Improved real-time photogrammetric stitching

Jason de Villiers[a,b] and Jaco Cronje[a]

[a]Council for Scientific and Industrial Research, Pretoria, South Africa;
[b]University of Cape Town, Cape Town, South Africa

ABSTRACT

This work extends earlier work on the real-time photogrammetric stitching of staring arrays of high resolution videos on commercial off the shelf hardware. The blending is both further optimised for Graphics Processor Unit (GPU) implementation and extended from one to two dimensions to allow for multiple layers or arbitrary arrangements of cameras. The incorporation of stabilisation inputs allows the stitching algorithm to provide space stabilised panoramas. The final contribution is to decrease the sensitivity to depth of the stitching procedure, especially for wide aperture baselines. Finally timing tests and some resultant stitched panoramas are presented and discussed.

Keywords: Depth from Defocus, Depth from Focus, Scintillation, Heat Shimmer, Passive Ranging

1. INTRODUCTION

Photogrammetric stitching is a means to generate panoramas from multiple images without requiring computationally expensive image processing in the form of searching for and then matching image features and creating a homography between the images. Rather, both the system and the cameras are pre-calibrated and use is made of these parameters to efficiently perform the stitching. The authors previously presented an overview of this procedure in 2009[1]

1.1 Blending

In this work blending refers to the determination of the appropriate weighted combination of pixels from each camera that can see each pixel in the stitch. This blending needs to work when zero, one or more cameras can see the point under consideration.

In operation, each camera will have a slightly different white balance due to the different exposure, iris and gain settings that may be applied to each camera according to their scene content. Thus, objects in the overlapping regions may appear slightly different in the two cameras. The correction of these colour imbalances is a research topic in its own right. Xu and Mulligan[2] provide an overview of the available methods.

1.2 Contribution

This work extends prior work on photogrammetric stitching. The blending is both extended to multiple dimensions and optimised for speed. The stabilisation of the panorama using externally determined rotation angles is presented. The stitching geometry is extended to decrease the sensitivity to depth in the panorama. These enhancements are all tested on a variety of hardware to show that real time operation is retained.

Further author information: (Send correspondence to J.d.V.)
J.d.V.: E-mail: jdvilliers@csir.co.za

2. OVERVIEW OF ORIGINAL STITCHING ALGORITHM

This section briefly describes the process of generating the photogrammetric stitch. This is to allow the improvements presented in this paper to be discussed in context. For mathematical details of the stitching algorithm refer to the original paper.[1]

For each pixel in the stitch image:

1. From the coordinate of the current pixel, determine the corresponding azimuth and elevation angles.
2. Create a vector from these angles.
3. Extend the vector to where it intersects the stitching sphere.
4. Calculate the point in space where the vector-sphere intersection occurs.
5. For each camera:
   a. Ray trace the point back to the camera's image plane.
   b. Scale by pixel size to get into image space.
   c. Correct for distortion and determine which pixel of the camera corresponds to this pixel in the stitch.
   d. Determine weighting of this pixel according to how close it is to the horizontal center of the cameras image.
   e. Set the weight to zero if:
      i the weight is less than zero.
      ii the calculated pixel positions falls outside the camera's image (four checks).
      iii the stitching point is behind the camera not in front of it.
6. Calculate the weighted sum of the pixels from each camera for this pixel in the stitch.

3. IMPROVED BLENDING

This section describes the improvements made to the blending algorithm. The improvements not only allow for camera's images to be blended horizontally and vertically but substantially improves the stitching processing rate (see Table 1 in Section 6).

3.1 Blending Algorithm

Blending is concerned with steps 5d, 5e and 6 of the stitching procedure as defined in Section 2. The original horizontal weighting used is an upwardly convex parabolic given in Eq. 1.

$$W_{h,i} = P_h^D \times (Res_h - P_h^D) \tag{1}$$

where:

$W_{h,i}$ = the horizontal weighting factor for camera i,

(P_h^D, P_v^D) = the coordinates of the distorted pixel position corresponding to the stitch point, and

(Res_h, Rev_v) = the resolution in pixels of the camera.

This weighting function peaks at the center of the image, descends quadratically down to zero at the image edges, and is negative if the pixel point is off the image to the left or right. This means that instead of doing two checks to see if the distorted point lies on the image horizontally and setting the weight to zero, one can merely take the maximum of the weight and zero, thus improving processing efficiency. If this same logic is applied to the vertical ordinate one can then be able to smoothly blend vertically displaced images (which would be required

for creating a hemispherical Field of View (FOV)) and improve the efficiency of checking to see if the distorted point lies on the image vertically. This new weighting function is given in Eq. 2.

$$W_{c,i} = max\left(0.0, P_h^D \times (Res_h - P_h^D)\right) \times max\left(0.0, P_v^D \times (Res_v - P_v^D)\right) \tag{2}$$

where:

$W_{c,i}$ = the combined weighting factor for camera i.

Two further improvements were made to optimise the blending algorithm. The first is a change in how the final blended value is calculated. Originally the weight and colour contribution for each camera was stored and then accessed at the end of the kernel program to calculate the weighted sum pixel value as per Eq. 3.

$$Colour_{out} = \frac{\sum_{i=1}^{n} (Colour_i \times W_{c,i})}{\delta_W + \sum_{i=1}^{n} W_{c,i}} \tag{3}$$

where:

δ_W = a small positive value (e.g. 10^{-4}) added to the denominator to avoid division by zero,
$Colour_i$ = the colour that camera i is contributing to the stitch, and
$Colour_{out}$ = the final colour for the current pixel of the stitch being calculated.

It is more efficient to operate in a 'fire and forget' mode of keeping a running total of the numerator and denominator of Eq. 3 and then performing a single division to calculate the output colour. Eq. 4 through Eq. 8 explain this mathematically, where Eq. 6 and Eq. 7 are performed once for each camera.

$$NSum_0 = 0.0 \tag{4}$$
$$DSum_0 = \delta_W \tag{5}$$
$$Nsum_i = NSum_{i-1} + Colour_i \times W_{c,i} \tag{6}$$
$$DSum_i = DSum_{i-1} + W_{c,i} \tag{7}$$
$$Colour_{out} = \frac{NSum_n}{DSum_n} \tag{8}$$

where:

$NSum_j$ = the current numerator sum after j cameras have been considered,
$DSum_j$ = the current denominator sum after j cameras have been considered, and
n = the number of cameras in the stitch.

The final improvement is to only process a camera if it is likely to be able to see the pixel. This is perhaps counter intuitive as branching is normally sub optimal in Graphics Processors Unit (GPU) kernels. This however is analogous to Step 5.e.i of Section 2. This is achieved by taking the dot product of the stitch unit vector for the current pixel with the optical axis (i.e. the camera's X axis in this formulation) and comparing this to the cosine of slightly more than half the diagonal FOV of the camera. This both eliminates the need to check if the point is behind the camera and decreases the mathematical computations and memory look ups required for each pixel. Eq.9 and Eq. 10 shows how the angular threshold and the cosine of angle between camera i and the

stitch vector respectively are calculated. Camera i is processed if $\theta_i \geqslant \theta_{th}$.

$$\theta_{th} = cos\big((0.5 + \delta_\theta) \times FOV_D\big) \tag{9}$$

$$\theta_i = \left(R_{C_i R} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \right) \bullet \vec{U}_{RPR} \tag{10}$$

where:

θ_{th} = the angular threshold to determine if a camera should be processed for this pixel,

δ_θ = small safety margin added to the diagonal FOV,

FOV_D = the diagonal FOV of the cameras,

θ_i = cosine of the angle between the stitch vector and camera i,

$R_{C_i R}$ = Euler rotation matrix of camera i relative to the stitch axis system, and

$\vec{U}_{RPR}$ = the unit vector to the current stitch pixel.

3.2 Blending Results

Figure 1 and Table 1 provide the results of the improved blending. The timing results are marked with improvements almost trebling the stitching times. Visually the results are subtle. In the extreme outer FOV of one of the cameras typically the largest residual errors after distortion correction are evident, resulting in small error in stitching. By biasing the blending weights towards the camera which has the point closest to the center of its FOV this effect is minimised. An example of this is evident in the marked sections of Figures 1(a) and 1(b). In the former there is slight blurring evident due to the errors induced from the upper FOV of the bottom camera. This blurring is much improved in Figure 1(b).

4. STABILISATION INPUT

Registration and stabilisation have many uses including surveillance, targeting, tracking, mosaicing and super-resolution. Stabilisation of narrow FOV images is an understood problem. Most often this entails the in-plane rotation, translation and perspective shearing of an image such that the image contents remain in a constant screen position despite any movement experienced by the camera. Examples of such trackers include the seminal Lukas-Kanade registration algorithm[3] and its assorted variants (e.g.[4]).

These algorithms are all implicit linear simplifications of the general case. For instance if an omnidirectional camera is tilted downwards the horizon will move up in the 'forward' part of the image, but also downwards in the rear direction and appear to rotate at the left and right. In fact, the horizon will form a sinusoid in the panorama - as evidenced by Figure 2(a). The shifting and rotating used is a local linearisation of this sinusoid for the panorama excerpt that is visible to the narrow FOV camera.

4.1 Stabilisation Mathematics

In order to stabilise the image content in the panorama it must be bent in an inverse sinusoid. Assuming that the angular motion of the camera array is known (either via an external system or image processing algorithms outside the scope of this work), then only a simple modification is required. The unit vector calculated from the current stitch pixel position in Step 2 of Section 2, is simply rotated by the inverse Euler rotation matrix as shown by Eq. 11.

$$\vec{U}'_{rp_i r} = R_{ar}^T \vec{U}_{rp_i r} \tag{11}$$

where:

$\vec{U}_{rp_i r}$ = the unit vector to the current pixel in the reference axes,

R_{ar} = rotation matrix express the rotation of the camera array relative to the reference axes, and

$\vec{U}'_{rp_i r}$ = the corrected vector for further use in the stitching process.

(a) Original Blending

(b) Improved Blending

Figure 1. Vertical Stitch of 2 Cameras

4.2 Stabilisation Results

Figure 2 and Table 1 show the visual and timing results for the stabilisation algorithm enhancement. Figure 2(b) clearly shows how the curvature of the horizon (evident in Figure 2(a)) induced by the camera array being tilted 20° downward, has been removed.

(a) Non-stabilised panorama

(b) Stabilised Panorama

Figure 2. 300° FOV panorama with camera pitched 20° downward.

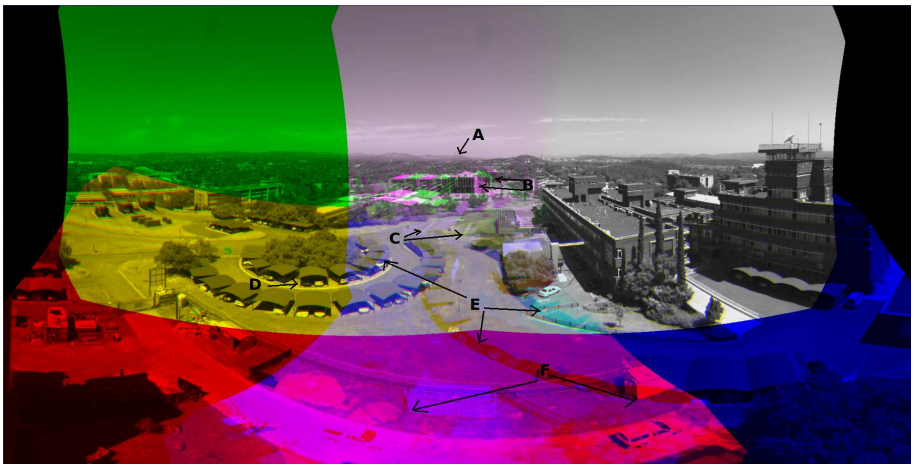
5. MULTI GEOMETRY STITCHING

The current stitching algorithm[1] makes the flat scene assumption and stitches on a spherical surface. This stitching algorithm is efficient but is only able to perform the stitching at a single distance. If an object in the scene is not at the distance at which the stitch was created, then it will not be rendered correctly. It will either be blurred if it is only slightly off the stitch distance, or it will be doubled if it is far from the stitch distance. The severity of this effect is directly proportional to the displacement between the cameras.

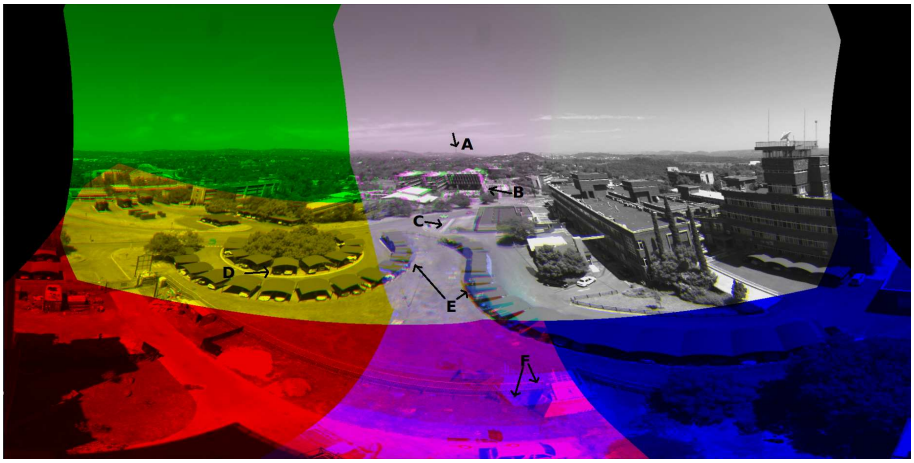
This effect is illustrated in Figure 3(a) which is a spherical stitch performed with a stitching radius of 1000m. All the images in Figure 3 are stitches generated from four cameras with a 5mm lens giving an approximate horizontal FOV of 82° on the 2/3" CMOS sensor of the Allied Vision GE1660 cameras used. The stitch extends from $-90°$ to 90° horizontally and from 60° below horizontal to 30° above horizontal vertically. The top two cameras have an approximately horizontal attitude and are looking at approximately $\pm 30°$ in yaw. The bottom two cameras have similar yaw values but are pitched downwards at approximately 45°. The left pair and right pair of cameras are approximately 16m apart. The top left camera's image has been coloured yellow, the bottom left coloured red, bottom right blue, and top right left in the original grayscale.

In Figure 3(a) the horizon (marked A) is stitched correctly. The building marked B (approximately 200m distant) is noticeably doubled. This effect gets markedly worse for the pathway (marked C), the central car ports (Marked E) and the extremely close container marked F. The left carports (Marked D) are not effected due to the negligible distance between the bottom left and top left cameras, this is true for all objects only visible to either left or right cameras but not both.

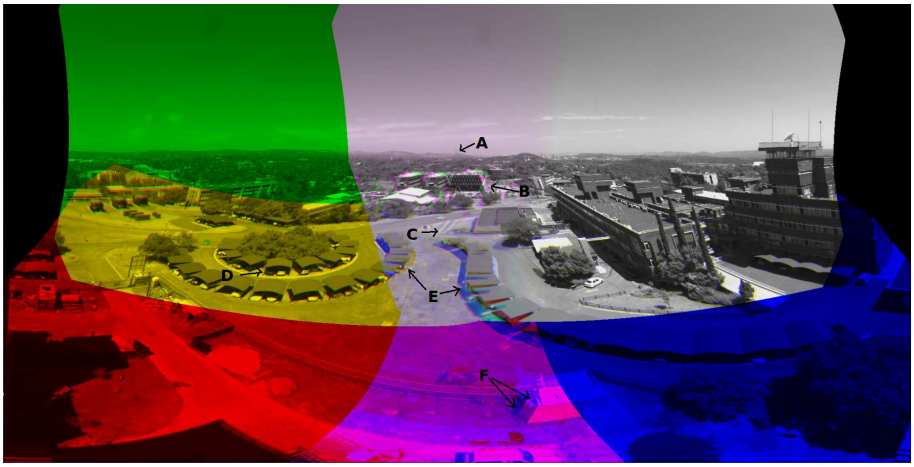
To alleviate this problem, the surface on which the stitch is calculated was modified so that the stitch is performed at different distances for different parts of the image as detailed in Section 5.1.



(a) Spherical Stitch



(b) Multi-Geometry Stitch, Plane 21m Below Cameras



(c) Multi-Geometry Stitch, Plane 19m Below Cameras

Figure 3. Spherical Stitches at Different Distances

5.1 Multi Geometry Algorithm

The current stitching algorithm uses only a radius to perform the stitching. This is the radius of a sphere whose center is at the origin of the axis system on which the camera's relative positions are defined. The stitch FOV and angular step sizes are used to generate discrete points on this sphere. The basic stitch algorithm is described in Section 2.

The new stitching algorithm adds a flat plane to this geometry. When the vector is created in Step 3 it is also extended until it intersects the flat plane. Then the point closest to the camera (the vector-plane intersection or vector-sphere intersection) is used from Step 4 onwards.

A plane can be defined in several manners. The method chosen here is the outward pointing unit normal vector of the plane, and the distance of the closest point of the plane from the origin. Using these definitions, Eq. 12 provides the distance to the intersection with the pixel's corresponding vector and the plane. Equation 13 provides the exact point of the intersection to be used in further steps in the stitching if necessary.

$$D_{rp_ir} = \frac{D_{pr}}{\vec{U}_{ra_ir} \bullet \vec{U}_{rpr}} \tag{12}$$

$$\vec{T}_{rp_ir} = D_{rp_ir} \times \vec{U}_{ra_ir} \tag{13}$$

where:

$\vec{U}_{ra_ir}$ = the unit vector created from the ith pair of azimuth and elevation angles,

D_{pr} = closest distance between the plane and origin,

$\vec{U}_{rpr}$ = the unit vector defining the plane,

D_{rp_ir} = distance along the ith angular vector to the plane, and

$\vec{T}_{rp_ir}$ = the point at which the ith angular vector intersects the plane.

5.2 Multi Geometry Results

Figures 3(b) and 3(c) show the stitching performed by the new algorithm. In this figure spherical stitching is performed at a distance 1000m, but a plane is defined with an outward facing normal of $[0, 0, -1]^T$ (i.e. a horizontal plane below the camera array) and a closest distance of either 19m or 21m. This plane approximates the ground surface, so all objects that are approximately at ground height are stitched correctly too. The definition of this plane can be updated in real-time. For instance it can be updated with inertial information so that even if the camera array is experiencing angular disturbances, potential objects of interest on the ground will still be stitched correctly.

In Figure 3(b) the plane is 21m below the camera.The horizon is still stitched correctly. The base of building B is stitched correctly and so is pathway C. Central carports E are much better stitched and close container F is much less displaced. In Figure 3(c) the plane is 19 m below the cameras. Building B is now correctly stitched half way up. Pathway C is slightly blurred. The central carports E and close in container F are both much better rendered.

This extreme sensitivity to depth is due to the large 16m displacement between the cameras horizontally, and the evident non planarity of the local terrain in the scene.

6. TIMING RESULTS

The timing results presented in Table 1 are representative of generating a 360° omnidirectional stitch with a 70° vertical field of view. The resolution of 4096×796 pixels in order to maintain the aspect ratio. 4096 is the maximum texture size common to all the GPUs used. The input data consisted of four 1.3MP 1380×1024 Allied Vision Prosilica GC1380 cameras and one 2.0MP 1600×1200 Prosilica GX1660 camera, all fitted with 5.0mm Navitar lenses. The algorithms were implemented in OpenGL Shading Language[5] using the Framework for Live Image Transformation.[6]

Table 1. Stitching Frame Rates

	Stitching Enhancement			Frame Rate Achieved (FPS)		
	Blending	Stabilisation	Multi Geom	NVidia ION2	Quadro 2000M	GTX460M
1				14	128	158
2			X	13	120	155
3		X		13	122	152
4		X	X	13	120	150
5	X			35	195	275
6	X		X	33	180	255
7	X	X		35	185	260
8	X	X	X	33	175	235

The optimisations of the blending algorithm yielded a marked improvement in processing rates on all the GPUs that were tested. Improvements of between 50% and 150% were observed. Adding the stabilisation and multi-geometry stitching had slight performance penalties, with the multi-view geometry having a slightly larger effect. Even with both of these enhancements, when combined with the new blending, allowed even the light weight 16-core NVidia Ion2 GPU being able to produce the high resolution stitch in real-time.

7. CONCLUSIONS

This work presented three improvements to the previously published stitching algorithm: an optimised two dimensional blending algorithm, stabilisation mathematics and less depth-sensitive multi-geometry stitching algorithm.

Together these algorithms allow even light-weight GPUs to generate high resolution stitches from moving camera arrays with decreased errors due to varying depths of the scene content.

REFERENCES

[1] de Villiers, J. P., "Real-time stitching of high resolution video on COTS hardware," in [*Proceedings of the 2009 International Symposium on Optomechatronic Technologies*], *ISOT2009* **9**, 46–51 (2009).

[2] Xu, W. and Mulligan, J., "Performance evaluation of color correction approaches for automatic multi-view image and video stitching," in [*Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*], 263 –270 (June 2010).

[3] Lucas, B. D., Kanade, T., et al., "An iterative image registration technique with an application to stereo vision," in [*Proceedings of the 7th international joint conference on Artificial intelligence*], (1981).

[4] Duvenhage, B., Delport, J. P., and de Villiers, J. P., "Implementation of the Lucas-Kanade image registration algorithm on a GPU for 3D computational platform stabilisation," in [*AFRIGRAPH '10*], 83–90, ACM, New York, NY, USA (2010).

[5] "OpenGL shader language." http://www.opengl.org/documentation/glsl/.

[6] Delport, J. P., "Framework for live image transformation," (2010). http://code.google.com/p/flitr/.