Structure from Motion System in Modern Browser

BY XU PU

HOMEPAGE - *ptx.digital*
EMAIL - *ptx.pluto@gmail.com*

Abstract

This thesis attempts to design a 3D reconstruction system on the modern web platform, including a structure from motion pipleline and its integration/optimization in the browser. It take advantage of the advanced capabilities of the modern browser and its cross platform nature, make 3D reconstruction avaliable everywhere.

1 Introduction

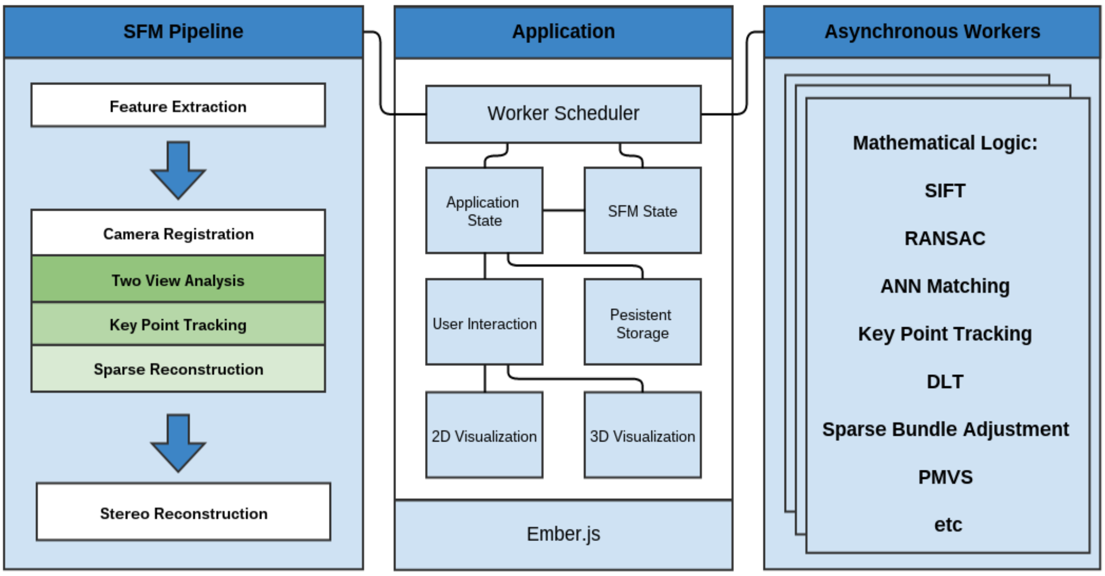
The title STRUCTURE FROM MOTION SYSTEM IN MODERN BROWSER implies two key aspects, *3D Reconstruction* and *Web Platform*, and both sides have great potential.

On one hand, structure from motion has always been a critical part of computer vision. In resent years, structure from motion has became increasingly relavent because of the rise of consumer drones and VR. Reconstruct 3D scene directly from photos taken by regular digital camera instead of specialized equipments can dramatically decreased the requirements for this important task, and open up the door to potentially unlimited resource for 3D reconstruction.

On the other hand, modern browser as a platform is gaining its popularity. It has became increasingly powerful over the years both in functionality and performance. Several novel features of modern browser has made implemeting an SFM system possible. First is *WebWorkers*, javascript is a single threaded language, *WebWorkers* enables parallel computing in clent-side javascript. Another one is *WebGL*, which is a interface for *OpenGL*. It made rendering of 3D models inside browser possible. Indexed DB is also a very important element, it implements a light weight object storage database, enables versatile persistent storage inside browser. Last but not the least, *ArrayBuffer* and *TypedArray* made handling large dataset using javascript practical.

This thesis attempts to design a 3D reconstruction system on the modern web platform, including a structure from motion pipleline and its integration/optimization in the browser. It take advantage of the advanced capabilities of modern browser and its cross platform nature, make 3D reconstruction avaliable everywhere.

2 Overview



. Framework of WebSFM

3 Geometry Context

3.1 Coordinate System

Points in the pipline have multiple cooridates according to different frames of reference.

- `Image Coordinate` – x
- `Camera Coordinate` – X_{cam}
- `World Coordinate` – X

Coorinate transitions:

- `Rotation Matrix(R) and Translation Vector(t)` – $X_{\text{cam}} = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix} \cdot X$
- `Calibration Matrix(K)` – $x = K \cdot [I, 0] \cdot X_{\text{cam}}$
- `Projection Matrix(P)` – $x = PX = K \cdot [R, t] \cdot X$

3.2 Camera Model

Camera model have multiple forms

- `Standard Linear Projective Camera Model (Undistroted)`
  - `Rotation Matrix (R)`
  - `Translation Vector (t)`
  - `Calibration Matrix (K)`
  - `Projection Matrix (P=K[R,t])`
- `Parameterized Full Camera (with distortion) -` 11 variables
  - `Euler Angles (r) -` 3 variables
  - `Translation Vector (t) -` 3 variables
  - `Principal Point (px, py) -` 2 variables
  - `Focal Length (f) -` 1 variable
  - `Radial Distortion (k1, k2) -` 2 variables

Parameterized full camera with ditortion is only used during camera registration. Distortion is ignored before camera registration; after registration cameras are undistorted.

4 Pipeline

The pipline is largely based on Bundler[3].

First generate SIFT features for each image (details in 4.1), then perform robust two view analysis on each pair of images (details in 4.2), matches produced by this process can then be used to track key points.

A *track* represents a point in the 3D space, it has one or no view on an image, we will find tracks by merging two view matches to a graph and traversing the match graph. Tracks that have more than one view on a single image is excluded as an obvious outlier.

Tracks are then used to initiate camera registration. We first choose a pair of camera to initiate using *five point algorithm*[4] to estimate first pair of camera parameter, then triangulate tracks they observe and run a two frame *sparse bundle adjustment* (detials in 4.4) to refine the initial estimation. After initiation, incremental recovery is performed (details in 4.3), and sparse reconstruction is obtained.

Currently WebSFM does not handle reconstruction beyoned sparse. PMVS/CMVS[6] can generate dense oriented point cloud, which can then be used to recover surface polygon using *Pission Surface Reconstruction*[2].

4.1 SIFT (Scale Invariant Feature Transform)

Lowe's SIFT[1] is our choice of feature extractor. SIFT features can be detected and matched regardless of its scale, orientation or lighting. Following is an overview of SIFT algorithm:

Algorithm SIFT

```
features := empty array
for each octave:
    scales space := progressive blur the base image of current octave
    dog space := subtract adjacent scales in scale space
    for each extrema in dog space:
        if principal curvature of extrema is too big or negative: break
        subpixel := interpolate extrema position in dog space, at (x,y,scale)
        orientations := domiant orientations of gradient histagram in sacle space
        for each orientation in orientations:
            descriptor := gradient distribution histagram in scale space
            features += (octave,subpixel,orientation,descriptor)
        end for
    end for
end for
```

Octave space start at -1 or 0, choose -1 if image is too small. In each octave, we will scan 3 DoG layers, which requires 5 layers in DoG space, and 6 layers in Guassian space. When bluring the image, larger sigma means larger kernel size, then directly leads to more time consumption, so blur is done progressively to keep sigma relatively small. We use default descriptor size stated in lowe's original paper, $4 \times 4 \times 8$ histagram and 128 dimensional Uint8 vector.

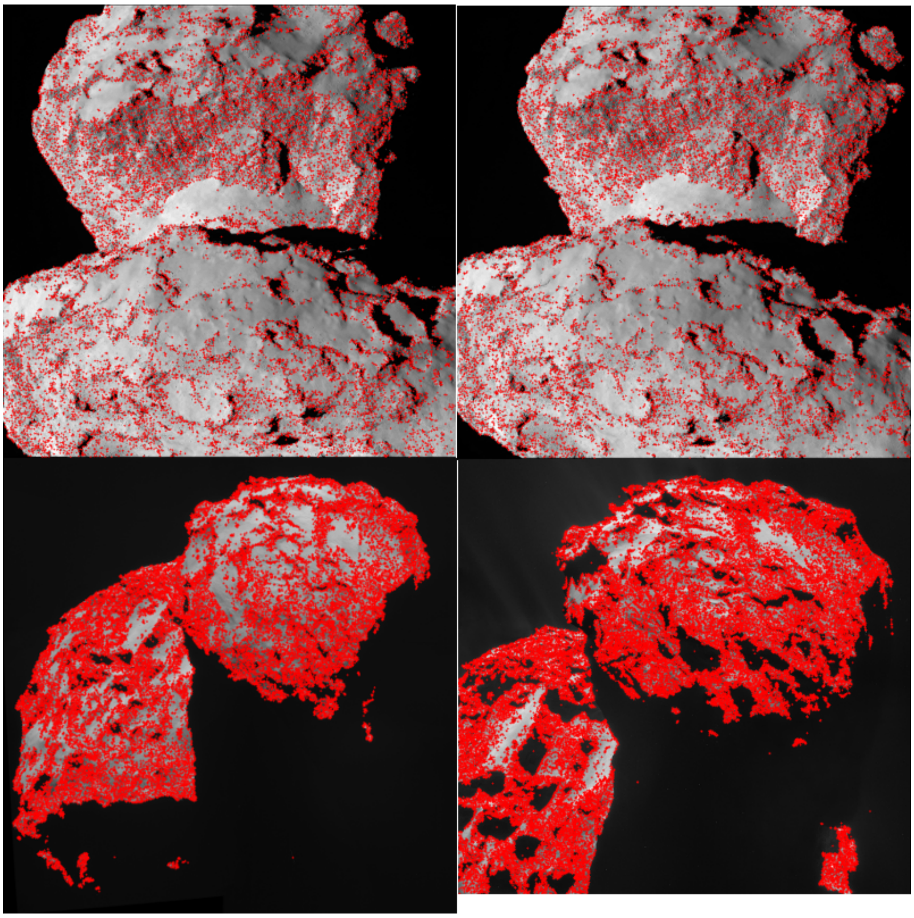


Figure. SIFT feature points

4.2 Robust Two-View Analysis

The objective of *Robust Two-View Analysis* is to find robust two-view features matches and accurate fundamental matrix estimation from SIFT features of two images.

We need to first obtain the *approximate nearest neighbor (ANN)*[5] matches of SIFT feature descriptors. For each set of descriptors, we construct a balanced kd-tree, which chooses splite plane based on the variance on each dimension in the subtree. Then use *approximate* nearest neighbor search rather than *strict* for performance concern. Instead using a threshold of the Euclidean distance of two descriptor to indicate a match, we use a priority queue to match each feature, then use a threshold on the ratio of distance between the best match and the second. Features also need to be the ANN of *each other* to be accepted.

Then we use two-view epipolar constrain to stablelize the matches. Each pair of point coorespondence must satisfy $x_1^T F x_2 = 0$, where F is the fundamental matrix. We exploit this attribute by combine RANSC and *normalized eight point algorithm* to filter outliers, and then estimate the *fundamental matrix* by SVD on the inliers then non-liear approximation by LMA.

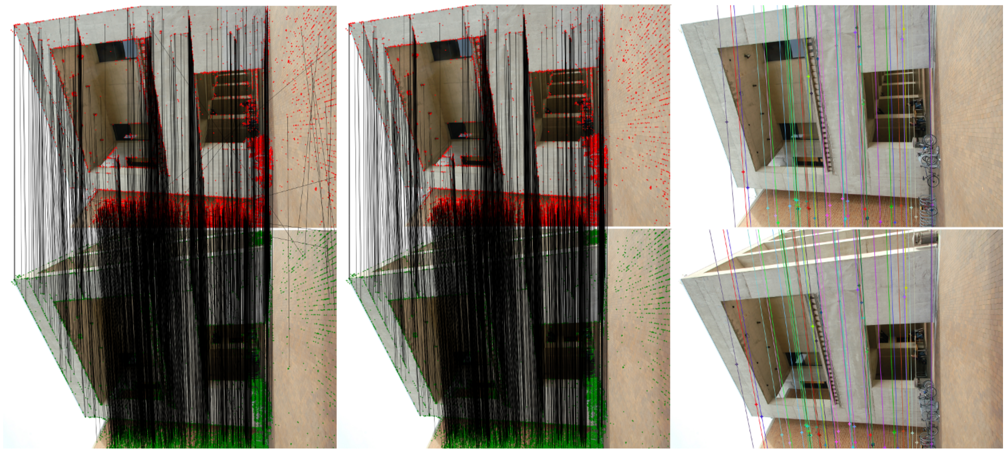


Figure 1. raw matches, robust matches and epipolar lines

4.3 Incremental Sparse Reconstruction

After the sparse structure is initiated, incremental recovery is performed. It will recover new cameras from recovered tracks, and triangulate more tracks, mean while use *SBA* and *Robust SBA* minimize projection error and keep the sparse structure robust, following is an overview of the algorithm:

Algorithm Incremental Sparse Reconstruction

`tracks` := input
`recovered cameras` := initial sturcture
`recovered tracks` := initial structure
`candidates` := un-recovered cameras view at least 20 `recovered tracks`
while `candidates` avaliable
    `selected` := candidate view the most `recovered tracks` and those above 75 percentile
    `new cameras` := recover camera models of `selected candidates` using DLT
    `recovered cameras` += `new cameras`
    sparse bundle adjustment on `new cameras` and `recovered tracks` they observe
    `new tracks` := triangulate tracks at maxium view angle within `recovered cameras`
    `recovered tracks` += `new tracks`
    global robust sparse adjustment
    `candidates` := un-recovered cameras view at least 20 `recovered tracks`
end while

Recover new camera parameters is accomplished by obtain its projection matrix through DLT and then decompose it into `(R, t, K)`, which can then be paramterized and refined.

When triangulate new points, big view angle is required. Only triangulate if the maximum view angle within recoverd cameras is larger then a threshold.

4.4 Sparse Bundle Adjustment (SBA)

The objective of *Sparse Bundle Adjustment (SBA)*[3] is to minimize projection error of a sparse reconstruction by perform non-linear approximation on camera paramters and 3D points. It is a modified implemention of *Levenberg-Marqurdt Algorithm* that take advantage of the sparse structure of muti-view geometry.

Sparse sturcture is exploited when calculating *Jacobian*, *damped Hessian* and δ_p. When solving

$$(H + d \cdot I) \cdot \delta_p = \epsilon$$

the damped Hessian can be splite into $\begin{pmatrix} A & B \\ B^T & V \end{pmatrix}$, where V is a block diagnal matrix, and equation can be solved using Shur's component.

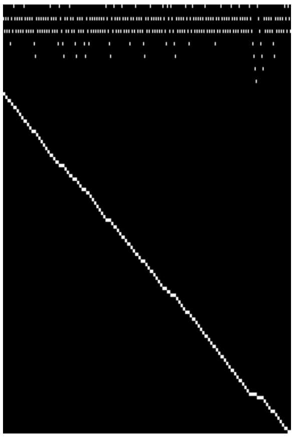


Figure 2. sparse jacobian matrix

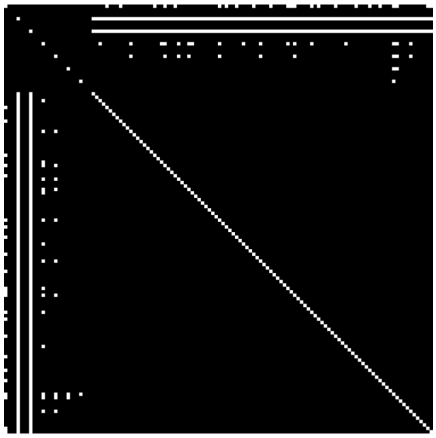


Figure 3. sparse hessian matrix

Robust SBA is SBA with outiler filter, it will refine the paramters and exclude outliers until no outliers can be found, following is an overview of the algorithm:

Algorithm Robust Sparse Bundle Adjustment

$\mathit{inliers}$:= input tracks
$\mathit{cameras}$:= input cameras
do
  sparse bundle adjustment on $\mathit{cameras}$ and $\mathit{inliers}$
  for each $\mathtt{camera}_{\mathtt{vi}}$ in $\mathit{cameras}$:
    d_{80} := 80 percentile of reprojection error on $\mathtt{camera}_{\mathtt{vi}}$
    $\mathtt{threshold}_{\mathtt{vi}}$:= clamp$(d_{80}, 4, 16)$
    $\mathtt{outliers}_{\mathtt{vi}}$:= tracks with reprojection error larger then $\mathtt{threshold}_{\mathtt{vi}}$
  end for
  $\mathit{outliers} := \bigcup \mathtt{outliers}_{\mathtt{vi}}$
  $\mathit{inliers} := \mathit{inliers} - \mathit{outliers}$
until $\mathit{ouliers}$ is empty

5 Application Framework

5.1 Parallelization

Structure from motion pipline is computational intense, but most tasks can run in parallel. Multi-thread computing can be accomplished using *WebWorkers*.

We encapsulate computation intense logic into tasks, and put all tasks into a single script file **worker.js**, which can have more than one instance inside browser. Then **application.js** can instantiated it and communicate with workers with aynchronous event interface. Tasks include:

- **SIFT Task** - Take ImageData and returns SIFT features
- **Matcher Task** - Take SIFT features of 2 images, returns robust two-view matches
- **Tracking Task** - Takes SIFT feature points and two-view matches, returns tracks.
- **Register Task** - Takes tracks, returns sparse point cloud and recovered camera parameters

5.2 Data Model

Large amount of data flow through the pipeline, it is critical to handle large dataset efficiently.

In WebSFM, a hierarchy of *ndarray*, *TypedArray*, and *ArrayBuffer* is used to handle large dataset, which uses *ArrayBuffer* as the underlying payload. For networking, it can be directly transfered through *XMLHttpRequest* as a native response type; for storage, it's supported in all implementations of *IndexedDB*; for multi-threading, it is a *TransferableObject* for *WebWorkers*. Data are stored in its minimum form through *TypedArray*, no JSON or text involved, which also means no parsing required.

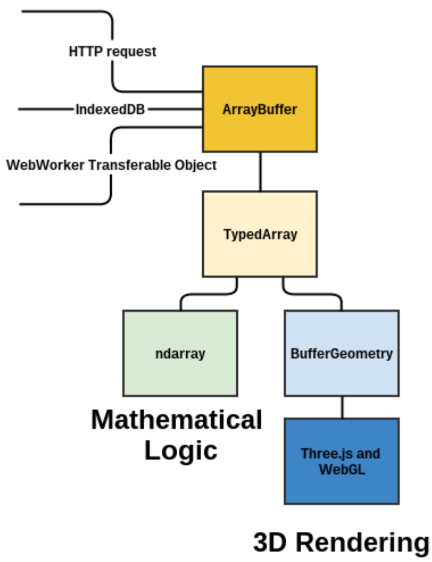


Figure 4. WebSFM data model

Following are the specifics of how data is represented as *ArrayBuffer*:

- SIFT features are stored in two seperate parts, *points* and *vectors*. *Points* is a $4 \times l$ ndarray with an underlying Float32 typed array as buffer, which stores *row*, *column*, *orientation*, *scale* of each feature point. *Vectors* is a $128 \times l$ ndarray with an underlying Uint8 typed array as buffer, which stores 128 dimensional descriptor of the cooresponding feature point in the *points* buffer. When matching features, only the *vectors* buffer is used, and when visualizing or tracking key points, only the *points* buffer is used.

- Surfels, both dense and sparse, is represented as two parts, *points* and *colors*. *Points* is a $3 \times l$ ndarray with an underlying FLOAT32 typed array as buffer, which stored the patch center coordinate. *Colors* is a $3 \times l$ ndarray with an underlying UINT8 typed array as buffer, which stores RGB color of the cooresponding patch in the *points* buffer.

Other data such as *camera paramters*, *feature matches* are insignificant in comparasion, so are directly stored as JSON objects.

To exam this approach, we take surfel model generated by PMVS which contains 450,000 patches. It costs 24.2MB in its original text form, 57.9MB in JSON form, which needs additional time for parsing. Using the data model described above, data is stored in its minimum form, *points* and *colors* have combined size of 6.8MB, and no parsing required.

5.3 Possible Improvement

WebGL is the door way to GPU acceleration inside browser, but haven't been fully exploited yet. The most time consuming tasks such as image convolution in SIFT and ANN matching can be GPU accelerated. This will be a future improvment.

5.4 Results

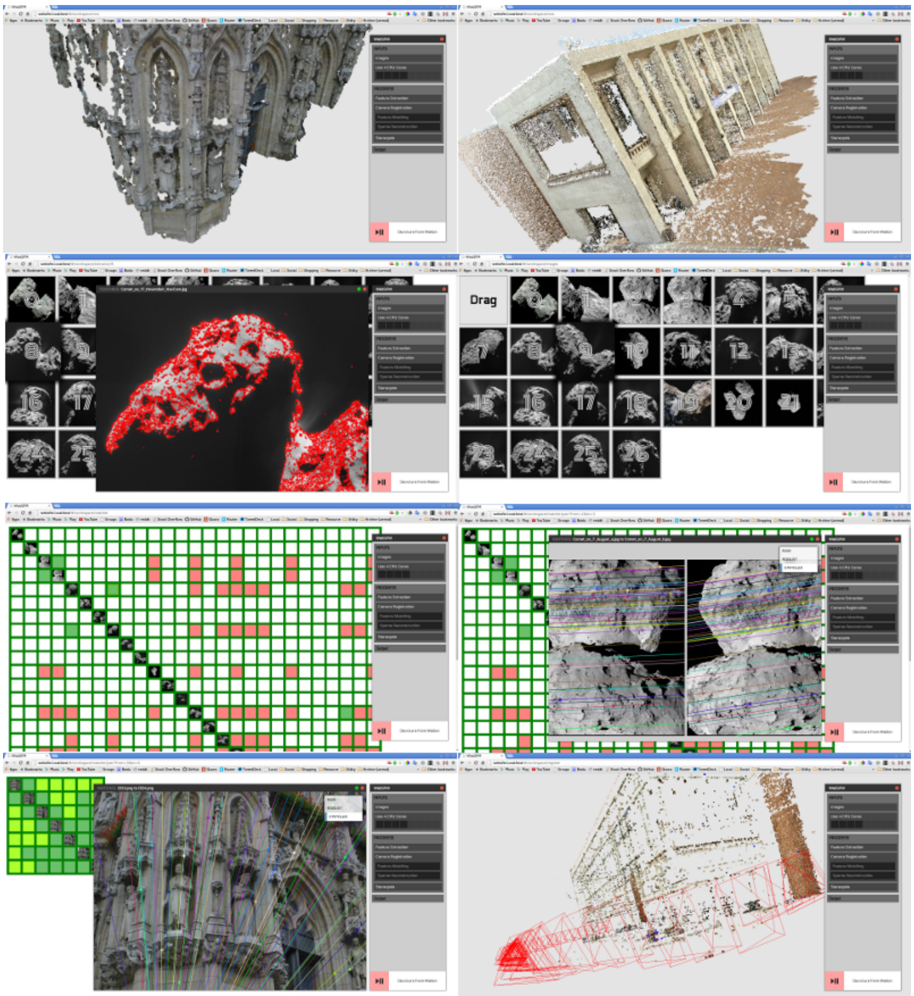


Figure 5. screen shots of WebSFM

Bibliography

[1] David G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, , 2004.

[2] M. Bolitho M. Kazhdan and H. Hoppe. Poisson surface reconstruction. *Symp. Geom. Proc.*, , 2006.

[3] Antonis A. Argyros Manolis I.A Lourakis. The design and implementation of a generic sparse bundle adjustment software package based on the levenberg-marquardt algorithm. , , 2004.

[4] D Nistér. An efficient solution to the five-point relative pose problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, , 2004.

[5] Nathan S. Netanyahu, Ruth Silverman, Angela Y. Wu Sunil Arya, David M. Mount. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. .

[6] Jean Ponce Yasutaka Furukawa. Accurate, dense, and robust multi-view stereopsis. .