Accurate, Dense, and Robust Multi-View Stereopsis

Authors:
Raúl Romaní Flores
Paul Alonzo Quio Añamuro

Source code compilation

Update `../base/numeric/mylapack.cc` From:

```
extern "C" {
#include <clapack/f2c.h>
#include <clapack/clapack.h>
};
```

To:

```
extern "C" {
//#include <clapack/f2c.h>
//#include <clapack/clapack.h>
#include <lapacke.h>
};
#define integer int
```

Update `../base/numeric/mylapack.h` From:

```
static void lls(std::vector<float>& A,
                std::vector<float>& b,
                long int width, long int height);

static void lls(std::vector<double>& A,
                std::vector<double>& b,
                long int width, long int height);
```

To:

```
static void lls(std::vector<float>& A,
                std::vector<float>& b,
                int width, int height);

static void lls(std::vector<double>& A,
                std::vector<double>& b,
                int width, int height);
```

Source code compilation

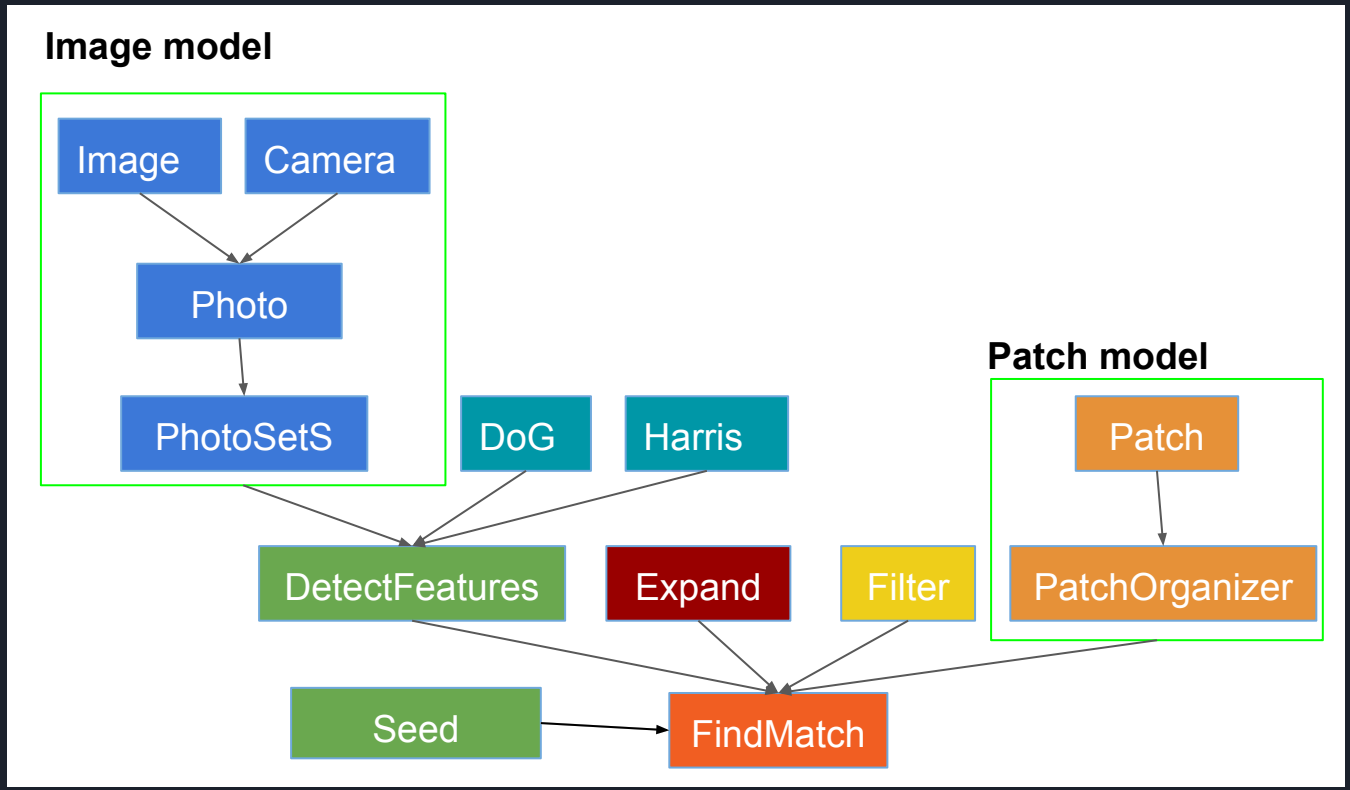
Install some dependencies

- libgtk2.0-dev
- libglew1.6-dev
- libglew1.6
- libdevil-dev
- libboost-all-dev
- libatlas-cpp-0.6-dev
- libatlas-dev
- imagemagick
- libatlas3gf-base
- libcminpack-dev
- libgfortran3
- libmetis-edf-dev
- libparmetis-dev
- freeglut3-dev
- libgsl0-dev
- libblas-dev
- liblapack-dev
- liblapacke-dev
- libjpeg-dev

And use make to generate executable files

```
alonzo@alonzo-VirtualBox:~/Documents/program/main$ make
g++ -O2 -Wall -Wno-deprecated    -c -o pmvs2.o pmvs2.cc
g++ -c -O2 -Wall -Wno-deprecated  ../base/pmvs/detectFeatures.cc
g++ -c -O2 -Wall -Wno-deprecated  ../base/pmvs/dog.cc
g++ -c -O2 -Wall -Wno-deprecated  ../base/pmvs/harris.cc
g++ -c -O2 -Wall -Wno-deprecated  ../base/pmvs/point.cc
g++ -c -O2 -Wall -Wno-deprecated  ../base/pmvs/detector.cc
g++ -c -O2 -Wall -Wno-deprecated  ../base/pmvs/findMatch.cc
g++ -c -O2 -Wall -Wno-deprecated  ../base/pmvs/expand.cc
g++ -c -O2 -Wall -Wno-deprecated  ../base/pmvs/filter.cc
g++ -c -O2 -Wall -Wno-deprecated  ../base/pmvs/optim.cc
../base/pmvs/optim.cc: In static member function ‘static double PMVS3::Coptim::my
_f_ssd(const gsl_vector*, void*)’:
../base/pmvs/optim.cc:762:9: warning: variable ‘flag’ set but not used [-Wunused-
but-set-variable]
     int flag;
         ^
g++ -c -O2 -Wall -Wno-deprecated  ../base/pmvs/patchOrganizerS.cc
g++ -c -O2 -Wall -Wno-deprecated  ../base/pmvs/seed.cc
g++ -c -O2 -Wall -Wno-deprecated  ../base/pmvs/option.cc
g++ -c -O2 -Wall -Wno-deprecated  ../base/image/image.cc
g++ -c -O2 -Wall -Wno-deprecated  ../base/image/camera.cc
g++ -c -O2 -Wall -Wno-deprecated  ../base/image/photoSetS.cc
g++ -c -O2 -Wall -Wno-deprecated  ../base/pmvs/patch.cc
g++ -c -O2 -Wall -Wno-deprecated  ../base/image/photo.cc
g++ -c -O2 -Wall -Wno-deprecated  ../base/numeric/mylapack.cc
g++  -lXext -lX11 -ljpeg -lm -lpthread -llapack -lgsl -lgslcblas -o pmvs2 pmvs2.o
 detectFeatures.o dog.o harris.o point.o detector.o findMatch.o expand.o filter.o
 optim.o patchOrganizerS.o seed.o option.o image.o camera.o photoSetS.o patch.o p
hoto.o mylapack.o  -lXext -lX11 -ljpeg -lm -lpthread -llapack -lgsl -lgslcblas
alonzo@alonzo-VirtualBox:~/Documents/program/main$ ls
camera.o          findMatch.o       liblapack.so.3  patchOrganizerS.o  point.o
detectFeatures.o  harris.o          Makefile        photo.o            run0.sh
detector.o        image.o           mylapack.o      photoSetS.o        run1.sh
dog.o             libblas.so.3      optim.o         pmvs2              run2.sh
expand.o          libgslcblas.so.0  option.o        pmvs2.cc           seed.o
filter.o          libgsl.so.0       patch.o         pmvs2.o
alonzo@alonzo-VirtualBox:~/Documents/program/main$
```

Project Class diagram



Main process of the algorithm

PMVS.cc

```
PMVS3::CfindMatch findMatch;
findMatch.init(option);
findMatch.run();
```

FindMatch.cc

```
// Detect features if not yet done
CdetectFeatures df;
const int fcsize = 16;
df.run(m_pss, m_num, fcsize, m_level, m_CPU);
```

1. Matching Detection

FindMatch.cc

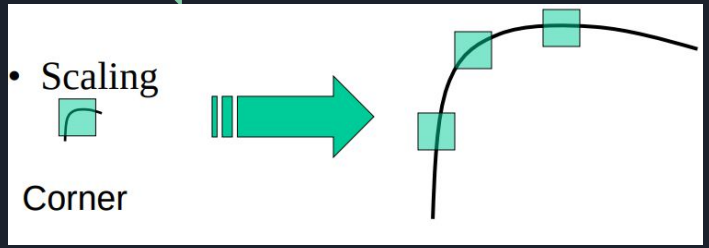
```
// Seed generation
m_seed.run();
m_seed.clear();

++m_depth;
m_pos.collectPatches();

//------------------------------
// Expansion
const int TIME = 3;
for (int t = 0; t < TIME; ++t) {
  m_expand.run();

  m_filter.run();

  updateThreshold();
```

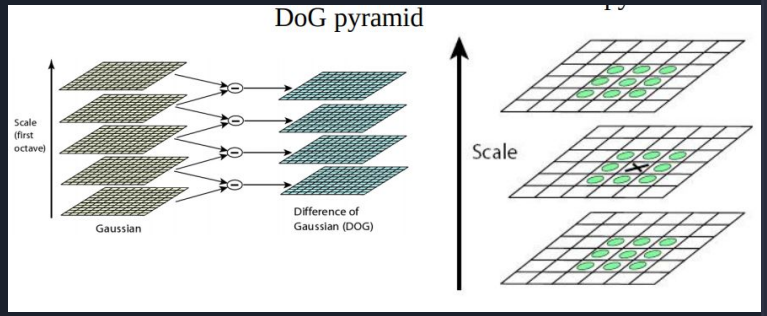
1.Matching Match across multiple pictures to reconstruct a sparse set of patches

2. Expansion dense set of patches

3. Filtering remove erroneos matches

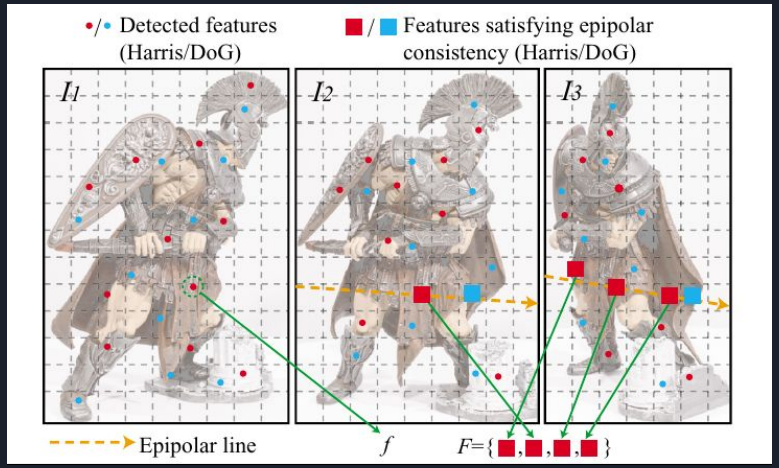
Detection






```
// Harris
{
    Charris harris;
    multiset<Cpoint> result;
    harris.run(m_ppss->m_photos[index].getImage(m_level),
               m_ppss->m_photos[index].Cimage::getMask(m_level),
               m_ppss->m_photos[index].Cimage::getEdge(m_level),
               m_ppss->m_photos[index].getWidth(m_level),
               m_ppss->m_photos[index].getHeight(m_level), m_csize, sigma, result);

    multiset<Cpoint>::reverse_iterator rbegin = result.rbegin();
    while (rbegin != result.rend()) {
        m_points[index].push_back(*rbegin);
        rbegin++;
    }
}

//-------------------------------------------------------------------------
// DoG
{
    Cdog dog;
    multiset<Cpoint> result;
    dog.run(m_ppss->m_photos[index].getImage(m_level),
            m_ppss->m_photos[index].Cimage::getMask(m_level),
            m_ppss->m_photos[index].Cimage::getEdge(m_level),
            m_ppss->m_photos[index].getWidth(m_level),
            m_ppss->m_photos[index].getHeight(m_level),
            m_csize, firstScale, lastScale, result);

    multiset<Cpoint>::reverse_iterator rbegin = result.rbegin();
    while (rbegin != result.rend()) {
        m_points[index].push_back(*rbegin);
        rbegin++;
    }
}
```

Matching




```
for (int p = 0; p < (int)m_ppoints[index][index2].size(); ++p) {
    // collect features that satisfies epipolar geometry
    // constraints and sort them according to the differences of
    // distances between two cameras.
    vector<Ppoint> vcp;
    collectCandidates(index, indexes,
                      *m_ppoints[index][index2][p], vcp);

    int count = 0;
    Cpatch bestpatch;
    //=========================================================================
    for (int i = 0; i < (int)vcp.size(); ++i) {
        Cpatch patch;
        patch.m_coord = vcp[i]->m_coord;
        patch.m_normal =
            m_fm.m_pss.m_photos[index].m_center - patch.m_coord;

        unitize(patch.m_normal);
        patch.m_normal[3] = 0.0;
        patch.m_flag = 0;

        ++m_fm.m_pos.m_counts[index][index2];
        const int ix = ((int)floor(vcp[i]->m_icoord[0] + 0.5f)) / m_fm.m_csize;
        const int iy = ((int)floor(vcp[i]->m_icoord[1] + 0.5f)) / m_fm.m_csize;
        const int index3 = iy * m_fm.m_pos.m_gwidths[vcp[i]->m_itmp] + ix;
        if (vcp[i]->m_itmp < m_fm.m_tnum)
            ++m_fm.m_pos.m_counts[vcp[i]->m_itmp][index3];

        const int flag = initialMatchSub(index, vcp[i]->m_itmp, id, patch);
        if (flag == 0) {
            ++count;
            if (bestpatch.score(m_fm.m_nccThreshold) <
                    patch.score(m_fm.m_nccThreshold))
                bestpatch = patch;
            if (m_fm.m_countThreshold0 <= count)
                break;
        }
    }
    if (count != 0) {
        Ppatch ppatch(new Cpatch(bestpatch));
        m_fm.m_pos.addPatch(ppatch);
        ++totalcount;
        break;
    }
}
```

Matching

```
// Set vImages vGrids.
if (m_fm.m_depth) {
    m_fm.m_pos.setVImagesVGrids(patch);

    if (2 <= m_fm.m_depth && check(patch))
        return 1;
}
```

```
int Coptim::check(Cpatch& patch) {
    const float gain = m_fm.m_filter.computeGain(patch, 1);
    patch.m_tmp = gain;

    if (gain < 0.0) {
        patch.m_images.clear();
        return 1;
    }

    {
        vector<Ppatch> neighbors;
        m_fm.m_pos.findNeighbors(patch, neighbors, 1, 4, 2);
        // Only check when enough number of neighbors
        if (6 < (int)neighbors.size() &&
                //if (8 < (int)neighbors.size() &&
                m_fm.m_filter.filterQuad(patch, neighbors)) {
            patch.m_images.clear();
            return 1;
        }
    }

    return 0;
}
```

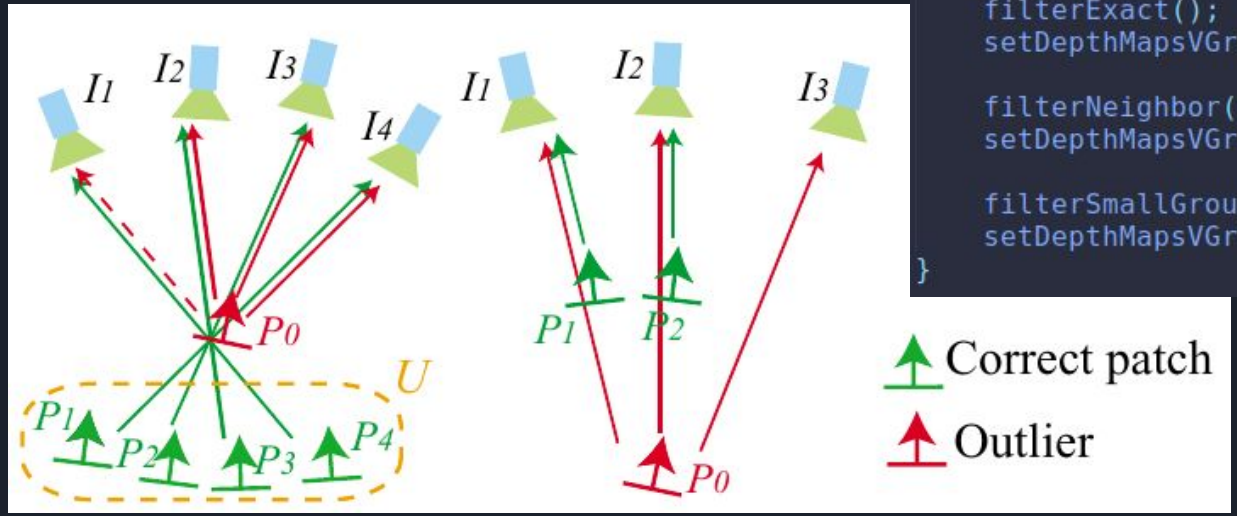
```
// starting with (index, indexs), set visible images by looking at correlation.
int Cseed::initialMatchSub(const int index0, const int index1,
                           const int id, Cpatch& patch) {
    //-------------------------------------------------------------------------
    patch.m_images.clear();
    patch.m_images.push_back(index0);
    patch.m_images.push_back(index1);

    ++m_scounts[id];

    //-------------------------------------------------------------------------
    // We know that patch.m_coord is inside bimages and inside mask
    if (m_fm.m_optim.preProcess(patch, id, 1)) {
        ++m_fcounts0[id];
        return 1;
    }

    //-------------------------------------------------------------------------
    m_fm.m_optim.refinePatch(patch, id, 100);

    //-------------------------------------------------------------------------
    if (m_fm.m_optim.postProcess(patch, id, 1)) {
        ++m_fcounts1[id];
        return 1;
    }

    ++m_pcounts[id];
    //-------------------------------------------------------------------------
    return 0;
}
```

Expansion

```
void Cexpand::expandThread(void) {
    pthread_rwlock_wrlock(&m_fm.m_lock);
    const int id = m_fm.m_count++;
    pthread_rwlock_unlock(&m_fm.m_lock);

    while (1) {
        Ppatch ppatch;
        int empty = 0;
        pthread_rwlock_wrlock(&m_fm.m_lock);
        if (m_queue.empty())
            empty = 1;
        else {
            ppatch = m_queue.top();
            m_queue.pop();
        }
        pthread_rwlock_unlock(&m_fm.m_lock);

        if (empty)
            break;

        // For each direction;
        vector<vector<Vec4f> > canCoords;
        findEmptyBlocks(ppatch, canCoords);

        for (int i = 0; i < (int)canCoords.size(); ++i) {
            for (int j = 0; j < (int)canCoords[i].size(); ++j) {
                const int flag = expandSub(ppatch, id, canCoords[i][j]);
                // fail
                if (flag)
                    ppatch->m_dflag |= (0x0001) << i;
            }
        }
    }
}
```

Filter

```
void Cfilter::run(void) {
    setDepthMapsVGridsVPGridsAddPatchV(0);

    filterOutside();
    setDepthMapsVGridsVPGridsAddPatchV(1);

    filterExact();
    setDepthMapsVGridsVPGridsAddPatchV(1);

    filterNeighbor(1);
    setDepthMapsVGridsVPGridsAddPatchV(1);

    filterSmallGroups();
    setDepthMapsVGridsVPGridsAddPatchV(1);
}
```



Filter outside

```
for (int j = 0; j < (int)m_fm.m_pos.m_pgrids[index][index2].size(); ++j) {
    const float bdepth = m_fm.m_pss.computeDepth(index,
                         m_fm.m_pos.m_pgrids[index][index2][j]->m_coord);
    if (pdepth < bdepth &&
            !m_fm.isNeighbor(*ppatch, *m_fm.m_pos.m_pgrids[index][index2][j],
                             m_fm.m_neighborThreshold1)) {
        maxpressure = max(maxpressure,
                          m_fm.m_pos.m_pgrids[index][index2][j]->m_ncc -
                          m_fm.m_nccThreshold);
    }
}
m_gains[p] -= maxpressure;
```

```
void Cfilter::filterOutside(void) {
    struct timeval tv;
    gettimeofday(&tv, NULL);
    time_t curtime = tv.tv_sec;
    cerr << "FilterOutside" << endl;
    //??? notice (1) here to avoid removing m_fix=1
    m_fm.m_pos.collectPatches(1);

    const int psize = (int)m_fm.m_pos.m_ppatches.size();
    m_gains.resize(psize);

    cerr << "mainbody: " << flush;

    m_fm.m_count = 0;
    pthread_t threads[m_fm.m_CPU];
    for (int i = 0; i < m_fm.m_CPU; ++i)
        pthread_create(&threads[i], NULL, filterOutsideThreadTmp,
    for (int i = 0; i < m_fm.m_CPU; ++i)
        pthread_join(threads[i], NULL);
    cerr << endl;

    // delete patches with positive m_gains
    int count = 0;

    double ave = 0.0f;
    double ave2 = 0.0f;
    int denom = 0;

    for (int p = 0; p < psize; ++p) {
        ave += m_gains[p];
        ave2 += m_gains[p] * m_gains[p];
        ++denom;

        if (m_gains[p] < 0.0) {
            m_fm.m_pos.removePatch(m_fm.m_pos.m_ppatches[p]);
            count++;
        }
    }
```

Filter exact

```
int index = -1;
for (int y = 0; y < h; ++y) {
    for (int x = 0; x < w; ++x) {
        ++index;
        for (int i = 0; i < (int)m_fm.m_pos.m_pgrids[image][index].size(); ++i) {
            const Cpatch& patch = *m_fm.m_pos.m_pgrids[image][index][i];
            if (patch.m_fix)
                continue;

            int safe = 0;

            if (m_fm.m_pos.isVisible(patch, image, x, y, m_fm.m_neighborThreshold1, 0))
                safe = 1;
            // use 4 neighbors?
            else if (0 < x && m_fm.m_pos.isVisible(patch, image, x - 1, y, m_fm.m_neighborThreshold1, 0))
                safe = 1;
            else if (x < w - 1 && m_fm.m_pos.isVisible(patch, image, x + 1, y, m_fm.m_neighborThreshold1, 0))
                safe = 1;
            else if (0 < y && m_fm.m_pos.isVisible(patch, image, x, y - 1, m_fm.m_neighborThreshold1, 0))
                safe = 1;
            else if (y < h - 1 && m_fm.m_pos.isVisible(patch, image, x, y + 1, m_fm.m_neighborThreshold1, 0))
                safe = 1;

            if (safe) {
                newimages[patch.m_id].push_back(image);
                newgrids[patch.m_id].push_back(TVec2<int>(x, y));
            }
            else {
                removeimages[patch.m_id].push_back(image);
                removegrids[patch.m_id].push_back(TVec2<int>(x, y));
            }
        }
    }
}
```

Filter neighbor

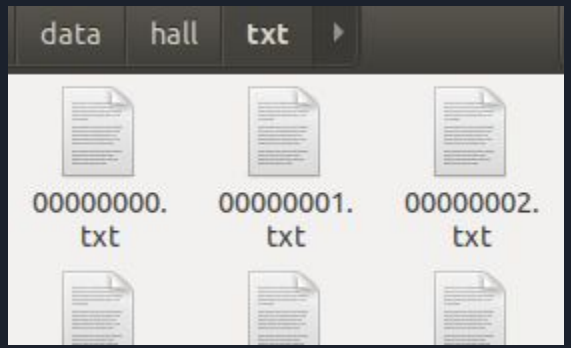
```
while (bpatch != epatch) {
    if (m_fm.isNeighborRadius(patch, **bpatch,
                              unit,
                              m_fm.m_neighborThreshold * scale,
                              radius))
        neighbors.push_back(*bpatch);
    ++bpatch;
}
```

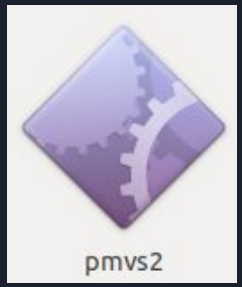
```
void Cfilter::filterNeighborThread(void) {
    const int size = (int)m_fm.m_pos.m_ppatches.size();
    while (1) {
        int jtmp = -1;
        pthread_rwlock_wrlock(&m_fm.m_lock);
        if (!m_fm.m_jobs.empty()) {
            jtmp = m_fm.m_jobs.front();
            m_fm.m_jobs.pop_front();
        }
        pthread_rwlock_unlock(&m_fm.m_lock);
        if (jtmp == -1)
            break;

        const int begin = m_fm.m_junit * jtmp;
        const int end = min(size, m_fm.m_junit * (jtmp + 1));

        for (int p = begin; p < end; ++p) {
            Ppatch& ppatch = m_fm.m_pos.m_ppatches[p];
            if (m_rejects[p])
                continue;

            vector<Ppatch> neighbors;
            //m_fm.m_pos.findNeighbors(*ppatch, neighbors, 0, 4, 2);
            m_fm.m_pos.findNeighbors(*ppatch, neighbors, 0, 4, 2, 1);

            //?? new filter
            if ((int)neighbors.size() < 6)
                //if ((int)neighbors.size() < 8)
                m_rejects[p] = m_time + 1;
            else {
                // Fit a quadratic surface
                if (filterQuad(*ppatch, neighbors))
                    m_rejects[p] = m_time + 1;
            }
        }
    }
}
```

Demo execution

```
alonzo@alonzo-desktop:~/Documentos/Projects/pmvs-2/program/main$
./pmvs2 "/home/alonzo/Documentos/Projects/pmvs-2/data/hall/" opti
on.txt

./pmvs2
/home/alonzo/Documentos/Projects/pmvs-2/data/hall/
option.txt----------------------------------------------
--- Summary of specified options ---
# of timages: 61 (range specification)
# of oimages: 0 (enumeration)
level: 2  csize: 2
threshold: 0.7  wsize: 7
minImageNum: 3  CPU: 4
useVisData: 1  sequence: -1
-----------------------------------------------
Reading images: **********
```

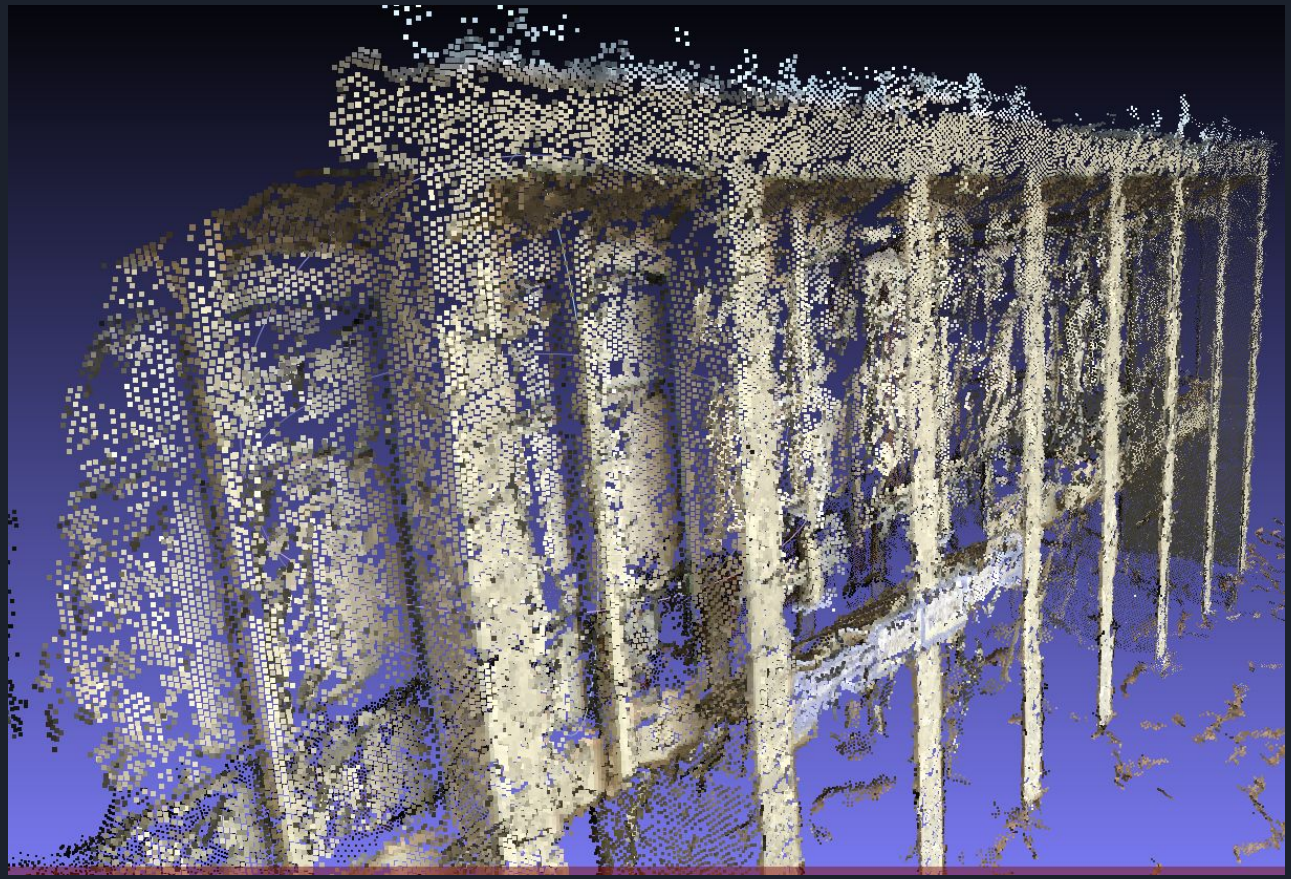
Demo execution





Results

Results - Meshlab



Thanks