Accurate, Dense, and Robust Multi-View Stereopsis

Raúl Romaní Flores * **Paul Alonzo, Quio Añamuro** **

* *Universidad Católica San Pablo (e-mail: raul.romani@ucsp.edu.pe).*
** *Universidad Católica San Pablo (e-mail: paul.quio@ucsp.edu.pe).*

AbstractIn this chapter proposed by Furukawa and Ponce [2010] we are going to see an algorithm that outputs a (quasi) dense set of rectangular patches covering the surfaces visible in the input images. The paper presents a multi-view stereopsis algorithm that has three main steps, matching, expansion and filtering. The paper also present a method for turning the resulting patch model into a mesh appropriate for image-based modeling. This algorithm was tested on several real datasets and achieving good results.

Keywords: Multi-view stereopsis, image-based modeling, camera calibration.

1. INTRODUCTION

The goal of multi-view stereo is to reconstruct a complete 3D object model from a collection of images taken from known camera viewpoints.

Most early work in multi-view stereopis tended to match and reconstruct all scene points independently (variational approach). Competing approaches mostly differ in the type of optimization techniques that they use. The variational approach has led to impressive progress, however, it typically requires determining a bounding volume (visual hull, bounding box or valid depth range) prior to initiating the optimization process, which not be feasible for outdoor scenes and/or cluttered images.

There are three classes of datasets, objects, scenes and crowded scenes. The algorithm effectively handles all three types of data, and outputs accurate object and scene models with fine surface detail despite low-texture regions, large concavities, and/or thin, high-curvature parts.

The multi-view stereopsis algorithm has three main steps: matching, expansion, and filtering. Matching, features found by Harris and Difference-of-Gaussians operators are matched across multiple pictures, yielding a sparse set of patches associated with salient image regions. Given these initial matches, the following two steps are repeated 3 times in all our experiments. Expansion, a technique similar to Lhuillier and Quan [2005] is used to spread the initial matches to nearby pixels and obtain a dense set of patches. (3) Filtering: visibility constraints are used to eliminate incorrect matches lying either in front or behind the observed surface.

2. KEY ELEMENTS OF THE PROPOSED APPROACH

Before getting into details of the multi-view stereopsis algorithm. we need to define some key elements.

A Patch p is a rectangle with center c(p), unit normal vector n(p) and a reference image R(p) oriented toward the cameras observing it; S(p) is the set of images where p should be visible, T(p) is the set of images where p is truly found. For each image is associated a rectangular grid composed of cells and attempt to reconstruct at least one patch in every cell. The key building blocks of the multi-view stereopsis algorithm are the methods for enforcing photometric consistency (by requiring that projected textures of every patch p be consistent in at least γ images) and enforcing visibility consistency(by requiring no patch p be occluded by any other patch in any image in the set of images where p should be visible S(p)).

Refer to the paper of Furukawa and Ponce [2010] for details.

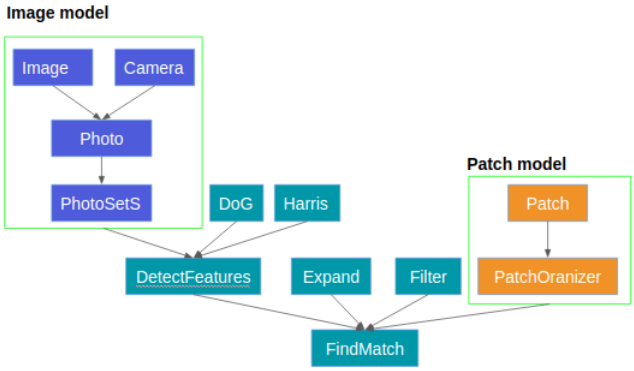

Figure 1. Class diagram of the C++ implementation of Furukawa and Ponce [2010] algorithm.

3. ALGORITHM

The multi-view stereopsis algorithm has three main steps: matching, expansion and filtering.

3.1 Matching

The purpose of this step is to reconstruct and initial set of patches, features lying in non-empty cell are skipped for efficiency.

First we detect corner and blob features in each image using Harris and Difference-Of-Gaussian. After these features have been found in each image, they are matched across multiple pictures to reconstruct a sparse set of patches.

```
// 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++;
    }
}
```

Figure 2. Whole matching process in file "/program/base/pmvs/detectFeatures.cc"

3.2 Expansion

The purpose of this step is to obtain dense patches. New neighbors (patches) are added iteratively to existing patches until they cover the surface visible in the scene. See figure 3.

Input: Patches P from the feature matching step.
Output: Expanded set of reconstructed patches.

Use P to initialize, for each image, Q_f, Q_t, and its depth map.
While P is not empty
  Pick and remove a patch p from P;
  For each image $I \in T(p)$ and cell $C(i,j)$ that p projects onto
    For each cell $C(i',j')$ adjacent to $C(i,j)$ such that $Q_t(i',j')$ is empty and p is not *n-adjacent* to any patch in $Q_f(i',j')$
      Create a new p', copying $R(p'), T(p')$ and $n(p')$ from p;
      $c(p') \leftarrow$ Intersection of optical ray through center of $C(i',j')$ with plane of p;
      $n(p'), c(p') \leftarrow \text{argmax}\tilde{N}(p')$;
      $S(p') \leftarrow \{$Visible images of p' estimated by the current depth maps $\} \cup T(p')$;
      $T(p') \leftarrow \{J \in S(p') | N(p', R(p'), J) \geq \alpha_1\}$;
      If $|T(p') < \gamma|$, go back to *For*-loop;
      Add p' to P;
      Update Q_t, Q_f and depth maps for $S(p')$;
Return all the reconstructed patches stored in Q_f and Q_t.

Figure 3. Patch expansion algorithm of Furukawa and Ponce [2010]. The values used for α_0 and α_1 in all our experiments are 0.4 and 0.7 respectively.

3.3 Filtering

The purpose of this step is to remove erroneous matches and enforce visibility constraints. Two filtering steps are

```
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);
```

Figure 4. Whole expansion process in file "/program/base/pmvs/expand.cc"

applied to the reconstructed patches to further enforce visibility consistency and remove erroneous matches. The first filter focuses on removing patches that lie outside the real surface (fig. 8, left). The second filter focuses on outliers lying inside the actual surface. Finally a a weak form of regularization is enforced (fig. 8, right).

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

    filterOutside();
    setDepthMapsVGridsVPGridsAddPatchV(1);

    filterExact();
    setDepthMapsVGridsVPGridsAddPatchV(1);

    filterNeighbor(1);
    setDepthMapsVGridsVPGridsAddPatchV(1);

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

Figure 5. Whole filter process in file "/program/base/pmvs/filter.cc"

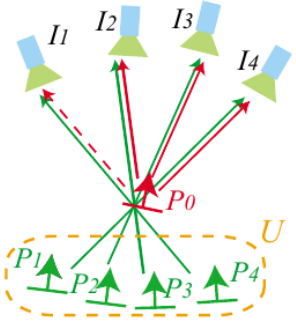

Figure 6. Filter outside representation take from Furukawa and Ponce [2010]

```
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, (void*)this);
    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++;
        }
    }

    if (denom == 0)
        denom = 1;
    ave /= denom;
    ave2 /= denom;
    ave2 = sqrt(max(0.0, ave2 - ave * ave));
```

Figure 7. Filter outside process in "/program/base/pmvs/filter.cc"

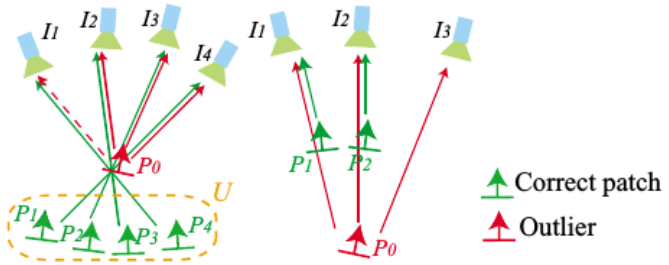

Figure 8. Feature matching algorithm of Furukawa and Ponce [2010]. The values used for α_0 and α_1 in all our experiments are 0.4 and 0.7 respectively.

4. EXPERIMENTS

4.1 How compile and execute PMVS program

The whole program pmvs-v2 can be downloaded from `https://www.di.ens.fr/pmvs/`. To compile the program pmvs-v2

4.2 Source code compilation

First you need to install some dependencies using the next command
sudo apt-get install 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 libg-fortran3 libmetis-edf-dev libparmetis-dev freeglut3-dev libgsl0-dev libblas-dev liblapack-dev liblapacke-dev libjpeg-dev

It also required to make some changes to the code, that it is explained in `https://github.com/mapillary/OpenSfM/blob/master/bin/export_pmvs.md`

```
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$
```

Figure 9. Compilation process

4.3 Input files

To perform the process we need some files as input, those are shown in the figure 10

Figure 10. Execution

4.4 Execution

To execute the program you should send the option file path and the option file to adjust the program settings. The process take several minutes(near to 15 minutes) ./pmvs2 "theoptionfilepath/" option.txt

After the process finished we found some files in the models folder, the ply file is the result of the reconstruction, this file could be opened using Meshlab or another tool.

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

Figure 11. Execution

5. RESULTS

The result of this process is an .ply file, this file could be opened in Meshlab or can be used to generate an object mesh, also generate a dense set of patches that can be used for image-based modeling.

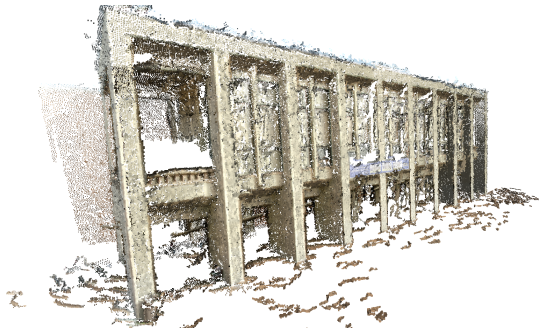

Figure 12. Result

REFERENCES

Furukawa, Y. and Ponce, J. (2010). Accurate, dense, and robust multiview stereopsis. *IEEE transactions on pattern analysis and machine intelligence*, 32(8), 1362–1376.

Lhuillier, M. and Quan, L. (2005). A quasi-dense approach to surface reconstruction from uncalibrated images. *IEEE transactions on pattern analysis and machine intelligence*, 27(3), 418–433.