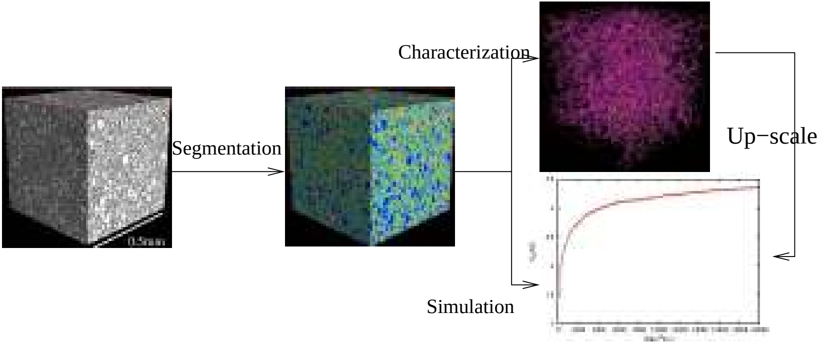
Population Library

Vincent Tariel

Shinoe software

History

1. Ph.D. : Geometry and diffusion transport in cement paste



Segmentation

Characterization

Up−scale

Simulation

2. For large volume of data, no 64 bits library in 2005, so
   1. 2005-2006 Population 1.0 in C
   2. 2006-2010 Population 2.0 in C++ in low-level generic programming
   3. 2010-now Population 3.0 in C++ in high-level generic programming

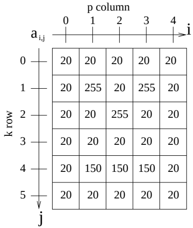
Outline

1. Program = Data + Algorithm
    - Function and IteratorE concepts
    - Classic algorithm
    - Region growing algorithm

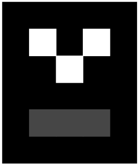
2. Productivity in complexity

Function concept : level 1

2d regular grid image with 1 byte (uchar) pixel :



	0	1	2	3	4
0	20	20	20	20	20
1	20	255	20	255	20
2	20	20	255	20	20
3	20	20	20	20	20
4	20	150	150	150	20
5	20	20	20	20	20



Mathematics

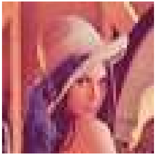
$$\begin{array}{ccc} f : \mathcal{D} \subset \mathbb{Z}^2 & \mapsto & (0, 1, \ldots, 255) \\ x & & y = f(x) \end{array}$$

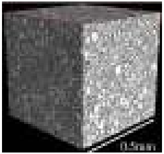
Programming

```
class ImageGrid2D_UC{
 pair<int,int> _domain;
 vector<vector<uchar> > _data;
 uchar& operator()(int i, int j);
 ImageGrid2D_UC(int sizei,int sizej);
};
int main(){
  ImageGrid2D_UC img(5,5);
  img(2,2)=120;
}
```

Function concept : level 2

Regular grid image :



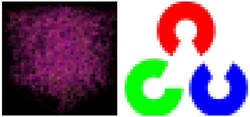


Mathematics	Programming
$\begin{array}{rcl} f : \mathcal{D} \subset \mathbb{Z}^d & \mapsto & F \\ x & & y = f(x) \end{array}$	see code below

```
template<int D,typename Type>
class ImageGrid{
 Point<D> _domain;
 vector<Type> _data;
 Type& operator()(Point<D> x);
};
typedef ImageGrid<2,uchar> ImageGrid2D_UC;
int main(){
  ImageGrid<2,ColorUC> img;
  img.load("lena.pgm");
  Point<2> x(5,5);
  cout<<img(x)<<endl;
}
```

Function concept : level 3

Any functions :



Mathematics

$$\begin{array}{ccc} f : \mathcal{D} \subset E & \mapsto & F \\ x & & y = f(x) \end{array}$$

Programming

```
class ConceptFunction{
 typedef ... Domain;
 typedef ... E;
 typedef ... F;
 ConceptFunction(Domain & d);
 Domain getDomain();
 F& operator()(E x);
};
//one model
template<int D,typename Type>
class ImageGrid{
  typedef Point<D> Domain;
  typedef Point<D> E;
  typedef Type F;
   ...
```

IteratorE concept

Mathematics

- IteratorETotal

$$\forall x \in \mathcal{D}$$

- IteratorEROI

$$\forall x \in R \subset \mathcal{D}$$

- IteratorENeighborhood

$$\forall x' \in N(x)$$

Programming

```
// definition  concept
class  ConceptIteratorE{
  typedef  ...  Domain;
  ConceptIteratorE(Domain d);
  bool  next();//next element and indicate  if
        the end
  E x()://return  the  current  element
};

// model IteratorETotal  definition  for  the
     ImageGrid model
class  ImageGridIteratorETotal ;

//associated  type  in  the  model
template<int D,typename Type>
class  ImageGrid{
  typedef ImageGridIteratorETotal
        IteratorETotal ;
   IteratorETotal :: Domain
        getIteratorETotalDomain();
  ...
};
```

Example erosion : level 1

Mathematics : $\forall f, h \in \mathcal{F}, \forall x \in E'$: $\quad h(x) = \min_{\forall x' \in N(x)} f(x')$

Programming :

```
Image2D_UC Erosion(const Image2D_UC & f,double norm, double radius){
    Image2D_UC erosion(f.getDomain());
    Image2D_UC::IteratorETotal itg(f.getIteratorETotalDomain());
    Image2D_UC::IteratorENeighborhood itl(f.getIteratorENeighborhoodDomain(norm,radius));
    while( itg .next()){
        unsigned char mini=numeric_limits<unsigned char>::max();;
         itl . init ( itg .x());
        while( itl .next()){
            mini = min(mini,in( itl .x()));
        }
        erosion ( itg .x())= mini;
    }
    return erosion ;
}
```

Example erosion : level 2

“free the object with some properties” = $\forall f, h \in \mathcal{F}$

```
template<typename Function>
Function Erosion(const Function & f,double norm, double radius){
    Function erosion(f.getDomain());
    typename Function::IteratorETotal itg(f.getIteratorETotalDomain());
    typename Function::IteratorENeighborhood itl(f.getIteratorENeighborhoodDomain(norm,
         radius));
    while(itg.next()){
        typename Function::F mini=numeric_limits<typename Function::F>::max();
        itl.init(itg.x());
        while(itl.next()){
            mini = min(mini,f(itl.x()));
        }
        erosion(itg.x())= mini;
    }
    return erosion;
}
```

Example erosion : level 3

“free the iteration”=$\forall x \in E'$ and $\forall x' \in N(x)$

```
template< typename Function,typename IteratorEGlobal,typename IteratorELocal>
Function FunctionProcedureAccumulateInGlobalLocalIteration( Function1 f, IteratorEGlobal
      itg, IteratorELocal  itl )
{
    Function erosion(f.getDomain());
    while(itg.next()){
        typename Function::F mini=numeric_limits<typename Function::F>::max();
        itl.init(itg.x());
        while(itl.next()){
            mini = min(mini,f(itl.x()));
        }
        erosion(itg.x())= mini;
    }
    return erosion;
}
```

Acculumator algorithm : level 4

“free the accumulator process”=

$\forall f, g \in \mathcal{F}, \forall x \in E' : h(x) = H(\{f(x') : \forall x' \in N(x)\})$

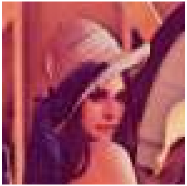
```
template< typename Function,typename IteratorEGlobal,typename IteratorELocal,template
      FunctorAccumulator>
Function  FunctionProcedureAccumulateInGlobalLocalIteration( Function1 f,  IteratorEGlobal
      itg ,  IteratorELocal    itl ,  FunctorAccumulator accumulator)
{
    Function  h(f.getDomain());
    while( itg .next()){
        accumulator. init ();
         itl . init ( itg .x());
        while( itl .next()){
            accumulator(in( itl .x()));
        }
        h( itg .x())= accumulator.getValue();
    }
    return  f;
}
```

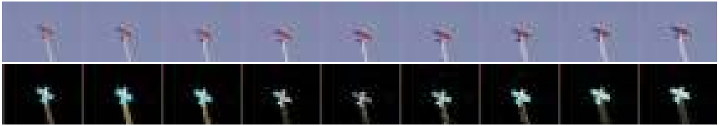
Accumulate algorithm

- Mathematics :

$$\forall x \in E' : \quad h(x) = H(\{f(x') : \forall x' \in N(x)\})$$

The accumulate functor H is a mapping from $\mathcal{P}(F)$ to F that can return

  - the max/min/median value, 
  - kernel convolution :

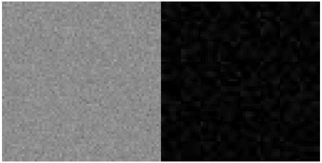


Generator algorithm

- Mathematics :

$$\forall x \in E' : h(x) = H()$$

The generator H can return :



  - $X \sim P$, random number

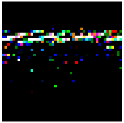
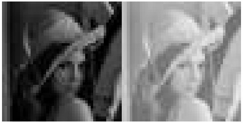
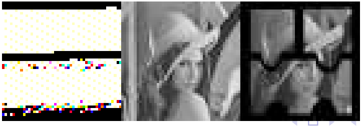
  - c a constant value
  - ...

Point algorithm

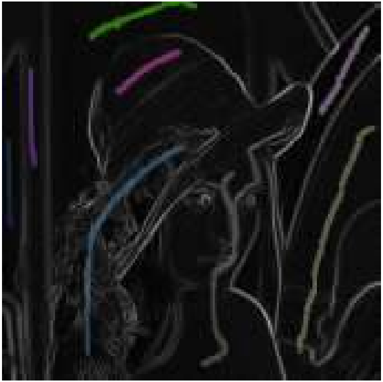
- Mathematics :

$$\forall x \in E' : h(x) = H(f(x))$$
$$\forall x \in E' : h(x) = H(f(x), g(x))$$

The unary/binary functor H as mapping from $F^{1 \text{ or } 2}$ to F can return :

   - $\begin{cases} 255 & \text{for} min \leq f(x) \leq max \\ 0 & \text{otherwise.} \end{cases}$ thresholded value, 
   - with function with a symbolic link 
   - $min(f(x), g(x))$ 

Watershed transformation

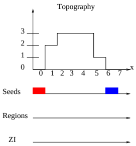


Watershed transformation

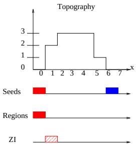
The watershed algorithm is :

1. Input : topographic surface, f, and seeds, $\{s_i\}_{0\leq i\leq n}$.
2. Sequential initialization of the regions with the seeds, $X_i^{t=0} = s_i$,
3. For $l = 0$ to the maximum level of the topographic surface, *//Region growing*
    1. While some pairs (i, x) satisfy $\max(l, f(x)) = l$ and $x \in Z_i^t$ with Z_i^t the outter boundary of the region i in excluding the other regions
        - Selection of the pair (j, y) that satisfies for the first time both conditions and still respects their until now.
        - Region growing : $X_j^{t+1} = X_j^t \cup \{y\}$
    2. End while
4. Return the regions.

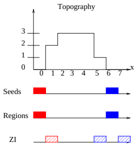
Watershed transformation



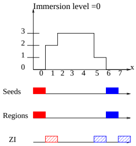
Init state



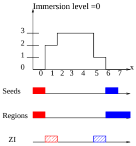
Init red region



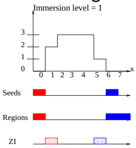
Init the blue region



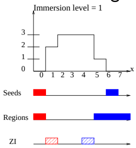
Ready to go



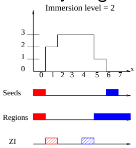
growth ($x = 7, i = blue$)



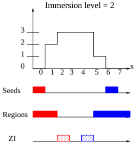
No more pair, l=1



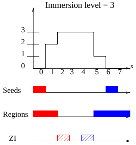
growth ($x = 5, i = blue$)



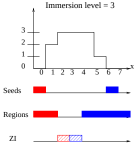
No more pair, l=2



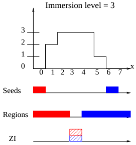
growth ($x = 1, i = red$)



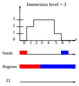
No more pair, l=3

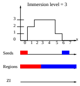


growth($x = 2, i = red$)
first one



growth ($x = 2, i = red$)





Watershed transformation

```
template<typename FunctionTopo,typename FunctionRegion>
FunctionRegion FunctionProcedureWatershed(const FunctionTopo & topo,const
     FunctionRegion & seed, typename FunctionTopo::IteratorENeighborhood & itn )
{
  FunctorTopography<FunctionTopo> functortopo(topo);
  Population<FunctionRegion,FunctorTopography<FunctionTopo> > pop(seed.getDomain(),
       functortopo,itn);
  typename FunctionTopo::IteratorETotal  it(topo.getDomain());
  // initialisation  of the regions with seeds
  while( it .next() ){
    if(seed( it .x() )!=0)
      pop.growth(seed( it .x() ), it .x() );
  }
  //region growing
  for( int  level =0;level <functortopo.nbrLevel() ; level ++){
    pop.setLevel ( level ) ;
    functortopo . setLevel ( level ) ;
    while(pop.next() ){
      pop.growth(pop.x(). first ,pop.x() .second);
    }
  }
  return pop.getRegion();
}
```

Implementation

classical algorithms : Voronoï tessellation, cluster to label, regional minima, distance function, watershed transformation, geodesic reconstruction and Adam's algorithm.

Implementation

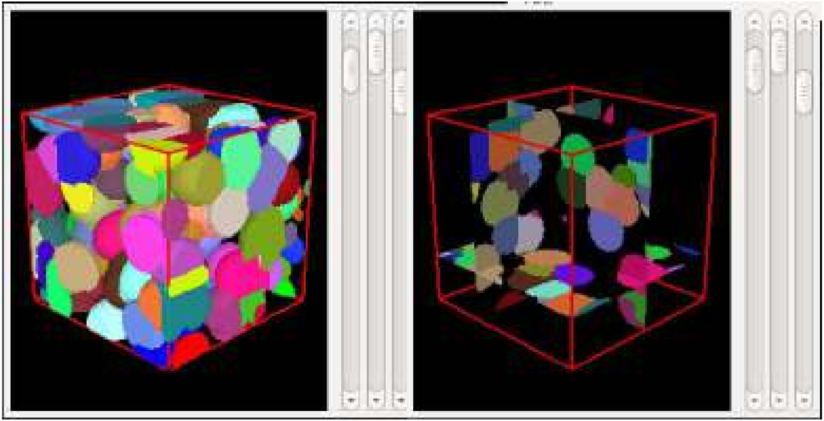
New algorithms :

- quasi-euclidean distance in quasi-linear time

Implementation

New algorithms :

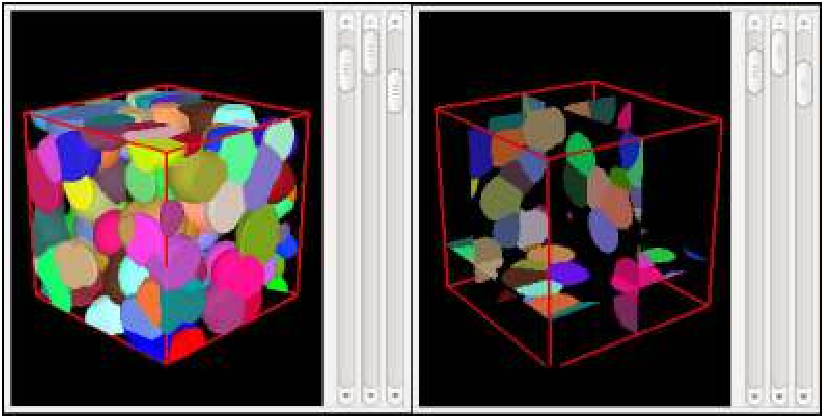
- adaptative meshing in phase field



Implementation

New algorithms :

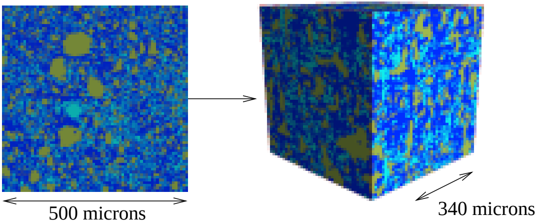
- adaptative meshing in phase field



Implementation

New algorithms :

- permutation localization in simulated annealing reconstruction



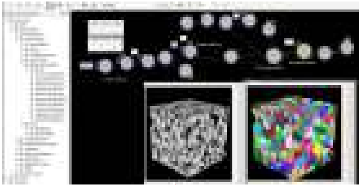
Outline

1. Program = Data + Algorithm

2. Productivity in complexity
   - Two scale programming paradigms

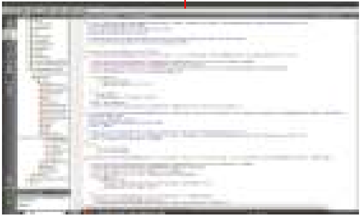
Cameleon language in collaboration with Cugnon de Sevricourt

Complexity



Macro−programming

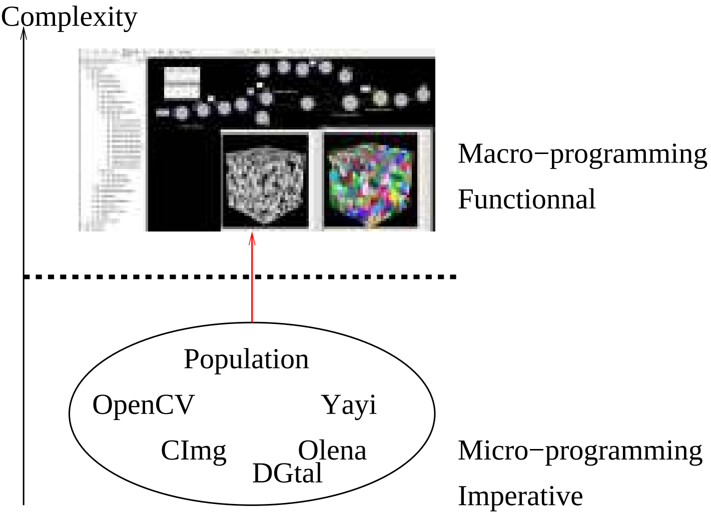
Functionnal



Micro−programming

Imperative

Cameleon language in collaboration with Cugnon de Sevricourt



Complexity

Macro−programming

Functionnal

Population

OpenCV

Yayi

CImg

Olena

DGtal

Micro−programming

Imperative

Conclusion

If time demonstration else :

1. with concept/model, you spend more time with a pencil than with a keyboard (for further explanation, my book is available at www.shinoe.com/population),
2. with cameleon integration, you democratize your libary and you can use other libraries in shared environement,
3. with IPOL, you democratize the acces of Science.