ImgLib2

Generische Bildverarbeitung in Java

Tobias Pietzsch, Stephan Preibisch & Stephan Saalfeld

Max Planck Institute of Molecular Cell Biology and Genetics, Dresden

Java User Group Saxony, 8. November 2012 http://imglib2.net

Generische Bildverarbeitung in Java

Tobias Pietzsch, Stephan Preibisch & Stephan Saalfeld

Max Planck Institute of Molecular Cell Biology and Genetics, Dresden

Java User Group Saxony, 8. November 2012

Motivation

- Image data sets in the life sciences:
  - *n*-dimensional,
  - multi-modal,
  - excessive size
- Algorithmic concepts from image processing are applicable, but Algorithm implementations are often not re-usable:
  - implemented for fixed dimensionality (often 2d),
  - specific data type,
  - limited image size.
- Goal: code that is independent of image dimensionality, data type, and data storage strategy.

Motivation

- Image data sets in the life sciences:
  - *n*-dimensional,
  - multi-modal,
  - excessive size
- Algorithmic concepts from image processing are applicable, but Algorithm implementations are often not re-usable:
  - implemented for fixed dimensionality (often 2d),
  - specific data type,
  - limited image size.
- Goal: code that is independent of image dimensionality, data type, and data storage strategy.

Find Maximum: ImageJ 2D

```java
/** ImageProcessor efficient two-dimensional */
public float findMax2(final ImageProcessor ip, final int[] location) {
  float max = ip.getf(0);
  int maxIndex = 0;
  final int n = ip.getPixelCount();
  for (int i = 1; i < n; ++i) {
    final float pixel = ip.getf(i);
    if (pixel > max) {
      max = pixel;
      maxIndex = i;
    }
  }
  location[1] = maxIndex / ip.getWidth();
  location[0] = maxIndex - location[1] * ip.getWidth();
  return max;
}
```

Find Maximum: ImageJ 3D

```java
/** ImageProcessor efficient three-dimensional */
public float findMax3(final ImagePlus imp, final int[] location) {
  final int n = imp.getWidth() * imp.getHeight();
  float max = (float) imp.getStack().getVoxel(0, 0, 0);
  int maxIndex = 0, maxZ = 0;
  for (int z = 0; z < imp.getNSlices(); ++z) {
    final ImageProcessor ip = imp.getStack().getProcessor(z + 1);
    for (int i = 0; i < n; ++i) {
      final float pixel = ip.getf(i);
      if (pixel > max) {
        max = pixel;
        maxIndex = i;
        maxZ = z;
      }
    }
  }
  location[2] = maxZ;
  location[1] = maxIndex / imp.getWidth();
  location[0] = maxIndex - location[1] * imp.getWidth();
  return max;
}
```

Find Maximum: ImageJ 4D

```java
/** ImageProcessor efficient four-dimensional */
public float findMax4(final ImagePlus imp, final int[] location) {
  final int n = imp.getWidth() * imp.getHeight();
  float max = imp.getStack().getProcessor(1).getf(0);
  int maxIndex = 0, maxZ = 0, maxT = 0;
  for (int t = 0; t < imp.getNFrames(); ++t) {
    for (int z = 0; z < imp.getNSlices(); ++z) {
      final ImageProcessor ip =
          imp.getStack().getProcessor(imp.getStackIndex(1, z + 1, t + 1));
      for (int i = 0; i < n; ++i) {
        final float pixel = ip.getf(i);
        if (pixel > max) {
          max = pixel;
          maxIndex = i;
          maxZ = z;
          maxT = t;
        }
      }
    }
  }
  location[3] = maxT;
  location[2] = maxZ;
  location[1] = maxIndex / imp.getWidth();
  location[0] = maxIndex - location[1] * imp.getWidth();
  return max;
}
```

Find Maximum: ImgLib2

```java
/** ImgLib2 generic */
public <T extends Comparable<T>> Cursor<T> findMax(
    final IterableInterval<T> img) {
  final Cursor<T> cursor = img.cursor();
  cursor.fwd();
  Cursor<T> max = cursor.copyCursor();
  while (cursor.hasNext())
    if (cursor.next().compareTo(max.get()) > 0) {
      max = cursor.copyCursor();
    }
  return max;
}
```

Find Maximum: ImgLib2 real coordinates

```java
/** ImgLib2 generic real coordinates */
public <T extends Comparable<T>> RealCursor<T> findMax(
    final IterableRealInterval<T> img) {
  final RealCursor<T> cursor = img.cursor();
  cursor.fwd();
  RealCursor<T> max = cursor.copyCursor();
  while (cursor.hasNext())
    if (cursor.next().compareTo(max.get()) > 0) {
      max = cursor.copyCursor();
    }
  return max;
}
```

ImgLib2

Library for *n*-dimensional data representation and manipulation.

Goals:

- Dimensionality-, type-, and storage-independent algorithms.
- Decouple algorithm development and data management.
- Extensibility (adding algorithms, pixel types, storage strategies).
- Adaptability (to existing data structures).

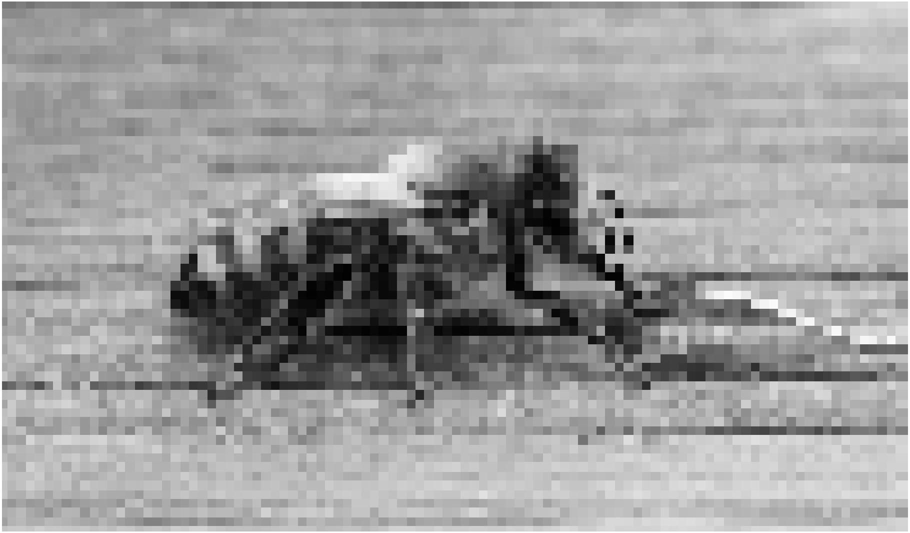
ImgLib2

- ImageJ2
- Fiji
- Knime Image Processing (KNIP)
- Omero

Outline

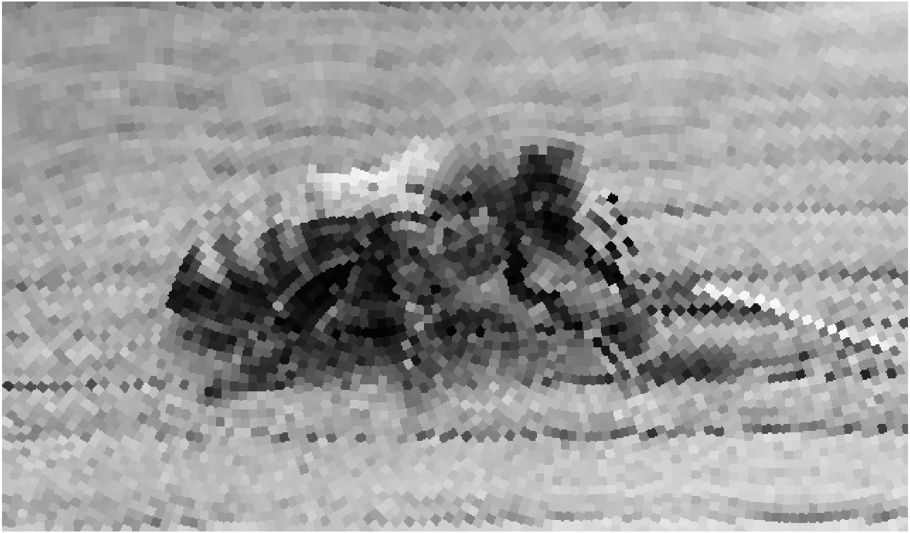
1. What is an Image in ImgLib2?
2. Architecture
   - Type Hierarchy - Pixels
   - Accessors - Image Access Patterns
   - Accessibles - Images
   - Virtualized Pixel Access
3. Performance

What is an Image in ImgLib2?

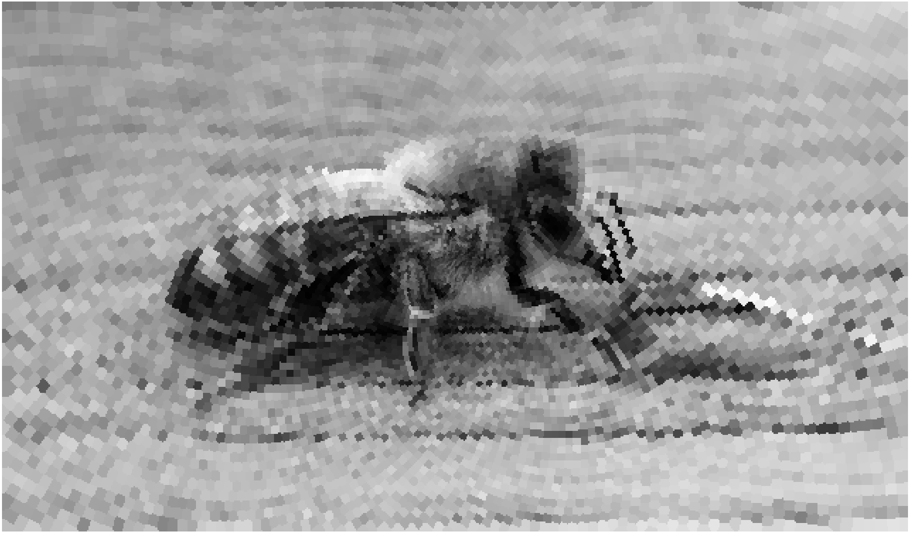


What is an Image in ImgLib2?

What is an Image in ImgLib2?



What is an Image in ImgLib2?



What is an Image in ImgLib2?

$$f : \Omega \to \mathbb{T} \quad \text{with} \quad \Omega \subset \mathbb{R}^n$$

- Arbitrary co-domain $\mathbb{T}$.
- Bounded or un-bounded domain Ω.
- Integer or real coordinates.
- Discrete (grid or sparsely sampled) or continuous domain.

Examples:

- 1D, 2D, . . . , *n*D pixel image.
- interpolated pixel image.
- (interpolated) sparse *n*D sample set.
- virtualized view into another image (transformed, sliced, . . .).
- procedurally generated image.

. . .

What is an Image in ImgLib2?

$$f : \Omega \to \mathbb{T} \quad \text{with} \quad \Omega \subset \mathbb{R}^n$$

- Arbitrary co-domain $\mathbb{T}$.
- Bounded or un-bounded domain Ω.
- Integer or real coordinates.
- Discrete (grid or sparsely sampled) or continuous domain.

Examples:

- 1D, 2D, . . . , *n*D pixel image.
- interpolated pixel image.
- (interpolated) sparse *n*D sample set.
- virtualized view into another image (transformed, sliced, . . .).
- procedurally generated image.

. . .

What is an Image in ImgLib2?

$$f : \Omega \to \mathbb{T} \quad \text{with} \quad \Omega \subset \mathbb{R}^n$$

- Arbitrary co-domain $\mathbb{T}$.
- Bounded or un-bounded domain Ω.
- Integer or real coordinates.
- Discrete (grid or sparsely sampled) or continuous domain.

Examples:

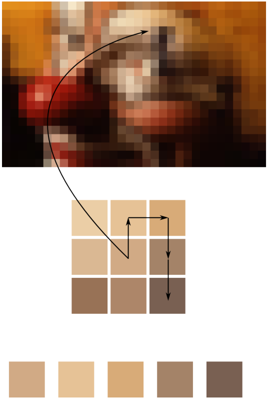
- 1D, 2D, . . . , *n*D pixel image.
- interpolated pixel image.
- (interpolated) sparse *n*D sample set.
- virtualized view into another image (transformed, sliced, . . .).
- procedurally generated image.

  . . .

Architecture

Main abstractions:

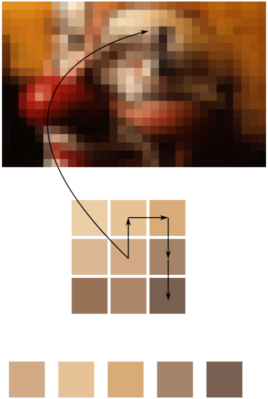
- Accessible ("Image")
  - Provides Accessors.
  - May provide bounds.
- Accessor
  - Is moved across the image.
  - Provides access to Types.
- Type ("Pixel value")
  - Represents sample value $\in \mathbb{T}$.
  - Operations on $\mathbb{T}$.



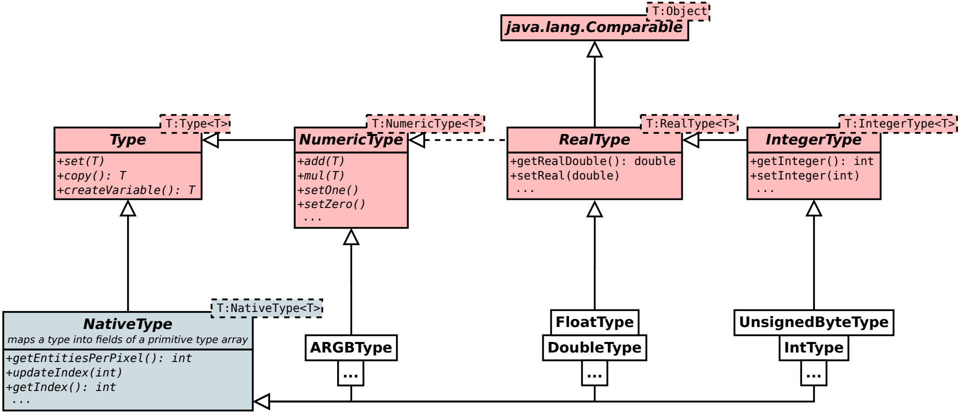
Architecture

Main abstractions:

- Accessible (“Image”)
  - Provides Accessors.
  - May provide bounds.
- Accessor
  - Is moved across the image.
  - Provides access to Types.
- Type (“Pixel value”)
  - Represents sample value $\in \mathbb{T}$.
  - Operations on $\mathbb{T}$.



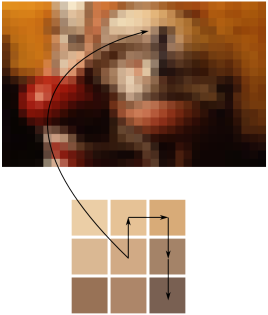
Types (Pixels)



Architecture

Main abstractions:

- Accessible (“Image”)
  - Provides Accessors.
  - May provide bounds.
- Accessor
  - Is moved across the image.
  - Provides access to Types.
- Type (“Pixel value”)
  - Represents sample value $\in \mathbb{T}$.
  - Operations on $\mathbb{T}$.



Access Patterns - Random Access

Random Access:

- Retrieve pixels at specific coordinates.

```
RandomAccess<T> access = image.randomAccess();
access.setPosition(new long[] {100, 100});
access.fwd(1);
...
T value = access.get();
...
```

Access Patterns - Random Access

Random Access:

- Retrieve pixels at specific coordinates.

```
RealRandomAccess<T> access = image.realRandomAccess();
access.setPosition(new double[] {42.1, 0.783});
access.fwd(1);
...
T value = access.get();
...
```

Access Patterns - Iteration

Iteration:

- Visit every pixel once.
- Iteration order is irrelevant.

```
RandomAccess<T> access = image.randomAccess();
for( long x = minX; x <= maxX; ++x ) {
  for( long y = minY; y <= maxY; ++y ) {
    for( long z = minZ; z <= maxZ; ++z ) {
      for( long t = minT; t <= maxT; ++t ) {
        for( long c = minC; c <= maxC; ++c ) {
          access.setPosition(new long[] {x,y,z,t,c});
          T value = access.get();
          ...
        }
      }
    }
  }
}
```

Access Patterns - Iteration

Iteration:

- Visit every pixel once.
- Iteration order is irrelevant.

```
Cursor<T> cursor = image.cursor();
while (cursor.hasNext()) {
  T value = cursor.next();
  ...
}
```

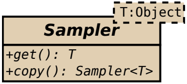
Access Patterns - Iteration

Iteration:

- Visit every pixel once.
- Iteration order is irrelevant.

```
for (T value : image ) {
   ...
}
```

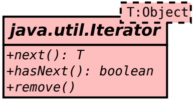
Accessors



T:Object

Sampler

+get(): T
+copy(): Sampler<T>

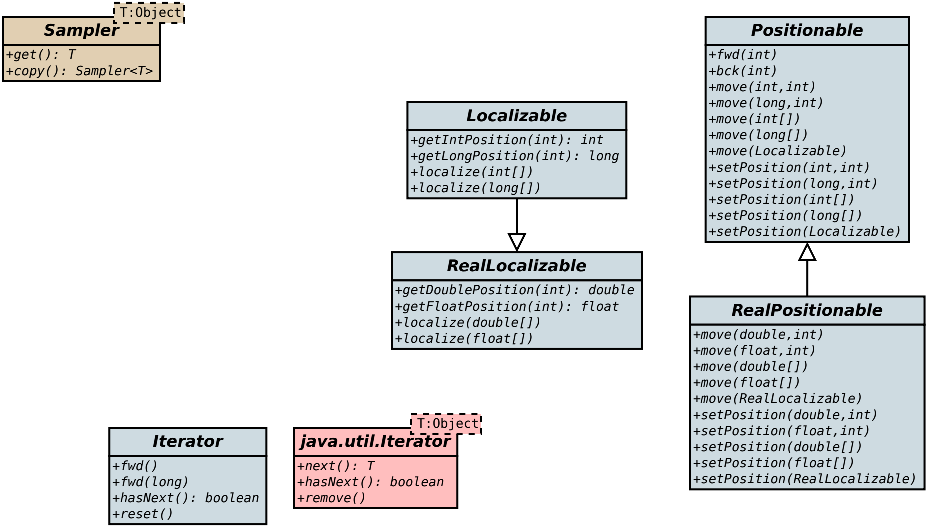


T:Object

java.util.Iterator

+next(): T
+hasNext(): boolean
+remove()

Accessors



Sampler T:Object

+get(): T

+copy(): Sampler<T>

Localizable

+getIntPosition(int): int

+getLongPosition(int): long

+localize(int[])

+localize(long[])

RealLocalizable

+getDoublePosition(int): double

+getFloatPosition(int): float

+localize(double[])

+localize(float[])

Positionable

+fwd(int)

+bck(int)

+move(int,int)

+move(long,int)

+move(int[])

+move(long[])

+move(Localizable)

+setPosition(int,int)

+setPosition(long,int)

+setPosition(int[])

+setPosition(long[])

+setPosition(Localizable)

RealPositionable

+move(double,int)

+move(float,int)

+move(double[])

+move(float[])

+move(RealLocalizable)

+setPosition(double,int)

+setPosition(float,int)

+setPosition(double[])

+setPosition(float[])

+setPosition(RealLocalizable)

Iterator

+fwd()

+fwd(long)

+hasNext(): boolean

+reset()

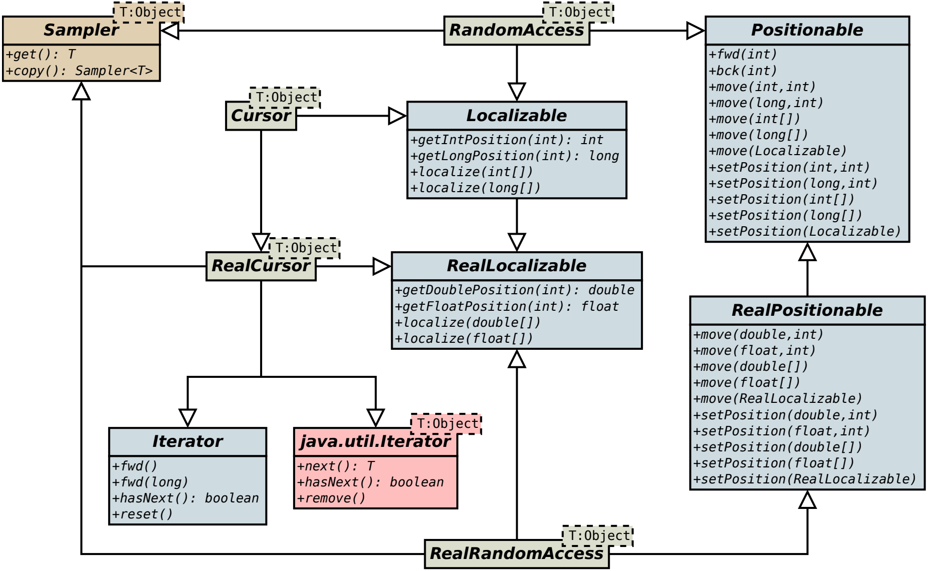
java.util.Iterator T:Object

+next(): T

+hasNext(): boolean

+remove()

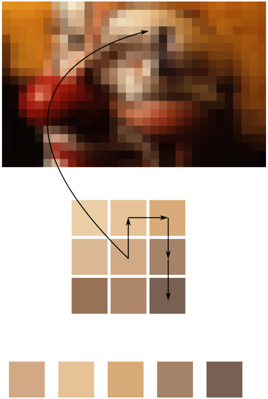
Accessors


T:Object
Sampler
+get(): T
+copy(): Sampler<T>
T:Object
RandomAccess
Positionable
+fwd(int)
+bck(int)
+move(int,int)
+move(long,int)
+move(int[])
+move(long[])
+move(Localizable)
+setPosition(int,int)
+setPosition(long,int)
+setPosition(int[])
+setPosition(long[])
+setPosition(Localizable)
T:Object
Cursor
Localizable
+getIntPosition(int): int
+getLongPosition(int): long
+localize(int[])
+localize(long[])
T:Object
RealCursor
RealLocalizable
+getDoublePosition(int): double
+getFloatPosition(int): float
+localize(double[])
+localize(float[])
RealPositionable
+move(double,int)
+move(float,int)
+move(double[])
+move(float[])
+move(RealLocalizable)
+setPosition(double,int)
+setPosition(float,int)
+setPosition(double[])
+setPosition(float[])
+setPosition(RealLocalizable)
Iterator
+fwd()
+fwd(long)
+hasNext(): boolean
+reset()
T:Object
java.util.Iterator
+next(): T
+hasNext(): boolean
+remove()
T:Object
RealRandomAccess

Architecture

Main abstractions:

- Accessible (“Image”)
  - Provides Accessors.
  - May provide bounds.
- Accessor
  - Is moved across the image.
  - Provides access to Types.
- Type (“Pixel value”)
  - Represents sample value $\in \mathbb{T}$.
  - Operations on $\mathbb{T}$.

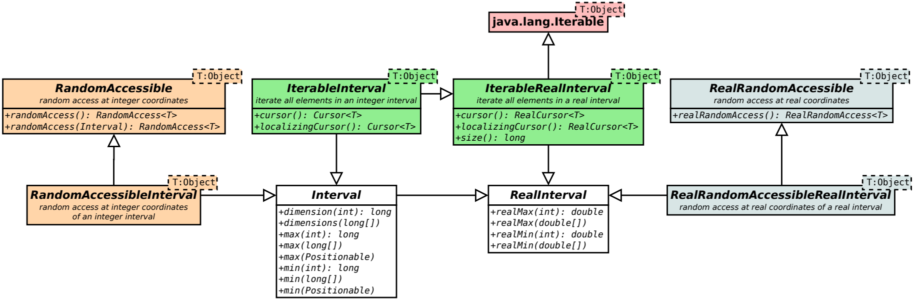


What is an Image in ImgLib2?

$$f : \Omega \rightarrow \mathbb{T} \quad \text{with} \quad \Omega \subseteq \mathbb{R}^n$$

$\Omega = \mathbb{R}^n$	RealRandomAccessible<T>
$\Omega = [\vec{min}, \vec{max}]; \vec{min}, \vec{max} \in \mathbb{R}^n$	RealRandomAccessibleRealInterval<T>
$\Omega = \{\vec{x}_1 \ldots \vec{x}_n\}; \vec{x}_i \in \mathbb{R}^n$	IterableRealInterval<T>
$\Omega = \mathbb{Z}^n$	RandomAccessible<T>
$\Omega = [\vec{min}, \vec{max}]; \vec{min}, \vec{max} \in \mathbb{Z}^n$	RandomAccessibleInterval<T>
$\Omega = \{\vec{x}_1 \ldots \vec{x}_n\}; \vec{x}_i \in \mathbb{Z}^n$	IterableInterval<T>

Accessibles (Collections)



T:Object
java.lang.Iterable

T:Object
RandomAccessible
random access at integer coordinates
+randomAccess(): RandomAccess<T>
+randomAccess(Interval): RandomAccess<T>

T:Object
IterableInterval
iterate all elements in an integer interval
+cursor(): Cursor<T>
+localizingCursor(): Cursor<T>

T:Object
IterableRealInterval
iterate all elements in a real interval
+cursor(): RealCursor<T>
+localizingCursor(): RealCursor<T>
+size(): long

T:Object
RealRandomAccessible
random access at real coordinates
+realRandomAccess(): RealRandomAccess<T>

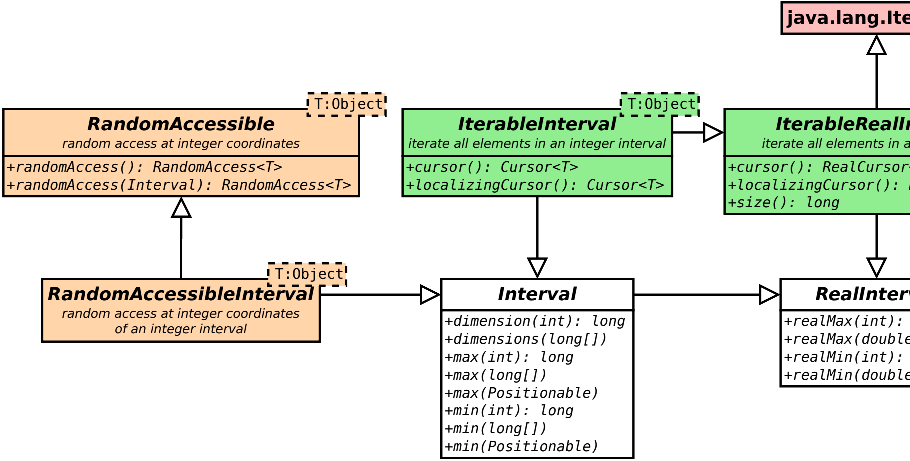
T:Object
RandomAccessibleInterval
random access at integer coordinates
of an integer interval

Interval
+dimension(int): long
+dimensions(long[])
+max(int): long
+max(long[])
+max(Positionable)
+min(int): long
+min(long[])
+min(Positionable)

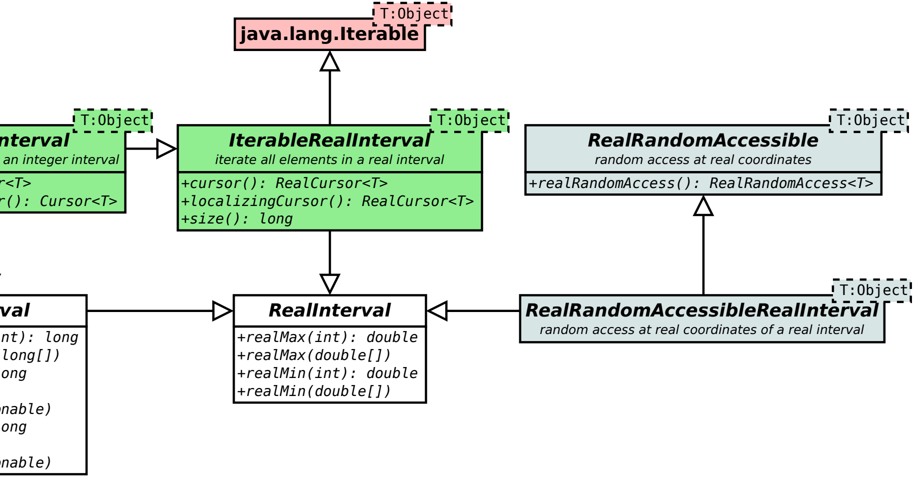
RealInterval
+realMax(int): double
+realMax(double[])
+realMin(int): double
+realMin(double[])

T:Object
RealRandomAccessibleRealInterval
random access at real coordinates of a real interval

Accessibles (Collections)



Accessibles (Collections)



T:Object
java.lang.Iterable

T:Object
nterval
an integer interval
r<T>
r(): Cursor<T>

T:Object
IterableRealInterval
iterate all elements in a real interval
+cursor(): RealCursor<T>
+localizingCursor(): RealCursor<T>
+size(): long

T:Object
RealRandomAccessible
random access at real coordinates
+realRandomAccess(): RealRandomAccess<T>

val
nt): long
long[])
ong
nable)
ong
nable)

RealInterval
+realMax(int): double
+realMax(double[])
+realMin(int): double
+realMin(double[])

T:Object
RealRandomAccessibleRealInterval
random access at real coordinates of a real interval

Accessibles - Neighborhoods

Accessibles are not restricted to contain Types.

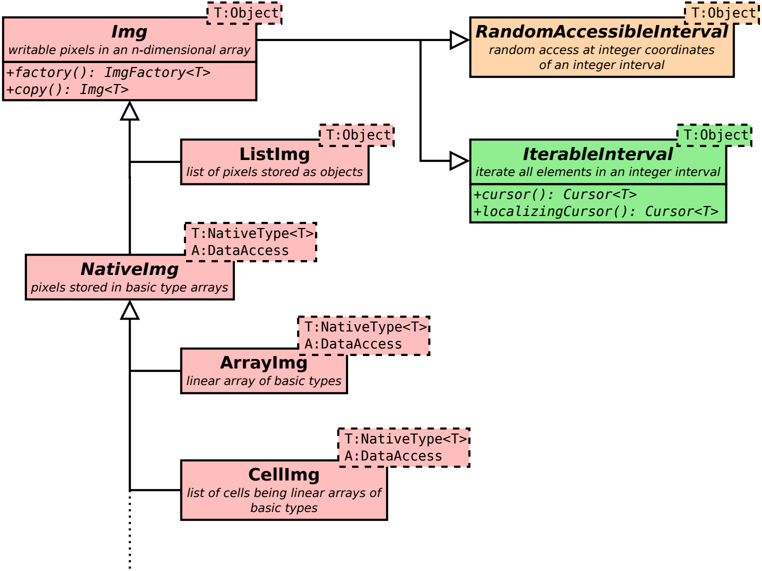
```
IterableInterval<T> image;
for (T value : image ) {
   ...
}
```

Accessibles - Neighborhoods

Accessibles are not restricted to contain Types.

```
IterableInterval<Neighborhood<T>> neighborhoods;
for (Neighborhood<T> neighborhood : neighborhoods) {
  for (T value : neighborhood ) {
    ...
  }
}
```

Pixel Images



T:Object

Img

writable pixels in an n-dimensional array

+factory(): ImgFactory<T>
+copy(): Img<T>

T:Object

ListImg

list of pixels stored as objects

T:NativeType<T>
A:DataAccess

NativeImg

pixels stored in basic type arrays

T:NativeType<T>
A:DataAccess

ArrayImg

linear array of basic types

T:NativeType<T>
A:DataAccess

CellImg

list of cells being linear arrays of basic types

T:Object

RandomAccessibleInterval

random access at integer coordinates of an integer interval

T:Object

IterableInterval

iterate all elements in an integer interval

+cursor(): Cursor<T>
+localizingCursor(): Cursor<T>

Virtualized Sample Access

$$f : \Omega \to \mathbb{T} \quad \text{with} \quad \Omega \subset \mathbb{R}^n$$

Views:

transparent, on-the-fly coordinate transformation

$$\begin{aligned} g : \Omega' \to \Omega \\ f \circ g = f' : \Omega' \to \mathbb{T} \end{aligned}$$

Converters:

transparent, on-the-fly value transformation

$$\begin{aligned} h : \mathbb{T} \to \mathbb{T}' \\ h \circ f = f' : \Omega \to \mathbb{T}' \end{aligned}$$

Virtualized Sample Access

```
RandomAccessibleInterval<UnsignedByteType> img;
```

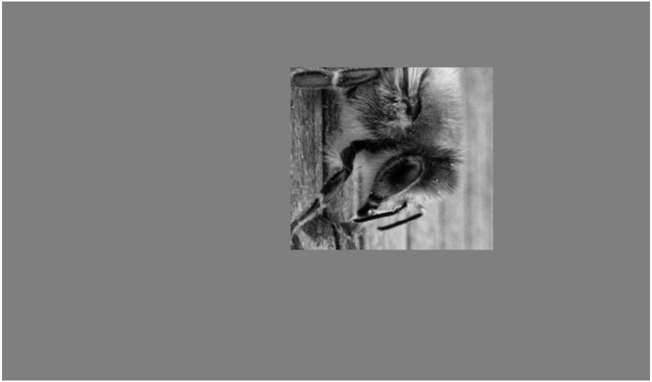
Virtualized Sample Access

```
Interval interval;
RandomAccessibleInterval<UnsignedByteType> cropped =
  Views.offsetInterval(img, interval);
```

Virtualized Sample Access

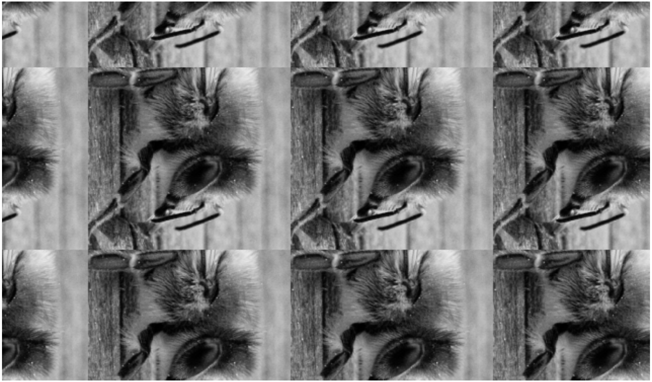
```
RandomAccessibleInterval<UnsignedByteType> rotated =
  Views.rotate(cropped, 0, 1);
```

Virtualized Sample Access



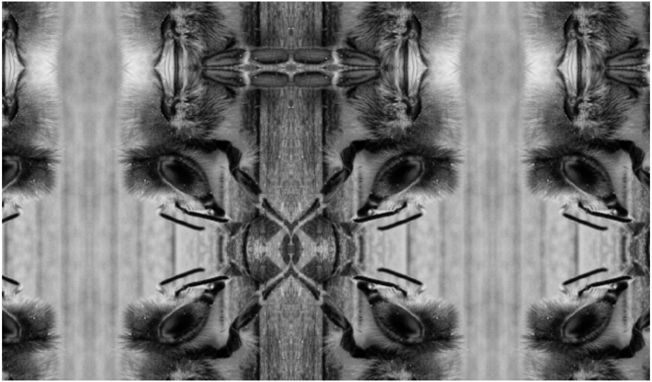
```
RandomAccessible<UnsignedByteType> extended =
  Views.extendValue(rotated, new UnsignedByteType(127));
```

Virtualized Sample Access



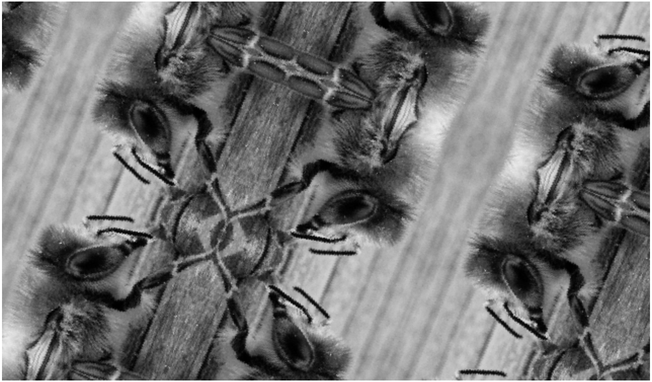
```
RandomAccessible<UnsignedByteType> extended =
  Views.extendPeriodic(rotated);
```

Virtualized Sample Access



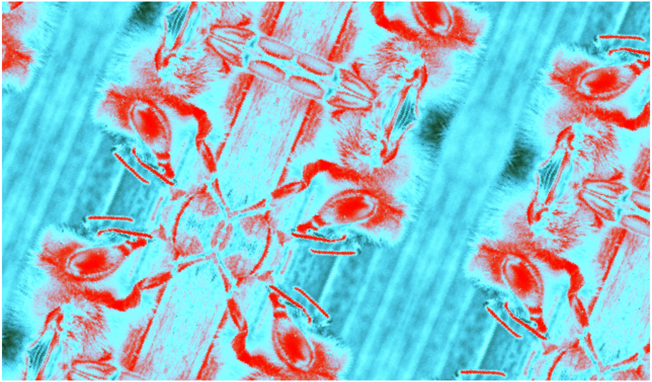
```
RandomAccessible<UnsignedByteType> extended =
  Views.extendMirrorSingle(rotated);
```

Virtualized Sample Access



```
RealRandomAccessible<UnsignedByteType> interpolated =
  Views.interpolate(extended,
    new NearestNeighborInterpolatorFactory<UnsignedByteType>());

AffineTransform affine;
RandomAccessible<UnsignedByteType> transformed =
  RealViews.affine(interpolated, affine);
```

Virtualized Sample Access



```
Converter<UnsignedByteType, ARGBType> lut;
RandomAccessible<ARGBType> converted =
  Converters.convert(transformed, lut, new ARGBType());
```

Performance Benchmarks

- Trivial operation: Invert all pixels
    - Iterate over all pixel (values).
- Complex operation: Compute center of mass
    - Sum *n*D coordinates weighted by pixel values.
    - Access coordinates and values.

- Java primitive type arrays (`byte[]/float[]`)
- ImageJ ImagePlus
- ImgLib2 ArrayImg
- ImgLib2 CellImg
- templated C++

Performance Benchmarks

- Trivial operation: Invert all pixels
  - Iterate over all pixel (values).
- Complex operation: Compute center of mass
  - Sum *n*D coordinates weighted by pixel values.
  - Access coordinates and values.

- Java primitive type arrays (`byte[]`/`float[]`)
- ImageJ ImagePlus
- ImgLib2 ArrayImg
- ImgLib2 CellImg
- templated C++

Benchmark A: trivial operation

Java primitive array

```java
private static void invert(final float[] img) {
  for (int i = 0; i < img.length; i++) {
    img[i] = -img[i];
  }
}
```

Benchmark A: trivial operation

ImageJ

```java
private static void invert(final ImagePlus imp) {
  final ImageStack stack = imp.getStack();
  final int numSlices = stack.getSize();
  for (int s = 1; s <= numSlices; ++s) {
    final ImageProcessor ip = stack.getProcessor(s);
    final int size = ip.getPixelCount();
    for (int i = 0; i < size; i++)
      ip.setf(i, -ip.getf(i));
  }
}
```

Benchmark A: trivial operation

ImgLib2

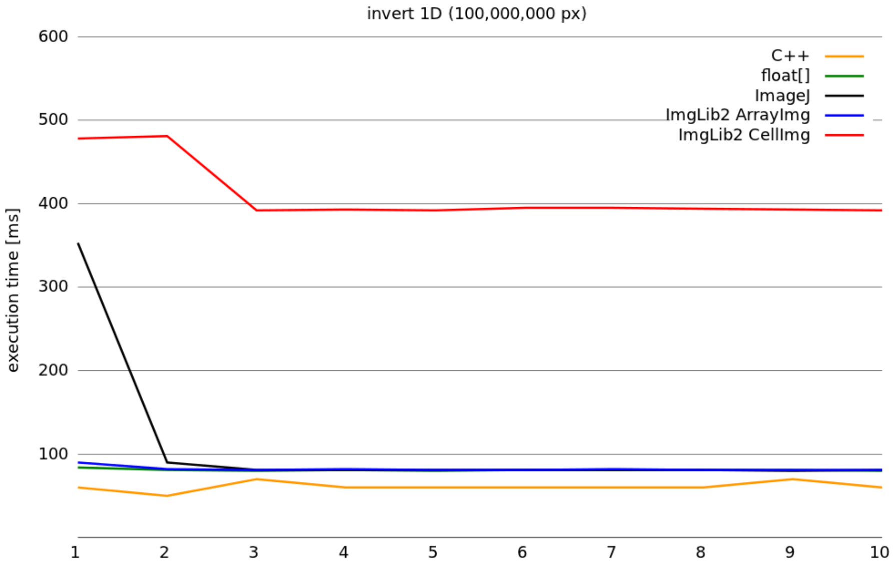
```java
static private <T extends RealType<T>> void invert(
    final IterableInterval<T> img) {
  for (final T t : img)
    t.mul(-1);
}
```

Benchmark A: trivial operation

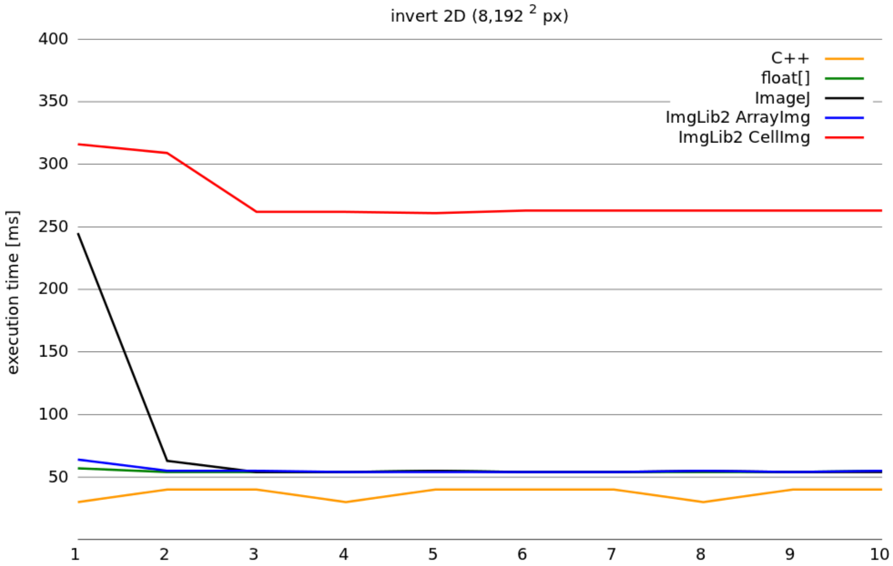
C++

```cpp
template<class T>
void invert(T* img, unsigned long long size) {
  for (unsigned long long i = 0; i < size; ++i) {
    img[i] = -img[i];
  }
}
```

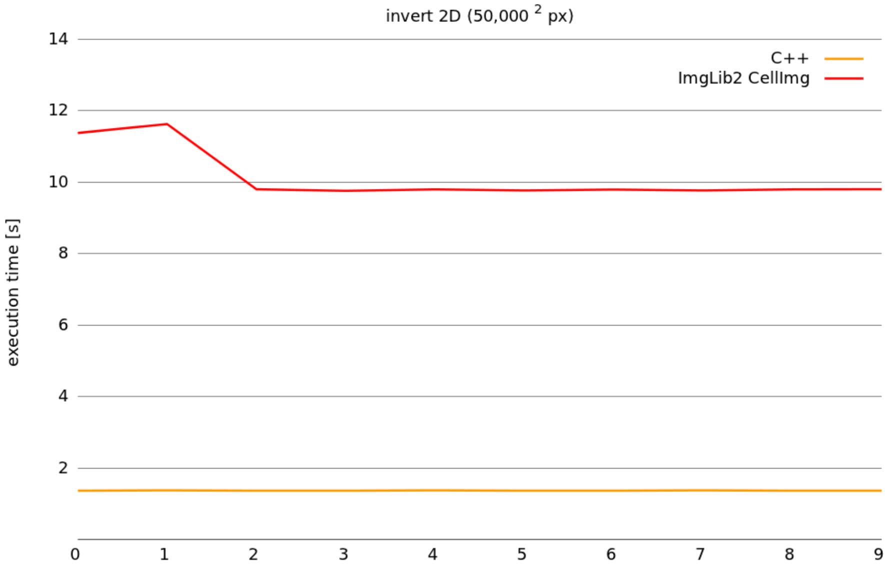
Benchmark A: trivial operation



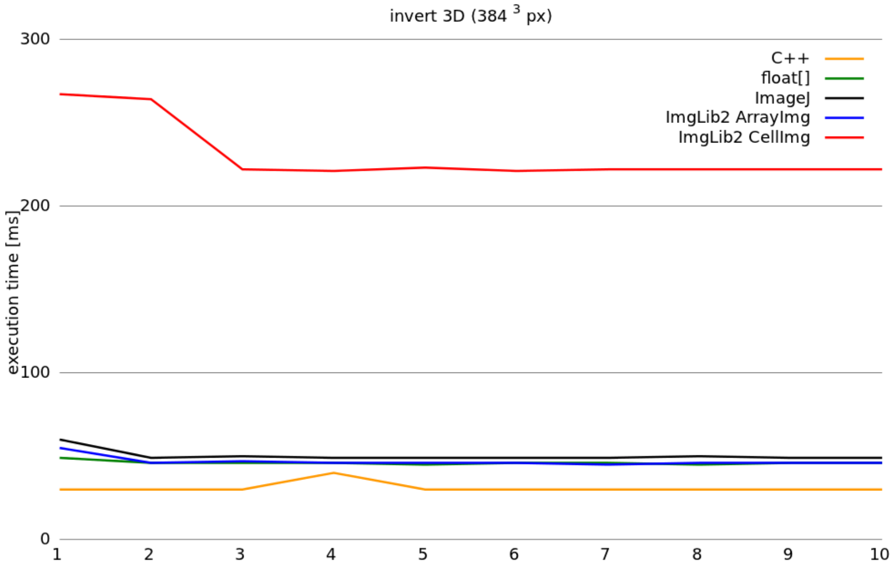
Benchmark A: trivial operation



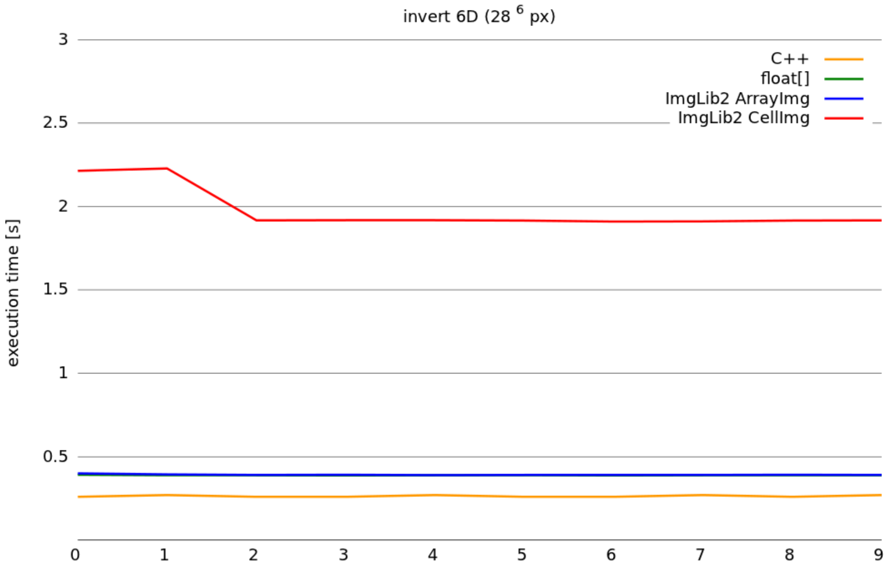
Benchmark A: trivial operation



Benchmark A: trivial operation



Benchmark A: trivial operation



Benchmark B: complex operation

Java primitive array (1D)

```java
private static double[] centerOfMass(
    final byte[] img, final int s0) {
  final RealSum c0 = new RealSum(), sum = new RealSum();
  for (int d0 = 0; d0 < s0; ++d0) {
    final double value = img[d0] & 0xff;
    c0.add(value * d0);
    sum.add(value);
  }
  final double s = sum.getSum();
  return new double[] { c0.getSum() / s };
}
```

Benchmark B: complex operation

Java primitive array (2D)

```java
private static double[] centerOfMass(
    final byte[] img, final int s0, final int s1) {
  final RealSum c0 = new RealSum(), c1 = new RealSum(),
      sum = new RealSum();
  int i = 0;
  for (int d1 = 0; d1 < s1; ++d1)
    for (int d0 = 0; d0 < s0; ++d0) {
      final double value = img[i++] & 0xff;
      c0.add(value * d0);
      c1.add(value * d1);
      sum.add(value);
    }
  final double s = sum.getSum();
  return new double[] { c0.getSum() / s, c1.getSum() / s };
}
```

Benchmark B: complex operation

Java primitive array (3D)

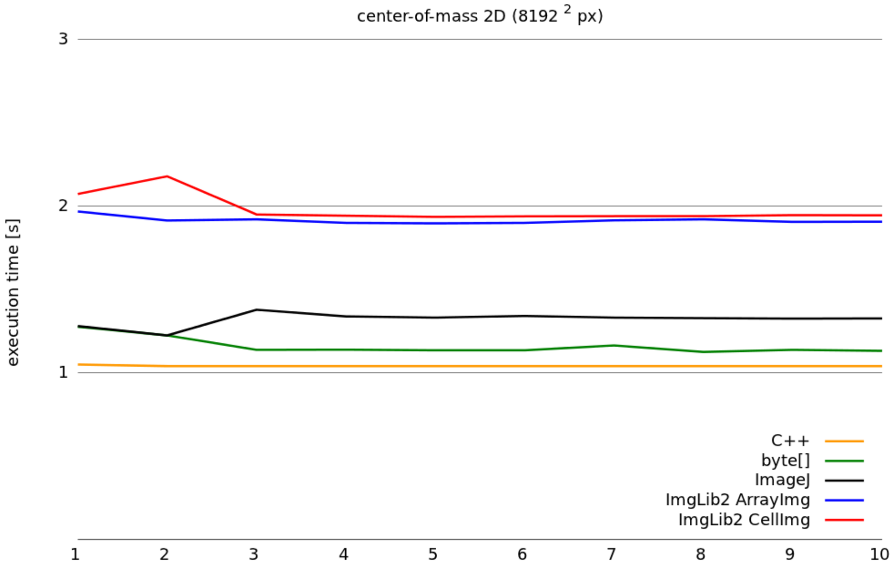
```java
private static double[] centerOfMass(
    final byte[] img, final int s0, final int s1, final int s2) {
  final RealSum c0 = new RealSum(), c1 = new RealSum(),
      c2 = new RealSum(), sum = new RealSum();
  int i = 0;
  for (int d2 = 0; d2 < s2; ++d2)
    for (int d1 = 0; d1 < s1; ++d1)
      for (int d0 = 0; d0 < s0; ++d0) {
        final double value = img[i++] & 0xff;
        c0.add(value * d0);
        c1.add(value * d1);
        c2.add(value * d2);
        sum.add(value);
      }
  final double s = sum.getSum();
  return new double[] {
      c0.getSum() / s, c1.getSum() / s, c2.getSum() / s };
}
```

Benchmark B: complex operation

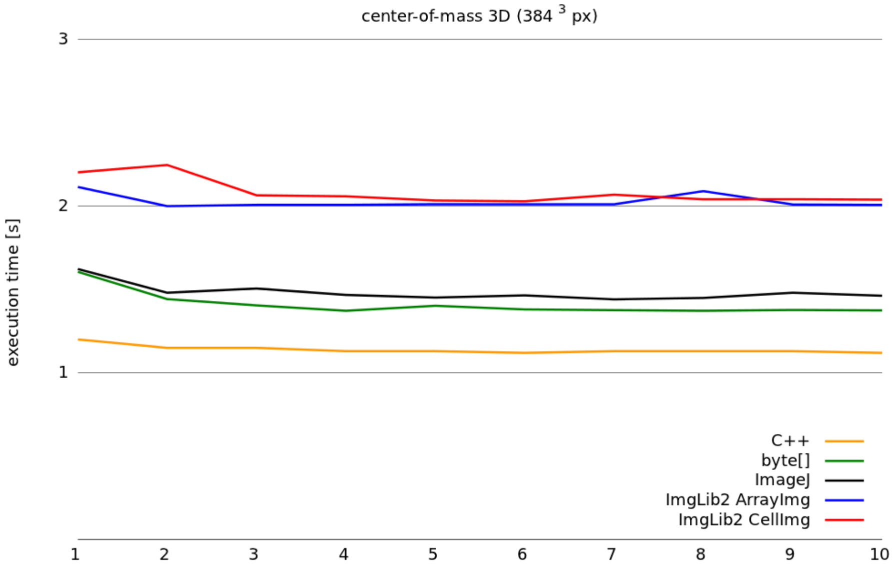
ImgLib2

```java
static private <T extends RealType<T>> double[] centerOfMass(
    final IterableInterval<T> img) {
  final RealSum[] c = new RealSum[img.numDimensions()];
  for (int d = 0; d < c.length; ++d)
    c[d] = new RealSum();
  final RealSum s = new RealSum();
  final Cursor<T> cursor = img.localizingCursor();
  while (cursor.hasNext()) {
    final double value = cursor.next().getRealDouble();
    s.add(value);
    for (int d = 0; d < c.length; ++d)
      c[d].add(cursor.getDoublePosition(d) * value);
  }
  final double[] centerOfMass = new double[c.length];
  final double sum = s.getSum();
  for (int d = 0; d < c.length; ++d)
    centerOfMass[d] = c[d].getSum() / sum;
  return centerOfMass;
}
```

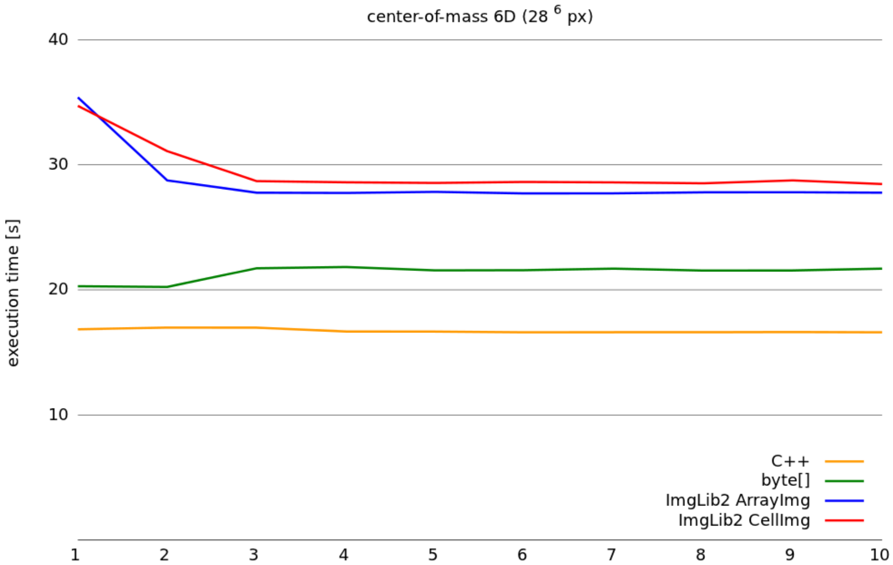
Benchmark B: complex operation



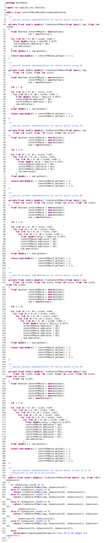
Benchmark B: complex operation



Benchmark B: complex operation

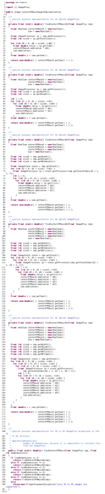


Benchmark B: complex operation



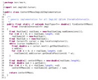
```java
package benchmark;

import net.imglib2.util.RealSum;

public class CenterOfMassByteNativeImplementation
{
  /**
   * special purpose implementation for native byte[] array 1D.
   */
  private final static double[] findCenterOfMass(final byte[] img, final int
size0)
  {
    final RealSum centerOfMass0 = new RealSum(),
                  sum = new RealSum();

    for (int d0 = 0; d0 < size0; ++d0) {
      final double value = img[d0] & 0xff;
      centerOfMass0.add(value * d0);
      sum.add(value);
    }
    final double s = sum.getSum();

    return new double[] { centerOfMass0.getSum() / s };
  }

  /**
   * special purpose implementation for native byte[] array 2D.
   */
  private final static double[] findCenterOfMass(final byte[] img,
      final int size0, final int size1)
  {
    final RealSum centerOfMass0 = new RealSum(),
                  centerOfMass1 = new RealSum(),
                  sum = new RealSum();

    int i = 0;

    for (int d1 = 0; d1 < size1; ++d1)
      for (int d0 = 0; d0 < size0; ++d0) {
        final double value = img[i++] & 0xff;
        centerOfMass0.add(value * d0);
        centerOfMass1.add(value * d1);
        sum.add(value);
      }
    final double s = sum.getSum();

    return new double[] { centerOfMass0.getSum() / s,
                          centerOfMass1.getSum() / s };
  }

  /**
   * special purpose implementation for native byte[] array 3D.
   */
  private final static double[] findCenterOfMass(final byte[] img,
      final int size0, final int size1, final int size2)
  {
    final RealSum centerOfMass0 = new RealSum(),
                  centerOfMass1 = new RealSum(),
                  centerOfMass2 = new RealSum(),
                  sum = new RealSum();

    int i = 0;

    for (int d2 = 0; d2 < size2; ++d2)
      for (int d1 = 0; d1 < size1; ++d1)
        for (int d0 = 0; d0 < size0; ++d0) {
          final double value = img[i++] & 0xff;
          centerOfMass0.add(value * d0);
          centerOfMass1.add(value * d1);
          centerOfMass2.add(value * d2);
          sum.add(value);
        }
    final double s = sum.getSum();

    return new double[] { centerOfMass0.getSum() / s,
                          centerOfMass1.getSum() / s,
                          centerOfMass2.getSum() / s };
  }

  /**
   * special purpose implementation for native byte[] array 4D.
   */
  private final static double[] findCenterOfMass(final byte[] img,
      final int size0, final int size1, final int size2, final int size3)
  {
    final RealSum centerOfMass0 = new RealSum(),
                  centerOfMass1 = new RealSum(),
                  centerOfMass2 = new RealSum(),
                  centerOfMass3 = new RealSum(),
                  sum = new RealSum();

    int i = 0;

    for (int d3 = 0; d3 < size3; ++d3)
      for (int d2 = 0; d2 < size2; ++d2)
        for (int d1 = 0; d1 < size1; ++d1)
          for (int d0 = 0; d0 < size0; ++d0) {
            final double value = img[i++] & 0xff;
            centerOfMass0.add(value * d0);
            centerOfMass1.add(value * d1);
            centerOfMass2.add(value * d2);
            centerOfMass3.add(value * d3);
            sum.add(value);
          }
    final double s = sum.getSum();

    return new double[] { centerOfMass0.getSum() / s,
                          centerOfMass1.getSum() / s,
                          centerOfMass2.getSum() / s,
                          centerOfMass3.getSum() / s };
  }

  /**
   * special purpose implementation for native byte[] array 5D.
   */
  private final static double[] findCenterOfMass(final byte[] img,
      final int size0, final int size1, final int size2, final int size3,
final int size4)
  {
    final RealSum centerOfMass0 = new RealSum(),
                  centerOfMass1 = new RealSum(),
                  centerOfMass2 = new RealSum(),
                  centerOfMass3 = new RealSum(),
                  centerOfMass4 = new RealSum(),
                  sum = new RealSum();

    int i = 0;

    for (int d4 = 0; d4 < size4; ++d4)
      for (int d3 = 0; d3 < size3; ++d3)
        for (int d2 = 0; d2 < size2; ++d2)
          for (int d1 = 0; d1 < size1; ++d1)
            for (int d0 = 0; d0 < size0; ++d0) {
              final double value = img[i++] & 0xff;
              centerOfMass0.add(value * d0);
              centerOfMass1.add(value * d1);
              centerOfMass2.add(value * d2);
              centerOfMass3.add(value * d3);
              centerOfMass4.add(value * d4);
              sum.add(value);
            }
    final double s = sum.getSum();

    return new double[] { centerOfMass0.getSum() / s,
                          centerOfMass1.getSum() / s,
                          centerOfMass2.getSum() / s,
                          centerOfMass3.getSum() / s,
                          centerOfMass4.getSum() / s };
  }

  /**
   * special purpose implementation for native byte[] array 6D.
   */
  private final static double[] findCenterOfMass(final byte[] img,
      final int size0, final int size1, final int size2, final int size3,
final int size4,
      final int size5)
  {
    final RealSum centerOfMass0 = new RealSum(),
                  centerOfMass1 = new RealSum(),
                  centerOfMass2 = new RealSum(),
                  centerOfMass3 = new RealSum(),
                  centerOfMass4 = new RealSum(),
                  centerOfMass5 = new RealSum(),
                  sum = new RealSum();

    int i = 0;

    for (int d5 = 0; d5 < size5; ++d5)
      for (int d4 = 0; d4 < size4; ++d4)
        for (int d3 = 0; d3 < size3; ++d3)
          for (int d2 = 0; d2 < size2; ++d2)
            for (int d1 = 0; d1 < size1; ++d1)
              for (int d0 = 0; d0 < size0; ++d0) {
                final double value = img[i++] & 0xff;
                centerOfMass0.add(value * d0);
                centerOfMass1.add(value * d1);
                centerOfMass2.add(value * d2);
                centerOfMass3.add(value * d3);
                centerOfMass4.add(value * d4);
                centerOfMass5.add(value * d5);
                sum.add(value);
              }
    final double s = sum.getSum();

    return new double[] { centerOfMass0.getSum() / s,
                          centerOfMass1.getSum() / s,
                          centerOfMass2.getSum() / s,
                          centerOfMass3.getSum() / s,
                          centerOfMass4.getSum() / s,
                          centerOfMass5.getSum() / s };
  }

  /**
   * special purpose implementation for native byte[] arrays 1D to 6D.
   * dispatches to the 1D to 6D versions.
   */
  public final static double[] findCenterOfMass(final byte[] img, final int[]
dimensions)
  {
    if (dimensions.length == 1)
      return findCenterOfMass(img, dimensions[0]);
    else if (dimensions.length == 2)
      return findCenterOfMass(img, dimensions[0], dimensions[1]);
    else if (dimensions.length == 3)
      return findCenterOfMass(img, dimensions[0], dimensions[1], dimensions
[2]);
    else if (dimensions.length == 4)
      return findCenterOfMass(img, dimensions[0], dimensions[1], dimensions
[2],
          dimensions[3]);
    else if (dimensions.length == 5)
      return findCenterOfMass(img, dimensions[0], dimensions[1], dimensions
[2],
          dimensions[3], dimensions[4]);
    else if (dimensions.length == 6)
      return findCenterOfMass(img, dimensions[0], dimensions[1], dimensions
[2],
          dimensions[3], dimensions[4], dimensions[5]);
    else
      throw new IllegalArgumentException("only 1D to 6D images are
supported");
  }
}
```

Java `byte[]` (1D-6D)



```java
package benchmark;

import ij.ImagePlus;

public class CenterOfMassImageJImplementation
{
  /**
   * special purpose implementation for 1D {@link ImagePlus}.
   */
  private final static double[] findCenterOfMass1D(final ImagePlus imp)
  {
    final RealSum centerOfMass0 = new RealSum(),
                  sum = new RealSum();

    final ImageProcessor ip = imp.getProcessor();
    final int size0 = ip.getPixelCount();

    for (int d0 = 0; d0 < size0; ++d0) {
      final double value = ip.getf(d0);
      centerOfMass0.add(value * d0);
      sum.add(value);
    }
    final double s = sum.getSum();

    return new double[] { centerOfMass0.getSum() / s };
  }

  /**
   * special purpose implementation for 2D {@link ImagePlus}.
   */
  private final static double[] findCenterOfMass2D(final ImagePlus imp)
  {
    final RealSum centerOfMass0 = new RealSum(),
                  centerOfMass1 = new RealSum(),
                  sum = new RealSum();

    final ImageProcessor ip = imp.getProcessor();
    final int size0 = ip.getWidth();
    final int size1 = ip.getHeight();

    int i = 0;
    for (int d1 = 0; d1 < size1; ++d1)
      for (int d0 = 0; d0 < size0; ++d0) {
        final double value = ip.getf(i++);
        centerOfMass0.add(value * d0);
        centerOfMass1.add(value * d1);
        sum.add(value);
      }
    final double s = sum.getSum();

    return new double[] { centerOfMass0.getSum() / s,
                          centerOfMass1.getSum() / s };
  }

  /**
   * special purpose implementation for 3D {@link ImagePlus}.
   */
  private final static double[] findCenterOfMass3D(final ImagePlus imp)
  {
    final RealSum centerOfMass0 = new RealSum(),
                  centerOfMass1 = new RealSum(),
                  centerOfMass2 = new RealSum(),
                  sum = new RealSum();

    final int size0 = imp.getWidth();
    final int size1 = imp.getHeight();
    final int size2 = imp.getNChannels();

    final ImageStack stack = imp.getStack();
    for (int d2 = 0; d2 < size2; ++d2) {
      final ImageProcessor ip = stack.getProcessor(imp.getStackIndex(d2 + 1,
1, 1));
      int i = 0;
      for (int d1 = 0; d1 < size1; ++d1)
        for (int d0 = 0; d0 < size0; ++d0) {
          final double value = ip.getf(i++);
          centerOfMass0.add(value * d0);
          centerOfMass1.add(value * d1);
          centerOfMass2.add(value * d2);
          sum.add(value);
        }
    }
    final double s = sum.getSum();

    return new double[] { centerOfMass0.getSum() / s,
                          centerOfMass1.getSum() / s,
                          centerOfMass2.getSum() / s };
  }

  /**
   * special purpose implementation for 4D {@link ImagePlus}.
   */
  private final static double[] findCenterOfMass4D(final ImagePlus imp)
  {
    final RealSum centerOfMass0 = new RealSum(),
                  centerOfMass1 = new RealSum(),
                  centerOfMass2 = new RealSum(),
                  centerOfMass3 = new RealSum(),
                  sum = new RealSum();

    final int size0 = imp.getWidth();
    final int size1 = imp.getHeight();
    final int size2 = imp.getNChannels();
    final int size3 = imp.getNSlices();

    final ImageStack stack = imp.getStack();
    for (int d3 = 0; d3 < size3; ++d3) {
      for (int d2 = 0; d2 < size2; ++d2) {
        final ImageProcessor ip = stack.getProcessor(imp.getStackIndex(d2 +
1, d3 + 1, 1));
        int i = 0;
        for (int d1 = 0; d1 < size1; ++d1)
          for (int d0 = 0; d0 < size0; ++d0) {
            final double value = ip.getf(i++);
            centerOfMass0.add(value * d0);
            centerOfMass1.add(value * d1);
            centerOfMass2.add(value * d2);
            centerOfMass3.add(value * d3);
            sum.add(value);
          }
      }
    }
    final double s = sum.getSum();

    return new double[] { centerOfMass0.getSum() / s,
                          centerOfMass1.getSum() / s,
                          centerOfMass2.getSum() / s,
                          centerOfMass3.getSum() / s };
  }

  /**
   * special purpose implementation for 5D {@link ImagePlus}.
   */
  private final static double[] findCenterOfMass5D(final ImagePlus imp)
  {
    final RealSum centerOfMass0 = new RealSum(),
                  centerOfMass1 = new RealSum(),
                  centerOfMass2 = new RealSum(),
                  centerOfMass3 = new RealSum(),
                  centerOfMass4 = new RealSum(),
                  sum = new RealSum();

    final int size0 = imp.getWidth();
    final int size1 = imp.getHeight();
    final int size2 = imp.getNChannels();
    final int size3 = imp.getNSlices();
    final int size4 = imp.getNFrames();

    final ImageStack stack = imp.getStack();
    for (int d4 = 0; d4 < size4; ++d4) {
      for (int d3 = 0; d3 < size3; ++d3) {
        for (int d2 = 0; d2 < size2; ++d2) {
          final ImageProcessor ip = stack.getProcessor(
              imp.getStackIndex(d2 + 1, d3 + 1, d4 + 1));
          int i = 0;
          for (int d1 = 0; d1 < size1; ++d1)
            for (int d0 = 0; d0 < size0; ++d0) {
              final double value = ip.getf(i++);
              centerOfMass0.add(value * d0);
              centerOfMass1.add(value * d1);
              centerOfMass2.add(value * d2);
              centerOfMass3.add(value * d3);
              centerOfMass4.add(value * d4);
              sum.add(value);
            }
        }
      }
    }
    final double s = sum.getSum();

    return new double[] { centerOfMass0.getSum() / s,
                          centerOfMass1.getSum() / s,
                          centerOfMass2.getSum() / s,
                          centerOfMass3.getSum() / s,
                          centerOfMass4.getSum() / s };
  }

  /**
   * special purpose implementation for 1D to 5D ImagePlus dispatches to the
1D
   * to 5D versions.
   *
   * @param numDimensions
   *          number of dimensions, because it is impossible to retrieve this
   *          from the ImagePlus.
   */
  public final static double[] findCenterOfMass(final ImagePlus imp, final
int numDimensions)
  {
    if (numDimensions == 1)
      return findCenterOfMass1D(imp);
    else if (numDimensions == 2)
      return findCenterOfMass2D(imp);
    else if (numDimensions == 3)
      return findCenterOfMass3D(imp);
    else if (numDimensions == 4)
      return findCenterOfMass4D(imp);
    else if (numDimensions == 5)
      return findCenterOfMass5D(imp);
    else
      throw new IllegalArgumentException("only 1D to 5D images are
supported");
  }
}
```

ImageJ (1D-5D)



```java
package benchmark;

import net.imglib2.Cursor;

public class CenterOfMassImgLib2Implementation
{
  /**
   * generic implementation for all ImgLib2 {@link IterableInterval}s.
   */
  public final static <T extends RealType<T>> double[] findCenterOfMass(
      final IterableInterval<T> img)
  {
    final RealSum[] realSums = new RealSum[img.numDimensions()];
    for (int d = 0; d < realSums.length; ++d)
      realSums[d] = new RealSum();
    final RealSum s = new RealSum();
    final Cursor<T> cursor = img.localizingCursor();
    while (cursor.hasNext()) {
      final double w = cursor.next().getRealDouble();
      s.add(w);
      for (int d = 0; d < realSums.length; ++d)
        realSums[d].add(cursor.getDoublePosition(d) * w);
    }

    final double[] centerOfMass = new double[realSums.length];
    final double sum = s.getSum();
    for (int d = 0; d < realSums.length; ++d)
      centerOfMass[d] = realSums[d].getSum() / sum;

    return centerOfMass;
  }
}
```

ImgLib2
(arbitrary type, container, dimensionality)

Acknowledgements

ImgLib2 is Open Source, licensed under BSD.

Contributors:

Tobias Pietzsch, Stephan Preibisch, Stephan Saalfeld (main authors)

Curtis Rueden (maven/eclipse integration, ImageJ/ImageJ2 integration, Bioformats bindings, display, benchmarks)

Lee Kamentsky (regions of interest, labeling, ImageJ2 integration)

Barry DeZonia (ops: modules for algorithm design, display)

Johannes Schindelin (Fiji/ImageJ2 integration)

Christian Dietz, Martin Horn (ops)

Albert Cardona (scripting framework, algorithms)

Jean-Yves Tinevez (algorithms)

Grant Harris (display)

Larry Lindsey (histograms)

Hackathons:

MPI-CBG, Dresden (Pavel Tomančák), EMBL, Heidelberg (Francesca Peri, Kota Miura), and LOCI, Madison (Kevin Eliceiri)

`http://imglib2.net/`

Pietzsch, T., Preibisch, S., Tomančák, P., and Saalfeld, S.
“ImgLib2 – generic image processing in Java,”
Bioinformatics (September 2012).