FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Zeitliches Upsampling von Bild-Sequenzen mit einem Nicht-Lokalen Durchschnitts Algoritmus

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Visual Computing

eingereicht von

BSc. Clemens Rögner
Matrikelnummer 0825045

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Mitwirkung: Dr. Johannes Hanika, Karlsruhe Institute of Technology (KIT)

Wien, 9. Oktober 2014

Clemens Rögner	Michael Wimmer

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Temporal Upsampling for Image Sequences Using a Non-Local Means Algorithm

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Visual Computing

by

BSc. Clemens Rögner
Registration Number 0825045

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Assistance: Dr. Johannes Hanika, Karlsruhe Institute of Technology (KIT)

Vienna, 9th October, 2014

Clemens Rögner | Michael Wimmer

Erklärung zur Verfassung der Arbeit

BSc. Clemens Rögner
Unterzwischenbrunn 21, A-3100 St. Pölten

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 9. Oktober 2014

Clemens Rögner

Danksagung

Diese Diplomarbeit wurde in Kollaboration mit der Computergrafik Gruppe des Karlsruher Instituts für Technologie (KIT) geschrieben. Der Algorithmus, welcher in dieser These vorgestellt wird, basiert auf einer Idee von Dr. Johannes Hanika, meinem Betreuer in Karlsruhe. Ich möchte mich hiermit herzlich bei ihm für seine uneingeschränkte Unterstützung bedanken und auch all den anderen Mitarbeiter, die es mir ermöglicht haben, diese Arbeit zu schreiben.

Weiters möchte ich mich auch noch bei Prof. Michael Wimmer bedanken, der den Kontakt zum KIT hergestellt hat und mir beim Abschluss der Arbeit geholfen hat.

Zu guter Letzt will ich hiermit dem Land Niederösterreich für das Stipendium danken.

Acknowledgements

This thesis is written in collaboration with the Computer Graphics Group of the Karlsruhe Institute of Technology. The method presented in this thesis is the result of an idea from my supervisor Johannes Hanika. I want to thank him for supporting me anytime his expertise was needed, as well as the entire Computer Graphics Group for making this thesis possible.

In addition to that, I want to thank Micheal Wimmer, who initiated the contact to the people at the Karlsruhe Institute of Technology and for helping me in the later stages of the thesis.

Furthermore, I want to thank the province of Lower Austria for the scholarship.

Kurzfassung

Computer-generierte Video Sequenzen mit einer höheren Frame-Rate (Bildrate) als 24 Bilder pro Sekunde, wie zum Beispiel 48 oder 60 Bilder pro Sekunde, werden, aufgrund ihrer höheren visuellen Qualität, in den entsprechenden Gebieten immer populärer. Dies hat aber zum Nachteil, dass mehr Zeit für das Berechnen benötigt wird.

Eine Lösung für dieses Problem ist das sogenannte Frame-Rate Upsampling, welches sich zeitlicher und räumlicher Kohärenz bedient um neue Frames (Bilder) zu approximieren, was daher die Dauer der Berechnung verringert. Dazu gibt es eine Vielzahl an Publikationen, welche dem Zweck der Echtzeitgrafik sowohl aber auch dem so genannten offline Rendering dienen.

In dieser These werden zwei neue Algorithmen für das Frame-Rate Upsampling vorgestellt. Beide zielen auf hoch-qualitative Computer generierte Bilder mit verschiedenen Global-Illumination Effekte ab. Diese zwei neuen Algorithmen verwenden eine denoising Methode für Videos – den non-local means Algorithmus– um die Farben für die Pixel in dem Frame zu finden, welcher approximiert werden soll. Um genau jene Farben zu finden, verwenden die vorgestellten Methoden entweder vorhandene Farbinformationen oder bedienen sich zusätzlicher Daten für jeden Pixel, welche mit minimalen weiteren Berechnungen aus jedem Global-Illumination Algorithmus extrahiert werden können. Die vorgestellten Methoden sollen so, besser als bisherige Publikationen, mit Reflexionen und transparenten Objekten umgehen.

Abstract

Computer-generated video sequences with a frame-rate higher than the usual 24 images per second, such as 48 or 60 frames per second, have become more popular in the respective industries, due to more visual fidelity. This, however, results in more computational costs for the same length of the video sequence.

One solution to this problem is the so-called frame-rate upsampling, which makes use of temporal and spatial coherence to approximate new frames and therefore saves computational time. Several methods have been published in this field, for the purposes of real-time rendering as well as for offline rendering algorithms.

In this thesis, two new algorithms for fame-rate upsampling are introduced. Those are targeted at high-quality computer-generated images that feature various global-illumination effects. The two new algorithms make use of a video denoising method –the non-local means algorithm– to find the appropriate pixel colors for the frame, that has to be upsampled. To find the corresponding pixels in another frame, the methods of this thesis either use existing color information or require additional data, which can be extracted from any global-illumination algorithm with minimal further computations. The proposed methods are aimed at handling reflections and refractions in the scene better than previous work.

Contents

Kurzfassung **xi**

Abstract **xiii**

Contents **xv**

List of Figures **xvi**

List of Tables **xx**

List of Algorithms **xxi**

1 Introduction **1**
1.1 Problem Description 1
1.2 Temporal Coherence 2
1.3 Challenges 3
1.4 Approach 4
1.5 Quality 5
1.6 Thesis overview 5

2 Background **7**
2.1 Light and Color 7
2.2 Global-Illumination 7
2.3 Optical Flow 11
2.4 Upsampling 12
2.5 Image Denoising 14

3 Previous Work **17**
3.1 Coherence in Global-Illumination Algorithms 17
3.2 Upsampling Techniques in Real-time Rendering 18
3.3 Upsampling using Video Coding 21
3.4 Non-local means denoising 22

4 Motivation and Overview **29**

4.1 Motivation . . . 29
4.2 General Approach . . . 30
4.3 Adapted Non-Local Means algorithm . . . 30

5 Motion-Vector Technique **35**
5.1 Algorithm . . . 35
5.2 Similarity measure . . . 36

6 Pixel-Similarity Technique **39**
6.1 Algorithm . . . 39
6.2 Similarity measure . . . 39

7 Results and Comparison **47**
7.1 Test Image Sequence . . . 47
7.2 Parameter Finding . . . 48
7.3 Comparison and Evaluation . . . 55
7.4 Summary . . . 66

8 Conclusion **67**
8.1 Future Work . . . 68

Bibliography **69**

List of Figures

2.1 Schematic of the rendering equation as it is described in Formula 2.1 . . . 8
2.2 Reflection of a light ray. . . . 9
2.3 Problem of moving reflective objects. Figure (a) shows the initial positions and the path that results into the green pixel. The blue arrow is the movement of the specular surface in Figure (b). The resulting pixel color should be brown as well. However, when the motion of the object is followed (hence sampling the same point on the specular surface as before) this could into a (false) green pixel. This green color is due a hit on the top of the cube in the initial position. . . . 10
2.4 Refraction of a light ray. . . . 11
2.5 The orange and the green line show a possible light path through a transparent object (blue rectangle). Due to internal reflection the path may not exist after the second surface-interaction. . . . 11

2.6 Optical flow of a circle moving from left to right. The according motion vector is colored blue. 12
2.7 Curve Fitting example. Sub-figure (a) depicts the original signal with its sampling points at t_i, Sub-figure (b) shows a polynomial fitting, Sub-figure (c) shows a natural cubic spline fitting with the blue dots marking the influence on $z(i)$ and Sub-figure (d) shows a Hermit cubic spline fitting with its derivatives $\dot{s}(i)$ (blue arrows) 13
2.8 Schematic of a noisy signal. Left: A possible, not altered, signal. The goal of denoising methods is to extract this signal. Middle: A (random) noise signal. Right: the combination of both signals. This is the problematic signal that has to be resolved via denoising methods. 15

3.1 Difference in the data usage of two methods proposed by Yang et al. [YTS+11]. The scene-assisted approach requires a rendering of the desired frame, whereas the image-based approach does not. In each case the purple cube represents the unknown shading information, the orange one represents the shading from the previous frame and the blue depicts the shading information of the upcoming frame 19
3.2 Three iterations of the algorithm by Yang et al. [YTS+11] which is used to estimate the movement of a pixel from one frame to another 20
3.3 Schematic of the non-local means algorithm for denoising introduced by Buades et al. [BCM05]. 22
3.4 Comparison of the non-local means denoising algorithm for a single image against other filtering methods. The number below the thumbnail images depict the mean squared error of the entire denoised image towards the original image. Pictures taken from Buades et al. [BCM05]. 23

4.1 Difference between using the original non-local means algorithm and the version with the list of best fits. 31
4.2 Size notation of the neighborhood K and similarity area S. The red square marks the center pixel. 34

5.1 The motion-vector technique in pictures. 37

6.1 The pixel-similarity technique in pictures. 40
6.2 Different types of light propagation on a surface point. 41
6.3 Validation plots of the similarity measure for diffuse surfaces. 43
6.4 Validation plots of the similarity measure for specular surfaces done for frame 000610. Figure a uses a measure using direction of the reflection and Figure b utilizes the point from where the most contributing light comes from. . . . 44
6.5 Validation plots of the similarity measure for transparent surfaces done for frame 002351. The figures on the left show the similarity values for two points on the glass cube. On the right side is the result of using that measure. . . . 46

7.1 This figure shows the test sequence along the time-line with an image example for each scene. 47

7.2 Values of the biggest dimension in motion vectors for each pair of frames across the entire test image sequence. 49

7.3 Values of the biggest dimension in motion vectors for each pair of frames across the entire test image sequence. 50

7.4 Polynomial fit for the function $f(x) = cos(2x) * sin(x/3)$. Figure (a) shows a fitting with with a degree 5 polynomial resulting in under-fitting, Figure (b) uses a 9-degree polynomial resulting in the best fit and (c) shows over-fitting with a polynomial degree of 11 50

7.5 This figure shows the consequences of too small similarity-area sizes for the motion-vector technique. Figure (a) was produced with an area size of 1 and shows noise at the green stripes due to the failed matching. The noise around the edges of the glass cube is also a result of the small area size. In Image (b) the area size is 5, which results in no noise at the green lines as well as reduced noise at the edges of the glass. Figure (c) shows the ground-truth frame, Figure (d) shows the motion vectors (pointing from white to red) due to the small area size of 1, and Figure (e) shows better motion vectors generated with a similarity size of 5. Notice that in Figure (d) motion is detected on the bumpy cube, when there is actually none. And finally, Figure (f) marks the place of the cutout in the frame. 51

7.6 This figure shows the consequences of too big similarity-area sizes for the motion-vector technique. Subfigure (a) was produced with an area size of 5 and shows 'color bleeding' to some extend due to the inability of the algorithm to find a similar pixel. In Image (b) this effect is amplified due to an increased similarity size of 10. Figure (c) shows the ground truth frame, Figure (d) shows the motion vectors (pointing from white to red) for the similarity area of 5 and Figure e) shows false motion vectors due to the big area size of 10. Notice that the vectors in Figure (e) do not cross the edge, which stands in contrast to the objects movement downwards to the left. And finally, Figure (f) marks the place of the cutout in the frame. 52

7.7 Tests for the size of the similarity area to use in the testing sequence. The blue line depicts the mean squared errors without a Gaussian kernel and the orange line shows a weighting of the area with the kernel. 53

7.8 Tests for the size of the similarity area of the pixel-similarity technique to use in the testing sequence. The blue line depicts the mean squared errors without a Gaussian kernel and the orange line shows a weighting of the area with the kernel. 53

7.9 Results for different maximal list sizes when its values are averaged to make up the resulting motion vector. The number below the pictures are the mean squared errors of those images. 54

7.10 Results for different maximal list sizes for frame 2351. The number below the pictures are the mean squared errors of those images. Notice the increased blurring due to the greater list size. 54

7.11 Figure (a) shows the mean squared error to the ground-truth frames of the motion-vector technique for each frame in the test sequence. The blue line is the variant with the iterative search and the orange on is the brute force variant Figure (b) shows the rendering times of both variants using an adaptive neighborhood size with a minimum of 10 56

7.12 Case of iterative search variant (mean squared error: 0.00391085) of the motion-vector technique outperforming the brute force variant (mean squared error: 0.00464652). 56

7.13 Figure (a) shows the mean squared error to the ground-truth frames of the pixel-similarity technique for each frame in the test sequence. The blue line is the variant with similarity measure using the position of the surface and the orange one uses that of the next surface-interaction of the refracted light Figure (b) shows the rendering times with an adaptive neighborhood size including a minimum of 10. 57

7.14 Mean squared error for the entire test sequence for the motion-vector technique (brute-force variant) as the blue line, the pixel-similarity technique (refraction interaction variant) as the orange line and the scene-assisten method proposed by Yang et al. [YTS+11] as the black line. 57

7.15 Problems of the motion-vector technique when detecting edges. The images in the upper row show the results and those in the bottom show the difference to the ground-truth frames. The 'border-effect' in the latter images highlight the problems of the motion-vector technique with correct edge location. . . . 59

7.16 Small non-linear motion (camera movement in this case) leads to false edge location. In this image an offset to the ground-truth frame of one pixel can be seen. The vertical blue line marks the position of where the bumps should be. 60

7.17 Edges are preserved with the pixel-similarity technique. The same is true for high-frequency textures. Slight blurring occurs due to usage of maximal list size greater than one. 61

7.18 Edges are preserved with the scene-assisted method by Yang et al. [YTS+11]. The same is true for high-frequency textures. Slight inaccuracies due to oversampling. 62

7.19 Ghost edges due to upsampling with video encoding. 62

7.20 Noise resulting into false motion vectors. In the case of the red and blue wall those do not matter, but on the edges of those two walls it does, since it results into a not discontinuous edge-line. 62

7.21 Specular surface with the motion-vector technique. Figures (a) to (i) show a cutout of an image sequence generated by the motion-vector technique. It shows, that small color changes do not result into problems, but as soon as they are significant (Figure (f) and (h)), the algorithm produces artifacts known as 'color bleeding'. In addition to that, the iterative search variant, produces a 'tearing effect' as seen in Figure (j). 63
7.22 Specular surface with the pixel-similarity technique. Figures (a) to (j) show a cutout of an image sequence generated with that technique. Some noise is visible at the highlights as well as small errors around the edges. The latter are a result of oversampling. 63
7.23 Specular highlight in frame 611 of the test sequence with various upsampling techniques. 64
7.24 Example of the motion-vector technique handling transparent objects. As can be seen from the figures above, edges lead to noise whilst the surface is free from it. 64
7.25 Example of the pixel-similarity technique handling transparent objects. This techniques leads to blurring and false edges at the back-side of the glass cube. 65
7.26 Glass cube in frame 2351 of the test sequence with various upsampling techniques. 65

List of Tables

3.1 Comparison of the optimized non-local means denoising method by Goossens et al. [GLPP08] on a GPU against a CPU version of the algorithm. The neighborhood size is given in width x height x temporal-window of previous frames 27

List of Algorithms

3.1 Non-local means algorithm for video denoising 24
3.2 Non-local means algorithm for video denoising optimized version 26

4.1 Maximal similarity detection 32
4.2 Injecting values into a list, that keeps those with the highest weight w . . . 33

CHAPTER 1

Introduction

1.1 Problem Description

Computer-generated video sequences are used in many areas for different purposes. Usually, those sequences consist of images (also called frames) displayed with 30, 25 or 24 frames per second. Generating one image on the computer (so-called rendering) requires extensive computations when the entire light transport in the scene has to be resolved. Recent developments head towards using more frames per second (either 48 or 60) to improve the visual experience [AFH+]. This increases the time to compute the video sequences even more, since more frames are required to produce. More details on those issues is given in the following sections.

1.1.1 CGI and Global Illumination

Computer-Generated Imagery (CGI) content is an important part in the movie industry. It enables the creation of scenes that otherwise would not be possible to put on screen, as well as reduces production costs compared to various non-CGI methods [KFC+10].

Traditionally, the movie industry used software that featured only direct illumination with several improvements to enable life-like effects. In recent years, the CGI standard shifted towards using rendering software that computes global illumination (GI). Before that, such rendering software was considered to restrict artists, but recent movie productions have shown that this is not true since the creative process can be altered to achieve the same goals [KFC+10].

The down-side of using a global-illumination renderer is that generating one image requires extensive computations since the light transport in the scene has to be calculated. This topic is discussed in detail in section 2.2.

1.1.2 Higher Frame Rate

When displaying a movie in a cinema, the frame-rate standard is 24 frames per second (fps). Displaying a video on a desktop pc is usually done at 24 or 30 fps. The display rate for television, however, is 60 fps, although some television programs are produced in cinema-like 24 fps [AFH+].

The benefit from a higher frame-rate is that it provides a better sense of motion, especially fast movement, to the viewer. In general, a higher frame-rate such as 60 frames per second resembles reality better and therefore creates an improved illusion [AFH+].

However, as mentioned before, rendering the same sequence at a higher frame-rate subsequently requires more computations. This either results in more time to generate the same sequence or requires better hardware to do so.

1.2 Temporal Coherence

One approach to solve this problem is to use calculations done in one frame to generate the next or previous frame, in other words: temporal coherence. In this thesis, the frames generated using temporal coherence are referred to as *interpolated frames* (I-frames), which approximate the ground-truth frame or *original frame* (O-frame). Frames that are computed by the renderer will be referred to as *base frames* (B-frames). The process of generating the I-frames is also known as *upsampling*.

Many methods that use temporal coherence to increase performance exist. For example, Yang et al. [YTS+11] introduce two methods that use so-called *motion vectors* (vectors that show the positional change for each pixel) generated via the expected motion of the underlying geometry for each pixel. With these motion vectors, Yang et al. [YTS+11] can generate frames for the purpose of real-time rendering, but are unable to deal with the problems of mirror- or glass-like objects, since these may result into image features that do not move coherently with the motion vectors.

Video encoding techniques produce motion vectors by looking at blocks of an image and search for their position in another frame [WSBL03]. This can then be used to upsample a missing frame and potentially deal with the moving image features of reflective and refractive surfaces, since the detected movement does not depend on the objects like in the work of Yang et al. [YTS+11]. This technique does not guarantee a color value for each pixel in the missing frame and therefore requires a heuristic to fill in the missing values. Furthermore, video encoding standards restrict the amount of maximal movement to reduce computing time [WSBL03].

A completely different approach was introduced by Havran et al. [HDMpS03] for Monte-Carlo path tracing. Their rendering architecture updates samples generated via ray tracing for each frame in relation to camera- and object movement. In addition to that, a custom acceleration data structure for ray-tracing is utilized to create multiple frames at once. This technique is specifically tailored to the global illumination algorithm at hand (path tracing), but does deal with the shortcomings of the previously mentioned

techniques, i.e., it deals with specular and refractive scene objects and provides a value for every pixel.

1.3 Challenges

With the shortcomings of the previously mentioned techniques in mind, the goal of this thesis is to develop an algorithm that deals with the following challenges:

- **Independence of rendering algorithm:** When a technique works independently of the global-illumination algorithm, changing the latter does not require additional coding to apply the technique. For example, this is not possible with the previously mentioned technique by Havran et al. [HDMpS03]. Furthermore, a technique that is independent of the rendering algorithm is an easy way to add another temporal-coherence method to the existing ones.

- **Individual result for each pixel:** A motion vector for each pixel reduces the errors made for each image and therefore enhances the resulting generated image. Using video encoding to do the upsampling can not achieve that, since the motion vectors are calculated for blocks of pixels. This is a problem when objects in the foreground move across a stationary background. In this case, motion vectors might also be applied to pixels of the background, which leads to errors in the resulting image.

- **Independence of object movement:** Global-illumination effects such as specular or transparent objects generate image features that may not move like the object causing it. But when the movement of the image features is bound to the object motion, the quality of the resulting images decreases, as is the case with the techniques of Yang et al. [YTS+11].

- **Reasonable computing time:** An upsampling algorithm that does not need significantly less time than the global-illumination algorithm, defeats the purpose of it and therefore is not desirable as such. This challenge is not a problem of the previously mentioned techniques, but is something that should be kept in mind for the proposed techniques if this thesis.

- **Movement constraints:** The amount of movement from one frame to another should not be limited by the algorithm itself. Video encoding, for example, puts a limit to the maximal amount of movement to save encoding time. Again, this challenge is something that will be important for the proposed techniques. It has to be mentioned that temporal-coherence methods in general put an indirect limitation on the maximal movement based on the rate the ground-truth data is refreshed.

1.4 Approach

This thesis introduces two methods for upsampling that are independent of the global-illumination algorithm that generates the frames, while still providing high-quality results. Both of them are image based in order to achieve the independence of the rendering algorithm. To overcome the accuracy issue of using the video-encoding for upsampling and the problems of the work of Yang et al. [YTS+11], the proposed methods use the non-local means algorithm, which was introduced by Buades et al. [BCM05], to detect similarities between two or more frames. This algorithm defines a fixed search area (in the spatial and temporal domain) to find a similar pixel. Whether two pixels in that neighborhood are considered to be similar is decided by comparing their surroundings.

The first approach, the motion-vector technique, applies the non-local means algorithm to the color of two base frames to find the movement of a pixel from one frame to the other. The resulting motion vectors can then be used to fill a certain pixel in the interpolated frame with the corresponding color. This technique works similar to what is done when using video encoding to do the upsampling, but without a heuristic to fill in the missing pixels. The motion-vector technique also counters the problems of the work by Yang et al. [YTS+11] when it comes to specular and transparent objects.

The second technique, the pixel-similarity method, requires additional data for each pixel in an I-frame, a so-called *stubby frame*. This data for each pixel is then used to search for similar ones (using the non-local means algorithm) in the adjacent B-frames to color the pixels in the I-frame. The data needed for this approach can be extracted from any global-illumination algorithm without altering it. This technique guarantees a color value for every pixel and overcomes inaccuracies of the motion-vector technique, while simultaneously dealing better with reflective and refractive surfaces than Yang et al. [YTS+11], but not as well as the motion-vector technique. The pixel-similarity technique is not as accurate as rendering algorithm-specific upsampling techniques (like the spatial–temporal architecture for animation-rendering by Havran et al. [HDMpS03]), but also not as complex to implement, a property shared with the motion-vector technique.

The non-local means approach is chosen because of the following reasons. First of all, it can generate an individual result for each pixel and potentially detect movement for each pixel across the entire image and is therefore not restricted like the work of Yang et al. [YTS+11]. Some optical flow techniques do not fulfill the requirement of unrestricted movement detection and individual pixel results (as mentioned in Section 2.3) and are thus not as good of a choice as the non-local means algorithm. In addition to that, the non-local means algorithm can be optimized for parallel execution on central processing units (CPUs) or graphic processing units (GPUs)[GLPP08]. The time to generate the interpolated images with the proposed algorithm is therefore significantly lower than the time to create a high-quality computer-generated image, even when the algorithm is applied to each pixel individually. Furthermore, the proposed techniques of this thesis, which use the non-local means algorithm, are independent of the rendering algorithm that produces the images and can also work independently of the object movement, because the calculations are done afterwards on data that is not bound to the objects movement.

The latter mentioned property allows for dealing with the shortcomings of the techniques proposed by Yang et al. [YTS+11], and the fact that the calculations are done after rendering puts the methods of this thesis in contrast to the global-illumination algorithm specific temporal-coherence methods, such as the one by Havran et al. [HDMpS03].

1.5 Quality

Both methods will be evaluated against other techniques by comparing the mean squared error against the ground-truth frames and how the proposed algorithms deal with the following image features:

- **Edge location:** The positioning of the objects in the interpolated frames has to be equal to those in the ground-truth ones.
- **Texture variety:** Details on textures with a high variance in color as well as smooth transition on textures with low variance in color should be preserved. The latter also includes transitions caused from changing intensities of the incoming light.
- **Specular surfaces:** Reflective materials can result in rapid color changes in the resulting image sequence, and the movement of perceived image features may be different to that of the underlying geometry. The handling of surfaces with varying magnitudes of specularity/roughness is an additional quality criteria for the comparison.
- **Transparent objects:** Light traveling through the surface and into the object can be observed on materials such as glass and crystals. Similar to specular surfaces, those refractive materials can result in different movement of image features than that of the geometry in the scene.
- **Continuous sequence:** The image series with the interpolated images should be free of discontinuities. One reason for that is incoherent movement within a scene. For example, when a objects in the foreground moves, but the background is stationary.

1.6 Thesis overview

Chapter 2 of this thesis is about the theoretical background. It is followed by a description of the previous work in Chapter 3. Chapter 4 contains a discussion of the motivation for the proposed techniques and its relation to previous work, as well as the necessary prerequisite for the techniques. Both techniques are then explained in detail in Chapter 5 and Chapter 6. The requirements to run the algorithms in practice and the results for a test image sequence are discussed in Chapter 7. This thesis is concluded in Chapter 8, which also gives an outlook to future work.

CHAPTER 2

Background

In this chapter, background knowledge for this thesis is provided. At first, light and its transport throughout a scene is described, followed by an explanation of the so-called optical flow, which describes movement of image features in an image sequence. After that, an introduction to the theory of upsampling is given, which together with optical flow builds the basis for frame-rate upsampling. The last section explains image denoising, because the proposed approach originates in that field of visual computing.

2.1 Light and Color

Light describes the part of the electromagnetic spectrum that is visible to the human eye. That part ranges from the wavelength of 380 nanometers (nm) to 780 nm. Those are not fixed boundaries since it depends on the individual human eye at hand. In addition to that, the part of the spectrum with a wavelength from 10 nm up to 380 nm (ultraviolet) and from 780 nm up to 1 mm (infrared) is also refereed to as light.

Aspects of visible light include intensity, propagation direction, wavelength spectrum and polarization. The term color usually describes the combination of the intensities of wavelengths for a specific sampling of light, although the definition of what exactly is meant by the term color is debatable.

Due to the fact that the human eye samples the spectrum of light at three different areas (red, green and blue), rendering softwares often use three corresponding intensity values to represent light. However, such an representation does not allow for computing certain visual phenomena such as dispersion.

2.2 Global-Illumination

Light interacts with surfaces and other participating media, which result in changes of its path and spectral properties. Computing the journey of the light from its source to

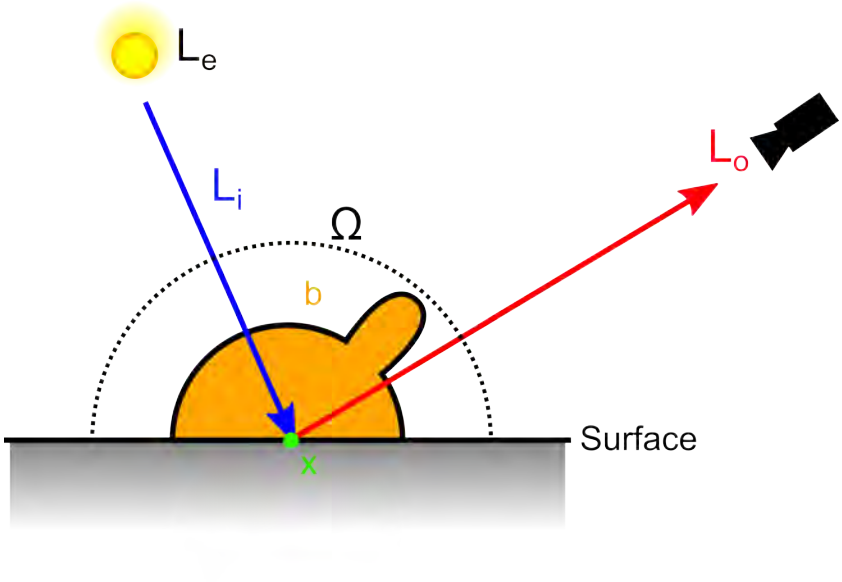


Figure 2.1: Schematic of the rendering equation as it is described in Formula 2.1

the observer in virtual scenes is commonly known as calculating global illumination.

As stated before global illumination is computationally expensive to compute. One of the reason for this is the nature of light transport itself which is described via the rendering equation [RDGK12] (as can be seen in Equation 2.1) and is depicted in Figure 2.1:

$$L_o(x, \omega_o) = L_e(x, \omega) + \int_\Omega b(x, \omega_o, \omega_i) L_i(x, \omega_i) cos\theta \, \mathrm{d}\omega_i \tag{2.1}$$

This variant of the rendering equation describes the outgoing light L_o coming from the point in space x with direction ω_o and integrates the incoming light into x over the hemisphere Ω. The other parts of the equation are:

- L_e: The emission of point x towards the direction ω_o.
- L_i: The incoming light from direction ω_i towards x.
- b: Is the bidirectional reflectance distribution function (BRDF) that describes the portion of light which is reflected from ω_i towards ω_o at point x according to the material. Several different ways to model such an BRDF exist, which all vary in their complexity and simultaneously their ability to represent various materials.
- $cos\theta$: Describes the reduction of the lights intensity when it comes from direction ω_i and hits the surface at x with a certain angle and thus increases the area it hits.

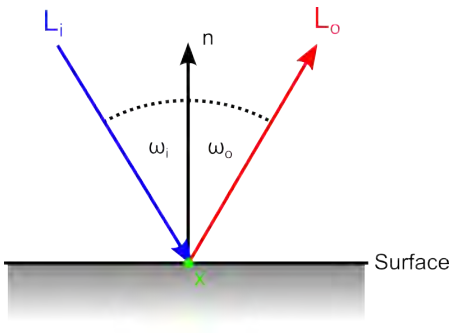


Figure 2.2: Reflection of a light ray.

Since the rendering equation consists of an integral that contains the light function itself, there is no analytical solution. In theory, this requires infinite computations to be resolved. In practice, this is avoided by introducing numerical thresholds to stop calculating when those do not significantly contribute to the result as well as simplified models, specialized algorithms and data structures to increase performance [RDGK12].

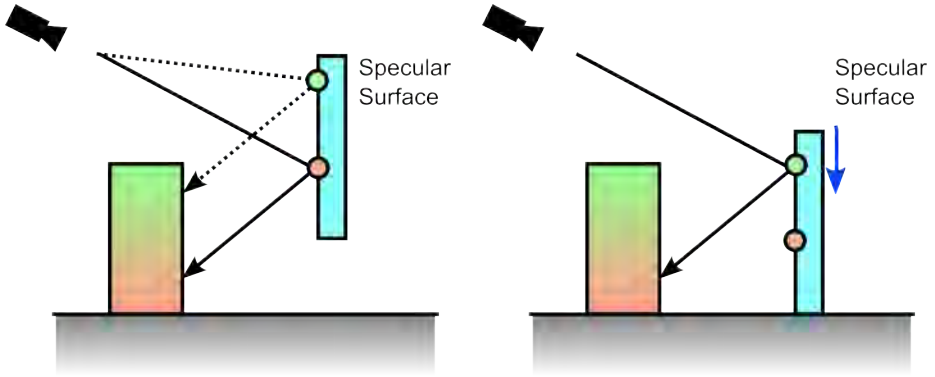
According to the SIGGRAPH Course 'Global-Illumination Across Industries' [KFC+10] the movie industry uses methods such as Monte-Carlo ray tracing with various optimizations or Point-Based global-illumination to calculate the light transport in a scene. A short overview of those and other global-illumination algorithms (including temporal and/or spatial coherence based optimizations) is given in Section 3.1.

2.2.1 Ideal Reflection

An ideal reflection occurs when incoming light is bounced off the surface, away from the object. The direction of the reflection is as follows: Given are an incoming light ray L_i, which hits a point x on a surface with a perpendicular vector n (the normal vector), and the outgoing light ray L_o. The angle between the incoming light ray ω_i and the normal vector then is always the same as the angle between the outgoing light ray and the normal ω_o, as seen in Equation 2.2 (wheres the $\cdot$ operator is the dot-product between two vectors) and Figure 2.2.

$$\omega_i = \omega_o or \frac{L_i}{\|L_i\|} \cdot \frac{L_o}{\|L_o\|} \tag{2.2}$$

Figure 2.3 shows what happens when a mirror translates: The color at the surface point will not stay the same. This highlights a problem of some previous work (for example the techniques introduced by Yang et al. [YTS+11], as described in Section 3.2) and the difficulties, that the proposed techniques have to deal with.



(a) Initial position (b) Applied movement

Figure 2.3: Problem of moving reflective objects. Figure (a) shows the initial positions and the path that results into the green pixel. The blue arrow is the movement of the specular surface in Figure (b). The resulting pixel color should be brown as well. However, when the motion of the object is followed (hence sampling the same point on the specular surface as before) this could into a (false) green pixel. This green color is due a hit on the top of the cube in the initial position.

2.2.2 Ideal Refraction

An ideal refraction occurs when a light ray hits a material, which allows the light to enter it. The direction of the refraction can be calculated using *law of refraction or Snell's law* (Equation 2.3), which states that the angle of the incident angle ω_i and the outgoing angle ω_o must be the reciprocal of the ratio between the respective materials indices of refraction η_i and η_o, as seen in Figure 2.4.

$$\frac{sin(\omega_i)}{sin(\omega_o)} = \frac{\eta_o}{\eta_i} \tag{2.3}$$

$$\omega_{crit} = sin^{-1}\frac{\eta_o}{\eta_i} \tag{2.4}$$

If the light in an object with an higher refractive index tries to leave it and the incident angle is beyond the so-called critical angle ω_{crit} (see Equation 2.4), the light path will be reflected internally, a so-called total internal reflection. This leads to an minimum of two changes in the light paths directions (as seen in Figure 2.5) before it arrives at the observer or another surface to interact with. The same is true when the light transport is seen in reverse with the starting point at the observer.

Refraction suffers from the same problems, in terms of moving geometry, as reflection, which is described in the previous section.

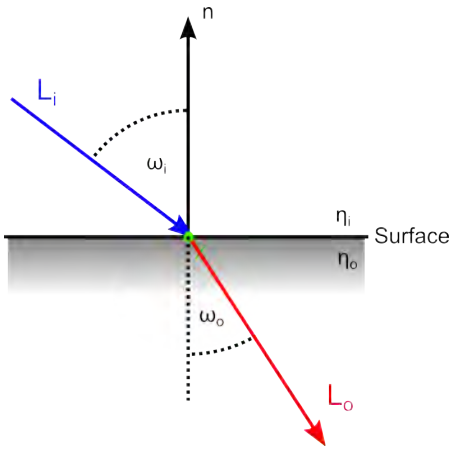


Figure 2.4: Refraction of a light ray.

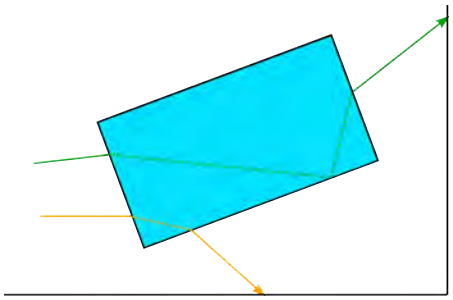


Figure 2.5: The orange and the green line show a possible light path through a transparent object (blue rectangle). Due to internal reflection the path may not exist after the second surface-interaction.

2.3 Optical Flow

Optical flow describes the apparent motion of objects through a scene from an observers standpoint. Such an optical flow of an object or feature is described via a vector in image space, a so-called motion vector as can be seen in Figure 2.6.

To determine the motion vectors in an image sequence, several techniques exist which all introduce additional constraints. Common methods are the Horn-Schunck method and improved variants of it as can be seen in the state of the art report of Sun et al. [SRB14]. The restraint of the Horn-Schunck method is that it assumes a coherent motion around the pixel for which it is calculated. At the edges of a moving object this will not be true an can result into false motion vectors [SRB14]. Another method is the Lucas-Kanade one. Its constraint is that the difference in time (and therefore change in the images) is kept to a minimum [SRB14]. As stated earlier, this can not be guaranteed when specular or transparent objects make up the scene.

Optical flow can be used in video compressing methods as it is described later in

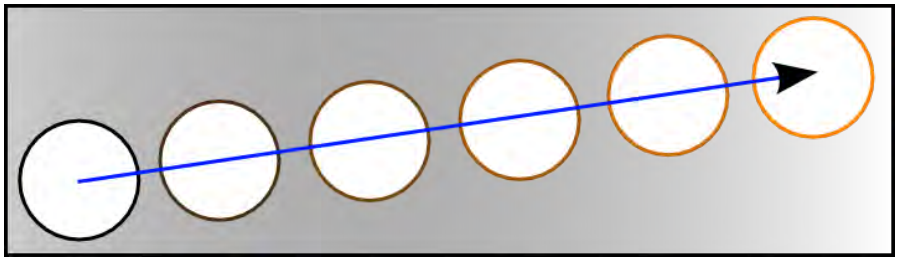


Figure 2.6: Optical flow of a circle moving from left to right. The according motion vector is colored blue.

Section 3.3. Furthermore, the notion of motion vectors are used in previous work as well as in the thesis itself and describes the positional change.

2.4 Upsampling

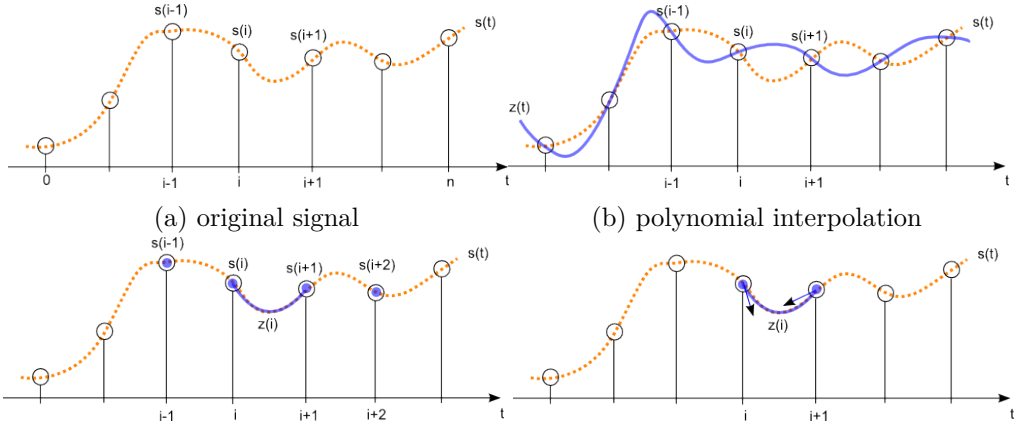
In signal processing, upsampling is often seen as the process of extending a sampled signal over a longer period of time. Another usage of the term upsampling, as it is used in this thesis, is to construct missing data points between a discrete set of known data points (aka sample points or samples). In mathematics this relates to the method of interpolation, which is explained in the following Subsection and is set into relation to frame-rate upsampling. Afterwards, the affinity and intricacies of upsampling for image-based renderings are described.

2.4.1 Relation to Interpolation

As mentioned before interpolation takes sampling points and constructs new values between those points. To define this problem properly: An unknown source function $s(t)$ is sampled n times at the locations $i = 0, 1, 2, ..., n$. Interpolation then tries to approximate that source function with $z(t)$ based on all values of $s(i)$. This can also be seen in Figure 2.7a.

One way to approximate the missing signal points is to perform a linear interpolation on an interval between two sampled points $s(i)$ and $s(i + 1)$. Unfortunately, this naive approach does not provide sufficient visual quality for a human observer. So does any other arbitrary weighting function of the two sampled points [AFH+].

Another way of doing so is to fit a polynomial curve through the sampled points. Such a function extends beyond the samples on the time-line and can lead to a problem known as 'over-fitting', where the difference to the original signal $\int_0^n z(t) - s(t)dt$ increases as more samples are used to generate such a polynomial curve. Furthermore, this global approach requires the original signal to be of polynomial nature for accurate results, which the rendering equation (as in Equation 2.1) does not fulfill. A schematic of such a polynomial fit is given in Figure 2.7b.



(a) original signal (b) polynomial interpolation

(c) Natural cubic spline interpolation (d) Hermit cubic spline interpolation

Figure 2.7: Curve Fitting example. Sub-figure (a) depicts the original signal with its sampling points at t_i, Sub-figure (b) shows a polynomial fitting, Sub-figure (c) shows a natural cubic spline fitting with the blue dots marking the influence on $z(i)$ and Sub-figure (d) shows a Hermit cubic spline fitting with its derivatives $\dot{s}(i)$ (blue arrows)

To avoid the problems of regular polynomial interpolation, spline interpolation uses piecewise polynomials of some degree. One of those is the natural cubic spline, which uses polynomials of degree three $\dot{z}(i)$ to interpolate between two adjacent sample points $s(i)$ and $s(i+1)$. To generate a continuous curve $z(t)$ throughout all sampling points, the first derivative $\dot{z}(t)$ and second derivative $\dddot{z}(t)$ of the spline have to be continuous. Furthermore, those derivatives have to be equal at the two adjacent sampled points: $\dot{z}(i) = \dot{z}(i+1)$ and $\dddot{z}(i) = \dddot{z}(i+1)$. Therefore, someone needs to use four sampling points to interpolate between two points as seen in Figure 2.7c.

Hermit cubic splines reformulate the natural cubic spline so that it is defined by the values of the sampling points $s(i)$ and the values of their first derivatives $\dot{s}(i)$. Such a definition is called the hermit form. Figure 2.7d depicts the principal of the Hermit cubic spline.

In current frame-rate upsampling methods (as mentioned in the work of Scherzer et al. [SYM+11] and described later in Section 3.2) the principal of using the sampled values and their derivatives to generate missing data is employed to create frames between those available. The motion vectors from optical flow, as described in Section 2.3, can be interpreted as the derivatives of an image in a sequence. However, some state-of-the-art methods do not use motion vectors that point towards the positional change of the shading but rather use motion vectors that point to the change in position of the underlying geometry [SYM+11]. Furthermore, the actual interpolation method of any splines can not be employed to frame-rate upsampling, because the change in such a series of frames is often not continuous and splines do not generate such an interpolation as can be seen above.

2.4.2 Application in Image Based Rendering

When it comes to generating frames out of information stored in each pixel from adjacent ones, there are two general approaches to finding the corresponding pixels with the data needed [SYM+11]:

- **Reverse reprojection:** This approach can be used, when there is data available for each pixel in the I-frame that points to those in the B-frame. The value of one pixel can then be set based on the data provided by such a pointer.
- **Forward reprojection:** The alternative is to start from the pixels in the B-frame. When there is a pointer towards the position in the I-frame, the data stored can then be injected into the pixels of the I-frame.

Both reprojection variants can lead to pixels with wrong or no data since it is essentially a non-linear warping. Reverse reprojection can have pixels with pointers that may not point towards the correct data. In forward reprojection those faulty pixels can be those for which no or multiple injection happen. In both cases some sort of heuristic has to be employed to correct those wrongly associated values, if the particular method does not allow for a ground-truth calculation of the faulty pixel values [SYM+11].

Those two approaches of finding corresponding data can be applied to upsampling methods that do no work on a pixel-basis, but use temporal or spatial coherence otherwise. The only requirement is that a relation between two corresponding data points is available in some way [SYM+11].

Some of those methods that use temporal coherence also require a decision on when to update the data that is used through time [SYM+11]. In the approach used for this thesis the I-frames are generated by the closest B-frames in time. For the use-case of upsampling 30 frames per second to 60, this relates to an unique pair of B-frames for each I-frame and as accurate data as possible for those.

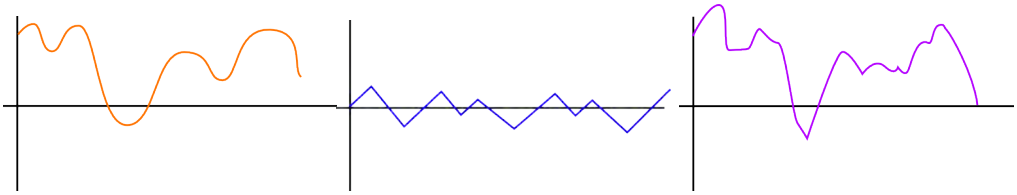
2.5 Image Denoising

Image noise is a random altering of the pixel colors from the ground truth image. Denoising is then the process of restoring the original image (which can be seen as a two dimensional signal) from a noisy image. The noise of an image can be interpreted as a two dimensional signal itself, which is added to the original image to make up the noisy one. A schematic of this relation for a one dimensional signal can be seen in Figure 2.8 and is described via the Equation 2.5 for both one dimensional signals and two dimensional images:

$$P_x = O_x + N_x \tag{2.5}$$

whereas

- O_x: Is the noise-free, original, signal at position x.



(a) basic signal (b) noise signal (c) combined signal

Figure 2.8: Schematic of a noisy signal. Left: A possible, not altered, signal. The goal of denoising methods is to extract this signal. Middle: A (random) noise signal. Right: the combination of both signals. This is the problematic signal that has to be resolved via denoising methods.

- N_x: Is the noise at position x.
- P_x: Is the combined signal of noise and the original signal.

An example of a denoising methods are local filter, which can be described by the following formula:

$$O_x = \frac{\sum_K \omega(x,k) P_k}{\sum_K \omega(x,k)} \tag{2.6}$$

whereas

- K: Is the neighborhood around x. In this neighborhood the algorithm searches for similar pixels.
- k: Is one position in the neighborhood K.
- ω: Is the weighting function between the signal values x and k.

This thesis approach makes use of such a denoising method, which is described in Section 3.4.

CHAPTER 3

Previous Work

3.1 Coherence in Global-Illumination Algorithms

As mentioned in Section 2.2, calculating global illumination(GI) [RDGK12] is done using various optimizations due to its computational complexity. One such method is to make use of temporal and/or spatial coherence, as the proposed algorithm of this thesis does as well. For global-illumination algorithms, those methods are usually tailored to the specific technique at hand. In the following list an overview of such techniques and the according optimization (using spatial/temporal coherence) is given:

- The **Finite element**(FE) method (introduced by Goral et al. [GTGB84]), also known as radiosity method, discretize the scene's surfaces into finite elements (surface patches) and calculates the light transport between them. This results into solving a linear system, where the amount of light between two patches is described via so-called form factors. Once the linear system is solved, the rendering of each frame is fairly inexpensive, since the incoming light is known for each patch. However, the form-factor calculation is quite expensive and the meshing of the scene has to be performed so that GI can be calculated accurately and efficiently [RDGK12]. Performance can be increased by using link structures to avoid visibility computations. If the change within a scene is small enough, those link structures can stay valid and do not need to be rebuilt [DSDD07].

- **Monte-Carlo ray tracing**(MCRT) calculates the path of the light via shooting rays into the scene from an observers standpoint. As soon as the ray interacts with a medium or a surface, the ray will be redirected into one possible direction resulting in a path through the scene. This requires a lot of rays to be shot into the scene so that the average of the rays converges to the solution. Increasing performance for MCRT can be done, for example, via importance sampling (shooting rays where the amount of light is expected to be high) and using data structures that work

with spatial and temporal coherence. To give an example: The former can be done via bi-directional path tracing[LW93] or Metropolis Light Transport [Vea98] and the latter with a method known as 'Irradiance Caching' which simply reuses calculations of a previous frames [KFC+10].

- **Photon Mapping**(PM) works compared to MCRT from the other side via shooting photons from the light source into the scene. Those photons are bounced around in the scene and stored into a map each time a surface is hit. The evaluation of incoming light at each point in space is then calculated via a density estimate or final gathering. PM is especially good in rendering caustics in a scene. Speedups for this method can be achieved via using tree structures for efficient storage of photons, using correcting photons when hitting dynamic objects for utilizing temporal coherence and many others as seen in Ritschel et al. [RDGK12].

- **Instant Radiosity**(IR) works similar to PM, but instead of estimating, every photon is treated like a light source, a so-called virtual point light (VPL). The indirect light for each point in space is then calculated by checking the contribution (visibility) from each light. Spatial coherence is adapted into this method by increasing the accuracy of the VPLs in areas where the contribution to the final result is high. Such a method is Metropolis IR, which is similar to Metropolis Light Transport for MCRT [RDGK12].

- **Point-based Global-Illumination** (PBGI) works similar to FE and uses VPLs. This approach approximates the geometry via small discs (the points) which are put into a hierarchical structure. Light is then transported between the points (utilizing the hierarchy) and stored at them. Now one can calculate the incoming light of a point in space by rendering the discs into a hemispherical map around the desired point. Apart from using 'Irradiance Caching', one can also interpolate the incoming light on the edges of a surface patch if the variation on that light is small [KFC+10].

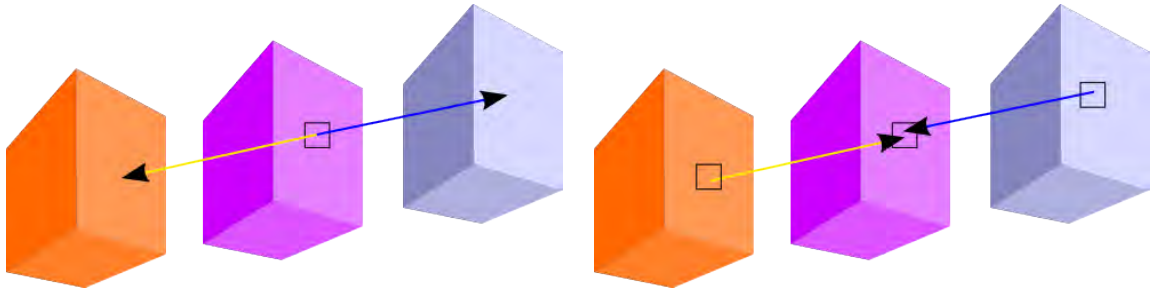
This thesis approaches differ from such tailored methods, because they are designed to be fairly independent from the global-illumination algorithm that is used to generate the image.

3.2 Upsampling Techniques in Real-time Rendering

In this section, an overview of frame-rate upsampling techniques is given. All of them focus on the usage in applications which want to achieve a real-time frame-rate of 60 fps.

3.2.1 Bidirectional Image Re-projection

Yang et al. [YTS+11] proposes several methods that use motion vectors to do the upsampling. In this case, motion vectors exist for each pixel (unlike in video compression as seen in Section 3.3) and can be calculated through the knowledge of the objects



Scene Assisted Image based

Figure 3.1: Difference in the data usage of two methods proposed by Yang et al. [YTS+11]. The scene-assisted approach requires a rendering of the desired frame, whereas the image-based approach does not. In each case the purple cube represents the unknown shading information, the orange one represents the shading from the previous frame and the blue depicts the shading information of the upcoming frame

position in the adjacent frame. The different methods all vary in the available data (as seen in Figure 3.1) for the upsampling and hence in the quality of their result.

Scene-Assisted approach

This method requires a complete rendering of the scene, but without calculating the shading. It also needs the motion vectors pointing to a frame backwards on the time line as well as motion vectors pointing to the next frame on the time line.

Those motion vectors are then used to look up the colors in their respective frame. It is therefore a reverse reprojection approach as described in Section 2.4.2. According to the authors, the scene assisted method produces the best results. This technique is used in this thesis for comparison with previous work and as a fallback for one of the proposed methods, which is described in Chapter 5.

Image-Based approach

The other technique proposed by Yang et al. [YTS+11] does not require any rendering of the I-frame. In this case, one has to search for the pixel in the B-frame that will move to the location of a desired (not yet colored) pixel in the I-frame and therefore qualifies as a forward reprojection as seen in Section 2.4.2.

This is done by executing an iterative search for the source pixel S. That search is initialized by the target pixel T's motion vector m_T. This vector is subtracted from the target pixels location giving a new pixel N. Then the motion vector of N is checked if it points towards the location of T. If it does, we found the pixel that moves to our target pixel. If it does not point to T, the motion vector of N is taken and subtracted from T. The process is repeated until a valid pixel that leads to T is found or a maximal number of iterations is exceeded, classifying that pixel as indeterminable. An example of such an iterative search is given in Figure 3.2.

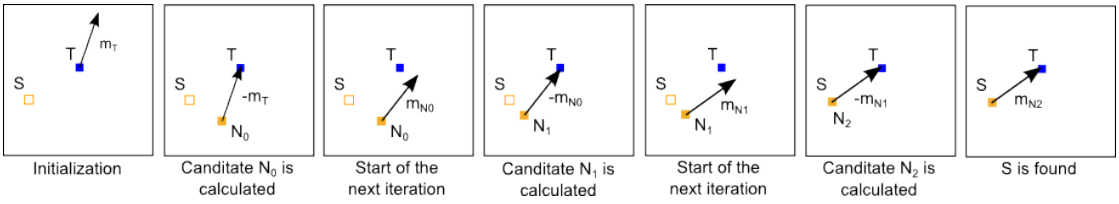


Figure 3.2: Three iterations of the algorithm by Yang et al. [YTS+11] which is used to estimate the movement of a pixel from one frame to another

On a pixel for which no valid source location in a B-frame can be found, heuristics have to be employed. For example, using the result of the first iteration or using the source location which comes closest to the target pixel when the according motion vector is added to it. Yang et al. [YTS+11] suggest a double iterative search to reduce the number of undetermined pixels: one with the previous B-frame and one with the next B-frame. Therefore, a heuristic only has to be employed when both iterative searches fail.

In this thesis the usage of that method is considered in the proposed techniques which is described in Chapter 5.

Occlusion Check

The techniques described in the previous section require a check if the calculated source pixel (the corresponding pixel in the B-frame) is actual valid. An invalid source pixel can occur when the motion vectors indicate a movement from a point in space to another whereas one of them is behind other geometry from the cameras point of view. To counter that problem, someone has to check if the sampled source pixels position in space and that of the target pixel relate to each other. Therefore positional information has to be included for each pixel. This, for example, can be a depth buffer as it is a by-product of rasterization, the most common method for real-time rendering. Furthermore, to determine if the positions of the two pixels relate to each other, the motion vectors have to contain not only the change in the two dimensional screen space, but also a third dimension to determine the depth change.

To sum it up, in the case of the scene assisted approach: one knows the depth information of the target and the source pixel, as well as the change of the depth information through the motion vector. A valid source pixel is then found if the depth change from the target pixel to the source pixel is equal or similar to the depth at the source pixel. This entire process can also be done by using world space coordinates for each pixel and the according three dimensional motion vectors, as it is done in both of the proposed techniques of this thesis.

The occlusion check for the image-base approach is more limited, since no information of the I-frame is available. In this case, one can compare the depth values which are indicated by the B-frame before the I-frame and the depth values from the one after it. Those depth values are computed by taking the depth of the source pixel and adding the motion vectors depth component. The target pixel is then colored based on the source

which indicates a depth value closer to the camera.

Results

The method of Yang et al.[YTS+11] generates, according to them, high quality output for their test scenarios. However they also point out that, as previously stated by Nehab et al.[NSL+07], the method does produce noticeable errors in scenes which feature highlights, transparency, and reflections as the lead to rapid movement relative to the object itself.

Furthermore, the image-based approach suffers from the errors that comes along when the iterative search does not find an appropriate source pixel, which occurs when objects move fast across frames, or when the motion vectors are inconsistent around the target pixel which leads the iterative search to not converge.

3.2.2 Iterative Image Warping

Bowles et al.[BMS+12] introduces a faster technique, which initializes the image-based approach described in [YTS+11] in a different way. The method subdivides one original frame into any number of quads. A quad is subdivided if the motion vectors in that region are not uniformly. Those quads are then rendered in the interpolated frame at the position the motion vectors indicate. Now one can find better initialization values on the pixels in the interpolated frames. Since this technique only improves performance and not quality, we do not compare it to our results.

3.3 Upsampling using Video Coding

In video coding temporal coherence is used to reduce the file size. The idea is to re-use blocks of pixels from one frame in an upcoming or previous one, so-called motion estimation. For this, motion vectors are utilized to define such a movement from one frame to another. The algorithm that is used in video coding to generate those motion vectors is generally refereed to as block matching or patch matching.

The first step of block matching is to partition the frame into similar sized blocks. The size depends on the coding standard at hand. The next step is to search in the other frame for a similar block. This search is performed within a local range of pixels, which again can be defined by the codec. The similarity of two blocks, with the middle pixels a and b and a size of d_x/d_y, in two different frames f_1 and f_2 can be defined by various measures, for example, the sum of squared errors (as seen in Equation 3.1) or sum of absolute differences (as seen in Equation 3.2) of the per pixel differences (using the color c). The motion vector then points towards the block with the highest similarity. For the two similarity measures that is the block with the lowest value [MT13].

$$SSD(a,b) = \sum_{d_x} \sum_{d_y} [c(f_1, a_x + d_x, a_y + d_y) - c(f_2, b_x + d_x, b_y + d_y)]^2 \tag{3.1}$$

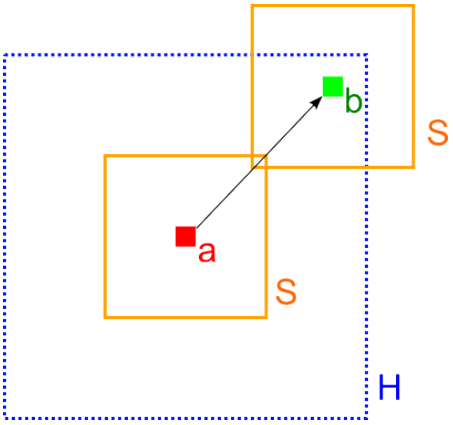


Figure 3.3: Schematic of the non-local means algorithm for denoising introduced by Buades et al. [BCM05].

$$SAD(a,b) = \sum_{d_x} \sum_{d_y} |c(f_1, a_x + d_x, a_y + d_y) - c(f_2, b_x + d_x, b_y + d_y)| \tag{3.2}$$

To improve performance, several strategies can be employed. Early termination at measuring the similarity is on option and another one is to move through the local search range in a particular pattern [MT13]. The H.264 standard also uses variable block size (4x4 up to 16x16 pixels), sub-pixel movement detection and multi-frame motion compensation [WSBL03].

The actual upsampling is done by moving the blocks according to their motion vectors and paint them into the missing I-frame, in other words a forward reprojection (see Section 2.4.2). Pixels that can not be filled by doing so are then colored by either interpolating the colors of the pixels at the same location in adjacent frames or taking the color of just one of those pixel.

3.4 Non-local means denoising

Apart from the local methods, as described in Section 2.5, non-local methods are another approach for denoising images. One such method is the non-local means algorithm which was introduced by Buades et al. [BCM05]. The algorithm works as follows: For one pixel the similarities to all other in a spatial neighborhood (the blue-doted area in Figure 3.3) around it are calculated. Two pixels are classified as similar when their surrounding (the orange areas in Figure 3.3) match. The color of the pixels in the neighborhood are then weighted by the similarity value and added together to make up the pixel color without noise. Mathematically, this method is described via the following formula (which uses the same notion as in Equation 2.5):

$$O_x = \frac{\sum_K \sum_S \omega(x+s, k+s)\sigma(s)P_k}{\sum_K \sum_S \omega(x+s, k+s)\sigma(s)} \tag{3.3}$$

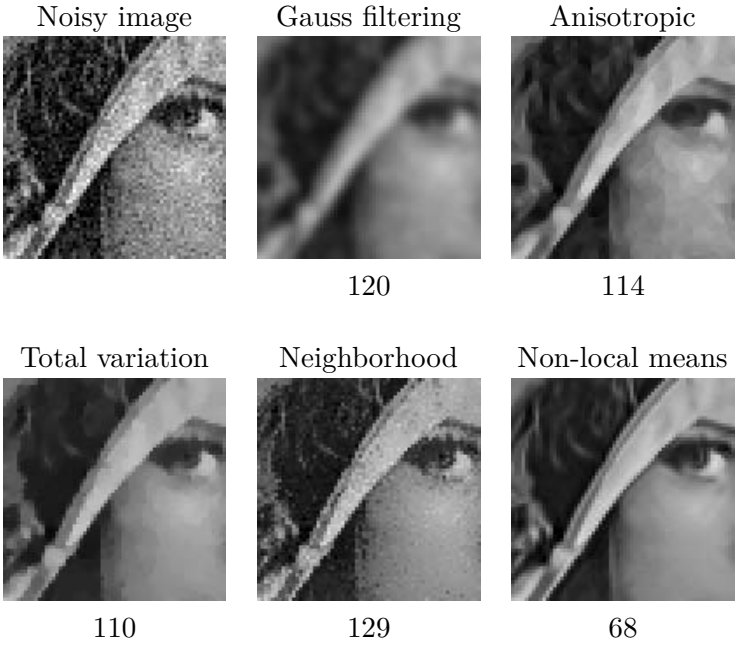


Figure 3.4: Comparison of the non-local means denoising algorithm for a single image against other filtering methods. The number below the thumbnail images depict the mean squared error of the entire denoised image towards the original image. Pictures taken from Buades et al. [BCM05].

whereas

- S: is the surrounding of a pixel on which similarity is defined by the algorithm. This will be refereed to as the similarity area around a pixel.
- s: is one point in the surrounding.
- σ: is the contributing factor of a point in the surrounding.

Buades et al. [BCM05] tested their algorithm on two different image against various other denoising methods. The results of one of them can be seen in Figure 3.4. The mean squared error measure also provides proof that the non-local means algorithm performs better than the other techniques. For more details on their findings, consult the original paper.

3.4.1 Non-local means for video denoising

When it comes to denoising video sequences, more than one dimension has to be taken into account. Two dimensions represent the image and affect the neighborhood of one pixel as well as the similarity area. The third dimension represents the time in the video.

This temporal dimension only affects the neighborhood and not the similarity area in the algorithm. The resulting denoising method is described in Algorithm 3.1.

Algorithm 3.1: Non-local means algorithm for video denoising

Input: An image sequence with the images $\mathbf{P_n}$ each of size WH
Output: An image sequence with the images $\mathbf{O_n}$ that do not contain the noise

for *xy within* WH **do**
  $\mathbf{W_{sum}} \Leftarrow 0$;
  for k_{xyn} *in neighborhood* K **do**
    $\mathbf{W} \Leftarrow 0$;
    for s_xy *in similarity area* S **do**
      $\mathbf{W} = \mathbf{W} + \xi(\mathbf{P_n}(x + s_x, y + s_y), \mathbf{P_{n+k_n}}(x + k_x + s_x, y + k_y + s_y))$
    end
    $\mathbf{W} = \kappa(\mathbf{W})$;
    $\mathbf{O_n}(x, y) = \mathbf{O_n}(x, y) + \mathbf{P_{n+k_n}}(x + k_x, y + k_y) * \mathbf{W}$;
    $\mathbf{W_{sum}} = \mathbf{W_{sum}} + \mathbf{W}$;
  end
  $\mathbf{O_n}(x, y) = \mathbf{O_n}(x, y)/\mathbf{W_{sum}}$
end
return $\mathbf{O_n}$

Whereas $\xi(a, b)$ is the function that calculates the difference of pixels in the similarity area. In this case, the function takes the absolute value of the color difference of those two points. Goosens et al. [GLPP08] also suggest the incorporation of thresholds to map the weight of not similar points in the neighborhood to zero. This is done in the function $\kappa(w)$, which in this case also scales the remaining range of values in a linear fashion.

When comparing pixels from two different frames X and K, the similarity measure can be seen as the likelihood that x (within X) moved to the location of k (within K). This reinterpretation is the key idea to the algorithm proposed in this thesis.

3.4.2 GPU accelerated non-local means denoising

From Equation 3.3 it can be seen, that the computational amount can be described as:

$$O(WH * K * S) \tag{3.4}$$

whereas

- WH: is the image resolution
- K: is the size of the neighborhood.
- S: is the size of the similarity area.

The work done by Goossens et al. [GLPP08] improves the work of Buades et al. [BCM05] by rearranging the process of calculating the method such that it benefits execution on the graphics processing unit(GPU) as well as the central processing unit(CPU) and therefore increases performance.

This is done by iterating over each pixels neighborhood first. The reason for this is the splitting of the sum to calculate the weight from each point in the neighborhood, which allows for the similarity measure to be executed in several so-called full-image passes.

Those full-image passes are calculations done for each pixel in an image, which can be executed independent from each other (meaning that no pixel calculations depends on another). This benefits the architecture of the GPU as its processing units are aligned in a parallel fashion and the memory is also optimized to be accessed in such a way.

The full-image passes that have to be done are the calculation of the difference between every pixel, resulting in a similarity weight for each pixel. After that, the sum of those weights has to be calculated and finally the contribution of the color with the sum of weights has to be applied to the result. A normalization of the contribution has to be done as well, which happens after the iteration over the neighborhood. To further improve the performance, the calculation of the sum of similarity weights can be split into two full-image passes, as it reduces the number of calculations and improves the memory access pattern which again benefits the GPU architecture. The resulting optimization to the technique can be seen in Algorithm 3.2.

Goosens et al.[GLPP08] tested their optimized version on a NVidia GeForce 9600GT GPU using DirectX 9.1 against an CPU implementation on 2.4GHz Intel Core(2) processor with 2048 MB RAM. The tests were executed on 99 frames of dimensions 720x480 with an added Gaussian noise with standard deviation 25/255 (PSNR=20.17dB). The results can be seen in Table 3.1.

For more details on the results, please consolidate the original paper of Goossens et al. [GLPP08].

Algorithm 3.2: Non-local means algorithm for video denoising optimized version

Input: An image sequence with the images $\mathbf{P_n}$ each of size WH

Output: An image sequence with the images $\mathbf{O_n}$ that do not contain the noise

$\mathbf{WS}[WH] \Leftarrow 0$;
for k_{xyn} *in neighborhood* K **do**
  $\mathbf{W}[WH] \Leftarrow 0$;
  $\mathbf{W}_{sum}[WH] \Leftarrow 0$;
  $\mathbf{W}_{sumTmp}[WH] \Leftarrow 0$;
  for xy *within* WH **do**
    $\mathbf{W}(x,y) = \xi(\mathbf{P_n}(x,y), \mathbf{P_{n+k_n}}(x+k_x, y+k_y))$
  end
  for xy*within* WH **do**
    for s_x *in similarity area* S **do**
      $\mathbf{W}_{sumTmp}(x,y) += \mathbf{W}(x+s_x, y)$
    end
  end
  for xy *within* WH **do**
    for s_y *in similarity area* S **do**
      $\mathbf{W}_{sum}(x,y) += \mathbf{W}_{sumTmp}(x, y+s_y)$
    end
  end
  for xy *within* WH **do**
    $\mathbf{W}(x,y) = \kappa(\mathbf{W}(x,y))$;
    $\mathbf{O_n}(x,y) = \mathbf{O_n}(x,y) + \mathbf{P_{n+k_n}}(x+k_x, y+k_y) * \mathbf{W}(x,y)$;
    $\mathbf{WS}(x,y) = \mathbf{WS}(x,y) + \mathbf{W}(x,y)$;
  end
end
for xy *within* WH **do**
  $\mathbf{O_n}(x,y) = \mathbf{O_n}(x,y) / \mathbf{WS}(x,y)$
end
return $\mathbf{O_n}$

Neighborhood Size	GPU msec/frame	CPU msec/frame
5 x 5 x 0	10.00	4021.00
5 x 5 x 3	24.79	N/A
7 x 7 x 0	19.06	7505.00
7 x 7 x 3	60.31	N/A
11 x 11 x 0	54.17	18230.00
11 x 11 x 3	184.48	N/A
21 x 21 x 0	231.56	50857.00
21 x 21 x 3	805.83	N/A

Table 3.1: Comparison of the optimized non-local means denoising method by Goossens et al. [GLPP08] on a GPU against a CPU version of the algorithm. The neighborhood size is given in width x height x temporal-window of previous frames

CHAPTER 4

Motivation and Overview

In this chapter a short motivation for the proposed algorithms is given, followed by an overview of the algorithms. In the last section, the core of both methods, the adaption of the non-local means denoising algorithm, is described.

4.1 Motivation

As mentioned before, one key feature of the proposed techniques is the independence of the rendering technique, that produces the images. The reason for this aspect, is the variety of methods available and the constant improvement of them, as described in Section 3.1. Therefore, an image-based approach to upsampling is chosen.

Yang et al. [YTS+11] introduced two such upsampling techniques aimed at real-time rendering applications. They use motion vectors, generated by the geometry of the objects in the scene, to accomplish the upsampling. This results in a problem: The optical features generated by specular and transparent objects may not move coherently with the motion of the objects themselves. This problem is described in Section 2.2 in more detail.

A solution to that is to calculate motion vectors from the color of two frames (as do the optical flow methods seen in Section 2.3) and do the upsampling based on that. Video encoding uses this technique to compress the data size. However, video encoding works on a block-bases, which results in inaccuracies.

To sum it up: we propose image-based techniques that calculate the origin of the pixels independently of the object's movement (unlike Yang et al. [YTS+11] described in Section 3.2.1) and are more precise than the video encoding technique (see Section 3.3), due to a unique motion vector calculation for each pixel.

4.2 General Approach

The idea to accomplish the above-mentioned goals is to search for similarities for each pixels across the frames. The way to do so is to use the non-local means denoising algorithm, which does just that: For each pixel it searches in its neighborhood for similar pixels based on the surroundings of the pixels. One can argue that optical flow methods do that as well, but unlike the non-local means algorithm, those suffer from certain constraints, such as coherence around a pixel (Horn-Schunck method) or minimal change (Lucas-Kanade method). For more details , please consider the state-of-the-art report of Sun et al. [SRB14] or Section 2.3. Apart from that, the non-local means can be optimized for GPUs as well as for parallel execution in general, as described by Goossens et al. [GLPP08] and in Section 3.4.1.

The first approach, the motion-vector technique, is inspired by the workings of the video encoding upsampling and by the iterative search approach of Yang et al. [YTS+11]. The proposed technique finds similar pixels within a pair of B-frames, giving every pixel a motion vector, which then can be used to generate the missing frame. Unlike in the approach of Yang et al.[YTS+11], the motion vectors are not bound to the geometry, which should counter the problem of reflections of refractions, as mentioned in the previous section. With the fast non-local means algorithm, introduced by Goossens et al. [GLPP08] and described in Section 3.4.2, it is possible to speed up the necessary calculations significantly.

The second introduced method, the pixel-similarity technique, is inspired as well by the paper of Yang et al. [YTS+11]. Unlike the other method, this one is based on the scene-assisted technique for upsampling. The first step of this method is to render an I-frame without the shading, but with data of the surfaces, which are sampled at the pixels locations. The search for similar pixels is then done based on the surface data and from the I-frame towards the B-frame. The latter then provides the missing color of the pixels.

4.3 Adapted Non-Local Means algorithm

The original version of the non-local means algorithm of Buades et al. [BCM05] (as described in Section 3.4) uses thresholds so that not every pixel of the neighborhood area contributes to the end result of the pixel. Otherwise, all pixels would contribute to the pixel color, leading to a 'blurring effect'. This is because the amount of pixels in the neighborhood area is in general big enough so that a lot of pixels with a small weight have a significant effect on the result. An example of this is shown in Figure 4.1, which shows some upsampled frames of the sequence with the original version of the non-local means algorithm. However, the above-mentioned thresholds require some knowledge about the value range of data at hand, and the required thresholds can potentially vary during the image sequence. Those varying thresholds can happen, for example, at close-up zooming sequences. Such a sequence starts with an arbitrary view of a scene and later-on shows just a small part of it, but in more detail. So when the non-local means is tuned for the

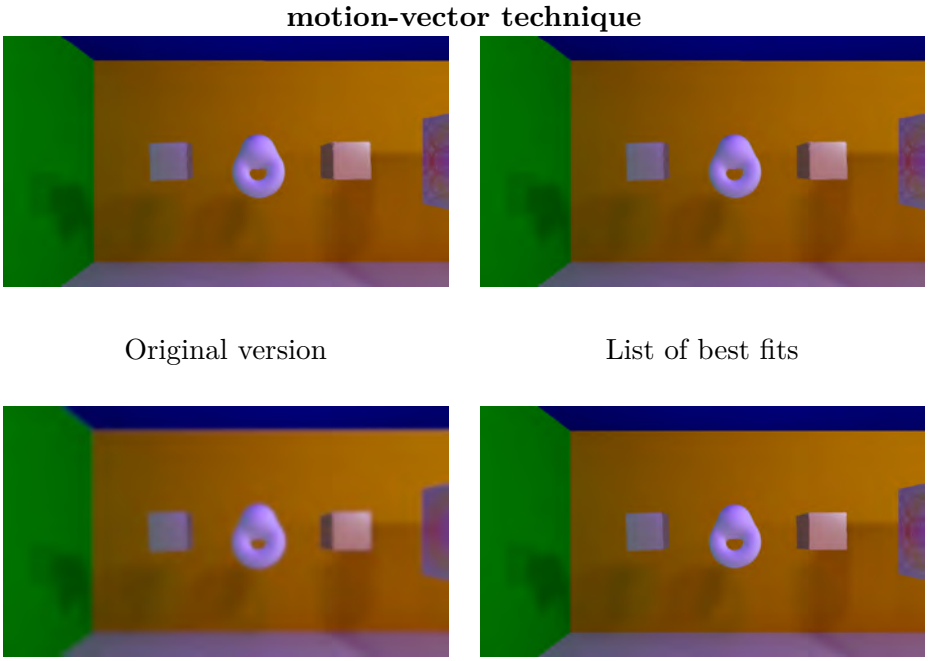


Figure 4.1: Difference between using the original non-local means algorithm and the version with the list of best fits.

starting point of view, it can occur that the tuning does not full-fill its purpose anymore.

To counter the problem of setting thresholds, a version of the algorithm that stores only a certain number of the neighboring pixels with the highest similarity is introduced in this thesis. This is accomplished by allocating a list structure for each pixel and only injecting a value of a neighboring pixel into it when its weight (which is calculated by the non-local means similarity measure) is bigger than any of those already in the list.

The resulting version of the non-local means method, with the performance enhancement proposed by Goosens et al. [GLPP08], can be seen in Algorithm 4.1. Furthermore, this version also features a weighting of the similarity area via a kernel function $\mathbf{K}(s)$, which scales with the distance to the middle of the area.

As mentioned before, the similarity between two pixels is based on their surrounding, the similarity area. Within those similarity areas, the difference between two values $\xi(a, b)$ is calculated differently for both techniques and will be described in the respective sections. The return value of that function is always a scalar (due to the nature of the non-local means algorithm) between 0 and 1 (chosen for convenience, where 1 indicates similar pixels and 0 the opposite).

It has to be mentioned that although the non-local means similarity detection is commutative ($\xi(a, b) = \xi(b, a)$), it is not guaranteed that when a search is started at pixel a and the highest similarity is found for pixel b, that when a second search started at b also results in the highest similarity present at pixel a. The reason for this is the

Algorithm 4.1: Maximal similarity detection

Input: An image **B**, an image **O** without the color information, both of size WH and an image-like list **E** containing the maximal N weights w and the associated values e for each pixel of WH

Output: An image **I** that contains the color information missing in **O**

for k_{xy} *in neighborhood* K **do**
  $\mathbf{W}[WH] \Leftarrow 0$;
  $\mathbf{W}_{sum}[WH] \Leftarrow 0$;
  $\mathbf{W}_{sumTmp}[WH] \Leftarrow 0$;
  for xy *within* WH **do**
    $\mathbf{W}(x, y) = \xi(\mathbf{O}(x, y), \mathbf{B}(x + k_x, y + k_y))$
  end
  for xy*within* WH **do**
    for s_x *in similarity area* S **do**
      $\mathbf{W}_{sumTmp}(x, y) += \mathbf{W}(x + s_x, y) * \mathbf{K}(s_x)$
    end
  end
  for xy *within* WH **do**
    for s_y *in similarity area* S **do**
      $\mathbf{W}_{sum}(x, y) += \mathbf{W}_{sumTmp}(x, y + s_y) * \mathbf{K}(s_y)$
    end
  end
  for xy *within* WH **do**
    $\mathbf{E}(x, y) \Leftarrow w = \mathbf{W}_{sum}(x, y), e = \mathbf{B}(x + k_x, y + k_y)$
  end
end
for xy *within* WH **do**
  $I(xy) = \sum_{\mathbf{E}(xy)} e_{xy} * w_{xy} / \sum_{\mathbf{E}(xy)} w_{xy}$
end
return I

different set of pixels for which the search is conducted at pixels a and b. In other words: Pixel b might find a better fitting pixel c (compared to a) in its neighborhood area which it does not share with pixel a. To conclude this, the non-local means algorithm is not commutative, although the similarity measure is.

The implementation of the list mentioned before is quite straight forward. First, memory space is allocated for each pixel based on the size of the list and the size of one list element, which includes the weight, the associated value and if the slot of the list is used. Inserting one element into the list is described in Algorithm 4.2. One new value is taken into the list either if there is an unused slot in the list or if the smallest weight in the list is not as big as the one which is about to be inserted. In that case the new element replaces the list entry with the smallest weight.

Algorithm 4.2: Injecting values into a list, that keeps those with the highest weight w

Input: Weight w and the associated value e
Output: List $\mathbf{E}$ containing the best N weight/value pairs

$w_{min} \Leftarrow \text{inf}$;
$i_{min} \Leftarrow 0$;
for i *from 0 to* N **do**
    if $\mathbf{E}(i)$ *is empty* **then**
        $\mathbf{E}(i)_w = w$;
        $\mathbf{E}(i)_e = e$;
        return;
    end
    if $\mathbf{E}(i)_w < w_{min}$ **then**
        $w_{min} = \mathbf{E}(i)_w$;
        $i_{min} = i$;
    end
end
$\mathbf{E}(i_{min})_w = w$;
$\mathbf{E}(i_{min})_e = e$;

In the forthcoming sections, the size of the neighborhood K and similarity area S are squares centered around one pixel. The actual size is then the number of pixels in one axis-aligned direction as can be seen in Figure 4.2. Hence, a neighborhood K of size 2 is a square of 5x5 with 25 pixels and one of size 3 is a square of 7x7 with 49 pixels.

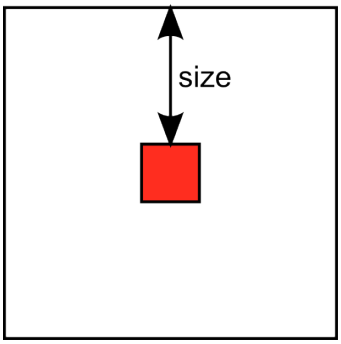


Figure 4.2: Size notation of the neighborhood K and similarity area S. The red square marks the center pixel.

CHAPTER 5

Motion-Vector Technique

The first proposed technique is the motion-vector technique. It finds optical flow based on two fully rendered color images and generates motion vectors for each pixel. From that, the coloring of the missing frame can be obtained. This technique works similar to what is done in upsampling using video encoding (described in Section 3.3), but generates motion vectors not just for a block of pixels, but for every pixel. The detection if two blocks are similar is done in the same way, the non-local means algorithm detects similarity between two pixels, but in the case of video encoding, the blocks themselves are the similarity areas. The selection of the best fitting location for the video encoding is (as described in Section 3.3) a search for the minimal difference value, which is the inverse operation of the non-local means search for the maximal similarity. However, the key aspect is that the motion vectors are not the same within a block of pixels when the non-local means algorithm is applied. This technique originates in the iterative search approach by Yang et al. [YTS+11], but in contrast to that approach, the proposed technique does not use the motion vectors from the object movement, but generates them based on the color information. This counters the problems of Yang et al. [YTS+11] with transparent and specular objects, where the movement of image features resulting from those objects is incoherent with what their motion vectors suggest.

5.1 Algorithm

This technique takes two B-frames (as seen in Figure 5.1a) and generates motion vectors by applying the non-local means algorithm (the maximum-detection variant as in 4.2) to the color information of those two B-frames as shown in Figure 5.1b. The next step of this technique is to iterate over all pixels and search for the sources in the B-frames indicated by the motion vectors as seen in Figure 5.1c. To do so, two different methods are used in this master thesis and compared to each other:

- **Iterative search:** This method was described by Yang et al. [YTS+11] and is explained in Section 3.2.1. As suggested by the same authors, this is done in both directions (forwards and backwards in time) to enhance results. If non of the two iterative searches for a pixel succeed, a heuristic to find the color for that pixel has to be employed. One reason for such a failed search can be inconsistent motion vectors generated by the algorithm. Another reason, as stated by Yang et al.[YTS+11], can be thin and fast-moving geometry. Since the scene-assisted approach of Yang et al.[YTS+11] provides reasonable results for pixels with a failed iterative search, the motion-vector technique uses that method as a fall-back. To avoid this problem at all, the method, described next can be used.

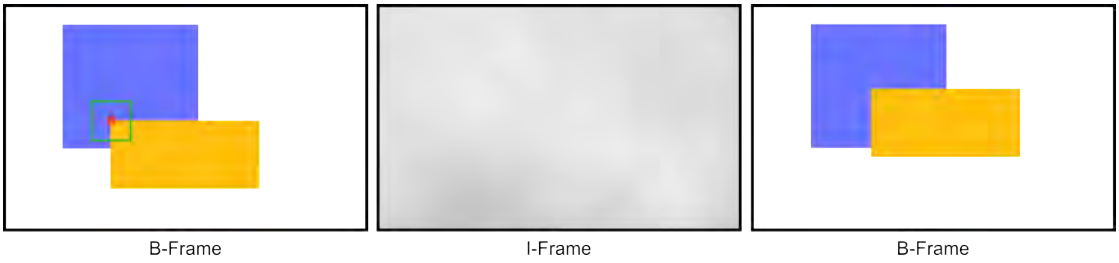
- **Brute force:** A search for the best-fitting motion vector is conducted in the neighborhood area (as defined by the non-local means algorithm) around one pixel. The best-fitting motion vector is the one that, when added to source pixel and multiplied with the time factor, comes closest to the target pixel's coordinates. The time factor describes the relative position of the I-frame on the time-line between two B-frames. Compared to the iterative search, this brute-force method needs to check more pixels, but does not rely on convergence of the motion vectors. It has to be mentioned that such a brute-force approach is only possible because the neighborhood area of the non-local means algorithm determines the maximal extent of the motion vectors.

In the following section, the function $\xi(a, b)$ to calculate the difference between two pixels is described, afterwards the configuration of the non-local means algorithm is explained, followed by a comparison and discussion of the results.

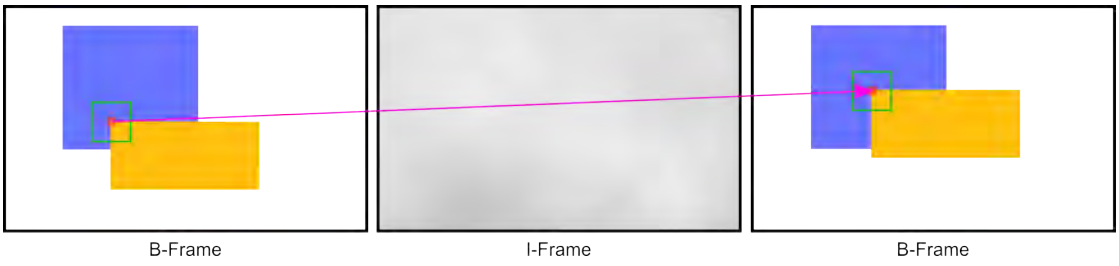
5.2 Similarity measure

As mentioned the motion-vector approach works by comparing the color values between two pixels a and b. Because $\xi(a, b)$ has to return just a single scalar (as described in Section 4.3), a mapping from the color components to that single value has to be found. The implementation used throughout this thesis has three color components: red, green and blue. To put this into a more general context, the amount of color components of one color representation is C. A color is then defined as similar to another, if all color components are somewhat similar to each other. Therefore, the product of all the color component's differences is used to make up the similarity measure.

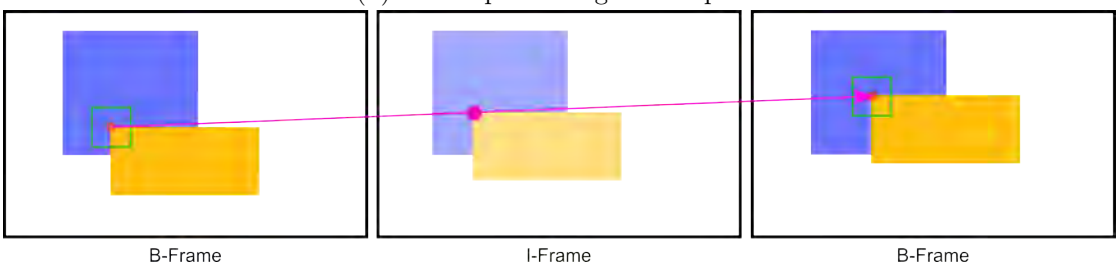
Since the similarity measure returns zero when the two pixels are not similar to each other, and the difference between two values of a pixel is zero when they are similar, a corresponding mapping of one scalar has to be used to achieve the desired return value of 1. A normal distribution function with a mean of zero can be such a mapping. Because only the most similar pixels, those with the highest values, are used the upsampling in the end, the actual value of the variance for such a normal distribution does not matter. Equation 5.2 shows a mapping of value x with the normal distribution function using the mean m and the variance v.



(a) 1^{st} Step: Starting point



(b) 2^{nd} Step: Finding similar pixel



(c) 3^{rd} Step: Finding motion vectors and coloring

Figure 5.1: The motion-vector technique in pictures.

$$\phi(x, m, v^2) = \frac{1}{v\sqrt{2 * \Pi}} e^{-\frac{1}{2}(\frac{x-m}{v})^2} \tag{5.1}$$

Again, because only the highest values are of interest to do the job, the fraction before the exponential component in Equation 5.2 can be disregarded. The resulting similarity measure for the motion-vector technique can be seen in Equation 5.2, whereas $c_i(p)$ returns the value of the particular color component i of the pixel p.

$$\xi(a, b) = \prod_{c_i=1}^{C} \phi(c_i(a) - c_i(b), m, v) \tag{5.2}$$

As mentioned before, the mean value m of the normal distribution has to be zero for the difference of two color components. The variance v is set to 0.25 in the actual implementation of the algorithm due to the range of color values of the rendering software at hand.

It has to be noted that a conversion to luminance would result in a loss of dimension, potentially increasing the possibility of an error. To be more specific: The proposed similarity measure returns a small value when one of the color components is significantly different. Calculating the luminance before or after the subtraction in Equation 5.2 would not have that property.

CHAPTER 6

Pixel-Similarity Technique

The second proposed technique is the pixel-similarity technique. It colors the pixels of the I-frame by looking at its surface attributes and searching for pixels in the B-frames with similar attributes. This technique is based on the scene-assisted approach introduced by Yang et al. [YTS+11], as it also requires a rendering of the I-frame without the shading information.

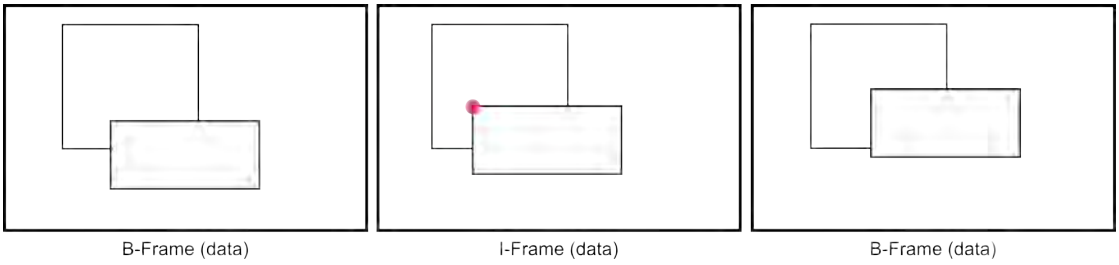
6.1 Algorithm

First, a rendering of the I-frame without any shading but with additional information for every pixel is generated, a so-called *stubby frame*, as seen in Figure 6.1a. A search for surface points which are expected to yield similar shading results is performed afterwards using the non-local means algorithm, as seen in Figure 6.1b. Then, pixel colors from the B-frame are picked (Figure 6.1c) based on the similarity of the pixel's additional data.

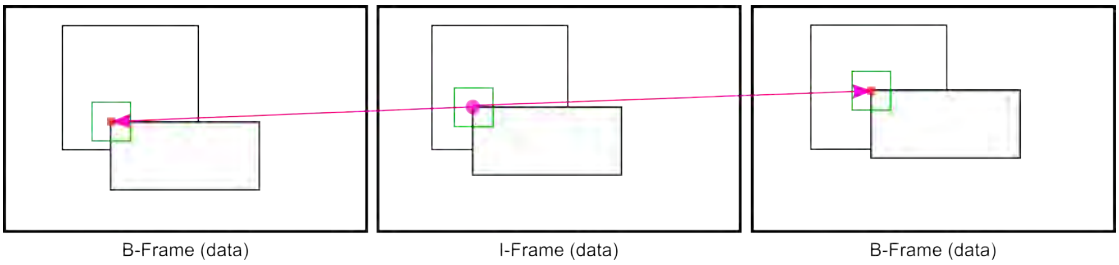
This technique uses the non-local means algorithm (as in Section 4.1) to get the best match in similarity. The selection of the similarity measure between two pixels is described in the following section, which is followed by a discussion about the right parameters for the algorithm and concludes with an in-depth evaluation of the results and comparison to the ground truth results.

6.2 Similarity measure

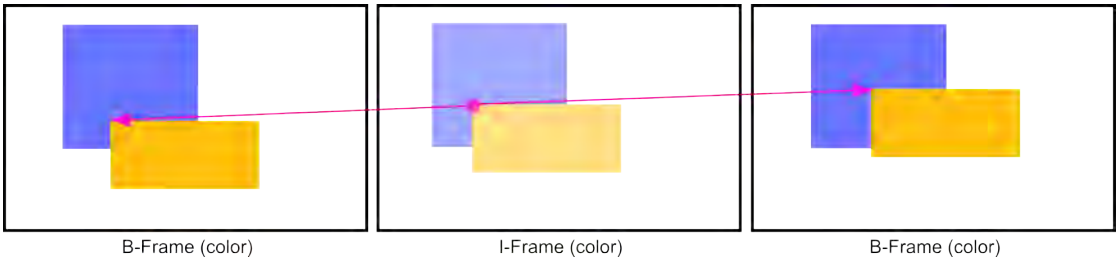
The similarity measure for of this technique is not as straight-forward as the motion-vector method, since it has not yet been defined which data of the I-frame pixels should be used. The search for such a metric is described in this section and separated into four components as explained in the following section.



(a) 1^{st} Step: Starting point at the stubby frame



(b) 2^{nd} Step: Finding similar pixel according to the data



(c) 3^{rd} Step: Coloring based on the similar pixels

Figure 6.1: The pixel-similarity technique in pictures.

6.2.1 Surface Type Separation

The first step in finding the similarity measure $\xi(a, b)$ is to identify the types of surfaces that the pixels a and b represent. The type of surface point in this context is its way of redirecting light, i.e., if the incoming light is reflected above the surface (specular or diffuse reflection) or underneath the surface (refraction). This separation is depicted in Figure 6.2.

Similar to the final similarity measure $\xi(a, b)$ (for pixels a and b), a function to compare two types of surfaces $\psi(a, b)$ returns zero (for different types) or one (for similar types). This relates to a similarity measure as seen in Equation 6.2.1, where $\xi_{above}(a, b)$ is the similarity in reflectance and $\xi_{under}(a, b)$ as the one in refractance.

$$\xi(a, b) = \psi(a, b) * \xi_{above}(a, b) * \xi_{under}(a, b) \tag{6.1}$$

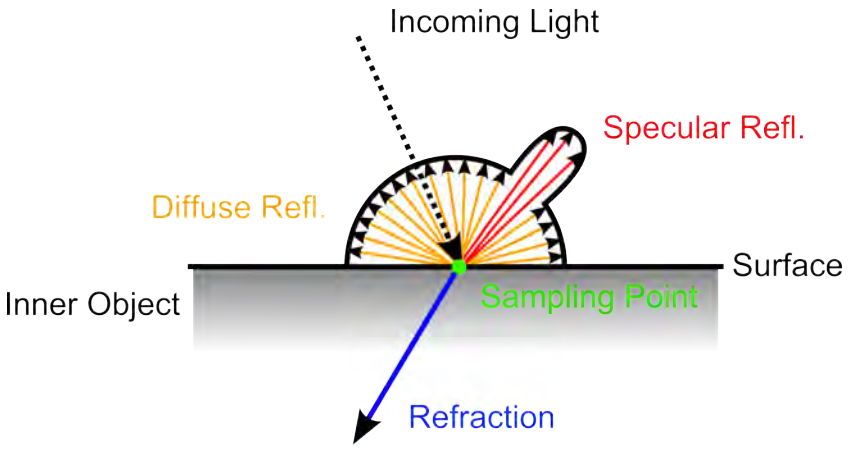


Figure 6.2: Different types of light propagation on a surface point.

In addition to comparing reflective and refractive properties, one may also want to differentiate which kind of reflection is happening above the surfaces. In the case of a Phong-BRDF (as it is used in the implementation) there are two kinds of reflection: diffuse and specular, which do not exclude each other, but are more or less present, depending on the specific surface. The Phong-BRDF is described in Formula 6.2.1 as part of the rendering equation as seen in Section 2.2.

$$b(x, \omega_o, \omega_i) = (n(x) \cdot \omega_i)\, d + \frac{k+2}{2\pi}(r(\omega_i, n(x)) \cdot \omega_o)^k s \tag{6.2}$$

where the diffuse factor d and the specular one s are limited to $d + s \leq 1$ and

- $n(x)$: is the surface normal vector at point x
- $r(\omega_i, n(x))$: is the reflection vector at point x for the direction of the incoming light ω_i.
- k: is the specular exponent, which describes the shape of the lobe.
- $v \cdot w$: the dot-product between two vectors

The resulting similarity measure for refractive surfaces and Phong-Reflectance is described in Equation 6.2.1. The three required measures (diffuse $\xi_{diff}(a, b)$, specular $\xi_{spec}(a, b)$, refractive $\xi_{under}(a, b)$) are described in the following sections.

$$\xi(a, b) = \psi(a, b) * [\mathbf{d} * \xi_{diff}(a, b) + \mathbf{s} * \xi_{spec}(a, b)] * \xi_{under}(a, b) \tag{6.3}$$

In the actual implementation of this master thesis, the type of surface can be identified via the variance of the outgoing ray bounces (since a ray-tracing algorithm is used). Using the factors of the underlying BRDF is another way of doing so.

The implementation identifies a surface as reflective or refractive, based on which variance actually exists. Determining if a reflective surface is specular or diffuse is also done using the variance: If the variance of the reflective bounce is maximal (over the entire hemisphere), the surface is considered to be diffuse. When the variance is minimal (converges to zero), the surface is of specular nature.

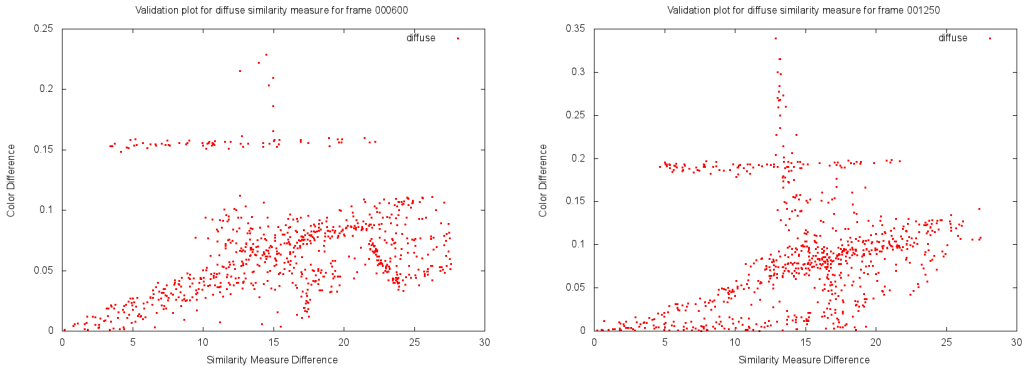
6.2.2 Diffuse Similarity Measure

To compare two diffuse surface points, an analysis of how the outgoing light is composed has to be done. For a diffuse reflection, all incoming light is weighted equally. So when two points are given with the same incoming light from all directions, the reflected color depends on the specific texture color at that point. To put this into a more general context, the texture color is the way of weighting the incoming light at that specific point. The assumption of all incoming light being equal is then more likely to be true when two points are close to each other.

Since the texture color is also bound to the surface point, one can assume that the outgoing light (the color at the pixel) is the same when the position is the same as well. This correlation of color and position for diffuse surface can be seen in Figure 6.3a, which was created by analyzing the diffuse colors of one frame in scene 1. The same is done for a frame in scene 2, as seen in Figure 6.3b. Within those plots, one point describes the difference between a pair of two points (one hand-picked and the others are randomly assigned to it). The horizontal axis show the value of the similarity measure for those two points and the vertical axis depicts the difference in color. The key point of those plots is to show that the area for when the similarity measure goes to zero (hence the points are qualified as equal) and the color difference is not zero (hence the final shading is not the same) is avoided. If there are points in that area, this would relate to the similarity measure qualifying two pixels as equal shaded, when they most definitely are not. Both of the Figures 6.3a and 6.3b show a spreading of the points along a line going from the bottom left corner to the right side going up. The vertical spreading in both figures shows pixels which are all of different color, but get the same measure value. The horizontal spreading shows pixels with the same color but different similarity values. But in the end, these observed features of the plot do not disqualify the measure, since there are no points in the above-mentioned area.

Since the proposed technique does not compare two points at the same location on the time-line, but on different ones, the possible movement of those points has to be taken into account. This positional change is part of the data available for rendering the stubby frame and leads to the measure for comparing diffuse surfaces as seen in Equation 6.2.2. In that formula, $pos(x)$ retrieves the world position of pixel x and $mov_{pos}(x, Y)$ returns the change of the position x to the point in time where the frame Y is located. Again, ϕ is a mapping via normal distribution (see Equation 5.2), and the variance of 0.5 does not matter since the pixel-similarity technique tries to find the maximal value of the pixel correlation.

$$\xi_{diff}(a, b) = \phi(pos(a) - (pos(b) + mov_{pos}(b, A)), 0, 0.5) \tag{6.4}$$



(a) Frame 600 (b) Frame 1250

Figure 6.3: Validation plots of the similarity measure for diffuse surfaces.

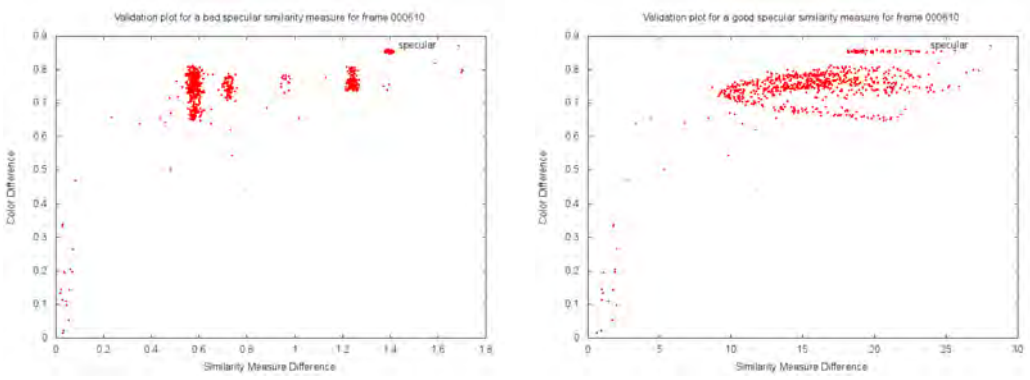
6.2.3 Specular Similarity Measure

In contrast to diffuse reflection the incoming light in the specular reflection is not equally weighted across all directions. This is because for this kind of reflection, a (small) specific part of all incoming light is the most important contributing factor to the final color of the pixel. Similar to the diffuse reflection, texture color also plays a role in the resulting outgoing light.

Taking the unequal weighting into account, one can argue that the direction where most of the light contributes to the final result is one of the key factors for a similarity measure. However, a lot of surface points in the image can have the same direction, which is most important for the resulting color. What those points might not have in common is the actual point in space from where the most contributing light is coming from. The correlation of that position and final coloring is validated by Figure 6.4b, which again is a comparison of colors and those particular position within a frame of scene 1. That the plain direction is not a good measure for the needs of the proposed technique is seen in Figure 6.4a. The reason for the latter plot disqualifying the measure are the points, which are closest to the vertical axis (on the most left side) with others of more similar coloring being not.

It has to be mentioned that when taking the position of the most contributing surface point into account, the assumption is made that the incoming light from this location stays the same over time and, more importantly, for every direction it is contributing light to. Hence, the assumption is that this particular point has a mostly diffuse reflection. When this is not the case, the similarity measure with the most contributing point in space does not hold up to its task. Such a failed assumption happens in the case of a refraction, as discussed in the following section.

The resulting similarity measure for specular reflection can be seen in Equation 6.2.3.



(a) Disqualified measure (b) Validated measure

Figure 6.4: Validation plots of the similarity measure for specular surfaces done for frame 000610. Figure a uses a measure using direction of the reflection and Figure b utilizes the point from where the most contributing light comes from.

Compared to the diffuse reflection, this measure does not contain a correction of the position over time, resulting in another restriction of that measure.

$$\xi_{spec}(a, b) = \phi(pos_{bounce}(a) - pos_{bounce}(b), 0, 0.5) \tag{6.5}$$

6.2.4 Refractive Similarity Measure

A refraction alters the path of the light when it interacts with a transparent object. Light is then either refracted or reflected, based on the incident angle and the refractive indices of the materials. Furthermore, when the light is inside an object, internal reflections can occur, causing the light path to bounce inside the object. The intricacies of refractions are described in Section 2.2 in more detail.

Hence, from an analytical stand point, a measure similar to the one for specular surfaces will not hold up to its task, even with the same assumptions made. Taking the point that the light path hits after it leaves the transparent object could counter that problem, but leads to another one: This point might not be as easy to extract from then underlying rendering algorithm, contradicting one of the key premises of this thesis: the independence of the software used to generate the images. A ray-tracing algorithm, as it is used for this thesis, might not have the issue of extracting the point of first surface interaction after the transparent object. But to highlight the problem of certain surface data not being available, the point of the first interaction after the transparent object is not used further on.

To actually find a good similarity measure with the given data points, one can employ another (non-analytical) way of finding the correlation between the data of a point and

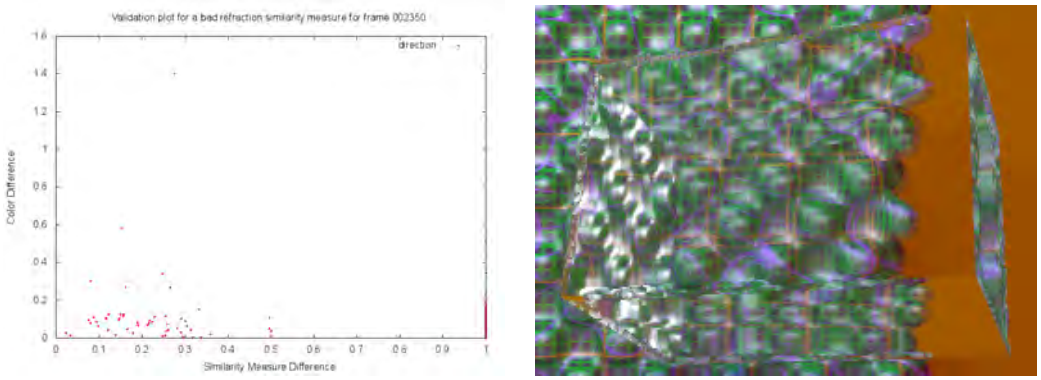
its color: Plotting each possible measure for the available data and finding the one which shows the best results. A selection of that process can be seen in Figure 6.5.

From Figure 6.5 it can be seen that in spite of the above statement, a measure using the position of the most (and only) contributing point in space delivers the 2nd best results in terms of mean squared error. Only a measure similar to diffuse reflection performs better, but arguably results into more 'blurring effect' on the surface of the transparent cube as well as false edges (those behind the surface), thus reducing the visual quality of the measure (see Section 1.5).

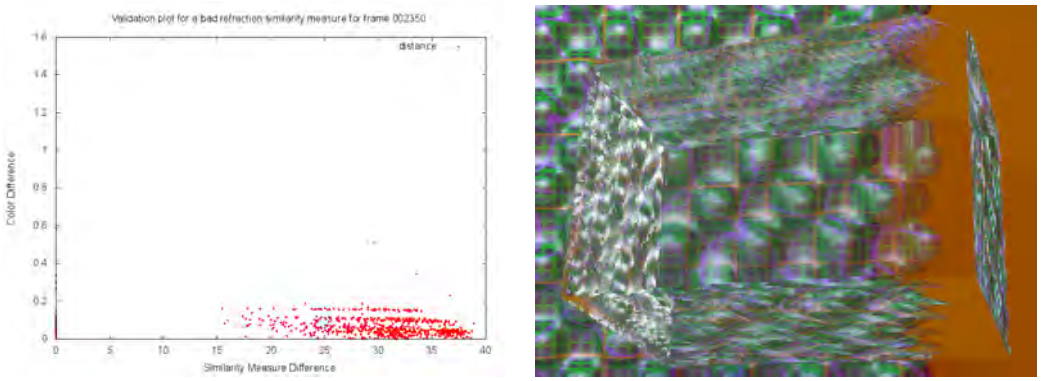
Therefore, the measure for refractive surfaces is either the one using the same method as for diffuse surfaces (Equation 6.2.4) or the one which is similar to the one for specular surfaces (Equation 6.2.4). For the final upsampling and further testing, the variant with the contributing point is used. This decision is based on the above statements. However, Figure 7.13, which shows the mean squared errors of the upsampling, also features the values of the other option.

$$\xi_{refr}(a,b) = \phi(pos(a) - (pos(b) + mov_{pos}(b,A)), 0, 0.5) \tag{6.6}$$

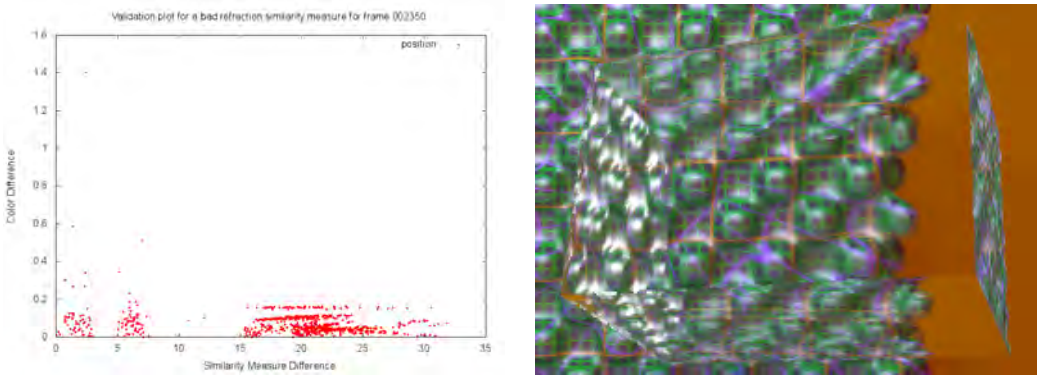
$$\xi_{refr}(a,b) = \phi(pos_{bounce}(a) - pos_{bounce}(b), 0, 0.5) \tag{6.7}$$



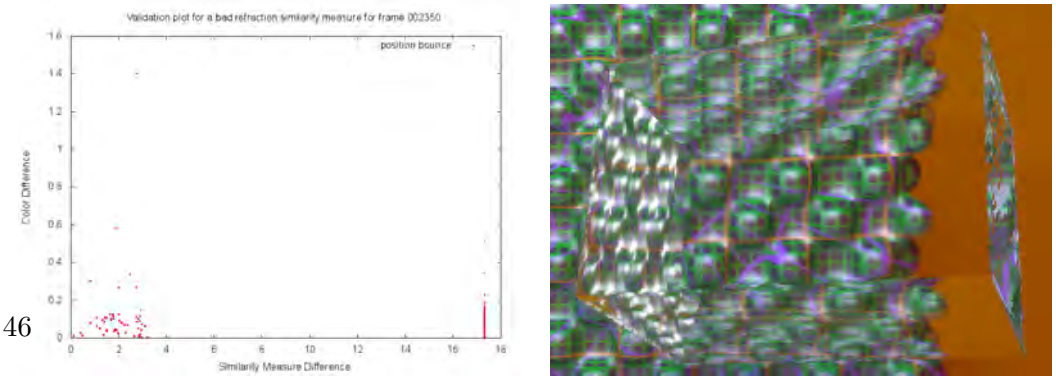
(a) Measure using the direction of the refraction. MSE: 0.01308331



(b) Measure using the distance of the refraction. MSE: 0.01419275



(c) Measure using the surface position. MSE: 0.00868144



(d) Measure using the position of the refraction next surface interaction. MSE: 0.01095594

Figure 6.5: Validation plots of the similarity measure for transparent surfaces done for frame 002351. The figures on the left show the similarity values for two points on the glass cube. On the right side is the result of using that measure.

CHAPTER 7

Results and Comparison

Before talking about the results of the actual techniques, the test image sequence is described in the first section, followed by the an explanation on how to find the parameters for the proposed techniques to achieve the best upsampling of that sequence.

7.1 Test Image Sequence

The test sequence (as seen in Figure 7.1) used to evaluate and compare the algorithms of this master thesis is one minute long and features a virtual room with various objects and materials. During the test sequence, the camera zooms in on three different sets of objects, which basically makes up three scenes. Those scenes all serve a purpose in terms of investigating the method's quality with regard to the definitions of quality in Section 1.5.

The scenes of the image sequence contain the following features:

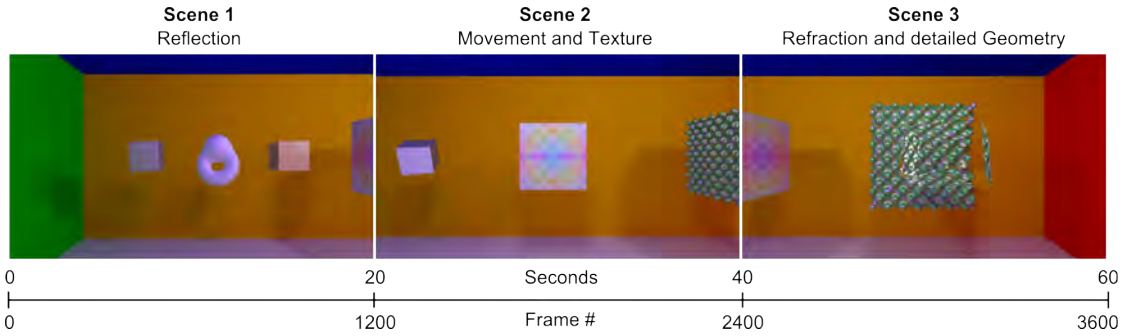


Figure 7.1: This figure shows the test sequence along the time-line with an image example for each scene.

- **Scene 1 - Reflection:** This scene shows rotating semi-reflecting cubes with a diffuse object in between. The purpose of this scene is to see if the methods (proposed and those from previous work) can handle rapid color changes on specular surfaces.

- **Scene 2 - Movement and Texture:** In this scene, a cube with a diffuse texture featuring a high color variation changes its size and position. This scene determines the method's performance in terms of edge location and texture variety when objects move in various ways.

- **Scene 3 - Refraction and Detailed Geometry:** This scene features a fast rotating transparent cube in front of a cube with a distorted surface and a diffuse grid-texture. The last scene represents the worst-case scenario for previous upsampling techniques as well as the proposed ones, since the movement of the optical flow is higher than in the other scenes, due to the refraction of the light and rotating speed of the glass cube (which is higher than those in scene 1). Furthermore, the geometry is more detailed than in any other scene.

The camera path is generated via a Catmull-Rom spline interpolation of given points. One segment of the camera path between the points P_0 and P_1 is calculated using two control points P_{-1} and P_2. The tension of the spline is set to 0.5.

7.2 Parameter Finding

The one thing missing from running the proposed techniques is to find the parameters of the non-local means algorithm. This search for the optimal parameters is described in the following subsections. To be more specific: The values for the neighborhood size, the similarity area and the list size of the non-local means algorithm for the motion-vector and the pixel-similarity technique are discussed. All tests for the motion-vector method are performed using the brute-force variant to detect the motion vectors to fill the missing pixel location. Testing for the pixel-similarity technique is done by using the similarity measure with the contributing point, as described in Section 6.2.

7.2.1 Neighborhood Size

The neighborhood size, which is the search radius, has to be equal to the maximal movement of one pixel across the passed time. In case of the motion-vector technique, this is the movement of a pixel across two B-frames. Since the pixel-similarity technique does not try to find the movement of a pixel across two B-frames, the neighborhood size has to be the maximal movement of one pixel from an I-frame to a B-frame.

Unfortunately, such a maximal movement cannot be exactly computed without applying the algorithm across the entire image size. This is due to the nature of light transport, which can not be predicted analytically (as seen in Section 2.2 and 2.4). An approximation to the maximal movement of one pixel can be taken from the motion

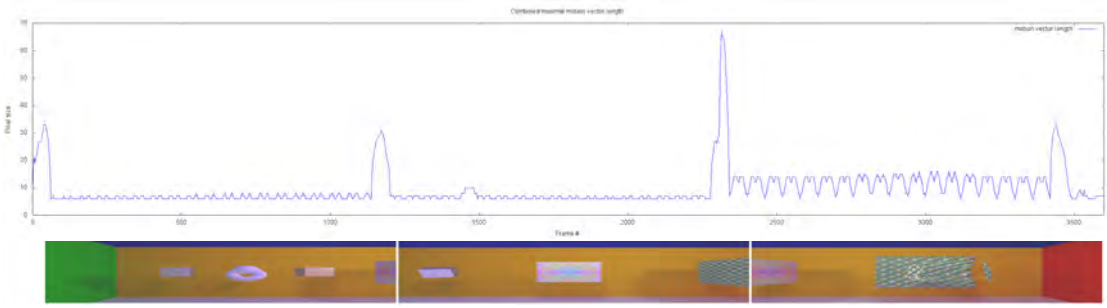


Figure 7.2: Values of the biggest dimension in motion vectors for each pair of frames across the entire test image sequence.

vectors generated by the movement of the underlying geometry, which are used in the approach of Yang et al. [YTS+11], described in Section 3.2.

Motion-Vector Technique

The approximation for the motion-vector technique is the sum of the maximal motion vectors of two adjacent frames for an I-frame. Figure 7.2 shows the biggest motion vectors for each pair of ground-truth frames of the test sequence, which the rendering software of this master thesis generated. Across the sequence, the biggest motion of a pair of B-frames (for an upsampling from 30 to 60 fps) is 67 pixels. As can be seen in Figure 7.2, not every frame has such a high expected motion. This can be used to reduce computational time for the entire sequence. Calculating the expected required neighborhood size for each I-frame independently results in reduced computation time for many I-frames, as can be seen in Figure 7.11b. To enhance this optimization, a minimum value for the neighborhood size of 6 is used to generate the upsampled sequence as well as an addition of 20% to what the motion vectors would suggest.

Pixel-Similarity Technique

In contrast to the motion-vector technique, the approximation for the pixel-similarity technique only requires the single maximal motion vector (as seen in Figure 7.3) length of both adjacent frames, not the combined one. For the test image sequence, this corresponds to a value of 34. Again, similar to the motion-vector technique, an optimization is made to reduce computational time. Therefore, a minimum value for the neighborhood size of 6 is chosen to generate the upsampled sequence as well as an addition of 20% to what the motion vectors would suggest.

7.2.2 Similarity Area and Kernel

To discuss this parameter setting, a few definitions have to made before-hand: A correct pixel is the ground-truth target pixel (in one frame) for one source pixel (in another

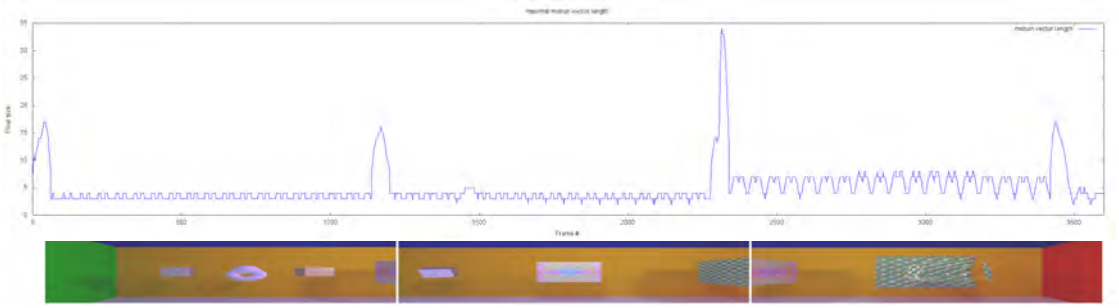
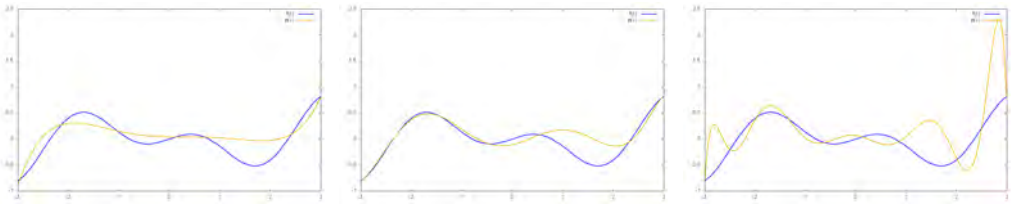


Figure 7.3: Values of the biggest dimension in motion vectors for each pair of frames across the entire test image sequence.



(a) Under-fitting (b) Best fitting (c) Over-fitting

Figure 7.4: Polynomial fit for the function $f(x) = cos(2x) * sin(x/3)$. Figure (a) shows a fitting with with a degree 5 polynomial resulting in under-fitting, Figure (b) uses a 9-degree polynomial resulting in the best fit and (c) shows over-fitting with a polynomial degree of 11

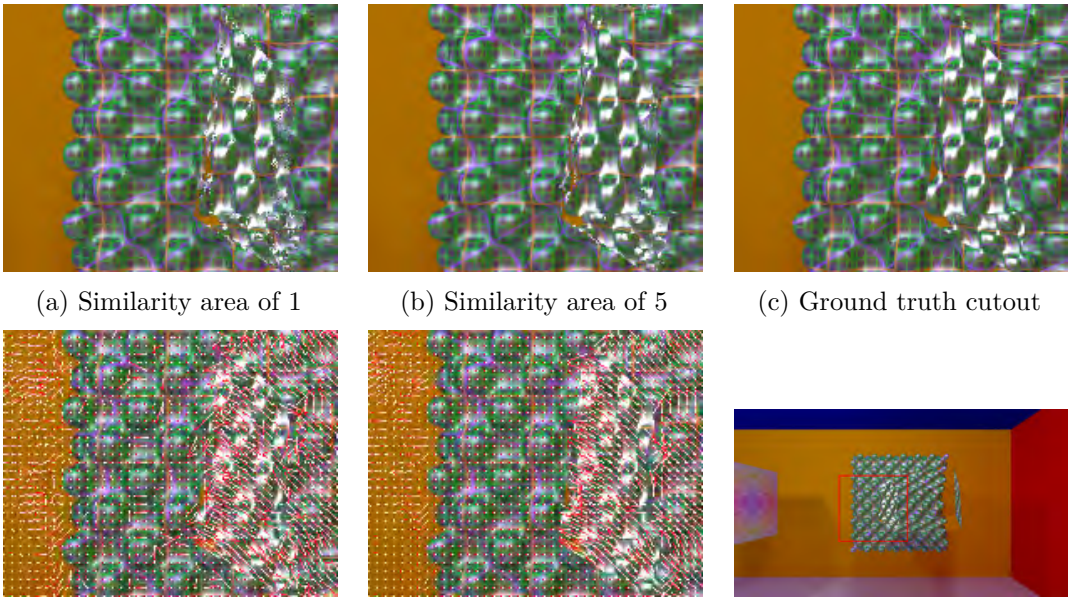
frame). A false pixel then is a target pixel which is not the ground-truth pixel. In other words: a source pixel moves to the correct pixel in another frame.

The size of the similarity area affects the algorithm as follows: The area has to be big enough so that the non-local means algorithm can detect differences in the area between the correct pixel and others. On the other hand, the similarity area should not be too big, so that there is still room for a change (caused by motion) within the area of the correct pixel.

Overall, this problem of finding the optimal area size is similar to choosing the degree for polynomial fitting (discussed briefly in Section 2.4). The size of the similarity area relates to the number of degrees chosen to make up the polynomial. Hence, the right value between over- and under-fitting has to be found. Figure 7.4 depicts the problems of a polynomial fit.

Figure 7.5 shows the results for the motion-vector technique when the similarity area is too small to detect the correct motion vectors. Repetitive image features (such as the grid texture on the cube with the bumps) can enhance the problems.

Figure 7.6, on the other hand, shows the problems of a too large area for the motion-vector technique. Bigger similarity-area sizes increase the chance of the actual best fit for a source pixel getting a decreased similarity value. This can be due to extensive change



(a) Similarity area of 1 (b) Similarity area of 5 (c) Ground truth cutout

(d) False motion vectors (e) Better motion vectors (f) Entire frame

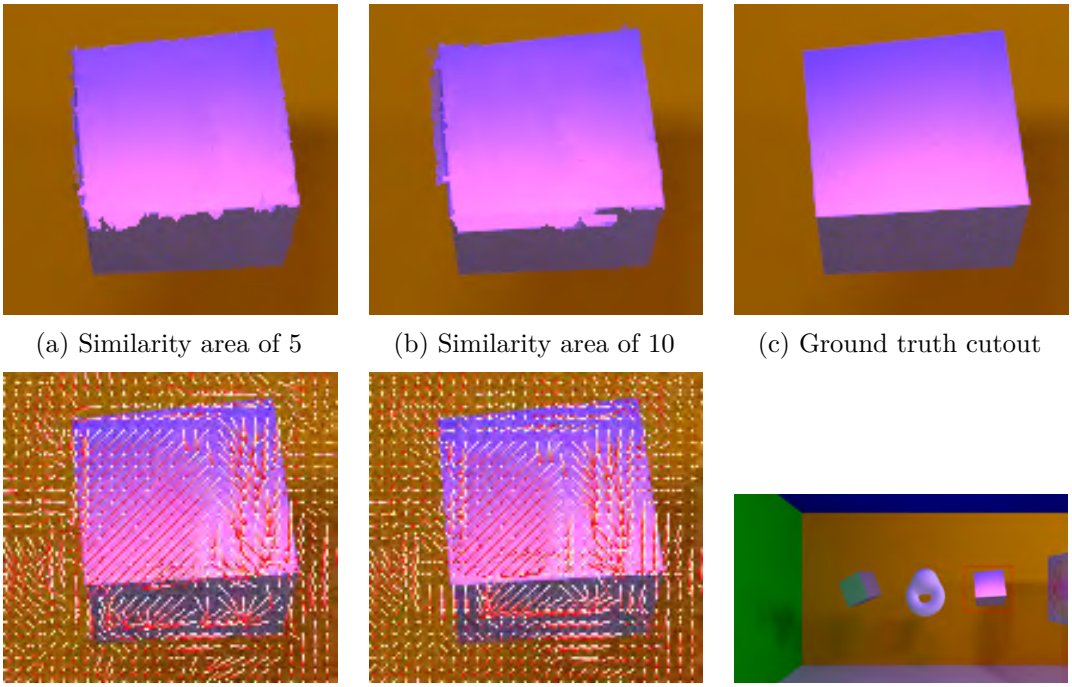
Figure 7.5: This figure shows the consequences of too small similarity-area sizes for the motion-vector technique. Figure (a) was produced with an area size of 1 and shows noise at the green stripes due to the failed matching. The noise around the edges of the glass cube is also a result of the small area size. In Image (b) the area size is 5, which results in no noise at the green lines as well as reduced noise at the edges of the glass. Figure (c) shows the ground-truth frame, Figure (d) shows the motion vectors (pointing from white to red) due to the small area size of 1, and Figure (e) shows better motion vectors generated with a similarity size of 5. Notice that in Figure (d) motion is detected on the bumpy cube, when there is actually none. And finally, Figure (f) marks the place of the cutout in the frame.

(motion) in the surroundings of the source pixel and the pixel of the actual best fit.

One way of finding the best value for the similarity area is simply by trial and error for a few exemplary frames of the sequence. The same also applies to determining whether a weighting by a Gaussian kernel should be applied or not. A Gaussian kernel in that context is $\mathbf{K}(s) = \phi(s, 0, S * 0.5)$, where S is the size of the similarity area. The multiplication factor of 0.5 is a result of initial trial-and-error testing. Other values between zero and one alter the results slightly depending on the similarity size, similar to the findings described below.

Motion-Vector Technique

Figure 7.7 shows tests with various sizes of similarity areas for two different frames. From this, it can be seen that the best similarity size varies depending on the frame, and the



(a) Similarity area of 5 (b) Similarity area of 10 (c) Ground truth cutout

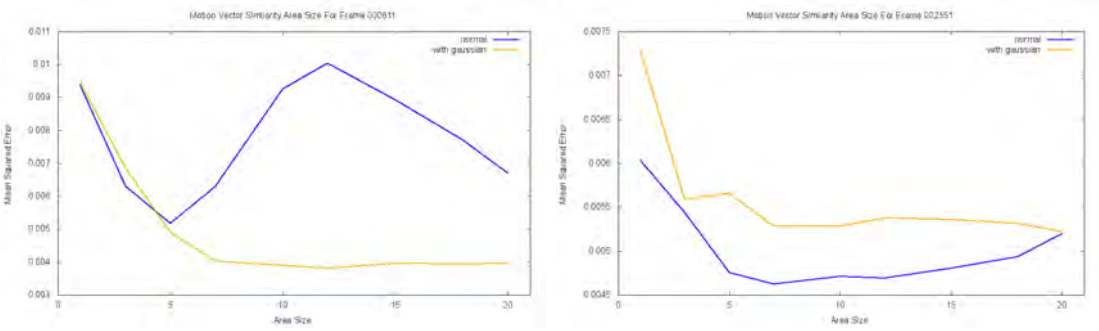
(d) Better motion vectors (e) False motion vectors (f) Entire frame

Figure 7.6: This figure shows the consequences of too big similarity-area sizes for the motion-vector technique. Subfigure (a) was produced with an area size of 5 and shows 'color bleeding' to some extend due to the inability of the algorithm to find a similar pixel. In Image (b) this effect is amplified due to an increased similarity size of 10. Figure (c) shows the ground truth frame, Figure (d) shows the motion vectors (pointing from white to red) for the similarity area of 5 and Figure e) shows false motion vectors due to the big area size of 10. Notice that the vectors in Figure (e) do not cross the edge, which stands in contrast to the objects movement downwards to the left. And finally, Figure (f) marks the place of the cutout in the frame.

same goes for the usage of the Gaussian kernel weighting. An adaptive decision for each individual I-frame, similar to the choice of the neighborhood area size, is not possible due to the lack of any heuristic that can determine the best configuration in advance. Based on the results above, the similarity area size of 5 without a Gaussian kernel is chosen to deliver the final results for comparison.

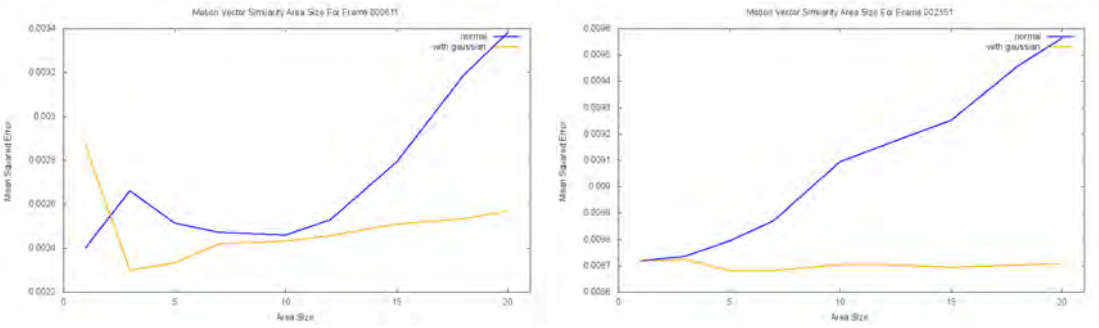
Pixel-Similarity Technique

Again, the actual values and whether a kernel weighting should be used can be found by trial and error. For the image sequence used in the master thesis, a value of 5 for the similarity area size and applied weighting with a Gaussian kernel function (see Equation 5.2) provides the best results. Figure 7.8 shows the tests performed.



(a) Tests for frame 611 (b) Tests for frame 2351

Figure 7.7: Tests for the size of the similarity area to use in the testing sequence. The blue line depicts the mean squared errors without a Gaussian kernel and the orange line shows a weighting of the area with the kernel.



(a) Tests for frame 611 (b) Tests for frame 2351

Figure 7.8: Tests for the size of the similarity area of the pixel-similarity technique to use in the testing sequence. The blue line depicts the mean squared errors without a Gaussian kernel and the orange line shows a weighting of the area with the kernel.

7.2.3 Maximal List Size

This section is about tuning the size of the list of best values for the adapted non-local means algorithm, as seen in Section 4.3.

Motion-Vector Technique

The maximal list size for the motion-vector technique is used for the reasons described in Section 4.3. To get the motion vectors for this technique, the values in the list are averaged. However, it can also happen that outliers make it into the list and therefore alter the result in a negative way. Because of this, the list size has to be limited to a specific amount.

A trial-and-error search for an appropriate list size for the test sequence showed that the list size of one (hence only the best result) delivers the most accurate results. The

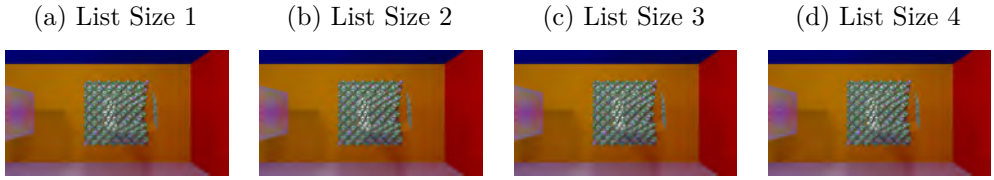


Figure 7.9: Results for different maximal list sizes when its values are averaged to make up the resulting motion vector. The number below the pictures are the mean squared errors of those images.

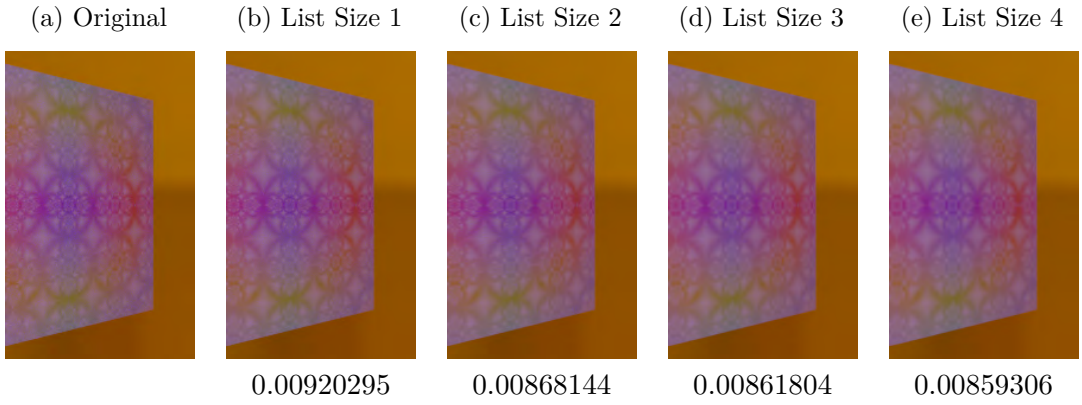


Figure 7.10: Results for different maximal list sizes for frame 2351. The number below the pictures are the mean squared errors of those images. Notice the increased blurring due to the greater list size.

tests shown in Figure 7.9 show that even small list sizes lead to false motion vectors and therefore to a worse upsampled image.

Pixel-Similarity Technique

Again, the optimal list size can be obtained via trial and error with the same problems in mind as for the motion-vector method. However, for the pixel-similarity technique, it is useful to take at least one pixel from the B-frame before and at least one from after the I-frame. The optimal maximal list size for the test sequence is in conclusion to the tests as seen in Figure 7.10, which shows decreased mean squared error for bigger list sizes, but also increased blurring. These two contradictions lead to the decision of using a list size of 2 for the final upsampling

7.3 Comparison and Evaluation

After finding the best values for the algorithm, the resulting images perform in the following way with regard to the quality definitions (as described in Section 1.5) and in comparison to the ground-truth frames.

7.3.1 Comparison with Ground Truth

The comparison with the ground-truth frames is done using the mean squared error measure.

Proposed Techniques

As can be seen from Figure 7.11, the motion-vector technique without the iterative image search has a slightly lower mean squared error than the one using it. However, for some frames the variant with iterative search outperforms the brute-force one. One such occasion in our test sequence is frame 2317, as seen in Figure 7.12. The reason for this is that the brute-force variant can choose a false positive as the best-fitting motion vector, whereas the iterative search variant has a chance to avoid a false positive due to the convergence towards the best solution. The iterative search performed better in 46 frames out of 1800 (3 percent).

The time it took to upsample one frame is on average 12.6% higher for the version without using iterative search on a Intel Core i7-4770K CPU with four 3.50GHz cores. Due to the overall better performance of the brute-force variant, it is chosen for the comparison with previous work.

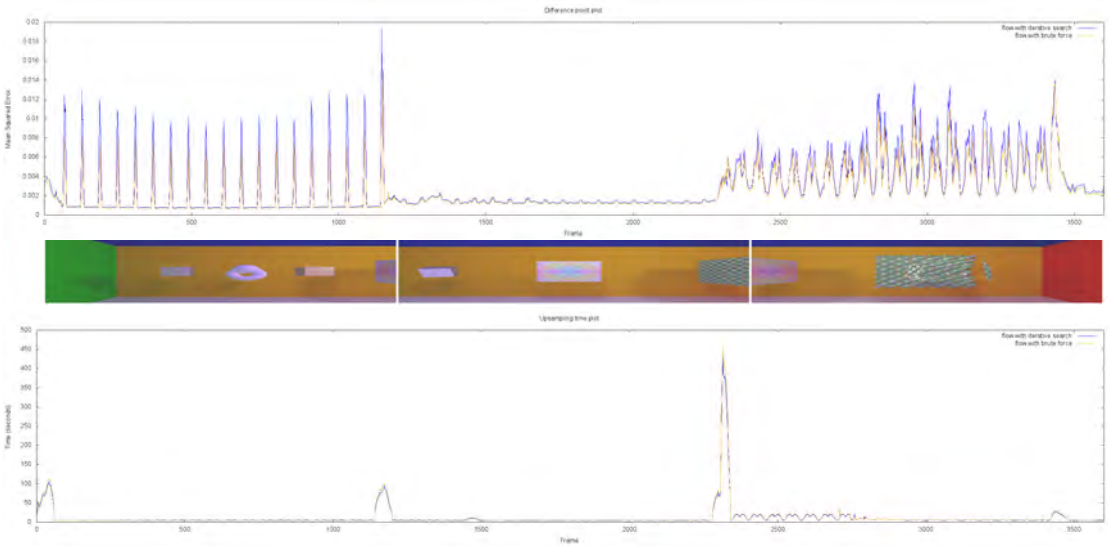
Figure 7.13 shows the results of the pixel-similarity technique. From that it can be seen that when it comes to the third scene, both variants of the similarity measure (using the position of the surface or the next surface interaction) perform quite similarly, with the exception that some peaks of the variant with the surface interaction are bigger and valleys are on some occasions lower.

An Intel Core i7-4770K CPU with four 3.50GHz cores was used to generate the times for upsampling with the pixel-similarity technique in Figure 7.13b. As mentioned before in Section 6.2, the variant with the refraction interaction for the similarity measure is better in terms of quality (described in Section 1.5). Therefore, it is chosen for the comparison with previous work.

Comparison with Previous Work

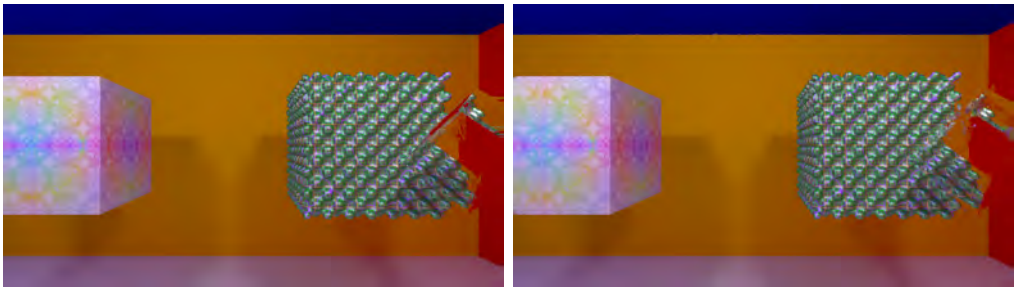
Figure 7.14 shows the comparison with previous work. However, the mean squared error of the upsampling using video encoding is not accurate, due to the quality loss of the compression. The plot displays that all techniques have problems when the cubes in scene 1 show a new side from one frame to another, which leads to the spikes there. As for the diffuse scaling and rotating cube with the detailed texture, all techniques seem to perform quite the same and the best compared to the other scenes. The motion-vector

(a) Mean squared error of both motion-vector technique variants.



(b) Time for calculating the missing I-frames of both variants.

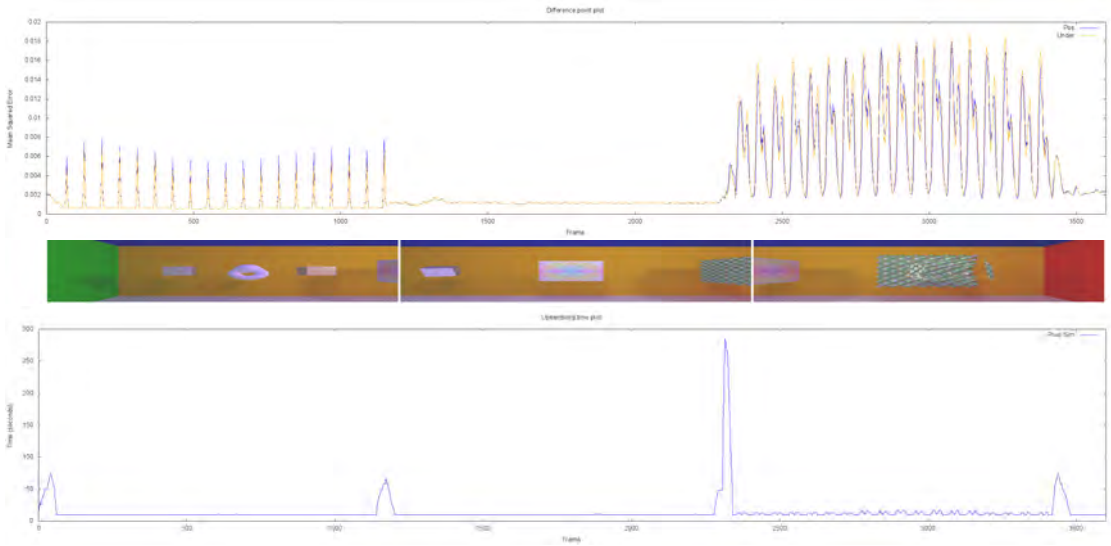
Figure 7.11: Figure (a) shows the mean squared error to the ground-truth frames of the motion-vector technique for each frame in the test sequence. The blue line is the variant with the iterative search and the orange on is the brute force variant Figure (b) shows the rendering times of both variants using an adaptive neighborhood size with a minimum of 10



(a) Iterative Search (b) Brute Force

Figure 7.12: Case of iterative search variant (mean squared error: 0.00391085) of the motion-vector technique outperforming the brute force variant (mean squared error: 0.00464652).

(a) Mean squared error of both pixel-similarity technique variants.



(b) Time for calculating the missing I-frames.

Figure 7.13: Figure (a) shows the mean squared error to the ground-truth frames of the pixel-similarity technique for each frame in the test sequence. The blue line is the variant with similarity measure using the position of the surface and the orange one uses that of the next surface-interaction of the refracted light Figure (b) shows the rendering times with an adaptive neighborhood size including a minimum of 10.

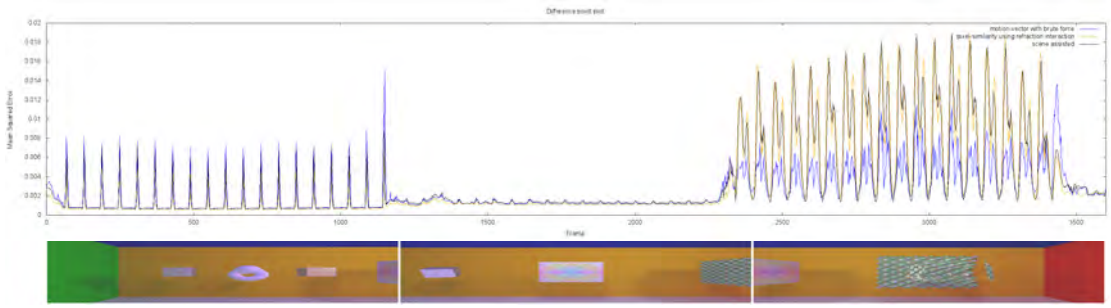


Figure 7.14: Mean squared error for the entire test sequence for the motion-vector technique (brute-force variant) as the blue line, the pixel-similarity technique (refraction interaction variant) as the orange line and the scene-assisten method proposed by Yang et al. [YTS+11] as the black line.

technique is significantly better than all the other ones when it comes to the transparent cube. The scene-assisted and pixel-similarity technique perform similar to each other with one out performing the other depending on the position of the cube.

7.3.2 Quality Performance

As stated in Section 1.5, the proposed techniques are aimed to handle several features. In the following subsections, the proposed techniques and previous work are compared against each other for each feature.

Edge location

As can be seen in Figures 7.15a and 7.15c, the **motion-vector technique** has troubles detecting the correct edges of rotating geometry. This is due to the nature of the non-local means algorithm, which is not rotation invariant. The same applies to some extent when an object undergoes scaling, as seen in Figure 7.15b and 7.15d. The scaling issues also arise when the object is translated along the depth axis of the camera. Another problem is when non-linear motion (in image space) occurs. Then the object's position is off compared to its ground-truth location, as seen in Figure 7.16, which highlights this problem via a small camera movement (which is generated via spline interpolation) resulting in an offset of just one pixel.

The edge location of the **pixel-similarity technique** is in general similar to the ground-truth frames as can be seen in the difference image of Figure 7.17. This is due to the partially rendered I-frame and the usage of the kernel for the similarity area, which emphases the center pixels of it. A slight blurring with surrounding colors can occur due to the maximal list size of two.

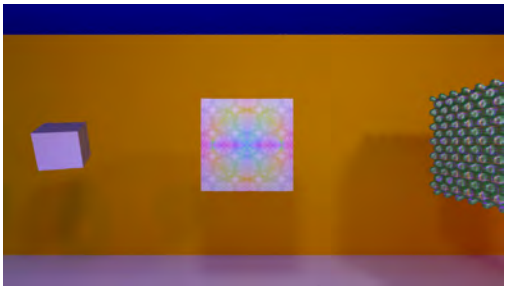
Similar to the pixel-similarity technique, the **scene-assisted technique** by Yang et al. [YTS+11] produces exact edges due to the rendering of the stubby frame. Again, slight inaccuracies occur due to oversampling as can be seen in Figure 7.18.

Upsampling with **video encoding** produces 'ghost edges' on the rotating cubes (as seen in Figure 7.19a and 7.19b), due to the inability of it to detect such movement. When it comes to scaling, slight steps along the edges are visible, as can be seen in Figure 7.19c and 7.19d.

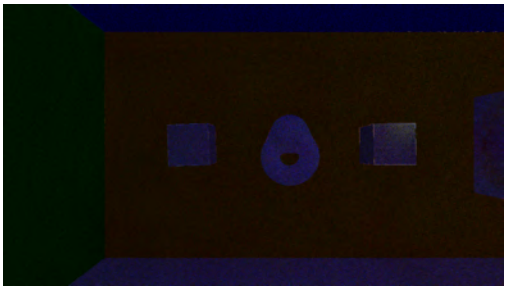
Texture variety

Detailed surfaces are beneficial for the **motion-vector technique** for small movements, as it is more likely to find the correct target location of a pixel. Figure 7.20 depicts the opposite case and shows that small noise (which can be seen as some sort of texture variety) can lead to detection of motion when there is none. In our test sequence, the actual coloring is only affected occasionally by this problem, because for the most part, the image features stay the same along the false motion vectors.

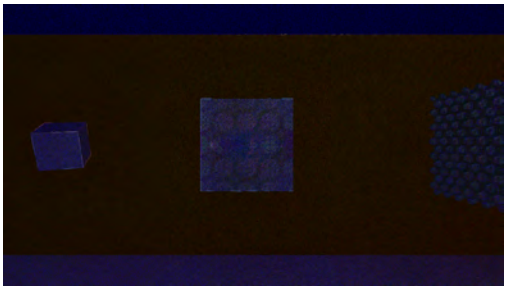
(a) Rotation



(b) Scaling



(c) Rotation - difference to ground truth



(d) Scaling - difference to ground truth

Figure 7.15: Problems of the motion-vector technique when detecting edges. The images in the upper row show the results and those in the bottom show the difference to the ground-truth frames. The 'border-effect' in the latter images highlight the problems of the motion-vector technique with correct edge location.

The **pixel-similarity technique** preserves detailed surface features, as can be seen in Figure 7.17. Again, a slight blur with the nearby colors happens due to the maximal list size.

Again, the **scene-assisted method** performs similar to the pixel-similarity technique and preserves the details of the textures as seen in Figure 7.18.

Due to the compression, the upsampling with video encoding blurs the textures and therefore reduces the visual quality as seen in Figure 7.19c.

Specular surfaces

The **motion-vector technique** has no problem detecting the correct movement of reflective image features when the coloring is similar across frames, as seen in Figures 7.21a to 7.21e. However, when one side of the mirror-like cubes disappears due to its rotation (Figures 7.21f to 7.21i), the algorithm fails to find the correct motion vectors around the edges, leading to the previously mentioned color bleeding (see Section 7.2.2). Furthermore, the iterative-search variant produces some sort of 'tearing effect' (as can be seen in Figure 7.21j), which the brute force does not. The reason for this 'tearing effect' is the inconsistency of the motion vectors around the edges, which leads to false convergence of the method.

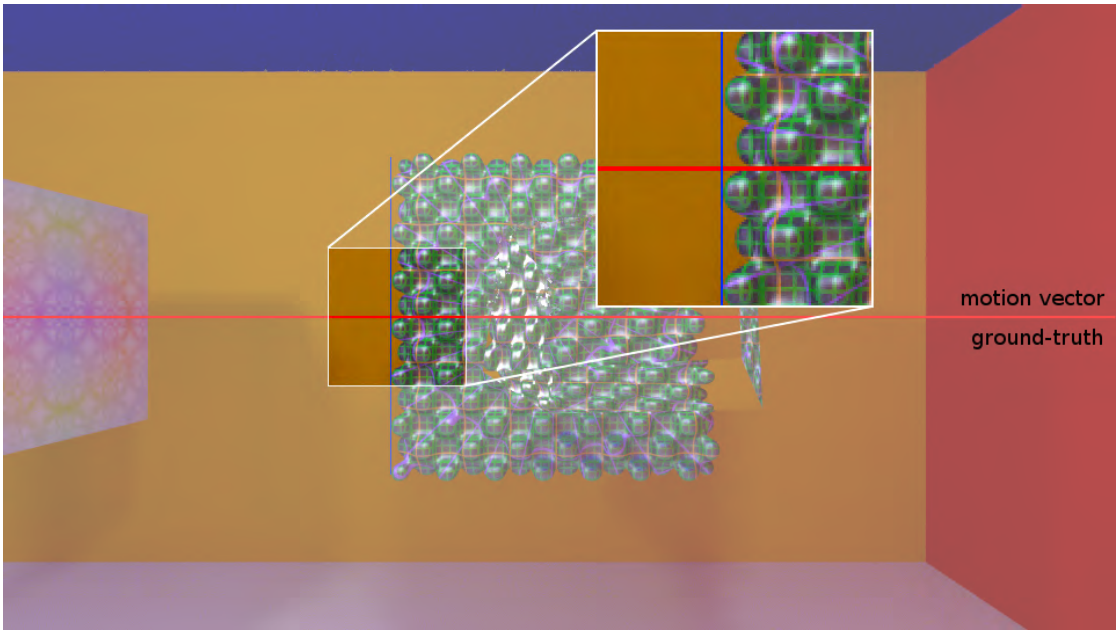


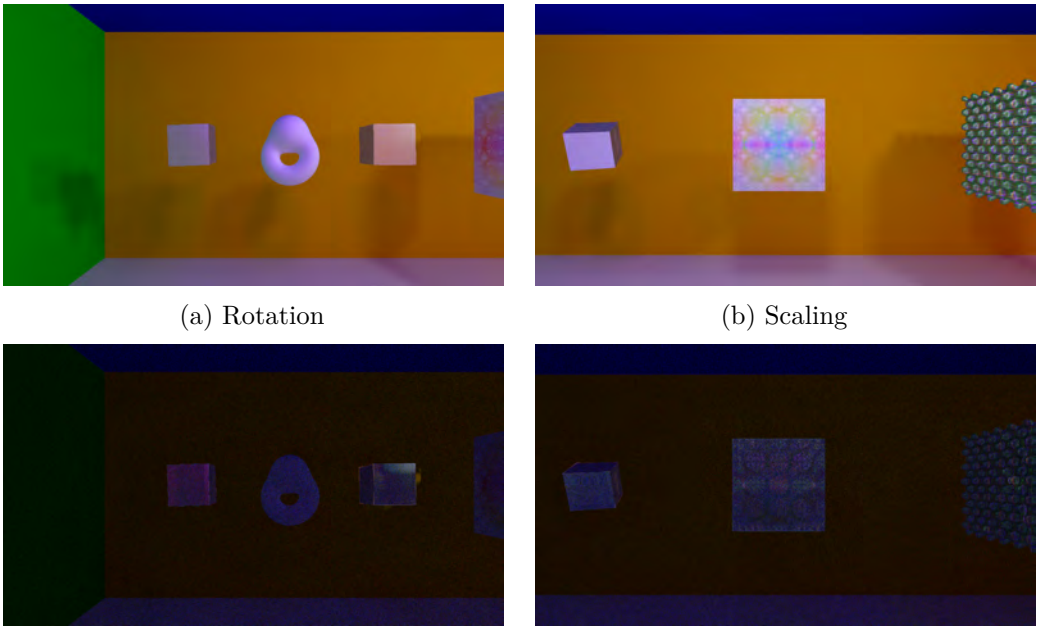
Figure 7.16: Small non-linear motion (camera movement in this case) leads to false edge location. In this image an offset to the ground-truth frame of one pixel can be seen. The vertical blue line marks the position of where the bumps should be.

With the **pixel-similarity technique**, the highlights on those objects are tracked correctly using the proposed similarity measure. Occasionally, the non-local means algorithm fails, due to its rotation invariance, to find the correct surface point, resulting into minor noise, as seen in Figure 7.22. Furthermore, over-sampling in the rendered frames leads to altered surface values resulting into false coloring at the edges of the reflective cubes, which is visible as a small border around them.

Unlike the proposed techniques, the **scene-assisted method** produces plausible looking, yet false specular highlights on reflective surfaces when compared to the ground-truth. As can be seen in Figure 7.23a, frame 611 of the test sequence has a pink highlight on the lower half at the front-side of the cube. The proposed techniques produce a similar result, as can be seen in Figure 7.23b and Figure 7.23c. However, the scene-assisted method generates a slightly bigger highlight (Figure 7.23d), which spreads to the upper left corner as well. The **video encoding** upsampling produces highlights which are not as bright in terms of color for the same frame.

Transparent objects

Similar to the specular surfaces, the **motion-vector technique** picks up the correct movement (as seen in Figure 7.24) when the amount of color change allows it. However, the fast rotation in that particular scene results in noise around the edges, as seen in Figure 7.24.



(a) Rotation (b) Scaling

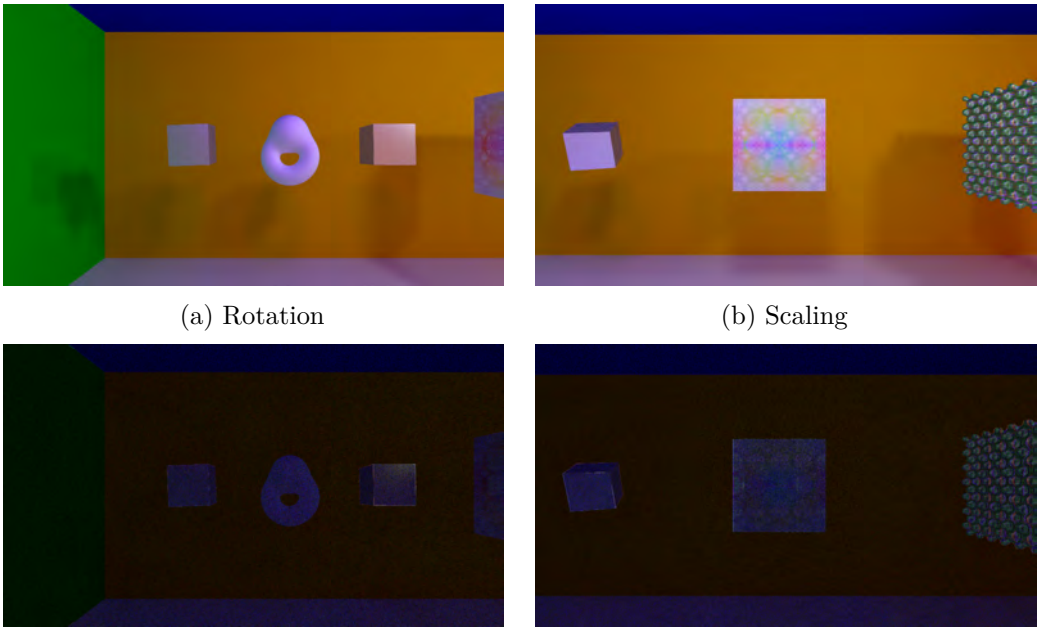
(c) Rotation - difference to ground truth (d) Scaling - difference to ground truth

Figure 7.17: Edges are preserved with the pixel-similarity technique. The same is true for high-frequency textures. Slight blurring occurs due to usage of maximal list size greater than one.

Since the perceived color of a glass-like object mainly originates from a light-surface interaction outside of it and data of this one is not available for the **pixel-similarity technique**, it fails to provide the correct coloring of such transparent surfaces as seen in Figure 7.25. It is possible to store such a value of the interaction, but only if the underlying technique allows it. This also highlights a problem that can occur at specular surfaces. Since only the information of the second light-surface interaction is available to the upsampling technique, light traveling through multiple mirror-like objects can result in false moving image features in the I-frames. A single transparent object acts like such a multiple mirror scene, because the perceived color originates in a light-surface interaction beyond the second one. These shortcomings are noticeable in blurring on the transparent surfaces as well as false edges on the side of the glass cube which does not face the camera.

The **scene-assisted method** handles transparency somewhat similarly to the pixel-similarity technique (as was already mention in Section 6.2). As can be seen from Figure 7.26d and 7.26c, both produce 'ghost edges' at the side of the cube that does not face the camera. Apart from that, both techniques generate blurred image features. In general, the pixel-similarity technique creates these two artifacts to a lesser extent, although this can vary depending on the frame and particular location on the cube.

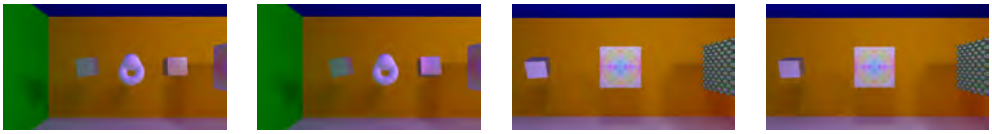
Similar to the motion-vector technique, the **video encoding** upsampling preserves



(a) Rotation (b) Scaling

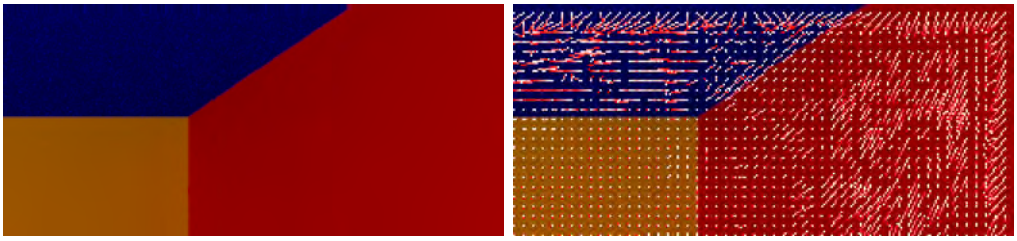
(c) Rotation - difference to ground truth (d) Scaling - difference to ground truth

Figure 7.18: Edges are preserved with the scene-assisted method by Yang et al. [YTS+11]. The same is true for high-frequency textures. Slight inaccuracies due to oversampling.



(a) Rotation 1 (b) Rotation 2 (c) Scaling 1 (d) Scaling 2

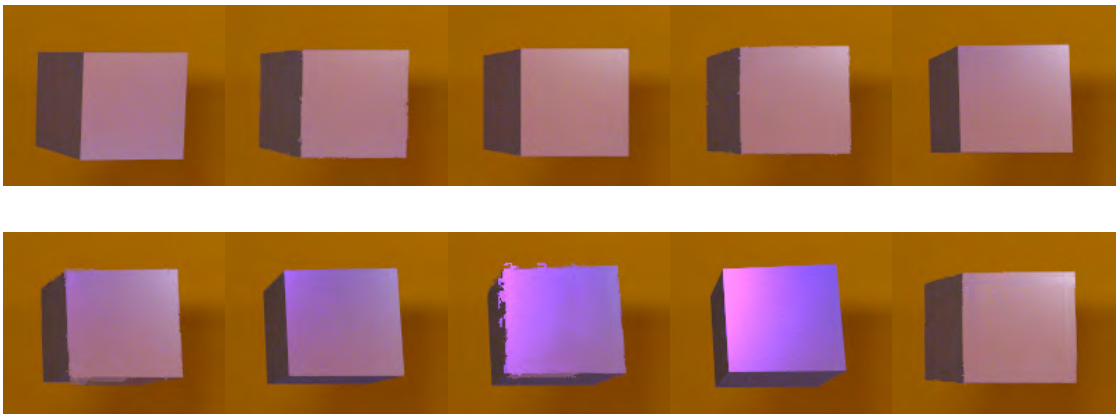
Figure 7.19: Ghost edges due to upsampling with video encoding.



(a) Result of noise (b) False motion vectors due to noise

Figure 7.20: Noise resulting into false motion vectors. In the case of the red and blue wall those do not matter, but on the edges of those two walls it does, since it results into a not discontinuous edge-line.

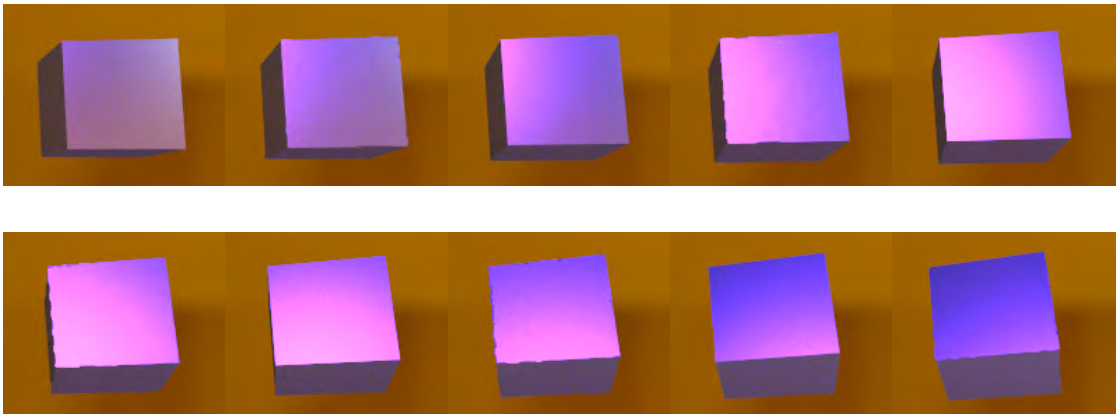
(a) B-frame (0838) (b) I-frame (0839) (c) B-frame (0840) (d) I-frame (0841) (e) B-frame (0842)



(f) I-frame (0843) (g) B-frame (0844) (h) I-frame (0845) (i) B-frame (0846) (j) Tearing

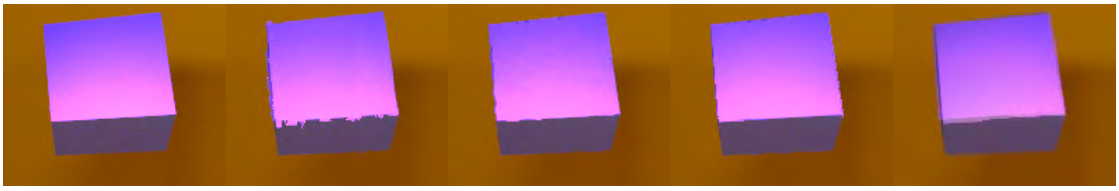
Figure 7.21: Specular surface with the motion-vector technique. Figures (a) to (i) show a cutout of an image sequence generated by the motion-vector technique. It shows, that small color changes do not result into problems, but as soon as they are significant (Figure (f) and (h)), the algorithm produces artifacts known as 'color bleeding'. In addition to that, the iterative search variant, produces a 'tearing effect' as seen in Figure (j).

(a) B-frame (0604) (b) I-frame (0605) (c) B-frame (0606) (d) I-frame (0607) (e) B-frame (0608)



(f) I-frame (0609) (g) B-frame (0610) (h) I-frame (0611) (i) B-frame (0612) (j) I-frame (0613)

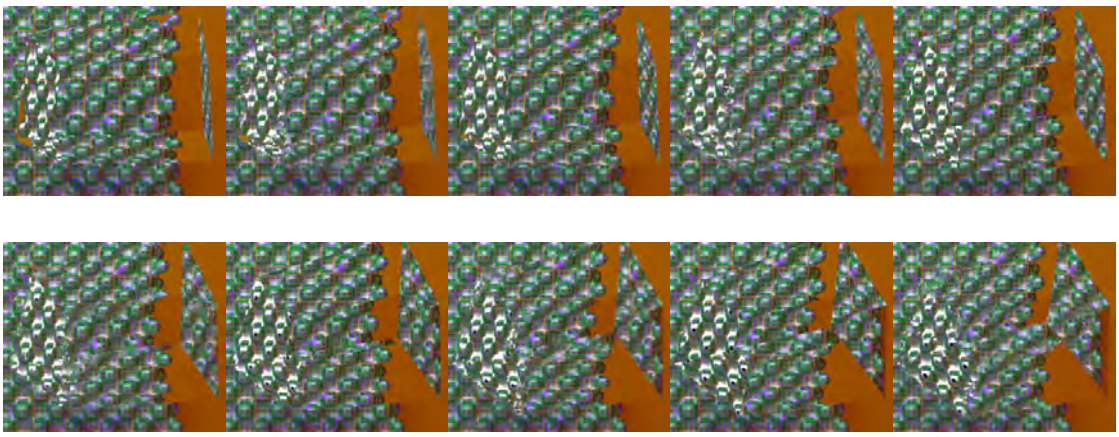
Figure 7.22: Specular surface with the pixel-similarity technique. Figures (a) to (j) show a cutout of an image sequence generated with that technique. Some noise is visible at the highlights as well as small errors around the edges. The latter are a result of oversampling.



(a) Ground-truth frame (b) motion-vector technique (c) pixel-similarity technique (d) scene-assisted method (e) video encoding upsampling

Figure 7.23: Specular highlight in frame 611 of the test sequence with various upsampling techniques.

(a) B-frame (2350) (b) I-frame (2351) (c) B-frame (2352) (d) I-frame (2353) (e) B-frame (2354)



(f) I-frame (2355) (g) B-frame (2356) (h) I-frame (2357) (i) B-frame (2358) (j) I-frame (2359)

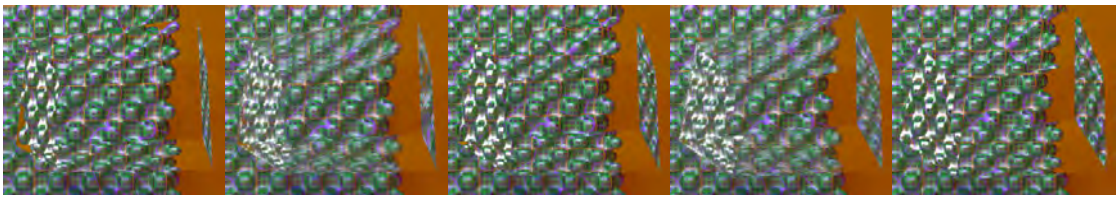
Figure 7.24: Example of the motion-vector technique handling transparent objects. As can be seen from the figures above, edges lead to noise whilst the surface is free from it.

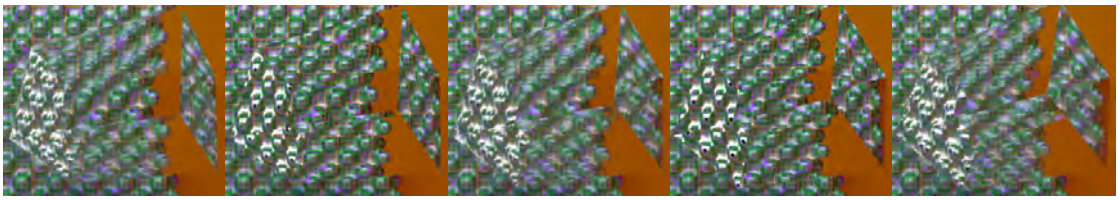
the optical flow of the transparent surfaces better than the pixel-similarity technique and the scene-assisted method, as can be seen in Figure 7.26b and 7.26e. However, the motion-vector technique is more precise when it comes to the edge location, and in general, the video encoding method generates a 'blurring effect' to some extend, as can be seen at the right side of the cube in Figure 7.26e.

Continuous sequence

Even when the resulting video is played with half of the intended 60 fps, the **motion-vector technique** does not show any jumping image features. However, the artifacts of specular surfaces are noticeable at any frame rate. The noise around the edges of the transparent cube and the 'tearing effect' on the refractive cube are not recognizable at half the playback speed.

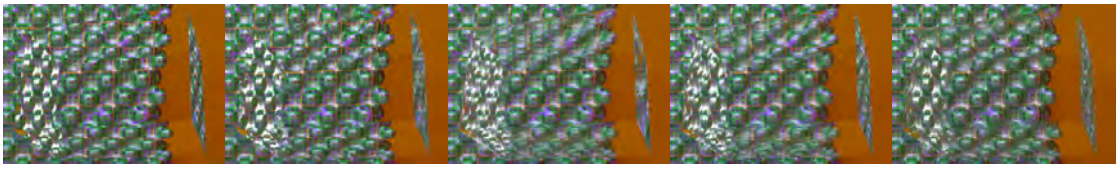
(a) B-frame (2350) (b) I-frame (2351) (c) B-frame (2352) (d) I-frame (2353) (e) B-frame (2354)





(f) I-frame (2355) (g) B-frame (2356) (h) I-frame (2357) (i) B-frame (2358) (j) I-frame (2359)

Figure 7.25: Example of the pixel-similarity technique handling transparent objects. This techniques leads to blurring and false edges at the back-side of the glass cube.



(a) Ground-truth frame (b) motion-vector technique (c) pixel-similarity technique (d) scene-assisted method (e) video encoding upsampling

Figure 7.26: Glass cube in frame 2351 of the test sequence with various upsampling techniques.

Unlike the motion-vector method, the artifacts of the **pixel-similarity technique** that come with the specular surfaces are not visible at the intended frame-rate of 60 fps. The noise becomes noticeable when the playback speed of the video is reduced to 30 fps. Likewise, the problems with refraction are noticeable at half the playback speed as small stuttering.

Small borders can be observed at half the playback speed (30 fps) of the **scene-assisted method** when it comes to the specular cubes in scene 1. Again, the problems with the refractions are not visible at 60 frames per second, but show up at half the speed.

Upsampling using **video encoding** shows sometimes stuttering geometry at the edges of the rotating cubes at the intended playback speed of 60 fps. When the video is played at half the speed, more artifacts become visible, such as slightly shifted parts of the detailed cube in scene 3.

7.4 Summary

Overall, the **video encoding** performs the worst of all techniques. However, the fact that the compression reduces the quality in the first place and that the values are not compared in the full high-definition range for colors (32 bit floating point for each color channel) does not allow a fair comparison when it comes to the mean squared error measure.

The **scene-assisted method** by Yang et al. [YTS+11] has no problems handling diffuse reflection, but struggles with the refractions and reflections in the test sequence. In particular, it has problems with rapid color changes, as mentioned by the authors. This happens, for example, when a side of a cube pops up from one frame to the other.

The presented **motion-vector technique** successfully improves the short comings of the video encoding and in general better than the scene-assisted method. It performs the best when it comes to refractions out of all techniques. However, when the cubes in the test sequence show a new side from one frame to the other, the technique has troubles handling it and leads to some sort of 'color bleeding'.

Last, but not least, the **pixel-similarity technique** outperforms all other techniques slightly when it comes to diffuse surfaces and reflections. The problems of this technique is the transparent cube in the last part of the image sequence. In terms of mean squared error, it cannot hold up to the motion-vector or the scene-assisted method. The visual quality in that case is somewhat on the same level as the scene-assisted method, but definitely worse compared to the motion-vector technique.

CHAPTER 8

Conclusion

The two proposed techniques of this thesis are the first to utilize the non-local means algorithm for upsampling. Furthermore, these methods are independent of the global-illumination algorithm that generates the B-frames, unlike other algorithms using temporal coherence to increase performance. Furthermore, the time to upsample the frames is significantly lower than the time to render them with a global-illumination algorithm.

The motion-vector technique uses the non-local means algorithm to detect movement across two B-frames. The pixels of the I-frame are then filled according to those motion vectors. It is more accurate than upsampling done via video encoding, since this one works on a block- and not on a pixel basis. The method lacks in accuracy when it comes to edge location, but successfully restores the optical flow of reflective and transparent objects. The former problem is the only case when this technique performs worse than the scene-assisted method by Yang et al. [YTS+11].

The pixel-similarity technique takes a stubby frame, i.e., a rendering of the I-frame without the shading, but with surface data for every pixel. The color for the pixels is then taken based on the similarity of that surface data at the neighboring B-frames. This technique is more accurate than the motion-vector technique, when it comes to edges. However, this technique has one major drawback: It only performs well enough when the data available allows it. In the test sequence, the technique had no problems with the reflective surfaces (where the appropriate data was available), but struggles with the transparent ones due to the lack of proper data.

Overall, the proposed techniques perform better than other image-based upsampling techniques. A downside of both techniques is the need for parameter finding, which has to be done for each image sequence manually. Furthermore, the resulting videos of both techniques show artifacts due to the rotation- and scale invariance of the non-local means algorithm.

8.1 Future Work

As can be seen from the statements made before, the proposed techniques have room for improvement. One problem of the non-local means similarity detection is that it can not handle rotation and scaling as well as translations that are aligned to the image axis. A similarity measure that deals with rotation and scaling better is therefore desirable. Another improvement to the proposed techniques could be the re-calculation of pixels that are considered to be part of an artifact in the resulting image. To do so, a way ro determine such pixels has to be found.

Bibliography

[AFH+] Armstrong, Flynn, Hammond, Jolly, and Salmon. High frame-rate television.

[BCM05] Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. A non-local algorithm for image denoising. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2 - Volume 02*, CVPR '05, pages 60–65, Washington, DC, USA, 2005. IEEE Computer Society.

[BMS+12] Huw Bowles, Kenny Mitchell, Robert W. Sumner, Jeremy Moore, and Markus Gross. Iterative image warping. *Computer Graphics Forum (proceedings of EUROGRAPHICS)*, 31(2), may 2012.

[DSDD07] Carsten Dachsbacher, Marc Stamminger, George Drettakis, and Frédo Durand. Implicit visibility and antiradiance for interactive global illumination. *ACM Trans. Graph.*, 26(3), July 2007.

[GLPP08] Bart Goossens, Quang Luong, Aleksandra Pizurica, and Wilfried Philips. An improved non-local denoising algorithm. In *2008 International Workshop on Local and Non-Local Approximation in Image Processing (LNLA 2008)*, pages 143–156, 2008.

[GTGB84] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. Modeling the interaction of light between diffuse surfaces. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '84, pages 213–222, New York, NY, USA, 1984. ACM.

[HDMpS03] Vlastimil Havran, Cyrille Damez, Karol Myszkowski, and Hans peter Seidel. An efficient spatio-temporal architecture for animation rendering, 2003.

[KFC+10] Jaroslav Křivánek, Marcos Fajardo, Per H. Christensen, Eric Tabellion, Michael Bunnell, David Larsson, and Anton Kaplanyan. Global illumination across industries. In *ACM SIGGRAPH 2010 Courses*, SIGGRAPH '10, New York, NY, USA, 2010. ACM.

[LW93] Eric P. Lafortune and Yves D. Willems. Bi-directional path tracing. In *PROCEEDINGS OF THIRD INTERNATIONAL CONFERENCE ON COMPUTATIONAL GRAPHICS AND VISUALIZATION TECHNIQUES (COMPUGRAPHICS âĂŹ93*, pages 145–153, 1993.

[MT13] Shilpa Metkar and Sanjay Talbar. *Motion Estimation Techniques for Digital Video Coding*. Springer Publishing Company, Incorporated, 2013.

[NSL+07] Diego Nehab, Pedro V. Sander, Jason Lawrence, Natalya Tatarchuk, and John R. Isidoro. Accelerating real-time shading with reverse reprojection caching. In *Proceedings of the 22Nd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware*, GH '07, pages 25–35, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[RDGK12] Tobias Ritschel, Carsten Dachsbacher, Thorsten Grosch, and Jan Kautz. The state of the art in interactive global illumination. *Comput. Graph. Forum*, 31(1):160–188, February 2012.

[SRB14] Deqing Sun, Stefan Roth, and MichaelJ. Black. A quantitative analysis of current practices in optical flow estimation and the principles behind them. *International Journal of Computer Vision*, 106(2):115–137, 2014.

[SYM+11] Daniel Scherzer, Lei Yang, Oliver Mattausch, Diego Nehab, Pedro V. Sander, Michael Wimmer, and Elmar Eisemann. A survey on temporal coherence methods in real-time rendering. In *EUROGRAPHICS 2011 State of the Art Reports*, pages 101–126. Eurographics Association, 2011.

[Vea98] Eric Veach. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford, CA, USA, 1998. AAI9837162.

[WSBL03] T. Wiegand, G.J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the h.264/avc video coding standard. *Circuits and Systems for Video Technology, IEEE Transactions on*, 13(7):560–576, July 2003.

[YTS+11] Lei Yang, Yu-Chiu Tse, Pedro V. Sander, Jason Lawrence, Diego Nehab, Hugues Hoppe, and Clara L. Wilkins. Image-based bidirectional scene reprojection. *ACM Trans. Graph.*, 30(6):150:1–150:10, dec 2011.