13.4 Umgebungs-Texturen (Environment Maps)

Bis vor wenigen Jahren war ein klares Erkennungszeichen von interaktiver 3D-Computergrafik das Fehlen von realistischen Spiegelungen. Dies war den sehr viel aufwändigeren Ray-Tracing-Verfahren (Abschnitt 12.1.2) vorbehalten. Durch die Einführung von Umgebungs-Texturen (*Environment Maps*, [Blin76]) können Objekt-Spiegelungen jedoch schon ziemlich gut auf aktuellen Grafikkarten in Echtzeit dargestellt werden. Das Grundprinzip ist einfach und gleicht in den ersten Schritten dem Ray-Tracing-Verfahren. Man schickt einen Strahl vom Augenpunkt auf einen Punkt der reflektierenden Oberfläche und berechnet mit Hilfe des Normalenvektors den Reflexionsvektor. Anstatt den reflektierten Strahl bis zur nächsten Objektoberfläche zu verfolgen, um dort ein lokales Beleuchtungsmodell auszuwerten, wie dies beim Ray-Tracing der Fall ist, wird jetzt die Richtung des Reflexionsvektors benutzt, um die Texturkoordinaten in einer Umgebungs-Textur zu bestimmen (Bild 13.18).

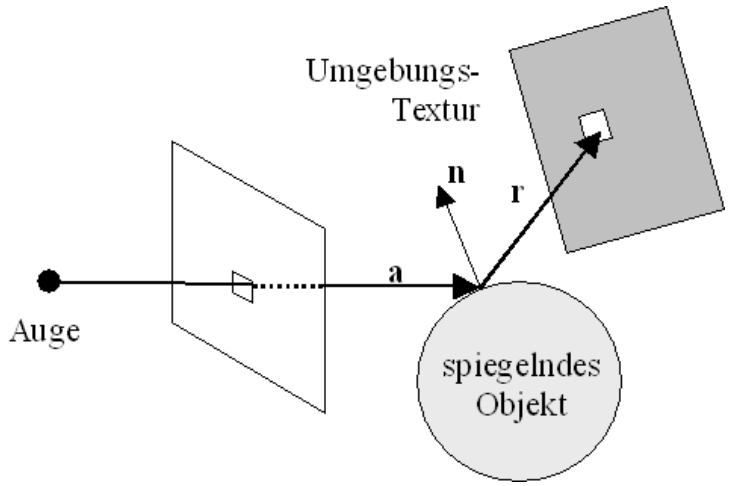


Bild 13.18: Das Prinzip des Environment Mappings: ein Beobachter blickt auf ein spiegelndes Objekt und sieht darin die Umgebung in der Richtung des Reflexionsvektors. Die Umgebung des Objekts wird in Form einer Umgebungs-Textur gespeichert.

Die Umgebungs-Textur enthält das Bild der Umgebung aus dem Blickwinkel des Objekts. Solange die Umgebung relativ weit entfernt von dem Objekt ist und sich die Umgebung nicht verändert, muss man sie nicht, wie beim Ray-Tracing, jedesmal neu berechnen, sondern kann sie in einer Textur abspeichern. Auch bei Objektbewegungen, die im Verhältnis zur Entfernung zwischen Objekt und Umgebung klein sind, kann die Umgebungs-Textur wieder verwendet werden. Im Umkehrschluss sind damit auch die Einschränkungen von *Environment Mapping* Techniken offensichtlich. Spiegelungen in Szenen, bei denen mehrere Objekte relativ nah beieinander sind, können nur für eine Objektkonstellation und einen Blickwinkel korrekt dargestellt werden. Falls sich Blickwinkel oder Objektpositionen ändern, müsste für jedes Bild und evtl. sogar für jedes Objekt vorab eine Umgebungs-

Textur erzeugt werden. Dies ist natürlich sehr rechenaufwändig und benötigt mehrere Durchläufe durch die Rendering Pipeline (*Multipass Rendering*, Abschnitt 13.1.1.2).

Die Berechnung der Texturkoordinaten für eine Umgebungs-Textur kann auf unterschiedlichen Genauigkeitsstufen ablaufen:

- per Vertex, dann müssen die pixel-bezogenen Texturkoordinaten wie üblich durch lineare Interpolation gewonnen werden.
- per Pixel, was zu besseren Ergebnissen führt, aber deutlich rechenaufändiger ist, da der Reflexionsvektor für jedes Pixel zu berechnen ist. Diese Variante setzt einen programmierbaren Pixel-Shader auf der Grafikkarte voraus.

In der interaktiven 3D-Computergrafik werden vor allem die zwei *Environment Mapping* Techniken eingesetzt, die auch in OpenGL verfügbar sind: Sphärische Texturierung (*Sphere Mapping*) und Kubische Texturierung (*Cube Mapping*).

13.4.1 Sphärische Texturierung (Sphere Mapping)

Die Grundidee der Sphärischen Texturierung (*Sphere Mapping*) besteht darin, eine ideal verspiegelte Kugel aus großer Entfernung mit einem starken Teleobjektiv zu betrachten, was im Grenzfall einer orthografischen Projektion entspricht [Mill84]. In der Kugel spiegelt sich deren gesamte Umgebung mit zunehmender Verzerrung zum Rand hin. Ein Foto dieser Kugel mit der gespiegelten Umgebung wird als Sphärische Textur (*Sphere Map*) bezeichnet. Die Kugel deckt innerhalb einer solchen Textur einen kreisförmigen Bereich mit Mittelpunkt $(0.5, 0.5)$ und Radius 0.5 ab (Bild 13.19-a). Die verspiegelte Kugel bildet die Richtung eines Reflexionsvektors auf einen Punkt der sphärischen Textur ab. Man benötigt also für die Berechnung der Texturkoordinaten diese Abbildung und den Reflexionsvektor.

Für jeden Reflexionsvektor $\mathbf{r}$ muss man Texturkoordinaten (s, t) bestimmen, die einem Punkt innerhalb des kreisförmigen Bereichs der sphärischen Textur entsprechen. Nachdem die sphärische Textur durch eine orthografische Projektion der Einheitskugel erzeugt wurde, ergibt sich der Zusammenhang zwischen den Texturkoordinaten (s, t) und einem Punkt (x, y, z) auf der Einheitskugel durch Skalierung um den Faktor 2 und Verschiebung um eine negative Einheit für die x- und die y-Koordinate, sowie aus der Kreisgleichung $r = 1 = \sqrt{x^2 + y^2 + z^2}$ für die z-Koordinate (Bild 13.19-a), d.h.:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 2s - 1 \\ 2t - 1 \\ \sqrt{1 - x^2 - y^2} \end{pmatrix} \tag{13.15}$$

Für die Einheitskugel im Ursprung gilt aber, dass die Einheitsnormalenvektoren $\mathbf{n^e}$ durch die Koordinaten (x, y, z) des jeweiligen Punkts auf der Oberfläche gegeben sind (Bild 13.19-b), d.h.:

$$\mathbf{n^e} = \begin{pmatrix} n_x^e \\ n_y^e \\ n_z^e \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 2s - 1 \\ 2t - 1 \\ \sqrt{1 - x^2 - y^2} \end{pmatrix} \tag{13.16}$$

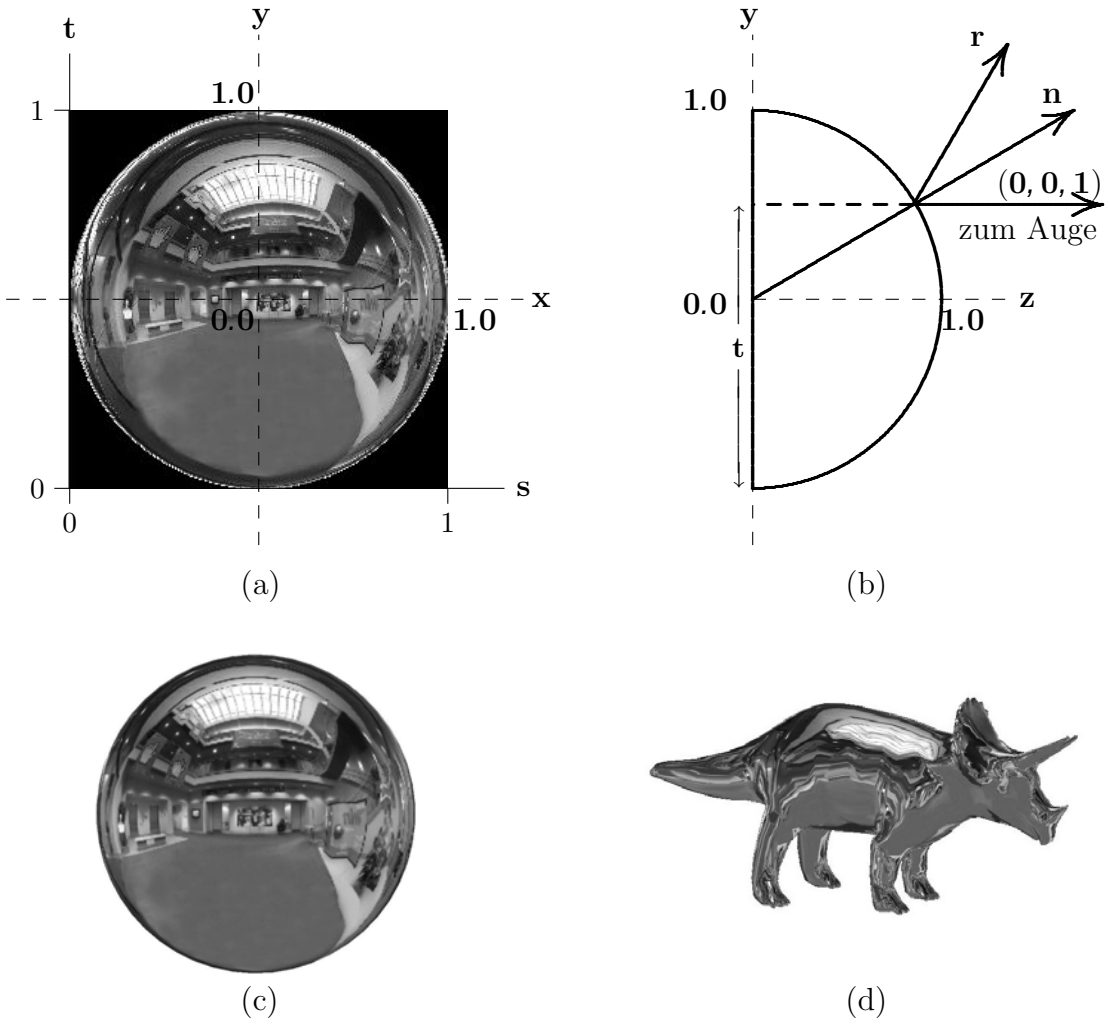


Bild 13.19: Sphärische Texturierung (*Sphere Mapping*): (a) Eine sphärische Textur ist das Foto einer spiegelnden Kugel. Die Kugel deckt in Texturkoordinaten einen kreisförmigen Bereich mit Mittelpunkt $(0.5, 0.5)$ und Radius 0.5 ab. (b) Die Seitenansicht der vorderen Halbkugel: der Normalenvektor $\mathbf{n}$ ist die Winkelhalbierende zwischen Augenpunktsvektor $(0, 0, 1)^T$ und Reflexionsvektor $\mathbf{r}$. Für die Einheitskugel im Ursprung gilt, dass die Koordinaten eines Punktes auf der Oberfläche gleich dem Einheitsnormalenvektor sind $((x, y, z)^T = (n_x^e, n_y^e, n_z^e)^T)$. (c) Das Mapping einer sphärischen Textur auf eine Kugel. (d) Das Mapping einer sphärischen Textur auf das Triceratops-Modell.

Als Nächstes bestimmt man den Zusammenhang zwischen dem Reflexionsvektor $\mathbf{r}$ und dem Normalenvektor $\mathbf{n}$. In Weltkoordinanten berechnet sich der Reflexionsvektor $\mathbf{r}$ aus dem Vektor $\mathbf{a}$ vom Augenpunkt zum Oberflächenpunkt und dem Normalenvektor $\mathbf{n}$ mit Hilfe des Reflexionsgesetzes (12.2) zu:

$$\mathbf{r} = \mathbf{a} - 2(\mathbf{a} \cdot \mathbf{n}) \cdot \mathbf{n} \tag{13.17}$$

Der Vektor $\mathbf{a}$ vom Augenpunkt zum Oberflächenpunkt ist in Weltkoordinaten durch die

negative z-Achse gegeben: $\mathbf{a} = (0, 0, -1)^T$. Setzt man dies in (13.17) ein, erhält man:

$$\begin{pmatrix} r_x \\ r_y \\ r_z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} - 2 \left(\begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} \cdot \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} \right) \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} + 2n_z \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} \quad (13.18)$$

Aufgelöst nach dem Normalenvektor ergibt:

$$\mathbf{n} = \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} = \frac{1}{2n_z} \begin{pmatrix} r_x \\ r_y \\ r_z + 1 \end{pmatrix} \quad (13.19)$$

Durch Normierung $|\mathbf{n}| = \sqrt{n_x^2 + n_y^2 + n_z^2} = \frac{1}{2n_z}\sqrt{r_x^2 + r_y^2 + (r_z+1)^2}$ entsteht der Einheitsnormalenvektor $\mathbf{n^e}$:

$$\mathbf{n^e} = \begin{pmatrix} n_x^e \\ n_y^e \\ n_z^e \end{pmatrix} = \frac{1}{\sqrt{r_x^2 + r_y^2 + (r_z+1)^2}} \begin{pmatrix} r_x \\ r_y \\ r_z + 1 \end{pmatrix} \quad (13.20)$$

Dieses Ergebnis ist sehr gut verständlich, da der Normalenvektor nichts Anderes ist als der winkelhalbierende Vektor zwischen dem Augenpunktsvektor $(0, 0, 1)^T$ und dem reflektierten Strahl $(r_x, r_y, r_z)^T$. Der Einheitsnormalenvektor ergibt sich daraus einfach durch Addition $(r_x, r_y, r_z+1)^T$ und Normierung (Bild 13.19-b). Durch Gleichsetzen von (13.16) und (13.20)

$$\begin{pmatrix} 2s - 1 \\ 2t - 1 \\ \sqrt{1 - x^2 - y^2} \end{pmatrix} = \begin{pmatrix} n_x^e \\ n_y^e \\ n_z^e \end{pmatrix} = \frac{1}{\sqrt{r_x^2 + r_y^2 + (r_z+1)^2}} \begin{pmatrix} r_x \\ r_y \\ r_z + 1 \end{pmatrix} \quad (13.21)$$

kann man die gesuchten Texturkoordinaten s, t in Abhänigkeit vom Reflexionsvektor $\mathbf{r}$ angeben:

$$s = \frac{r_x}{2\sqrt{r_x^2 + r_y^2 + (r_z+1)^2}} + \frac{1}{2} \quad (13.22)$$

$$t = \frac{r_y}{2\sqrt{r_x^2 + r_y^2 + (r_z+1)^2}} + \frac{1}{2} \quad (13.23)$$

Die Berechnung des Reflexionsvektors mit (13.17) und der Texturkoordinaten mit (13.22) und (13.23) wird von OpenGL vorgenommen, wenn die automatische Texturkoordinatengenerierung auf den Modus `GL_SPHERE_MAP` eingestellt wurde:

```
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);
```

Weitere Parameter, wie bei den anderen Generiermodi (`GL_OBJECT_LINEAR`, `GL_EYE_LINEAR`), gibt es in diesem Fall nicht. Die Ergebnisse, die man mit der sphärischen Projektion der Texturkoordinaten in Verbindung mit einer sphärischen Textur für konvexe spiegelnde Oberflächen erzielen kann, sind durchaus beeindruckend (Bild 13.19-c,d).

Sphärische Texturierung ist auf nahezu allen heutigen Grafikkarten verfügbar, da man nur eine normale Textur benötigt und die Texturkoordinatengenerierung im Rahmen einer Textur-Matrix implementiert werden kann. Sinnvolle Ergebnisse kann man jedoch nur erzielen, wenn die Textur sphärisch ist, d.h. das Bild einer spiegelnden Kugel enthält. Solch eine Textur kann aber relativ einfach erzeugt werden. Entweder durch Fotografieren der realen Umgebung mit einem extremen Weitwinkelobjektiv (Fischauge) bzw. einer versilberten Christbaumkugel mit einem starken Teleobjektiv oder durch 3D-Computergrafik z.B. mit Hilfe von Ray-Tracing-Verfahren bzw. Verzerrung planarer Texturen.

Sphärische Texturierung funktioniert bei konkaven Oberflächen nur unzureichend, da keine Eigenspiegelungen möglich sind. Ein weiterer Nachteil der sphärischen Texturierung ist, dass sie streng genommen nur für einen Blickwinkel gilt. Bewegt sich der Beobachter beispielsweise um das spiegelnde Objekt herum, sieht er nicht die andere Seite der Umgebung, sondern immer nur die gleiche. Daher entsteht der Eindruck, dass sich nicht der Beobachter um das Objekt bewegt, sondern dass sich das Objekt dreht. Diese Schwäche kann erst durch die im nächsten Abschnitt dargestellte kubische Texturierung beseitigt werden.

13.4.2 Kubische Texturierung (Cube Mapping)

Bei der kubischen Texturierung (*Cube Mapping*, [Gree86]) verwendet man nicht nur eine Umgebungs-Textur, wie bei der sphärischen Texturierung, sondern sechs 2-dimensionale Umgebungs-Texturen, die die Flächen eines Kubus' bilden. Im Zentrum des Kubus befindet sich das zu texturierende Objekt. Die sechs Einzeltexturen, die zusammen die kubische Textur bilden, werden einfach dadurch gewonnen, dass man die Umgebung aus der Position des Objektmittelpunkts sechs mal mit einem Öffnungswinkel von 90° fotografiert oder rendert, und zwar so, dass die sechs Würfelflächen genau abgedeckt werden. Die sechs Einzeltexturen müssen also an den jeweiligen Rändern übergangslos zusammen passen. In Bild 13.20 ist ein Beispiel für eine kubische Umgebungs-Textur dargestellt.

Eine kubische Textur wird formal als eine Bildmatrix $\mathbf{T}$ beschrieben, die von den drei Variablen (s, t, r) eines orthogonalen Texturkoordinatensystems abhängt, genau wie bei einer 3-dimensionalen Textur. Allerdings werden die Texturkoordinaten als Richtungsvektor betrachtet, der angibt, welches Texel man sieht, wenn man vom Zentrum in Richtung des Vektors geht (Bild 13.21). Möchte man wieder die Spiegelung der Umgebung in einem Objekt bestimmen, wird der Reflexionsvektor $\mathbf{r}$ als Richtungsvektor benutzt. Die Vektoren müssen in diesem Fall nicht normiert werden. Die Koordinate des Reflexionsvektors mit dem größten Absolutwert bestimmt, welche der sechs Texturen ausgewählt wird (z.B. wird durch den Vektor $(1.3, -4.2, 3.5)$ die $-Y$ Fläche ausgewählt). Die verbleibenden zwei Koordinaten werden durch den Absolutwert der größten Koordinate geteilt, so dass sie im Intervall $[-1, +1]$ liegen. Anschließend werden sie mit dem Faktor $\frac{1}{2}$ skaliert und um

(e7): In Teil 3 liegen nur Punkte mit $d_x = d_y = 0$. Teil 2 endet mit einem solchen Punkt. Sonst wie die fünfte Bedingung (90^o-Ecke).

Mit diesen Bedingungen wird für den Bildpunkt in der Position (x, y) ermittelt, ob eine Systematik vorliegt oder nicht.

(f) Berechnung des Kantenbildes durch eine Kombination des Rang- und des Systematikbildes wie oben beschrieben. Die Bewertung erfolgt dabei gemäß (20.10).

Ende des Algorithmus

Abschließend zu diesem Algorithmus noch einige Bemerkungen: Durch die Auswertung des Rang- und des Systematikbildes werden hier positive Eigenschaften der einzelnen Verarbeitungsschritte kombiniert. Sowohl Rang- als auch Systematikbild ermöglichen eine betragsunabhängige Kantendetektion. Die Rangauswertung liefert eine gute Geschlossenheit von Kantenzügen, während die Systematikauswertung bei der Entfernung von irrelevanten Kantenstücken in homogenen oder verrauschten Bildbereichen herangezogen wird. Beide Zwischenschritte (Rang und Systematik) und das Endergebnis sind Binärbilder (logische Bilder), in denen die Grauwerte der Bildpunkte die Bedeutung „liegt auf einer Kante“ und „liegt auf keiner Kante“ haben.

An vielen Stellen dieses Kantendetektors werden Schwellwerte verwendet. Das hat zur Folge, dass das Verfahren sehr sorgfältig auf die jeweilige Problemstellung zu adaptieren ist. Das Ergebnis wird dabei sicher verbessert, wenn in der Vorverarbeitung Störungen (Rauschen, Scannerfehler, usw.) korrigiert werden.

Da einzelne Verarbeitungschritte durchaus rechenzeitintensiv sind, vor allem, wenn größere Umgebungen verwendet werden, wird man bei Echtzeitanwendungen möglicherweise Probleme bekommen. Hier kann die Verwendung von sehr schnellen allgemeinen Prozessoren, die Parallelisierung von Teilabläufen oder die Verwendung von Spezialhardware Abhilfe schaffen. Bei vielen praktischen Anwendungen ist es außerdem nicht notwendig, den gesamten Bildbereich zu verarbeiten, sondern die Verarbeitung kann auf einen kleinen Bildausschnitt (*area-of-interest*) beschränkt werden.

20.6 Der Canny-Kantendetektor

Einen Kantendetektor mit einigen interessanten Eigenschaften stellt der Canny-Kantendetektor dar. Er wurde von Canny 1983 [Cann83, Cann86] vorgestellt. Damals waren schon eine Vielzahl von Operatoren bekannt, Canny's Beitrag war, mit einem mathematisch-signaltheoretischen Ansatz aus (mathematisch formulierten) Designkriterien einen bezüglich dieser Kriterien optimalen Kantendetektor abzuleiten.

Informell ausgedrückt sind dies folgende Kriterien:

- Geringe Fehler erster und zweiter Art, d.h. es sollen keine Kanten übersehen werden und keine Kanten an Stellen detektiert werden, an denen sich keine befinden,

- Genaue Lokalisierung von Kanten, d.h. ein Detektor sollte Kanten möglichst genau dort lokalisieren, wo sie auch im Originalbild zu finden sind,
- Nur eine Antwort des Detektors auf eine Kante, nicht mehrere Antworten an mehreren Stellen (im wesentlichen als Ergänzung zum zweiten Kriterium).

Canny betrachtet dabei das eindimensionale Signal senkrecht zur jeweils betrachteten Kante im Bild. Als bezüglich der obigen Kriterien optimalen Kantendetektor ermittelt er das Maximum der Ableitung des rauschgefilterten Signals.

Die Realisierung des Canny-Kantendetektors erfolgt - wie schon in Canny's Originalarbeit skizziert - meist in einem dreischrittigen Verfahren:

- Glättung des Bildes mit einem Gauß'schen Kern
- Bestimmung des Gradienten (Kantenfilterung)
- Kantentracking (Nicht-Maxima-Unterdrückung und Tracking mit Hysterese)

A20.3: Canny-Kantendetektor.

Voraussetzungen und Bemerkungen:

◇ $\mathbf{S}_e = (s_e(x, y))$ ein Grauwertbild;

◇ Eingabeparameter des Canny-Kantendetektors: σ: Standardabweichung für die Glättung mit einem Gauß'schen Kern; h, l: oberer und unterer Schwellwert für das Kantentracking mit Hysterese;

Algorithmus:

(a) Glätte $\mathbf{S}_e$ mit einem Gauß'schen Kern mit Standardabweichung σ;

(b) Bestimmung des Gradienten (Kantenfilterung):

(ba) Wende auf $\mathbf{S}_e$ Kantenfilteroperatoren in zwei orthogonalen Richtungen an;

(bb) Bilde aus dem Ergebnis Gradientenbetrag und -richtung;

(c) Kantentracking:

(ca) Unterdrücke im Gradientenbetragsbild diejenigen Pixel, die entlang der Gradientenrichtung kein Maximum darstellen;

(cb) Verfolge Kanten mit Hysterese: Ausgehend von Pixeln, die über der oberen Schwelle h liegen, markiere alle Pixel, die über der unteren Schwelle l liegen als zur Kante gehörend.

Ende des Algorithmus

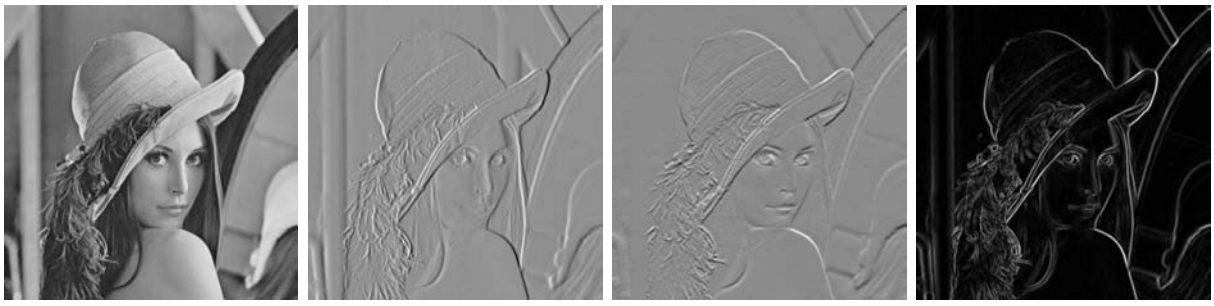
Bei der Glättung des Bildes wird ein Gauß'scher Kern mit - je nach Verrauschtheit des Bildmaterials - vorwählbarer Standardabweichung σ verwendet. Diese ist einer der Parameter des Canny-Kantendetektors. Ziel dieses Schrittes ist die Verminderung des Bildrauschens, da die hochfrequenten Rauschanteile sich nachteilig auf den nachfolgenden Schritt der Gradientenbildung auswirken.

Zur Gradientenbildung wird meist einer der bekannten Kantenfilteroperatoren in zwei orthogonalen Richtungen angewendet (Robert's Cross, Prewitt, Sobel, Bild 20.12-b,c), um daraus Betrag und Richtung des Gradienten zu bestimmen.

Im Gradienten-Betragsbild (Bild 20.12-d) werden dann diejenigen Pixel "unterdrückt", d.h. zu Null gesetzt (*non-maxima-suppression*), die entlang der Gradientenrichtung (also senkrecht zur Kante) kein Maximum aufweisen. Canny schlägt vor, für die Bestimmung der Nachbarpixel entlang des Gradienten zwischen den entsprechenden Pixeln der 8-er Nachbarschaft zu interpolieren. Allerdings wird hierzu auch häufig die Gradientenrichtung meist in die vier (betragsfreien) Richtungen der Achter-Nachbarschaft (*horizontal, vertikal, entlang der Hauptdiagonalen, entlang der Nebendiagonalen*) diskretisiert und die Maximumsunterdrückung entlang der drei benachbarten Pixel in dieser Richtung durchgeführt.

Auf den übrigbleibenden, Kanten im Bild entsprechenden Pixeln wird dann ein Linienverfolgungsverfahren mit Hysterese angewendet: Um schwache, oft auch aus Restrauschen entstehende, Kanten zu unterdrücken, werden nur diejenigen Kanten verwendet, bei denen mindestens ein Pixelwert über einer oberen Schranke h liegt. Um für diese Kanten Unterbrechungen durch schwächere Kantenbereiche zu vermeiden, wird eine Hysterese eingeführt, d.h. für eine einmal erkannte Kante reicht es aus, wenn alle Pixel über einer unteren Schranke l liegen.

Die obere und untere Schwellwerte (h, l) bilden neben der oben erwähnten Standardabweichung die anderen beiden der drei Parameter des Canny-Kantendetektors.



(a) (b) (c) (d)

Bild 20.12: (a) Die Original-Lena, (b) Gradient dx, (c) Gradient dy, (d) Betrag des Gradienten

(a) (b) (c) (d)

Bild 20.13: Auswirkungen unterschiedlicher Standardabweichungen zum Glätten des Bildes: (a) $\sigma = 1.0$, (b) $\sigma = 2.0$, (c) $\sigma = 3.0$, (d) $\sigma = 4.0$,

(a) (b) (c) (d)

Bild 20.14: Auswirkung unterschiedlicher Hystereseschwellen für das Kantentracking, leichte Glättung ($\sigma = 4.0$), starkes verbleibendes Rauschen: (a) $h = 0.8, l = 0.3$,(b) $h = 1.0, l = 0.6$,(c) $h = 1.0, l = 0.8$,(d) Detail mit offenen Y-Verbindungen

(a) (b) (c) (d)

Bild 20.15: Auswirkung unterschiedlicher Hystereseschwellen für das Kantentracking, starke Glättung ($\sigma = 4.0$), leichtes verbleibendes Rauschen: (a) $h = 1.0, l = 0.1$,(b) $h = 0.3, l = 0.3$,(c) $h = 1.0, l = 0.3$,(d) $h = 1.0, l = 0.6$

Sollte die Topologie der detektierten Kanten in der jeweiligen Anwendung eine besondere Rolle spielen, so ist auf ein spezielles 'Feature' des Canny-Kantendetektors zu achten: Bedingt durch die Nicht-Maxima-Unterdrückung beim Kantentracking werden 'Y'-Verbindungen dreier (oder Kreuzungen mehrerer) Kanten nicht vollständig detektiert (siehe Detail aus der rechten, oberen Ecke des Originalbilds im Teilbild 20.14-d). Hier ist gegebenenfalls ein Nachbearbeitungsschritt zum Schließen der Y-Verbindungen oder eine alternative Trackingstrategie erforderlich.

20.7 Kanten und Linien mit morphologischen Operationen

Die mathematische Morphologie stellt der digitalen Bildverarbeitung und Mustererkennung mächtige Instrumente zur Bewältigung vieler Problemstellungen zur Verfügung. Die Grundlagen dazu sind in Kapitel 19 zusammengestellt. In diesem Abschnitt werden morphologische Verfahren erläutert, die im Zusammenhang mit der Verarbeitung von Kanten und Linien interessant sind.

Zuvor aber noch eine Zusammenfassung, wie Kanten- oder Linienbilder entstehen können: Grauwertkanten treten in Grauwertbildern an Helligkeitsübergängen auf, die eine gewisse Systematik aufweisen. Grauwertkanten können aber auch, wie schon oben dargestellt, das Ergebnis einer Textur- oder Farbmerkmalsverarbeitung sein. Wird auf ein derartiges Grauwertbild ein Verfahren zur Kantenextraktion angewendet (Abschnitte 18.4, 20.3 und 20.4), so sind im Ergebnisbild die markanten Grauwertübergänge in der Regel durch Linien dargestellt. Das so erzeugte Linienbild kann ein Grauwertbild oder ein Binärbild sein. Ist es ein Grauwertbild, so sagt ein hoher Wert aus, dass die Wahrscheinlichkeit, dass der Bildpunkt auf einer Grauwertkante liegt, hoch ist. Bei einem binären Linienbild bedeutet der Grauwert 0, dass der Bildpunkt zum Hintergrund, und der Grauwert 1, dass er zu einer Kante gehört.

Linienbilder können natürlich auch das direkte Ergebnis einer Digitalisierung sein, so z.B. wenn eine Strichgrafik entweder per Software oder mit einem Digitalisierungsgerät (Videosensor, Scanner) in ein Rasterbild umgewandelt wird.

20.7.1 Extraktion des Randes von Segmenten

Ein einfaches Verfahren, wie man aus einem Binärbild, das unterschiedliche Segmente enthält, den Rand dieser Segmente gewinnen kann, ist eine Dilatation oder Erosion mit einer anschließenden Differenz zum Originalbild. Vorausgesetzt wird ein Binärbild (Zweipegelbild), in dem sich die Segmente dunkel (Grauwert 0) vor einem hellen Hintergrund (Grauwerte 1 oder 255) abheben.

Bei einer Dilatation und einer anschließenden Differenzbildung des dilatierten Bildes mit dem Original erhält man den inneren Rand der Segmente, während bei der Erosion und einer anschließenden Differenzbildung Ränder erzeugt werden, die die Segmente ganz enthalten (Bilder 20.16).