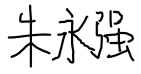
Optical Flow Computation for 360° Images

Yongqiang Zhu

MSc Digital Entertainment
The University of Bath
October 2020

This dissertation may be made available for consultation within the University Library and may be photocopied or lent to other libraries for the purposes of consultation.

Signed: 

Optical Flow Computation for 360° Images

submitted by

Yongqiang Zhu

for the degree of MSc Digital Entertainment of the

University of Bath

October 2020

DECLARATION

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of MSc Digital Entertainment in the Department of Computer Science. No portion of the work in this thesis has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Signature of Author 朱永强

Yongqiang Zhu

Abstract

Virtual Reality is becoming more popular in recent years, so more researches have been made for 360° images. The computation of 360° optical flow has not been focused even though some researchers have used them for their researches. In this thesis, we present a 360° optical flow computation method, using Icosahedron projection for 360° images. We have also implemented 2 existing methods for comparison. We use a 360° image dataset with ground truth flows, to compare the performance for the 3 approaches. We use 3 common optical flow metrics: Endpoint Error, Angular Error and Frame Interpolation Error, and we also propose 2 new metrics for evaluating 360° flows: Great Circle Distance and Spherical Angular Error. We show that our approach can provide more accurate 360° flows, and the computation of 360° flows is worth digging deeper in the future.

Contents

1 Introduction **8**

2 Literature and Technology Review **11**

2.1 Optical Flow 11

2.1.1 Traditional Optical Flow Algorithm 12

2.1.2 Optical Flow Algorithms with Convolutional Neural Network 14

2.2 Panoramic Image Projection 15

2.2.1 Equirectangular Projection 16

2.2.2 Cubemap Projection 17

2.2.3 Icosahedron Projection 18

2.2.4 Continuous Octahedron Projection 19

2.2.5 Method Discussions 19

3 Design and Approach **21**

3.1 Equirectangular Projection 22

3.2 Cubemap Projection 22

3.3 Icosahedron Projection 25

4 Implementation **27**

4.1 Implementation for Equirectangular Projection 27

4.2 Implementation for Cubemap Projection 27

4.2.1 Convert an Equirectangular Images to 20 Cubemap Images 28

4.2.2 Cubemap Face Number Classification 29

4.2.3 Merge 6 Cubemap Flows to Equirectangular Format . . 30

4.3 Implementation for Icosahedron Projection 31

4.3.1 Convert an Equirectangular Image to 20 Icosahedron Images 32

4.3.2 Triangle Number Classification 32

4.3.3 Merge 20 Icosahedron Flows to Equirectangular Format 33

4.4 Image Padding 33

4.4.1 Padding for Equirectangular Images 34

4.4.2 Padding for Cubemap Images 34

4.4.3 Padding for Icosahedron Images 35

4.5 Conversion Between Coordinate Systems 35

4.5.1 Equirectangule and Spherical 36

4.5.2 Spherical and World Cartesian 37

4.5.3 World Cartesian and Cubemap 38

4.5.4 World Cartesian and Icosahedron 38

4.5.5 Other Conversions 39

4.6 Relevant Techniques 39

4.6.1 Bilinear Interpolation 39

4.6.2 Barycentric Coordinate 40

5 Testing and Evaluation **41**

5.1 Testing methodology 41

5.1.1 Endpoint Error (EPE) 41

5.1.2 Angular Error (AE) 42

5.1.3 Great Circle Distance (GCD) 42

5.1.4 Spherical Angular Error (SAE) 43

5.1.5 Frame Interpolation Error (FIE) 44

5.2 Testing dataset 44

5.3 Result evaluation 46

5.3.1 Flow Algorithm Mode 46

5.3.2 Decision on Image Padding 46

5.3.3 Result Analysis 47

6 Conclusion **51**

6.1 Summary 51

6.2 Future Work 52

.1 Appendix A 56

Chapter 1

Introduction

Panorama is a type of image which processes several images as input and combines them to create a wider view of a scene. Image stitching is a common approach to create panorama since it has become a camera feature in modern smartphones. However, when the panorama is displayed using a Virtual Reality (VR) headset, it would look like a picture in the frame, rather than a recreation of the scene. If the goal is to convince users that they are virtually in the scene, this is far from immersive. The main reason is that while stitching multiple flat images together can give much wider view, it does not provide any depth information, making it unconvincing.

Bertel et al. [2019] has presented MegaParallax, a method for creating a 360° panorama scene using a sequence of images following a circular trajectory. The result scene not only has binocular disparity, i.e. slightly different views for left eye and right eye, but also provide motion parallax, i.e. change in position of viewpoint leads to change in view, making it much more convincing than conventional panorama in VR. The sequence of images are needed for optical flow computation to find objects, but there is no professional requirement for the images, so they can be captured using consumer level devices such as modern smartphones, making it approachable to many people. However there is a limitation with that: the field of view is limited, the views from above and below are generally not captured, so the result VR render would be lacking the sky and the ground.

This limitation could be resolved by using 360° images, i.e. spherical images, since they cover much bigger field of view. However, another question is raised: how to compute optical flows for 360° images? Optical flow is a term in visual computing which describes the relative motions occur between two images or video frames. Not only MegaParallax may be beneficial from using 360° flows, but also more researchers are looking into 360° image processing [Lee et al., 2018, Zhang et al., 2019] due to the advancement of VR in the recent years, the methods for computing 360° optical flow should be explored. Although there are many existing flow computation algorithms, like listed in the Middlebury benchmark[1] [Baker et al., 2011], they are all developed for images in the conventional planar format. Some researchers have computed flows for 360° images by treating the Equirectangular images like normal planar images and directly using existing flow algorithms [Kang and Cho, 2019, Lebreton et al., 2018, Matzen et al., 2017, Zhang et al., 2020]. Besides using Equirectangular images directly, Zhang et al. [2020] have also tried computing 360° flows using Cubemap projection, and they reported that Equirectangular projection performed better than Cubemap projection for their specific dataset. However, all Equirectangular projection images suffer from great distortion as image contents are heavily stretched out at the pole areas, which may seriously impact the accuracies of the computed 360° flows. Besides content distortion due to image projection, there is another challenge for computing 360° flows. Unlike conventional flows where they are all observed inside the images, since spherical images are self-connected, flows may be lost due to motions going from one face to another for multiple faces projection, or going from one end to the other end for single face projection, e.g. going from the rightmost to the leftmost in Equirectangular projection.

In this thesis, we present a method for computing 360° flows by using Icosahedron projection for 360° images. Optical flow is computed individually for each Icosahedron face, and the flows are stitched back together in the end to create a complete 360° flow. Since Icosahedron works well for approximating sphere, the projection could cause less distortion and so the projected image contents are more accurate. We also attempt to tackle the issue of motions moving across faces, by implementing image padding into the method. We select a few sets of synthetic images as our testing dataset, and we use some

[1] https://vision.middlebury.edu/flow/eval/

common optical flow metrics such as Endpoint Error (EPE), Angular Error (AE) and Frame Interpolation Error (FIE) [Baker et al., 2011] to compare the performance of our method and some existing approaches. Given that 360° images are different from conventional images, we have also developed two other metrics which are specific to evaluating 360° optical flows: Great Circle Distance (GCD) and Spherical Angular Error (SAE). The test results show that our method generally performs better than other tested approaches, but also reveal some limitations. Our method could fall behind in computing large flows due to its fragmented projection approach, but this also gives a possible direction for future researches.

This report is structured as follow: Chapter 2 explains what optical flow in general is, and reviews some traditional optical flow algorithms and some more state-of-the-art algorithms, as well as some projection methods for spherical 360° images. Chapter 3 gives a brief overview of the working pipeline in this thesis, and more detailed pipelines for different approaches. Chapter 4 shows how the pipelines are being implemented, and explains details of some visual computing techniques. Chapter 5 uses some testing results to explain how some decisions are made, and evaluate the performance of our proposed method and other approaches. Chapter 6 concludes this report and looks into some possibilities for future researches.

Chapter 2

Literature and Technology Review

In this chapter, we have a look at what optical flow in general is, how traditional algorithms compute it, and what are the state-of-the-art approaches. Moreover, we look at a few projection methods for 360° panoramic image, such as Equirectangle and Cubemap etc.

2.1 Optical Flow

In visual computing, the term optical flow is describing the relative motion of an object or a scene when compared to another state of itself, such as two image frames in a video sequence. Since it is a measurement of relative motion, generally in an image frame, the object or the scene does not necessarily need to be moving for observing optical flow, and vice versa. It is about the variation in visible pixel positioning, so if a video contains a plain colour circle with self rotation, and the background does not change, then even though the circle is moving it would still appear as stationary, so no flow can be observed. On the other hand, if there are two image frames with the same content, except the second frame has a pixel translated to the right, then optical flow could be observed.

The concept of optical flow can serve many purposes in computer vision. In a video sequence or real-time camera footage if an object has relative motion to the scene, then its flow may be found while moving, therefore optical flow can be used to track objects [Yamamoto et al., 1995]. Objects inside the same scene or image may all move independently and differently, therefore images or scenes can be segmented by grouping nearby flows with similar directions [Klappstein et al., 2009].

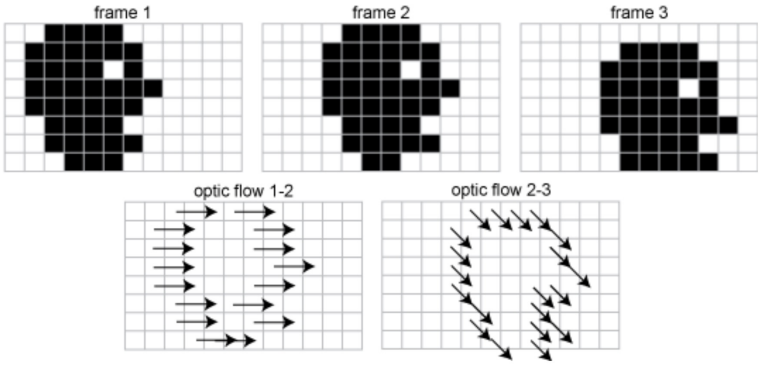


Figure 2-1: Simple optical flow visualisation using pixel images [Raudies, 2013].

Figure 2-1 gives a clear demonstration of how optical flow vectors can be represented. Frame 1, frame 2 and 3 are the same in terms of content; however, the object in black is positioned slightly differently in the three frames. This causes optical flow because of the pixelwise translation. Frame 2 has moved to the right from frame 1, hence the flow vectors 1-2 are pointing to the right. Similarly, frame 3 has moved right and down from the second frame, therefore all flows point toward the bottom right. However, real images and scenes are generally much more complicated than Figure 2-1, so researchers attempt to develop algorithms for approximating optical flows.

2.1.1 Traditional Optical Flow Algorithm

Horn-Schunck [Horn and Schunck, 1981] and Lucas-Kanade [Lucas and Kanade, 1981] optical flow estimations are the earliest algorithms for approximating optical flow. Over the past few dozen years, researchers have developed other methods for flow computation. Lucas-Kanade focuses on the local window,

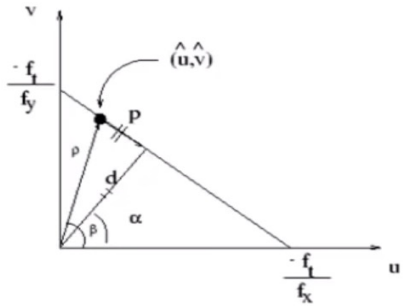


Figure 2-2: Plotting the line equation derived from brightness consistency [Horn and Schunck, 1981].

whereas Horn-Schunck views input images in a global sense, they have distinct advantages over each other. Researchers have also developed algorithms which combine the characteristics from both [Bruhn et al., 2005].

Horn & Schunck

Horn-Schunck optical flow method is one of the earliest algorithms for approximating flows between images [Horn and Schunck, 1981]. It is computed base on an assumption of brightness consistency between images as shown in Equation (2.1), and there is smoothness constraint. Horn-Schunck optical flow is computed using Equation (2.2), the first part of the integral represents the brightness consistency constraint and the second part represents the smoothness constraint. The flows along the horizontal and vertical direction are u and v respectively, x and y are the dimensions of a 2D frame, and t represents the order of the frame. Figure 2-2 plots the brightness consistency equation. Optical flow vectors can be efficiently obtained by approximating the function result iteratively until it converges.

$$f_x u + f_y v + f_t = 0 \tag{2.1}$$

$$\int\int \{(f_x u + f_y v + f_t) + \lambda(u_x^2 + u_y^2 + v_x^2 + v_y^2)\} dxdy \tag{2.2}$$

Lucas & Kanade

Lucas-Kanade [Lucas and Kanade, 1981] is a different approach for estimating optical flow compare to Horn-Schunck as it focuses on local flows rather than dense flows. Based on the same brightness consistency assumption in Equation (2.1), Lucas-Kanade creates a mask, i.e. a window around a pixel. Since every pixel is bounded in more than 1 window, there are enough constraints to estimate its u and v flows. Lucas-Kanade also requires the assumption of local smoothness, where all flows within a window are roughly equal. Equation (2.3) and Equation (2.4) calculates u and v respectively.

$$u = \frac{\sum f_x f_y \sum f_y f_t - \sum f_y^2 \sum f_x f_t}{\sum f_x^2 \sum f_y^2 - (\sum f_x f_y)^2} \tag{2.3}$$

$$v = \frac{\sum f_x f_t \sum f_x f_y - \sum f_x^2 \sum f_y f_t}{\sum f_x^2 \sum f_y^2 - (\sum f_x f_y)^2} \tag{2.4}$$

2.1.2 Optical Flow Algorithms with Convolutional Neural Network

Since the concepts of deep learning and neural network have been introduced and become more and more popular, researchers then have developed methods for flow computation using those techniques [Dosovitskiy et al., 2015, Revaud et al., 2015]. In fact, according to the flow benchmark databases such as Middlebury[Baker et al., 2011], KITTI[Geiger et al., 2012], MPI Sintel[Butler et al., 2012] and HD1K[Kondermann et al., 2016], more than a hundred optical flow algorithms have now been developed.

DeepFlow

Revaud et al. [2015] has purposed a flow algorithm called DeepFlow, which integrates some Convolutional Neural Network (CNN) training into traditional optimisation based flow methods. They have trained a CNN called DeepMatching for matching features between images, which can even be used on non-rigid deformed objects or objects with repeating textures. The CNN is

also scale and rotation invariant. Since feature matching plays an important role in optical flow computation, they have used DeepMatching as the matching mechanism for DeepFlow. They use Equation (2.5) as their optimisation function, where E_D denotes the constancy for colour and gradient in images, smoothness E_S denotes the norm of the gradient flow and E_M denotes feature matching, and this is where DeepMatching is integrated. Their benchmark results have shown that DeepFlow can match the performance with other state-of-the-art optical flow methods.

$$E(\boldsymbol{\omega}) = \int_{\Omega} E_D + \alpha E_S + \beta E_M \boldsymbol{dx} \tag{2.5}$$

FlowNet

FlowNet is another optical flow method using CNN, and it is proposed by Dosovitskiy et al. [2015]. FlowNet has 2 different CNN approaches to flow computation. The first one is FlowNetSimple, where a generic network with only convolution layers is used to process the image pair. The second one is FlowNetCorr, which starts by processing the image pair separately using the same network stream to generate two feature maps, then uses a dedicated correlation layer for matching feature vectors of the two maps. They have combined unpooling and convolution to create upconvolutional layers, to expand the feature maps and refine into higher resolution. The CNN method has obtained competitive results in optical flow benchmarks, and their approach has shown that it is feasible to compute flow by training CNN end-to-end.

2.2 Panoramic Image Projection

360° images or videos are viewed on a spherical shell in the 3D space. With this display method, it can easily be adapted into Virtual Reality. However, in general images and videos are processed on a 2D plane, it is difficult to do editing on a VR shell. Therefore to process this type of image or video, it would require to unwrap the spherical shell onto a 2D surface, mostly on rectangular planes. The process of unwrapping is called projection, and due to different needs for image processing or computation simplicity, many different

projection methods have been developed to serve the specific requirements, i.e. a 360° image can be segmented in many ways and each of them could self-fit onto different 2D planes while retaining the image details.

2.2.1 Equirectangular Projection

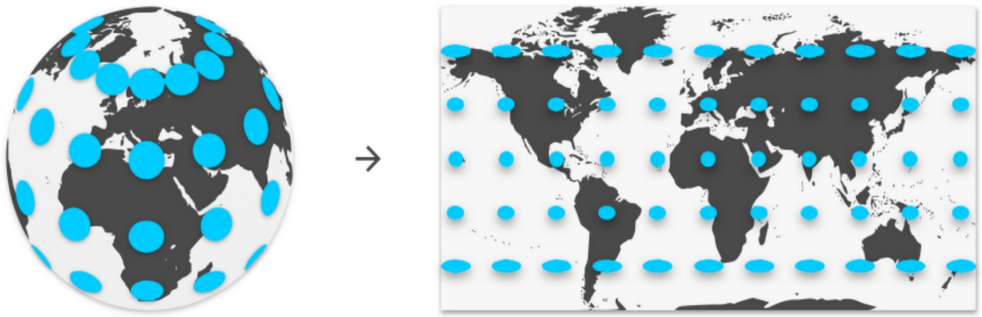


Figure 2-3: Flatten the Earth onto a 2D rectangular frame [Brown, 2017]. All blue circles cover the same amount of Earth area, so they are the size on the sphere. However, Equirectangular projection stretches the pole areas a lot more than the equator, so blue circles at the pole areas of the Equirectangular image look different from the circles near the equator.

One of the most popular projection methods is called Equirectangular projection. This method was not born in the filed of visual computing, but rather a well known object, the world map [Snyder, 1997]. Since the earth is a sphere roughly speaking, by projecting every circle of latitude to a horizontal line with identical length on a flat surface, the earth can be flatten and projected onto a 2D rectangular map. The left edge is connected to the right edge in terms of content, to retain the close loop. This idea can be visualised in Figure 2-3. This concept was soon being used in computer vision for image projection. Output images and videos from consumer 360° cameras like Insta360 One X are represented in this Equirectangular projection format.

However there are problems worth paying attention to. On a hemisphere the magnitude of every circle of latitude is different from each other, but when projected onto the 2D map they have equal length, it means that the pole and the equator, which is a single point and the longest circumference on a

Figure 2-4: A 360° image with Equirectangular projection.

sphere respectively, are reserved into the same size in space. The middle line contains details of the equator, whereas every pixel on the top line represents the same pole point on the sphere. Which in a sense pixel density for this projection method is not consistent along the vertical direction at all. The further away from the equator, the more expanded it gets when projecting onto 2D rectangular frame, therefore the image would be very distorted. Figure 2-4 is an example of 360° image represented in Equirectangular projection.

2.2.2 Cubemap Projection

Figure 2-5: A 360° image with Cubemap projection.

Cubemap is another method of projection which has been widely used. This idea was first addressed to environment mapping in computer graphics

[Greene, 1986]. The concept of this project is simple: first place the 360° image sphere into a bounding box, in this case a cube, then every face of the cube act as a mirror, reflects a part of the surface on the sphere. The cube would be unfolded and lay on the 2D plane with 6 squares. 6 faces together would contain all the details about the 360° image. Figure 2-5 is an example of such projection method, and it contains the same content as Figure 2-4. This projection method is frequently used in video games, and it has been implemented in game engines such as Unity and Unreal Engine.

There is a lot less image distortion in the Cubemap projection if we compare Figure 2-5 and Figure 2-4. This is an improvement over the Equirectangular projection. However, the problem about pixel density consistency still remains. Since every face is mirroring off a part of the sphere, the centre of a face is closer, and the edges are further away from the sphere surface. Therefore pixel density is not uniform on a Cubemap face.

2.2.3 Icosahedron Projection

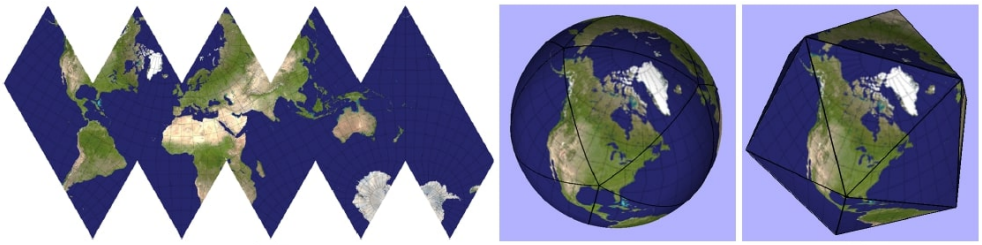


Figure 2-6: Icosahedron projection for the world map. a) Unwrapped Icosahedron world map; b) Spherical world map; c) Icosahedron world map.

Icosahedron projection has also been used for projecting the world map [Fisher, 1943], the projection result can be seen in Figure 2-6[1]. A great benefit of Icosahedron projection is that it can discretize a given sphere uniformly and accurately. Therefore researchers have been using it to train convolutional neural networks for spherical i.e.360° image data [Lee et al., 2018, Zhang et al., 2019].

[1] https://worldbuilding.stackexchange.com/questions/109633/how-to-generate-world-map-for-minimal-distortion

2.2.4 Continuous Octahedron Projection

Figure 2-7: A 360° image with continuous octahedron projection [Pei et al., 2018].

Recently a new projection method called Continuous Octahedron Projection has been introduced [Pei et al., 2018]. Figure 2-7 is an example of such image projection. The left half is a square made of 4 triangles, which are the four faces on the top half of an octahedron, and they contains the image details of the upper hemisphere of the 360° image. The right half is a similar case but contains the lower hemisphere of the 360° image. It is suggested that this representation would have relatively uniform pixel density distribution. With the left half actually connected to the right half in terms of content, there is no internal discontinuity for this type of projection.

2.2.5 Method Discussions

There are more projection methods such as the general Octahedron projection [Praun and Hoppe, 2003], and the Equi-Angular Cubemap projection [Brown, 2017]. Pei et al. [2018] compared different image projection methods and created a chart in terms of pixel density and internal boundary discontinuity, as shown in Figure 2-8. Green colour means the pixel density distribution is sparse, whereas warm colours like yellow and oranges mean image contents are more densely projected.

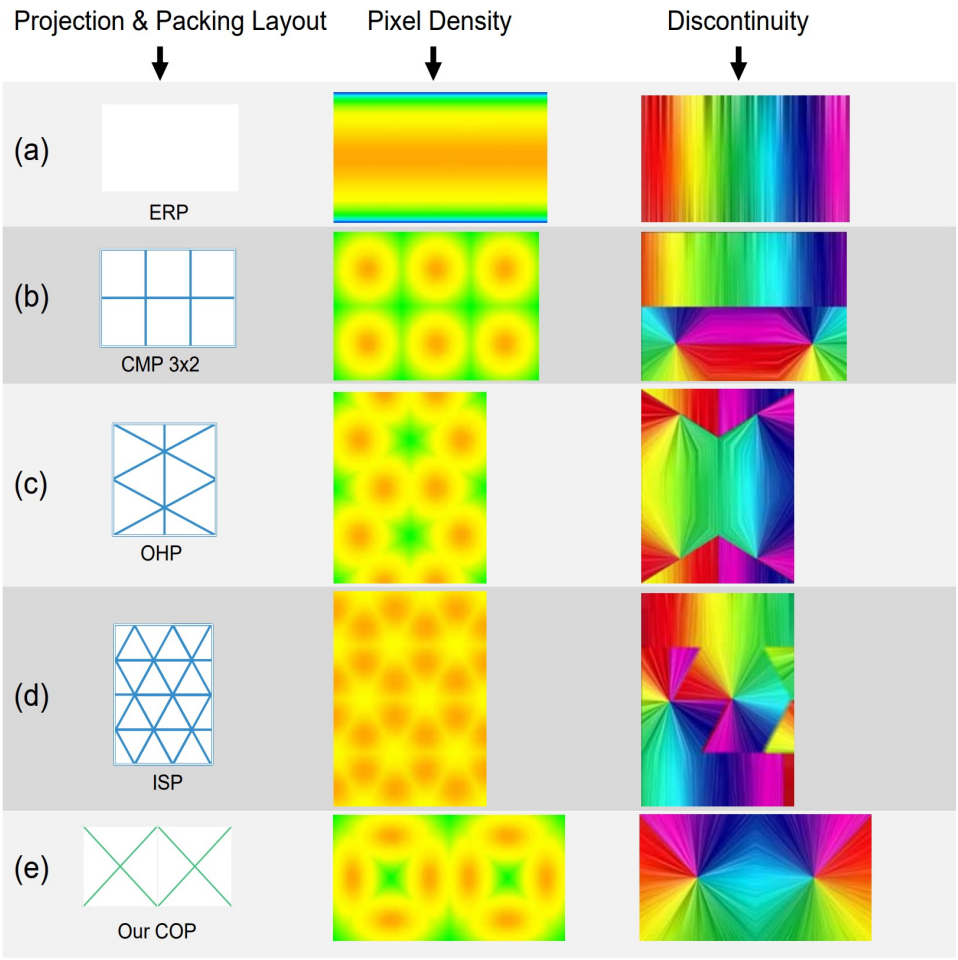


Figure 2-8: A comparison of different 360° image projection representation in terms of pixel density and internal discontinuity: (a) Equirectangular projection; (b) Standard Cubemap reshapes into 2 by 3 layout; (c) Octahedron projection; (d) Icosahedron projection; (e) Continuous octahedron projection [Pei et al., 2018].

Chapter 3

Design and Approach

Figure 3-1 has demonstrated the pipeline for 360° images flow computation in this thesis. Both the input images and the output flow are in the Equirectangular format since this format is one of the most widely used projection for 360° images [Zhang et al., 2020, Kang and Cho, 2019, Lebreton et al., 2018, Matzen et al., 2017]. The input can be processed using different approaches such as converting into different projections for flow computation.

In this chapter the pipeline details for Equirectangular, Cubemap and Icosahedron projections are presented. Equirectangular projection image requires no transformation, optical flow algorithms can process it directly, therefore it is time efficient to work with. However its performance may be impacted

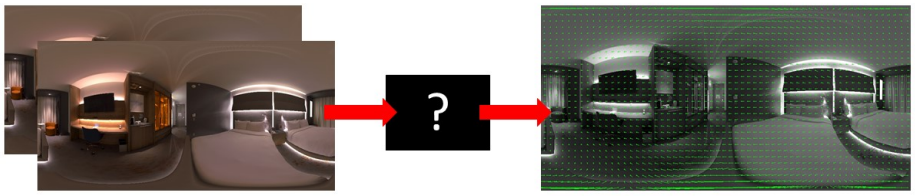


Figure 3-1: Optical flow computation pipeline for 360° images: on the left input two Equirectangular images, then in the middle flows are computed using various methods, on the right output a flow with Equirectangular format.

by the great distortion at the pole areas. Cubemap is also a popular projection for 360° images in general, and it can project the pole with much less distortion, but since Cubemap has 6 faces, it requires extra time for image transformations and flow stitching, the flow computation process is slower. Icosahedron projection has even more faces so it is the slowest projection of the three to work with. On the other hand, Icosahedron projection can accurately approximate the original sphere, resulting in minimal distortion for the projection.

The computed 360° flows will be compared with the ground truth flow, and the following metrics will be used for evaluating the performance for each approach: Endpoint Error (EPE), Angular Error (AE), Great Circle Distance (GCD), Spherical Angular Error (SAE) and Frame Interpolation Error (FIE). Evaluation results of the Equirectangular projection act as a baseline, and they are used to compare with evaluation results of the other two approaches.

3.1 Equirectangular Projection

Equirectangular projection projects a 360° image onto rectangle frame with some distortion in content, which means conventional flow algorithms can directly process Equirectangular images as input. However flow algorithms are designed to work with conventional images, they do not know there is content stretching at the pole areas in an Equirectangular image, and they are not aware of the wraparound format of the image. Accuracy of the result 360° flow could be heavily impacted by these 2 issues. Figure 3-2 shows the rundown of how flow from Equirectangular projection can be computed.

3.2 Cubemap Projection

Figure 3-3 is the rundown of how flow from Cubemap projection is computed. The input Equirectangular images is projected onto the 6 faces of the Cubemap, each is seen as an individual rectangular sub-image. It requires more processing effort than using Equirectangular projection directly, but content distortion at the pole areas is significantly less. Optical flow algorithm pro-

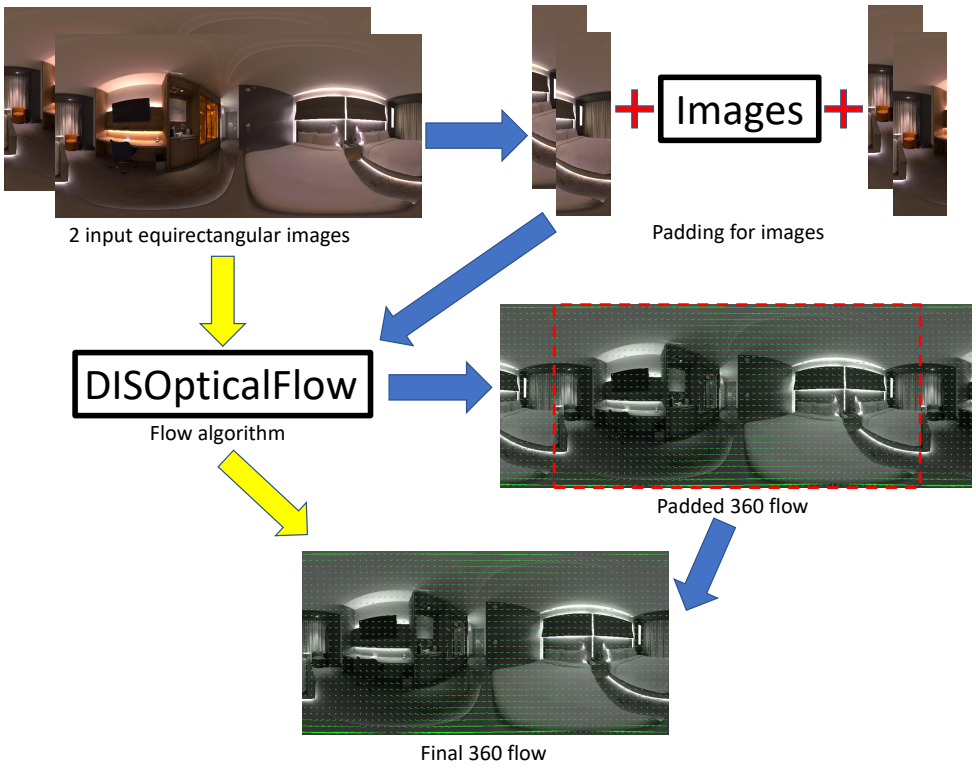


Figure 3-2: Implementation pipeline for Equirectangular projection. Input images can be processed directly by the flow algorithm, or being padded first. If the images are padded then the computed flow would need to be cropped to the same dimensions as the input images.

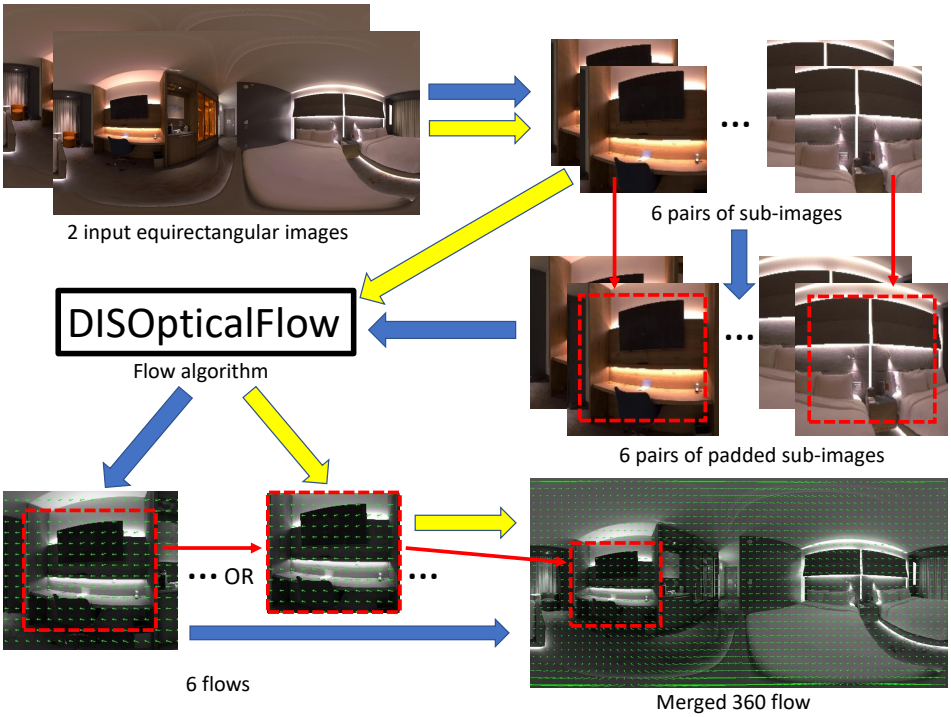


Figure 3-3: Implementation pipeline for Cubemap projection. Input images are converted into 6 pairs of image, each refer to a face on the Cubemap. Images can be padded or not padded before running flow algorithm. Flows will be computed separately for the 6 sets, and since flows are interpolated in the end, padded flows do not need to be cropped. In the end they are merged back into one flow, in the Equirectangular format.

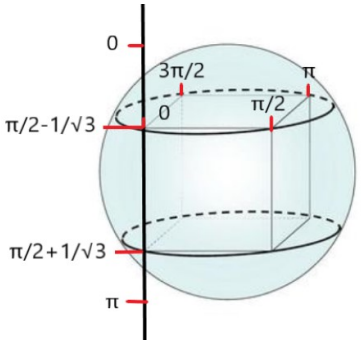


Figure 3-4: Longitude and latitude values for vertices in a Cubemap

cesses the sub-images directly to compute sub-flows, or pad the images before processing. The 6 sub-flows are interpolated back into the Equirectangular format. Cubemap projection has 8 unique vertices, the vertex longitude and latitude values are shown in Figure 3-4.

3.3 Icosahedron Projection

Figure 3-5 shows the rundown of how flow from Icosahedron projection is computed. The input Equirectangular images are projected onto the 20 faces of the Icosahedron. Since every Icosahedron face is a triangle, it needs image padding to form a rectangular sub-image, so that it can be processed by the optical flow algorithm. 20 sub-flows are computed from the 20 pairs of sub-image, then they are interpolated back into the Equirectangular format. This fragmented projection approach is twofold: it suffers the least from image content distortion, but its overall processing time is the longest. It could also miss out long flow displacement.

The Icosahedron can be unwrapped as shown in Figure 3-6. Each vertex is labelled and its corresponding spherical coordinate can refer to the axis values. All vertices from $NP1$ to $NP5$ are the north pole on the sphere, similarly all vertices from $SP1$ to $SP5$ are the south pole. Pole points can rotate around itself without losing their positions, so they can have multiple latitudes. Therefore technically there are 12 distinct vertices in the Icosahedron projection.

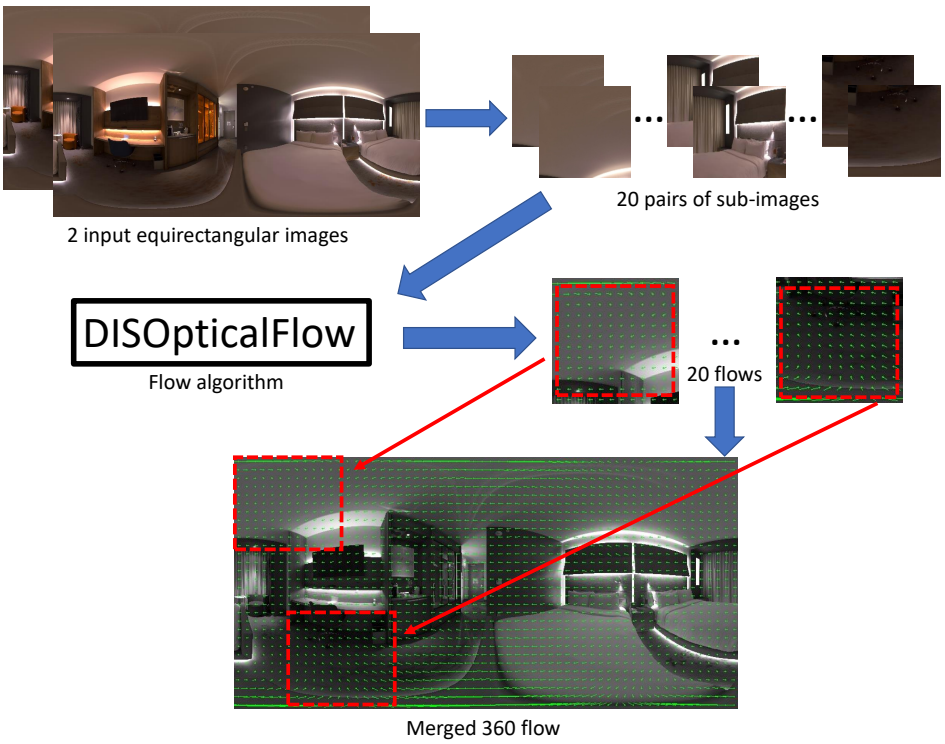


Figure 3-5: Implementation pipeline for Icosahedron projection. Input images are converted into 20 pairs of image, each refer to a face on the Icosahedron. Flows are computed separately for the 20 image pairs, and in the end they are merged back into one flow, in the Equirectangular format.

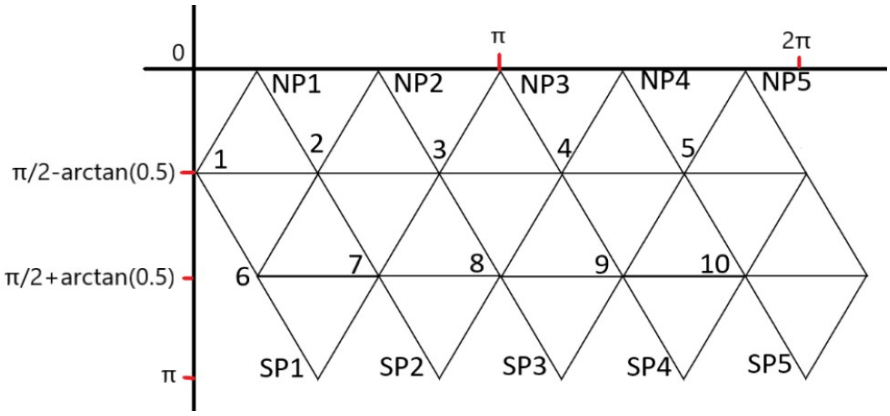


Figure 3-6: The triangle layout for an Icosahedron: longitude and latitude values for vertices.

Chapter 4

Implementation

In this chapter, we first look at the implementation details of the Equirectangular, Cubemap and Icosahedron flow pipelines, and how images are padded for each projection method. We also explain the details of some visual computing techniques used in the implementations such as bilinear interpolation, barycentric coordinate and coordinate conversion.

4.1 Implementation for Equirectangular Projection

We only need to implement image padding for Equirectangular images since they can be directly processed by optical flow algorithms. The details for image padding will be presented later in this chapter.

4.2 Implementation for Cubemap Projection

In this section we show the implementation details of the Cubemap pipeline, which includes projecting Equirectangular images onto 6 Cubemap faces, and stitching 6 Cubemap flows back into the Equirectangular format.

4.2.1 Convert an Equirectangular Images to 20 Cubemap Images

The process of converting an Equirectangular image to 6 Cubemap images requires finding corresponding Equirectangular coordinate for every pixel on every Cubemap image. Once every pixel on every Cubemap image finds a Equirectangular coordinate, it can find its RGB or grayscale value from the Equirectangular image, using bilinear interpolation. Then the 6 Cubemap images can be created using all the interpolated RGB or grayscale values.

Every Cubemap face can be seen as 2 triangles instead of 1 square like shown in Figure 4-2, and all Cubemap vertices have coordinate values predefined, so we could use their Cubemap coordinates to find barycentric coordinates for every pixel in every Cubemap image. After obtaining the barycentric coordinate for a pixel, its world Cartesian coordinate can be calculated using the world Cartesian coordinates of the three corresponding triangle corners and its barycentric coordinate.

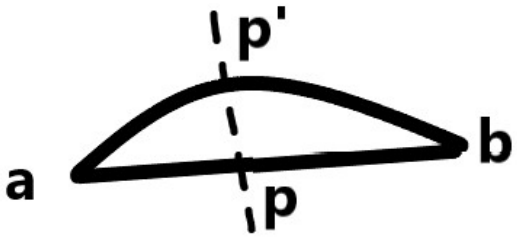


Figure 4-1: A cross-section view of the sphere with the triangle plane.

However this computed world Cartesian coordinate cannot be converted into Equirectangular coordinate directly. Figure 4-1 has demonstrated the issue with that. In the image $\boldsymbol{a}$ and $\boldsymbol{b}$ are two of the triangle corners, and $\boldsymbol{p}$ is the computed point. Apart from the three triangle corners, all other points on the triangle are not on the sphere surface. Therefore Equation (4.1) is needed for finding its corresponding point $\boldsymbol{p'}$ on the sphere surface.

$$\boldsymbol{p'} = \frac{\boldsymbol{p}}{\|\boldsymbol{p}\|} \tag{4.1}$$

Once $\boldsymbol{p'}$ is calculated it can be used to obtain the corresponding Equirectangular coordinate using world Cartesian to spherical and spherical to Equirectangular coordinate conversions. As soon as every pixel on a Cubemap image finds its Equirectangular coordinate, its RGB or grayscale value can be find from the Equirectangular image, using bilinear interpolation. A Cubemap image is created using those interpolated pixel values. All result images look similar to Figure 4-7.

4.2.2 Cubemap Face Number Classification

Every vertex on the Cubemap has pre-defined coordinate values. Therefore the face number for each pixel on the Equirectangular image can be obtained using those pre-defined values. Every face has 4 corners, only 3 are required to form a triangle like shown in Figure 4-2. The triangle has 3 corner vertices $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$ in world Cartesian coordinate, they form a unique plane in the 3D space. Every two corner vertices form a directional vector on the plane, and the normal $\boldsymbol{n}$ can be calculated as the cross product of two directional vector, this is expressed in Equation (4.2).

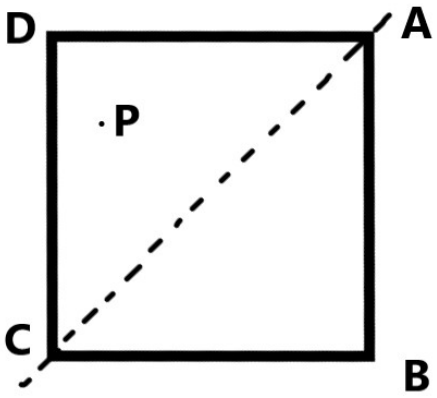


Figure 4-2: Cubemap image visualisation

$$\boldsymbol{n} = (\boldsymbol{b} - \boldsymbol{a}) \times (\boldsymbol{c} - \boldsymbol{a}) \tag{4.2}$$

Assume that we want to classify pixel $\boldsymbol{p}$, in the 3D space we can form a line by connecting it with the sphere centre $(0, 0, 0)$. Every point $\boldsymbol{l}$ on the line

can be calculated using Equation (4.3), where d is a scalar value.

$$\boldsymbol{l} = \boldsymbol{p} \times (d + 1) \tag{4.3}$$

$$d = \frac{(\boldsymbol{a} - \boldsymbol{p}) \cdot \boldsymbol{n}}{\boldsymbol{n} \cdot \boldsymbol{p}} \tag{4.4}$$

The scalar d of the intersection between that line and the triangle plane can be calculated using Equation (4.4), if $\boldsymbol{n} \cdot \boldsymbol{p} = 0$ then there is no intersection, otherwise the intersection coordinate can be calculated using Equation (4.3). However, the classification process does not stop here. The line could intersect with the plane outside the triangle area, or the intersected triangle plane is on the different hemisphere from $\boldsymbol{p}$. These situations need to be handled. The maximum distance between the triangle and the sphere centre would be the radius of the sphere since the 3 triangle corner vertices are on the sphere surface. If the magnitude of $\boldsymbol{l}$ is greater than that, the intersection definitely does not happen inside the triangle. We can also use the barycentric coordinate $\boldsymbol{\lambda}$ to identify whether the intersection point lies within the triangle. The condition will be true if and only if all λ_a, λ_b and λ_c are between 0 and 1.

The top image in Figure 4-3 shows the result for running the classification function. Pixels with the same face number are painted with the same colour. However, there are some black dots in the circled areas, and they are unclassified pixels. Each blank pixel is classified using a 3 by 3 window: given a pixel (r, c) on the Equirectangular image, there are 8 neighbouring pixels around it in a 3 by 3 window, the blank pixel will have the same number as the most occurring triangle number in the window. This 3 by 3 window can be visualised in Figure 4-4. The bottom image in Figure 4-3 shows the final classification result.

4.2.3 Merge 6 Cubemap Flows to Equirectangular Format

Like mentioned previously, a pixel position (r, c) on the Equirectangular image has a corresponding triangle on a Cubemap face, and we are able to calculate

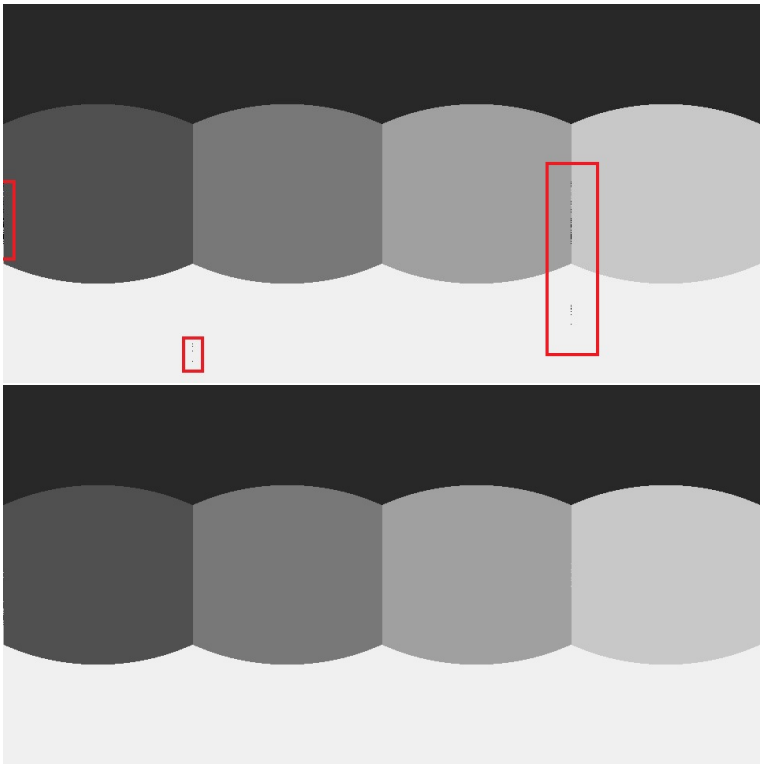


Figure 4-3: Cubemap face number classification in the Equirectangular image. Top: classification function leaves some blank pixels, i.e. black dots in the red circled areas; Bottom: all blank pixels are filled using a 3 by 3 window.

the intersection point on the triangle plane in the 3D space. By applying world Cartesian to Icosahedron coordinate conversion, we can obtain its corresponding Cubemap coordinate (j, i). Since most of the time the calculated Cubemap coordinate is not integer coordinate, bilinear interpolation function is used to find its 4 neighbouring coordinate locations and their corresponding bilinear interpolation coefficients. Then these interpolation details are used to merge 6 Cubemap sub-flows to a flow in the Equirectangular format.

4.3 Implementation for Icosahedron Projection

In this section we show the implementation details of the Icosahedron pipeline, which includes projecting Equirectangular images onto 20 Icosahedron faces,

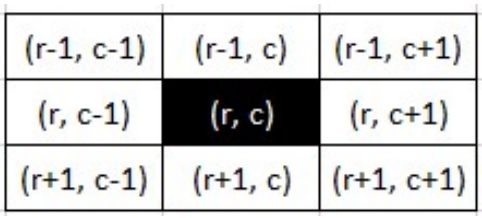


Figure 4-4: Blank pixel in the middle and its 8 neighbouring pixels in the 3 by 3 window.

and stitching 20 Icosahedron flows back into the Equirectangular format. The implementation is similar to Cubemap projection, so some details from Section 4.2 are not repeated.

4.3.1 Convert an Equirectangular Image to 20 Icosahedron Images

The same method in Section 4.2.1 can be used for converting an Equirectangular image into 20 Icosahedron images, since every Icosahedron face is a triangle. Once every pixel on every Icosahedron image finds an Equirectangular coordinate, it can find its RGB or grayscale value from the Equirectangular image, using bilinear interpolation. Then the 20 Icosahedron images can be created using all the interpolated RGB or grayscale values. All result images look similar to Figure 4-8.

4.3.2 Triangle Number Classification

Triangle number classification in Icosahedron projection is fundamentally the same as Section 4.2.2. There are 20 Icosahedron faces and every face is a triangle, with 3 corner vertices. All vertices have pre-defined coordinate values. The Icosahedron triangle number can be calculated and derived using Equation (4.2), Equation (4.3) and Equation (4.4). The top image in Figure 4-5 visualises the result of Icosahedron triangle number classification. There are still blank pixels so they are filled using the 3 by 3 neighbouring window method. The final result is the bottom image in Figure 4-5.

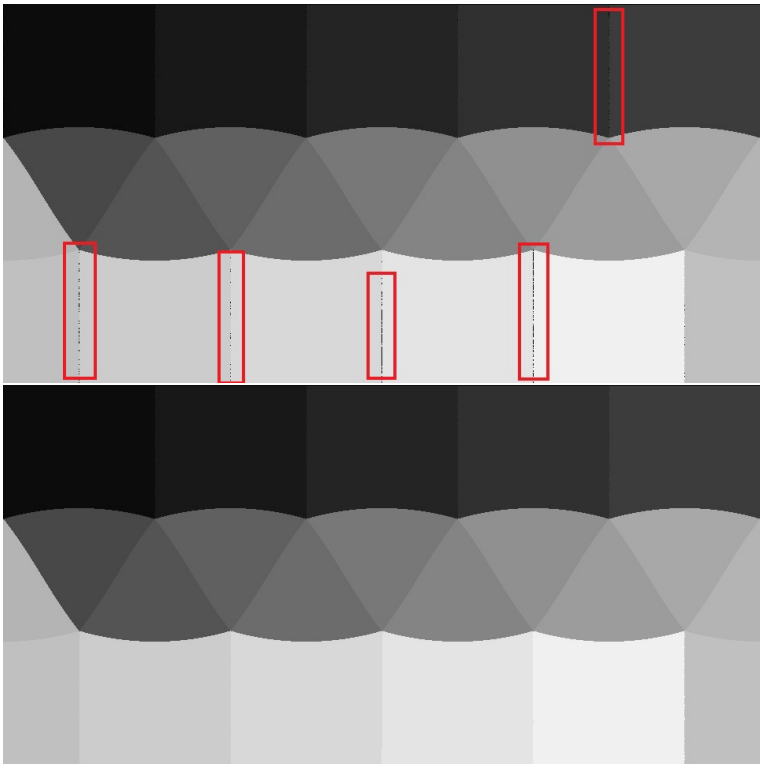


Figure 4-5: Icosahedron triangle number classification in the Equirectangular image. Top: classification function leaves some pixel blank i.e. black dots like shown in the circled areas; Bottom: blank pixels have been filled using a 3 by 3 window.

4.3.3 Merge 20 Icosahedron Flows to Equirectangular Format

Each pixel position (r, c) on the Equirectangular image has a Icosahedron triangle number, and the corresponding Icosahedron coordinate (j, i), like explained in Section 4.2.3. Then we can find the bilinear interpolation details, and use them to merge 20 Icosahedron sub-flows to a flow in Equirectangular format.

4.4 Image Padding

Some researchers pad images before computing optical flows for 360° images [Zhang et al., 2020, Kang and Cho, 2019]. Image padding provides overlay

areas between projection faces, so it helps to tackle the issue where motions could start at one face and end at another face on a multi-face projection such as Cubemap or Icosahedron. Motions on an Equirectangular image could start from the rightmost but end at the leftmost, due to its wraparound format, and image padding can avoid flow algorithms from missing out those motions. Images would need to be padded differently depending on the projection forms, therefore in this section we present the image padding details for Equirectangular, Cubemap and Icosahedron images.

4.4.1 Padding for Equirectangular Images

An Equirectangular image is self connected on its left and right boundaries. Therefore padding can be achieved by adding a section of the left pixels to the right and a section of the right pixels to the left, like shown in Figure 4-6.

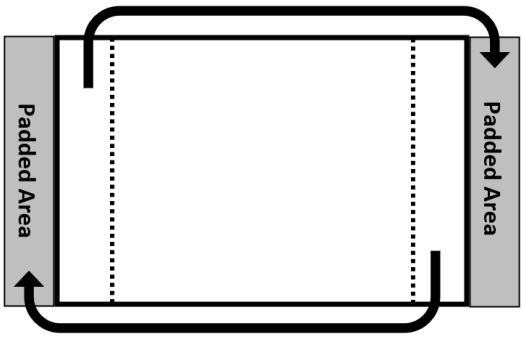


Figure 4-6: Padding for Equirectangular image. The white rectangle is the original image, the grey areas on the left and right are the padded areas. The arrows and the dotted lines explain where the padded contents come from.

4.4.2 Padding for Cubemap Images

Figure 4-7 demonstrates the padding for a Cubemap image. A Cubemap face has 4 corner vertices $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ and $\boldsymbol{D}$ with pre-defined coordinate values. We can ignore one of them and take the other 3, e.g. $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$ to form a triangle. For any pixel $\boldsymbol{P}$ on the padding image, its barycentric coordinate

regarding $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$ can be calculated. Then by coordinate conversion and bilinear interpolation, it can find its corresponding image value on the original Equirectangular image.

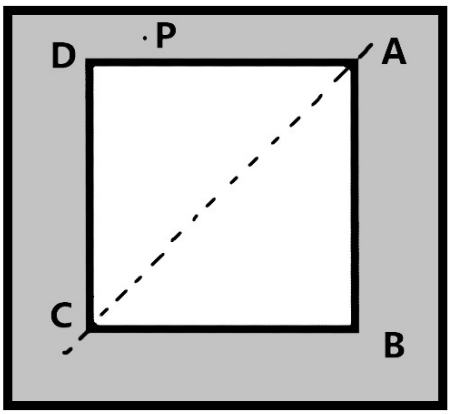


Figure 4-7: Padding for a Cubemap image. The middle white square is the Cubemap face, and the grey area is the padding.

4.4.3 Padding for Icosahedron Images

Since every Icosahedron face is a triangle, it always requires padding for general image processing, i.e. optical flow computation in this case. Padding of an Icosahedron image can be visualised in Figure 4-8. Since the coordinate values for the 3 triangle vertices $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$ are pre-defined, for any pixel $\boldsymbol{P}$ on the padding image, its barycentric coordiante $\boldsymbol{\lambda}$ can be calculated with respect to $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$. Pixels which are outside the triangle have values greater than 1 or less than 0 in its $\boldsymbol{\lambda}$, making its barycentric coordinate less meaningful, but the values can still be used to find its corresponding image value from the input Equirectangular image using bilinear interpolation.

4.5 Conversion Between Coordinate Systems

Several coordinate systems have been used to serve different purposes in the thesis. Equirectangular coordinate describes pixel location (r, c) in an Equirectangular image, where r is the vertical displacement and c is the hor-

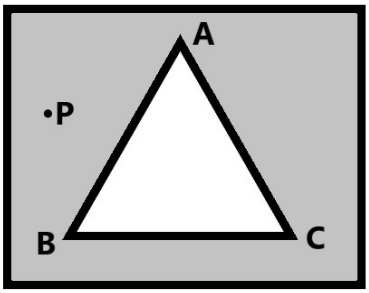


Figure 4-8: Padding for Icosahedron image. The middle white triangle is the Icosahedron face, and the grey area is the padding.

izontal displacement from the origin. Spherical coordinate uses θ and ψ to describe the latitude and longitude of a given point on the sphere in a 3D space. World Cartesian coordinate describes any point in the 3D space using (x, y, z). Icosahedron has 20 faces and Cubemap has 6 faces, but the coordinate system on each face has the same properties as Equirectangular coordinate, where (r, c) define the pixel location.

4.5.1 Equirectangule and Spherical

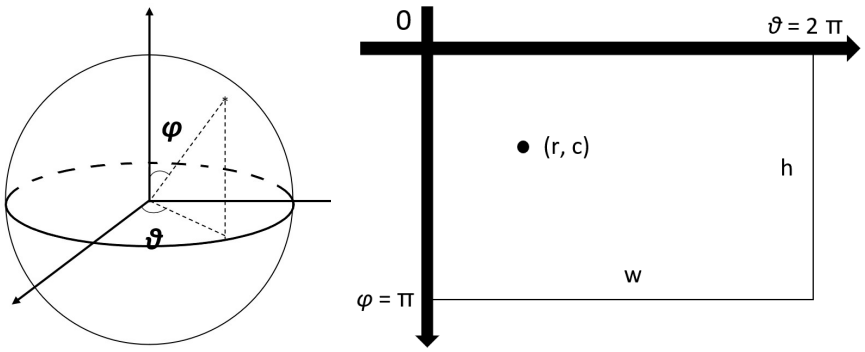


Figure 4-9: Spherical coordinate system on the left and Equirectangular coordinate system on the right.

Figure 4-9 and Equation (4.5) show the range of values for θ and ψ, where ψ is 0 at north pole and π at south pole, and θ increases along the anticlockwise direction from the top-down view.

$$\begin{aligned} 0 \leq \psi < \pi \\ 0 \leq \theta < 2\pi \end{aligned} \tag{4.5}$$

In order to maintain the consistency, the orientation of all Equirectangular images in the project has been defined as shown in Figure 4-9. The image has height h and width w, and for a point on the image it is defined by its row value r and column value c. Both r and c start at 0 from the top left corner, which can also refer to the north pole in Figure 4-9. When the row value is at maximum, its corresponding longitude ψ in the sphere will be at maximum. The same rule applies to column value and latitude θ.

Therefore the ratio between r and h will equal to the ratio between ψ and π, similarly the ratio between c and w will be the same as the ratio between θ and 2π. Coordinates can be converted between spherical coordinate and Equirectangular coordinate using these rules. Equation (4.6) converts from Equirectangular coordinate to spherical coordinate and Equation (4.7) converts from spherical coordinate to Equirectangular coordinate.

$$\theta = \frac{2\pi \times c}{w}, \psi = \frac{\pi \times r}{h} \tag{4.6}$$

$$r = \frac{h \times \psi}{\pi}, c = \frac{w \times \theta}{2\pi} \tag{4.7}$$

4.5.2 Spherical and World Cartesian

The sphere in the 3D space has assumed its radius as 1 and the centre as $(0, 0, 0)$. Therefore for any point (x, y, z) on the sphere it satisfies the condition in Equation (4.8).

$$\begin{aligned} (x-0)^2 + (y-0)^2 + (z-0)^2 = r^2 \\ x^2 + y^2 + z^2 = 1 \end{aligned} \tag{4.8}$$

Given the spherical coordinate (θ, ψ) we can calculate the world Cartesian

coordinate (x, y, z) using Equation (4.9).

$$\begin{aligned} x &= \sin\psi \times \cos\theta \\ y &= \sin\psi \times \sin\theta \\ z &= \cos\psi \end{aligned} \tag{4.9}$$

Combining Equation (4.8) and Equation (4.9) we can obtain Equation (4.10), to calculate the spherical coordinate (θ, ψ).

$$\begin{aligned} \theta &= \arctan\frac{y}{x} \\ \psi &= \arccos z \end{aligned} \tag{4.10}$$

4.5.3 World Cartesian and Cubemap

There are 6 faces in a Cubemap projection, so the face number needs to be calculated before processing. The method for finding a face number is explained in Section 4.2.2. As explained previously, we can select 3 of the corner vertices to form a triangle for a given face, and the 3 triangle vertices are known as $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$. For any point $\boldsymbol{p}$ on the face, its barycentric coordinate $\boldsymbol{\lambda}$ can be calculated using $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$. Since the 3 triangle vertices have also got pre-defined Cubemap coordinates $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$, the Cubemap coordinate $\boldsymbol{P}$ of the point can be calculated using Equation (4.11).

$$\boldsymbol{P} = \lambda_a \times \boldsymbol{A} + \lambda_b \times \boldsymbol{B} + \lambda_c \times \boldsymbol{C} \tag{4.11}$$

4.5.4 World Cartesian and Icosahedron

Coordinate conversion between world Cartesian and Icosahedron is similar to Section 4.5.3, except Icosahedron has 20 faces, and every face is a triangle. The world Cartesian coordinates for the 3 triangle corners are $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$ for a face, and they can be used for calculating the barycentric coordinate $\boldsymbol{\lambda}$ for any point $\boldsymbol{p}$ on the same face. Since the Icosahedron coordinates for the 3 corners are pre-defined as $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$, the Icosahedron coordinate $\boldsymbol{P}$ of the

point can be calculated using Equation (4.11).

4.5.5 Other Conversions

There are other conversions required in the project such as Equirectangule to world Cartesian, but this type of conversion can be obtained from a series of conversions mentioned earlier, i.e. Equirectangule to spherical then spherical to world Cartesian.

4.6 Relevant Techniques

4.6.1 Bilinear Interpolation

During coordinate system conversion, most of the time a converted coordinate would not have integer coordinates in the other coordinate system. In order to find its corresponding data value in the other coordinate system, it would need to be interpolated from the neighbouring integer coordinates. Bilinear interpolation is one of the interpolation algorithms. Bilinear interpolation finds its 4 nearest integer coordinates, weigh each of the value by distance then sum them up. This can be visualised in Figure 4-10. The bilinear interpolated value can be calculated using Equation (4.12).

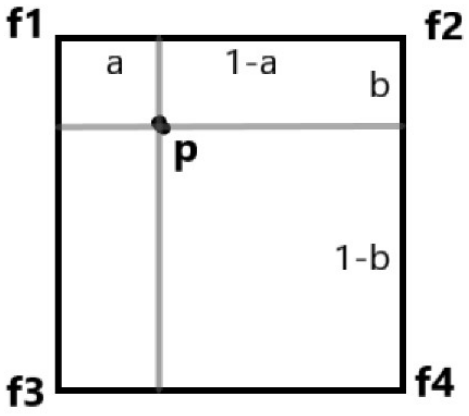


Figure 4-10: Bilinear interpolation

$$\boldsymbol{p} = (1-a)(1-b)\boldsymbol{f_1} + a(1-b)\boldsymbol{f_2} + (1-a)b\boldsymbol{f_3} + ab\boldsymbol{f_4} \tag{4.12}$$

4.6.2 Barycentric Coordinate

Barycentric coordinates system describes a point in the triangle with scales to the three triangle vertices. In Figure 4-11 there are three vertices A, B and C and a point in the triangle, all with their own Cartesian coordinates, and these values can be used to calculate the barycentric coordinate. Equation (4.13) shows how the barycentric coordinate $(\lambda_A, \lambda_B, \lambda_C)$ can be calculated from the four Cartesian coordinates where (x, y) is the target point.

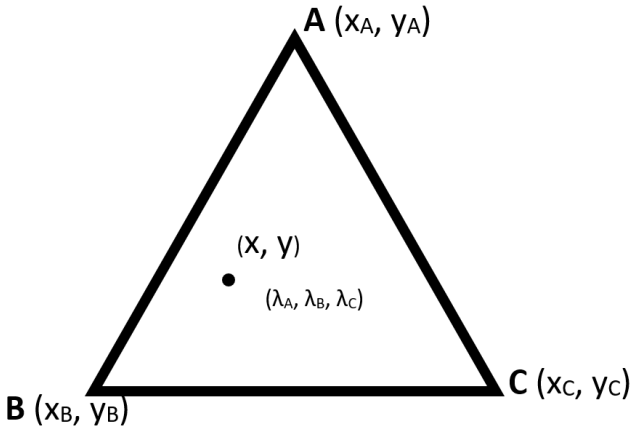


Figure 4-11: Barycentric coordinate

$$\begin{aligned}
\lambda_A &= \frac{(y_B - y_C) \times (x - x_C) + (x_C - x_B) \times (y - y_C)}{(y_B - y_C) \times (x_A - x_C) + (x_C - x_B) \times (y_A - y_C)} \\
\lambda_B &= \frac{(x_A - x_C) \times (y - y_C) - (y_A - y_C) \times (x - x_C)}{(y_B - y_C) \times (x_A - x_C) + (x_C - x_B) \times (y_A - y_C)} \\
\lambda_C &= 1 - \lambda_A - \lambda_B
\end{aligned} \tag{4.13}$$

Chapter 5

Testing and Evaluation

In this chapter we present the metrics for evaluating the performance of 360° optical flow computation, and what dataset is used. Additionally, we explain some decision making by showing some testing results, and show the final evaluation results for the 3 projection methods.

5.1 Testing methodology

The three projection methods implemented in Chapter 4 are evaluated for 360° optical flow computation. In this section, we are going to explain what evaluation metrics are being used and how they are measured, in the following order: endpoint error, angular error, great circle distance, spherical angular error and frame interpolation error.

5.1.1 Endpoint Error (EPE)

The endpoint error on the Equirectangular format measures the euclidean distance between the endpoints of the ground truth flow and the computed flow. EPE is considered as the absolute error between flows. This is a popular metric for measuring the performance of an optical flow algorithm [Kondermann et al., 2016, Butler et al., 2012, Geiger et al., 2012, Baker et al., 2011, Otte

and Nagel, 1994]. The results will be measured in pixels.

$$EPE = \|\boldsymbol{F}_{\mathrm{gt}} - \boldsymbol{F}_{\mathrm{c}}\| \tag{5.1}$$

5.1.2 Angular Error (AE)

The angular error is the measure of the angle between the ground truth flow vector $\boldsymbol{F}_{\mathrm{gt}}$ and the calculated flow vector $\boldsymbol{F}_{\mathrm{c}}$, on the Equirectangular plane. The angle α between 2 vectors can be calculated using dot product, as shown in Equation (5.2). AE has also been used by many researchers to evaluate the performance of optical flow algorithms[Fleet and Jepson, 1990, Barron et al., 1994, Baker et al., 2011]. The result angles will be measured in degrees.

$$\begin{aligned} \boldsymbol{F}_{\mathrm{gt}} \cdot \boldsymbol{F}_{\mathrm{c}} &= \|\boldsymbol{F}_{\mathrm{gt}}\|\|\boldsymbol{F}_{\mathrm{c}}\| \cos\alpha \\ \cos\alpha &= \frac{\boldsymbol{F}_{\mathrm{gt}} \cdot \boldsymbol{F}_{\mathrm{c}}}{\|\boldsymbol{F}_{\mathrm{gt}}\|\|\boldsymbol{F}_{\mathrm{c}}\|} \\ AE = \alpha &= \arccos \frac{\boldsymbol{F}_{\mathrm{gt}} \cdot \boldsymbol{F}_{\mathrm{c}}}{\|\boldsymbol{F}_{\mathrm{gt}}\|\|\boldsymbol{F}_{\mathrm{c}}\|} \end{aligned} \tag{5.2}$$

EPE or AE on its own may not correctly reflect performance in some cases, and Figure 5-1 shows some examples. On the left the ground truth flow and the computed flow are colinear, so AE would be 0, but the magnitudes of the flows are quite different resulting in big EPE. On the right flows have very small EPE, but with big number in AE. Even though these are some extreme cases for flows and they would be rare in practice, they would still suggest that flow could be more accurately analysed by measuring EPE and AE together.

5.1.3 Great Circle Distance (GCD)

Great Circle Distance[1] is the endpoint distance between the ground truth endpoint and the computed endpoint on the sphere surface. In another word, it is the arc length between the two endpoints. As shown in Equation (5.6), GCD can be represented by the radial angle β between the two vectors $\boldsymbol{v_0}$ and

[1] https://en.wikipedia.org/wiki/Great-circle_distance

Figure 5-1: Visualizations of endpoint error and angular error. White arrows are ground truth flows and red arrows are computed flows. Left: perfect AE as vectors are colinear, but terrible EPE; Right: respectable EPE but terrible AE.

$\boldsymbol{v_1}$, assuming it is a radius 1 unit sphere.

$$\boldsymbol{v_0} = \boldsymbol{F}_{\text{gt}} + \boldsymbol{p} \qquad \boldsymbol{v_1} = \boldsymbol{F}_{\text{c}} + \boldsymbol{p} \tag{5.3}$$

$$GCD = r \arccos \frac{\boldsymbol{v_0} \cdot \boldsymbol{v_1}}{r^2} \tag{5.4}$$

$$\boldsymbol{v_0} \cdot \boldsymbol{v_1} = \|\boldsymbol{v_0}\|\|\boldsymbol{v_1}\| \cos\beta = r^2 \cos\beta \tag{5.5}$$

$$GCD = r\beta = \beta \tag{5.6}$$

5.1.4 Spherical Angular Error (SAE)

Spherical angular error is the angular measure on the sphere surface. Figure 5-2 has shown what angles on a sphere surface look like. A, B and C are the spherical angles between sphere arcs, and a, b and c are the radial angles like mentioned in GCD. The value of the spherical angle can be calculated using Equation (5.7), and it will be measured in degrees.

$$SAE = A = \arccos \frac{\cos a - \cos b \times \cos c}{\sin b \times \sin c} \tag{5.7}$$

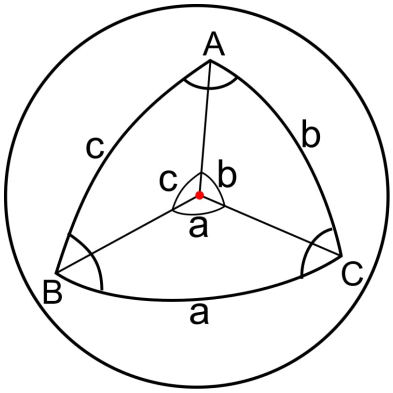


Figure 5-2: Angle measures on a sphere surface

5.1.5 Frame Interpolation Error (FIE)

One of the common applications for optical flow is image warping. Therefore the pixel difference between a ground truth image and an interpolated image can also be used to evaluate the performance of the flow algorithm. Baker et al. [2011] has suggested that FIE can is calculated by square rooting the sum-of-square-difference between the interpolated image $\boldsymbol{I}_{\mathrm{c}}$ and the ground truth image $\boldsymbol{I}_{\mathrm{gt}}$ as shown in Equation (5.8).

$$FIE = \sqrt{\sum_{(i,j)} (\boldsymbol{I}_{\mathrm{gt}}(i,j) - \boldsymbol{I}_{\mathrm{c}}(i,j))^2} \tag{5.8}$$

5.2 Testing dataset

The testing dataset is a synthetic set of images generated using OpenGL [Straub et al., 2019]. It includes 6 set of images, each set has a particular scene, and they are all indoor scenes. There are 180 images in each set as well as their corresponding ground truth flows. Every image is in Equirectangular form, with height of 480 pixels and width of 960 pixels. Samples of the dataset are shown in Figure 5-3.

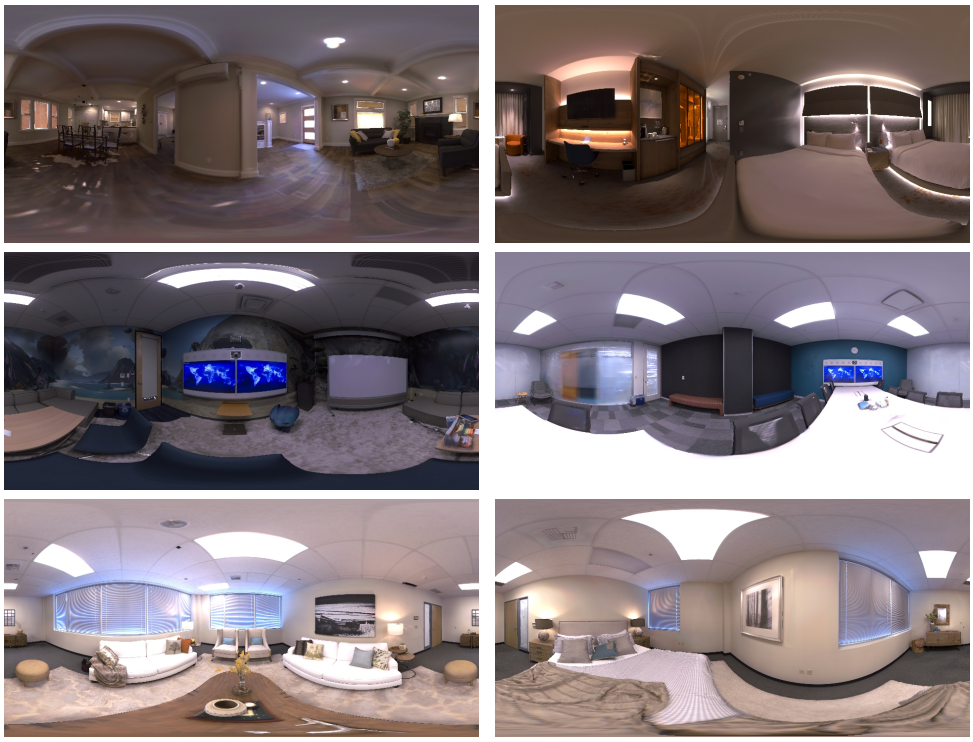


Figure 5-3: Images from the 6 sets of ground truth data. Top left – apartment; Top right – hotel; Middle left – office0; Middle right – office4; Bottom left – room0; Bottom right – room1. [Straub et al., 2019]

	DIS Mode	apartment	hotel	office0	office4	room0	room1
EPE	**Ultrafast**	3.957	4.816	4.480	6.167	3.567	4.205
	Fast	3.866	4.623	4.353	6.045	3.448	4.047
	Medium	3.618	4.323	4.161	5.853	3.099	3.645
AE	**Ultrafast**	7.814	11.466	10.441	17.232	4.927	6.720
	Fast	7.237	10.279	9.539	16.570	4.239	5.671
	Medium	5.808	8.423	8.644	14.635	2.782	4.066

Table 5.1: Results for comparing different speed mode of DISOpticalFlow function. All runs are using direct Equirectangular projection, with no padding.

5.3 Result evaluation

In this section we have presented some test results, to explain the decision of flow algorithm mode and why all images are padded before computing optical flows. We have also presented the performance evaluations for computing 360° optical flows using Equirectangle, Cubemap and Icosahedron.

5.3.1 Flow Algorithm Mode

DISOpticalFlow[2] is used in this thesis for general optical flow computation, and there are 3 speed modes according to its documentation: ultrafast, fast and medium. We have used direct Equirectangluar projection for computing flows, EPE and AE as evaluation metrics to test the performance of the 3 modes. Table 5.1 shows the results of the test. It is obvious and consistent that medium mode has the best performance among the 3. Therefore we have decided to use medium mode for the rest of the testing.

5.3.2 Decision on Image Padding

Image padding has been tested to verify its necessity for computing 360° optical flows. The test uses Equirectangular projection, both padded and unpadded, and EPE and AE are used for performance evaluation. The results in Table 5.2 suggests that image padding can improve 360° flow computation performance.

[2]https://docs.opencv.org/3.4/da/d06/classcv_1_1optflow_1_1DISOpticalFlow.html

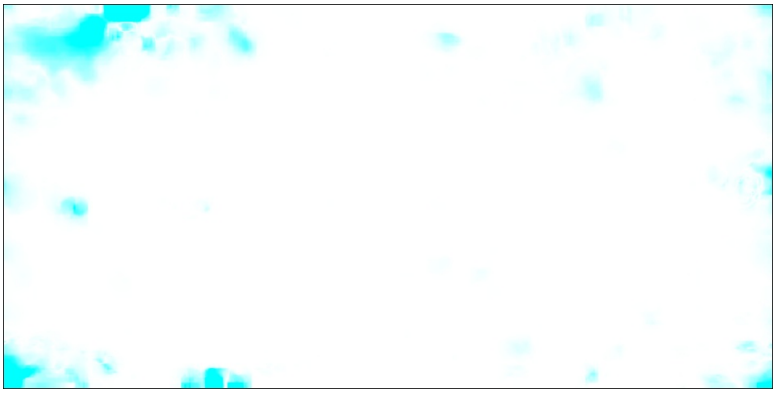


Figure 5-4: Visualization of flow differences between unpadded and padded Equirectangular projection. Pixels with great differences are in cyan, and white means the 2 flows have small to no difference.

	Padding	apartment	hotel	office0	office4	room0	room1
EPE	**No**	3.618	4.323	4.161	5.853	3.099	3.645
	Yes	3.601	4.297	4.132	5.802	3.084	3.619
AE	**No**	5.808	8.423	8.644	14.635	2.782	4.066
	Yes	5.629	8.261	8.429	14.573	2.728	3.898

Table 5.2: Results for evaluating whether image padding provides performance benefit in optical flow computation for 360° images. All runs are using Equirectangular projection, and medium mode for DISOpticalFlow.

The flow difference between the padded and unpadded projection is visualized in Figure 5-4, where most of the differences are near the image boundaries. Error could accumulate when the number of projection faces increases. Therefore image padding is applied to every projection method in the final test.

5.3.3 Result Analysis

Table 5.3 is performance results for the three projection methods explained in Chapter 4. Every value in the table is the average result of an image set. All images have been padded before running DISOpticalFlow, and all runs are using medium mode for DISOpticalFlow. Figure 5-5 shows examples of the optical flow results using the three projection methods, in comparison with the

ground truth. The results suggest that both Cubemap and Icosahedron outperform Equirectangle by a great margin in every evaluation metric for every dataset, and Icosahedron has the best performance among the 3 approaches. The figure also shows flows computed using Cubemap or Icosahedron are more accurate than using Equirectangle, especially at the top and the bottom where the pole areas are at.

It is interesting to see that while Cubemap has better EPE than Icosahedron in the apartment set and the hotel set, the GCD of Icosahedron are actually better than Cubemap for both sets. This suggests the importance of using GCD in evaluating flow, since long displacements near the poles in Equirectangular form would affect the performance in EPE, but they might be insignificant in the original spherical form.

While in most cases Icosahedron performs better than Cubemap, Cubemap outperforms Icosahedron in the dataset office4, which can be seen in both Table 5.3 and Figure 5-5. Moreover, office4 has the worst result among all datasets in every evaluation metric. We can see an example of the office4 dataset in Figure 5-3, there is a long white table spreads across the whole south pole area, and there are repetitive patterns in the ceiling across the north pole area. Cubemap may reserve its advantage in this case over Icosahedron by processing the south pole area (as well as the north pole) using 1 image rather than several. However, since there is only one dataset with such image layout, we cannot conclude that the these factors are the direct causations of the performance values, but it is worth to investigate deeper in the future. We also observe obvious seams in Icosahedron and Cubemap flows in Figure 5-5, this may be a con of using multiple-face projections.

	Projection	apartment	hotel	office0	office4	room0	room1
EPE	**Equirectangle**	3.601	4.297	4.132	5.802	3.084	3.619
	Cubemap	1.590	2.468	2.287	3.646	1.259	1.476
	Icosahedron	1.591	2.509	2.121	4.053	1.154	1.281
AE	**Equirectangle**	5.629	8.261	8.429	14.573	2.728	3.898
	Cubemap	3.419	6.850	5.363	7.472	2.602	3.609
	Icosahedron	3.111	6.367	4.984	9.496	2.296	3.236
GCD	**Equirectangle**	2.385	3.887	3.367	5.537	2.460	3.118
	Cubemap	1.829	3.497	2.488	4.433	2.190	2.798
	Icosahedron	1.704	3.397	2.306	4.741	2.059	2.580
SAE	**Equirectangle**	6.139	8.897	8.424	12.828	3.834	5.166
	Cubemap	3.645	7.242	5.395	7.539	2.852	3.975
	Icosahedron	3.374	6.925	5.098	9.266	2.557	3.589
FIE	**Equirectangle**	14.671	18.569	19.563	35.022	28.718	26.902
	Cubemap	11.778	15.716	14.256	26.217	23.733	22.442
	Icosahedron	11.679	15.233	13.795	26.559	23.394	21.935

Table 5.3: 360° image optical flow result evaluation for Equirectangular, Cubemap and Icosahedron projections. Evaluation metrics include EPE measured in pixels, AE in degrees, GCD in 10^{-3}units, SAE in degrees and FIE. All projection methods have applied image padding. All runs are using medium mode for DISOpticalFlow.

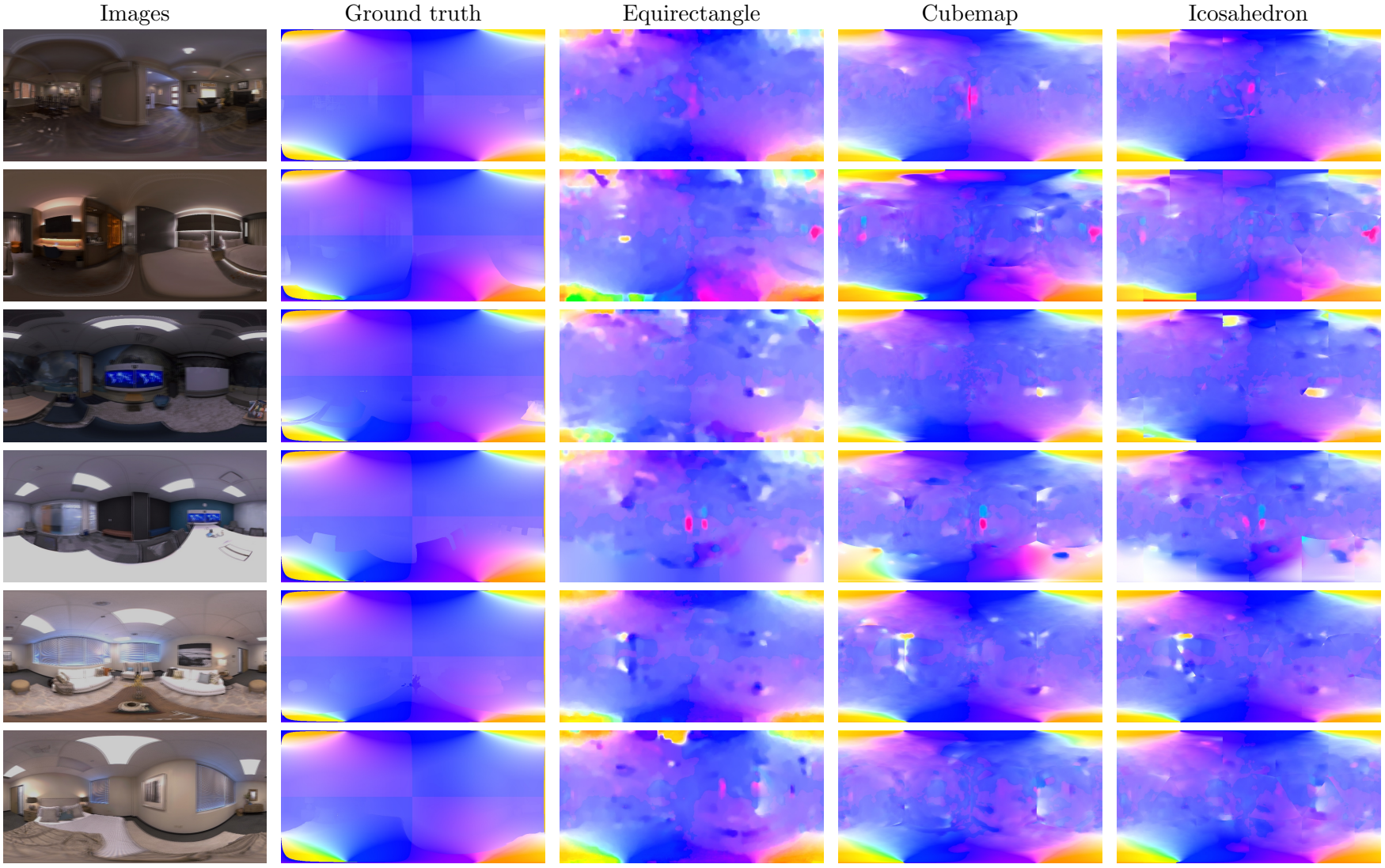


Figure 5-5: Examples of the optical flow results using the three projection methods, in comparison with the ground truth. Dataset are in the following row order: apartment, hotel, office0, office4, room0, room1.

Chapter 6

Conclusion

In this chapter we summarise this report and talk about possible directions for future researches.

6.1 Summary

In Chapter 2 we have looked at some backgrounds about optical flow computation, and different projection methods for spherical images. The pipelines for computing 360° flow using different projections are described in Chapter 3, and Chapter 4 shows the implementation details. Then in Chapter 5 we show the performance evaluations for the 3 projections, both statistically and visually.

In this report we have proposed a method for computing 360° optical flow, using Icosahedron projection for 360° images. We have also looked at two other existing methods which use Equirectanglar projection and Cubemap projection, and compare the performance of the 3 methods for computing 360° flows. While Equirectangle is used by many researchers for computing 360° optical flow due to its implementation simplicity, the evaluation results suggest that by using a different projection method like Cubemap or Icosahedron, much more accurate 360° flows can be obtained. In addition, the results show that 360° flows calculated using our proposed method have the best accuracy among the three.

There are some limitations and problems in the current implementation. For Cubemap and Icosahedron flow visualization in Figure 5-5, obvious seams can be observed at where the face boundaries are. In addition, the 360° flow pipelines for both Cubemap and Icosahedron projections involve flow interpolation, although it can be calculated mathematically, the result of interpolating motions is meaningless, and interpolation for both flows and images may cause loss in details. The testing dataset in this thesis is limited to indoor scenes only, so we could not evaluate how the three methods perform for outdoor images.

6.2 Future Work

There is a lot more to explore, for better understanding 360° optical flow, and finding some well-performed computation method. Like mentioned in Chapter 2, there are more projection methods for spherical images yet to be tested. We could use Equirectangle as a baseline, to see how other projections perform compare to the baseline, and what are the ideal image conditions for each projection method, since the evaluation results in this thesis show that Cubemap may out-perform Icosahedron in some special cases.

Moreover, it is suspected that large objects in plain colours or objects with repetitive patterns may have impact on the performance of 360° flow computation. Therefore in the future we could develop different datasets with specific conditions embedded, to identify how those conditions could affect the 360° flow performance, and ideally find solutions to handle them.

Bibliography

Simon Baker, Daniel Scharstein, JP Lewis, Stefan Roth, Michael J Black, and Richard Szeliski. A database and evaluation methodology for optical flow. *International journal of computer vision*, 92(1):1–31, 2011.

John L Barron, David J Fleet, and Steven S Beauchemin. Performance of optical flow techniques. *International journal of computer vision*, 12(1): 43–77, 1994.

Tobias Bertel, Neill D. F. Campbell, and Christian Richardt. MegaParallax: Casual 360 panoramas with motion parallax. *IEEE Transactions on Visualization and Computer Graphics*, 2019. doi: 10.1109/TVCG.2019.2898799. URL `https://richardt.name/megaparallax/`. to appear.

Chip Brown. Bringing pixels front and center in vr video, Mar 2017. URL `https://blog.google/products/google-ar-vr/bringing-pixels-front-and-center-vr-video/`.

Andrés Bruhn, Joachim Weickert, and Christoph Schnörr. Lucas/kanade meets horn/schunck: Combining local and global optic flow methods. *International journal of computer vision*, 61(3):211–231, 2005. URL `https://link.springer.com/article/10.1023/B:VISI.0000045324.43199.43`.

D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. A naturalistic open source movie for optical flow evaluation. In A. Fitzgibbon et al. (Eds.), editor, *European Conf. on Computer Vision (ECCV)*, Part IV, LNCS 7577, pages 611–625. Springer-Verlag, October 2012.

Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick Van Der Smagt, Daniel Cremers, and Thomas

Brox. Flownet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2758–2766, 2015.

Irving Fisher. A world map on a regular icosahedron by gnomonic projection. *Geographical Review*, 33(4):605–619, 1943.

David J Fleet and Allan D Jepson. Computation of component image velocity from local phase information. *International journal of computer vision*, 5 (1):77–104, 1990.

Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361. IEEE, 2012.

Ned Greene. Environment mapping and other applications of world projections. *IEEE Computer Graphics and Applications*, 6(11):21–29, 1986. URL `https://ieeexplore.ieee.org/document/4056759`.

Berthold KP Horn and Brian G Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981. URL `https://www.sciencedirect.com/science/article/pii/0004370281900242`.

Kyoungkook Kang and Sunghyun Cho. Interactive and automatic navigation for 360° video playback. *ACM Transactions on Graphics (TOG)*, 38(4): 1–11, 2019.

Jens Klappstein, Tobi Vaudrey, Clemens Rabe, Andreas Wedel, and Reinhard Klette. Moving object segmentation using optical flow and depth information. In *Pacific-Rim Symposium on Image and Video Technology*, pages 611–623. Springer, 2009. URL `https://link.springer.com/chapter/10.1007/978-3-540-92957-4_53`.

Daniel Kondermann, Rahul Nair, Katrin Honauer, Karsten Krispin, Jonas Andrulis, Alexander Brock, Burkhard Gussefeld, Mohsen Rahimimoghaddam, Sabine Hofmann, Claus Brenner, et al. The hci benchmark suite: Stereo and flow ground truth with uncertainties for urban autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 19–28, 2016.

Pierre Lebreton, Stephan Fremerey, and Alexander Raake. V-bms360: A video extention to the bms360 image saliency model. In *2018 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*, pages 1–4. IEEE, 2018.

Yeonkun Lee, Jaeseok Jeong, Jongseob Yun, Wonjune Cho, and Kuk-Jin Yoon. Spherephd: Applying cnns on a spherical polyhedron representation of 360 degree images. *arXiv preprint arXiv:1811.08196*, 2018.

Bruce D Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. 1981. URL `https://ri.cmu.edu/pub_files/pub3/lucas_bruce_d_1981_2/lucas_bruce_d_1981_2.pdf`.

Kevin Matzen, Michael F Cohen, Bryce Evans, Johannes Kopf, and Richard Szeliski. Low-cost 360 stereo photography and video capture. *ACM Transactions on Graphics (TOG)*, 36(4):1–12, 2017.

Michael Otte and H-H Nagel. Optical flow estimation: advances and comparisons. In *European conference on computer vision*, pages 49–60. Springer, 1994.

Qikai Pei, Juan Guo, LU Haiwen, Guilona Ma, Wensong Li, and Xinyu Zhang. Cop: A new continuous packing layout for 360 vr videos. In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 655–656. IEEE, 2018. URL `https://ieeexplore.ieee.org/document/8446523`.

Emil Praun and Hugues Hoppe. Spherical parametrization and remeshing. In *ACM Transactions on Graphics (TOG)*, volume 22, pages 340–349. ACM, 2003. URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.471.3763&rep=rep1&type=pdf`.

F. Raudies. Optic flow. *Scholarpedia*, 8(7):30724, 2013. doi: 10.4249/scholarpedia.30724. URL `http://www.scholarpedia.org/article/Optic_flow`. revision #149632.

Jerome Revaud, Philippe Weinzaepfel, Zaid Harchaoui, and Cordelia Schmid. Deep convolutional matching. *arXiv preprint arXiv:1506.07656*, 2015.

John P Snyder. *Flattening the earth: two thousand years of map projections*. University of Chicago Press, 1997.

Julian Straub, Thomas Whelan, Lingni Ma, Yufan Chen, Erik Wijmans, Simon Green, Jakob J Engel, Raul Mur-Artal, Carl Ren, Shobhit Verma, et al. The replica dataset: A digital replica of indoor spaces. *arXiv preprint arXiv:1906.05797*, 2019.

Shinya Yamamoto, Yasushi Mae, Yoshiaki Shirai, and Jun Miura. Realtime multiple object tracking based on optical flows. In *Proceedings of 1995 IEEE International Conference on Robotics and Automation*, volume 3, pages 2328–2333. IEEE, 1995. URL `https://ieeexplore.ieee.org/abstract/document/525608`.

Chao Zhang, Stephan Liwicki, William Smith, and Roberto Cipolla. Orientation-aware semantic segmentation on icosahedron spheres. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3533–3541, 2019.

Y. Zhang, F. Dai, Y. Ma, H. Li, Q. Zhao, and Y. Zhang. Saliency prediction network for 360° videos. *IEEE Journal of Selected Topics in Signal Processing*, 14(1):27–37, 2020.

.1 Appendix A

This form must be attached to the dissertation as an appendix.

Department of Computer Science

12-Point Ethics Checklist for UG and MSc Projects

Student YONGQIANG ZHU

Academic Year or Project Title OPTICAL FLOW COMPUTATION FOR 360 IMAGES

Supervisor CHRISTIAN RICHARDT

Does your project involve people for the collection of data other than you and your supervisor(s)? YES / NO

If the answer to the previous question is YES, you need to answer the following questions, otherwise you can ignore them.

This document describes the 12 issues that need to be considered carefully before students or staff involve other people ('participants' or 'volunteers') for the collection of information as part of their project or research. Replace the text beneath each question with a statement of how you address the issue in your project.

1. *Have you prepared a briefing script for volunteers?* YES / NO

   Briefing means telling someone enough in advance so that they can understand what is involved and why – it is what makes informed consent informed.

2. *Will the participants be informed that they could withdraw at any time?* YES / NO

   All participants have the right to withdraw at any time during the investigation, and to withdraw their data up to the point at which it is anonymised. They should be told this in the briefing script.

3. *Is there any intentional deception of the participants?* YES / NO

   Withholding information or misleading participants is unacceptable if participants are likely to object or show unease when debriefed.

4. *Will participants be de-briefed?* YES / NO

   The investigator must provide the participants with sufficient information in the debriefing to enable them to understand the nature of the investigation. This phase might wait until after the study is completed where this is necessary to protect the integrity of the study.

5. *Will participants voluntarily give informed consent?* YES / NO

Participants MUST consent before taking part in the study, informed by the briefing sheet. Participants should give their consent explicitly and in a form that is persistent –e.g. signing a form or sending an email. Signed consent forms should be kept by the supervisor after the study is complete.

6. *Will the participants be exposed to any risks greater than those encountered in their normal work life (e.g., through the use of non-standard equipment)?* YES / NO

Investigators have a responsibility to protect participants from physical and mental harm during the investigation. The risk of harm must be no greater than in ordinary life.

7. *Are you offering any incentive to the participants?* YES / NO

The payment of participants must not be used to induce them to risk harm beyond that which they risk without payment in their normal lifestyle.

8. *Are you in a position of authority or influence over any of your participants?* YES / NO

A position of authority or influence over any participant must not be allowed to pressurise participants to take part in, or remain in, any experiment.

9. *Are any of your participants under the age of 16?* YES / NO

Parental consent is required for participants under the age of 16.

10. *Do any of your participants have an impairment that will limit Their understanding or communication?* YES / NO

Additional consent is required for participants with impairments.

11. *Will the participants be informed of your contact details?* YES / NO

All participants must be able to contact the investigator after the investigation. They should be given the details of the Supervisor as part of the debriefing.

12. *Do you have a data management plan for all recorded data?* YES / NO

All participant data (hard copy and soft copy) should be stored securely, and in anonymous form, on university servers (not the cloud). If the study is part of a larger study, there should be a data management plan.