Photogrammetric Stitching Report

Finn Petrie

2019-27-5

1 Introduction

The project aims to extend the findings in the Computer Science and Archeology collaboration paper " Photogrammetric Debitage Analysis: Measuring Maori Toolmaking Evidence". The original paper found a significant statistical difference in the calculation of a stone's maximum width, and dimension by the photogrammetric measurement method, versus human measurement, suggesting the stones were not being properly reconstructed. The goal of this project was to find a way to reduce this statistical difference between the reconstructions and the ground-truth, human measurement. The approach was to stitch the reconstructions of the opposing surfaces - dorsal and ventral - together, and then test for maximum dimension, and width. Initially, we shall outline the methods attempted, as well as considered methods. Furthermore, we will discuss problems faced, and where to go next. This document shall also describe how to use the pipeline that was created for this project.

2 Alignment methods

2.1 Alignment by shape measurement

To stitch opposing surfaces fundamentally requires a linear transform that results in an alignment on a single plane in which the two surface's openings meet. This is inherently a shape measurement problem. Our method chose to model the problem by aligning their silhouttes. A shape measurement metric was developed by taking samples of points on the corresponding silhouettes at regular angles, and then summing the distance between these points over a sampled range of 360°s. In application, we concerned ourselves with trying to find an alignment between the $2D$ convex hulls of the surfaces' projections onto their two larger principle components, which gives us an easy mathematical approximation of silhouette.
The current method takes a ventral and dorsal surface, appropriately setup in R^3, computes their projections onto their two largest principle components, then computes the projections' convex hulls. As above, a suitable alignment would correspond to a $2D$ rotation such that the area between the two hulls

is minimised. We can setup such a problem in the following way: given two convex hulls C_d and $C_v \in R^2$, parameterised as corresponding curves in polar coordinates, $\gamma_{C_d}(\theta)$, $\gamma_{C_v}(\theta)$ where $\theta \in [0, 2\pi]$, we want to find a rotation R such that $\int_0^{2\pi} \sqrt{\gamma_{C_d}(\theta)^2 - \gamma_{Cv}(\theta)^2} d\theta$ is minimal. The following problem can be solved in our discrete domain by representing C_d, C_v in polar coordinates, fitting their point clouds with interpolating functions $f_{C_d}(t)$, $f_{C_v}(t)$ and then finding R such that $\sum_{n=0}^{360} \sqrt{f_{C_d}(\frac{n\times\pi}{180})^2 - f_{C_v}(\frac{n\times\pi}{180})^2}$ is minimal. We use the following algorithm to solve this problem:

$minError = \infty$
for *theta in range 0, 360* **do**
  $e = 0$
  $i = 0$
  while $i \leq 360$ **do**
    $e = e + \sqrt{f_{C_d}(i)^2 - f_{C_v}(i)^2}$
  end
  if $e < minError$ **then**
    $minError = e$;
    $minTheta = theta$
  end
  Rotate C_d by 1°
end
return C_d *rotated by minTheta*

Although the above algorithm has a nested loop structure, and therefore appears to be $O(n^2)$, we note that the time spent in these loops is independent of the input size. $\implies$ the running time is $O(c)$, where c is a constant, and is thus practical for any application no matter the size of the input's point sets. Still, the constant is large, here $c = 360^2$, and may grow depending on how many samples we wish to take, and how many discrete rotations we wish to make. Noting that we are only concerned with rotations within a certain quadrant, optimisation may therefore be done by heuristically guessing a decent rotation that moves us into such a quadrant, and thus approximately minimises the error function. Using binary search from their on the shape distance metric resulting from R to rotate convergingly toward a global minimum. This would reduce c by a large factor.

2.1.1 Interpolating Function

Intuitively, the accuracy of R is dependent on both, the number of samples taken over the interpolating function, and the accuracy of the interpolating function itself. Due to time constraints, we chose to use linear interpolation, but a better rotation R could be found using Cubic B-Splines, or Bezier Curves. Scipy's wrapper of Fortran's FITPACK was initially chosen for its cubic B-Splines, before I decided to write my own simple linear interpolation to avoid the library's confusing structure. Further work in Cubic Spline interpolation would likely result in a better alignment.

2.2 Alignment via ICP

Iterative Closest Point is also an option for finding a global alignment. ICP is prone to finding local minima, so to reduce the complexity of the optimisation domain, we chose to match silhouettes, and use a stochastic ICP to reduce the likelihood of the algorithm finding a local minima. The method is similar to the shape measurement method, where we setup a dorsal and ventral surface in R^3, making them lie in the same plane. We again compute their projections onto their two larger principle components, and then compute convex hulls C_d, C_v as a model of silhouette. ICP is then run, resulting in an alignment between one convex hull and another. The rotation and translation matrix resulting from the match is then applied to the dynamic convex hull's surface. This gives an approximate alignment.

2.3 Combined Method

Using both the shape measurement optimisation, and then ICP may result in an alignment that is closer to the global minimum. This is done by reducing the shape error metric between the silhouettes as above, and then using ICP to find what could possibly be a better alignment, noting that ICP performs best when the pointclouds to match are setup well. Note however, that a better alignment is not guaranteed, and using ICP can result in worse alignments. It is best to check the result of this method against the original shape metric to see if the alignment has improved.

2.4 Further Alignment

Given the time constraint, a statistical analysis between the maximum width and dimension of the aligned surfaces versus ground truth has not been carried out. In the case that this alignment does not render a correct reconstruction, we could choose to remove noise and use physical constraints to find better alignments. We may use the fact that the surfaces were reconstructed on a plane to do further alignment. Using these planes as physical constraints (estimated as either a RANSAC of the original photogrammetric point cloud, or the span of the two larger principle components of the surface) and sandwiching the surfaces between them such that the highest point of a surface S is tangent to their opposing surface S''s plane sets the two planes up to be rotated until they have stable support on these planes. Using the tangent points as a pivot would be the first step toward this. The following algorithm sandwiches the surfaces between their planes, a stable support alignment was not developed, but likely requires a rotation using the plane-projected points as a pivot for the surface.

Data: Surface, SurfacePlane, OpposingPlane
Compute principle components of Surface
Project the minimum principle component onto the opposing plane
translation = mininimum principle component - projected point
for *point in Surface* **do**
| point = point + translation
end
for *point in Surface Plane* **do**
| point = point + translation
end

2.5 Considered Alignment Methods

Two other possible approaches to alignment were considered. A simplified version of the Spherical Demons algorithm, and Brendan McCane's curve alignment algorithm. These approaches are still valid, and their implementation may lead to interesting results.

2.6 Reconstruction

Once the ventral and dorsal are correctly, or approximately stitched, then we would be able to use SfM software, such as Colmap, Photoscan, or CMVS-PMVS to create a dense reconstruction and mesh of the new object. To add this functionality, the current alignment pipeline would need to be modified to accomodate cameras of the original reconstruction. This could be done by making a .bundle parser to initialise cameras, and then forwarding each camera through the pipeline, transforming them with each transform their surfaces' undergo. Then importing the resulting stitched point cloud and cameras back into the original software and using the software's dense reconstruction and meshing methods. Further work in creating the masks of the surface's convex hulls may need to be done here so that the cameras do not reconstruct elements that have been deleted from our scene, such as other stones.

3 Using the Pipeline

Currently there are two segments of the pipeline. The alignment method discussed in this report is encapsulated into a Python program, where a "Stitching Pipeline" object simply sets up two imported .ply files for matching, and then calls a composition Matching object to align the surfaces. The other segment is a C++ library that supports some rudimental pointcloud functionality, the optional ICP alignment of the pipeline, and the sandwich-plane alignment. This report shall discuss the C++ library shortly as most of its header files are well documented.

3.1 Stitching Pipeline, Python

The functionality of the Python segment is fairly automatic. It simply requires the user to instantiate a pipeline object with the paths to the two surfaces, and then call *.setup*() followed by *.run*(). However, depending on the original orientation of the two surfaces, the program may need to be supplied with an index being an axis to remove, ($x = 0$, $y = 1$, $z = 2$) so as to project the surfaces to their larger principle components. This only needs to be added to the $self.flattenSurface(surface, axis)$ calls in *.setup*() The stages of the pipeline are outlined below.

Input: = Ply Surfaces
(1) compute covariance matrix of both clouds, use its eigen-vector matrix as their bases
(2) compute change of basis matrix to the standard basis, i, j, k
(3) remove their translations
(4) rotate the dynamic points so that they oppose the static point cloud
(5) project both clouds onto their larger principle components (the axes that the two larger eigen-vectors of the covariance matrix were transformed to)
(6) compute their convex hulls
(7) express the convex hulls in polar coordinates
(8) interpolate these convex hulls using linear interpolation
(9) run the alignment algorithm as outlined in 2.1
Output = the aligned dynamic surface.

3.1.1 *.setup*()

This function orientates the surfaces correctly. It computes a change of basis matrix for each pointcloud, transforming their principle components such that they correspond to the standard basis, i, j, k. From here it computes a quaternion rotation so that surface specified as *dynamic* opposes the *static* surface. Since the surfaces now lie in a plane due to the change of basis transform, we just compute a rotation of π about the largest principle component. We then project the newly aligned point clouds onto the plane spanned by their larger principle components, and then compute their convex hulls.

3.1.2 *.run*()

Here we instantiate a Matching object, this object handles the interpolation function, as well as the shape measuring algorithm outlined in 2.1. Step by step, this object re-represents the convex hulls in polar coordinates, computes a linear interpolation parameterised by some t in range $[0, 2\pi]$, and then runs through the alignment algorithm, computing the distance between each cloud's interpolation at a given angle, θ, and summing these distances up as our error

metric. It iterates over this for 360°. Once complete, it rotates our cloud to the θ which minimised our shape distance error metric.

3.2 Stitching Pipeline, C++

Given two approriately setup surfaces and their convex hulls, the C++ library can be used to compute an ICP match, as well as the sandwiching of the two surface's between their opposing surface's planes. In the case of ICP, four .ply files must be read into the program using the $PlyFile(pathToFile)$ object. A ventral surface and convex hull, and a dorsal surface and convex hull. From here the user needs to instantiate a $IterativeClosestPoint$ object, taking the four .ply objects as arguments to its constructor. Then a call to $IterativeClosestPoint :: computeStochastic(int, bool)$ where the integer is the number of indices to randomly sample from, and the boolean indicates whether the algorithm is to apply its rotation matrix to the surfaces. The method $translatePlanes(Plane1, Plane2)$ handles the sandwiching.

4 Conclusion

Currently our shape metric alignment results in an approximately correct alignment, the correctness being dependent on how many samples we wish to take of our parameterised curves and how many degrees we wish to rotate per iteration. Additionally, ICP can be used to possibly find a better alignment, but it is often uncertain as to whether the alignment given will be closer or further to the optimal. A statistical analysis of the alignment method's maximum dimension and width compared to those of ground truth needs to be performed. Sandwiching the surfaces between planes is a direction that may allow exploitation of their physical constraints for an alignment based on both surface's stable support on their opposing surface's planes. Ultimately, development of a better interpolation function to optimise the alignment further, as well as using binary search on the interpolation function to further converge to a global minimum would be an important direction to head in, as well as figuring out how to import the alignment into SfM software for further reconstruction.