Accurate, Dense, and Robust Multi-View Stereopsis

Authors:
Raúl Romaní Flores
Paul Alonzo Quio Añamuro

Source code compilation

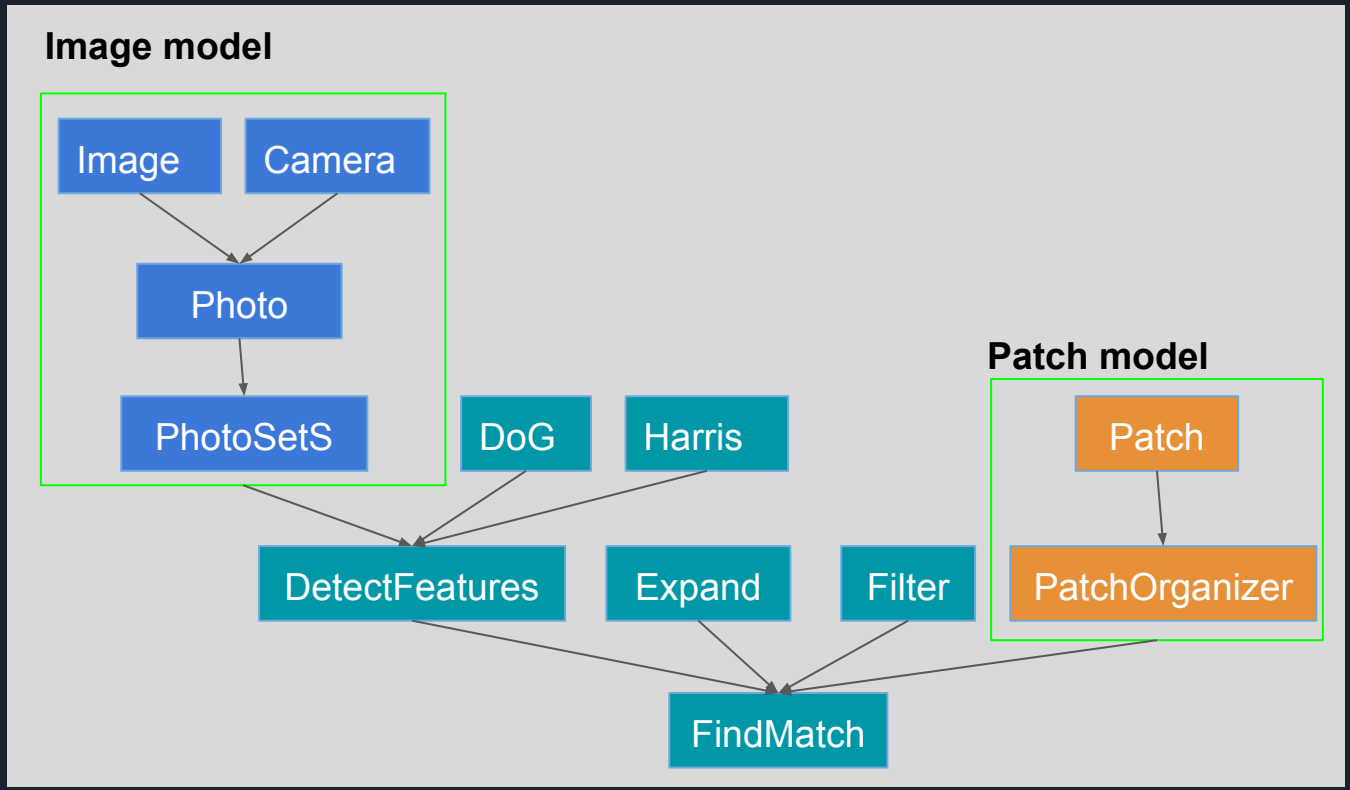
Install some dependencies

- 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);
```

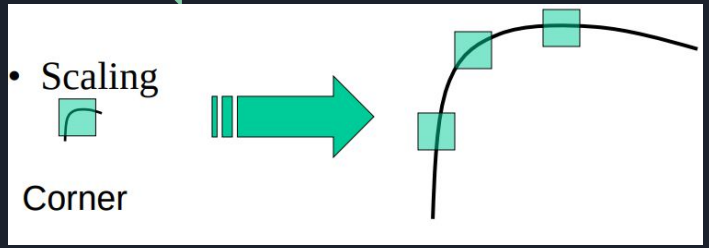
1. Matching

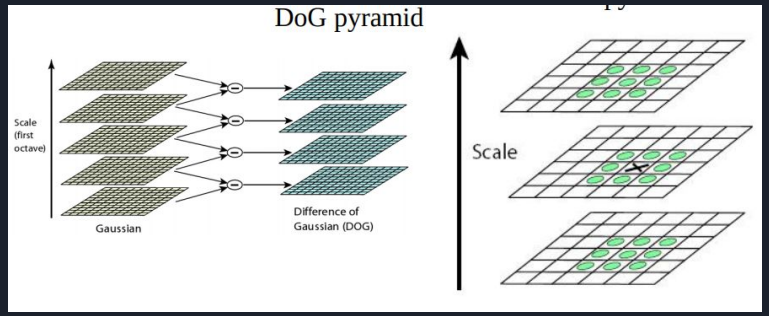
FindMatch.cc

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

  m_filter.run();

  updateThreshold();

  cout << "STATUS: ";
  for (int i = 0; i < (int)m_optim.m_status.size(); ++i) {
    cout << m_optim.m_status[i] << ' ';
    if (i % 10 == 9)
      cout << endl;
  }
  cout << endl;

  ++m_depth;
}
```

2. Expansion
3. Filtering

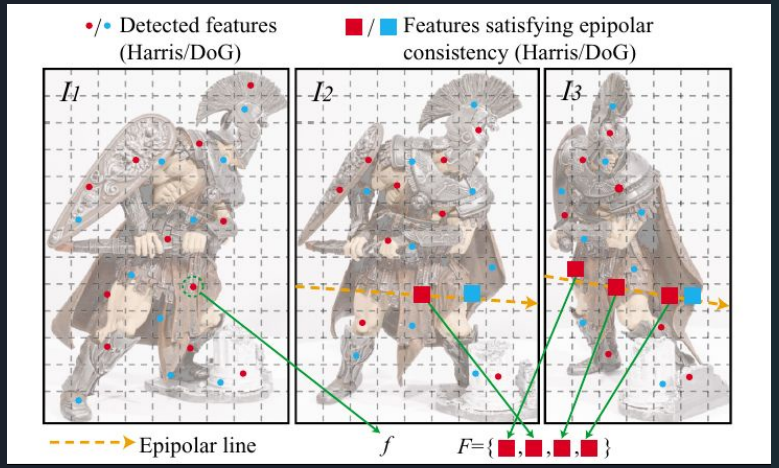
Detection





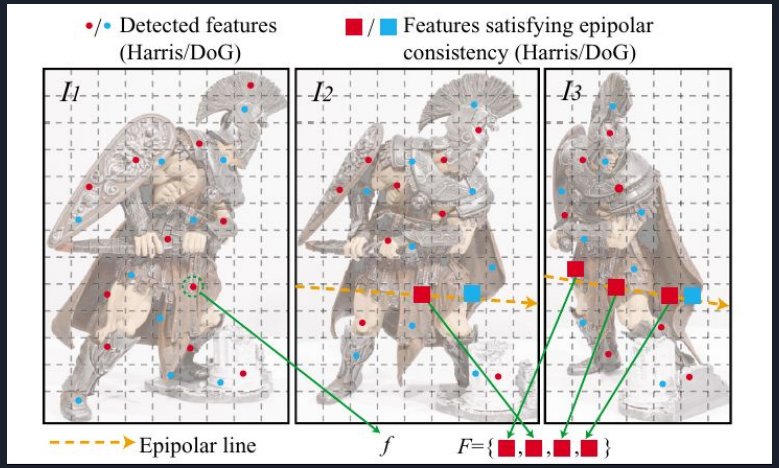

```
// Harris
{
    Charris harris;
    multiset<Cpoint> result;
    harris.run(m_ppss->m_photos[index].getImage(m_level),
               m_ppss->m_photos[index].Cimage::getMask(m_level),
               m_ppss->m_photos[index].Cimage::getEdge(m_level),
               m_ppss->m_photos[index].getWidth(m_level),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



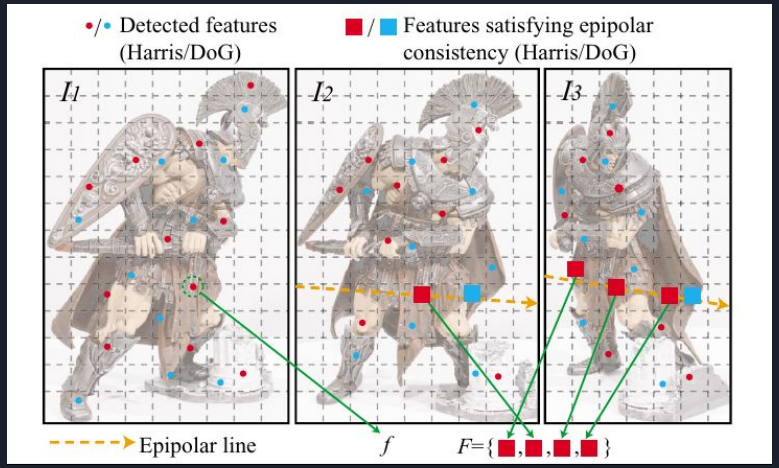

```
void Cseed::initialMatch(const int index, const int id) {
    vector<int> indexes;
    m_fm.m_optim.collectImages(index, indexes);

    if (m_fm.m_tau < (int)indexes.size())
        indexes.resize(m_fm.m_tau);

    if (indexes.empty())
        return;

    int totalcount = 0;
    //=================================================================
    // for each feature point, starting from the optical center, keep on
    // matching until we find candidateThreshold patches
    const int gheight = m_fm.m_pos.m_gheights[index];
    const int gwidth = m_fm.m_pos.m_gwidths[index];
    int index2 = -1;
    for (int y = 0; y < gheight; ++y) {
        for (int x = 0; x < gwidth; ++x) {
            ++index2;
            if (!canAdd(index, x, y))
                continue;
            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;
```

Matching




```
            //=============================================================================
            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




```
// 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::run(void) {
    m_fm.m_count = 0;
    m_fm.m_jobs.clear();
    m_ecounts.resize(m_fm.m_CPU);
    m_fcounts0.resize(m_fm.m_CPU);
    m_fcounts1.resize(m_fm.m_CPU);
    m_pcounts.resize(m_fm.m_CPU);
    fill(m_ecounts.begin(), m_ecounts.end(), 0);
    fill(m_fcounts0.begin(), m_fcounts0.end(), 0);
    fill(m_fcounts1.begin(), m_fcounts1.end(), 0);
    fill(m_pcounts.begin(), m_pcounts.end(), 0);

    time_t starttime = time(NULL);

    m_fm.m_pos.clearCounts();
    m_fm.m_pos.clearFlags();

    if (!m_queue.empty()) {
        cerr << "Queue is not empty in expand" << endl;
        exit (1);
    }
    // set queue
    m_fm.m_pos.collectPatches(m_queue);

    cerr << "Expanding patches..." << flush;
    pthread_t threads[m_fm.m_CPU];
    for (int c = 0; c < m_fm.m_CPU; ++c)
        pthread_create(&threads[c], NULL, expandThreadTmp, (void*)this);
    for (int c = 0; c < m_fm.m_CPU; ++c)
        pthread_join(threads[c], NULL);

    cerr << endl
         << "---- EXPANSION: " << (time(NULL) - starttime) << " secs ----" << endl;

    const int trial = accumulate(m_ecounts.begin(), m_ecounts.end(), 0);
    const int fail0 = accumulate(m_fcounts0.begin(), m_fcounts0.end(), 0);
    const int fail1 = accumulate(m_fcounts1.begin(), m_fcounts1.end(), 0);
    const int pass = accumulate(m_pcounts.begin(), m_pcounts.end(), 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;
      }
    }
  }
}
```

Expansion

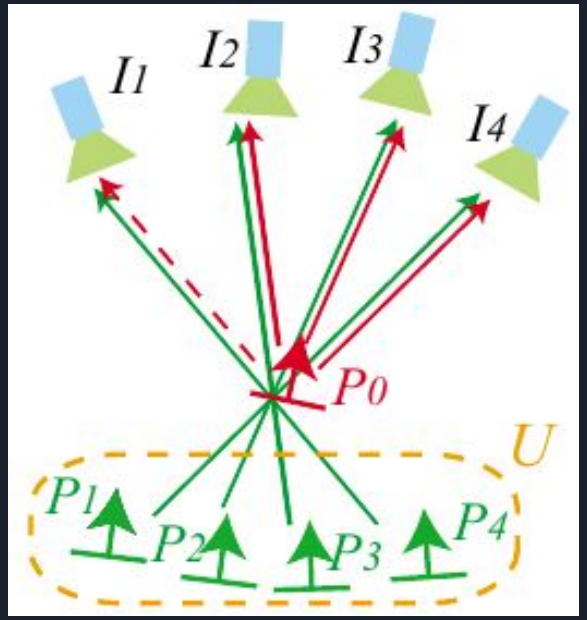
```
int Cexpand::expandSub(const Ppatch& orgppatch, const int id,
                       const Vec4f& canCoord) {
  // Choose the closest one
  Cpatch patch;
  patch.m_coord = canCoord;
  patch.m_normal = orgppatch->m_normal;
  patch.m_flag = 1;

  m_fm.m_pos.setGridsImages(patch, orgppatch->m_images);
  if (patch.m_images.empty())
    return 1;

  //------------------------------------------------------------
  // Check bimages and mask. Then, initialize possible visible images
  if (m_fm.m_pss.getMask(patch.m_coord, m_fm.m_level) == 0 ||
      m_fm.insideBimages(patch.m_coord) == 0)
    return 1;

  // Check m_counts and maybe m_pgrids
  const int flag = checkCounts(patch);
  if (flag)
    return 1;

  // Check edge
  m_fm.m_optim.removeImagesEdge(patch);
  if (patch.m_images.empty())
    return 1;

  ++m_ecounts[id];
  //------------------------------------------------------------
  // Preprocess
  if (m_fm.m_optim.preProcess(patch, id, 0)) {
    ++m_fcounts0[id];
    return 1;
  }
```

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

  //------------------------------------------------------------
  if (m_fm.m_optim.postProcess(patch, id, 0)) {
    ++m_fcounts1[id];
    return 1;
  }
  ++m_pcounts[id];

  //------------------------------------------------------------
  // Finally
  Ppatch ppatch(new Cpatch(patch));

  //patch.m_images = orgppatch->m_images;
  const int add = updateCounts(patch);

  m_fm.m_pos.addPatch(ppatch);

  if (add) {
    pthread_rwlock_wrlock(&m_fm.m_lock);
    m_queue.push(ppatch);
    pthread_rwlock_unlock(&m_fm.m_lock);
  }

  return 0;
}
```

Filter

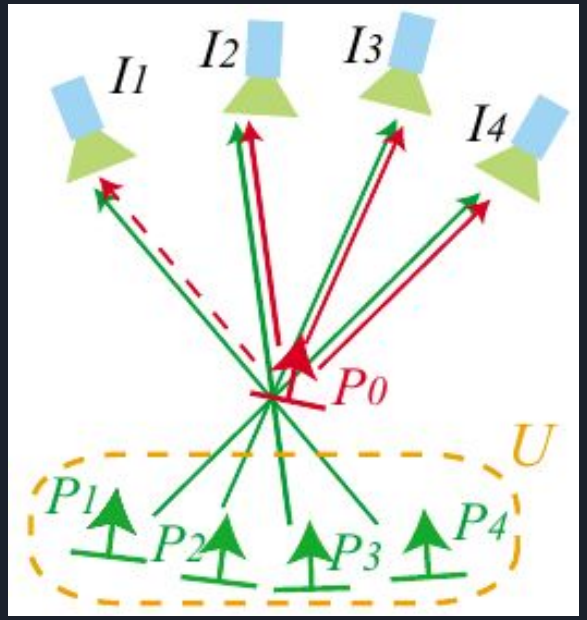
```
void Cfilter::run(void) {
    setDepthMapsVGridsVPGridsAddPatchV(0);

    filterOutside();
    setDepthMapsVGridsVPGridsAddPatchV(1);

    filterExact();
    setDepthMapsVGridsVPGridsAddPatchV(1);

    filterNeighbor(1);
    setDepthMapsVGridsVPGridsAddPatchV(1);

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

Filter outside



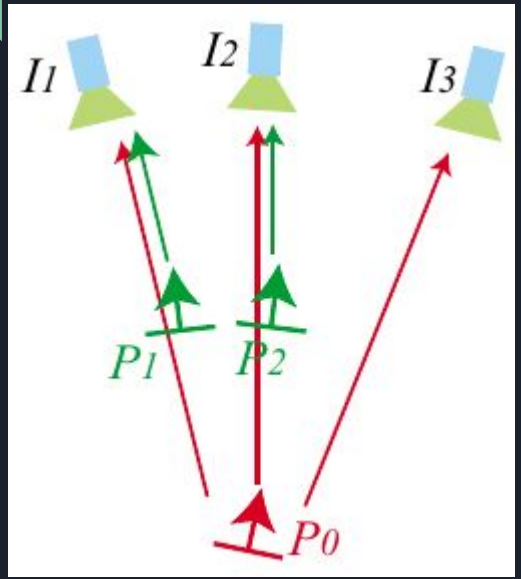

```
void Cfilter::filterOutsideThread(void) {
    pthread_rwlock_wrlock(&m_fm.m_lock);
    const int id = m_fm.m_count++;
    pthread_rwlock_unlock(&m_fm.m_lock);

    const int size = (int)m_fm.m_pos.m_ppatches.size();
    const int itmp = (int)ceil(size / (float)m_fm.m_CPU);
    const int begin = id * itmp;
    const int end = min(size, (id + 1) * itmp);

    for (int p = begin; p < end; ++p) {
        Ppatch& ppatch = m_fm.m_pos.m_ppatches[p];
        m_gains[p] = ppatch->score2(m_fm.m_nccThreshold);

        const int size = (int)ppatch->m_images.size();
        for (int i = 0; i < size; ++i) {
            const int& index = ppatch->m_images[i];
            if (m_fm.m_tnum <= index)
                continue;
            const int& ix = ppatch->m_grids[i][0];
            const int& iy = ppatch->m_grids[i][1];
            const int index2 = iy * m_fm.m_pos.m_gwidths[index] + ix;
            float maxpressure = 0.0f;
            for (int j = 0; j < (int)m_fm.m_pos.m_pgrids[index][index2].size(); ++j) {
                if (!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;
        }
```

Filter outside



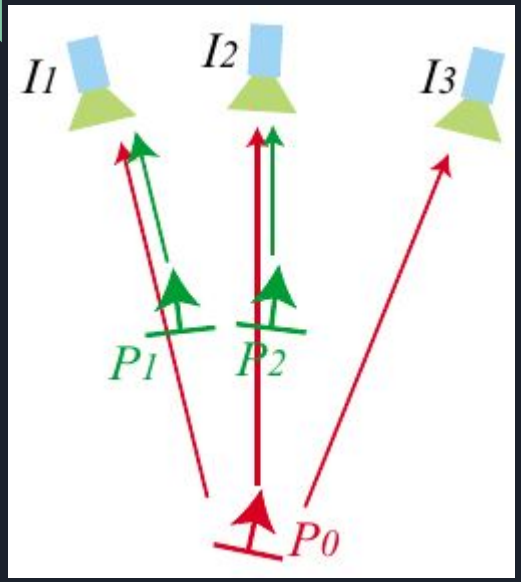

```
        const int vsize = (int)ppatch->m_vimages.size();
        for (int i = 0; i < vsize; ++i) {
            const int& index = ppatch->m_vimages[i];
            if (m_fm.m_tnum <= index)
                continue;
            const float pdepth = m_fm.m_pss.computeDepth(index, ppatch->m_coord);
            const int& ix = ppatch->m_vgrids[i][0];      const int& iy = ppatch->m_vgrids[i][1];
            const int index2 = iy * m_fm.m_pos.m_gwidths[index] + ix;
            float maxpressure = 0.0f;

            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;
        }
    }
}
```

Filter exact




```
void Cfilter::filterExactThread(void) {
    const int psize = (int)m_fm.m_pos.m_ppatches.size();
    vector<vector<int> > newimages, removeimages;
    vector<vector<TVec2<int> > > newgrids, removegrids;
    newimages.resize(psize);
    removeimages.resize(psize);
    newgrids.resize(psize);
    removegrids.resize(psize);

    while (1) {
        pthread_rwlock_wrlock(&m_fm.m_lock);
        const int image = m_fm.m_count++;
        pthread_rwlock_unlock(&m_fm.m_lock);

        if (m_fm.m_tnum <= image)
            break;

        cerr << '*' << flush;

        const int& w = m_fm.m_pos.m_gwidths[image];
        const int& h = m_fm.m_pos.m_gheights[image];
        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?
```

Filter exact

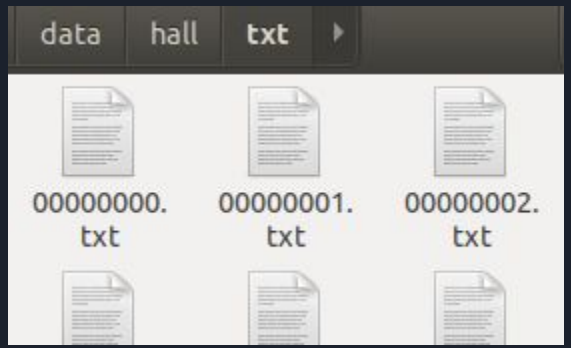


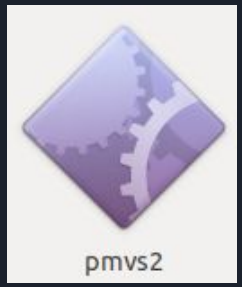

```
                    // use 4 neighbors?
                    else if (0 < x && m_fm.m_pos.isVisible(patch, image, x - 1, y, m_fm.m_neighborThreshold1, 0))
                        safe = 1;
                    else if (x < w - 1 && m_fm.m_pos.isVisible(patch, image, x + 1, y, m_fm.m_neighborThreshold1,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

    pthread_rwlock_wrlock(&m_fm.m_lock);
    for (int p = 0; p < psize; ++p) {
        m_newimages[p].insert(m_newimages[p].end(),
                              newimages[p].begin(), newimages[p].end());
        m_newgrids[p].insert(m_newgrids[p].end(),
                             newgrids[p].begin(), newgrids[p].end());
        m_removeimages[p].insert(m_removeimages[p].end(),
                                 removeimages[p].begin(), removeimages[p].end());
        m_removegrids[p].insert(m_removegrids[p].end(),
                                removegrids[p].begin(), removegrids[p].end());
    }
    pthread_rwlock_unlock(&m_fm.m_lock);
}
```

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: **********
```

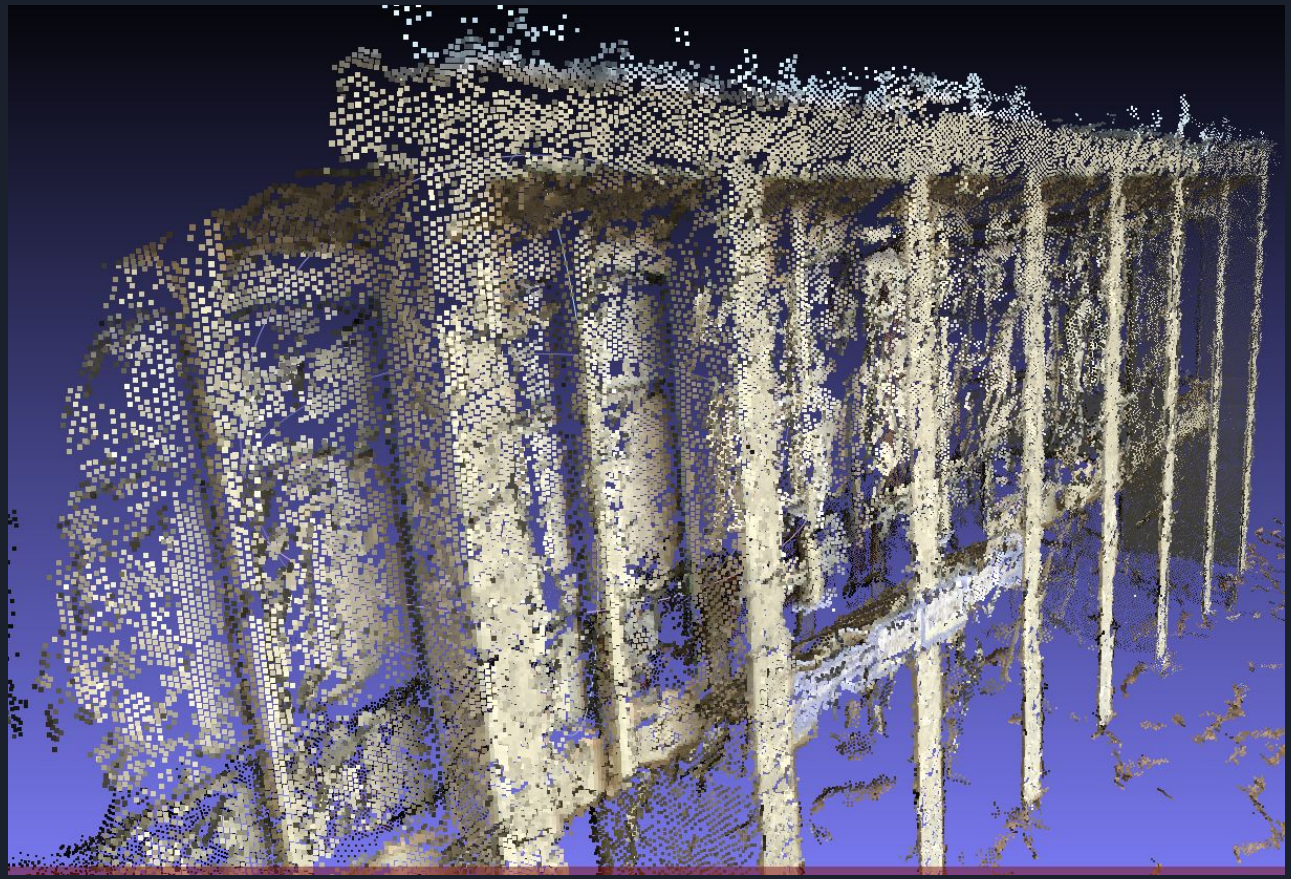
Demo execution





Results

Results - Meshlab



Thanks