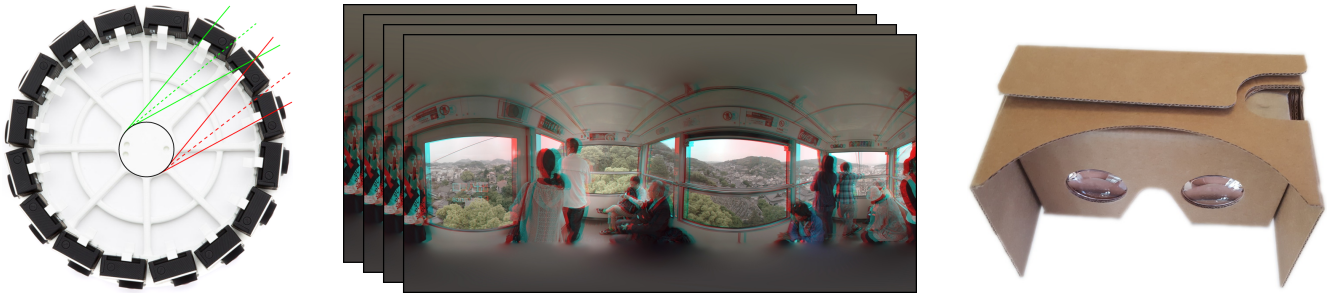
Jump: Virtual Reality Video

Robert Anderson David Gallup Jonathan T. Barron Janne Kontkanen
Noah Snavely Carlos Hernández Sameer Agarwal Steven M. Seitz

Google Inc.



(a) **(b)** **(c)**

Figure 1: *The Jump system produces omnidirectional stereo (ODS) video. (a) The multi-camera rig with the ODS viewing circle overlaid and three rays for the left (green) and right (red) stitches. Solid rays pass through a camera and can be sampled directly. Dashed rays are sampled from interpolated views. (b) A stitched ODS video generated from 16 input videos, shown in anaglyphic stereo here. (c) A VR headset in which the video can be viewed.*

Abstract

We present Jump, a practical system for capturing high resolution, omnidirectional stereo (ODS) video suitable for wide scale consumption in currently available virtual reality (VR) headsets. Our system consists of a video camera built using off-the-shelf components and a fully automatic stitching pipeline capable of capturing video content in the ODS format. We have discovered and analyzed the distortions inherent to ODS when used for VR display as well as those introduced by our capture method and show that they are small enough to make this approach suitable for capturing a wide variety of scenes. Our stitching algorithm produces robust results by reducing the problem to one of pairwise image interpolation followed by compositing. We introduce novel optical flow and compositing methods designed specifically for this task. Our algorithm is temporally coherent and efficient, is currently running at scale on a distributed computing platform, and is capable of processing hours of footage each day.

Keywords: Panoramic stereo imaging, Video stitching

Concepts: •**Computing methodologies** → **Computational photography;** *Virtual reality;*

SA '16 Technical Papers, December 05-08, 2016, , Macao
ISBN: 978-1-4503-4514-9/16/12
DOI: http://dx.doi.org/10.1145/2980179.2980257

1 Introduction

As virtual reality (VR) headsets become widely available, capturing content for VR has emerged as a research problem of growing importance. In this paper we introduce Jump, a complete VR capture system, spanning camera design, stitching algorithms, serving format, and display that is optimized to capture real scenes and events for today's VR headsets and video serving platforms. In particular, we seek to achieve the following three criteria:

Immersion: The viewer should feel *immersed*, i.e., present within the captured scene.

Stereopsis: The viewer should see the recorded content in stereo.

Editing and streaming: the content should be represented in a form that can be edited using existing tools, streamed reliably on todays networks, and rendered in real time on todays headsets.

To satisfy these criteria, we present a solution based on omnidirectional stereo (ODS) video [Ishiguro et al. 1990; Peleg et al. 2001]. This format, based on creating a time-varying panorama for each eye, is advantageous because scene content may be represented as a traditional video and streamed on existing video serving platforms (it does not require depth or 3D information, nor does it require that a view interpolation algorithm be run on client-side). Nevertheless, creating omnidirectional video content introduces a number of unique challenges.

First, the ODS projection was not designed for VR. For example, there is no proof that this projection model is capable of producing perspective stereo pairs that can be properly fused. We show that ODS in fact *violates* the epipolar constraint when reprojected to perspective in a VR headset, and we analyze its performance in detail. Our main conclusion is that the violation is minor and only significant for objects that are very close or at extreme angles.

A second major challenge is designing a camera and stitching system for producing ODS video. We perform a detailed analysis of optimal ODS multicamera rigs as a function of field of view, number of

cameras, and camera placement. From this analysis we present an ODS camera system with no moving parts or unusual hardware, consisting of several off-the-shelf cameras on a ring.

The third key contribution of this paper is an ODS video stitching algorithm which produces high quality output completely automatically. This algorithm has been implemented at scale and has processed millions of frames of video and produced content that has been viewed by the general public more than 20 million times. As part of this stitching algorithm we introduce novel optical flow and compositing methods designed specifically for this task.

The remainder of this paper is structured as follows: Section 2 gives an overview of related work while Section 3 gives a summary of the ODS projection followed by an analysis of the distortion it introduces when used for VR video. In Section 4 we detail our capture setup, deriving constraints on feasible rig geometries and quantifying the distortion introduced at capture time. Section 5 describes our approach for stitching content from such a rig, and in Section 6 we discuss our results.

2 Related work

There are many potential formats which could be used for VR video. Lightfields [Levoy and Hanrahan 1996] provide the greatest level of immersion if they can be captured for a suitable volume. Lightfield capture has been demonstrated using 2D arrays of cameras [Wilburn et al. 2005; Yang et al. 2002], however even if arrays such as these were generalized to capture over a sphere, the amount of data that would have to be transmitted for client-side rendering is very large and a practical solution for this has not yet been demonstrated. Also, editing lightfields is itself a challenging problem [Jarabo et al. 2014]. A second format that allows for 3D translation of the viewer is free viewpoint video [Carranza et al. 2003; Collet et al. 2015; Zitnick et al. 2004; Smolic 2011], which allows the viewer to move freely within some volume. While advances have been made in reducing the data rate of these captures [Collet et al. 2015] editing this content remains a challenging problem. Concentric mosaics [Shum and He 1999] allow a viewer to look in any direction and allow for movement on a disc, however severe vertical distortion is introduced as the viewer moves radially on the disc. Omnidirectional stereo (ODS) [Ishiguro et al. 1990; Peleg et al. 2001] allows the user to look around but not to move. This is consistent with a large fraction of existing VR headsets (GearVR [Samsung 2015] and Cardboard [Google 2014]) which track head rotation but not translation. Correct stereo is supported, provided the user does not roll their head (which most users tend to not do). Since the output is a pair of panoramic videos (one for the left eye and one for the right) transmission only requires twice as much data as monoscopic video and many common editing operations are easy (color correction, cross fades, etc). There are also existing tools for making more complex stereo aware edits [Koppal et al. 2010]. For these reasons we propose ODS as a practical format for delivering VR video today.

Some attempts have been made to capture ODS video directly. The system of Tanaka and Tachi uses a rotating prism sheet to capture the relevant rays, but this requires a complex setup and the resulting video is of low quality [Tanaka and Tachi 2005]. The mirror based system of Weissig *et al.*has the advantage of no moving parts and significantly higher video quality, but the vertical field of view is limited to 60 degrees [Weissig et al. 2012]. The term "omnidirectional stereo" has also been used to describe a different projection in which two panoramic images are captured with a vertical baseline [Shimamura et al. 2000; Gluckman et al. 1998]. Video content for this projection can be captured by two cameras with curved mirrors. While that vertical baseline is useful for estimating depth from stereo correspondence, it is not useful for generating VR video.

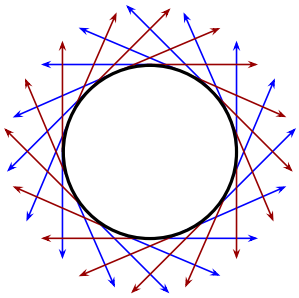


Figure 2: *The omnidirectional stereo (ODS) projection which captures stereo views in all directions by sampling rays tangential to a viewing circle. Red and blue rays comprise left and right eyes respectively.*

Most existing methods for capturing ODS panoramas rely on the scene being static [Peleg et al. 2001; Richardt et al. 2013] or having only a small amount of motion suitable for video textures [Couture et al. 2011], thereby allowing a panorama to be captured by rotating a single camera. To allow video with large motion to be captured, we use 16 cameras placed radially on a rig and use view interpolation between adjacent cameras on that ring to provide the necessary intermediate images. Previous mosaicing methods have used optical flow [Richardt et al. 2013] or depth [Rav-Acha et al. 2008] to aid compositing multiple images into a single scene, but in both cases a much denser set of input images was used, 100-300 in the case of [Richardt et al. 2013] instead of our 16.

In addition to the academic research in this area there are also many companies producing 360 degree video cameras. Some such as the Ricoh Theta or the Gear360 capture monoscopic video. This allows for compact cameras but does not meet our target of immersion since the stereo cue is missing. Some companies are producing stereoscopic 360 degree cameras such as Jaunt, Facebook and Nokia however it is difficult to evaluate these systems as they use proprietary stitching methods.

3 Omnidirectional stereo projection

Panoramas have been around for more than a hundred years, as their ability to render a scene in all directions has made them popular for scene visualization and photography. The much more recent discovery of stereo panoramas presents exciting opportunities for VR video, as they provide a compact representation of stereo views in all directions. First shown in [Ishiguro et al. 1990] and popularized by [Peleg et al. 2001], the omnidirectional stereo (ODS) projection, shown in Figure 2, produces a stereoscopic pair of panoramic images by mapping every direction in 3D space to a pair of rays with origins on opposite sides of a circle whose diameter is the interpupillary distance. The ODS projection is therefore *multi-perspective*, and can be conceptualized as a mosaic of images from a pair of eyes rotated 360 degrees on a circle. Throughout this paper we call the circle that the rays originate from the *viewing circle*.

Formally, for any direction defined by an elevation ϕ and azimuth θ, the viewing ray directions in vector form are $[\sin(\theta)\sin(\phi), \cos(\phi), \cos(\theta)\sin(\phi)]$ and the viewing ray origins are $\pm[r\cos(\theta), 0, -r\sin(\theta)]$ where $+$ is for the right eye, $-$ is for the left eye, and r is generally half of the average viewer's interpupillary distance[Dodgson 2004]. The image for each eye can then be stored as an equirectangular panorama of width w, where $x = {}^{\theta w}/_{2\pi}$ and $y = {}^{\phi w}/_{2\pi}$, or as any other panoramic image such as a cube map.

3.1 Distortion when viewing in VR

Most prior work on ODS has produced equirectangular or cylindrical panoramas which are viewed as stereo pairs on a flat screen [Ishiguro et al. 1990; Peleg et al. 2001]. It has been shown that when these panoramas are instead viewed in a cylindrical display, with the viewer positioned in the center, some distortion is introduced [Couture et al. 2010]. When displaying ODS panoramas in a VR headset a different type of distortion occurs, since we must render perspective views depending on where the user is looking. In this section we show that the resulting distortion introduces vertical disparity, but that this distortion is small in the center of the user's field of view and only increases towards the edges.

To render a perspective view from an ODS panorama, each ray in the ODS panorama should ideally be projected onto the scene's geometry and then back into a perspective view. If, as in our case, the scene geometry is not known, one can instead use a proxy geometry consisting of an infinite sphere. This is equivalent to only considering the direction of each ray but not its origin and is correct for distant content but introduces distortion for nearby content as shown in Figure 3.

Consider a point at $(p_x, p_y, 0)$ being projected into an ODS panorama with a viewing circle of radius r, as shown in Figure 4. This point projects into an ODS panorama at (θ, ϕ) for the left eye and $(-\theta, \phi)$ for the right eye where

$$\theta = \cos^{-1}\left(\frac{r}{p_x}\right) \qquad \phi = \tan^{-1}\left(\frac{p_y}{\sqrt{p_x^2 - r^2}}\right) \qquad (1)$$

Due to the rotational symmetry of the ODS projection, this generalizes to any point. If we now generate a pair of perspective images looking horizontally at an angle α from the point then the rays that view this point will have the following directions for the left and right eyes respectively:

$$\begin{bmatrix} \cos(\phi)\sin\left(\frac{\pi}{2} - \theta - \alpha\right) \\ \sin(\phi) \\ \cos(\phi)\cos\left(\frac{\pi}{2} - \theta - \alpha\right) \end{bmatrix} \qquad \begin{bmatrix} \cos(\phi)\sin\left(\theta - \frac{\pi}{2} - \alpha\right) \\ \sin(\phi) \\ \cos(\phi)\cos\left(\theta - \frac{\pi}{2} - \alpha\right) \end{bmatrix}$$

We can find the point to which these rays project into the left and

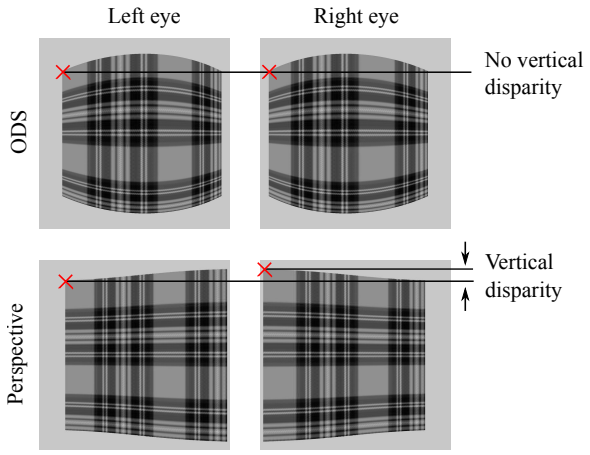


Figure 3: *A synthesized, textured square imaged by the ODS projection and then projected to perspective. The images have an interpupillary distance of* 6*cm and about* 100° *FOV. The square has been placed at the unusually close distance of* 30*cm to exaggerate the vertical parallax for the figure.*

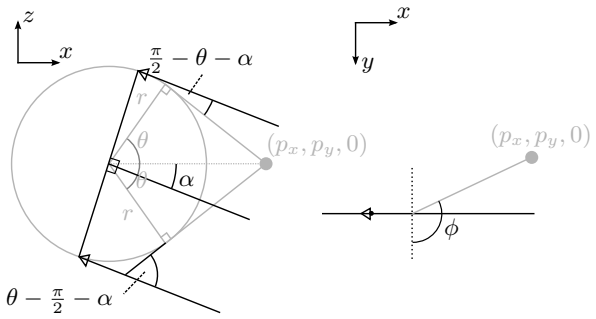
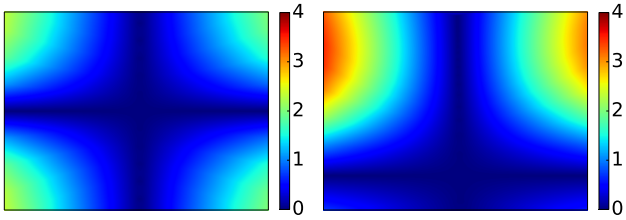


Figure 4: *Top and side views of a point being projected into ODS space (greyed out) and the angles of the rays that view it with respect to two perspective views for a horizontal view rotated α from the point.*



(a) **(b)**

Figure 5: *Vertical parallax in degrees introduced when projecting points on a cylinder of 1m radius from an ODS panorama into a perspective view. Both images are left eye perspective views with a 110 degree horizontal field of view. (a) Looking horizontally. (b) Looking 30 degrees above the horizon.*

right perspective images, assuming a focal length of f:

$$\begin{bmatrix} f\tan\left(\frac{\pi}{2} - \theta - \alpha\right) \\ f\tan(\phi)\sec\left(\frac{\pi}{2} - \theta - \alpha\right) \end{bmatrix} \qquad \begin{bmatrix} f\tan\left(\theta - \frac{\pi}{2} - \alpha\right) \\ f\tan(\phi)\sec\left(\theta - \frac{\pi}{2} - \alpha\right) \end{bmatrix}$$

It can be seen that unless $\theta = \frac{\pi}{2}$, corresponding to a point infinitely far away, $\phi = 0$, corresponding to a point lying on the horizon, or $\alpha = 0$, corresponding to the perspective view looking directly towards the point, then some vertical parallax will be introduced. The closer the point is, the greater this vertical parallax will be.

If this vertical parallax is too large then it can cause problems fusing the left and right eye images. Qin *et al.*[2004] suggest that vertical parallax of half a degree is noticeable. Figure 5a shows that when looking horizontally there is little vertical parallax for points 1m horizontally from the camera. When the user looks up, as in Figure 5b, there is greater distortion towards the edge of the images. This distortion exceeds the limit suggested by Qin *et al.*but fortunately this distortion only occurs towards the edge of a viewer's field of view where it is much less noticeable. If they turn to look at the region that has large distortion in this figure then it will move closer to the center of their field of view and so will exhibit less distortion. Further investigation is required to determine whether this has an impact of comfort for long term viewing.

This distortion is particularly severe when a user is looking straight up or straight down. For the camera design proposed in this paper this is not an issue as it only has a vertical field of view of 120 degrees.

4 ODS capture

Directly capturing the rays necessary to build an ODS panorama is difficult for time varying scenes. While this has been attempted [Tanaka and Tachi 2005] the quality of such approaches is currently below that of the computational approaches to ODS capture. Most previous approaches [Ishiguro et al. 1990; Peleg et al. 2001; Richardt et al. 2013] capture ODS panoramas by rotating a camera on a circle of diameter greater than that of the ODS viewing circle as shown in Figure 6. We use the same approach for capturing ODS video except that instead of rotating a single camera we use a small number of stationary cameras and hallucinate the missing viewpoints needed to produce the ODS panorama. This method of capture introduces some vertical distortion which we analyze below.

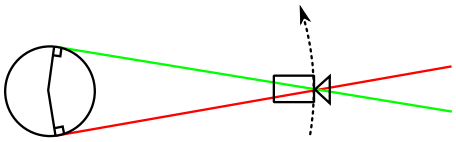


Figure 6: *In the 2D case, all rays tangential to a viewing circle can be captured by rotating a single camera on a larger circle.*

4.1 Vertical distortion

Figure 7a shows the vertical stretching introduced when capturing an ODS panorama with cameras on a ring with larger diameter than the viewing circle. This stretching drops off rapidly as content moves further from the camera. We can quantify this distortion by considering a point offset by z_1 horizontally from the camera at a height h as shown in figure 7b. The ODS projection of the point will appear to be Δh higher and the vertical stretching introduced is given by

$$\frac{\Delta h}{h} = \frac{z_0}{z_1} = \frac{\sqrt{R^2 - r^2}}{z_1}. \quad (2)$$

This distortion can be reduced by minimizing the distance between the cameras and the ODS viewing circle. For the case of static scenes it can be completely avoided by rotating a pair of cameras on the viewing circle, facing tangential to the viewing circle, rather than rotating a single camera on a much larger circle as is common in prior work [Peleg et al. 2001; Richardt et al. 2013].

Generating an ODS panorama by taking the parallel rays to those captured is equivalent to using a proxy geometry of an infinite sphere. If a prior on scene depth is available then a different proxy geometry could be used. For example, if shooting indoors then a sphere with a smaller diameter could be used, or even a cuboid fit to the room. If geometry is available that exactly matches the scene, this distortion could be completely avoided, however a point in 3D space which is visible to the ODS panorama may not be visible to the cameras due to occlusion, and this will cause holes in the panorama.

The analysis in this section assumes that we are capturing images from a circle. In practice since we are interpolating between a small number of cameras the interpolated views lie on a regular polygon instead. This means that the vertical distortion described above varies depending on viewing angle. For a rig with 16 cameras the distance between the ideal circle and the actual interpolated position is 2% of the rig radius and so this effect is very small.

4.2 Rig design for video capture

Our goal is to design a video camera rig that can capture all of the rays needed by the ODS projection shown in Figure 2, while maximizing image quality and minimizing image distortions.

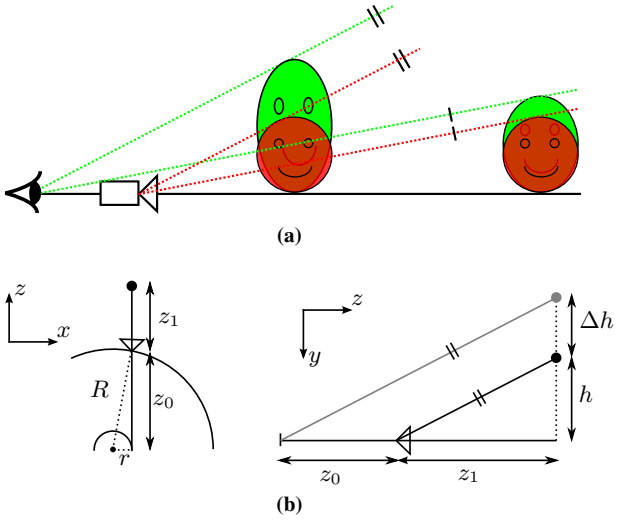


Figure 7: *(a) Vertical distortion caused by capturing with cameras which lie on a larger circle than the ODS viewing circle. The red faces are viewed by a camera. When the parallel rays are used for rendering from a viewpoint behind the camera this leads to the vertically stretched green faces. (b) Top and side views of this distortion being applied to a point.*

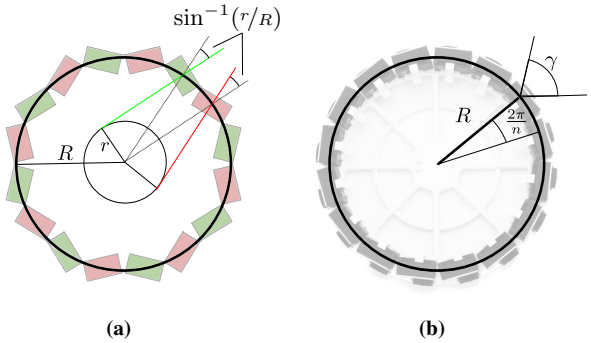


Figure 8: *Sketch of a tangential (left) and radial (right) layout. (a) In the tangential case, the cameras align with the ODS left/right rays by rotating $\pm sin^{-1}(r/R)$ w.r.t. the radial direction. (b) The radial rig geometry is fully defined by its radius R, number of cameras n, and the horizontal field of view γ of the cameras.*

We place the cameras on a circle of radius R which is greater than the radius of the ODS viewing circle r. An ODS ray which passes through a camera will do so at an angle $\sin^{-1}(r/R)$ to the normal of the circle on which the cameras lie. Two distinct camera layouts are possible: a tangential layout and a radial one, as shown in Figure 8.

The tangential layout dedicates half of the cameras to capturing rays for the left image and the other half to capturing rays for the right image, and aligns each camera so that an ODS ray which passes through it does so along its optical axis. On the other hand, the radial layout uses all of the cameras to collect rays for both the left and right images and so each camera faces directly outwards.

The advantage of the radial design is that image interpolation occurs between adjacent cameras, while for the tangential design it must occur between every other camera, which doubles the baseline for the view interpolation problem and makes it more challenging. The

disadvantage of the radial design is that since each camera must capture rays for the left and right image, the horizontal field of view required by each camera is increased by $2\sin^{-1}(r/R)$. In practice this means that the radial design is better for larger rig radii and the tangential design is better for smaller radii.

The cameras we chose to use are around 3cm wide and therefore limit how small the rig can be made. This means that the radial design is more appropriate and all further discussion is based on this layout.

The rig geometry is fully described by 3 parameters (see Figure 8b): the radius of the rig R, the horizontal field of view of the cameras γ, and the number of cameras n. We have several conflicting goals:

- Minimize rig diameter R, thereby reducing vertical distortion as described in Section 4.1.
- Minimize the distance between adjacent cameras, thereby reducing the baseline for view interpolation.
- Have a sufficient horizontal field of view for each camera, so that content at least some distance d from the rig center can be stitched.
- Maximize each camera's vertical field of view, which results in a large vertical field of view in the output video.
- Maximize overall image quality, which generally requires using large cameras.

We now describe a rig geometry that achieves these properties.

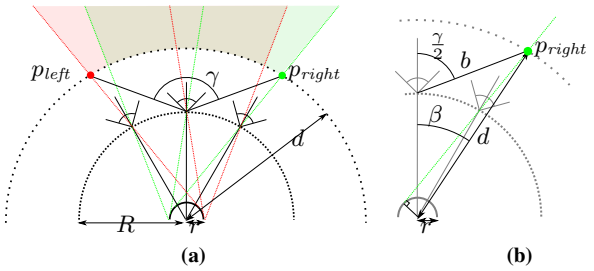


Figure 9: *(a) To successfully stitch all points with distances of at least d from the rig center, the central camera must observe all points in the shaded regions (red for right eye and green for left eye). The extreme points p_{left} and p_{right} constrain the camera's field of view to be at least γ. (b) Here we visualize the intermediate values used when defining γ in (6).*

4.3 Rig geometry

We assume that between adjacent cameras in a ring we can synthesize views on a straight line lying between the two cameras and that these synthesized views can only include points observed by both cameras. Figure 9 shows the volume that must be observed by one camera in order to allow stitching for all points with distances from the rig center of at least d.

Given a ring of radius R containing n cameras, we can derive the minimum required horizontal field of view for each camera γ as follows:

$$\beta = \frac{2\pi}{n} + \cos^{-1}\frac{r}{d} - \cos^{-1}\frac{r}{R} \tag{3}$$

$$b^2 = d^2 + R^2 - 2dR\cos\beta \tag{4}$$

$$\pi - \frac{\gamma}{2} = \cos^{-1}\left(\frac{R^2 + b^2 - d^2}{2Rb}\right) \tag{5}$$

$$\gamma = 2\cos^{-1}\left(\frac{d\cos\beta - R}{\sqrt{d^2 + R^2 - 2dR\cos\beta}}\right). \tag{6}$$

Figure 10 visualizes the relationship between the horizontal field of view γ, the rig radius R, and the number of cameras n, as described in (6). Smaller rig radii require a larger field of view, as rays from the viewing circle intersect the rig at an angle further from normal to the circle. However, as points get close to the rig radius, the required field of view also rises rapidly. For a 40 cm minimum distance, this leads to an optimal rig radius between 10 and 15cm. Figure 10 also shows that increasing the number of cameras for a fixed rig radius reduces the field of view requirements; in addition stitching quality will be improved as image interpolation will be over a shorter baseline. In practice the physical size of the cameras limits how many cameras can be used.

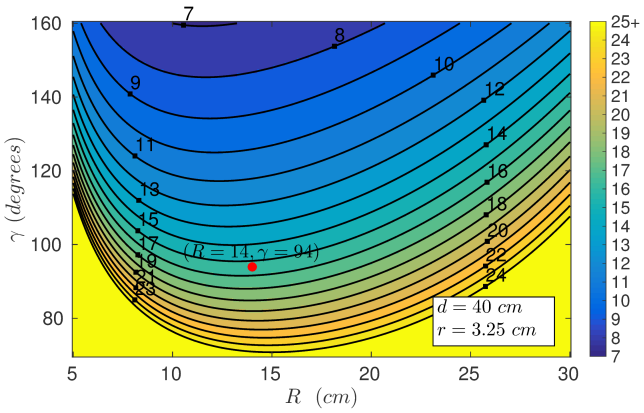


Figure 10: *Minimum number of cameras as a function of rig radius R and horizontal field of view γ. The interpupillary distance is set to 6.5cm and the minimum distance d is set to 40cm. Increasing the number of cameras reduces field of view requirements while increasing the viewing circle radius increases the required field of view. The design choice in this work is shown as a large red dot at $R = 14$cm and $\gamma = 94°$.*

We chose to use GoPro cameras due to their large field of view (94×120 degrees) and reasonably small size. Our design uses 16 cameras on a 28cm diameter ring (see the red dot in Figure 10). Using more cameras would increase the rig diameter leading to more vertical distortion. Reducing the number of cameras would mean we could make the rig smaller but would increase the distance of the nearest point we could stitch.

5 Stitching pipeline

This section describes our stitching pipeline which takes 16 video streams and produces a single stitched ODS video. The individual stages in the pipeline are shown in Figure 11. To run in a timely manner it is crucial that the work can be distributed across many machines. It is also crucial that results are temporally coherent. To allow this the optical flow implementation operates on blocks of frames (40 in all results here) and is temporally coherent within

each block. By using overlapping blocks and discarding the first and last 5 frames of each block, discontinuities at block boundaries are minimized. All other stages operate on each frame individually and are designed so that small changes in their inputs will not produce large changes in their outputs.

5.1 Calibration

We use a standard structure from motion approach [Hartley and Zisserman 2003], with priors provided by the nominal rig layout, to calibrate the intrinsics and the relative pose of the individual cameras in the rig. Each capture is calibrated individually, using 5 frames from the first minute of footage.

5.2 Flow estimation

Interpolating views between cameras in the ring requires a per-pixel correspondence between each pair of adjacent cameras. We solve the general 2D optical flow correspondence problem instead of the simpler 1D stereo problem so that we are robust to correspondences which do not follow epipolar geometry, such as specularities and, since we use cameras with rolling shutter, fast moving objects. Before estimating flow between adjacent cameras we transform the images to remove the effect of camera orientation. This means that horizontal flow is a good approximation to inverse depth (disparity), a fact that is later used during compositing.

Optical flow is a well-studied problem, with classic methods that seek smooth solutions that satisfy brightness constancy assumptions [Horn and Schunk 1981; Lucas and Kanade 1981], to more recent approaches that use feature descriptors to address large displacements [Brox and Malik 2011] or appearance variation [Liu et al. 2011]. However, the problem of correspondence for view interpolation in our setting has different and somewhat contradictory goals:

Visual quality over metric fidelity: Techniques with low endpoint error on standard flow benchmarks [Baker et al. 2011; Menze and Geiger 2015] often produce dramatic artifacts such as temporal flickering or poorly localized edges when used for our task. Our algorithm is designed to minimize visual artifacts, not endpoint error.

Speed: Top flow techniques on the KITTI benchmark [Menze and Geiger 2015] take minutes or hours per megapixel, and would therefore take several compute-years or compute-millennia to process an hour of our footage[1]. In contrast, our approach takes 1.1 seconds per megapixel on cheap commodity hardware, without the use of GPUs or FPGAs.

[1] An hour of footage contains 18.6 million megapixels of flow computation: 16 videos of 5.4 megapixel images at 30 FPS with forward and backward flow.

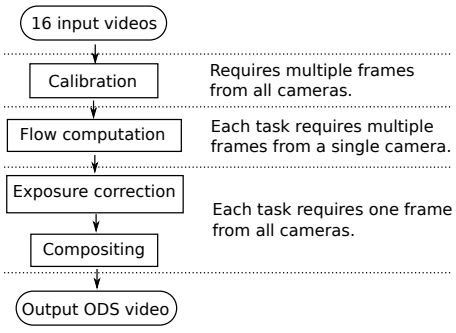


Figure 11: *Structure of the pipeline. Flow is computed on blocks of 40 frames separately for each camera. Exposure correction and compositing are carried out independently for each frame.*

Temporal coherence: Our flow estimates must be coherent from one frame to the next. This is a slightly different problem to traditional temporally consistent optical flow since we are computing flow between images which are separated both spatially and temporally.

The requirements of speed and visual quality are often at odds with each other, as most fast techniques oversmooth at flow discontinuities [Kroeger et al. 2016], and most edge-aware techniques [Krähenbühl and Koltun 2012; Revaud et al. 2015] are more expensive than their non-edge-aware counterparts.

We present an edge-aware and temporally-consistent optical flow algorithm built upon a fast tile-based alignment procedure and a temporal extension of the bilateral solver [Barron and Poole 2016], a technique for efficiently inducing joint edge-aware smoothness.

The algorithm consists of four stages:

1. Locally normalize each image to discard variation due to different exposure or contrast settings.
2. Compute a coarse tile-based alignment, in which for each non-overlapping 32×32 tile in the reference image we perform a brute force sub-pixel accurate search for its best-matching tile in a 256×64 region of the neighboring image.
3. Upsample the per-tile flow field into a per-pixel flow field, while generating a per-pixel confidence measure which reflects the reliability of that pixel's flow.
4. The flow/confidence estimate is used as input to a temporally-consistent bilateral solver, which finds the flow field that best resembles the input flow (where confidence is large) while being as bilateral-smooth as possible (smooth within spatio-temporal regions but not across edges).

We describe our flow algorithm in terms of a single "reference" image I_0 (the image for which flow is computed) and a neighboring image I_1, though this process is performed 32 times for each temporal frame (16 camera pairs, forward and backward).

First, we locally normalize each image by subtracting the local mean and dividing by the local standard deviation:

$$I_0' = \frac{I_0 - \mathrm{box}\left(I_0, r\right)}{\sqrt{\epsilon^2 + \mathrm{box}\left(\left(I_0 - \mathrm{box}\left(I_0, r\right)\right)^2, r\right)}} \tag{7}$$

where $\mathrm{box}\left(\cdot, r\right)$ is a box filter of radius $r = 32$, and $\epsilon = 0.001$ prevents normalization issues in flat image regions. I_1' is defined similarly. Grayscale images are used for this matching step simply for the sake of efficiency.

Given these normalized images, for each non-overlapping 32×32 tile in I_0' we perform a brute-force search for the best matching tile with a horizontal motion in $[0, 224]$ and a vertical motion in $[-16, 16]$, using the single-scale alignment technique of [Hasinoff et al. 2016]. For each tile T_0' in the normalized reference image I_0' we compute a 224×32 sum of squared differences (SSD) distance image:

$$D(u, v) = \sum_{y=1}^{32} \sum_{x=1}^{32} \left(T_0'(x, y) - T_1'(x + u, y + v)\right)^2 \tag{8}$$

where T_1' is a subimage of I_1'. This can be accelerated using sliding-window box filtering and FFTs, as has been done with normalized cross-correlation [Lewis 1995]. We use a Halide implementation [Ragan-Kelley et al. 2012] to further improve performance.

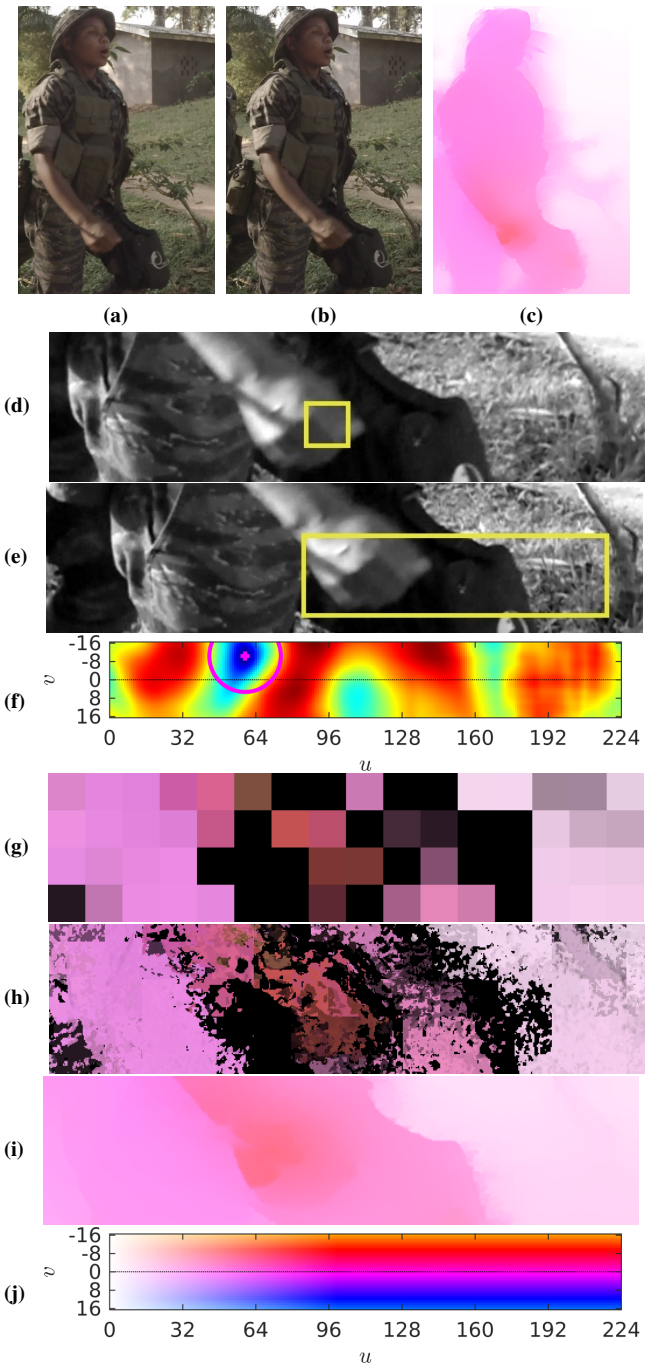


Figure 12: *Given two images in (a) and (b), our flow algorithm produces the edge-aware flow field in (c). We visualize each step of our flow algorithm for a cropped region of these images. For each non-overlapping tile in image* 0 *(d) we identify the larger search area in image* 1 *(e) and compute a normalized SSD surface (f), from which we produce a motion estimate and confidence (shown here as the radius of the circle). Despite this being a stereo pair, significant vertical motion is visible in (f) due to rolling shutter. With our per-tile flow and confidence in (g) we perform a per-pixel upsampling and confidence adjustment to get the proposed flow in (h) (visualized with saturation* $\propto u$*, hue* $\propto v$*, and value* $\propto c^{1/8}$*, as shown in the legend in (j)). This noisy and incomplete flow/confidence is fed into a temporally-consistent bilateral solver to produce the final edge-aware flow field in (i).*

We then extract a subpixel flow estimate from D by fitting a quadratic to the 3×3 window surrounding the argmin of $D(u, v)$ and localizing its minimum:

$$D(u,v) \approx \frac{1}{2}[u\ v]\,\mathbf{A}_i \begin{bmatrix} u \\ v \end{bmatrix} + \mathbf{b}_i^{\mathrm{T}} \begin{bmatrix} u \\ v \end{bmatrix} + c_i \tag{9}$$

$$(U_i, V_i) = -\mathbf{A}_i^{-1}\mathbf{b}_i \tag{10}$$

We can also use this quadratic to produce a confidence for tile i:

$$C_i = \exp\left(\frac{\log|\mathbf{A}_i|}{\sigma_A} - \frac{c_i}{\sigma_c^2}\right) \tag{11}$$

where $\sigma_c = 256$ and $\sigma_A = 5$ determine the importance of the SSD value, and the curvature of SSD, respectively. C_i is large iff the two tiles match well and the match is well-localized. See Figure 12f for a visualization of this process.

These per-tile flow and confidence estimates $\{U_i, V_i, C_i\}$ (shown in Figure 12g) then undergo a series of heuristic transformations to model assumptions about outliers, low-texture regions, repeated texture, object boundaries which do not align with tile boundaries, and forward/backward symmetry (see the supplementary material for details). This results in a per-pixel flow/confidence, where for each pixel i we have $\hat{u}_i$ as horizontal motion, $\hat{v}_i$ as vertical motion, and $\hat{c}_i$ as our estimated confidence of $\hat{u}_i$ and $\hat{v}_i$ (shown in Figure 12h). This flow field is noisy and incomplete, but the flow estimate tends to be accurate when the confidence is large. With this, we can use the bilateral-solver [Barron and Poole 2016] to produce a smoothed estimate of the flow-field which respects edges in the video sequence, while resembling our noisy flow estimate in confident regions (shown in Figure 12i). We use the bilateral solver to solve the following:

$$\underset{\{u_i,v_i\}}{\text{minimize}}\ \frac{\lambda}{2}\sum_{i,j}\hat{W}_{i,j}\left\|\begin{bmatrix} u_i \\ v_i \end{bmatrix} - \begin{bmatrix} u_j \\ v_j \end{bmatrix}\right\|_2^2 + \sum_i \hat{c}_i \left\|\begin{bmatrix} u_i \\ v_i \end{bmatrix} - \begin{bmatrix} \hat{u}_i \\ \hat{v}_i \end{bmatrix}\right\|_2^2 \tag{12}$$

where $\{u_i, v_i\}$ is the smoothed flow field estimated by the solver. The solver contains a smoothness term built around $\hat{W}$, a (bistochastized) bilateral affinity matrix W. To generalize the bilateral solver to video sequences, we need only modify W to include a temporal term in addition to the spatial xy and color ℓuv terms used in [Barron and Poole 2016]:

$$W_{i,j} = \exp\left(-\frac{(p_i^\ell - p_j^\ell)^2}{2\sigma_l^2} - \frac{\left\|[p_i^u, p_i^v] - [p_j^u, p_j^v]\right\|_2^2}{2\sigma_{uv}^2} - \frac{\left\|[p_i^x, p_i^y] - [p_j^x, p_j^y]\right\|_2^2}{2\sigma_{xy}^2} - \frac{(p_i^t - p_j^t)^2}{2\sigma_t^2}\right) \tag{13}$$

where for each pixel i, p_i^ℓ is luma, (p_i^u, p_i^v) is chroma, (p_i^x, p_i^y) is spatial position, and p_i^t is time (the pixel's frame in the video sequence). The parameters ($\sigma_\ell = 16$, $\sigma_{uv} = 8$, $\sigma_{xy} = 12$, and $\sigma_t = 1$) determine the size of the luma, chroma, spatial, and temporal support of the solver. This approach of enforcing temporal consistency by connecting each pixel to its nearby pixels in the video sequence implicitly reasons about object motion by assuming motion is small and temporally smooth for images with the same color, which works well in practice and avoids the need for estimating *temporal* flow across adjacent frames, as is often required by other techniques [Lang et al. 2012]. Our approach is similar to the temporal smoothing technique used in [Meka et al. 2016] for intrinsic image separation, though that approach relies on using randomly sampled connections while the bilateral solver gives us a dense "fully connected" temporal smoothness prior. We solve the problem in Eq. 12 using the same bilateral-space optimization approach as presented in [Barron and Poole 2016], but we optimize over the entire video sequence in a 6-dimensional bilateral-temporal space, rather than a 5-dimensional space.

5.3 Exposure correction

To handle scenes with large exposure variation, which are common in panoramic capture, each camera in the rig autoexposes independently. This means that adjacent cameras may have very different settings (in practice we have observed up to a $3\times$ difference in exposure between adjacent cameras). We need to compensate for this exposure difference before compositing. If we do not then the same point in the scene may have very different exposures in the left and right eye stitches, which makes it difficult for a human observer to fuse the imagery when viewing it in VR.

To compensate for exposure we estimate the average image intensity within the overlapping region of each image pair. Since we already have correspondence estimated between the two images we use this when deciding which regions of the image overlap. For each image i we calculate the average intensity in the region that overlaps with the next image N_i and the average intensity in the region which overlaps with the previous image P_i. We then estimate a gain to apply to each image g_i aiming to minimize

$$\sum_{i=1}^{n} \left(g_i N_i - g_{i+1} P_{i+1}\right)^2 + \epsilon \left(1 - g_i\right)^2 \tag{14}$$

where indices are calculated using modulo arithmetic, so that index $n+1$ is equivalent to index 1, and $\epsilon = 0.001$ is a small value which controls the strength of our prior, that gains should be close to one.

We apply the gains g_i estimated in this way to each image before compositing. This means that the final stitch is a high dynamic range (HDR) image if the input images had different exposures. We found this method for estimating exposure correction to be robust enough that no temporal regularization was needed. Figure 13a shows a crop of a stitch generated with no exposure correction. The exposure is very different for the crowd to the left of the rink in the left-eye and right-eye panoramas which makes it difficult to fuse. Figure 13b shows the same scene after exposure correction has been applied, with the resulting HDR image clamped to 8 bits. The scene is now equally exposed in all directions and there is less variation between the left and right eyes, however when clamping to 8 bits the rink becomes blown out.

Ideally the HDR stitch would be transmitted and viewed using an HDR display, but to leverage existing video streaming platforms we must generate an 8 bit video. Local tone mapping for video is a challenging problem [Aydin et al. 2014] and extending it to be stereo consistent is even more challenging. Here we propose a simple heuristic approach which generates results faithful to the original input images while avoiding blowing out regions.

Given the gains g_i already calculated, we have an estimate of how exposure of the input cameras varies around the rig and we aim to match this in the output. For each column of both left- and right-eye panoramas we estimate a gain by projecting a ray horizontally for that column, finding the two cameras on the rig which it passes between, and taking a weighted average of the gain for those two cameras. Concretely, if the column's longitude is θ_c and the cameras' gains and longitudes when projected into the panorama are (θ_0, g_0) and (θ_1, g_1) then the gain for that column g_c is given by:

$$g_c = \frac{\theta_c - \theta_0}{\theta_1 - \theta_0} g_0 + \frac{\theta_1 - \theta_c}{\theta_1 - \theta_0} g_1. \tag{15}$$

For each column in the output stitch we then take the max of the gain estimates for the left eye and right eye and divide the values in the column by this value. This favors making the stitch darker rather than lighter and ensures that we never blow out any content that isn't blown out in the input videos. While this approach is not stereo consistent (nearby objects will appear in different columns and therefore have different gains applied) the variation this causes is small enough to not cause problems when fusing. Figure 13c shows the result of applying this approach.

Figure 13: *A scene containing significant exposure variation, with the top half of each image cropped from the left eye's panorama and the bottom half cropped from the right eye's. (a) No exposure correction - the left edge of the image is hard to fuse due to exposure differences. (b) Exposure correction applied and resulting HDR image clamped to 8 bits - the rink is blown out. (c) Exposure correction followed by tone mapping to match input exposures.*

5.4 Projection into ODS

Given the optical flow calculated in Section 5.2 we can synthesize an image at any point between adjacent cameras on the ring. To generate an ODS stitch we could synthesize hundreds of intermediate images around the ring and then use existing techniques [Peleg et al. 2001]; however this approach is very slow and it is much more efficient to project each input pixel in the original images directly into its position in the ODS stitch.

This direct projection into the ODS stitch can be performed very efficiently if we make two assumptions. As we linearly interpolate from one camera to the next:

- The heading of an ODS ray which passes through the center of the interpolated camera varies linearly.
- The projection of a 3D point in the interpolated camera varies linearly.

Under these two assumptions, when interpolating between two cameras we just need to find the fraction of the way between the cameras at which the ODS heading for the interpolated camera and the interpolated feature are equal. Concretely, when interpolating between cameras with ODS headings θ_0 and θ_1 we should project a point whose heading is θ_a in the first camera and θ_b in the second (found through optical flow) to a heading θ_p in the ODS stitch where

$$\theta_p = \frac{(\theta_1 - \theta_b) * \theta_0 + (\theta_0 - \theta_a) * \theta_1}{\theta_b - \theta_a + \theta_0 - \theta_1}. \tag{16}$$

The assumptions listed earlier are good approximations unless points come too close to the rig. Figure 14 shows the error introduced by this approximation for points at several different depths for our rig geometry. For points 1m from the camera, the maximum value of

this error is around 0.05° which corresponds to an error of 1 pixel in our full resolution stitches (8192 pixels wide).

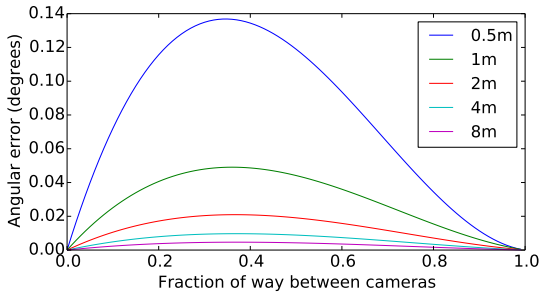


Figure 14: *The error introduced, for points at various distances from the rig center, by using the approximation in Eq. 16 rather than synthesizing intermediate views explicitly and generating an ODS stitch from those.*

5.5 Compositing

The rendering approach described in Section 5.4 generates a variable number of splats on each pixel in the stitched video. Each input pixel creates one splat that lands somewhere in the output as determined by the optical flow. When compositing at the same angular resolution as the input images, more than one splat typically lands in every output pixel. Pixels that receive zero splats are rare and are filled by a post process diffusion step.

We do not use elaborate splat kernels as is typical in point splatting [Gross and Pfister 2007]. Instead, we project the center of each input pixel to the output, and create bilinearly-weighted fragments on all four neighboring output pixels, yielding four output fragments per input pixel (per eye).

The fragments in the same output pixel have to be composited together using a method that accounts for occlusion, produces anti-aliased edges, is spatially and temporally coherent, and visually pleasing.

The problem of compositing surface splats has been considered before, as in Zwicker *et al.*'s work [2001], but their approach produces coherent results only if the depths associated with the splats can be very accurately determined.[2] The same is true for a classic z-buffering approach where only the splat with the smallest depth value is shown. This problem can be partially addressed with soft z-buffering [Pulli et al. 1997], but both hard and soft z-buffering can also yield jagged edges due to their inability to model partial coverage.

In our case, the depth ordering of the fragments is achieved by sorting based on the flow vector length, which is approximately equal to disparity. Since optical flow is based on captured content, it is unrealistic to assume that the disparity is exact. In a video sequence, this means that the relative ordering of fragments may change from frame to frame, causing temporal flickering. Also, subtle changes between neighboring pixels may cause spatial discontinuities.

To be spatially and temporally coherent, we require that the composited result should be C^0-continuous with respect to disparities. I.e. an infinitesimal change in disparity should produce an infinitesimal change in the result. Simply averaging contributions together regardless of disparity is C^0-continuous, but ignoring occlusion in this way

[2] See the accompanying video for a demonstration of coherency issues caused by fluctuating depth/disparity and Zwicker *et al.*'s. method.

(a) **(b)** **(c)**

Figure 15: Compositing Methods. *(a) Results by averaging all contributions to each pixel. (b) Results using our continuous compositing method. (c) The difference between (a) and (b).*

produces "ghosting" artifacts as can seen in Figure 15. Averaging is suitable for combining contributions that represent the same surface, whereas separate surfaces should be composited in disparity order using the over-operator [Porter and Duff 1984]. As in [Zwicker et al. 2001], the problem is to group contributions into surfaces and then combine the surfaces together using the over-operator, but while also satisfying the requirement of continuity (e.g., avoiding hard clustering of fragments which can lead to lack of coherence).

To achieve this goal, we use an *interval-based* compositing algorithm, where we assign a finite disparity range to each fragment (see Figure 16a-b). Intuitively, this turns the fragment into a volumetric object, where the disparity range models the uncertainty over the optical flow. In our system, we assign a constant-sized disparity range to each fragment, though other choices are possible.

Baran *et al.* [2011] used a similar idea to achieve continuity for compositing paint strokes. In their technique well-separated fragments are composited in depth order, whereas fragments at close depths are composited in stroke order. Transition between the two modes happens smoothly. What follows can be seen as an application of this technique to surface splatting. We explain this in detail below.

Each pixel is composited by sorting the endpoints of the disparity ranges and processing these endpoints in order of increasing disparity. This allows us to consider one homogeneous span at the time (see Figure 16c). The color contribution that a fragment makes along such a span is proportional to the length of the span divided by the total fragment range (denoted by constant k). The alpha-premultiplied RGBA-color $\mathbf{c}'_i$ of span i is computed by adding together contributions from all the fragments whose disparity range overlaps with that span:

$$\mathbf{c}'_i = \frac{d_{i+1} - d_i}{k} \sum_{j \in \theta_i} \lambda \mathbf{f}_j \qquad (17)$$

where d_i refers to the disparity at the beginning of span i, θ_i to

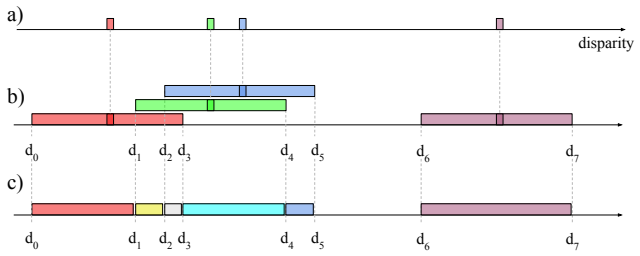


Figure 16: *The stages of our interval-based compositing algorithm explained using a single pixel. (a) First, the splatting algorithm has assigned four fragments into the pixel. (b) Next, the fragments are expanded by applying a constant offset to both directions in disparity space. (c) Finally, spans of constant color are created and composited using the* over*-operator.*

the group of fragments on span i, and $\mathbf{f}_j$ to the RGBA-color of the fragment j in the premultiplied format. λ is a constant multiplier that should be set to the smallest value such that surfaces still look solid. This parameter is needed because, as is common in surface splatting, we cannot guarantee that the weights or alpha values of splats assigned to a pixel construct an exact partition of unity [Gross and Pfister 2007]. For the same reason, we must normalize the color $\mathbf{c'}_i$ by the alpha channel if α_i exceeds one:

$$\mathbf{c}_i = \begin{cases} \mathbf{c}'_i/\alpha_i, & \text{if } \alpha_i > 1 \\ \mathbf{c}'_i, & \text{otherwise} \end{cases} \tag{18}$$

Finally, the composited color for each pixel is obtained by combining the contributions of the individual spans with an over-operator:

$$\mathbf{c}_{composite} = \mathbf{c}_0 \text{ over } \mathbf{c}_1 \text{ over } .....\mathbf{c}_n \tag{19}$$

In practice all the above can be done in three stages: 1) convert fragments to start and end points 2) sort the start and end points 3) compute the composited color in a single sweep through the start and end points.

For all results we set $\lambda = 5.23$ and $k = 0.0055$, which were experimentally found (and must be adjusted depending on image resolution). Tuning k allows for a balance between coherence and faithful modeling of occlusions. A very large k yields results similar to the averaging method, whereas a very small k corresponds to combining all the input fragments with the over-operation, which lacks coherence similar to classic z-buffering. Our setting of k is small enough such that occlusions with well separated background and foreground disparities are handled correctly with the over-operator, but fragments with similar disparity are averaged. In other words, for small peturbations of the input flow there will only be small changes in the output, even if the ordering of the fragments changes.

6 Results

We have tested our system on a wide variety of scenes, stitching millions of frames of content. Our results are best viewed in a VR headset, and a selection of videos processed with our proposed system can be found at https://goo.gl/2Of8Dm. Due to the current limitations of internet streaming platforms, these videos cannot be streamed at their full resolution (8192 pixels wide), so full resolution viewing is limited to local playback. ODS still images taken from a selection of videos are shown in Figure 18.

6.1 Failure cases

Overall we have found stitching quality to be high for the majority of scenes, but there are some cases in which artifacts are introduced. Figure 17 shows the main failure cases we have observed:

Objects too close: If objects come closer to the camera than the limits described in Section 4.3 then stitching is not possible.

Thin structures: Objects which are smaller than the tile size used in our optical flow algorithm may be assigned incorrect depths, resulting in a "ghosting" effect in the final rendering.

Semi-transparent surfaces: Because we only estimate a single flow value per input pixel, pixels with multiple depths (ie, transparent surfaces) may exhibit distortion.

Flow mismatches: Challenging scenes may result in incorrect flow fields (see Figure 17d). This can produce significant artifacts but is very rare in practice.

We found that the impact of these errors, especially thin structures and semi-transparent surfaces, was significantly reduced by ensuring that results are temporally coherent. Most viewers do not notice if small or transparent objects have been assigned incorrect depths and are "ghosting", but viewers are likely to notice if those same objects are ghosting more or less over time, or switching abruptly between depths.

A different type of failure case is very fast motion. In this case stitching quality remains high, but at playback time, due to the fact that we capture video at 30fps, it become very obvious that objects are moving in discrete thirtieth of a second steps instead of smoothly. This effect is very noticeable when viewing the video in a VR headset although it can be alleivated by capturing and playing back at higher framerates.

6.2 Computational cost

The stitching algorithm must handle a large amount of data. Input is sixteen 2704×2028 30 FPS video streams and the output is a single 8192×8192 video stream, with a 8192×4096 panorama for each eye (with the left eye on top of the right). The algorithm takes about 60 seconds per frame (where each frame consists of 16 images) on a single machine, meaning an hour of video would take 75 days to process. To allow for timely processing we use a large number of machines in parallel. The per-frame timings for a representative run on 390,000 frames are shown in Table 1. The total time per frame is about five times that of when running on a single machine, due to the single machine being able to run some operations in parallel and to the additional overhead necessary to run in a distributed architecture. For example, flow must be saved to disk between stages of the pipeline, and to reduce the amount of disk used it must be compressed, which takes some time. Even with this overhead, parallelizing computation over 1000 cores allows for an hour of footage to be processed in $\sim$10 hours.

The majority of time is spent in optical flow computation, despite the fact that each individual flow field takes $\sim$5 seconds to compute. This is because for each frame of output, sixteen forwards and backwards flow fields must be computed, and 25% of those flow fields are discarded due to our use of overlapping temporal blocks to ensure temporal smoothness. Peak memory usage occurs during compositing and is 16GB.

Our algorithm is modular, and different flow or compositing methods can be used. By running flow on downsampled images or replacing compositing with a simpler scheme we can significantly reduce run time at the expense of output quality.

Figure 18: *Still stereo frames taken from several stitches, represented here as anaglyphs.*

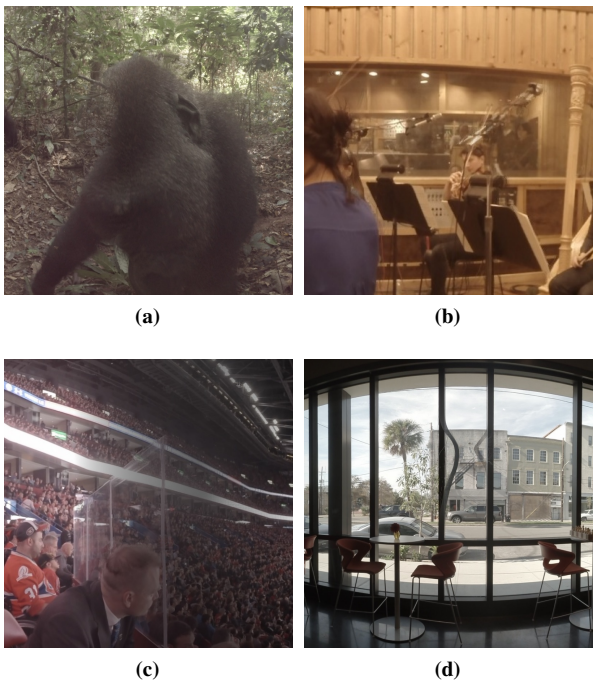


Figure 17: Failure cases - *crops from one eye of the ODS stitches. (a) As objects get closer than the limits described in section 4.3 stitching completely breaks down, here the gorilla is ~15cm from the camera. (b) Thin structures can be missed in the flow computation stage which leads to ghosting on the mic stand and bow. (c) Semi-transparent surfaces can deform as shown here. (d) In very rare cases flow mismatches occur. Here a combination of the correct match being occluded and repeated texture providing a good match in the wrong location leads to a severe warp.*

Operation	Time (sec.)
Flow computation	183
Compositing	54
Frame IO and rectification	40
Flow compression/decompression	38
Post processing/one off setup	6
Total	321

Table 1: *Mean processing time per frame, averaged over* $390,000$ *input frames.*

7 Conclusion

We presented a VR video capture, stitching, and rendering system that captures high quality ODS video for display in today's VR headsets. Our system has so far processed over 150 hours of footage and produced videos with a total of over 20 million views.

Our contributions include a detailed analysis of distortions introduced by using ODS for VR video, and showed that while vertical parallax is introduced, it is small enough to be acceptable in practice. We also characterized the design space of possible multi-camera ODS rigs. Based on this analysis, we focused on one specific rig design that optimizes the design tradeoff given cameras that are currently available off the shelf. Our stitching algorithm includes novel optical flow and compositing algorithms that yield state-of-the-art results with a limited run-time budget.

While the system works remarkably well, its reliance on optical flow can cause failures in a number of situations, including very large motions (e.g., objects significantly closer than a meter), transparency, thin structures, and repetitive content. We expect that performance will continue to improve due both to developments in hardware (smaller, more closely-spaced cameras), and software (e.g., better flow algorithms).

Having a system capable of generating large amounts of ODS video allows for further study of this format. Specifically it would be interesting to investigate the effects of long-term viewing of ODS video and whether it induces fatigue. While ODS video supports only head rotation (i.e., three degrees of freedom), we hope to see VR video solutions in the future that additionally support head translation, enabling full 6DOF viewing experiences. In principle, the parallax information that we compute through optical flow could be used to derive depthmaps and re-render the scene from any new viewpoint. In practice, however, we have found this re-rendering task to be much more prone to visible artifacts, as people appear to be more sensitive to motion parallax errors than stereo parallax errors. Capturing 6DOF VR video is a fascinating and critically important topic for future work, and we look forward to seeing major progress on this topic in the coming years.

References

AYDIN, T. O., STEFANOSKI, N., CROCI, S., GROSS, M., AND SMOLIC, A. 2014. Temporally coherent local tone mapping of hdr video. *TOG*.

BAKER, S., SCHARSTEIN, D., LEWIS, J. P., ROTH, S., BLACK, M. J., AND SZELISKI, R. 2011. A database and evaluation methodology for optical flow. *IJCV*.

BARAN, I., SCHMID, J., SIEGRIST, T., GROSS, M., AND SUMNER, R. W. 2011. Mixed-order compositing for 3d paintings. *TOG*.

BARRON, J. T., AND POOLE, B. 2016. The fast bilateral solver. *ECCV*.

BROX, T., AND MALIK, J. 2011. Large displacement optical flow: Descriptor matching in variational motion estimation. *TPAMI*.

CARRANZA, J., THEOBALT, C., MAGNOR, M. A., AND SEIDEL, H.-P. 2003. Free-viewpoint video of human actors. *TOG*.

COLLET, A., CHUANG, M., SWEENEY, P., GILLETT, D., EVSEEV, D., CALABRESE, D., HOPPE, H., KIRK, A., AND SULLIVAN, S. 2015. High-quality streamable free-viewpoint video. *TOG*.

COUTURE, V., LANGER, M. S., AND ROY, S. 2010. Analysis of disparity distortions in omnistereoscopic displays. *ACM Transactions on Applied Perception (TAP)*.

COUTURE, V., LANGER, M. S., AND ROY, S. 2011. Panoramic stereo video textures. *ICCV*.

DODGSON, N. A. 2004. Variation and extrema of human interpupillary distance. *SPIE: Stereoscopic Displays and Applications*, 3646.

GLUCKMAN, J., NAYAR, S. K., AND THORESZ, K. J. 1998. Real-time omnidirectional and panoramic stereo. *Proc. of Image Understanding Workshop*.

GOOGLE, 2014. Google Cardboard. https://en.wikipedia.org/wiki/Google_Cardboard.

GROSS, M., AND PFISTER, H. 2007. *Point-Based Graphics*. Morgan Kaufmann Publishers Inc.

HARTLEY, R., AND ZISSERMAN, A. 2003. *Multiple view geometry in computer vision*. Cambridge university press.

HASINOFF, S. W., SHARLET, D., GEISS, R., ADAMS, A., BARRON, J. T., KAINZ, F., CHEN, J., AND LEVOY, M. 2016. Burst photography for high dynamic range and low-light imaging on mobile cameras. *SIGGRAPH Asia*.

HORN, B. K. P., AND SCHUNK, B. G. 1981. Determining optical flow. *Artificial Intelligence*.

ISHIGURO, H., YAMAMOTO, M., AND TSUJI, S. 1990. Omnidirectional stereo for making global map. *ICCV*.

JARABO, A., MASIA, B., BOUSSEAU, A., PELLACINI, F., AND GUTIERREZ, D. 2014. How do people edit light fields? *SIGGRAPH*.

KOPPAL, S. J., ZITNICK, C. L., COHEN, M. F., KANG, S. B., RESSLER, B., AND COLBURN, A. 2010. A viewer-centric editor for 3d movies. *Computer Graphics and Applications*.

KRÄHENBÜHL, P., AND KOLTUN, V. 2012. Efficient nonlocal regularization for optical flow. *ECCV*.

KROEGER, T., TIMOFTE, R., DAI, D., AND GOOL, L. J. V. 2016. Fast optical flow using dense inverse search. *ECCV*.

LANG, M., WANG, O., AYDIN, T., SMOLIC, A., AND GROSS, M. 2012. Practical temporal consistency for image-based graphics applications. *SIGGRAPH*.

LEVOY, M., AND HANRAHAN, P. 1996. Light field rendering. *CGIT*.

LEWIS, J. 1995. Fast normalized cross-correlation. *Vision interface*.

LIU, C., YUEN, J., AND TORRALBA, A. 2011. Sift flow: Dense correspondence across scenes and its applications. *TPAMI*.

LUCAS, B. D., AND KANADE, T. 1981. An iterative image registration technique with an application to stereo vision. *IJCAI*.

MEKA, A., ZOLLHOEFER, M., RICHARDT, C., AND THEOBALT, C. 2016. Live intrinsic video. *SIGGRAPH*.

MENZE, M., AND GEIGER, A. 2015. Object scene flow for autonomous vehicles. *CVPR*.

PELEG, S., BEN-EZRA, M., AND PRITCH, Y. 2001. Omnistereo: Panoramic stereo imaging. *TPAMI*.

PORTER, T., AND DUFF, T. 1984. Compositing digital images. *SIGGRAPH*.

PULLI, K., HOPPE, H., COHEN, M., SHAPIRO, L., DUCHAMP, T., AND STUETZLE, W. 1997. View-based rendering: Visualizing real objects from scanned range and color data. *Proc. Eurographics Workshop on Rendering*.

QIN, D., TAKAMATSU, M., AND NAKASHIMA, Y. 2004. Measurement for the panum's fusional area in retinal fovea using a three-dimension display device. *Journal of Light & Visual Environment*.

RAGAN-KELLEY, J., ADAMS, A., PARIS, S., LEVOY, M., AMARASINGHE, S., AND DURAND, F. 2012. Decoupling algorithms from schedules for easy optimization of image processing pipelines. *SIGGRAPH*.

RAV-ACHA, A., ENGEL, G., AND PELEG, S. 2008. Minimal aspect distortion (mad) mosaicing of long scenes. *IJCV*.

REVAUD, J., WEINZAEPFEL, P., HARCHAOUI, Z., AND SCHMID, C. 2015. EpicFlow: Edge-Preserving Interpolation of Correspondences for Optical Flow. *CVPR*.

RICHARDT, C., PRITCH, Y., ZIMMER, H., AND SORKINE-HORNUNG, A. 2013. Megastereo: Constructing high-resolution stereo panoramas. *CVPR*.

SAMSUNG, 2015. Samsung Gear VR. https://en.wikipedia.org/wiki/Samsung_Gear_VR.

SHIMAMURA, J., YOKOYA, N., TAKEMURA, H., AND YAMAZAWA, K. 2000. Construction of an immersive mixed environment using an omnidirectional stereo image sensor. *Workshop on Omnidirectional Vision*.

SHUM, H.-Y., AND HE, L.-W. 1999. Rendering with concentric mosaics. *CGIT*.

SMOLIC, A. 2011. 3d video and free viewpoint videofrom capture to display. *Pattern recognition*.

TANAKA, K., AND TACHI, S. 2005. Tornado: Omnistereo video imaging with rotating optics. *TVCG*.

WEISSIG, C., SCHREER, O., EISERT, P., AND KAUFF, P. 2012. *The ultimate immersive experience: panoramic 3D video acquisition*. Springer.

WILBURN, B., JOSHI, N., VAISH, V., TALVALA, E.-V., ANTUNEZ, E., BARTH, A., ADAMS, A., HOROWITZ, M., AND LEVOY, M. 2005. High performance imaging using large camera arrays. *TOG*.

YANG, J. C., EVERETT, M., BUEHLER, C., AND MCMILLAN, L. 2002. A real-time distributed light field camera. *Rendering Techniques 2002*.

ZITNICK, C. L., KANG, S. B., UYTTENDAELE, M., WINDER, S., AND SZELISKI, R. 2004. High-quality video view interpolation using a layered representation. *TOG*.

ZWICKER, M., PFISTER, H., VAN BAAR, J., AND GROSS, M. 2001. Surface splatting. *CGIT*.

Jump: Virtual Reality Video Supplement

Robert Anderson David Gallup Jonathan T. Barron Janne Kontkanen
Noah Snavely Carlos Hernández Sameer Agarwal Steven M. Seitz

Google Inc.

1 Coarse Tile Flow Upsampling

In the main paper we described a technique for producing a coarse per-tile alignment between a pair of images, in which a brute-force normalized SSD computation is used to produce a set of horizontal and vertical displacements and a corresponding confidence of that displacement. That is, for the set of non-overlapping 32×32 tiles in the image of interest we have $\{U_i, V_i, C_i\}$, where U_i, V_i, and C_i are the horizontal displacement, vertical displacement, and confidence, respectively. From this per-tile flow/confidence field we will produce a per-pixel flow/confidence field. To do this, we will apply a series of heuristic operations which lower the confidence of tiles likely to have incorrect flow estimates. From this refined per-tile alignment we produce an upsampled per-pixel flow/confidence field $\{\hat{u}_i, \hat{v}_i, \hat{c}_i\}$ via an adaptive upsampling process which attempts to best warp the tile flow to the structure of the reference image. Each image pair's "forward" per-pixel flow/confidence field is then combined with that pair's corresponding "backward" per-pixel flow/confidence field to model our assumption that the flow field should be symmetric. Our resulting flow/confidence field is fed into the bilateral solver as described in the main paper, which causes the flow to be denoised and inpainted in low-confidence regions but preserved in high-confidence regions. The bilateral solver is an aggressive smoothing operator which performs global optimization across entire video sequences. It is very effective at inpainting low-confidence regions but is not robust to incorrect flow estimates with high confidence. Our coarse flow refinement and upsampling procedure is therefore designed to be very conservative when assigning high confidence to pixels. A small number of very large confidence pixels which reliably indicate motion, are sufficient to inpaint large regions of the image in an edge-aware fashion.

1.1 Repeated Texture

Recall that each tile's motion was estimated by taking the (subpixel) argmin of an SSD image:

$$(U_i, V_i) \approx \arg\min_{u,v} D_i(u, v). \tag{1}$$

Simplifying the structure of $D_i(u, v)$ down to a single point ignores a great deal of information that may be present in $D_i(u, v)$. For example, if there is repeated texture in the other image, then there may be many local minima in $D_i(u, v)$ which have nearly as small a SSD as the global minimum. If this is the case we would ideally propagate all of these minima, however since the filtering stage requires a single estimate for displacement we instead reduce the confidence for this tile. To this end, after extracting the global minimum from $D_i(u, v)$ we extract a second minimum which is at least 32 pixels away from (U_i, V_i) in u and v. Let $d_i = \min_{u,v}(D_i(u, v))$ be the global minimum of D_i corresponding to (U_i, V_i), and let d'_i be the value of D_i at this second minimum. The tile's confidence C_i is updated based on the ratio between d_i and d'_i as follows.

$$C_i \leftarrow C_i \exp\left(-w_r \min(1, \max(0, \frac{(\max(\epsilon_d, d_i) - d'_i r_0)^2}{{d'_i}^2 (r_1 - r_0)^2}))\right), \tag{2}$$

where w_r is a weight that controls the overall effect of the term, ϵ_d is a small value that ensures that this term still has an effect even as d approaches zero and r_0 and r_1 give the range of ratios over which this term transitions from having no effect to having full effect. We use $w_r = 100$, $\epsilon_d = 50$, $r_0 = 0.6$ and r_1=0.8.

1.2 Low Variance Tiles

If the tile of the reference image being matched has very little image texture, then the motion estimated for that tile should be assigned a low confidence. If we were to simply compute the SSD between non-normalized image tiles, using the determinant of the $\mathbf{A}$ matrix in the confidence would naturally encourage this property. But because our images are pre-normalized to have a mean of zero and a standard deviation of 1, our SSD measure assumes all tiles are comparably textured. To this end, for each tile we look at the non-normalized tile and compute it's variance $\mathrm{var}(T)$ then update the tile confidence using

$$C_i \leftarrow C_i \exp\left(-\max\left(0, \frac{w_v}{\mathrm{var}(T)} - \epsilon_v\right)\right). \tag{3}$$

w_v is a weight which controls the strength of this term and ϵ_v is a threshold on variance below which we consider a tile to have low variance. We use $w_v = 100$ and $\epsilon_v = 25$. When calculating $\mathrm{var}(T)$ pixel values in the input image range from 0 to 255.

1.3 Outlier Tiles

We observe that accurate tile flow estimates tend to have nearby tiles with similar flow. Therefore, if none of a tile's neighbors have a flow which is sufficiently close to that tile's flow, we reduce that tile's confidence:

$$C_i \leftarrow C_i \exp\left(-\min_{j \in \mathrm{neigh}(i)} \left(\frac{(U_i - U_j)^2}{\sigma_u^2} + \frac{(V_i - V_j)^2}{\sigma_v^2}\right)\right) \tag{4}$$

Where $\mathrm{neigh}(i)$ are the 4-connected neighbors of tile i and σ_u and σ_v control the scale of the expected variation between a tile's flow and its neighbor. We set $\sigma_u = 16$ and $\sigma_v = 1$, thereby allowing for large neighboring variation in horizontal motion between neighbors (ie, large depth discontinuities) while discouraging vertical motion. By taking the min over each neighbor, a tile's confidence can remain high provided there is at least one neighbor with a similar flow.

1.4 Image Aware Upsampling

Mapping a per-tile flow/confidence field to a per-pixel field requires an upsampling step. Straightforward choices for this upsampling

operation can have a large negative impact on the quality of the output. For example, using nearest-neighbor upsampling produces a blocky flow field, which also does not respect the structure of the reference image. Using bilinear or bicubic upsampling often produces egregious oversmoothing artifacts, as interpolation incorrectly assumes that a pixel between four tiles has a motion which is some average of those four tile's motions, when the pixel's motion is likely best modeled as being similar to one or more tiles but not similar to the average of all tiles. We therefore use a modified nearest-neighbor upsampling procedure: we look at the motions of the four tiles which "bound" each pixel and assign each pixel the motion which minimizes the error between a 3×3 window centered on that pixel and the corresponding window in the alternate image indicated by that tile's motion.

$$t_i = \underset{t \in \text{bound}(x,y)}{\arg\min} \sum_{a=-1}^{1} \sum_{b=-1}^{1} \left| I'_0(x+a, y+b) - I'_1(x+a+U_t, y+b+V_t) \right|$$

Where $\text{bound}(x, y)$ is the list of four tiles which surround pixel i, t_i is the tile index which we identify as producing the minimum residual error for pixel i, and I'_0 and I'_1 are the normalized grayscale images for tile-matching as defined in the main paper. With this we can produce a per-pixel flow/confidence field, where the per-pixel flow is simply the per-tile flow using these tile assignments, and the per-pixel confidence is the per-tile confidence attenuated by the per-pixel image residual.

$$\hat{u}_i = U_{t_i} \qquad \hat{v}_i = V_{t_i} \tag{5}$$

$$\hat{c}_i = C_{t_i} \exp\left(- \frac{\max\left(0, \left| I'_0(x,y) - I'_1(x+\hat{u}_i, y+\hat{v}_i) \right| - \epsilon_{up}\right)}{\sigma_{up}} \right)$$

Where $\sigma_{up} = 0.5$ and $\epsilon_{up} = 0.2$. Updating the per-pixel confidence in this way means that pixels which are not well explained by any nearby tile have very low confidence, and will therefore be inpainted during optimization.

1.5 Flow Asymmetry

Now that we have our per-pixel "forward" and "backward" flow fields for each image pair, we can reason about the symmetry or asymmetry these flow fields. If the estimated flow from pixel i in image 0 maps to pixel j in image 1, we would also expect the estimated flow from pixel j in image 1 to map back to pixel i in image 0. If this property does not hold, then the forward flow estimate at pixel i should not be trusted, and its confidence will be decreased accordingly. In a small abuse of notation, here let $\hat{u}_f(x, y) = u_i$ where pixel i is located at position (x, y) in our "forward" flow field, and let $\hat{u}_b(x, y) = u_i$ for our "backward" flow field (with $\hat{v}_f$, $\hat{v}_b$ defined similarly).

$$\begin{aligned} a_i &= \left(\hat{u}_f(x,y) + \hat{u}_b(x + \hat{u}_f(x,y), y + \hat{v}_f(x,y))\right)^2 \\ &\quad + \left(\hat{v}_f(x,y) + \hat{v}_b(x + \hat{u}_f(x,y), y + \hat{v}_f(x,y))\right)^2 \\ \hat{c}_i &\leftarrow \exp\left(- \frac{a_i}{\sigma_{sym}^2} \right) \hat{c}_i \end{aligned} \tag{6}$$

Where $\sigma_{sym} = 4$. The asymmetry measure a_i is squared Euclidean distance between the flow at pixel i and the the negative backward flow at the pixel in the alternate image that i maps to according to its estimated flow. This same update can also be applied to the "backward" flow.