let's intersect lines and join points, first in homogeneous coordinates.

```
void intersect_lines(const vec_t& l1, const vec_t& l2, vec_t& p)
{
   cross_product(l1, l2, p);
}
void join_points(const vec_t& p1, const vec_t& p2, vec_t& l)
{
   cross_product(p1, p2, l);
}
void cross_product(const vec_t& l1, const vec_t& l2, vec_t& p)
{
   p[0] =  l1[1]*l2[2] - l1[2]*l2[1];
   p[1] = -l1[0]*l2[2] + l1[2]*l2[0];
   p[2] =  l1[0]*l2[1] - l1[1]*l2[0];
}
```

- This code is correct, efficient, and robust.

  It handles all cases, even degenerate ones.

Computational Geometry is about
. . . algorithms & coding

Now let's intersect lines in Cartesian coordinates.
A line is now $y = kx + d$, stored as a vector (k, d).

```
bool intersect_lines(const vec_t& l1, const vec_t& l2, vec_t& p) {
   if (l1[0]==l2[0]) {
      if (l1[1]==l2[1]) {
         return COINCIDENT_LINES;
      }
      return PARALLEL_LINES;
   }
   p[0] = (l2[1]-l1[1])/(l1[0]-l2[0]);
   p[1] = l1[0]*p[0] + l1[1];
}
```

- This code is somewhat efficient, not robust, and *not even correct*.

Computational Geometry is about
. . . algorithms & coding

Now let's intersect lines in Cartesian coordinates.
A line is now $y = kx + d$, stored as a vector (k, d).

```
bool intersect_lines(const vec_t& l1, const vec_t& l2, vec_t& p) {
   if (l1[0]==l2[0]) {
      if (l1[1]==l2[1]) {
         return COINCIDENT_LINES;
      }
      return PARALLEL_LINES;
   }
   p[0] = (l2[1]-l1[1])/(l1[0]-l2[0]);
   p[1] = l1[0]*p[0] + l1[1];
}
```

- This code is somewhat efficient, not robust, and *not even correct*.
  Again, no copying;   BUT: no reuse of code for `join_points`

Computational Geometry is about
. . . algorithms & coding

Now let's intersect lines in Cartesian coordinates.
A line is now $y = kx + d$, stored as a vector (k, d).

```
bool intersect_lines(const vec_t& l1, const vec_t& l2, vec_t& p) {
   if (l1[0]==l2[0]) {
      if (l1[1]==l2[1]) {
         return COINCIDENT_LINES;
      }
      return PARALLEL_LINES;
   }
   p[0] = (l2[1]-l1[1])/(l1[0]-l2[0]);
   p[1] = l1[0]*p[0] + l1[1];
}
```

- This code is somewhat efficient, not robust, and *not even correct*.
  It's unstable numerically (==, /)—say `if(fabs(l1[0]-l2[0])<EPS)`

Computational Geometry is about
. . . algorithms & coding

Now let's intersect lines in Cartesian coordinates.
A line is now $y = kx + d$, stored as a vector (k, d).

```
bool intersect_lines(const vec_t& l1, const vec_t& l2, vec_t& p) {
   if (l1[0]==l2[0]) {
      if (l1[1]==l2[1]) {
         return COINCIDENT_LINES;
      }
      return PARALLEL_LINES;
   }
   p[0] = (l2[1]-l1[1])/(l1[0]-l2[0]);
   p[1] = l1[0]*p[0] + l1[1];
}
```

- This code is somewhat efficient, not robust, and *not even correct*.
  It doesn't handle parallel lines, but that's Euclidean geometry's fault.

Computational Geometry is about
. . . algorithms & coding

Now let's intersect lines in Cartesian coordinates.
A line is now $y = kx + d$, stored as a vector (k, d).

```
bool intersect_lines(const vec_t& l1, const vec_t& l2, vec_t& p) {
   if (l1[0]==l2[0]) {
      if (l1[1]==l2[1]) {
         return COINCIDENT_LINES;
      }
      return PARALLEL_LINES;
   }
   p[0] = (l2[1]-l1[1])/(l1[0]-l2[0]);
   p[1] = l1[0]*p[0] + l1[1];
}
```

- This code is somewhat efficient, not robust, and *not even correct*.
  But it IS the code's fault that it doesn't handle vertical lines! Need:

```
class line {
  bool is_vertical;
  vec_t k_and_d;
};
```

Computational Geometry is about
. . . algorithms & coding

Now let's intersect lines in Cartesian coordinates.
A line is now $y = kx + d$, stored as a vector (k, d).

```
bool intersect_lines(const vec_t& l1, const vec_t& l2, vec_t& p) {
   if (l1[0]==l2[0]) {
      if (l1[1]==l2[1]) {
         return COINCIDENT_LINES;
      }
      return PARALLEL_LINES;
   }
   p[0] = (l2[1]-l1[1])/(l1[0]-l2[0]);
   p[1] = l1[0]*p[0] + l1[1];
}
```

- This code is somewhat efficient, not robust, and *not even correct*.
  But it IS the code's fault that it doesn't handle vertical lines! Need:
- It's much harder to get right, because of more special cases.
- It needs an extra variable, the boolean return value.