SNAP GeoCoding

New architecture overview

Tom Block

Brockmann Consult GmbH

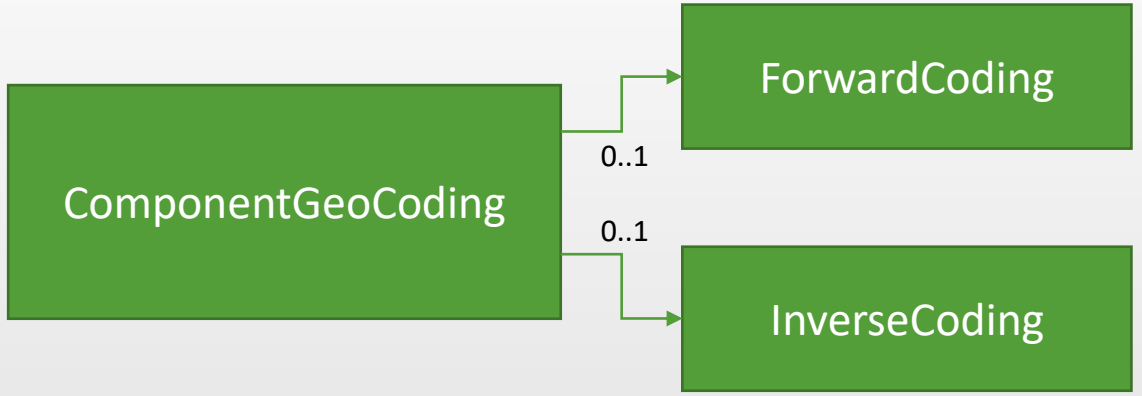
esa

Motivation

- Performance issues
- Lack of precision
- Artefacts
- Code structure

Code Structure

- Separate forward and inverse transformation
- Keep GeoCoding interface
- Separate create and initialize
- Re-use common operations
- Encapsulate mathematics
- Allow reader to assemble geocoding most appropriate for data
- Library of building blocks
- Improved test-coverage



Forward: pixel (x/y) -> geo (lon/lat)

Inverse: geo (lon/lat) -> pixel (x/y)

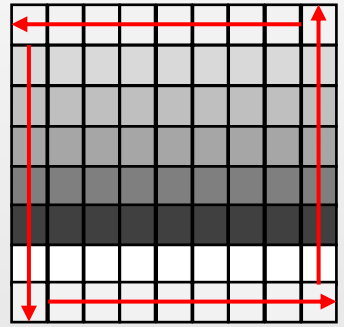
Forward/Inverse

- ForwardCoding
  - Transform (x/y) to (lon/lat)
  - Either interpolation (tie-points) or direkt lookup (pixel)
  - Implementations:
    - TiePointBilinearForward, **TiePointSplineForward**, PixelForward, **PixelInterpolatingForward**
- InverseCoding
  - Transform (lon/lat) to (x/y)
  - Search closest pixel in a geodetic sense, evt. interpolate (tie-points)
  - Implementations:
    - TiePointInverse, **PixelGeoIndexInverse**, PixelQuadTreeInverse

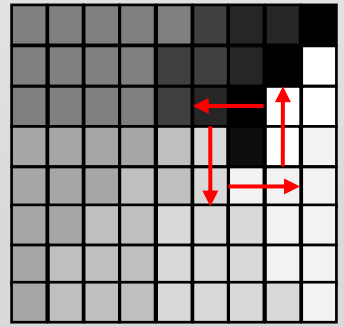
Common Operations

- ComponentGeoCoding implements two common tasks
  - Detection of anti-meridian
  - Detection of poles
- Result of checks used during initialisation phase
- Checks can be skipped if not useful

1. Check longitude difference on borders of raster
2. If Δ(lon) > 180 deg -> anti meridian



1. Find pixel with abs(lat) > threshold
2. Check longitude difference on 3x3 neighbourhood
3. If **once** Δ(lon) > 180 deg -> pole

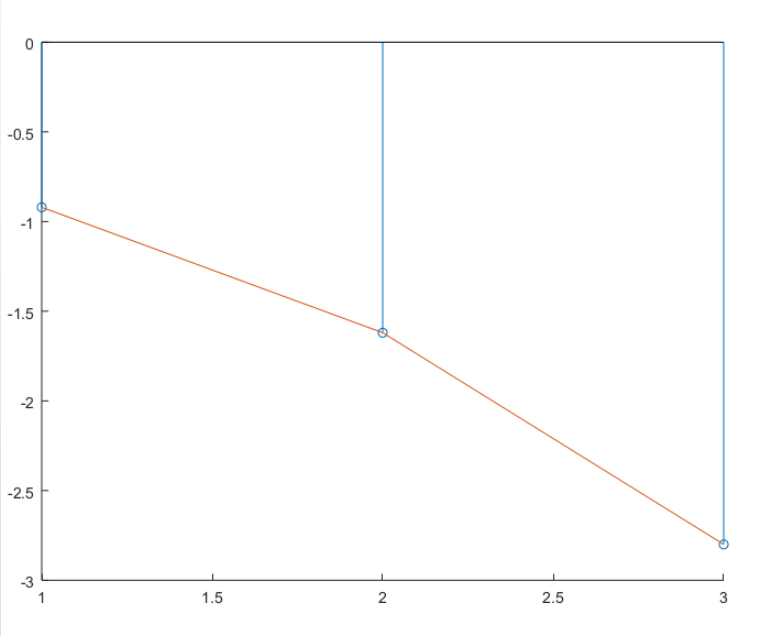


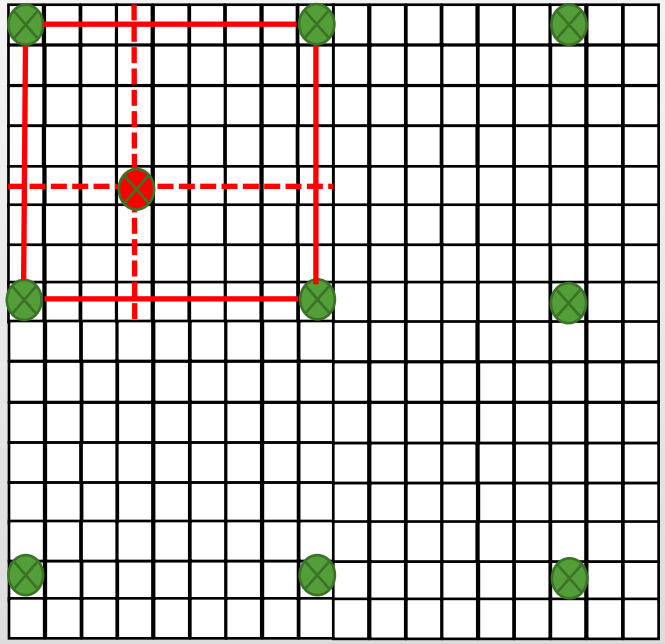
ForwardCoding implementations

- TiePointBilinearForward
- TiePointSplineForward (new)
- PixelForward
- PixelForwardInterpolating (new)

TiePointBilinearForward

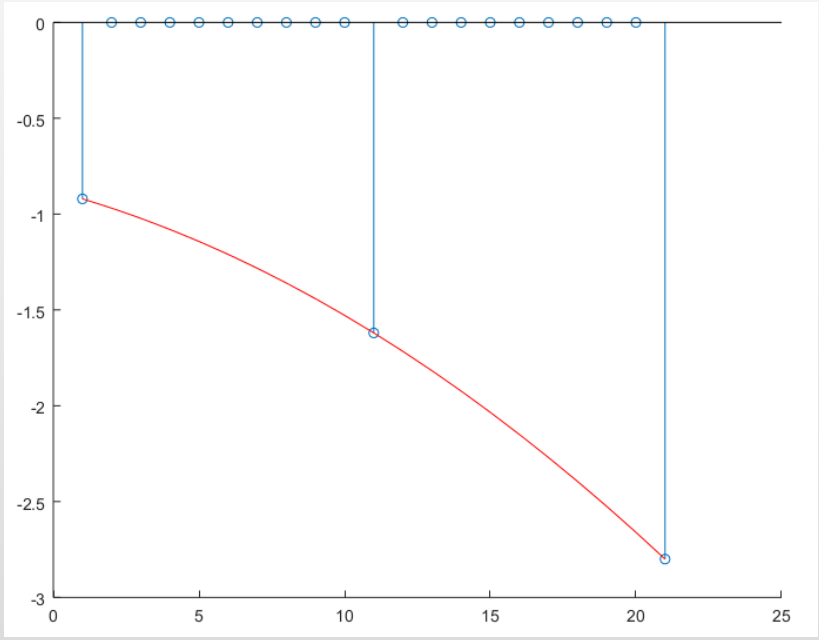
- ForwardCoding for tie-point based geolocations
- Uses bilinear interpolation

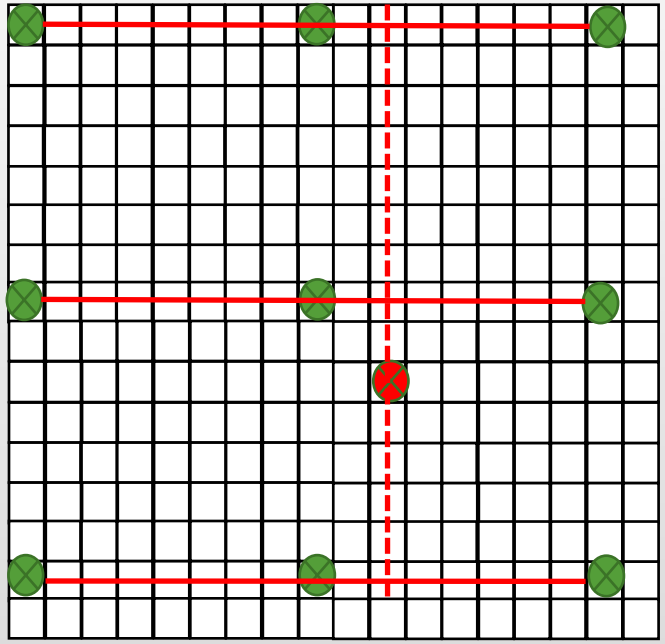




TiePointSplineForward

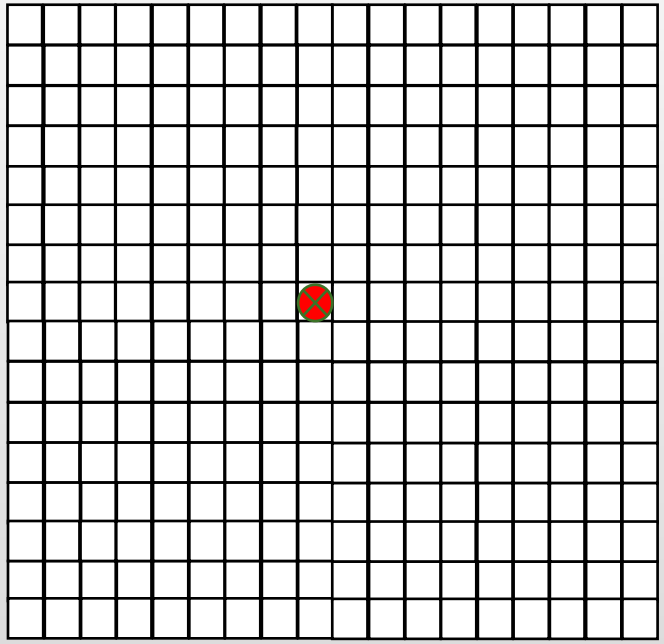
- ForwardCoding for tie-point based geolocations
- Used 2D spline interpolation





PixelForward/PixelForwardInterpolating

- ForwardCoding for pixel based geolocations
- Simply read (lon/lat) at (x/y)
- Interpolating mode uses bilinear interpolation between the four neighboring pixels



InverseCoding implementations

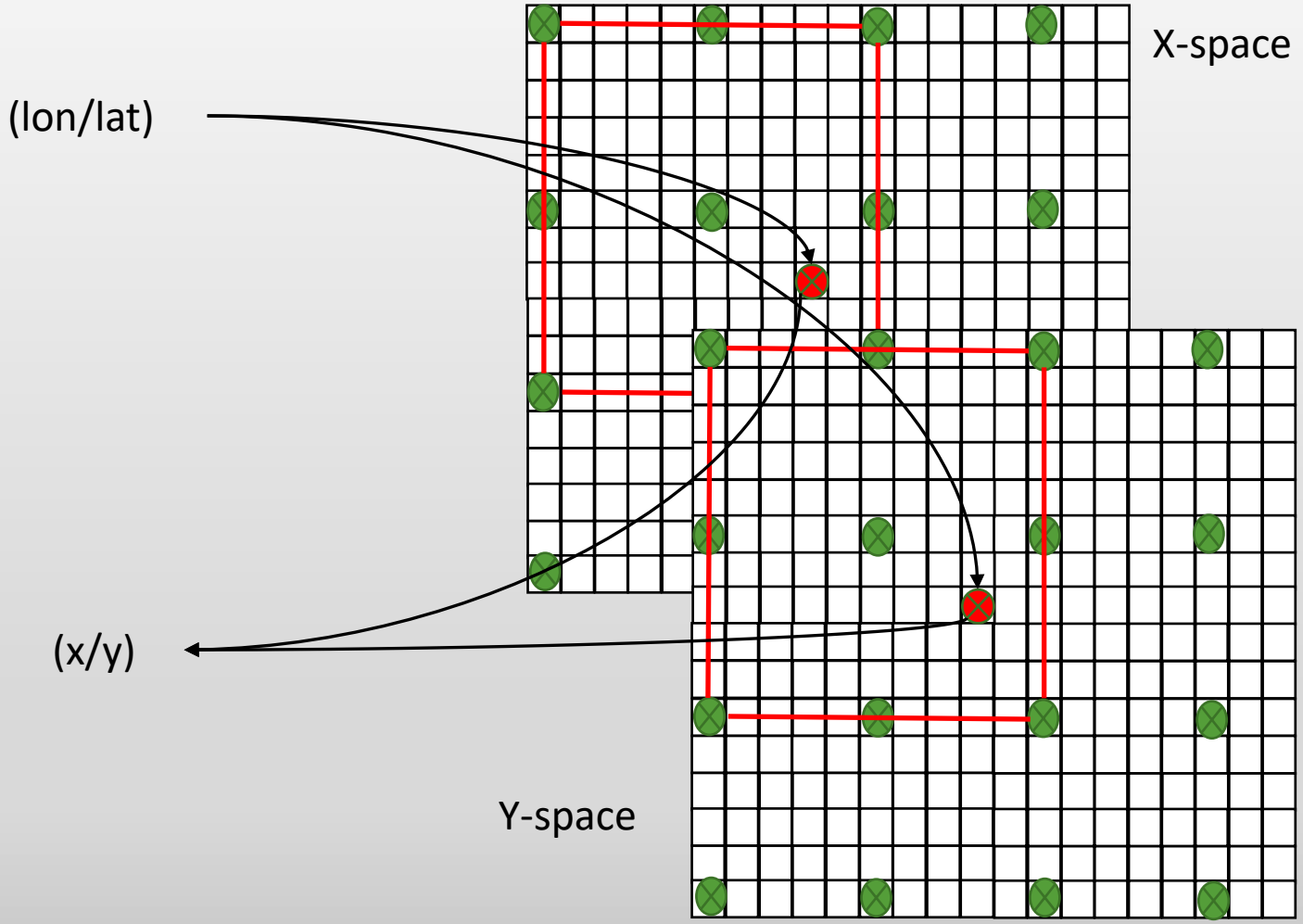
- TiePointInverse
- PixelQuadTreeInverse
- PixelGeoIndexInverse (new)

TiePointInverse

- Interpolation in lon/lat space
- Separate interpolation for x and y
- Use tiled approach (100-300 tiles per product)
- For each tile find optimal approximation by minimising RMS error
- Approximations
  - linear/bilinear
  - Quadratic/biquadratic
  - Cubic/bicubic
  - 4th order/bi 4th order

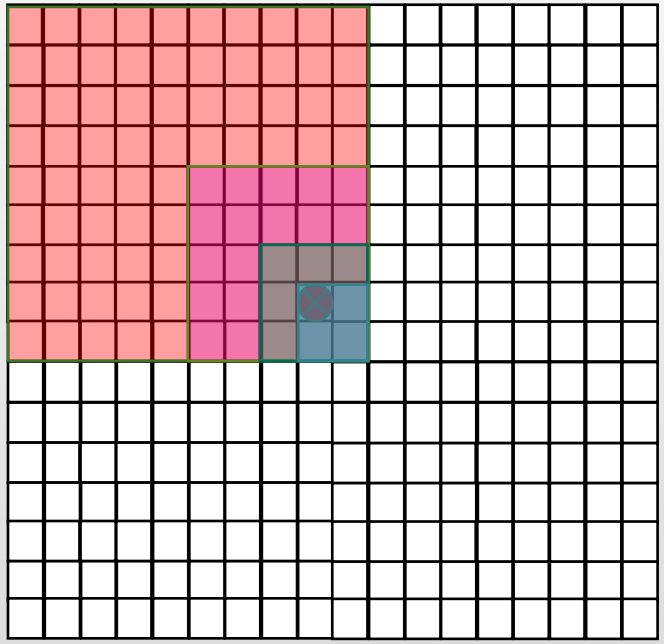
TiePointInverse (II)

- getPixelPos()
  - Find approximation with minimal squared distance (lon/lat)
  - Calculate approximation for x
  - Calculate approximation for y



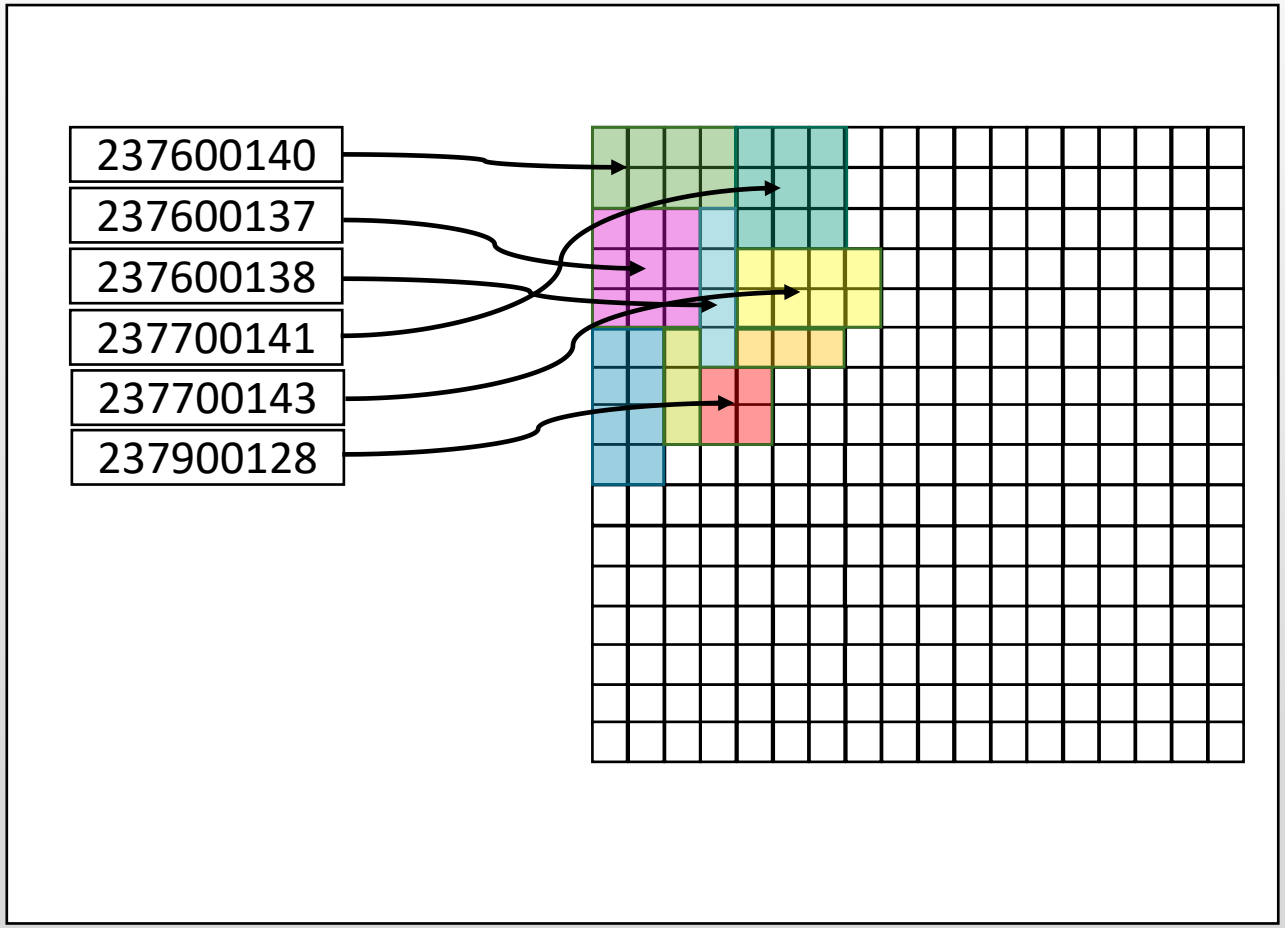
PixelQuadTreeInverse

- Search by quad-tree recursion
- Inside/outside detection using lon/lat bounding box for each quad-tree section
- Recursion stops at 2 x 2 region
- Return pixel with smallest squared distance to (lon/lat)



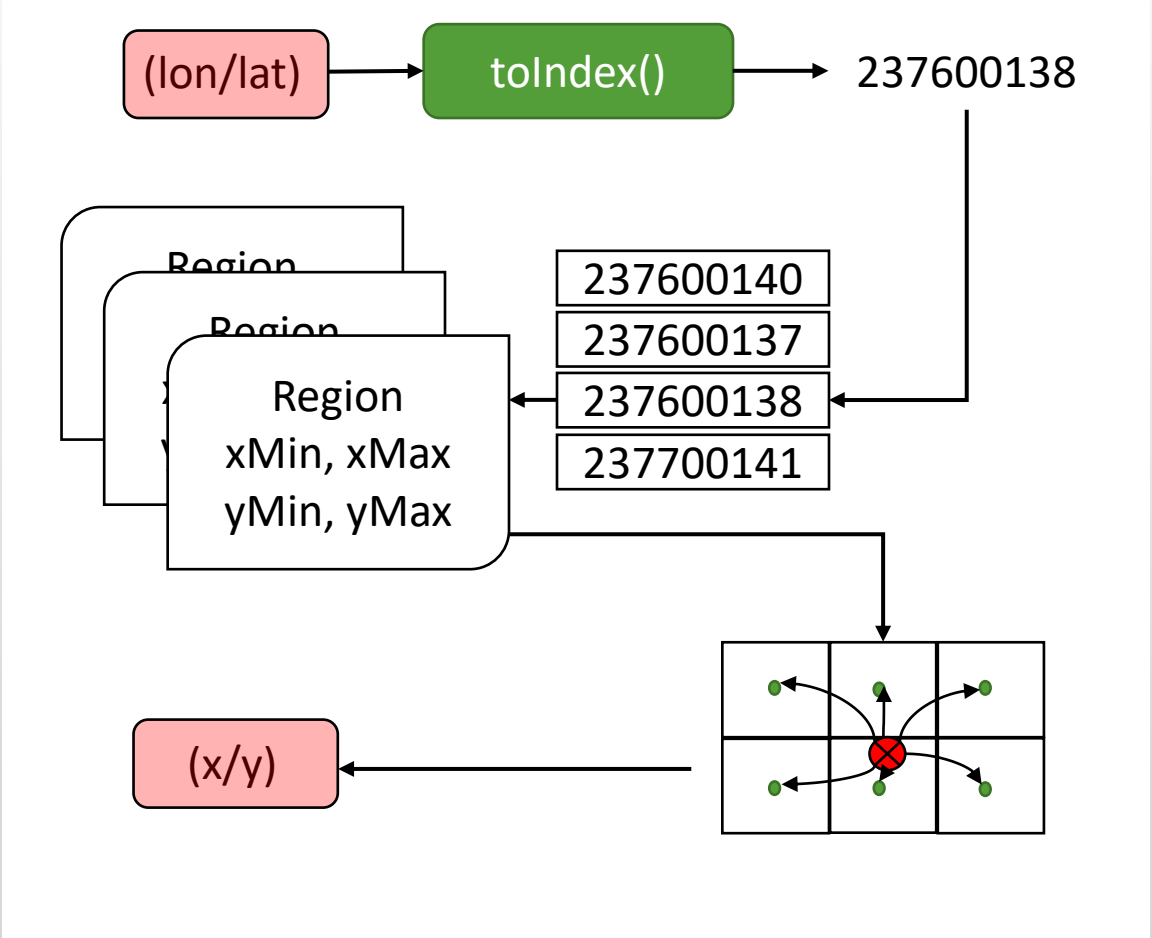
PixelGeoIndexInverse

- Build index of small regions
- Index expression combines (lon/lat) into one index value
- Index expression depends on sensor resolution
- Index storage as B-Tree, search complexity *O*(ln(N))



PixelGeoIndexInverse (II)

- getPixelPos()
  1. (lon/lat) -> index
  2. Search region for index
  3. Find pixel position with minimal geodetic distance to (lon/lat)
  4. Add x/y – offsets
- Complexity
  - ln(N) for search, i.e. 14 long compares for an index of 1e6.
  - 4-12 geodetic distance calculations

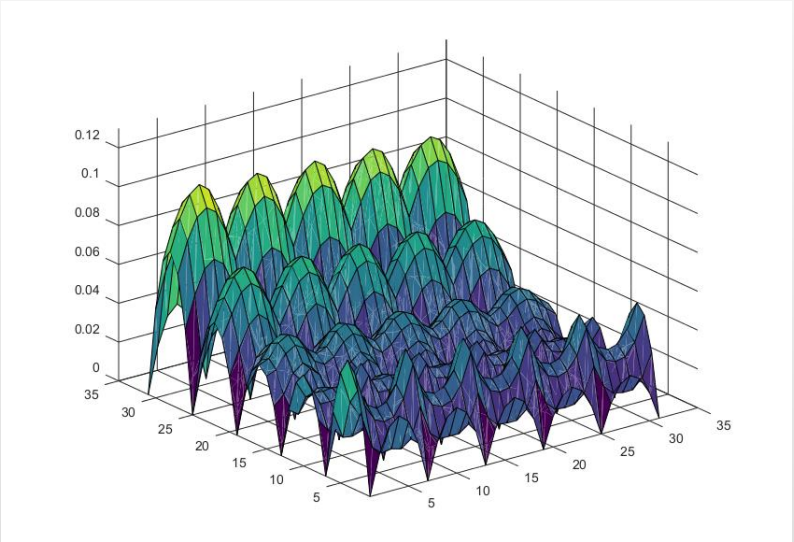


Performance

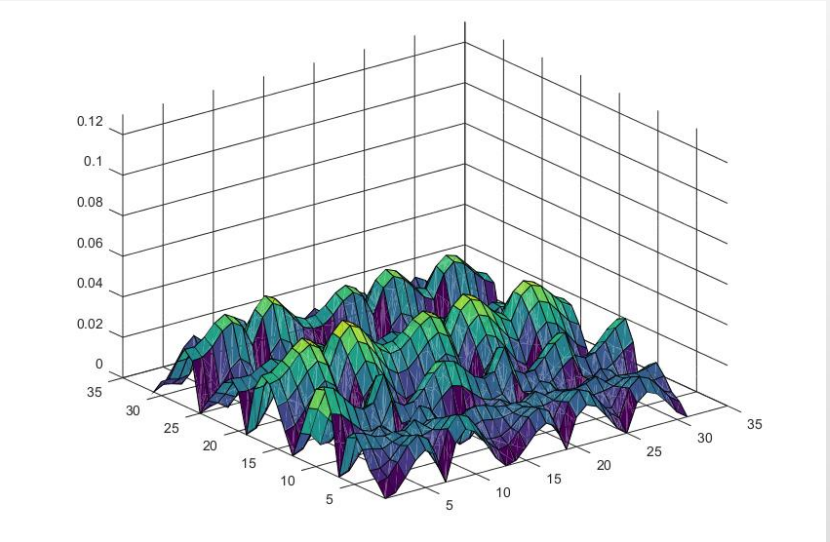
- Correctness of transformation
  - Absolute
  - Relative/instrinsic
- Runtime performance
  - Set-up time
  - Conversion time

Performance (II)

- Compare TiePointForward
- Bilinear vs Spline
- Testdata:
  - AMSU-B subset
  - WGS-72 (i.e. on ellipsoid)
  - Subsampled
  - Re-interpolated
- Absolute interpolation errors can be calculated



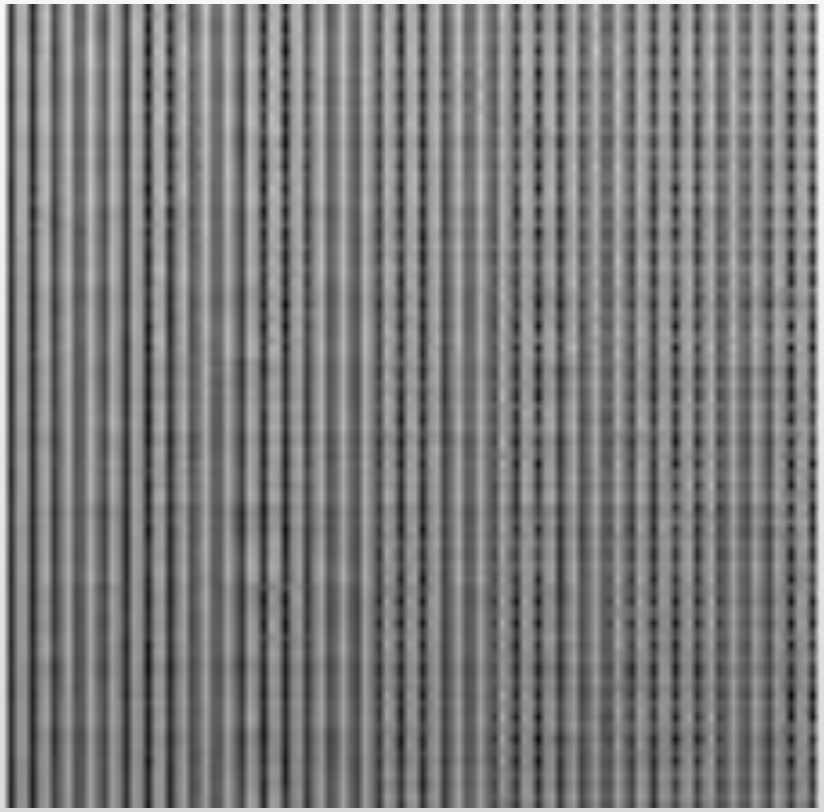
bilinear



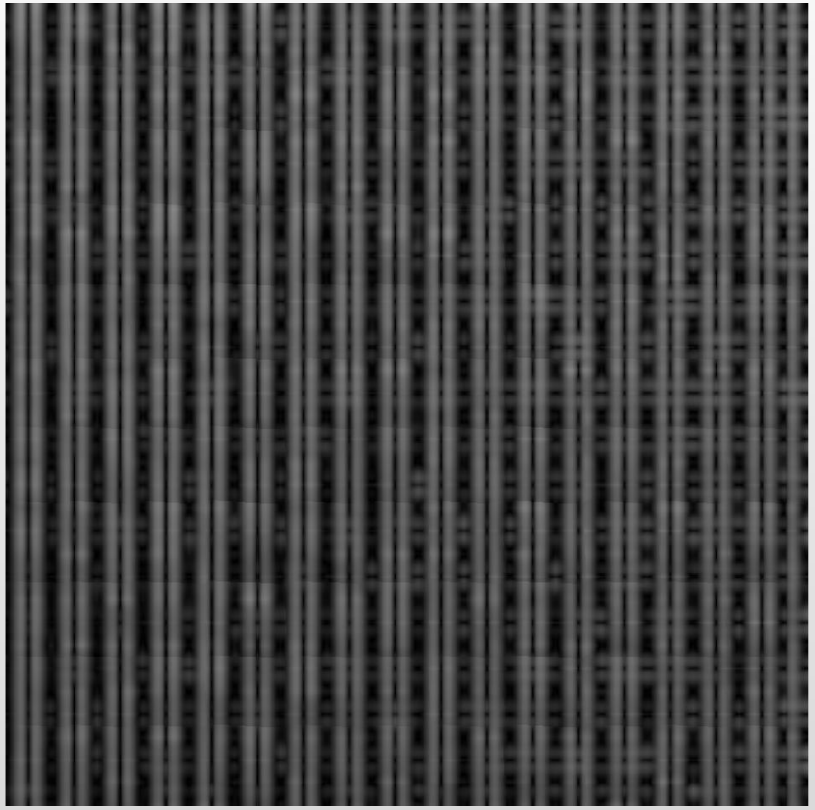
spline

Performance (III)

- Total error analysis
- Example: MERIS FR
- Displays distance error for combined forward and inverse transformation
- (x/y) -> (lon/lat) -> (x/y)
- Values are $\sqrt{\Delta x^2 + \Delta y^2}$ in fractional pixel sizes



bilinear
min: 2.38e-6, max: 0.0125,
mean: 0.0073



spline
min: 2.05e-6, max: 0.0084,
mean: 0.0038

Performance (IV)

Product	Init Bilinear	Conv Bilinear	Init Spline	Conv Spline
MERIS FR (2241 x 2241)	0.108	0.353	0.109	0.383
MERIS FRS (4481 x 4161)	0.300	1.169	0.278	1.284
AATSR (512 x 44032)	1.044	1.407	1.050	1.643
ASAR WSM (5819 x 39429)	0.061	15.417	0.054	15.254

- Compare TiePointForward Bilinear vs Spline
- Testdata:
  - ENVISAT data
- Runtime performance (average of three tests)

Performance (V)

Product	Init QuadTree	Conv QuadTree	Init GeoIndex	Conv GeoIndex
OLCI (2728 x 4865)	0.001	23.155	3.789	5.855
SYN (4091 x 4865)	0.001	23.463	3.668	5.598

- Compare PixelInverse QuadTree vs GeoIndex
- Testdata:
  - SENTINEL3 DEM corrected data
- Runtime performance (average of three tests)

Open issues

- Special treatment for extreme sizes e.g. 2 x 4200 px
- Implement handling of self-overlapping orbits for inverse transformations
- Implement handling of poles
- Define and implement framework to parametrize product reader

Ideas

- Allow decorator-pattern
- Possible (generic) decorators
  - Fill value handling
  - Flag masks
  - Geometric constraints

The end!

Thank you!