Systematic doming error in Structure-from-Motion (SfM) derived topographic data

Topographic data such as point clouds derived from vertical imagery often show large scale deformations that are due to radial lens distortions. Small errors in the calibration of the camera model propagate into elevation estimates, leading to a systematic error often expressed as a central *doming* (e.g. Rosnell and Honkavaara, 2012; Javernick et al., 2014; James and Robson, 2014).

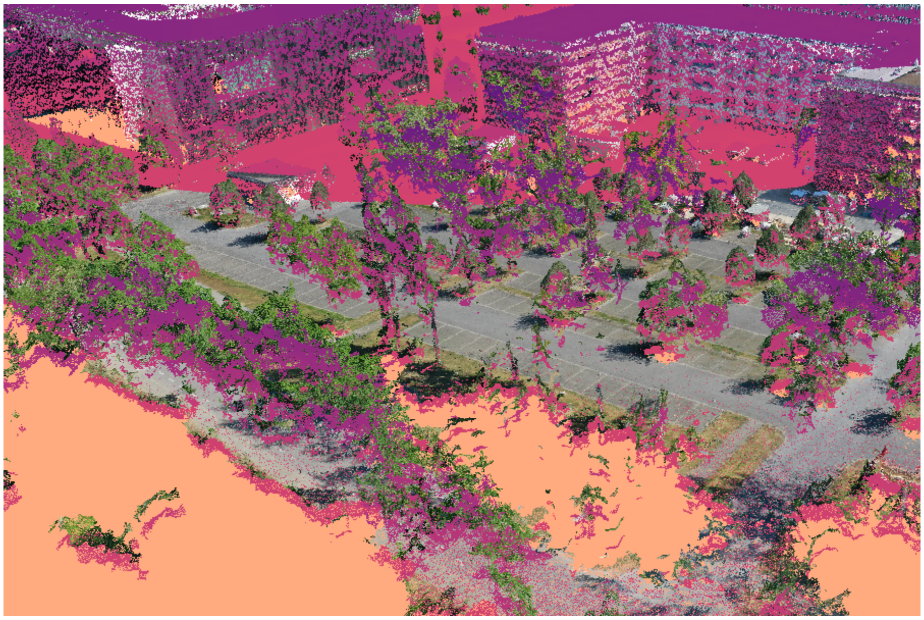

Figure 1: Campus Golm SfM doming error

We quantify this systematic doming error by the z-component of distance vectors between SfM points and corresponding closest points in a geo-referenced lidar point cloud. CloudCompare can compute these cloud-to-cloud distances and can store them inside LAS files.

Reading such a LAS file in Python using the module **laspy**:

```
fname = 'data/...c2cdistances.las'
print('reading', fname)
f = laspy.file.File(fname)
x = f.x
y = f.y
z = f.z
classification = f.classification
dz = getattr(f,'c2c_absolute_distances_(z)').copy()
```

```
f.close()
```

Vertical distances **dz** can get quite large at steep surfaces such as from buildings, but also due to changes in vegetation and constructions. Therefore, we limit the point cloud only to ground classified points with moderate **dz** (-2, 2):

```
print('c2c absolute distances (z):')
print('pre min, mean, max:', dz.min(), dz.mean(), dz.max())
j = (-2 <= dz) * (dz <= 2)
xj, yj, dzj = x[j], y[j], dz[j]
dzj *= 10
print('post min, mean, max:', dzj.min(), dzj.mean(), dzj.max())

print('extract ground class ..')
pts = np.transpose((xj, yj, dzj))
pts = pts[classification[j] == 2]
```

As the error model for the systematic doming error we choose a sphere. A least squares fit to a 3D sphere can be done in the following way:

```
def fitsphere(pts):
    # CuPy should be faster for large point clouds
    # i.e., pts.shape[0] > 1e5
    from scipy.linalg import lstsq

    A = np.zeros((pts.shape[0], 4))
    x, y, z = pts[:,0], pts[:,1], pts[:,2]
    A[:,0] = x * 2
    A[:,1] = y * 2
    A[:,2] = z * 2
    A[:,3] = 1
    f = np.zeros((len(x), 1))
    f[:,0] = x*x + y*y + z*z
    c, resid, rank, sval = lstsq(A, f)
    r = np.sqrt(c[0]*c[0] + c[1]*c[1] + c[2]*c[2] + c[3])

    # return radius and center of sphere
    return (r, c[0], c[1], c[2])
```

Applied to the point cloud:

```
print('fit spherical error model to ground component ..')
ptsmin = pts.min(axis = 0)
R, x0, y0, z0 = fitsphere(pts-ptsmin)
x0 += ptsmin[0]
y0 += ptsmin[1]
z0 += ptsmin[2]
```

```python

# transform (x, y, z) -> (x, y, z-zsphere)
zsphere = z0 - np.sqrt(R*R - (x-x0)**2-(y-y0)**2)
zsphere /= 10

# corrected point cloud cpts
cpts = np.transpose((x, y, z-zsphere))

print('export corrected point cloud to LAZ file ..')
exportlas('%s_spherical_corrected.laz' % fname[:-4], cpts[:,2], cpts)
```

The residuals **r** are given by **r** = **dz - zsphere** and if the error model is correct these should be small and flat in space. We test the flatness with a least squares plane fit on the residuals:

```python
A = np.transpose((x, y, np.ones(x.shape)))
c, resid, rank, sval = lstsq(A, dz - zsphere)
print(c)

# z of the residual plane
zrp = c[0]*x + c[1]*y + c[2]
```

Exporting to LAS files can be done with the following function:

```python
def exportlas(fname, var, pnts, cmap = None, classification = None):
    import laspy
    import os
    from subprocess import call

    if cmap is None:
        from matplotlib.cm import magma_r
        cmap = magma_r

    fn, fe = os.path.splitext(fname)

    # var -> v: [0,1]
    v = var - np.min(var)
    v /= v.max()

    # RGB colors
    rgb = cmap(v)
    rgb = rgb[:, :3]
    rgb *= 65535
    rgb = rgb.astype('uint')

    # LAS file
    header = laspy.header.Header()
```

```
    header.data_format_id = 2
    f = laspy.file.File('%s.las' % fn, mode = 'w', header = header)
    f.header.scale = [0.001, 0.001, 0.001]
    f.header.offset = [pnts[:,0].min(), pnts[:,1].min(), pnts[:,2].min()]
    f.x = pnts[:, 0]
    f.y = pnts[:, 1]
    f.z = pnts[:, 2]
    if classification is not None:
        f.classification = classification
    f.set_red(rgb[:, 0])
    f.set_green(rgb[:, 1])
    f.set_blue(rgb[:, 2])
    f.close()

    # try laszip
    try:
        r = call(['laszip', '%s.las' % fn])
        if not r:
            r = call(['rm', '%s.las' % fn])
    except:
        pass
```