UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Markerloses Tracking unter Verwendung von Analyse durch Synthese auf Basis der Ähnlichkeitsbestimmung photorealistischer Bilder

Diplomarbeit

zur Erlangung des Grades eines Diplom-Informatikers
im Studiengang Computervisualistik

vorgelegt von
Thorsten Habelitz

Erstgutachter:	Prof. Dr.-Ing. Stefan Müller (Institut für Computervisualistik, AG Computergraphik)
Zweitgutachter:	Dipl.-Inf. Martin Schumann

Koblenz, im Mai 2009

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

.....
(Ort, Datum) (Unterschrift)

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Augmented Reality bezeichnet die gleichzeitige Darstellung von realen Bildelementen und Elementen der virtuellen Realität. Ihr Ziel ist es, die als Videokamerabild oder durch ein *Head-Up-Display* wahrgenommene Realität durch zusätzliche Informationen und Wahrnehmungen in Form von Computergraphiken und Textinformationen zu erweitern. Dazu muss das Kamerabild so interpretiert werden, dass die Bewegung eines Objekts oder der Kamera selbst verfolgt werden (*Tracking*) kann, um die Kamerapose und letztendlich die eingeblendeten Informationen korrekt zu bestimmen. Neben der Information durch eine Kamera können auch weitere Sensoren eingesetzt werden, um die Verlässlichkeit der berechneten Kamerapose zu verstärken oder um das gesamte Trackingsystem robuster oder schneller zu machen.

Zur Interpretation des Kamerabilds unterscheidet man zwischen *markerbasiertem* und *markerlosem Tracking*. Markerbasiert bedeutet, dass im Kamerabild nach einer vorab definierten, eindeutigen Markierung gesucht wird (*Matching*), ihre Orientierung und Position im Raum interpretiert wird und das Trackingsystem daraus die Pose berechnet. Die *ARToolKit Library* ist beispielsweise eine performante und leicht zugängliche Grundlage für markerbasiertes Tracking.

Nachteil dieser Trackingmethode ist die Platzierung der Marker in der Umgebung. Würde ein Marker verdeckt, hätte ein ausschliesslich darauf basierendes System keine Möglichkeit mehr, Information aus dem Kamerabild zu erhalten. Das Finden von natürlichen Markern (*Features*) oder die Interpretation der gesamten Bildinformation ist Grundlage des markerlosen Trackings. Ein aufwendiger und nicht immer praktikabler Ansatz ist hier, vorab viele Referenzbilder, beispielsweise Photos, für alle möglichen Kamerapositionen und -orientierungen zu generieren und diese mit dem Kamerabild des Trackingsystems zu vergleichen.

Die Idee von *Analyse durch Synthese* ist, das synthetische Bild aus einem 3D Modell des zu trackenden Objekts zu generieren. Das Rendering kann dabei angepasst werden, um den Vergleich mit dem Kamerabild zu fördern. Ansätze sind hier beispielsweise der Vergleich von Kantenbild und Geometrie oder der von stilisierten oder realistischen bzw. detailreichen Bildern.

In dieser Arbeit wird der Ansatz verfolgt, das Rendering möglichst photorealistisch zu gestalten, um es ohne zusätzlichen Bearbeitungsschritt mit dem Kamerabild vergleichen zu können. Besonderes Augenmerk liegt

auf der Anzahl, Position und Farbe der Lichtquellen, realistischen Schattenwürfen und einer detailreichen Geometrie des Modell. Untersucht wird die Ähnlichkeit beider Bilder im Bezug auf interessante Merkmale, die in beiden Bildern vorkommen gefunden und verglichen werden müssen. Um die Beleuchtungssituation jedes Kameraframes einzufangen, wird eine *High Dynamic Range* (*HDR*) Videokamera verwendet.

Auf der anderen Seite steht die Performanz solch eines Systems im Hinblick auf die Verwendung im *AR* Kontext. Photorealistische Computergraphik hat in den letzten Jahren durch programmierbare Graphikhardware und performante Lichtsimulationsansätze auch eine Bedeutung für echtzeitfähige Anwendungen erlangt.

Ziel dieser Diplomarbeit ist es, die Möglichkeiten des Vergleichs von photorealistischen Renderings mit einem Kamerabild zu untersuchen. Es werden etablierte Techniken im Bereich der Ähnlichkeitsbestimmung und des photorealistischen Renderings auf der *GPU* genutzt, um über die Wirkung der Kombination dieser Techniken und ihrer Parameter eine Aussage zu treffen.

1.2 Aufbau

Kapitel 2 bietet einen Überblick über die Definition und vorhergehenden Arbeiten in den Bereichen *Analyse durch Synthese* und photorealistischer Beleuchtungssimulation. In Kapitel 3 wird die wesentliche Technik der *HDR* Kameras, ihre *Kalibrierung* und *Tonemapping*verfahren näher betrachtet. Anschliessend wird der verwendete Algorithmus zur Merkmalsextraktion (*Scale-invariant feature transform*, *SIFT*) und das Ähnlichkeitsmaß beschrieben. Kapitel 4 stellt die verwendete Methode der Beleuchtung vor, ausserdem die darauf basierende Schattengenerierung. Die Ergebnisse des gesamten Tracking Systems werden in Kapitel 5 vorgestellt und analysiert. Kapitel 7 gibt einen Ausblick über mögliche Richtungen und Verbesserungen ausgehend von dieser Arbeit.

2 Überblick

2.1 Analyse durch Synthese

Der Ansatz *Analyse durch Synthese* wurde schon 1961 im Rahmen der Spracherkennung bzw. -analyse von Bell (1961) et al. geprägt. Durch die technische Entwicklung schneller digitaler Computer und Erkenntnisse über die Parameter, die ein Sprachsignal ausmachen, wurde die Idee vertieft, ein synthetisches Signal zu generieren, das mit dem ursprünglichen Sprachsignal vergleichbar sein sollte. Beide Signal werden anschliessend verglichen und der Fehler bzw. die Ähnlichkeit gemessen. Solange die Ähnlichkeit nicht einen bestimmten Schwellwert erreicht, werden weiterhin Signale generiert, bis das System konvergiert. Auch schon in dieser Arbeit wurde darauf hingewiesen, die Schritte bis zur Konvergenz, also die Anzahl der im Computer erzeugten Spektren, möglichst gering zu halten, damit sich der Ansatz *Analyse durch Synthese* mit anderen Analysemethoden messen kann.

Der Ansatz lässt sich auch auf die Analyse von Bildern übertragen. Die synthetische Komponente wird vom Computer erzeugt und basiert auf einem 3D Modell, welches nach Durchlauf der Rendering Pipeline mit einem 2D Kamerabild verglichen werden kann. Durch diese aufwendige Vorarbeit erhält man bei hoher Ähnlichkeit (einzelner Merkmale) zusätzliche Informationen über die Tiefe jedes Pixels und die geometrisch korrekte Darstellung des gesamten Modells, sowie der Kamera relativ zum Objekt.

Moeslund (1999) verwendet *Analyse durch Synthese*, um modellbasiert die Bewegung eines Menschen zu tracken (Motion Capturing). Probleme liegen hier in der Unbeweglichkeit des Modells, das im Vergleich mit den möglichen Bewegungen des menschlichen Körpers und der Kleidung zu unflexibel ist und so die Bestimmung der Ähnlichkeit erschwert. Vergleichbar mit dem markerbasiertem Tracking entstehen bei so einem komplexen Modell wie dem menschlichen Körper auch Fehler und Ungenauigkeiten, wenn wichtige Teile des Bildes verdeckt sind.

Eine weitere Möglichkeit besteht darin, die Geometrie des 3D Modells mit einem Kantenbild zu vergleichen. Mit dem Ziel, den Rückprojektionsfehler zu minimieren, werden Punkt- und Linienkorrespondenzen gesucht, um aus ihnen eine Kamerapose zu berechnen. Als Einschränkung für die Suche nach Merkmalen kann angenommen werden, dass diese nur im Umkreis um ein im einen Bild gefundenes Merkmal oder auf einer Linie liegen können. Eine Kombination dieser Korrespondenzen werden von Ewering (2006) und Achilles (2008) verwendet, um die Verlässlichkeit der berechneten Pose zu erhöhen.

Schumann (2008) generiert aus den Parametern der Rotation und Translation in drei Dimensionen verschiedene Ansichten des Modells, so dass anschliessend jeweils die Ähnlichkeit zwischen dem synthetischen Bild und dem Kamerabild bestimmt werden kann. Die ähnlichste Ansicht bildet die Grundlage für das folgende Kameraframe, um sich so iterativ der korrekten Kamerapose zu nähern. Das verwendete Ähnlichkeitsmaß basiert hier nicht auf dem Vergleich einzelner Merkmale, sondern es werden die Intensitäten aller Pixel der Bilder verglichen. Um Kamerabild und Computergraphik auf die selbe Abstraktionsebene zu bringen und so die Ähnlichkeitsbestimmung zu verbessern, werden die Bilder stilisiert.

Analyse durch Synthese ist ein Ansatz, der durch die steigende Leistungsfähigkeit moderner Computer realisiert wird und in seinen Möglichkeiten der Genauigkeit, Verlässlichkeit und Schnelligkeit mit der Hardware skaliert. Im Hinblick auf eine Verwendung in der AR muss der Aufwand der Ähnlichkeitsbestimmung, des Renderings und der Analyse des Kamerabild sinnvoll aufgeteilt sein.

2.2 Beleuchtung

In dieser Diplomarbeit soll versucht werden, das synthetische Bild der Methode *Analyse durch Synthese* so photorealistisch wie möglich zu rendern, um ohne weitere Verarbeitungsschritte die Merkmalskorrespondenzen in den Bildern zu finden.

Nakamae u. a. (1986) haben gezeigt, dass virtuelle Objekte, in diesem Fall Gebäude, glaubhaft in Photographien eingebracht werden können. Dazu wird etwa die Position der Kamera gemessen und die Position der Sonne durch Datum und Zeitpunkt bestimmt, um eine konsistente Beleuchtung zu erzeugen. Debevec (1998) nutzt das Bild einer Light Probe, welches von einer HDR Kamera aufgenommen wurde, um virtuelle Objekte zu beleuchten. Einen ähnlichen Ansatz verfolgen Sato u. a. (1999), indem das Bild einer mit einem Fischaugenobjektiv ausgestatteten Kamera Grundlage für die Beleuchtung ist.

Havran u. a. (2005) erweitern diesen Ansatz, indem eine HDR Videokamera mit Fischaugenobjektiv genutzt wird, um bewegte Bilder, sogenannte *Video Environment Maps*, zu generieren. Besonders wird hier auch auf die Kohärenz der besten Lichtquellen von Frame zu Frame geachtet, so dass unnatürliches Flackern der Beleuchtung bzw. der Schatten vermieden wird. Schatten werden durch ein *Multi-Pass Shadow-Mapping* für jede Lichtquelle erzeugt und tragen durch Blending zur gesamten Beleuchtung innerhalb eines Frames bei.

Das hier entwickelte Trackingsystem basiert auf *Analyse durch Synthese* und photorealistischer Beleuchtungssimulation. Es erlaubt grundsätzlich die interaktive Bewegung der Kamera und des zu trackenden Objekts. Die Beleuchtung beschränkt sich auf das direkte Licht, das heißt unendlich weit entfernte Lichtquellen, wobei die Beleuchtungssituation allerdings interaktiv verändert werden kann.

3 Bildverarbeitung

3.1 Kamera

Es werden zwei HDR Kameras verwendet. Eine fängt durch ein Fischaugenobjektiv das einfallende Licht der darüber liegenden Hemisphäre ein. Abbildung 1 zeigt das von dieser Kamera gelieferte Bild. Zu beachten ist, dass diese Fischaugenkamera idealerweise genau an der Position des zu trackenden Objekts liegen sollte, um die korrekte Beleuchtungssituation aufzunehmen. Da dies jedoch physikalisch nicht möglich ist, wird die Kamera möglichst nahe neben dem Objekt positioniert und nach oben ausgerichtet. Ausserdem wird die Kamera in einer Ebene mit dem Objekt positioniert, um den Fehler durch einen Höhenunterschied zu minimieren. Die zweite HDR Kamera ist auf das Objekt gerichtet und bildet die Grundlage für den Vergleich mit dem synthetischen Bild.

Abbildung 1: Bild der HDR Kamera mit Fischaugenobjektiv

Die verwendeten HDR Videokameras sind sogenannte *HDRC* (*HDR CMOS*) Kameras des Typs *GEVILUX CAM1xCL Color*. HDRC Kameras zeichnen sich durch einen erhöhten Dynamikumfang aus, es können Szenen mit extremen Helligkeitsunterschieden erfasst werden. Diese hohe Kontrastauflösung wird durch eine logarithmische Komprimierung der einfallenden Helligkeit auf ein begrenztes Signal erreicht, ähnlich der logarithmischen Adaption des menschlichen Auges. Der entsprechende digitale Datenstrom hat eine Auflösung von 12 Bit.

3.1.1 Bildentstehung

Dieses Kapitel beschreibt die Transformationen der Geometrie des 3D Modells. Die Parametrisierung des Bildes der Kamera mit Fischaugenobjektiv

wird in Kapitel 4.2 beschrieben.

Im Folgenden soll die Transformation der Geometrie in Objektkoordinaten zu Pixelkoordinaten beschrieben werden. Die Geometrie in Pixelkoordinaten bildet die Syntheseeinheit im Ansatz *Analyse durch Synthese*. Die Transformation entspricht der Matrizenmultiplikation mit *Modelview-*, *Projection-* und *Viewport-Matrix*.

Zuerst wird von Objektkoordinaten nach Weltkoordinaten transformiert, um das Objekt in der Welt zu positionieren. Anschliessend wird die Kamera durch Rotation und Translation positioniert, so dass das Objekt aus Sicht der Kamera gezeigt wird, es befindet sich nun in Kamerakoordinaten. Dies kann durch die $4x4$ *Modelview-Matrix* M, die mit einem Punkt p_w multipliziert den Punkt p_k ergibt, beschrieben werden:

$$p_k = Mp_w = \begin{pmatrix} Rotationsmatrix & Translationsvektor^T \\ 0 & 1 \end{pmatrix} p_w \qquad (1)$$

Die Projektion der Kamerakoordinaten auf Bildkoordinaten geschieht durch die *Projection-Matrix* P:

$$p_b = Pp_k = \begin{pmatrix} 2\frac{f_x}{w} & 0 & -2\frac{p_x}{w} + 1 & 0 \\ 0 & 2\frac{f_y}{h} & 2\frac{p_y}{h} - 1 & 0 \\ 0 & 0 & \frac{far+near}{near-far} & 2\frac{far*near}{near-far} \\ 0 & 0 & -1 & 0 \end{pmatrix} p_k \qquad (2)$$

Brennpunkt (f_x, f_y) und Hauptpunkt (p_x, p_y) entsprechen den bei der intrinsischen Kamerakalibrierung gemessenen Werten. Das Bild sollte dabei natürlich die selben Werte für Breite w und Höhe h haben. Der Abstand zwischen Far und $near$ sollte so gering wie möglich gewählt werden, dass die Präzision des Tiefenpuffers entsprechend genau bleibt. Abschliessend werden Bildkoordinaten noch in Pixelkoordinaten transformiert:

$$p_p = Vp_b = \begin{pmatrix} m_x & 0 & c_x \\ 0 & m_y & c_y \\ 0 & 0 & 1 \end{pmatrix} p_b = VPMp_w \qquad (3)$$

(c_x, c_y) beschreibt die Translation in den Ursprung des neuen Koordinatensystems, für OpenGL typischerweise in der linken, unteren Ecke. (m_x, m_y) ist die Skalierung durch die neue Breite und Höhe.

Damit können die Transformation in intrinsische (*Viewport-* und *Projection-Matrix*) und extrinsische (*Modelview-Matrix*) aufgeteilt werden.

3.1.2 Kalibrierung

Um das Bild der HDR Videokamera mit dem der virtuellen Kamera zu vergleichen, müssen bestimmte Eigenschaften der Kamera durch Kalibrierung bestimmt werden. Man unterscheidet dabei zwischen intrinsischer, photometrischer und kolorimetrischer Kalibrierung.

Die intrinsische Kalibrierung bestimmt diejenigen Parameter, die sich durch eine bestimmte Brennweite, sowie allgemein durch das Verhältnis von Objektiv und Sensor ergeben. Grundlage für die intrinsische Kamerakalibrierung sind verschiedene Aufnahmen eines Kalibriermusters. Die weitverbreitete *Camera Calibration Toolbox for Matlab* und die Implementation in C++ im Rahmen der *OpenCV Library* nutzen ein Schachbrettmuster mit bekannten Ausmaßen, um über die stark kontrastierenden, schwarzen und weißen Flächen Punkte und aus diesen die Parameter der Kamera zu berechnen. Die hier genutzten Parameter sind von einer vorhergehenden Kalibrierung der Kamera (durch *Camera Calibration Toolbox for Matlab*) übernommen, nachdem sie durch die Kalibrierung verschiedener Aufnahmen (durch die Kalibierung mit *OpenCV*) bestätigt werden konnten. Die Parameter, die in der Projektionsmatrix der virtuellen Kamera verwendet werden sind: Brennpunkt (f_x, f_y) = (823.18351, 822.50013), Hauptpunkt (p_x, p_y) = (255.66649, 241.28068), bei einer Bildgröße von 512 x 496 Pixeln. Verzerrung und Skew werden vernachlässigt.

Photometrische Kalibrierung bestimmt die Abbildung der ausgegebenen RGB Farbwerte der Kamera auf die Leuchtdichte, es soll also der Prozess der Kamera kalibriert werden, der die tatsächliche Leuchtdichte in eine Ausgabe umwandelt. Ist die Abbildung bekannt, kann der RGB Farbwert des Bildes genutzt werden, um mit der korrekten Lichtfarbe zu beleuchten, das Bild mit Hilfe von Tonemapping korrekt darzustellen oder um die Reflektionseigenschaften eines Materials zurückzurechnen.

Die Leuchtdichte lässt sich durch die Multiplikation des Vektors des RGB Farbwertes, welcher von der Kamera ausgegeben wird, mit dem Vektor $(0.2126, 0.7152, 0.0722)$ berechnen. Dies entspricht der Transformation vom *sRGB Farbraum* in den *CIE XYZ Farbraum*, wobei die mittlere Zeile der Transformationsmatrix genutzt wird. Die Y Kurve entspricht der $V(\lambda)$ Kurve, also der Helligkeit, die durch das menschliche Auge wahrgenommen wird. Die so resultierende Leuchtdichte wird vor allem durch den Anteil des Grün Wertes beeinflusst, während Rot weniger Einfluss besitzt und Blau kaum zur Helligkeit beiträgt. Eine für die HDR-Kameras zu bestimmende spezifische Transformationsmatrix würde die korrekte Bestimmung der Leuchtdichte verbessern.

Kolorimetrische Kalibrierung umfasst in diesem Fall die Abbildung des

RGB Farbraums der Kamera auf einen standardisierten Farbraum. U.a. umfasst dies die Darstellung der korrekten Farbtemperatur, welche durch einen Weißabgleich bestimmt wird. Grundlage ist die Aufnahme einer weißen Fläche bzw. eines sogenannten *ColorCheckers*, einer Tafel mit standardisierten Farben.

3.1.3 Weißabgleich

Um die HDR Farbwerte korrekt auf dem Bildschirm darzustellen, kann eine Reihe von Abbildungen vorgenommen werden: Weißabgleich, Farbsättigung, Tonemapping und eine zusätzliche Gamma-Korrektur. Ziel ist, den subjektiven Eindruck von Kamerabild und synthetischem Bild anzugleichen.

Die hier verwendete Methode des manuellen Weißabgleichs wird auf das komprimierte LDR Bild, das heißt erst nach dem Tonemapping, angewandt. Dabei wird zunächst manuell eine geeignete weiße oder graue Fläche im Bild ausgewählt und deren durchschnittlicher Farbwert bestimmt. Das Motiv hier ist der *ColorChecker*, eine Tafel mit standardisierten Farben. Anschliessend wird eine Abbildung der Farben in Form einer *Lookup-Tabelle* initialisiert. Da der grüne Farbkanal wahrscheinlich die genaueste Repräsentation der Farbe im Bild enthält, basiert die nachfolgende Manipulation der Lookup-Tabelle auf diesem Wert. Ausserdem wird der Wert von Blau gespeichert.

Die Kurve bzw. jeder Wert r der *Lookup-Tabelle* für Rot wird nun durch 255 geteilt (Diese Methode setzt ein 8 Bit Bild voraus) und mit dem zuvor bestimmten Wert von Grün, der Referenz, multipliziert.

$$r' = g_{white} * \frac{r}{255} \tag{4}$$

Nun muss noch die Kurve von Blau angepasst werden, um alle drei Farbkanälen aneinander anzugleichen:

$$b' = \begin{cases} g_{white} * \frac{b}{b_{white}} & \text{falls } b' < 255 \\ 255 & \text{sonst} \end{cases} \tag{5}$$

3.1.4 Farbsättigung

Durch das Tonemapping bzw. die gewählten Parameter für α (und γ) werden nicht unbedingt korrekt gesättigte Farben erzeugt, so dass durch eine zusätzliche Farbsättigung vorab ein besserer Bildeindruck entstehen kann.

Voraussetzung für eine korrekte Darstellung ist, dass zuvor ein Weißabgleich vorgenommen wird. Um einen beliebigen Farbwert $c = (r, g, b)^T$ zu verstärken, muss dieser zuerst auf die normalisierte Weißachse $w = (\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}})^T$ projeziert werden:

$$v = (c \bullet w) * w \tag{6}$$

$$v = (r * \frac{1}{\sqrt{3}} + r * \frac{1}{\sqrt{3}} + r * \frac{1}{\sqrt{3}}) * (\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}})^T \tag{7}$$

$$v = (\frac{r+g+b}{3}, \frac{r+g+b}{3}, \frac{r+g+b}{3})^T \tag{8}$$

Für jeden Farbkanal k lässt sich die mit Faktor δ skalierte, gesättigte Farbe berechnen:

$$c'_k = \delta(c_k - v) + v \tag{9}$$

Abbildung 2 zeigt die Wirkung der Farbverstärkung.

(a) (b)

Abbildung 2: Farbverstärkung: Niedrige Farbverstärkung (a) und hohe Farbverstärkung (b) in Szene 2

3.1.5 Tonemapping

Tonemapping bezeichnet in diesem Fall die Abbildung der HDR Farbwerte auf die durch den Bildschirm darstellbaren Werte. Dabei wird zusätzlich zu dem auf der Arbeit von Reinhard u. a. (2002) basierenden Tonemapper abschliessend eine Gamma-Korrektur durchgeführt. Tonemapping, Farbsättigung und Weißabgleich werden nach der Lichtsimulation und dem Matching auf der GPU berechnet, das heißt, sie dienen nur der Darstellung. Lichtsimulation, Matching, etc. rechnen mit den HDR Werten in 12 Bit. Eingabetexturen für den Fragmentshader sind das Kamerabild und das virtuelle Objekt, das als Ausgabetextur eines *Frame Buffer Objects* vorliegt.

Reinhard u. a. (2002) orientieren sich an dem in der Photographie eingesetzten *Zonen System* von Ansel Adams, welches bestimmte Bereiche von Leuchtdichten Zonen zuordnet, um diese auf eine geringere Anzahl Zonen für den Druck abzubilden. Das Mapping wird durch den sogenannte *Key* beeinflusst, der die subjektive Helligkeit der gesamten Szene beschreibt. Der *Key* wird approximiert, indem zuerst vorab die durchschnittliche, logarithmisch skalierte Leuchtdichte bzw. Helligkeit aus allen Pixeln (x, y) berechnet wird:

$$\overline{L}_w = \frac{1}{N} \exp\left(\sum_{x,y} \log(\delta + L_w(x, y))\right) \tag{10}$$

$L_w(x, y)$ ist die Leuchtdichte des Pixels (x, y), welche, wie in Kapitel Kalibrierung beschrieben, ermittelt wird. δ ist ein möglichst kleiner Wert ungleich 0, um die Singularität bei fehlerhaften, schwarzen Pixeln im Bild zu vermeiden. Ziel ist, diese Helligkeit auf ein geeignetes Grau abzubilden:

$$L(x, y) = \frac{\alpha}{\overline{L}_w} L_w(x, y) \tag{11}$$

α ist der sogenannte *Key-Wert*, welcher, auf einer von 0 bis 1 skaliert, den Grauwert beschreibt, auf den abgebildet werden soll. Typischerweise wird hier für ein normal belichtetes Bild der Wert 0.18 gewählt. Zuletzt werden hohe Leuchtdichten im Bild noch durch folgende Gleichung komprimiert:

$$L_d(x, y) = \frac{L(x, y)}{1 + L(x, y)} \tag{12}$$

Dieser Schritt garantiert die Abbildung der HDR Information in den LDR Raum. Hohe Leuchtdichten werden annähernd mit $\frac{1}{L}$ skaliert, während niedrige Leuchtdichten mit 1 skaliert werden.

Zusätzlich gibt es noch die Möglichkeit, die Ausgabe durch einen Gamma-Wert zu korrigieren:

$$p(x,y) = \left(\frac{p(x,y)}{L_{max}}\right)^{\gamma} \tag{13}$$

L_{max} ist die maximale Leuchtdichte des aktuellen Bilds, γ hat den Wert 2.2. Abbildung 3 zeigen die Auswirkungen eines unterschiedlich gewichteten *Key-Wertes* auf die Szene, ähnlich der Über- und Unterbelichtung.

(a) (b) (c)

Abbildung 3: Tonemapping Key: High key (a), normal key (b) und low key (c) in Szene 2

3.1.6 Implementation

Das Trackingsystem verarbeitet HDR Bilder, welche erst abschliessend zur Darstellung in ihrer Dynamik komprimiert werden. Dabei wird das synthetische Bild in ein Frame Buffer Object gerendert, um es anschliessend durch den in 1 vorgestellten Fragment Shader zu komprimieren.

Eingabe des Shaders sind Texturen des Kamerabilds und des synthetischen Bilds. Ausserdem werden die Parameter δ der Farbsättigung und γ der zusätzlichen Gamma-Korrektur durch Benutzereingaben berücksichtigt. Der *Key* des Tonemappings von Reinhard u. a. (2002) wird indirekt über die Eingabe von α gewichtet mit der durchschnittlichen Leuchtdichte beeinflusst, im selben Schritt wird auch die maximale Leuchtdichte berechnet.

```
uniform sampler2D tex_1, tex_2;
uniform float delta_value;
uniform float gamma_value;
uniform float key_value;
uniform float max_luminance_value;
```

```
void main (void)
{
    vec2 at = gl_TexCoord[0].xy;
    vec3 pixel_1 = texture2D(tex_1, at).xyz;
    vec4 pixel_2_alpha = texture2D(tex_2, at).xyzw;

    vec3 pixel_2 = pixel2_alpha.xyz;

    float c = 0.333 * (pixel_1.x + pixel_1.y + pixel_1.z);
    pixel_1 = delta_value * (pixel_1 - c) + c;

    vec3 y_scales = vec3(0.2127, 0.7152, 0.0722);
    float y = dot(y_scales, pixel_1);
    float y_a = key_value * y;
    float y_d = y_a / (y_a + 1.0);
    pixel_1 = y_d / y * pixel_1;

    pixel_1 = pow((pixel_1 / max_luminance_value), gamma_value);

    gl_FragColor.rgb = pixel_2_alpha.w * pixel_1 + pixel_2;
    // gl_FragColor.rgb = (pixel_2 + pixel_1) / 2.0;
};
```

Listing 1: *Fragment Shader: HDR Tonemapping*

3.2 Tracking

Um das Tracking durch den Ansatz *Analyse durch Synthese* möglich zu machen, muss das System einmalig initialisiert werden. Dazu wird das 3D Modell als Grundlage des synthetischen Bildes geladen. Ausserdem wird begonnen, die einzelnen Bilder, die vorab gespeicherten Frames eines Videos, zu laden. Die gesamte Sequenz der Bilder kann vorab nicht geladen und zwischengespeichert werden, um beispielsweise beim Tracking einen schnelleren Ablauf zu erzeugen, weil davon ausgegangen wird, dass die Bilder von einer laufenden Kamera stammen, die kontinuierlich neue Bilder aufnimmt. Für diese Arbeit werden allerdings vorab aufgenommene Sequenzen von Bildern verwendet und von der Festplatte geladen, eine direkte Anbindung an die beiden HDR Kameras wäre jedoch auch möglich. Das 3D Modell wird vorab per Hand durch Rotation und Translation in die ungefähre Pose des Kamerabildes gebracht. Die initiale Poseberechnung wird hier vernachlässigt.

Im Folgenden soll die Optimierung des Ähnlichkeitsmaßes im Rahmen von *Analyse durch Synthese* zum Finden der besten Lösung beschrieben werden. Eine Lösung bzw. das Ähnlichkeitsmaß basiert auf Punkten, welche durch den SIFT Punktdetektor gefunden und durch den SIFT Deskriptor beschrieben werden. Anschliessend werden diese Deskriptoren verglichen (*Matching*), um über das Ähnlichkeitsmaß die Pose der Kamera zu berechnen. Abschliessend sollen die wesentlichen Schritte der Implementierung vorgestellt werden.

3.2.1 Optimierung Analyse durch Synthese

Aufgabe der Optimierung ist, die Ähnlichkeitsfunktion im Rahmen von *Analyse durch Synthese* so anzupassen, dass ihr Fehler möglichst gering wird. Die Idee ist, Ansichten aller möglichen Transformationen des Modells zu generieren und diese mit dem Kamerabild zu vergleichen. Mögliche Transformationen sind Translation in beide Richtungen auf allen drei Achsen, sowie Rotation in beide Richtungen um alle drei Achsen. Dabei wird davon ausgegangen, dass sich die Ähnlichkeitsfunktion durch eine lokale Suche optimieren lässt, da lediglich von kleinen Bewegungen der Kamera ausgegangen wird. Insgesamt ergeben sich so 12 mögliche Ansichten, zusätzlich kann noch die aktuelle, unveränderte Ansicht weiter verwendet werden, falls ihre Ähnlichkeit höher als die der neu generierten Ansichten ist. Diejenige Ansicht mit der besten Ähnlichkeit bildet die Grundlage für das nächste Kameraframe. Abbildung 4 zeigt die 6 Ansichten bei einer Rotation von 10 Grad um die jeweilige Achse. Hinzu kommt die aktuelle Ansicht.

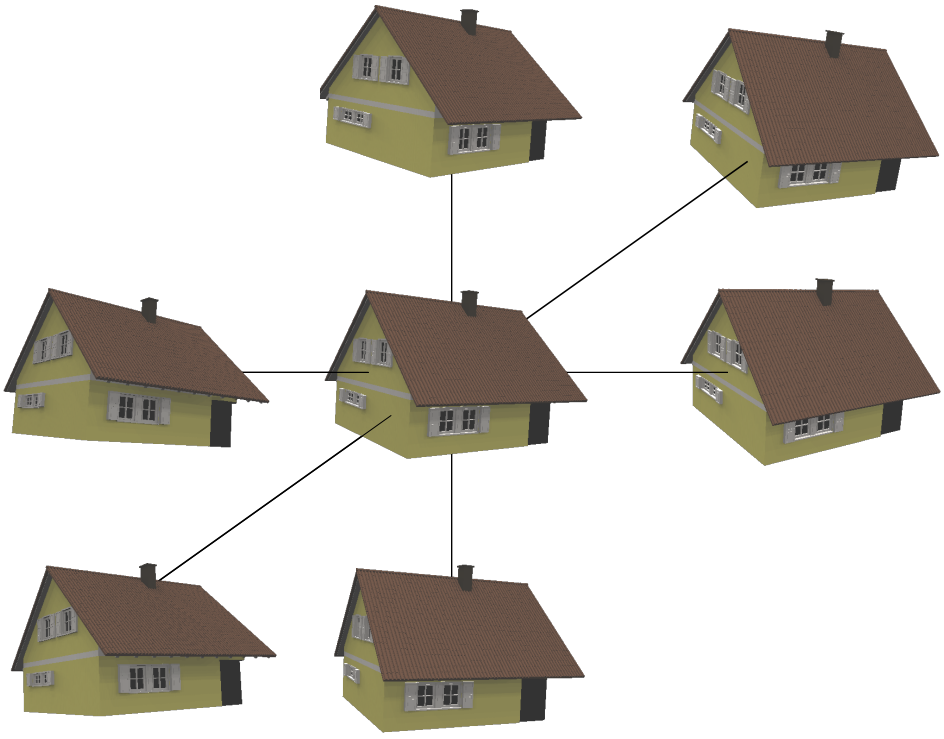


Abbildung 4: Ansichten der Rotation um 10 Grad um die drei Achsen, ausserdem die aktuelle Ansicht

Der Grad der Rotation und Translation wird abhängig von der Ähnlichkeit zwischen Kamerabild und synthetischem Bild adaptiv angepasst. Bei hoher bzw. zunehmender Ähnlichkeit wird davon ausgegangen, dass sich das globale Maximum in der Nähe befindet, so dass auch nur eine geringe Rotation oder Translation ausreicht, um die Funktion zu optimieren. Rotation und Translation werden verkleinert.

Auf der anderen Seite wird bei geringer bzw. abnehmender Ähnlichkeit von einer schnellen Bewegung ausgegangen oder dass das Trackingsystem erst initialisiert wird. Rotation und Translation werden entsprechend vergrößert. Sollte über mehrere Frames keine Veränderung der Ähnlichkeit vorkommen, kann dies auf ein lokales Maximum hindeuten. In diesem Fall werden Rotation und Translation je nach Anzahl der Frames, die dieses Maximum andauert, ebenfalls vergrößert oder verkleinert, um mögliche bessere Ansichten zu finden oder um sie auszuschliessen.

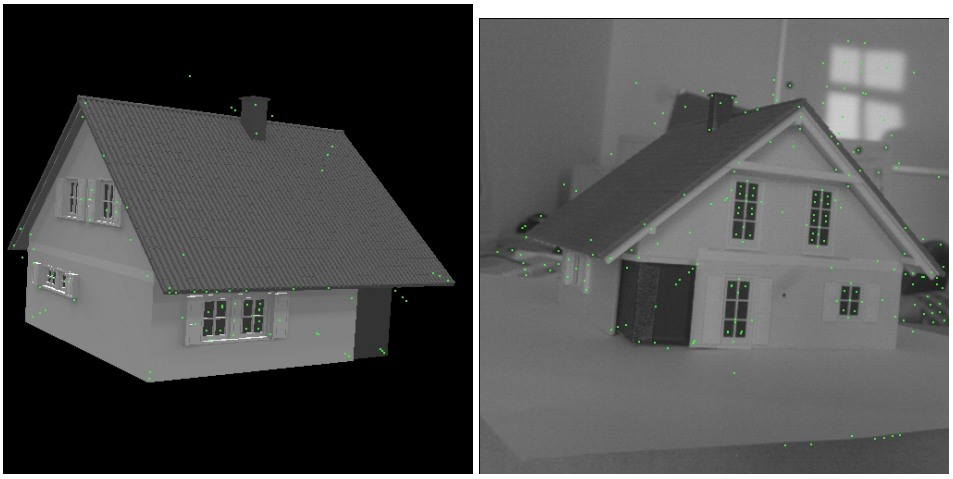
Ausserdem ist Aufgabe der Optimierung, die Ähnlichkeit in minimaler Zeit, das heißt nach möglichst geringer Anzahl von Frames, zu maximieren. Dem gegenüber stehen lokale Maxima, die erkannt oder vermieden werden müssen, um keine fehlerhafte Pose zu generieren. Die Idee ist, durch Einstellung der Parameter für Rotation und Translation möglichst lokalen Maxima vorzubeugen. Falls jedoch über mehrere Frames keine Veränderung bzw. Verbesserung der Ähnlichkeit festgestellt wird, wird wie oben genannt die Translation und Rotation verändert.

3.2.2 Punktdetektor SIFT

Um die Ähnlichkeit von Kamerabild und synthetischem Bild zu bestimmen, werden Interessenspunkte bzw. Merkmale in beiden Bildern gesucht, damit diese miteinander verglichen werden können. Diejenigen Interessenspunkte, die in beiden Bildern auftauchen, werden einander zugeordnet. Diese Punktkorrespondenzen bilden die Grundlage für die Ähnlichkeitsbestimmung. Abbildung 5 zeigt durch den verwendeten SIFT Detektor gefundene Merkmale in einem Kamera- und synthetischem Bild.

Scale Invariant Feature Transform (*SIFT*) von Lowe (2004) ist ein Verfahren, welches Merkmale unabhängig von ihrer Skalierung und Orientierung findet bzw. zueinander zuordnet. Genau genommen beschreibt SIFT dabei einen *Difference of Gaussians* (*DoG*) Detektor und einen eigenen Deskriptor. Der SIFT Deskriptor, welcher die gefundenen Merkmale beschreibt und Grundlage des Vergleichs ist, wird im nächsten Kapitel näher beschrieben.

Interessenspunkte findet der Detektor an Extremstellen im differentiellen Skalenraum eines Bildes. Der Skalenraum wird erstellt, indem das Bild $I(x, y)$ mit einem Gaußfilter $G(x, y, \sigma)$ gefaltet wird:



(a) (b)

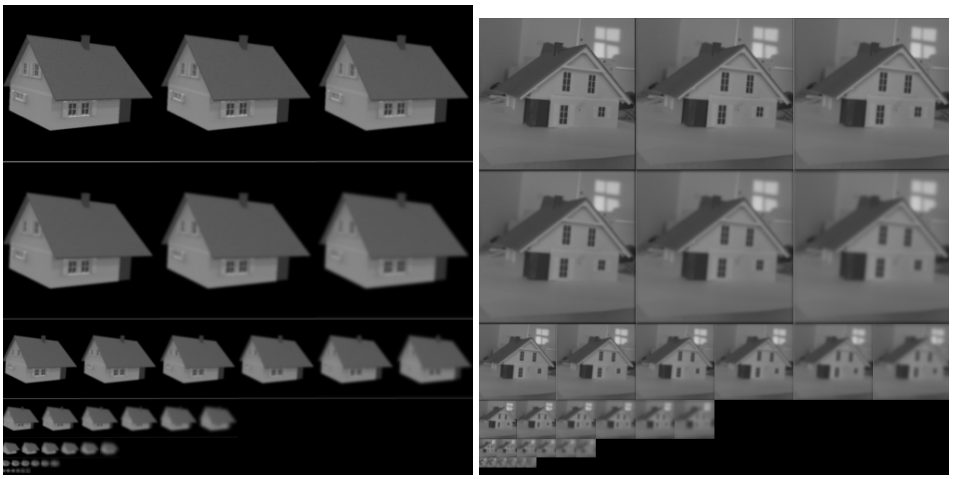
Abbildung 5: SIFT Merkmale eines synthetischen Bilds (a) und eines Kamerabild (b)

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{14}$$

Zuerst wird das Bild in verschiedenen Auflösungen erstellt, welche zusammen übereinander gelagert der Struktur einer Pyramide ähneln. Abbildung 6 zeigt die Ebenen einer Pyramide für ein Kamera- und synthetisches Bild. Auf jeder dieser Ebenen der Pyramide, also in einer bestimmten Bildauflösung, wird nun das Bild mehrmals mit konstantem Abstand k in Richtung σ inkrementell mit einem Gaußfilter gefaltet. Diese Ebenen werden *Oktaven* genannt. Die Differenz zwischen zwei benachbarten Ebenen innerhalb einer Oktaven, daher auch *Difference of Gaussians* (*DoG*), bildet den differentiellen Skalenraum. (Abbildung 7) DoG ist eine Approximation des *Laplacian of Gaussian* (*LoG*) Operators $\sigma\nabla^2 G$, welcher sehr stabile, invariante Merkmale findet, wenn die Normalisierung durch den Skalenparameter σ^2 verwendet wird.

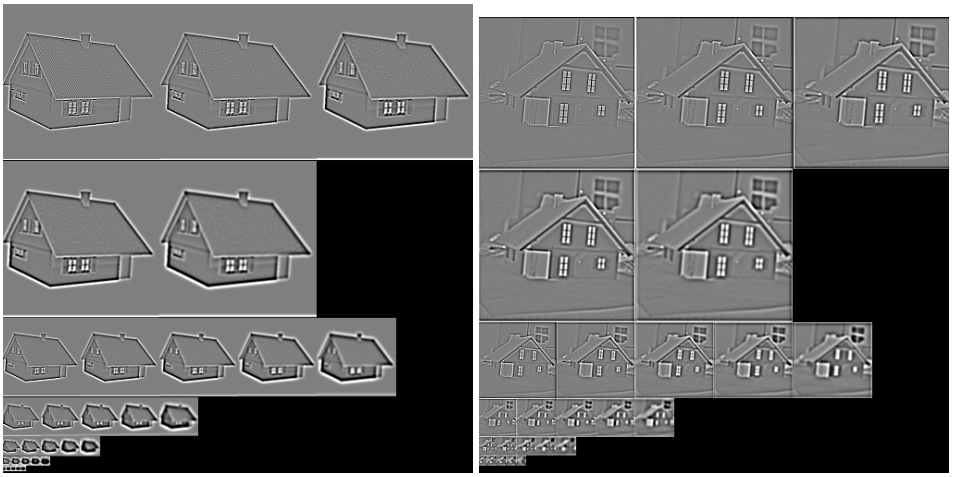
$$\begin{aligned} D(x, y, \sigma) &= (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) && (15)\\ &= L(x, y, k\sigma) - L(x, y, \sigma) && (16)\\ &\approx ((k-1)\sigma^2\nabla^2 G) * I(x, y) && (17) \end{aligned}$$

Die Extremstellen dieses differentiellen Skalenraums, die lokalen Maxima und Minima, werden als Merkmalskandidat angesehen, wenn dieser Punkt größer oder kleiner als alle Punkte in seiner Nachbarschaft ist. Die



(a) (b)

Abbildung 6: Gaußpyramiden



(a) (b)

Abbildung 7: DoG

Nachbarschaft umfasst die umliegenden 8 Punkte auf der selben Ebene innerhalb der Oktave, sowie jeweils 9 Punkte auf der nächsten und vorherigen Ebene.

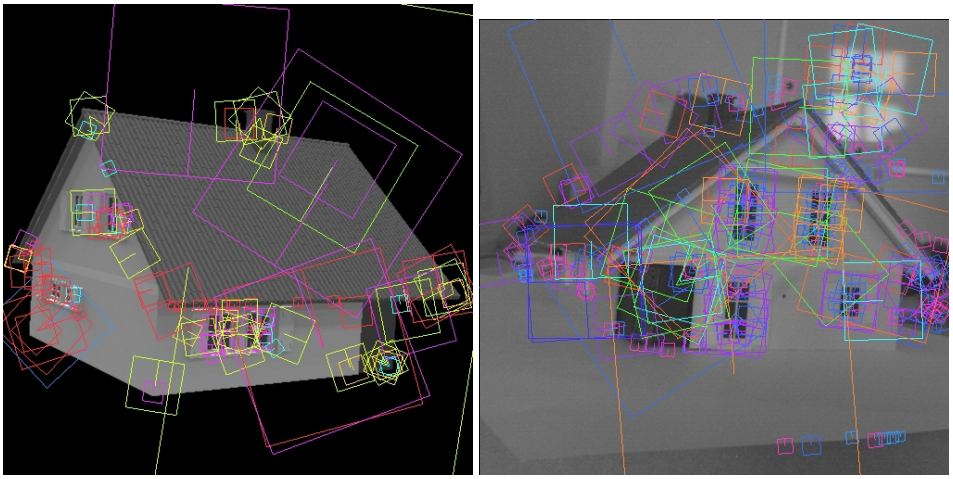
Die Merkmalskandidaten werden anschliessend näher betrachtet, um ungeeignete Kandidaten auszuschliessen. Es wird die genauere *Subpixelposition* des Kandidaten bestimmt, ausserdem der Kontrast dieses Extremums durch die Differenz mit der Nachbarschaft. Liegt der Kontrast unter einem bestimmten Schwellwert, wäre es sinnvoll, dieses Extremum nicht weiter zu berücksichtigen. Zusätzlich müssen diejenigen Kantenpixel verworfen werden, deren Position auf der Kante durch Rauschen nicht eindeutig bestimmt werden kann. Dazu wird die *2x2 Hesse-Matrix* dieses Kandidaten bestimmt, dabei muss das Verhältnis des größeren Eigenwerts zum kleineren unter einem bestimmten Schwellwert liegen.

Die Orientierung eines Merkmals relativ zum Bild wird durch seine Nachbarschaft bestimmt. Gewichtet mit der Steigung (Abbildung 8) und einem Gauß gewichteten Fenster werden die Nachbarschaftspunkte in ein Histogramm eingebracht, welches auf 36 mögliche Orientierungen in 10 Grad Schritten aufgeteilt ist. Diejenige Orientierung mit dem größten Gewicht bestimmt letztendlich die Orientierung des Merkmals. Abbildung 9 zeigt durch Quadrate markierte Merkmale, wobei die Orientierung der Quadrate der Orientierung der Merkmale entspricht.

(a) (b)

Abbildung 8: Gradient der Pyramide

Grabner u. a. (2006) haben Verbesserungen des *SIFT* Ansatzes von Lowe vorgeschlagen, welcher recht aufwendig ist und sich nicht ohne weiteres in Echtzeitumgebungen einsetzen lässt. Lowe verwendet für die erste Oktave



(a) (b)

Abbildung 9: Quadrate markieren Merkmale und beschreiben Orientierung und Skalierung

das Bild in doppelter Auflösung, wobei bei dem Ansatz von Grabner u.a. mit der originalen Auflösung begonnen wird. Ausserdem wird der Gaußfilter durch einen Mittelwertfilter, genauer der *Difference of Gaussians* durch *Difference of Mean* (*DoM*), ersetzt.

Grundlage des Mittelwertfilters ist die Umwandlung der Bilder zu *Integralbildern*. Dieses von Viola und Jones (2001) im Rahmen der Objekterkennung vorgeschlagene Format bildet für jeden Punkt $p(x, y)$ die Summe aller Punkte $p(x', y')$ mit $x' \leq x$ und $y' \leq y$. Sobald das Integralbild berechnet ist, kann der Mittelwert einer rechteckigen Region in konstanter Zeit berechnet werden, unabhängig von der Größe dieser Region. Die inkrementelle Faltung kann so durch Vergrößerung dieser Region effizient berechnet werden. Die berechneten Merkmale sind nicht so präzise wie die durch *DoG* oder *LoG* gefundenen Merkmale, sie eignen sich für Objekterkennung und Tracking, sind allerdings weniger geeignet, die *Fundamentalmatrix* zu schätzen.

Die Implementierung von SIFT auf der GPU (*SiftGPU*) von Wu (2007) verfolgt zwar den ursprünglichen Ansatz von Lowe, gleicht dies aber durch die parallele Verarbeitung der Pixel aus. Dadurch kann die Pyramide aus Gauß gefilterten Bildern verschiedener Auflösung, sowie der *Difference of Gaussians* im Vergleich zur Berechnung auf der CPU in kürzerer Zeit durchgeführt werden. Ausserdem wird eine effiziente Datenstruktur der Kandidaten, der sogenannten Keypoints, genutzt, um parallel den Deskriptor und die Orientierung eines Merkmals zu bestimmen. *SiftGPU* setzt aktuelle Graphikhardware voraus und wird im hier entwickelten Trackingsystem

eingesetzt, um die Merkmale zu finden und sie miteinander zu vergleichen. Es werden *CG*, *GLSL* und *CUDA* unterstützt, wobei in diesem System die *GLSL* Variante benutzt wird, um konsistent mit dem übrigen Shadercode zu bleiben. Eine zusätzliche Auswertung der korrespondierenden Punktmerkmale, näher beschrieben in Kapitel *Matching*, ist jedoch nötig, um ihre Ähnlichkeit präziser zu bestimmen.

3.2.3 Deskriptor SIFT

Der von Lowe entwickelte *SIFT Deskriptor* beschreibt Merkmale eindeutig genug, um etwa auch aus einer großen Menge von möglichen Korrespondenzen die korrekte herauszufinden. Der Deskriptor enthält insgesamt 128 Einträge, welche Informationen von 8 quadratischen 4x4 Regionen in der Nachbarschaft des zu untersuchenden Merkmals beschreiben. Der Gradient und die Orientierung jedes Punktes innerhalb dieser Regionen wird berechnet, wobei diese relativ zum untersuchenden Merkmal rotiert werden, um invariant gegenüber Rotation zu sein. Um das Merkmal herum werden diese Regionen mit einer kreisförmigen Gaußmaske gewichtet, um den Abstand der Gradienten zum Merkmal zu betonen.

Für jede dieser 4x4 Regionen wird ein Gradientenhistogramm erstellt, welches die Gradienten einer bestimmten Orientierung in 45 Grad Schritten zusammenfasst. Durch diese Histogrammbildung wird der Deskriptor tolerant gegenüber in ihrer Position innerhalb einer Region verschobenen Gradienten. Die Einträge der Histogramme bilden die Einträge des SIFT Deskriptors.

Es existieren ausserdem Ansätze, die Größe des Deskriptors zu reduzieren. Beispielsweise haben Mikolajczyk und Schmid (2005) zwar für den Deskriptor eine größere, und damit eindeutigere Umgebung als Basis gewählt, reduzieren die eigentliche Größe des Deskriptors jedoch auf 64 Einträge. Bay u. a. (2006) nutzen nicht die Gradienten, sondern *Haar-Wavelet-Responses* der Nachbarschaft, um einen SIFT ähnlichen Deskriptor mit 64 Einträgen zu erstellen. Genau wie Grabner u. a. (2006) nutzen sie *Integralbilder* als Grundlage.

Grabner u.a. ignorieren in ihrem Ansatz im Vergleich zum SIFT Deskriptor von Lowe die Gewichtung in der Nachbarschaft. Auch werden hier für die Rotationsinvarianz nicht die einzelnen Punkte rotiert, sondern die Histogramme. Statt *Orientierungshistogrammen* verwenden sie *Integralhistogramme*, die auf separaten *Integralbildern* der Regionen beruhen. Analog steigt der Vorteil gegenüber dem ursprünglichen Ansatz von Lowe mit der Anzahl der berechneten Deskriptoren.

SIFT auf der GPU (*SiftGPU*) gibt neben der Orientierung, Skalierung und Position eines Merkmals den Lowe SIFT Deskriptor aus, nutzt dabei aber intern die effiziente Datenstruktur der sogenannten Histogrammpyramide, um durch diese die Bild- bzw. Merkmalsinformation in eine Liste zu komprimieren. Dieser Ansatz von Ziegler u. a. (2006) verspricht u.a. die Analyse von zweidimensionalen Bild- und dreidimensionalen Volumendaten auf der GPU effizient zu gestalten. Ziegler u.a. betonen den Vorteil einer Implementation allein auf der GPU, *SiftGPU* verwendet jedoch eine hybride GPU / CPU Implementation.

Erster Schritt der Komprimierung ist die Konvertierung in ein Binärbild. Mit Hilfe eines Diskriminators wird dabei entschieden, ob ein Pixel zur Beschreibung der Liste beiträgt oder wegfällt. Das Binärbild bildet die Grundlage der *Histogrammpyramide*, welche ähnlich der Pyramide des SIFT Operators aufgebaut ist. Im Folgenden wird die Summe von je vier Pixeln, in diesem Kontext auch Zellen genannt, gebildet, bis die Spitze der Pyramide, bestehend aus einer Zelle, erreicht ist. Die Struktur der Histogrammpyramide ähnelt damit einem *Quadtree*.

Die Punktliste beinhaltet die Koordinaten der im Binärbild mit 1 markierten Pixel. Diese Koordinaten werden auf Basis der *Histogrammpyramide* gebildet, indem diese von Ebene zu Ebene traversiert wird. In *SiftGPU* wird diese Technik genutzt, um den Texturzugriff zu reduzieren, indem nur die für die Merkmale relevanten Informationen übertragen werden.

3.3 Matching und Berechnung der Pose

Matching beschreibt den Vergleich der gefundenen SIFT Merkmale und die adaptive Anpassung von Rotation und Translation der Kamera, sowie des durchschnittlichen Abstands zwischen korrespondieren Merkmalen. Für den Vergleich werden, wie in Kapitel 3.2.1 beschrieben, mehrere Ansichten generiert, um durch Auswahl der ähnlichsten Ansicht die virtuelle Kamerapose dem Kamerabild anzugleichen. Ausgabe von *SiftGPU* ist eine Menge von korrespondierenden Merkmalen, welche jedoch noch fehlerhafte Korrespondenzen enthalten kann und entsprechend gefiltert werden muss. Diese Menge beschreibt die ähnlichsten Deskriptoren, ohne dabei die Position im Bild zu berücksichtigen. Fehler können durch gleichartige Elemente im Bild, beispielsweise mehrere gleiche Fenster bei Gebäuden, entstehen. Das in diesem Trackingsystem verwendete Matching nutzt also neben dem Vergleich der Deskriptoren auch zusätzlich die Position der Merkmale in beiden Bildern.

Die Implementation von *SiftGPU* nutzt eine *Brute-Force-Methode* (*exhaustive search*), um die Korrespondenz von Merkmalen zwischen zwei Deskriptormengen festzustellen. Dieser Aufwand wird möglich durch die oben beschriebene, effiziente Datenstruktur des Deskriptors und die Umsetzung auf der GPU. Die Grundidee ist, dass jeder Deskriptor der einen Menge mit jedem der anderen Menge multipliziert wird, um so die beste Merkmalskorrespondenz zu bestimmen.

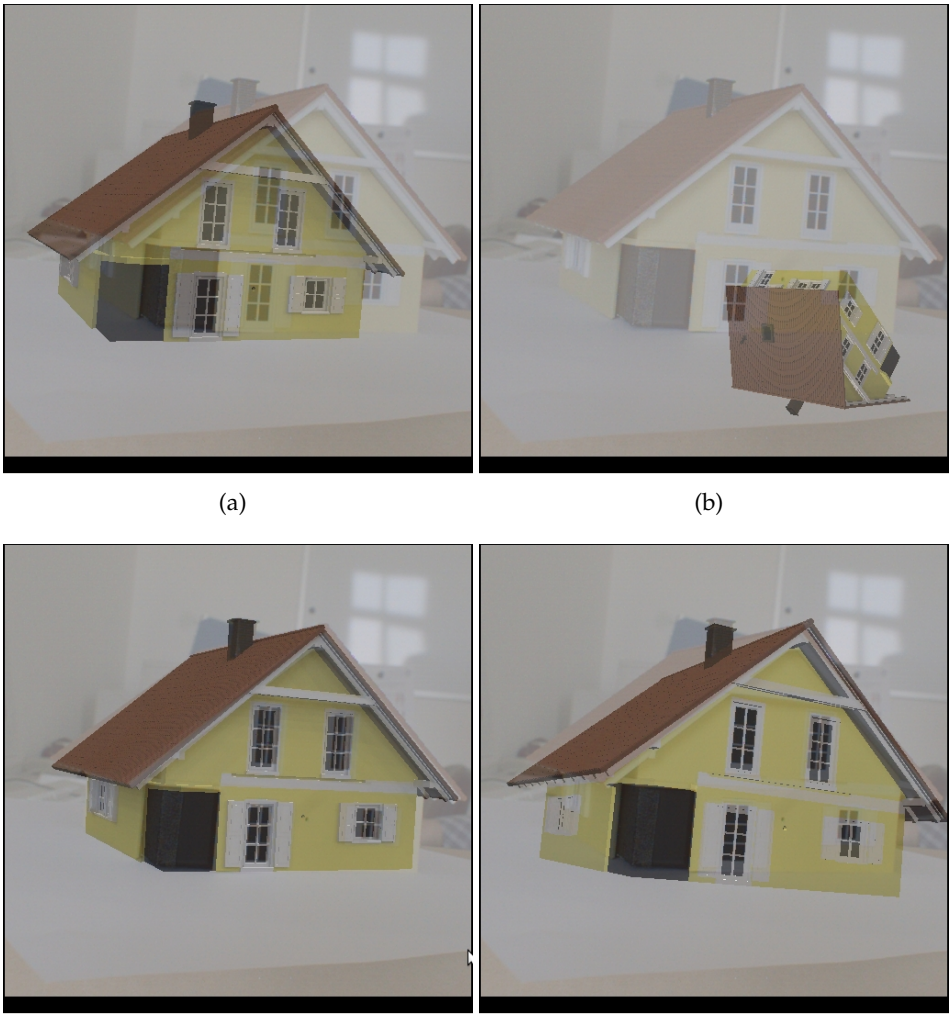
Lowe (2004) schlägt Verbesserungen vor, um die Anzahl der Vergleiche der Deskriptoren einzuschränken. U.a. wird ein Algorithmus vorgeschlagen, der mögliche, korrespondierende Merkmale nach ihrem Abstand priorisiert (*Best-Bin-First*). Nachdem eine bestimmte Anzahl von Korrespondenzkandidaten getestet ist, wird der Algorithmus abgebrochen und das wahrscheinlich korrespondierende Merkmal ausgegeben. Lowe beschreibt den Vorteil dieser Suche bei einer großen Datenbank von 100000 Merkmalen, während im hier entwickelten Trackingsystem durchschnittlich 200 Merkmale gefunden werden und getestet werden müssen.

Das Matching muss in unterschiedlichen Situationen funkionieren, das heißt, es muss zu jeder Zeit eine bestimmte Anzahl von korrespondierenden Merkmalen gefunden werden. Wird das Trackingsystem gestartet, unterscheiden sich synthetisches Bild, also die Position der virtuellen Kamera, und das Kamerabild stark. Auch bei schneller Bewegung der Kamera unterscheiden sich beide Bilder stark voneinander. Um auf der anderen Seite ein gutes Matching noch weiter zu verbessern, muss der Suchraum, der durch die Generierung der Ansichten entsteht, entsprechend verkleinert werden. Die Lösung ist eine adaptive Anpassung der Parameter Translation und Rotation der virtuellen Kamera, sowie des maximal zulässigen,

durchschnittlichen Abstands der Positionen der Merkmale, kurz des Suchfensters um ein Merkmal.

Das Trackingsystem wird mit hohen Werten für Rotation und Translation und einem großen Suchfenster initialisiert, um die Pose der Kamera möglichst schnell anzunähern. Die initiale Ansicht durch die virtuelle Kamera muss hier jedoch vorab dem Kamerabild angenähert werden, um den Fehler und die Zeit der Korrektur des initialen Trackings gering zu halten. Das System arbeitet mit einem experimentell bestimmten Schwellwert der Ähnlichkeit, der Anzahl der Merkmalskorrespondenzen, um zu bestimmen, wann die Parameter adaptiv angepasst werden müssen. Liegt die Ähnlichkeit über dem Schwellwert, wird die nächst kleinere Parameterstufe gewählt. Umgekehrt werden die Parameter erhöht, wenn die Ähnlichkeit unter einem Schwellwert liegt. Ändert sich die Ähnlichkeit über einen gewissen Zeitraum nicht, deutet dies auf ein lokales oder das globale Maximum hin. Rotation, Translation und das Suchfenster werden erhöht bzw. vergrößert, um das globale Maximum zu bestätigen oder um das lokale Maximum zu widerlegen. Die Annahme, dass sich die Kamera langsam bewegt, wird durch diese adaptiv angepasste Optimierung unterstützt.

Abbildung 10 zeigt unterschiedliche Stadien der Poseberechnungen durch dieses System.



(a) (b)

(c) (d)

Abbildung 10: Vergleich Poseberechnung: Gute Optimierung (a), schlechte Optimierung (b), mögliches globales Maximum (c), mögliches lokales Maximum (Merkmale an der Vorderseite des Gebäudes) (d)

4 Computergraphik

Das synthetische Bild im Ansatz *Analyse durch Synthese* soll in diesem Tracking System möglichst photorealistisch gerendert werden. Dazu wird das direkte Licht simuliert, welches durch *Importance Sampling* des Bilds der HDR-Fischaugenkamera aufgenommen wird, um ein als ideal diffus angenommenes Modell durch mehrere gerichtete Lichtquellen (*Directional Lights*) zu beleuchten (vgl. *Image based lighting*). Dieser photorealistische Ansatz auf Basis des Importance Samplings wird durch realistischen Schattenwurf jeder gerichteten Lichtquelle unterstützt. Da sich der Vergleich der Bilder auf das zu trackende Modell beschränkt, wird nur die Selbstverschattung berechnet.

4.1 Direkte Beleuchtung

Direkte Beleuchtung beschreibt den Lichtweg von einer oder mehreren Lichtquellen zum Objekt. Im Gegensatz dazu stammt das indirekte Licht von Materialien, die direktes Licht reflektieren. Hier wird davon ausgegangen, dass das Licht von gerichteten Lichtquellen stammt, die unendlich weit entfernt sind und deren Lichtstrahlen parallel zueinander auf das Objekt treffen. Die Berechnung solch einer Beleuchtung ist besonders einfach und kann effizient umgesetzt werden.

Es wird angenommen, dass das ideal diffus reflektierende Material M_d in diesem Trackingsystem von diffusem Licht L_d bestrahlt wird, die Leuchtdichte ist unabhängig vom Betrachtungswinkel. Das Licht I_o ist abhängig von dem Kosinus des Winkels θ zwischen dem normalisierten Richtungsvektor der gerichteten Lichtquelle und der normalisierten Oberflächennormalen des Materials:

$$I_o = L_d M_d \cos\theta \tag{18}$$

Ziel ist, das Integral der direkten Beleuchtung L_i zu lösen, um eine photorealistische Beleuchtung zu simulieren:

$$L_i = \frac{\rho}{\pi} \int_{2\pi} L_i(\omega_i) \cos(\theta_i) d\omega_i \tag{19}$$

Integrale lassen sich durch die *Monte Carlo Integration* schätzen. Das Prinzip ist, das Integral durch den Erwartungswert einer Zufallsvariablen darzustellen und den Erwartungswert durch Samples abzuschätzen. Diese N Samples können mit einer beliebigen Dichte $p(x)$ gebildet werden, solange $p(x)$ die Eigenschaften einer Dichte besitzt. Sampling wird im nächsten

Kapitel 4.2 näher beschrieben. Die Approximation der direkten Beleuchtung L_i lässt sich durch *Monte Carlo Integration* bilden:

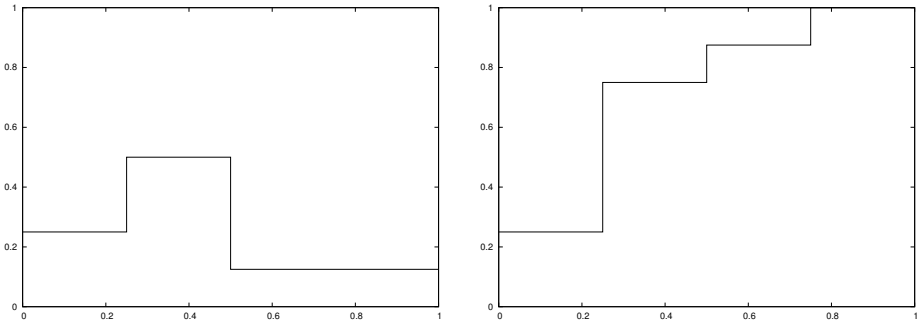
$$L_i \approx \frac{\rho}{\pi N} \sum_{i=1}^{N} \frac{L_i(\omega_i)\cos(\theta_i)}{p(x)} \tag{20}$$

Grundlage der direkten Beleuchtung dieses Trackingsystems ist das Bild der HDR-Videokamera mit Fischaugenobjektiv, aus dem Richtung und Farbe der gerichteten Lichtquellen bestimmt werden. Das Bild dieser Kamera hat eine Farbtiefe von 12 Bit, weshalb eine photorealistische Lichtsimulation durchgeführt werden kann, wenn die Materialeigenschaften des zu beleuchtenden Materials bekannt sind.

4.2 Sampling

Die Dichtefunktion der *Monte Carlo Integration* bestimmt, an welcher Stelle Samples erzeugt werden. Im Allgemeinen ist die Wahl der Dichtefunktion beliebig, jedoch kann sie im Rahmen der Beleuchtung durch das Fischaugenkamerabild die Qualität und Zeit der Berechnung von beleuchteten Szenen beeinflussen. Wenn direkte Lichtquellen sehr klein sind, aber dennoch die Beleuchtung der Szene bestimmen, ist es bei einer beliebigen Dichtefunktion unwahrscheinlich, dass genau diese Lichtquellen getroffen und Samples erzeugt werden. Importance Sampling beschreibt eine Dichtefunktion, die an wichtigen Stellen Samples erzeugt. Dazu muss sie der Form der Funktion ähneln, deren Integral berechnet werden soll.

Die Methode der inversen Verteilungsfunktion (*Inverse CDF-Methode*) für den 2D Fall wird im folgenden Kapitel beschrieben. Durch sie kann die Zufallsvariable mit der gewünschten Dichte durch Eingabe von gleichverteilten Zufallszahlen bestimmt werden. Ausserdem wird im diskreten Fall statt der kontinuierlichen Dichtefunktion eine Wahrscheinlichkeitsfunktion verwendet, deren Verteilungsfunktion dementsprechend auch diskret ist (Abbildung 11).



(a) (b)

Abbildung 11: Eine (diskrete) Wahrscheinlichkeitsdichtefunktion (a) und Verteilungsfunktion (b)

4.2.1 Importance Sampling

Die Idee von *Importance Sampling* ist, die Dichte für die *Monte Carlo Integration* der Beleuchtung so zu wählen, dass Samples aus den wichtigsten Lichtquellen im Bild genommen werden können. Es bietet sich an, die zweidimensionale Dichtefunktion so zu wählen, dass sie dem ankommenden Licht entspricht. Die Wahrscheinlichkeit eine Richtung auszuwählen, soll von der Helligkeit, der Leuchtdichte, abhängig sein.

Das Bild der Kamera mit Fischaugenobjektiv wird zunächst in das *Latitude Longitude Format* (Abbildung 12) reparametrisiert, um als Grundlage einer zweidimensionalen Dichte zu entsprechen. Vorab muss dazu das Zentrum und der Radius des Bildes der Kamera mit Fischaugenobjektiv bestimmt werden. Danach werden die Richtungen der Hemisphäre über (θ, ϕ) parametrisiert für $\theta \in [0, \frac{\pi}{2}]$ und $\phi \in [0, 2\pi]$ p.

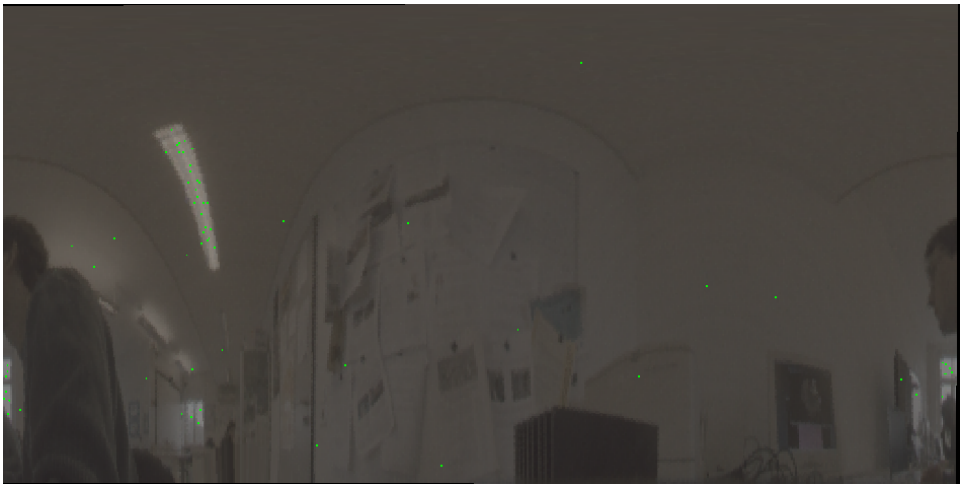


Abbildung 12: Bild der HDR Kamera mit Fischaugenobjektiv im Latitude Longitude Format. Durch Importance Sampling gefundene Samples grün markiert

Die Dichtefunktion entspricht der aus dem RGB Bild rekonstruierten Leuchtdichte. Dazu wird jedes Pixel mit y = (0.2127, 0.7152, 0.0722) multipliziert. Der daraus resultierende, skalare Wert der Leuchtdichte wird zusätzlich mit $\sin(\theta)$ gewichtet, um die Dichtefunktion zu gewichten, so dass mehr Samples um den Pol der Hemisphäre erzeugt werden.

4.2.2 Inverse CDF Methode

Die *Inverse CDF Methode* ähnelt sich grundsätzlich für den 1D und 2D Fall. Während für den 1D Fall aus einer Zufallszahl ein 1D Sample erzeugt wird, wird im 2D Fall aus zwei Zufallszahlen ein 2D Sample erzeugt. Ausserdem wird im diskreten Fall statt der durch eine Dichtefunktion stetig verteilten Zufallsvariable eine Wahrscheinlichkeitsfunktion genutzt, um die Wahrscheinlichkeit, ein Sample einer bestimmten Pixelbreite x und -höhe x in einem zweidimensionalen Bild zu berechnen, zu beschreiben. Die aus der Dichte- oder Wahrscheinlichkeitsfunktion resultierende Verteilungsfunktion beschreibt die Wahrscheinlichkeit, dass eine Zufallszahl in einem bestimmten Intervall bzw. Bereich liegt. Die inverse Verteilungsfunktion lie-

fert also durch Eingabe von $(0, 1)$-verteilten Zufallszahlen eine Zufallsvariable der gewünschten Dichte- bzw. Wahrscheinlichkeitsfunktion und Verteilung.

Zunächst soll der allgemeine 1D Fall einer kontinuierlichen Dichtefunktion erläutert werden, um den Algorithmus danach auf den diskreten 2D Fall zu erweitern. Zuerst muss die Funktion $f(x)$ normalisiert werden, um einer Dichtefunktion $p(x)$ zu entsprechen. Dazu muss das Integral über diese Funktion, also die Fläche unter der Kurve, genau 1 ergeben:

$$p(x) = cf(x) \tag{21}$$

$$\Rightarrow \quad \int_0^1 p(x) = \int_0^1 cf(x) = 1 \tag{22}$$

$$\Rightarrow \quad c\left[F(x)\right]_0^1 = 1 \tag{23}$$

$$\Rightarrow \quad \frac{1}{F(1) - F(0)} = c \tag{24}$$

Die Verteilungsfunktion kumuliert die Wahrscheinlichkeiten für diskrete Zufallsvariablen durch Summenbildung und integriert die Dichte für stetig verteilte Zufallsvariablen. Die Verteilungsfunktion $F(x)$ wird aus der normalisierten Dichtefunktion bzw. Wahrscheinlichkeitsfunktion gebildet:

$$F(x) = \int_0^1 p(x)dx \tag{25}$$

Die inverse Verteilungsfunktion $F^{-1}(x)$ bildet nun eine $(0, 1)$-verteilte Zufallszahl in die gewünschte Dichte $p(x)$ ab.

Die Anzahl der Samples, die pro Frame erzeugt werden können, ist beschränkt. Durch die hier verwendete Implementation der Beleuchtung und Schatten, sowie der verwendeten Graphikhardware, sind maximal 64 Lichtquellen möglich, weshalb die wichtigen Lichtquellen im Kamerabild unbedingt durch diese Samples abgedeckt werden müssen. Durch das *Importance Sampling* werden dort Samples erzeugt, wo eine hohe Leuchtdichte besteht. Die Problematik liegt in der Generierung der Zufallszahlen. Abgesehen davon, dass populäre, eingebaute Zufallszahlengeneratoren wie *rand* und *srand* nicht unbedingt zufällig Zahlen erzeugen, tritt selbst bei besseren Generatoren das Problem auf, dass die durch das *Importance Sampling* erzeugten 64 Samples von Frame zu Frame ihre Position verändern. Durch die geringe Anzahl Samples und ihre Positionsveränderung von Frame zu Frame entsteht beim Rendering der Beleuchtung und Schatten ein deutlich wahrnehmbares Flackern.

Eine *Halton Sequenz* beschreibt eine Sequenz von Zahlen, welche Eigenschaften mit echten (Pseudo-)Zufallszahlen teilen, allerdings nicht zufällig sind. Der Vorteil solch einer Sequenz ist, dass sich die Position der Samples wenig verändert und so das Flackern der wechselnden Lichtquellen von Frame zu Frame verringert wird. Die *Halton Sequenz* deckt dabei den Samplingbereich optimal ab, indem zur Berechnung Primzahlen als Basis genommen werden. Diese Primzahlen sollten dabei idealerweise möglichst klein gewählt werden.

Folgender Code berechnet die 1D Halton Sequenz mit Basis $b_1 = 2$. Das Ergebnis ist in diesem Fall $\frac{1}{2}, \frac{1}{4}, \frac{3}{4}, \frac{1}{8}, \frac{5}{8}, \frac{3}{8}, \frac{7}{8}, \frac{1}{16}, \frac{9}{16}, ...$ Dabei wird die i-te Nummer der Sequenz erzeugt:

```
for(int i = 1; i < num_points_+1; i++)
{
    int digit = 0;
    float half = 0.5f;
    float out = 0.0f;
    float in = (float)i;

    while(in != 0.0f)
    {
        digit = (int)in % base_1_;
        out += digit * half;
        in = (in - digit) * 0.5;
        half = half * 0.5;
    }
}
```

Listing 2: *1D Halton Sequenz mit Basis 2*

Die *2D Halton Sequenz* für das *2D Latitude Longitude Bild* der Kamera mit Fischaugenobjektiv hat die Basen $b_1 = 2$ und $b_2 = 3$, welche optimal über den Samplingbereich verteilt sind, wenn N zweidimensionale Samples erzeugt werden mit $N = b_1^{k_1} b_2^{k_2}$, wobei k_1 und k_2 natürliche Zahlen sind. In diesem Trackingsystem wird eine feste Anzahl von 64 Lichtquellen bzw. zweidimensionale Samples genutzt, wodurch der Samplingbereich nicht optimal abgedeckt wird. Da die Anzahl der Lichtquellen konstant ist, wird die *Halton Sequenz* nur zur Initialisierung des Trackingsystems einmalig berechnet.

Die *Inverse CDF Methode* lässt sich zusammenfassen:

1. $f(x)$ normalisieren: $p(x) = cf(x)$
2. Verteilungsfunktion $F(x)$ bestimmen
3. Inverse Verteilungsfunktion $F^{-1}(x)$ bestimmen

4. Durch $(0,1)$-verteilte Zufallszahlen ξ_i die Zufallsvariable X_i mit gewünschter Dichte bestimmen: $X_i = F^{-1}(\xi_i)$

Statt einer kontinuierlichen Dichtefunktion soll die Wahrscheinlichkeitsfunktion des diskreten Leuchtdichtebildes der Kamera mit Fischaugenobjektiv erstellt werden. Die Pixel bilden dabei eine stückweise konstante Funktion $p(x)$. $p(x)$ ist durch N Samples v_n über $x \in [0,1]$ definiert als:

$$p(x) = \begin{cases} v_0 & \text{falls } x_0 < x < x_1 \\ v_1 & \text{falls } x_1 < x < x_2 \\ ... & ... \end{cases} \tag{26}$$

Das Integral von $p(x)$ ist gleich der Summe über die Samples v_n geteilt durch die Anzahl Samples N:

$$\int_0^1 p(x)dx = \sum_{i=0}^{N-1} \frac{v_i}{N} = F(x) \tag{27}$$

Die entsprechende kumulative Verteilungsfunktion $F(x)$ ist stückweise linear. Die Wahrscheinlichkeit P, dass Samples zwischen i und j generiert werden, beträgt:

$$P(i < X \leq j) = F(x_j) - F(x_i) \tag{28}$$

Für das Sampling aus dem zweidimensionalen, diskreten Bild der Leuchtdichten wird dessen Funktion $f(x)$ normalisiert, um der Dichtefunktion $p(x)$ zu entsprechen. In diesem Fall muss das Integral über die Dichtefunktion, das Volumen unter der Kurve, 1 ergeben.

$$p(x,y) = \frac{f(x,y)}{\int\int f(x,y)dxdy} \tag{29}$$

$$= \frac{f(x,y)}{\sum\sum f(x,y)} \tag{30}$$

Die normalisierte Funktion $p(x,y)$ wird in eine bedingte Wahrscheinlichkeit $p(y|x)$ und eine Randverteilung $p(x)$ aufgeteilt. Im zweidimensionalen Fall werden zwei $(0,1)$-verteilte Zufallszahlen ξ_1 und ξ_2 generiert, wobei ξ_1 zuerst in die Dichtefunktion, in die Randverteilung, $p(x)$ abgebildet wird. ξ_2 wird anschliessend durch die bedingte Wahrscheinlichkeit $p(y|x)$ in die zweidimensionale Dichte $p(x,y)$ abgebildet.

$$p(x, y) = p(y|x)p(x) \tag{31}$$

Die Randverteilung $p(x)$ vernachlässigt einen Teil der Zufallsvariablen von $p(x, y)$, indem nur das Integral über y gebildet wird:

$$p(x) = \int_y p(x, y)dy \tag{32}$$

$$= \frac{\sum_y f(x, y)}{\sum\sum f(x, y)} \tag{33}$$

Die bedingte Wahrscheinlichkeit $p(y|x)$ lässt sich aus den bekannten $p(x, y)$ und $p(x)$ berechnen:

$$p(y|x) = \frac{p(x, y)}{p(x)} \tag{34}$$

$$= \frac{\frac{f(x,y)}{\sum\sum f(x,y)}}{p(x)} \tag{35}$$

4.3 Schatten

Bei direkter Beleuchtung werden durch starke Lichtquellen Schatten erzeugt, deren Schattenkanten unter Umständen auch natürliche Merkmale beschreiben können und deshalb ebenso wichtig wie die Merkmale der Geometrie sind. Für das Matching wird nur die Selbstverschattung des virtuellen Objekts berücksichtigt.

In dieser Implementation wird das *Shadow Mapping* Verfahren genutzt, um für jede der 64 Lichtquellen den Schattenwurf zu berechnen. Durch Überblenden der einzelnen Schatten entstehen bei korrektem Sampling realistische, weiche Schatten. *Shadow Mapping* lässt sich effizient durch Shader implementieren. Ausserdem gibt es weitere Verbesserungen unter anderem durch zusammenfassen der *Shadow Maps* in einer großen Tiefentextur (*Textur Atlas*), um Schreib- und Lesezugriff zu optimieren. Durch *Shadow Mapping* kann beliebig komplexe Geometrie effizient verschattet werden, da das Verfahren bildbasiert arbeitet. Abbildung 13 zeigt die Beleuchtung und Schatten eines Objekts durch 4 Lichtquellen.

Nachteile sind *Aliasing-Artefakte* an den Schattenrändern bei geringer Shadow Map Texturauflösung und falsche Selbstverschattung (*Z-Fighting*). Diese Probleme können durch eine höhere Texturauflösung gemindert werden.

4.3.1 Shadow Mapping

Um eine *Shadow Map* zu erzeugen, wird die Szene aus Sicht der Lichtquelle gerendert. Da eine gerichtete Lichtquelle unendlich weit entfernt ist, kann die Szene orthographisch projeziert werden. Anschliessend wird der *Z-Buffer* aus Sicht der Lichtquelle in einer Tiefentextur, der sogenannten *Shadow Map*, gespeichert. Jedes Fragment muss nun in das Koordinatensystem der Lichtquelle transformiert werden, um dessen Tiefe gegen die Tiefentextur zu testen. Ist die Tiefe größer als der Wert der Tiefentextur, ist also das Fragment weiter entfernt, liegt es im Schatten. Sind beide Tiefenwerte gleich, wird das Fragment logischerweise beleuchtet. Abbildung 14.

Bounding Sphere Um die Auflösung der Tiefentextur effizient zu nutzen, muss das Objekt sie möglichst ausfüllen. Dazu wird diejenige Kugel (*Bounding Sphere*) gesucht, die alle *Vertices* optimal umfasst. Der Ursprung der Objektkoordinaten soll dabei im Zentrum der Bounding Sphere liegen, so dass durch Angabe eines Radius' die Parameter der *orthographischen Projektion* in *OpenGL* durch *glOrtho* leicht bestimmbar sind. Diese Vereinfachung garantiert nicht die minimale *Bounding Sphere*, sie genügt jedoch, um die *Shadow Map* Textur besser auszunutzen. Abbildung 15 zeigt die durch eine *Bounding Sphere* definierte *orthographischen Projektion* des Objekts.

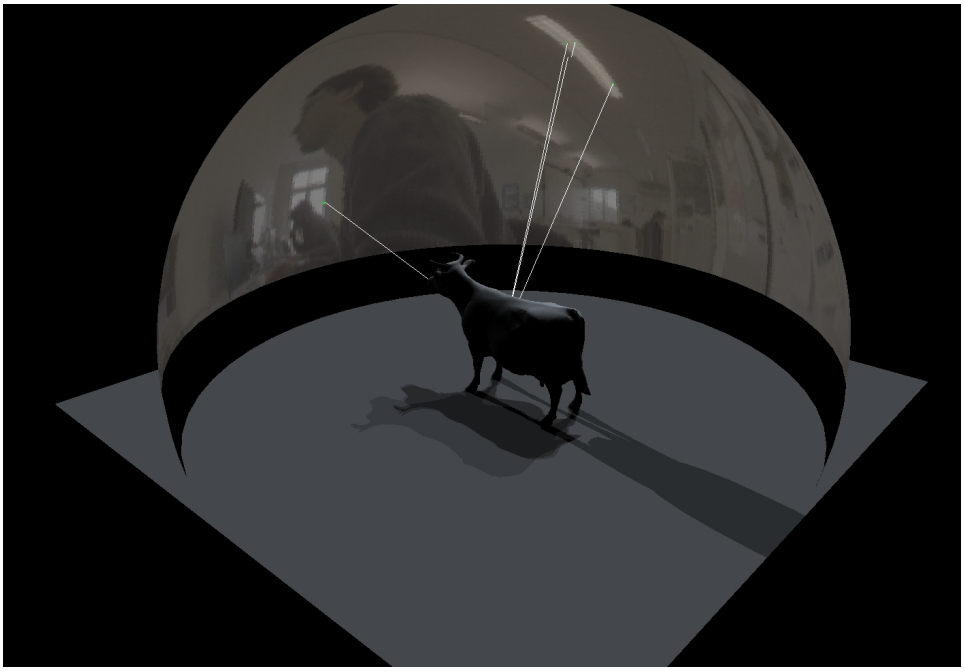


Abbildung 13: Beleuchtung und Schatten des Modells durch 4 Lichtquellen. Importance Sampling der über dem Objekt liegenden Hemisphäre

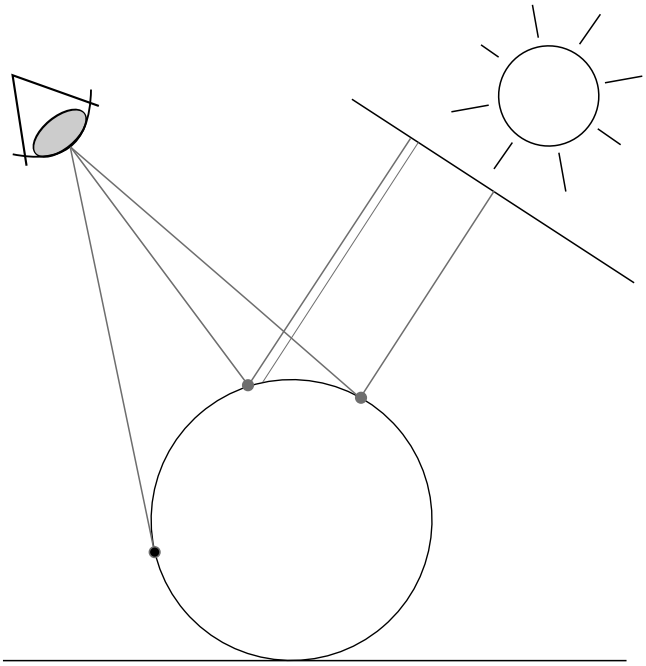


Abbildung 14: Shadow Mapping

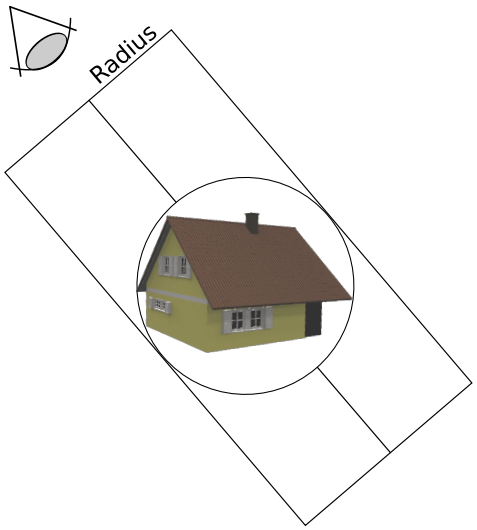


Abbildung 15: Durch eine Bounding Sphere verbesserte orthographische Projektion

Die *Bounding Sphere* wird hier vorab einmalig bestimmt, indem sie ausgehend von einem Startvertex wächst, wenn ein Vertex ausserhalb der *Bounding Sphere* gefunden wird. Der Algorithmus zur inkrementellen Generierung einer *Bounding Sphere* ist:

```
Initialisierung: Startvertex und Radius ist 0

Solange Vertices vorhanden sind
{
    Falls der Abstand zwischen Startvertex und Vertex kleiner
        gleich Radius ist, ignoriere Vertex
    Andernfalls lasse Bounding Sphere wachsen, neuer Radius ist
        Abstand zwischen Startvertex und Vertex
}
```

Listing 3: *Pseudo Code: Bounding Sphere*

Entscheidend ist hier die Reihenfolge der *Vertices*. Um die bestmögliche *Bounding Sphere* zu finden, wird der Algorithmus mehrmals durchlaufen, wobei die Reihenfolge der *Vertices* für jeden Durchlauf zufällig gemischt wird. Falls der neu berechnete Radius kleiner als der alte Radius ist oder dies der erste Durchlauf ist, wird der neue Radius als bestes Ergebnis gespeichert. Der Algorithmus produziert schon nach wenigen Durchläufen einen guten Radius der *Bounding Sphere* mit Zentrum im Ursprung des Objektkoordinatensystems.

Shadow Map Atlas Das Sampling liefert die Positionen bzw. Richtungen und Farben der gerichteten Lichtquellen. Ziel ist, die daraus entstehenden Shadow Maps in einer großen Textur (*Textur Atlas*) zu speichern. Dazu wird ein *Frame Buffer Object* (*FBO*) erstellt, das den quadratischen *Textur Atlas* repräsentieren soll. Über eine Schleife werden nun alle Ansichten der Lichtquellen gerendert und in das *Frame Buffer Objekt* eingetragen. Die so entstehende Tiefentextur bildet die Grundlage für den Vergleich mit dem Rendering aus Sicht der virtuellen Kamera, um den Schattenwurf zu bestimmen.

Der *Viewport* entspricht genau einer *Shadow Map*, der quadratische *Textur Atlas* wird also optimal ausgenutzt, wenn 4^n Lichtquellen vorhanden sind. In diesem System werden beispielsweise 64 Lichtquellen genutzt. Da der Radius der Bounding Sphere vorab berechnet wurde, kann nun die *orthographische Projection Matrix* durch *glOrtho(-radius, radius, -radius, radius, near, far)* erzeugt werden. Für die *Modelview Matrix* wird die virtuelle Kamera auf die Richtung bzw. die Position der Lichtquelle gesetzt.

Shadow Mapping und Beleuchtung Schatten und Beleuchtung werden in einem Schritt durch einen Vertex und Fragment Shader erzeugt. Im Vertex Shader werden zunächst die ursprüngliche Vertex Position, die normalisierte Vertex Normale, die Materialfarbe berechnet und an den Fragment Shader übergeben. Die transformierte Vertex Position wird ebenfalls berechnet.

Eingaben des Fragment Shaders sind somit der Shadow Map Atlas, die Arrays der Richtungen und Farben der Lichtquellen, die Anzahl der Lichtquellen und die Projection Matrix des Lichts P, welche für alle Lichtquellen gleich ist. Um den Vergleich der Tiefenwerte im Rahmen des *Shadow Mappings* machen zu können, muss die Vertex Position in das Koordinatensystem der Lichtquelle transformiert werden. Die ausserdem zur Transformation benötigte *Modelview Matrix* kann aus dem Richtungsvektor der Lichtquelle berechnet werden, indem die Funktionalität von *gluLookAt()* nachgebildet wird. Zusätzlich werden Richtungen und Farben der Lichtquellen zur Beleuchtung im selben Shader genutzt. Eine weitere Matrix V für den Viewport transformiert das Koordinatensystem von $[-1, 1]$ nach $[0, 1]$.

Aus den Vektoren der Richtung der Lichtquelle $f = (-x, -y, -z)^T$ und $up = (0, 1, 0)^T$ wird die *Modelview Matrix* MV gebildet:

$$f = \frac{f}{||f||} \tag{36}$$

$$s = f \times up \tag{37}$$

$$u = \frac{s}{||s||} \times f \tag{38}$$

$$MV = \begin{pmatrix} s_0 & s_1 & s_2 & 0 \\ u_0 & u_1 & u_2 & 0 \\ -f_0 & -f_1 & -f_2 & -1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{39}$$

Der Vertex im Lichtkoordinatensystem ist demnach:

$$v_{Licht} = V * P * MV * v \tag{40}$$

Nun wird der Tiefenwert des transformierten Vertex mit dem der im *Textur Atlas* gespeicherten Shadow Map verglichen. Ist der Tiefenwert der *Shadow Map* kleiner als der Tiefenwert des transformierten Vertex, wird das Fragment verschattet. Andernfalls wird die Beleuchtung durch die gerichtete Lichtquelle ausgegeben. Eine Schleife läuft über alle Lichtquellen, um die Beleuchtung, Schatten und Farbe für jedes Fragment zu akkumulieren. Die Ausgabe wird anschliessend normalisiert, indem durch die Anzahl der Lichtquellen geteilt wird. Abschliessend wird sie noch mit der Materialfarbe gewichtet.

```
varying vec3 normal;
varying vec3 material;
varying vec4 vertex;

void main(void)
{
    vertex =  gl_Vertex;
    normal = normalize(gl_Normal);
    material =  gl_Color;
    gl_Position = ftransform();
};
```

Listing 4: *Vertex Shader: Beleuchtung Pass*

```
#define NUMPOINTS 64

uniform sampler2D texshadow;
uniform vec3 lightdirs[NUMPOINTS];
uniform vec3 lightcolors[NUMPOINTS];
uniform mat4 proj;
```

```
varying vec3 normal;
varying vec3 material;
varying vec4 vertex;

void main(void)
{
    vec3 lightsum;
    vec3 lightdir;
    vec3 lightcolor;
    vec2 at;
    vec4 shadow_coord;

    at = vec2(0.0,0.0);
    lightsum = vec3(0.0, 0.0, 0.0);

    for(int i = 0; i < NUMPOINTS; i++)
    {
        lightdir = lightdirs[i];
        //normalize(lightdir);

        // glulookat
        vec3 f = normalize(vec3(-lightdir.x, -lightdir.y, -
            lightdir.z));
        vec3 up = vec3(0.0,1.0,0.0);
        vec3 s = cross(f,up);
        s = normalize(s);
        vec3 u = cross(s,f);

        // column-major
        mat4 mv = mat4(s[0],u[0],-f[0],0.0,s[1],u[1],-f[1],0.0,s
            [2],u[2],-f[2],0.0,0.0,0.0,-1.0,1.0);
        // [-1,1] to [0,1]
        mat4 bias = mat4(0.5, 0.0, 0.0, 0.0, 0.0, 0.5, 0.0, 0.0,
             0.0, 0.0, 0.5, 0.0, 0.5, 0.5, 0.5, 1.0);

        shadow_coord = bias * proj * mv * vertex;

        vec4 shadow_coord_normalized = shadow_coord /
            shadow_coord.w;

        vec2 atlas_coord = at + (8.0 * shadow_coord_normalized.
            st / float(NUMPOINTS));
        // numerator is sqrt(NUMPOINTS)
        float factor = 8.0 / float(NUMPOINTS);

        float depth_shadowmap = texture2D(texshadow, atlas_coord
            ).z;

        float shadow = 1.0;
        if(shadow_coord.w > 0.0 && (atlas_coord.x - at.x) <
            factor && (atlas_coord.y - at.y) < factor)
        {
            if(depth_shadowmap < shadow_coord_normalized.z -
                0.00003 && shadow_coord_normalized.s > 0.0 &&
```

```
                shadow_coord_normalized.t > 0.0)
            {
                shadow = 0.0;
            }
        }

        lightcolor = lightcolors[i];
        lightcolor *= shadow;
        lightcolor *= max(dot(normal, lightdir), 0.0);

        at.x += factor;
        if(at.x >= 1.0){at.x = 0.0; at.y += factor;}
        lightsum +=  4.0 * lightcolor / float(NUMPOINTS);
    }

    lightsum = material * lightsum;

    gl_FragColor.rgb = lightsum;
};
```

Listing 5: *Fragment Shader: Beleuchtung*

5 Ergebnisse

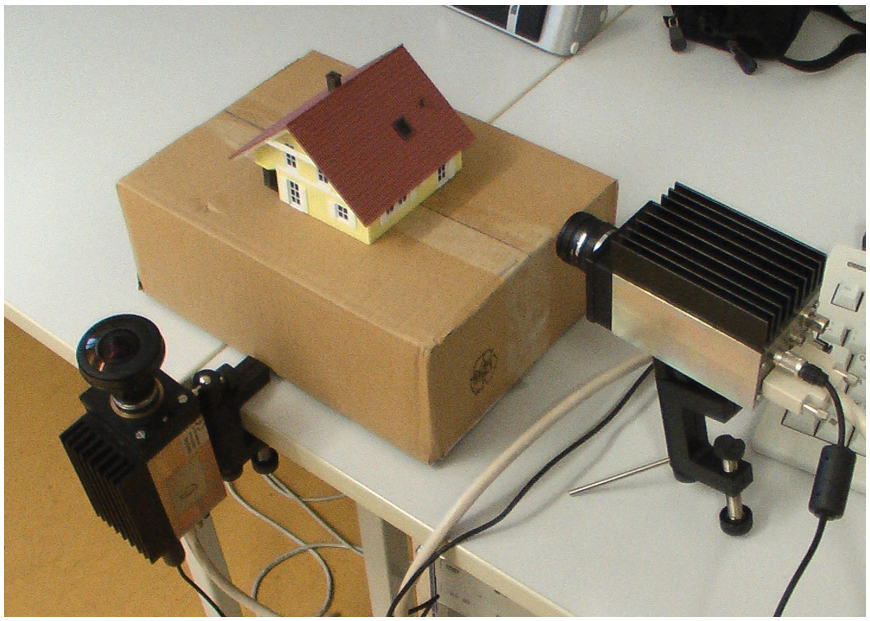


Abbildung 16: Versuchsaufbau

Im Folgenden werden bestimmte Eigenschaften des synthetischen Bilds und des Kamerabilds betrachtet, um ihre Ähnlichkeit näher zu beurteilen. Die Eigenschaften sind die Anzahl der gefundenen natürlichen SIFT-Merkmale und die Ähnlichkeit ihrer Deskriptoren mit und ohne fehlerhafte Korrespondenzen. Gültige Merkmalskorrespondenzen dürfen in beiden Bildern einen Abstand von 10 Pixeln nicht überschreiten. Idealerweise liegen bei korrekter Geometrie und Bildentstehung gültige Merkmalskorrespondenzen an der selben Position. Die Pose der virtuellen Kamera wird jedoch für den oben genannten Vergleich manuell so transformiert, dass beide Bilder ungefähr übereinstimmen. Die Toleranz im Abstand der Merkmale liegt unter anderem dem zugrunde.

Betrachtet werden jeweils 20 Frames zweier Referenzvideos, welche keine Transformation des Objekts zeigen. Stattdessen soll so die selbst in dieser kurzen Zeit wechselnde Beleuchtungssituation eingefangen werden, als auch das Bildrauschen der HDR Kamera relativiert werden. Das synthetische Bild beider Referenzvideos wird jeweils mit unterschiedlichen Parametern in den Bereichen Material, Schatten und Anzahl der durch das Sampling bestimmten Lichtquellen erzeugt, um die Auswirkungen dieser Parameter auf das Matching bzw. Tracking zu verdeutlichen. Die Videos haben eine Auflösung von 512 x 512 Pixeln, wobei das Kamerabild der HDR Videokameras mit einer Auflösung von 512 x 496 Pixeln vorliegt. Die

Referenzvideos, im Folgenden auch Szene 1 und 2 genannt, zeigen unterschiedliche Ansichten ähnlicher Beleuchtungssituationen: Aus Sicht eines zur direkten Beleuchtung beitragenden Fensters und aus der entgegengesetzten Ansicht.

Im Bereich Material wird neutral gefärbtes Material und an das reale Modell farblich angenähertes Material miteinander verglichen. Neutral gefärbtes Material beschreibt eine RGB Farbe mit gleichen Farbwerten. Die Materialfarbe wurde vorab manuell bestimmt und entspricht nicht dem tatsächlichen Reflexionsgrad des Materials. Diese Vereinfachung des photorealistischen Anspruchs soll im Kontext dieses Trackingsystems überprüft werden.

Im Bereich Schatten werden vollständig opake, schwarze Schatten, transparente Schatten und fehlende Schatten verglichen. Transparente Schatten entsprechen der stärkeren Betonung eines nicht genauer definierten, ambienten Lichts. Opake Schatten bezeichnet den Schatten einer Lichtquelle. Durch das Überblenden der Lichtsimulationen aller Lichtquellen innerhalb eines Frames entstehen jedoch auch Effekte wie weiche Schattenränder.

Der Vergleich mit unterschiedlicher Anzahl an Lichtquellen beschränkt sich auf Schritte von 4^n Lichtquellen, welche durch die Hardware und die Implementation limitiert sind.

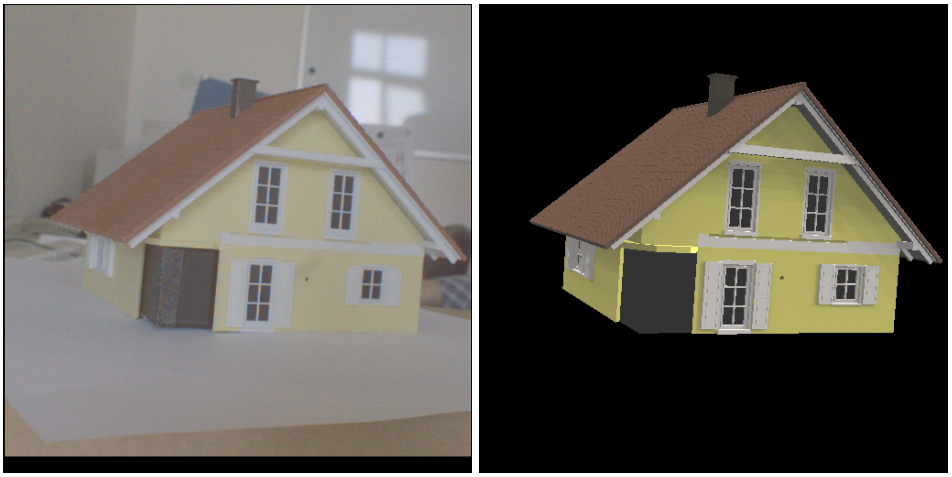
Das hier verwendete 3D-Modell eines Modellhauses wurde in *Blender* modelliert und mit Normalen und Materialfarbe als *OBJ* Datei und *MTL* Datei exportiert. Das Modell hat 94970 *Vertices* und 178672 Dreiecke. Details, wie beispielsweise die Dachziegel des Dachs, wurden ebenfalls mit Geometrie modelliert. Es wurden keine Texturen verwendet.

Abbildung 17 zeigt das synthetische Bild und das Kamerabild. Das synthetische Bild ist Resultat der Beleuchtungssimulation durch die HDR Videokamera mit Fischaugenobjektiv. Da der Vergleich nur im Bereich des Modells stattfinden soll, wird der Hintergrund schwarz gezeichnet. Merkmale werden jedoch über das gesamte Bild gesucht, die Ausmaskierung verhindert lediglich, dass dort Merkmalskorrespondenzen (*Matches*) gefunden werden.

Die Ausgabe des Kamerabilds ist durch *Tonemapping*, Gamma-Korrektur und Farbverstärkung verändert, sie dient nur der Darstellung und bildet den letzten Schritt dieses Trackingsystems. Die Lichtsimulation arbeitet im Wesentlichen mit den unveränderten Kamerabildern. Abbildung 18 zeigt die Überblendung von Kamerabild und synthetischem Bild.

In einer *AR* Umgebung würde der Benutzer nur das Kamerabild sehen, das synthetische Bild dient lediglich zur Berechnung der Pose der Kamera.

Da hier vor allem die Ähnlichkeit von SIFT Merkmalen und die Ähnlichkeit von Kamerabild und synthetischem Bild betrachtet werden soll, wird in dieser Arbeit meist das durch das 3D-Modell überlagerte Kamerabild gezeigt.



(a) (b)

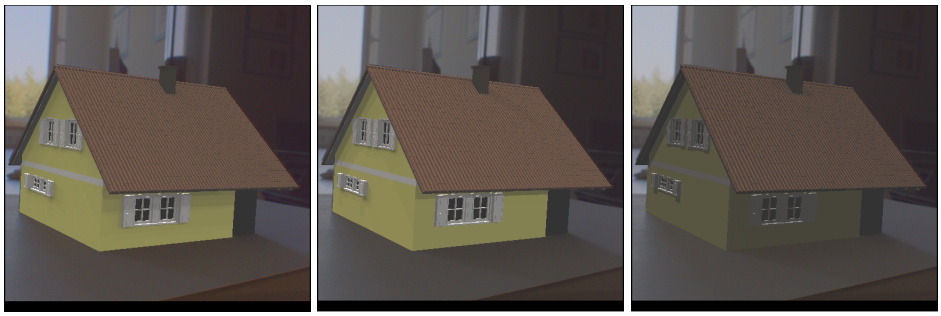
Abbildung 17: Ausgangslage für extitAnalyse durch Synthese: Kamerabild (a) und synthetisches Bild (b) in Szene 1

(a) (b) (c)

Abbildung 18: Darstellung: Opakes synthetisches Bild bzw. Objekt (a), Gleichmäßiges Blending (b) und Opakes Kamerabild (c) in Szene 2

(a) (b) (c)

Abbildung 19: Vergleich Anzahl Lichtquellen: 64 Lichtquellen (a), 16 Lichtquellen (b), 4 Lichtquellen (c) in Szene 1



(a) (b) (c)

Abbildung 20: Vergleich Anzahl Lichtquellen: 64 Lichtquellen (a), 16 Lichtquellen (b), 4 Lichtquellen (c) in Szene 2

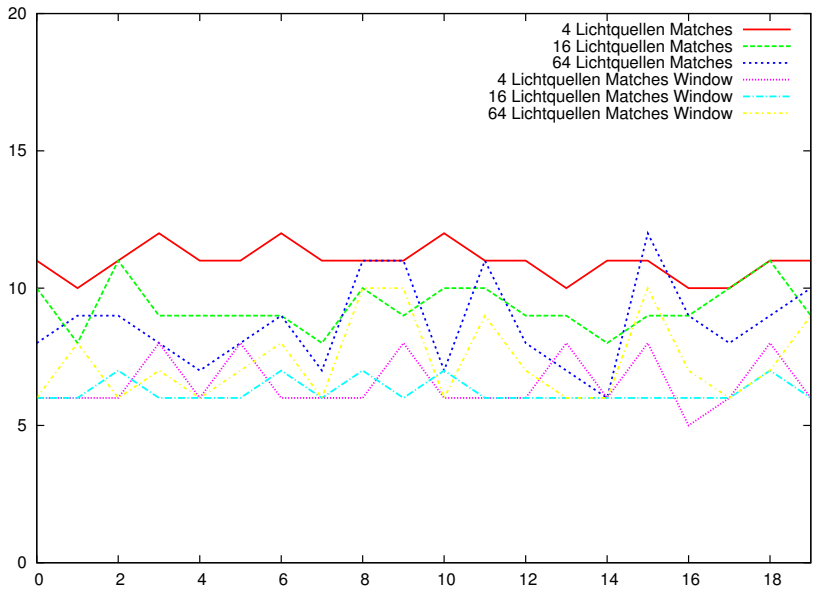


Abbildung 21: Anzahl fehlerhafter (Matches) und gültiger (Matches Window) Merkmalskorrespondenzen bei unterschiedlicher Anzahl Lichtquellen in Szene 1. X-Achse Framenummer, Y-Achse Anzahl Merkmalskorrespondenzen.

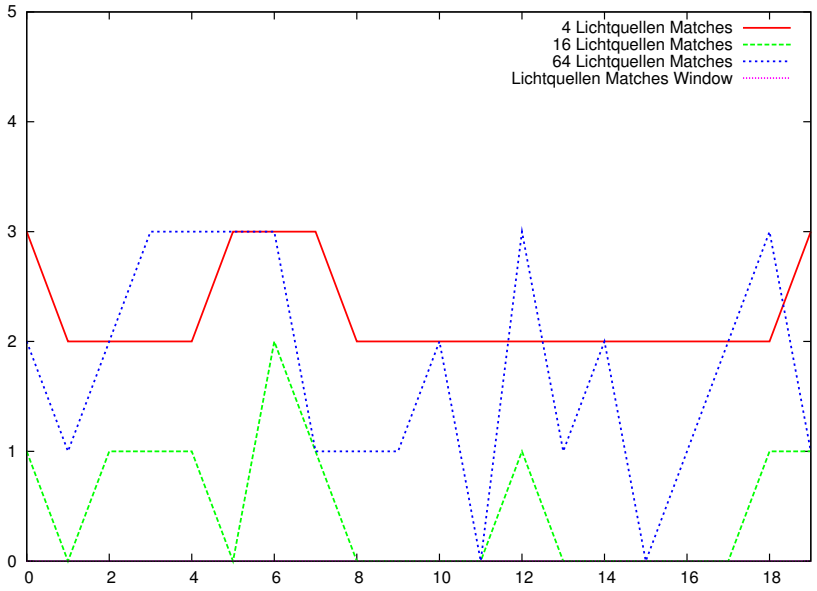


Abbildung 22: Anzahl fehlerhafter (Matches) und gültiger (Matches Window) Merkmalskorrespondenzen bei unterschiedlicher Anzahl Lichtquellen in Szene 2. X-Achse Framenummer, Y-Achse Anzahl Merkmalskorrespondenzen.

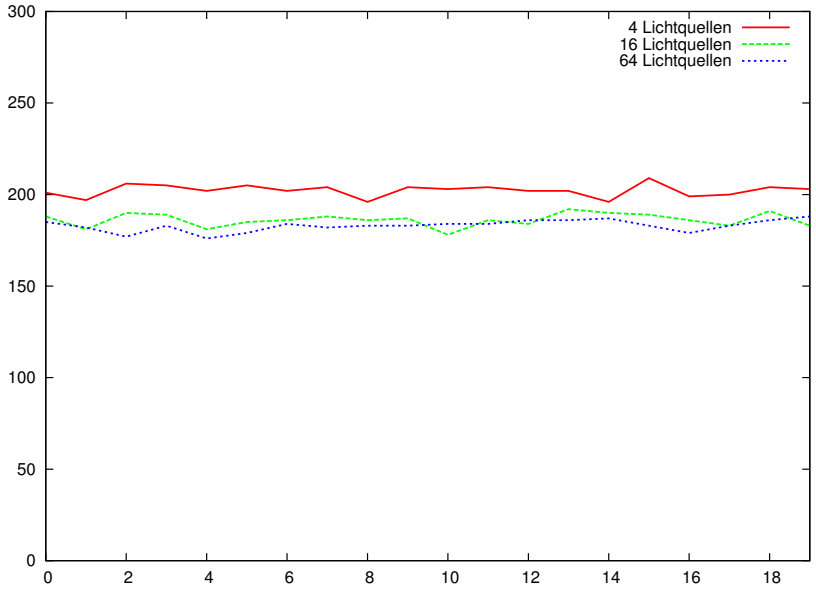


Abbildung 23: Anzahl SIFT Merkmale bei unterschiedlicher Anzahl Lichtquellen in Szene 1. X-Achse Framenummer, Y-Achse Anzahl SIFT Merkmale.

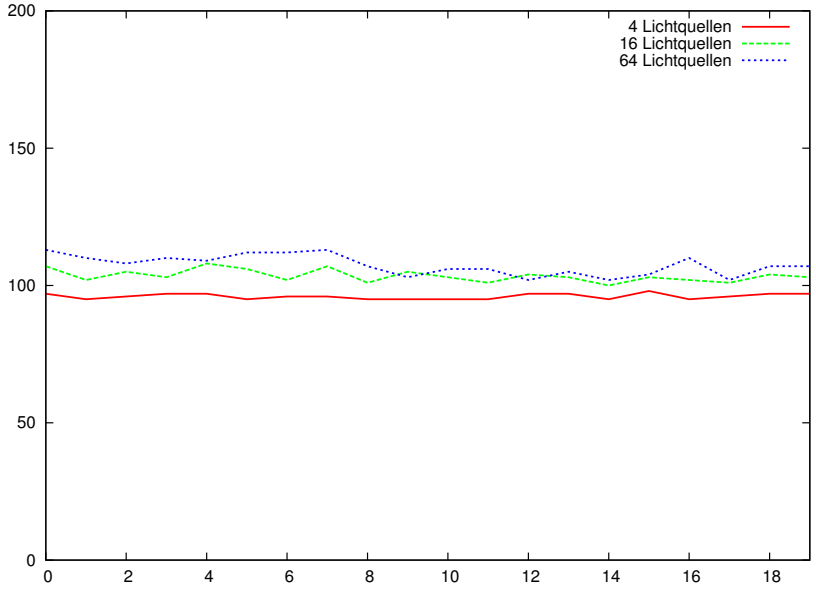
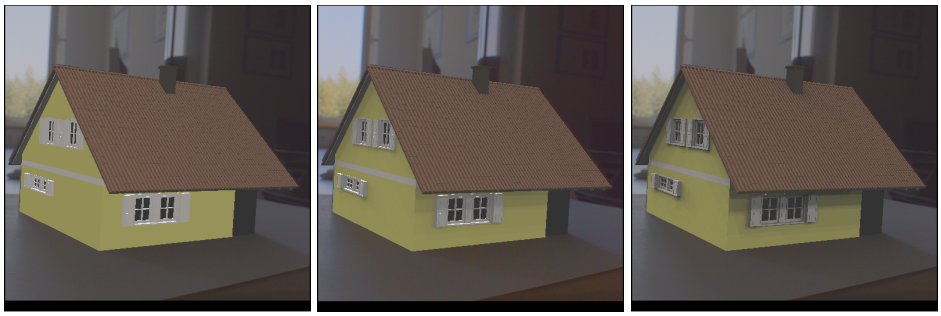


Abbildung 24: Anzahl SIFT Merkmale bei unterschiedlicher Anzahl Lichtquellen in Szene 2. X-Achse Framenummer, Y-Achse Anzahl SIFT Merkmale.

(a) (b) (c)

Abbildung 25: Vergleich Schatten: Opake Schatten 0.0 (a), teilweise transparente Schatten 0.5 (b) und ohne Schatten (c) in Szene 1



(a) (b) (c)

Abbildung 26: Vergleich Schatten: Ohne Schatten (a), teilweise transparente Schatten 0.5 (b) und opake Schatten 0.0 (c) in Szene 2

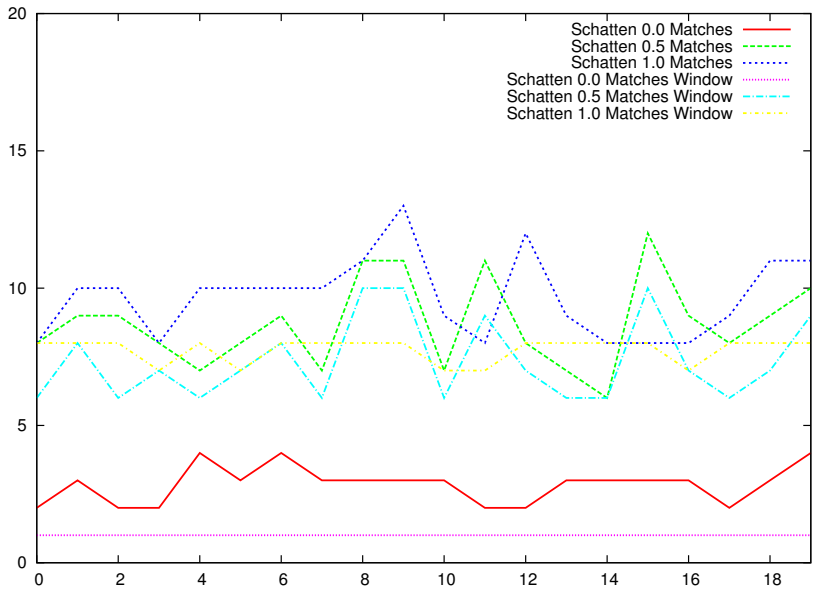


Abbildung 27: Anzahl fehlerhafter (Matches) und gültiger (Matches Window) Merkmalskorrespondenzen bei unterschiedlichen Schatten in Szene 1. X-Achse Framenummer, Y-Achse Anzahl Merkmalskorrespondenzen.

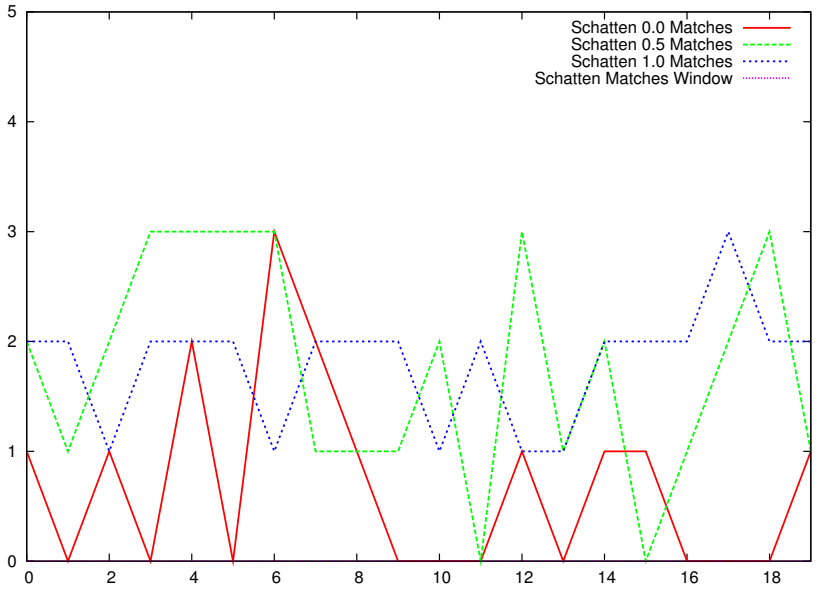


Abbildung 28: Anzahl fehlerhafter (Matches) und gültiger (Matches Window) Merkmalskorrespondenzen bei unterschiedlichen Schatten in Szene 2. X-Achse Framenummer, Y-Achse Anzahl Merkmalskorrespondenzen.

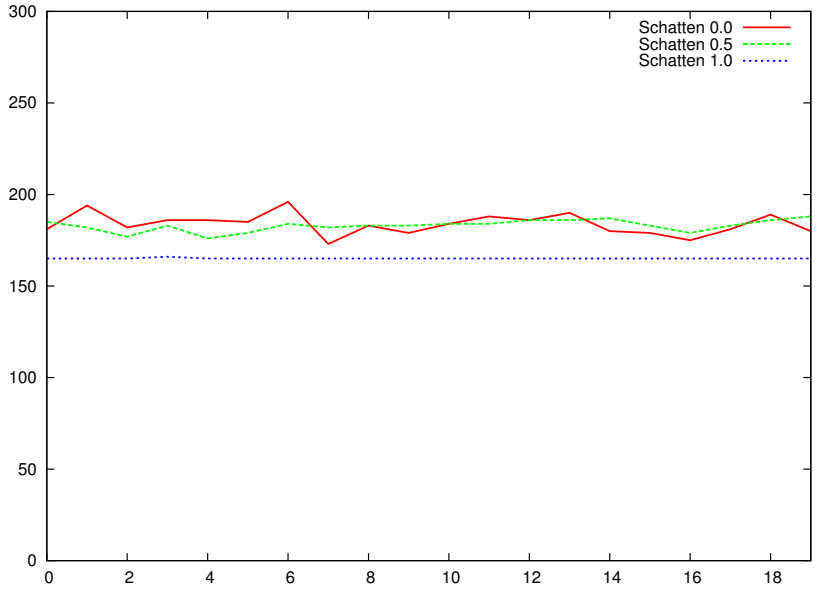


Abbildung 29: Anzahl SIFT Merkmale bei unterschiedlichen Schatten in Szene 1. X-Achse Framenummer, Y-Achse Anzahl SIFT Merkmale.

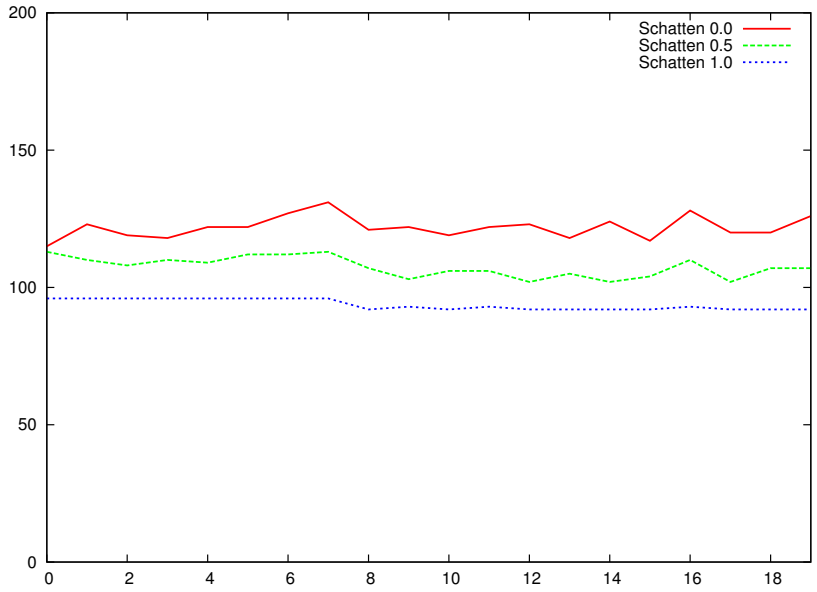
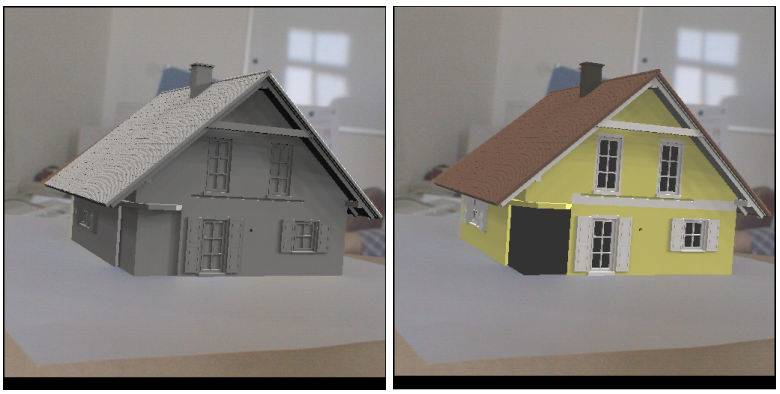
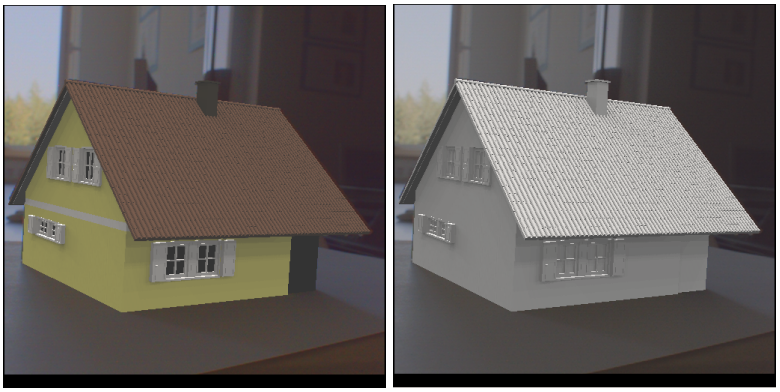


Abbildung 30: Anzahl SIFT Merkmale bei unterschiedlichen Schatten in Szene 2. X-Achse Framenummer, Y-Achse Anzahl SIFT Merkmale.



(a) (b)

Abbildung 31: Vergleich Material: Neutrales Material (a) und Materialfarbe (b) in Szene 1



(a) (b)

Abbildung 32: Vergleich Material: Materialfarbe (a) und neutrales Material (b) in Szene 2

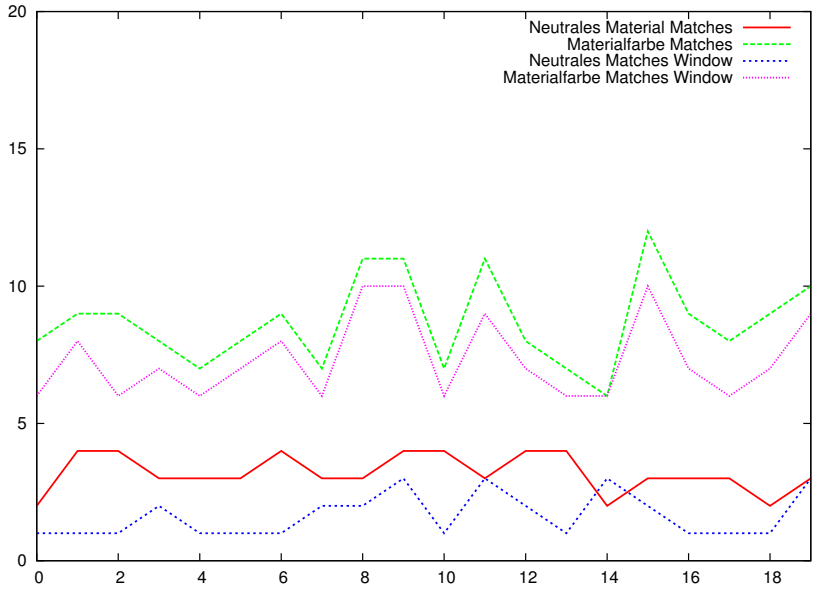


Abbildung 33: Anzahl fehlerhafter (Matches) und gültiger (Matches Window) Merkmalskorrespondenzen bei unterschiedlichen Materialien in Szene 1. X-Achse Framenummer, Y-Achse Anzahl Merkmalskorrespondenzen.

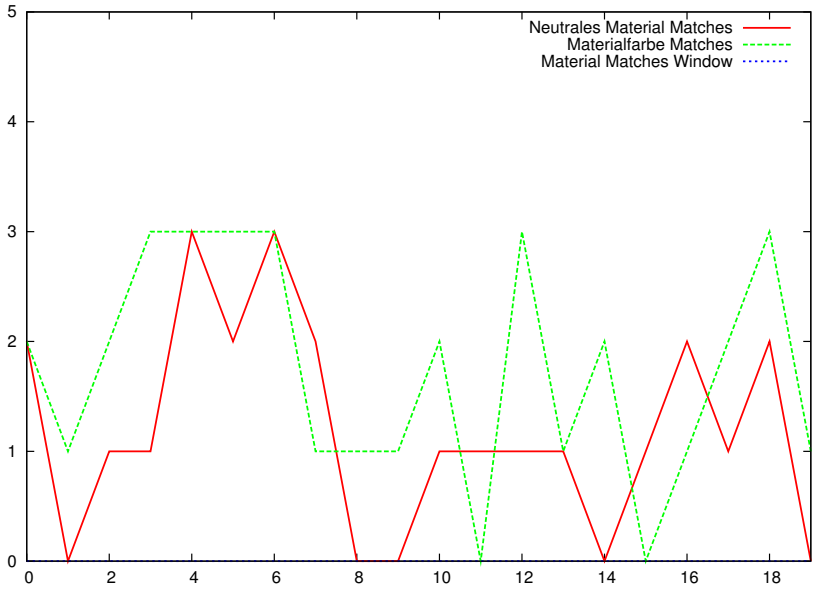


Abbildung 34: Anzahl fehlerhafter (Matches) und gültiger (Matches Window) Merkmalskorrespondenzen bei unterschiedlichen Materialien in Szene 2. X-Achse Framenummer, Y-Achse Anzahl Merkmalskorrespondenzen.

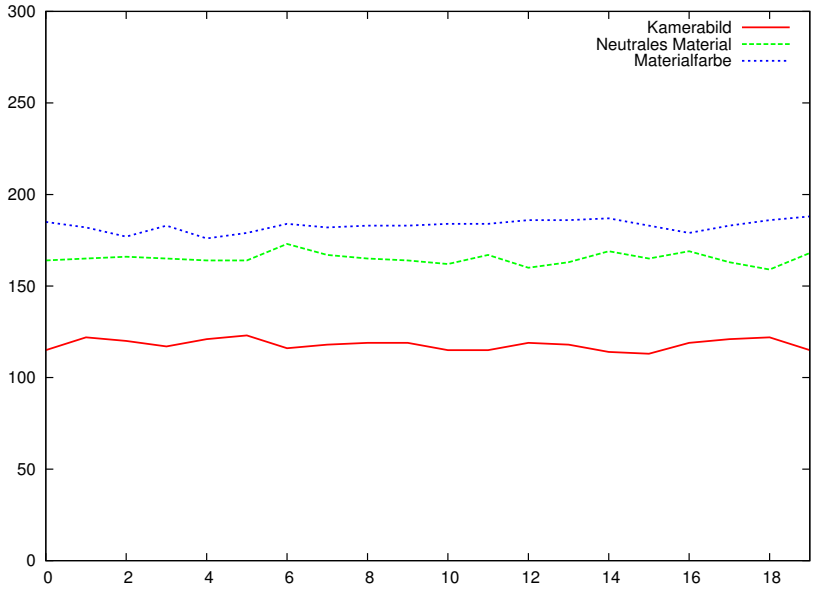


Abbildung 35: Anzahl SIFT Merkmale bei unterschiedlichen Materialien in Szene 1. X-Achse Framenummer, Y-Achse Anzahl SIFT Merkmale.

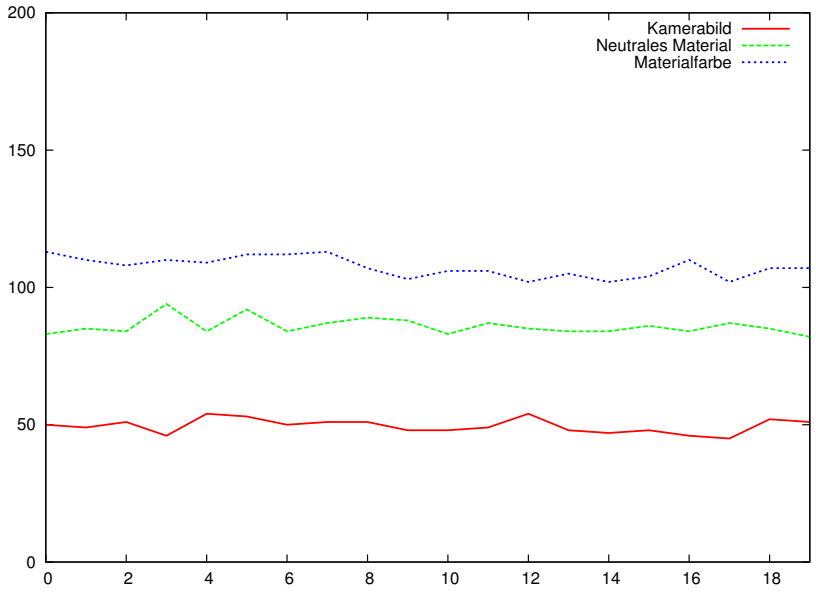


Abbildung 36: Anzahl SIFT Merkmale bei unterschiedlichen Materialien in Szene 2. X-Achse Framenummer, Y-Achse Anzahl SIFT Merkmale.

5.1 Fazit

Der Vergleich der durchschnittlichen Anzahl der SIFT Merkmale mit der Anzahl der Merkmalskorrespondenzen zeigt eine deutliche Differenz. Ein Grund ist das teils sehr unterschiedliche Ergebnis der Detektion von Merkmalen in Kamera- und synthetischem Bild durch den hier verwendeten SIFT Detektor. Deshalb können die unterschiedlich stark gewichteten Parameter des Renderings für Material, Anzahl der Lichtquellen und Schatten auch nur einen Trend beschreiben. Ein Annähern von Kamera- und synthetischem Bild kann durch auf *Analyse durch Synthese* abgestimmte Deskriptoren und Matching geschehen oder durch eine photorealistischere Lichtsimulation.

Das nicht optimale Matching ist der Grund, dass auf eine Evaluierung des verwendeten Trackings, der automatischen Berechnung der Kamerapose, verzichtet wird. Verschiedene Aufnahmen von Beleuchtungssituationen mit und ohne Transformationen des Objekts zeigten durch wechselhaft geringe oder nicht ausreichende Merkmalskorrespondenzen eine nicht stabile Berechnung der Kamerapose. Deshalb soll der Schwerpunkt auf der Beurteilung der Anzahl der Merkmale und Merkmalskorrespondenzen liegen.

Abbildung 19 und 20 zeigen den Vergleich des Samplings und der anschliessenden Beleuchtung des Modells durch 4, 16 und 64 Lichtquellen. Die Kurven der Anzahl der Merkmale in Abbildungen 23 und 24 und der Merkmalskorrespondenzen in Abbildungen 21 und 22 zeigen Unterschiede zwischen beiden Szenen. Die Reduktion der Anzahl der Lichtquellen in Szene 1 erzeugt im Vergleich zu Szene 2 mehr Schatten mit kontrastreichen Kanten. Allgemein zeigt Szene 1 mehr deutliche Fenster, welche die Differenz zwischen Szene 1 und 2 ausmachen. Gleichzeitig wird dadurch die Bedeutung von Beleuchtung und Schatten(kanten) relativiert.

In Abbildung 25 und 26 werden unterschiedlich transparente Schatten betrachtet. Wie die Kurven in den Abbildungen 29 und 30 zeigen, erzeugen Schattenkanten zusätzliche SIFT Merkmale und sind damit potentiell für Matching und Tracking wichtig. Der Vergleich der Merkmalskorrespondenzen in Abbildung 27 und 28 zeigt jedoch gerade bei opaken Schatten, welche in diesem Vergleich die stärksten Schattenkanten erzeugen, deutlich weniger gültige und fehlerhafte Korrespondenzen.

Abbildung 31 und 32 zeigen den Vergleich von neutralem Material und unterschiedlichen Materialfarben in Szene 1 und 2. Sowohl die Anzahl der Anzahl der SIFT Merkmale (Abbildungen 35, 36) als auch die Anzahl der Merkmalskorrespondenzen (Abbildungen 33, 34) macht deutlich, dass eine realistischere Materialfarbe das Matching und Tracking verbessern kann.

6 Zusammenfassung

In dieser Arbeit wurde ein Trackingsystem beschrieben, welches auf der Idee von *Analyse durch Synthese* beruht. Dazu wurde in Bildern einer HDR Videokamera und eines photorealistischen Renderings des zu trackenden Objekts nach natürlichen SIFT Merkmalen gesucht. Als Objekt wurde hier ein Modellhaus in einer leicht abstrahierten, realen Umgebung genutzt, um mögliche Einsatzmöglichkeiten eines *markerlosen Trackings* anzudeuten.

Das Ziel war, den grundsätzlichen Mehrwert von photorealistischen Computergraphiken mit Berücksichtigung ihres Berechnungsaufwands näher zu betrachten. Ausserdem sollten Parameter des Renderings (Lichtsimulation, Schatten, Material) in ihrem Beitrag zu einem besseren Matching und Tracking beurteilt werden. Photorealistische Computergraphik ist je nach Anspruch an die Ähnlichkeit mit der Realität bzw. mit dem Kamerabild aufwendig zu berechnen, so dass in einem interaktiven *Augmented Reality* System die relevanten Faktoren des Renderings bestimmt werden müssen, um die Interaktivität zu erhalten.

In dieser Arbeit wurde in Ansätzen der Weg verfolgt, das Rendering dem Kamerabild so weit anzugleichen, dass derselbe Merkmalsdetektor optimalerweise die gleichen Merkmale in Kamera- und synthetischem Bild findet. Die Ergebnisse zeigen, dass der verwendete SIFT Deskriptor (oder seine Implementation) nicht optimal ist, um ähnliche Merkmale in einem HDR Kamerabild und einem Rendering zu finden, damit modellbasiertes Tracking möglich wird. Auf der anderen Seite würden durch einen Mehraufwand in der Berechnung der Lichtsimulation wahrscheinlich bessere Korrespondenzen gefunden werden.

7 Ausblick

Die Syntheseeinheit im Ansatz *Analyse durch Synthese* durch eine photorealistische Computergraphik darzustellen, ist ein vielversprechender Ansatz. Jedoch ist eine Verbesserung der Qualität und Verlässlichkeit der natürlichen Merkmale mit aufwendigen Berechnungen im Bereich des Detektors oder des photorealistischen Renderings verbunden. Eine Idee ist, die Anzahl der Lichtquellen zu erhöhen bzw. die für das direkte Licht relevanten Lichtquellen genauer zu definieren. Havran u. a. (2005) haben Möglichkeiten vorgeschlagen, die Anzahl der Lichtquellen durch *Clustering* und Diskretisierung unter Berücksichtigung der Geschwindigkeit des Systems zu erhöhen.

In diesem Trackingsystem wurden möglichst viele Berechnungen der Licht- und Schattensimulation, des Merkmaldetektors und des Matchings auf der GPU durchgeführt. Eine ausgewogenere Aufteilung auf CPU und GPU wäre sinnvoll.

Zur Verbesserung der photorealistischen Computergraphik können nichtdiffuse Materialien berücksichtigt werden. Ausserdem sollte neben direktem Licht auch indirektes Licht berücksichtigt werden. Indirektes Licht in einer realen Umgebung kann dabei von in Grundzügen modellierten umliegenden Objekten stammen oder je nach Geometrie des zu trackenden Objekts von sich selbst.

Die hier verwendeten HDR Videokameras haben neben einer geringen Bildauflösung auch ein starkes Farbrauschen. Eine höhere Auflösung des Kamerabilds oder eine Reduktion des Rauschens kann die Detektion von Punktmerkmalen wie SIFT verbessern.

Statt Merkmale auf HDR Bildinformationen zu detektieren, können auch die durch Tonemapping, Weißabgleich, Farbsättigung, Gamma-Korrektur etc. veränderten Bilder Grundlage des Detektors sein. Eine Idee wäre auch, die von RGB- auf Leuchtdichteinformationen abgebildeten Bilder zu benutzen. Etablierte Trackingsysteme arbeiten meist auf LDR bzw. Schwarz-Weiß-Bildern.

Das hier verwendete Matching arbeitet mit unterschiedlich großen Parameterstufen für Rotation, Translation und Abstand der Merkmalskorrespondenzen. Die Wahl der Optimierung, der richtigen Werte dieser Parameter und die adaptive Anpassung der Parameterstufe durch die Anzahl und Ähnlichkeit der gefundenen Merkmalskorrespondenzen sollte verbessert werden.

Eine Alternative zu diesem Ansatz ist die Berechnung der Kamerapose durch einen *robusten Schätzer*. Ein *robuster Schätzer*, genauer ein *Maximum-Likelihood-Artigen Schätzer*, hilft auch bei verrauschten Merkmalskorrespon-

denzen die Kamerapose durch Minimierung des Rückprojektionsfehlers zu approximieren.

Die Modellierung des Tracking-Objekts muss sehr genau sein, da unter Umständen durch kleine Ungenauigkeiten bei der Lichtsimulation unerwünschte Merkmale erzeugt werden. Trotzdem könnten flache Details, die beispielsweise nicht zur Selbstverschattung beitragen, durch Texturen ersetzt werden. SIFT arbeitet mit Details der Texturen und das Rendering würde durch die reduzierte Geometrie schneller berechnet werden.

Literatur

[Achilles 2008] ACHILLES, Sabine: Markerloses Tracking unter Verwendung von Analyse durch Synthese auf Basis von Featuredetektoren. Diplomarbeit. Universität Koblenz-Landau, 2008

[Bay u. a. 2006] BAY, Herbert ; TUYTELAARS, Tinne ; GOOL, Luc V.: Surf: Speeded Up Robust Features. In: *9th European Conference On Computer Vision*. Graz, Austria, 2006

[Bell 1961] BELL, C. G.: Reduction of Speech Spectra by Analysis-by-Synthesis Techniques. In: *The Journal of the Acoustical Society of America* 33(12) (1961), S. 1725–1736

[Debevec 1998] DEBEVEC, Paul: Rendering synthetic objects into real scenes: bridging traditional and image-based graphics with global illumination and high dynamic range photography. In: *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM Press, 1998, S. 189–198. – ISBN 0-89791-999-8

[Ewering 2006] EWERING, Dag: Modellbasiertes Tracking mittels Linien- und Punktkorrelationen. Diplomarbeit. Universität Koblenz-Landau, 2006

[Grabner u. a. 2006] GRABNER, Michael ; GRABNER, Helmut ; BISCHOF, Horst: Fast Approximated SIFT. In: *ACCV 2006: Asian Conference on Computer Vision*, 2006, S. 918–927

[Havran u. a. 2005] HAVRAN, Vlastimil ; SMYK, Miloslaw ; KRAWCZYK, Grzegorz ; MYSZKOWSKI, Karol ; SEIDEL, Hans-Peter: Importance sampling for video environment maps. In: *SIGGRAPH '05: ACM SIGGRAPH 2005 Sketches*. New York, NY, USA : ACM Press, 2005, S. 109

[Lowe 2004] LOWE, D. G.: Distinctive Image Features from Scale-Invariant Keypoints. In: *Int. J. Comput. Vision* Bd. 60(2), 2004, S. 91–110

[Mikolajczyk und Schmid 2005] MIKOLAJCZYK, Krystian ; SCHMID, Cordelia: A performance evaluation of local descriptors. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* Bd. 27, 2005, S. 1615–1630

[Moeslund 1999] MOESLUND, Thomas B.: The Analysis-by-Synthesis Approach in Human Motion Capture: A Review. In: *The 8th Danish conference on pattern recognition and image analysis*, 1999

[Nakamae u. a. 1986] NAKAMAE, Eihachiro ; HARADA, Koichi ; ISHIZAKI, Takao ; NISHITA, Tomoyuki: A montage method: the overlaying of

the computer generated images onto a background photograph. In: *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM Press, 1986, S. 207–214. – ISBN 0-89791-196-2

[Reinhard u. a. 2002] REINHARD, Erik ; STARK, Michael ; SHIRLEY, Peter ; FERWERDA, James: Photographic tone reproduction for digital images, URL `http://www.cs.utah.edu/~reinhard/cdrom/`, 2002

[Sato u. a. 1999] SATO, Imari ; SATO, Yoichi ; IKEUCHI, Katsushi: Acquiring a Radiance Distribution to Superimpose Virtual Objects onto a Real Scene. In: *IEEE Transactions on Visualization and Computer Graphics* 5 (1999), /, Nr. 1, S. 1–12. – URL `citeseer.ist.psu.edu/article/sato99acquiring.html`

[Schumann 2008] SCHUMANN, Martin: Markerloses Tracking unter Verwendung von Analyse durch Synthese auf Basis der Ähnlichkeitsbestimmung stilisierter Bilder. Diplomarbeit. Universität Koblenz-Landau, 2008

[Viola und Jones 2001] VIOLA, Paul ; JONES, Michael J.: Robust Real-time Object Detection. In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* Bd. 1, 2001, S. 511–518

[Wu 2007] WU, Changchang: *SiftGPU: A GPU Implementation of Scale Invariant Feature Transform (SIFT)*. `http://cs.unc.edu/~ccwu/siftgpu`. 2007

[Ziegler u. a. 2006] ZIEGLER, Gernot ; TEVS, Art ; THEOBALT, Christian ; SEIDEL, Hans-Peter: GPU point list generation through histogram pyramids / Max-Planck-Institut für Informatik. Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany, June 2006 (MPI-I-2006-4-002). – Research Report. – ISSN 0946-011X