INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Master's Thesis

Pixel-based 2-DoF Synthesis of 360° Viewpoints Using Flow-based Interpolation

Rosalie Kletzander

Draft from February 1, 2021

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Master's Thesis

Pixel-based 2-DoF Synthesis of 360° Viewpoints Using Flow-based Interpolation

Rosalie Kletzander

Aufgabensteller:	Prof. Dr. Dieter Kranzlmüller
Betreuer:	Prof. Dr. Jean-François Lalonde (Université Laval, Kanada) Markus Wiedemann
Abgabetermin:	2. Februar 2021

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 26. Januar 2021

. . . . . .
(Unterschrift des Kandidaten)

Abstract

Virtual Reality technology allows users to experience virtual environments by interacting with them, and navigating within them. These environments tend to be either meticulously modeled in 3D by hand, or prerecorded using 360° cameras. The advantage of using 360° images is that they achieve a high level of realism with little effort, however, they often limit the user to a single viewpoint. Image-based rendering, or image-based synthesis aims to create novel viewpoints based on captured viewpoints, in the best case enabling a user to navigate freely around a scene. There are a number of different approaches to image-based synthesis, many using some form of feature correspondence to extract information, such as scene geometry, from the captured images. Extracting scene geometry from images can be problematic, since inaccurate scene geometry can lead to severe artefacts. This thesis proposes a pixel-based 2-DoF synthesis algorithm that combines basic reprojection using proxy geometry with flow-based interpolation. The use of simple proxy geometry, such as a sphere, in place of estimated scene geometry makes it possible to synthesize novel views within a scene without knowledge about the geometry. However, the difference of the proxy geometry from the scene geometry can lead to severe artefacts, just like when using inaccurately estimated scene geometry. Flow-based interpolation, which is generally used for interpolating between pairs of images, is leveraged in order to alleviate some of these artefacts. A proof-of-concept implementation of the approach is presented, and tested with a select set of parameters, using different virtual and real scenes. The results are then evaluated based on mathematical error metrics, as well as visible artefacts. The results of the evaluation show that the flow-based method improves the basic method in a number of cases.

Contents

1. Introduction

Over the past decade, Virtual Reality (VR) technology has experienced a resurgence in popularity due to the development of a number of affordable, consumer-quality head-mounted displays[1]. These displays allow a user to experience and interact with a virtual environment in 3D, for example by playing games, or by taking virtual tours of cities, historical landmarks, or remote locations in nature.

Some of these environments are modeled meticulously in 3D, while others use 360° images taken on location. Often, the 3D-modeled environments allow a lot of freedom of movement, enabling the user to "walk" around and inspect elements of the scene at will. Unfortunately, modeling a real environment by hand to be viewed interactively requires an enormous amount of time and effort, and even then it is very difficult to achieve photo-realism. An alternative is to capture the location with a 360° camera, which records all of the surroundings in a single image. Viewing these images offers photo-realism (as they are actual photos), but often awkward navigation, for example forcing the user to "jump" from one image location to the next, instead of being able to "walk" smoothly around the scene. Nevertheless, the ease of capturing 360° images with modern 360° cameras, along with the significant advantage of providing photo-realism, makes this an attractive method for creating immersive VR environments.

The difficulties of navigating an environment captured through 360° cameras are not a new phenomenon. Such problems are also relevant in virtual tours that exist outside of the realm of VR, viewed on regular computer or smartphone screens, for example interactive tours of museums, real-estate, or other locations of interest. A prominent example is Google's Street View, which allows users to navigate streets and monuments around the world by way of 360° images.

Whether a scene is viewed stereoscopically with a head-mounted display, or monoscopically on a flat screen, the greatest obstacle at the moment is the problem of interactive, user-driven navigation. Ideally, a user would be able to go anywhere they liked in the scene, and view anything they wanted from any angle and at any level of detail. Unfortunately, for environments captured by way of a 360° camera, this type of interaction would require a prohibitive amount of data, as well as being impossible to manually execute, since a separate image would have to be captured at every possible viewing position.

An alternative to capturing all of these viewpoints manually is to generate them digitally. This requires capturing a much smaller subset of images and using them to generate the desired, novel viewpoints. Generating new images based on already captured images is generally known as *image-based rendering* (IBR), or image-based synthesis. There are many different approaches to synthesizing new images from captured ones, and they can generally be categorized in two different ways:

[1] The price of a consumer-quality VR headset is between approximately 20 and 1000 USD as of January 2021, including headsets for use with a smartphone (`https://www.tomsguide.com/us/best-vr-headsets,review-3550.html`, *accessed Jan 13, 2021*)

- by the degrees of freedom (DoF) that the technique can synthesize images with, and the spacial limitations on the movement within the scene
- by the type and amount of information extracted from the captured images

The area and the degrees of freedom (DoF) required for navigation usually depend on the goal and requirements of the application: A virtual tour on a pre-defined path for example, would only require generating intermediate views between existing viewpoints (i.e. synthesis with 1-DoF) for a smooth transition. For a less constrained virtual tour that enables users to navigate freely, generating viewpoints with 2-DoF at eye-height could be sufficient. Users could move around and look in any direction but not change their viewing height. Some applications may require three degrees of freedom, for example in order to enable users to closely inspect certain objects from all angles, but restrict them from moving away from the object.

Depending on the type of scene, the required fidelity, and the real-time requirements, different IBR techniques leverage different amounts and types of information extracted from the input images. On one end of the scale, there are approaches that extract as much geometry information as possible from the set of images and try to reconstruct the 3D geometry of the scene as closely as possible. These approaches can suffer from the problem of extracting accurate geometry information, since errors in geometry can lead to unappealing results. Other approaches try to extract some information from the image, such as feature correspondences, or motion vectors (optical flow), which enables interpolation between pairs of images, i.e. a smooth transition with 1-DoF. At the other end of the scale, there are approaches that use no semantic image information whatsoever. Instead, they may use color values, simple proxies for the scene geometry, or information about the relative location of captures, but they tend to operate on a pixel level. Although there is no specific label for this kind of synthesis, for the purpose of this thesis, it will be referred to as *pixel-based synthesis*.

Markus: what scale

JF says cite some examples for both types

One very basic form of pixel-based synthesis is to use a simple proxy ("stand-in") geometry, for example a sphere, in place of the real scene geometry[2]. This allows for resampling the captured images in order to create a new viewpoint at any location within the scene, without having to estimate or record the actual scene geometry. However, a significant difference between the proxy and the actual geometry can lead to severe distortions and artefacts in the resampled images. Although this basic form of pixel-based rendering using proxy geometry may be unsuitable for scenes where the real geometry differs greatly from the proxy geometry, it may be possible to improve these results by combining the basic technique with a 1-DoF interpolation method.

Problem Statement

A number of 1-DoF interpolation techniques exist, many of which use some form of feature correspondence. Flow-based interpolation is one of these techniques that has already been used successfully for 360° images. It uses motion vectors ("optical flow") between pairs of images to interpolate new viewpoints between them. The goal of this thesis is to answer the following research questions:

[2]The term "proxy geometry" is used in this thesis as a term for the model used in place of the real scene geometry. It can be anything from a simple geometric shape such as a sphere, to a simplified version of the real scene geometry

1. Can flow-based interpolation be used in 2-DoF pixel-based synthesis with proxy geometry?

2. If it can, does the flow-based interpolation improve the accuracy of the results compared to the basic pixel-based synthesis?

In order to measure the accuracy of the different results, the synthesized images are compared with the ground truth images using two different error metrics, as well as being assessed visually.

Scope

As the number of possible different environments is infinite, as well as the positions at which images can be captured, it is necessary to decrease the parameter space for testing and evaluation. First of all, only indoor scenes are examined. The scenes are assumed to be static, meaning the objects within the scene do not change their positions. To reduce the complexity of the implementation, as well as the number of necessary input images to be captured, only 2-DoF synthesis is considered. Specifically, all captured and synthesized viewpoints are located on a plane at approximate eye-level parallel to the ground.

The parameters that will be examined are:

Difference between the scene geometry and the proxy geometry The proxy geometry used in the approach is a sphere of the approximate size of the scene. Scenes of different basic shapes, containing different objects are tested, in order to gauge the effect of the difference between the proxy model and the scene. The "difference" of the model to the sphere is difficult to quantify in a meaningful way, so it is not measured but done intuitively.

Density of captured viewpoints

Position of synthesized points relative to captured points

These parameters will not be examined exhaustively, as this is impossible, given the infinite possibilities of scene shapes and object placements, alone. Instead, for each parameter to be examined, a scenario is designed that tests this parameter in a reasonable context.

Methodology

In order to implement and evaluate an algorithm that combines basic pixel-based synthesis and flow-based interpolation, this thesis follows an approach consisting of two distinct phases: implementation and evaluation (Figure 1.1). The implementation consists of first developing a basic pixel-based synthesis algorithm using proxy geometry. Then, based on existing 1-DoF flow-based interpolation algorithms (presented in Section 2.2), the extended pixel-based algorithm using flow-based interpolation is developed. The approach and implementation are presented in Chapter 3.

The evaluation step is further divided into three phases. In the preparation phase, based on the parameter space described in detail in Section 4.1, data for testing and evaluation is acquired. This data includes virtual and real scenes. Additionally, the error metrics are

adapted for 360° images (Section 4.2). Then, in the synthesis and testing phase, the images are synthesized using the implementations developed in the implementations step. Using the result images and the ground truth data generated during the data acquisition, error calculation is performed with the adapted metrics. Finally, using the calculated error values, along with the synthesized images, the results are analyzed. The analysis is made up of two parts: First, an analysis is performed on the virtual data that explores the limits of the chosen parameters (Section 4.3). Then, the results using the real data are examined as a proof-of-concept (Section 4.4).

Contributions

The contributions of this thesis are:

- An algorithm for 2-DoF pixel-based synthesis algorithm with flow-based interpolation
- A basic proof-of-concept implementation of this algorithm
- An evaluation of the results of this algorithm with a comparison of the basic pixel-based algorithm versus with flow-based interpolation, proving that flow-based interpolation does improve the results of the basic algorithm in the majority of the examined cases

reformulate, this is too contrived

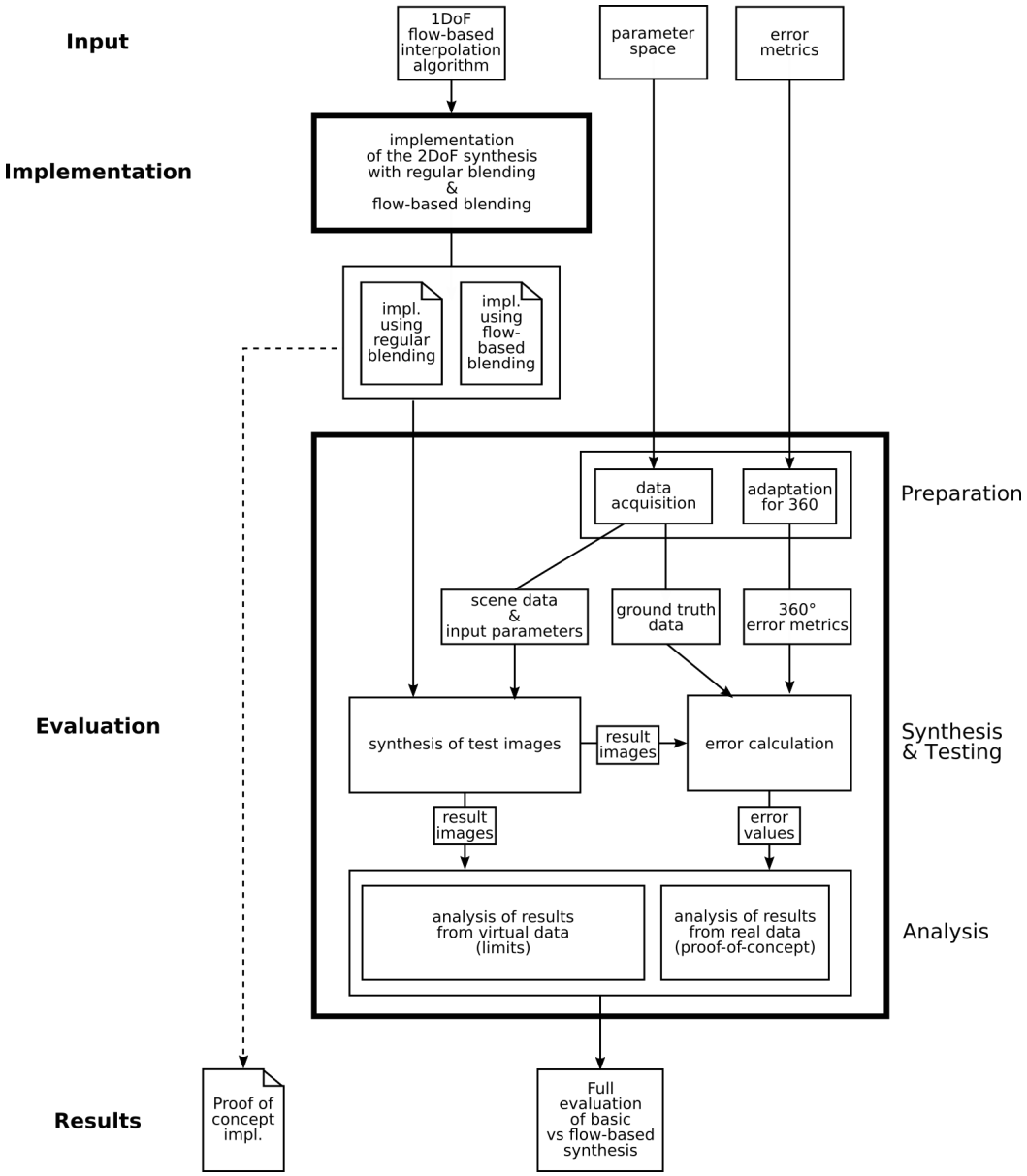


Figure 1.1.: Methodical approach

2. Background and Related Work

Before diving into the details of the pixel-based rendering approach with flow-based interpolation that is the focus of this thesis, it is important to outline the basic concepts and methods used in this approach (Section 2.1), as well as to understand the state-of-the-art of image-based rendering techniques (Section 2.2).

2.1. Fundamentals

The images used in this thesis, as well as many other approaches, are 360° images. Since 360° images differ significantly from "regular" images in how they are captured and visualized, it is important to understand how 360° cameras capture their surroundings, how the captured data can be mapped to a planar surface, and what the most common mappings for 360° images are (Section 2.1.1). Also, the concept of optical flow is introduced, as it is a prerequisite for a number of image-based rendering techniques, including the flow-based interpolation used in this thesis.

2.1.1. 360° Images

Capturing an image with a 360° camera differs significantly from capturing an image with a regular camera. A regular camera captures incoming rays of light with a limited field of view. The sensors on the camera (or the film, for analog cameras), are arranged on a plane and register the wavelengths of incoming rays. This process represents a projection of the scene onto a plane. The measured light values can then be stored directly as a planar image (Figure 2.1a).

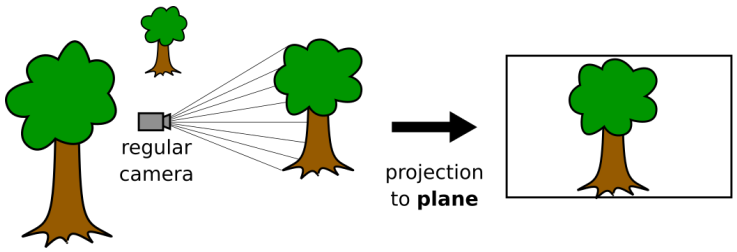
A 360° camera[1], on the other hand, captures light rays with an field of view of 360°. This means that the sensors are arranged in a way that captures light rays from the entire surroundings. For the sake of simplicity, this can be pictured as a number of sensors in the form of a sphere[2]. The camera must then perform an additional conversion in order to transform the light values captured on a sphere to a planar image (Figure 2.1b). [SI14]

The projection onto a flat surface is necessary, since image data is generally stored in 2D, and the majority of viewing devices are planar (e.g., computer or smartphone screens). The process of translating data from a 3D model to a 2D image and vice versa is well known in computer graphics and is called *uv mapping* or *texture mapping*.

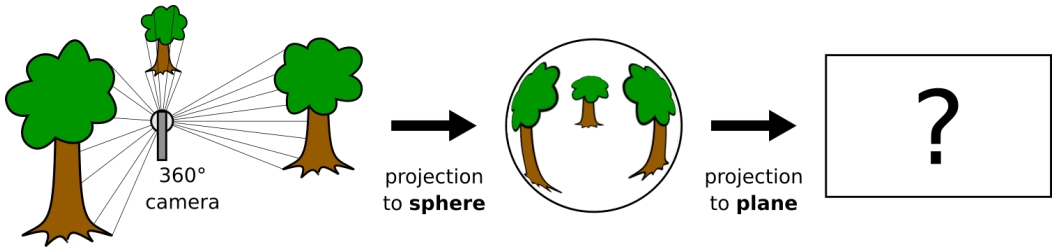
Specifically, the process of uv mapping for spherical geometry is needed for mapping the data from the sphere to a planar image. This process describes a bijective operation in

[1]The term "omnidirectional camera" is also used informally, however, this term is less exact, since an omnidirectional camera can also be a camera that captures a single hemisphere, instead of the full scene [SI14].

[2]The sensors cannot actually take the form of a perfect sphere, since the camera needs to have some form of casing. Instead, several lenses are usually used ("polydioptric cameras" [SI14]), and the image stitched together in software.



(a) Image capturing process with a regular camera: The sensors are arranged on a plane, so capturing the rays corresponds to a projection to a plane.



(b) Image capturing process with a 360° camera: The sensors are arranged in an approximation of a sphere, which means an additional step is needed to convert the captured data to a planar representation.

Figure 2.1.: Capturing an image with a regular camera compared to a 360° camera

which the points (x,y,z) on the sphere (described by *unit directions* which are unit vectors) are associated with pixel positions in image coordinates (u,v). Figure 2.2 shows an example mapping between the unit sphere and a planar image. In this example, the poles of the sphere are mapped to the entire top and bottom pixel row of the image, and the equator is mapped to the row of pixels in the vertical center of the image. This means that the areas near the poles are *oversampled*, which indicates that the mapping function is not *equal area* [RWPD05, p.450] i.e. it does not conserve how much area a pixel value occupies.

In the case of a 360° camera mapping the captured light rays to a planar image for storage, the image values are some type of color values. However, other kinds of information can also be uv-mapped to a shape, for example illumination data, depth values (bump mapping), and more.

The most common mappings for 360° images are the *cube map*, the *ideal mirrored sphere*, the *angular map*, and the *equirectangular map* [RWPD05, p. 535]. The image data can be projected using any of these mappings with only minimal data loss (by interpolation). These mappings are briefly presented in the following paragraphs.

Cube Map The cube map is a mapping that splits the image data into six separate square views, one in each direction (top, front, left, right, back, bottom). This is the equivalent of capturing the surroundings with six different cameras with a field of view of 90° each, and then stitching the resulting images into a shape that can be "folded" into a cube (see Figure 2.3a), which also gives this mapping its name.

Due to the projection of a spherical surface to a plane, there is some distortion towards the edges of each face. However, this distortion is comparable to the distortion at the edges of a regular, planar image, which is a significant advantage compared to other mappings (see Figure2.3b,d,f,h[3]). The disadvantage is that each face is projected separately, which

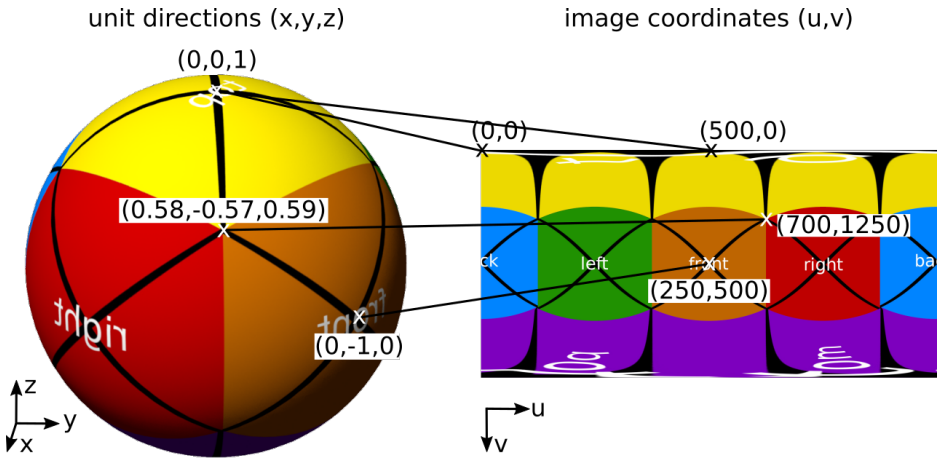


Figure 2.2.: Example of uv mapping for spherical geometry

leads to directional discontinuities at the many seams. This type of mapping is often used to simulate complex environments in 3D scenes (e.g., for game or animation graphics), as it is easy to use and reduces render time significantly compared to a 3D model of the same environment.

[RWPD05, p. 540]

Ideal Mirrored Sphere The ideal mirrored sphere is a mapping to a circle within a square. It represents how the surroundings would be reflected if one placed a small sphere with a perfectly reflective surface ("mirrored" sphere) into a scene and then photographed it using an orthographic camera. This mapping, like all the mappings presented here, shows the complete surroundings, albeit considerably distorted toward the edges. Figure 2.3cb shows where each direction is mapped and the extent of the distortion. It is clear that the farther away from the "front" area, the more distorted the mirrored sphere mapping is. The ideal mirrored sphere mapping can be used for calculating average illumination color for high dynamic range calculations; however, the type of distortion at the edges can cause problems with sampling, which is why the angular map mapping tends to be preferred. [RWPD05, p. 535]

Angular Map At first sight, the angular map seems very similar to the ideal mirrored sphere. It also maps to a circle within a square, however it samples the input in such a way that the back of the image is allotted more space and is less distorted than the mirrored sphere (see Figure 2.3e). [RWPD05, p. 537]

Equirectangular Map The equirectangular, or latitude-longitude (latlong) mapping is a common type of mapping in cartography. The data is mapped to a rectangular image space, in which the width is twice the height. The azimuth (around the circumference) of the unit directions is mapped to the map's horizontal coordinate and the elevation to the vertical

[3]Figure 2.3b does not perfectly represent the distortion in cube maps. It was chosen as a baseline because cube maps have relatively little distortion compared to other mappings and it visualizes which parts of the image are mapped where and how they are distorted in other mappings.

coordinate. The main problem of this representation is well known in cartography: The distortion increases significantly towards the poles, as can be seen in Figure 2.3h. Otherwise, this mapping is convenient as it has very few seams and all pixels are valid (i.e. there are no "black" areas). It is used as a storage format for 360° images. [RWPD05, p. 538]

While all of these projections are static, showing the entirety of the 360° image at once, it is also possible to view 360° images interactively. In this case the field of view tends to be limited, so only a certain part of the image needs to be projected: the part of the image the viewer is "facing" virtually. Once the viewing direction has been determined, the projection can be calculated such that the center of the image has minimal distortion. Theoretically, any of the above projections could be used for this.

2.1.2. Optical Flow

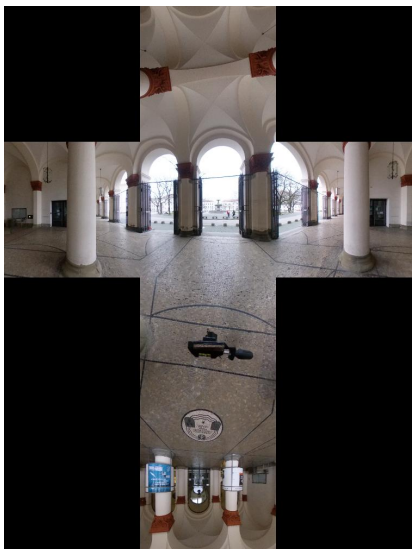
Optical flow describes the displacement of specific points between two images. It is generally used on consecutive frames of video sequences, for example for semantic segmentation, structure-from-motion, data compression or other applications where information about movement between images is required. To illustrate, Figure 2.4 shows two consecutive frames of a video sequence. On a high level, an optical flow algorithm should recognize that the pixels representing the bicyclist are moving towards the bottom left of the image, and the pixels representing the background are moving to the right (because the camera is panning slightly to the left).

There are two types of optical flow: sparse optical flow and dense optical flow. Sparse optical flow algorithms calculate the motion of several select points that can be either chosen manually, or by some kind of automatic selection (e.g., based on features). This type of optical flow can be used to track only specific objects in a scene (e.g., the direction and relative velocity of a certain car in traffic).

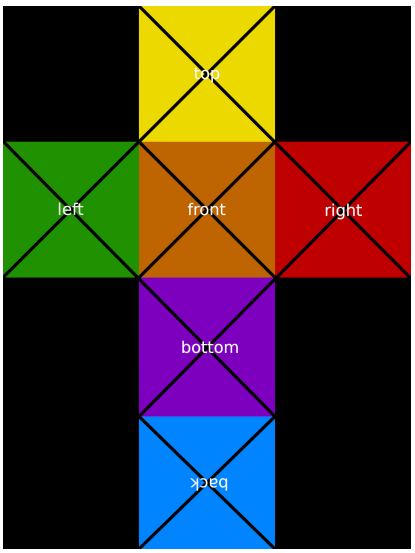
Dense optical flow algorithms compute the motion of *each pixel* between two images, instead of single points. This can be used for more general object tracking (e.g., direction and relative velocity of complete surroundings in traffic), to estimate 3D geometry (in structure-from-motion algorithms), or to identify static sections of the image for video compression [FBK15]. Dense optical flow can also be used for image synthesis, such as in Richardt et al's Megastereo [RPZSH13] described in Section 2.2.2, which is also the basis of the flow-based interpolation presented in this thesis.

There are a number of optical flow algorithms, ranging from methods using parametrization, or regularization [FBK15] to methods relying on Deep Learning [SET20]. Although these algorithms differ greatly in approach, they share the type of result, which is a vector field. For dense optical flow, this vector field contains a vector for each pixel, describing the displacement of this pixel between the input images. Sparse optical flow only contains a vector for each pre-chosen point, not for every pixel.

Figure 2.5 shows two different visualizations of the vector field calculated by the dense optical flow algorithm by Farnebäck [Far03] between the frames in Figure 2.4. Figure 2.5a is a color-based visualization: the hue encodes the vector direction and the saturation encodes the vector length for each pixel. Using this visualization, it is possible to roughly distinguish two separately moving areas of the image, which could be used for semantic segmentation. Figure 2.5b shows the pixel displacements with vectors: The origin of the vector is shown by a point and the direction and length of the vector are represented by a line.

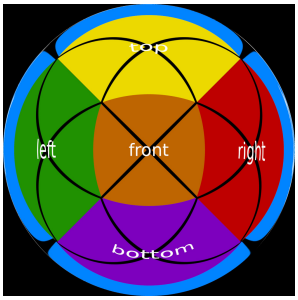


(a) Cube map example



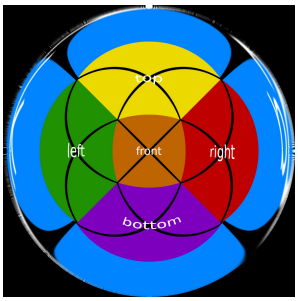
(b) Cube map distortion visualization

(c) Mirrored sphere mapping example



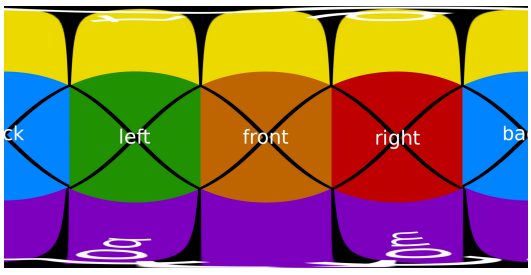
(d) Mirrored sphere distortion visualization

(e) Angular mapping example



(f) Angular mapping distortion visualization

(g) Equirectangular (latlong) mapping example

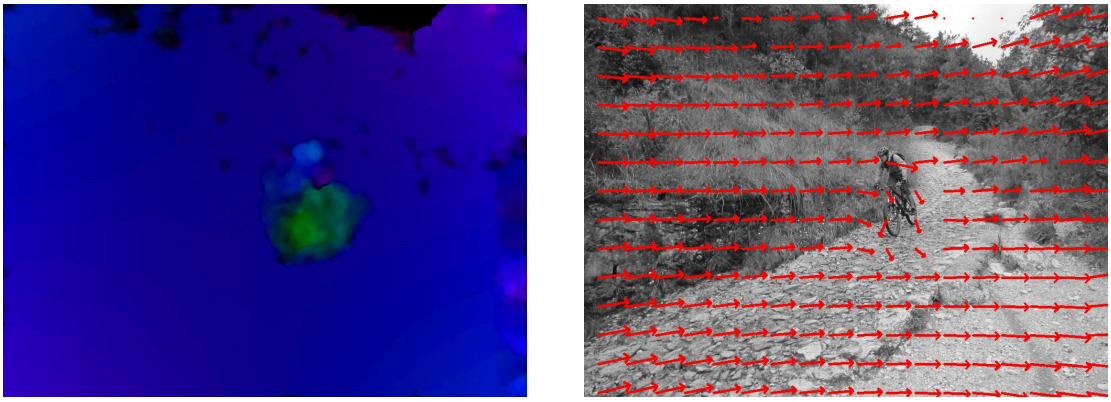


(h) Equirectangular distortion visualization

Figure 2.3.: Common mappings for 360° images

(a) Frame 1 (b) Frame 2

Figure 2.4.: Example frames that optical flow is calculated on



(a) Visualization of optical flow with color: the hue encodes the vector direction, the saturation encodes the vector length for each pixel.

(b) Visualization of optical flow with arrows: The vectors are represented by arrows (only a limited number of vectors are shown)

Figure 2.5.: Optical flow visualizations

These visualizations can help show if and how well an optical flow algorithm is working. Although there are a large number of different algorithms, most of them still do not effectively deal with common issues such as occlusions, too-large displacements and intensity changes [FBK15]. Occlusions are problematic, since the displacements between two images may reveal or cover image areas that, as a result, have no correspondence in the previous image. This problem is exacerbated when displacements are very large (e.g., due to fast-moving objects). Large displacements are also problematic by their very nature, as most algorithms are not designed to handle them. How these limitations affect the use of optical flow for image synthesis will be discussed in Chapters 3 and 4.

2.2. Related Work

Image-based Rendering (IBR) and viewpoint interpolation[4] started attracting interest with the advent of virtual walkthroughs, for example for Apple's QuickTime® VR, in order to

save render time by generating views from images instead of using complex 3D scenes including textures, lights and complex geometric models [Che95]. Chen and Zhang, in their survey on image-based rendering [ZC04], use the terms *source description* and *appearance description* to compare these basic rendering techniques: "Traditional" rendering techniques use *source description*, i.e. the scene is described by the objects within it, their positions and properties. On the other hand, IBR techniques try to achieve the same goal through *appearance description*: Instead of trying to describe the scene through the objects it contains, IBR rendering techniques try to model the *light rays* that reach the viewer. The reason why this is feasible is that human vision itself has little to do with 3D geometry; it is the processing of a dense set of light rays by the brain which are "captured" by the eye. This process can also be performed by a capture device like a camera. As a result, it is not necessary to meticulously model the source, it is sufficient to model the "appearance" of a scene.

While the differentiation between source description and appearance description is helpful in understanding the basic differences between image-based and traditional rendering, many image-based rendering methods also utilize some form of source description, predominantly in the form of 3D geometry. In a different survey on image-based rendering techniques, Kang and Shum [SK00] classify IBR rendering techniques on a continuum, which ranges from techniques using no geometry, to techniques using implicit geometry (or more generally, feature correspondences), to techniques using explicit geometry.

The algorithm presented in this thesis is a combination of two different approaches: the first step uses no geometry whatsoever, and the second step uses feature correspondences to correct some of the problems that arise in the first step. This differentiation guides the related work presented here: The first section presents synthesis approaches using no geometry, and the second section presents approaches using implicit geometry or feature correspondences.

2.2.1. Image Synthesis without Image Correspondences

A theoretical model for image synthesis with no geometry or other image features (i.e., "pure" appearance description) was developed by Adelson et al. [AB91]: the *plenoptic function* (Equation 2.1). The plenoptic function is a 7D function that describes the observable light at every point in space V_x, V_y, V_z, from every direction θ, φ, at every wavelength λ, at every possible point in time t.

$$P = P(\theta, \varphi, \lambda, t, V_x, V_y, V_z) \tag{2.1}$$

In practice, it is infeasible, if not impossible, to cover all dimensions of this function, as this would require a capture device at every location, at every point in time, capturing light rays coming from every direction. However, by making different assumptions, IBR techniques try to reconstruct simplified versions of the plenoptic function. Common assumptions are the reduction of wavelengths to RGB, the removal of the temporal dimension, and the assumption that the light is constant along a ray in empty space (i.e., light does not change its wavelength over distance) [ZC04]. The methods for resampling the plenoptic function, or a lower-dimensional version are diverse, from Light Field Rendering [LH96], which uses

[4] As there seems to be no explicit difference between the terms "image-based rendering", "viewpoint interpolation" and "viewpoint synthesis" in the literature, they will be used interchangeably in this thesis, although "interpolation" is favored for simpler blending techniques, and "synthesis" is used to describe more complex algorithms.

a capturing rig to uniformly sample the surroundings with a very high sample rate, but is able to construct views in real time, to more recent approaches that use neural networks to enhance the samples [NJ18]. The following section presents approaches that concern themselves specifically with 360° images, as this is most relevant for this thesis.

"A Simple Method for Light Field Resampling" [Kaw17]

Kawai [Kaw17] approaches the problem of synthesizing new images with two degrees of freedom without using 3D geometry. Their basic setup is to capture four 360° images at each corner of a rectangular area and use resampling to synthesize a new image anywhere within this area.

The resampling is done by inserting a virtual sphere centered at the synthesized viewpoint representing a projection screen on which to project rays from the captured viewpoints. The locations of the captured viewpoints are known, so the outbound rays of these viewpoints can be calculated by using the image-to-world coordinate conversion from the equirectangular representation. The intersections of these captured rays with the virtual sphere are calculated and the corresponding pixel values are used. The projection screen where no rays have intersected are approximated by repeating the reprojection with different resolutions.

In cases where several rays share an intersection, they propose several methods. The first is to take an average of the rays. As an alternative, they suggest employing a rating based on the inner product of the ray direction and the viewing direction and using the ray with the smallest score. A final option is to prioritize one specific captured viewpoint over the others to completely avoid ghosting artefacts.

To evaluate their method they capture four viewpoints in a scene (the distances between the viewpoints are not mentioned) and calculate an image along a diagonal. They then compare the results of the different ray combination methods by describing visual artefacts.

"On the Use of Ray-tracing for Viewpoint Interpolation in Panoramic Imagery" [SLDL09]

Shi et al. [SLDL09] examine how ray tracing can be used to calculate arbitrary new viewpoints based on knowledge of relative positions between the viewpoints which are stored as cube maps. For every pixel in the target image, a ray is cast into the scene. Since the geometry of the scene is unknown, an alternative method is used to find the correct value of that point. They do this by introducing a color consistency constraint, which compares the pixel values of the rays cast from the reference images. The assumption is that if the colors of different rays are the same, the rays must be intersecting the same point.

In order to calculate an intersection with the scene, they propose two different methods: a brute-force depth search using no scene geometry which searches along all of the captured rays until the pixel values are similar enough to fulfill their color constraint requirements, or a guided depth search using sparse 3D reconstruction.

To evaluate their method, they use a set of five captured input images with a maximum distance of one meter, from which they remove one to use as ground truth. They evaluate the algorithm by comparing the brute-force to the guided depth search based on the artefacts in the results and the computation time.

"Unconstrained Segue Navigation for an Immersive Virtual Reality Experience" [HDR+17]

Herath et al. [HDR+17] propose a system that enables casual users to capture their surroundings with a smartphone in a grid, and then navigate that environment with two degrees of freedom. In order to interpolate between two captured 360° images (1-DoF), they differentiate between faces that are parallel to the axis of movement and faces that are perpendicular to the axis of movement. For faces that are parallel, they stitch the faces of two adjacent viewpoints together and interpolate by using a sliding window. For faces that are perpendicular, they calculate a homography between the faces of two adjacent viewpoints and morph the image accordingly. To interpolate any image within a rectangular area bounded by four captured viewpoints (2-DoF), they recursively interpolate intermediary viewpoints until they reach the desired position.

As the focus of this work is on the whole process of capture, navigation, and viewing, the interpolation step is not explicitly evaluated.

2.2.2. Image Synthesis using Image Correspondences

Leveraging image correspondences for synthesis has been a popular method almost since the beginning of viewpoint synthesis. Chen and Williams [CW93] were one of the first to use "the morphing method" that simultaneously blends the shape and texture of two images using image correspondences. A comparable method, based on optical flow, is used by Richardt et al. [RPZSH13] for planar images.

Adaping planar algorithms (e.g., optical flow, structure-from-motion) for 360° images is a common challenge in 360° image synthesis. Kolhatkar et al. [KL10] and Huang et al. [HCCJ17] solve this problem by extending the faces of the cube maps to account for pixels moving across borders. [KL10] then use optical flow for interpolation between two images; whereas [HCCJ17] estimates the scene geometry with a structure-from-motion algorithm to extend monoscopic 360° videos to stereo. Zhao et al. [ZWF+13] propose a method for adapting sparse correspondence matching for the spherical domain, circumventing the need to use an extended cube map.

Morphing the images to create a new viewpoint can be done with pixel-based blending [RPZSH13], [KL10] or by triangulating the image and calculating homographies between the triangles [HCCJ17], [ZWF+13].

The flow-based blending approach in this thesis builds on the approaches of [RPZSH13] and [KL10], which are presented in more detail in the following sections.

"Megastereo: Constructing High-Resolution Stereo Panoramas" [RPZSH13]

Richardt et al. [RPZSH13] present an approach which combines planar images captured casually on a radius to create a panoramic image that is viewable in stereo in high resolution. In place of scene geometry, they use a cylindrical imaging surface that is concentric to the capture center. For each eye at a given viewing orientation, they project a ray into the scene (Figure 2.6a) and calculate the deviation angle α between the desired ray and the nearest captured rays (Figure 2.6b).

Because linearly blending the rays of the two closest captures would lead to artefacts due to the difference between the real geometry and the cylindrical surface (Figure 2.6c), and

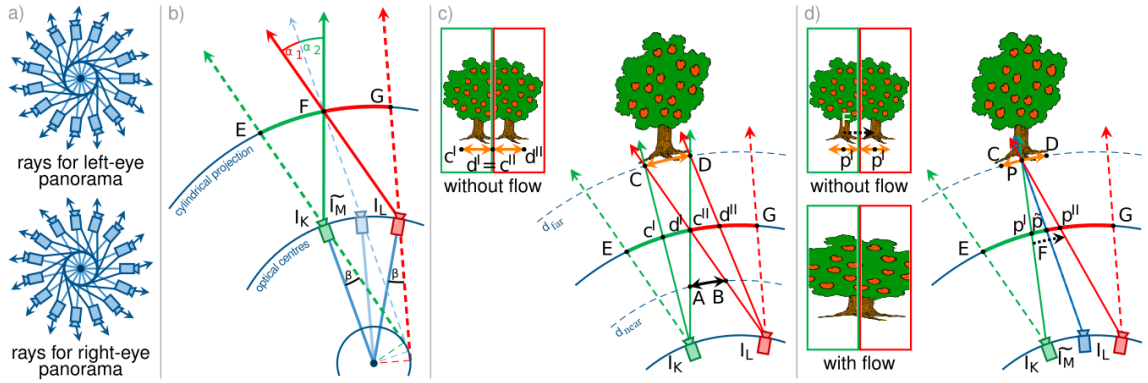


Figure 2.6.: (a) Illustration of rays required for creating a stereoscopic panorama and (b) deviation angles α. (c) Duplication and truncation artefacts caused by the aliasing. (d) Flow-based upsampling to synthesize required rays. *Adapted from [RPZSH13]*

using a nearest-neighbor technique would result in discontinuities, they propose a "flow-based blending" technique: For each ray of the final image that is not a captured ray (deviation angle 0), a new ray is synthesized using optical flow. The vertical image strip captured by the synthesized ray is interpolated by taking the two closest viewpoints I_K and I_L and interpolating $\widetilde{I_M}$ (Figure 2.6d) using the optical flow vectors $F_{k\to l}$ and $F_{l\to k}$. The corresponding strip is then taken from this new viewpoint which contains the matching ray.

The interpolated image $\widetilde{I_M}$ at point η between the images I_K and I_L is calculated by shifting I_K by $\eta \cdot F_{k\to l}$ and by shifting I_L by $(1-\eta) \cdot F_{l\to k}$. The two shifted images are then blended linearly, using η as the weight. Instead of calculating the entire image for each ray, only the necessary image areas are extracted and the interpolation is calculated pixel-wise.

To evaluate their method, they leverage datasets used by other approaches, as well as capturing their own images. They visually compare the results, noting improvements based on visible artefacts.

"Real-Time Virtual Viewpoint Generation on the GPU for Scene Navigation" [KL10]

Kolhatkar and Laganière [KL10] propose a method for smoothly interpolating between pairs of 360° images. Their approach is similar to the approach in Megastereo, where optical flow between the images is used to incrementally morph the two images. Since they use 360° images, they extend the cube map representation to account for points moving across edges, which is the method that is used in this thesis, as well. To reduce artefacts in the obtained optical flow, they perform a matching and smoothing step. They then implement their algorithm on the GPU, which allows them to interpolate between images in real-time.

They evaluate their method by capturing scenes at "reasonable" distances and removing every other image in order to obtain ground truth data. The computation time of the optical flow calculation of the extended cubes is measured, as well as the actual interpolation time. Furthermore, they compare the interpolated results with ground truth images, both visually and by using a per-pixel metric.

	method			evaluation				
	input type	**DoF**	**extracted features**	**defined parameter space?**	**visual eval.**	**error metrics**	**computational cost**	**comparison to other approaches**
[Kaw17]	360° images	2	none	✗	✓	✗	(✓)	✗
[SLDL09]	360° images	2	none/ dense geo	(✓)	✓	✗	(✓)	✗
[HDR+17]	360° images	2	none	✗	✗	✗	✓	✗
[RPZSH13]	planar images	1	dense flow	✗	✓	✗	(✓)	✓
[KL10]	360° images	1	dense flow	✗	✓	✓	✓	✗
[HCCJ17]	360° video	3*	dense geo	✗	✓	✗	✓	✓
[ZWF+13]	360° video	1	sparse feature	(✓)	✓	✓	✓	✓

*on a constrained path

Table 2.1.: Comparing the methods and evaluations of different approaches

2.2.3. Discussion

The overview of the presented approaches (Table 2.1) show that of the presented approaches, most of the approaches using feature correspondences only allow for synthesis with 1-DoF, whereas the approaches using no correspondences allow for synthesis with 2-DoF. Furthermore, most of the approaches do not test their methods on a defined parameter space, and those that do (marked in parentheses) do not test different parameters, but only declare which parameters were used for the minimal tests. Almost all approaches evaluate their results visually, but only two use error metrics. All of the approaches evaluate the computational cost, but for two out of the seven methods (marked in parentheses), there is no real-time requirement.

3. Pixel-based 2-DoF Synthesis of 360° Viewpoints with Flow-based Blending

The approach presented in this chapter combines two types of methods presented in the previous chapter: At its base, the basic pixel-based synthesis uses no geometry or correspondences whatsoever. It is similar to Kawai's "A Simple Method for Light Field Resampling" [Kaw17] in that it also uses a sphere to resample the surroundings. However, where Kawai's work suggests using only one viewpoint for resampling to avoid ghosting artefacts (i.e., doubled edges), the pixel-based synthesis presented in this chapter uses flow-based blending to try to remove ghosting artefacts. The flow-based blending is based on the method presented in Richardt et. al's Megastereo [RPZSH13].

artefact vs artifact

First, the general approach is presented (Section 3.1) and then the details of the implementation are described (Section 3.2).

maybe edit

3.1. Approach

This section presents the assumptions and simplifications made for the pixel-based synthesis of 360° viewpoints (Section 3.1.1), the basic pixel-based approach using proxy geometry (Section 3.1.2) and an improvement to the basic approach based on flow-based blending from Richardt et al.'s Megastereo [RPZSH13] (Section 3.1.3).

3.1.1. Assumptions

In order to simplify the process of synthesis, some assumptions are made based on the scene and the viewpoints in the scene:

- the scene is static
- all images are captured on a plane parallel to the floor (viewpoint plane)
- all synthesized viewpoints are located inside the scene boundaries and are also located on the viewpoint plane
- the positions and orientations of the captures are known
- the scale (max radius) of the scene is known

Furthermore, for the moment, it is assumed that the optical flow algorithm used in the flow-based blending calculates an acceptable result between any pair of viewpoints.

3.1.2. Basic 2-DoF Synthesis

With these assumptions and using a basic proxy geometry with approximately the same scale as the captured scene, it is possible to synthesize new viewpoints with varying accuracy, depending on the scene. The process presented here for basic 2-DoF synthesis is a combination of texture lookup through raytracing, and mosaicking by using a constraint based on the ray deviation angle.

Raytracing-based Texture Lookup

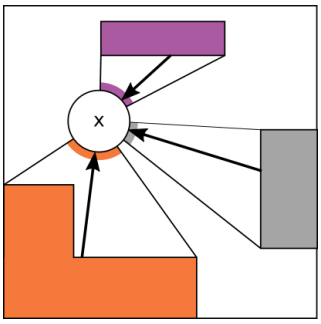
The first step is to map the texture (i.e., pixel values) of an existing viewpoint to a new viewpoint according to its position in the scene. Theoretically, any 360° viewpoint can be mapped to any other, since each 360° image captures every point in the scene. This is only theoretically the case, since image resolution and occlusions in the scene will conceal some areas from some viewpoints whereas they are visible for others. However, at this point, this will be ignored and it will be assumed that each viewpoint image contains all the points of the scene albeit at different image coordinates and different sampling rates[1].

In addition, 3D geometry of the scene is needed for raytracing. However, since the approach in this thesis does not capture or infer any real geometry, a proxy geometry is used that has approximately the same scale as the scene that was captured. The proxy geometry is a sphere, as this is a simple, very general geometry to represent a variety of different scenes. The radius of the sphere is chosen so that the sphere contains all possible points in the scene, for which the scale of the scene needs to be known. Under these assumptions, it is possible to map the image at one viewpoint to a new position by combining raytracing and texture lookup.

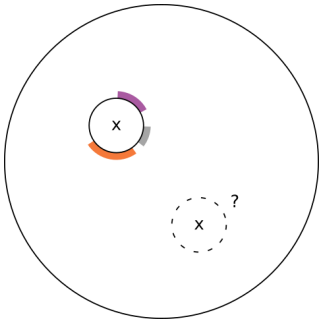
In order to do this, several steps of raytracing are necessary, which are visualized in Figure 3.1. Figure 3.1a shows how a camera at a specific viewpoint captures the light rays reflected from the objects in the scene. The captured pixel values are visualized on a circle around the center of projection of the camera (for simplicity's sake, only one row of pixels is shown). Once the viewpoints have been captured (there is only one viewpoint in this example), a new viewpoint is ready to be synthesized. The proxy geometry is visualized as a circle[2] in Figure 3.1b, with the new viewpoint to be synthesized represented by a dotted circle around a center of projection. For each pixel of the synthesized image, a ray is projected into the scene (Figure 3.1c) and its intersection with the scene is calculated. Then, the ray from the center of projection of the captured viewpoint to the scene intersection is calculated, which, when normalized, is equivalent to a unit direction vector of the unit sphere. Using the unit direction and the image mapping function, the pixel value at that position is retrieved (Figure 3.1e) and copied back to the new viewpoint (Figure 3.1f). In this way, the pixel values (i.e., texture) of a captured viewpoint are mapped to the new viewpoint (Figure 3.1g). Figure 3.1h compares the mapped values to the actual scene. It is immediately clear that most points have the value they would have had, had the viewpoint been captured instead of synthesized (ground truth value); however some are incorrect. This is due to the disparity between the proxy geometry and the real scene geometry.

[1] By design, areas closer to the camera are captured with a higher sampling rate per point than areas farther away.

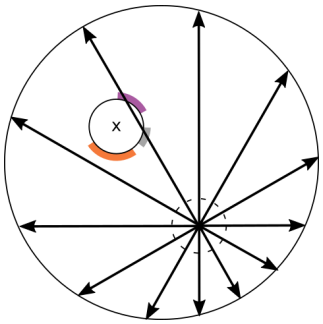
[2] In this example, the sphere does not surround the complete scene (the corners of the scene are outside the circle). This is only for visualization purposes, normally the sphere would contain the complete scene, including the corners.



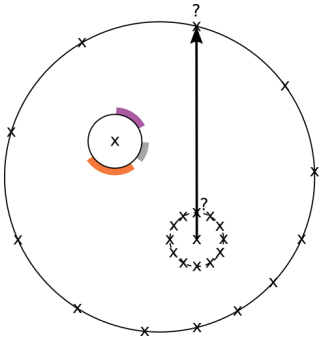
(a) The real scene viewed from above: The camera captures light rays reflecting from objects



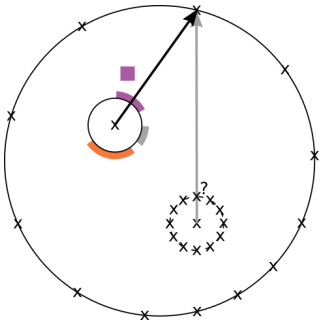
(b) A new viewpoint to be synthesized using the proxy geometry (sphere)



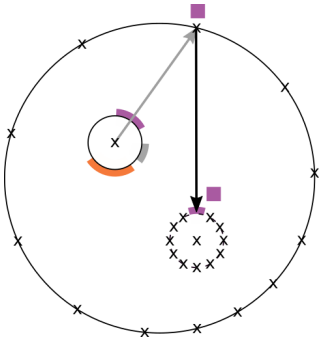
(c) A ray for each pixel in the synthesized viewpoint is traced to its intersection with the proxy geometry



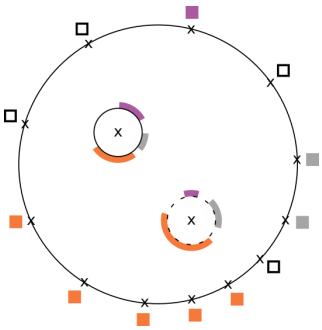
(d) For each ray, a texture lookup is performed



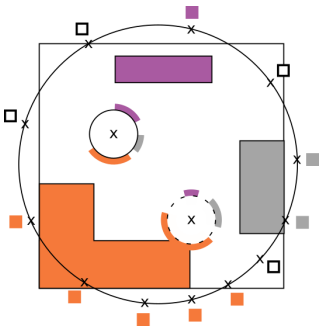
(e) From the intersection of the ray and the proxy geometry, a second ray is traced to the center of projection of the captured viewpoint and the pixel value is looked up



(f) The pixel value is copied back to the new viewpoint



(g) This process is repeated for all pixels



(h) The resulting texture mapping from the captured viewpoint to the proxy geometry in comparison with the original scene

Figure 3.1.: Process of texture lookup through raytracing

Deviation-angle-based Mosaicking

The example in Figure 3.1 contains only one captured viewpoint, so the choice of which viewpoint to use for texture lookup is trivial. In cases where several viewpoints are available, a choice must be made as to which viewpoint should be used. In the case where the real scene has the same geometry as the proxy sphere, this choice is inconsequential, since the raytracing is always accurate. However, this scenario is unrealistic, since the number of spherical rooms containing no objects is negligible. Thus, as soon as the real scene differs from the proxy sphere, some viewpoints yield better results than others. Figure 3.2a shows how a discrepancy between the real scene and the proxy sphere can lead to inaccurate results.

When comparing rays from different viewpoints, two metrics can be examined: the euclidean distance of the captured viewpoint from the synthesized viewpoint and the deviation angle between the rays. Figure 3.2b visualizes the two metrics and in the example, the $vp2$ with the smaller deviation angle, is a better match. In fact, assuming that there is no obscuring element in the air such as fog, and disregarding diffusion and scattering over distance, the same light ray is captured by any viewpoint located on the ray in question. This means the closer the deviation angle is to zero, the more accurate the result will be, regardless of the distance of the viewpoints. In this simplified scenario, the best viewpoint would have a deviation angle of zero (Figure 3.2c). However, sampling rates and resolution also have an effect on the sampled point, so the euclidean distance cannot be completely ignored.

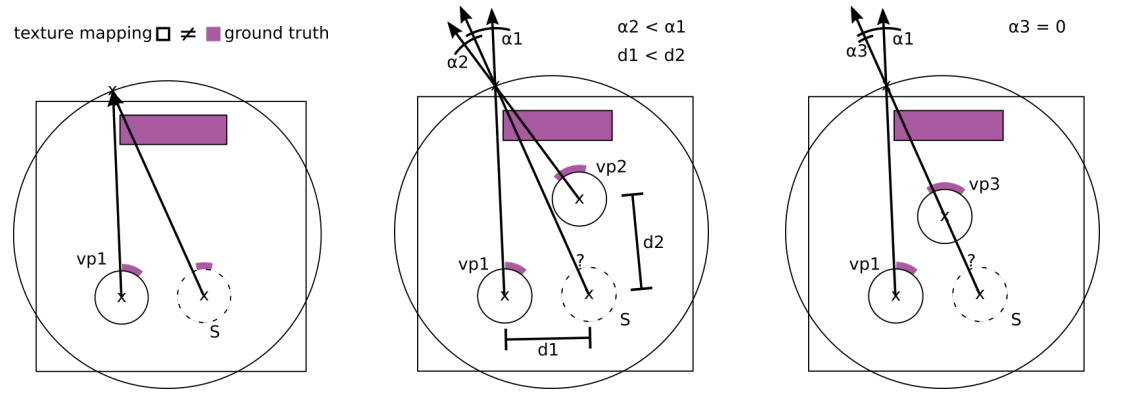
Instead of combining these metrics in a function that might have to be weighted differently depending on the scene size, the distribution viewpoints, or other factors, the choice of which viewpoints to use is divided into two different steps: the choice of input viewpoints from all captured viewpoints for the synthesis of a specific point and the choice of which of the input viewpoints to use for each ray of the synthesized point.

The pre-selection of input viewpoints from all captured viewpoints is based on the assumption that the closer the captured viewpoints are to the synthesized viewpoint, the more accurate the sampling of the surroundings will be (i.e., the relative size of the objects). As a result, all viewpoints further than a certain distance are discarded from the input.

After selecting the most appropriate captured viewpoints, the synthesized image is created by comparing the deviation angles of these viewpoints for each ray (i.e., pixel). The two closest viewpoints are then blended together on a per-pixel basis, so that there are no abrupt edges between mosaic areas. The blending function is presented in more detail in Section 3.2.

3.1.3. 2-DoF Synthesis using Flow-based Blending

Using this basic 2-DoF synthesis works fairly well as long as the real scene geometry corresponds roughly to the proxy sphere geometry. The basic shape of many rooms can be approximated by a sphere; however the objects within these rooms can diverge greatly from the proxy geometry. In these cases, ghosting and doubling artefacts become visible, such as areas appearing twice, not at all, or two areas overlapping inconsistently. This problem is exacerbated when the synthesized viewpoint is very close to an object, as is visualized in Figure 3.3: In this example the synthesized viewpoint is very close to a detailed object whose geometry diverges significantly from the proxy sphere geometry. The values of the points captured by the two viewpoints $vp1$ and $vp2$ differ (orange and purple), and neither of them is the desired ground truth value (gray) (Figure 3.3a). In order to improve the result, an adapted variation of the flow-based blending method from Richardt et al.'s al.'s Megastereo



(a) Differences in the real scene from the proxy geometry can lead to inaccurate texture mappings

(b) The deviation angle α of the rays can be compared, as well as the euclidean distance d

(c) The optimal case (disregarding sample resolution) is when the deviation angle is 0, as for viewpoint 3 in this example

Figure 3.2.: Choosing the appropriate viewpoint to improve the result

[RPZSH13] is introduced. This method allows interpolation between two 360° viewpoints using optical flow. Figure 3.3b shows how the interpolation can be used to achieve a more accurate result: A new viewpoint $vp1\text{-}2$ is interpolated between $vp1$ and $vp2$ such that the interpolated viewpoint is located on the ray in question[3] This new viewpoint is then used for the texture lookup to create the synthesized image with the goal of improving the accuracy of the mapped point.

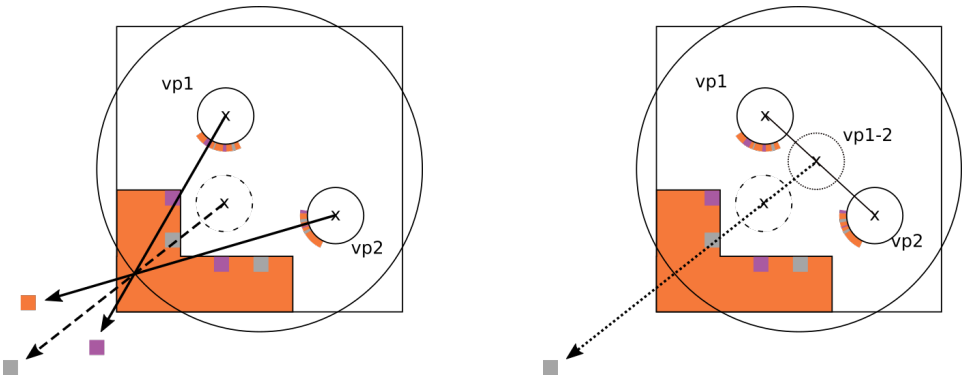
Adapting Flow-based Blending in Megastereo for 1-DoF Interpolation of 360° Images

Megastereo [RPZSH13] aims to generate high-resolution stereo panoramas by combining images captured on a circle. Their approach is to combine corresponding strips of the captured images and to create a view for each eye (see Section 2.2.2). In order to mitigate artefacts such as ghosting, they use "flow-based blending" to combine two images A and B. This consists of using the optical flow vectors $F_{A\to B}$ and their inverse $F_{B\to A}$. To get the interpolated image at position δ between image A and B, first, image A is shifted by $\delta \cdot F_{A\to B}$ and image B is shifted by $(1-\delta) \cdot F_{A\to B}$, yielding I_A and I_B, respectively. Then, I_A is multiplied by $(1-\delta)$ and I_B by δ and these pixel values are added together to give the resulting interpolation. This is described by the following function, in which each pixel at position x of the synthesized image S is defined by:

$$\begin{aligned} S(x) = (1-\delta) \cdot A(x + \delta \cdot F_{A\to B}(x)) \\ + \delta \cdot B(x + (1-\delta) \cdot F_{B\to A}(x)) \end{aligned} \tag{3.1}$$

The flow-based blending in Megastereo operates on planar images. In order to use it for 360° synthesis, it is necessary to adapt the method for 360° images.

[3]Positioning the interpolated viewpoint directly on the ray is only possible for rays that are on the 2D plane containing all the viewpoints. All other cases must be approximated.



(a) Detailed areas where real geometry is very different from the proxy geometry are problematic for basic 2-DoF Synthesis

(b) Interpolating between input viewpoints using optical flow may improve results

Figure 3.3.: Introducing flow-based blending to improve accuracy

A 360° image can be projected in several ways, as described in Section 2.1.1. The output of these projections is a planar image, meaning that it would be possible to apply flow-based blending directly. However, optical flow algorithms are generally designed to handle planar images without seams or distortions and would most likely produce unexpected results if used naively on planar projections of 360° images. As a result, the 360° images must first be projected and adapted in such a way that optical flow can be calculated accurately on them.

Of the projections presented in Section 2.1.1, only the cube map representation is applicable. Spherical representations are impractical, as aligning seams is not feasible and the distortion towards the edges is extreme. The equirectangular representation has only four seams to handle, but also distorts the image greatly around the poles. The cube map representation contains a number of seams but does not distort the image more than a planar image would. The challenge presented by the many seams of the cube map is to be able to track the points that move across the seams created by the six faces. Figure 3.4 shows an example of different points moving across seams, illustrating why calculating optical flow on each face separately would not be enough to track the points moving across the seams. Figure 3.4 also shows the linear discontinuities at the seams (e.g., at the upper edge of the carpet): Since each cube face was captured by a different virtual camera, angles are not consistent across seams. As a result, it is not possible to use the cube map as it is, since the optical flow assumes linear movement[4].

To solve this problem, an *extended* cube map is introduced, which was also used by Huang et al. [HCCJ17] and Kolhatkar et al. [KL10] to adapt 360° images for structure-from-motion and optical flow algorithms. Instead of projecting a field of view of 90° for each camera, which covers exactly 360° of the image, the extended cube map uses a larger field of view for each camera (in this case, 150°). Consequently, some areas of the scene are represented several times, as the areas of the image that are near a seam are represented on each face that is adjacent to the seam. This way, when calculating optical flow on each face separately, points that move across where the seam would be in a regular cube map remain on the face

[4]Optical flow uses vectors to describe movement, which are inherently linear

(a) Viewpoint A in cube map representation (90° FoV per face)

(b) Viewpoint B in cube map representation (90° FoV per face)

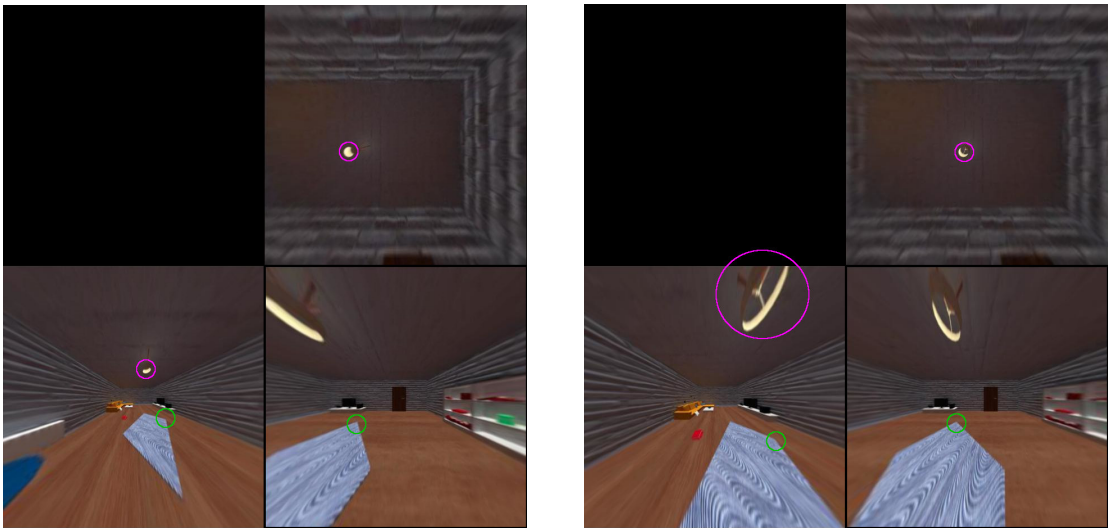
Figure 3.4.: Points in the scene moving across seam edges need to be tracked by optical flow (the back face of the cube map was omitted for simplicity's sake)

with the corresponding projection. Figure 3.5 shows the front, top, and left faces of the extended cube map representation of the cube map shown in Figure 3.4. In the extended cube map, the tracked points do not traverse a seam on any of the faces, meaning optical flow can be calculated on the entire original face.

Nonetheless, this method is still limited by the field of view used by the virtual cameras. If the maximum displacement is larger than the face extension, the extended cube map will not be sufficient, as the points can also traverse the extended seams. Also, the larger the field of view, the more the image will be distorted towards the edges of a face, which may lead to distorted optical flow results. Both problems are visible in Figure 3.5: The lamp in the left face has such a large displacement between a and b that it is partly cut off in b. On top of being cut off, it is also greatly distorted, which will result in distorted optical flow values for that area.

In general, this means that displacement between two images is limited. However, the displacement that is trackable by optical flow algorithms is also limited. The effect of these limitations will be explored in Chapter 4.

Despite these limitations, using the extended cube map makes it possible to calculate optical flow on each face separately, meaning that Megastereo's flow-based blending method can be applied to two 360° viewpoints A and B: First, the extended cube map projections A_{ext} and B_{ext} are created from the image data. From this point, each set of faces A^i_{ext} and $B^i_{ext}, i \in [top, left, front, right, bottom, back]$ is handled separately. Optical flow $F^i_{A \to B}$ and inverse optical flow $F^i_{B \to A}$ are calculated for A^i_{ext} and B^i_{ext}. Then, the shifted image is calculated using Equation 3.1. Finally, for each face, the extended parts of the extended cube map are clipped so that each face once again has a field of view of 90°, resulting in the blended 360° image at position δ between viewpoints A and B. Since the flow-based blending method is applied to two complete images instead of image strips like in Megastereo, it is equivalent to interpolation with one degree of freedom (i.e., on a line) between viewpoints A



(a) Viewpoint A in extended cube map representation (150° FoV per face)

(b) Viewpoint B in extended cube map representation (150° FoV per face)

Figure 3.5.: Points that traversed a seam in the regular cube map can be tracked across the original seams in the extended cube map

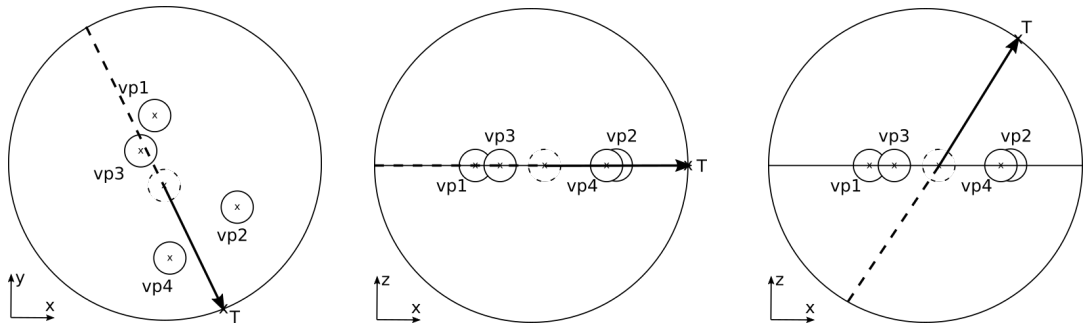
and B. This interpolated viewpoint can then be used for texture lookup just like a captured viewpoint.

2-DoF Synthesis with Flow-based Blending

Adapting Megastereo's flow-based blending for 1-DoF interpolation of 360° images allows the creation of a new viewpoint between A and B that is closer to the actual ray (Figure 3.3b). In order to leverage this to improve the basic 2-DoF synthesis, the 1-DoF interpolation needs to be integrated in the 2-DoF synthesis algorithm. For each pixel of the synthesized image, a set of input viewpoints A and B needs to be chosen for use in the 1-DoF interpolation. Then, based on the positions of A and B, and the ray in question, an interpolation distance δ must be calculated that defines the position of the 1-DoF interpolation between A and B.

For these steps, an approximation needs to be made due to the 2-DoF restriction: Figure 3.6a and b show the ideal case, where the target point T is on the same horizontal plane as the captured points and the synthesized point (the viewpoint plane). In this case, there are a number of different positions directly on the ray that are also on the plane. Depending on the input viewpoints, the most convenient can be chosen and an interpolated viewpoint calculated at that position. However, this is only the case for all target points *on this plane*. All other target points in the scene lie either above or below the viewpoint plane, for example in Figure 3.6c, where T is above the plane. In these cases, the only intersection of the ray and the viewpoint plane is at the synthesized viewpoint. Finding the set of viewpoints A and B that allow the closest 1-DoF interpolation to this point is not trivial, as it would require comparing the minimum distance of the point to all vectors between all the possible sets of viewpoints. In order to simplify this problem, all the rays that are not on the viewpoint plane are approximated.

To approximate a ray pointing at target point T at the spherical coordinates (r, θ, φ), the

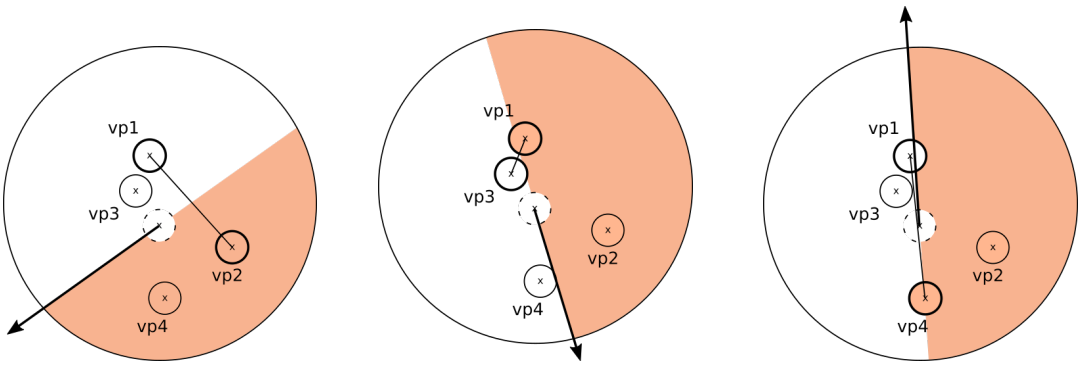


(a) Scene view from above: The target point T is on the viewpoint plane

(b) Scene view from the side: The target point T is on the viewpoint plane

(c) Scene view from the side: The target point T can also be above (or below) the viewpoint plane

Figure 3.6.: Example of different target points in the scene



(a) A: vp1, B: vp2

(b) A: vp1, B: vp3

(c) A: vp1, B: vp4

Figure 3.7.: Examples of the choice of viewpoints A and B for 1-DoF interpolation based on deviation angle and position on either side of the ray. The two sides of the ray are color coded in white and orange.

elevation θ is reduced to 0, assigning the spherical coordinates to $(r, 0, \varphi)$, which is equivalent to moving T to the viewpoint plane by the shortest path. This will yield less accurate results as the actual ray moves towards the poles, since the deviation angle between the actual ray and the approximated ray increases towards the poles. However, this approximation is computationally and mathematically much simpler and should yield acceptable results.

With this approximation, the viewpoints A and B used for 1-DoF interpolation can be chosen. As in basic 2-DoF synthesis, the metric for choosing A and B is the deviation angle. The actual rays, not the approximated rays, are used for the calculation and comparison of the deviation angles, since there is no need to use the approximated rays at this point. However, for the choice of A and B, an additional constraint is included: The two viewpoints chosen must be on either side of the approximated ray, so that there is an intersection between the vector connecting the viewpoints A and B and the approximated ray (see Figure 3.7).

Once the viewpoints A and B are chosen for each target point T, the interpolation distance $\delta \in [0, 1]$ is calculated, which is the point on the vector $\overrightarrow{AB}$ that intersects the approximated ray. The calculation of δ is a simple line intersection calculation, explained in Section 3.2.

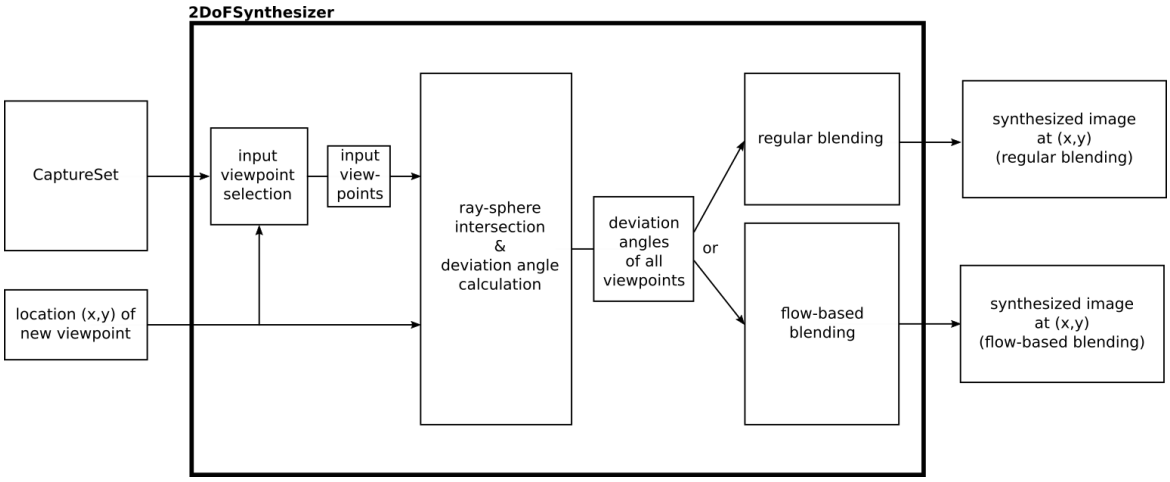


Figure 3.8.: System diagram of the 2DoFSynthesizer

Using the chosen viewpoints A and B, and the calculated interpolation distance δ, a 1-DoF interpolation is calculated for each approximated ray. Using this interpolated viewpoint, a texture lookup is then performed for each pixel associated with that ray. This results in a mosaicked image where each image area (vertical strip in equirectangular representation) is an interpolated, reprojected viewpoint. The 1-DoF interpolation step should improve some of the artefacts caused by the use of the proxy sphere instead of the actual scene geometry. Its effectiveness and limitations are explored in Chapter 4.

3.2. Implementation Details

This section presents the technical and mathematical details on which the basic 2-DoF synthesis with regular blending and the 2-DoF synthesis with flow-based blending are based. The basic 2-DoF synthesis with regular blending and the 2-DoF synthesis with flow-based blending are implemented in Python3 [Pyt18], using the libraries NumPy [The19b], OpenCV [The19a], SciPy [The20], and scikit-image [vdWSN+14]. For the conversion between different 360° projections, as well as the calculation of the extended cube map, the library "skylibs" [Hol20] is used. The synthesis process is shown in Figure 3.8: The captured data, which is encapsulated in the CaptureSet class, is passed to the *2DoFSynthesizer*, along with the desired location of the viewpoint to be synthesized. Using this data, the input viewpoints are selected, and passed on for ray-sphere intersection and deviation angle calculation. Then, depending on whether regular blending, or flow-based blending is used, the blending is performed using the results of the previous step, producing the synthesized image.

3.2.1. Preprocessing

The input data consists of a set of captured viewpoint images in equirectangular representation, a text file containing the metadata (positions and orientations) of these viewpoints, and the approximate scene radius. In order to easily and intuitively access the locations and image data of the captured viewpoints, the data is encapsulated in the CaptureSet class. The CaptureSet class first parses the metadata; then, with this information, rotates all the

images so that they have the same orientation, and shifts the set of viewpoints so that it is centered around the origin (0,0,0). This is done under the assumption that images were captured in a regular distribution throughout the room. The proxy sphere representing the scene is also centered at the origin. Instead of storing the image data directly in the CaptureSet, the file paths are stored so that the images can be dynamically loaded when needed. The CaptureSet can then be used by the 2DoFSynthesizer to synthesize a new image either using regular blending or flow-based blending at any given location within the scene[5].

3.2.2. Basic 2-DoF Synthesis using Regular Blending

For the basic 2-DoF Synthesis, the steps described in detail in this section are the selection of input viewpoints, the calculation of the ray-scene intersection and deviation angles, as well as the blending function introduced in Section 3.1.2 and the texture lookup which reprojects the captured viewpoints.

Selecting Appropriate Input Viewpoints

Before calculating ray-scene intersections and performing texture lookup, the most appropriate input viewpoints are selected from the complete set of input viewpoints. In reality, resolution does play a role in the resampling, meaning that captured viewpoints that have a large euclidean distance from the synthesized viewpoint will not necessarily yield the "correct" value. Since the euclidean distance is not factored in to the weighting function (see 22), the captured viewpoints are filtered before synthesis. All viewpoints outside a certain radius (approx. 1m) are discarded. If the set of viewpoints within the radius is empty, all viewpoints are used, regardless of distance.

Calculating Ray-sphere Intersection

The ray-sphere intersection is used in the raytracing-based texture lookup to find the scene points captured by the synthesized viewpoint (see 3.1c). This raytracing process is a basic raytracing technique used in computer graphics. The vectors representing the rays of a viewpoint can be easily derived from the unit directions of the 360° image (see Section 2.1.1): Each unit direction is a vector on the unit sphere, representing the location of an image value (pixel). These coordinates are in *model space*, meaning that they are centered around zero. Translating them into *world space* moves the rays to their respective location in the scene. From this location, the rays represented by the vectors are cast into the scene, where they will intersect with the proxy geometry.

The intersections of these rays with the proxy sphere geometry can be calculated analytically: The proxy sphere, which is centered at the origin, can be represented implicitly by Equation 3.2. A point P defined by this equation represents a point on the surface of the sphere (Equation 3.3). The equation describing any point on the ray can be expressed by Equation 3.4, where O is the origin of the ray, which is the center of projection of the new viewpoint, t is the length of the ray and D is a unit vector describing the direction.

[5]Given that it is within the convex hull of the captured viewpoints

$$x^2 + y^2 + z^2 - R^2 = 0 \tag{3.2}$$
$$P^2 - R^2 = 0 \tag{3.3}$$

$$P = O + tD \tag{3.4}$$

The point P in Equation 3.3 can be substituted with the equation of any point on the ray which yields Equation 3.5. This equation can be developed into Equation 3.6, which is a quadratic function with $a = D^2$, $b = 2OD$, $c = O^2 - R^2$ (Equation 3.7).

$$|O + tD|^2 - R^2 = 0 \tag{3.5}$$
$$D^2t^2 + 2ODt + O^2 - R^2 = 0 \tag{3.6}$$

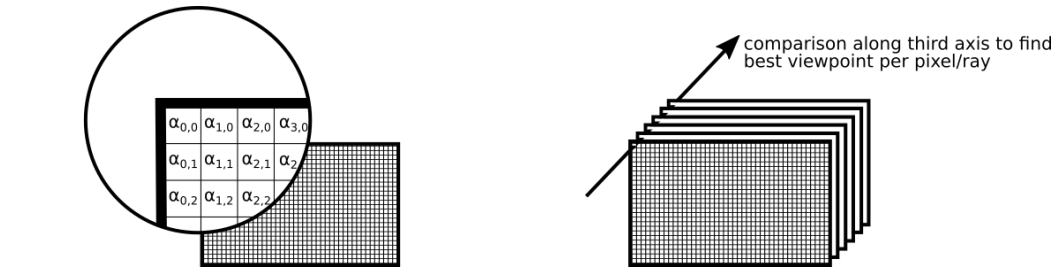
$$a = D^2, b = 2OD, c = O^2 - R^2$$
$$f(t) = at^2 + bt + c \tag{3.7}$$
$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{3.8}$$

This equation can then be solved for t. Since the radius of the sphere is chosen so that it contains the complete scene and no viewpoints are synthesized outside of the scene, the quadratic function will always have two solutions (i.e., two intersections): one for which the vector length t is negative, and one for which the vector length t is positive. Since the original ray used for the calculation is unidirectional (i.e., it cannot invert its direction), it needs to be extended by a positive value. The original ray, being a unit ray of length 1, can then be multiplied by the positive t, which yields the intersection point.

The vectors and intersection points are each calculated and stored in latlong representation (i.e., a matrix of vectors of the same shape as the latlong image), which means that they can easily be associated with the unit directions, as well as the uv coordinates (and thus, pixel values) using the latlong mapping function. By storing the values in this representation (i.e., 3D matrix), Numpy's vectorization can be used, which greatly facilitates implementation.

Deviation Angle Calculation

The deviation angle calculation is a simple angle calculation between vectors. This calculation is performed for each ray of the synthesized point and the corresponding rays of all of the input viewpoints. The rays are defined by their origin (location of the viewpoint), and the intersection points calculated in the previous step. The deviation angles per viewpoint are stored in latlong representation (Figure 3.9a). The deviation angles for all viewpoints are stacked in a three-dimensional matrix, which allows comparison of the deviation angles per pixel/ray of all the viewpoints (Figure 3.9b) in the next steps.



(a) The deviation angles of one input viewpoint in latlong format

(b) The stacked deviation angles of all input viewpoints

Figure 3.9.: Visualization of deviation angle storage

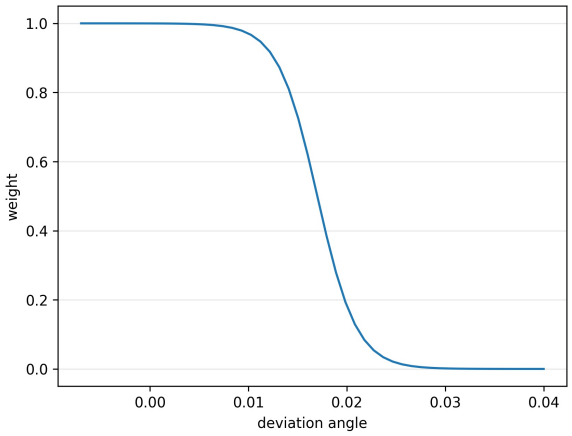


Figure 3.10.: The inverse sigmoid function used for weighting

"Regular" Knn Blending

The previously calcuated and stored deviation angles are used for the regular blending step. The storage of the deviation angles in the three dimensional matrix (Figure 3.9b) makes the selection of the best viewpoint per pixel very straightforward, since the id of the k best input viewpoints can easily be extracted and the corresponding pixel value retrieved. With this information, the "regular", k-nearest-neighbor (knn) blending is performed.

The regular blending function combines the values of the rays of the k closest deviation angles α. The idea is to weight deviation angles of 0 very highly and all larger deviation angles with exponentially low values. This is done with an inverse sigmoid function (Equation 3.9, visualized in Figure 3.10). The parameters of the function were found by trial and error.

$$w(\alpha) = \frac{1}{(1 + e^{500 \cdot (\alpha - 0.017)})} \tag{3.9}$$

Then, the per-pixel weights w are normalized so that their sum for each pixel is one. In this way, the final image is not oversaturated. Blending the two best viewpoints per pixel results in smoother transitions between the mosaicked areas. Figure 3.11 shows the difference between using $k = 1$ and $k = 2$: It is clearly visible that the border between two mosaicked areas (e.g., on the rug in the center of the room) is very abrupt in Figure 3.11a, because the

(a) $k = 1$ (b) $k = 2$

Figure 3.11.: K-nearest-neighbor blending with different values for k. The images are clipped from the latlong representation of an image synthesized using regular blending.

two areas do not align perfectly. This border is much smoother in Figure 3.11b, since the two areas are gradually blended into one another. Using $k > 2$ does not have much impact, since most deviation angles where $k > 3$ are too large to have an effect.

Texture Lookup

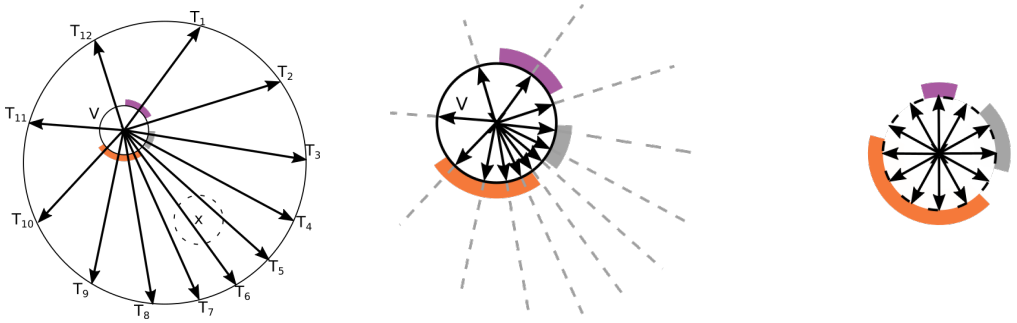
Finally, using the viewpoints selected through the deviation angles, the texture lookup shown in Figure 3.1d-f is performed by resampling using uv coordinates. Given a captured viewpoint at the location V and the ray-scene intersections from the synthesized viewpoint T_i, where i denotes which ray is being examined, the rays from the V to the intersections can easily be calculated by $T_i - V$ (see Figure 3.12a). These rays are then normalized to have length 1 and returned to model space (see Figure 3.12b). The normalized rays in model space are in the same format as the unit directions and can therefore be transformed to image coordinates using the latlong mapping function (see Section 2.1.1), implemented in the library Skylibs [Hol20]. These image coordinates (i.e., uv coordinates) can be used to resample the data, which is equivalent to actually performing a ray-by-ray texture lookup, but much faster. The result of the resampling along with the "new" rays is shown in Figure 3.12c. This resampling based on uv coordinates is also implemented in Skylibs and utilizes Scipy's function *scipy.ndimage.map_coordinates*. Finally, the pixel values of the different remapped viewpoints are multiplied by the normalized weights associated with that viewpoint and the weighted latlong images are added up to give the final synthesized image.

3.2.3. Flow-based Blending

The 2-DoF synthesis with flow-based blending also utilizes the ray-sphere intersection, the deviation angle calculation and the texture mapping. The details of the flow-based blending steps, i.e., the input viewpoint selection, the choice of viewpoints A and B for the 1-DoF interpolation, the calculation of the interpolation distance δ, as well as the 1-DoF interpolation are explained in this section.

Selecting Appropriate Input Viewpoints

The adaptation of optical flow algorithms for 360° images, as well as most optical flow algorithms themselves, are limited to a maximum displacement. This is not considered in



(a) The rays $T_i - V$ in world space

(b) The normalized rays $T_i - V$ in model space

(c) The result of resampling based on uv coordinates

Figure 3.12.: Texture lookup by uv remapping

the algorithm itself, so it is handled in the input viewpoint selection. The goal of the input selection is to find a minimal convex hull around the point to be synthesized from the set of captured viewpoints. If there is at least one captured viewpoint in each quadrant around the synthesized viewpoint, then the closest point from each quadrant is selected. If there is no point in one of the quadrants, the points in the quadrants adjacent to the empty quadrant are selected so that they are as close to the empty quadrant as possible. Since the synthesized point is in the convex hull of all the captured points, there must exist a solution for the minimal convex hull. In the worst case, the minimal convex hull is so large that the optical flow algorithm fails.

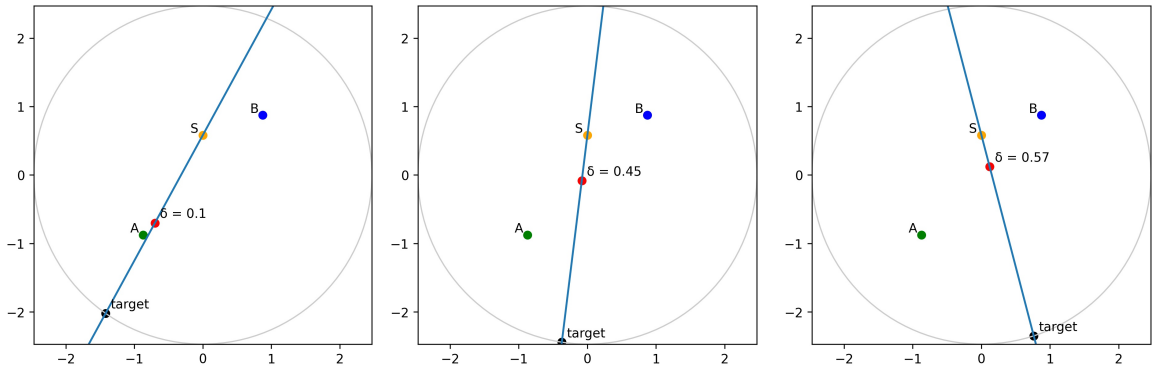
Choosing the Viewpoints A and B

The choice of the input viewpoints A and B for the 1-DoF interpolation is important in that two viewpoints need to be chosen that are on either side of the ray in question (see Figure 3.7), so that the line on which the images can be interpolated actually intersects the approximated ray. Since all the viewpoints are on a plane, the "side" a viewpoint is on is defined by whether the deviation angle is between 0 and 180° (one side) or between 180 and 360° (other side). To get the best deviation angles on either side, the angles are sorted and the angles closest to 0 and closest to 360 are chosen.

The only problem with this process would arise if there was no viewpoint on the other side of the axis. However, because of the restriction that all synthesized points must be within the convex hull of the captured points, this can never happen. In the worst case scenario, there is only one viewpoint on the other side of the axis with a very large deviation angle. This is acceptable, since this viewpoint is not directly used, instead the 1-DoF interpolation creates a new viewpoint with an ideally very small deviation angle.

Determining the 1-DoF Interpolation distance δ

Given the two viewpoints A and B, it is now possible to calculate the intersection of $\overrightarrow{AB}$ and the approximated ray (elevation $\theta = 0$) between the synthesized point S and the target point P in the scene (Figure 3.13). The calculation of the intersection between two lines is a method adapted from [Wei]. Using these four points A,B,P and S, two infinite lines can be defined (Equation 3.10). The intersection point at $t * \overrightarrow{AB}$ is given by Equation 3.11.



(a) (b) (c)

Figure 3.13.: Depending on the scene point, a different δ is calculated based on the intersection of $\overrightarrow{AB}$ and $\overrightarrow{S, target}$

$$\overrightarrow{AB} = A + t * (B - A) \quad \overrightarrow{SP} = S + u * (P - S) \tag{3.10}$$

$$t = \frac{(x_A - x_S)(y_S - y_P) - (y_A - y_S)(x_S - x_P)}{(x_A - x_B)(y_S - y_P) - (y_A - y_B)(x_S - x_P)} \tag{3.11}$$

Since the synthesized viewpoint S is within the convex hull, there is always an intersection t $\in [0, 1]$. This value can then directly be used as δ. The only case that merits an exception is if the synthesized viewpoint is directly on the border of the convex hull, i.e., directly on the line between two captured viewpoints. In this case, the ray that is parallel to vector $\overrightarrow{AB}$ is equal to the line AB. As a result t is not defined and δ must be found by dividing the distance $|\overrightarrow{AS}|$ by $|\overrightarrow{AB}|$.

1-DoF Interpolation

Given the two viewpoints A and B and the interpolation distance δ, the interpolated image at δ between A and B can be calculated. The different steps in the process are: extending the cube map, calculating optical flow on the extended cube map, shifting the images by the optical flow, and transforming the shifted, extended cube map back into latlong representation so that it can be remapped in order to be used for blending.

Extended Cube Map The class *ExtendedCubeMap* uses the 360° image data and the virtual camera provided by skylibs [Hol20] to "extend" the cube map by capturing a virtual image for each face of the cube with a 150° field of view. This is done for both viewpoints A and B. The *ExtendedCubeMap* class handles the extended faces and can perform different functions on them, such as the optical flow calculation, which is required for the next step.

Optical Flow Calculation and Image Shifting The optical flow algorithm used in the implementation is Farnebäck's algorithm implemented by OpenCV (*cv2.calcOpticalFlowFarneback*) [The19a]. For calculating optical flow between two *ExtendedCubeMaps* A and B, the *ExtendedCubeMap* class provides a function *optical_flow*, which takes as arguments the optical flow

algorithm, and a second *ExtendedCubeMap* to use for optical flow calculation. Passing the optical flow function dynamically greatly simplifies exchanging the optical flow algorithm for future work.

The *ExtendedCubeMap* then calculates the optical flow *separately* between each corresponding pair of extended faces. On top of the optical flow from *ExtendedCubeMaps* A to B, the inverse optical flow from *ExtendedCubeMaps* B to A is required as well. The inversion of optical flow is nontrivial, since inverting the optical flow vectors is not enough: In order to calculate the inverse optical flow field, the original optical flow field must first be shifted by its own vectors and then inverted: For example, a vector (1,3) at position (10,10) in the optical flow field must be shifted to position (10+1, 10+3) and then reversed to (-1, -3). Alternatively, the optical flow can just be calculated on the *ExtendedCubeMaps* in reverse, i.e., from B to A. This option is used in the proof-of-concept implementation in order to avoid bugs, and since the impact on performance is not very significant for images with small resolution, which are used the experiments presented in Chapter 4.

Using the optical flow, the inverted optical flow, and the interpolation distance δ, the shifted images I_A and I_B are calculated separately for each face by again using Scipy's *scipy.ndimage.map_coordinates*, with image coordinates shifted by the optical flow vectors and δ, and the inverse optical flow vectors and $(1 - \delta)$, respectively. The shifted *ExtendedCubeMap* images are then combined by multiplying them by $(1 - \delta)$ and δ, respectively, and adding the pixel values. The result is an interpolated *ExtendedCubeMap* at interpolation distance δ.

Before passing this on to the blending step of the 2-DoF synthesis, the *ExtendedCubeMap* is transformed back into a regular cube map by clipping each face back to its original size and mapping the cube map back to a latlong map, since all other processing steps use the latlong format.

Flow-based Blending

The flow-based blending step combines all of the previous steps: First, for each ray, the viewpoints A and B are selected and the interpolation distance δ is calculated. Then, a mask is created for each set (A, B, δ), masking all pixels that do not belong to the set, and the interpolated image for that set is calculated and reprojected to the location of the synthesized viewpoint. This is repeated for all sets.

The complexity of the flow-based blending is directly bound to the precision of δ. Given a precision of two decimal points results in 101 different interpolated images ($\delta \in$ [0.00, 0.01, 0.02 ... 0.99, 1.00]), whereas precision of one decimal point results in only 11 calculated images. A precision of two is used in the implementation, since this is an acceptable compromise between complexity and a result with smoother transitions.

Finally, the masked interpolated latlong images are added together to give the final result.

3.2.4. Performance

As this is a first attempt at 2-DoF synthesis, performance was not deemed important, and no parallelizations were incorporated. Since the algorithms operate in a pixel-wise fashion, the computation time increases exponentially for larger image resolutions ($O(n^2)$ complexity). The performance of the flow-based blending depends on the location of the synthesized point in relation to the captured points. In the worst case, one interpolated image must

be calculated per pixel, at best one interpolated image must be calculated for the whole image (if a synthesized viewpoint is directly on a line between two captured viewpoints). The non-optimized interpolation of a whole image for pixel regions (and in the worst case, single pixels), makes the flow-based blending significantly slower than the regular blending.

The synthesis of an image of 1000x500 pixels with no optimization using regular blending with 4 input viewpoints takes approximately 4s on a single Intel Xeon (Skylake) processor, whereas flow-based blending with the same input takes between 10 and 30 minutes, depending on the constellation of the viewpoints. Fortunately, many of the operations are "embarrassingly parallel", meaning they can be very easily parallelized in the future.

3.2.5. Implementation-related Problems

Unfortunately, there are some implementation-related problems that are unrelated to the algorithm, but nonetheless have an effect on the result. The extension of the regular cube map to the *ExtendedCubeMap* with the skylibs library slightly distorts the image for an as-of-yet unknown reason, so that when the map is clipped back down to its regular size, it is not identical to the original. Although the difference is not extreme, it is apparent.

The other, more visible problem, also from the skylibs library, concerns the conversion from the cubemap back to the latlong representation. Due to a bug in this operation, a black line sometimes appears at on or more borders of a face. Since a synthesized image is made up of mosaicked, reprojected areas, it is possible that in some cases, these lines are shifted into the middle of different faces.

Although these errors seem relatively small, it is necessary to keep them in mind during the evaluation, since they only apply to the flow-based blending and may skew the results.

4. Evaluation and Results

Unfortunately, there are no publicly available benchmarks for 360° image synthesis with two degrees of freedom without 3D geometry, as not many methods exist that try to achieve this. Since the approach presented in Chapter 3 is a first, basic attempt at solving this problem, this chapter presents a basic evaluation of the algorithm, based on mathematically calculable error metrics. These metrics measure the accuracy of a synthesized image compared to the ground truth and are used to assess the effect of a limited number of parameters.

First, possible parameters of the algorithm and of the scene are discussed (Section 4.1), followed by the presentation of the evaluation methodology (Section 4.2). Then, virtually generated scenes are used to evaluate the effect of different combinations of select parameters in a controlled environment (Section 4.3). Based on the knowledge gained in the this evaluation, a proof-of-concept evaluation is performed on a real scene (Section 4.4). Finally, the limitations of the evaluation and algorithm are discussed (Section 4.5).

4.1. Parameters

Before defining the parameters to test in the limitation and proof-of-concept evaluations, this section gives an overview of possible parameters in the context of the 2-DoF algorithm presented in Chapter 3. The 2-DoF algorithm already makes a few assumptions, for example the constraint to the viewpoint plane, the fact that the scene is static, and more (stated in Section 3.1 on page 19). These assumptions are upheld in the evaluation, as they are prerequisite to the algorithm.

The remaining parameters (that are not constrained by the assumptions) can be divided into two categories:

Internal parameters i.e., parameters based within the algorithm, such as the blending type and the selection of input viewpoints.

External parameters i.e., parameters based on the properties of the captured scene, such as the viewpoint density, or the geometry of the scene.

The internal parameters can be modified after the scene has been captured, the external ones cannot. The most prominent internal parameters based on the implementation from Chapter 3 are:

- location of synthesized points within the scene (near walls, objects, etc)
- location of synthesized points relative to the captured points
- blending type, i.e., flow-based blending or deviation-angle-based knn blending ("regular blending")
- optical flow algorithm used for flow-based blending

There are more internal parameters that could theoretically be modified, such as the knn blending function (Equation 3.9 on page 31), or the method of ray approximation for 2-DoF in flow-based blending (Section 3.1.3), but these will be assumed immutable for this evaluation, as the variation of these parameters would require developing new functions, which is outside the scope of this thesis.

As for the possible external parameters, the number of different possible scenes is infinite, but the assumed key parameters are:

- type of scene (outdoor, indoor, etc) → size and shape of scene
- objects within the scene
- density of captured viewpoints
- distribution of captured viewpoints

External parameters such as the camera settings (e.g., aperture, shutter speed, white balance) and the lighting throughout the scene are not considered; it is assumed that all the captures have the same camera settings and white balance parameters. Furthermore, the evaluation is restricted to indoor scenes of approximately $25m^2$. This reduces the parameter space significantly, since indoor scenes tend to be enclosed by walls, which enforces a maximal distance of objects to the camera.

The evaluation presented in this thesis aims to examine the effects of a few select internal and external parameters, instead of exhaustively examining all of them. In order to do this, a *scenario* is designed for each selected parameter that attempts to demonstrate the effect of this parameter on the accuracy of the result. It must be noted that limiting the evaluation to specific scenarios reduces the testing space but might also lead to missing some interactions between parameters.

4.2. Evaluation Methodology

The evaluation is divided into two distinct phases: an evaluation of limits using virtual scenes, and a proof-of-concept evaluation using real scenes. Both evaluations follow the approach depicted in Figure 4.1, and consist of four steps: *scenario definition*, where a scenario is designed to test a specific parameter, *synthesis*, where the synthesized images are calculated using the 2-DoF synthesis presented in Chapter 3, *error calculation*, where the accuracy of the synthesized images is measured, and *result analysis*, where the cause and effect of the parameters is examined. The details of these steps are described in the following sections.

Scenario Definition

A scenario is defined by the parameter that it tests, the static parameters that are used, and the scene where the data was captured. Although a scenario is designed to test a specific parameter, which then is "dynamic" (i.e., will be modified throughout the scenario), there might also be more dynamic parameters. For example, in a scenario for exploring viewpoint density, the blending type might also be modified to see what effect the viewpoint density has on regular and flow-based blending. The static parameters include the location of the synthesized viewpoints, for example, or any other parameter that remains unchanged

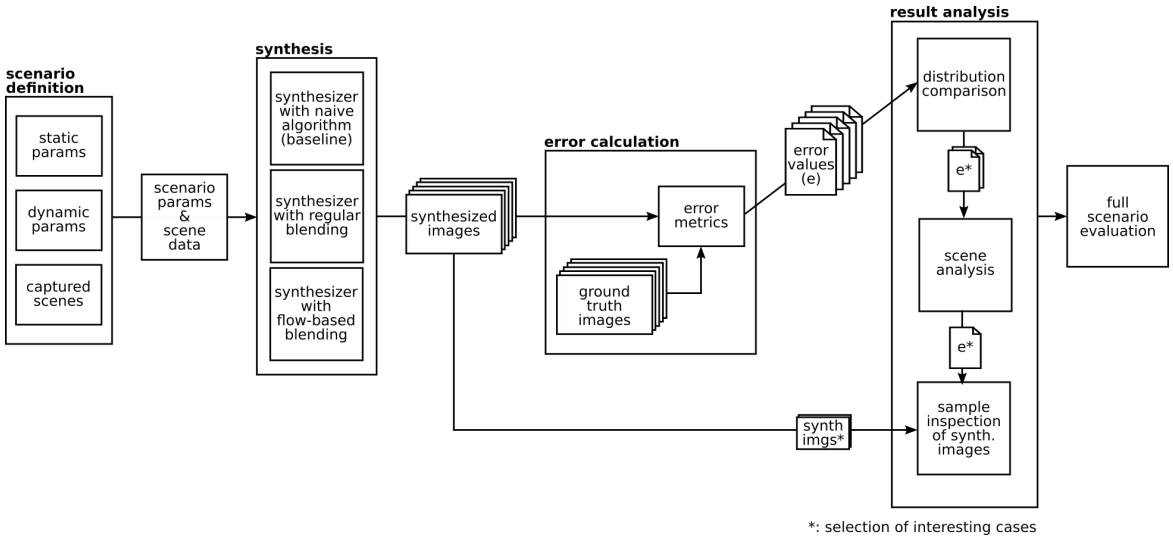


Figure 4.1.: Methodology for the evaluation of a scenario

throughout the scenario. One other defining factor of a scenario is the scene data that the scenario is tested on. Although the scene is part of the parameters (the external parameters, to be exact), it merits particular mention, as it contains the actual image data that is used for synthesis. This data, along with the other parameters defined in the scenario, are then passed on to the synthesis step.

Synthesis

The synthesis step consists of two parts: the 2-DoF synthesis presented in Chapter 3 using either flow-based blending or regular blending depending on the scenario parameters, and a baseline synthesis using a naïve algorithm. The naïve algorithm consists of simply selecting the nearest neighbor viewpoint based on euclidean distance. The input parameters are the same for both algorithms and both results are passed on to error calculation. The results of the naïve algorithm serve as a baseline comparison to verify whether the developed 2-DoF algorithm is an improvement to a naïve approach. If either the regular or the flow-based blending generally performed worse than the naïve algorithm, this would be an indication of a substantial flaw in the approach.

Error Calculation

There are many properties that a synthesis algorithm can be evaluated for, for example execution speed, visual acceptability (based on user studies), number of artefacts, or distance from the ground truth. In this evaluation, mathematical error metrics are used to compare each result to its ground truth image. Two different metrics are chosen based on different image features so that potential limitations of each metric can be compensated for by the other. However, it must be taken into account that neither metric was designed specifically for the task of measuring accuracy of 360° synthesized images. As a result, it is necessary to monitor and verify their results by visual examination, especially in cases where the results do not correspond.

(a) (b) (c) L1 error visualization of (a) and (b)

Figure 4.2.: Example visualization of L1 RGB error. The RGB error values have been intensified so that they are more visible.

L1 error on RGB The first metric is the L1 error calculated on the ground truth and result images in RGB color space. This error metric calculates the mean absolute difference of the RGB values and therefore indicates the mean accuracy of each pixel of the image. The difference of the RGB color values $r, g, b \in [0, 1]$ (for floating point values) of each pixel of the ground truth (gt) and synthesized (s) image is calculated, then all of the differences are added together, and divided by the number of pixels (P) in the image (Equation 4.1). The error value range is $e_{L1} \in [0, 3]$.

$$e_{L1} = \frac{1}{|P|} \cdot \sum^{P} |(P_r^{gt} - P_r^s)| + |(P_g^{gt} - P_g^s)| + |(P_b^{gt} - P_b^s)| \tag{4.1}$$

The L1 error can also be visualized by calculating the sum of absolute differences per pixel without averaging the values. Figure 4.2 shows an example visualization of the L1 error between two images. The visualization encodes areas of the image where there is a very large difference with a value closer to white and areas where there is no difference as black, which highlights areas of the image that are problematic. Since the sum of the differences between all three color values is used, the visualization can contain oversaturated white areas where the pixel value is larger than 1. Although this reduces the ability to judge the severity of the error values in the image, it helps in visually recognizing problematic areas, which is more important for this evaluation.

The L1 error is useful because it gives a rough estimate of how accurately each pixel is synthesized. The visualization indicates in which areas the synthesized image is inaccurate, which is helpful for classifying problems. However, a drawback of the L1 error is that it relies on color values, so images with large differences in pixel values will generally produce a higher error value than images with smaller differences in pixel values, even though the distortion and displacement may be the same.

As in the case of optical flow calculation, some adjustment must be made to adapt this metric for 360° images. Since the equirectangular projection is not equal-area, the areas towards the poles would intrinsically have higher weighting, since RGB L1 is calculated per pixel. To avoid this problem, the cube map projection is used, since it does not significantly distort the image. The average value is then calculated using the six faces of the cube,

omitting the black background.

SSIM error on Grayscale The metric to complement the RGB L1 error is a variation of the structural similarity index (SSIM) [ZBSS04], which measures the *structural similarity* between two images. Instead of comparing the images pixel by pixel, the SSIM uses the luminance, contrast and structure of the images for comparison. It compares these locally, i.e., it operates on smaller areas instead of the image as a whole. Consequently, it is possible that the SSIM does not register small displacements in the scene if the objects are not distorted. However, the additional comparison with the RGB L1 error should mitigate this potential problem.

The SSIM metric in general, and the implementation used in this evaluation[1] return a value $SSIM \in [-1, 1]$ with 1 signifying an extremely similar image and -1 signifying a very different image. In order to be able to compare it more easily with the RGB L1 error, the SSIM value is converted to an error value $e_{SSIM} \in [0, 1]$, with 0 signifying an identical image (0 error) and 1 signifying a very different image.

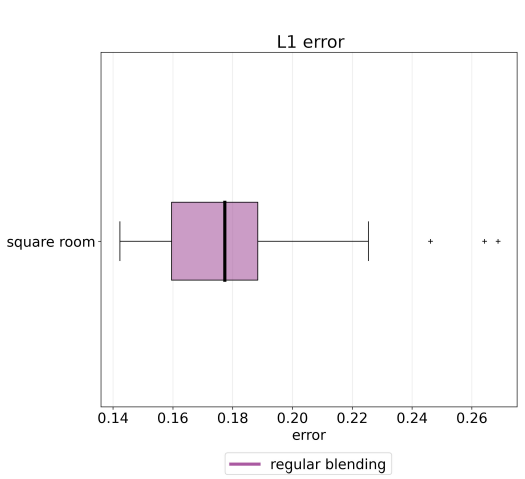
The SSIM error is calculated on the grayscale image in cubemap representation. There is no need to use an RGB image, since it does not use the color values of an image. The cubemap representation is again used to avoid possible problems with distortion.

Result Analysis

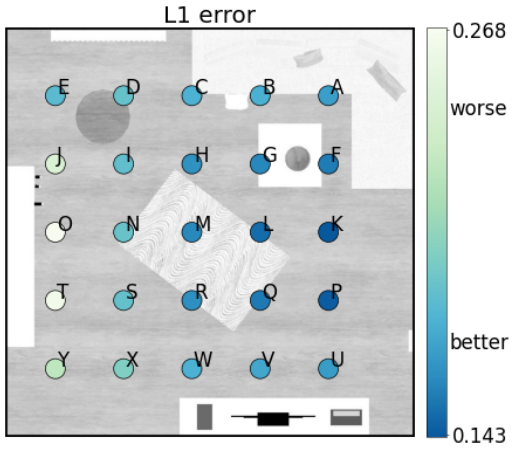
For each scenario evaluation, the number of result images depends on the tested parameters and the number of synthesized viewpoints. For each result image, there are two error metrics to be considered. In order to effectively analyze this potentially very large number of results, it is necessary to break them down by creating different visualizations that highlight different attributes. At first, an overview is created, from which interesting cases are selected and examined in more detail.

Distribution Comparison The first step of the analysis is a comparison of error value distribution. In order to compare all the error values of a scenario, they are plotted using a boxplot (Figure 4.3a). The different parameter combinations of the scenario are plotted on the y axis (e.g., viewpoint densities a, b, c) and the error distribution (i.e., the error values of all the synthesized viewpoints) is plotted on the x axis. The box plot shows the distribution of these values: The thick black line in the colored box is the median value (approx. 0.184 in Figure 4.3a); the colored ranges to the left and right of the median describe the "interquartile range" (IQR), the range of the closest half of the data (25% on each side). The "whiskers" of the plot extend to the minimum and maximum of the values. The minimum and maximum are defined as $1.5 \cdot IQR$. Any data outside of the range between the minimum and the maximum, are the "outliers", depicted as small crosses. The boxplot gives a general overview of how the error values of the specific scene are distrubute. The distribution of the error values of the results in Figure 4.3a, for example, shows that the first three quartiles of the results are fairly close to the median (between approx. 0.14 and 0.19), whereas the fourth quartile extends, over almost the same range (0.19 to 0.225) and there are some extreme outliers. This indicates that there are some viewpoints that performed significantly worse than most of the others.

[1] skimage.metrics.structural_similarity [vdWSN+14]



(a) Boxplot example for distribution comparison of the results of regular blending in the "square room"



(b) Visualization for scene analysis: The error values from the boxplot in (a) are mapped to colors at the positions of the viewpoints in the scene. Light green represents the worst value (0.268) and dark blue the best value (0.143).

Figure 4.3.: Different types of result visualizations for L1 error values for example results of regular blending in a scene

Scene Analysis Based on the insights gained in the distribution comparison, several interesting cases are selected for closer analysis. These cases are examined by color coding the error values and assigning the colors to the positions in the scene. This puts the error values in context with the scene surroundings. Figure 4.3b shows the synthesized points in the context of the scene, color coded by their error values. The maximum and minimum values of the points (also clearly visible in Figure 4.3a) are coded as light green and dark blue, respectively. This visualization gives a more detailed overview over the values of the different points. In Figure 4.3b, for example, the synthesized points near the right wall of the room have much better values than the row on the left side of the room. The four light green values are clearly the outliers that were visible in Figure 4.3a. Using this information, it is possible to draw some conclusions about the effect of the position of the synthesized viewpoint relative to the objects in the scene, and select a few synthesized images that merit closer examination.

Sample Inspection In order to further understand the effects of the parameters on specific positions, some of the synthesized viewpoints from the scene analysis are examined manually by comparing the synthesized image to the ground truth image. The visual examination may also reveal information that the error metrics are unable to extract, such as specific types of artefacts. For example, by using the information presented in Figure 4.3b, it is possible to choose one of the best results, for example synthesized point "K" near the right wall in the middle. The close inspection of the synthesized images is shown in Figure A.1 (page 82[2]): In the left column from top to bottom are the ground truth image, the synthesized image using regular blending, and the synthesized image using flow-based blending. In the right column are the L1 difference images to the ground truth image. They can help with understanding the error values. For example, the accuracy of the rug in the bottom face (marked in green)

is improved in the flow result compared to the regular result in Figure A.1. However, the flow result also has a fairly large artefact at the top of the door in the left face (magenta ring), which is not the case in the regular result.

4.3. Evaluation using Virtual Scenes

In the first part of the evaluation, virtual scenes are used to evaluate the performance of the 2-DoF synthesis using regular blending and flow-based blending. The performance in the context of this evaluation is defined as the accuracy of the synthesized image based on the L1 and SSIM error metrics, as well as visual examination. Virtual scenes allow full control over the external parameters, for example the scene geometry, and the positions of the captured and synthesized viewpoints, as well as enabling automatic generation of the images at the chosen positions. As a result, it is possible to test setups that would be unfeasible for real scenes, for example setups using a large number of captured viewpoints at varying locations in different scenes. The selected internal and external parameters that will be explored in the scenarios tested in the virtual scenes are:

- proxy-scene difference (how dissimilar is the scene geometry from the proxy sphere geometry, including the objects within the scene)
- density of the captured viewpoints
- position of the synthesized viewpoints relative to the captured viewpoints

A scenario is designed for each of the three parameters, containing several different setups to test the performance and limitations of the algorithm developed in Section 3.

4.3.1. Data Acquisition and Featured Scenes

Three different scenes were modeled for testing, using the animation software Blender [Ble20]: the *checkersphere*, the *square room* and the *oblong room*.

Checkersphere The *checkersphere* (Figure 4.4) is a perfect sphere with a radius of approximately 2m. Its surface is covered with a checkerboard pattern with alternating colors of dark blue, dark red, and dark green. Although this kind of room is not likely to exist in reality, it represents an interesting case, since the scene geometry is identical to the proxy geometry. The checkerboard pattern was chosen so that distortions or inaccuracies would be more visible.

Square Room The *square room* (Figure 4.5) is a room whose basic shape is a perfect cube with a side length of 3.5m. It contains an assortment of furniture[3]: In one corner, there is an orange, L-shaped couch with dark blue and white cushions on it. In front of the couch is a small, white marble coffee table with a dark blue bowl on it. Several small, simple black and white pictures are hanging on the wall behind the couch. There is a blue and white rug in the middle of the room, and to the left of the couch are a round blue table, as well as a white radiator. On the wall next to the blue table is a white marble bookshelf containing

[2]The synthesized images are shown in Appendix A, in order to avoid interrupting the flow of the text, since they take up a fairly large amount of space.

several red books, as well as three wine bottles, and two decorational objects in green and purple. Across from the couch is a low marble cabinet with a black monitor, a black laptop and a black speaker on it. To the left of the cabinet is a wooden door with a gray handle. The walls are brick, painted a dark purple, and there is a lamp with a white lampshade hanging from the middle of the ceiling.

The intention of using the square room is to allow approximate accuracy for the reprojection step, since the general cube shape of the room is still somewhat similar to the proxy sphere geometry, while at the same time offering a more realistic indoor setting.

Oblong room The *oblong room* (Figure 4.6) has a room size of approximately 5.5m x 3.5m and contains the same basic elements as the square room. It has the exact same furniture layout as the square room, except that the basic shape of the room is different. The walls are also a dark blue, instead of dark purple.

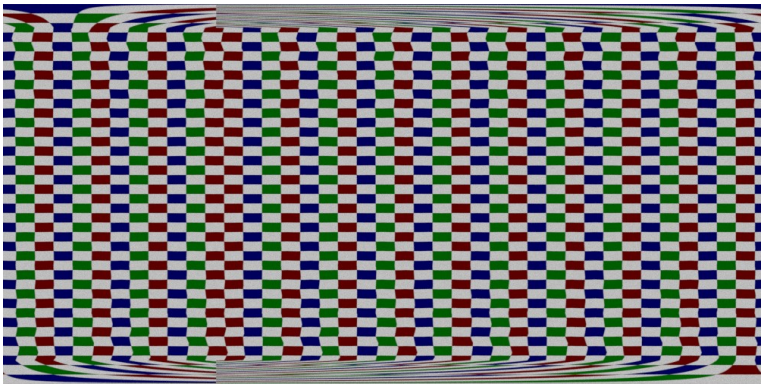
Given the different scenes, it is necessary to choose comparable viewpoints to capture that will be used as input for the synthesis. Since the scenes all have similar scale, the viewpoint layout was chosen to be identical in all of the scenes. This means that all scenes contain 36 captured viewpoints, aligned in a regular grid of 6x6 viewpoints, with 60cm spacing. The grid of viewpoints is centered within the scene. This means that in the checkersphere (Figure 4.7a) and the square room (Figure 4.7b), the viewpoints cover approximately the complete scene, and in the oblong room, there is about a 1m area on each side of the grid that is not covered (Figure 4.7c). It is necessary to take this difference in viewpoint coverage into account for the evaluation, since the viewpoints in the oblong room have a larger distance to two of the walls, which may have an effect on the accuracy of the results.

Since Blender is designed for creating animated movies, and not the capture of static viewpoints, some adjustments had to be made to be able to capture the chosen viewpoints (as well as the ground truth images): After choosing the positions of the input viewpoints for the scenes, the position of the camera for each captured viewpoint was stored as a keyframe, so that the batch of viewpoints could be rendered like an animation. A Blender script was implemented in order to automatically assign the viewpoints and the ground truth points to the keyframes, and write out the metadata. This way the locations of the viewpoints would always be perfectly accurate, and the "capture" of the viewpoints required no manual effort. The images were rendered with a resolution of 1000x500 for all of the scenes, in order to reduce the computation time for the image synthesis in the tests.

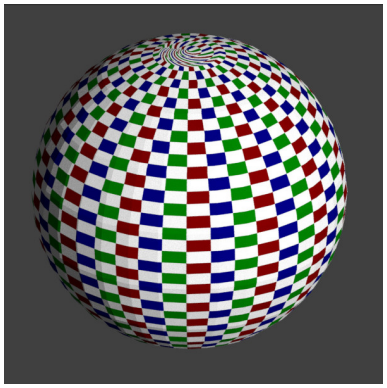
4.3.2. Synthesizing Optical Flow with Blender

Using Blender to create virtual scenes not only facilitates capture, but also offers an alternative to calculating optical flow. As mentioned in Section 2.1.2, most optical flow algorithms

[3]The furniture models used in the square room and the oblong room are adapted from `https://www.cgtrader.com/free-3d-models/interior/living-room/low-poly-interior-57731178-c955-4625-9e44-109c8eea5ee2`, by user "miha29076", and the textures are adapted from `https://www.poliigon.com/texture/plaster-17`, `https://www.poliigon.com/texture/fabric-denim-003`, `https://www.poliigon.com/texture/wood-fine-dark-004`, and `https://www.poliigon.com/texture/interior-design-rug-starry-night-001`. *All accessed last on September 23, 2020*

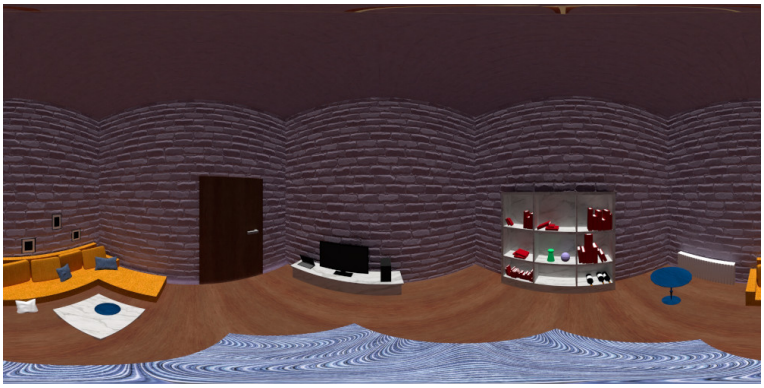


(a) Latlong image from the center of the sphere



(b) View from the outside for reference

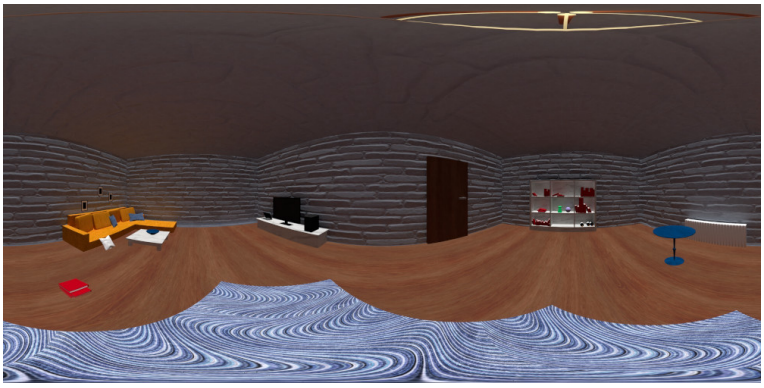
Figure 4.4.: Overview of the “checkersphere”



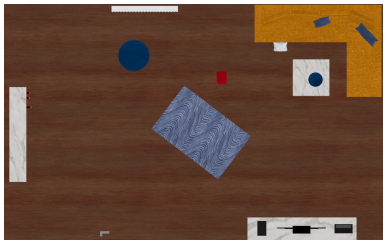
(a) Latlong image from the center of the room

(b) Top view

Figure 4.5.: Overview of the “square room”

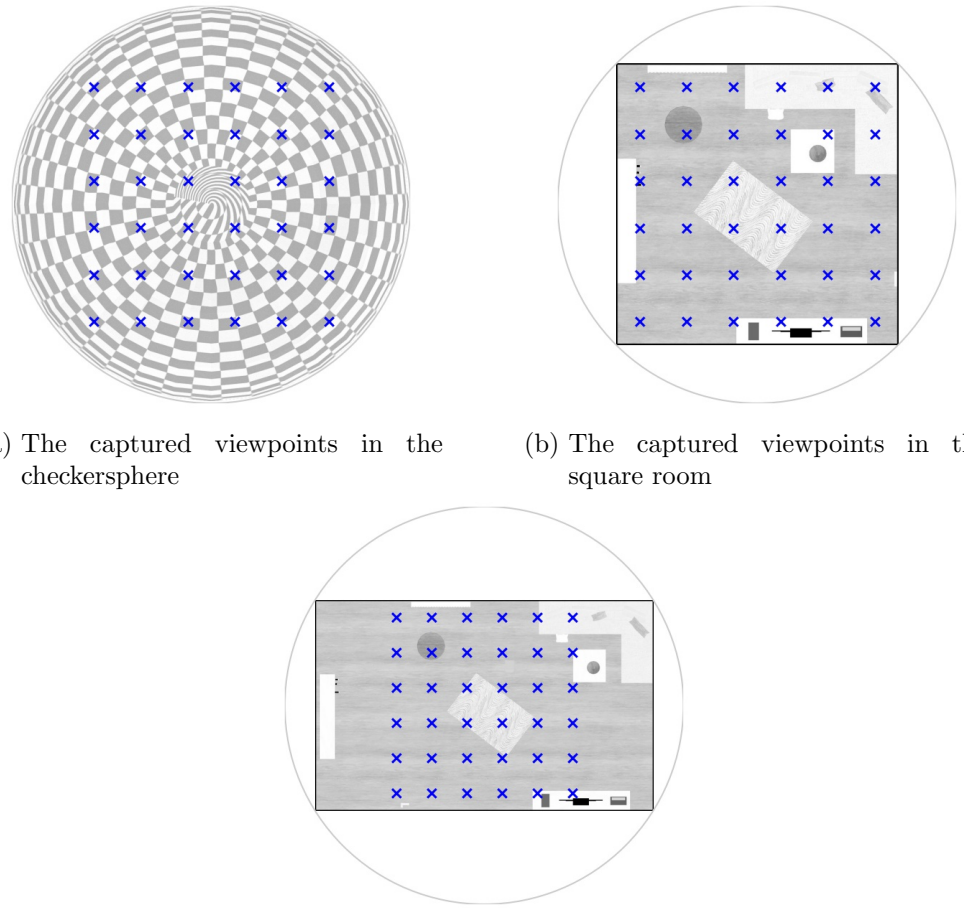


(a) Latlong image from the center of the room



(b) Top view

Figure 4.6.: Overview of the “oblong room”



(a) The captured viewpoints in the checkersphere

(b) The captured viewpoints in the square room

(c) The captured viewpoints in the oblong room

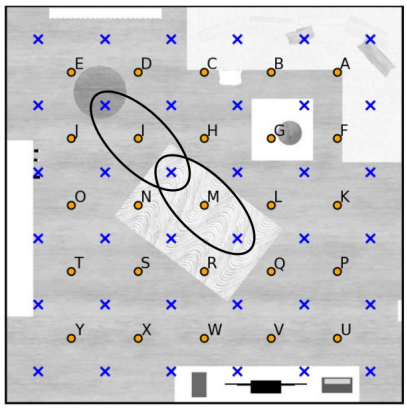
Figure 4.7.: The grid of captured viewpoints in each scene, including the proxy geometry (not to scale)

struggle with large displacements. The flow-based blending step in the 2-DoF synthesis algorithm, on the other hand, assumes decent optical flow and there is no attempt to judge whether the optical flow calculation is feasible between two selected viewpoints. As a result, given the wrong circumstances (two viewpoints A and B that are far apart), the optical flow algorithm may fail, leading the flow-based blending to output undesirable results. The success of the optical flow algorithm is a prerequisite for the success of the 2-DoF algorithm with flow-based blending.

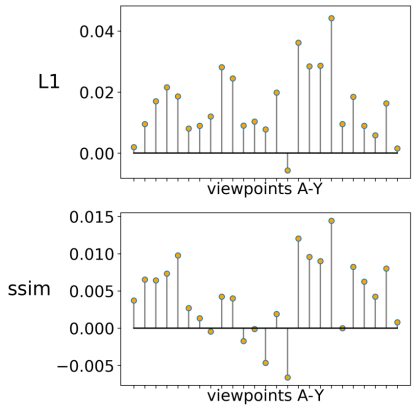
However, the focus of this evaluation is not the accuracy of an arbitrary optical flow algorithm. In the best case, it would be possible to emulate "perfect" optical flow, thus decoupling the success of the optical flow from the success of the flow-based blending. While this is practically impossible for real scenes, virtual scenes theoretically contain all necessary information for retrieving "ground truth" optical flow. Blender, for example, is capable of "rendering" motion vectors using its vector speed render pass, which calculates the movement between points from one frame to the next in pixel space. The result is a motion vector field, which corresponds to the result of a "classic" optical flow algorithm. This "ground truth" optical flow (in the optimal case) was first presented by Butler et al. [BWSB12] as a benchmark for optical flow algorithms.

In order to demonstrate the improvement of Blender optical flow compared to Farnebäck optical flow, which is the optical flow algorithm used in the implementation, Figure 4.8a shows a scene setup in which 25 viewpoints (A-Y) are synthesized using Farnebäck and Blender optical flow at the interpolation distance $\delta = 0.5$ between captured points. Figure 4.8b shows the improvement of the results using Blender optical flow: In all but one case for the L1 values, and in the majority of the SSIM values the Blender optical flow improved the error values. The visual difference of the 1-DoF interpolation is clear: Figure 4.8c shows the viewpoint "I" interpolated using Farnebäck optical flow, and Figure 4.8d the same viewpoint using Blender optical flow. There are distincly fewer artefacts with Blender optical flow, for example the rug and the couch both have a much more distinct outline, and the bookshelf is also clearer. Figure 4.8e and Figure 4.8f show the same for interpolated viewpoint "M". In this case, the TV cabinet and the rug are much clearer in the Blender optical flow version (Figure 4.8f).

It is notable that using Blender's optical flow tends to improve the results, compared to Farnebäck's algorithm, but that does not mean that the resulting optical flow is necessarily accurate. For example, the bookshelf in the right and middle faces in Figure 4.8d still shows warping and doubling effects, indicating that there are still some inaccuracies. The same is true for the coffee table and the blue round table in the bottom face. There are several possible reasons for this, mostly based on the fact that the process in Blender, like most optical flow algorithms, is designed for frame-to-frame use, and has in this case been "misused" for distances that are infeasible for an actual animation. Nevertheless, no definitive explanation can be made at this point, since this would require in-depth understanding of Blender's vector speed render pass, which is outside of the scope of this thesis. Based on the results shown in Figure 4.8, and experience gained from testing both variants, the Blender optical flow is used for the tests, because, even though it is not completely accurate, it still seems to mostly yield better results than Scipy's implementation of Farnebäck's optical flow algorithm and as such decouples (to a degree) the success of the 2-DoF synthesis with flow-based blending from the success of the optical flow algorithm.



(a) The interpolated viewpoints (A-Y) for testing optical flow

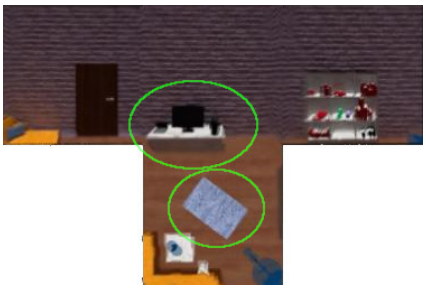


(b) The improvement of error values using Blender optical flow vs Farnebäck optical flow

(c) 1-DoF interpolated viewpoint "I" using Farnebäck optical flow

(d) 1-DoF interpolated viewpoint "I" using Blender optical flow

(e) 1-DoF interpolated viewpoint "M" using Farnebäck optical flow



(f) 1-DoF interpolated viewpoint "M" using Blender optical flow

Figure 4.8.: Comparing 1-DoF interpolation results using Farnebäck to results using Blender optical flow

4.3.3. Scenarios and Results

Using the generated scenes and optical flow, as well as the parameters defined for this evaluation, it is now possible to design, test, and evaluate the following scenarios:

- "Different Scene Geometries"
- "Density of Captured Viewpoints"
- "Position of Synthesized Viewpoints Relative to Captured Viewpoints"

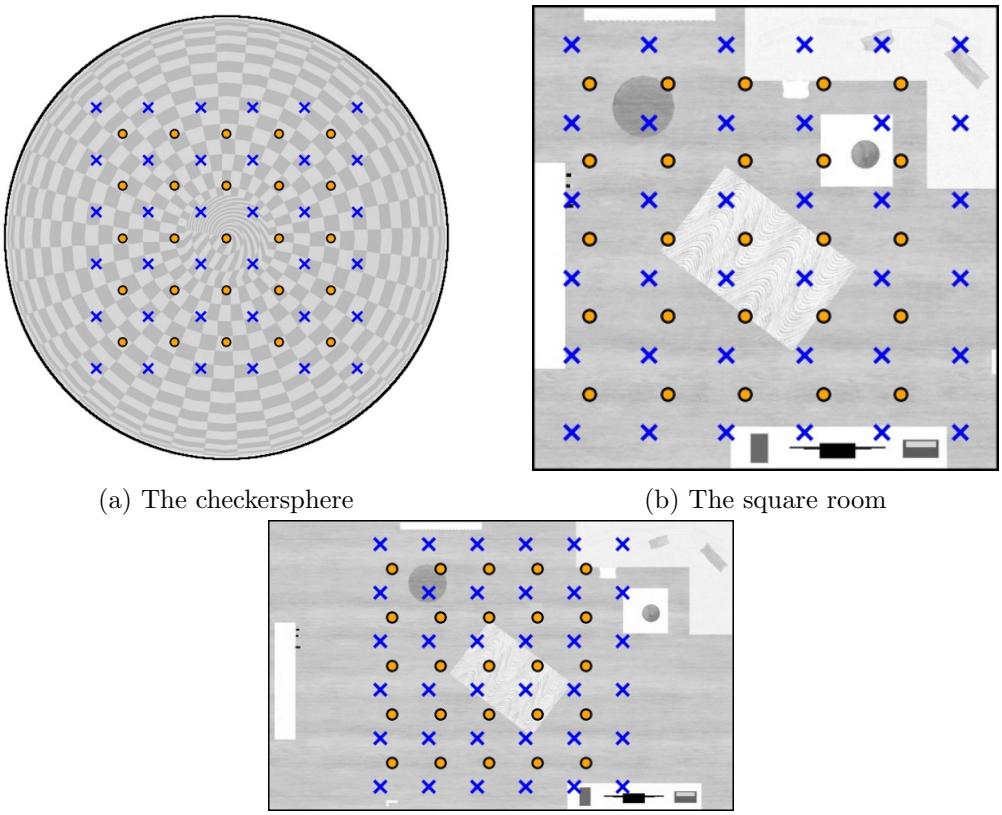
This section presents the scenes, viewpoint setups, and tested parameters used in each of these scenarios, as well as the results of the tests, which are evaluated using distribution comparison, scene analysis, and sample inspection, as described in Section 4.2. First the effect of the tested parameter on the regular blending results is examined, then the effect on the flow-based blending results is examined, and finally, the results of the flow-based blending are compared to the results of the regular blending.

Different Scene Geometries

The first parameter to be examined is the effect of different scenes on the accuracy of the results. There are two attributes of a scene that may have an influence on the result: the basic shape of the scene (e.g., sphere, cube, rectangular prism, or arbitrary polygon), and the objects within the scene. All the scenes presented in 4.3.1 are used, as they differ both in their basic shape, as well as in the arrangement of the objects within, although the square room and the oblong room are more similar to each other than to the checkersphere.

The arrangement of captured viewpoints in all the scenes is identical (a 6x6 grid with a spacing of 60cm), and 25 viewpoints are synthesized in each scene (Figure 4.9). The synthesized points are near or in the center of each grid cell, since these are the areas where the deviation angles are the highest, and where the largest artefacts for the regular blending are expected to emerge. In the square and oblong rooms, the synthesized points are slightly offset from the center of each grid cell. The offset is important in order to test actual 2-DoF synthesis, instead of just 1-DoF interpolation, since synthesizing in the center of a grid cell could be done with only 1-DoF interpolation (e.g., by interpolating by 0.5 between the top right to the bottom left captured viewpoint, as was done in Section 4.3.2 for testing Blender optical flow). No offset was used in the checkersphere scene, since the checkersphere scene is expected to have excellent results for the regular blending (since the proxy geometry and scene geometry are identical). In this case it is more interesting to use one of the presumably best positions for the flow-based blending to see how well it holds up in comparison.

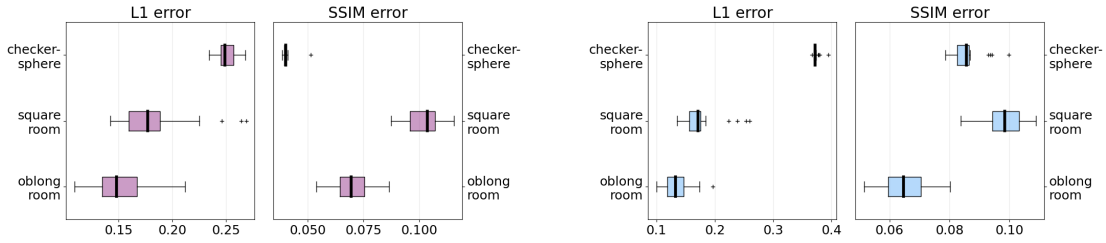
Regular Blending Results Figure 4.10a shows the distributions of the error values for the regular blending results in the three scenes. The most striking feature of this distribution is that the checkersphere results show the highest error values for the L1 error, whereas they show the lowest values for the SSIM error. At first, this seems surprising, both because the error metrics do not "agree", as well as because the results of the regular blending are expected to be very good since the scene geometry is identical to the proxy geometry. The reason for the ambiguous results is the sensitivity of the L1 metric to different color values. L1 errors on the RGB images of the checkersphere will produce a higher value in general because of the checkerboard texture: The difference between a dark blue pixel of a dark



(a) The checkersphere (b) The square room

(c) The oblong room

Figure 4.9.: The captured (blue) and synthesized (orange) viewpoints in the different scenes (scenes are not to scale)



(a) Regular blending (b) Flow-based blending

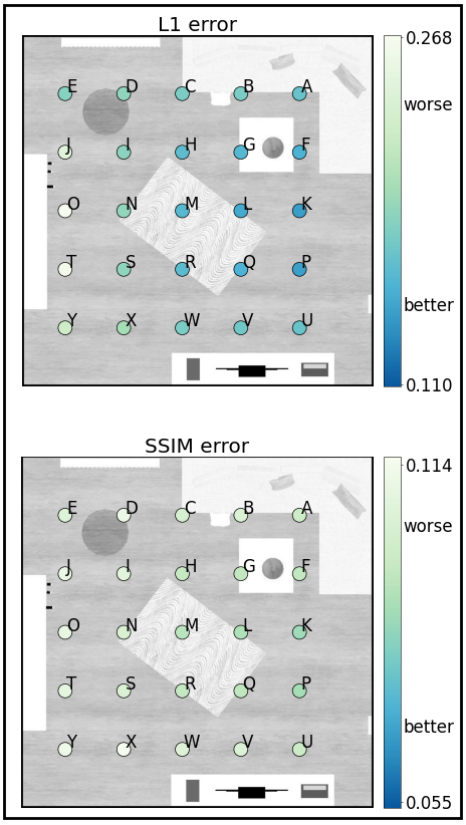
Figure 4.10.: Comparing the distributions of the results in different scenes

checkerboard field, and a white pixel on a white checkerboard field is close to 1, whereas the difference between a dark brown pixel and a dark purple pixel (e.g., between the door and the wall in one of the rooms) will produce a lower error value, although the distortion or displacement may be identical. The SSIM error metric, which does not take the color values of the pixels into account, produces a distribution that is much closer to the expected result: Since the scene geometry is identical to the proxy geometry, the result of the reprojection should be almost perfect. And in fact, when visually comparing the results of a synthesized viewpoint to the ground truth (Figure A.2, page 83), the only difference is a little bit of blurriness due to sampling differences. The L1 difference depicted in the right column of Figure A.2 shows how the blurriness caused the high L1 error.

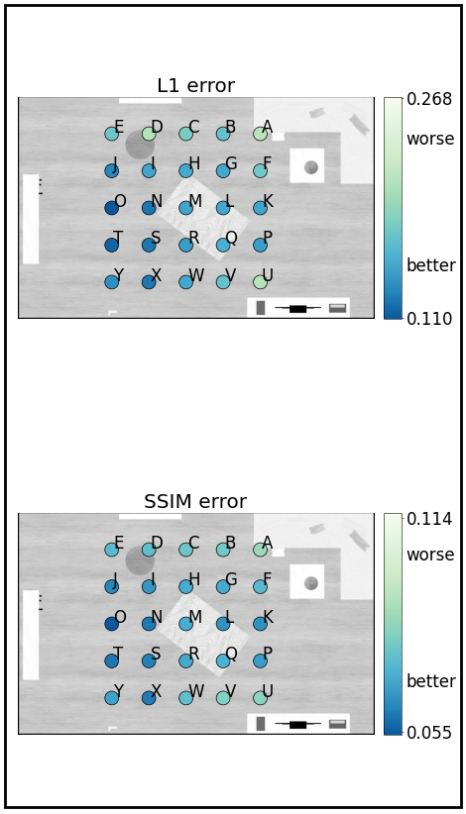
The trend of the error metrics of the other two rooms is consistent: Both the L1 and SSIM error values of the square room are generally higher than those of the oblong room. In this case, comparing the RGB color values is more reliant, since both rooms use the same textures. The results however, are surprising: Although the basic geometry of the square room is closer to the proxy geometry, the error values from the square room are generally higher than those from the oblong room. A closer scene analysis (Figure 4.11) shows the probable reason for this: The scene analysis visualization of the square room (Figure 4.11a) shows that the error values are particularly high near the bookshelf on the left side of the room, whereas in the oblong room (Figure 4.11b) the bookshelf is farther away, and the values on the left side of the room are very low. Comparing the synthesized images at location "O" (directly across from the bookshelf) for the two scenes (Figure A.3, page 84) confirms this impression: The bookshelf is so close to viewpoint "O" in the square room, that there are extreme inaccuracies in the synthesis due to large deviation angles. In the oblong room, the bookshelf is much further away, making the devation angles much smaller and the result more accurate. The larger net distance to the walls and some of the objects in the oblong room is the likely reason for the lower error results throughout the scene.

As for the accuracy of the synthesized viewpoints relative to objects in the scenes, results in the square room have already been shown to have very high error values directly by the bookshelf (high detail, close proximity). As for the rest of the room, the error values seem moderately high in the second row from the bookshelf (D, I, N, S, X), and in the top row (E, D, C, B, A) and the bottom row (Y, X, W, V, U), and lowest in the center and right side of the room (H, G, F, M, L, K, R, Q, P). This is likely due to the proximity of the viewpoints to the walls, and the objects in the scene. The closer the viewpoint is to the walls (i.e., the edge of the scene), the higher the deviation angles are, and thus, the ghosting artefacts and positional inaccuracies. The bookshelf is especially close to the viewpoints, since it is at eye-height, instead of below (and thus further away, with smaller deviation angles), like the majority of other objects. The same effects are visible in the oblong room, although the synthesized viewpoints are generally further away from objects, due to the shape of the scene. Near the top wall (E, D, C, B, A), the error values are generally higher, and especially so over the blue table (D) and over the sofa (A). In the bottom row, the viewpoints near the TV cabinet (U, V) are also higher than in the rest of the scene, whereas the bottom left viewponts (O, N, T, S, Y, X) are the lowest in the scene, and also relatively far away from any objects.

Flow-based Blending Results The distribution of error values of the synthesis using flow-based blending in the different scenes (Figure 4.10b) show similar tendencies as the error



(a) Error values of regular blending in the square room



(b) Error values of regular blending in the oblong room

Figure 4.11.: Scene analysis of regular blending results in the square and oblong rooms

values of the regular blending: The L1 values of the checkersphere are very high, and the results of the oblong room generally have lower values than those of the square room. One exception is the SSIM error value of the checkersphere: Whereas the SSIM error in the checkersphere scene is significantly lower than the other two for the regular blending results, in the flow-based blending results, the SSIM error yields worse results than the oblong room, but still mostly better than the square room. The worse performance of the flow-based blending in the checkersphere is likely due to the fact that the flow-based blending introduces some inaccuracies, for example due to imperfect optical flow, or ray approximation. Figure A.2 (page 83) shows slightly higher inaccuracies for the flow result, as well as some artefacts and noise, that are the likeliest causes for the higher error values.

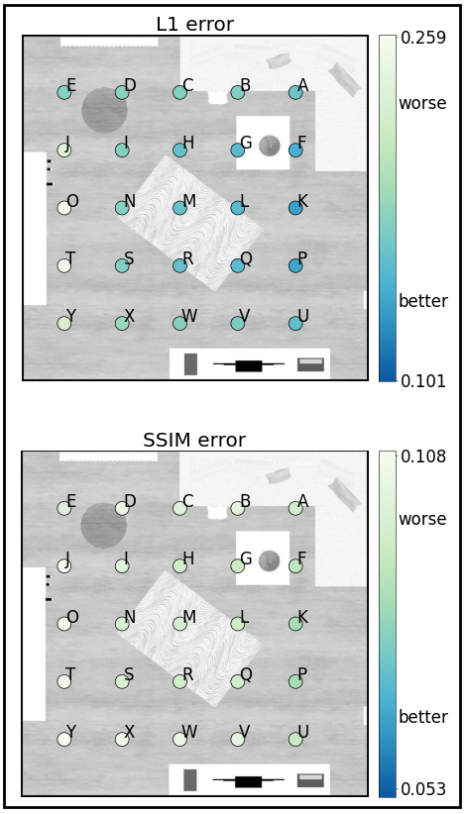
In the other cases, the results of the flow-based blending mirror those of the regular blending: The error values in the oblong room are generally lower than those in the square room. A look at the scene visualization in Figure 4.12 shows that the flow-based blending results show very similar tendencies as the regular blending results. Like in the regular blending case, the reason for the worse performance of the square room is very likely also the relative postion of the walls and objects to the synthesized viewpoints. The most severe case is the area around the bookshelf. In the case of the regular blending, the reason this area was problematic were the large deviation angles leading to inaccurate reprojections. Generally, the flow-based blending should alleviate this problem, however, in this case it did not (or not considerably). Looking at the example viewpoint "O" in Figure A.4 (page 85) shows why the flow-based blending also produced an inaccurate image: Due to the optical flow calculation failing in proximity to the bookshelf, which was demonstrated in Section 4.3.2, the normally straight lines of the books are warped, but a comparison to the ground truth shows that they are warped in a way that bears no resemblance to the actual position or shape.

The general relation of the accuracy of the synthesized viewpoints to the proximity of the objects in the scene is very similar to the results of the regular blending. Viewpoints that are closer to objects or walls tend to have higher error values (e.g., W, V, U in the square room and D, A, and U in the oblong room) while viewpoints that are further away from objects have lower error values (L, K in the square room, O, N, T, S in the oblong room). These tendencies are much more pronounced in the oblong room, where the differences in distance are higher.

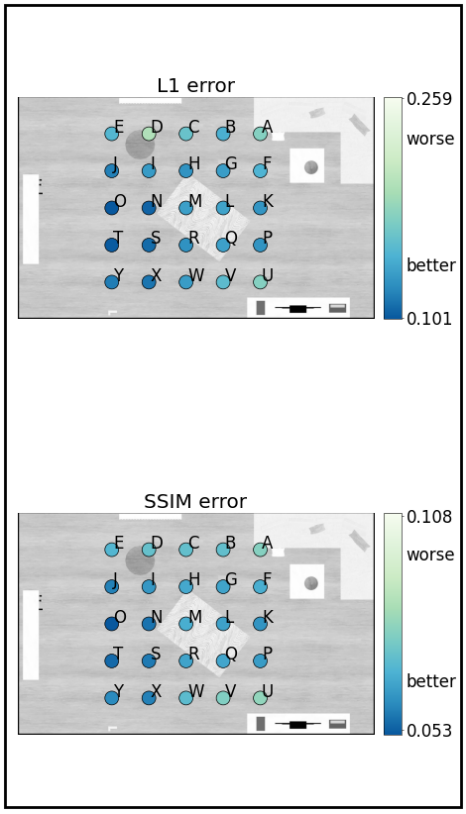
Comparing Regular Blending to Flow-based Blending Results The distributions of the error values of both the regular blending (purple) and the flow-based blending (blue) are shown in Figure 4.13, as well as the error value distribution of the naïve algorithm (orange) for comparison[4]. The graph shows that the error values of the results of the flow-based blending are generally slightly better than those of the regular blending (except in the case of the checkersphere) and that the error values of the results of the regular blending tend to be distinctly better than those of the naïve algorithm.

In the case of the checkersphere, where the scene geometry is identical to the proxy geometry, the regular blending distinctly outperforms the flow-based blending. However, this is to be expected. The goal of the flow-based blending approach aims to improve problems that arise due to the difference between the real and proxy geometries, and it introduces possible

[4]The naïve algorithm error values of the checkersphere were omitted because they were so much higher than the other values that the scale of the plot became too small.



(a) Error values of flow-based blending in the square room



(b) Error values of flow-based blending in the oblong room

Figure 4.12.: Scene analysis of flow-based blending results in the square and oblong rooms

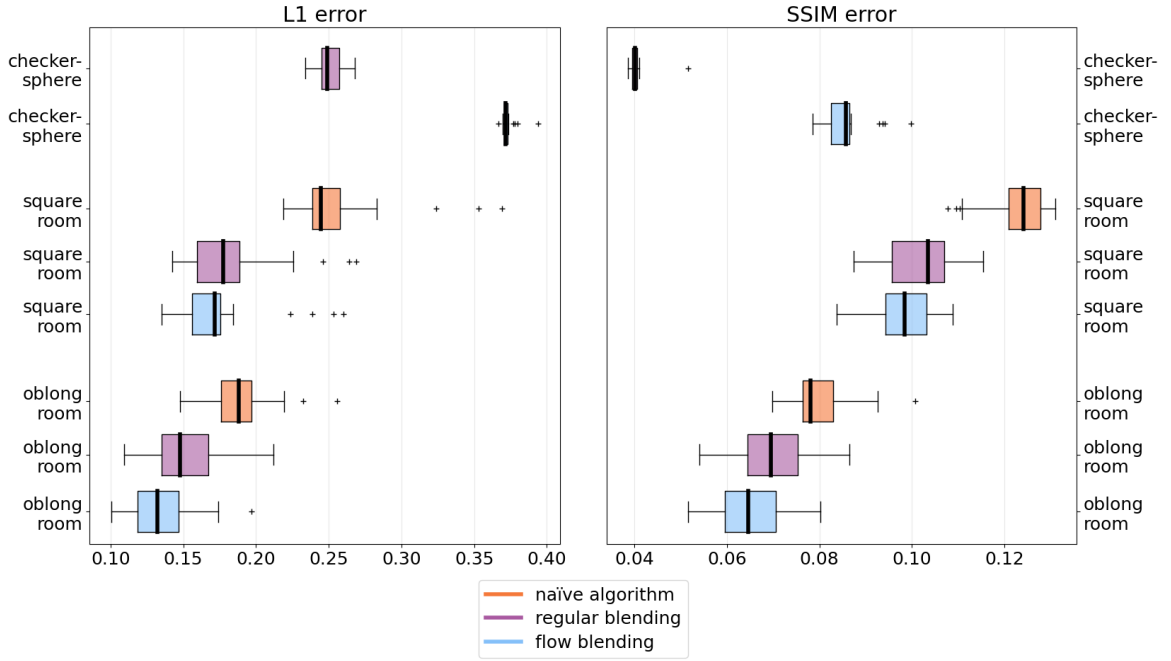


Figure 4.13.: Comparing the distribution of the results in different scenes for regular blending and flow-based blending

inaccuracies in order to do so (e.g., viewpoint selection, optical flow, ray approximation). Therefore, it is not surprising that the results are less accurate than the results of the regular blending.

For the square room and the oblong room, the $\Delta L1$ and $\Delta SSIM$ (i.e., the difference, or "improvement") of the flow-based blending compared to the regular blending are shown in Figure 4.14. Positive $\Delta L1$ and $\Delta SSIM$ values (red) signify that the flow-based blending produced a *worse* result (i.e., higher error value) than the regular blending, and negative $\Delta L1$ and $\Delta SSIM$ (blue) signify that the flow-based blending produced a *better* result (i.e., lower error value) than the regular blending, with stars marking the highest and lowest values. At a glance, it is clear that in all cases in the oblong room, and in all but one or two cases in the square room (depending on the metric) the flow-based blending improves the result based on the L1 and SSIM error metrics.

Figure A.5 (page 86) shows the viewpoint with the lowest (best improved) $\Delta L1$ difference in the square scene. The most visible difference is the rug on the bottom face: In the regular blending result, it shows some ghosting (doubled, offset edges), which has been mostly fixed in the flow-blending result. Other than that, the white coffee table with the blue bowl in the left face is slightly blurrier, but covers a more accurate area than the result of the regular blending. The bottom part of the bookshelf is more accurate. Otherwise the two results are very similar visually.

For the "worst improved" viewpoint L (Figure A.6, page 87) the regular and flow results are also visually very similar. The most visible differences are that the rug shows some ghosting artefacts in the regular blending result, which are gone in the flow-based result. However, the flow-based blending introduced some new artefacts, namely a sharp discontinuity and offset on the rug, and a distorted edge on the coffee table.

Figure A.7 (page 88) shows viewpoint "A" with the best improvement in the oblong scene. Here, the most visible difference is the couch in the left and bottom faces. Where the inner edge of the couch shows severe ghosting artefacts in the regular image, it is much cleaner in the flow-based result. The rug also covers a more accurate area of the image. However, the flow-based blending also introduces new artefacts, for example in the bottom face, where a part of orange couch appears detached from the rest of the couch.

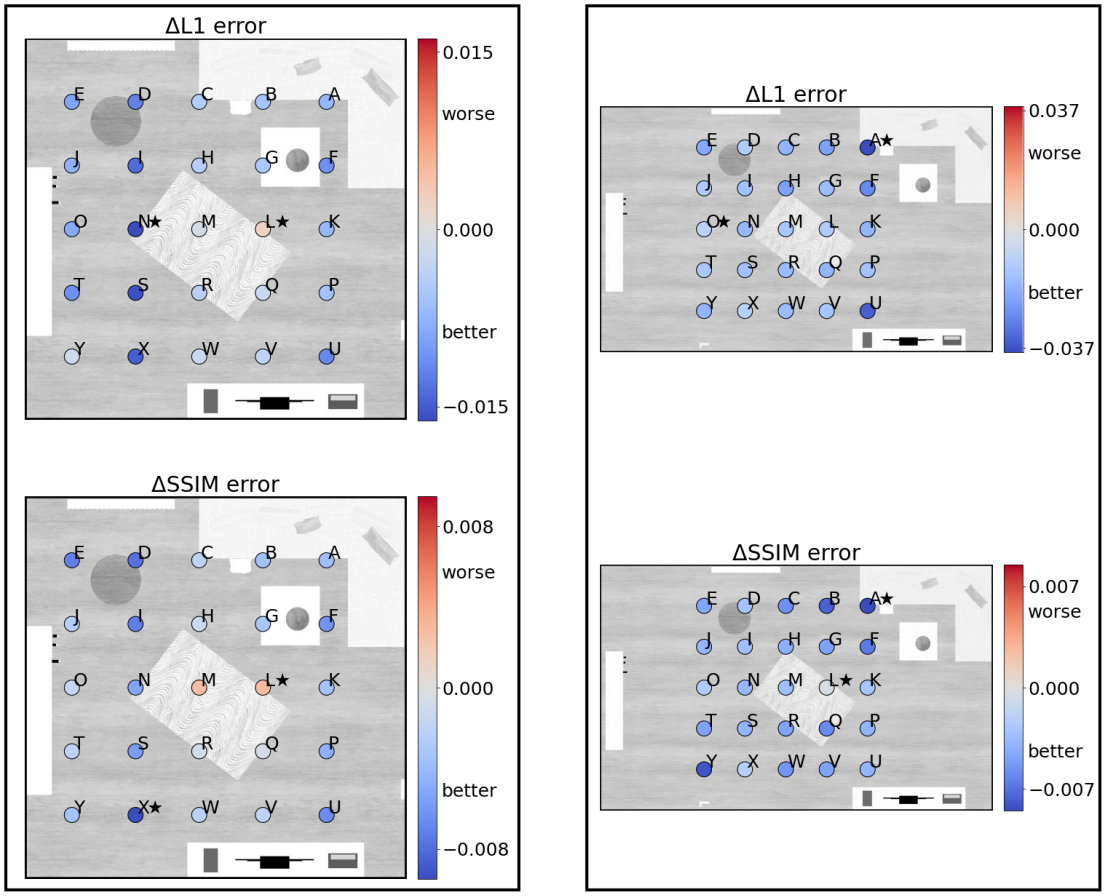
The "worst improved" viewpoint L (Figure A.8) shows even more of these artefacts: The rug in the bottom face has a few extreme discontinuities, and so does the coffee table in the left face (although the coffee table is positioned more accurately in the flow-based blending result). These severe discontinuities are caused by the selection of input viewpoints for flow-based blending: For each ray of the synthesized image, two input viewpoints are selected based on their deviation angle, and whether they are "on either side" of the ray in question (see Section 3.1.3). This means that while traversing the rays of a synthesized viewpoint, there comes a point where the selected input viewpoints for the 1-DoF interpolation A and B suddenly switch to B and C (this will be described in more detail in Section 4.3.4). In these cases, the less accurate the reprojection step is, the more extreme the discontinuity at that place in the image seems to be. Although these discontinuities are visually extremely irritating, it is possible that they do not have a large impact on the error metrics.

Scenario Synopsis The goal of this scenario was to evaluate the effects of the scene geometry on the accuracy of the results of regular and flow-based blending. This includes the details of the geometry given by the shape and placement of objects within the scene, as well as the general shape of the scene. Concerning the geometry details, i.e., the objects within the scene, the results of the scenario are fairly clear: both blending techniques performed better, the further away specific objects were from the synthesized point. The closer the synthesized point was to an object, the more ghosting and doubling artefacts, and positional inaccuracies showed up in the regular blending results. In some of these cases, the flow-based blending synthesized a more accurate image, showing fewer positional inaccuracies and correctly synthesizing some of the object shapes. However, the flow-based blending also introduced some new artefacts, predominantly abrupt discontinuities. These discontinuities were not necessarily recognized by the error metrics and needed to be identified by visual inspection. In the case of extreme proximity to an object (e.g., in front of the bookshelf), both the regular blending and the flow-based blending produced highly inaccurate results. In the case of the flow-based blending, this was due to inaccurate optical flow. However, even though the optical flow was inaccurate, the flow-based blending result was not worse (in terms of the error metrics) than the regular blending result.

As for the impact of the general shape, the results are not very significant, due to the choice of the oblong and square rooms along with the chosen captured and synthesized viewpoints. The different distances of the objects to the synthesized viewpoints within the two scenes overpowered most effects that the different basic shape might have had. It was clear, however, that in the case where the scene geometry matched the proxy geometry, the flow-based blending performed worse than the regular blending.

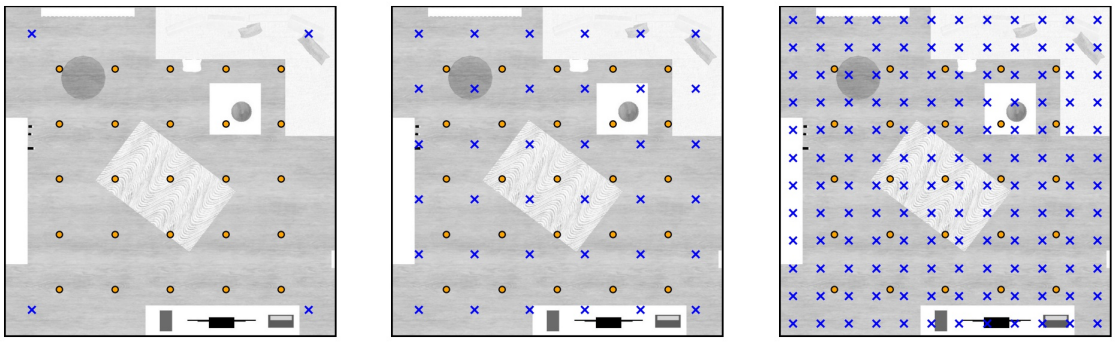
Density of the Captured Viewpoints

The previous scenario demonstrated that close proximity of a synthesized viewpoint to an object can have a strong adverse effect on the accuracy of the result. A possible way to



(a) Square room (b) Oblong room

Figure 4.14.: $\Delta L1$ and $\Delta SSIM$, "improvement" of flow-based blending over regular blending results in the square and oblong rooms. The stars denote the best and worst cases.

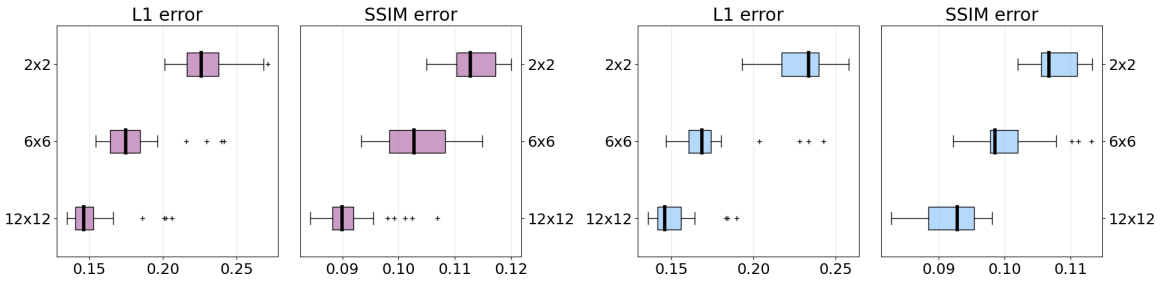


(a) A density of 4 viewpoints (2x2 grid)

(b) A density of 36 viewpoints (6x6 grid)

(c) A density of 144 viewpoints (12x12 grid)

Figure 4.15.: The different captured viewpoint densities (blue) in the square room with the synthesized viewpoints (orange)



(a) Regular blending

(b) Flow-based blending

Figure 4.16.: Comparing the distributions of the results in the square room with different densities separately

mitigate this problem could be to increase the density of the captured viewpoints near objects and walls. The scenario presented in this section explores the impact of viewpoint density on the accuracy of the results.

To test the effect of viewpoint density, the square room is used, since the viewpoint grid covers the entire area of the room, which guarantees higher proximity to all of the objects. Three different versions of the grid of captured viewpoints are used: a 2x2 grid with a spacing of approximately 2.3m (Figure 4.15a), the 6x6 grid with a spacing of approximately 60cm, which was also used in the previous scenario (Figure 4.15b), and a 12x12 grid with a spacing of approximately 30cm (Figure 4.15c). The 2x2 grid was chosen since it is the minimal grid to cover the entire room, and the 12x12 grid was chosen, since it halves the distance of the 6x6 grid and as such, retains the relative position of the captured viewpoints compared to the synthesized viewpoints. If a different grid was used, for example 10x10, some synthesized viewpoints would be closer to captured viewpoints than other synthesized viewpoints, which may have an effect on the overall results. Like in the previous scenario, 25 viewpoints are synthesized, located near the center of each grid cell. They are offset slightly from the exact center, like in the previous scenario, to demonstrate true 2-DoF synthesis.

Regular Blending Results Figure 4.16a shows the general distribution of the results of the regular blending with varying viewpoint densities. Unsurprisingly, the accuracy of the results improves, the higher the density of the input viewpoints is. In the 6x6 and the 12x12 setup, there are several outliers with a higher error value for the L1 error, which are likely due to the proximity of some points to the bookshelf. In the 2x2 setup, there are no significant outliers, which means that the distribution of the error values is more uniform throughout the scene.

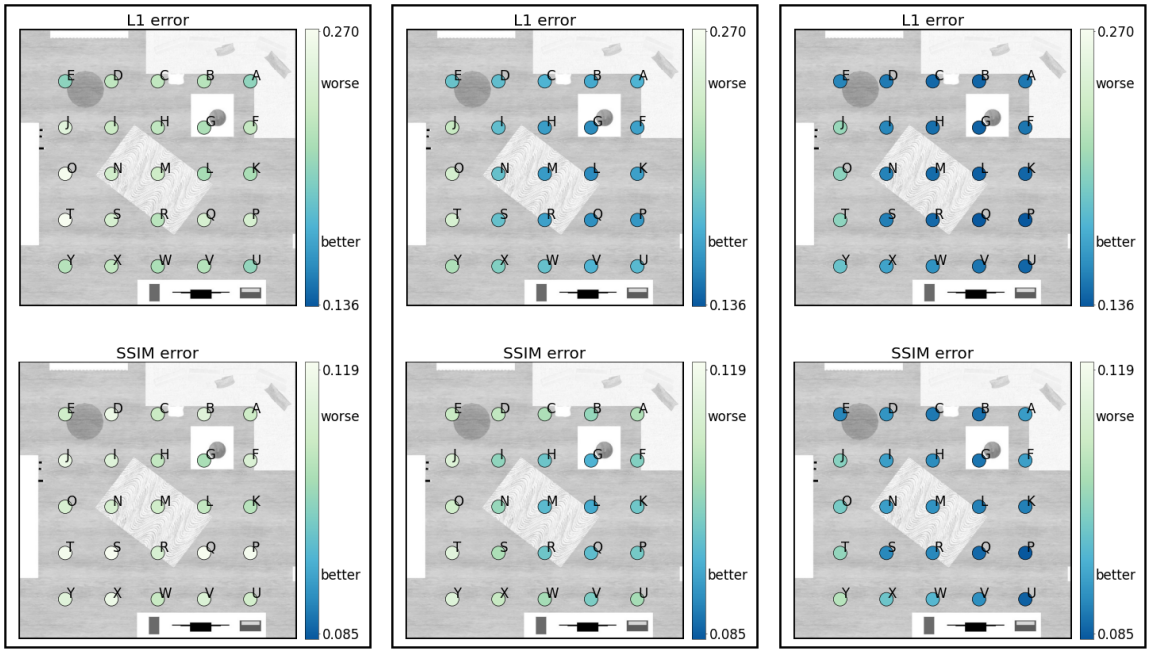
A look at the scene visualization in Figure 4.17 confirms these assumptions: The outliers in the 6x6 and 12x12 setups are caused by the bookshelf on the left side of the room. In the case of the 2x2 setup, the error values are fairly high throughout, compared to the other two setups, which is the reason that the viewpoints near the bookshelf do not register as outliers. Other than the area around the bookshelf, both the 6x6 and the 12x12 scenes have a fairly uniform distribution for the L1 error. For the SSIM error, the areas close to the walls have a slightly higher error than those in the middle of the scene. This is in line with the results of the previous scenario, where the error values near objects or walls tended to be higher.

Although the error values near objects and walls are slightly higher in both the 6x6 and 12x12 setups, the improvement when going from the 6x6 to the 12x12 density is also higher in these areas, as can be seen in Figure 4.18a, which shows the improvement when using the 12x12 scene versus the 6x6 scene. In areas near walls (e.g., the top row for both metrics, viewpoints A-E), or objects (in front of the bookshelf for L1 (O, T), or in front of the TV screen for L1 and SSIM (U,V,W)), the improvement tends to be higher than near the center of the scene (e.g., viewpoints H, G, M, L).

In order to understand more about how the density affects ghosting and other artefacts in the images, one of the better and one of the worse results for all of the setups is examined visually. According to the general tendencies of the values in the different setups (Figure 4.17), synthesized viewpoint T near the bottom left of the room, next to the bookshelf has a comparatively high error value in both metrics for all three scenes and synthesized viewpoint G, above the coffee table towards the top right corner has a comparatively low error value for all of the scenes.

Figure A.9 (page 90) shows why T has a high error value: for the 2x2 scene, while most of the scene is moderately accurate (the objects are in approximately the right place), the bookshelf is shown from the wrong perspective, i.e., from the side versus from the front. The reason for this is that the viewpoint used for reprojection captured the bookshelf from the side. Using only this information, it is impossible to reconstruct the front of the bookshelf. One other effect of the fairly large jump in perspective using only regular blending are the warped walls. This is due to using the sphere as proxy geometry. The warping problems are much less visible in the 6x6 and 12x12 results, since the reprojection jumps are much smaller due to the captured viewpoints being much closer together. The L1 difference images on the right do show an improvement of the accuracy in the 12x12 image over the 6x6 image, but the bookshelf still has many inaccuracies due to its proximity and high detail.

While the difference between the 12x12 and the 6x6 image is less evident in one of the worse results, Figure A.10 (page 91) (viewpoint G) better illustrates the effect of the denser grid. The 2x2 image is an excellent demostration of the problem with using input images with large deviation angles in conjunction with a proxy geometry that is not identical to the scene geometry: The rays used to synthesize this image captured the coffee table at different positions and the reprojection does not account for this, since it only approximates the scene geometry by using the proxy geometry. As a result, the coffee table appears four times. In



(a) Results of regular blending with a 2x2 grid

(b) Results of regular blending with a 6x6 grid

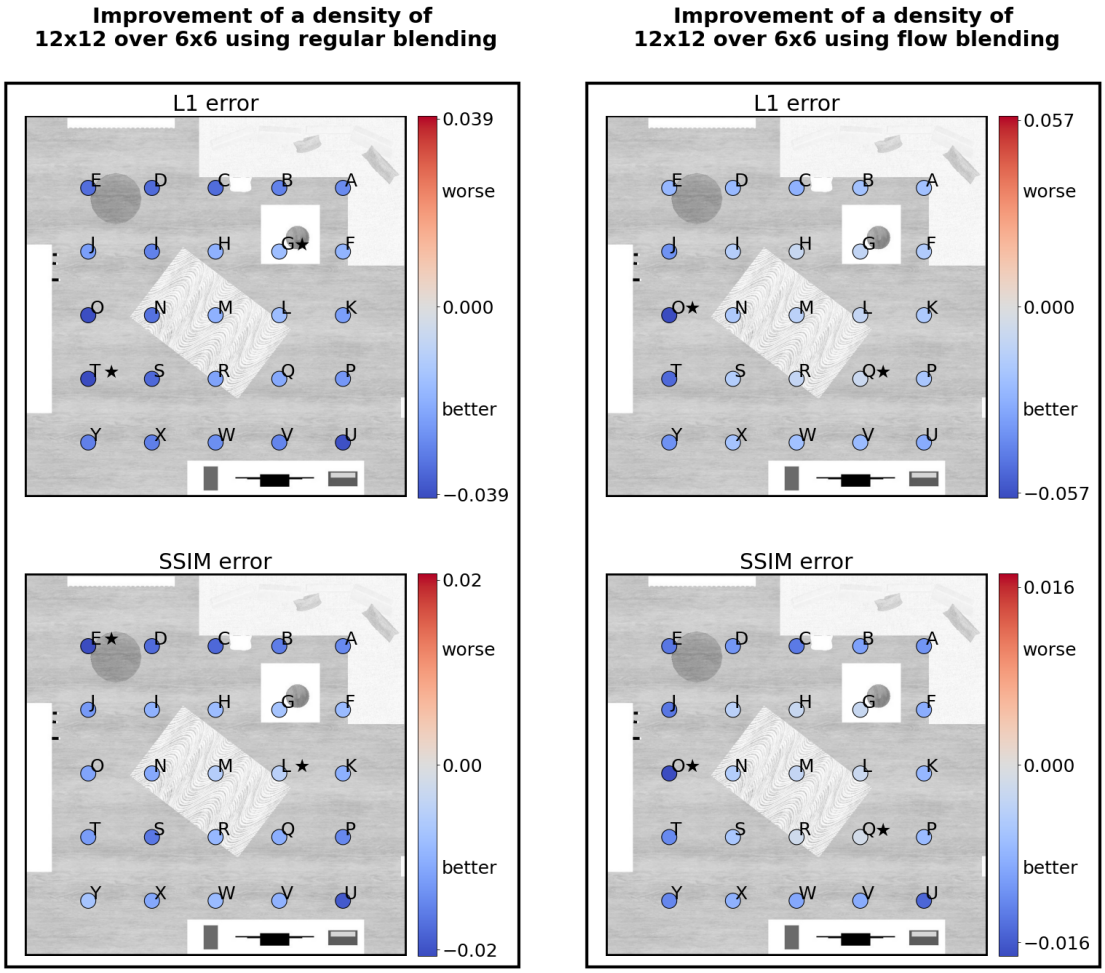
(c) Results of regular blending with a 12x12 grid

Figure 4.17.: Scene analysis of the regular blending results in the square room with different densities

this example, the reprojection errors decrease visibly as the density increases: in the 6x6 image, a slight doubling effect of the coffee table is still visible, whereas in the 12x12 image, the table only appears slightly blurry.

Flow-based Blending Results The error values of the flow-based blending results (Figure 4.16) show similar tendencies as the regular blending results. Like for the regular blending, the 6x6 and 12x12 setups have several outliers in the L1 metric, but otherwise a fairly close distribution. The scene visualization (Figure 4.19 also shows a similar pattern: In the 2x2 scene, the error values are generally high, whereas in the 6x6 and 12x12 scenes, the error values are high near the bookshelf, but generally similar everywhere else. Like in the regular blending scene, the improvement of the values in the 12x12 setup compared to the 6x6 setup is slightly higher near the walls. Since the tendencies are very similar as in the regular blending, it is more interesting to examine the comparison of the regular blending to the flow-based blending, instead of the flow-based blending by itself, as there do not seem to be any findings other than the improvement of the results with higher density, especially near walls and objects.

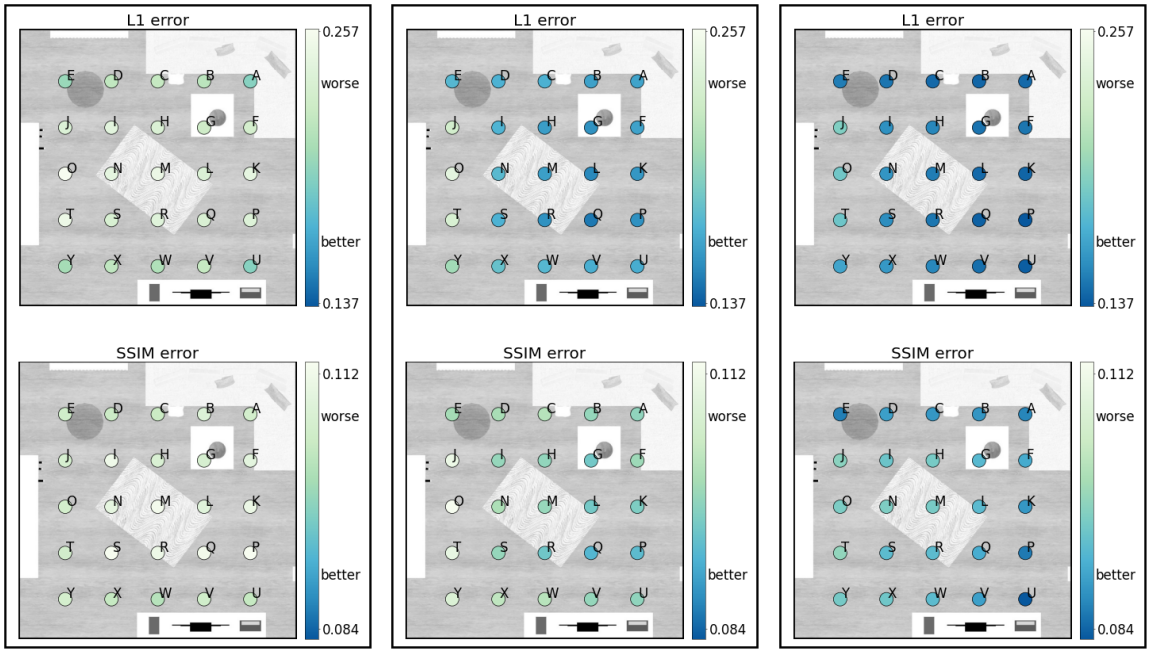
Comparing Regular Blending to Flow-based Blending Results At a glance, it is clear that both the regular blending and the flow-based blending performed distinctly better than the naïve algorithm. Other than that, the distributions of the regular blending and flow-based blending results shown in Figure 4.20 are inconclusive. Only the results of the 6x6 scene show a consistent improvement of the flow-based results compared to the regular results



(a) Improvement for regular blending

(b) Improvement for flow-based blending

Figure 4.18.: Improvement of results using 12x12 density compared to 6x6 density



(a) Results of flow-based blending with a 2x2 grid

(b) Results of flow-based blending with a 6x6 grid

(c) Results of flow-based blending with a 12x12 grid

Figure 4.19.: Scene analysis of the flow-based blending results in the square room with different densities

for both error metrics. For the 2x2 scene, the median L1 error is slightly higher for the flow-based blending, while the error range is lower. The SSIM error range and median for the 2x2 scene are both lower for the flow-based blending. In the 12x12 scene, the L1 error shows an almost identical distribution, whereas for the SSIM result, the general distribution of the flow-based blending is wider (ignoring outliers), which implies that some results were improved while others were worsened.

Figure 4.21 shows the $\Delta L1$ and $\Delta SSIM$ of the flow-based blending versus the regular blending for the 2x2 and the 12x12 scenes. Both the 2x2 and the 12x12 scenes show improved values nearer to the walls, and a slightly worse values near the center of the room. However, although this overview shows similar tendencies, a look into the synthesized images shows some of the fallacies of the metrics.

For the 2x2 scene, the regular blending produced relatively inaccurate results, due to the extreme perspective changes. The flow-based blending relies on optical flow, which does not handle large displacements well (even with Blender optical flow). The results show that in the 2x2 scene, the optical flow algorithm did not produce very accurate results in general, either, which is visible in both the "best" and "worst improvement" images. The "best improved" viewpoint "T" (Figure A.11, page 92) is the image that was one of the worst rated for the regular blending, since it did not show the bookshelf from the correct perspective. In the flow-based version, a blurry shadow of the bookshelf is visible, however, the whole image shows blurry distortions of all of the objects. So although the metrics show improved values, visually it is much harder to distinguish the different objects. The same holds true for the "worst improvement" image, where both metrics measured a slight increase in error value

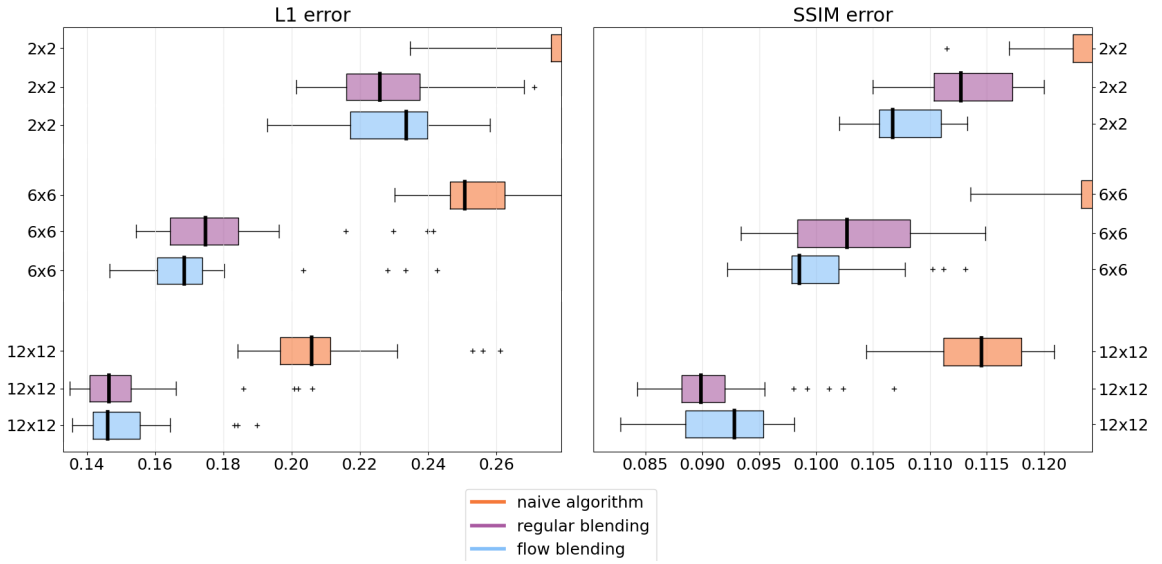
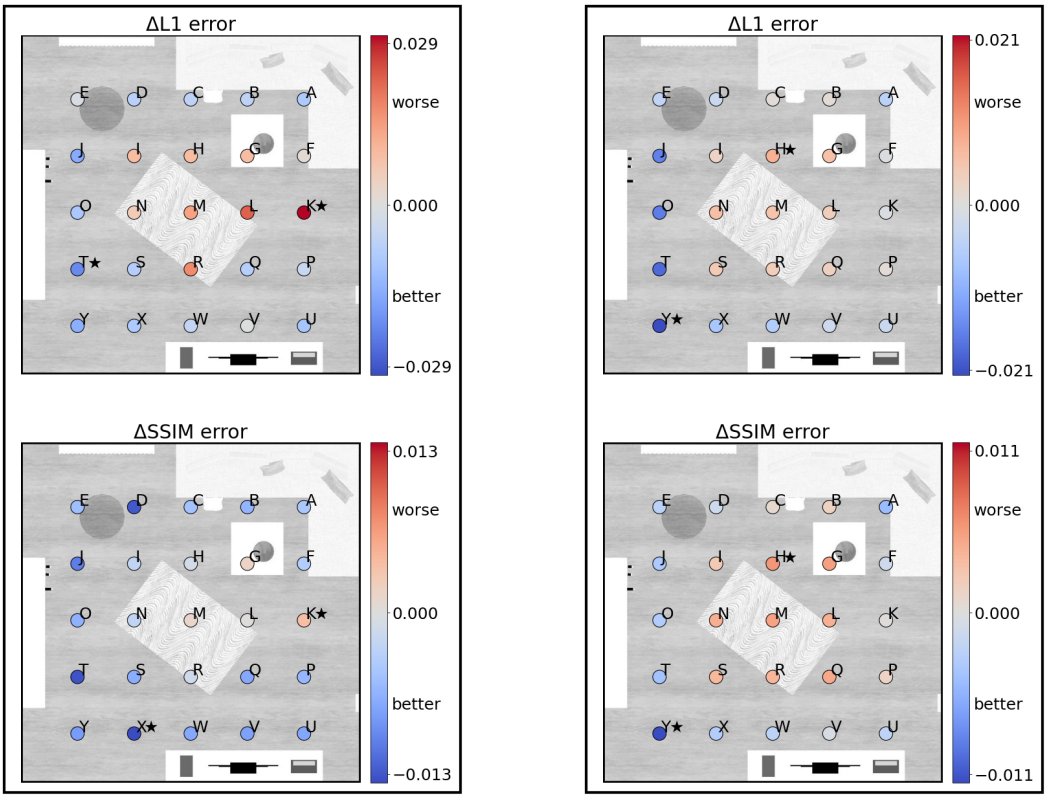


Figure 4.20.: Distributions of error values in the square room with different densities of captured viewpoints for the regular blending and flow-based blending

from the regular to the flow-based result. Figure A.12 (page 93) shows point "K", and while the regular blending result shows extreme artefacts due to doubling, the flow-based version, due to the failure of optical flow, exacerbates the problem: some objects, such as the rug, are reduced to a blurry spot, whereas others, such as the coffee table, suffer from even worse blurring. In summary, it is safe to say that the captured viewpoints in the 2x2 density are too far apart to produce satisfying results with either blending technique.

As for the 12x12 scene, the error metrics show a slight decrease in error values near the walls, and a slight increase of error values near the middle of the scene. However, when looking at the best and worst improved viewpoints ("Y" and "H", respectively), very little difference is actually visible. In the best improved image (Figure A.13, page 94), the flow-based blending slightly improves the accuracy of the bookshelf in the right and back faces, although some ghosting artefacts remain. It also does not display an artefact present in the front face of the regular blending result (a white blurry spot, which appears due to the use of an input viewpoint that had small deviation angles in that area, but seems to have captured a part of the bookshelf with the respective rays). However, other than that, the two images are visually extremely similar. The same holds true for the worst improved image (Figure A.14, page 95): Looking at the L1 difference images, it is hard to see any difference between the two, but when comparing the synthesized images, it is noticeable that the flow-based blending synthesized cleaner outlines on the rug and the coffee table than the regular blending, where there are some ghosting artefacts. This change hardly has an effect on the accuracy however, and possibly did not have any impact on the error values. A factor that was possibly detrimental to the error value of the flow-based result are the black lines that appear near the face edges and are not present in the regular blending result. These black lines originate from a bug in one of the external libraries (see Section 3.2.5) and only affect the flow-based blending. In cases where the synthesized images are so similar, it is possible that these artefacts skew the results in favor of the regular blending.

The 6x6 scene is more straightforward. Figure 4.22 shows the improvement of the flow-



(a) 2x2 setup

(b) 12x12 setup

Figure 4.21.: $\Delta L1$ and $\Delta SSIM$, "improvement" of flow-based blending over regular blending results in the 2x2 and 12x12 setups. The stars denote the best and worst cases.

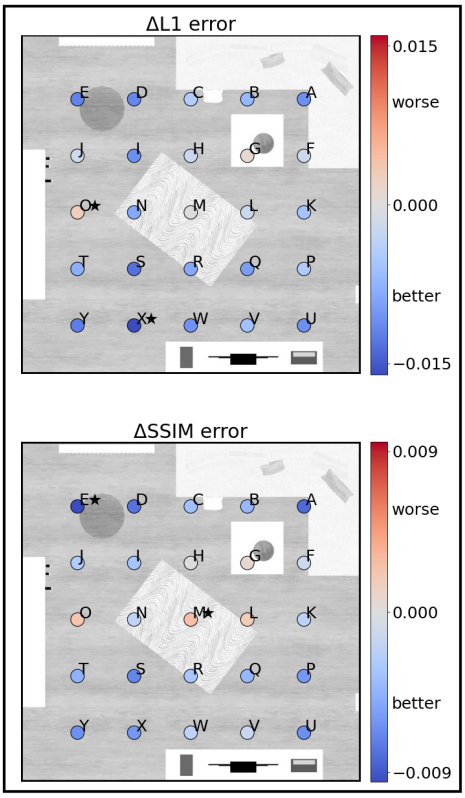


Figure 4.22.: $\Delta L1$ and $\Delta SSIM$, "improvement" of flow-based blending results over regular blending results in the 6x6 setup

based blending results over the regular blending results. As the improvement is very similar to the improvement in the previous scenario for the square room (Figure 4.14a), where an improvement was achieved by the flow-based blending in most cases, the 6x6 setup is not examined in more detail.

Scenario Synopsis In summary it can be said that for the tested room (and presumably in most other cases, as well), increasing the density of the captured viewpoints also increases the accuracy of the results. In the tested cases, the improvement was more apparent near objects and walls. As a result, it is most likely best to capture an irregular grid of viewpoints, with a denser distribution of captured viewpoints near objects and walls, and a sparser distribution where there are no objects.

In the extreme case of the 2x2 density, both the regular and the flow-based blending lead to extreme artefacts, due to the large jumps in reprojection and the failure of the optical flow algorithm. For the 6x6 density, the results were unambiguous: both metrics produced lower error values for the majority of flow-based blending results. The 12x12 density resulted in comparably low error values and few artefacts for both regular and flow-based blending. In this case, the metrics were ambiguous, since the difference between the two versions was very small. A visual evaluation of some of the samples indicates that the flow-based blending improved ghosting artefacts, which visually improved the images, but also introduced artefacts due to a bug in an external library, which may have skewed the

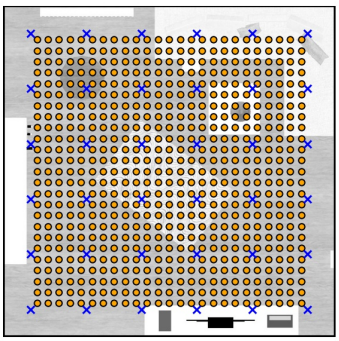


Figure 4.23.: The dense grid of synthesized viewpoints (orange) and the 6x6 grid of captured viewpoints (blue) in the square room

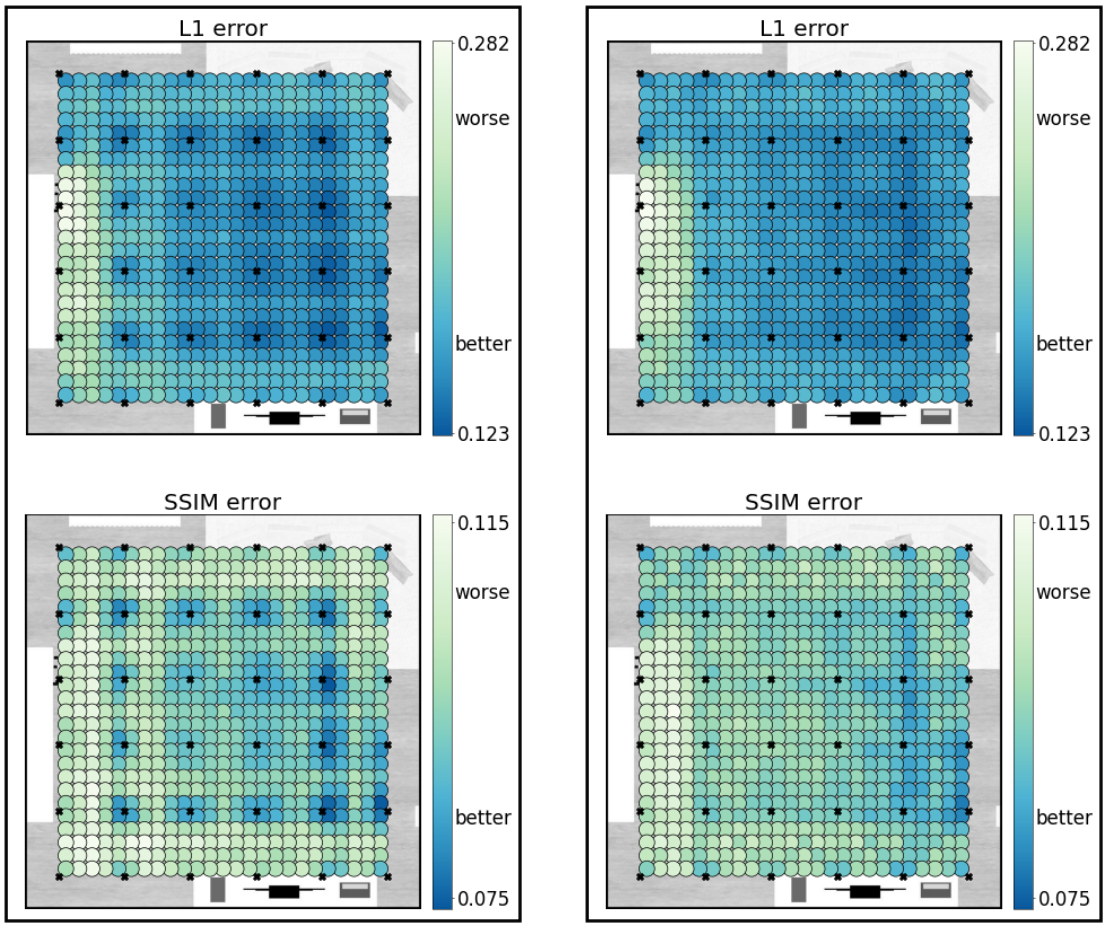
results unfavorably.

Position of Synthesized Viewpoints Relative to Captured Viewpoints

In most of the previous results, the flow-based blending showed an improvement over the regular blending. However, the locations of the synthesized viewpoints were chosen explicitly so that the regular blending would most likely have the most difficulty: areas near the center of the grid cells where the deviation angles would be comparably high. Naturally, this is not the only location that comes in question for synthesis. In fact, all locations within the convex hull can potentially be synthesized by the 2-DoF algorithm. In order to gain a more general understanding of the impact of the relative positons of the captured viewpoints on regular versus flow-based blending, this scenario synthesizes a dense grid of viewpoints, instead of choosing a few select locations in the scene, as was done in the previous scenarios.

Based on the insights gained from the last two scenarios, the square room is used with a captured viewpoint density of 6x6. The square room gives the advantage of covering the whole possible space, and a density of 6x6 with a spacing of 60cm is a conceivable distance for extrapolation to real scenes. A grid of 25x25 viewpoints is synthesized, totaling 625 synthesized points (Figure 4.23).

Regular Blending Results In this scenario, only the position of the viewpoints within a single scene is examined, so the distribution can be inspected directly in the scene analysis visualization (Figure 4.24a). The dense coverage of synthesized viewpoints gives a fairly detailed picture of the effects of the location relative to the scene, and especially relative to the location of the captured viewpoints. It is immediately striking that the synthesized viewpoints in the close vicinity of a captured viewpoint have a distincly lower error value than viewpoints that are farther away. The exceptions to this are the synthesized viewpoints near the walls and near the bookshelf. The observation of higher error values near the bookshelf and walls are consistent with the observations in the previous scenarios. However, this scenario shows that the error values are higher near walls and objects, independent of whether the synthesized viewpoints are close to the captured viewpoints or not. This phenomenon is clearly visible in the L1 error visualization, but even more so in the SSIM error visualization, where the accuracy dropoff is even more extreme.



(a) Results of regular blending

(b) Results of flow-based blending

Figure 4.24.: Scene analysis of regular and flow-based blending results

Flow-based Blending Results The error values of the flow-based blending results (Figure 4.24b) display a different pattern than those of the regular results: While the error values are lower in the vicinity of captured viewpoints, they are comparably low in the vertical and horizontal spaces between the captured viewpoints, as well. This implies that it does not have a significant impact on the error values, whether a synthesized viewpoint is very close to a captured viewpoint, or anywhere between a set of horizontally or vertically adjacent viewpoints. In general, the majority of the values is fairly similar, with clear outliers only visible near the bookshelf.

Comparing Regular Blending to Flow-based Blending Results Figure 4.25 shows the $\Delta L1$ and $\Delta SSIM$ error values of the results of the flow-based blending compared to the results of the regular blending. Here, the results are striking: In general, the error values of synthesized points in close proximity to captured points go up (i.e., the accuracy decreases) when using flow-based blending. An exception to this are the captured points near the walls of the room: In these cases, the flow-based blending improves the results in the majority of cases. Also, in the areas where the distance to the captured viewpoints is highest, the flow-based

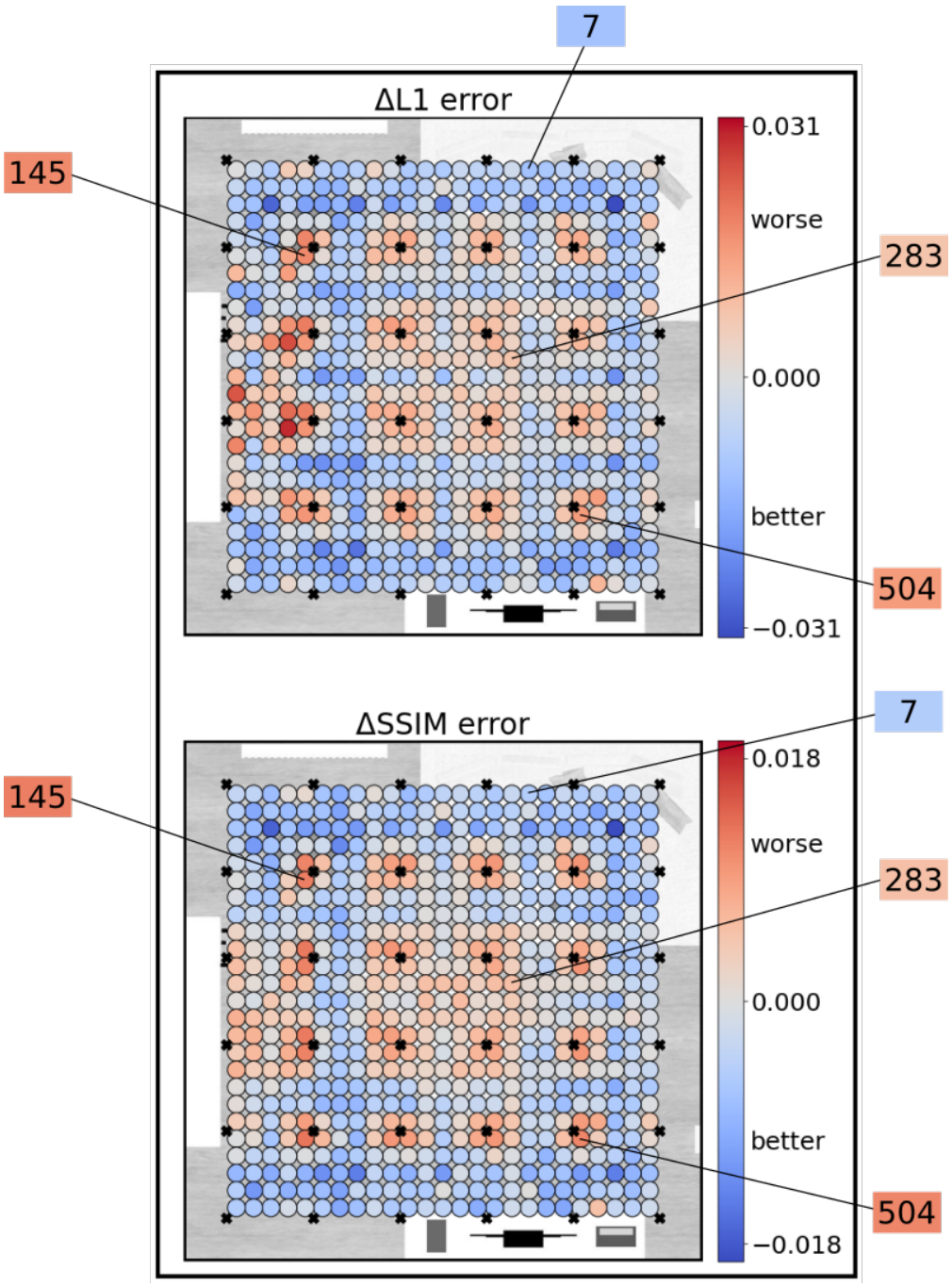


Figure 4.25.: $\Delta L1$ and $\Delta SSIM$, "improvement" of flow-based blending results over regular blending results for 625 synthesized images in the square room. The annotated points are examined more closely.

blending also performs better. An area where the results are ambiguous is the area near the bookshelf. Here, the improvement or degradation cannot be clearly attributed to the proximity to a captured viewpoint or the proximity of the bookshelf. It must be kept in mind, however, that in the case of the bookshelf, the Blender optical flow does not yield good, or even acceptable results, which has a direct, detrimental effect on the flow-based blending, so the results near the bookshelf should not be taken into strong consideration.

Many cases where the flow-based blending yielded a better result than the regular blending have already been shown in the previous scenarios. In this scenario, it is more interesting to examine the cases in which the flow-based blending performed only slightly better, or worse than the regular blending, in order to understand the problems the flow-based blending may still have.

Starting with an example where the flow-based blending performed slightly better than the regular blending, Figure A.15 (page 96) shows viewpoint 7 at the top edge of the room. Although the error value is not significantly lower than that of the regular result, the accuracy is visibly improved, as well as there being no discontinuities or doubled edges. This is especially visible on the coffee table, the rug, and the pictures on the wall. A possible reason for the comparatively small improvement of the error values despite the visually clear improvement could be the black line artefacts that are caused by the external library bug.

Next, a result near the middle of the scene is examined (Viewpoint 283), where the flow-based blending performed slightly worse than the regular blending, even though the viewpoint is located farther away from a captured viewpoint. Figure A.16 shows that part of the cause for this is a severe displacement artefact in the regular blending result, which decreases the accuracy. Other than this, the images are extremely similar.

Finally, the most unexpected case: directly adjacent to a captured viewpoint, where the flow-based blending performed worse than the regular blending. Figure A.17 (page 98) shows viewpoint 145 near the top right corner of the room. The results of the regular and flow-based blending are very similar, except some distinctive artefacts in the flow-based blending result, where the blue table is slightly misshapen, and the coffee table displays a slight displacement. However, these are visually fairly unobtrusive. Figure A.18 (page 99) shows viewpoint 504, near the bottom right of the room. Here, the difference between the flow-based and the regular results is barely visible: The coffee table has a slightly less accurate position. In this case, the black line artefacts may again be part of the reason why the flow-based blending result has a higher error value.

Scenario Synopsis In summary, in this scenario, the results of the flow-based blending generally yielded lower error values than the regular blending results in areas close to objects and walls, as well as in areas further away from captured viewpoints. On the other hand, the regular blending generally performed better than the flow-based blending in close proximity to captured viewpoints. This was more pronounced towards the center of the scene, where the regular blending generally performs better due to higher distances to detailed objects. However, since the results of both techniques were visually very similar near captured viewpoints, it is possible that the worse performance of the flow-based blending was in part due to the external library bug, which introduced black lines and a slight pixel offset in only the flow-based blending results.

4.3.4. Discussion

The evaluation of the three scenarios using virtual data gave a first impression on the general performance of the flow-based blending compared to the regular blending under different circumstances, as well as indicating some of the basic problems and challenges of the flow-based approach.

In many of the tested cases the flow-based blending produced lower error values than the regular blending. However, the evaluation also uncovered some factors that decreased the accuracy of the results, originating from the implementation details as well as the underlying approach. The most important of these factors are:

- failure of the optical flow algorithm
- bugs in one of the external libraries
- deviation-angle-based choice of input viewpoints with abrupt change
- ray approximation

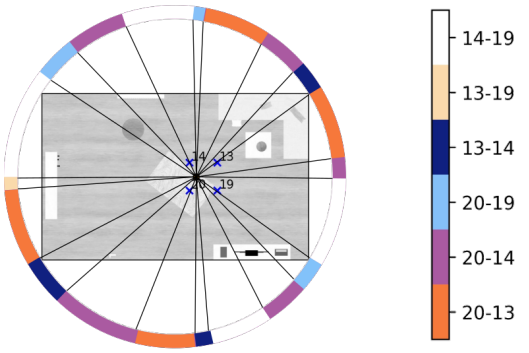
In general, there were certain objects in the square and oblong rooms for which the Blender optical flow yielded better results than for others. A notable example is the rug, which was synthesized with clear edges without warping or blurring effects in all of the examined images (there were some discontinuities, but these are not due to the 1-DoF interpolation). The coffee table is an example for the optical flow being partially accurate, since it was often synthesized in an accurate position, but with blurred details. For the bookshelf, on the other hand, the Blender optical flow almost always yielded inaccurate results, which resulted in warped, blurry, and distorted areas. It is clear that the accuracy of the flow-based blending is directly dependent on the accuracy of the optical flow used in the 1-DoF interpolation, so it is possible that better results could have been achieved with a more accurate optical flow algorithm.

Another external factor that may have caused elevated error values for the flow-based blending is the reprojection problem in the external library Skylibs [Hol20]. This bug leads to slight pixel offsets and thin black lines in the flow-based blending results, exclusively, as the regular blending does not use the problematic operation. As a result, it may have had a noticeable impact on the $\Delta L1$ and $\Delta SSIM$ error values in cases where the results were very similar. However, this error is very difficult to quantify, since the flow-based blending result contains strips of a number of 1-DoF interpolated images, which display the problem to varying degrees. As a result, it can only be assumed that this bug may have a noticeable impact on the accuracy in the flow-based blending results.

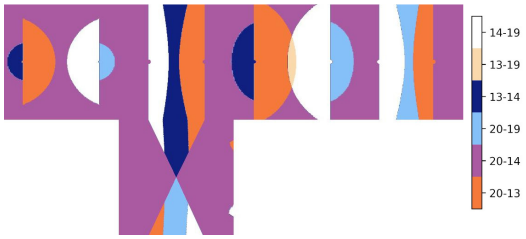
The accuracy of the optical flow, and the bug in the external library are both implementation-related problems, as they are not part of the approach, and may be improved by using different algorithms or libraries. However, there are also some problems that are intrinsic to the approach, and thus need to be discussed in more detail.

One of the most common artefacts in the results of the flow-based blending were severe discontinuities and jumps. This type of artefact is caused by the choice of input viewpoints for the 1-DoF interpolation. Figure 4.26 breaks down the cause of the problem step by step: Looking at an arbitrary synthesized viewpoint in the oblong room (Figure 4.26a) that displays this problem, the abrupt discontinuities are clearly visible, for example in the rug in the bottom and right faces. In the synthesis process for each pixel of this image, two

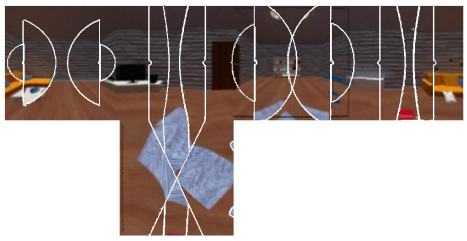
(a) An arbitrary viewpoint in the oblong room



(b) The rays in the scene at which the choice of viewpoints switches (approximated)



(c) The panels of the image that have been assigned to a set of input viewpoints A, B $\in [13, 14, 19, 20]$



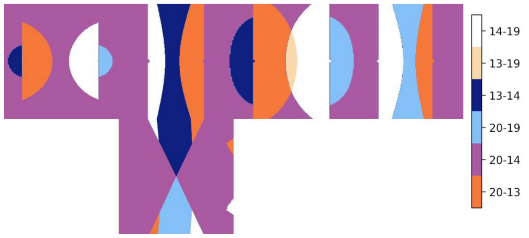
(d) The panels applied to the image

Figure 4.26.: The input viewpoint choice problem in the oblong room

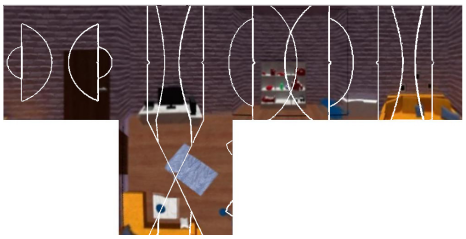
viewpoints A and B are chosen that are on either side of the ray corresponding to the pixel (explained in detail in Section 3.1.3). If more than one viewpoint is found on either side of the ray, the points with the smallest deviation angles are chosen. The interpolation distance between these points A and B is then calculated and the 1-DoF interpolation is performed. Finally the interpolated point is reprojected to the position of the synthesized viewpoint. This succession of operations is performed for each ray of the synthesized image. As a result, there are areas in the synthesized image, where, from one pixel to the next, a different set of input viewpoints is chosen. Figure 4.26b shows the synthesized point in the scene (the oblong room), along with the (approximated) rays where the choice of input viewpoints changes. The surrounding captured viewpoints are viewpoints 13, 14, 19 and 20. This process leads to image areas ("panels") originating from a 1-DoF interpolation between different sets of input viewpoints (Figure 4.26c). When applying these panels to the synthesized image (Figure 4.26d), it becomes clear that these are the source of the abrupt continuities.

The switch between viewpoint sets within the image does not always seem to be a problem. For example, when comparing a very similar viewpoint from the square room with the example from the oblong room, (Figure 4.27), some discontinuities are hardly visible, for example at the coffee table, or on the door. Other discontinuities are visible, but less noticeable, since the displacement is relatively small, for example on the rug, or on the TV cabinet. Since each panel is reprojected separately, it is possible that the severity of the discontinuity is dependent on the accuracy of the reprojection, which again is dependent

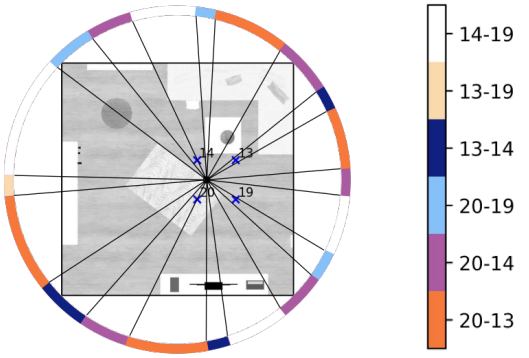
(a) A similar viewpoint to Figure 4.26 in the square room



(b) The panels of the image that have been assigned to a set of input viewpoints A, B $\in [13, 14, 19, 20]$



(c) The panels applied to the image



(d) The panels visualized in the scene (approximation)

Figure 4.27.: The input viewpoint choice problem in a the square room

on the difference between the scene geometry and the proxy geometry. This would explain why the discontinuities are less severe in the square room than the oblong room, however, since the scenario that evaluated the effect of the scene geometry focused mostly on objects within the scene instead of the basic scene geometry, no definitive assertion can be made at the moment.

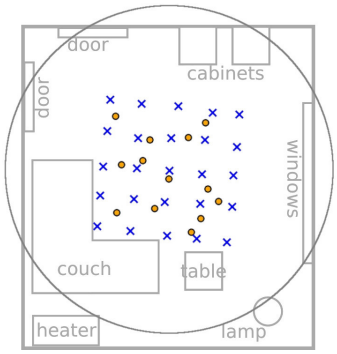
The other possible source of inaccuracies, and a possible contributor to the displacements, is the ray approximation that is necessary for choosing the input viewpoints (detailed in Section 3.1.3). However, since the ray approximation is uniform for all of the flow-based images, it is difficult to judge how much of an impact it has on the results. Based on the analyzed images, it does not seem to create noticeable artefacts that can unambiguously be traced back to it.

4.4. Proof-of-Concept Evaluation using a Real Scene

Following the extensive evaluation of the virtual scenes, this section presents the results of the 2-DoF synthesis tested on a real scene. The goal of this "proof-of-concept" evaluation is to determine to what extent the insights gained in the evaluation of the virtual scenes hold true when applied to data captured in the real world.

As for the internal and external parameters, only a single scene is tested, using a grid of captured viewpoints with a single density. Within this grid, several viewpoints are synthe-

(a) Latlong image from the center of the room



(b) The captured (blue) and synthesized (orange) viewpoints in the scene

Figure 4.28.: Overview of the real scene

sized at random positions, and the regular blending results are compared to the flow-based blending results.

4.4.1. Data Acquisition

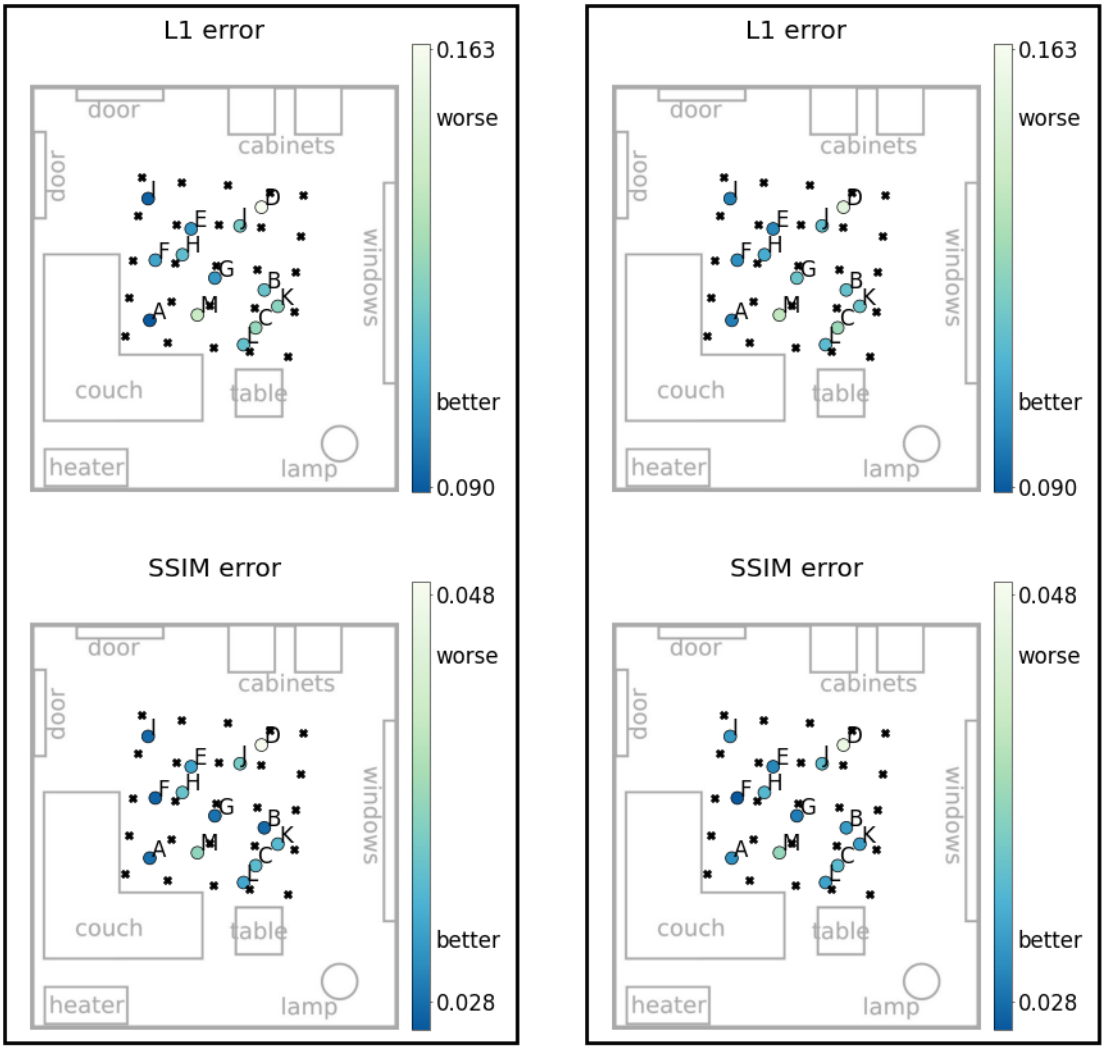
The images were captured using a Ricoh Theta Z1 360° camera with a tripod in an approximately 3m by 2.5m room with a sloped roof and simple furnishings (Figure 4.28a). In order to have approximately uniform distances between the captured viewpoints, a 5x5 grid was mapped out on the floor. The spacing used for the grid was 40cm, as this was estimated to be a feasible distance for optical flow calculations. For the ground truth data, the tripod was randomly placed within the boundaries of the grid, capturing 13 different viewpoints.

After acquiring the image data, the structure-from-motion library OpenSfM [Map] was used to automatically determine the positions and rotations of all of the images. The positions of the ground truth viewpoints were then used as positions for the 2-DoF synthesis. Instead of using a radius encompassing the complete area of the room, a slightly smaller radius was chosen so that the proxy geometry would be closer to the actual scene geometry. Figure 4.28b shows the acquired scene, including the proxy geometry as a gray circle. The proxy geometry is centered around the points, which were captured slightly offset from the center of the scene. This should be kept in mind, as this was not tested in the virtual scenes and may decrease reprojection accuracy.

4.4.2. Results

The same evaluation procedure that was used in the virtual scenes is used for the real scene: First, the L1 and SSIM error values are calculated using the ground truth data, then the scene is analyzed using this data. Based on the results within the scene, interesting points are selected for closer inspection.

Figure 4.29 shows the scene analysis visualization for the regular and flow-based blending. At a glance, they are very similar: The error values of the results of both methods are fairly close together. The error values of the viewpoints towards the left of the scene seem to be slightly lower than those on the right side of the scene, however, there is no clear



(a) Regular blending

(b) Flow-based blending

Figure 4.29.: Scene analysis of error values for regular and flow-based blending results

pattern. Since the results are so similar, first the general tendencies of the error values are examined (e.g., why the error values are especially high for both blending types at a specific viewpoints), and then the results of the regular blending are compared to the results of the flow-based blending.

General Tendencies

First of all, to get an impression of the general look and accuracy of a synthesized image in the real scene, one of the better rated points, viewpoint "I" is inspected. Figure A.19 (page 100) shows that both the regular blending, and the flow-based blending worked fairly well in this case: Most of the objects are positioned accurately, the largest error being on the tripod in the bottom face, which is not really of interest, and on the windows. The windows have a comparably high error, not necessarily because they were synthesized incorrectly, but

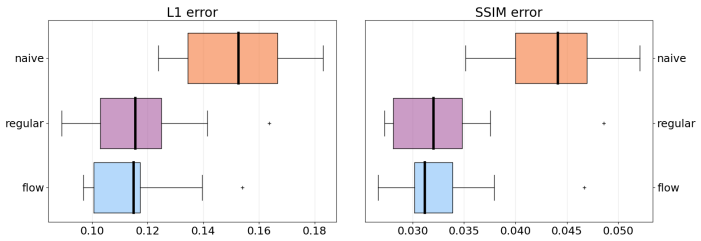


Figure 4.30.: Distribution of the error values of the results from the real scene

mostly because the lighting is different between the captured viewpoints and the ground truth viewpoints[5]. The black lines from the external library bug are unfortunately fairly disrupting visually, since most of the background is white, which makes them stand out (which will also contribute to a higher L1 value).

A closer look at Figure 4.29 shows that viewpoint "D" has particularly high error values for both metrics. Figure A.20 (page 101) shows that the high error values for both methods are due to the proximity of the cabinet elements on that side of the room. Unsurprisingly, the regular blending does not correctly reproject these, since they are not part of the proxy geometry. However, the flow-based blending also fails, presumably because the displacement between the input viewpoints was too large for the optical flow algorithm to handle.

Comparing Regular Blending to Flow-based Blending Results

Figure 4.30 shows the distribution of the error values of the results of the naïve algorithm, the regular blending, and the flow-based blending. At a glance, it is clear that the regular and flow-based blending once again performed better than the naïve blending algorithm. However, the error values of the regular and flow-based blending are extremely close, and the distribution is inconclusive.

Figure 4.31 shows the Δ L1 and SSIM of the regular blending compared to the flow-based blending. For most of the viewpoints, the flow-based blending performed slightly better than the regular blending for both metrics. However, all of the values are very close together, and the difference between the highest and lowest Δ L1 and SSIM is generally very small. In this case, the error values alone are not sufficient to understand the differences between the results of the regular versus the flow-based blending.

A closer inspection of viewpoint "G" (Figure A.21, page 102) shows that most of the details (outlines of the couch, reading lamp, cabinets) are more accurate in the flow-based blending result. However, like in the virtual scenes, the flow-based blending also introduced some artefacts, such as noticeable displacements on the wall and door in the front face, as well as some extreme ghosting on the top face, where the 1-DoF interpolation was not able to correctly calculate optical flow of the ceiling lamp. The large RGB difference of the pixels of the ceiling lamp in the flow-based blending result is most likely also the reason why the $\Delta L1$ error value is significantly higher than the $\Delta SSIM$ value for viewpoint "G".

Examining a result that has a relatively high $\Delta L1$ and $\Delta SSIM$ (i.e., decrease in accuracy

[5]The changes in illumination are due to the timing of the captures: The input viewpoints were captured first, and the ground truth points were captured a little later, when there was already less light. As a result, the synthesized viewpoints (which use the more illuminated captured viewpoints as input) will be compared to the "ground truth" viewpoints that will have less illumination from outside the window.

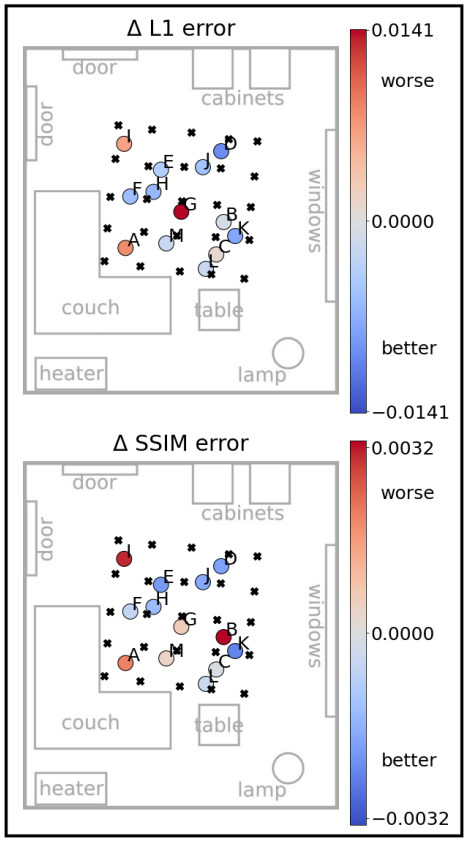


Figure 4.31.: $\Delta L1$ and $\Delta SSIM$, "improvement" of flow-based blending results over regular blending results for the real scene.

for flow-based blending), viewpoint "A" (Figure A.22, page 103), also clearly demonstrates two of the problems of the flow-based blending: The red pillow in the front face is distorted due to failed optical flow, and the discontinuities are clearly visible, both in the front face on the back of the couch, and in the back face on the window. However, the flow-based blending synthesized the cabinets more accurately.

Viewpoint "K" (Figure A.23), where the flow-based blending result produced lower L1 and SSIM values than the regular blending result, shows a distinct inaccuracy in the shape of the lamp in the regular blending result, which is accurate in the flow-based blending result. Also, the area around the lamp, including the small table, and the edge between the wall and the floor is more accurate in the flow-based rendering result.

Judging by the inspected results, it is likely that the remaining results show similar patterns: The flow-based blending results perform better than the regular blending result in some areas, but failed optical flow, discontinuity artefacts, and possibly artefacts due to the external library bug prevent the flow-based blending results from being unambiguosly better.

4.4.3. Discussion

In summary, the proof-of-concept evaluation showed that the insights gained in the evaluation of virtual scenes held true in the real scene: In a number of cases, the flow-based blending performed better than the regular blending, according to the L1 and SSIM metrics. Visually, the flow-based blending managed to improve some of the artefacts introduced by using the proxy geometry, however, it also introduced artefacts caused by the abrupt change in input viewpoints for 1-DoF interpolation, which were visually irritating. Also, in the places where the optical flow algorithm failed, the results were blurred or warped, which is visually more noticeable than the regular blending result, where the objects were left undistorted but possibly viewed from an inaccurate position. Nonetheless, the evaluation in the real scene showed that 1-DoF interpolation can be used in order to improve the accuracy of basic pixel-based synthesis.

4.5. Limitations and Future Work

The detailed evaluation of virtual scenes and the proof-of-concept evaluation of the real scene showed that flow-based interpolation can improve the results of basic pixel-based synthesis with proxy geometry. However, there are some limitations to this evaluation, as well as to the algorithm itself. These will be presented in the following sections, along with ideas on how the approach and evaluation can be improved in the future.

add future work to evaluation

4.5.1. Limitations of the Evaluation

An evaluation is only ever as significant as the parameter space it covers and the metrics it uses. In this case, the evaluation used a relatively small parameter space, and tested the results using predominantly mathematical, pixel-based error metrics. This limits the significance of the evaluation in the following ways:

Firstly, only a limited number of external parameters were tested, for example leaving out more extreme room shapes that deviate strongly from the basic rectangular shape. It is possible that, given a strong deviation from the proxy geometry, the artefacts in the flow-based blending caused by the abrupt change in viewpoint would increase to a degree where

the regular blending would be preferable. Also, only indoor scenes were considered. It is possible that the behavior of the algorithms is is completely different for outdoor scenes with much larger distances between the viewpoint and the scene geometry. Furthermore, the parameters that were tested were not tested exhaustively, so it is possible that some interactions between parameters were overlooked.

Secondly, the metrics that were used (i.e., the L1 error and the SSIM error), can show an improvement between different results, but it is difficult to quantify this improvement. For example, it is not possible to explicitly compare the error values of different scenes (unless they contain very similar colors and patterns), since the L1 error metric is very dependent on pixel color values, and it is difficult to differentiate the exact cause-and-effect of the SSIM error metric. Futhermore, the L1 and SSIM error metrics give no indication on believability or visual preference, since they cannot measure the severity of artefacts for human perception. This means that it is possible that users may prefer the regular blending results over flow-based blending results, due to the smoother transitions of the regular blending between different areas. Although user acceptability was not a criterium for this evaluation, it is crucial for the intended area of appliation, namely Virtual Reality. Another factor that plays into the visual acceptability is temporal consistency, i.e., whether the parallax movement of objects in the scene is believable when navigating through the scene using synthesized views. This is also not tested in the evaluation.

Lastly, due to the bug in the external library, the results of the flow-based blending had a non-quantifiable disadvantage. It is possible that an implementation without the bug would have performed measurably better, but it is also possible that the bug had no significant impact.

4.5.2. Limitations of the Algorithm and Future Work

Naturally, the pixel-based 2-DoF synthesis using flow-based interpolation has its limitations as well. Some of these are intrinsic to the algorithm, and others may be overcome by modifying or adapting the implementation.

The most obvious limitation of the approach is the reliance on decent optical flow. The optical flow algorithms used in this thesis did not always manage to do this, which had a noticeable impact on the results. It could be very advantageous to explore different optical flow algorithms, especially ones that focus on capturing large displacements. Furthermore, it could be possible to undistort the extended cube map before calculating optical flow (e.g., by using a method like the one presented in [SLL19]), which could also improve the accuracy of the optical flow.

The other, visibly most detrimental limitation at the moment is the effect of the abrupt change in the selection of viewpoints for 1-DoF interpolation which results in visual discontinuities. In order to improve this, one approach would be to use a proxy geometry that is closer to the scene geometry, that could for example be approximated using a structure-from-motion algorithm to calculate a sparse scene geometry. An alternative would be to change the selection of input viewpoints for the 1-DoF interpolation, or to introduce some kind of constraint (e.g., a color constraint) in order to blend and soften the abrupt edges.

Finally, the implementation also offers a lot of opportunities for parallelization and the offloading of operations to the GPU, which could enable real-time navigation

5. Conclusion

The approach and implementation presented in this thesis laid the groundwork for a pixel-based 2-DoF synthesis using 1-DoF interpolation. The results of the evaluation showed that, in the majority of cases where the basic reprojection produced significant artefacts due to the difference between the scene geometry and the proxy geometry, the flow-based blending was able to improve the accuracy of the results. However, the evaluation also uncovered some of the limitations of the flow-based approach, predominantly some severe artefacts based on ...

Using the insights gained in the evaluation, it will be possible to improve the 2-DoF synthesis with flow-based interpolation approach, and leveraging its potential for parallelization will make it possible to synthesize images in real-time. This could potentially enable casually captured environments to be experienced interactively, which could enhance a broad range of Virtual Reality applications.

A. Synthesized Images

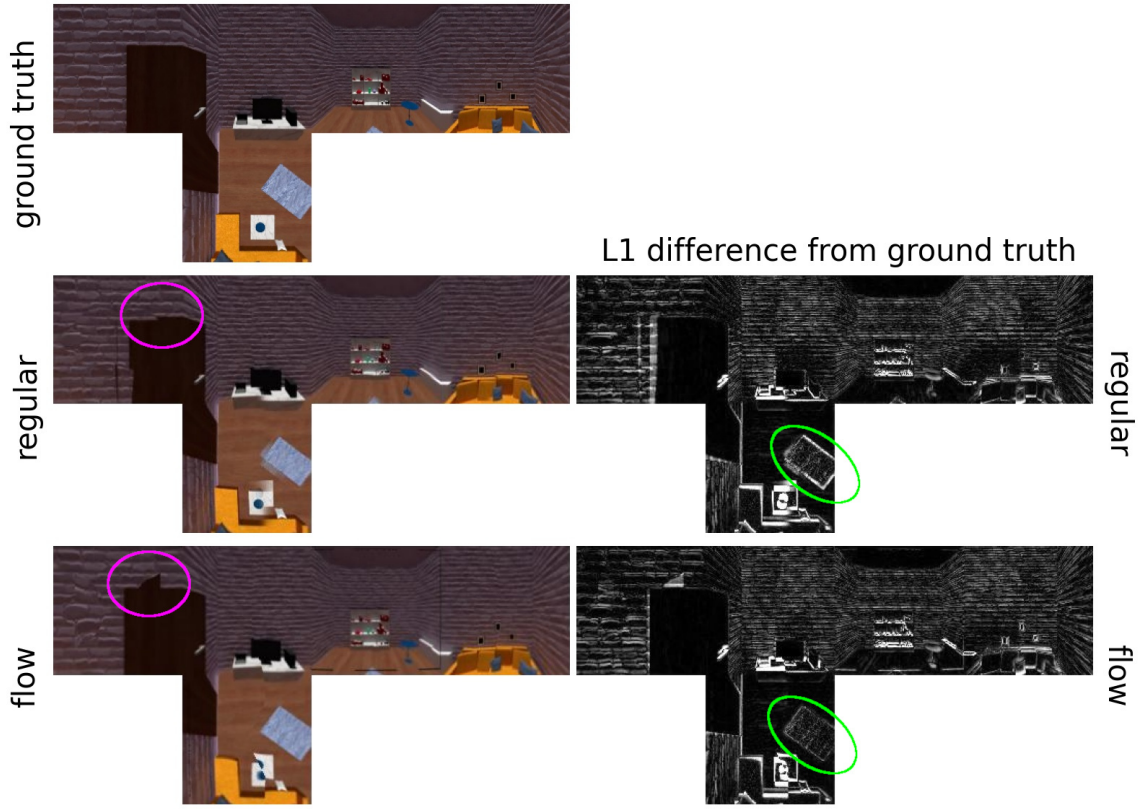


Figure A.1.: Sample inspection of example viewpoint "K": The images are in cube map representation, as this is tends to be more intuitive to understand than latlong representation. Depending on the content, different faces may be omitted.

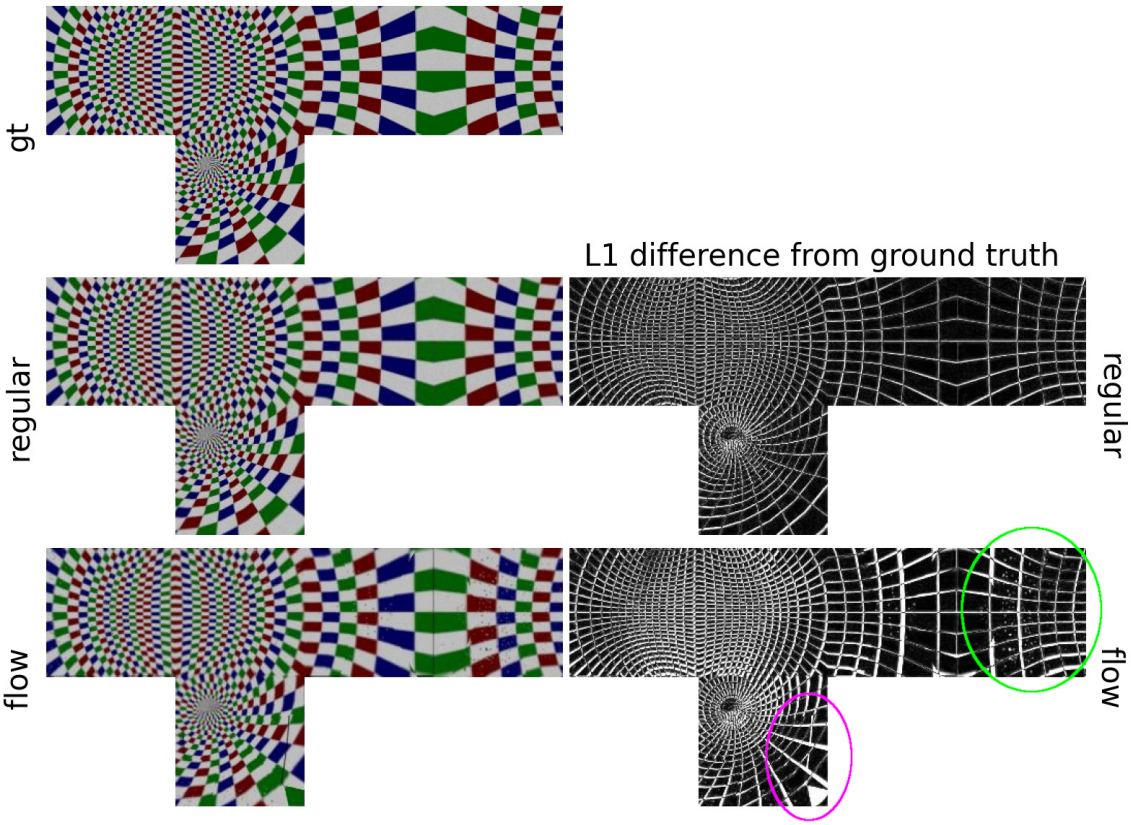


Figure A.2.: Results for synthesized viewpoint “Y” in the checkersphere: The regular blending result is very close to the ground truth, except for some blurriness. The flow-based blending result shows some inaccuracies (magenta) and noise (green)

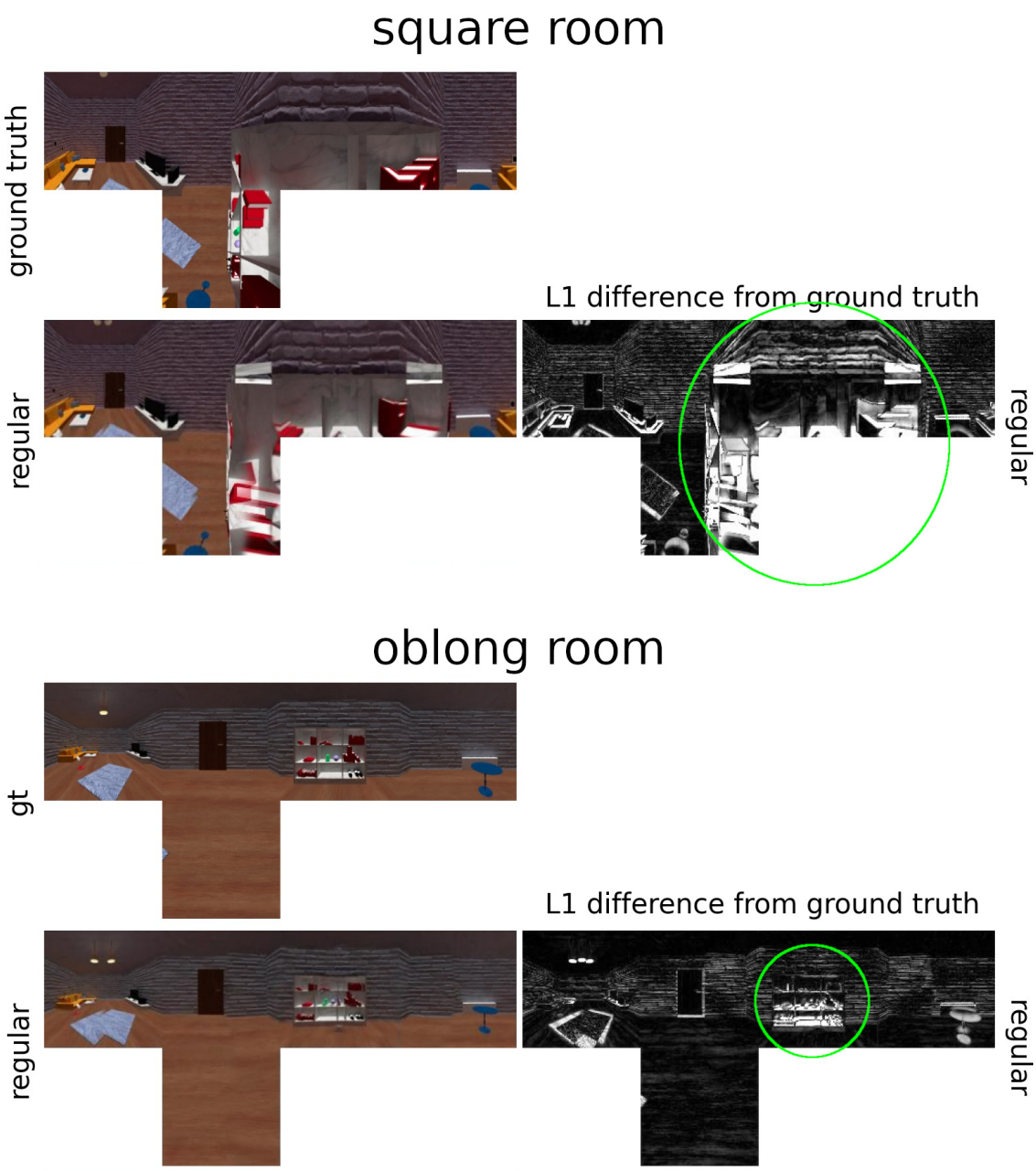


Figure A.3.: Regular blending result of viewpoint "O" in the square and oblong rooms: The bookshelf has a strong impact on the difference in error values (marked in green)

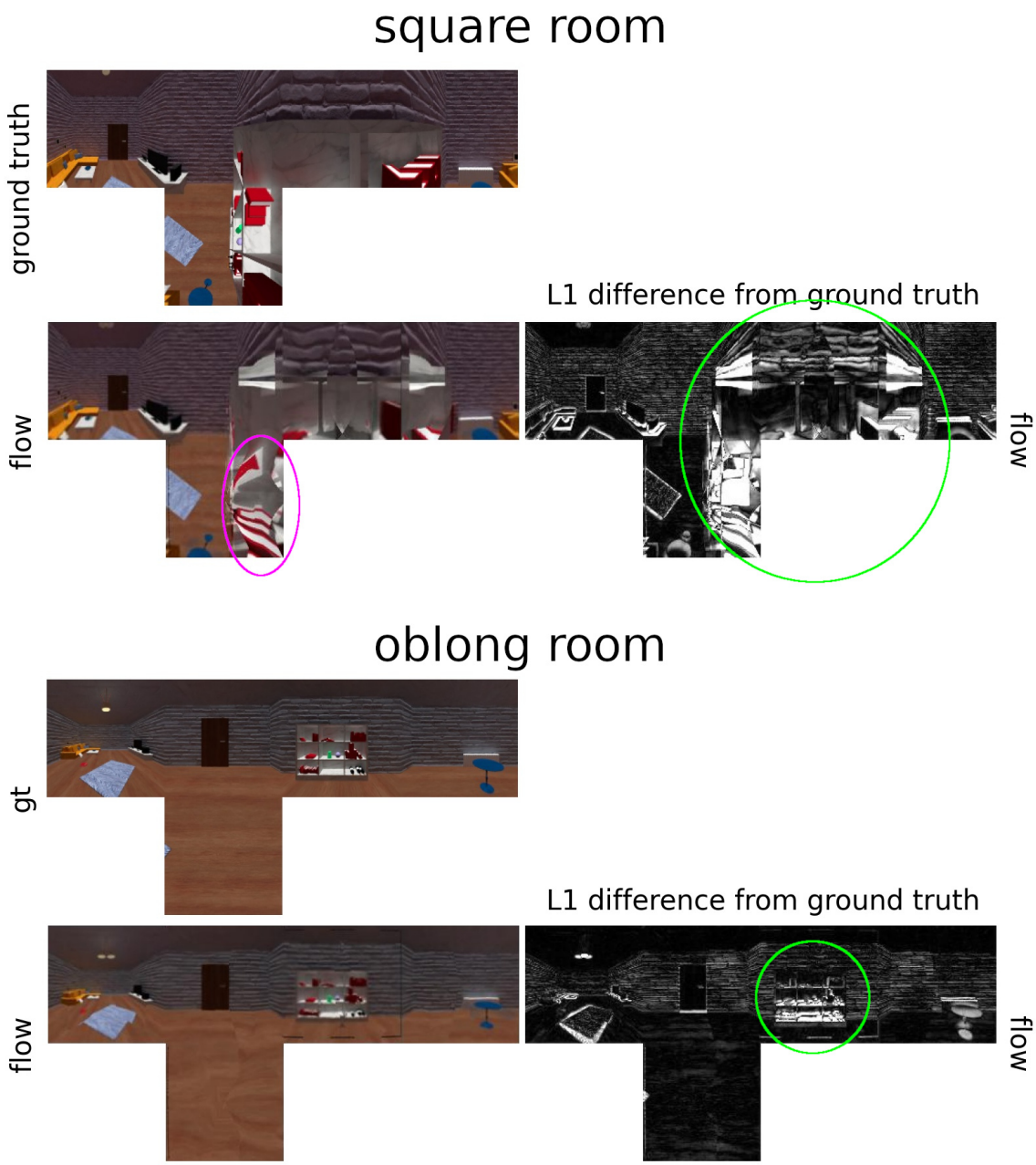


Figure A.4.: Flow-based blending result of viewpoint "O" in the square and oblong rooms: The bookshelf has a strong impact on the difference in error values (green) and the details of the bookshelf are warped due to inaccurate optical flow (magenta).

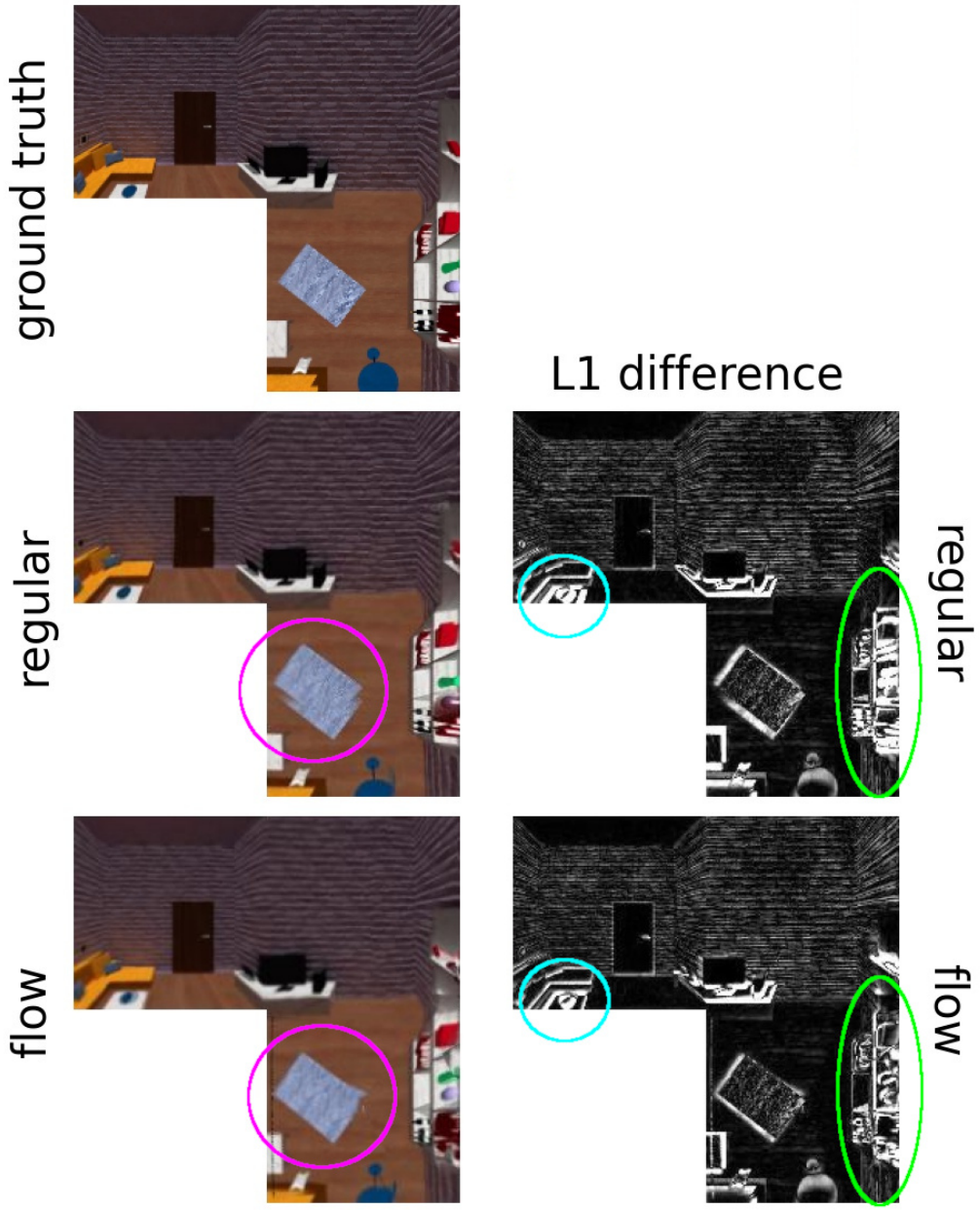


Figure A.5.: Synthesized point "N" in the square room (best improvement for L1, good for SSIM): The flow-based blending removed the ghosting artefacts on the rug (magenta) and improved the accuracy on the coffee table (cyan), and the lower part of the bookshelf (green). The rest of the scene is very similar for both results.

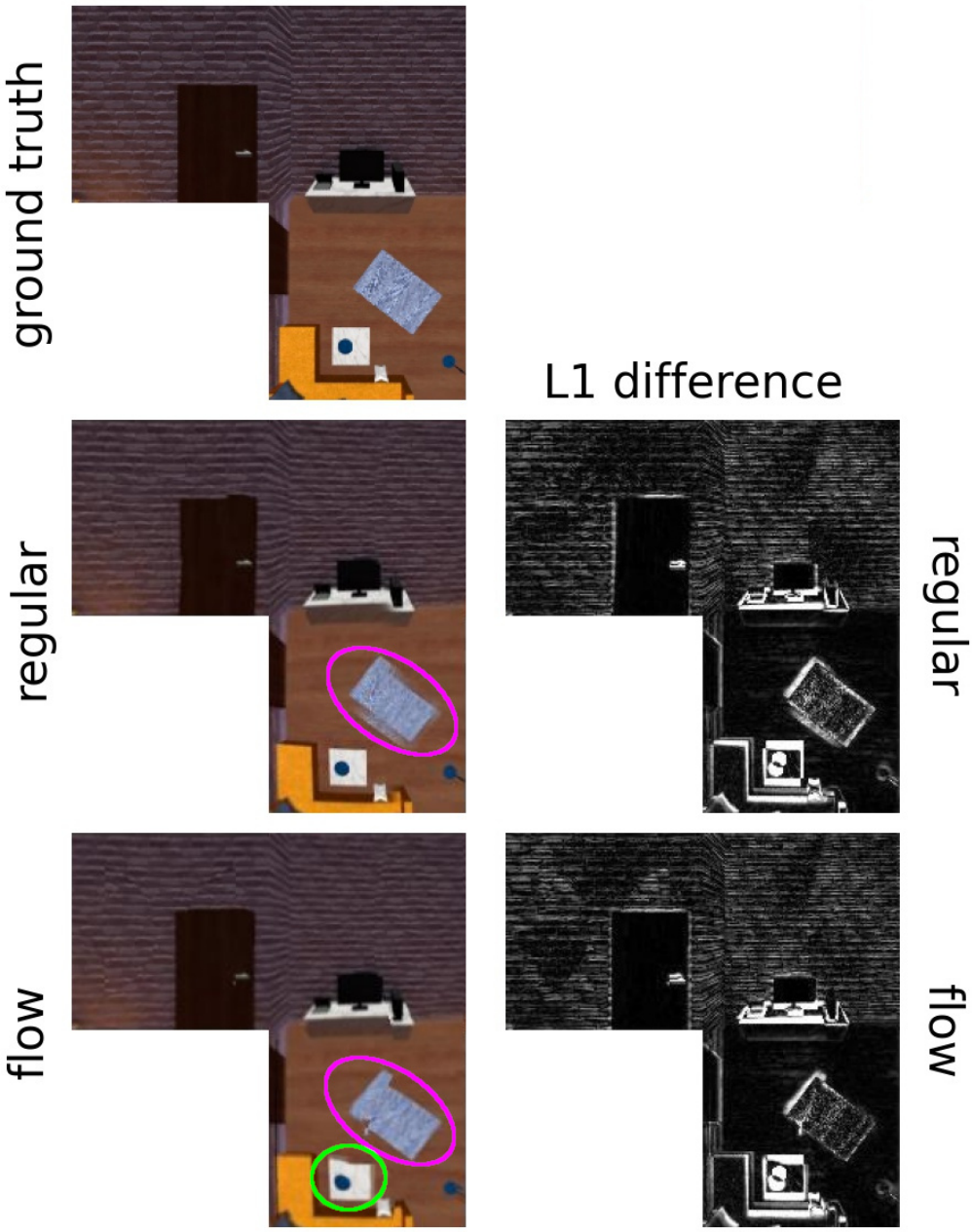


Figure A.6.: Synthesized point "L" in the square room (worst "improvement": slight increase of error for both metrics): The ghosting artefact on the rug in the regular blending result was replaced by a displacement artefact in the flow-based blending (magenta), which also introduced a new artefact, namely the warped top edge of the coffee table (green). Otherwise the scenes are very similar.

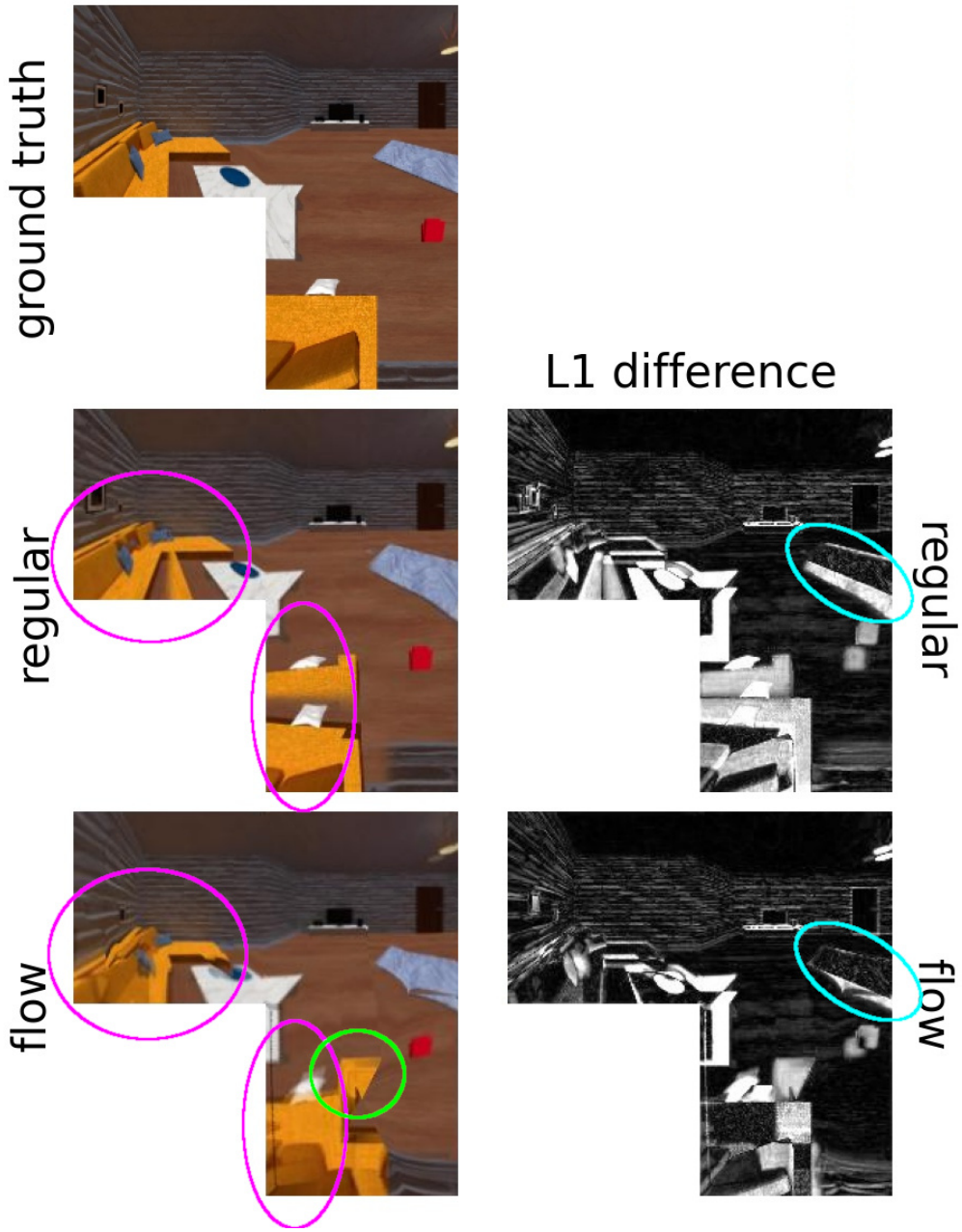


Figure A.7.: Synthesized point "A" in the oblong room (best improvement): The flow-based blending drastically improved ghosting artefacts on the couch (magenta) and the accuracy of the rug (cyan), but also introduced new artefacts (green).

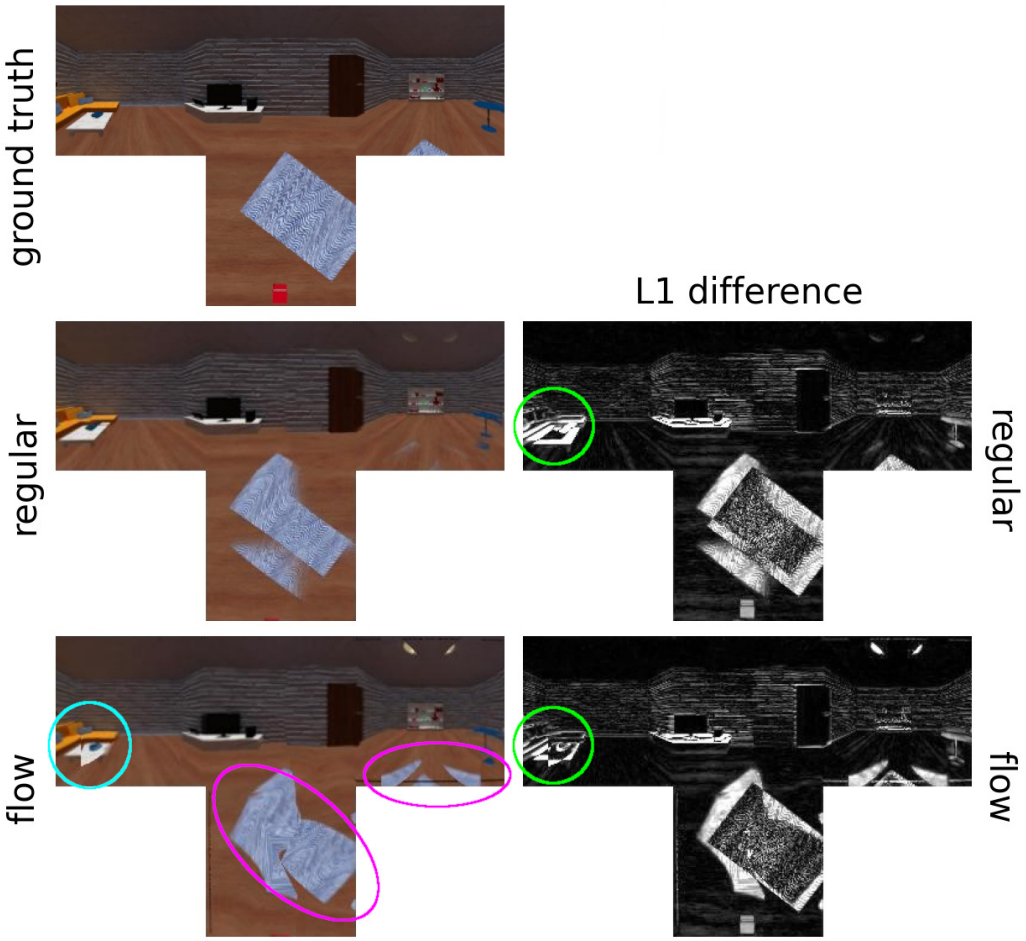


Figure A.8.: Synthesized point “L” in the oblong room (worst improvement): The flow-based blending result introduced some severe discontinuity artefacts on the rug (magenta) and on the coffee table (cyan), although the coffe table is positioned more accurately in the flow-based blending result (green)

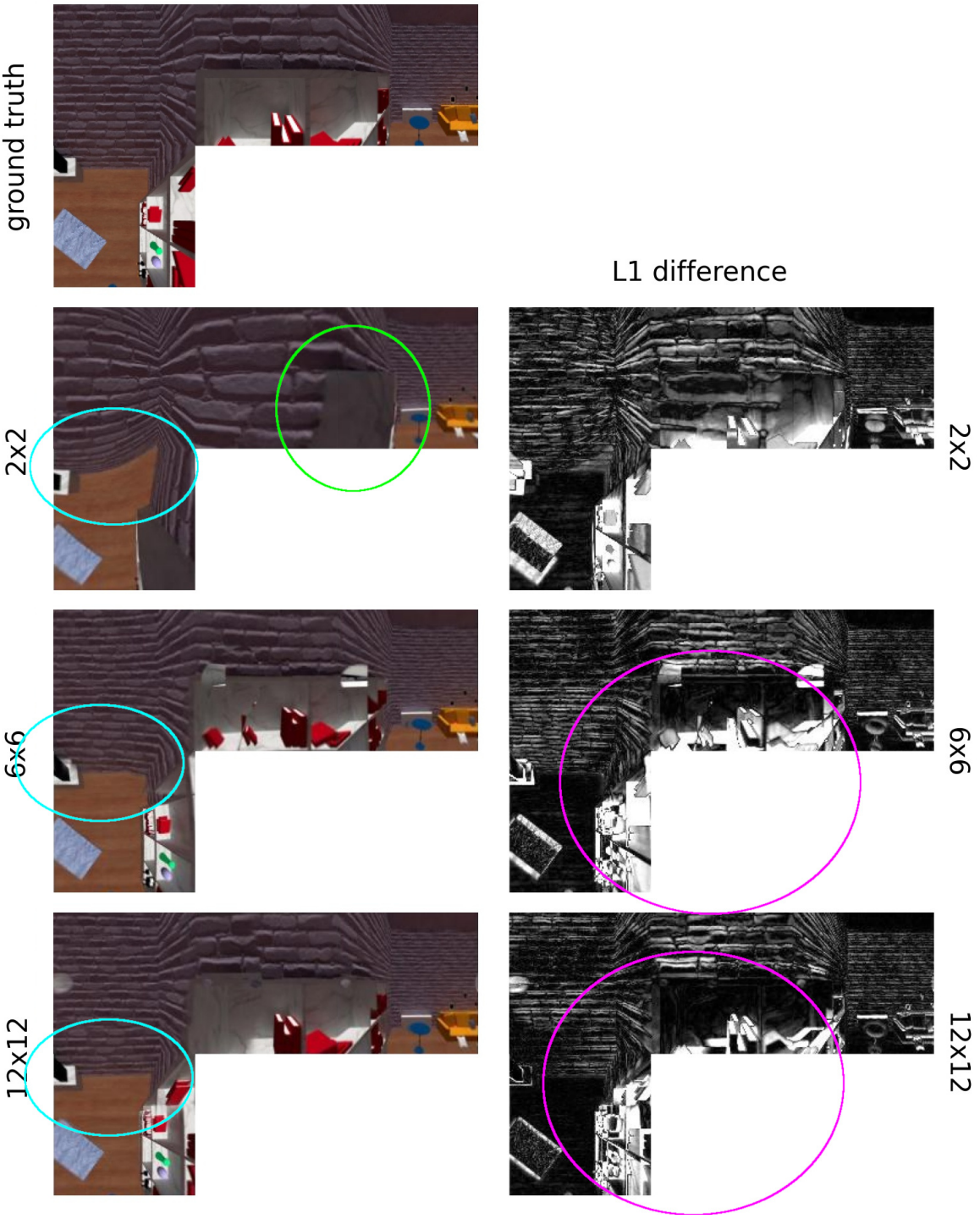


Figure A.9.: The regular blending results for point "T" (one of the worse results) in the square room with a viewpoint density of 2x2, 6x6, and 12x12: The 2x2 image shows the bookshelf from the wrong perspective (green), and the walls are noticeably warped (cyan), which is improved in the 6x6 and 12x12 images. The bookshelf is more accurate in the 12x12 image than the 6x6 image (magenta), but still shows some inaccuracies.

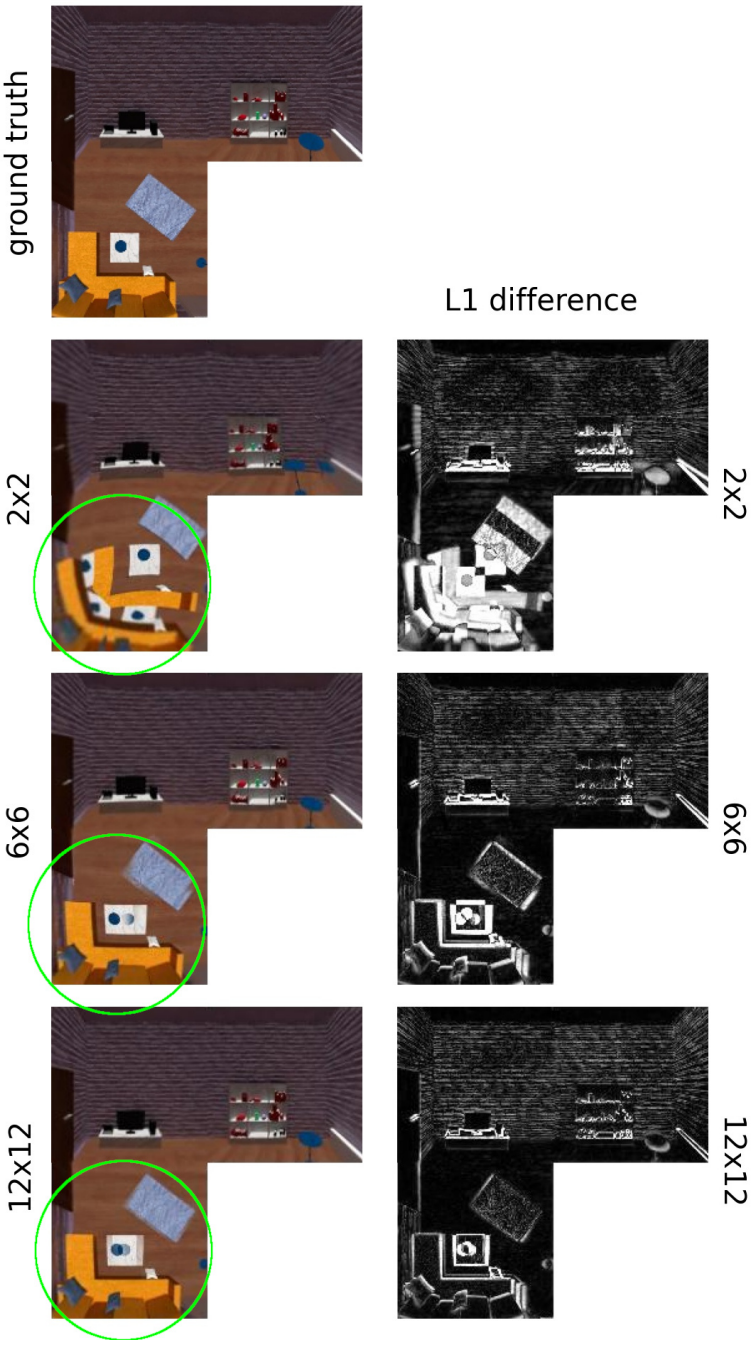


Figure A.10.: The regular blending results for point “G” (one of the better results) in the square room with a viewpoint density of 2x2, 6x6, and 12x12: Both the accuracy and the ghosting effects are visibly reduced, the higher the density of the captured viewpoints is. This is especially clear for the coffee table (green).

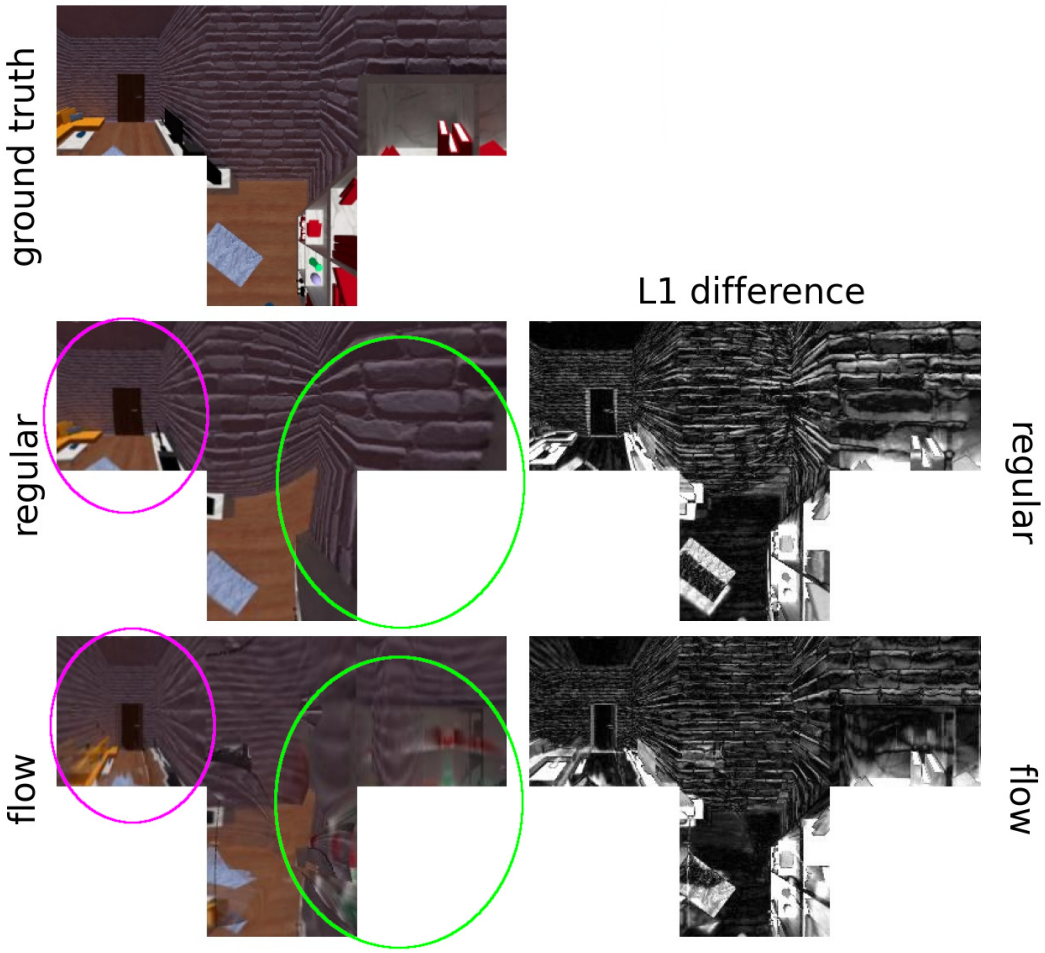


Figure A.11.: Synthesized point "T" in the 2x2 setup (best improvement for L1, good for SSIM): The flow-based blending synthesized a blurry approximation of the bookshelf (green), but also distorted and blurred the rest of the image (magenta), due to inaccurate optical flow from captured viewpoints with a too-large distance.

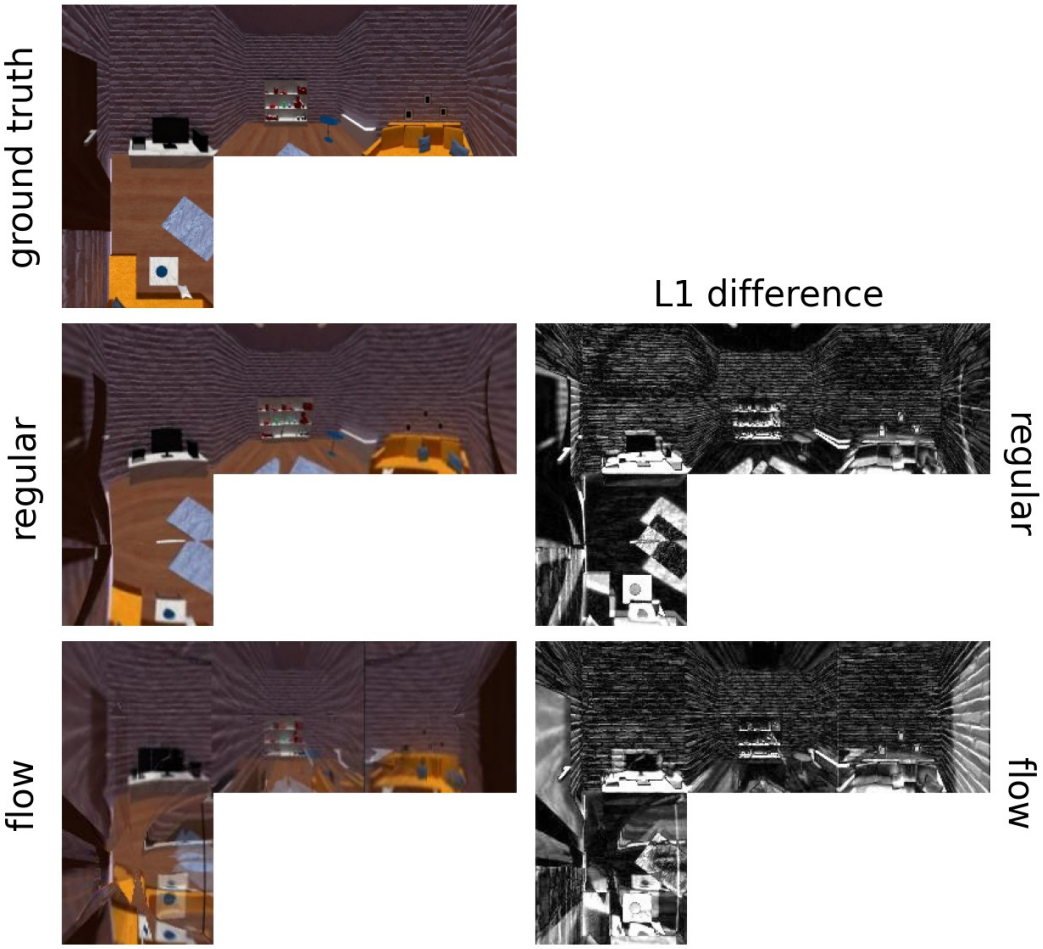


Figure A.12.: Synthesized point “K” in the 2x2 setup (worst “improvement”: slight increase of error): Although the regular blending result shows many doubling artefacts and incorrect positioning problems, the results of the flow-based blending are visually much worse due to inaccurate optical flow.

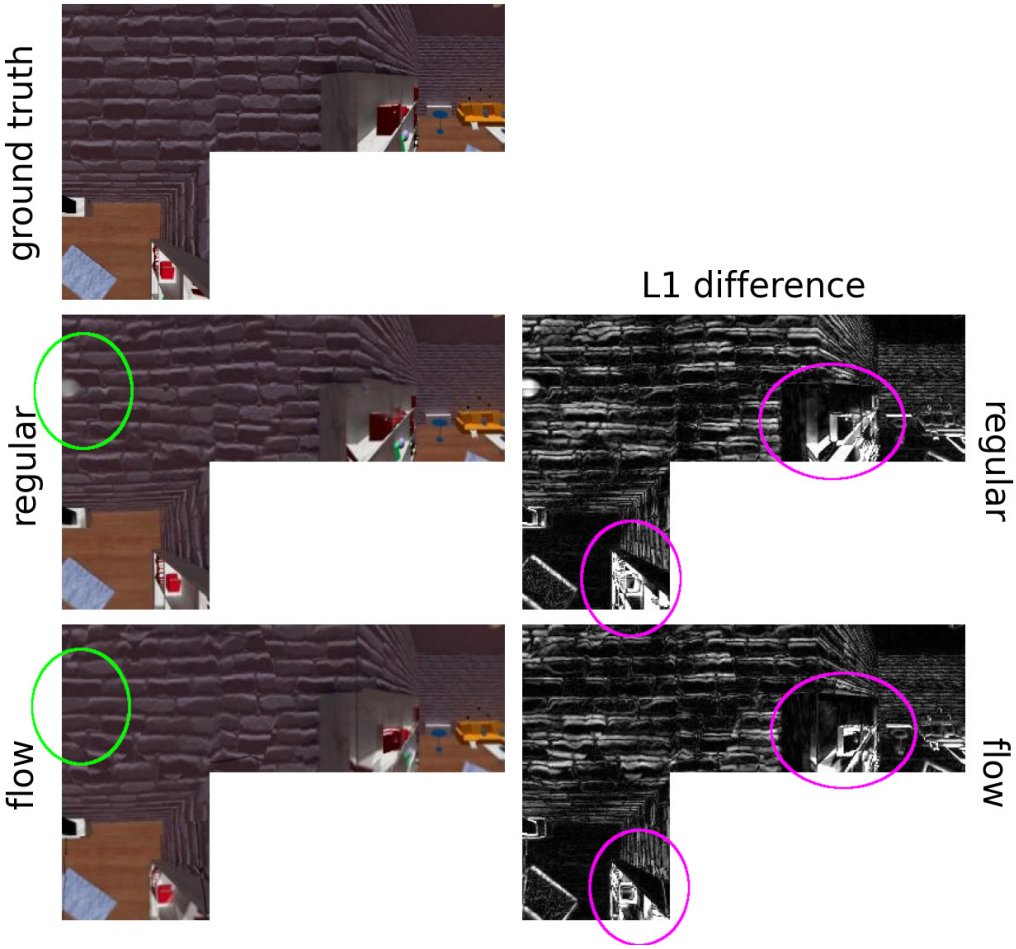


Figure A.13.: Synthesized point "Y" in the 12x12 setup (best improvement for L1 and SSIM): The flow-based blending result synthesizes the bookshelf slightly more accurately (magenta), and also does not display an artefact present in the regular blending result (green). Other than that, the results are very similar.

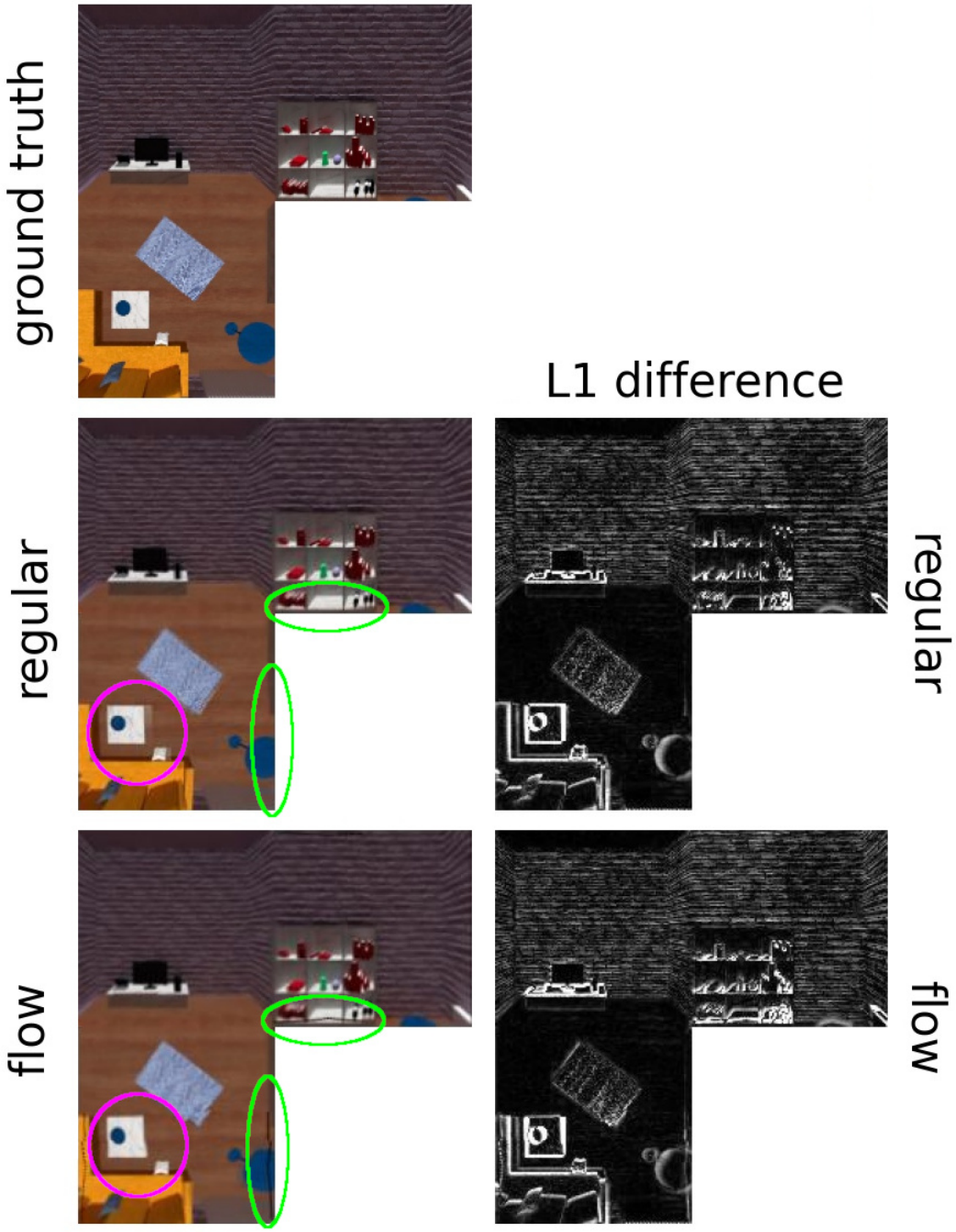


Figure A.14.: Synthesized point "H" (worst "improvement": slight increase of error): The flow-based blending result synthesizes the coffee table and white pillow with cleaner edges (magenta), however, due to a bug in an external library, the flow-based blending result contains some black lines (green) that possibly skew the error values in favor of the regular blending.

Figure A.15.: Synthesized point 7: The accuracy of the pictures on the wall is better in the flow-based blending result (magenta), and the shapes and position of the rug and the coffee table are also more accurate (green). However, the black line artefacts (cyan) are also fairly severe.

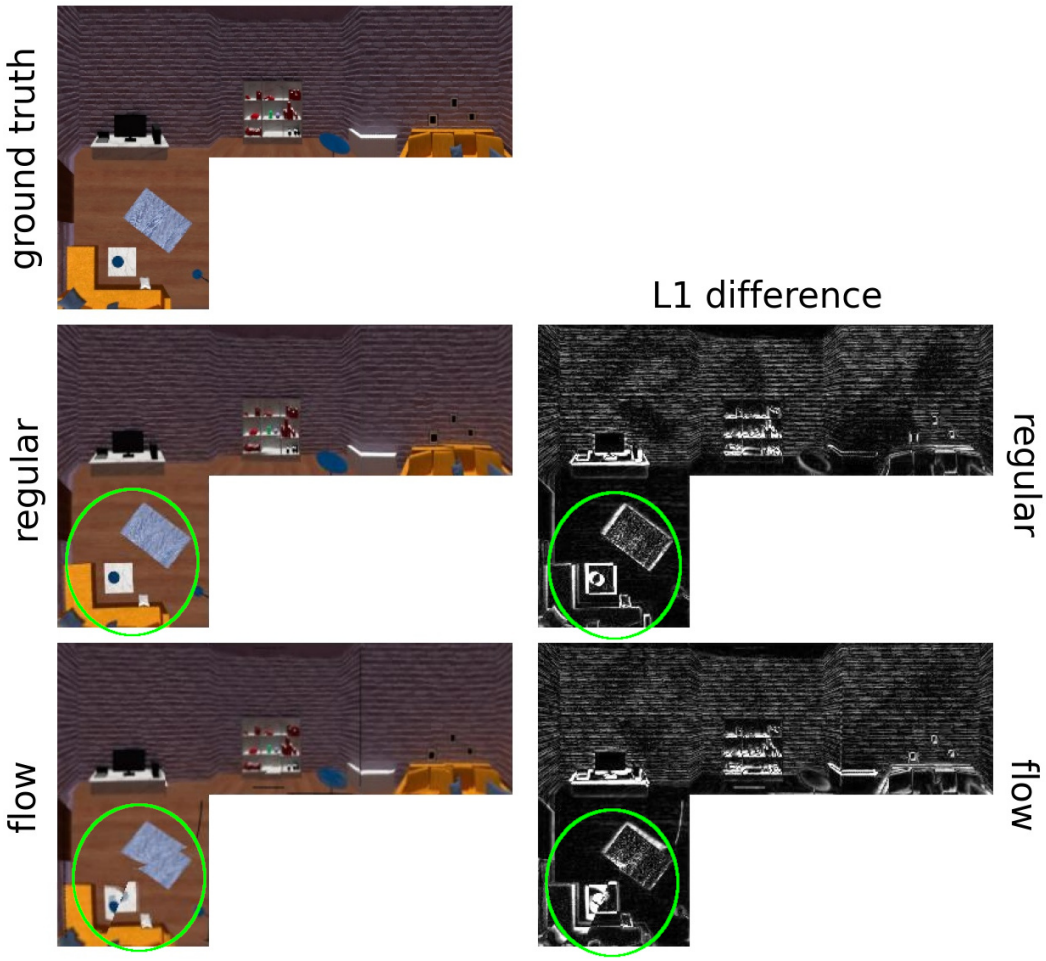


Figure A.16.: Synthesized point 283: Apart from a severe displacement artefact in the regular blending image, which also decreases the accuracy on the coffee table, the results are very similar.

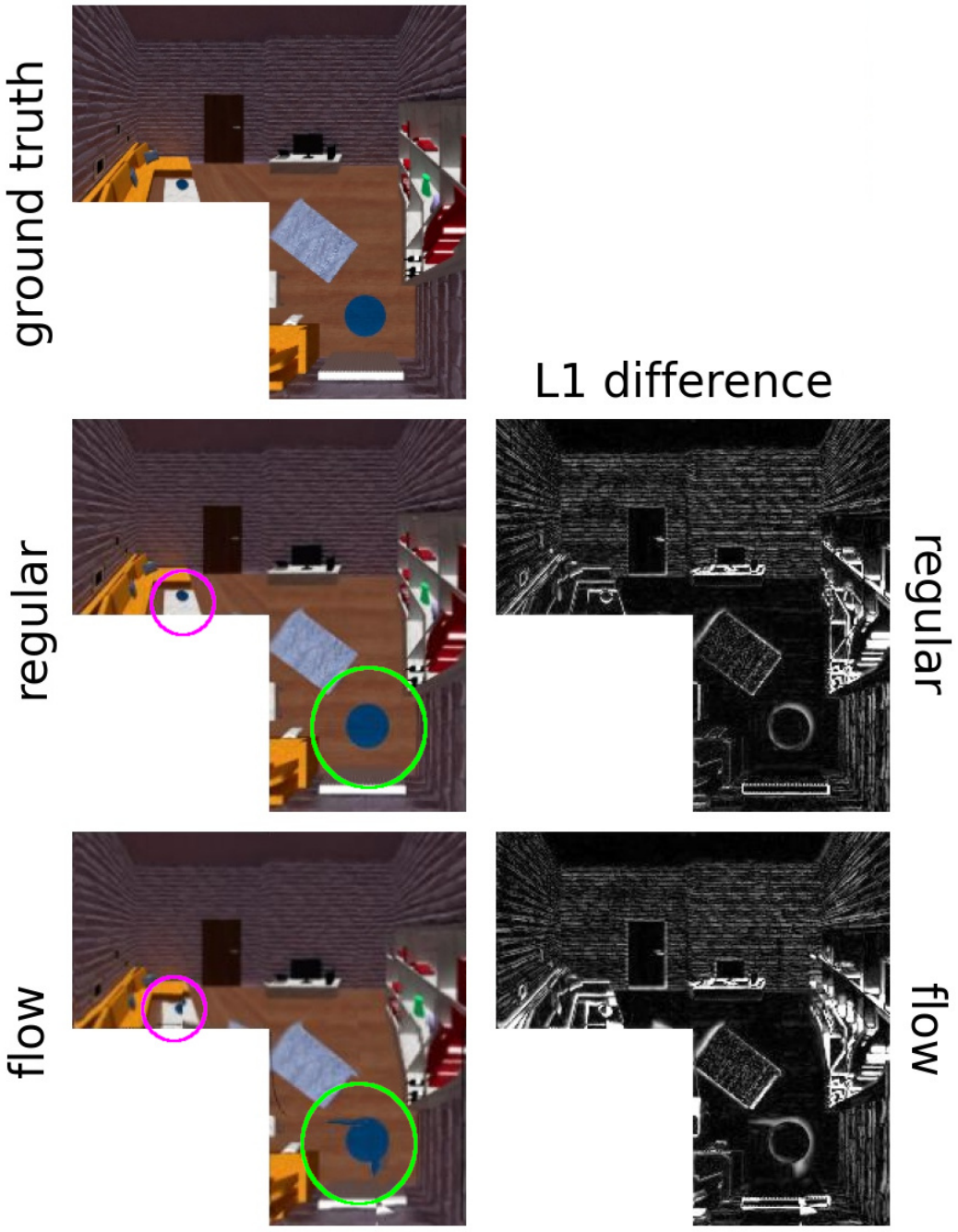


Figure A.17.: Synthesized point 145: The flow-based blending introduced a slight displacement on the coffee table (magenta) and a distortion on the blue table (green). Otherwise the results are very similar.

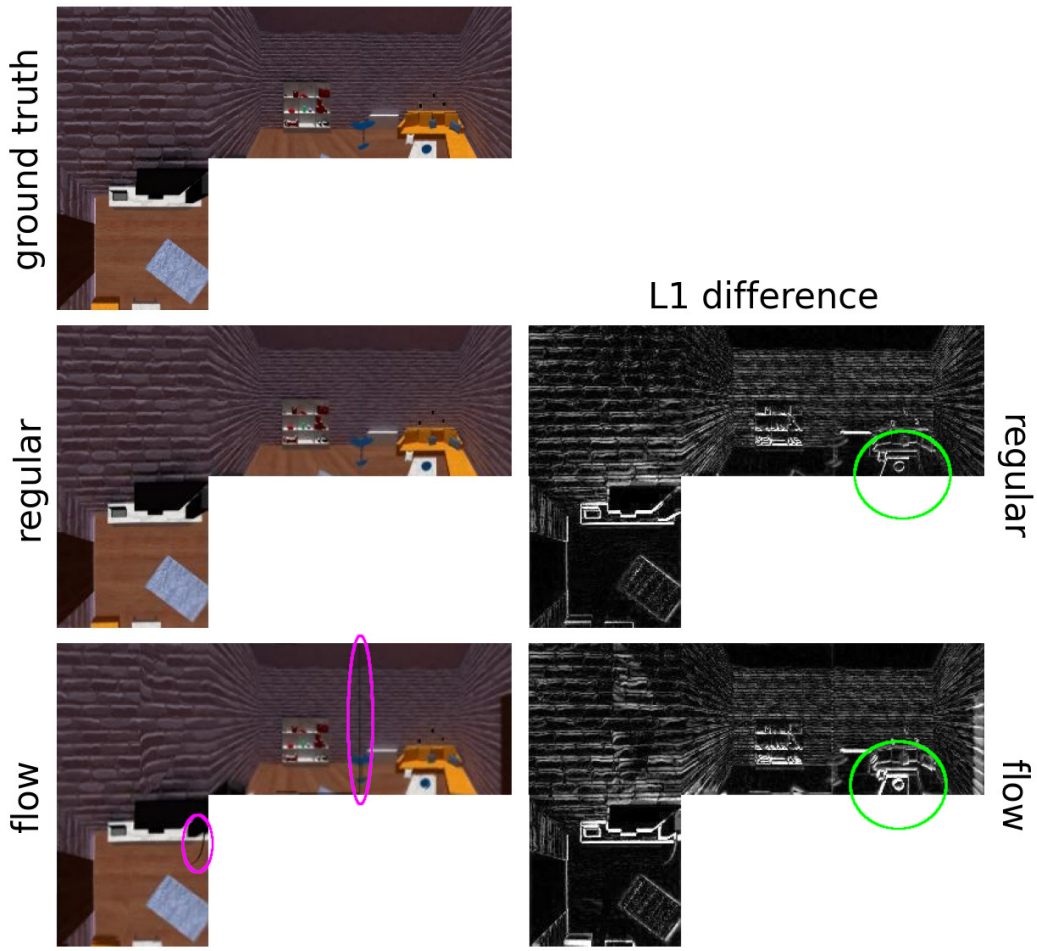


Figure A.18.: Synthesized point 504: The position of the coffee table in the flow-based blending result is less accurate than in the regular result (green). Otherwise the results are almost identical, except for the black lines caused by the external library bug (magenta).

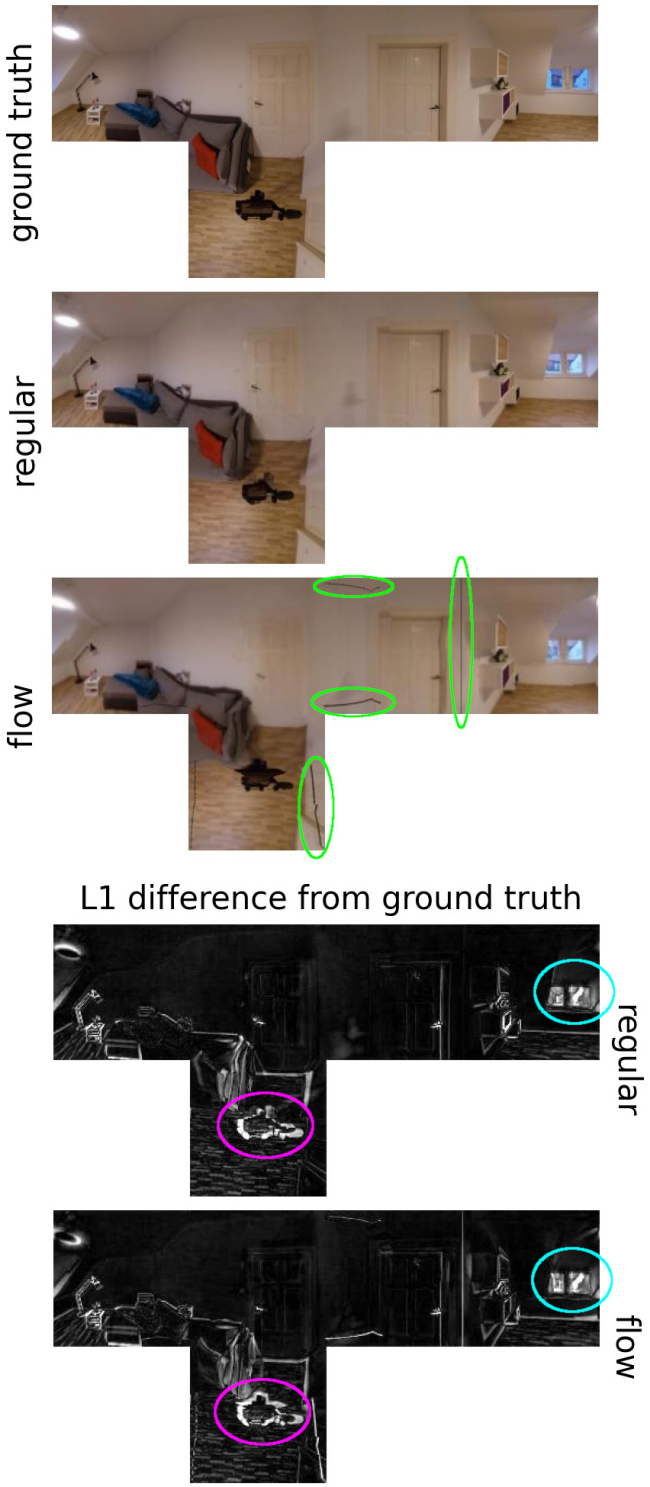


Figure A.19.: Viewpoint "I" (relatively low values for both regular and flow-based blending): Most of the scene is fairly accurate for both blending techniques. The largest positional inconsistencies are the tripod (magenta) and outside of the windows (cyan). The exernal library bug also causes very visible artefacts in the flow-based result (green).

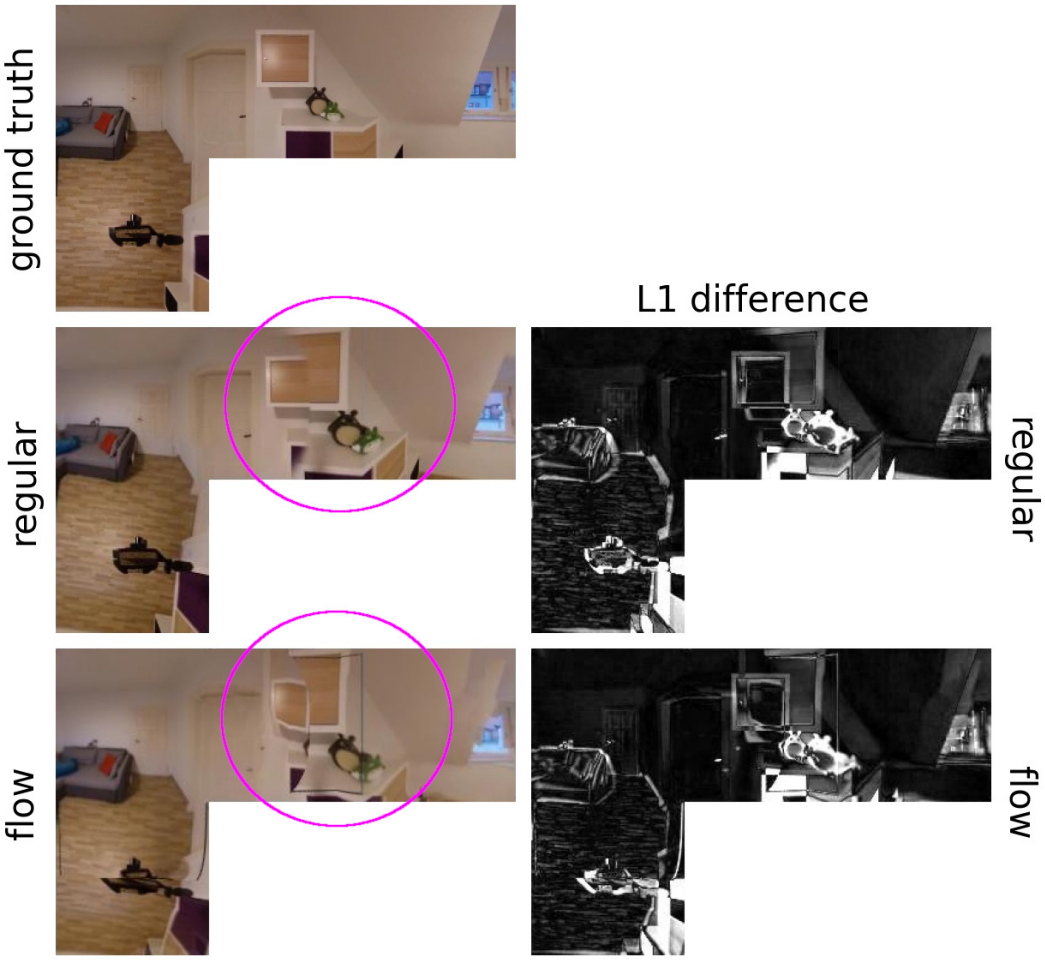


Figure A.20.: Viewpoint "D" (high values for both regular and flow-based blending): The high error values are due to the proximity to the cabinet, which was not reprojected correctly in the regular blending, and for which the optical flow algorithms also seems to have failed (magenta).

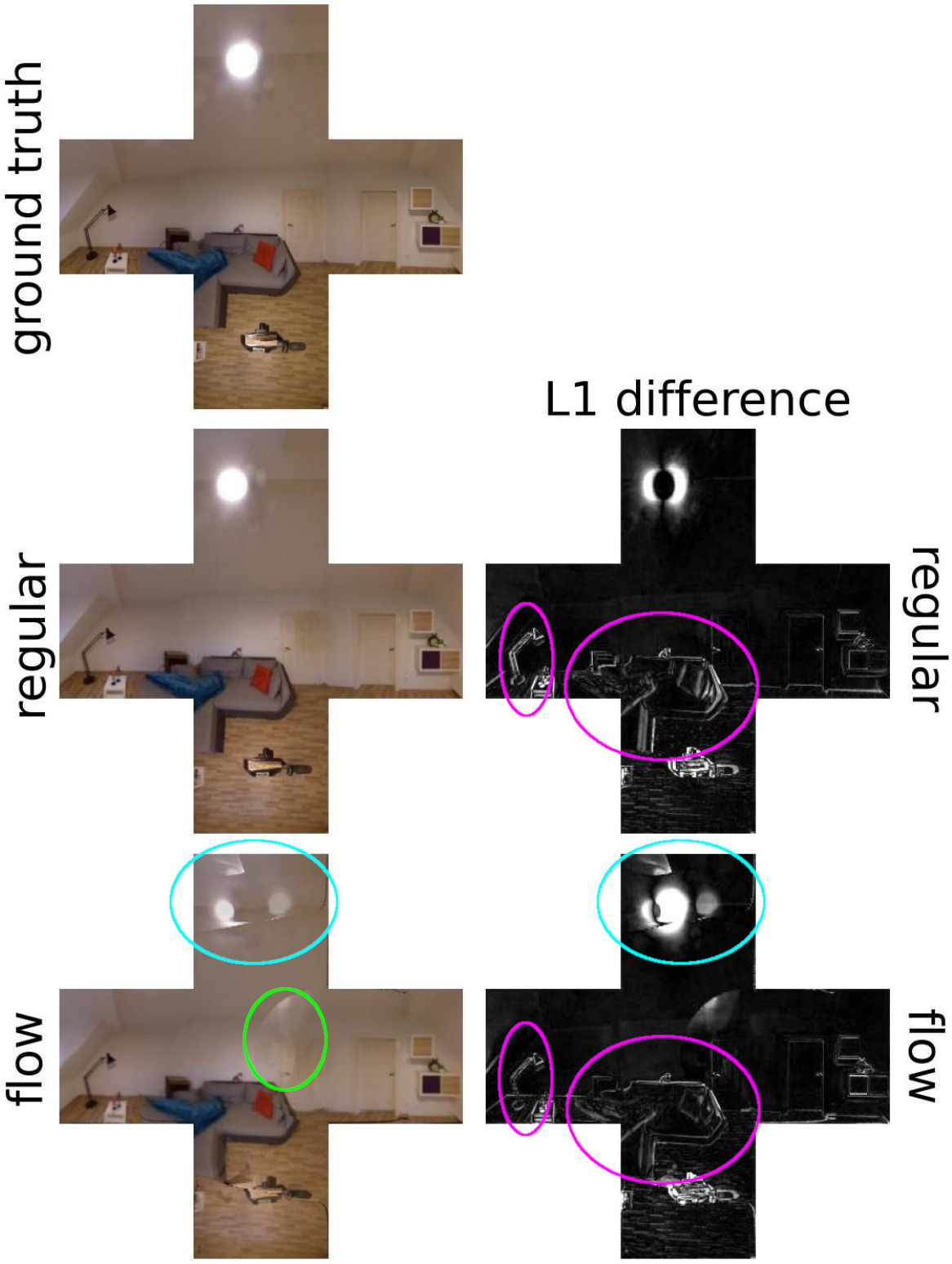


Figure A.21.: Viewpoint "G" (L1 much higher for flow-based blending, SSIM only slightly higher): Most elements of the scene are more accurate in the flow-based result (magenta), however, it does introduce some artefacts, including on the door and wall (green) and on the ceiling lamp, where the optical flow was inaccurate (cyan).

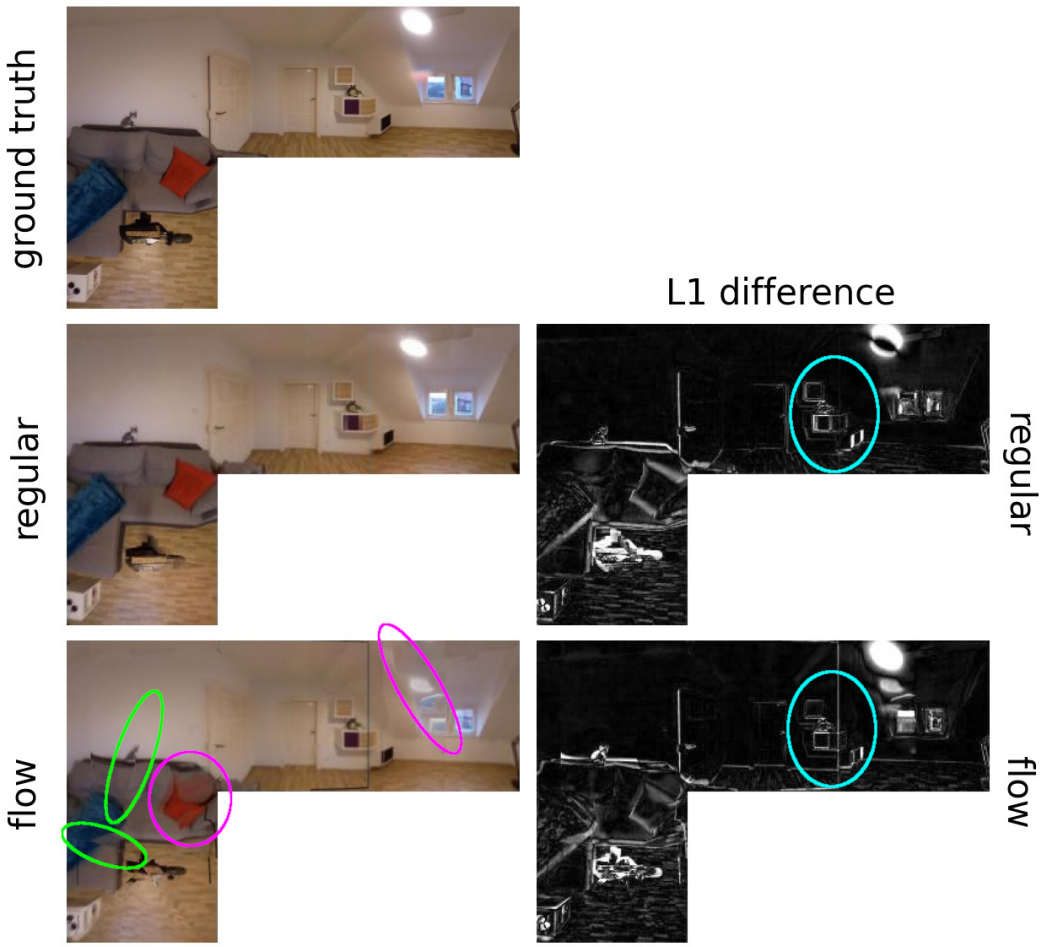


Figure A.22.: Viewpoint “A” (error values higher for flow-based blending): The flow-based blending result shows some ghosting artefacts due to failed optical flow (magenta), and discontinuities (green), but also has a more accurate positioning of the cabinets (cyan).

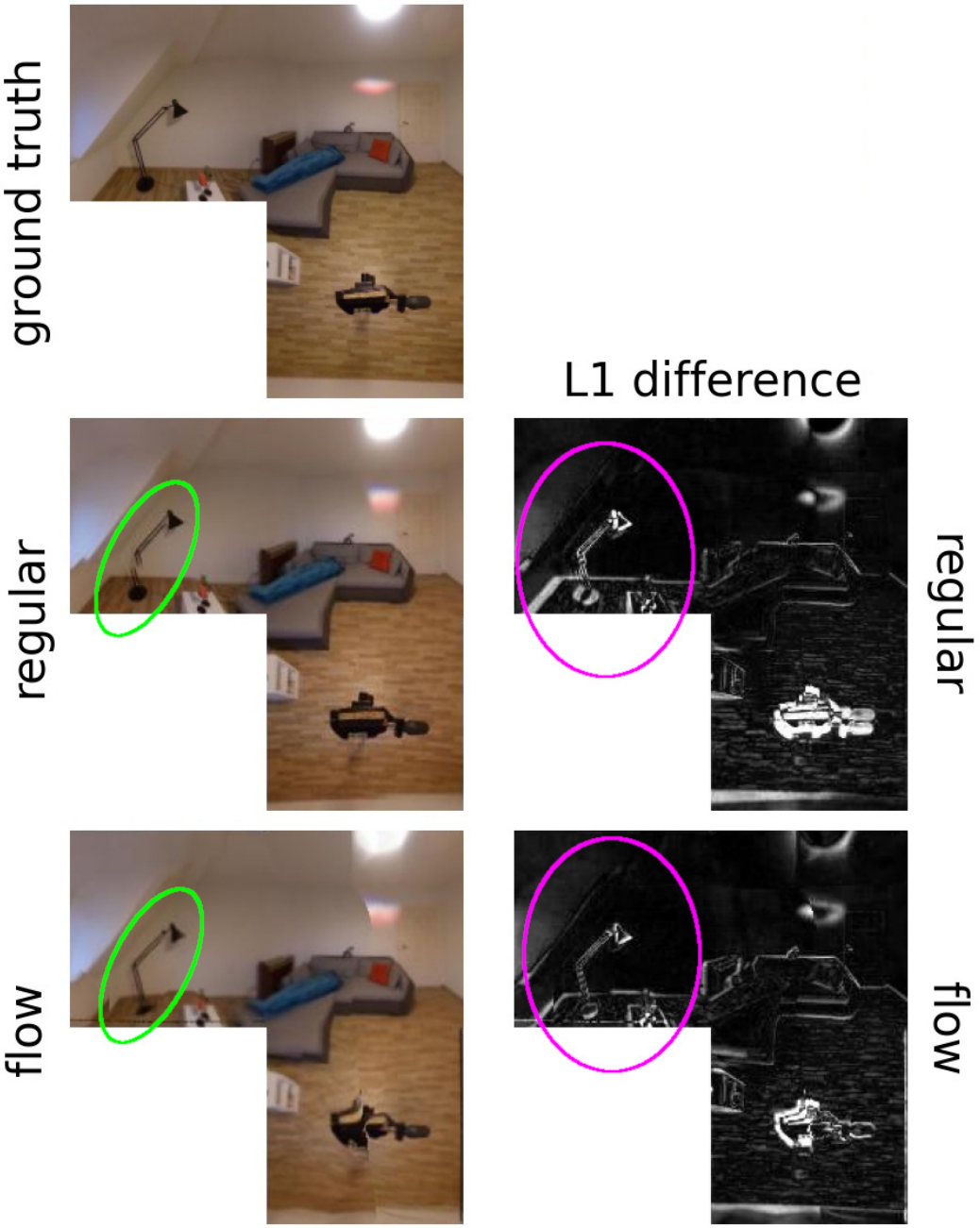


Figure A.23.: Viewpoint "K" (flow-based blending produced better results than the regular blending): The shape of the lamp is more accurate in the flow-based blending result (green), and the whole area around the lamp (the small table and the edge between the walls and floor is also in a more accurate position.

List of Figures

Bibliography

[AB91] Edward H. Adelson and James R. Bergen. The plenoptic function and the elements of early vision. In *Computational Models of Visual Processing*, pages 3–20. MIT Press, 1991.

[Ble20] Blender Online Community. Blender - a 3d modelling and rendering package, 2020.

[BWSB12] Daniel J. Butler, Jonas Wulff, Garrett B. Stanley, and Michael J. Black. A naturalistic open source movie for optical flow evaluation. In Andrew Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid, editors, *Computer Vision – ECCV 2012*, pages 611–625, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[Che95] Shenchang Eric Chen. QuickTime VR: An Image-Based Approach to Virtual Environment Navigation. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, page 29–38, New York, NY, USA, 1995. Association for Computing Machinery.

[CW93] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, page 279–288, New York, NY, USA, 1993. Association for Computing Machinery.

[Far03] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In Josef Bigun and Tomas Gustavsson, editors, *Image Analysis*, pages 363–370, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[FBK15] Denis Fortun, Patrick Bouthemy, and Charles Kervrann. Optical flow modeling and computation: A survey. *Computer Vision and Image Understanding*, 134:1 – 21, 2015. Image Understanding for Real-world Distributed Video Networks.

[HCCJ17] J. Huang, Z. Chen, D. Ceylan, and H. Jin. 6-DOF VR videos with a single 360-camera. In *2017 IEEE Virtual Reality (VR)*, pages 37–44. IEEE Computer Society, 2017.

[HDR+17] Sachini Herath, Vipula Dissanayake, Sanka Rasnayaka, Sachith Seneviratne, Rajith Vidanaarachchi, and Chandana Gamage. Unconstrained segue navigation for an immersive virtual reality experience. *Engineer: Journal of the Institution of Engineers, Sri Lanka*, 50:13, 10 2017.

[Hol20] Hold, Yannick (Soravux). Skylibs. `https://github.com/soravux/skylibs`, 2020.

[Kaw17] Naoki Kawai. A simple method for light field resampling. In *ACM SIGGRAPH 2017 Posters*, SIGGRAPH '17, New York, NY, USA, 2017. Association for Computing Machinery.

[KL10] S. Kolhatkar and R. Laganière. Real-time virtual viewpoint generation on the gpu for scene navigation. In *2010 Canadian Conference on Computer and Robot Vision*, pages 55–62, 2010.

[LH96] Marc Levoy and Pat Hanrahan. Light field rendering. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, page 31–42, New York, NY, USA, 1996. Association for Computing Machinery.

[Map] Mapillary. OpenSfM. `https://www.opensfm.org/docs/`.

[NJ18] Z. Nian and C. Jung. High-quality virtual view synthesis for light field cameras using multi-loss convolutional neural networks. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 2605–2609. IEEE Computer Society, 2018.

[Pyt18] Python Software Foundation. Python 3.7.9 documentation. `https://docs.python.org/3.7/`, 2018.

[RPZSH13] Christian Richardt, Yael Pritch, Henning Zimmer, and Alexander Sorkine-Hornung. Megastereo: Constructing high-resolution stereo panoramas. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1256–1263. IEEE Computer Society, June 2013.

[RWPD05] Erik Reinhard, Greg Ward, Sumanta Pattanaik, and Paul Debevec. *High Dynamic Range Imaging: Acquisition, Display, and Image-Based Lighting (The Morgan Kaufmann Series in Computer Graphics)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[SET20] Stefano Savian, Mehdi Elahi, and Tammam Tillo. *Optical Flow Estimation with Deep Learning, a Survey on Recent Advances*, chapter 12, pages 257–287. Springer International Publishing, 01 2020.

[SI14] Davide Scaramuzza and Katsushi Ikeuchi. Omnidirectional camera. *Computer Vision: A Reference Guide*, 2014.

[SK00] Harry Shum and Sing Bing Kang. Review of image-based rendering techniques. In King N. Ngan, Thomas Sikora, and Ming-Ting Sun, editors, *Visual Communications and Image Processing 2000*, volume 4067, pages 2 – 13. International Society for Optics and Photonics, SPIE, 2000.

[SLDL09] F. Shi, R. Laganiere, E. Dubois, and F. Labrosse. On the use of ray-tracing for viewpoint interpolation in panoramic imagery. In *2009 Canadian Conference on Computer and Robot Vision*, pages 200–207. IEEE Computer Society, 2009.

[SLL19] YiChang Shih, Wei-Sheng Lai, and Chia-Kai Liang. Distortion-free wide-angle portraits on camera phones. *ACM Trans. Graph.*, 38(4), July 2019.

[The19a] The OpenCV team. OpenCV 4.2 documentation. `https://docs.opencv.org/4.2.0/`, 2019.

[The19b] The SciPy community. NumPy v1.16 Manual. `https://numpy.org/doc/1.16/`, 2019.

[The20] The SciPy community. SciPy v1.5.2 Reference Guide. `https://docs.scipy.org/doc/scipy-1.5.2/reference/`, 2020.

[vdWSN+14] Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu, and the scikit-image contributors. scikit-image: image processing in Python. *PeerJ*, 2:e453, 6 2014.

[Wei] Weisstein, Eric W. Line-Line Intersection. `https://mathworld.wolfram.com/Line-LineIntersection.html`.

[ZBSS04] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[ZC04] Cha Zhang and Tsuhan Chen. A survey on image-based rendering—representation, sampling and compression. *Signal Processing: Image Communication*, 19(1):1 – 28, 2004.

[ZWF+13] Q. Zhao, L. Wan, W. Feng, J. Zhang, and T. Wong. Cube2video: Navigate between cubic panoramas in real-time. *IEEE Transactions on Multimedia*, 15(8):1745–1754, 2013.