Holo3 Polymap Surface reconstruction

Pierre BERNARD
supervised by Christophe PRUD'HOMME and Vincent CHALVIDAN

August 2020

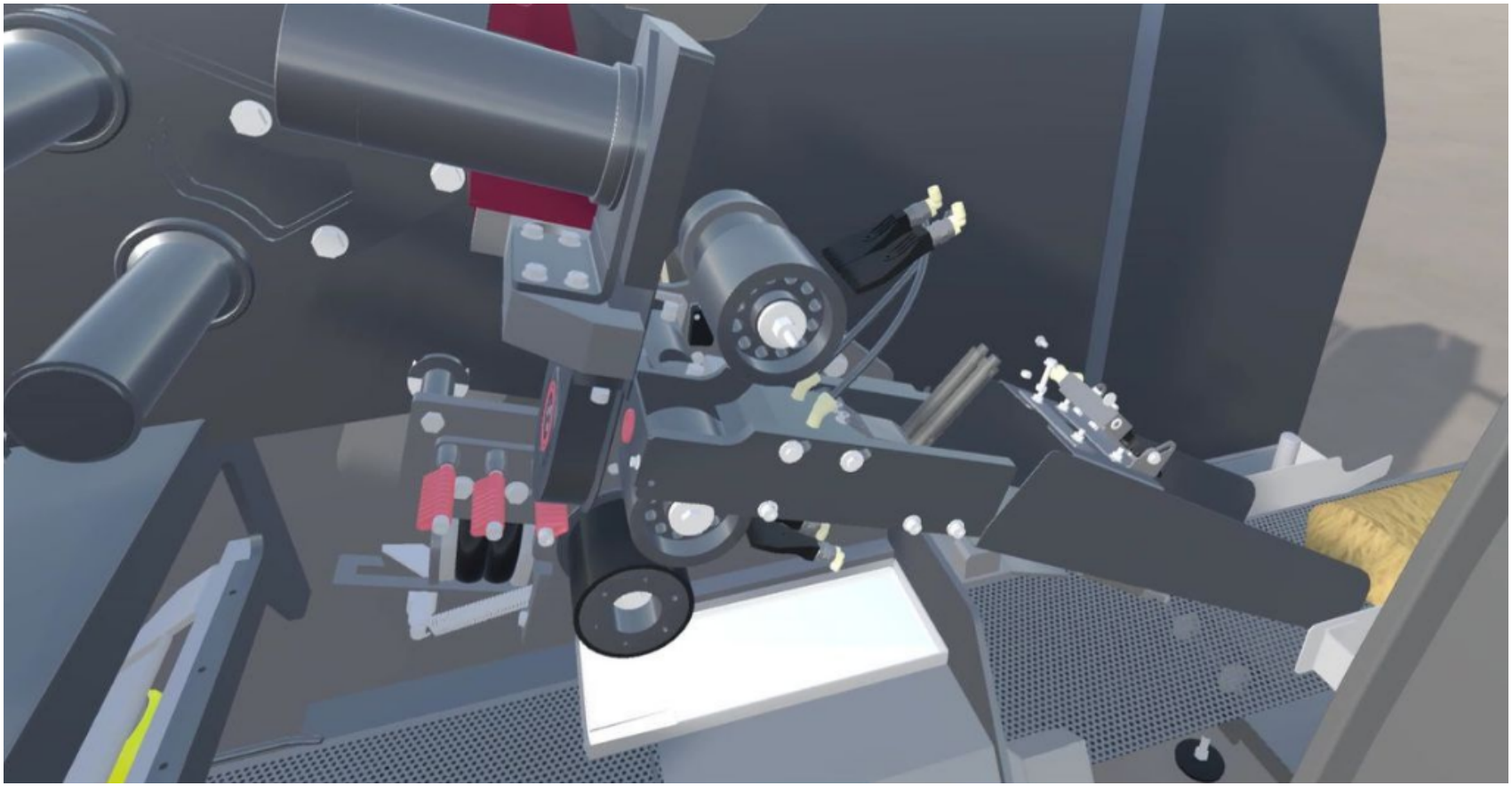


Figure: Immersive Presentation

Polymap

Precision and Speed

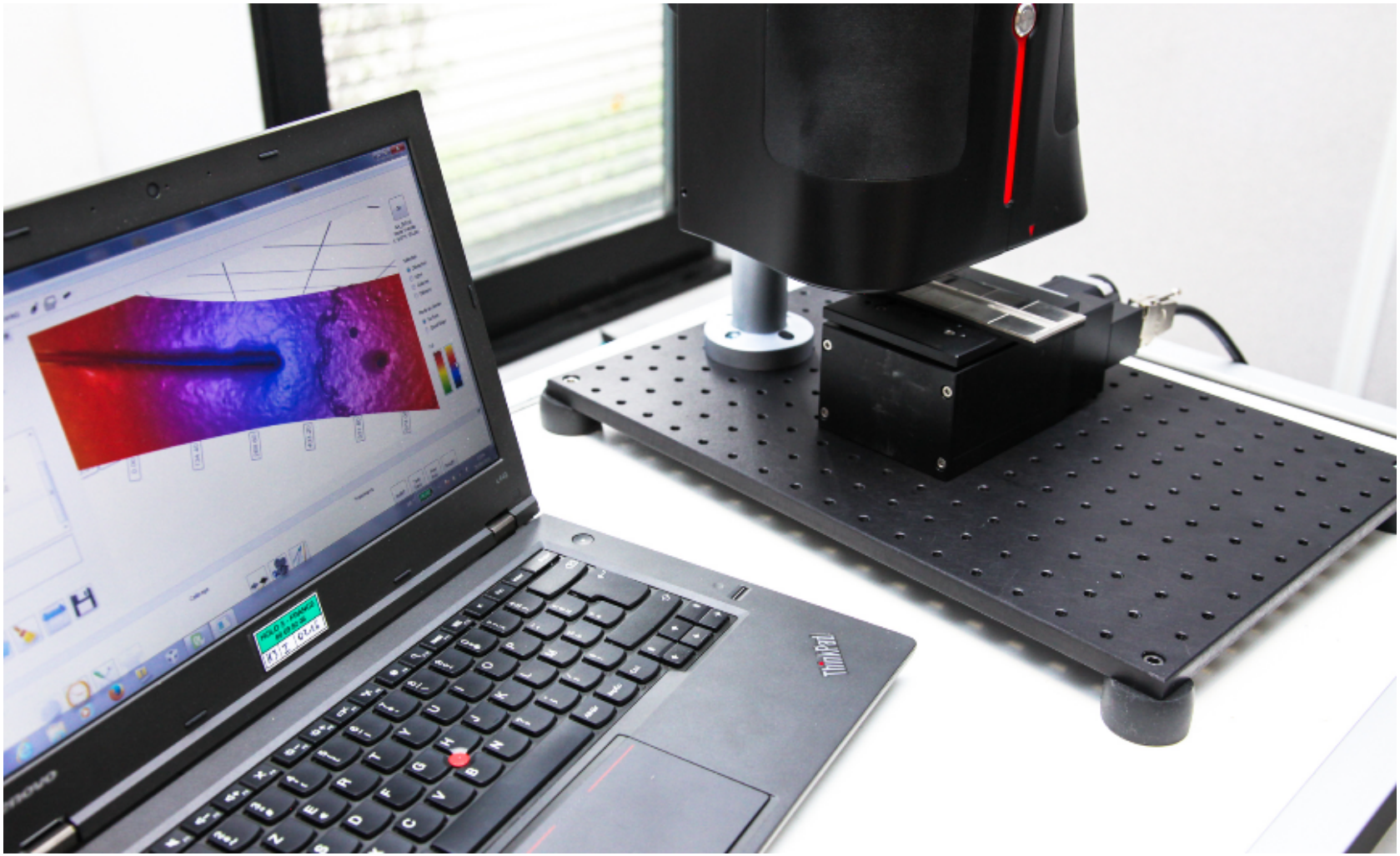
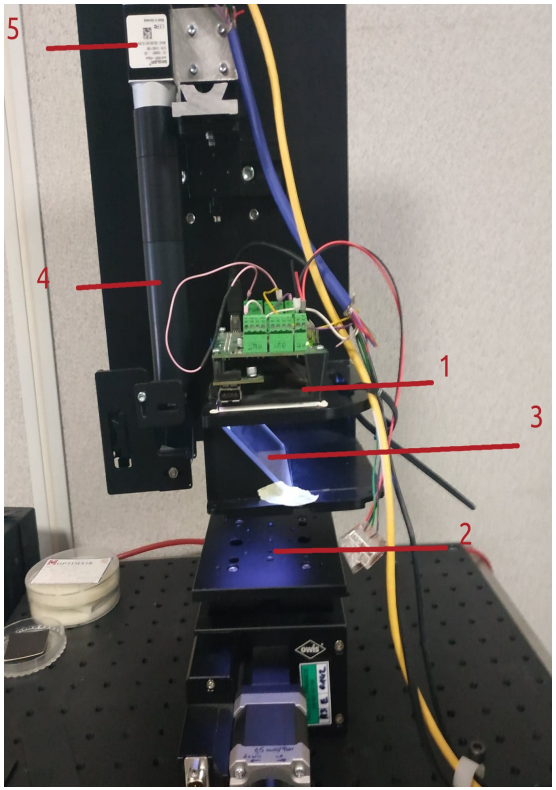


Figure: Polymap

Polymap

How it works



1. LCD screen
2. Emplacement of the studied object
3. Beam Splitter splitting incoming incident light by transmitting half of the light and reflecting half of the light
4. Lense
5. Camera

Polymap

How it works

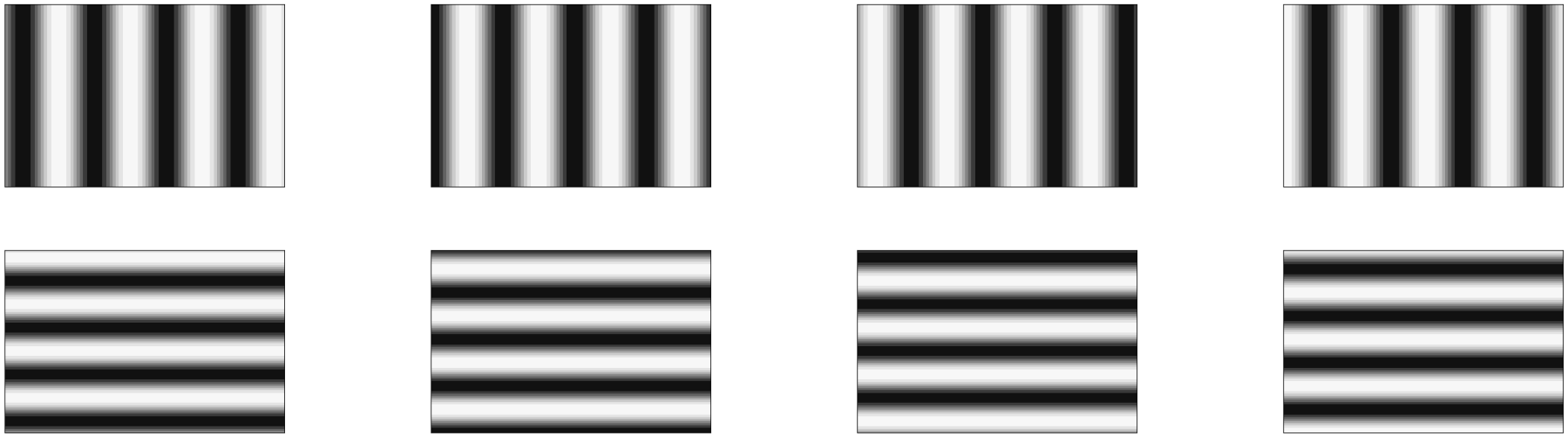


Figure: Fringes displayed by LCD screen

We **know** the intensity $I_i(x,y) \forall i \in 0,1,2,3$ for each phase ϕ_i used above. From this information, we can compute :

- $I_m(x,y) = \frac{I_0+I_1+I_2+I_3}{4}$ the map of **mean intensity**

and we have the **relationship** $I_i = I_m(1+\gamma.cos(\phi+\frac{i\pi}{2})) \forall i \in 0,1,2,3$:

$$\begin{cases} I_0 &= I_m[1+\gamma cos(\phi)] \\ I_1 &= I_m[1+\gamma cos(\phi+\frac{\pi}{2})] = I_m[1-\gamma sin(\phi)] \\ I_2 &= I_m[1+\gamma cos(\phi+\pi)] = I_m[1-\gamma cos(\phi)] \\ I_3 &= I_m[1+\gamma cos(\phi+\frac{3\pi}{2})] = I_m[1+\gamma sin(\phi)] \end{cases}$$

$$\begin{cases} I_0 - I_2 &= 2I_m\gamma cos(\phi) \\ I_3 - I_1 &= 2I_m\gamma sin(\phi) \end{cases}$$

Data Computation

from the previous relationship,

$$\begin{cases} I_0 - I_2 &= 2I_m\gamma cos(\phi) \\ I_3 - I_1 &= 2I_m\gamma sin(\phi) \end{cases}$$

we can compute :

- The map of **relative contrast** $\gamma = \frac{\sqrt{(I_0-I_2)^2+(I_3-I_1)^2}}{2I_m}$
- $\phi_w = arctan2(I_3 - I_1, I_0 - I_2)$ the wrapped Phase map
- And from all of the phase maps we can construct the **unwrapped phase** map ϕ using bigger size pattern to ensure continuity

From phase data, HOLO3 can then compute **local slopes** along vertical and horizontal axis.

Surface reconstruction

Variational formulation

Let $P = (p, q)$ where p and q are the slopes according respectively to the x and the y axis, we get the natural EDP

$$\nabla z = P \text{ on } \Omega$$

We multiply this problem by a test funcion $V : \Omega \to \mathbb{R}^2$ that can be seen as the gradient of a function test $v : \Omega \to \mathbb{R}$, integrate it and we get

$$\int_\Omega \nabla z \nabla v = \int_\Omega P \nabla v$$

and by using the Green Formula with the natural boundary condition $\frac{\partial z}{\partial n} = (p, q) \cdot n$ on $\partial\Omega$ we get a Poisson problem :

$$\int_\Omega \Delta z v = \int_\Omega \nabla \cdot P v$$

Surface Reconstruction

Additional condition to get unicity

The previous problem has a solution up to a constant, to ensure that the problem is well posed we select the zero-mean solution, i.e

$$\begin{cases} \Delta z & = \nabla P \text{ on } \Omega \\ \int_\Omega z & = 0 \end{cases}$$

Surface Reconstruction

Testsuite

- To study the convergence of the reconstruction, we can build manufactured surfaces.

$$z(x, y) = \sin(x) + 0.1y + 0.2\cos(5x)\sin(y)$$

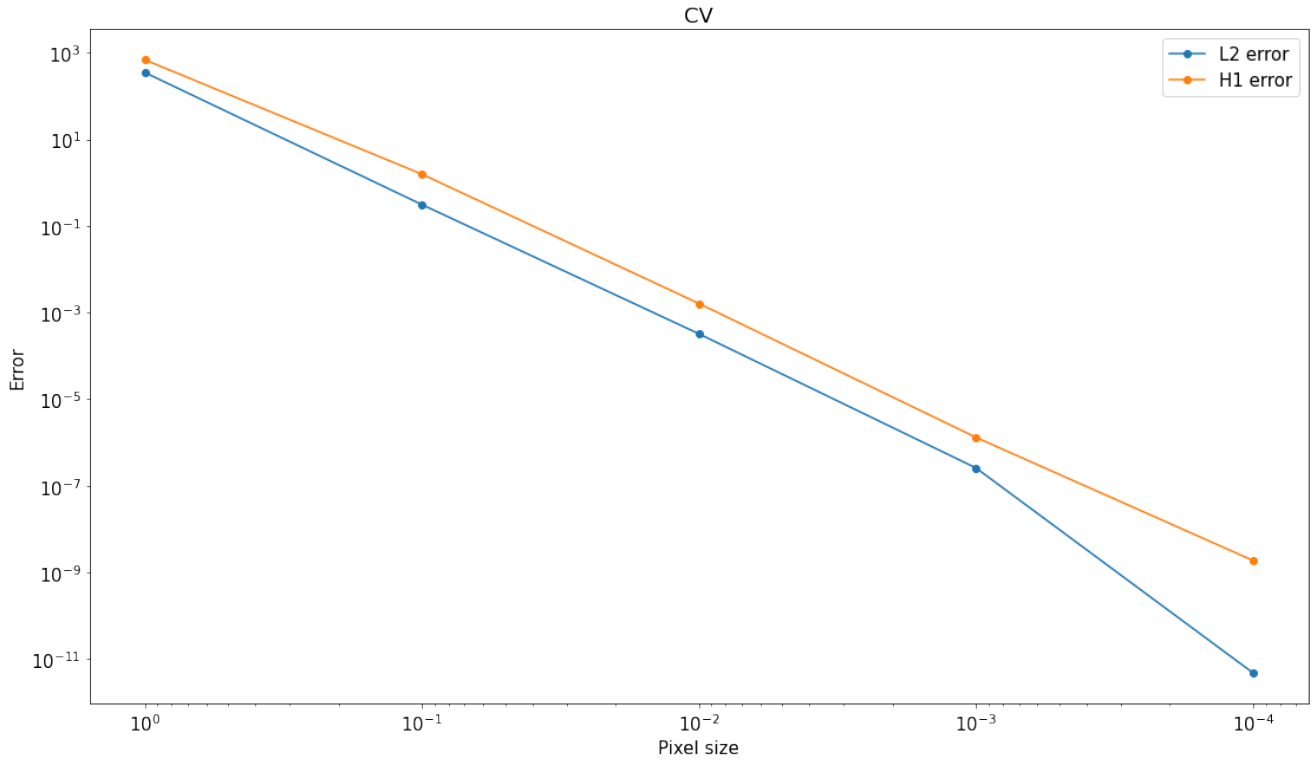
- we just need to compute slopes by hand and pass it to the application

```
1 if( !soption( "polymap.p" ).empty() && !soption( "polymap.q"
      ).empty() )
2 {
3     px.on( _range = elements( support( Vh ) ), _expr = expr(
      soption( "polymap.p" ) ) );
4     py.on( _range = elements( support( Vh ) ), _expr = expr(
      soption( "polymap.q" ) ) );
5 }
```

Surface Reconstruction

Testsuite

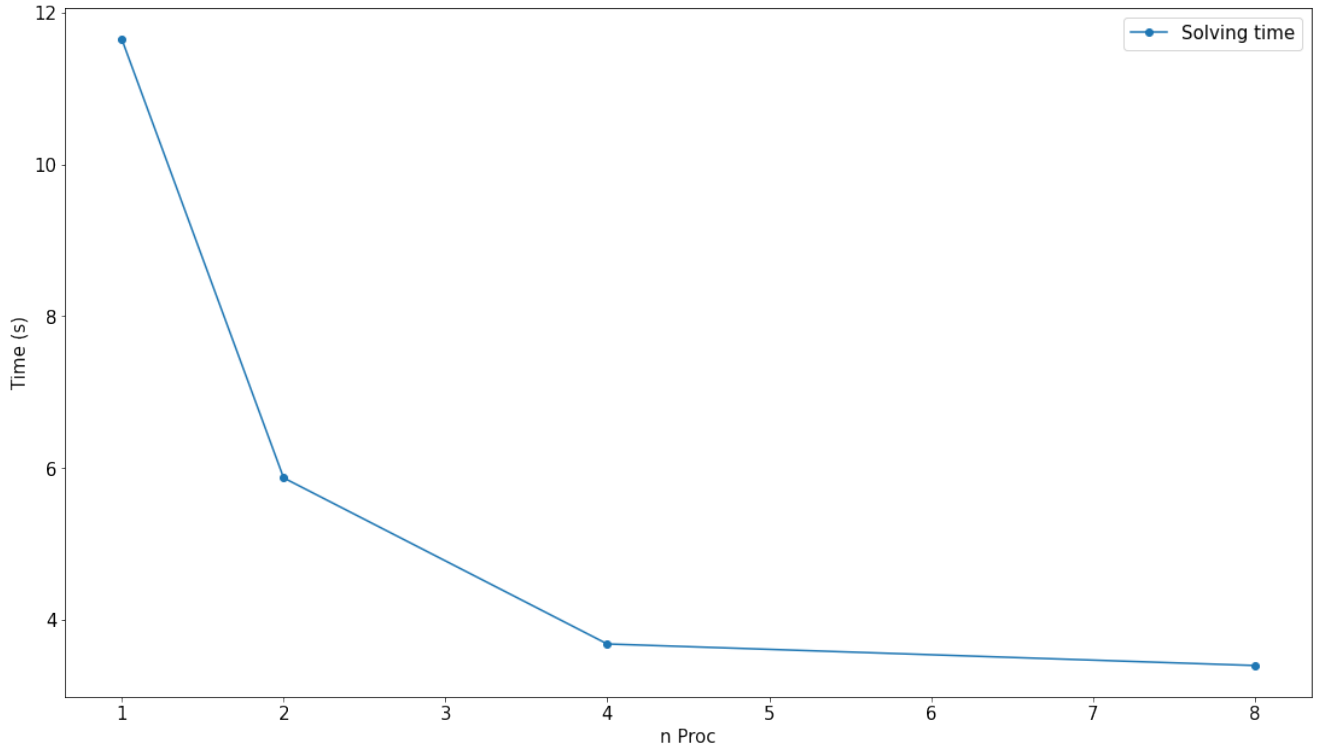
We are working on pixels so the natural choice is to work on Q1 Lagrange elements.



- Error L2 converging in h^2
- Error H1 converging in h

Surface Reconstruction

Benchmarking



Inclined surfaces

Motivation

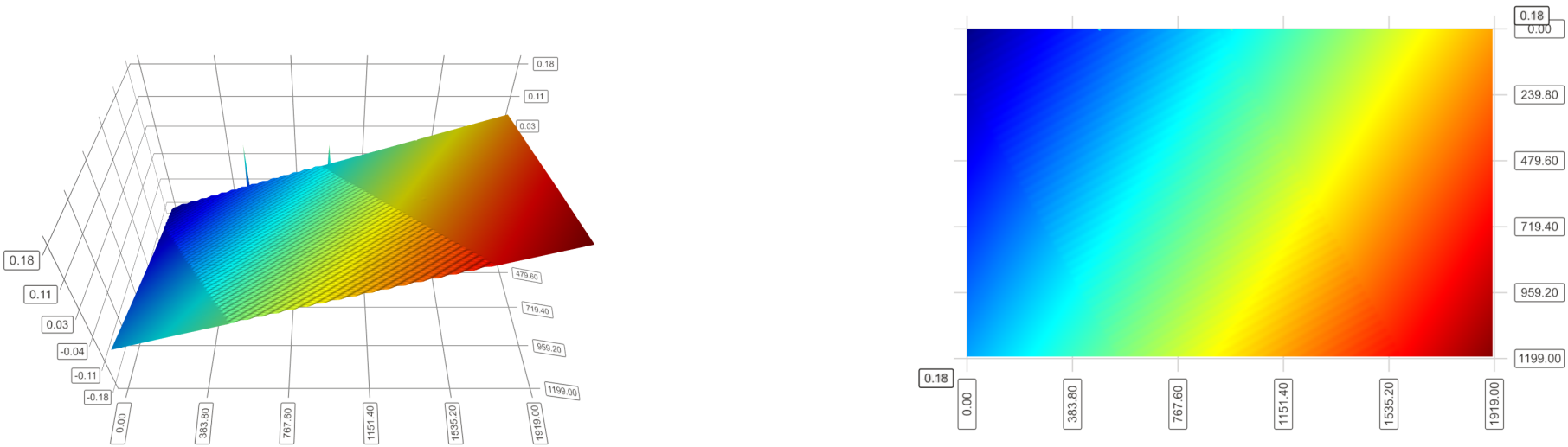


Figure: Inclined surface make the reconstruction harder to read

Inclined surfaces

Strategy

Let suppose that the surface is **globally flat** i.e. $p(x, y) = \alpha + g(x, y)$ and $q(x, y) = \beta + h(x, y)$ with g and h zero mean functions.
Then we just need to study $g(x, y)$ and $h(x, y)$ instead of $p(x, y)$ and $q(x, y)$

Warning

If the surface isn't globally flat, doing so will cause a loss of information and will lead to the creation of artefacts

Inclined surfaces

Implementation

```
auto mpx = mean( _range = elements(support( Vh )),
                 _expr = idv( px ) )( 0, 0 );
auto mpy = mean( _range = elements(support( Vh )),
                 _expr = idv( py ) )( 0, 0 );

px.add( -mpx );
py.add( -mpy );

mpx = mean( _range = elements( support( Vh ) ),
            _expr = idv( px ) )( 0, 0 );
mpy = mean( _range = elements( support( Vh ) ),
            _expr = idv( py ) )( 0, 0 );
```

Inclined surfaces

Results

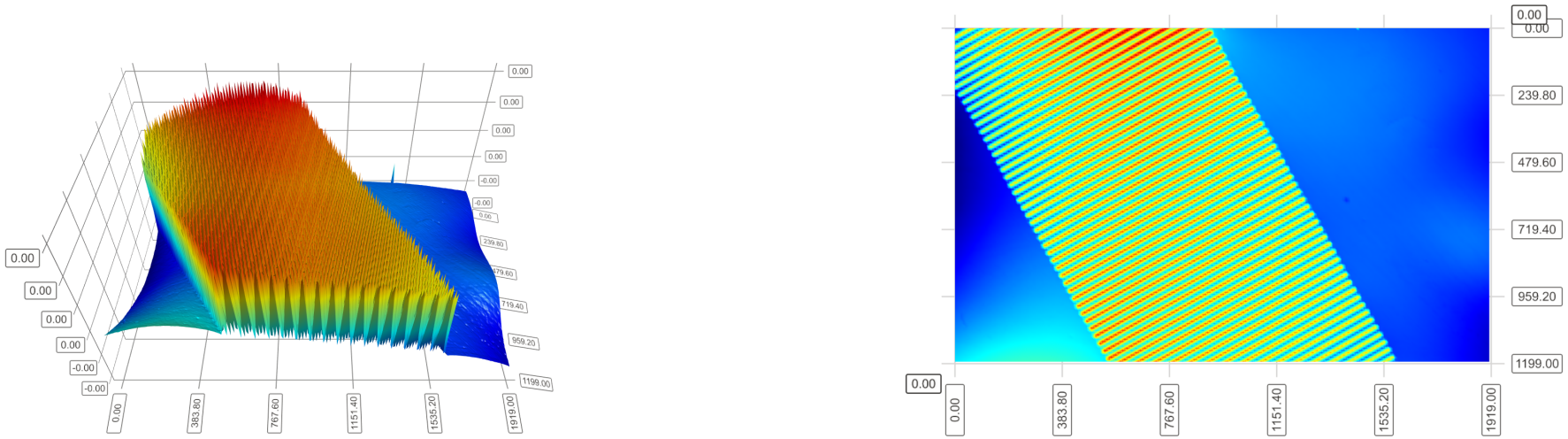
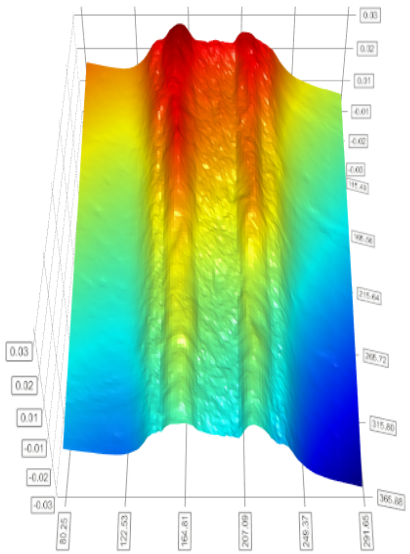


Figure: The local slopes are far more readable

Corrupted Area



Causes :

- Roughness of the surface (multi reflection)
- Strong Variation of slopes of the surfaces
- Real slopes with a value that exceeds the limit of measurement
- Light pollution from the environment
- Camera

The solution : Indicators

We know that the corrupted areas has some characteristics :

- Over/under exposure induces a **low contrast**
- **Phase spikes** when appears when unwrapping phases
- **Phase out of possible range** appears at the border of the surface when it is so curvy that reflection goes outside the camera's screen

Indicators

Contrast threshold

Let I_i being the intensity of the ray reflected i times and R the reflectivity of the studied surface.

$$I_n = I_{n-1} \times R = ... = I_0 R^n$$

We can estimate $\overline{R_0}$ as the mean of the contrast map and since $0 < \overline{R_0} < 1$, we can build a threshold

$$R' = \frac{\overline{R_0}^2 + \overline{R_0}}{2}$$

To improve our threshold, we can use this first one to remove corrupted data from the estimation of R and compute a second one that will not be corrupted.

$$R'' = \frac{\overline{R_1}^2 + \overline{R_1}}{2} \text{ where } \overline{R_1} = mean(C \setminus \{c_{ij} | c_{ij} < R'\})$$

Indicators

Implementation

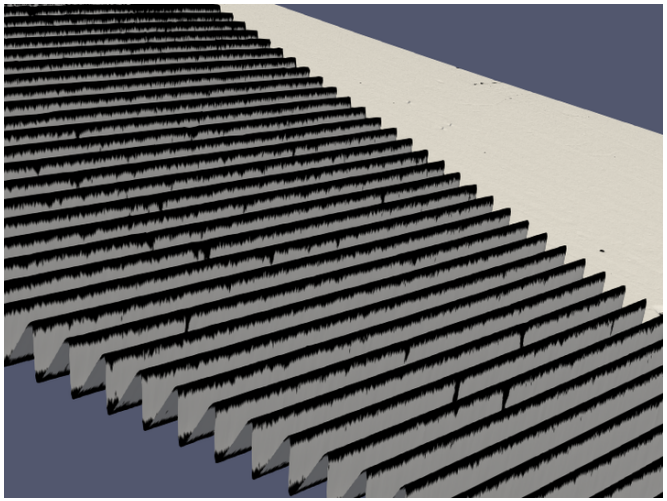
As we are using Eigen library to be as fast as possible, we want to avoid loops on each pixels. For these reason, to compute the second threshold we need to replace each corrupted contrast by the median of each contrast using only eigen operations.

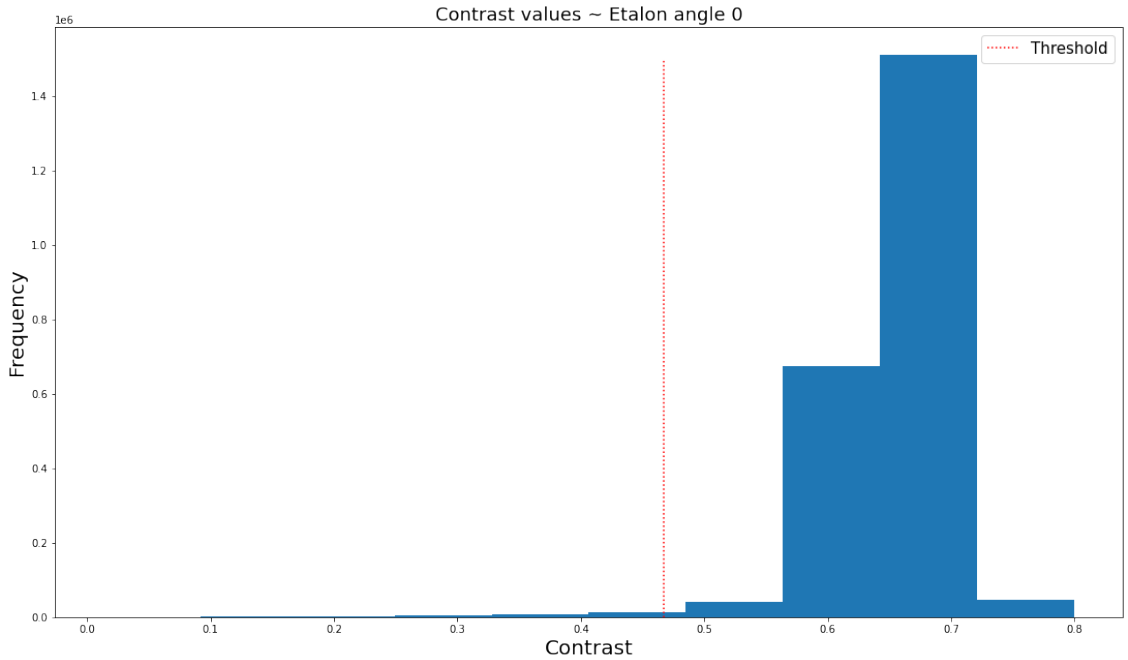
```
1 auto cxy_row { cxy.reshaped() };
2 std::sort( cxy_row.begin(), cxy_row.end() );
3
4 auto med = cxy_row.size() % 2 == 0 ?
5     cxy_row.segment( (cxy_row.size()-2)/2, 2 ).
    mean() :
6     cxy_row( cxy_row.size()/2 );
```

We can then use our threshold to select corrupted data

```
image_t greaterThanR = (cxy.array() > R).select(
    cxy, matrix_1.array()*med);

r = greaterThanR.mean();
R = ( r*r + r )/2.;

Dcontrast_ = (cxy.array() > R).select(matrix_1,
    matrix_0);
```

Low Contrast detection results





Indicators

Strategy

When studying complex surfaces, to be able to see the whole surface even the darkest areas, we need to increase the luminosity even if it means to **overexpose** some areas.
This choice introduces a noise on phase data. However this noise is known to be $2n\pi$.
Therefore it is easy to detect, we just need to detect when the difference between two neighboring pixels is greater than 2π

We can then use our threshold to select corrupted data

```
1 image_t pxU = image_t::Zero(nrows_, ncols_);
2 pxU.block(1, 0, nrows_-1, ncols_) = px.array().
    block(0, 0, nrows_-1, ncols_);
3 image_t dpxU = image_t::Zero(nrows_, ncols_);
4 dpxU = (px.array()-pxU.array()).abs();
```

Indicators

Implementation

We create 7 other pictures in every possible direction the same way are created the same way and compute the mean of phase variation in every direction.

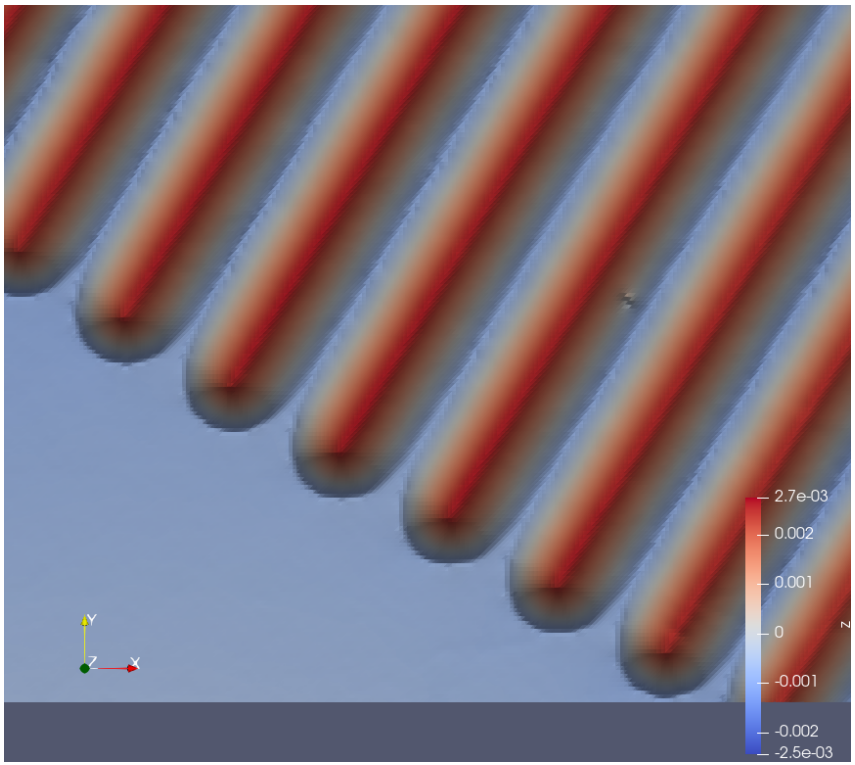
```
image_t dpxMean = (dpxU.array() + dpxR.array() + dpxD.array
    () + dpxL.array() + dpxUR.array() + dpxDR.array() +
    dpxDL.array() + dpxUL.array()) / 8;
```

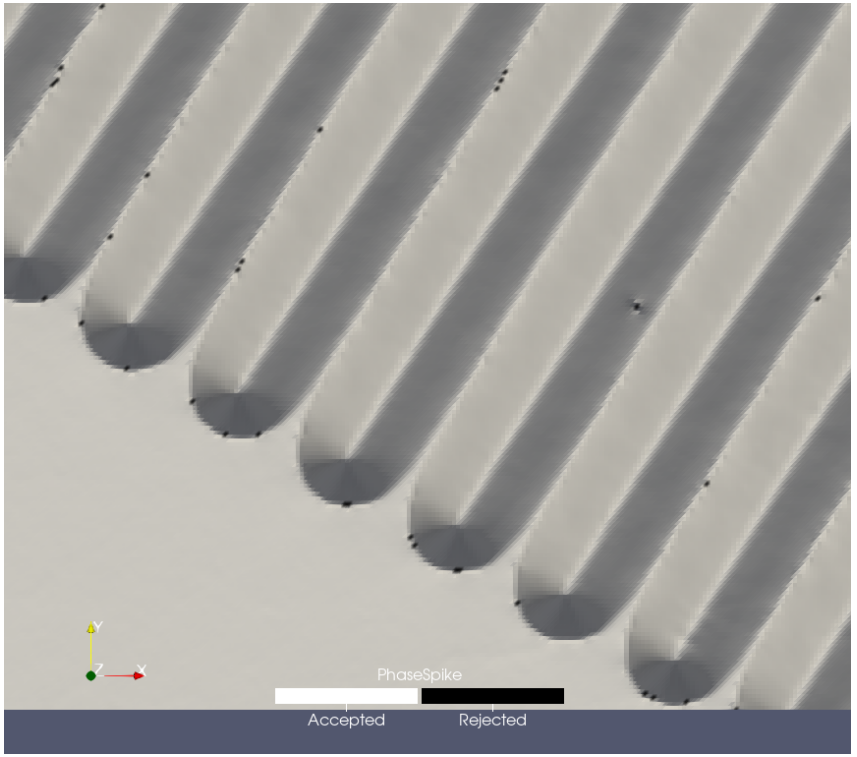
We do the same with the vertical phase map and then select the pixels having either an horizontal or vertical big phase variation.

```
DphaseSpike_ = (dpxMean.array() < M_PI && dpyMean.array() <
    M_PI).select(matrix_1, matrix_0);
```

Indicators

Phase spike results





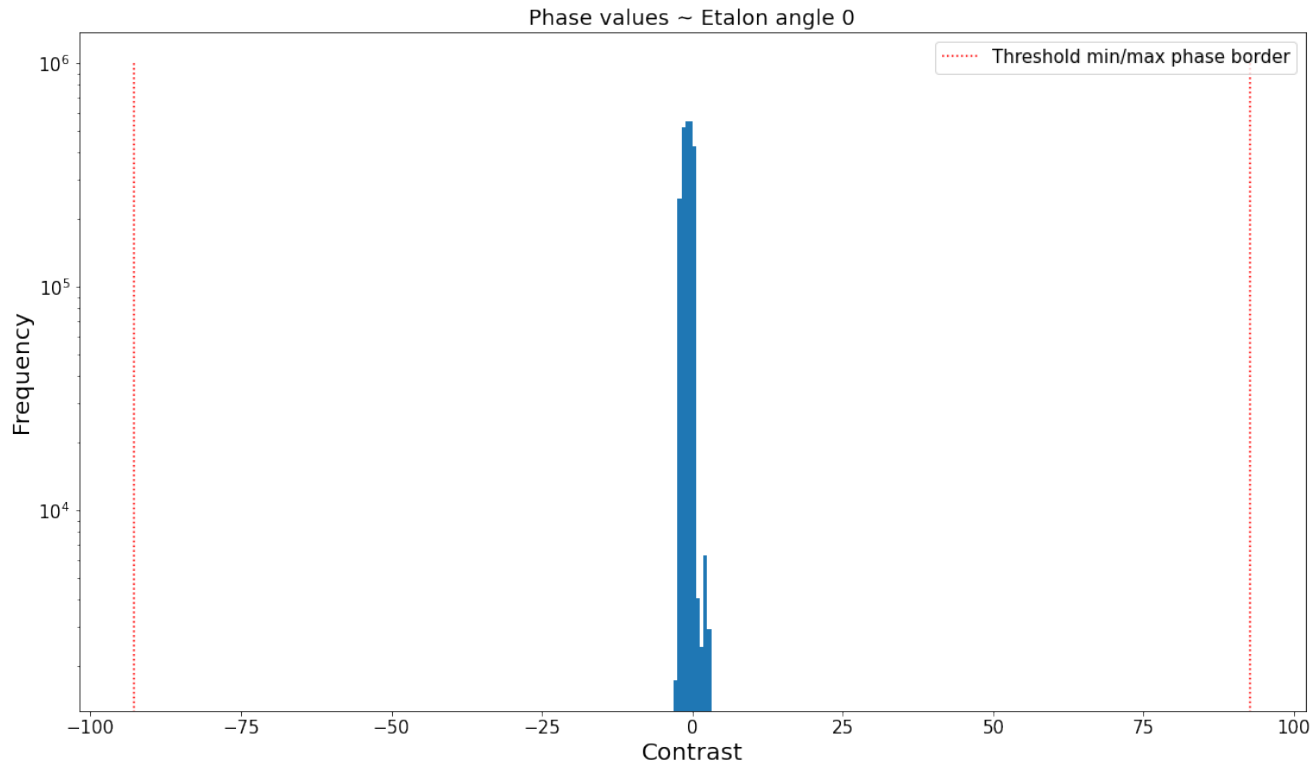
Indicators

Phase out of range Strategy

We know that wrapped phases are $\in [-\pi, \pi]$ so unwrapped phases are $\in [-n_{fringes}\pi, n_{fringes}\pi]$.
So any phase out of this interval is the phase of a corrupted pixel.

Indicators

Phase out of range results



Warning

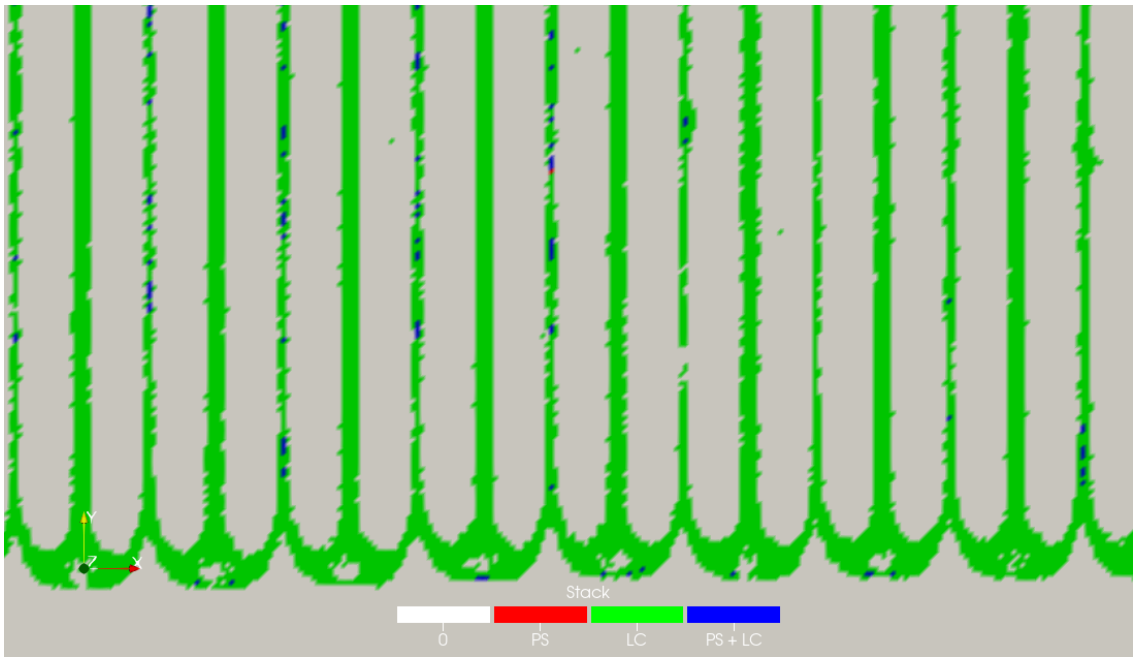
In Polymap, with the current settings, this indicator doesn't detect any pixel at all. However it might be helpful if the focal length or the resolution of the camera changes.

Indicators

Overlapping indicators

One pixel may be detected by multiple indicators. We can stack them by adding them together. We are using powers of 2 to identify each sum.

```
1 stack_ = phaseSpike.array() * 1 + phaseBorder.array()*2 +
      lowContrast.array()*4;
```



Context

- The client application runs on Windows 10
- The server is a linux application
- WSL allows to run linux apps on Windows (major distribution are available)
- But how about creating a service for surface reconstruction deployed on WSL but could be accessible to other machine for analysis

A http service

Use JSON as format exchange and Representational state transfe (REST)[a] API to create a HTTP service

- list all cases or specific cases
- configure cases depending on polymap calibration
- run reconstruction on entire image
- run reconstruction on a region using a mask
- compute indicators

[a] Representational state transfer (REST) is a software architectural style that defines a set of constraints to be used for creating Web services – Wikipedia

Specs for the exchanges

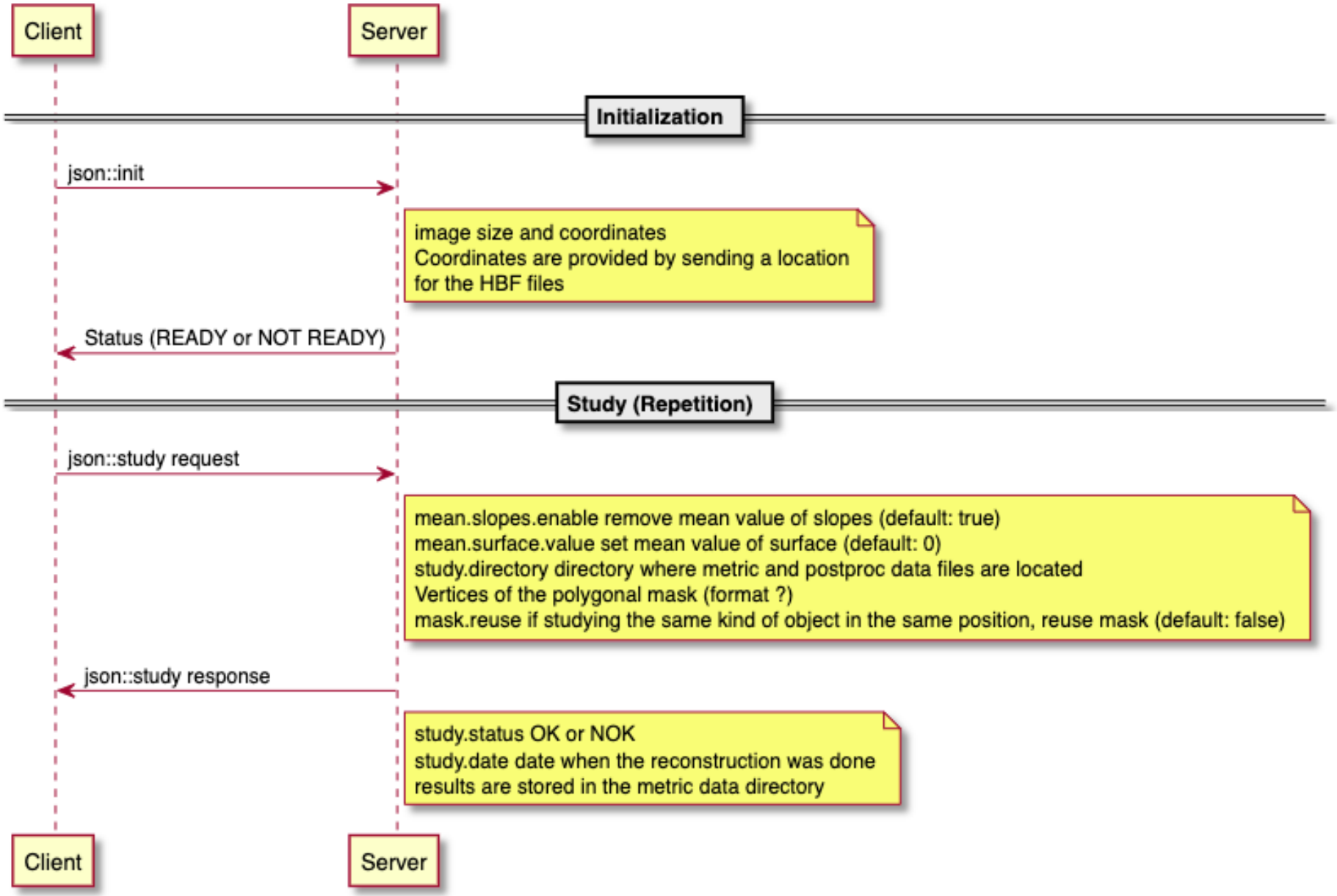
Use JSON:API

A Request Example using JSON:API

```
        https://localhost:8080/cases

```

```
HTTP/1.1 200 OK
Content-Type: application/vnd.api+json

{ "data": [  {
   "attributes": {...
    },
   "id": "deformed",
   "links": {    "self": "http://cemosis.feelpp.org:8080/
    cases/deformed"    },
   "type": "cases"  },
  {
   "attributes": { ... },
   "id": "mesureEtalonSin",
   "links": {    "self": "http://cemosis.feelpp.org:8080/
    cases/mesureEtalonSin"   },
   "type": "cases"
  }, ] }

```

Client/Server communications



Thanks

Thanks to Christophe PRUDHOMME and Vincent CHALVIDAN