Institut für Computergraphik
und Algorithmen
Technische Universität Wien
Karlsplatz 13/186/2
A-1040 Wien
AUSTRIA
Tel: +43 (1) 58801-18601
Fax: +43 (1) 58801-18698

Institute of Computer Graphics
and Algorithms
Vienna University of Technology
email:
technical-report@cg.tuwien.ac.at
other services:
http://www.cg.tuwien.ac.at/

TECHNICAL REPORT

Image-based Reprojection Using a Non-local Means Algorithm

Clemens Rögner* Michael Wimmer* Johannes Hanika† Carsten Dachsbacher†

TR-186-2-05-2
April 2015

Keywords: temporal upsampling, image reprojection, offline rednering, optical flow, image-based rendering

Image-based Reprojection Using a Non-local Means Algorithm

Clemens Rögner* Michael Wimmer* Johannes Hanika†
Carsten Dachsbacher†

*Vienna University of Technology †Karlsruhe Institute of Technology

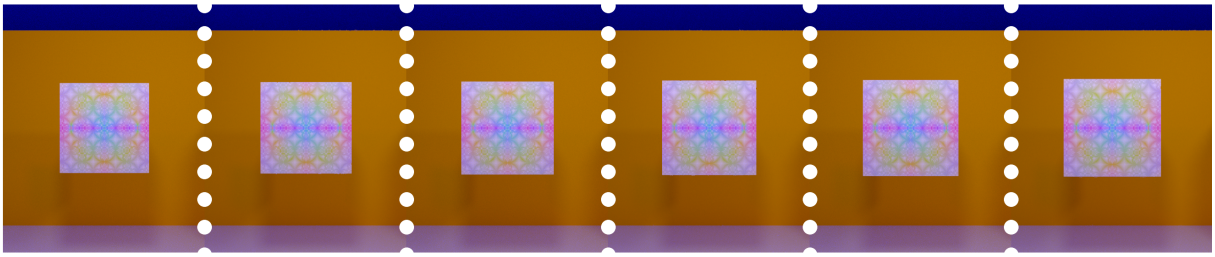


Figure 1: Part of an image sequence, which is upsampled using our algorithm. We produce motion vectors via a non-local means algorithm, which gives us more precise results.

Abstract

We introduce an image-based approach to increase the framerate of image sequences generated with offline rendering algorithms. Our method handles in most cases reflections and refractions better than existing image-based temporal coherence techniques. The proposed technique is also more accurate than some image-based upsampling methods, because it calculates an individual result for each pixel.

Our proposed algorithm takes a pair of frames and generates motion vectors for each pixel. This allows for adding a new frame between that pair and thus increasing the framerate. To find the motion vectors, we utilize the non-local means denoising algorithm, which determines the similarity of two pixels by their surrounding and re-interpret that similarity as the likelihood of movement from one pixel to the other. This is similar to what it is done in video encoding to reduce file size, but in our case is done for each pixel individually instead of a block-wise approach, making our technique more accurate. Our method also improves on work in the field of real-time rendering. Such techniques use motion vectors, which are generated through knowledge about the movement of objects within the scene. This can lead to problems when the optical flow in an image sequence is not coherent with the objects movement. Our method avoids those problems. Furthermore, previous work has shown, that the non-local means algorithm can be optimized for parallel execution, which significantly reduces the time to execute our proposed technique as well.

CR Categories: I.3.3 [Computer Graphics]: Three-Dimensional Graphics and Realism—Display Algorithms I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Radiosity;

Keywords: temporal upsampling, image reprojection, offline rednering, optical flow, image-based rendering

1 Introduction

The demand of the CGI Industry for high-quality images such as the ones generated with global illumination rendering algorithms and increased framerates leads to more computational time to generate the image sequences. A common approach to reduce the rendering time is to exploit temporal coherence across the frames in the sequence. Many

*e-mail:croegner|wimmer@cg.tuwien.ac.at@gmx.at
†e-mail:hanika|dachsbacher@kit.edu

different algorithms that utilize temporal coherence exist.

For example, there are several techniques specifically tailored for the global-illumination rendering algorithms at hand. All of them re-use calculations from a previous point in time to increase the performance. One such an approach was introduced by [Havran et al. 2003] 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. However, tailored approaches, such as the one by [Havran et al. 2003], are often complex and therefore difficult to implement.

A different approach to use temporal coherence is to work on an image-base, which is often used for interactive or real-time applications. Those approaches work with the final results of the rendering algorithms, the images, and generate additional frames based on them, so-called upsampling. In general image-based approaches are less difficult to implement, like the one of [Yang et al. 2011]. They propose two techniques, which rely on knowledge about the movement of objects within a virtual scene to generate so-called motion vectors (which describe the positional change across the two frames). Because of this, their approach, unlike the previous mentioned one, has problems handling color changes (the so-called optical flow) of reflections and refractions, because such surface types can lead to optical flow that is incoherent with the objects motion.

Another approach for using image-based upsampling is to utilize the motion compensation of video coding techniques. To safe memory space, video encoding, partitions frames into pixel blocks of various size and searches for possible locations of those blocks in adjacent frames. This results into motion vectors for each block. Generating an additional frame then can be done by moving the block to its projected position along those motion vectors. Compared to the work of [Yang et al. 2011] this technique works independently of the objects movement, but is more inaccurate because not every pixel has an individual motion vector and furthermore, parts of the upsampled frame, which cannot be colored by the existing blocks, have to be filled via an heuristic.

As one can see from the statements made above, upsampling techniques aimed towards renderings featuring global-illumination effects, either suffer from their complexity, can not handle certain shading features (such as transparency and specularity) properly or lack in accuracy. Therefore, we propose an image-based method that deals with those issues.

Our approach uses two rendered frames and executes the following two steps to generate a new frame: First, it calculates a motion vector for each pixel in one frame by detecting a similar pixel in the other. This process is repeated the other way around, resulting in an additional set of motion vectors. The second step of our techniques is to find the color for each pixel in the frame that is about to be generated. This last step can be done in two different ways as is shown in this paper. However, the main contribution of this paper is the usage of an adapted non-local means denoising algorithm to detect the similarities between two pixel for the first step. The core functionality of the non-local means algorithm is to qualify two pixels as similar based on their surrounding ones. As mentioned before, this similarity is used to generate the motion vectors between two frames. Therefore, our technique interprets the similarity between two pixels as the likelihood that one pixel moved to the position of the other one. The usage of the non-local means algorithm is chosen, because it generates results for every pixel (unlike the usage of the video codings motion compensation), it makes our technique independent of the objects movement (unlike the work of [Yang et al. 2011]), which leads to a better handling of reflection and refraction compared to previous work, and furthermore, it can be optimized for parallel execution to reduce its computational time.

2 Previous Work

In this section we will explain the image-based techniques, which are relevant for our proposed method, in more detail. All of those methods interpolate frames (further referred to as *I-frames*) based on one are more base frames (so-called *B-frames*). 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 as defined by [Scherzer et al. 2011]:

- **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.

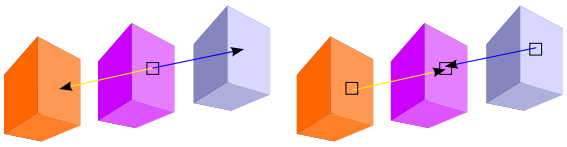
In the last part of this section, the denoising algorithm, which our approach utilizes, is explained (including an optimization for parallel execution).

2.1 Video compression

Video compression standards include some sort of motion compensation to reduce the file size of the video. The key idea is to store the motion of image parts so that pixel values of those parts can be reused in other frames [Wiegand et al. 2003]. As mentioned before, this can be used to artificially generate new frames. The motion detection is done via so-called block matching, which works as follows: The first step 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 absolute differences (as seen in Equation 1) of the per pixel differences (using the color c). The motion vector then points towards the block with the highest similarity, which is the block with the lowest value using the sum of absolute differences [Wiegand et al. 2003].

$$SAD(a,b) = \sum_{d_x}\sum_{d_y} |c(f_1, a_x + d_x, a_y + d_y) - c(f_2, b_x + d_x, b_y + d_y)| \quad (1)$$

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 (as defined by [Scherzer et al. 2011]). Pixels that cannot be filled by doing so are then colored by using a heuristic, such as interpolating the colors of the pixels at the same location in adjacent frames.



(a) Scene Assisted (b) Image based

Figure 2: Difference in the data usage of two methods proposed by [Yang et al. 2011]. 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.

2.2 Bidirectional Image Re-projection

[Yang et al. 2011] propose 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 the previous section) and can be calculated through the knowledge of the objects position in the adjacent frame. The different methods all vary in the available data (as seen in Figure 2) for the upsampling and hence in the quality of their result.

The scene-assisted 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 defined by [Scherzer et al. 2011]. According to the authors, the scene-assisted method produces the best results.

The image-based approach proposed by [Yang et al. 2011] 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. This is done by executing an iterative search for the source pixel S (which has the color information). That search is initialized by the target pixel T's (the pixel that is about to be colored) 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

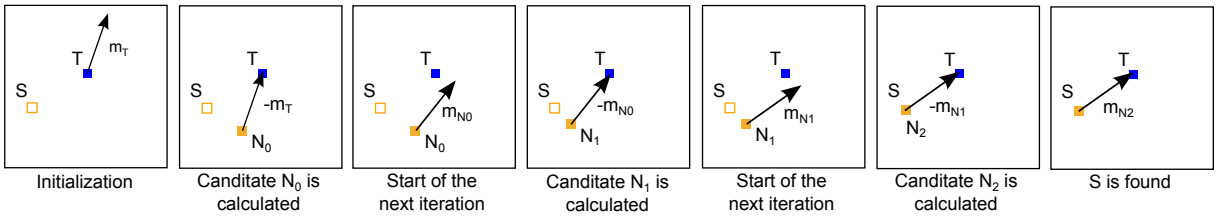


Figure 3: Three iterations of the algorithm by Yang et al. [Yang et al. 2011] which is used to estimate the movement of a pixel from one frame to another. This technique is also used in our approach.

it does, the pixel that moves to the target pixel is found. 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. We use this technique of finding the color in our approach, as described in Section 3.

2.3 Non-local means video denoising

Image noise is a problem that can be caused by various things, such as photodiode leakage currents in cameras. To reduce noise in an image one can apply so-called denoising methods. [Buades et al. 2005] introduced such a method, the non-local means algorithm. The algorithm works as follows: For one pixel the similarities to all other pixel in a spatial neighborhood (the blue-doted area in Figure 5) around it are calculated. Two pixels are classified as similar when their surrounding (the orange areas in Figure 5) matches. The color of the pixels in the neighborhood are then weighted by the similarity value and added together to make up the denoised pixel color. Mathematically, this method can be described via the Equation 2:

$$O_x = \frac{\sum_K \sum_S \omega(x+s, k+s)\sigma(s)P_k}{\sum_K \sum_S \omega(x+s, k+s)\sigma(s)} \qquad (2)$$

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.
- 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.
- P_k: Is the color at location k.

The work done by [Goossens et al. 2008] improves the work of [Buades et al. 2005] 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, because it allows for splitting the sum to calculate the weight from each point in the neighborhood. This is beneficial for parallel execution since the similarity measure can now be executed in several full-image passes, which the memory access pattern.

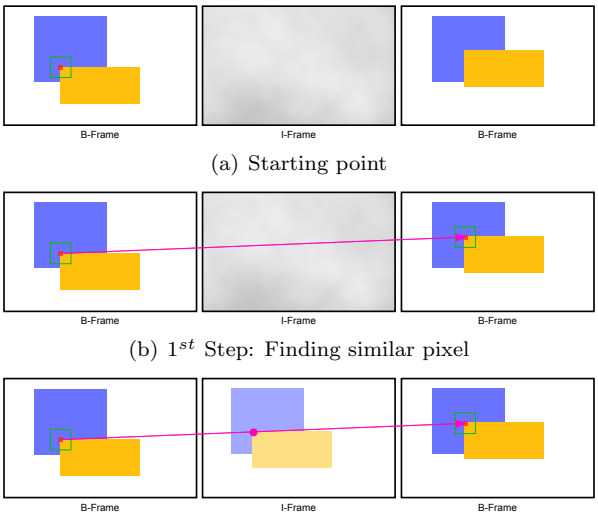
The full-image passes that have to be done are the calculation of the difference between every pixel, resulting in a similarity weight for each pixel. After that, the sum of those weights has to be calculated and finally the contribution of the color with the sum of weights has to be applied to the result. 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. Those optimizations to the technique are also applied to our method, as seen in Algorithm 1.

3 Algorithm

As mentioned before, our proposed techniques takes two B-frames (as seen in Figure 4(a)) and generates motion vectors by applying the non-local means algorithm (the maximum-detection variant as in 3.1) to the color information of those two B-frames as shown in Figure 4(b). The resulting similarity value for a pixel in one B-frame to the pixels in the other B-frame can be interpreted as the likelihood that the one pixel moved to the location of the other. The actual motion vector for one pixel in a B-frame is then taken from the pixel in the other B-frame with the highest similarity value. 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 4(c). As suggested by [Yang et al. 2011], we perform this search on the previous and next frame to counter the problem of occluded areas when only one frame is used. Hence, our technique requires motion vectors, which point from one frame to another, and motion vectors, which point the other way around. The search for the source pixel in one frame can be done using two different methods:

- **Iterative search:** This method was described by [Yang et al. 2011] and is explained in Section 2.2. If the iterative search for a pixel does not succeed, a heuristic to find the color for that pixel has to be employed. We suggest taking the result of the first iteration, similar to what [Yang et al. 2011] suggest in their work. One reason for such a failed search can be inconsistent motion vectors generated by the algorithm. Another reason, as stated by [Yang et al. 2011], can be thin and fast-moving geometry. To avoid the problem of using a heuristic at all, the method, described next can be used.

- **Brute force:** Since the non-local means algorithm searches around one pixel in a specific radius (the neighborhood area), the best-fitting motion vector has to be within that radius. 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 timeline 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.



(a) Starting point

(b) 1^{st} Step: Finding similar pixel

(c) 2^{nd} Step: Finding motion vectors and coloring

Figure 4: Our technique in pictures.

3.1 Adpated Non-Local Means Algorithm

The key to finding the motion vector is, as stated before, the non-local means algorithm. This algorithm qualifies two pixels as similar, based on if their surroundings match, which is further referred to as the similarity area S. The search for such similar pixels is conducted in a certain area around one pixel, which we call the neighborhood area K. This principal is depicted in Figure 5. The original algorithm for the purpose of denoising takes the weighted mean of all the pixels in the neighborhood into account to make up the final coloring of the pixel. Since our approach has to find the position of the most similar pixel, we do not calculate the weighted mean, but rather take the pixel with the highest similarity. In other words: we conduct a search for the maximum. This also does not require any thresholds for the calculation of the similarity, like it is done in the original version. Those thresholds can also cause a problem when there are significant changes in the image sequence that would require adjustment during it. Furthermore, those thresholds also requires some knowledge about the color range at hand. With our maximum-detection

variant of the non-local means algorithm, we avoid both problems.

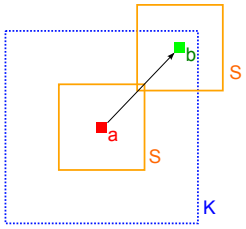


Figure 5: Schematic of the non-local means algorithm for denoising introduced by [Buades et al. 2005].

Therefore, our adapted non-local means algorithm works as follows: For each pixel a in B-frame **A** conduct a search in B-frame **B** for similar pixel b, which are in the neighborhood area around the same pixel-location of a in frame **B**. To calculate the similarity value of pixel a and pixel b, compare their surroundings, the similarity area. This comparison is done by calculating the difference of each pixel in the similarity area and adding those differences up. Save the location of pixel b with the highest similarity as the location to which pixel a is most likely to have moved to. Algorithm 1 depicts the workings of our proposed non-local means variant, which includes the performance improvements for parallel execution proposed by [Goossens et al. 2008], as described in Section 2.3.

It has to be noted, that $\xi(a, b)$ in Algorithm 1 describes the so-called similarity measure, a function that calculates the difference between two pixels (a and b) and returns one if they are equal and zero otherwise. This definition comes from the original version of the non-local means algorithm by [Buades et al. 2005] and is kept for convenience. The actual content of this function is described in the following section. Furthermore, we investigate if the usage of a weighting function for the differences of the similarity area's pixels based on the distance to the center improves our technique. In our implementation we used a so-called Gaussian kernel mapping (as in Equation 3) with a variance of 0.5 times the size of the similarity area and a mean that is equal to the coordinate of the center pixel of the similarity area. Werther such a weighting is useful for a specific image sequence or not, is discussed in Section 4. It has to be mentioned that although the similarity measure is commutative ($\xi(a, b) = \xi(b, a)$), the non-local means algorithm is not, because the

Algorithm 1: Maximal similarity detection

Input: Two images **A** and **B**, both of size WH.
Output: An image-like structure **L** that will contain the motion vectors for the pixels movement from **A** to **B**

$\mathbf{L}[WH] \Leftarrow 0$;
$\mathbf{V}[WH] \Leftarrow 0$;
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**
    $\mathbf{W}(x, y) = \xi(\mathbf{A}(x, y), \mathbf{B}(x + k_x, y + k_y))$
  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**
    if $\mathbf{W}_{sum}(x, y) > \mathbf{V}(x, y)$ **then**
      $\mathbf{L}(x, y) = k_{xy}$;
      $\mathbf{V}(x, y) = \mathbf{W}_{sum}(x, y)$;
    end
  end
end
return L

neighborhood area of a does not cover the same area as the one around b.

3.2 Similarity Measure

As mentioned earlier our 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 the previous section), 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 for the upsampling in the end, the actual value of the variance for such a normal distribution does not matter. Equation 3 shows a mapping of value x with the normal distribution function using the mean m and the variance v.

$$\phi(x, m, v^2) = \frac{1}{v\sqrt{2 * \Pi}} e^{-\frac{1}{2}(\frac{x-m}{v})^2} \tag{3}$$

Again, because only the highest values are of interest to do the job, the fraction before the exponential component in Equation 3 can be disregarded. The resulting similarity measure for our technique can be seen in Equation 4, whereas $c_i(p)$ returns the value of the particular color component i of the pixel p.

$$\xi(a, b) = \prod_{c_i=1}^{C} \phi(c_i(a) - c_i(b), m, v) \tag{4}$$

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. However, the actual value of the variance does not matter, because we detect the maximum and the variance does not affect that.

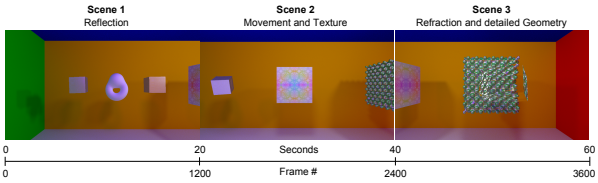


Figure 6: This figure shows the test sequence along the time-line with an image example for each scene. Scene 1 shows two rotating objects with reflective surfaces, each one with a different magnitude of specularity. Scene 2 features a scaling and rotating cube with a detailed diffuse texture. The last scene features a rotating transparent cube.

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 4 would not have that property.

4 Parameter finding

To apply our proposed algorithm to an actual image sequence, one has to set the parameters for the non-local means algorithm, namely the size of the neighborhood area and similarity area. To test our algorithms performance and show what has to be done to run it, we generated a sequence consisting of three scenes, each with different materials, as can be seen in Figure 6.

4.1 Neighborhood Area

The neighborhood area defines the maximal amount of movement from frame to frame that can be detect by our algorithm in screen space. When this value is too small, our algorithm produces so-called 'ghosting' artifacts. A bigger size than the minimal does not effect the algorithm itself, but only the time to generate the frames, hence it increases the computational cost. Finding the correct value of maximal movement across the image sequence (which relates to the minimal neighborhood size) cannot be done analytically (due to the unpredictability of the light transport), which requires the usage of heuristics.

One can simply make an elaborate guess to set the neighborhood size. Another heuristic is to calculate the maximal extend of motion vectors that are generated by the objects expected movement, like it is done in the work of [Yang et al. 2011]. In our implementation of the algorithm, we use the latter heuristic and add 20% of the maximal movement in pixels across to B-frames to make up the final radius of the neighborhood area. For our test scene (as described in Section 5) this gave us a sufficient size of the neighborhood area, so that no 'ghosting' artifacts occur.

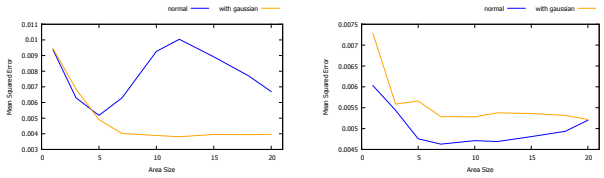
4.2 Similarity Area

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 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 (target) 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 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. 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.

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 is true to determine if a Gaussian kernel should be applied to the similarity area. Again, an analytical solution for finding the right value for the size is not possible. For our test image sequence (as described in Section 5), we chose a radius of 5 without the use of a Gaussian kernel weighting after conducting a trial-and-error test, which is shown in Figure 7. Note that a Gaussian kernel might be useful for another sequence.



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

Figure 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.

5 Results

After fixing the parameters for the image sequence at hand, we can conclude the following about the results of our proposed technique.

5.1 Objective Comparison

To evaluate the performance of our technique in comparison to the ground-truth frames, we choose the mean-squared error measure to do so. In addition to the ground-truth comparison we are able to show how our method performs compared to the work of [Yang et al. 2011] with the same metric. A fair comparison with the usage of the motion compensation in video coding is not possible, due to the quality loss (color- range and depth) of the coding techniques themselves. Furthermore, we compare our work to the scene-assisted method proposed by [Yang et al. 2011] and not the other, since the former method provides better results in terms of quality according to the authors.

From Figure 8 it can be seen, that our proposed technique deals significantly better with transparency than the previous work by [Yang et al. 2011]. However, the scene-assisted technique delivers a lower mean squared error for some frames with the reflective surfaces (the reason for this will be explained in the following section), but for the majority of those frames their performance can be considered equal. This similar performance is mainly due to the fact that those surfaces are not of perfect specular nature (unlike the transparent cube), but rather semi-specular, which reduces the problems of the technique by [Yang et al. 2011]. When it comes to the second scene, featuring the moving diffuse cube with the detailed texture, our technique delivers similar results as the scene-assisted method as

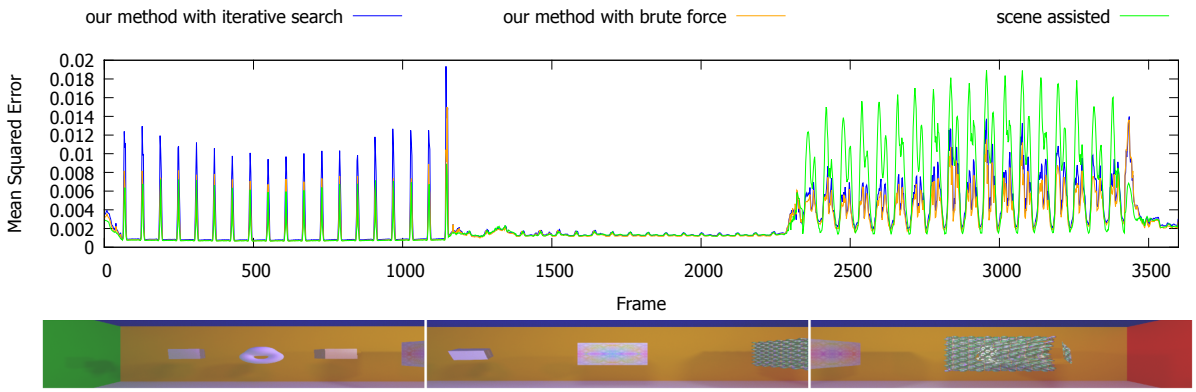


Figure 8: This figure shows the mean squared error difference to the ground-truth frames. The blue line depicts the performance of our method with the iterative search, while the orange line is the one with the brute force method. The green line shows the performance of the scene-assisted method of [Yang et al. 2011].

well. When looking at the graph one can notice the spikes in the differences to the ground-truth frames for all techniques in every scene. Those occur when a side of a cube appears in one frame that was not visible in the previous one. In general, this always happens when new geometry appears in the frame.

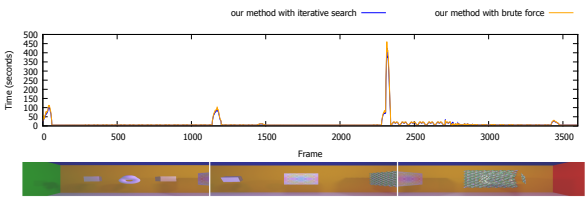


Figure 9: This figure shows the times to generate each frame in seconds on an Intel Core i7-4770K CPU with four 3.50GHz cores. The biggest neighborhood area size for our test sequence is 67.

Figure 9 shows the time to generate each frame in seconds on an Intel Core i7-4770K CPU with four 3.50GHz cores. The time is on average 12.6% higher for the brute force variant of our proposed technique. The reason for the different times to generate each frame is the size of the neighborhood area, which is calculated via the heuristic mentioned in Section 4.1.

5.2 Perceptional Findings

Although the mean squared error provides a good measure to depict which technique performs better in comparison to the ground-truth frames, a visual inspection of resulting frames reveals the artifacts that the techniques produce.

Figure 12 shows the improvement of our method compared to others, when it comes to handling transparent objects. The scene-assisted method by [Yang et al. 2011] produces ghosting artifacts all across the glass cube, which is the result of their motion vectors not pointing in the same direction as the path of the image features would suggest, as can be seen in Figure 10. Using the motion compensation of video coding techniques for upsampling, results in a slight blur across the cube and some false edges due to the block based matching for finding the motion vectors. As can be seen from the comparison with the other techniques, our method preserves the edges better than others, but still produces a little bit of noise around those. Apart from that, our method has troubles restoring the fine detail on the right side of the glass cube, but again, does it better than the other techniques.

The performance of our technique for the semi-specular surfaces in the first scene is for some frames, as mentioned earlier, not as good as the previous work. As can be seen from Figure 13, our method produces 'tearing' artifacts on the edges of the cube, when a sudden change in color for a significantly large amount of pixels occurs (not in terms of mean rapid movement, but rather in terms of a 'flash-like'-appearance of new color). Figure 14 shows a sequence of another five frames that highlight this problem of our approach as well. In such a case, the method of [Yang et al. 2011] has less troubles since it does not rely on the actual coloring to

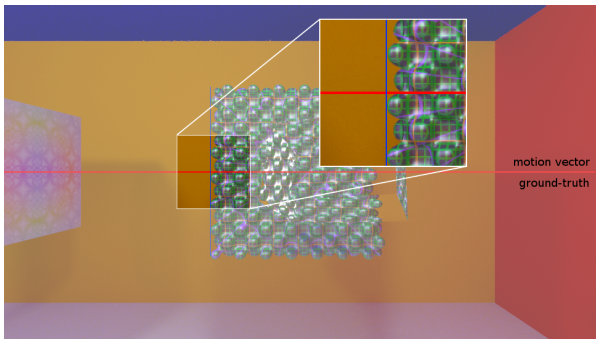


Figure 10: Non-linear motion poses a problem for our approach, since the linear motion vectors can not compensate for such movement.

find the motion vectors, although a slight 'border'-effect can be observed with their approach. Upsampling via video encoding has the same problem as our technique, but since it classifies the motion vectors as not usable (due to the low similarity of the blocks) and therefore uses a blending heuristic to generate the new frame, the visual error is not as severe compared to our method. When such a change in color for a sufficient amount of pixels does not occur, our proposed technique performs equally to the scene-assisted method, as mentioned in the previous section and as can be seen in Figure 8.

Another problem of our approach is non-linear motion. Since the camera path in our test sequence is generated via catmull-rom splines, the linear motion-vectors of our technique can lead to false positioning of the scene, as can be seen in Figure 10. This is also a problem of the upsampling using video coding, but is not a problem for the scene-assisted approach by [Yang et al. 2011].

6 Conclusion

We proposed an algorithm that upsamples frames from an image sequence generated with global-illumination rendering algorithms to reduce the overall time for computing that sequence. Our technique generates motion vectors by calculating the similarity of two pixels via the non-local means algorithm. The motion vectors then can be used to generate a new frame. One of the advantages our algorithm over previous work is that our method does not rely on knowledge about the movement of objects in the scene. This property leads to a better handling of reflections and refractions in some scenarios than other methods.

However, our method has problems with rapid appearance of new geometry or color. Furthermore, our algorithm requires some trial and error testing for finding the correct parameters to be applied to an image sequence.

For future work, we would like to find ways to qualify areas of the frames that may cause artifacts as such, which then could be used to perform a ground-truth rendering of those areas.

References

Buades, A., Coll, B., and Morel, J.-M. 2005. 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*, IEEE Computer Society, Washington, DC, USA, CVPR '05, 60–65.

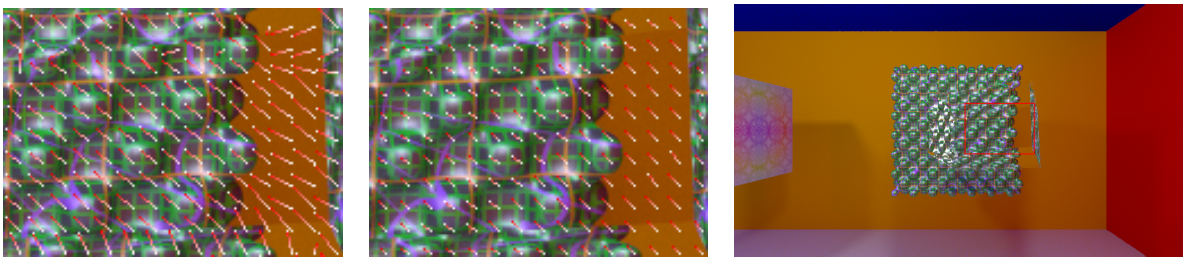
Goossens, B., Luong, Q., Pizurica, A., and Philips, W. 2008. An improved non-local denoising algorithm. In *2008 International Workshop on Local and Non-Local Approximation in Image Processing (LNLA 2008)*, 143–156.

Havran, V., Damez, C., Myszkowski, K., and peter Seidel, H., 2003. An efficient spatio-temporal architecture for animation rendering.

Scherzer, D., Yang, L., Mattausch, O., Nehab, D., Sander, P. V., Wimmer, M., and Eisemann, E. 2011. A survey on temporal coherence methods in real-time rendering. In *EUROGRAPHICS 2011 State of the Art Reports*, Eurographics Association, 101–126.

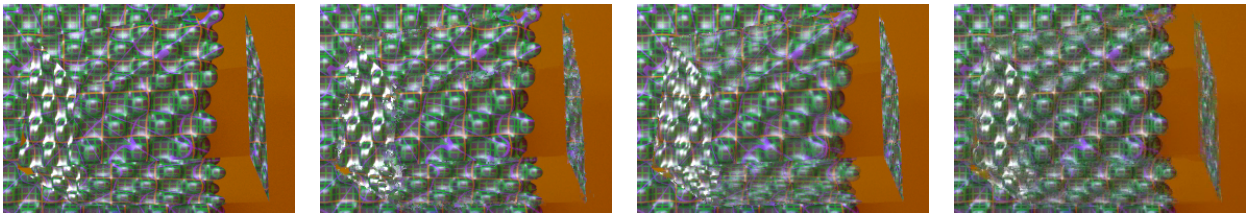
Wiegand, T., Sullivan, G., Bjontegaard, G., and Luthra, A. 2003. Overview of the h.264/avc video coding standard. *Circuits and Systems for Video Technology, IEEE Transactions on 13*, 7 (July), 560–576.

Yang, L., Tse, Y.-C., Sander, P. V., Lawrence, J., Nehab, D., Hoppe, H., and Wilkins, C. L. 2011. Image-based bidirectional scene reprojection. *ACM Trans. Graph. 30*, 6 (dec), 150:1–150:10.



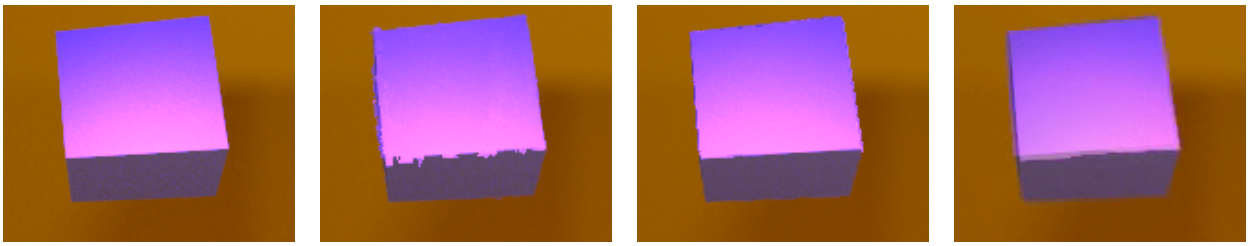
(a) Our method (b) Scene assisted (c) Frame 2351

Figure 11: Comparison of the motion vectors generated with our method (left figure) and with the approach by [Yang et al. 2011] (middle figure). Apart from the different length of the motion vectors, one can also notice a different alignment of the motion vectors from our method compared to the one of [Yang et al. 2011]. The motion vectors point from the white towards the red. The right figure shows the part of frame 2351 from our test sequence that is used for this comparison.



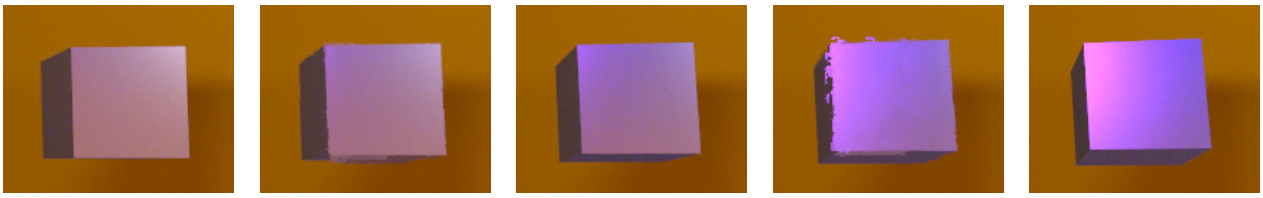
(a) Ground-truth frame (b) Our method (brute force) (c) Scene-assisted method (d) Video coding upsampling

Figure 12: Glass cube in frame 2351 of the test sequence with various upsampling techniques. The scene-assisted method by [Yang et al. 2011] produces ghosting due to false motion vectors. Upsampling via video encoding results into slight blurring and some artifacts around the edges. Our method does not suffer from the problems of the other methods, but produces some noise around the edges and can not handle the right side of the cube well, compared to the ground-truth frame.



(a) Ground-truth frame (b) Our method (brute force) (c) Scene-assisted method (d) Video coding upsampling

Figure 13: Glass cube in frame 611 of the test sequence with various upsampling techniques. Our method produces a 'tearing' artifact due to a flash-like color change on the surface. Upsampling via video coding uses a heuristic in such a case and the scene-assisted method of [Yang et al. 2011] is not affected by such a color change.



(a) B-frame #842 (b) I-frame #843 (c) B-frame #844 (d) I-frame #845 (e) B-frame #846

Figure 14: Sequence produced with our proposed technique (brute-force variant). 'Tearing' artifacts appear due to rapid color change across a major part of the surface. One can also observe the ghosting in Subfigure (b) as a result of the bottom side of the cube, which was not visible in the previous frame.