

Interaktive NPR-Darstellung von Metalloberflächen in Airbrushtechnik

Tibor Schütz

Quelle: www.deinmeister.de

Inhaltsverzeichnis

1	Einleitung	1
2	Airbrush als gestalterisches und künstlerisches Medium	3
2.1	Einsatzgebiete	3
2.2	Farbauftrag	3
2.3	Bildgestaltung	4
2.4	Spezielle Effekte	7
2.4.1	Spittering	7
2.4.2	Kratztechnik	7
2.4.3	Sterneffekte	7
2.4.4	Spitzlichter	7
3	Eigenschaften von Metalloberflächen	10
3.1	Physikalische Eigenschaften	10
3.1.1	Beispieleigenschaften einiger wichtiger Metalle	12
3.2	Strukturelle Eigenschaften	13
4	Entwurfskriterien	14
5	Simulation des Farbauftrags	15
5.1	Grundprinzip: Dithering im Bildraum	15
5.2	Verbesserung durch weichere Stufenfunktionen	17
5.3	Verwendung der gewonnenen Intensitätsverteilung	22
5.4	Animationsfähigkeit	22
5.5	Nachbildung von Schraffurlinien	25
6	Wahl eines geeigneten Beleuchtungsmodells	27
6.1	Diffuser Anteil	27
6.2	Spekularer Anteil	28
6.3	Environment Mapping	29
6.4	Anisotropie	31
6.5	Bumpmaps	34

7	Erweiterte Lichteffekte	35
7.1	Glanzlichter	35
7.2	Sterneffekte	36
8	Implementierung	37
8.1	Verwendete API-Bestandteile	37
8.2	VertexShader	38
8.3	PixelShader	39
8.3.1	Parameter für Lichtquellen und Umgebungslicht	39
8.3.2	Parameter des Materials	41
8.3.3	Parameter des Sprüheffekts	42
8.3.4	Vorbereitung interpolierter Werte	44
8.3.5	Berechnung und Modulation des Sprühparameters	45
8.3.6	Beleuchtungsberechnung	46
8.4	Glanzlichter	50
8.5	Sterneffekte	51
A	Ergebnisse/Bilder	52
B	Definitionen und Begriffe	61
C	Quellenverzeichnis	66
D	Shader-Quelltexte	68
D.1	Vertexshader des Hauptdurchgangs	69
D.2	Pixelshader des Hauptdurchgangs	71
D.3	Vertexshader zur Erzeugung der spekularen Textur (erster Durchgang)	77
D.4	Pixelshader zur Erzeugung der spekularen Textur (erster Durchgang)	79
D.5	Vertexshader für Glanzlichter (zweiter Durchgang)	83
D.6	Pixelshader für Glanzlichter (zweiter Durchgang)	84
D.7	Vertexshader für Sterneffekte (zweiter Durchgang)	85
D.8	Pixelshader für Sterneffekte (zweiter Durchgang)	88
D.9	Shader zur Erzeugung einer vorberechneten zylindrischen Environmentmap mit automatisch generierten Farben für Ober- und Unterseite	89
D.10	Shader zur Erzeugung einer vorberechneten zylindrischen Environmentmap mit Überblendung zu vorgegebenen Farben für Ober- und Unterseite	90

Kapitel 1

Einleitung

Bei der Darstellung dreidimensionaler Objekte und Szenen gibt es in der Computergraphik zwei wichtige Schwerpunkte: Die Erstellung photorealistischer Bilder einerseits, sowie mehr oder weniger skizzierte Grafiken mit Schwerpunkt auf einer möglichst guten Erkennbarkeit der Details andererseits. Daneben gibt es viele Ansätze, auch künstlerische Techniken nachzubilden, welche unter dem Sammelbegriff des nicht-photorealistischen Renderings (NPR) gerade in letzter Zeit einen deutlichen Aufschwung erfahren haben.

Zu den verbreitetsten NPR-Techniken gehört das Cool-to-Warm-Shading, bei dem die normale Beleuchtung mit Schattierungen durch eine Beleuchtung mit warmem Licht und entgegengesetztem kaltem Gegenlicht ersetzt wird; diverse texturbasierte Verfahren zur Nachbildung von refraktierenden, spiegelnden und metallischen Oberflächen sowie das Cartoon-Rendering. Zu Letzterem gibt es eine Vielzahl an Verfahren, deren gemeinsame Hauptmerkmale das Nachzeichnen von Objektkanten und Silhouetten zusammen mit der Verwendung von wenigen Helligkeits- und Farbstufen für Schattierungen bilden. Mittlerweile gibt es auch Ansätze, Metasymbole für Bewegungen und Emotionen wie Staubwolken, Bewegungslinien oder Gedankenblitze geeignet in das Cartoon-Rendering einzubinden.

Eine andere Klasse nichtrealistischer Darstellung bilden Verfahren, die weniger auf eine klare Darstellung sondern mehr auf die Nachbildung künstlerischer Stilrichtungen abzielen. Dazu zählen Kupferstiche und Radierungen, bei denen die Darstellung von Helligkeit und Konturen vorwiegend durch die Dichte und Lage der verwendeten Punkte bzw. Linien erzielt wird, ebenso wie Schraffurzeichnungen, bei denen mit unterschiedlich breiten Linien Schattierungen realisiert werden. In der Regel folgen diese Linien den Oberflächen, was in den verwendeten Verfahren in der Regel eine aufwendige Vorverarbeitung erfordert. Vergleichsweise einfach sind dagegen Techniken, welche Kohlezeichnungen nachbilden und auf Texturen zur Modulation der Schattierungen entsprechend Papier und gezeichneter Oberflächen beruhen.

Den Abschluß bilden Painterly-Verfahren zur Nachbildung von Malereien. Die Farbe und mitunter auch Form und Lage der Pinselstriche werden dabei einem Referenzbild entnommen. Für animierte Bilder ist allerdings auch hier ein gewisser Aufwand an Vorverarbeitung nötig, sofern man die Pinselstriche den Objekten folgen lassen möchte.

Bisher weitgehend unbeachtet blieb dagegen der Airbrush, mit dem es im Bereich der darstellenden Künste ein Medium gibt, welches Abstraktion erfolgreich mit Realismus verbindet. Eine genauere Analyse zeigt jedoch, daß viele Verfahren mit dem Airbrush über Entsprechungen in der Computergraphik verfügen oder leicht in diese zu übernehmen sind. Darüber hinaus eignen sie sich besonders für eine Realisierung mittels gängiger Rasterisierungshardware, was sie speziell für geschwindigkeitskritische, interaktive Anwendungen geeignet macht. Die im folgenden beschriebene vergleichsweise einfache Erzeugung semirealistischer Graphiken im Airbrushstil bietet hier eine interessante Alternative zu photorealistischen Verfahren, da letztere für qualitativ hochwertige Bilder schnell einen immensen Aufwand erfordern.

Der Schwerpunkt wird im Folgenden auf der Darstellung von Metalloberflächen samt ihrer vielfältigen Erscheinungsformen liegen. Aus der Fülle gestalterischer Techniken und physikalischen Grundlagen werden diejenigen vorgestellt, die für die Umsetzung relevant sind.

Kapitel 2

Airbrush als gestalterisches und künstlerisches Medium

2.1 Einsatzgebiete

Von der Lackierung in der gewerblichen Produktion über die Retusche von Photos und Erstellung realistisch wirkender Bilder bis hin zu freier künstlerischer Gestaltung findet der Luftpinsel Gebrauch. Besonders beliebt ist er bei der Erstellung von Illustrationen und Werbegraphiken aufgrund der perfekt wirkenden Farbverläufe und der klaren Darstellung unter Verzicht auf störende Details. Auch zur Ergänzung bestehender Liniengraphiken aus Konstruktionsplänen oder der Colorierung von Cartoons wird er deshalb öfters eingesetzt.

2.2 Farbauftrag

Um Verwirrungen zu vermeiden, soll im Folgenden - ähnlich wie in [1] - zwischen der Farbe als Eigenschaft und „Farbe“ als Material unterschieden werden. Letztere wird für den Rest des Dokuments ungeachtet des tatsächlichen Aufbaus als Pigment bezeichnet, wohingegen mit Farbe die Farbe von Pigmenten, Oberflächen oder Licht gemeint ist.

Im Airbrush werden vorwiegend zwei Pigmenttypen eingesetzt: Lasuren und Tuschen mit subtraktiver Farbmischung und geringer Deckkraft sowie deckende Pigmente, deren Mischverhalten direkt dem Anteil des aufgetragenen Pigments an der Oberfläche entspricht. Seltener werden auch Glimmerpigmente mit additivem Mischverhalten verwendet, aufgrund des stark metallischen Glitzereffekts sind sie vorwiegend bei Effektlackierungen anzutreffen.

Lasuren werden aufgrund ihrer mangelnden Deckfähigkeit stets von hellen nach dunklen Farbtönen angewendet, wohingegen deckende Pigmente auch ein Vorgehen in umgekehrter oder beliebiger Reihenfolge erlauben. Auffällig ist, daß in beiden Fällen die meist weiße Farbe des Untergrunds mit in die farbliche Gestaltung eingeht. Dies liegt weniger an dem

geringeren Arbeitsaufwand oder der begrenzten Deckfähigkeit des eingesetzten Pigments, sondern vor allem an der meist unübertroffenen hohen Strahlkraft reinen Papiers als auch vieler grundierten Oberflächen im Vergleich zu den eingesetzten Pigmenten.

Aufgrund des hohen Aufwands für das Reinigen des Airbrushs beim Wechsel des Pigments als auch aus gestalterischen Gründen beschränken sich Airbrushbilder meist auf wenige Grundfarben. Aus diesen entsteht dann durch Mischung mittels unterschiedlich starkem Auftrag des Pigments die vollständige Farbpalette. Stärke, Verlauf und Breite des Pigmentauftrags können dabei durch Variation des Verhältnisses von Pigment und Luft, dem Sprühwinkel und dem Abstand zur Oberfläche fein dosiert werden.

Während vollständig gleichförmige Farbflächen sehr leicht erstellbar sind erfordert ein großflächiger Farbverlauf viel Geschick und Übung. Aus diesem Grund sind weite, gleichmäßige Farbverläufe eher selten anzutreffen. Wesentlich bedeutendere Folgen hat jedoch die Einschränkung, daß ein Airbrush das Pigment selbst mit dem präzisesten Gerät stets auf einer Fläche verteilt.

Aus diesem Grund werden meist zusätzliche Mittel eingesetzt: Mit einem Pinsel lassen sich Details einfügen, feine Linien können mit Aquarell-, Filz- und Tuschestiften ergänzt werden. Ebenso kann einem gleichförmigem Pigmentauftrag durch Wischen, Tupfen und Radieren mit den unterschiedlichsten Materialien mehr Struktur verliehen werden [2,3,5].

2.3 Bildgestaltung

Am Anfang steht meist eine Vorzeichnung anhand eines freien Enturfs oder einer Photographie. Zur Erhöhung des realistischen Eindrucks und der beabsichtigten Wirkung wird eine Perspektive mit ein, zwei oder drei Fluchtpunkten eingesetzt. Darüber hinaus wird durch die Wahl einer warmen oder kalten, gebrochenen oder satten Farbpalette die Stimmung des Bildes festgelegt.

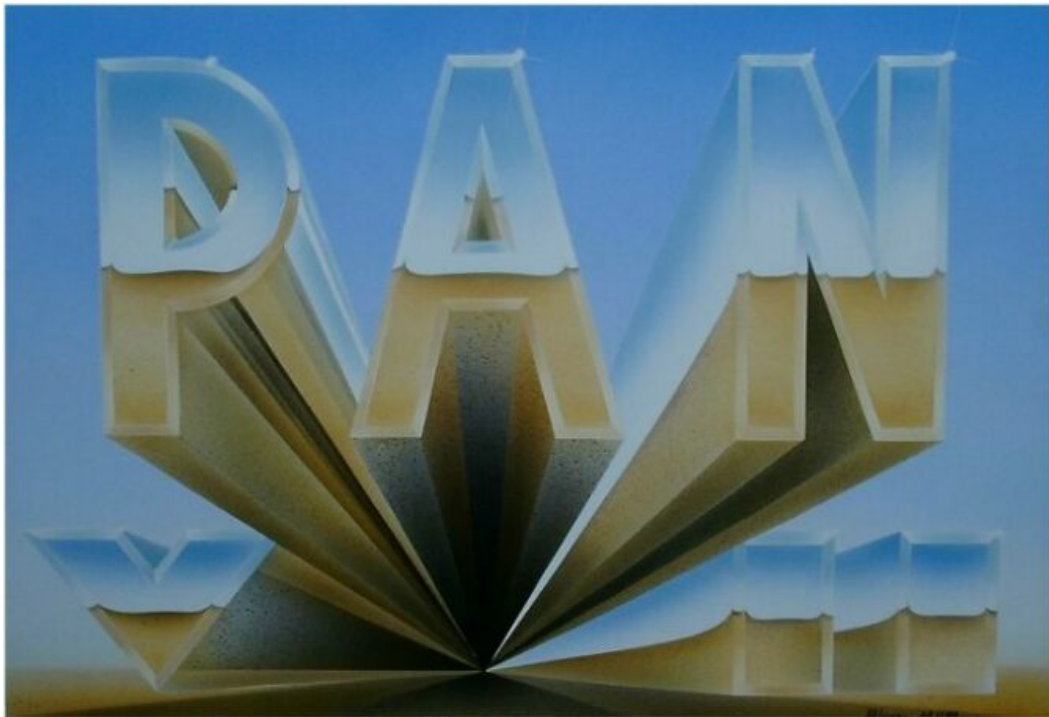
Während ein Airbrush von Haus aus für weiche, organische Verläufe geeignet ist sind für die Darstellung mehr oder minder harter Kanten Hilfsmittel nötig. Aus diesem Grund gehören Masken verschiedenster Art zur Eindämmung des Sprühstrahls zum zweitwichtigsten Werkzeug eines Künstlers. Zum Maskieren eignen sich spezieller Decklack, aber auch Folie, geschnittenes oder gerissenes Papier, Watte sowie fast jeder sonstige Gegenstand. Je nach Abstand zum Untergrund und der Randschärfe dieser Maske ergeben sich unterschiedliche Verläufe und Strukturen. Darüber hinaus lassen sich Schablonen durch Verschieben auch für Abstufungen und Schlagschatten einsetzen.

Reflektierte Lichter werden anhand eines Referenzbildes oder mittels gedachter Lichtquellen umgesetzt. Meist verfügen diese über parallel auftretende Lichtstrahlen, wie sie vor allem für Sonnenlicht als bedeutendste Lichtquelle zutreffen. Für gestalterische Zwecke sind Abweichungen von der Realität allerdings durchaus gängig.

Besondere Sorgfalt erfordert die Umsetzung von Metall. Am besten verdeutlichen dies die Zitate einiger bekannter Künstler: „*Wichtig ist, [...] daß Licht und Schatten auf Metall wie in einem Spiegel projiziert werden, wobei sie der äußeren Form des Gegenstands folgen.*“ [2] sowie „*Es gibt keine konkrete Darstellung von Gold, sondern nur die Farben, die sich in der goldenen Fläche spiegeln.[...]*“ [3]. Streng physikalisch trifft dies zwar auch auf alle anderen Materialien zu, ist dort aber für die menschliche Wahrnehmung weitgehend unbedeutend.

Spiegelungen werden im Airbrush typischerweise nur stark idealisiert dargestellt. In Innenräumen ist dies meist das Abbild eines oder mehrerer Fenster, für im Freien stehende Objekte wird je eine Farbe für die Himmelskuppel sowie dem Boden gewählt, welche durch einen teils unregelmäßigen Farbverlauf am Horizont getrennt werden.

Die Wirkung sich selbst abschattender Oberflächen wird gerne überzeichnet, um den plastischen Eindruck zu betonen. Darüber hinaus finden Schatten nur spärlich Anwendung, etwa an eng benachbarten Stellen, an denen sein Fehlen schnell auffallen würde, oder zur Hervorhebung von Objekten und ihrer Lage.



Einsatz eines idealisierten Horizonts und vereinfachter Schatten
(Quelle: Airbrush-Galerie Kai Blume - www.airbrush-galerie-blume.de)

2.4 Spezielle Effekte

2.4.1 Spittering

Nicht immer ist ein homogener Farbverlauf ideal. Um rauhe Oberflächen wie Sand, Stein oder unedles, oxidationsanfälliges Metall darzustellen eignet sich ein gesprenkelter Pigmentauftrag besser. Dieser kann durch Verändern der Luftzufuhr erzielt werden. Falls nur wenige Flecken erzeugt werden sollen, etwa für einen Sternenhimmel oder beginnende Korrosion, kann das Pigment auch mittels Pinsel oder Bürste auf den Untergrund gespritzt werden.

2.4.2 Kratztechnik

Mittels Messer oder Sandpapier lassen sich Pigmentschichten abtragen, so daß die darunterliegende Farbe wieder zum Vorschein kommt. Neben der Herstellung feinsten Details eignet sich diese Technik vor allem zur Darstellung gebürsteter und gefräster Oberflächen, bei denen zwar die Linienstruktur, nicht mehr aber einzelne Linien wahrnehmbar sind[6].

2.4.3 Sterneffekte

Optische Systeme zeigen oftmals, nicht selten sogar beabsichtigt, sternförmige Ausläufer mit 4, 6 oder 8 Ästen rund um helle Lichtquellen und Reflexionen. Dieser Effekt wird im Airbrush meist als Kombination von Sprühtechnik für den Kern sowie Kratztechnik für die Äste nachgebildet. Im Gegensatz zu den flächigen Reflexionen mit Glanzlichtern verdeutlicht er jedoch bevorzugt punktförmige Spitzlichter an besonders hochwertig polierten Oberflächen, kann aber auch mit flächigen Spitzlichtern kombiniert werden[3].

2.4.4 Spitzlichter

Ein generelles Problem fast aller Maltechniken ist der begrenzte Dynamikumfang der Pigmente, welcher für hoch reflektierende Oberflächen von Wasser und Metallen nicht ausreicht. Um sie dennoch darstellen zu können, werden die hellsten Bereiche zusätzlich mit Weiß (also dem hellsten verfügbaren Pigment) wolkenartig übersprüht, wodurch die Überstrahlung hellen Lichts in benachbarte Bereiche nachgebildet wird. Richtig eingesetzt entspricht dies durchaus dem Verhalten überbelichteter Bereiche in einer realen Kamera[2,3].



Beugungssterne und ein skizzierter Horizont sowie Einsatz von Masken, Radierer, Pinsel und Spittering (Quelle: Airbrush-Galerie Kai Blume - www.airbrush-galerie-blume.de)



Chromeffekt hervorgehoben durch Reflektionen mit hohem Detailgrad und Kontrast
(Quelle: A.D. Cook Fine Art - <http://adcookfineart.com>)

Kapitel 3

Eigenschaften von Metalloberflächen

3.1 Physikalische Eigenschaften

In Luft oder Vakuum (Brechungsindex=1) gelten für den als Betragsquadrat des Reflexionskoeffizienten R definierten Reflexionsgrad ρ mit Brechungsindex n folgende Fresnel-Formeln[7,9]:

$$R_{parallel} = \frac{n^2 * \cos(\alpha) - \sqrt{n^2 - \sin(\alpha)^2}}{n^2 * \cos(\alpha) + \sqrt{n^2 - \sin(\alpha)^2}}$$

$$R_{Senkrecht} = \frac{(\sqrt{n^2 - \sin(\alpha)^2} - \cos(\alpha))^2}{n^2 - 1}$$

Dabei wird der Winkel α gegen die Oberflächennormale gemessen, $R_{parallel}$ gilt für parallel und $R_{senkrecht}$ für senkrecht zur Reflexionsebene polarisiertes Licht. Diese vereinfachten Formeln gelten strenggenommen nur für nichtmagnetische Materialien, was aber für die folgenden qualitativen Betrachtungen keine Einschränkung darstellt.

Hier lassen sich zwei wichtige und zudem polarisationsunabhängige Eigenschaften ableiten: Erstens geht der Betrag des Reflexionskoeffizienten für flachen Lichteinfall gegen 1, wodurch das gesamte einfallende Licht unabhängig von der Farbe reflektiert wird. Zweitens beträgt der Reflexionskoeffizient für senkrechten Lichteinfall $\frac{n-1}{n+1}$, so daß Farbe und Intensität des reflektierten Lichts nur vom frequenzabhängigen Brechungsindex abhängen.

In Metallen ist durch die frei beweglichen Elektronen zusätzlich der Leitungsstrom sowie in Folge die auftretende Dämpfung einfallender elektromagnetischer Wellen durch den elektrischen Widerstand des Metalles zu berücksichtigen. Dazu wird die reelle Dielektrizitätskonstante ϵ_r um einen imaginären Anteil erweitert, so daß auch der Brechungsindex n zu einer komplexen Zahl wird. Für den imaginären Anteil gilt bei Wellen mit Frequenz f :

$$\text{Im}(n) = \sqrt{\frac{\mu_r \sigma}{4\pi \epsilon_0 f}}$$

Hierbei sind als Materialparameter μ_r als relative Permeabilität, die Leitfähigkeit σ sowie die Vakuumdielektrizitätskonstante ϵ_0 einzusetzen.

In weiten Teilen des sichtbaren Bereichs ist

$$n = \sqrt{\frac{\epsilon_r \mu_r + i\sqrt{\sigma \mu_r}}{4\pi f \epsilon_0}} \text{ näherungsweise } \text{Im}(n) \gg 1,$$

woraus die hohe Reflektivität selbst bei senkrechtem Lichteinfall folgt.

Die Eindringtiefe T elektromagnetischer Wellen bis zu einem Abfall auf $\frac{1}{e}$ ihrer ursprünglichen Intensität ergibt sich zu

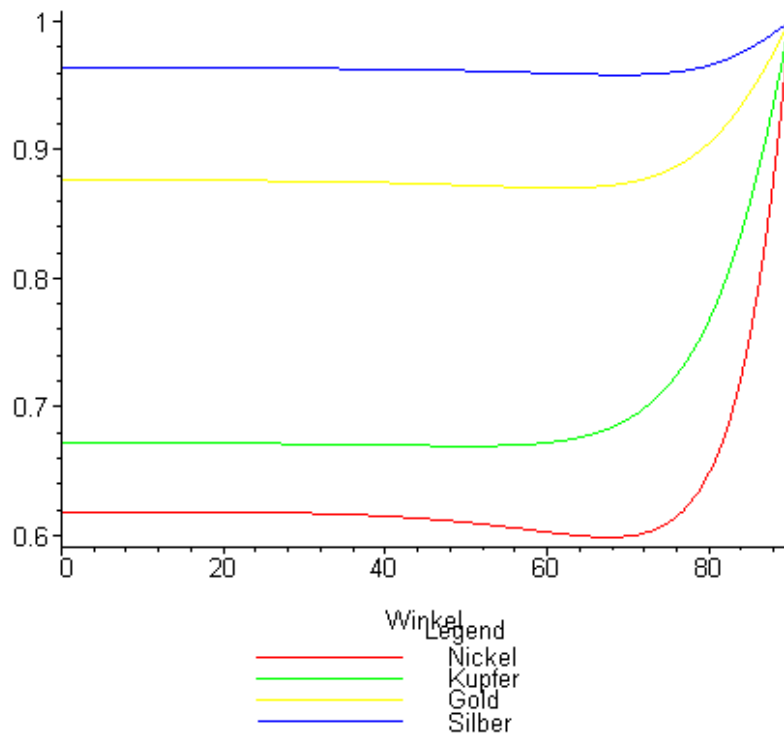
$$T = \frac{\lambda}{4\pi \frac{\text{Im}(n)}{\text{Re}(n)}},$$

wobei λ die Wellenlänge im Metall bezeichnet. Somit findet die Reflexion sichtbaren Lichts praktisch ausschließlich an der Oberfläche statt.[7,8]

3.1.1 Beispieleigenschaften einiger wichtiger Metalle

Sämtliche Angaben beziehen sich auf unpolarisiertes, senkrecht einfallendes grünes Licht ($\lambda = 530nm$)

Metall	$Re(n)$	$Im(n)$	Reflexionsgrad ρ	Relative Eindringtiefe in λ
Nickel	1,80	3,32	0,62	0,043
Kupfer	0,83	2,60	0,67	0,025
Gold	0,31	2,88	0,88	0,009
Silber	0,12	3,45	0,96	0,003



Winkelabhängigkeit des Reflexionsgrades

3.2 Strukturelle Eigenschaften

Da aufgrund der geringen Eindringtiefe praktisch keine Streuung des Lichts innerhalb des Metalls auftritt hängt das Reflexionsverhalten stark von der herstellungs- und verwendungsbedingten Oberflächenstruktur ab. Wichtige im Alltag vorkommende Feinstrukturen sind:

- Geätzte, leicht glitzernde Oberflächen mit mikroskopisch rauher isotroper Struktur
- Nahezu perfekt ebene polierte oder gewalzte Oberflächen, mitunter mit leichter Anisotropie
- Gegossene, eher matte Oberflächen mit rauher Struktur
- Gebürstete, gedrehte oder gefräste Bauteile mit stark anisotroper Reflexion und glatter Oberfläche

Anisotropie tritt bei den genannten Formen in einer stärkeren Reflexion in Abhängigkeit zur Bearbeitungsrichtung auf. Darüber hinaus wirken sich durch den hohen Reflexionsgrad auch leichte Variationen der Grobstruktur, wie sie durch Schrumpfungen an gegossenen Gegenständen, punzierten, gehämmerten und verbogenem Blechen sowie durch Applikationen wie Schrauben und Nieten entstehen, stark aus.

Kapitel 4

Entwurfskriterien

Neben der Berücksichtigung oben genannter Schlüsseigenschaften sollte das zu entwickelnde Verfahren sich möglichst nahtlos in bestehende Anwendungen einfügen und kombinierbar mit anderen Verfahren sein. Dies bedeutet, daß es unabhängig von der gewählten Form der Verdeckungsrechnung anwendbar sowie möglichst keine Annahmen über die Tiefen-, Stencil-, Alpha- und Scissorstufen der Graphikhardware sowie dem gewählten Rendertarget voraussetzen sollte. Bestehende 3D-Objekte sollten ohne manuelle Änderung direkt einsetzbar sein, ebenso die bisher verwendeten Lichtquellen und Transformationen.

Von speziellem Interesse war auch die Eignung für interaktive, animationsfähige Anwendungen. Somit sollten sich Änderungen der Objekt- und Beobachungsposition sowie beliebiger Parameter kontinuierlich und mit gleichbleibender Bildqualität auswirken. Die verwendeten Parameter sollten intuitiv verständlich und leicht bedienbar sein, aus Geschwindigkeitsgründen sollte das Verfahren möglichst innerhalb eines Renderdurchgangs erfolgen und auf weiteres Postprocessing verzichten.

Kapitel 5

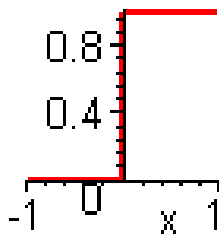
Simulation des Farbauftrags

Kern des Verfahrens ist die Nachbildung des Pigmentauftrags. Diese sollte orthogonal zu der Berechnung der Farbwerte und Intensitäten erfolgen, wodurch spätere Erweiterungen problemlos unterstützt werden.

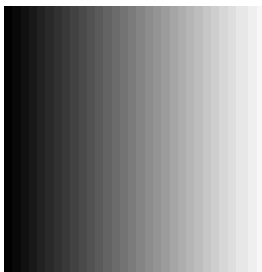
5.1 Grundprinzip: Dithering im Bildraum

Für den fertigen Bildeindruck ist nur das Aussehen, nicht aber das Zustandekommen des Pigmentauftrags relevant. Das Aussehen hängt dabei direkt von dem Verhältnis der mit Pigment bedeckten zur unbedeckten Fläche ab, so daß der Farbauftrag mittels eines Ditheringverfahrens nachgebildet werden kann. Ein- und Ausgabedaten des Dithervorgangs sind jedoch keine Farbwerte, sondern die mittlere und lokal resultierende Intensität des Pigments. Das Verfahren läuft im Bildraum ab, um eine gleichbleibende und objektunabhängige Streuauflösung wie auf dem Papier zu gewährleisten. Davon abgesehen entspricht es der schon in [10] verwendeten Vorgehensweise, die letztendlich eine Variante des geordneten Ditherings ist.

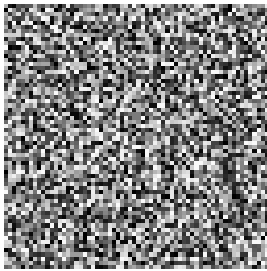
Basis des Ditherings ist die Zuordnung von individuellen Zufallswerten zu den Pixeln im Bildraum. Diese sind zwischen 0 und 1 gleichverteilt (Erwartungswert = 0,5) und werden von der ebenfalls zwischen 0 und 1 liegenden lokalen Intensität subtrahiert. Die Differenz x beider Werte wird mittels einer Stufenfunktion $S(x)$ in den resultierenden lokalen Intensitätswert umgesetzt.



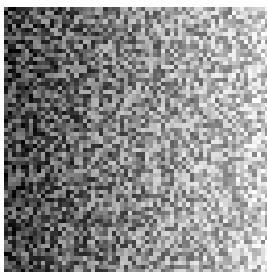
Verwendete Stufenfunktion $S(x)$



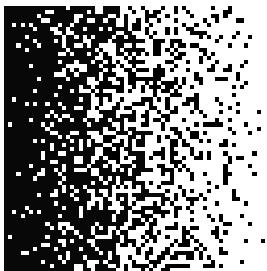
Die ursprüngliche Intensität eines Farbverlaufs



1:1 den Pixeln zugeordnete Zufallswerte



Pixelweise ausgewertete Differenz x

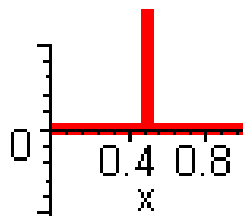


Nach Anwendung von $S(x)$

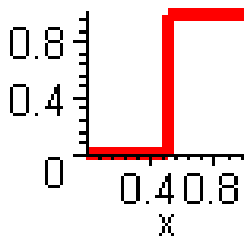
Die Zuordnung der Zufallswerte erfolgt am geschicktesten durch eine Textur, deren Texel 1:1 auf entsprechende Pixel abgebildet werden und mit gleichverteilten Werten im Intervall $[0..1]$ versehen sind. An den zur Erzeugung verwendeten Zufallsgenerator sind relativ geringe Anforderungen nötig, da durch die Texturwiederholung sich die Werte ohnehin entsprechend der Texturgröße wiederholen. Somit genügen schon einfache Modulo-Generatoren den Anforderungen. Wichtig ist jedoch, daß es sich tatsächlich um eine Gleichverteilung der Werte handelt, da nur so die Gleichheit von mittlerer Intensität nach dem Dithering und ursprünglicher Intensität garantiert ist. Eine solche Textur kann aus einer beliebigen Textur mit N Zufallswerten hergestellt werden, indem dem i -t größten Zufallswert der Wert $I = \frac{i-1}{N-1}$ zugeordnet wird.

5.2 Verbesserung durch weichere Stufenfunktionen

Weil alle simulierten Pigmentflecken dieselbe Größe haben wirkt der auf diese Weise erzielte Verlauf ziemlich rauh. Betrachtet man nun den Pigmentauftrag stochastisch, so entspricht die Pigmentfleckengröße einer Zufallsdichte $D(x)$ mit der gewählten Stufenfunktion als dazugehörigen Zufallsfunktion $S(x)$. Die Differenz x wird hierbei als per $x := 0.5 * (x + 1)$ auf das Intervall $[0..1]$ abgebildet angenommen.

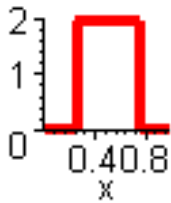


Dichte der Stufenfunktion: $\frac{dS(x)}{dx}$

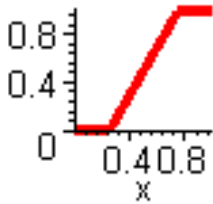


Stufenfunktion $S(x)$

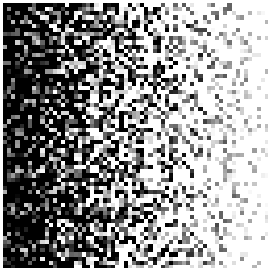
Ein besseres Ergebnis erhält man, indem man von einer Gleichverteilung zwischen einer kleinsten und einer größten Pigmentfleckengröße auf dem Papier ausgeht. Die dazugehörige Zufallsfunktion ist eine Verallgemeinerung der Stufenfunktion: $S_{lin}(x) = Saturate(w*x + 0.5*(1-w))$ und konvergiert für $w \rightarrow \infty$ gegen diese.



Dichte der Stufenfunktion ($w = \frac{1}{2}$): $\frac{dS_{lin}(x)}{dx}$



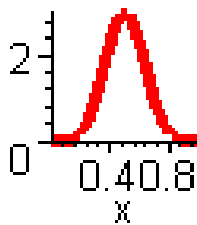
Stufenfunktion $S_{lin}(x)$



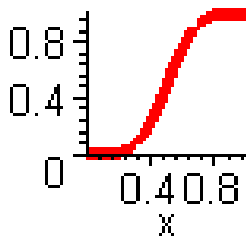
Nach Anwendung von $S_{lin}(x)$

Realistischer ist es jedoch, davon auszugehen daß schon die gesprühten Tropfen selbst einer Gleichverteilung unterliegen. Unter der Annahme, daß das Pigment noch flüssig bleibt und sich somit die bedeckte Fläche als Summe vieler Tropfengrößen ergibt, wird die Zufallsdichte bei längerem Sprühen zu einer Glockenkurve[14]. Ein qualitativ vergleichbares Ergebnis ergibt sich unter Annahme einer normalverteilten Zufallsdichte. Auch hier kann die Zufallsfunktion als verallgemeinerte Stufenfunktion gesehen werden.

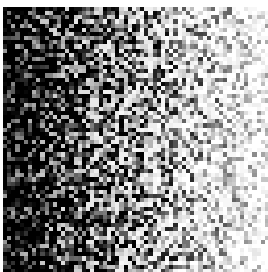
Dichte der Stufenfunktion ($w = \frac{1}{2}$): $\frac{dS_{linsum}(x)}{dx}$



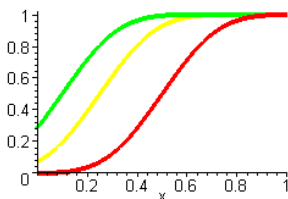
Stufenfunktion $S_{linsum}(x)$



Nach Anwendung von $S_{linsum}(x)$



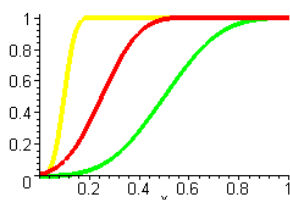
Problematisch an den weicheren Stufenfunktionen ist, daß für Intensitätswerte am Rand des Intervalls $[0..1]$ nach Sättigung im Rasterisierer nicht mehr der lineare Zusammenhang $I = \text{Mittelwert}(S(x))$ gilt. Insbesondere sind die Mittelwerte 0 und 1 nicht mehr möglich!



Darstellung von $S_{linsum}(x - Intensitaet)$ für Intensitäten von 0.5, 0.25 und 0.1

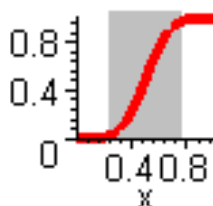
Beheben läßt sich dies, indem die Intensität von dem Intervall $[0..1]$ auf das Intervall $[-\frac{w}{2}..1 + \frac{w}{2}]$ abgebildet wird. Danach sind Mittelwerte von 0 und 1 wieder möglich, es besteht aber weiterhin kein linearer Zusammenhang zwischen ursprünglicher Intensität und dem resultierenden Mittelwert mehr.

Ein adaptiver Korrekturansatz, bei dem die Breite der Stufenfunktion auf $\frac{2 * \min(Intensitaet, 1 - Intensitaet)}$ beschränkt wird, stellt auch den linearen Zusammenhang wieder her. Allerdings erfolgt dies auf Kosten einer nicht mehr konstanten Streuverteilung, was allerdings nur bei weiten Verteilungen nachteilig ist.



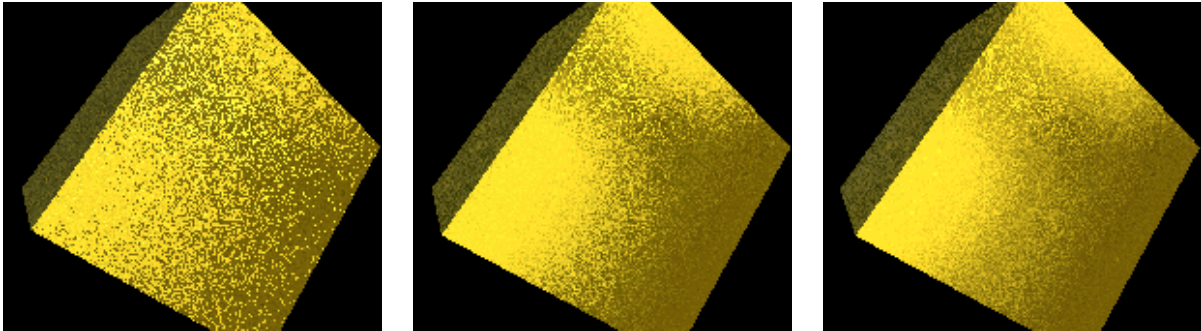
Darstellung von $S_{linsum}(x - Intensitaet)$ für dieselben Intensitäten nach passender Skalierung der Breite

Auf heutigen Bildschirmen ist die Bildauflösung um ein vielfaches geringer als die mit einem Airbrush minimal mögliche Pigmenttropfengröße. Es ist also zweckmäßig, von einer teilweise homogenen Pigmentverteilung auf den Pixeln auszugehen. Dies entspricht Zufallswerten gleichen Erwartungswerts, aber geringerer Varianz bei Bildung der Differenz x . Die Breite der Stufenfunktion kann zugleich zur Simulation unterschiedlich rauher Oberflächen herangezogen werden. Wird die Varianz darüber hinaus mit der Entfernung moduliert, erhält man den Effekt von mit zunehmender Entfernung schwächer werdender Details.

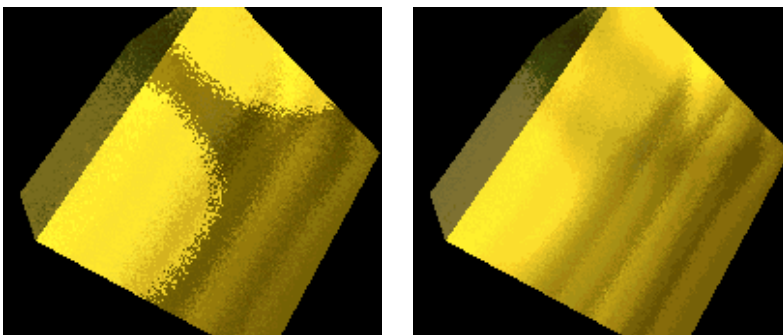


Bei geringerer Varianz überstreicht x einen entsprechend kleineren Teil des Definitionsbereichs von $S_{linsum}(x - Intensitaet)$

Je nach gewählter Breite der Stufenfunktion und Varianz der Zufallswerte ergeben sich somit Graphiken, die im Grenzfall bis hin zu photorealistischen Techniken oder Cartoonstil reichen:



Verschiedene Stufenfunktionsbreiten bei hoher Varianz



Verschiedene Stufenfunktionsbreiten bei geringer Varianz

5.3 Verwendung der gewonnenen Intensitätsverteilung

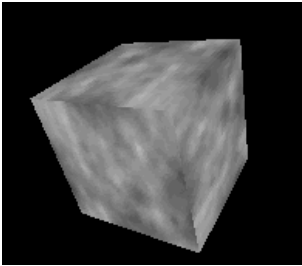
Die im Airbrush eingesetzten Pigmente mischen sich meist deckend oder subtraktiv, während in der Computergraphik eine additive Farbmischung bevorzugt wird. Da hier nur der resultierende Farbauftrag, nicht aber sein Zustandekommen modelliert wird, stellt dies keine Einschränkung dar. Es ist allerdings darauf zu achten, daß die beschriebene Modellierung stets nur skalare Werte liefern kann. Somit muß bei der Beleuchtungsberechnung als Erstes die skalare Intensität der einzelnen Beleuchtungskomponenten ermittelt und auf diesen getrennt der Sprühvorgang angewendet werden. Erst dann können die so erhaltenen lokalen Intensitäten mit den dazugehörigen Farbwerten multipliziert und aufsummiert werden. Andernfalls würden sich innerhalb ein- und desselben simulierten Pigmentauftrags unterschiedliche Farbwerte ergeben.

5.4 Animationsfähigkeit

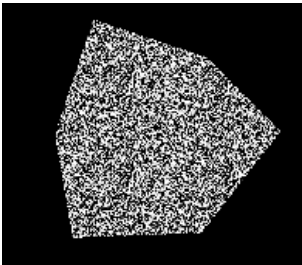
Bildraumbasierte Verfahren haben meist den Nachteil, daß bei bewegten Objekten die Abhängigkeit des Verfahrens von der Position im Bildraum sichtbar wird. Der Effekt ähnelt der Betrachtung durch eine angerauhte Glasscheibe und wird auch als „Shower Door Effect“ bezeichnet[12]. Um ihn zu vermeiden, wird jedes Objekt mit einer Modulationstextur versehen, welche kontinuierliche, richtungsunabhängige Verläufe zwischen 0 und 1 verhält. Derartige Texturen können z.B. durch stark gefilterte Perlin-Noise-Texturen oder überlagerte Sinusfunktionen realisiert werden. Der Wert dieser Textur an einem Punkt wird zu dem Zufallswert im Bildraum modulo 1 hinzuaddiert:

$$\text{Zufallswert}_{\text{animiert}} = \text{frac}(\text{Zufallswert} + \text{Modulationswert})$$

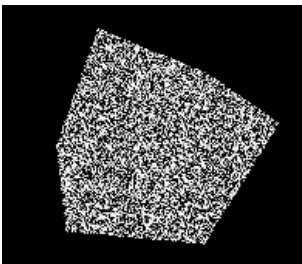
Auf diese Weise ändert der Pigmentierungsgrad eines Pixels, die Stärke der Änderung ist dabei direkt vom Ausmaß der Objektbewegung abhängig und vermittelt den Eindruck, als ob das Bild mit dem Airbrush überarbeitet werden würde. Als angenehmen Nebeneffekt verringern sich auch die Regelmäßigkeiten, welche durch die Wiederholung der Zufallstextur entstehen.



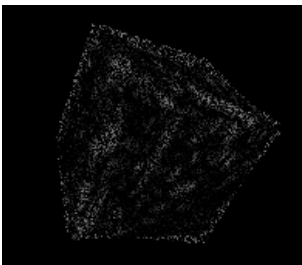
Würfel mit Modulationstextur auf der Oberfläche



Sprüheffekt mit Modulation auf einem rein ambient beleuchteten Würfel



Derselbe Würfel, leicht gedreht



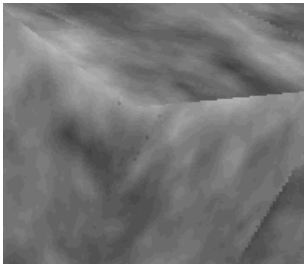
Die Differenz beider Bilder verdeutlicht die Auswirkung auf animierte Bilder

Die Texturkoordinaten für die Modulationstextur können oftmals direkt aus bestehenden Texturkoordinaten eines Objekts übernommen werden. Andernfalls können geeignete Koordinaten durch Betrachtung des an der Objekt Oberfläche gespiegelten Sichtvektors eines im Objektraum positionierten Betrachters gewonnen werden. Hierbei treten zwar an Kanten Sprünge in den Texturkoordinaten auf, die beim Rendern aber ohnehin durch die Intensitätssprünge in der Beleuchtung des Objekts überdeckt werden.

Kritischer ist jedoch, daß bei sehr kleinen Sichtwinkeln auch die Änderung der resultierenden Texturkoordinaten pro Pixel sehr klein wird und in Folge die Effektivität der Modulation sinkt. In diesen Fällen sollte die Auflösung der Modulationstextur dynamisch als Funktion der Texturkoordinatenänderung pro Pixel erhöht werden.



Modulationstextur nach Zoom auf eine Ecke des obigen Würfels



Dieselbe Szene mit dynamisch angepaßten Modulationstexturkoordinaten

5.5 Nachbildung von Schraffurlinien

Polierte und gebürstete Oberflächen werden im Airbrush häufig durch feine Liniestücke in Bearbeitungsrichtung dargestellt. Die Richtung der Linien kann als 2D-Vektoren auf der Oberfläche spezifiziert werden, welche anschließend in die ebenfalls zweidimensionale Bildebene projiziert werden. Die Darstellung der Linien ist somit nichts anderes als die Visualisierung eines 2D-Vektorfelds, wobei das Aussehen nur von Richtung der Linien und der Bildauflösung abhängen soll, nicht aber von Lage, Betrag oder Vorzeichen der Vektoren.

Ein geeignetes Verfahren stellt das LIC(Line Integral Convolution)-Verfahren dar, bei dem ein symmetrischer eindimensionaler Filterkern auf eine zweidimensionale Zufallstextur angewendet wird (man vergleiche auch mit [11]). Als Orientierung des eindimensionalen Filters in der zweidimensionalen Ebene wird dabei der jeweils lokal darzustellende Vektor verwendet[13].

Als Ausgangstextur wird die schon für die Intensitätsstreuung verwendete Zufallstextur verwendet. Um den Filter möglichst effizient zu realisieren, sollte einerseits jeder Abtastpunkt einen anderen Zufallswert ermitteln, andererseits sollte der Filterkern auf das kleinstmögliche Umfeld eines Abtastpunktes beschränkt sein. Der größtmögliche Abstand zweier Pixel tritt entlang der Diagonalen auf und beträgt somit $\sqrt{2} * Pixelgroesse$.

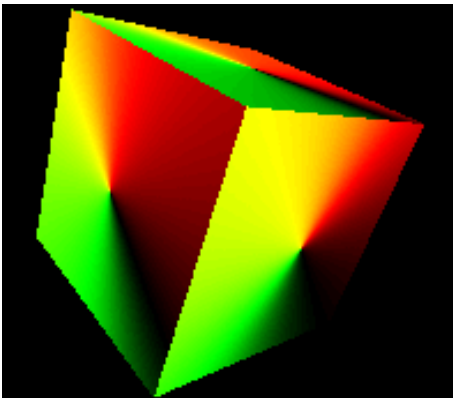
Leider ergibt die Summe I mehrerer gleichverteiler Zufallswerte i keinen gleichverteilten Zufallswert mehr. Stattdessen ergibt sich für die ungewichtete Summe aus n gleichverteilten Zufallswerten mit Werten zwischen 0 und 1 folgende Zufallsdichte[14]:

$$D(I) = \frac{1}{2^{(n-1)!}} * \sum_{k=0}^n ((-1)^k * \binom{n}{k} * (I - k)^{n-1} * \text{Signum}(I - k))$$

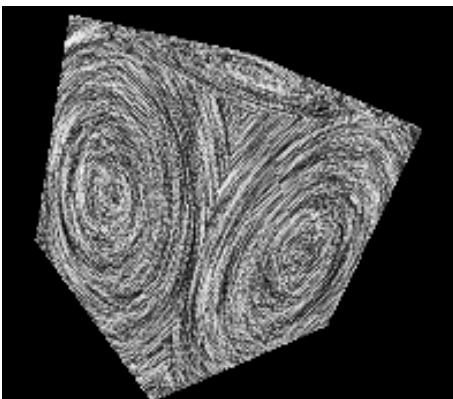
Gleichverteilte Werte i erhält man, indem man die dazugehörige Zufallsfunktion $S(x)$ heranzieht: $i = S(I)$. Für eine feste Anzahl von Abtastpunkten kann diese vorberechnet und in einer Textur für das Intervall $[0..n]$ abgelegt werden. Die so gewonnenen Werte werden wie bei dem schon beschriebenen Verfahren eingesetzt.

Ein kontinuierlicher Übergang zwischen Zufallspunkten und Zufallslinien ist durch lineare Interpolation beider Techniken möglich. Dasselbe Ergebnis wäre zwar auch über eine entsprechend stärkere Gewichtung des mittleren Filterpunktes möglich, würde aber zugleich die Gestalt von $S(x)$ von der Gewichtung abhängig machen.

Farbcodierte Vektoren $(r, g, b) = \frac{(x, y, z) + 1}{2}$ in Welt- und in Bildkoordinaten



Errechnete Schraffur



Kapitel 6

Wahl eines geeigneten Beleuchtungsmodells

Prinzipiell ist für den Airbrusheffekt jedes Beleuchtungsmodell geeignet, solange es sich in einen Ambient-Diffusen und einen spekularen Term zerlegen läßt und für den spekularen Term reflektierte Sichtvektoren berechnen kann. Die Verteilung dieser Vektoren soll dabei deren Anteil an der spekularen Reflexion entsprechen. Aufgrund der geringen Eindringtiefe genügt die Betrachtung der Strahlenoptik für nahezu alle im Alltag vorkommenden Metalloberflächen.

Im Folgenden wird eine Variante des Phong-Modells vorgestellt, welches an das von Ashikmin und Shirley in [15] präsentierte Modell angelehnt ist und einen guten Kompromiß aus erzielter Qualität zu aufzuwendender Leistung und leichter Handhabbarkeit darstellt. Ebenso wie dieses setzt es sich aus einem spekularen und einem diffusen Term zusammen und unterstützt sowohl einen winkelabhängigen Fresnelterm zur Farbkorrektur als auch die Nachbildung anisotroper Oberflächen, verwendet aber einen einfacheren Term für diffuse Reflexionen. Darüber hinaus wird es für den Einsatz in einer rein lokalen Beleuchtungsberechnung zusätzlich um einen ambienten Term sowie eine Environmentmap ergänzt.

6.1 Diffuser Anteil

Der diffuse Anteil berücksichtigt den durch Mehrfachreflexion an Mikrofacetten der Oberfläche in beliebige Richtungen gestreuten Anteil des Lichts. Er setzt sich aus einem Lambertschen Term für unter dem Winkel α gegen die Oberflächennormale gerichtet einfallendes Licht sowie einem konstanten Term für das aus der Umgebung einfallende Licht zusammen:

$$\begin{aligned} \text{DiffusesLicht} = & \\ \text{DiffuseFarbe} * (\sum_{\text{Lichtquellen}} (\cos(\alpha) * \text{LichtintensitaetLichtquelle}(\cos(\alpha)))) & \\ + \text{LichtintensitaetAmbient} & \end{aligned}$$

Im klassischen Lambertschen Modell ist $\mathbf{LichtintensitaetLichtquelle}(\cos(\alpha))$ für positive $\cos(\alpha)$ eine positive Konstante und ansonsten gleich 0. Es bietet sich aber an, hier ähnlich dem Cool-to-Warm-Shading mit einem Gegenlicht (mit positivem oder negativem Vorzeichen) zu arbeiten, um den plastischen Eindruck zu erhöhen[16]. Somit wird $\mathbf{LichtintensitaetLichtquelle}(\cos(\alpha)) = \mathbf{LichtintensitaetLichtquelle}$ für $\cos(\alpha) > 0$ und ansonsten gleich der Lichtintensität des Gegenlichts.

Für die DiffuseFarbe gilt aufgrund von Mehrfachreflexion:

DiffuseFarbe \approx

Mittelwert(SpekulareFarbe^{AnzahlReflexionen} * AnteilMehrfachreflexionen

und somit ist sie je nach Oberflächenbeschaffenheit wesentlich stärker gefärbt und schwächer als die spekulare Farbe. Bei Verwendung von Gegenlichtern sollte zudem die ambiente Intensität angepaßt werden, um die Änderung der Gesamthelligkeit durch das Gegenlicht zu kompensieren. Aufgrund der geringen Intensität des diffusen Anteils kann für Renderingzwecke auf die Berücksichtigung des Fresnelterms sowie Abschwächung bei hoher spekularer Reflexionsintensität aufgrund der Energieerhaltung verzichtet werden.

6.2 Spekularer Anteil

Der Spekulare Anteil wird nach dem Phong oder dem Blinn-Phong-Modell berechnet:

SpekularesLicht =

SpekulareFarbe * ($\sum_{\mathbf{Lichtquellen}} (\mathbf{Saturate}(\cos(\mathbf{Spekularwinkel}))^{\mathbf{Spekularexponent}} * c * \mathbf{LichtintensitaetLichtquelle})$)

Im Phong-Modell ist der Spekularwinkel gleich dem Winkel zwischen Sichtstrahl und dem reflektiertem Lichtstrahl, welcher mit dem Winkel zwischen einfallendem Licht und reflektierten Sichtvektor identisch ist. Besser geeignet [18], aber für jede Lichtquelle getrennt zu berechnen, ist der Spekularwinkel als Winkel zwischen dem Halfway-Vektor und der Oberflächennormale, wie er im Blinn-Phong-Modell verwendet wird.

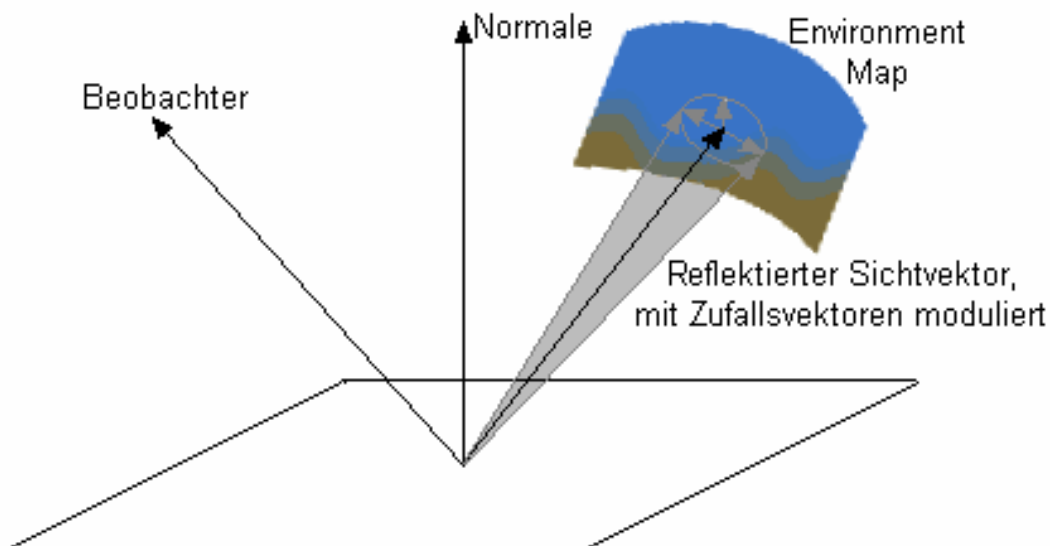
Dies liegt in der physikalischen Bedeutung des Halfway-Vektors: Dieser gibt die Normale einer unter den gegebenen Licht- und Sichtrichtungen perfekt reflektierenden Oberfläche an. Betrachtet man nun eine aus Mikrofacetten bestehende Oberfläche mit dem Spekularexponenten als Maß für deren Glattheit, so gibt der Term $\mathbf{cos}(\mathbf{Spekularwinkel})^{\mathbf{Spekularexponent}}$ mit dem Normalisierungsfaktor c für die Energieerhaltung den Anteil der reflektierenden Mikrofacetten an der Gesamtoberfläche an. Im Gegensatz dazu verfügt das normale Phong-Modell über keine direkte Interpretation für eine aus Mikrofacetten zusammengesetzte Oberfläche.

Die spekulare Farbe entspricht direkt der Farbe des vom Metall reflektierten weißen Lichts und kann für unpolarisiertes Licht als Mittelwert der Fresnelterme für polarisiertes Licht bei entsprechender Frequenz berechnet werden. Der Normalisierungsfaktor c ergibt sich zu $c = \frac{\text{Spekularexponent}+1}{2\pi}$ und kann mit der spekularen Farbe vormultipliziert werden, so daß hierfür keine gesonderte Konstante nötig ist.

Im BRDF-Modell von Ashikmin und Shirley wird der Korrekturfaktor c noch mit $\frac{1}{\max(\text{Lichtvektor} \cdot \text{Normale}, \text{Sichtvektor} \cdot \text{Normale})}$ multipliziert. Dieser kann aber bei ohnehin rein lokaler Beleuchtungsberechnung wie im Blinn-Phong-Modell durch den nicht-reziproken Term $\frac{1}{\text{Lichtvektor} \cdot \text{Normale}}$ ersetzt werden, welcher bei Anwendung der BRDF durch den ebensogroßen Faktor für einfallendes Licht kompensiert wird und somit entfällt.

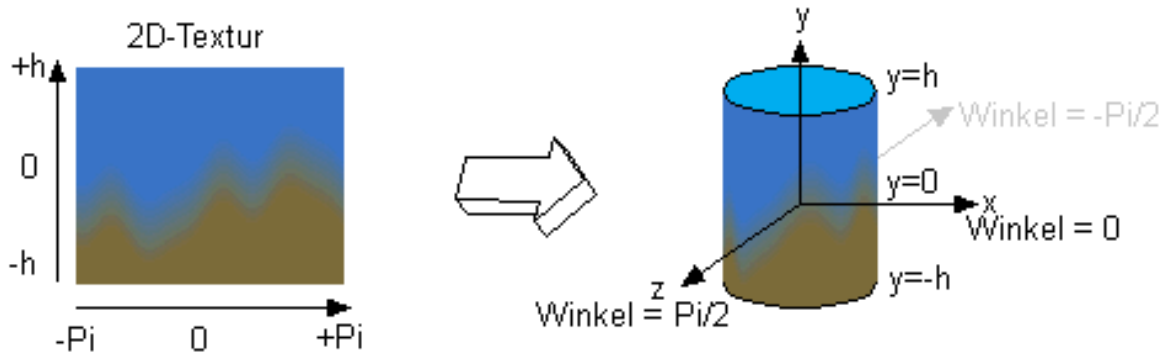
6.3 Environment Mapping

Für die Nachbildung von Spiegelungen wird das einfallende Licht einer Environment Map entnommen. Um verschieden raue Oberflächen zu simulieren wird dazu der reflektierte Sichtvektor mit zufälligen Vektoren so moduliert, daß der endgültige Reflexionsvektor derselben Verteilung folgt wie der Intensitätsverlauf eines spekulär reflektierten Lichtes aus derselben Richtung. Hierbei genügt es, nur ein einzigen Wert pro Pixel aus der Environment-map auszulesen, da das dadurch erzeugte Muster sich optisch kaum von der Streuung des simulierten Airbrushes unterscheidet. Voraussetzung dazu ist allerdings eine ausreichend tiefpaßgefilterte Environment Map, da starke Änderungen der Farbe innerhalb eines Winkels in Größenordnung des Streukegels des modulierten Reflexionsvektors zu einem unruhigen Gesamtbild führen würden.



Die einfachste Realisierung einer passenden Environment Map stellen CubeMaps dar. Allerdings ist die Generierung solcher Environment Maps vergleichsweise umständlich. Da sich eine stark vereinfachte Umgebung als Kombination je einer einfarbigen Boden- und Himmelshemisphäre deuten läßt, bietet es sich an, den polaren Winkel der Reflexionsrichtung zu verwenden, um zwischen der Farbe des Himmels und der des Bodens zu interpolieren. Um mehr Variation zu erhalten, kann in Horizontnähe die Farbe zusätzlich in Abhängigkeit vom azimuthalen Winkel variiert werden.

Eine besonders für stark glänzende Oberflächen interessante Variation bietet sich, wenn man dazu eine gewöhnliche, zweidimensionale Textur verwendet, welche z.B. einem Panoramaphoto entstammen kann. Damit ergibt sich für das Aussehen der Environmentmap ein Zylinder, dessen Mantel die 2D-Textur bildet und dessen Ober- und Unterseiten der Himmels- bzw. Bodenfarbe entsprechen. Für die Koordinaten der Textur ergeben sich somit $u = \frac{1}{2} + \frac{\arctan(\frac{x}{z})}{2\pi}$ und $v = \frac{1}{2} \frac{\sqrt{1-y^2}}{h}$ für einen normalisierten reflektierten Vektor (x,y,z) und Texturkoordinaten u,v innerhalb $[0..1]$ auf der Mantelfläche sowie v außerhalb $[0..1]$ für die Ober- und Unterseite.



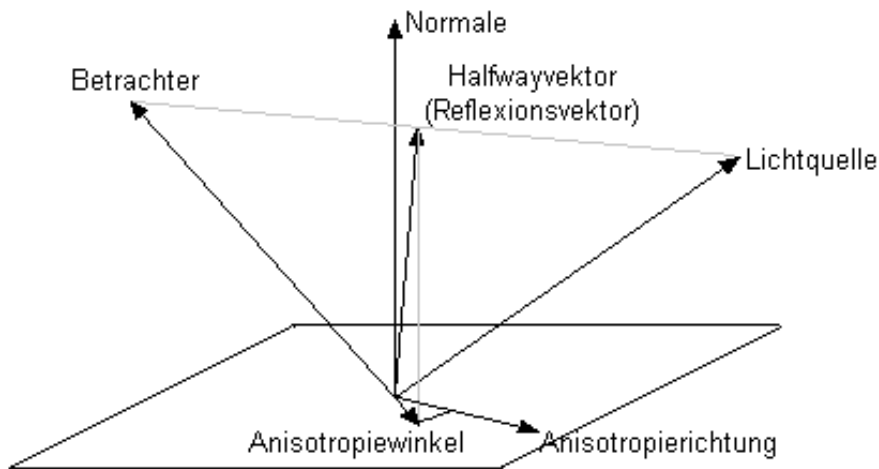
6.4 Anisotropie

Einfache Anisotropie einer Oberfläche kann als Richtung maximaler Rauigkeit mit senkrecht dazu liegender Richtung minimaler Rauigkeit modelliert werden[15]. Der dazugehörige Spekularexponent ergibt sich als lineare Interpolation zwischen einem minimalen und einem maximalen Spekularexponenten in Abhängigkeit des Winkels zwischen dem auf die Oberfläche projizierten Halfwayvektor und der Anisotropierichtung der Oberflächenebene:

$$\text{Spekularexponent} = \text{SpekularexponentMin} * \text{Orientierung} + \text{SpekularexponentMax} * (1 - \text{Orientierung})$$

mit

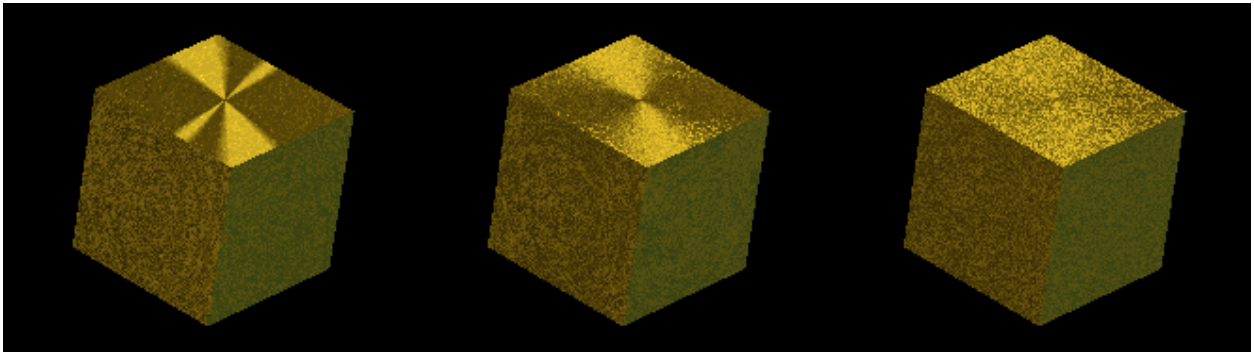
$$\text{Orientierung} = \cos(\text{Anisotropiewinkel})^2 = \frac{(\text{Halfwayvektor} \bullet \text{Anisotropieachse})^2}{1 - (\text{Halfwayvektor} \bullet \text{Normale})^2}$$



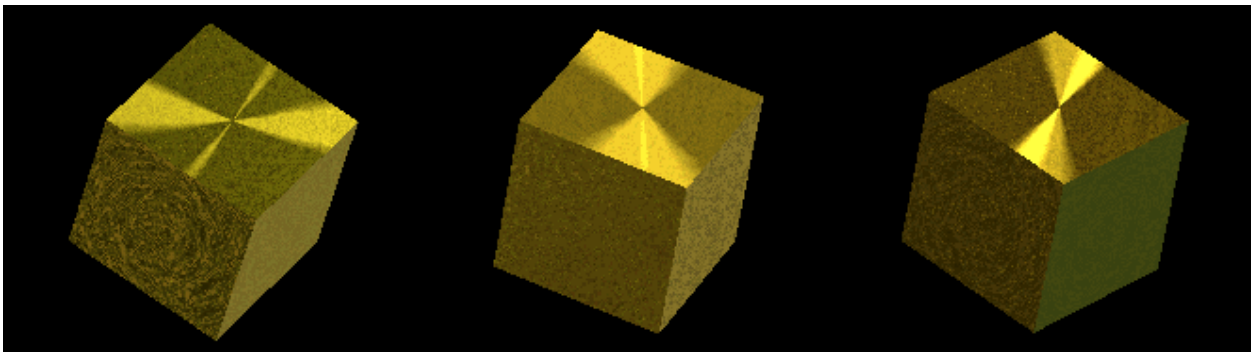
Auch hier gibt es eine anschauliche physikalische Interpretation, indem man analog zum isotropen Blinn-Phong-Modell den Halfwayvektor als Normale einer reflektierenden Mikrofacette nimmt. Die Verteilung der Mikrofacettennormalen hängt jetzt allerdings auch von deren Richtung parallel zur Oberfläche ab, und entsprechend wird auch der Spekularexponent als Maß der Glattheit zu einer Funktion der Orientierung des Halfwayvektors parallel zur Oberfläche.

Die Normalisierungskonstante c kann auch hier wieder vorberechnet werden und ergibt sich zu:

$$c = \frac{\sqrt{(\text{SpekularexponentMin} + 1) * (\text{SpekularexponentMax} + 1)}}{2\pi}$$

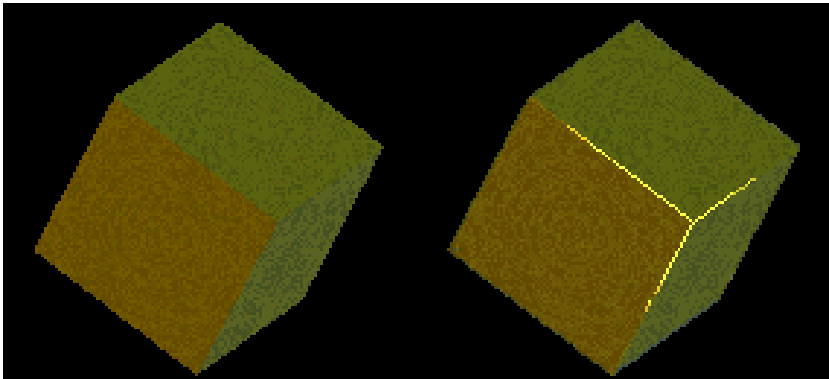


Verschiedene Verhältnisse der Spekularexponenten von 1:100, 1:4, 1:1.5



Auswirkungen verschiedener Betrachterpositionen auf die anisotrope Reflektion

Ein Spezialfall anisotroper Oberflächen sind Kanten: An diesen verteilen sich die Oberflächennormalen im gesamten Bereich zwischen den Normalen der angrenzenden Seiten. Somit können durch Linien mit entsprechend niedrigeren Spekularexponenten für senkrecht zur Kante einfallendes Licht die meist vernachlässigten Reflexionen an Kanten nachgebildet werden. Als Normale der Kante wird der Mittelwert der angrenzenden Seitennormalen verwendet.

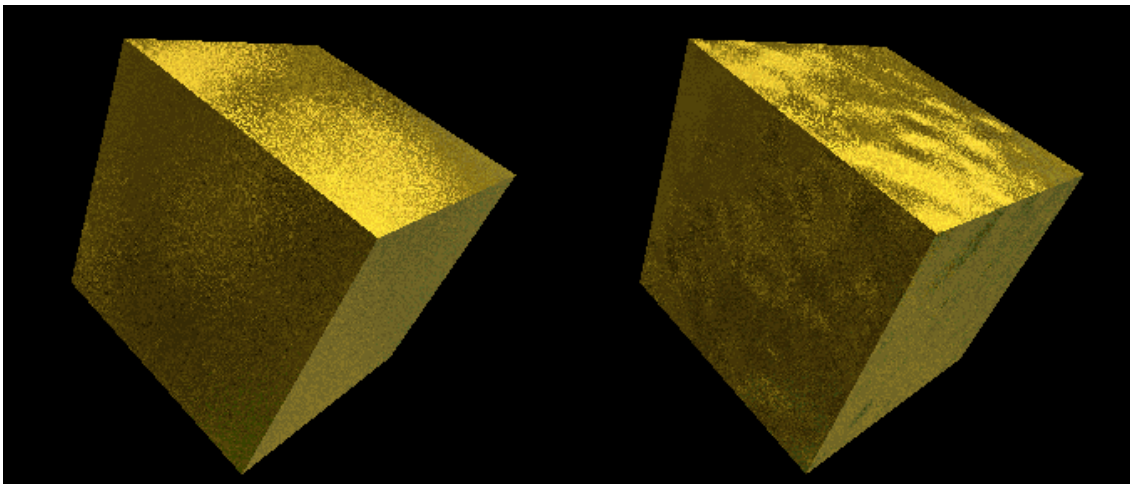


Würfel ohne und mit nachgezeichneten Kanten

An für sich sollte auch die Verteilung des reflektierten Sichtvektors für Spiegelungen von der Anisotropie beeinflusst werden. Allerdings ist die Auflösung mit nur einem Abtastwert pro Pixel und vorgefilterter Environmentmap in der Praxis so grob, daß die Auswirkungen der Anisotropie kaum sichtbar sind und somit vernachlässigt werden können. Auch bei Airbrusharbeiten werden anisotrope Effekte üblicherweise nur über die Verteilung der hellsten Lichtreflexe imitiert.

6.5 Bumpmaps

Nachdem mit der Normalen und der dazu orthogonalen Anisotropierichtungsebene ein fertiges kartesisches Koordinatensystem im Tangentialraum festgelegt ist liegt es nahe, dieses auch für Änderungen der Oberflächennormale zur Simulation von Unebenheiten auf Pixelebene zu verwenden. Als geeignete Bumpmap-Verfahren bieten sich insbesondere Offset Maps (Textur enthält einen Vektor, der zur Oberflächennormale hinzuaddiert wird) und Rotation Maps (Textur beschreibt Drehung der Oberflächennormale) an[17]. Die durch Filterung bei Texturverkleinerung verschwindenden Unterschiede der Normalen können dabei nicht nur in einem entsprechend angepaßten Spekularexponenten aufgefangen werden[18,19], sondern auch durch entsprechende Änderung der Varianz der Zufallswerte weiter zur Darstellung einer unebenen Oberfläche im Airbrushstil verwendet werden. Geeignet sind Bumpmaps vor allem für Normalenänderungen niedriger Amplitude und Frequenz, da signifikante Änderungen in der Höhe im Airbrush besser durch direkte Nachbildung über präzise Techniken wie Masken wiedergegeben werden, während kleinräumige Strukturen direkt durch die Art des Pigmentauftrags simuliert werden. Als generelle Grenzen von Bumpmaps treten zudem bei großen Amplituden und flachen Betrachtungswinkeln der flache Texturcharakter unangenehm in Erscheinung, wohingegen sich feine Texturdetails bei Texturfilterung schnell gegenseitig aufheben.



Würfel mit ebener und unebener Oberfläche via Bumpmap

Kapitel 7

Erweiterte Lichteffekte

7.1 Glanzlichter

Reflexionen, die den gewählten Dynamikbereich überschreiten, strahlen in ihre Umgebung aus. Zur Nachbildung dieses Effekts wird ein an HDR-Rendering[20] angelehntes Verfahren mit zwei Durchgängen angewendet:

Im ersten Schritt wird eine Textur erstellt, welche nur die Lichtintensität der spekularen Beleuchtungskomponente des Bildes enthält. Diese kann je nach Stärke der Überstrahlung mit einem multiplikativen Faktor moduliert sein.

Im zweiten Schritt werden im Bildraum die gefilterten Werte aus der spekularen Textur ausgelesen und in weißer Farbe additiv dargestellt. Zur vereinfachten Nachbildung des Sprüheffektes genügt es, je nach Position im Bildraum einige exemplarische Werte innerhalb des Filterkerns zu entnehmen, so daß kein vollständiger und aufwendiger Filterkern nötig ist. Die Lesepositionen \mathbf{X}_t aus der spekularen Textur kann ggf. um den Betrag b in Richtung des Lichtes (in Bildebene projiziert) verschoben sein und ergibt sich für den Pixel am Ort \mathbf{X}_p im i -ten Durchgang zu:

$$\mathbf{X}_t = \mathbf{X}_p + 2\text{D-Zufallsvektor}(i, \mathbf{X}_p) + b * \text{Lichtrichtung}$$

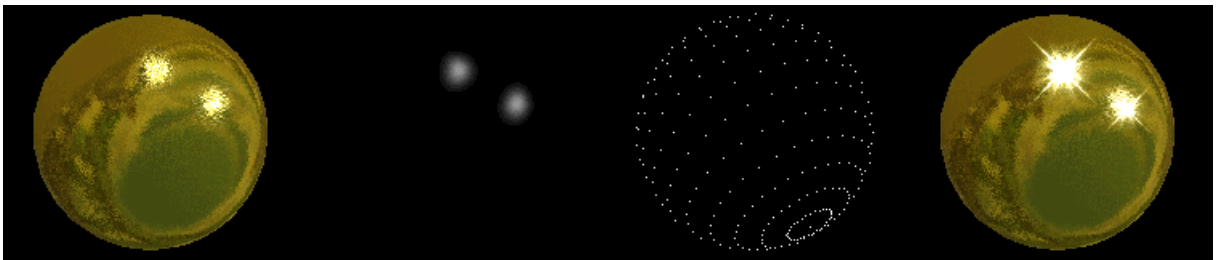


Teekessel normal gerendert, spekulare Intensität, nach Filterung, resultierendes Bild

7.2 Sterneffekte

Sternförmige Lichtreflexe könnten zwar ebenfalls in einem zweiten Durchgang durch Filterung aus der spekularen Textur gewonnen werden, erfordern aber einen in der Praxis nicht vertretbaren Rechenaufwand. Sinnvoller ist es, die spekulare Textur nur an einigen exemplarischen Punkten auszuwerten und entsprechend der aufgefundenen Intensität die sternförmigen Effekte zu platzieren. Zu beachten ist, daß alle Sterne dieselbe Orientierung und Gestalt haben, einzig die Helligkeit und Größe darf variieren.

Passende Testpunkte können als 3D-Vertices von Hand den zu zeichnenden Objekten hinzugefügt werden, oder sie können direkt als zufällige Auswahl aus den bestehenden Vertices eines Objekts entnommen werden. Angezeigt werden sie idealerweise als Billboards bzw. Pointsprites.



Kugel mit zwei Lichtquellen, deren spekulare Intensität, für den Effekt getestete Orte, fertiger Effekt



Zur Darstellung verwendetes Sprite

Kapitel 8

Implementierung

8.1 Verwendete API-Bestandteile

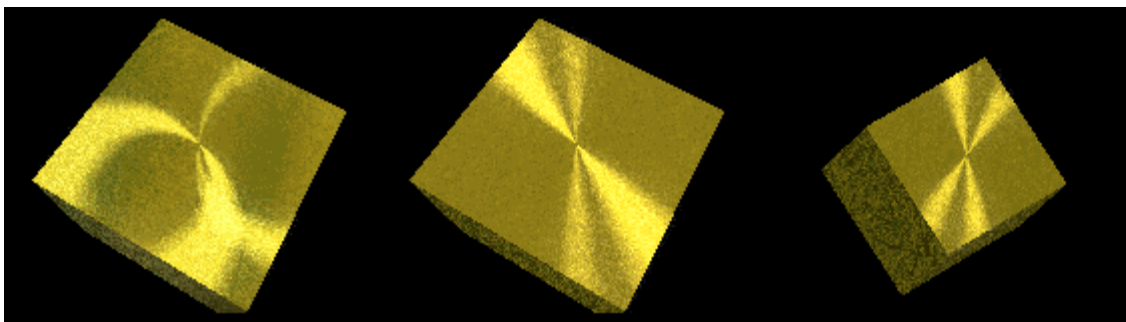
Realisiert wurde das beschriebene Verfahren in Direct3D 9 mit 2.0x-Shadern in HLSL. Diese sind die Minimalanforderung, um das gesamte Rendering (bis auf die zusätzlichen Effekte) in einem einzigen Pass zu erledigen und sind auf praktisch allen aktuellen Standardgraphikkarten verfügbar.

Die gesamte Beleuchtungsberechnung findet aufgrund der zu erwartenden hohen Spekularexponenten komplett im Pixelshader statt. Ein dazugehöriger Vertexshader ist dennoch nötig, um die lokalen Parameter an diesen übergeben zu können. Davon abgesehen entspricht er praktisch der Standardfunktionalität der fest vorgegebenen Graphikpipeline von DirectX 9. Im Pixelshader sind prinzipiell beliebig viele Lichtquellen möglich, deren Parameter denen der vorgegebenen Lichtquellen in D3D 9 entsprechen.

Der Dithervorgang zur Sprühsimulation ist streng genommen nur für einen linearen Zusammenhang zwischen berechneter Farbintensität und auf dem Bildschirm sichtbarer Helligkeit korrekt. Andernfalls müssen die Werte per Gammakorrektur konvertiert werden. In D3D 9 wird aber innerhalb der Shader ein linearer Zusammenhang vorausgesetzt[21], so daß keine weiteren Vorkehrungen hierzu nötig sind.

8.2 VertexShader

Im Vertexshader werden sämtliche Vektoren in Weltkoordinaten transformiert, so daß im Pixelshader keine weiteren Transformationen für die dort hinzukommende EnvironmentMap und die Lichtquellen mehr nötig sind. Zu beachten ist, daß die Koordinaten für die Zufallstextur im Bildraum schon hier vorberechnet werden (abgesehen von der perspektivischen Division), da in 2.0x-Pixelshadern kein Zugriff auf die Position eines Pixels möglich ist. Falls nötig, werden geeignete Koordinaten für die Modulationstextur ebenfalls generiert. Das Flag `LOCALVIEWER` verhält sich wie der D3D-Renderstatus `D3DRS_LOCALVIEWER` und ist nicht nur für orthogonale Projektion sinnvoll, sondern kann auch durch einen übergroßen Sichtwinkel entstehende Effekte vermeiden.



Anisotroper Würfel mit `LOCALVIEWER=true` und Sichtwinkel von 45 Grad, Würfel mit `LOCALVIEWER=false` und Sichtwinkel von 45 Grad sowie Würfel mit `LOCALVIEWER=true` und Sichtwinkel von 2 Grad

8.3 PixelShader

Im Pixelshader erfolgt die komplette Beleuchtungsberechnung. Hier sind 2.0a bzw. 2.0b-Shader im Vorteil, da sie durch eingebaute Kontrollstrukturen sowie höherer Zahl an erlaubten Anweisungen eine unterschiedliche Zahl an Lichtquellen ohne Neuübersetzung des Shaders und in einem Durchgang unterstützen. Der Shader im Detail:

8.3.1 Parameter für Lichtquellen und Umgebungslicht

In der Struktur `Lichtquelle` werden die Parameter einer jeden Lichtquelle abgelegt, wobei die Werte `Ambient`, `Diffus`, `Spekular`, `Typ` und `Vektor` weitgehend denen der vordefinierten Standard-D3D-Lichtquellen entsprechen und um eine Gegenlichtfarbe `GegenlichtDiffus` ergänzt werden. Sämtliche Farbwerte liegen als RGBA-Farben vor. Im Gegensatz zu D3D werden Spotlights und entfernungsabhängige Abschwächung nicht unterstützt, können aber durch Erweiterung des Shaders mit berücksichtigt werden.

Zusätzlich kann eine einzelne Environment Map im RGBA-Format verwendet werden, deren Farbwert mit `LichtfarbeEnv` multipliziert wird, um die Farbe der Umgebung dynamisch anpassen zu können. Die richtungsabhängige Farbe aus der Environment Map wird über die Funktion `EnvFarbe()` ausgelesen. Für diese Funktion gibt es mehrere Möglichkeiten. Am einfachsten zu implementieren ist die Verwendung vorberechneter CubeMaps, gespeichert in der Textur `EnvMap`:

```
float4 Envmap_Cube(float3 Richtung)
{
    return texCUBE(EnvMap,Richtung);
}
```

Eine andere Variante verzichtet auf Texturen und berechnet anhand einiger Parameter eine einfache Umgebung. Benötigt werden als benutzerdefinierte Parameter die beiden Farbwerte, die gewünschte Position des Horizonts relativ zur y-Komponente eines normalisierten Richtungsvektors und als Eingabe einen normalisierten Richtungsvektor. Über eine Summe mehrere Sinusfunktionen, skaliert mit dem Parameter `Horizontvarianz`, wird die Höhe zusätzlich zwecks Imitation einer einfachen Landschaft moduliert. Die Schärfe des Boden-Himmel-Farbübergangs wird über `Horizontverlauf` eingestellt:

```
//Parameter
float4 HorizontfarbeOben; //RGB-Farbe des Himmels
float4 HorizontfarbeUnten; //RGB-Farbe des Bodens
float Horizonthoehe;
float Horizontverlauf;
float Horizontvarianz;
```

```
float4 EnvmapParamHorizont(float3 Richtung)
{
    //ein möglicher Horizontverlauf auf Basis von Sinusfunktionen
    float Horizontstruktur = Horizontvarianz * (0.25 * sin(5. * Richtung.x * PI)
                                                +0.5 * cos(3. * Richtung.x * PI)
                                                +1.0 * sin(1. * Richtung.x * PI));

    //interpoliert zwischen 2 Farben
    float Horizontposition = 0.5 - (0.5 * (Richtung.y + 1.) - Horizonthoehe)
                                * Horizontverlauf;
    float Farbmischung = saturate(Horizontposition + Horizontstruktur);
    return lerp(HorizontfarbeOben,HorizontfarbeUnten,Farbmischung);
}
```

Die beschriebene zylindrische Environment Map erlaubt es, bestehende (Panorama-)Bilder als Environmentmap zu verwenden. Die Pixel am oberen und am unteren Bildrand sollten jeweils in einer Farbe vorliegen, da diese auch für die Ober- und Unterseite des Zylinders mittels Clamp-Adressmodus verwendet werden. Um eine lückenlose Interpolation am Rand der Textur zu ermöglichen ist für die horizontale Texturkoordinate Wraparound zu aktivieren. Mit einem normalisierten Richtungsvektor als Eingabe liefert folgende Funktion den passenden Farbwert:

```
float XzuY; //Parameter: Seitenverhältnis der eingesetzten Textur

float4 Envmap_Zylinder(float3 Richtung)
{
    //Addressmodi: x=wrap, y=clamp
    float2 TexKoord;
    TexKoord.x = 1. - (atan2(Richtung.x,Richtung.z) / (2 * PI));
    TexKoord.y = 0.5 * (1. - XzuY * Richtung.y / sqrt((1. - Richtung.y * Richtung.y)));
    return tex2D(EnvMap, TexKoord);
}
```

Das zylindrische Mapping läßt sich vereinfachen, wenn die Ausgangstextur nur eine Hälfte des Zylinders abdeckt und für die andere Hälfte gespiegelt wird. Verwendet man eine passend vorverzernte Textur, genügt für das Auslesen der Environment Map folgende einfache Funktion:

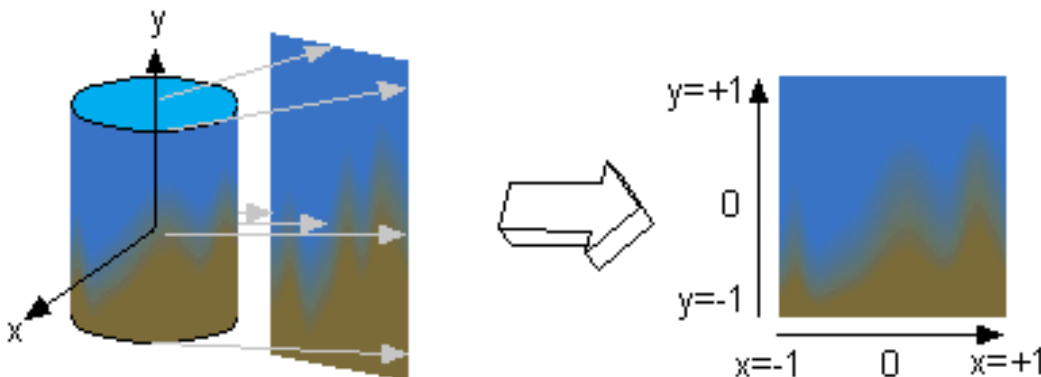
```
float4 Envmap_Halbzyylinder(float3 Richtung)
{
    //Addressmodi: x=mirror, y=clamp
    float2 TexKoord = 0.5 * (1 + Richtung);
    return tex2D(EnvMap, TexKoord);
}
```

Dazugehöriger Shader zum Erzeugen einer passend verzerrten Textur als Rendertarget:

```
const float PI=3.14159265359;
sampler envmap;
float XzuY; //Seitenverhältnis der eingesetzten Textur

float4 Halbzylindertextur(float2 Texpos : TEXCOORD0 ) : COLOR
{
    //Achtung: Eingabeposition in Rendertargetkoordinaten
    //liegt in [-1..1]^2, nicht in [0..1]^2 !
    //berechne z-Komponente des Richtungsvektors anhand der Texturkoordinaten
    //der verzerrten Textur (Richtungsvektor ist dabei nicht normalisiert):
    float Texposz=sqrt(1. - Texpos.x * Texpos.x);
    //berechne halbzylindrisches Mapping
    float2 envmap_uv;
    envmap_uv.x = (atan2(Texpos.x, Texposz) / PI) + 0.5;
    //halbe Höhe aufgrund Spiegelung der Ausgangstextur:
    envmap_uv.y = 0.5 * (1. + 0.5 * XzuY * Texpos.y / sqrt((1. - Texpos.y * Texpos.y)));
    return tex2D(envmap, envmap_uv);
}
```

Erweiterte Shader, mit der auch Bilder ohne konstante Farbe am oberen und unteren Rand verwendet werden können, finden sich im Anhang.



8.3.2 Parameter des Materials

Die Farbwerte `SpekulareFarbe` für einfach und `DiffuseFarbe` für mehrfach reflektiertes Licht ergeben sich direkt aus der nachzubildenden Metalloberfläche, der `Spekularexponent` wird entsprechend dem Blinn-Phong-Modell der Oberflächenrauigkeit angepaßt.

Um die interaktive Handhabung zu vereinfachen berechnen sich die anisotropen `Spekularexponenten` zu `Spekularexponent/Isotropie` sowie `Spekularexponent*Isotropie`,

ein Wert von 1 für **Isotropie** ergibt somit eine isotrop reflektierende Oberfläche.

Zur Darstellung komplexerer Anisotropiestrukturen kann der Tangentenvektor mittels einer Anisotropietextur in der Ebene gedreht werden: Die Textur speichert dabei direkt den Wert der gedrehten Tangente, ausgedrückt im zweidimensionalen Tangente-Binormale-Koordinatensystem.

Als Bumpmap wird eine Offset Map verwendet, so daß die zum Variieren der Normale verwendeten Offsetvektoren in derselben zweidimensionalen Form wie die gedrehten Tangentenvektoren der Anisotropie gespeichert werden können. Weil alle Offsetvektoren somit in der Oberflächenebene liegen, sind die resultierende Normalen nicht mehr normalisiert. Dies ist allerdings kein Nachteil, da aufgrund der metalltypisch hohen Spekularexponenten sowieso normalisiert werden sollte, um Interpolationsartefakte der Bumpmap zu vermeiden. Der zweite Nachteil von derartigen Offsetmaps, der eingeschränkte Bereich resultierender Vektoren (maximal 45 Grad Abweichung zur ursprünglichen Normalen bei Offsetwerten innerhalb $[-1..1]$ in Tangenten- bzw. Binormalenrichtung) ist ebenfalls unkritisch: Dies reicht selbst für bei Wellblech oder Riffelblech üblichen Werten aus, größere Abweichungen sind aus schon genannten Gründen ohnehin nicht empfehlenswert. Sowohl bei der Anisotropie als auch der Bumpmap ist darauf zu achten, daß sich bei Texturwiederholung die Vektoren durch Interpolation an den Kanten gegenseitig aufheben können.

8.3.3 Parameter des Sprüheffekts

Die Parameter **Streuung** und **Verlauf** beeinflussen Varianz der Zufallswerte und die Breite der Stufenfunktion bei Berechnung der lokalen Intensität. Für den Einsatz im Environmentmapping und den Glanzlichtern enthält die Zufallstextur pro Texel gleich 4 verschiedene Zufallswerte. Für die Modulationstextur genügt dagegen ein Wert pro Texel. Über die **z_Daempfung** kann die Abnahme der Varianz in Abhängigkeit von der z-Position des gerenderten Objekts variiert werden, um eine entfernungsbedingte Abnahme an sichtbaren Details nachzubilden.

Als Anzahl der Abtastpunkte für die Schraffurlinien hat sich in der Praxis ein Wert von 5 als Minimum für einen gut sichtbaren Effekt herausgestellt. Mehr Abtastpunkte verbessern die Qualität, führen aber schnell zu Engpässen bei der Übersetzung von 2.0/2.0a-Shadern mangels dynamischer Flußkontrolle bei zugleich stark begrenzter Länge. Die **Relinearisierer-1D**-Textur dient zur Wiederherstellung gleichverteilter Zufallswerte nach Anwendung des Schraffurfilters.

Für die Sprühfunktion `Spray(Intensitaet, Spruehparameter)` gibt es mehrere Möglichkeiten. Die einfachste ist mit konstanter Korrektur und linearem Verlauf zwischen 0 und 1:

```
float Spray(float Intensitaet, float Spruehparameter)
{
    float LokaleIntensitaet = Intensitaet - Spruehparameter;
    float Korrektur = 1. - Streuung;
    return saturate((LokaleIntensitaet - 0.5) / Verlauf * Korrektur + 0.5);
}
```

Dieselbe Funktion mit adaptiver Korrektur:

```
float Spray(float Intensitaet, float Spruehparameter)
{
    float LokaleIntensitaet = Intensitaet - Spruehparameter;
    float Korrektur = 2. * saturate(min(Intensitaet, 1. - Intensitaet))
        * (1. - Streuung);
    return saturate(0.5 + ((LokaleIntensitaet - 0.5) / (Verlauf * Korrektur)));
}
```

Die bei Annahme mehrerer Schichten aus gleichverteilten Sprühtropfen entstehende S-Kurve als Zufallsfunktion läßt sich näherungsweise über die in der Shadersprache schon enthaltene `Smoothstep`-Funktion darstellen. Bei dieser handelt es sich praktisch um eine Hermite-Interpolation, welche einen Eingabewert x auf das Intervall $[a..b]$ begrenzt und anschließend den Wert $Smoothstep(a, b, x) = 3y^2 - 2y^3$ mit $y = \frac{x-a}{b-a}$ zurückgibt. Die entsprechenden Varianten für konstante und adaptive Korrektur sind damit:

```
float Spray(float Intensitaet, float Spruehparameter)
{
    //konstante Korrektur
    float LokaleIntensitaet = Intensitaet - Spruehparameter;
    float Korrektur = 1. - Streuung;
    return smoothstep(-Verlauf * Korrektur, Verlauf * Korrektur,
        2. * (LokaleIntensitaet - 0.5));
}

float Spray(float Intensitaet, float Spruehparameter)
{
    //adaptive Korrektur
    float LokaleIntensitaet = Intensitaet - Spruehparameter;
    float Korrektur = 2. * saturate(min(Intensitaet, 1. - Intensitaet))
        * (1 - Streuung);
    return smoothstep(-Verlauf * Korrektur, Verlauf * Korrektur,
        LokaleIntensitaet - 0.5);
}
```

8.3.4 Vorbereitung interpolierter Werte

Zuerst werden die auf Vertexbasis interpolierten Werte `TexNormale`, `Tangente`, `Binormale`, `Textur` (Texturkoordinaten) und `TexturMod` (Modulationstexturkoordinaten) vorbereitet und ggf. normalisiert. Anschließend werden die Bump- sowie Anisotropietexturen ausgewertet und die Anisotropietangente in Bildkoordinaten vorbereitet:

```
//BumpMap
float2 Offset = tex2D(BumpMap,Textur);
float3 Normale = normalize(TexNormale + Offset.x * Tangente + Offset.y * Binormale);

//Anisotropiehauptachse (Tangente der Schraffur)
float2 Rotation = normalize(tex2D(AnisoMap,Textur).xy);
float3 Anisotropieachse = Rotation.x * Tangente + Rotation.y * Binormale;
//Anisotropietangente projiziert und skaliert
float3 Anisotropietangente = mul(matViewProjection,float4(Anisotropieachse.xyz,0));
float2 AnisotropietangenteBild = 1. / (TexGroesse / sqrt(2.))
    * normalize(Anisotropietangente.xy
        - Anisotropieachse.z
        * normalize(BildPosition.xy));
```

8.3.5 Berechnung und Modulation des Sprühparameters

Für die Variation des Sprüheffekts wird der Z-Wert des Pixels verwendet:

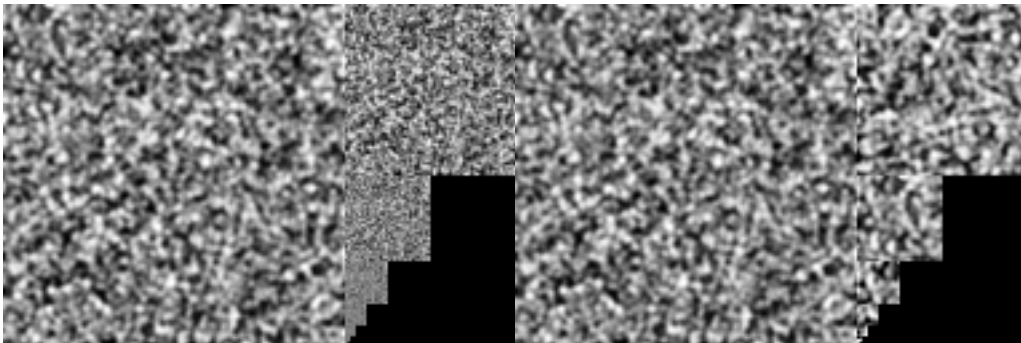
`LokaleStreuung = Streuung*(1.-BildPosition.z/BildPosition.w*z_Daempfung)`

Dieser muß, ebenso wie die Koordinaten `TexPos` für die Position des Pixels im Bildraum, noch perspektivisch dividiert werden. Der entsprechende Zufallswert eines Pixels wird anschließend mit dem Wert `ZufallswertAniso` aus dem LIC-Verfahren gemischt. Letzterer ergibt sich als relinearisierte Summe über die benachbarten Zufallswerte in Richtung der projizierten Anisotropietangente:

```
//berechne linear gefilterten Wert
float4 ZufallswertAniso = Zufallswert;
for(int i = 0;i < ((Abtastpunkte - 1) / 2);i++)
{
    TexPosP-= AnisotropietangenteBild;
    TexPosN+= AnisotropietangenteBild;
    ZufallswertAniso+= tex2D(Zufallswerte, TexPosP);
    ZufallswertAniso+= tex2D(Zufallswerte, TexPosN);
}
//erzeuge gleichverteilte Werte
ZufallswertAniso = tex2D(Relinearisierer, ZufallswertAniso / Abtastpunkte);
//Mische Schraffur mit normalem Zufallswert
Zufallswert = lerp(ZufallswertAniso, Zufallswert, Isotropie);
```

Der fertige Sprühparameter eines Pixels berechnet sich dann zu `float4 Spruehparameter = LokaleStreuung*(0.5-frac(Modulation+Zufallswert));`.

Um zu verhindern, daß die Modulationstextur bei starker Skalierung ineffizient wird kann man Mipmapping einsetzen, wobei die größte Mipmap so gewählt wird, daß sie bei der größten zu erwartenden Skalierung gute Werte liefert. Die darunterliegenden Mipmaps werden nicht durch Filterung gewonnen, sondern enthalten entsprechend kleinere Ausschnitte der ursprünglichen Textur mit maximalem Kontrast, so daß die Auflösung praktisch konstant bleibt. Lineare Filterung zwischen den Mipmaps ist zur Vermeidung von sichtbaren Mipmapstufen bei Animation nötig.



Vergleich von Mipmaps durch bilineare Filterung mit Mipmaps aus kontrastkorrigierten Ausschnitten

Alternativ können die Texturkoordinaten dynamisch über deren Gradienten angepaßt werden. Anders als bei Mipmaps wird hier allerdings nicht bei Verdoppelung der Texturkoordinatenänderung auf die nächste Stufe umgestiegen, sondern ein einfacher zu implementierender linearer Zusammenhang zwischen Gradient und gewähltem Texturkoordinatenmultiplikator verwendet:

```
//skaliere Modulationstextur anhand des sichtbaren Bereichs
//für annähernd verzerrungsfreies Texturmapping genügt Analyse einer Komponente
float TexModFaktor = (1. / ModTexBreite) / length(ddx(TexturMod));
//berechne Koordinaten der grob aufgelösten Stufe:
float2 TexMod1 = TexturMod * (TexModFaktor - frac(TexModFaktor));
//berechne Koordinaten der fein aufgelösten Stufe:
float2 TexMod2 = TexturMod * (TexModFaktor - frac(TexModFaktor) + 1.);
//interpoliere zwischen den Stufen:
float Modulation = lerp(tex2D(Modulierer, TexMod1), tex2D(Modulierer, TexMod2),
                        frac(TexModFaktor));
```

8.3.6 Beleuchtungsberechnung

Da schon im Vertexshader alle zur Beleuchtung nötigen Vertexdaten in Weltkoordinaten transformiert werden sind zur Beleuchtungsberechnung keine weiteren Matrizenmultiplikationen mehr nötig. Nur die noch fehlende Berechnung des Lichtvektors für punktförmige Lichtquellen erfolgt im Pixelshader, so daß die maximale Anzahl der Lichtquellen nur durch die Fähigkeiten der Grafikhardware (maximale Anzahl an Instruktionen und freier Register für Konstanten) begrenzt wird.

Für die Materialfarbe wird der Fresnelterm mittels der Schlickschen Näherung zu **Fresnelfarbe(Einfallswinkel) = Materialfarbe + (1 - Materialfarbe) * (1 - Einfallswinkel⁵)** berechnet, die Materialfarbe gilt dabei für senkrecht einfallendes Licht. Auf die Fresnelkorrektur des diffusen Terms wird aufgrund seines bei Metallen geringen Wertes im Vergleich zum spekularen Term verzichtet.

Bei der Implementierung ist wie zuvor beschrieben darauf zu achten, zuerst die skalaren Intensitäten einer Lichtquelle zu berechnen und anschließend mit der Sprühfunktion die lokale Intensität einer Lichtquelle zu bestimmen. Erst dann kann die Summe über die Produkte der resultierenden Intensitäten mit ihren dazugehörigen Material- und der Lichtfarben gebildet und im `Pixel` abgelegt werden.

Für das restliche aus der Umgebung stammende Licht wird eine `EnvironmentMap`-Funktion eingesetzt, die mittels des reflektierten Sichtvektors ausgelesen wird. Da es sich hierbei um eine spekulare Reflexion handelt, wird der ermittelte Lichtwert ebenfalls mit der Fresnel-Materialfarbe multipliziert. Für den Streukegel der Reflexion wird eine leicht berechenbare Näherung angewandt:

Es wird der Winkel betrachtet, innerhalb dessen ein vorgegebener Anteil der spekularen Reflexion liegt. Dieser bildet sich (bei isotroper Reflexion) auf der durch alle möglichen Reflexionsvektoren gebildeten Halbkugel als Kreis ab. Für hohe Spekularexponenten wird der Winkel klein, so daß der Durchmesser des Kreises mit dem Winkel in erster Näherung identisch ist. Vergleicht man ein Quadrat mit Kantenlänge d und einen Kreises mit Durchmesser d miteinander, so ergibt sich ein Verhältnis von $\frac{\pi}{4} : 1$ (rund 0.79) zwischen Quadrat- und Kreisfläche.

Dies bedeutet, für einen Kreis mit Durchmesser d auf der Halbkugel, welcher rund 79% der spekularen Reflexion umfaßt, besteht dasselbe Verhältnis zwischen spekularer Reflexion innerhalb und außerhalb des Kreises wie für den Inkreis eines Quadrats mit demselbem Mittelpunkt und Kantenlänge d , welches die gesamte spekulare Reflexion aufnehmen würde. Somit kann dieses Quadrat zur Simulation der spekularen Streuung herangezogen werden.

Den Durchmesser d erhält man im Phong-Modell mit Spekularexponent n aus

$$\frac{\pi}{4} = \int_0^{2\pi} \int_0^{\frac{d}{2}} \frac{n+1}{2\pi} * \cos(a)^n * a * da * dw$$

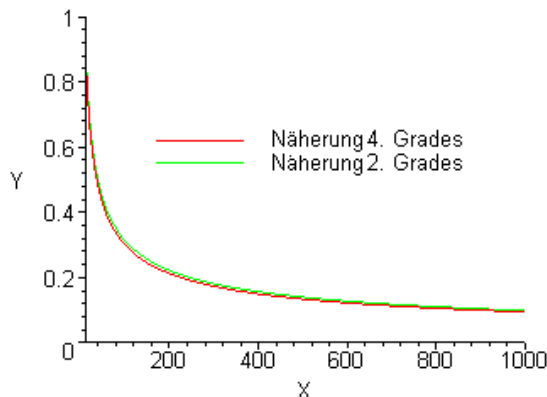
Entwickelt man $\cos(a)^n$ als Reihe, so erhält man in quadratischer Näherung $\cos(a)^n = 1 - \frac{n}{2} * x^2$, und somit

$$\frac{\pi}{4} = -\frac{1}{128}(n+1) * n * d^4 + \frac{1}{8}(n+1) * d^2$$

mit der Lösung

$$d = 2\sqrt{\frac{2n+2+\sqrt{4n^2+8n+4-2\pi n^2-2\pi n}}{(n+1)*n}}$$

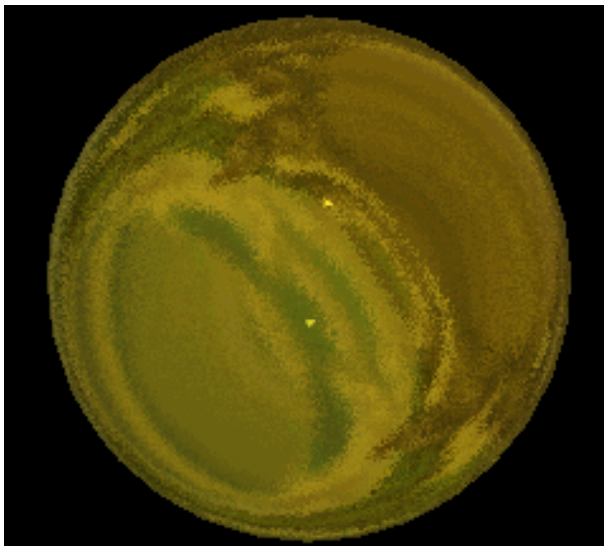
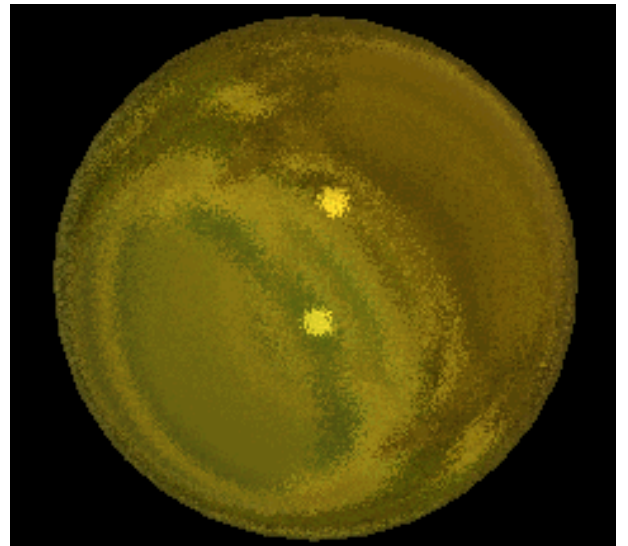
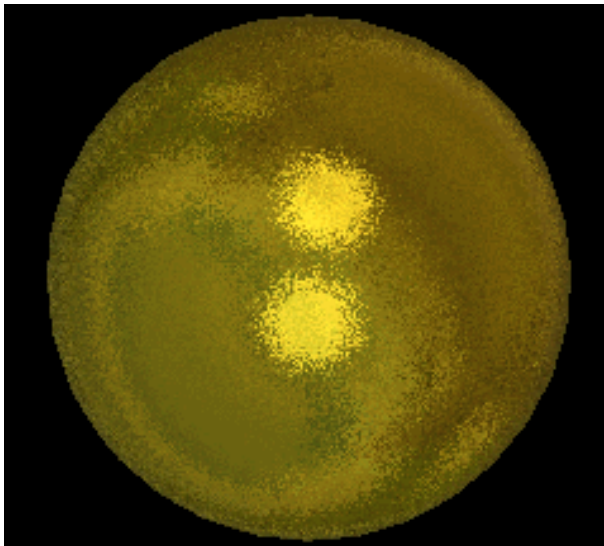
Diese kann für $n \gg 1$ gut über $d = \frac{3}{\sqrt{n}}$ angenähert werden.



Funktionsverlauf für die exakten Lösungen der Cosinus-Reihenentwicklung quadratischer bzw. vierter Ordnung sowie der beschriebenen Näherung. Die Kurven für die exakte Lösung quadratischer Ordnung und der Näherung sind in der Grafik deckungsgleich.

Die so ermittelte Kantenlänge wird direkt zu Skalierung eines Zufallsvektors in $[-1..1]^2$ orthogonal zum Reflexionsvektor verwendet und diesem hinzuaddiert. Eine weitere Vereinfachung besteht darin, einen im Mittel rund 10% längeren Zufallsvektor unter Verzicht auf Orthogonalität aus $[-1..1]^3$ zu entnehmen - die entstehenden Variationen in der Verteilung fallen praktisch nicht auf. Ein passender Filter für die Environment Map sollte so gewählt sein, daß ein Texels in etwa eine Fläche in Größenordnung des Kreisradiuses als Durchmesser abdeckt. Bilinear gefilterte Mipmaps bieten sich somit an. Die Wahl der Mipmapstufe kann direkt über den berechneten Kreisdurchmesser erfolgen oder der Graphikhardware überlassen werden. Letzteres ist zwar implementierungsspezifisch, lieferte aber auf den getesteten Karten in Verbindung mit einem Mipmapbias-Wert von etwa -0.5 vergleichbare Ergebnisse.

```
//Environment Map
//winkelunabhängig, da schon im Fresnelterm berücksichtigt
float IntensitaetEnvmap = 1.;
float Streuweite = saturate(3. / sqrt(Spekularexponent));
float4 RefFarbeEnv = SpekulareFarbeFresnel * LichtfarbeEnv;
float3 EnvreflektVekt = ReflektVekt + Spruehparameter.xyz * Streuweite;
float4 EnvAnteil = RefFarbeEnv * EnvFarbe(EnvreflektVekt) * IntensitaetEnvmap;
Pixel+= EnvAnteil;
```



Kugel mit Environmentmap und zwei parallelen Lichtquellen für Spekularexponenten von 50, 500 und 5000 (ohne Berücksichtigung der energieerhaltenden Konstante)

8.4 Glanzlichter

Die spekulare Intensität wird mit denselben Funktionen wie das normale Bild berechnet. Ihr Wert kann anschließend entweder zusammen mit dem normalen Bild im Alphakanal oder in einer Multielementtextur gespeichert werden. Oder sie wird mit einem separaten, nur die Intensitätsberechnungen enthaltenden Shader in einem zweiten Durchgang in eine separate Textur gerendert. In diesem Fall kann die Auflösung der erzeugten Textur je nach erwünschter Genauigkeit geringer ausfallen. Falls möglich, sollte zusätzlich noch der Wert aus der Modulationstextur für den nachfolgenden Durchgang mit abgespeichert werden.

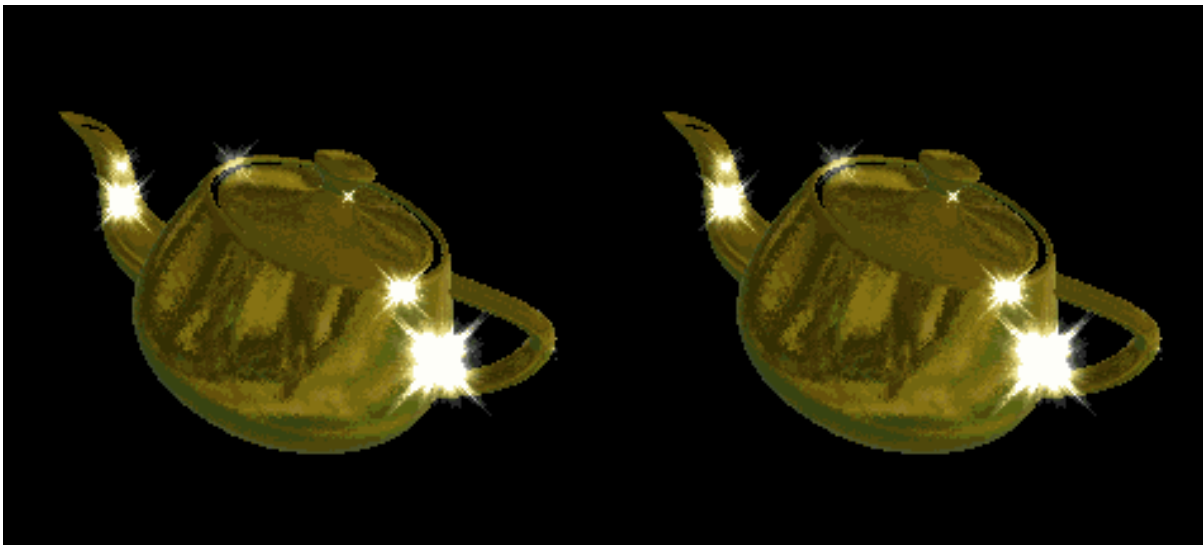
Der zweite Durchlauf läuft ausschließlich im Bildraum ab und benötigt dazu neben der spekularen Textur nur noch ein bildfüllendes Rechteck als Eingabe für den Vertexshader. Für den Offset des Lichteffektes wird die Richtung der ersten Lichtquelle verwendet, da keine genaueren Daten auf Vertexbasis verfügbar sind. Die berechneten Pixel können dann per additivem Alphablending dem Bild hinzugefügt werden.

Der dazugehörige Pixelshader liest pro Pixel 4 Werte im Umkreis **Spitzlichtweite** eines Pixels aus und gewichtet diese anhand ihres Abstands zum Mittelpunkt. Die Position des Effektes kann dabei um **Spitzlichtoffset** in Richtung der hellsten Lichtquelle (oder auch einer anderen Vorzugsrichtung) verschoben werden.

8.5 Sterneffekte

Um die Belastung der Graphikkarte zu verringern, werden nur diejenigen Pointsprites in voller Größe gezeichnet, an denen auch eine entsprechende spekulare Reflexion vorliegt. Da in 2.0x-Vertexshadern leider nicht auf die spekulare Textur zugegriffen werden kann, wird stattdessen pro Vertex die spekulare Intensität nochmals berechnet. Nachteilig ist, daß hier im allgemeinen weder die modifizierte Normale noch die Anisotropierichtung verfügbar ist. Anstatt sie in einem Vorverarbeitungsschritt den Vertexdaten hinzuzufügen, begnügt sich der hier gezeigte Ansatz allerdings damit, auf entsprechende Korrekturen zu verzichten. Der dadurch in Kauf genommene Verlust an potentiellen Sterneffekten hält sich für mäßige Anisotropie und Bumpmaps niedriger Amplitude in vertretbaren Grenzen. Die Leseposition in der spekularen Textur wird über die COLOR-Register übergeben, da die Texturkoordinaten bei der Expansion der Pointsprites zu Rechtecken überschrieben werden. Die berechneten Pixel können dann per additivem Alphablending dem Bild hinzugefügt werden.

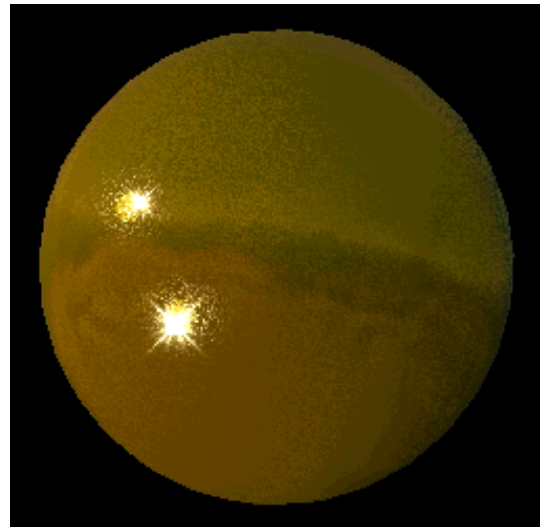
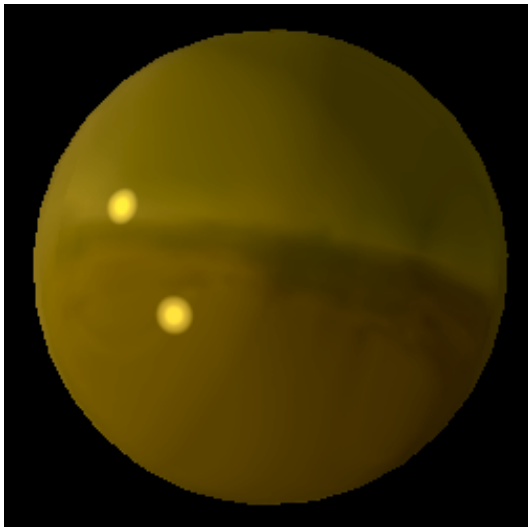
Im Pixelshader wird die Sternentextur nur noch anhand der automatisch generierten Texturkoordinaten ausgelesen und mit dem Wert aus der spekularen Textur multipliziert. Damit ist zugleich die Verdeckung nicht sichtbarer Reflexe erledigt.



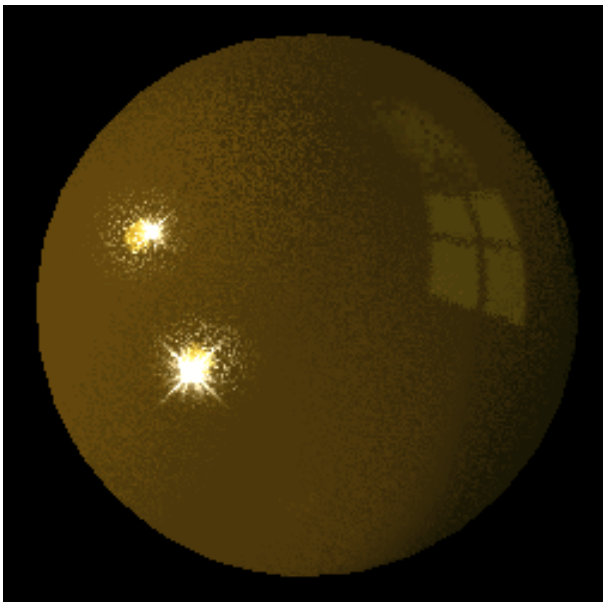
Teekessel mit und ohne spekulare Vorskalierung der Effektsprites. Der einzige Unterschied zeigt sich an dem kleinem Spitzlicht des Henkels. Spekularexponent=533, Isotropie=0.25, Bumpmapvariation unterhalb 5 Grad

Anhang A

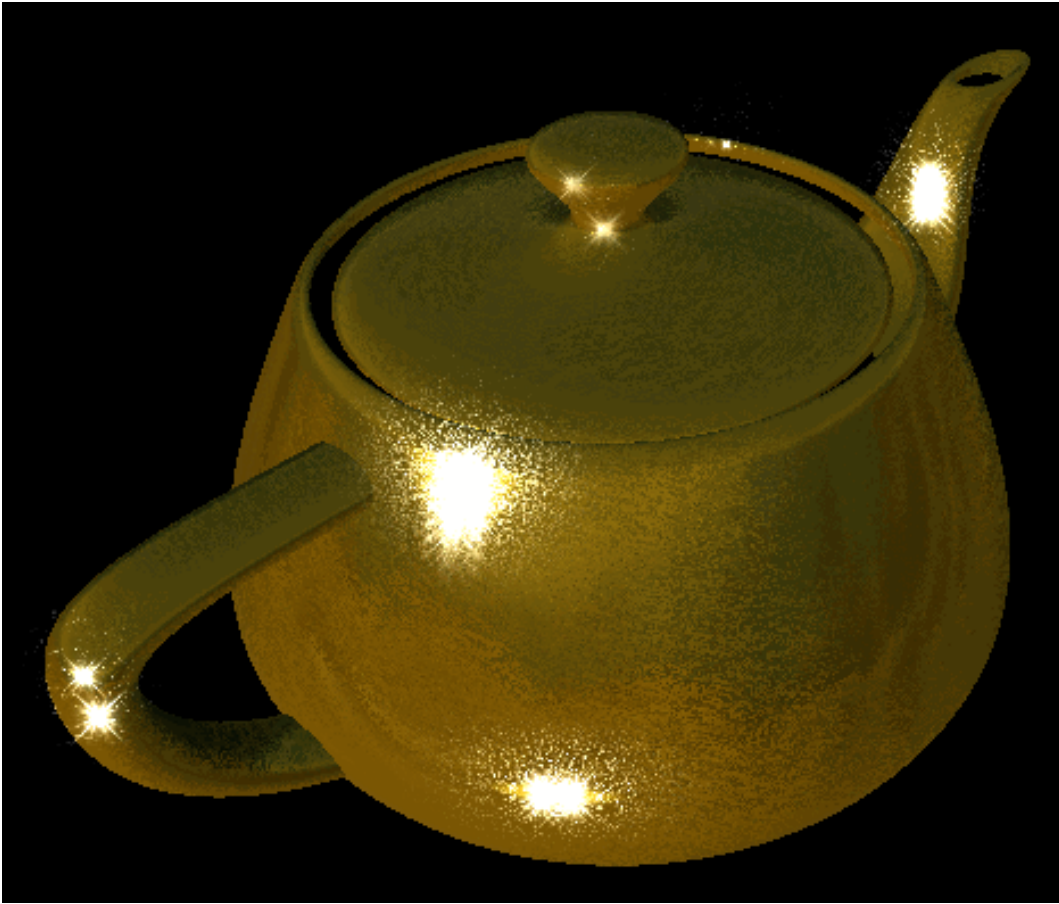
Ergebnisse/Bilder



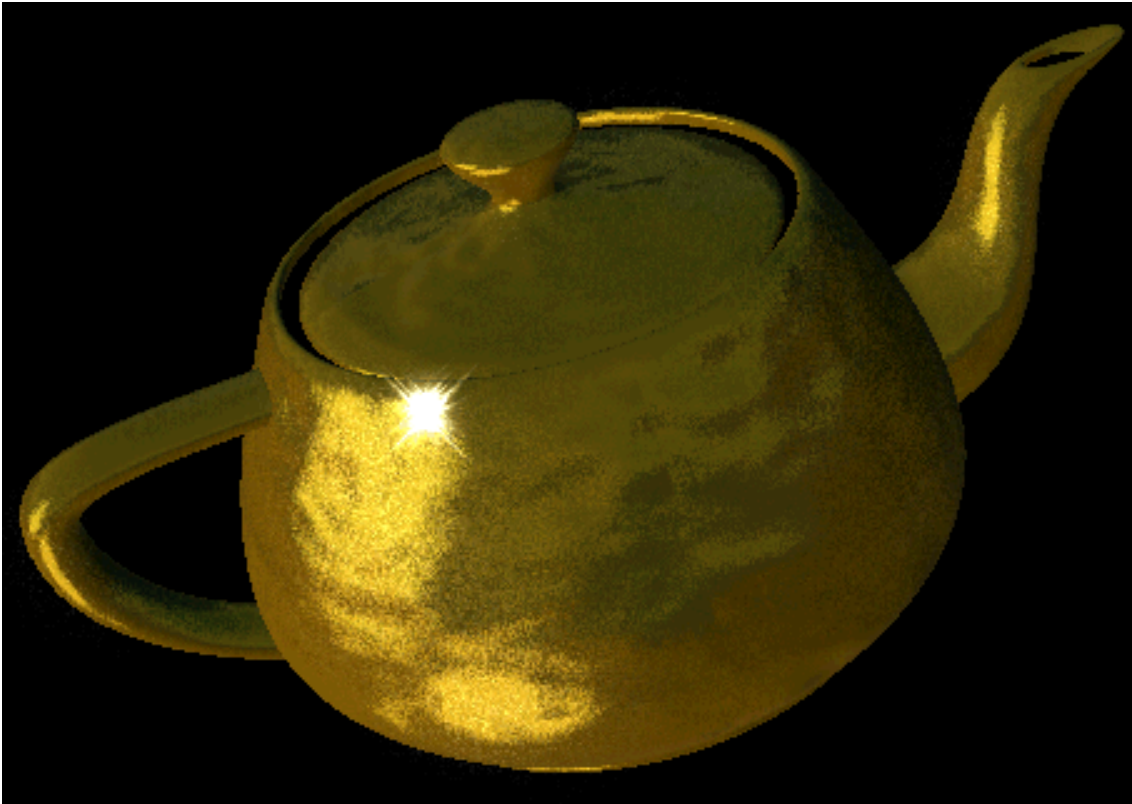
Vergleich einer Kugel mit zwei Lichtquellen dargestellt mit normalem Phong-/Lambertshading und Airbrushshading. (Isotropie = 1, Spekularexponent = 500; Environmentmapfilter: Vorgefilterte Textur für Phongshading, automatischer Mipfilter für Airbrushshading)



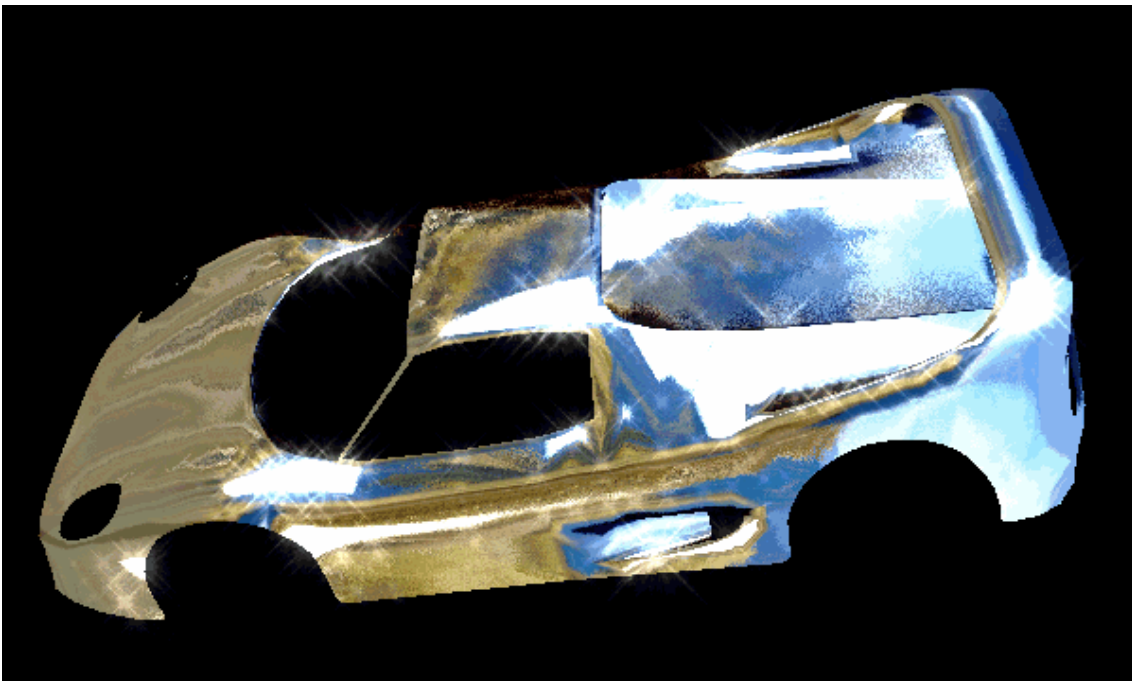
Dieselbe Kugel mit dem vereinfachten Abbild eines Fensters als Environmentmap zur Darstellung von Objekten in Innenräumen



Gebürstete Teekessel aus zwei verschiedenen Materialien



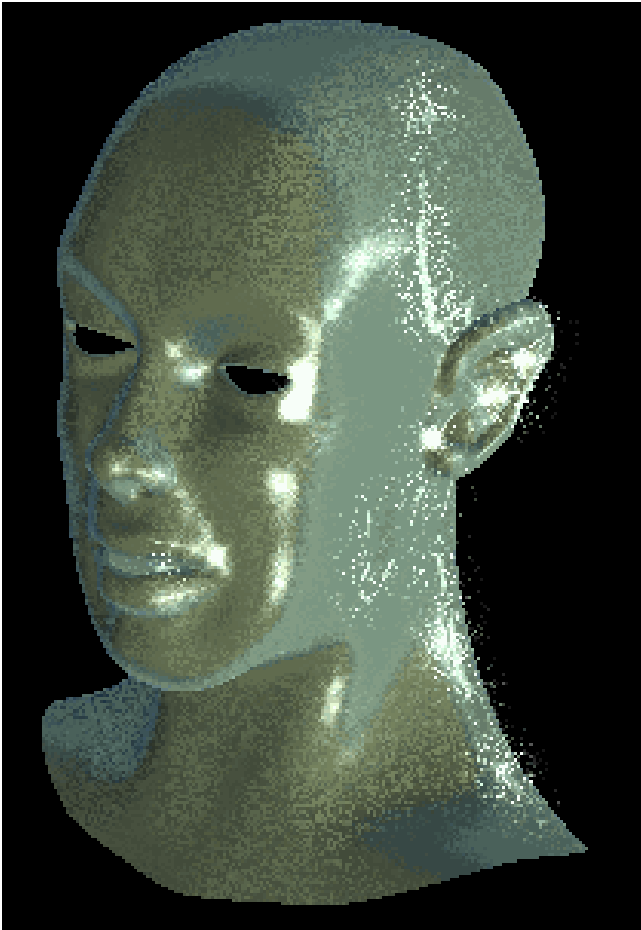
Selbiger Teekessel, verbeult statt gebürstet



Chrom-Effekt mit hartem Verlauf, niedriger Streuung und starkem Gegenlicht

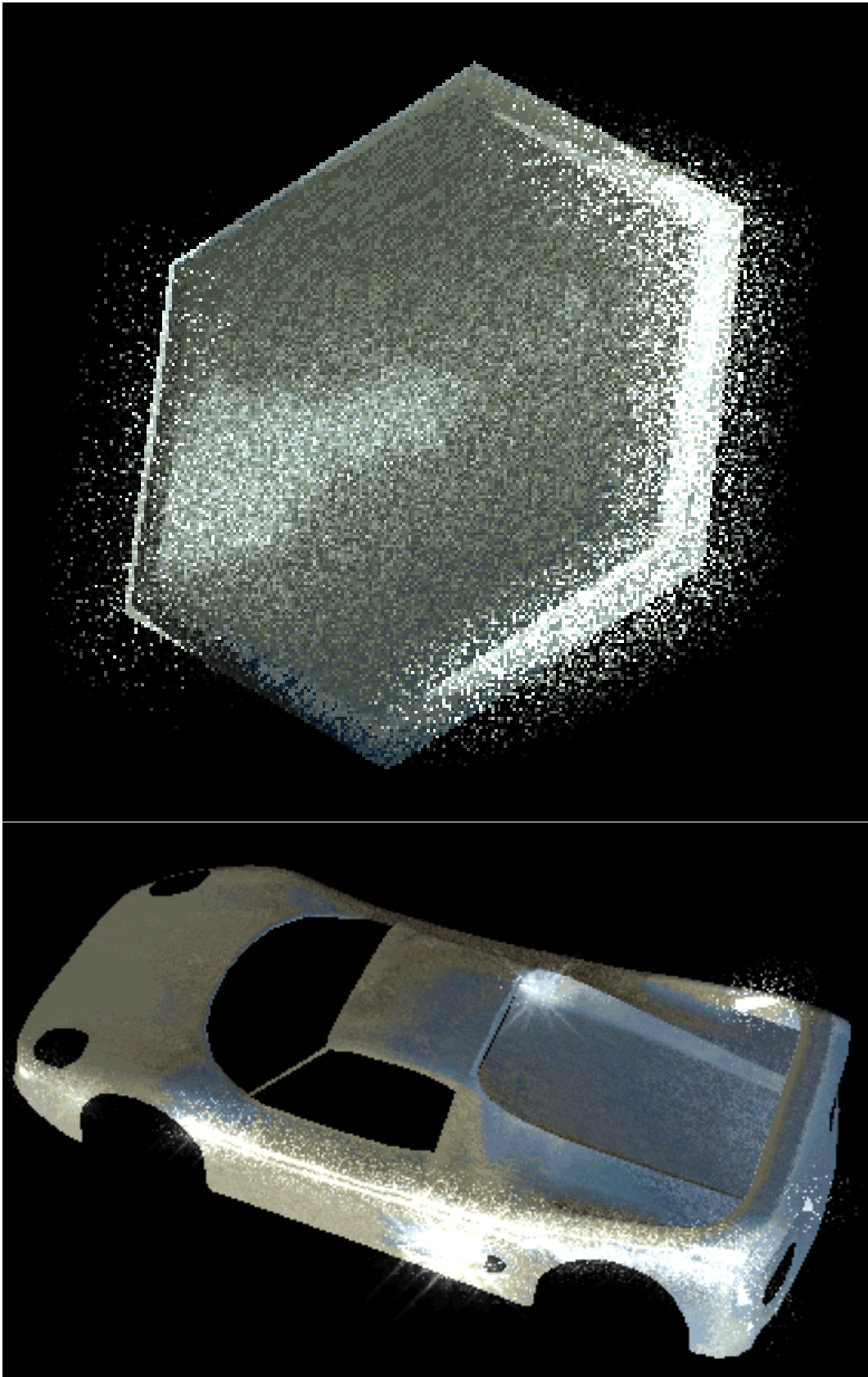


Gegenstände aus unedleren Metallen mit rauherer Oberfläche, dargestellt durch hohe Varianz mit überhöhter Z-Dämpfung

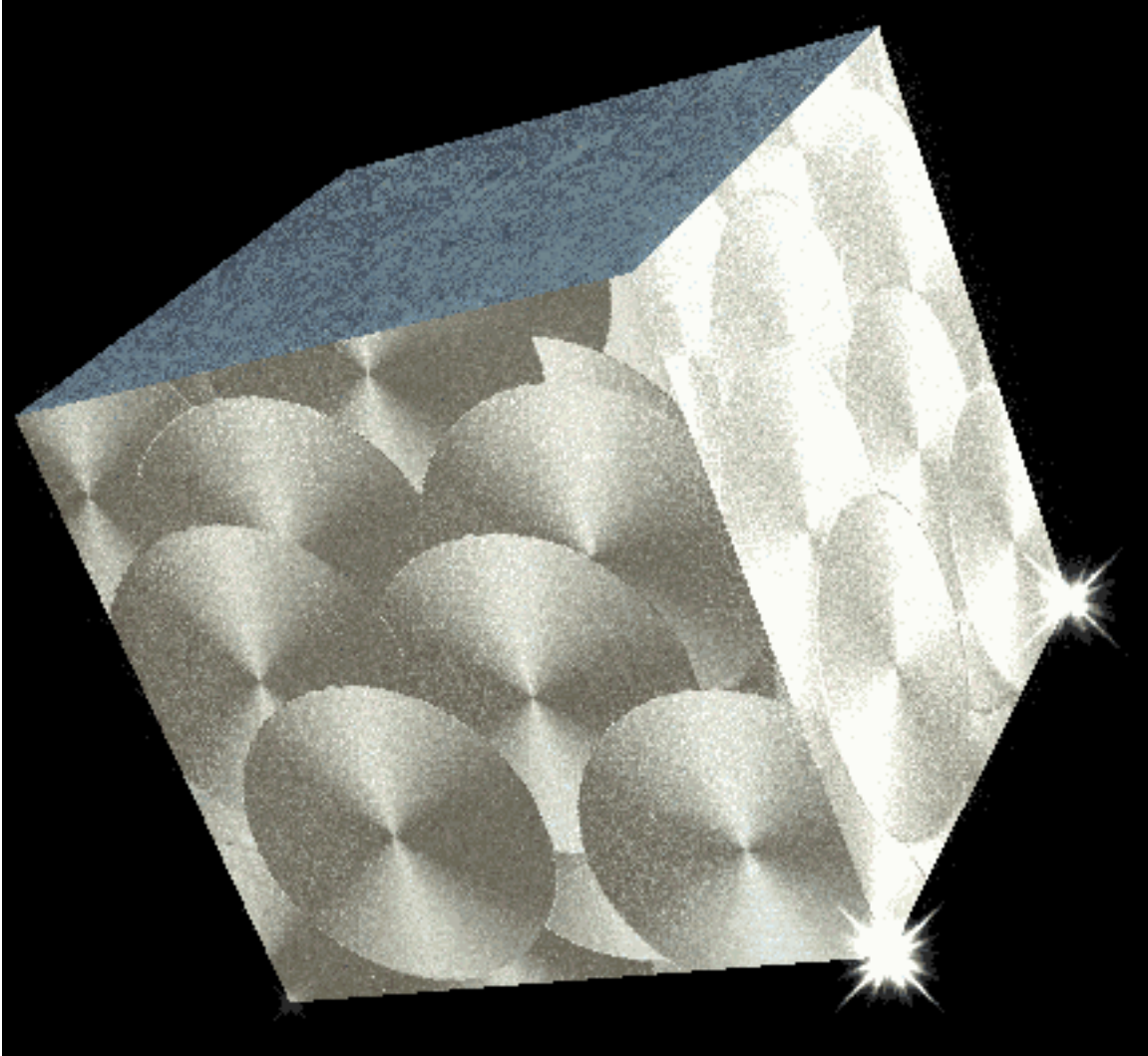




Verschiedene Objekte aus Buntmetallen mit geringer Anisotropie



Objekte mit geradlinig gebürsteten Oberflächen



Kreisförmig gebürsteter Würfel

Anhang B

Definitionen und Begriffe

- **ANISOTROPIE**
Eigenschaft, die in Bezug auf Metalloberflächen die Abhängigkeit des Reflexionsverhaltens von der Orientierung des einfallenden Lichtes parallel zur Oberfläche kennzeichnet. Der hier betrachtete Fall einfacher Anisotropie kann durch Angabe einer Hauptrichtung sowie die Stärke der Anisotropie vollständig spezifiziert werden.
- **BILLBOARD**
Zweidimensionale Figur, dessen Position der eines in Bildkoordinaten transformierten Vertexes entspricht. Je nach Verwendungszweck wird die Größe der Figur anhand der Z-Position in Kamera- oder Projektionskoordinaten skaliert. Die Texturkoordinaten der Figur sind dabei unabhängig von der Position in Bildkoordinaten der Position in der Figur fest zugeordnet.
- **BLINN-PHONG-MODELL**
Empirisches Modell, welches eine Modifikation des Phong-Modells darstellt und den Intensitätsverlauf des gerichtet-diffusen Anteils stattdessen als Funktion des Winkel zwischen der Oberflächennormale und dem Halfwayvektor zwischen Lichtstrahl und Sichtichtung modelliert.
- **BRDF**
Eine BRDF (Bi-directional Reflectance Distribution Function) beschreibt das Verhältnis von reflektiertem zum pro Raumwinkelement einfallenden Licht an einem Punkt auf einer Oberfläche. Als Parameter werden die Richtungen des einfallenden und des reflektierten Lichtes verwendet, so daß sich im Allgemeinen eine vierdimensionale Funktion und für isotrope Oberflächen eine dreidimensionale Funktion der Einfallswinkel α und Reflexionsrichtung β ergibt:

$$BRDF(\alpha_{Polar}, \alpha_{Azimuthal}, \beta_{Polar}, \beta_{Azimuthal}) \\ = \frac{\text{Reflektierte Intensität}}{\text{Einfallende Intensität} * \cos(\alpha_{Polar}) * \text{Raumwinkelement}}$$

Die Abhängigkeit des Reflexionsverhaltens von der Farbe des einfallenden Lichts wird üblicherweise dadurch berücksichtigt, indem man mehrere skalare BRDF $BRDF_{\lambda}(\alpha_{Polar}, \alpha_{Azimuthal}, \beta_{Polar}, \beta_{Azimuthal})$ für jeweils monochromatisches Licht zu einer Vektorfunktion $\mathbf{BRDF}(\alpha_{Polar}, \alpha_{Azimuthal}, \beta_{Polar}, \beta_{Azimuthal})$ für alle Farben zusammenfaßt.

- **FARBE**
Eigenschaft des Lichts, gekennzeichnet durch die Verteilung der enthaltenen Wellenlängen. Als Eigenschaft einer Oberfläche mit der Farbe des von ihr reflektierten weißen Lichts identisch
- **FILTERKERN**
Ein Filterkern gibt an, wie sich der gefilterte Wert eines Punktes aus den Werten der umgebenden Punkte errechnet. Für die gängigste Realisierung als gewichtete Summe über endlich viele benachbarte Punkte ist er durch Angabe der endlich vielen Wichtungsfaktoren vollständig spezifiziert.
- $Frac(x)$
Nachkommaanteil von x
- **GEORDNETES DITHERING**
Quantisierungsverfahren, um beliebige Eingabewerte anhand eines Vergleichswerts auf vorgegebene diskrete Werte abzubilden. Dabei hängt der Vergleichswert nur von der Position des Eingabewerts ab und kann somit als Textur gespeichert werden.
- **HDR-RENDERING**
Verfahren, welche auch Helligkeitswerte außerhalb des Intervalls $[0..1]$ zulassen. Diese werden anschließend durch eine nichtlineare Abbildung (z.B. Addition von Werten außerhalb des Bereichs nach Tiefpaßfilterung, anschließende Gammakorrektur) wieder in den Bereich $[0..1]$ abgebildet.
- **(RELATIVE) INTENSITÄT**
Relative Intensitäten geben den Anteil an einer dazugehörigen Gesamtintensität an, liegen also stets innerhalb $[0..1]$. Intensitäten kommen als Maß der Stärke des einfallenden bzw. reflektierten Lichts oder der von Pigment bedeckten Oberfläche vor.
- **MIKROFACETTE**
Perfekt spiegelnder ebener Bereich beliebig kleiner Oberfläche, der zur Modellierung nicht perfekt spiegelnder, rauher Oberflächen verwendet wird. Das Reflektionsverhalten der Gesamtoberfläche ergibt sich dann (mit Ausnahme eines farbbestimmenden Faktors) vollständig aus der Verteilung der Facettennormalen.
- **MODULO-ZUFALLSGENERATOR**
Pseudozufallsgenerator der Form $x_{n+1} := (x_n * a + b) \bmod c$ mit geeigneten Parametern a, b, c, x_0

- POINTSPRITE
 Als Punkt gerendeter Vertex, der durch Aktivieren von `D3DRS_POINTSPRITEENABLE` von Direct3D im Bildraum zu einem Quadrat mit automatisch generierten Texturkoordinaten erweitert wird. Die Achsen des Quadrats entsprechen denen des Bildkoordinatensystems, die Seitenlänge des Quadrats wird im Vertexshader berechnet. Die generierten Texturkoordinaten bilden eine Textur achsenparallel auf das Quadrat ab und liegen innerhalb $[0..1]^2$.
- PHONG-MODELL
 Rein empirisch begründetes Modell, welches den Intensitätsverlauf des spekularen Term genannten gerichtet-diffusen Anteils des reflektierten Lichts als Funktion des Winkels zwischen einfallendem Lichtstrahl und der an der Oberfläche gespiegelten Sichtichtung modelliert: $SpekularerTerm \sim Saturate(\cos(Spekularwinkel))^{Spekularexponent}$
 Der ungerichtet-diffuse Anteil, diffuser oder Lambertscher Term genannt, ist eine Funktion des Winkels zwischen einfallendem Licht und der Oberflächennormale: $DiffuserTerm \sim Saturate(\cos(Einfallswinkel))$
 Das fertige Phong-Modell besteht aus der Summe dieser beiden Terme.
- PIGMENT
 Material zur Erzeugung von Oberflächen mit bestimmter Farbe
- $Saturate(x)$
 $Saturate(x) := Median(\{0, x, 1\})$
- STRAHLENOPTIK
 Teilbereich der Optik, in dem die Ausbreitung des Lichtes durch senkrecht zur Wellenfront der Lichtwellen stehende Linien (Lichtstrahlen) beschrieben wird. Auf diese Weise ist eine einfache geometrische Beschreibung und Berechnung möglich, welche in der Praxis als Näherung ausreichend genaue Ergebnisse liefert. Bedingung dafür ist allerdings, daß die beteiligten Strukturen keine Variationen auf Größenordnung der Lichtwellenlänge aufweisen dürfen.
- **Vektoren**
 Vektoren werden in den beschriebenen Formeln durch Fettschrift kenntlich gemacht. Für Addition $\mathbf{a} + \mathbf{b}$ und Skalarprodukt $\mathbf{a} \bullet \mathbf{b} = dot(\mathbf{a}, \mathbf{b})$ zweier Vektoren gelten die üblichen Regeln der linearen Algebra, eine Multiplikation $\mathbf{a} * \mathbf{b}$ wird wie die Addition komponentenweise durchgeführt. Skalare Werte werden ebenfalls wie Vektoren behandelt, wobei fehlende Komponenten durch Wiederholung der existierenden Komponente ergänzt werden.
 2D-Vektoren werden für Textur-/Tangential- und Bildkoordinaten verwendet,
 3D-Koordinaten enthalten RGB-Farbwerte, Richtungsvektoren, Normalenvektoren sowie Positionsdaten.
 4D-Koordinaten werden für homogene Ortskoordinaten und xRGB-Farbwerte verwendet.

- KOORDINATENSYSTEME

Folgende Koordinatensysteme werden im Kontext des Shaders verwendet:

- TANGENTIALKOORDINATEN

Lokales 2D-Koordinatensystem auf der Oberflächenebene eines Objekts. Zur alleinigen Angabe von Richtungsvektoren genügt die Angabe eines Tangential- sowie eines Binormalenvektors für jeden Vertex, der Nullpunkt kann über Texturkoordinaten angegeben werden. Wird im Shader zur Angabe der Anisotropieachsen, Texturkoordinaten sowie Offsetvektoren verwendet. Erweiterung zu einem 3D-Koordinatensystem wird durch Hinzunahme der Oberflächennormale ermöglicht.

- OBJEKTKOORDINATEN

Lokales 3D-Koordinatensystem, in welchem die Ortsvektoren, Tangenten, Binormalen sowie Normalenvektoren der Vertices eines einzelnen Polygonmodells vorliegen. Die Ortsvektoren werden durch eine vierte, konstante Komponenten (in D3D = 1) zu einem 4D-Vektor in homogenen Koordinaten erweitert.

- WELTKOORDINATEN

Globales homogenes 4D-Koordinatensystem, in welches die Objekte durch Translation, Rotation, Skalierung und ggf. Projektion in der kompletten darzustellenden Szene positioniert werden. In diesen Koordinaten liegen auch Richtungen und Positionen der Lichtquellen vor.

- KAMERA KOORDINATEN

Globales homogenes 4D-Koordinatensystem, in dem die X und Y-Achsen parallel zur Bildebene vorliegen. Der Betrachter steht im Ursprung des Koordinatensystems, seine Sichtrichtung entspricht der positiven Z-Achse.

- PROJEKTIONSKOORDINATEN

Globales 4D-Koordinatensystem, in welchem die in Kamerakoordinaten vorliegenden Positionsdaten entsprechend der verwendeten Projektion vorbereitet werden. Für perspektivische Projektion entspricht die W-Koordinate nach Transformation der Z-Koordinate in Kamerakoordinaten, bei orthogonaler Projektion einer Konstante. In Projektionskoordinaten liegen die X- und Y-Komponenten der Ortsvektoren entsprechend des Sichtbereiches skaliert vor, die Z-Komponente wird so skaliert, daß die sichtbaren Werte innerhalb $[0..1]$ liegen.

- BILDKOORDINATEN

Globales 2D-Koordinatensystem, in welchem die X- und Y-Komponenten der Ortsvektoren nach der perspektivischen Division vorliegen. Dieses Koordinatensystem ist mit dem des Bildschirms bzw. des Rendertargets identisch.

- ZUFALLSDICHTE

Funktion, welche diskreten Zufallswerten die Häufigkeit ihres Auftretens zuordnet. Für unendlich viele, dicht geordnete Zufallswerte geht sie in eine kontinuierliche Funktion über. Diese ist so normiert, daß ihr Integral über alle Zufallswerte 1 ergibt.

- ZUFALLSFUNKTION

Monoton steigende Funktion mit Werten zwischen 0 und 1, deren Verlauf die Verteilung dazugehöriger Zufallswerte festlegt. Ihre Ableitung ist die Zufallsdichte.

Anhang C

Quellenverzeichnis

- 1 W.R.Cramer: Airbrush und Autolackdesign.
Ravensburger Buchverlag, 1992, ISBN 3-473-48022-3
- 2 M.F.Canal(Hrsg.): Airbrush.
Edition Michael Fischer, 2003, ISBN 3-933033-77-2
- 3 J.M.Parramón, M.Ferrón: Das große Technikbuch Airbrush.
Edition Michael Fischer, 1992, ISBN 3-924433-80-1
- 4 C.M.Mette: Airbrush: Eine Einführung.
Ravensburger Buchverlag, 1988, ISBN 3-473-48016-9
- 5 C.M.Mette: Airbrush: Eine Fortsetzung.
Ravensburger Buchverlag, 1989, ISBN 3-473-48020-7
- 6 J.Martin: Airbrush Technik, Metall-Effekte.
Edition Michael Fischer, 1990, ISBN 3-924433-84-4
- 7 W.Nolting: Grundkurs Theoretische Physik 3 Elektrodynamik.
Friedr. Vieweg & Sohn Verlagsgesellschaft, 1997, ISBN 3-528-16933-8
- 8 Wissenschaft-Online: Lexikon der Optik: Metalloptik.
<http://www.wissenschaft-online.de/abo/lexikon/optik/2003>
- 9 Wissenschaft-Online: Lexikon der Optik: Reflexion.
<http://www.wissenschaft-online.de/abo/lexikon/optik/2786>
- 10 B.Freudenberg, M.Masuch, T.Strothotte:
Real-Time Halftoning: A Primitive For Non-Photorealistic Shading.
Thirteenth Eurographics Workshop on Rendering, 2002
- 11 Steve "Gliebster" Gliebe: Tutorials: Metallic Surface.
<http://gliebster.com/tutorials/?type=photoshop&tutorial=metal>

- 12 Daniel Sperl: Künstlerisches Rendering für Echtzeit-Applikationen.
Diplomarbeit Fachhochschule Medientechnik und -design Hagenberg, 2003
- 13 B.Cabral, L.Leedom: Imaging vector Fields using line integral convolution.
Proceedings of SIGGRAPH '93, Computer Graphics, pp. 263-272, 1993
- 14 Mathematics:Probability and Statistics:Statistical Distributions:
Continuous Distributions:Uniform Sum Distribution.
MathWorld, Wolfram Research Inc, <http://mathworld.wolfram.com/UniformSumDistribution.html>
- 15 M.Ashikhmin, P.Shirley:
An Anisotropic Phong Light Reflection Model. University of Utah, 2000
- 16 A.Gooch, B.Gooch., P.Shirley, E.Cohen:
A Non-Photorealistic Lighting Model For Automatic Technical Illustration.
Department of Computer Science, University of Utah, 1998
- 17 M.J.Kilgard: A Practical and Robust Bump-mapping Technique for Today's GPUs. NV-
DIA Corporation, 2000
- 18 A.Schilling: Towards Real-Time Photorealistic Rendering: Challenges and Solutions.
WSI/GRIS, Universität Tübingen, 1997
- 19 M.Toksvig: Mipmapping Normal Maps. NVIDIA Corporation, 2004
- 20 A.S.Shastry: High Dynamic Range Rendering.
<http://www.gamedev.net/reference/articles/article2108.asp>
- 21 DirectX 9.0 Programmer's Reference. Microsoft Corporation, 2002

Anhang D

Shader-Quelltexte

D.1 Vertexshader des Hauptdurchgangs

```

#define LOCALVIEWER true
#define Texturdaten true

float4x4 matWeltKameraProjektion; //Objektkoordinaten -> Projektionskoordinaten
float4x4 matWelt; //Objektkoordinaten -> Kamerakoordinaten
float4x4 matWeltInvTrans; //Objektkoordinaten -> Kamerakoordinaten für die Normalen
float4x4 matKameraInv; //Kamerakoordinaten -> Weltkoordinaten
float TexGroesse = 64.; //gibt Größe der quadratischen Zufallstextur in Texeln an
float BildGroesseX; //Abmessungen des zu rendernden Bildes in Pixeln
float BildGroesseY;

struct PixelVertex
{
    float4 Position : POSITION; //Vertexposition in Projektionskoordinaten
    float4 BildPosition : TEXCOORD0; //Kopie von POSITION für den Pixelshader
    float3 Normale : TEXCOORD1;
    float3 Tangente : TEXCOORD2;
    float3 Binormale : TEXCOORD3;
    float4 Textur2x : TEXCOORD4; //enthält 2 2D-Koordinaten:
                                //Textur in .xy sowie Modulationstextur in .zw
    float3 KameraVekt : TEXCOORD5; //Sichtrichtung
    float3 WeltPosition : TEXCOORD6; //Vertexposition in Weltkoordinaten
};

PixelVertex main(
    float4 VertexPosition : POSITION,
    float3 VertexNormale : NORMAL,
    float3 VertexTangente : TANGENT, //Tangente ggf. aus Texturkoordinaten ermittelt
    float2 VertexTex : TEXCOORD0) //enthält Texturkoordinaten
{
    PixelVertex Res;

    //transformiere Position in den Projektionsraum
    Res.Position = mul( matWorldViewProjection, VertexPosition );
    Res.BildPosition = float4((BildGroesseX / 2.) / TexGroesse,
                              (BildGroesseY / 2.) / TexGroesse, 1., 1.)*Res.Position;

    //übergebe Texturkoordinaten
    Res.Textur2x.xy = VertexTex;
}

```

```
//ermittle Sprühmodulationstexturkoordinaten
if(Texturdaten)
{
    //verwende existierende Texturkoordinaten für Modulationstextur
    Res.Textur2x.zw = VertexTex; //verwende existierende Koordinaten
}
else
{
    //verwende Spiegelung von (1,0,0) aus in Objektkoordinaten
    Res.Textur2x.zw = 2 * VertexNormale.x * VertexNormale - float3(1,0,0);
}

//transformiere Normale (nach außen gerichtet) in Weltkoordinaten
Res.Normale = normalize(mul(matWorldInvTrans,float4(VertexNormale.xyz,0.))).xyz;
//transformiere Tangente/Binormale in Weltkoordinaten
Res.Tangente = normalize(mul(matWorld,float4(VertexTangente.xyz,0.))).xyz;
Res.Binormale = cross(Res.Normale,Res.Tangente);

//ermittle Position in Weltkoordinaten (benötigt für Punktlichtquellen)
Res.WeltPosition = mul(matWorld,VertexPosition);

//ermittle Sichtrichtung (Kamera->Vertex) in Weltkoordinaten
if(LOCALVIEWER)
{
    //Kameraposition = Ursprung der Kamerakoordinaten
    Res.KameraVekt = normalize(mul(matWorld,VertexPosition).xyz
        - mul(matViewInv,float4(0,0,0,1)).xyz);
}
else
{
    //verwende unendlich weit entfernte Kamera
    //mit Sichtrichtung in z-Richtung in Kamerakoordinaten
    Res.KameraVekt = normalize(mul(matViewInv,float4(0,0,1,0)).xyz);
}

return Res;
}
```

D.2 Pixelshader des Hauptdurchgangs

```
//allgemeine Konstanten
const float PI=3.14159265359;

//Lichtquellen
//Anzahl der Lichtquellen
#define LichtNum 2
//aktuell verwendete Lichtquellen
#define MinLichtNum 0
#define MaxLichtNum 2
const bool Punktlicht=true;
const bool Richtungslicht=false;
struct Lichtquelle
{
    float4 Ambient;
    float4 Diffus;
    float4 GegenlichtDiffus;
    float4 Spekular;
    float3 Vektor; //Position oder Richtung
    bool Typ;      //Punktlicht=true oder Richtungslicht=false;
};
Lichtquelle Licht[MaxLichtNum];
float4 LichtfarbeEnv;
sampler EnvMap;
float4 EnvFarbe(float3 Richtung); //liest Farbe des Umgebungslichts

//Material
float4 DiffuseFarbe;
float4 SpekulareFarbe;
float Spekularexponent;
float Isotropie;
#define Abtastpunkte 5
sampler AnisoMap;
sampler BumpMap;
sampler decaltexture;
```

```
//Sprüheffekt
float Spruehparameter; //globale Variable, verwendet in spray(...)
float Streuung;
float Verlauf;
float z_Daempfung;
float TexGroesse;
float ModTexBreite;
sampler Zufallswerte;
sampler Modulierer;
sampler Relinearisierer;
float4x4 matKameraProjektion; //Kamerakoordinaten->Projektionskoordinaten
float Spray(float Intensitaet,float Spruehparameter); //berechnet lokale Intensität

float4 main(float4 BildPosition : TEXCOORD0,
            float3 IntNormale    : TEXCOORD1,
            float3 IntTangente   : TEXCOORD2,
            float3 IntBinormale  : TEXCOORD3,
            float4 Textur2x      : TEXCOORD4, //enthält 2 2D-Koordinaten
            float3 IntKameraVekt: TEXCOORD5,
            float3 WeltPosition  : TEXCOORD6) : COLOR
{
    //berechne lokale Geometrie (in Weltkoordinaten)

    //Objektparameter
    float2 Textur = Textur2x.xy;
    float2 TexturMod = Textur2x.zw;
    float3 TexNormale = normalize(IntNormale);
    float3 Tangente = normalize(IntTangente);
    float3 Binormale = normalize(IntBinormale);
```

```

//BumpMap
float3 Normale = TexNormale;
float3 Offset = tex2D(BumpMap,Textur);
Normale = normalize(TexNormale + Offset.x * Tangente + Offset.y * Binormale);

//Anisotropiehauptachse (Tangente der Schraffur)
float2 Rotation = normalize(tex2D(AnisoMap,Textur).xy);
float3 Anisotropieachse = Rotation.x * Tangente + Rotation.y * Binormale;
//Anisotropietangente projiziert und skaliert
float3 Anisotropietangente = mul(matViewProjection,float4(Anisotropieachse.xyz,0));
float2 AnisotropietangenteBild = 1. / (TexGroesse/sqrt(2.))
    * normalize(Anisotropietangente.xy
        - Anisotropieachse.z
        * normalize(BildPosition.xy));

//ermittle Sprühparameter
//moduliere Streuung mit Entfernung
float LokaleStreuung = Streuung;
if(Tiefendaempfung_An) LokaleStreuung*= (1. - (BildPosition.z / BildPosition.w)
    * z_Daempfung);

//skaliere Modulationstextur anhand des sichtbaren Bereichs
float Modulation = tex2D(Modulierer,TexturMod);

//Lese Zufallswert (mit linearem Filter in Richtung der Anisotropietangente)
float4 Zufallswert = tex2Dproj(Zufallswerte,BildPosition);

//projiziere Position in Bildraum
float2 TexPosP = BildPosition.xy / BildPosition.w;
float2 TexPosN = BildPosition.xy / BildPosition.w;
//berechne gefilterten Wert
float4 ZufallswertAniso = Zufallswert;
for(int i = 0;i < ((Abtastpunkte - 1) / 2);i++)
{
    TexPosP-= AnisotropietangenteBild;
    TexPosN+= AnisotropietangenteBild;
    ZufallswertAniso+= tex2D(Zufallswerte,TexPosP);
    ZufallswertAniso+= tex2D(Zufallswerte,TexPosN);
}
//erzeuge gleichverteilte Werte:
ZufallswertAniso = tex2D(Relinearisierer,ZufallswertAniso / Abtastpunkte);
Zufallswert = lerp(ZufallswertAniso,Zufallswert,Isotropie);

```

```
//Moduliere Zufallswert und skaliere
float4 Spruehparameter = LokaleStreuung*(0.5 - frac(Modulation + Zufallswert));

//globale Beleuchtungsparameter
float4 SpekularerAnteil = 0;
float4 DiffuserAnteil = 0;
float3 KameraVekt = normalize(IntKameraVekt);
float3 ReflektVekt = reflect(KameraVekt, Normale);

//Materialfarbe
float4 SpekulareFarbeFresnel = SpekulareFarbe;
float4 DiffuseFarbeFresnel = DiffuseFarbe;
//Fresnel-Farbkorrektur
float FresnelRef = pow(1.-saturate(dot(-KameraVekt, Normale)), 5.);
SpekulareFarbeFresnel = lerp(SpekulareFarbe, 1., FresnelRef);
//diffuse Fresnel-Farbkorrektur ist vernachlässigbar:
DiffuseFarbeFresnel = DiffuseFarbe;

for(int i=MinLichtNum; i<MaxLichtNum; i++)
{
    //lokale Beleuchtungsparameter
    float3 LichtVekt;
    if(Licht[i].Typ==Richtungslicht)
    {
        LichtVekt = -normalize(Licht[i].Vektor.xyz);
    }
    else
    {
        LichtVekt = normalize((Licht[i].Vektor-WeltPosition).xyz);
    }
    float3 Spekularorientierung;
    float cos_Spekularwinkel;
    float cos_LichtNormale = dot(LichtVekt, Normale);

    Spekularorientierung = normalize(-KameraVekt + LichtVekt);
    cos_Spekularwinkel = dot(Spekularorientierung, Normale);
}
```

```

//berechne Beleuchtung
float SpekularexponentAniso = Spekularexponent;

//berechne anisotropen Spekularwert
//(für Blinn-Phong vom Ort der Lichtquelle abhängig)
float cos_ReflektAnisotropieachse = dot(Spekularorientierung,Anisotropieachse);
float cos_ReflektNormale = dot(Spekularorientierung,TexNormale);
//Tangentiale Komponente des Richtungsvektors ^2:
float OrientierungAniso = pow(cos_ReflektAnisotropieachse,2);
//Länge des in die Ebene projizierten Richtungsvektor ^2:
float KorrekturAniso = 1./(1.-pow(cos_ReflektNormale,2));
SpekularexponentAniso = lerp(Spekularexponent / Isotropie,
                            Spekularexponent * Isotropie,
                            OrientierungAniso * KorrekturAniso);

//(hier: Phong mit diffusem negativem Gegenlicht
//(Ambientes Licht ggf. anpassen))
float IntensitaetAmbient = 1.;
float IntensitaetDiffus = abs(cos_LichtNormale);
float IntensitaetSpekular = pow(saturate(cos_Spekularwinkel),
                               SpekularexponentAniso);
float4 RefFarbeAmbient = DiffuseFarbeFresnel * Licht[i].Ambient;
float4 RefFarbeDiffus = ((cos_LichtNormale > 0)
                        ? DiffuseFarbeFresnel * Licht[i].Diffus
                        : DiffuseFarbeFresnel * Licht[i].GegenlichtDiffus);
float4 RefFarbeSpekular = SpekulareFarbeFresnel * Licht[i].Spekular;

//wende Beleuchtung an
//Diffuse Reflexion
float4 DiffusesLicht = RefFarbeDiffus
                    * Spray(IntensitaetDiffus,Spruehparameter.z);
float4 AmbientesLicht = RefFarbeAmbient * IntensitaetAmbient;
DiffuserAnteil+= DiffusesLicht + AmbientesLicht;
//Spekulare Reflexion
SpekularerAnteil+= RefFarbeSpekular
                  * Spray(IntensitaetSpekular,Spruehparameter.w);
}

//resultierende Farbe
float4 Pixel = SpekularerAnteil+DiffuserAnteil;

```

```
//Environment Map
float IntensitaetEnvmap = 1.; //für Metalle praktisch winkelunabhängig
float Streuweite = saturate(3. / sqrt(Spekularexponent));
float4 RefFarbeEnv = SpekulareFarbeFresnel * LichtfarbeEnv;
float3 EnvreflektVekt = ReflektVekt + Spruehparameter.xyz * Streuweite;
float4 EnvAnteil = RefFarbeEnv * EnvFarbe(EnvreflektVekt) * IntensitaetEnvmap;
Pixel+= EnvAnteil;

return Pixel;
}
```

D.3 Vertexshader zur Erzeugung der spekularen Textur (erster Durchgang)

```
#define LOCALVIEWER true
#define Texturdaten true

float4x4 matWeltKameraProjektion; //Objektkoordinaten -> Projektionskoordinaten
float4x4 matWelt; //Objektkoordinaten -> Kamerakoordinaten
float4x4 matWeltInvTrans; //Objektkoordinaten -> Kamerakoordinaten für die Normalen
float4x4 matKameraInv; //Kamerakoordinaten -> Weltkoordinaten
float TexGroesse; //gibt Größe der quadratischen Zufallstextur in Texeln an
float BildGroesseX; //Abmessungen des zu rendernden Bildes in Pixeln
float BildGroesseY;

struct PixelVertex
{
    float4 Position : POSITION;
    //TEXCOORD0 : nicht benötigt
    float3 Normale : TEXCOORD1;
    float3 Tangente : TEXCOORD2;
    float3 Binormale : TEXCOORD3;
    float4 Textur2x : TEXCOORD4; //enthält 2 2D-Koordinaten
    float3 KameraVekt : TEXCOORD5;
    float3 WeltPosition : TEXCOORD6;
};

PixelVertex main(
    float4 VertexPosition : POSITION,
    float3 VertexNormale : NORMAL,
    float3 VertexTangente : TANGENT,
    float2 VertexTex : TEXCOORD0)
{
    PixelVertex Res;

    //transformiere Position in den Projektionsraum
    Res.Position = mul( matWorldViewProjection, VertexPosition );

    //übergebe Texturkoordinaten
    Res.Textur2x.xy = VertexTex;
```

```
//ermittle Sprühmodulationstexturkoordinaten
if(Texturdaten)
{
    Res.Textur2x.zw = VertexTex; //verwende existierende Koordinaten
}
else
{
    //verwende Spiegelung von (1,0,0) aus im Objektraum
    Res.Textur2x.zw = 2 * VertexNormale.x * VertexNormale - float3(1,0,0);
}

//transformiere Normale (nach außen gerichtet) in Weltkoordinaten
Res.Normale = normalize(mul(matWorldInvTrans,float4(VertexNormale.xyz,0.))).xyz;

//transformiere Tangente/Binormale in Weltkoordinaten
Res.Tangente = normalize(mul(matWorld,float4(VertexTangente.xyz,0.))).xyz;
Res.Binormale = cross(Res.Normale,Res.Tangente);

//ermittle Position in Weltkoordinaten (für Punktlichtquellen)
Res.WeltPosition = mul(matWorld,VertexPosition);

//ermittle Sichtrichtung (Kamera->Vertex) in Weltkoordinaten
if(LOCALVIEWER)
{
    //Kameraposition = Ursprung der Kamerakoordinaten
    Res.KameraVekt = normalize(mul(matWorld,VertexPosition).xyz
        - mul(matViewInv,float4(0,0,0,1)).xyz);
}
else
{
    //verwende unendlich weit entfernte Kamera
    //mit Sichtrichtung in z-Richtung in Kamerakoordinaten
    Res.KameraVekt = normalize(mul(matViewInv,float4(0,0,1,0)).xyz);
}

return Res;
}
```

D.4 Pixelshader zur Erzeugung der spekularen Textur (erster Durchgang)

```
//allgemeine Konstanten
const float PI=3.14159265359;

//Lichtquellen
//Gesamtzahl Lichtquellen
#define LichtNum 2
//aktuell verwendete Lichtquellen
#define MinLichtNum 0
#define MaxLichtNum 2
const bool Punktlicht = true;
const bool Richtungslicht = false;
struct LichtquelleSpekular
{
    float3 Vektor; //float4 Position oder Richtung
    bool Typ; //Punktlicht=true oder Richtungslicht=false;
};
Lichtquelle Licht[MaxLichtNum];

//Material
float4 SpekulareFarbe;
float Spekularexponent;
float Isotropie;
sampler AnisoMap;
sampler BumpMap;

//Effekt
float Spitzlichtstaerke; //Effektstärke durch Skalierung der Intensität einstellen
sampler Modulierer;
```

```
float4 main(float3 IntNormale    : TEXCOORD1,
            float3 IntTangente   : TEXCOORD2,
            float3 IntBinormale  : TEXCOORD3,
            float4 Textur2x      : TEXCOORD4, //enthält 2 2D-Koordinaten
            float3 IntKameraVekt: TEXCOORD5,
            float3 WeltPosition  : TEXCOORD6) : COLOR
{
    //berechne lokale Geometrie (in Weltkoordinaten)

    //Objektparameter
    float2 Textur = Textur2x.xy;
    float2 TexturMod = Textur2x.zw;
    float3 TexNormale = normalize(IntNormale);
    float3 Tangente = normalize(IntTangente);
    float3 Binormale = normalize(IntBinormale);

    //BumpMap
    float3 Normale = TexNormale;
    float3 Offset = tex2D(BumpMap,Textur);
    Normale = normalize(TexNormale + Offset.x * Tangente + Offset.y * Binormale);

    //Anisotropiehauptachse (Tangente der Schraffur)
    float2 Rotation = normalize(tex2D(AnisoMap,Textur).xy);
    float3 Anisotropieachse = Rotation.x * Tangente + Rotation.y * Binormale;

    //skaliere Modulationstextur anhand des sichtbaren Bereichs
    float Modulation = tex2D(Modulierer,TexturMod);

    //globale Beleuchtungsparameter
    float IntensitaetSpekular = 0;
    float3 KameraVekt = normalize(IntKameraVekt);
    float3 ReflektVekt = reflect(KameraVekt,Normale);
```

```
for(int i=MinLichtNum;i<MaxLichtNum;i++)
{
    //lokale Beleuchtungsparameter
    float3 LichtVekt;
    if(Licht[i].Typ==Richtungslicht)
    {
        LichtVekt = -normalize(Licht[i].Vektor.xyz);
    }
    else
    {
        LichtVekt = normalize((Licht[i].Vektor-WeltPosition).xyz);
    }
    float3 Spekularorientierung;
    float cos_Spekularwinkel;
    Spekularorientierung = normalize(-KameraVekt+LichtVekt);
    cos_Spekularwinkel = dot(Spekularorientierung,Normale);

    //berechne Beleuchtung
    float SpekularexponentAniso = Spekularexponent;
    if(Anisotropie_An)
    {
        //berechne anisotropen Spekularwert
        //(für Blinn-Phong vom Ort der Lichtquelle abhängig)
        float cos_ReflektAnisotropieachse = dot(Spekularorientierung,Anisotropieachse);
        float cos_ReflektNormale = dot(Spekularorientierung,TexNormale);
        //Tangentiale Komponente des Richtungsvektors^2 :
        float OrientierungAniso = pow(cos_ReflektAnisotropieachse,2);
        //Länge des in die Ebene projizierten Richtungsvektor^2 :2
        float KorrekturAniso = 1. / (1. - pow(cos_ReflektNormale,2));
        SpekularexponentAniso = lerp(Spekularexponent / Isotropie,
                                     Spekularexponent * Isotropie,
                                     OrientierungAniso * KorrekturAniso);
    }
    //(hier: Phong mit diffusem negativem Gegenlicht
    //(Ambientes Licht ggf. anpassen))
    IntensitaetSpekular+= pow(saturate(cos_Spekularwinkel),SpekularexponentAniso);
}
```

D.4: Pixelshader zur Erzeugung der spekularen Textur (erster Durchgang)

```
//resultierende Farbe
float4 Pixel;
Pixel.rgb=Spitzlichtstaerke * IntensitaetSpekular;

//übergebe Wert der Modulationstextur an zweiten Durchgang
Pixel.a = Modulation;

return Pixel;
}
```

D.5 Vertexshader für Glanzlichter (zweiter Durchgang)

```

float TexGroesse; //Größe der quadratischen Zufallstextur in Texeln
float BildGroesseX; //Abmessungen des zu rendernden Bildes in Pixeln
float BildGroesseY;
float4 LichtVekt1; //Lichtrichtung der hellsten Lichtquelle

float4x4 matKamera; //Weltkoordinaten -> Kamerakoordinaten

struct PixelVertex
{
    float4 Position:    POSITION;
    float4 BildPosition: TEXCOORD0;
    float2 BildLichtVekt: TEXCOORD1;
};

PixelVertex main( float4 VertexPosition: POSITION )
{
    PixelVertex Res;

    //übergebe Koordinaten (Tiefe: vordere Z-Grenze)
    Res.Position.xy = VertexPosition;
    Res.Position.zw = float2(0,1);

    //transformiere Bildkoordinaten in entsprechende Texturkoordinaten
    Res.BildPosition.x = 0.5 * ( Res.Position.x + 1.);
    Res.BildPosition.y = 0.5 * (-Res.Position.y + 1.);
    Res.BildPosition.zw = float2((BildGroesseX / 2.) / TexGroesse,
                                (BildGroesseY / 2.) / TexGroesse) * Res.Position;

    //projiziere Lichtvektor in Bildebene
    float3 BildLichtVekt = float3(1,-1,1)
                          * mul(matView,float4(LichtVekt1.xyz,0.)).xyz;
    Res.BildLichtVekt = normalize(BildLichtVekt.xy);

    return Res;
}

```

D.6 Pixelshader für Glanzlichter (zweiter Durchgang)

```

float Spitzlichtweite; //Streuradius eines Texels in der spekularen Textur
float Spitzlichtoffset; //Stärke der Verschiebung des Streukreises in Richtung
                        //der hellsten Lichtquelle

sampler Zufallswerte;
sampler SpekularTex; //enthält spekulare Intensität in R,G und B dupliziert sowie
                    //Modulationswert im Alphakanal

float4 main( float4 TexPos2x: TEXCOORD0, float2 LichtVektProj: TEXCOORD1) : COLOR0
{
    //ermittle Effektparameter
    float4 Spruehparameter = frac(tex2D(Zufallswerte, TexPos2x.zw)
                                + tex2D(SpekularTex, TexPos2x.xy).aaaa) - 0.5;
    float4 Spruehoffset = Spruehparameter * Spitzlichtweite;
    float2 Lichtoffset = Spitzlichtoffset * LichtVektProj;

    //berechne Effekt
    float2 Leseposition = TexPos2x.xy + Lichtoffset;
    float4 Spitzlichtfarbe = 0;
    Spitzlichtfarbe+= tex2D(SpekularTex, Leseposition + Spruehoffset.xy)
                    * (1. - 2. * length(Spruehparameter.xy));
    Spitzlichtfarbe+= tex2D(SpekularTex, Leseposition + Spruehoffset.zw)
                    * (1. - 2. * length(Spruehparameter.zw));
    Spitzlichtfarbe+= tex2D(SpekularTex, Leseposition + Spruehoffset.yw)
                    * (1. - 2. * length(Spruehparameter.yw));
    Spitzlichtfarbe+= tex2D(SpekularTex, Leseposition + Spruehoffset.xz)
                    * (1. - 2. * length(Spruehparameter.xz));
    return Spitzlichtfarbe;
}

```

D.7 Vertexshader für Sterneffekte (zweiter Durchgang)

```
// D3DRS_POINTSPRITEENABLE ist vor Verwendung des Shaders zu aktivieren

#define LOCALVIEWER true

float4x4 matWeltKameraProjektion;
float4x4 matWelt;
float4x4 matWeltInvTrans;
float4x4 matKameraInv;
float Spekularexponent;
float Spritegroesse; //maximale Größe des Sprites im Bildraum in Pixeln

//Lichtquellen
//Gesamtzahl Lichtquellen
#define LichtNum 2
//aktuell verwendete Lichtquellen
#define MinLichtNum 0
#define MaxLichtNum 2
const bool Punktlicht = true;
const bool Richtungslicht = false;
struct LichtquelleSpekular
{
    float3 Vektor; //float4 Position oder Richtung
    bool Typ; //Punktlicht=true oder Richtungslicht=false;
};
Lichtquelle Licht[MaxLichtNum];

struct PixelVertex
{
    float4 Position : POSITION;
    float2 BildPosition : COLOR0;
    float Punktgroesse : PSIZE;
};
```

```
PixelVertex main(
    float4 VertexPosition : POSITION,
    float3 VertexNormale  : NORMAL,
    float2 VertexTex      : TEXCOORD0)
{
    PixelVertex Res;

    //transformiere Position in den Projektionsraum
    Res.Position = mul( matWeltKameraProjektion, VertexPosition );
    //berechne Zufalls- und Spekularwertkoordinaten
    Res.BildPosition.x = ( Res.Position.x/Res.Position.w+1.)*0.5;
    Res.BildPosition.y = (-Res.Position.y/Res.Position.w+1.)*0.5;

    //transformiere Normale in Weltkoordinaten
    float3 Normale = normalize(mul(matWeltInvTrans,float4(VertexNormale.xyz,0)).xyz);

    //Anisotropie/Bumpmap entfällt, da erst ab VS3.0 möglich

    //globale Beleuchtungsparameter
    float IntensitaetSpekular = 0;
    float3 WeltPosition = mul(matWelt,VertexPosition);
    //ermittle Sichtrichtung (Kamera->Vertex) in Weltkoordinaten
    float3 KameraVekt;
    if(LOCALVIEWER)
    {
        KameraVekt = normalize(WeltPosition-mul(matKameraInv,float4(0,0,0,1)).xyz);
    }
    else
    {
        KameraVekt = normalize(mul(matKameraInv,float4(0,0,1,0)).xyz);
    }
    float3 ReflektVekt = reflect(KameraVekt,Normale);
```

```
for(int i=MinLichtNum;i<MaxLichtNum;i++)
{
    //lokale Beleuchtungsparameter
    float3 LichtVekt;
    if(Licht[i].Typ==Richtungslicht)
    {
        LichtVekt = -normalize(Licht[i].Vektor.xyz);
    }
    else
    {
        LichtVekt = normalize((Licht[i].Vektor-WeltPosition).xyz);
    }
    float3 Spekularorientierung;
    float cos_Spekularwinkel;
    Spekularorientierung = normalize(-KameraVekt+LichtVekt);
    cos_Spekularwinkel = dot(Spekularorientierung, Normale);

    //berechne Beleuchtungsintensität nach Blinn-Phong
    IntensitaetSpekular+= pow(saturate(cos_Spekularwinkel),Spekularexponent);
}

Res.Punktgroesse = IntensitaetSpekular*Spritegroesse;

return Res;
}
```

D.8 Pixelshader für Sterneffekte (zweiter Durchgang)

```
sampler SpekularTex; //spekulare Textur mit Intensität in R, G und B dupliziert
sampler EffektSprite; //Textur mit Sterneffekt

float Spritehelligkeit; //Helligkeit des Effekts zur Kompensation von Mehrfach-
                        //überblendung und zu geringen spekularen Intensitäten

float4 main(float2 TexEffekt: TEXCOORD0, float4 TexPos: COLOR0) : COLOR0
{
    return Spritehelligkeit * tex2D(EffektSprite, TexEffekt)
        * tex2D(SpekularTex, TexPos.xy);
}
```

D.9 Shader zur Erzeugung einer vorberechneten zylindrischen Environmentmap mit automatisch generierten Farben für Ober- und Unterseite

```
const float PI = 3.14159265359;

sampler envmap;

float Horizontuebergang; //Abstand des Randübergangs von der Bildmitte
float Horizontverlauf; //Verlauf des Übergangs
float xzuy;

#define Abtastpunkte 8 //Anzahl Abtastpunkte für Mittelung der Randfarbe

float4 Envmap_Halbzyylinder_Randfarbe(float2 Texpos : TEXCOORD0 ) : COLOR
{
    //berechne halbzyklindrisches Mapping
    float2 envmap_uv;
    float Texposz = sqrt(1. - Texpos.x * Texpos.x);
    envmap_uv.x = (atan2(Texpos.x, Texposz) / PI) + 0.5;
    envmap_uv.y = 0.5 * (1. + 0.5 * Xzuy * Texpos.y
        / sqrt((1. - Texpos.y * Texpos.y)))

    //ermittle nächstliegenden Rand der Textur in y-Richtung
    float RandpositionY = 0.5 + 0.5 * sign(Texpos.y);

    //Berechne passende Randfarbe als Mittelwert des vorhandenen Bildrandes
    float4 Randfarbe = 0.;
    for(int i = 0; i < Abtastpunkte; i++)
    {
        Randfarbe += tex2D(envmap, float2(i / Abtastpunkte, RandpositionY)) / Abtastpunkte;
    }

    //Mischungsverhältnis Randfarbe zu Bildfarbe
    float Mischung = pow(saturate((abs(Texpos.y) - Horizontuebergang)
        / (1. - Horizontuebergang)), Horizontverlauf);

    return lerp(tex2D(envmap, envmap_uv), Randfarbe, Mischung);
}
```

D.10 Shader zur Erzeugung einer vorberechneten zylindrischen Environmentmap mit Überblendung zu vorgegebenen Farben für Ober- und Unterseite

```
const float PI=3.14159265359;

sampler envmap;

float Horizontuebergang; //Abstand des Randübergangs von der Bildmitte
float Horizontverlauf; //Verlauf des Übergangs
float XzuY;
float4 FarbeOben;
float4 FarbeUnten;

float4 Envmap_Halbzyylinder_2xFarbe(float2 Texpos : TEXCOORD0 ) : COLOR
{
    //berechne halbzylindrisches Mapping
    float2 envmap_uv;
    float Texposz = sqrt(1. - Texpos.x * Texpos.x);
    envmap_uv.x = (atan2(Texpos.x, Texposz) / PI) + 0.5;
    envmap_uv.y = 0.5 * (1. + 0.5 * XzuY * Texpos.y
        / sqrt((1. - Texpos.y * Texpos.y)))

    //ermittle Farbe des nächstliegenden Rands der Textur in y-Richtung
    float Randposition = 0.5 + 0.5 * sign(Texposy);
    float4 Randfarbe = lerp(FarbeOben, FarbeUnten, Randposition);

    //Mischungsverhältnis Randfarbe zu Bildfarbe
    float Mischung = pow(saturate((abs(Texposy) - Horizontuebergang)
        / (1. - Horizontuebergang)), Horizontverlauf);

    return lerp(tex2D(envmap, envmap_uv), Randfarbe, Mischung);
}
```