FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Color Adjustment of Colored Range Images

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Computergraphik & Digitale Bildverarbeitung

eingereicht von

Gabriele Hebart
Matrikelnummer 0626511

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Associate Prof. Dipl.Ing. Dipl.Ing. Dr.techn. Michael Wimmer
Mitwirkung: Dipl.Ing. Claus Scheiblauer

Wien, 27.11.2011

(Unterschrift Verfasserin) (Unterschrift Betreuung)

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Color Adjustment of Colored Range Images

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Visual Computing

by

Gabriele Hebart
Registration Number 0626511

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Associate Prof. Dipl.Ing. Dipl.Ing. Dr.techn. Michael Wimmer
Assistance: Dipl.Ing. Claus Scheiblauer

Vienna, 27.11.2011

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Gabriele Hebart
Münzgrabenstraße 215b/2, 8010 Graz

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.

(Ort, Datum) (Unterschrift Verfasserin)

Abstract

With the increased availability of imaging systems for three dimensional objects, virtual models of statues, buildings, archaeological excavation sites, or even whole cities can be created. A popular technique to build up these models is to generate a point cloud by combining multiple scans from a 3D scanner. The number of necessary scans depends on the size of the original object. The combination of the single scans leads to various color discontinuities in the overall model, which have to be corrected by post-processing.

This thesis shows two different approaches to automate the color correction process and improve the visual appearance of a point cloud. The first approach concentrates on the images for the colorization of the discrete scans, and tries to adjust them in a way to a achieve a common light situation for all of the images. A model created from point clouds which are colored by the use of the processed images, shows less color discontinuities.

In contrast to the first approach the second approach affects the points of an already colored point cloud. The basic idea is to recolor the single points according to the surrounding points and a specific filter. The result of the second approach depends on a number of settings and the choice of the applied filter. An appropriate combination of these factors should change the colors of a point cloud and make them appear smoother and less noisy.

Several point clouds together with the photos which were used to colorize them are employed to test and validate these two approaches.

Kurzfassung

Der Einsatz von Punktwolken, welche mit Hilfe von 3D Scannern erzeugt werden, hat zur Visualisierung von großen Szenen wie Statuen, Gebäuden, archäologische Ausgrabungsstätten und Städten, stark an Beliebtheit gewonnen.

Um mit dieser Technik das gesamte Objekt erfassen zu können sind mehrere Scans erforderlich, welche später zu einer Punktwolke zusammengefügt werden. Die Notwendigkeit von mehreren Scans führt jedoch dazu, dass sich die Bedingungen für die einzelnen Scans (wie zum Beispiel die Tageszeit) verändern, wodurch große Farbunterschiede im fertigen Modell auftreten, die in einem Nachbearbeitungsschritt ausgebessert werden müssen.

In dieser Diplomarbeit werden zwei Verfahren angewendet um die Korrektur der Farbunterschiede zu automatisieren und die Farbunterschiede zu minimieren und dadurch eine Verbesserung des Gesamterscheinungsbildes zu erzielen. Das erste Verfahren hat zum Ziel die Lichtsituation der einzelnen Fotos welche während des Scan Prozesses gemacht wurden, anzugleichen. Ein Punktemodell dessen Punkte mit diesen modifizierten Fotos eingefärbt wurden, zeigt weniger Farbunterschiede.

Im zweiten Verfahren wird die Farbe einzelner Punkte der gesamten Punktwolke anhand direkt angrenzender Punkte lokal verändert. Dies geschieht durch die Verwendung der angrenzenden Punkte in verschiedenen Filtern. Dadurch sollen die Farbunterschiede minimiert werden, wodurch ein allgemein homogeneres Gesamtbild des Punktemodells erzeugt wird. Die Ergebnisse sind von mehreren Einstellungen und der Wahl des Filters abhängig.

Zur Validierung werden beide Verfahren auf unterschiedliche Punktwolken und den Fotos mit denen diese Punktwolken eingefärbt wurden angewendet, und mit dem Original der Punktwolke verglichen.

Contents

CHAPTER 1

Introduction

With the use of automated range finding devices like a 3D scanner, the acquisition of the geometry of large real-world scenes gained more and more popularity. These devices make it possible to visualize statues, buildings, even archaeological excavation sites and cities. With the information recorded by a 3D scanner, a 3D point cloud can be generated. To get a complete 3D model of the desired object it is necessary to make multiple scans from different positions. Photographs of the object are taken during the scan process as well. These pictures are later used to color the point clouds acquired by the 3D scans.

The number of necessary scans depends on the size of the original object. To obtain a complete model of the scanned object the different single scans are merged together. For smaller objects, like statues, very often 3D meshes are generated out of point clouds. The use of points as rendering primitives instead of a mesh has become a popular alternative especially for rendering of complex scenes.

1.1 Problem Definition and Objectives

If one part of the scanned object appears on different scans from different positions, then the color for this part of the model probably differs on each of the corresponding images. The color of one part depends for example on the distance from the point to the scanner, the scanner orientation, and on the light situation (daylight, the use of spot lights). So when the different 3D scans are merged, the surface of the final model can have great variations on it, especially on overlapping regions. Figure 1.1 shows two examples of such point clouds. In Figure 1.1(a) the color discontinuities which appear when multiple scans are merged together are very recognizable. The boundaries of every scan are visible, for example on the floor. Figure 1.1(b) is a good example for an irregular surface on a point cloud model. For example on the column, a lot of dark spots are visible.

(a)

(b)

Figure 1.1: These images show color artefacts in the color of a 3D point cloud. Figure (a) is a good example for the color discontinuities which arise from the merge of different scans. Figure (b) shows an example of an irregular surface.

1.2 Contribution and Course of Investigation

Until now the correction of these errors in colored point clouds is done manually, which takes a lot of time, especially for very large point clouds. This work presents two approaches to automate this process:

- The first approach is an image-based approach, where the pictures which were taken during the scan process are modified.
- The second approach is a point-based approach where the points of the point cloud are recolored in object space according to their surrounding neighbor points.

One reason for the separation of the work into two parts is that the lighting when taking photographs influences the color of the different points a lot. The different scans, and therefore the different images, are made at different daytimes. For example pictures which are taken at noon can appear very bright or even overexposed, while images of the same object taken in the late afternoon or on very cloudy days may appear darker. Here the whole picture has a specific lighting. That means that the whole images differ, not just some parts of it. The basic approach to improve the color appearance of the point cloud is to modify the images so that all of the images which are taken to color the points in the point cloud have the same lighting mood.

The point-based approach then tries to get a smoother colored surface of the model and also tries to remove the noise. For this the points are recolored with the help of their neighbors. Some criteria for the selection of the neighbor points are developed to get the best result. Then some image processing techniques are used for the assignment of the final color.

1.3 Structure of the Work

This thesis is organized as follows:

Chapter 2 gives an overview of the state the art of the important topics covered in this thesis. The first part is about the range image acquisition pipeline, where the important steps about the generation of colored 3D point cloud are described. The next part presents different data structures used for the rendering of large point clouds. Then the different color adjustment methods for point clouds and 3D meshes coming from range images are described. These are followed by image-processing algorithms to remove noise from images. The chapter finishes with a short overview of the k-nearest neighbor search in point clouds.

Then Chapter 3 presents two existing methods to bring the images responsible for the colorization of the points in the same lighting mood. First a method in decorrelated color space is described, then one in correlated color space. These two methods are then compared.

Chapter 4 describes the method used for the already colored point cloud. First the kd-tree data structure used is described. This is followed by the techniques used for the calculation of the final point color.

The outcome of Chapter 4 in combination with the method described in Chapter 3 is presented in chapter 5. Here the outcome of the different settings which can be made for the point based approach is presented and discussed.

Then Chapter 6 closes this thesis with a summary and some possible enhancements for future work.

CHAPTER 2

State of the Art

This chapter gives an overview of the state the art of the important topics covered in this thesis. Section 2.1 deals with the range image acquisition pipeline and describes the how the final point cloud model was generated. After that, section 2.2 shows some examples how these point clouds can then be efficiently stored and shows some different rendering algorithms. As this thesis is about color adjustment for colored range images, section 2.3 gives an overview of current approaches for combining the color from multiple scans and on how to decrease color discontinuities. Section 2.4 then shows some methods on how to reduce noise in images. Section 2.5 then describes some methods for the k-nearest neighbor search in point clouds.

2.1 Range Image Data Acquisition Pipeline

Range Images are images that store depth values of an object. As these depth values are dependent of a certain viewpoint it is possible to obtain a 3D position of the points of the image, and later on it is possible to generate a 3D model out of one or more range images. But first they have to be generated, then registered and colored [9, 13].

2.1.1 Range Scanner Hardware

There are different types of laser scanners. The two most common types are the *triangulation* laser scanner and the *Time-of-Flight* laser scanner.

Triangulation System

In the triangulation system the laser projects a light spot on the object of interest and then this light spot is detected by a camera. With the known distance between the laser and the sensor, and the angle between the laser beam and the position of the spot on the camera image, it is possible to determine the position of the spot on the object by the use of triangulation. This principle is shown in figure 2.1(a).

Alternatively, another approach to obtain the shape of an object is to project a stripe or a pattern instead of a spot on the object. The approach with the stripe is shown in figure 2.1(b). It is also possible to work with more than one camera and laser on an object.

A disadvantage of the laser-based triangulation system is that the quality of the results is constrained by the quality of the surface material of the object. Properties like the shininess or subsurface scattering influence the reflection behavior of the projected laser light.

Time-of-Flight System

The time-of-flight laser scanner sends out a pulse of light and waits for it to return after the light is reflected by the object. With the round-trip-time of the light the scanner calculates the distance between the object and the scanner. Figure 2.2 shows the functionality of this scanner type.

The time-of-flight system has the same disadvantage like the triangulation system that it is dependent of the quality of the surface.

2.1.2 Range Image Registration

Usually more than one scan of an object is necessary to fully capture an object, and these scans must be merged together after the scanning. The process of putting these scans together in one coordinate system is called *registration*. The most often used methods for registration are presented in the following.

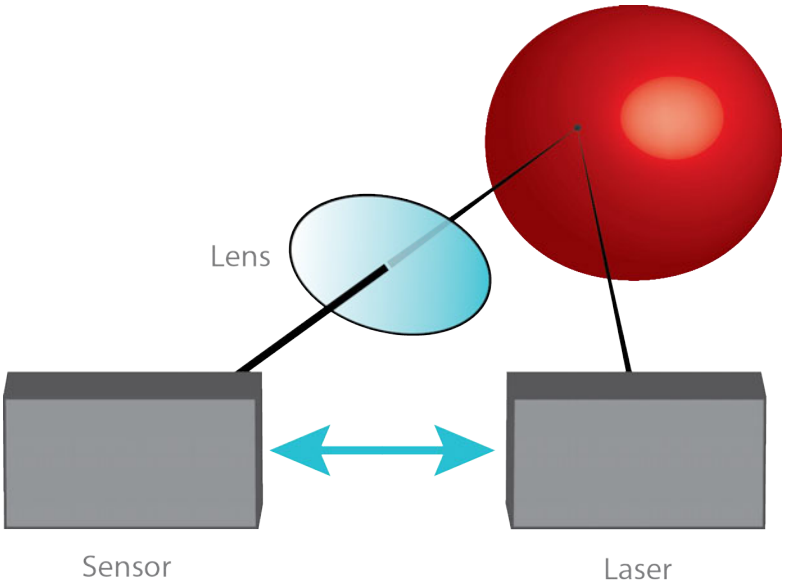
One possibility, used with high-end scanner, is that the scanner tracks its GPS[1] position and orientation precisely so the matching of the separate scans can be made easily. The registration process can be alleviated when at least four retro-reflective targets markers are placed at the object which is scanned. These markes must be visible from more than one scanposition [44]. The disadvantage of the method with markers is that the markers are then visible in the scan. These can indeed be removed but then some details of the model are missing.

If there is no data about the scanning setting available it is necessary to match the separate scans by their actual data. There are many approaches of how to match the scans automatically, for example the *Iterative Closest Point (ICP)* algorithm and various modifications of it [7, 10, 29, 47]. Here points from two different scans which correspond to the same surface point are found and then aligned.

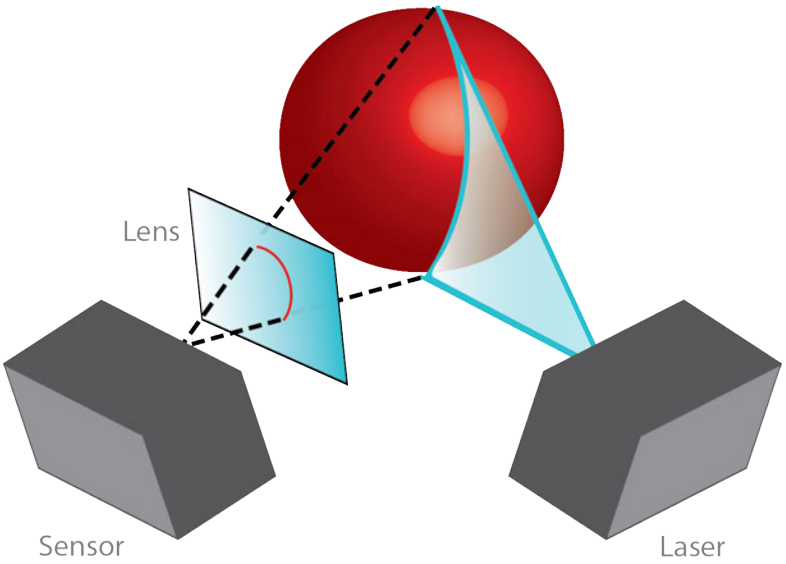
Silhouettes can also be used to align corresponding scans. The single scans have to be already converted to mesh which should be aligned later. Therefore the silhouettes appearing in the images taken are compared with those on the image obtained by rendering the meshes [28]. With that information the meshes can be merged together.

Manual registration of the images is also possible, where the user selects corresponding points of the model on the different images which where taken [37], or an automatic image-based approach where specific features are detected on the images taken during the scan.

[1] Global Positioning System



(a) Triangulation with a projected spot



(b) Triangulation with a projected stripe

Figure 2.1: Principle of a triangulation scanner

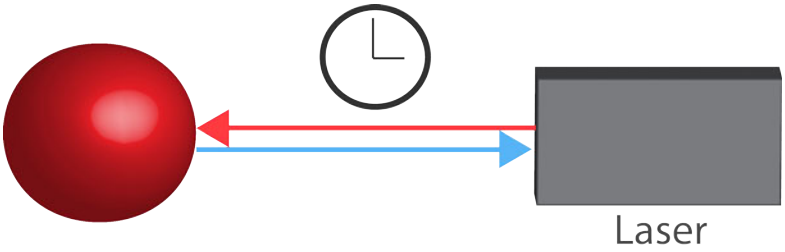


Figure 2.2: The round trip time of the laser beam is calculated.

Figure 2.3: 3D range scanner with a mounted camera [20].

2.1.3 Coloring of Range Images

With the recording and registration methods described earlier just the position of the points is calculated, the next step is to get information about the color of the points. Therefore a cameras is mounted on the scanner hardware and with that camera one or more pictures made are of the field of view of the scanner during or after the scan process. With the information about the position of the scanner and the camera the color can later be mapped on the 3d point cloud like a texture in a post processing step. Figure 2.3 shows a 3D scanner with a mounted camera which takes pictures during the scan process.

If there are no images made during the scan process but at some other time, there are some approaches which combine these images to generate a texture for the geometric model afterwards [31]. In this case the colorization of the model happens after the registration process. If there are already color images of the scanned object available, these can be used for the registration process [22, 26].

When the different scans are aligned a good basis for generating a single mesh out of the point cloud is given [29].

2.2 Range Image Data Structures and Rendering

The rendering of the obtained point cloud data sets is a challenging task. To visualize this data either a mesh can be generated or the points can be rendered individually. Especially for extremely big point clouds, like detailed scans of statues, buildings or cultural heritage acquisition, which exceed the limit of the main memory, the rendering becomes more complicated. To render these scenes it is necessary to think of new data structures, which are out-of-core compatible and ideally support some *level of detail* system. Here are some examples of rendering methods for very large point clouds.

2.2.1 Sequential Point Trees

Dachsberger et al. [14] are using a method to process the level of detail transitions on the GPU using *Sequential Point Trees*. The point set is put into an octree and different levels of detail are calculated. The normals and positions of these new points are the averaged values of their child nodes. Therefore nodes higher in the hierarchy represent a coarser level of detail than the lower ones. Dependent of the *geometric error* e_g the minimum and maximum distance values r_{min}, r_{max} are stored with the nodes and are used to decide which level of detail should be rendered for the current viewpoint.

2.2.2 Instant Points

Instant Points describes a data structure composed of Nested Octrees [49]. This algorithm is a modification of *SPT*, described in section 2.2.1, and introduces a new data structure, the so called *Memory Optimized Sequential Point Tree (MOSPTs)*. Furthermore a so called outer octree is used, which acts as a traversal structure and holds as nodes *MOSPTs*. The outer octree is used for view-frustum culling and the out-of-core handling, while the MOSPTs are used for efficient rendering and hold all actual points. Figure 2.4 shows the 2D version of a nested octree, a nested quadtree with a depth of two.

2.2.3 Layered Point Clouds

Gobetti and Marton [21] make an LOD tree out of an evenly spaced point set. The root node holds a coarse representation of the entire model. All other points are equally subdivided in two subtrees, where each node is a refinement of its parent node, the nodes also store the average spacing r_j between the samples. Later when the tree is traversed for rendering, the average sample distance of the nodes which pass view frustum culling, is projected on the screen to obtain the splat size. The points of each node that is visited during rendering traversal are rendered. When the splat size is under a certain threshold the points associated with the node are still rendered, but the traversal stops. Otherwise it continues the refinement with its children. Figure 2.5 shows two different LODs of Michelangelo's David.

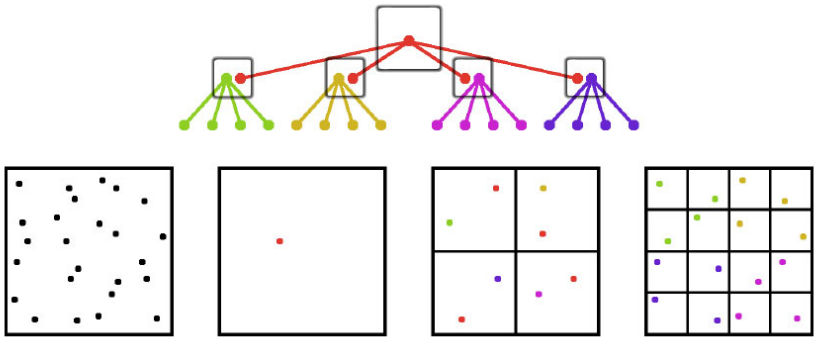
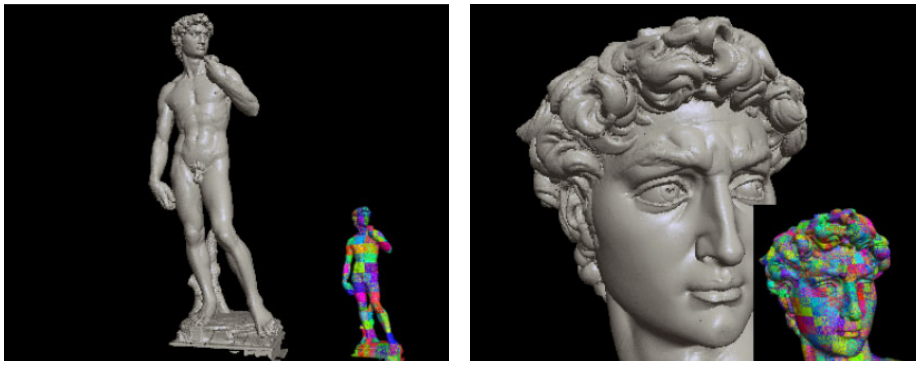


Figure 2.4: A nested quadtree with the depth of two [49].



(a) David with 4M samples. (b) David with 28M samples.

Figure 2.5: Two different LOD representations of David [21].

2.2.4 QSplat

Qsplat [40] makes use of a hierarchy of bounding spheres. When building the hierarchy, the bounding sphere of the complete model is recursively split into smaller spheres. The spheres must have must have the right sizes so that the nodes touch each other in a way that there are no holes during rendering. The nodes store the average information of their subtrees like their bounding spheres, their subnodes, normal and optionally a color. For rendering this tree, the nodes which pass the frustum culling are traversed until one node is a leaf node, or the size of the sphere projected on the screen exceeds a certain threshold. Figure 2.6 shows such a bounding sphere hierarchy with a branching factor of maximal four.

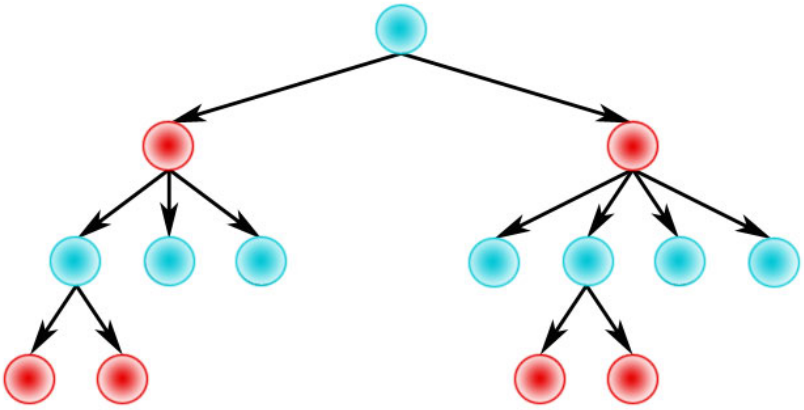


Figure 2.6: Bounding sphere hierarchy with a branching factor 4.

2.3 Color Adjustment for Geometric Models from Range Images

Not only color processing is an issue when processing point clouds [48], for topics like hole filling and outlier removal also a lot of research has been done. Unlike the hole filling and outlier removal which are handled in the post-processing step, the color is in most cases already matched during or directly after the registration process 2.1.2 [22, 26, 34].

The color matching can happen when separate colored point clouds are merged together or when coloring the already registered point cloud or mesh, with the color coming from a texture [11]. In that case the color processing can be done when the texture is generated.

Getting a smooth colored representation is a challenging task due to the different lighting conditions when the pictures are made. Light sources like the sun, a flash light from the camera, or other lights which were positioned around the object, influence the images taken. So when these images are merged together to color the model, visual discontinuities and artifacts are visible over the model. Figure 2.7 shows an example of a point cloud representation of the stephansdom floor colored by the unprocessed images which were taken during the scan process.

There are some approaches how to solve this problem and improve the visual appearance of the model.

2.3.1 Using the Color for the Registration

As mentioned in 2.1.2 the *Iterative Closest Point (ICP)* algorithm is a popular algorithm for the registration of multiple scans. This algorithm can be extended for improving the registration process by including the color values [22, 26]. Other algorithms like the *Projective Surface Matching (PSM)* [34] also use the color values to improve the registration process.

Bernardini et al. [8] use a geometric-based alignment of the different scans only and use an image-based alignment from high-quality images after that. The accuracy of the point pairs is improved by looking for the corresponding points in the local neighborhood of the images.

Figure 2.7: Visual discontinuities coming from the different lighting conditions are quite noticeable.

Using color during for the registration helps to decrease the registration error which also results in less color variations, especially when the colors from the different scans are blended together when generating a texture.

Figure 2.8 shows how using the image-based alignment in addition to geometry-based alignment improves the resulting image.

(a) (b)

Figure 2.8: Result image once without (a) and with (b) image-based registration [8].

2.3.2 Calculate Color from multiple Images

As the previous part was about the influence of the color on the point cloud registration this part is mostly about meshes, which were created from point clouds. Creating triangle meshes from

point clouds is a common task, especially for smaller models. For these meshes a texture has to be generated from the images taken during the scan process. For example if multiple parts of a model are merged together to one mesh, some parts of the model appear in more than one scan and therefore these parts appear in more than one image too. So there is a lot of different color information available for that one part. When the texture for the mesh is generated, the right color for these parts, which appear in more than one scan, has to be chosen from the different available images. There are some possibilities how to choose the color for these parts of the mesh.

If there are several images from different scan positions available for one query point, the final outcome of the color can be improved by checking from which camera positions the query point was visible and discard the occluded points from the color calculation.

One possibility to check whether a point was visible from a certain position or not is with a *z-buffer* approach. Here the mesh is rendered and a depth map of the scene from the camera position is made and then the depth values are compared in the depth map with the depth value of the point [29, 31], if the difference between those two values exceeds a certain threshold, the point is most likely occluded. An alternative method to the *z-buffer* approach is to use ray casting. [34] used coarse shadow volumes to detect possible occluded points from certain scan positions.

2.3.3 Image Combination

The colorization of one 3D point on a mesh which appears in more than one scan, can be calculated by blending the different color values together. The blending of the different colors can result in a smoother appearance of the whole mesh model than choosing one color which should be the most accurate for the query point. But blending different values together can also result in a blurry appearance.

There are some approaches which use the combination of various textures for the final color and therefore different weighting functions were introduced which should lead to a satisfying result. Johnson et al. [26] have first registred the different point clouds with the help of the color ICP and then generated a mesh out of these point clouds. Later they produce a texture out from the available textures for that model. They consider for example the normals of the different contributing points and calculate the dot product of them with the normal from the query point to gain the different weights for the individual contributing points. Figure 2.9 shows the weighted blending operation of different color values to one final color.

Neugebauer et al. [31] first calculate the visibility of one point from each camera position and then they build a weighting function with three weights. The first weight is calculated from the sampling density of the images, the second weight is for a smooth transition between images, and the third weight tries to eliminate outliers.

Other weighting factors can be the viewing direction or the feathering [33]. The feathering weight was introduced to correct the illumination differences on the silhouettes of scans.

Rankov et al. [35] worked on an algorithm for combining multiple textures to one big texture. Their work is just about the textures, with no 3D mesh belonging to it. They remove the color changes by blending the overlapping parts of the images. A look up table for the overlap-

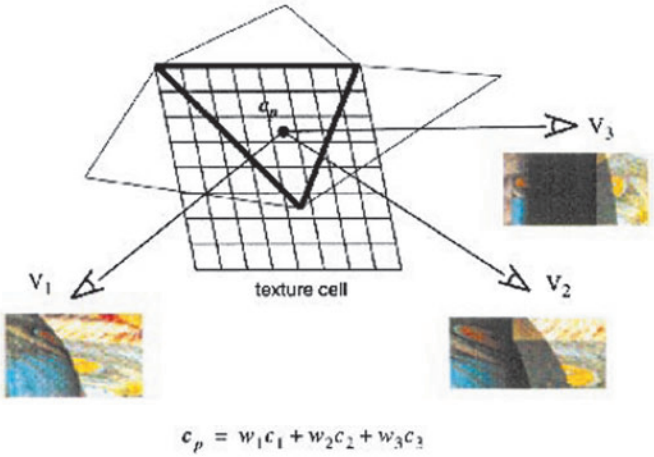


Figure 2.9: Calculate the color of a pixel with a weighted average of colors for that pixel. Image taken from [26].

ping region, filled with weights, is generated for each image. Figure 2.10 shows this blending procedure.

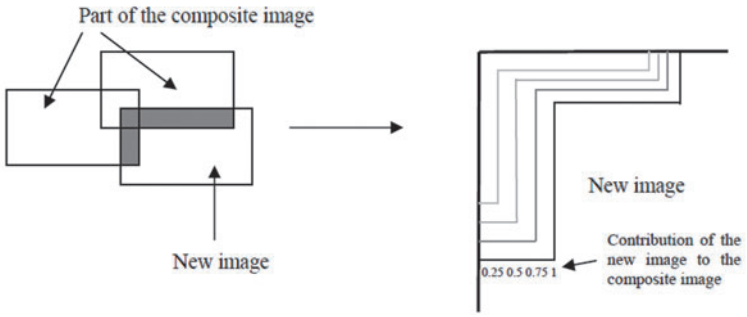


Figure 2.10: Blending procedure from Rankov et al. [35].

2.3.4 Texture Construction from Patches

Callieri et al. [11] use *texture averaging* for their meshes to remove color discontinuities and differences from the final texturing. They subdivide the model into patches and assign parts of the different images to these patches and try to keep these patches as large as possible. After that they try to remove the discontinuities by interpolating between the edges of the pathes. Figure 2.11 shows the outcome of their texture averaging algorithm.

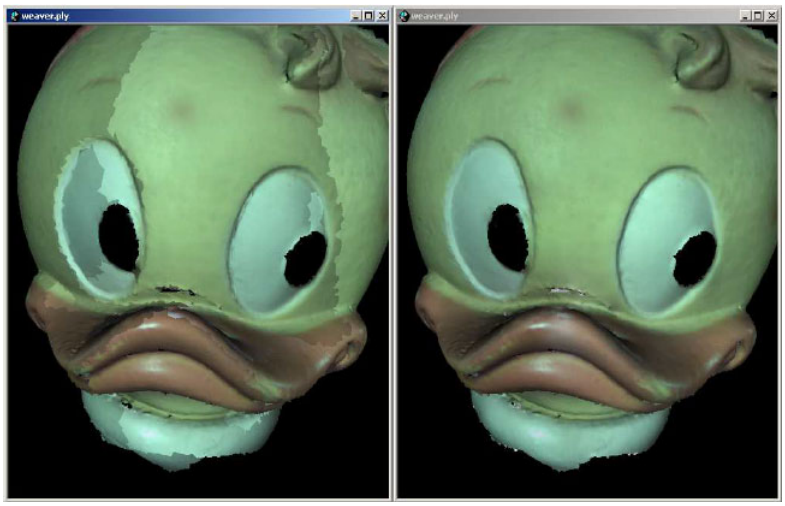


Figure 2.11: Using texture averaging removes large-area color discontinuities. Image taken from [11].

2.3.5 Removal of Lighting Artifacts from the Images

Other techniques try to solve the problem with the different lighting conditions by removing the artifacts occurring therefrom. Such artifacts can be the specular highlights, cast shadows and the different light intensities depending on the light direction.

One approach to improve the color appearance of the 3D model is by correcting artifacts evoked by flashlights, presented by Dellepiane et al. [15]. Here a *color correction space* is made, which means that there is a color correction matrix for each point in the camera frustum. These matrices differ for each camera. Then the colors from the images made with one camera can be corrected afterwards by transforming the pixel values to that color correction space. Equation 2.1 shows the transformation from the color (RGB) to the corrected color (R', G', B') with the color correction matrix c_{ij}.

$$\begin{pmatrix} R' \\ G' \\ B' \end{pmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \end{bmatrix} \begin{pmatrix} R \\ G \\ B \\ 1 \end{pmatrix} \tag{2.1}$$

Additional to that the relative position of the flash light is calculated and with the information about the position of the flash light and the color correction the visual artifacts like the specular highlights, by comparing the luminance of an point on the different photos, are removed. Shadows on the model are corrected as well by constructing a depth map for the flash light.

It is also possible to remove the lighting artifacts if at least three different images from one view are made, each one with a different, well known, lighting condition [37]. With a collection of different images which all contain the same points under different lighting conditions it is possible to detect shadowed pixels, but also pixels with a very low intensity, pixels with a spec-

ular highlight, or pixels with conspicuous different saturation values. The diffuse lighting term, as well as the normal, can then be calculated by solving a linear system, set up of the different pixel values under the different light directions. With those two factors, the diffuse term and the normal, the illumination-invariant color can be obtained. This method is just possible for small indoor objects where full control can be taken over the lighting scene.

Bannai et al. [5] used a different method to remove illumination differences between multiple images. They calculate color transformations from patches (kein Beistr) which should have the same color. The color transformations minimize the color error over all pixels in those patches. The patches are then transformed to have the same appearance. Later a global transformation for all overlapping pixels is calculated, based on a chosen image A with the best appearance. With this transformation all other images are transfered to the same color space as A.

Levoy at al. [29] took every image made of the scanned object twice, the first time with and the second time without their spotlight. The difference image between those two images shows the object as if it was only lit by the spotlight. By the generation of that image the different daylight conditions can be compensated.

2.4 Methods for Noise Removing in Images

In this section some image processing methods to reduce the Gaussian and impulse noise in images are described.

2.4.1 Linear Smoothing Filter

The *Mean* and the *Gauss Filter* are both linear smoothing filters. For the mean filter the value of a pixel is replaced by the mean values of the pixels neighborhood (including the pixel itself).

The gauss filter weights the different pixel values from the pixel as well as the pixels neighborhood with a gaussian function. For example for the *standard normal distribution* the weighting function would be

$$w(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} \tag{2.2}$$

where x is the distance between the pixel and its neighbor pixel.

2.4.2 Median Filter

For the purpose of signal smoothing Tukey [46] introduced the *median filter*. Regarding image processing the median filter should reduce the noise in the image but still preserve the edges. Therefore for every pixel in an image its value is replaced with the median of the neighboring pixels. The median is the middle value of an sorted list of odd numbers. For an multiband image this calculation has to be done for each band seperately which can lead to errors near the edges.

2.4.3 Bilateral Filter

The *bilateral filter* was introduced by [32] and named and extended by [45]. Bilateral filtering is a non-linear image processing filter. This filter combines a *domain* and a *range* filter. The domain part measures how geometrically close the points are and the range part measures how similar their values are. So the source pixel is then replaced by a combination of similar and close-by values. The weight for the bilateral filter can then be described as:

$$w(x,\xi) = c(\xi,x)\, s(\mathbf{f}(\xi),\mathbf{f}(x)) \tag{2.3}$$

where $\mathbf{f}(x)$ is the pixel value of image $\mathbf{f}$ at $\mathbf{x}$.

In this equation the geometric closeness between the pixel at value $\mathbf{x}$ and the neighbor point ξ is computed in $c(\xi,\mathbf{x})$. The photometric closeness is computed with $s(f(\xi),\mathbf{f}(x))$. These functions can be Gaussian functions or other functions where the weights can decreased to zero.

The Gaussian case of the spatial weighting function is:

$$c(\xi,x) = e^{-\frac{1}{2}\left(\frac{d(\xi,x)}{\sigma_d}\right)^2} \tag{2.4}$$

where $d(\xi,x)$ stands for the Euclidean distance between $\mathbf{x}$ and ξ.

The radial weighting function is then:

$$s(\xi,x) = e^{-\frac{1}{2}\left(\frac{\delta(\mathbf{f}(\xi),\mathbf{f}(x))}{\sigma_r}\right)^2} \tag{2.5}$$

where $\delta(\mathbf{f}(\xi),\mathbf{f}(\mathbf{x}))$ is the intensity difference between the two pixels.

This combination of range and domain filtering results in a filter that removes small differences between pixels caused by noise. So the overall image is smoothed but the edges are kept and one other advantage is that this filter does not only work for gray scale images but also for multi band images.

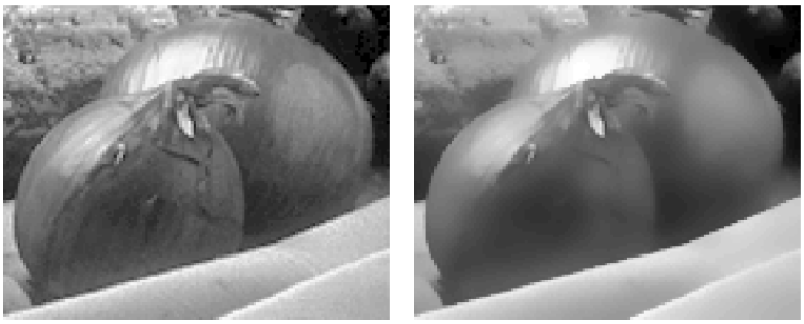
Figure 2.12 from [45] shows an example image before and after the bilateral filtering.

2.4.4 Modifications of the bilateral filter

Not only Gaussian noise can cause corruption in images, impulse noise is another type of noise which can appear in images. Impulse noise replaces pixel values of an image with some random values. One example of an impulse noise is the „Salt-and-Pepper“ noise where the noise-pixels are completely black or white. Figure 2.13 from [19] shows one example of such noise.

To remove not only Gaussian noise but also impulse noise Garnett et al. [19] extended the bilateral filter to a filter which can remove impulse noise as well. Therefore they created the ROAD-Statistic („Rank-ordered Absolute Differences“). For the ROAD calculation the m points

(a) Unprocessed Image (b) Bilateral Filtered Image



(c) Detail from 2.12(a) (d) Detail from 2.12(b)

Figure 2.12: Shows a picture before and after bilateral filtering, and a close-up of the onions in that picture. Images taken from [45]

Figure 2.13: Image with impulse noise. Image taken from [19]

which have the most similar pixel values as **x** out of the neighbor points are used. ROAD is defined as:

$$ROAD_m(\mathbf{f}(x)) = \sum_{i=1}^{m} r_i(\mathbf{f}(x)) \tag{2.6}$$

where $2 \leq m \leq 7$ and $r_i(\mathbf{f}(x))$ are the euclidian distances of the color values from the m neighbors to the value from **x**.

This statistic detects the m-Neighbors which are the closest to the pixel and shows how close they are to the pixel value, because impulse noise pixels always differ much from their neighbor pixels. And non-impulse-noise pixels, even those on the edges, should have at least four close neighbor pixels.

Besides the domain and range filter, Garnett et al. added a third filter to the weighting function, the „impulsive“ weight. But the impulsive weight cannot be added to the function straight away because the *range* part of the bilateral filter will detect edges where the *impulsive* part will detect noise. Some adjustments have to be made so that the *range* and *impulsive* help each other instead of blocking each other, and the weight *range* part of the filter is dependent of how „impulsive“ one pixel is. The „joint impulsivity“ of the pixel value **x** and the neighbor pixel value ξ is computed as:

$$J(x,\xi) = 1 - e^{-\frac{1}{2}\left(\frac{\frac{ROAD(\mathbf{f}(x))+ROAD(\mathbf{f}(\xi))}{2}}{\sigma_J}\right)^2} \tag{2.7}$$

This results in a new weight of the neighbor pixel ξ with respect to **x**:

$$w(x,\xi) = c(\xi,x)\, s(\mathbf{f}(\xi),\mathbf{f}(x))^{1-J(x,y)} i(\mathbf{f}(\xi))^{J(x,y)} \tag{2.8}$$

2.5 K-Nearest-Neighbor Search in Point Clouds

Finding k nearest neighbors *(kNNs)* in point clouds is a well discussed topic. Nearest neighbors are used for example for the calculation of normals, surface reconstruction, feature detection or for noise removal. For the simplification of the point cloud the count of the neighbors of a point and the local normal change are important [51] therefore the nearest neighbors of a point areneeded too.

2.5.1 *k*-Nearest Neighbor

The nearest neighbor is the point that is the geometrically closest point to a query point q from a point cloud $P = \{p_1, p_2, p_3, ..., p_n\}$. This geometrical closeness can be calculated with the euclidean distance between the points. The k-nearest neighbor points are then the k points which are closest to the query point. Therefore the k-nearest neighbor search can be defined as the search for these k neighbor points in the same point cloud P.

The simplest, but most costly, way to find the *k*-nearest neighbors would be a linear search. Therefore all points in *P* would be compared to *q*. The calculation time of the linear search for a point cloud containing *N* points with dimensionality *d* would be $O(Nd)$.

A widely used structure, and several variations of it, for accelerating the *k*-nearest neighbor search is to make use of an *kd-tree* [6] which performs the search in $O(logN)$ time.

To find the *k*-nearest neighbors, a priority queue, where the *k* nearest points so far are stored, is created. And then the tree is traversed, starting from the root node. Therefore a ball is created around the query point. The radius of this ball is dependent on the largest distance of the neighbor points found so far. A „bounds-overlap-ball“ and a „ball-within-bounds“ test [17] is made on each node visited to decide whether to continue recursive traversing of the tree or to stop. The traversing continues if the ball overlaps the node. When the ball is within the bounds of the node, the node is a perfect match and no other nodes need to be traversed any more. Nodes which do not overlap with or lie within the ball are completely discarded from the search, because no potential neighbors would be found in them. If the current node is a leaf node, all points within are tested as potential neighbors. Friedman et al. [17] made some adjustments to the kd-tree to minimize the number of nodes which need to be traversed. To accelerate the search-algorithm, when building the kd-tree, the axis with the largest spread is chosen as discriminator axis for the splitting of the node into sub nodes. The bucket size was considered as well.

Xiao and Huang [51] make the radius of the ball around the query point dependent on the point cloud density, so that approximately *k* points fit in the bounding sphere.

The algorithms mentioned previously are working fine when all points lie in the main memory. Sankaranarayanan et al. [42] proposed an *kNN* algorithm especially for very large point clouds. They used a hierarchical spatial data structure like an *R-Tree* or an *bucket PR-Tree* [41]. The tree structure remains in main memory while the leaf nodes, containing the actual point data, are stored on the hard disk. For finding the *kNNs* of point *t* the neighborhood of the node *N*, containing *t*, is created with the help of the distance estimates *MinDist* and *MaxDist*. Figure 2.14 shows these two distance estimates. This neighborhood, called *locality* of node *N*, contains all *kNNs* for its containing points. Later this locality is searched to find the actual neighbor points for point *t*.

2.5.2 Approximate Nearest Neighbor

Arya et al. [4] proposed approximate nearest neighbors for nearest neighbor search. An approximate nearest neighbor is a point which is a neighbor point of query point *t* whose distance to *t* lies within a $(1+\varepsilon)$ factor of the distance to the real neighbor point, where ε is a positive real number. The approximate nearest neighbor calculation has the most advantage when working with dimensions more than 8 [1], because in this case the computation of the exact nearest neighbors is a rather difficult task.

Muja and Lowe [30] compared the search performance of different algorithms for approximate nearest neighbor. Their conclusion was that either the *k-means* tree, proposed by Fukunaga and Narendra [18] or *multiple randomized kd-trees* [43] brought the best result, dependent on the dataset.

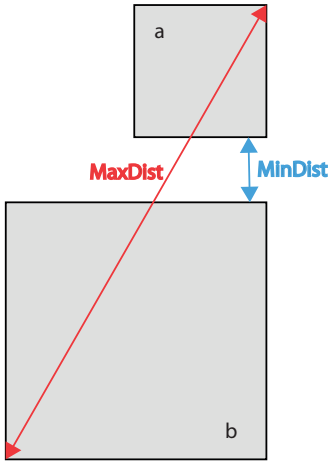


Figure 2.14: Distance estimates to find the neighborhood of node *N*.

2.5.3 *k*-Nearest Neighbor Graph

A nearest neighbor graph contains a set of points $\{p_1, p_2, p_3, ..., p_n\}$ and a set of edges, where one edge connects a point p_i with its nearest point p_j. In a k-nearest neighbor graph the point i is connected with the points N_i^k which is the set of k closest points to point i.

For example Connor and Kumar [12] show how to create a k-nearest neighbor graph by using Morton order [41] of points.

CHAPTER 3

Methodology for Image-Based Approach

The reason for irregular color of the points of 3D models in the outdoor area is that the lighting differs a lot on the pictures taken, because it depends on the daytime and the weather. Pictures taken at noon are often overexposed and some 3D points get a very light or white color. Dependent on the lighting is the lighting mood. The lighting mood of an image is the palette of colors used in the image. The lighting mood is dependent on the intensities of the colors, shadows and the specular effects on the objects. The lighting mood at noon is different to the lighting mood at sunset. When colorizing a point cloud of the outdoor area with these pictures, the different lighting moods in the pictures will lead to color inconsistencies across the point model.

The approach to solve this problem is to give all the pictures the same lighting mood, then the scene, no matter how many different scans and pictures are used for it, should be colored in one uniform style.

The color differences which are coming from the shadows as well as from the specular behavior of the material of the objects in the images cannot be affected later. The color differences between the images which are caused by the different color temperatures can be adjusted by using white or color balance. White balance is used to change the intensities of the colors on an image so that the are independent of the color temperature of the light source. When images are taken by a camera the white balance can be made automatically. Here the camera looks for a white or very bright spot in the picture and changes the other colors according to that spot. As an alternative to the automatic white balance a manual white balance can be made where the white spot is defined by the user of the camera.

The white balance settings differ from scan to scan. For some 3D scans the same white balance settings were used for all images of the object. On other 3D scans the same white color settings are only used per scan position. But the white balance settings for one set of images from one scan position are always the same, because the camera settings are always made once per image set. The images used in this section were made with the same manual white balance

settings. As a manual white balance was already made during the scan process the use of an automatic white balance in a postprocess step will be no use.

This chapter covers two approaches for solving that problem. First the color transfer in decorrelated color space from Reinhard et al. [36] and the color transfer in correlated color space from Xiao et al. [50]. These two papers are described in Section 3.2 and Section 3.3 respectively.

After the presentation of the two approaches it is tested if the images can be improved by using an automatic white balance algorithm.

3.1 Contribution

The idea behind this chapter is to bring all images to the same lighting mood so that they all look like they were taken at the same time of the day. This approach works well for images taken in the outside area like in Figure 3.8, except for images which are overexposed so that the color histogram of the image is only a narrow distribution of color values.

3.2 Color Transfer in Decorrelated Color Space

One color-correction method is to borrow one images' color characteristics and apply them to another picture. One example for this is to let a picture which was taken at noon appear like it was shot in the evening.

Usually these pictures are in the RGB color space. The three channels in this RGB space are correlated, which means that if a pixel color value should be changed, all three color channels have to be changed. To avoid this, the $l\alpha\beta$ space from Rudermann et al. [38] is used. In this color space the channels are only slightly correlated.

3.2.1 The $l\alpha\beta$ space

The $l\alpha\beta$ space is an advancement of the *LMS* color space[1]. In the $l\alpha\beta$ color space, the l channel is achromatic, the α channel represents the yellow-blue channel, and the β channel the red-green channel.

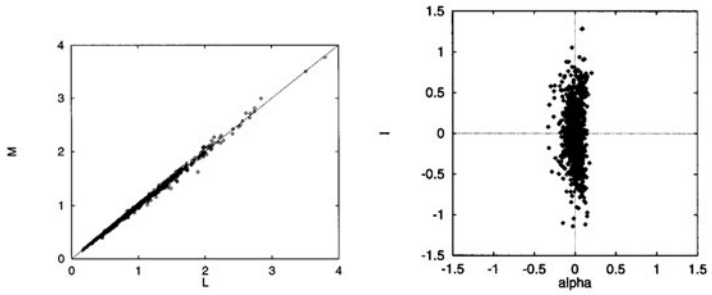
The *LMS* color space describes a color space which represents the response of the cones of the human eye on short, medium and long wavelengths. The following formula shows how to convert from the *RGB* color space to the *LMS* color space:

$$\begin{bmatrix} L \\ M \\ S \end{bmatrix} = \begin{bmatrix} 0.3811 & 0.5783 & 0.0402 \\ 0.1967 & 0.7244 & 0.0782 \\ 0.0241 & 0.1288 & 0.8444 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{3.1}$$

On scatter plots of L-M and L-S pairs all their data lies near their diagonals, which means that this data is strongly correlated, and the distribution along the diagonals is asymmetrical.

[1] http://en.wikipedia.org/wiki/LMS_Color_Space

This can be seen in Figure 3.1(a) where in the upper right corner are less points then in the lower left corner.



(a) L versus S (b) l versus α

Figure 3.1: Scatter plots for L-S pair in correlated color space and L-α pair in decorrelated color space. Images taken from [36]

Rudermann et al. [39] propose to transfer these channels in the logarithmic space (base 10) as this will improve the symmetry of the data points. The channel data is not bounded to have only positive values and can have negative values too.

$$\begin{aligned} \mathcal{L} &= \log L \\ \mathcal{M} &= \log M \\ \mathcal{S} &= \log S \end{aligned} \tag{3.2}$$

Next the axes have to be rotated so that they are decorrelated. The rotation is done with the orthogonal transformation technique principal-components analysis (PCA). This analysis has as result three orthonormal principal axes (l, α, β).

$$\begin{bmatrix} l \\ \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{3}} & 0 & 0 \\ 0 & \frac{1}{\sqrt{6}} & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & -2 \\ 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} \mathcal{L} \\ \mathcal{M} \\ \mathcal{S} \end{bmatrix} \tag{3.3}$$

3.2.2 Color Correction

For a correct color transfer, the source and the target image have to be transfered from *RGB* to $l\alpha\beta$ color space. The target image is the image whose color characteristics should be changed to those of the source image. Reinhard et al. [36] transfer the image to the $l\alpha\beta$ color space and continue with the calculation of the mean and the standard deviation along each axis for the two images.
The mean has to be subtracted from each data point:

$$
\begin{aligned}
l^* &= l - l^{mean} \\
\alpha^* &= \alpha - \alpha^{mean} \\
\beta^* &= \beta - \beta^{mean}
\end{aligned} \tag{3.4}
$$

where $l^{mean}, \alpha - \alpha^{mean}, \beta - \beta^{mean}$ are again the mean values of each channel in the $l\alpha\beta$ color space.

Next the data points have to be scaled by the standard deviation from the source image divided by the standard deviation from the target image:

$$
\begin{aligned}
l' &= \frac{\sigma^l_{src}}{\sigma^l_{tgt}} l^* \\
\alpha' &= \frac{\sigma^\alpha_{src}}{\sigma^\alpha_{tgt}} \alpha^* \\
\beta' &= \frac{\sigma^\beta_{src}}{\sigma^\beta_{tgt}} \beta^*
\end{aligned} \tag{3.5}
$$

where $\sigma^l_{src}, \sigma^\alpha_{src}, \sigma^\beta_{src}, \sigma^l_{tgt}, \sigma^\alpha_{tgt}, \sigma^\beta_{tgt}$ are the standard deviation values for the different channels of the source and target image.

The last thing is to add the already calculated mean to the result above:

$$
\begin{aligned}
l &= l' + l^{mean} \\
\alpha &= \alpha' + \alpha^{mean} \\
\beta &= \beta' + \beta^{mean}
\end{aligned} \tag{3.6}
$$

where $l^{mean}, \alpha^{mean}, \beta^{mean}$ are again the mean values of each channel in the $l\alpha\beta$ color space.

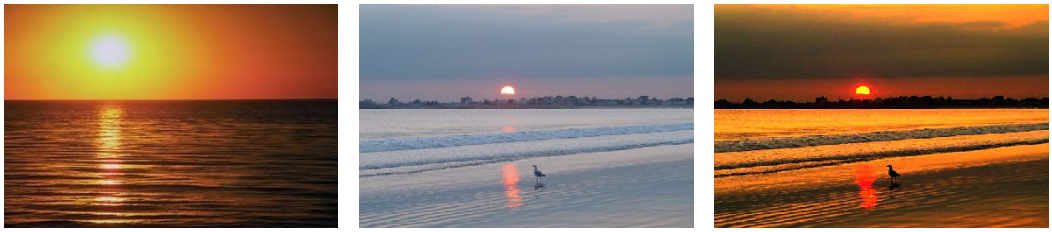
Reinhard et al. [36] finish the color transfer with the transformation from the $l\alpha\beta$ color space back to the *RGB* color space.

3.2.3 Results and Conclusion

The implementation of the Reinhard et al. [36] suggestion for color transfer works fine with the example images from their paper. Figure 3.2 shows an example of this implementation.

Figure 3.3 shows the color transfer algorithm used with pictures made during an outdoor 3D scan. As shown in Figure 3.3(c) the result image is not correct. The colors are too bright and glaring. Some other trials showed that this algorithm is not working for pictures that are not very colorful. These pictures use a small part of the histogram.

The next example, for Figure 3.4 the Reinhard et al. [36] color transfer algorithm is used in the *RGB* color space. Here the result image looks much different, but the result is still insufficient for an accurate color correction for the outdoor 3D scans.



(a) Source image (b) Target image (c) Result image

Figure 3.2: Color Transfer example pictures using the $l\alpha\beta$ color space.

(a) Source image (b) Target image (c) Result image

Figure 3.3: Images from an outdoor scan using the $l\alpha\beta$ color space

3.3 Color Transfer in Correlated Color Space

In the paper „Color Transfer in Correlated Color Space“ from Xiao et al. [50] an approach is presented which is similar to Reinhard et al. [36]. The results achieved with this approach are also similar to the approach from Reinhard et al. [36] but here an solution is found which solves the color transfer in *RGB* color space and avoids to transfer the colors into the $l\alpha\beta$ color space. To avoid the transformation in the $l\alpha\beta$ color space rotation is added additionally to the scaling and translating of the values in the color correction. The results of these two methods are very similar but tests showed that the color transfer in correlated color space produced a bit better results than the method in decorrelated color space.

Xiao et al. [50] consider a *RGB* value as three dimensional stochastic variable and an image is a set full of these variables. Here the characteristics of the source images are transfered to the target image with the calculation of a rotation, translation, and scale matrix.

The covariance matrix between the three components of the image has to be calculated as

(a) Source image (b) Target image (c) Result image

Figure 3.4: Images from an outdoor scan using the *RGB* color space

well as the mean of the pixel values along each axis. To obtain the rotation matrix the covariance matrix has to be decomposed with the *SVD* [27] algorithm.

3.3.1 Color Correction

In this approach statistics are used for applying the characteristics of one image to another one. So the first step of the method of Xiao et al. [50] is to calculate the mean along each axis for the source- and target image as well as the covariance matrix between the three components of an *RGB* value for the whole picture.

The covariance matrix shows the covariances of the three axis. This matrix is always symmetric and positive semidefinite and its diagonal consists of the eigenvalues.

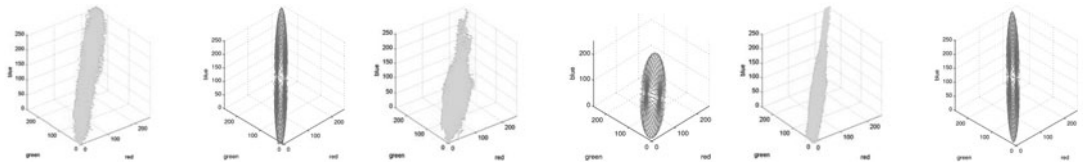
This approach is about morphing ellipsoids to fit another one. Figure 3.5 shows that the ellipsoid of the result image, Figure 3.5(c), is the ellipsoid from the target image morphed so its characteristics match the characteristics of the the ellipsoid of the source images. The ellipsoids are defined by the mean, which gives the center coordinates, and the covariance, its eigenvalues and vectors are responsible for the length and orientation.

The next step is to decompose the covariance matrix with the *SVD* algorithm, the result are three matrices U, Σ,V.

$$Cov = U * \Sigma * V \tag{3.7}$$

U and V are orthogonal matrices, these represent the eigenvectors of Cov. Σ is the diagonal matrix and contains the resulting eigenvalues:

$$\Sigma = diag(\lambda^R, \lambda^G, \lambda^B) \tag{3.8}$$



(a) Source image (b) Target image (c) Result image

Figure 3.5: Morphing the ellipsoid of the target image to fit the source images ellipsoid. Images taken from [50]

where $\lambda^R, \lambda^G, \lambda^B$ are the eigenvalues of Cov.

Then the transformation matrix can already be calculated out of the rotation, scale and translation matrices:

$$I = T_{src} * R_{src} * S_{src} * S_{tgt} * R_{tgt} * T_{tgt} * I_{tgt} \tag{3.9}$$

I is a 1x3 matrix with the RGB values. Now the rotation, scale, translation and the I matrices are calculated.

$$R_{src} = U_{src} \qquad R_{tgt} = U_{tgt}^{-1}$$

$$S_{src} = \begin{bmatrix} s_{src}^r & 0 & 0 & 0 \\ 0 & s_{src}^g & 0 & 0 \\ 0 & 0 & s_{src}^b & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad S_{tgt} = \begin{bmatrix} s_{tgt}^r & 0 & 0 & 0 \\ 0 & s_{tgt}^g & 0 & 0 \\ 0 & 0 & s_{tgt}^b & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{src} = \begin{bmatrix} 1 & 0 & 0 & t_{src}^r \\ 0 & 1 & 0 & t_{src}^g \\ 0 & 0 & 1 & t_{src}^b \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_{tgt} = \begin{bmatrix} 1 & 0 & 0 & t_{tgt}^r \\ 0 & 1 & 0 & t_{tgt}^g \\ 0 & 0 & 1 & t_{tgt}^b \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.10}$$

The elements of these matrices are the following:

$$
\begin{aligned}
t^r_{src} &= R^{mean}_{src} & t^r_{tgt} &= -R^{mean}_{tgt} \\
s^r_{src} &= \sqrt{\lambda}^R_{src} & s^r_{tgt} &= \frac{1}{\sqrt{\lambda^R_{tgt}}} \\
t^g_{src} &= G^{mean}_{src} & t^g_{tgt} &= -G^{mean}_{tgt} \\
s^g_{src} &= \sqrt{\lambda}^G_{src} & s^g_{tgt} &= \frac{1}{\sqrt{\lambda^G_{tgt}}} \\
t^b_{src} &= B^{mean}_{src} & t^b_{tgt} &= -B^{mean}_{tgt} \\
s^b_{src} &= \sqrt{\lambda}^B_{src} & s^b_{tgt} &= \frac{1}{\sqrt{\lambda^B_{tgt}}}
\end{aligned} \tag{3.11}
$$

where $R^{mean}, G^{mean}, B^{mean}$ are the mean values of all pixels from an image on the axes.

3.3.2 Results and Conclusion

The implementation was made with *C++* and *OpenCV*, the *Open Computer Vision Library*[2]. *SVD* is for example already implemented in OpenCV.

Applying the approach from Xiao et al. [50] on the images from the scans produced the best results. This algorithm also works with pictures that are not very colorful and have very unbalanced histograms. One result of this algorithm is shown in Figure 3.6. The source and target images are the same as in Figure 3.3 and Figure 3.4, but the result image shown in Figure 3.6(c) was generated with this algorithm. This image has not too bright parts like in Figure 3.3(c) and is more like the source image compared to Figure 3.4(c).

Figure 3.7 shows the three result images produced with the algorithm from Reinhard et al. [36] and Xiao et al. [50] for direct comparison. In that comparison it is apparent that the color transfer method from Reinhard et al. [36] and Xiao et al. [50] produce almost the same result image. The outcome of the color transfer in the correlated color space is a bit better than the result from the decorrelated color space because some parts in Figure 3.7(a) appear very bright, almost glowing, which does not happen in Figure 3.7(c). Figure 3.7(b) has an overall uniform appearance and would appear as good result, but when this image is compared with the target image it is noticeable that the color transfer from the source to the target image did not succeed because the result image is more desaturated than the source image and does not contain the lighting mood of the target image at all. The best indicator for that is the color of the sky and the green area in that image.

The best result would be if the green bushes in the source image have the same color as in the target image, this is none of the three result images the case. But as the result image made with the algorithm from Reinhard et al. [36] in *RGB* color space produces too desaturated results, the color transfer method from Xiao et al. [50] is used for further tests.

[2] http://opencvlibrary.sourceforge.net/

(a) Source image (b) Target image (c) Result image

Figure 3.6: Images from an outdoor scan using the statistic based approach.

(a) Result image from the decorrelated color space approach in the $l\alpha\beta$ color space.

(b) Result image from the decorrelated color space approach in the *RGB* color space

(c) Result image from the correlated color space approach.

Figure 3.7: Result images from Section 3.2 and Section 3.3.

3.4 Enhancement to the Color Transfer Method

3.4.1 Source Image for the Color Transfer

3D scans produce multiple color images and therefore the question arises which image should be used as source image. To get good result images it may be the better way not to just choose one representative image in terms of scenery and color palette image as source image but to produce a source image from all the target images. One example for such a source image is displayed in Figure 3.8.

Figure 3.8: Source image produced from a set of images.

For this, an average image of all target images is produced. That means the mean RGB value for every pixel is calculated. If there are many overexposed images in the set of target images, to include them in the calculation of the source image can distort the final result after the color adjustment. So it is better to weight them less or even discard them for producing the source image. Figure 3.9 shows such source images created with different weighting of the overexposes images. Figure 3.10 shows how this weighting has an influence on the resulting images. In Figure 3.10(c) the overexposed images were completely discarded, this produced the best results.

To detect if an image is overexposed the histogram of the image is evaluated. If more then 10% of all pixels in all three channels are in the upper or in the lower end of the histogramm, the image is rated as overexposed image.

3.5 Automatic White Balancing

As already mentioned white balancing is used to adjust the color intensities of an image so that these are invariant to the illumination color temperature. For the automatic white balancing the

(a) Source image 100% (b) Source image 50% (c) Source image 0%

Figure 3.9: Source images created where overexposed images were weighted differently. They were weighted with 100% (a), 50% (b) or were discarded (c).

(a) Result image 100% (b) Result image 50% (c) Result image 0%

Figure 3.10: Result images from the use of the source images from Figure 3.9.

method from Huo et al. [24] was used. This algorithm uses a local automatic white balancing algorithm where only gray color points are used for the color temperature estimation. In comparison to local white balancing algorithms global automatic white balancing algorithms use all pixels for the color temperature estimation.

In the paper from Huo et al. [24] the image is first converted to the *YUV* color space[3] where the *U* and *V* channels, the two chrominance components, are used to find the gray color points. The transformation to the *YUV* color space is shown in equation 3.12.

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.299 & -0.587 & 0.886 \\ 0.701 & -0.587 & -0.114 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{3.12}$$

All pixels in the image which fulfill the following criterion:

$$F(Y,U,V) = \frac{(|U|+|V|)}{Y} < T \tag{3.13}$$

where T is a defined threshold. In the implementation for this thesis the value for the threshold is set to 3.

When gray pixels are found the mean values for *U* and *V* channels, $|U|, |V|$ of the gray pixels are calculated and then the two mean values are compared to find out which channel in the *RGB* color space needs adjustment. If $|U|$ is bigger then $|V|$ then the blue color channel needs adjustment and $|U|$ is used as error value. Otherwise the red channel needs adjustment and $|V|$ is used as error value.

If this error value is under a certain threshold the calculation of the automatic white balance is finished. Otherwise the error value is scaled so that the color of the pixels is changed smoothly.

The error value is used to calculate the transformation of the *RGB* pixel values. This is done by:

$$g_i = g_i - \varphi \tag{3.14}$$

where *g* is the gain vector for adjusting the color values in the *RGB* color space and *i* indicates the color channel which needs to be adjusted. φ is the weighted error value.

The gray color pixel estimation and the following adjustment of the color values is repeated until the error value is under a certain threshold or after a certain number of iterations.

3.5.1 Results

The adjustment of the automatic white balance algorithm changes only a little in the images. One image before and after the automatic white balance is shown in Figure 3.11. So the automatic white balance is not applied on the images in this thesis because when these images were taken a manual white balance was already set for the camera. And so the automatic white balancing does not really improve the images.

[3] http://en.wikipedia.org/wiki/YUV

(a) Original image

(b) Image after automatic white balancing.

Figure 3.11: Images before (b) and after (a) automatic white balancing.

3.6 Color Transfer of the Images for the Point Cloud

When the image-based approach is used for a point cloud the number of image which should be processed is dependent of the quality of the images. Generally all images which are used to color the point cloud should be processed to get a point cloud with a uniform coloring. But if for example, the scans of three different scan positions are used for one point cloud and the color characteristics in the images from two scan positions are the same and only the images from the third scan position differ, then the color characteristics should only be transfered to the images from the third scan position.

For the user the best way of using the color transfer method for the images which are used to color the point cloud, would be to use the same target image for all images. But to use one source image for all images results in a lot of completely wrong colored images, especially for images taken in the outside area. To use one source image for all images from one scan position even results in wrong results. Therefore the best way is to manually choose the image which should be used as source image for a specific subset of all images which is quite time consuming and it is possible that no image can be found to improve the color characteristics of the image.

If the source image is generated out of the target images it is important that not too many target images are used because that can lead to a source image which generates desaturated result images.

In Chapter 5 two applications of the image based approach are shown.

CHAPTER 4

Methodology for Point-Based Approach

The following chapter deals with color adjustment for already colored point clouds. This method was implemented for the *Scanopy Rendering System* [49]. As already mentioned, the starting point for this processing step is a point cloud merged together from points of separate scans, taken from different scan positions. The only information available are the different scan positions and the information which point belongs to which scan position. No other information about the scanning setting, like the time or the light situation, is available. Therefore most of the approaches to correct color discontinuities proposed in Section 2.3 cannot be applied here.

For the method in this thesis the neighbors of the points play an important role. In *Scanopy* the data structure used for the storage and rendering of points is the *Modifiable Nested Octree* [49]. The problem with this structure is that the neighbor points can lie in the different levels in the hierarchy, which represent different *levels of detail*. The process of finding point neighbors in this data structure is therefore quite time consuming because a lot of nodes have to be visited to find the true nearest neighbors. To accelerate the process of finding the neighbor points of each point, a separate data structure is integrated into Scanopy. For that purpose a kd-tree was chosen.

In this chapter the calculation of the new color for a query point according to its neighbor points is described first. For this, the different neighbor point selection criteria are explained. Then the weighting function for the final point color is presented as well as the different filters which can used to blend the neighbor point color values with the query point color value together.

As the k-nearest neighbors are needed for the point-based color adjustment the kd-tree, the data structure which is used for the nearest neighbor search, is described in the second part of this chapter.

4.1 Contribution

For this part of the thesis the contribution is the following: An approach to automatize the color correction of the individual points of the point cloud and change the color of the points so that an overall smooth appearance is created. The color noise is removed with a set of different filters applied to the point cloud. This happens with the help of the nearest neighbors in object space.

4.2 Color Adjustment on Point Clouds

The *k*-nearest neighbors for each point can be used to change the color of the query point. The goal of this step is to remove noise of the coloring as well as to smooth the color discontinuities coming from the different lighting settings during the scanning of the model.

This step can be divided in two parts. First the neighbor points which influence the color of the query point are chosen carefully from the previously found *k*NN. Then these neighbor points are weighted on how much they should influence the color of the query point. The final color of the query point is then calculated by blending the weighted color values of the query point and the neighbor points together. Here different blending functions can be used. The optimal result would be that noisy points would be removed, and the overall appearance of the point cloud would be smoother, while the edges should remain sharp.

4.2.1 Selection criteria for choosing neighbor points

Not all neighbor points around the query point may help to improve the color of the query point. So some selection criteria are made to sort these neighbors out which are probably not „good“ neighbor points. When the *k*NN points are searched, some criteria are added to the search. With some of these criteria, points around the query point are not even put in the *k*NN queue.

- Number of neighbors
- Maximum neighbor point distance
- The position of the scanner / camera from which the color was acquired.

Number of Neighbor Points

The number of neighbor points, *k*, is the number of sought-after neighbor points and therefore the maximum number of points around the query point which can be used to change the color of the query point. This number has to be chosen carefully because it can make a big difference. If the number is too small it can happen that changing the color of the query point has almost no effect. On the other side if the number of neighbors is too big it may occur that details get smoothed out or that the result looks too blurry because too many color values get averaged.

Maximum distance of a neighbor point

If only a few points are available in some regions and one or more of the k nearest neighbors would be far away of the query point, this can lead to errors in the recolorization. To avoid including points which are far away, it is useful to define a maximum distance up to which neighbor points are considered for the recoloring.

Distance to Scan Position Influence

In [11] it is described that the camera which should capture the point of interest best should be the one which points towards the point along the normal of the point. For recoloring the points, the relationship between the scan position and the point is important too. When choosing the neighbor points which influence the color of the query point, a better result can be expected if these neighbor points are captured from a scan position with a good view on the area of the query point. Therefore the points in the neighborhood of query point p which got their color from the camera with the best view of p are preferred for recoloring p.

Figure 4.1 shows an example of a situation where the scan position should influence the choice of the neighbor points. The neighbors around the query point come from two scan positions. First the distance from the query point to the scan position is measured to find out which scan position is closer. In this figure scan position 1 is much closer to the query point than scan position 2, so it is assumed that the colors of the points coming from that scan position are more accurate than the colors from the points of scan position 2. So if there are enough points available from scan position 1 these should be chosen as neighbor points instead of the points from scan position 2.

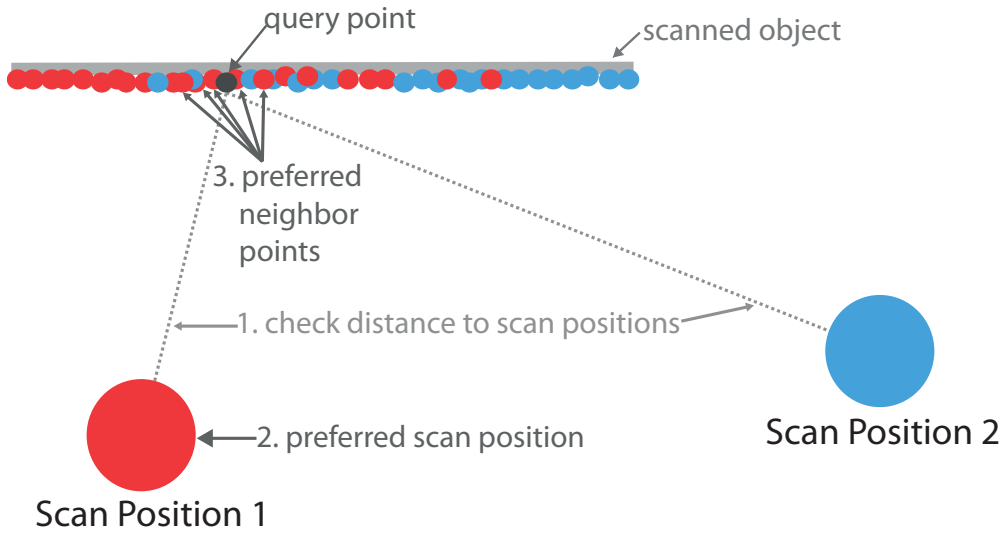


Figure 4.1: A 2D-representation of a point cloud scanned from two scan positions. The red dots come from scan position 1 and the blue dots come from scan position 2. Around the query point lie points from both scan positions.

4.2.2 Point Normal Influences

With the selection criteria for choosing the neighbor points a set of neighbors with a potentially good color are chosen for the recoloring of the point. For the selection criteria only the point position was considered. The next step is to consider the point normals too. For example if the neighbor point belongs to a different surface than the query point and therefore the normals of the neighbor point and the query points look in totally different directions, then the probability that the colors of these two points are not similar is very high.

So these two factors are used to calculate a weight which defines how much influence the neighbor point color should have on the final point color according to its normal:

- The relation between the normal of the query point and that of the neighbor point.
- The scan direction point normal relationship.

The weight calculated with these two factors is then integrated in the different blending functions which are used to calculate the new color for the query point. If an image based filter is used which weights the different neighbor point colors then this weight is combined with the weight coming from the normal calculations.

Normal Influence

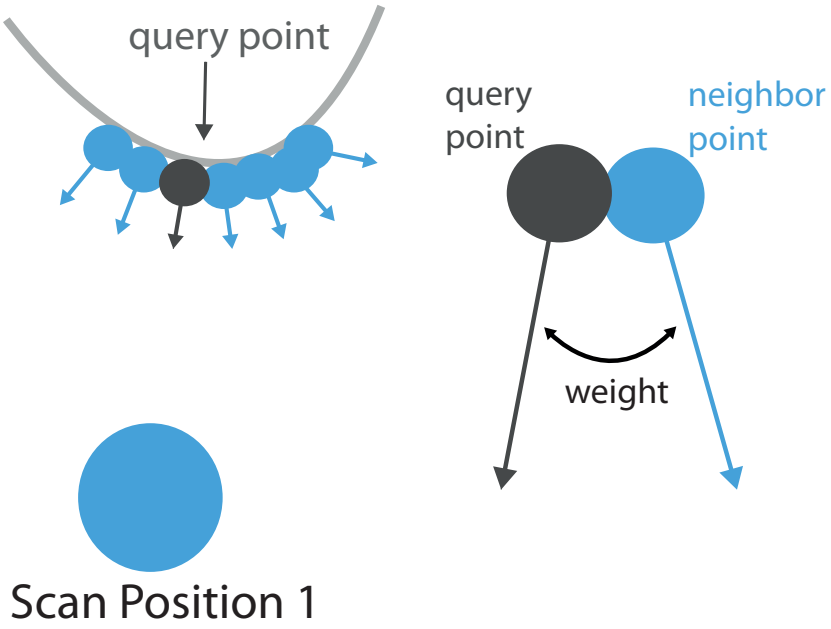
If the positions of two points are very close it does not mean that the colors of those points need to be similar. The color of a point depends on the surface attributes. For example if the object has a bumpy surface, the normals of close points can look in different directions, which causes different lighting effects on the object. Figure 4.2(a) shows a curved surface with points that are close to each other but whose normals look in different directions. If the surface is smooth, the points will look in the same direction and the illumination is smoother. To regard the surface normal is a good way to find out about the surface quality. Figure 4.2(b) shows the direct comparison of the two normal vectors. A weight is calculated from the difference of those two vectors.

If the normal of the point looks in the same direction as the normal of its neighbor point then it can be assumed that the points have a similar shading and that the possible color differences are not coming from the shading.

So the difference between the normal of the query point and those of the neighbor points is considered too by calculating a weight. This weight and other weights, calculated later in the recoloring process, are responsible for the influence of the neighbor color to the final point color.

This weight is calculated by the *dot product* between the normal of the query point and the normal of the neighbor point. The dot product is clamped to the range between zero and one. This linear weighting function is given in Figure 4.3.

As an alternative the dot product can be weighted with a Gaussian function, the visualization for that function is given in Figure 4.4 .



(a) (b)

Figure 4.2: This image shows that the normals of points which are very close to each other can look in different directions. The difference between those two normals is then weighted.

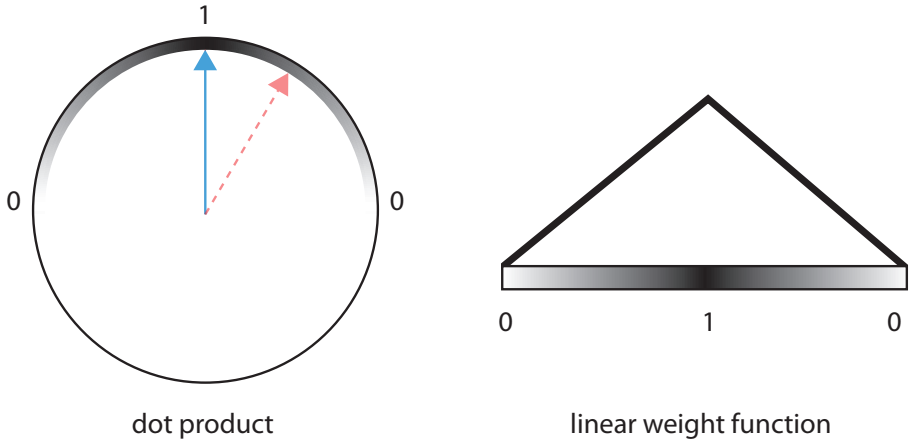


Figure 4.3: Linear weight for the dot product between the query point normal and the neighbor point normal.

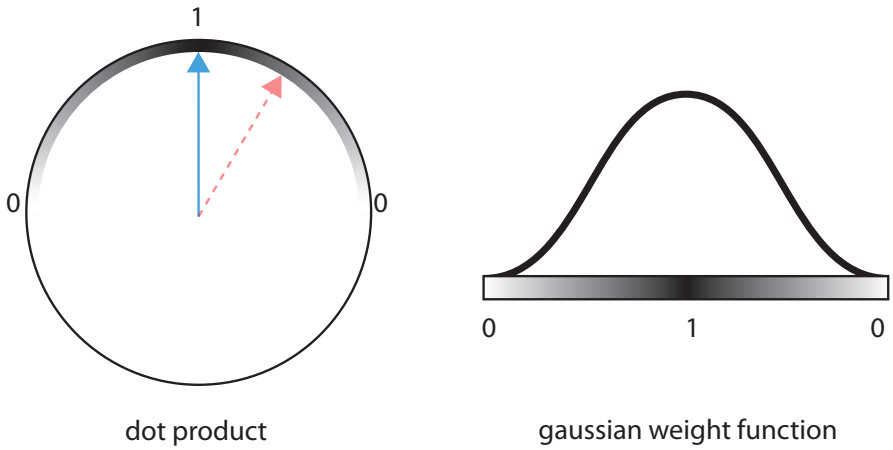


Figure 4.4: Gaussian weight function for the dot product between the query point normal and neighbor point normal.

Scan Position - Point Normal Relationship Influence

As third factor the relationship between the neighbor point normal and the vector between the neighbor point and its scan position is considered. If the angle between the neighbor point's normal and the vector to its scan position is larger than 90 degrees, the color of the neighbor point will corrupt the blended color for the query point. An example for such a situation is shown in Figure 4.5.

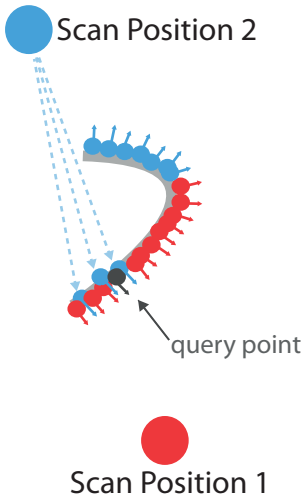


Figure 4.5: The blue points were scanned from scan position 2 but their normals look to the opposite side, like the red dots. That is an indication for the fact that the color can differ much from the surrounding red points, which were colored from the view point.

Therefore this is an additional weight and can be calculated like the weight for the query point normal and neighbor point normal relationship shown in Figures 4.3 and 4.4.

4.2.3 Blending functions for the final point color

If there are neighbors available for a query point, which were chosen according to the qualities described in Section 4.2.1, the final color for the query point has to be calculated. For this several methods are available. These methods are taken from image processing, especially the latter four methods are nonlinear filters used for noise reduction. These image processing methods produce weights which are combined with the weights coming from the calculations presented in Section 4.2.2. These four methods were already specified in 2.4.

As these blending functions for the final point color come from image processing some adjustments have to be made so that these filters can be applied on the point cloud. For this, first the geometric distance between the points and the colors of the points needs to be calculated for example for the gauss and the bilateral filter. The bilateral and trilateral filter need the distance between the intensity values of the query point and the neighbor point too. If these distances are known the weight for the point color coming from the specific filter and therefore the final weighting function for the final color of the query point can be calculated.

Geometrical distance of points

When calculating the distance between the query point and one neighbor point, the Euclidean distance is used. That means the calculation for the distance d is then:

$$d_{(q,n)} = \sqrt{(q_x - n_x)^2 + (q_y - n_y)^2 + (q_z - n_z)^2} \tag{4.1}$$

where q is the position of the query point and n is the position of the neighbor point.

Color distance of points

Calculating the difference in the color values between the query point and a neighbor point can be done by using the Euclidean distance (see equation 4.1). Instead of the x, y, z values the r, g, b are used.

As an alternative to the Euclidean distance another distance can be used for measuring the difference between the two intensity values. For this, a color metric method presented from *CompuPhase* is used which calculates the color difference like the human eye perceives it [2].

The calculation for this color distance is then:

$$r_{mean} = \frac{cq_r + cn_r}{2} \tag{4.2}$$

$$\Delta r = cq_r - cn_r \tag{4.3}$$

$$\Delta g = cq_g - cn_g \tag{4.4}$$

$$\Delta b = cq_b - cn_b \tag{4.5}$$

$$\Delta c = \sqrt{\left(2 + \frac{r_{mean}}{256}\right) * \Delta r + 4 * \Delta g^2 + \left(2 + \frac{255 - r_{mean}}{256}\right) * \Delta b^2} \tag{4.6}$$

where cq_x are the color channels of the query point and cn_x are the color channels of the neighbor point.

Calculation of the final color with a weight function

If the gauss, mean, bilateral or trilateral filter is chosen for the color adjustment the final color of the point is calculated with a weighting function. This weighting function is used because the mentioned filters are used to calculate a weight for each neighbor point. This weight defines how much influence the neighbor point color should have on the final color for the point.

As already mentioned in Section 4.2.2, if the relation between the query point normal and the neighbor normal or the neighbor normal and its scan position is considered, a weight depending of the normals of the points is calculated. When a specific filter is chosen for blending the colors of the neighbor points with the query point, the weight coming from considering the normals of the points is integrated in the filter. That means the final weight for a neighbor point color is the weight from the image based filter with respect to the geometry of the object. The final weights of the neighbor points are summed up so that the new color for the query point can be normalized by the sum of weights.

When a filter like the median filter is used which does not weight the different neighbor point colors, then the weight coming from normal calculations is ignored too.

For the calculation of the final color with the gauss or mean filter first the colors of the neighbor points are added up according to their final weight. Then the color of the query point is added to the function too. The query point color is also weighted from the filter and the weight coming from the relationship between the point normal and the vector to the scan position is added too.

So the final coloring of the query point can be:

$$c_{filtered}(q) = \frac{1}{w_{sum}(q)} \left(\left(\sum_{i=1}^{k} c(n_i) * nw(n_i) * fw(n_i) \right) + c(q) * nw(n_q) * fw(n_q) \right) \tag{4.7}$$

with the normalization:

$$w_{sum}(q) = \left(\sum_{i=1}^{k} nw(n_i) * fw(n_i) \right) + nw(n_q) * fw(n_q); \tag{4.8}$$

Here $c(q)$ is the color of the query point, $c(n_i)$ the color of the i-th neighbor point, $nw(n_i)$ the weight according to the calculations around the normal of the neighbor point (4.2.2) and $fw(n_i)$ the weight coming from the used filter. The term $nw(n_q)$, where n_q is the normal of the query point, is only the weight for the query point coming from the relation to the scanposition. $fw(n_q)$ is the filter weight for the query point.

When the bilateral or trilateral filters are used, the query point color is already considered when the intensity differences are calculated. The color of the query point is therefore not additionally added. Then the calculation of the final color is:

$$c_{filtered}(q) = \frac{1}{w_{sum}(q)} \sum_{i=1}^{k} c(n_i) * nw(n_i) * fw(n_i) \tag{4.9}$$

with the normalization:

$$w_{sum}(q) = \sum_{i=1}^{k} nw(n_i) * fw(n_i) \tag{4.10}$$

With the information about the geometric distance between points and the distance of the intensity values different filters can be used to blend the color values together. The different blending modes which can be used for recoloring the query point from the neighbor points are described below.

Mean Filter

For the mean filter the mean of all color values (including the color value of the query point) is used. So the weight for this filter is:

$$fw(n_i) = \frac{1}{(k+1)} \tag{4.11}$$

where k is the number of neighbor points for the query point. It is $k+1$ because the query point color is used too.

Gauss Filter

The gauss filter is used to weight the different points according to their distance to the query point because points which are farther away should have less influence on the final point color. For this, the filter generates a weight where the distance between the query point and the neighbor point is weighted with the standard normal distribution. The weight calculation for one neighbor point is therefore:

$$fw(n_i) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}d^2_{(q,n)}} \tag{4.12}$$

where d is the Euclidean distance between the query point position and the neighbor point position.

Median Filter

To remove noise in the visual appearance of the point cloud the median filter is used. The median filter is a simple noise removal filter which takes the median of all color values available for the final point color. For this, no weight needs to be calculated for the median filter. Here the *rgb* color values of the query point and of the neighbor points are put in a list which is then sorted. This is done for each channel separately. Then the *rgb* color values of the query point are replaced by the median values of the sorted list.

Minimum Filter

Here the sorted lists for the *rgb* color values are used too. In contrary to the median filter, for the minimum filter the smallest color values are taken for the query point.

Maximum Filter

For the maximum filter the highest values of the sorted *rgb* color values lists are taken.

Bilateral Filter

In this thesis the bilateral filter from Tomasi et al. [45] is used. This filter technique is already described in detail in section 2.4.3. With the use of this filter it is tried to remove gaussian noise by considering the intensity distances. As for the gauss filter the geometric distance of the neighbor point is weighted too. So the final weight of the filter is composed of the geometric distance and the intensity distance between the query point and the neighbor points.

When the weight from the normal calculations in Section 4.2.2 is added to the weight from the bilateral filter the final weighting function equates to a cross bilateral filter [16].

In this version the Euclidean distance or the color metric distance, described in section 4.2.3, can be used for the distance of the intensity values of the points. The additional weight coming from the normal calculations is added too.

For this, the weight is calculated by :

$$fw(n_i) = e^{-\frac{1}{2}\left(\frac{gd_{(q,n)}}{\sigma_d}\right)^2} * e^{-\frac{1}{2}\left(\frac{id_{(q,n)}}{\sigma_r}\right)^2} \tag{4.13}$$

where $gd_{(q,n)}$ is the geometric distance between the query point and the neighbor point and $id_{(q,n)}$ is the intensity distance between the two points. σ_d and σ_r are the two scale values which can be set by the user to adjust the filter so that it produces the best outcome.

Bilateral Filter (each channel separately)

This version of the bilateral filter differs from the previously described filter version by calculating the filter weight for each color channel separately. That means, instead of calculating the distance between the *rgb* values of the points, the distance is calculated for the red channel, the blue channel and the green channel separately and therefore results in three different weights.

Trilateral Filter

For the trilateral filter the modified version of the bilateral filter from Garnett et al. [19] is used. This filter is explained in section 2.4.4. The filter was added because it should remove impulse noise as well as gaussian noise. Before the filter can be used, the *ROAD* values for the k neighbor points of the query point have to be calculated, which leads to an additional pass. Because of the additional pass for the *ROAD* values, the performance of the color adjustment calculation decreases when using this filter compared to the other filters.

First the ROAD values have to calculated:

$$ROAD_m(q) = \sum_{i=1}^{m} id_{(q,n_i)} \tag{4.14}$$

where q is the point for which the ROAD value is calculated. $id_{(q,n)}$ is the intensity distance between the query point end the i-th neighbor point.

With the ROAD values for each point, the weight can be calculated by :

$$fw(n_i) = e^{-\frac{1}{2}\left(\frac{gd_{(q,n)}}{\sigma_d}\right)^2} * \left(e^{-\frac{1}{2}\left(\frac{id_{(q,n)}}{\sigma_r}\right)^2}\right)^{1-J(q,n)} * e^{-\frac{ROAD(n)^2}{2*\sigma_J{}^2}J(q,n)} \tag{4.15}$$

where $gd_{(q,n)}$ is the geometric distance between the query point and the neighbor point and $id_{(q,n)}$ is the intensity distance between the two points. σ_d and σ_r are the two scale values which can be set by the user to adjust the filter so that it produces the best outcome.

And the „joint impulsivity“ is calculated by:

$$J(q,n) = 1 - e^{-\frac{1}{2}\left(\frac{\frac{ROAD(q)+ROAD(n)}{2}}{\sigma_J}\right)^2} \tag{4.16}$$

where $ROAD(q)$ and $ROAD(b)$ are the ROAD values for the query point and the neighbor point.

Trilateral Filter (each channel separately)

Like in the second version of the bilateral filter, for this version of the trilateral filter the filter weight is created for each color channel separately. The ROAD values are calculated for each channel separately too.

4.3 Kd-Tree

A k-dimensional tree (short kd-tree) was introduced by [6] and is a data structure used to store points from $\mathbb{R}^n$. The kd-tree is a binary search tree, that means that every node of the tree has hat most two child nodes. This makes this data structure very useful for queries like the nearest-neighbor-search. For operations like fast insertion of points and fast deletion of points the kd-tree is not qualified. As a method to find the k-nearest neighbors for every point in a point

cloud is needed for the point-based color adjustment, this is done with the kd-tree.

A node of the kd-tree can either be an inner node or a leaf node. Inner nodes are nodes which are further subdivided into sub nodes. These inner nodes can be split in one of the k dimensions into two parts. Then the inner node points to its two sub nodes. Which axis is chosen for splitting the node varies, it can for example be a randomly chosen axis, or the axis with the widest spread. Leaf nodes are the nodes which actually store point data. Contrary to [25] the implementation of the kd-tree in this thesis stores point data only in the leaf nodes. The inner nodes just contain pointers to their child nodes.

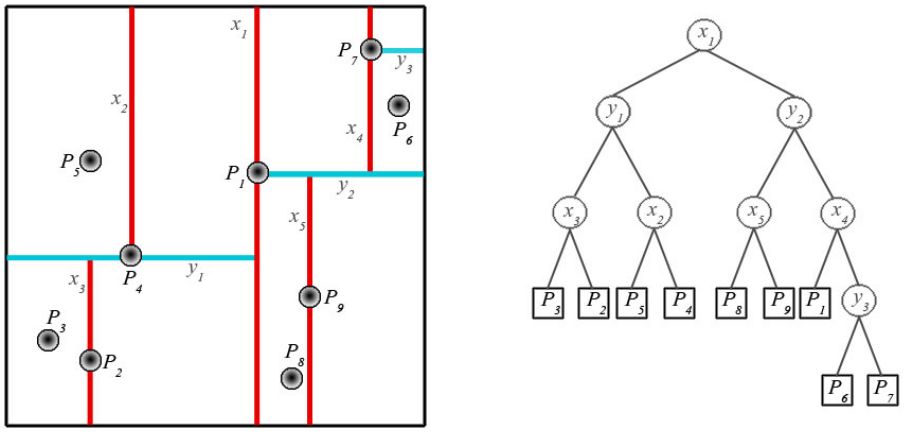


Figure 4.6: 2-d tree

In Figure 4.6 a 2-d tree is given as an example consisting of nine points. The left side shows how the axes are split. First the cloud is split along the x-axis, x_1. Then the left and the right side are split along the y-axis, y_1 and y_2. After that it is again split along the x-axis, and this recursion repeats until all points are divided into leaf nodes. Each leaf node stores at most a user-defined amount of points.

4.3.1 Out-of-Core Implementation Ability

The implementation of the kd-tree for this thesis is prepared for out-of-core processing of point clouds. Therefore the kd-tree only keeps the coarse structure of the tree in the main memory. The point data from the leaf nodes of the tree are stored in files on the disk. If the node data is needed for the nearest neighbor search, rendering or color processing, the detailed information about the points in the leaf node can be loaded into main memory.

Section 2.2 gives an overview of currently used data structures for large point sets which are used for spatial queries. Hubot et al. [23] use a fast version of the kd-tree where the point information is compressed so that the whole kd-tree fits into the main memory. If the kd-tree is

just used for the color adjustment, the total point information is not needed in the main memory the whole time. When searching for the nearest neighbors for example, only the node for whose points the nearest neighbors are searched can be loaded in the main memory. If any other node may contain potential neighbor points this node is then loaded into main memory too. All other points are not needed at this step.

The size of the point clouds which were used for this thesis was limited to fit in the main memory. Therefore the out-of-core ability was not used and the complete tree was kept in the main memory.

4.3.2 Structure

The kd-tree class holds the root node of the tree data structure. Every kd-tree node holds pointers to its two sub nodes, except for the leaf nodes which hold the actual point data instead. Figure 4.7 shows the structure of this kd-tree, including the information stored with each point, like the normal, the scan position from which the point was scanned, the point position and its color. To traverse the tree not only downward but also upward each node also holds a pointer to its parent node.

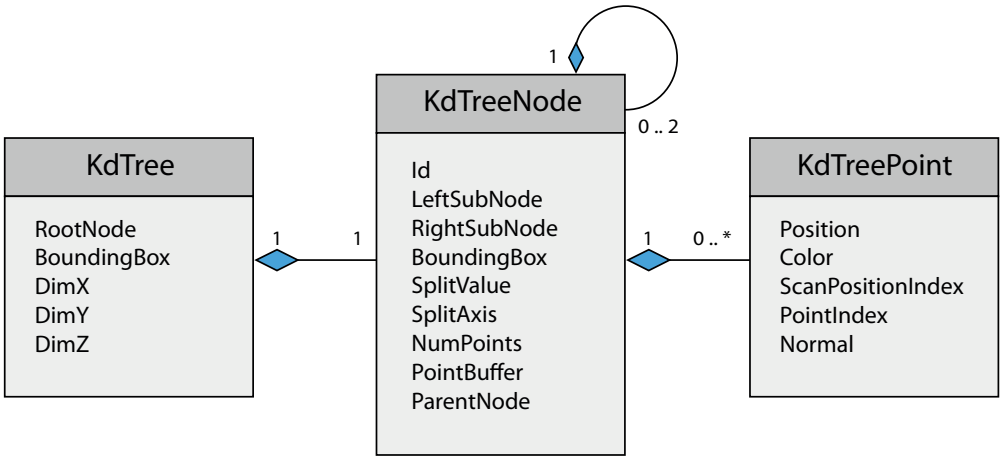


Figure 4.7: Structure of kd-tree

4.3.3 Construction

For the construction of the kd-tree the user can define the maximum number of points one leaf node should contain. As mentioned before in this version of the kd-tree the available axes are cycled to get the split axes. When creating the kd-tree the point files are loaded and the median value of the positions of the points in the first split axis is found and the data is divided into two parts. Then the resulting two parts of the data are recursively split until the resulting amount of points is equal to or smaller than the chosen bucket size. Then the algorithm stops and the leaf nodes are stored on the disk with a link to their parent nodes. The basic algorithm for subdividing one kd-tree node into two sub nodes is outlined in Algorithm 13.

```
1  Algorithm: SubdivideNode
   input: PointBuffer
2  if NumPoints > MaxPointsPerNode then
3  |   choose splitting axis
4  |   sort PointBuffer according at the splitting axis
5  |   search median value
6  |   split PointBuffer at the median value in two buffers
7  |   create left and right subnode
8  |   calculate bounding box for left and right subnode
9  |   leftNode← SubdivideNode( LeftPointBuffer)
10 |   rightNode← SubdivideNode( RightPointBuffer)
11 else
12 |   save point buffer in node
13 end
```

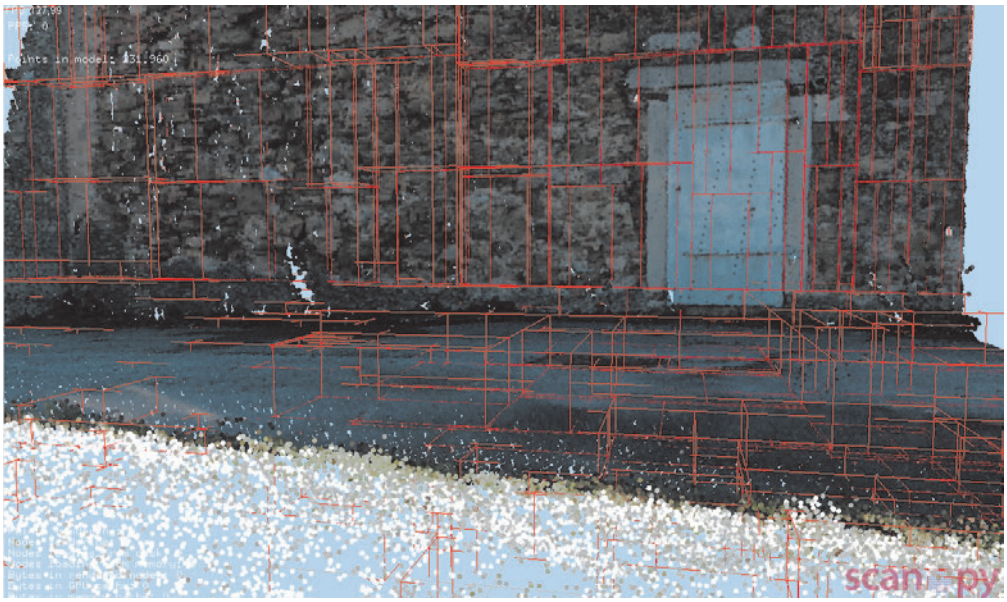
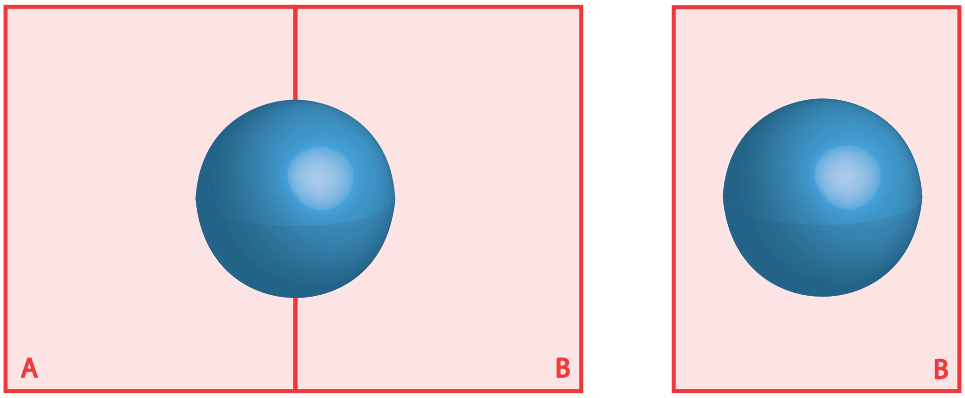


Figure 4.8: Node visualization of a kd-tree in scanopy.



(a) „bounds-overlap-ball“ (b) „ball-within-bounds“

Figure 4.9: (a) shows the „bounds-overlap-ball“ and (b) the „ball-within-bounds“ test. Here the blue sphere is the sphere created around the query point and the red boxes are the bounding boxes of the kd-tree nodes.

4.3.4 k-Nearest-Neighbor Search

For the calculation of the *k*-nearest neighbors the approach from Friedman et al. [17], which was shortly described in section 2.5, was used. A bounding sphere is created for the query point, the radius for the bounding sphere can be chosen by the user. Then the kd-tree is traversed starting from the root node. The traversal stops at each node which fails the „bounds-overlap-ball“ or „ball-within-bounds“ test. If the bounding sphere of the query point and the node bounding box overlap, the „bounds-overlap-ball“ test will pass. The „ball-within-bounds“ passes if the bounding sphere of the query point is within the bounding box of the node. Figure 4.9 visualizes these two tests. If the nodes pass these tests their subnodes will be tested until the algorithm finds the leaf nodes which contain the real neighbor for the query point.

A queue which stores pointers to the nodes that pass the „bounds-overlap-ball“ and „ball-within-bounds“ test is created. When all these leaf nodes containing the potential neighbor points are found, the queue holding these leaf nodes is processed. Therefore a priority queue of size *k* is created. The node closest to the query point is tested first for potential neighbors. The distance between the points in this node and the query point is calculated and the *k* points with the smallest distance to the query point are stored in the queue. Then the distance to the other nodes in the node queue is calculated. Those nodes in the node queue, whose distance to the query point is smaller then the distance of the point with the maximum distance in the point priority queue, are searched for neighbors too.

Out-of-core ability of *kNN* search

This nearest neighbor search is usable for very large point clouds, since for the „bounds-overlap-ball“ or „ball-within-bounds“ test the actual node need not be loaded into the main memory, because the tree structure of the kd-tree always remains in the main memory. If those nodes pass

the test then the individual nodes, holding the potential neighbor nodes, must be loaded in the main memory, one after another.

4.4 Application of the point-based color adjustment

To use the color adjustment method presented in the first part of this Chapter the point cloud has to stored in a kd-tree structure. Then the k-nearest points can be searched for each point in the point cloud. With the nearest neighbors and a chosen filter, the new color of the query point can be calculated. The settings for the nearest neighbor search, like the maximum number of neighbors or the maximum distance, can be adjusted in a *GUI*[1] Dialog integrated into *Scanopy*. The neighbor point selection criteria as well as the settings for the bilateral or trilateral filter can be set too. The user can also choose the filter which should be used for the point-based color adjustment. Results of the point-based approach are shown and compared in Chapter 5

[1] Graphical User Interface

CHAPTER 5

Results and Conclusions

This chapter shows results achieved with the color adjustment methods from Chapter 3 and Chapter 4. The different settings for the point-based approach are compared on different models to identify their strengths and weaknesses. The influence of the selection criteria for the neighbor points on the final result is presented too.

Section 5.1 gives a short overview how these results were generated. Section 5.2 shows results produced with the image-based approach as well as with the point-based approach and then the combination of the two methods is presented. The image-based approach was used for the point model of the Leopoldsberg in Section 5.3. In Section 5.4 the point-based color adjustment approach is used for the Hanghaus of Ephesos model.

5.1 Color adjustment testing pipeline

For testing, point clouds made with scanners from the company *Riegl*[1] are used. This point cloud data is registered and colored from the images taken during the scan process, with their program *RiSCAN PRO* [44]. These images are available in the *JPG* format. To test the color transfer method from Chapter 3, these textures are changed and then the point cloud is newly colorized from these processed images. For the color transfer, the method „Color Transfer in Correlated Color Space“ from Xiao et al. [50] (described in Section 3.3) is used.

After that, the point cloud is exported and converted to a kd-tree that can be used in *Scanopy* to test the point-based approach. As already mentioned in Chapter 4, a GUI[2] dialog was therefore included in Scanopy to allow the user to define the color adjustment settings which should be used.

All these point clouds consist of different scans coming from more then one scan position, so the visual artifacts coming from the merge of these scans can be well observed. For the

[1] http://www.riegl.com/

[2] Graphical User Interface

Table 5.1: Time used for building the kd-tree and the color adjustment

	1.8M Points	5M Points
Build up kd-tree	30 seconds	1.5 minutes
Color adjustment (gauss, with OpenMP)	7.5 minutes	32 minutes
Color adjustment (gauss)	15 minutes	75 minutes

comparison of the different color adjustment settings a close-up view of one part of the model, where the color discontinuities are quite visible, is used. The difference of the settings are quite obvious here, especially on small details of the model, like a door or window frame.

The time it takes to build up a kd-tree and use the point-based color adjustment is shown in Table 5.1. Here the calculation times for a point model consisting of 1.8M points and one with 5M points is shown. For the color adjustment, the same settings were used, the Gauss filter with 5 neighbor points. The speed-up of the color adjustment process because of the use of the *OpenMP API* [3] for multi-threading is shown too. These results are produced on a Core i5-760 with 2.8 GHz and 8 GB RAM.

5.2 Stephansdom

5.2.1 Image-based color adjustment

The part of the Stephansdom containing the altar was used for testing the color adjustment methods. This part consists of twelve different scan positions. All 78 textures for these scan positions where processed two times. One time with the generated source image (see Section **??**) and then with one image chosen out of the 78 available images used as source image. Figure 5.1 shows the two source images used for the color transfer. Figure 5.2 shows the different results compared with the original coloring. The images used for the coloring of the Stephandsdom were made with different cameras and different white balance settings for each scan position.

In comparison to the original point cloud (Figure 5.2(a) and 5.2(b)) the point cloud with the colors from the color transfer with the generated source image (Figure 5.2(c) and 5.2(b)) has a more uniform appearance but the whole color representation is desaturated. The usage of a chosen source image out of all images used for the color transfer results in a point cloud with a more uniform look than the original coloring of the point cloud. The point cloud still looks a bit desaturated but the colors of this point cloud are more uniform because the very dark parts from the original coloring are brightened up.

The areas in the images marked with the red circles show large black areas in the original point cloud. The same area is not black anymore in the processed point clouds, but still these areas are not colored with the right color.

5.2.2 Point-based color adjustment with different filters

The original coloring of the next scene is shown in Figure 5.3. The outcome of the different filters are presented in Figure 5.4 and in Figure 5.5. The settings which were used for the color

(a) (b)

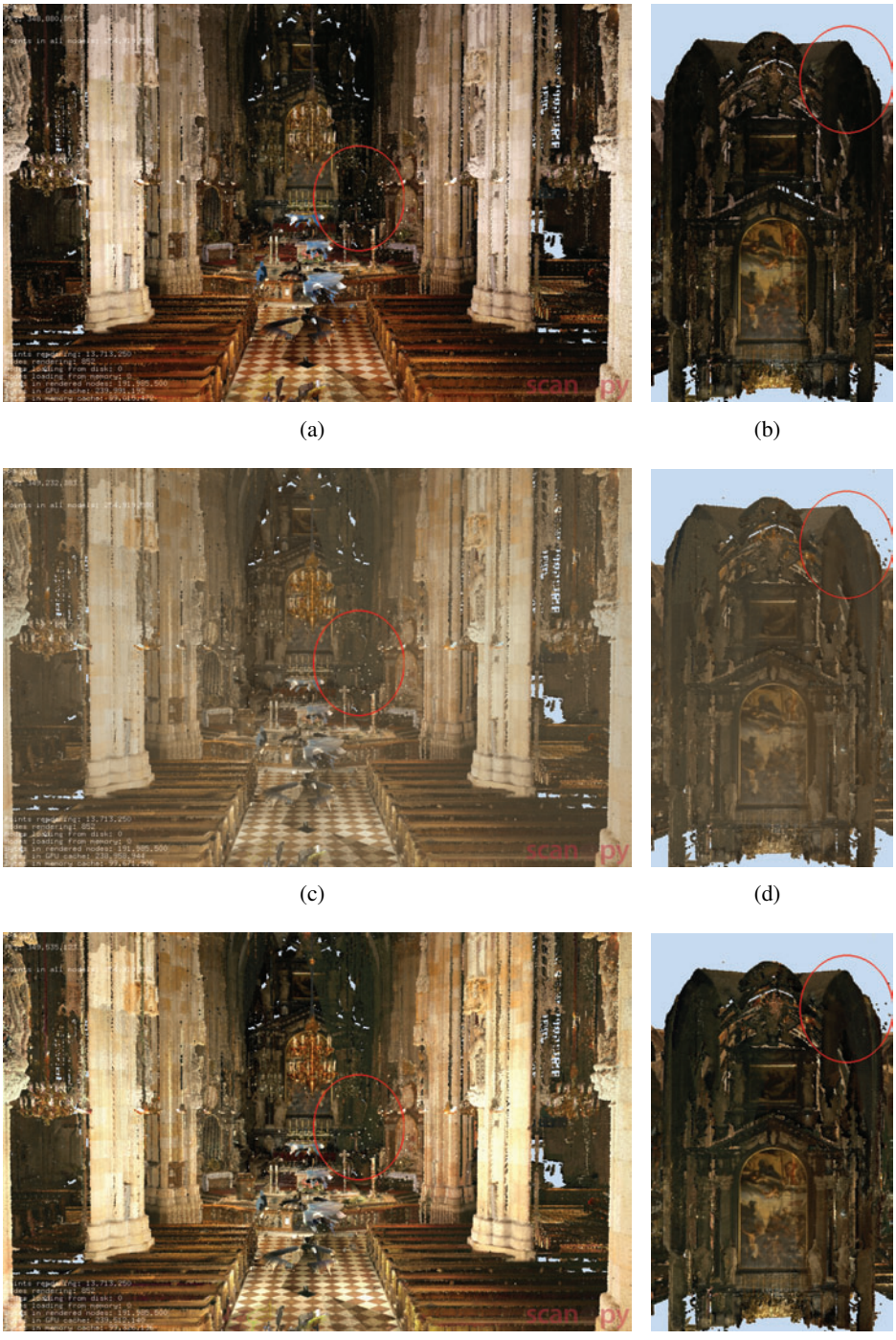
Figure 5.1: These images were used as source images for the color transfer. Figure (a) is the automatically generated source image of all 78 images. Figure (b) is a user-selected source image.

Table 5.2: Settings for the calculation of Figures 5.4 and 5.5

Subfigure	Filter	kNN	Max Distance	Scales (σ_d, σ_r, σ_I)	Scan Pos Influence	Normal Relation	Normal-Scan Pos Relation
Fig. 5.4(a)	Gauss	5	0.2		x	x	x
Fig. 5.4(c)	Bilateral	5	0.1	10, 100	x	x	x
Fig. 5.4(e)	Trilateral	5	0.1	15, 50, 50	x	x	x
Fig. 5.5(a)	Mean	5	0.1		x	x	x
Fig. 5.5(c)	Median	10	0.1		x	x	x
Fig. 5.5(e)	Median	5	0.1		x	x	x

adjustment for each image are listed in Table 5.2. The abbreviation „Scan Pos“ stands for scan position. If a checkmark in the last three columns is set these neighbor point selection criteria were used.

Here the best results are achieved with the median, bilateral and the trilateral filter. The median filter is used two times with a different number of kNN. The first time with ten neighbor points (Figure 5.5(c)), the second time with five neighbor points (Figure 5.5(e)). When the result images from the median filter are compared it is well visible that the usage of too many neighbor points reduces details. The best result was achieved with five kNN Points.



(a) (b)

(c) (d)

(e) (f)

Figure 5.2: Results of using the image-based approach. The first row shows ((a), (b)) the model with its original colors. For the second row ((c), (d)) the automatically generated image (Figure 5.1(a)) was used as source image. And for the third row ((e), (f)) Figure 5.1(b) was used as source image.

(a) Original (b) Original closeup

Figure 5.3: Original coloring of the scene for the point-based color adjustment.

Table 5.3: Settings for the calculation of Figures 5.6 and 5.7

Subfigure	Filter	kNN	Max Distance	Scales (σ_d, σ_r, σ_I)	Scan Pos Influence	Normal Relation	Normal-Scan Pos Relation
Fig. 5.6(b)	Trilateral	5	0.1	15, 30, 30	x	x	x
Fig. 5.6(c)	Trilateral	5	0.1	15, 30, 30		x	x
Fig. 5.7(a)	Trilateral	5	0.1	15, 30, 30			x
Fig. 5.7(b)	Trilateral	5	0.1	15, 30, 30		x	
Fig. 5.7(c)	Trilateral	5	0.1	15, 30, 30	x	x	x

5.2.3 Point-based color adjustment with different neighbor point selection criteria

In Figure 5.6 and 5.7 the influence of the different selection criteria for choosing neighbor points can be observed. The color metric (see Section 4.2.3) is used as one color adjustment setting. The filter setting for the trilateral filter was the same for all screenshots. The parameter „CM“ in Table 5.3 stands for the color metric.

The influence of preferring the points from the nearest scan positions can be well observed in Figure 5.6(b). A clean cut is made at the border between the two nearest scan positions. In comparison to Figure 5.6 and 5.7 where the consideration of the nearest scan position helps to remove the black areas on the model, here the results without the usage of the nearest scan position look better. Therefore it can be said that it depends strongly on the model or part of the model if the usage of the nearest scan position improves the result achieved by the color adjustment.

For the color adjustment of the point cloud in Figure 5.6(b) 68 % of potential neighbor points

(a) Gauss

(b) Gauss closeup

(c) Bilateral

(d) Bilateral closeup

(e) Trilateral

(f) Trilateral closeup

Figure 5.4: One set of results of using the point-based approach. These are produced with the settings in Table 5.2

(a) Mean

(b) Mean closeup

(c) Median 1

(d) Median 1

(e) Median 2

(f) Median 2

Figure 5.5: The second set of results of using the point-based approach. These are produced with the settings in Table 5.2

(a) Original

(b) Setting 1

(c) Setting 2

Figure 5.6: Results produced with the different selection criteria for choosing neighbor points enabled. These are produced with the settings in Table 5.3

(a) Setting 3

(b) Setting 4

(c) Setting 5

Figure 5.7: Results produced with the different selection criteria for choosing neighbor points enabled. These are produced with the settings in Table 5.3

have been rejected because of their scan position. 0.01 % of the potential neighbor points have been rejected because their normal pointed in a completely different direction than the normal of the query point. And finally 0.0001 % have been rejected because the normal of these neighbor points looked in a completely different direction than the vector between the neighbor point and the scan position. For the result in Figure 5.4(a) 39% of the potential neighbor poins are rejected because of their scan position, 0.04 % because of their normal and 0.0001 % because of the normal - scanposition relationship.

For the generation of the point cloud in Figure 5.6(c) all criteria based on the normal were used. Here changes occur on geometric edges like the steps or the altar. The usage of the normal based criteria helps to emphasize the borders and make less blurry edges.

The usage of the color metric results in a point cloud which looks quite noisy. Therefore the usage of the color metric does not help to improve the result images.

5.2.4 Point-based color adjustment with different settings for the bilateral and trilateral filter

The results of using the point-based color adjustment methods on the scene from Figure 5.8 are shown in Figure 5.9 and Figure 5.10. Here different settings for the bilateral and trilateral filter were brought into focus. These settings can be found in Table 5.4.

Figure 5.8: Original coloring of the second scene from the stephansdom.

On these images it can be observed that the bilateral filter helps to create a smoother surface but noise is sometimes detected as edges. Instead of removing these pixels they are even pointed out. The trilateral filter classifies the noisy points much more accurately and therefore good results can be achieved. Figure 5.9(a), 5.9(b), 5.10(a) and 5.10(b) show how important the settings are for the bilateral and trilateral filter. It is possible to achieve completely varying results with the same filter but different settings. The usage of the bilateral and trilateral filter

(a) Bilateral 1

(b) Bilateral 2

(c) Bilateral separate

Figure 5.9: Results produced with different settings of the bilateral filter. These are produced with the settings in Table 5.4

(a) Trilateral 1

(b) Trilateral 2

(c) Trilateral separate

Figure 5.10: Results produced with different settings of the trilateral filter. These are produced with the settings in Table 5.4

Table 5.4: Settings for the calculation of Figures 5.9 and 5.10

Subfigure	Filter	kNN	Max Distance	Scales (σ_d, σ_r, σ_I)	Scan Pos Influence	Normal Relation	Normal-Scan Pos Relation
Fig. 5.9(a)	Bilateral	5	0.1	1, 50		x	x
Fig. 5.9(b)	Bilateral	5	0.1	10, 30		x	x
Fig. 5.9(c)	Bilateral seperate	5	0.1	10, 30		x	x
Fig. 5.10(a)	Trilateral	5	0.1	5, 10, 30		x	x
Fig. 5.10(b)	Trilateral	5	0.1	15, 30, 30		x	x
Fig. 5.10(c)	Trilateral seperate	10	0.1	15, 30, 30		x	x

for each *RGB* color channel separately results in some completely wrong colored pixels and the result looks a little bit more blurry.

5.2.5 Combination of the point-based approach and the image based approach

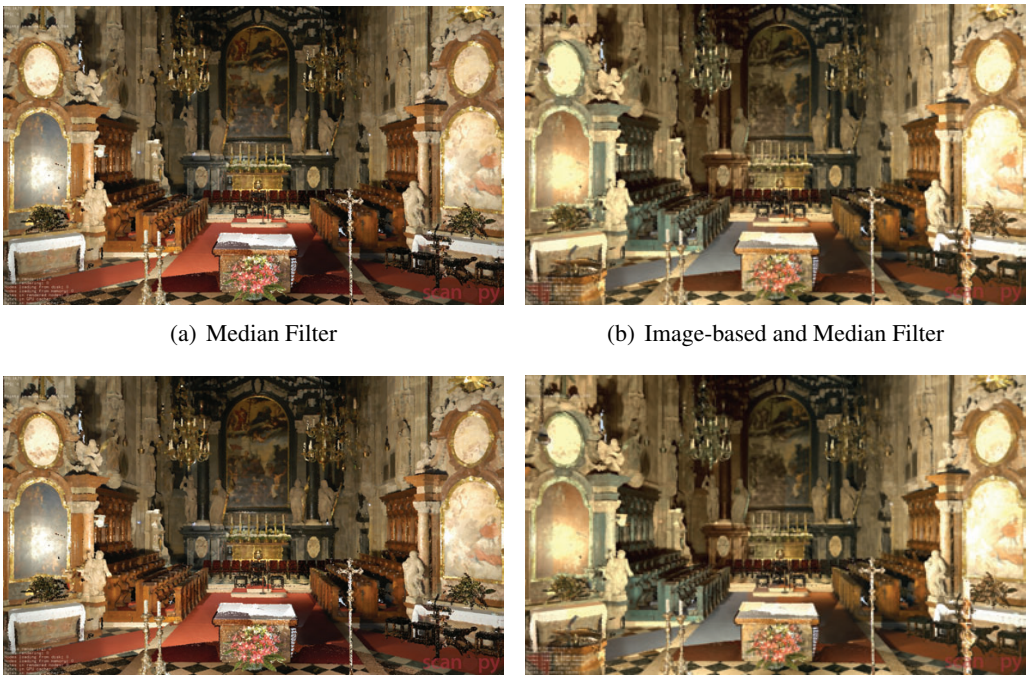
Figure 5.11 shows two result images from the combination of the point-based approach and the image-based approach. The point-based approach is used on the point cloud which was colored from the processed images. This point cloud is shown in Figure 5.2(e) and 5.2(f). For the point-based approach the same settings as for Figure 5.5(e) and 5.4(e) were used.

The images where only the point-based approach was used show better results than those where a combination of the two approaches was used. On Figure 5.11(b) and 5.11(d) the benches on the left side have a different color than on the right side. The carpet on the floor has a wrong color too on the images were a combination of the two approaches is used. On these images it can be observed that the image-based approach is not necessary to remove color discontinuities because these can be quite good removed when only the point-based approach is used.

5.3 Leopoldsberg

For the point model of the Leopoldsberg the image-based approach was tested too. The images taken for this point cloud have great variations in it. Examples for such images are shown in Figure 5.12. Figure 5.13 shows the original colored point cloud. The results produced with the image-based approach are presented in Figure 5.14. All images made during the scan process were made with the same camera and different exposure times but for all images the same manual white balance was set for the camera.

Using the image-based approach for this model unfortunately did not improve the final coloring of the model. The images for the point cloud had such big variations in it that it was not possible to transfer them all in the same color space without getting wrongly colored images. The color transfer works well when only a few images are used or when the images which were used do not differ that much. Much better results were achieved for the pictures used in Sec-



(a) Median Filter (b) Image-based and Median Filter

(c) Trilateral Filter (d) Image-based and Trilateral Filter

Figure 5.11: Results from a combination of the point-based and image-based method (b, d) compared to the results where only the point-based approach was used (a, c).

tion 4, but these pictures had not such differences in lighting. For the generation of the images in Figure 5.14 twenty images were used and that results in quite wrong images. So unfortunately it was not possible with this approach to improve the visual quality of this point cloud.

5.4 Hanghaus of Ephesos

5.4.1 Point-based color adjustment with different filter

In Figure 5.16 and Figure 5.17 some results for the Hanghaus of Ephesos are shown. Here just the point-based approach was tested. The settings for the different modes tested are shown in Table 5.5. Figure 5.15 is a screenshot of the original colored Hanghaus.

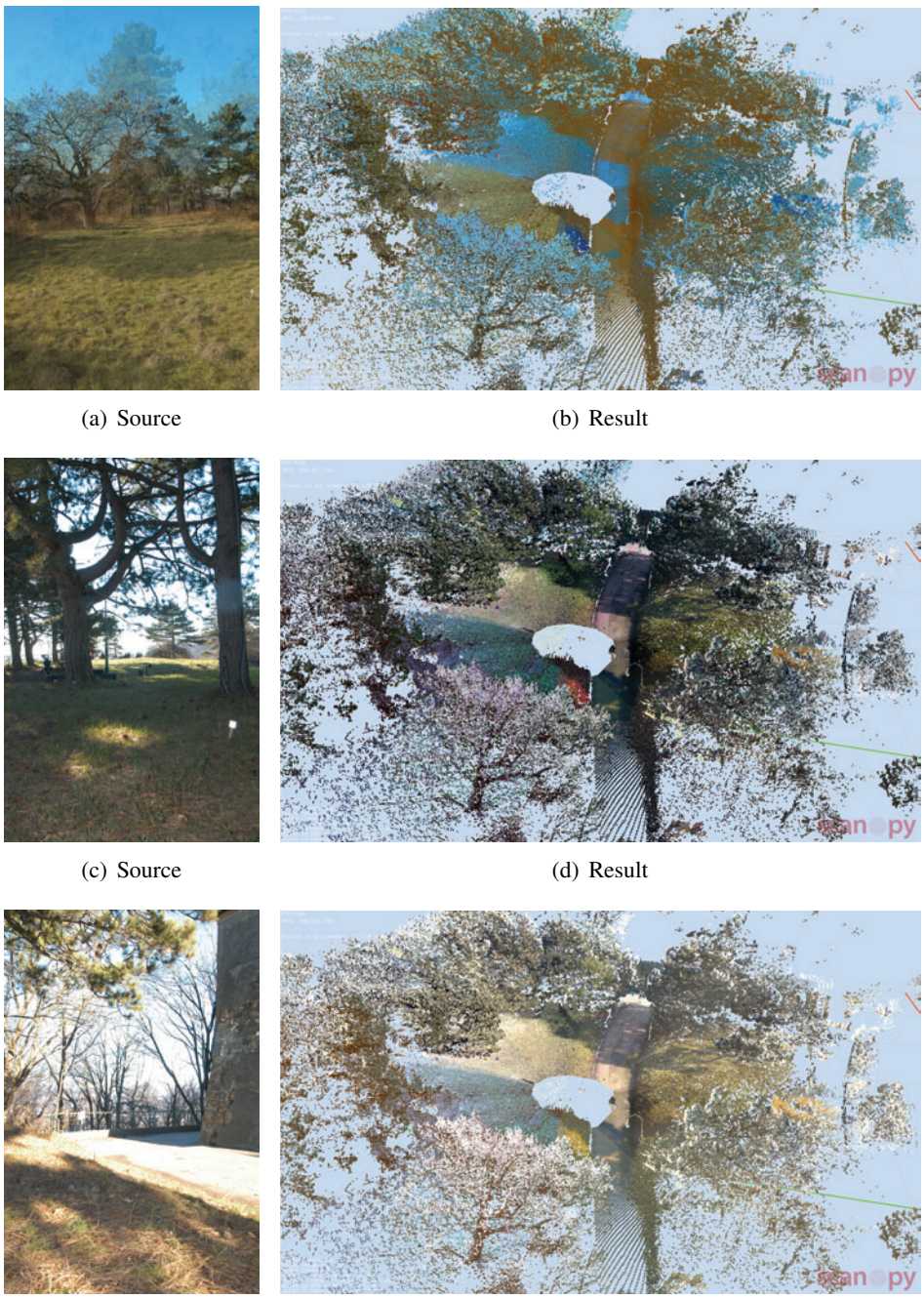
5.5 Conclusion

In this chapter results from the use of the point-based approach and the image-based approach on three different point clouds was presented.

(a) (b)

Figure 5.12: Images taken of the scanned object during the scan process.

Figure 5.13: Original colored point cloud.



(a) Source (b) Result

(c) Source (d) Result

(e) Source (f) Result

Figure 5.14: Results of using the image-based approach. The first row shows the source images used for the recoloring. The second column shows a resulting point cloud.

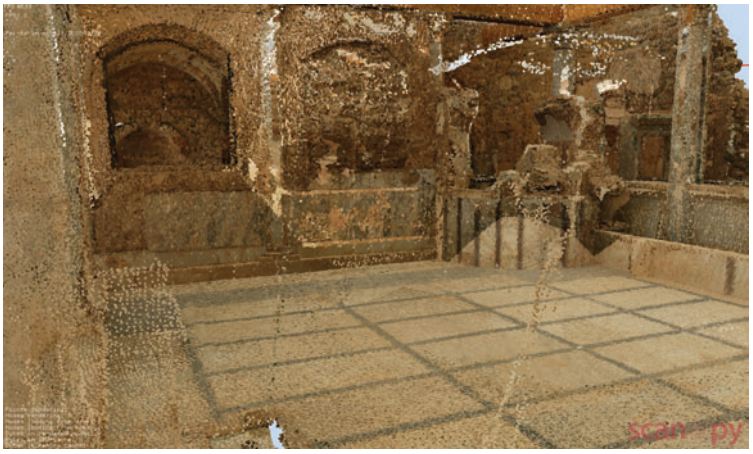


Figure 5.15: Original coloring of the hanghaus scene.

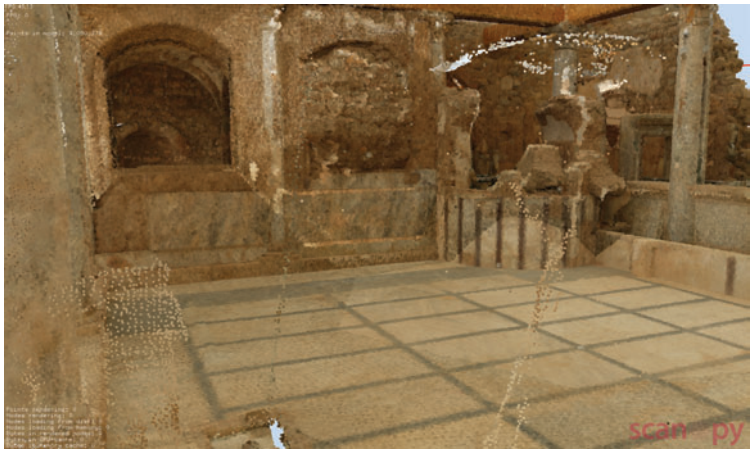
Table 5.5: Settings for the calculation of Figures 5.16 and 5.17

Subfigure	Filter	kNN	Max Distance	Scales (σ_d, σ_r, σ_I)	Scan Pos Influence	Normal Relation	Normal-Scan Pos Relation
Fig. 5.16(a)	Gauss	5	0.2		x	x	x
Fig. 5.16(b)	Mean	5	0.2		x	x	x
Fig. 5.16(c)	Bilateral	5	0.3	3, 30	x	x	x
Fig. 5.17(a)	Median	5	0.1		x	x	x
Fig. 5.17(b)	Trilateral	10	0.3	15, 30, 30	x	x	x

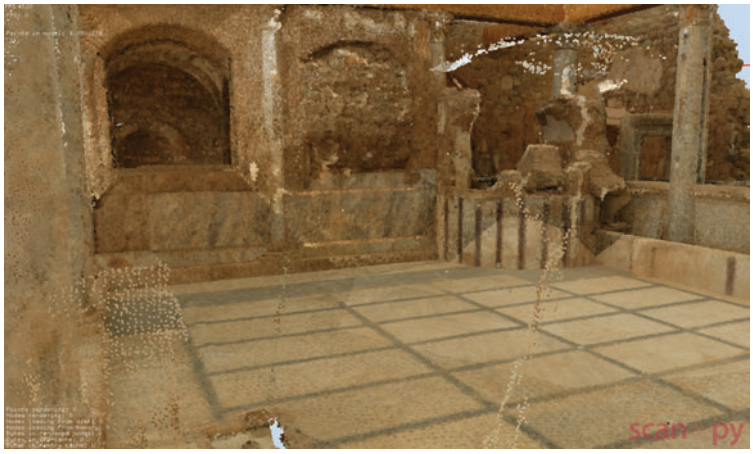
Quite good results were achieved with the point-based approach, especially with the use of the neighbor point selection criteria and the median and trilateral filter. The filter work how they are supposed to do. With those two filters the Gaussian and impulse noise was quite good removed and the resulting point clouds have a smooth and uniform color appearance. Unfortunately there are still some noisy spots which could not be removed. For example the flowers in the front of the point cloud shown in Figure 5.4(e). These flowers still contain black noise. But here the colors of the points vary too much so that the filter cannot detect these dark points as wrongly colored points.

The use of the neighbor point selection criteria help to improve the final results too, especially the criteria where points from the nearest scan position are preferred as neighbor points. The dark areas appearing in Figure 5.3(a) can be mostly removed by using this criterion.

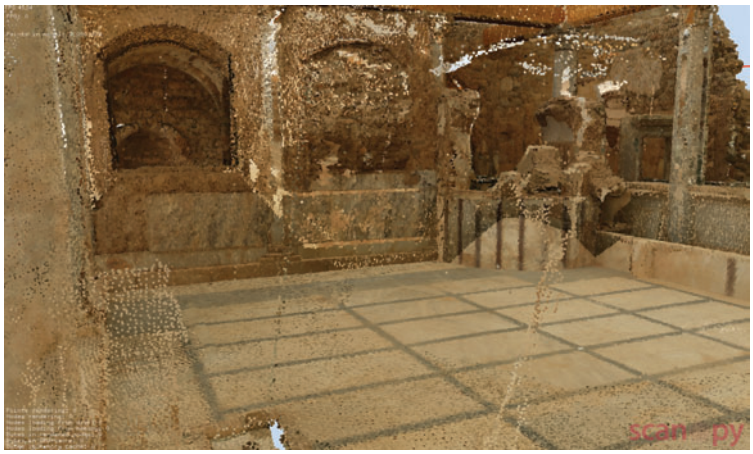
In comparison to the point-based approach the results achieved with the image-based approach did not improve the color appearance. Mainly this is because the number of images used is too big and the lighting mood on the images which are used to color the point cloud varies too much. The approach worked quite ok with a small amount of images, if the color of one



(a) Gauss

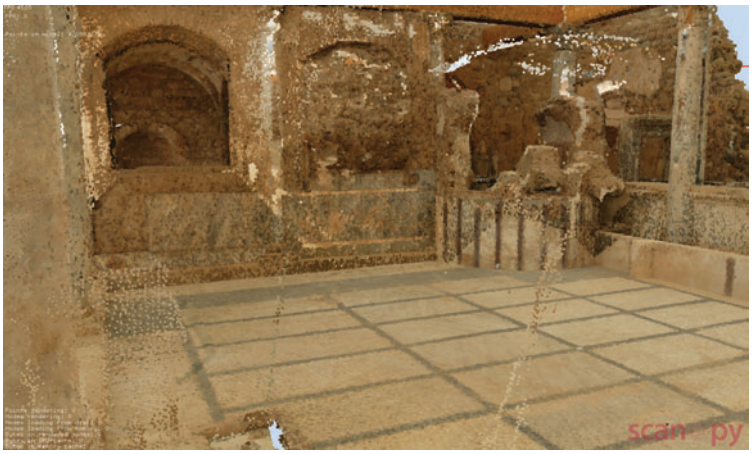


(b) Mean

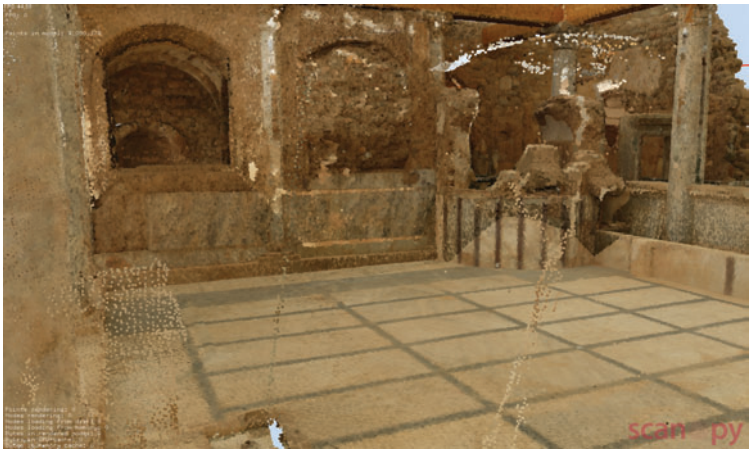


(c) Bilateral

Figure 5.16: One set of results of using the point-based approach on the Hanghaus model. These are produced with the settings in Table 5.5



(a) Median



(b) Trilateral

Figure 5.17: These are more results of using the point-based approach on the Hanghaus model. These are produced with the settings in Table 5.5

image is transfered to the one other image, but when more images should have the same color characteristics the number of wrongly colored images tops the number of good colored images.

To achieve the best result only the point-based approach should be used for the color adjustment of the already colored point clouds. The preferred filter should be the trilateral filter because this filter removes noise but still preserves the details in the color of the point cloud. As an alternative the median filter produces quite good results too but here more detail in the color of the point cloud gets lost. The settings for the trilateral filter are different for every point cloud so it may take more than one attempt to produce the desired result.

CHAPTER 6

Summary

This thesis presents an image based approach and a point based approach to improve the visual appearance of point clouds. The combination of single scans leads to various color discontinuities in the overall model which should be decreased or even removed automatically.

The image based approach tries to improve the coloring of the point cloud by transforming all images taken for the colorization of the point cloud to the same lighting mood. This works well if the lighting situation in the images does not differ too much or if this approach is used for a small amount of images. Otherwise the new images produced by transfering the images in the color space of another image can have wrong colors and cannot be used at all.

The second approach is a point based approach. Here an already colored point cloud is processed. The single points are recolored according to the surrounding points and a specific filter. A data structure for the k-nearest neighbor search was implemented and different criteria for the usage of the neighbor points where presented. For the final recoloring of the points, a set of different filters is available. These filters should help to smooth or even remove the noise of the coloring of the point cloud, but the edges and details in the texturing should still be preserved.

6.1 Future Work

6.1.1 Image based approach

The approach presented in Section 3 generates result images which are quite good when a small amount of images is used for the color transfer, or when the lighting mood of the images does not differ very much. This approach may be improved if a color transfer method is found which also works for images with big variations in the lighting.

6.1.2 Point based approach

The results which can be achieved with the point based approach help to increase the visual quality of the point cloud. One drawback of this method is the time it takes to compute the kNN for a big point cloud. This can possibly be improved.

Bibliography

[1] Ann: A library for approximate nearest neighbor searching. Accessed Nov. 3, 2011.

[2] Colour metric. Accessed Nov. 2, 2011.

[3] Openmp. Accessed Nov. 11, 2011.

[4] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45:891–923, November 1998.

[5] Nobuyuki Bannai, Alexander Agathos, and Robert B. Fisher. Fusing multiple color images for texturing models. In *In Proceedings of 2nd International Symposium on 3D Data Processing, Visualization and Transmission*, pages 558–565. IEEE Computer Society, 2004.

[6] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *CACM*, 18(9):509–517, 1975.

[7] Robert Bergevin, Marc Soucy, Hervé Gagnon, and Denis Laurendeau. Towards a general multi-view registration technique. *IEEE Trans. Pattern Anal. Mach. Intell.*, 18:540–547, 1996.

[8] Fausto Bernardini, Ioana M. Martin, and Holly Rushmeier. High-quality texture reconstruction from multiple scans. *IEEE Transactions on Visualization and Computer Graphics*, 7(4):318–332, October 2001.

[9] Fausto Bernardini and Holly Rushmeier. The 3d model acquisition pipeline. *COMPUTER GRAPHICS forum*, 2:149–172, 2002.

[10] Paul J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(2):239–256, February 1992.

[11] Marco Callieri, Paolo Cignoni, and Roberto Scopigno. Reconstructing textured meshes from multiple range rgb maps. In *7th Int.l Fall Workshop on Vision, Modeling, and Visualization 2002*, pages 419–426. IOS Press, 2002.

[12] Michael Connor and Piyush Kumar. Fast construction of k-nearest neighbor graphs for point clouds. *Visualization and Computer Graphics, IEEE Transactions on*, 16:599–608, 2009.

[13] Brian Curless. From range scans to 3d models. *Computer Graphics*, 33:38 – 41, 1999.

[14] Carsten Dachsbacher, Christian Vogelgsang, and Marc Stamminger. Sequential point trees. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 657–662. ACM Press, 2003.

[15] Matteo Dellepiane, Marco Callieri, Massimiliano Corsini, Paolo Cignoni, and Roberto Scopigno. Improved color acquisition and mapping on 3d models via flash-based photography. *ACM Journal on Computing and Cultural Heritage*, 2:9:1–9:20, March 2010.

[16] Elmar Eisemann and Frédo Durand. Flash photography enhancement via intrinsic relighting. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 673–678. ACM, 2004.

[17] Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3:209–226, September 1977.

[18] K. Fukunage and P. M. Narendra. A branch and bound algorithm for computing k-nearest neighbors. *IEEE Trans. Comput.*, 24:750–753, 1975.

[19] Roman Garnett, Timothy Huegerich, Charles Chui, and Wenjie He. A universal noise removal algorithm with an impulse detector. *IEEE Transactions on Image Processing*, 14:1747 – 1754, 2005.

[20] RIEGL Laser Measurement Systems GmbH, October 2011. Accessed Oct. 2, 2011.

[21] Enrico Gobbetti and Fabio Marton. Layered point clouds: a simple and efficient multiresolution structure for distributing and rendering gigantic point-sampled models. *Computers Graphics*, 28:815–826, 2004.

[22] Guy Godin, Marc Rioux, and Rejean Baribeau. Three-dimensional registration using range and intensity information. In *Proc. SPIE 2350*, volume 2350, pages 279–290, 1994.

[23] E. Hubo, T. Mertens, T. Haber, and P. Bekaert. The quantized kd-tree: Efficient ray tracing of compressed point clouds. *Symposium on Interactive Ray Tracing*, 0:105–113, 2006.

[24] Jun-Yan Huo, Yi-Lin Chang, Jing Wang, and Xiao-Xia Wei. Robust automatic white balance algorithm using gray color points in images. *IEEE Transactions on Consumer Electronics*, 2:541–546, 2006.

[25] Henrik Wann Jensen. *Realistic Image Synthesis Using Photon Mapping*. AK Peters, 2001.

[26] Andrew Edie Johnson and Sing Bing Kang. Registration and integration of textured 3d data. In *Proceedings of the International Conference on Recent Advances in 3-D Digital Imaging and Modeling*, NRC '97, pages 234–. IEEE Computer Society, 1997.

[27] Konstantinos Konstantinides and Kung Yao. Statistical analysis of effective singular values in matrix rank determination. *IEEE transactions on acoustics, speech and signal processing*, 36(5):757–763, 1988.

[28] Hendrik P. A. Lensch, Wolfgang Heidrich, and Hans-Peter Seidel. Automated texture registration and stitching for real world models. In *Proceedings of the 8th Pacific Conference on Computer Graphics and Applications*, PG '00, pages 317–, Washington, DC, USA, 2000. IEEE Computer Society.

[29] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. The digital michelangelo project: 3d scanning of large statues. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, pages 131–144. ACM Press/Addison-Wesley Publishing Co., 2000.

[30] Marius Muja and David G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *In VISAPP International Conference on Computer Vision Theory and Applications*, pages 331–340, 2009.

[31] Peter J. Neugebauer and Konrad Klein. Texturing 3d models of real world objects from multiple unregistered photographic views. *Computer Graphics Forum*, 18:245–256, 1999.

[32] K. J. Overton and T. E. Weymouth. A noise reducing preprocessing algorithm. In *Proceedings of the IEEE Computer Science Conference on Pattern Recognition and Image Processing*, pages 498–507, 1979.

[33] Kari Pulli, Habib Abi-Rached, Tom Duchamp, Linda G. Shapiro, and Werner Stuetzle. Acquisition and visualization of colored 3d objects. In *Proceedings of the 14th International Conference on Pattern Recognition-Volume 1 - Volume 1*, ICPR '98, pages 11–, 1998.

[34] Kari Pulli, Simo Piiroinen, Tom Duchamp, and Werner Stuetzle. Projective surface matching of colored 3d scans. In *3-D Digital Imaging and Modeling, 2005. 3DIM 2005. Fifth International Conference on*, pages 531 – 538, 2005.

[35] Vladan Rankov, Rosalind J. Locke, Richard J. Edens, Paul R. Barber, and Borivoj Vojnovic. An algorithm for image stitching and blending. In *Proceedings of SPIE*, pages 190–199, 2005.

[36] Erik Reinhard, Michael Ashikhmin, Bruce Gooch, and Peter Shirley. Color transfer between images. *IEEE Comput. Graph. Appl.*, 21(5):34–41, 2001.

[37] C. Rocchini, P. Cignoni, C. Montani, R. Scopigno, Istituto Scienza, and Consiglio Nazionale Delle Ricerche. Acquiring, stitching and blending diffuse appearance attributes on 3d models. *The Visual Computer*, 18:186–204, 2002.

[38] Daniel L. Ruderman, Thomas W. Cronin, and Chuan-Chin Chiao. Statistics of cone responses to natural images: implications for visual coding. *Journal of the Optical Society of America A*, 15:2036–2045, 1998.

[39] Daniel L. Ruderman, Thomas W. Cronin, and Chuan-Chin Chiao. Statistics of cone responses to natural images: Implications for visual coding. *Journal of the Optical Society of America A*, 15:2036–2045, 1998.

[40] Szymon Rusinkiewicz and Marc Levoy. Qsplat: A multiresolution point rendering system for large meshes. In *Siggraph 2000, Computer Graphics Proceedings*, pages 343–352, 2000.

[41] Hanan Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann Publishers Inc., 2006.

[42] Jagan Sankaranarayanan, Hanan Samet, and Amitabh Varshney. A fast all nearest neighbor algorithm for applications involving large point-clouds. *Comput. Graph.*, 31:157–174, April 2007.

[43] Chanop Silpa-Anan and Richard Hartley. Optimised kd-trees for fast image descriptor matching. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8, June 2008.

[44] Riegl Laser Management System. Riscan pro data sheet, September 2009.

[45] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Proceedings fo the Sixth International Conference on Computer Vision*, ICCV '98, pages 839–. IEEE Computer Society, 1998.

[46] John W. Tukey. *Exploratory Data Analysis*. Addison-Wesley, 1977.

[47] Greg Turk and Marc Levoy. Zippered polygon meshes from range images. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, SIGGRAPH '94, pages 311–318. ACM, 1994.

[48] T. Weyrich, M. Pauly, S. Heinzle, S. Scandella, and M. Gross. Post-processing of scanned 3d surface data. In *Symposium On Point-Based Graphics*, pages 85–94, 2004.

[49] Michael Wimmer and Claus Scheiblauer. Instant points. In *Proceedings Symposium on Point-Based Graphics 2006*, pages 129–136. Eurographics, 2006.

[50] Xuezhong Xiao and Lizhuang Ma. Color transfer in correlated color space. In *VRCIA '06: Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications*, pages 305–309, New York, NY, USA, 2006. ACM.

[51] Zhaoxia Xiao and Wenming Huang. Kd-tree based nonuniform simplification of 3d point cloud. In *Proceedings of the 2009 Third International Conference on Genetic and Evolutionary Computing*, WGEC '09, pages 339–342, Washington, DC, USA, 2009. IEEE Computer Society.