DIGITAL GEOMETRY

UGE M2 INFORMATIQUE “IMAGERY SCIENCE”

TP1 : Counting and measuring grains

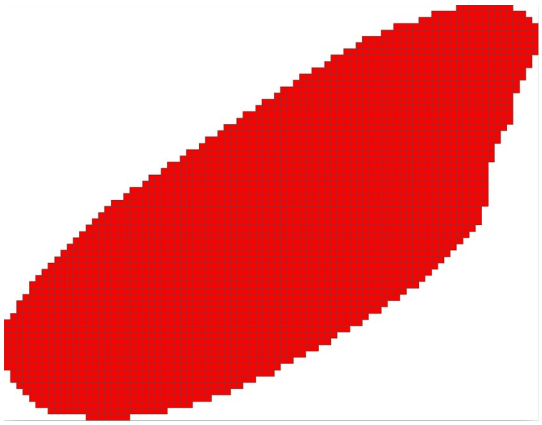
Step 1 - DOWNLOAD AND OBSERVE SEGMENTED IMAGES

Characteristics of rice grains with the naked eye :

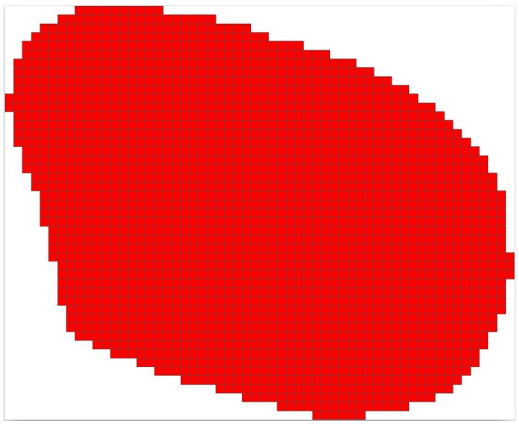
- (A) - **Basmati** rice is long and thin.
- (C) - **Japanese** rice is the opposite of *basmati* rice, it is small and thick, it looks light like a ball in general.
- (B) - **Camargue** rice is between basmati rice and Japanese rice, it is a little long and a little thick.

Step 2 - COUNT GRAINS

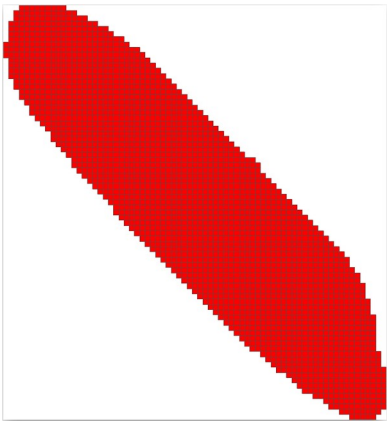
- **Basmati** rice : 141 components detected → 124 after elimination
- **Camargue** rice : 132 components detected → 112 after elimination
- **Japanese** rice : 147 components detected → 138 after elimination



Camargue



Japanese



Basmati

```
typedef Object<DT4_8, DigitalSet> ObjectType; // Digital object type

Board2D aBoard, aBoard2, aBoard3; // we will use these objects to save output
Image image = PGMReader<Image>::importPGM (argv[1]); // you have to provide a correct path

// 1) Create a digital set of proper size
DigitalSet set2d (image.domain());

// 2) Use SetFromImage::append() to populate a digital set from the input image
SetFromImage<Z2i::DigitalSet>::append<Image>(set2d, image, 0, 255);

// 3) Create a digital object from the digital set
ObjectType set( dt4_8, set2d );
std::vector< ObjectType > objects; // All conected components are going to be stored in it
std::back_insert_iterator< std::vector< ObjectType > > inserter( objects ); // Iterator us

// 4) Use method writeComponents to obtain connected components
set.writeComponents(inserter);
std::cout << "Number of components : " << objects.size() << " (before elimination). \n"; /

// 4,2) Elimination border grains
objects.erase(std::remove_if(objects.begin(), objects.end(),
                             [image](ObjectType o) -> bool
                             {
                                 for(auto p : o.pointSet()) {
                                     if(p[0] == image.domain().lowerBound()[0] ||
                                        p[0] == image.domain().upperBound()[0] ||
                                        p[1] == image.domain().lowerBound()[1] ||
                                        p[1] == image.domain().upperBound()[1])
                                        { return true; }
                                 };
                                 return false;
                             }), objects.end());
std::cout << "Number of components : " << objects.size() << " without border grains. \n";
```

Step 3 - EXTRACT DIGITAL OBJECT BOUNDARY

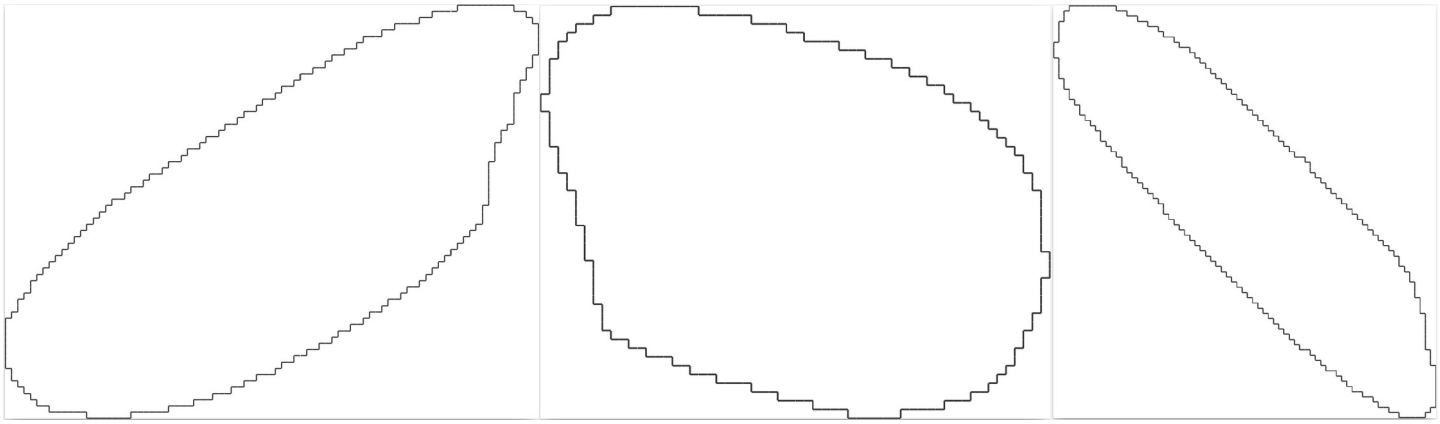
```
template<class T>
Curve getBoundary(T & object)
{
    KSpace kSpace; // Khalimsky space
    // We need to add a margine to prevent situations such that an object touch the bourder of the domain
    kSpace.init( object.domain().lowerBound() - Point(1,1), object.domain().upperBound() + Point(1,1), true);

    // 1) Call Surfaces::findABel() to find a cell which belongs to the border
    std::vector<Z2i::Point> boundaryPoints; // Boundary points are going to be stored here
    SCell aCell = Surfaces<Z2i::KSpace>::findABel(kSpace, object.pointSet(), 10000);

    // 2) Call Surfece::track2DBoundaryPoints to extract the boundary of the object
    Curve boundaryCurve;
    SurfelAdjacency<2> SAdj( true );
    Surfaces<Z2i::KSpace>::track2DBoundaryPoints(boundaryPoints, kSpace, SAdj, object.pointSet(), aCell);

    // 3) Create a curve from a vector
    boundaryCurve.initFromVector(boundaryPoints);
    return boundaryCurve;
}
```

```
// 5 (Step 3) boundary
Curve bondary = getBoundary(objects[0]);
aBoard2 << bondary;
```



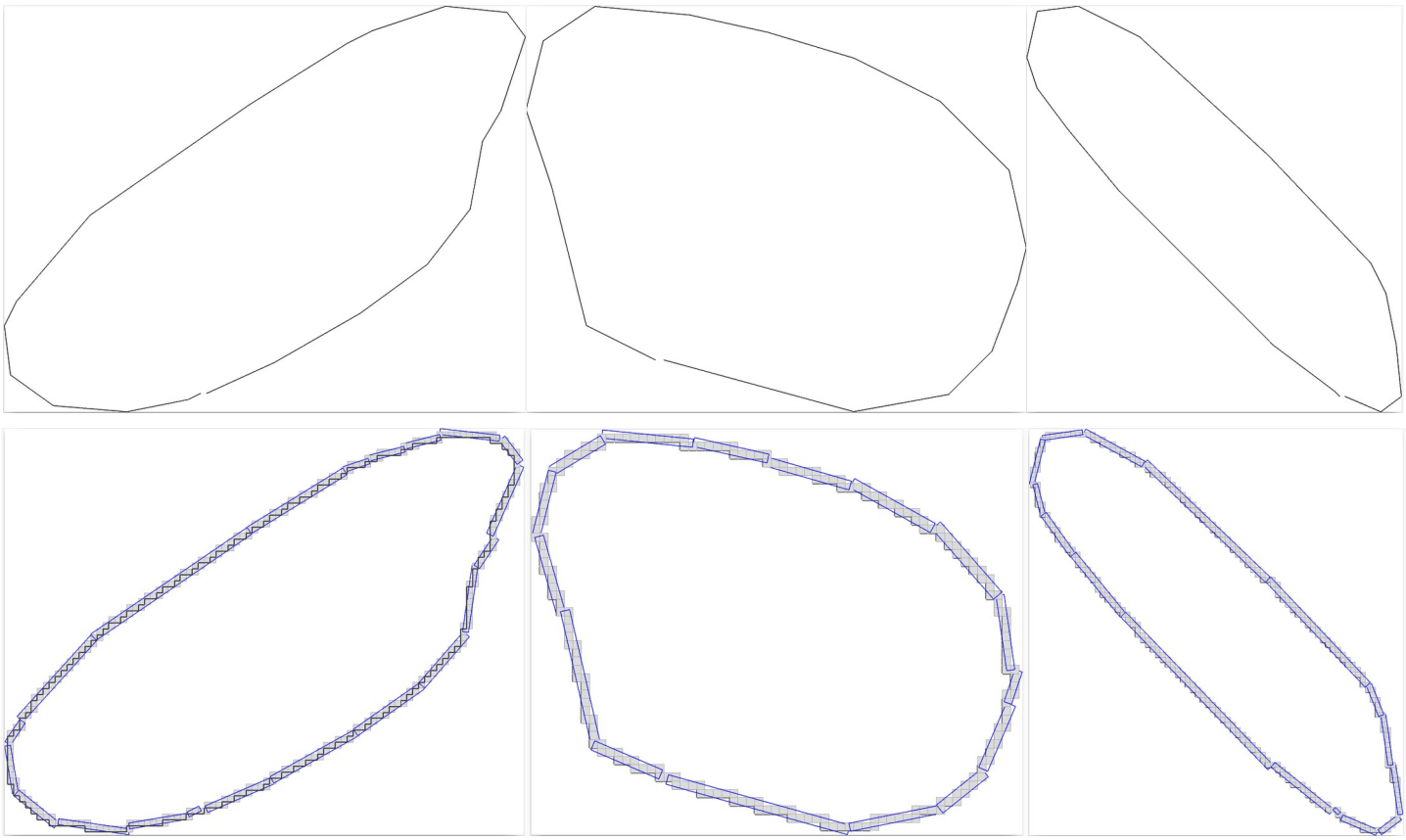
Camargue *Japanese* *Basmati*

Step 4 - POLYGONIZE DIGITAL OBJECT BOUNDARY

```
// 6 (Step 4)
typedef Curve::PointsRange Range; // Container of digital points
typedef Range::ConstIterator ConstIterator; // Iterator on the container
typedef StandardDSS4Computer<ConstIterator> SegmentComputer; // StandardDSS4 computer
typedef GreedySegmentation<SegmentComputer> Segmentation;

Range range = bondary.getPointsRange(); // Construction of the computer
SegmentComputer recognitionAlgorithm; // Segmentation
Segmentation theSegmentation(range.begin(), range.end(), recognitionAlgorithm);

string styleName = "";
for ( Segmentation::SegmentComputerIterator it = theSegmentation.begin(), itEnd = theSegmentation.end(); it != itEnd; ++it )
{
  aBoard3 << SetMode( "ArithmeticalDSS", "Points" ) << it->primitive(); // Cube line (optional)
  aBoard3 << SetMode( "ArithmeticalDSS", "BoundingBox" ) // Blue rectangle
          << CustomStyle( "ArithmeticalDSS/BoundingBox", new CustomPenColor( Color::Blue ) )
          << it->primitive();
}
```



Camargue *Japanese* *Basmati*

Step 4.2 and 5 - CALCULATE AREA AND PERIMETER

- Average area of **Basmati** rice :
  - With number of grid points → 2249.97
  - With area of the polygon → 2229.69
- Average area of **Camargue** rice :
  - With number of grid points → 2693.56
  - With area of the polygon → 2671.89
- Average area of **Japanese** rice :
  - With number of grid points → 2068.46
  - With area of the polygon → 2048.06

When we look at the means of the two methods for the area, we notice that the polygon method is lower than the grid points method on all grains of rice. Knowing that the polygon method gives more precise calculation of the area than the grid point one, this means that more detail you give, more the estimated area decreases and it gets closer to the true value.

We can conclude that the **multigrid convergence is conserved**.

Calculating the air of a grain of rice is useful to deduce its volume but it does not tell us anything about its shape, so it is an important element but not the main one.

- Average perimeter of **Basmati** rice :
  - With the boundary → 292.048
  - With the perimeter of the polygon → 222.056
- Average perimeter of **Camargue** rice :
  - With the boundary → 272.107
  - With the perimeter of the polygon → 204.839
- Average perimeter of **Japanese** rice :
  - With the boundary → 220.362
  - With the perimeter of the polygon → 165.848

When we look at the means of the two methods for the perimeter, we notice that the polygon method is lower than the boundary method on all grains of rice. Knowing that the polygon method gives more precise calculation of the area than the grid point one, this means that more detail you give, more the estimated area decreases and it gets closer to the true value.

Contrary to the air calculation, we can conclude that the **multigrid convergence is not conserved**, because normally we should have much similar values, the lowering of resolution should not affect the values obtained .

Just like the calculation of the area, the perimeter helps for certain information but alone, it does not allow to break a grain of rice

```cpp
int v1_x, v1_y, v2_x, v2_y, area = 0, perimeter = 0;
vector<int> sum_area_1, sum_area_2, sum_perimeter_1, sum_perimeter_2;
// 7 (Step 4-5)
for(auto o : objects)
{
    Curve b = getBoundary(o);
    Range r = b.getPointsRange();
    SegmentComputer recognitionAlgorithm;
    Segmentation theSegmentation(r.begin(), r.end(), recognitionAlgorithm);
    auto it0 = theSegmentation.begin();
    area = 0;
    perimeter = sqrt(pow(it0->front()[0] - it0->back()[0] , 2) + pow(it0->front()[1] - it0->back()[1], 2));

    for ( Segmentation::SegmentComputerIterator it = ++it0, itEnd = theSegmentation.end(); it != itEnd; ++it )
    {
        v1_x = it->back()[0]  - it0->back()[0];
        v1_y = it->back()[1]  - it0->back()[1];
        v2_x = it->front()[0] - it0->back()[0];
        v2_y = it->front()[1] - it0->back()[1];
        area += (v1_x * v2_y - v2_x * v1_y);
        perimeter += sqrt(pow(it->front()[0] - it->back()[0] , 2) + pow(it->front()[1] - it->back()[1], 2));
    }
    sum_area_1.emplace_back(o.size());
    sum_area_2.emplace_back(area / 2);
    sum_perimeter_1.emplace_back(b.size());
    sum_perimeter_2.emplace_back(perimeter);
}
```

```cpp
std::cout << "-----------------------------------------\nAverage of the area of all grains of rice: \n";
std::cout << " - with number of grid points : " << accumulate(sum_area_1.begin(), sum_area_1.end(), 0.0) / sum_area_1.size() << endl;
std::cout << " - with area of the polygon : " << accumulate(sum_area_2.begin(), sum_area_2.end(), 0.0) / sum_area_2.size() << endl;
std::cout << "-----------------------------------------\n";
std::cout << "Perimeter of the area of all grains of rice: \n";
std::cout << " - with the boundary  : " << accumulate(sum_perimeter_1.begin(), sum_perimeter_1.end(), 0.0) / sum_perimeter_1.size() <<
std::cout << " - with the perimeter of the polygon  : " << accumulate(sum_perimeter_2.begin(), sum_perimeter_2.end(), 0.0) / sum_perim
```

Step 6 - PROPOSE AND CALCULATE CIRCULARITY

For calcul .. I will use this a mathematical formulation : **$4\pi \times \text{Area} / \text{Perimeter}^2$**
This formulation give a number between 1 and 0, the number 1 correspond to a perfect circle and 0 to the opposite.

- Average circularity of **Basmati** rice :
  - With the cells → 0.538879
  - With the polygon → 0.336161
- Average circularity of **Camargue** rice :
  - With the cells → 0.743125
  - With the polygon → 0.456674
- Average circularity of **Japanese** rice :
  - With the cells → 0.866402
  - With the polygon → 0.535404

When we observe the results obtained on the different averages. We deduce that Japanese rice has a much more pronounced round shape than the other two rices, and that basmati rice didn't have a round shape at all.

This validates ur theory at the first step, which gave Japanese rice a ball shape, basmatie rice a rod shape (far from a ball) and that camarge rice would be an in-between.