Otto-von-Guericke-Universität Magdeburg

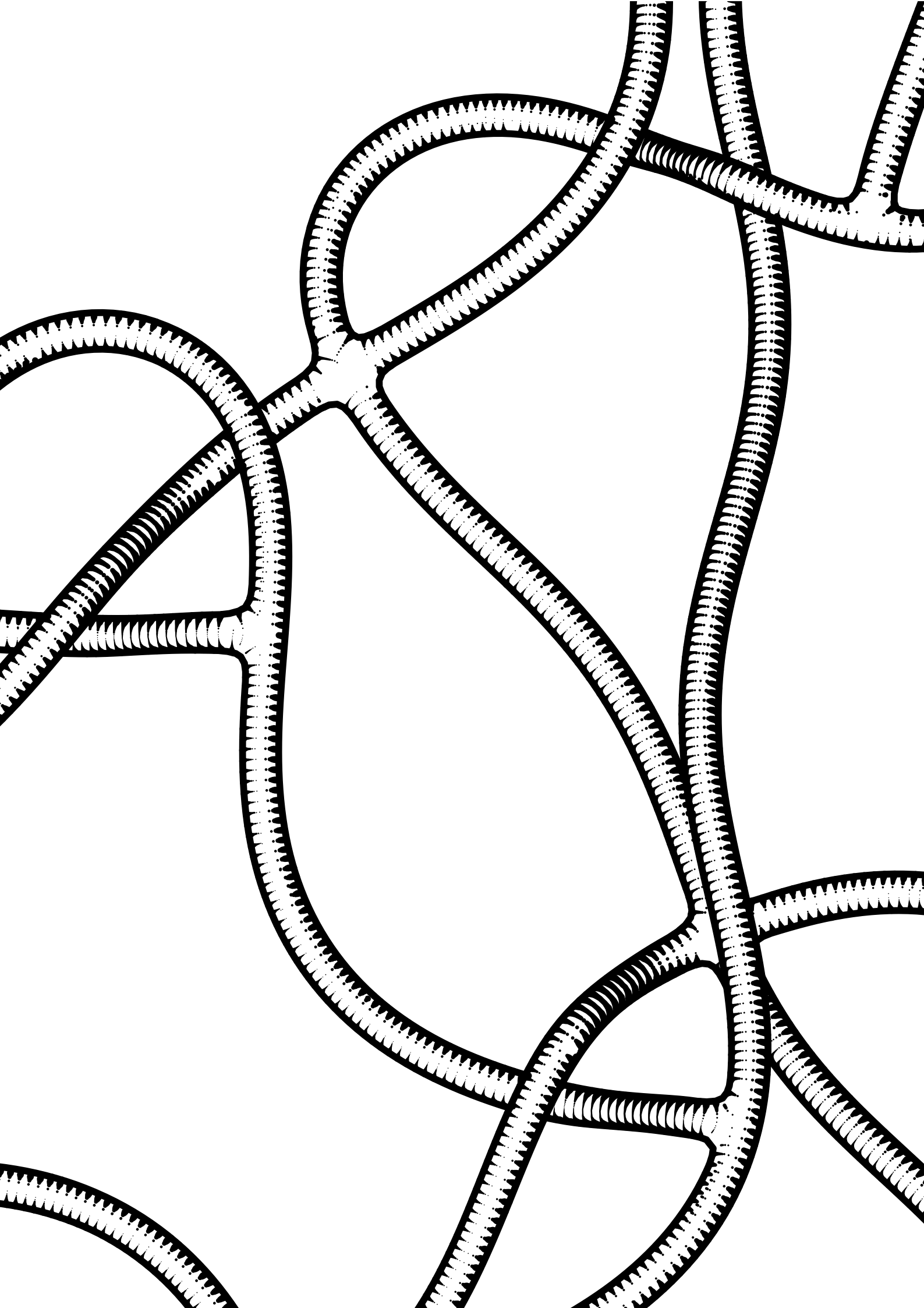
Fakultät für Informatik
Institut für Simulation und Graphik

Studienarbeit

Erstellung qualitativ hochwertiger Illustrationen mit OpenNPAR

Johannes Zander
No. 158281

Betreuer:
Dipl.-Inf. Tobias Isenberg
Dr.-Ing. Stefan Schlechtweg



Inhalt

0. Einleitung 5
1. Illustration 6
2. Verwandte Arbeiten 8
3. Richtungsfelder 10
3.1 Krümmungsanalyse 10
3.2 Optimierung 11
3.3 Nachbarschaftliche Spannungen 12
3.4 Umgebungs Spannung 13
3.5 Minimierung 13
4. Stromlinien 15
4.1 Integration 15
4.2 Abbruchkriterien 16
4.3 Liniennähe 17
4.4 Implementierungs-Details 19
5. Visualisierung 20
5.1 Verdeckte Linien 20
5.2 Schattierung 20
5.3 Virtuelle Maschine 22
6. Vektor-Linien 23
6.1 OpenGL 23
6.2 Vierecke 23
6.3 Gehrungsgrenze 24
6.4 Runde Linien 24
6.5 Implementierungs-Details 24
6.6 Negative Linienstärke 26
6.7 Punktierte Linien 26
6.8 Punkt-Abstände 26
6.9 Minimale Punkt-Abstände 27
6.10 Ausgabe 28
7. OpenNPAR 30
8. Beispiele 32
9. Fazit 34
Literatur 36
Bild-Referenzen 37
WWW-Referenzen 38

0. Einleitung

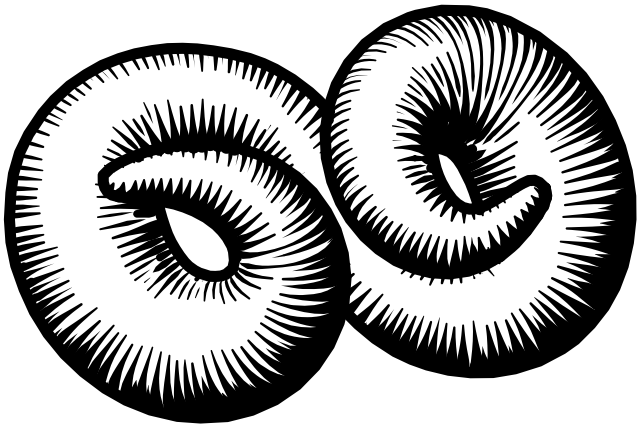
Über lange Zeit war ein Kerngebiet der Computergraphik das photorealistische-Rendering, also das Erzeugen von Bildern die Photographien nahe kommen. Im Gegensatz dazu entwickelt sich das nicht-photorealistische-Rendering (NPR) jedoch in eine andere Richtung, nämlich der Nachahmung von Bildern und Graphiken, die von Menschen entworfenen wurden. Ein Teilgebiet davon ist das Imitieren von klassischen Zeichentechniken. Das ist auch das Anliegen dieser Arbeit, wobei der Schwerpunkt auf die Schattierung von polygonalen Objekten gelegt wurde. Letztlich lassen sich so aus vorhandenen 3D-Modellen auflösungsunabhängige Vektorzeichnungen gewinnen, die sich dann weiterverarbeiten lassen.

Die grundlegende Idee dieser Arbeit ist, die Oberfläche eines polygonalen Netzes gleichmässig mit Linien zu überziehen, die jedoch nicht willkürlich verlaufen, sondern die Form betonen. Für die Darstellung werden Sichtbarkeit und Schattierung dieser Linien berükksichtigt, um das endgültige Bild zu erstellen.

Um dies zu realisieren ergeben sich die folgenden Schritte: In einer Vorverarbeitungsphase wird erst bestimmt, in welche Richtungen die Linien laufen sollen (Kapitel 3). Diese werden dann im Objektraum möglichst gleichmässig auf der Modell-Oberfläche plaziert (Kapitel 4). Nach der Vorverarbeitung werden nicht sichtbare Linien entfernt und die Strichstärke der Linien festgelegt (Kapitel 5). Dabei wird auf ein zur Laufzeit veränderbares Schattierungsmodell zugegriffen, durch das eine Reihe unterschiedlicher Effekte erzielt werden können. Die Darstellung selbst (Kapitel 6) erfolgt letztlich entweder interaktiv in einem Vorschau-Modus oder es wird eine qualitativ hochwertige Vektor-Beschreibung angelegt, die dann z.B. für den Print-Bereich weiterverwendet werden kann.

Die im folgenden beschriebenen Algorithmen wurden in C++ realisiert. Dazu wurde ein vorhandenes Software-System zur Darstellung von nicht-photorealistischem-Rendering umfangreich erweitert. Bei dem System handelt es sich um das NPR Framework OpenNPAR, dass am Institut für Simulation und Graphik der Otto-von-Guericke-Universität Magdeburg entwickelt wurde (Kapitel 7).

Ein grosses Dankeschön an meine beiden Betreuer *Tobias Isenberg* und *Stefan Schlechtweg* für ihre hervorragende Betreuung und Zusammenarbeit sowie an *Petra Neumann* für das Modell der Kannenpflanze.



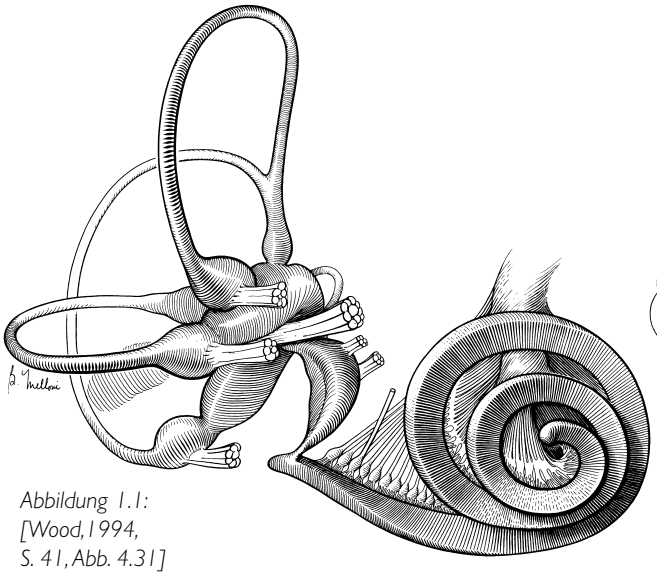


Abbildung 1.1:
[Wood, 1994,
S. 41, Abb. 4.31]

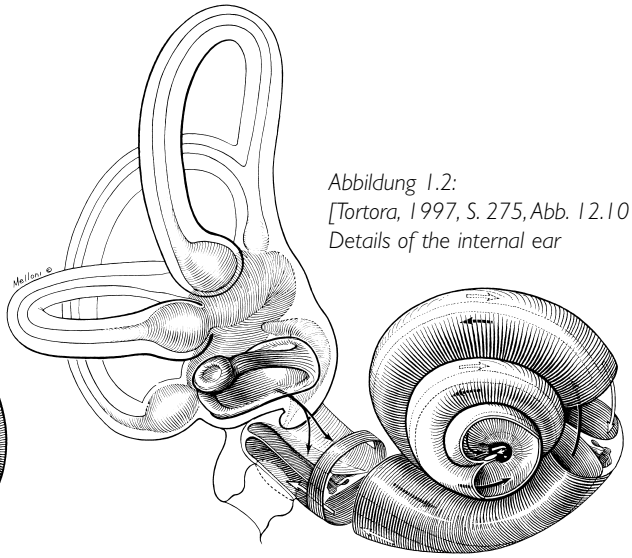


Abbildung 1.2:
[Tortora, 1997, S. 275, Abb. 12.10
Details of the internal ear

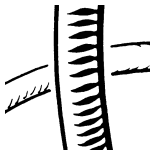
1. Illustration

Möchte man sich an der Nachahmung von Illustrationen versuchen, so ist der erste und wichtigste Schritt eine genaue Betrachtung des echten illustrativen Handwerks. Dabei sollte das Augenmerk insbesondere auf technische und stilistische Besonderheiten gelegt werden.

Um die Unterschiede zwischen den Darstellungen von ein und dem selben Objekt zu zeigen, werden drei unterschiedliche Versionen des Innenohrs und zusätzlich noch eine schematische Zeichnung des Kleinhirns untersucht, um einige alternative Stilmittel zu verdeutlichen:

Abbildung 1.1

Bei dieser Abbildung handelt es sich um eine sehr hoch detaillierte Tuschzeichnung. Eine Betonung der Kontur findet für die primäre Form statt, mit zunehmender Entfernung wird die Linie jedoch dünner und enthält an Überschneidungen sogar Auslassungen zur klareren Trennung nicht zusammenhängender Formen.



Zur besseren Verdeutlichung von Plastizität und Form wurde die Oberfläche in weiten Teilen mit einem gleichmässigen Linienmuster überzogen. Dabei folgen die Linien wie schon von [Hodges, 1998, S.99] beschrieben Ellipsen, die sich entlang der Strukturen ziehen. Diese Linien überwinden jedoch keine stärkeren Knicke in der Oberfläche, sondern enden dort. So wird die Diskontinuität aneinandergrenzender Flächen weiter betont. Durch die Variation der Linienstärke werden Schatten und Schattierung wiedergegeben. Stärkere Aufhellungen erfolgen durch Auslassungen, was besonders an den Glanzlichtern deutlich wird.

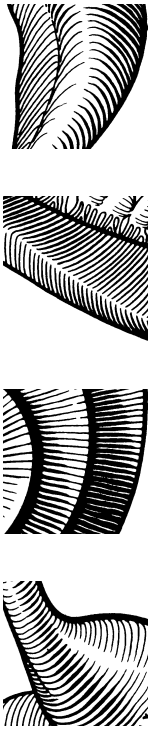


Abbildung 1.2

Da diese Zeichnung vom gleichen Illustrator stammt wie die vorhergehende, sind Ähnlichkeiten nicht von der Hand zu weisen. Es wurde jedoch mehr Augenmerk auf verborgene Strukturen gelegt. Diese wurden mit Hilfe von Transparenz und Schnitten offengelegt.

Abbildung 1.3:
[Rogers, 1992, S. 58, Abb. 4.15]
The right inner ear
(von Hand erstellte Rekonstruktion)

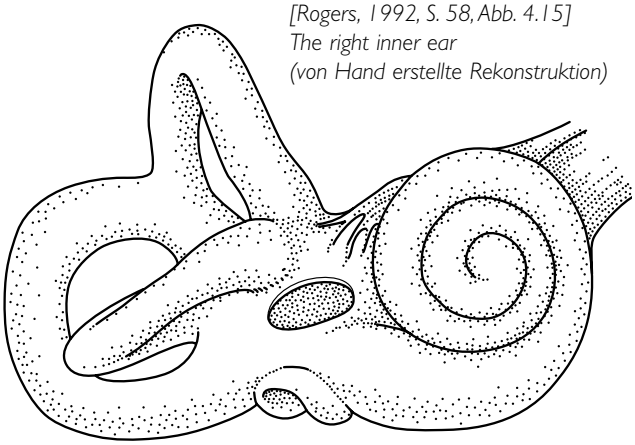
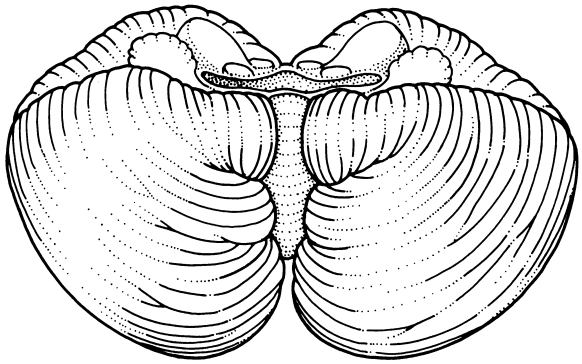


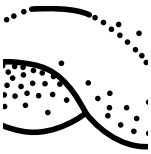
Abbildung 1.4:
[Rogers, 1992, S. 324, Abb. 21.9b]
The cerebellum



Auch wurde versucht, Prozesse die im Inneren ablaufen, mit Pfeilen zu verdeutlichen. So laufen schwarze und weisse Pfeile durch die verschiedenen Kammern im Inneren der Schnecke. Dinge, die für die zu kommunizierenden Ziele nicht wichtig sind, wurden dagegen weggelassen. Die Ansätze von Nervensträngen fehlen hier zum Beispiel vollständig.

Abbildung 1.3

Diese Arbeit unterscheidet sich grundsätzlich von den beiden vorhergehenden. Es wurde bedeutend weniger Wert auf die Wiedergabe von Details gelegt, sondern mehr auf die allgemeine Form. Linien geben nur die Kontur wieder und haben eine gleichmässige Linienstärke. Teilweise gehen sie auch in Punkte über, besonders wenn die Kontur in die Grundform stösst oder zur Andeutung von Knicken in der Oberfläche. Ein anderer wichtiger Unterschied ist die Art der Schattierung, denn diese wird nur über Punktieren erreicht.



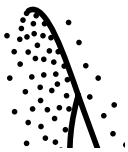
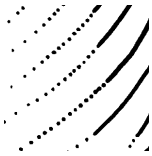


Abbildung 1.4

In dieser Zeichnung wird schematisch das Kleinhirn gezeigt. Ähnlich wie in der vorhergehenden Illustration wird auch hier Schattierung nicht durch Modulation der Linienstärke erreicht. Stattdessen werden Linien in Punktfolgen überführt, um den Tonwert zu variieren. Dies tritt zum einen bei Linien auf, die stärkere Knicke in der Oberfläche repräsentieren und so zu einer charakteristischen Texturierung führen und zum anderen auch bei der flächigen Schattierung. Dabei ist vor allem anzumerken, dass bei der Punktierung nicht in alle Richtungen gleichmässig voneinander entfernte Punkte verwendet wurden, sondern diese entlang von quasi parallelen Linien verlaufen.



2. Verwandte Arbeiten

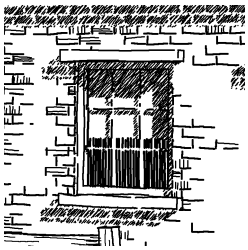
Der Bereich der nicht-photorealistisch darstellenden Computergraphik entwickelt sich schon über einen Zeitraum von gut 20 Jahren. Dementsprechend existiert inzwischen auch ein reicher Fundus an Arbeiten, deren Ziele und Methoden denen dieser Arbeit ähneln. Daher folgt eine Übersicht an relevanten Techniken:

[Salisbury, 1994]

Es werden Stroke-Textures angelegt, in denen Striche gespeichert werden, um eine bestimmte Struktur zu erzielen. Dabei ist es auch möglich, den einzelnen Strichen Prioritäten zuzuordnen, so dass Striche zu Gruppen zusammengefasst werden, die jeweils einen bestimmten Tonwert repräsentieren. Dabei haben Striche höherer Priorität den Vorrang vor solchen mit niedrigerer. Zur Darstellung wählt der Benutzer eine Textur und kann mit der Maus Tonwerte auftragen, wobei automatisch solange Striche aus der Textur gewählt werden, bis der gewünschte Tonwert erreicht wird.

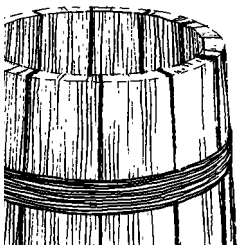
[Winkenbach & Salesin, 1994]

Die Technik aus [Salisbury, 1994] lässt sich auch verwenden um 3D Modelle mit Texturen zu überziehen. Zusätzlich zum Tonwert ist es möglich den Detailgrad gezielt über vom Benutzer plazierten Detail-Segmente zu steuern, wodurch Auslassungen nachgeahmt werden, die bei echten Illustrationen üblich sind. Ein Vorteil der Technik ist, dass sich die Dichte der Striche gezielt an unterschiedliche Ausgabe-Auflösung anpassen lässt.



[Winkenbach & Salesin, 1996]

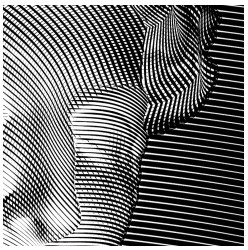
Anstelle von polygonalen Modellen wurde der Renderer aus [Winkenbach & Salesin, 1994] so erweitert, dass er auf Modellen mit parametrischen Oberflächen arbeitet. Die Parametrisierung lässt sich nutzen, um den Verlauf von Strichen auf der Objekt-Oberfläche festzulegen. Um einen gleichmässigen Tonwert der Striche auf der Oberfläche erreichen zu können, wird eine Metrik definiert, über die Abstände zwischen den in den Bildraum projizierten Strichen gemessen werden kann und über die die Linienstärke angepasst wird.



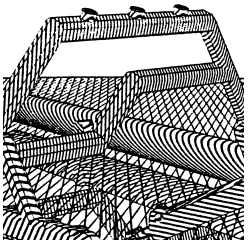
[Pnueli & Bruckstein, 1994]

Ein Graustufenbild wird anhand von Potentialfeldern in Niveaulinien zerlegt, so dass diese einer Rasterung des Bildes entsprechen und Bildinhalten folgen. Gleichzeitig ist es auch möglich, diese Linien in Punktfolgen zu überführen, um Techniken nachzuempfinden, die von Illustratoren benutzt werden, um Halbtonbilder zu erstellen.

[Oustromoukhov, 1999]



Von Hand werden auf einem gegebenen Graustufenbild mit Béziersplines Regionen umschlossen und mit verzerrbaren Sägezahn-Streifenmuster gefüllt und interaktiv so verzerrt, dass sie möglichst entlang der Hauptmerkmale des Bildes verlaufen. So wird das Bild schrittweise in einzelne Schichten zerlegt, die dann über verschiedene Verknüpfungs-Modi kombiniert werden und so Gravurebenen erzeugen, die als Grundlage verwendet werden, um das Bild zu rastern.

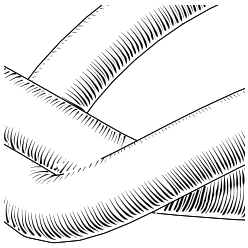


[Leister, 1994]

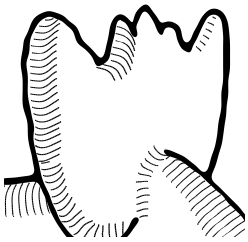
Der Ansatz basiert auf einem modifizierten Raytracer, bei dem insbesondere der Schritt verändert wurde, in dem die Farbinformation für die jeweiligen Pixel des Bildes erzeugt werden. Anstelle physikalische Beleuchtungs-Prozesse zu simulieren, wird eine prozedurale Textur ausgewertet, wodurch für Gravuren typische Streifenmuster entstehen, die in einem finalen Schritt binarisiert werden müssen und durch Hinzufügen von Objektkonturen vervollständigt werden.

[Deussen, 1999]

Von der Geometrie wird ein Innenskelett abgeleitet. Entlang des Skelettes und orthogonal zu diesem werden Schnittebenen mit dem Modell geschnitten. Die entstehenden sichtbaren Schnittkanten werden in einem Postprocessing von der Linienstärke an die Schattierung angepasst und dargestellt. Das Verfahren kann jedoch leider nur auf nicht verzweigende Strukturen angewendet werden, was die Einsatzmöglichkeiten einschränkt.

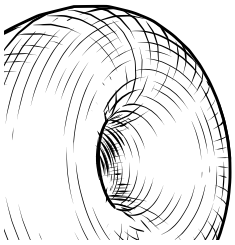


[Rössl & Kobbelt, 2000]



Für alle Knoten des Modells werden mit Hilfe diskreter Krümmungsanalyse die Hauptkrümmungen bestimmt. Diese werden über das Modell linear interpoliert und aus Kammerasicht in einen G-Buffer projiziert. Bildteile können dann interaktiv mit Linien überzogen werden, die sich an diesen Hauptkrümmungen orientieren. Um Turbulenzen und Rauscheinflüsse zu unterdrücken, werden davon jedoch nur einige markante Linien gewählt und die dazwischen liegenden durch Interpolation bestimmt, welche dann als Grundlage für die finale Darstellung benutzt werden.

[Hertzmann & Zorin, 2000]



Auch hier wird mittels diskreter Krümmungsanalyse die Hauptkrümmung für jeden Knoten bestimmt, aus der jedoch ein Richtungsfeld für Kreuzschraffuren konstruiert wird, welches mit einer globalen Optimierungs-Strategie verbessert wird, so dass das gesamte Feld möglichst homogene Richtungen enthält. Dieses Feld wird über die Kamerasicht in die Ebene projiziert, wo dann Feldlinien entlang beider Richtungen verfolgt werden, um Kreuzschraffuren zu erhalten.

3. Richtungs-felder

Da die Linien auf der Modell-Oberfläche nicht in willkürlichen Richtungen verlaufen sollen sondern diese bestimmt sein sollen, braucht es zuerst eine Form mit der diese über die gesamte Oberfläche beschrieben werden können. Dazu wird ein Tensor Feld verwendet, das im weiteren als Richtungsfeld bezeichnet wird. In diesem wird für jeden Knoten eines Modells mit Hilfe eines Einheitsvektors in der Tangentialebene die bevorzugte Richtung festgehalten.

Es gibt viele verschiedene Möglichkeiten die Richtungen in diesem Feld festzulegen, so wäre es z.B. denkbar die Richtung eines Innenskeletts als Grundlage zu verwenden (siehe [Deussen, 1999]), aber in dieser Arbeit wurde der Fokus auf die Krümmungsanalyse gelegt. Die Krümmung bietet sich an, da sie direkt von der Objekt-Geometrie abgeleitet werden kann und dazu bereits mehrere Arbeiten existieren, die erfolgreich die Krümmung benutzen, um den Verlauf von Linien festzulegen (siehe [Interrante, 1997], [Girshick, 2000] und [Herzmann & Zorin, 2000]). Diese Arbeiten stützen sich dabei auch auf psychologische Studien, die den Schluss nahelegen, dass Linien entlang der Hauptkrümmungsrichtung besonders gut dazu geeignet sind, dem Betrachter auch noch nach der Projektion in die Ebene die dreidimensionale Form des Objekts zu verdeutlichen.

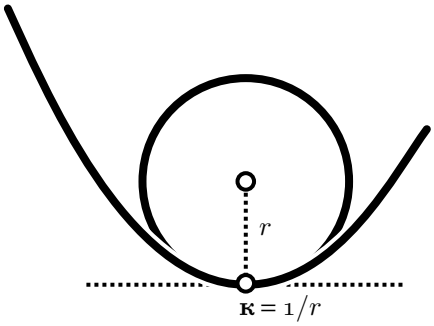


Abbildung 3.1: Krümmungskreis an einem Funktions-Graphen.

3.1 Krümmungsanalyse

Die Krümmung definiert sich bei einem Funktions-Graphen über den *Krümmungskreis*. Dieser verläuft an einem Kurvenpunkt tangential zum Graphen (siehe Abb. 3.1) und weist die gleiche erste und zweite Ableitung auf. Der Kehrwert des Kreisradius entspricht direkt der *Krümmung* κ, wobei das Vorzeichen Auskunft über den Drehsinn gibt: für ein positives κ ergibt sich eine Linkskurve und entsprechend für ein negatives eine Rechtskurve.

Um dieses Konzept auf Flächen zu erweitern, wird eine Flächenkurvenschar durch einen Flächenpunkt betrachtet, die sich durch den Schnitt der Fläche mit allen Ebenen ergibt, die sowohl den Flächenpunkt als auch den dazugehörigen Normalenvektor enthalten; dies entspricht letztlich einer Drehung einer Ebene, die diese Eigenschaft erfüllt, um den Normalenvektor. Die Krümmungen der einzelnen Kurven der Flächenkurvenschar lassen sich wie gehabt bestimmen, wobei zwei der Flächenkurven von besonderer Bedeutung sind: die Flächenkurve mit maximaler und minimaler Krümmung, deren Krümmungen respektive mit $\kappa 1$ und $\kappa 2$ bezeichnet werden. Die Richtungen, in denen die zu $\kappa 1$ und $\kappa 2$ gehörenden Flächenkurven verlaufen, heissen *Hauptkrümmungs-Richtungen* und sind zueinander orthogonal und spannen, bedingt durch ihre Konstruktion, die Tangentialebene auf, bil-

den also zusammen mit dem Normalenvektor eine orthonormale Basis. Das Produkt von $\kappa 1$ und $\kappa 2$ wird als *gausssche Krümmung* bezeichnet und der Mittelwert von $\kappa 1$ und $\kappa 2$ als *mittlere Krümmung*.

Versucht man dies jedoch auf Polygonnetze zu übertragen, so stellt man fest, dass bei einer diskreten Darstellung allein schon die Bestimmung eines Krümmungskreise auf Polygonzügen zu Problemen führt. Diese hätten an Knoten einen Radius von 0 und innerhalb der Liniensegmente einen unendlich Radius. Daher versucht man in der Praxis einen Polygonzug lokal über eine Kurve anzunähern, welche dann wie gehabt analysiert werden kann. Bei Flächen geht man ähnlich vor und definiert z.B. analog eine Quadrik durch einen Knoten und seine Nachbarn. In dieser Arbeit wurde ein ähnliches Verfahren verwendet, das jedoch darauf beruht ein Taylorpolynom zweiter Ordnung an einen Knoten und seine Nachbarn anzupassen (siehe auch [Rössl, 1999]). Dieses Verfahren war in ähnlicher Form im zugrundeliegenden Software-System schon implementiert und konnte sich dort in der Praxis bereits als äusserst robuster Algorithmus für die diskrete Krümmungsanalyse bewähren, weshalb es auch für diese Arbeit benutzt wurde.

Neben der Bestimmung der Krümmung selbst ist es jedoch auch wichtig, diese Information zu nutzen, um daraus für jeden Knoten bestimmen zu können, in welche Richtung eine Linie an dieser Stelle verlaufen soll. Bei handgefertigten Illustrationen beobachtet man dabei oft, dass Linien zylindrisch um elongierte Strukturen verlaufen (siehe auch Abb. 1.1 und 1.2). Das hat den Vorteil, dass insbesondere die durch die Projektion entstehende Verzerrung von Kreisscheiben Aufschluss über die Orientierung der zugrundeliegenden Struktur gibt (siehe [Hodges, 1989, S. 99]). Möchte man einen ähnlich Effekt erzielen, so reicht eine einfache Heuristik, um die Richtung festzulegen.

Dabei wählt man je nach Form der Oberfläche entweder die minimale oder maximale Krümmungsrichtung. Da immer die Krümmungsrichtung mit der stärkeren absoluten Krümmung von Interesse ist, reicht es aus, als Indikator das Vorzeichen der mittleren Krümmung zu betrachten, da sich bei einer Addition immer der Wert mit dem grösseren Vorzeichen behaupten kann und das Vorzeichen des Terms diktiert. Ist es positiv, so wählt man die maximale Krümmungsrichtung, andernfalls die minimale (siehe Abb. 3.2).

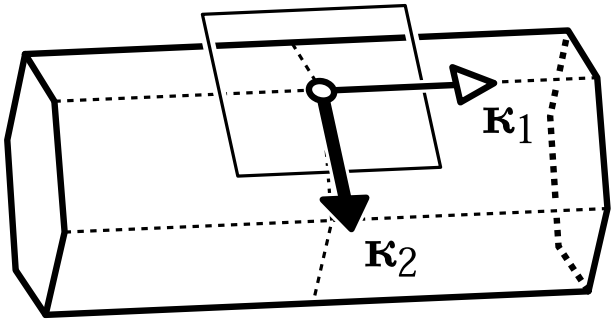


Abbildung 3.2: Wahl der Krümmung für einen Knoten auf der Oberfläche eines diskreten Zylinders.

3.2 Optimierung

Leider existieren grundlegende Probleme bei der Bestimmung der Linienorientierung durch diskrete Krümmungsanalyse. So ist zum einen nicht für jeden Knoten ein Wert definiert; auf ebenen Flächen oder Kugeloberflächen existieren keine eindeutig orientierten minimalen und maximalen Krümmungsrichtungen. Ein weiteres Problem ist die Anfälligkeit des Verfahrens gegenüber hochfrequenten Änderungen in der Oberfläche, z.B. durch Rauschen oder feine Details. Diese manifestieren sich in

nicht homogenen Richtungsfeldern. Versucht man davon Schraffuren abzuleiten, so werden sie sehr unnatürlich und sind, im Gegensatz zu Schraffuren von echten Künstlern, mehr verwirrend als Form betonend (Abb. 3.4).

Um gegen Probleme mit hohen Frequenzen vorzugehen, kann man diese filtern, indem man vorher z.B. das Modell glättet oder mit einem Mesh-Simplification-Algorithmus Details entfernt (siehe [Praun, 2001]). Dadurch wird aber nur einem Teil der Probleme entgegengewirkt. Zum einen bleiben Stellen, an denen keine Krümmungsrichtungen definiert sind, und das Problem wird unter Umständen sogar verschärft, da Glättungsfunktionen oft dazu neigen den Anteil von ebenen oder runden Bereichen in einem Modell zu erhöhen und so noch weniger Knoten eine definierte minimale und maximale Krümmungsrichtung zugewiesen ist.

Daher wurde in dieser Arbeit der Fokus mehr auf Verfahren gelegt, die versuchen, nach der Berechnung eines Feldes die Richtungen zu homogenisieren und nicht definierte Bereiche zu ergänzen. Dabei ist zunächst einmal ein Kriterium wichtig, mit dessen Hilfe bewertet werden kann, welche Regionen nicht homogen sind. Dies wird über einen Energieterm gelöst, der angibt, wie stark die Unterschiede zwischen den einzelnen Richtungsvektoren sind. Je höher der Wert, desto höher die Inhomogenität. Zur Berechnung wurden zwei Varianten getestet, die im Folgenden erläutert werden.

3.3 Nachbarschaftliche Spannungen

Für einen Knoten betrachtet man die Spannungen zu den angrenzenden Knoten. Dabei wird jeweils für beide Knoten einer Kante das Problem in die Ebene verlagert, indem die Kante in die dortige Tangetialebene projiziert wird und das lokale Koordinatensystem so transformiert wird, dass die Projektion entlang einer der Koordinatenachsen verläuft. Mit Hilfe dieser Transformationen lassen sich an beiden Knoten die Richtungsvektoren in ein einheitliches Koordinatensystem bringen und Differenzen bilden. Dabei ist darauf zu achten, dass die Spannung am grössten sein soll, wenn die beiden Richtungen orthogonal zueinander stehen und bei linearer Abhängigkeit der Richtungsvektoren minimal werden soll. Um dies zu modellieren wird der Winkel zwischen den beiden transformierten Richtungsvektoren in folgende Funktion eingesetzt:

$$1 - \cos(2\theta)$$ *Formel 3.1*

Dies hat gegenüber der direkten Arbeit mit Winkeln den Vorteil, dass sich eine stetige Ableitung bilden lässt, was später noch wichtig sein wird.

Um die komplette Spannung an einem Knoten zu berechnen, führt man obigen Schritt für alle angrenzenden Kanten durch und summiert die Ergebnisse.

3.4 Umgebungs Spannung

Das Verfahren entspricht in weitesten Teilen dem vorhergehenden, jedoch werden nicht nur die angrenzenden Kanten betrachtet, sondern alle Kanten, die in einer Umgebung mit wohldefiniertem Ausmass um den Knoten herum existieren. Die Größe der Umgebung wird mit einer Kugel beschrieben, wobei der Radius eine vom Benutzer einstellbare Größe ist. Um alle beteiligten Kanten zu finden, werden alle Flächen, die sich um den Startknoten herum befinden, vermerkt. Solange an diese Menge eine Nachbarfläche angrenzt, die zumindest teilweise innerhalb der Kugel liegt, wird die Menge um diese Fläche erweitert. Dies wird solange fortgeführt, bis sich die Menge nicht mehr erweitern lässt. Dann werden alle Kanten betrachtet , die Bestandteil der in der Menge enthaltenen Flächen sind. Für jede Kante wird wie zuvor die Spannung berechnet, diese jedoch noch gewichtet, in Abhängigkeit davon wie viel der Kannte sich im Inneren der Kugel befindet (Abb. 3.3). Durch Aufsummieren der gewichteten Spannungen erhält man den Energieterm für die Umgebung eines Knotens.

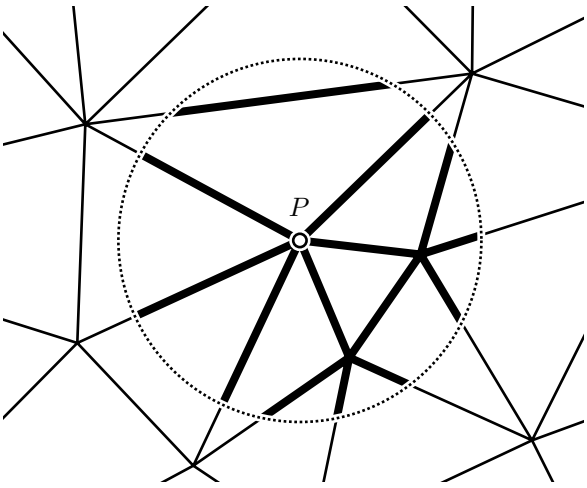


Abbildung 3.3:
Alle Kanten die sich in der Umgebung um Punkt P herum befinden. Für die Wichtung der Kanten wird betrachtet, wieviel einer Kante im Inneren der Umgebung liegt, im Verhältnis zur gesamten Länge einer Kante.

3.5 Minimierung

Nachdem nun ein Homogenitäts-Kriterium vorliegt, wählt man mittels eines vom Benutzer gewählten Schwellwerts alle Knoten aus die diesem Kriterium nicht genügen und versucht, dort die Homogenität zu erhöhen. Dieses Ziel wird erreicht, indem der Energieterm minimiert wird, denn dadurch wird auch gleichzeitig die Inhomogenität des Feldes vermindert, was ja das ursprüngliche Ziel war. Um das zu erreichen, gibt es mehrere Möglichkeiten. Am einfachsten ist es, jeden Knoten für sich zu betrachten und abhängig von der lokalen Nachbarschaft die Richtungsvektoren so zu verändern, dass der Energieterm minimiert wird. Ein einfaches Verfahren ist z.B. die lokale Relaxierung des Richtungsfeldes. Dazu wird für jeden Knoten ermittelt, wie weit der Richtungsvektor von denen in seiner direkten Nachbarschaft abweicht, davon wird der Durchschnitt gebildet und der Richtungsvektor wird um diesen Betrag zurückgedreht, wodurch die Spannung zu seinen Nachbarn minimiert wird. Durch mehrfaches Anwenden dieser Technik lässt sich das Richtungsfeld bis zu einem bestimmten Punkt zunehmend verbessern. Es entstehen dabei jedoch oft Situationen in denen das Verfahren an seine Grenzen stösst und Spannungen in Richtungsfelder nicht mehr weiter aufgelöst werden können, weil Zonen in einem stabilen aber nicht minimalen Zustand verharren, der nur aufgelöst werden könnte, indem sich mehrere Richtungsvektoren gleichzeitig in eine Richtung drehen, was jedoch durch die lokale Sicht des Verfahrens nicht möglich ist (Abb. 3.5).

Um solche Probleme zu vermeiden, macht es Sinn ein globales Optimierungsverfahren ein-

zusetzen (vergleiche [Hertzmann & Zorin, 2000]). Ein gängiger und frei verfügbarer Algorithmus zur globalen Optimierung von Systemen mit einer sehr hohen Anzahl von Variablen ist das L-BFGS-B Verfahren (siehe [Zhu & Byrd, 1997]). Dieses lässt sich als fertiges Paket beziehen, muss jedoch, da es sich um FORTRAN-Code handelt, für eine weitere Verwendung in einem C/C++ Projekt erst portiert werden. Mit Hilfe von [F2C] lässt sich dies jedoch mit relativ geringem Aufwand bewerkstelligen. Für die Optimierung selbst sind neben einem globalen Energieterm noch die zu optimierenden Variablen nötig und, da es sich bei L-BFGS-B um ein Gradientenabstiegs-Verfahren handelt, auch deren Ableitungen. Optimiert werden die Winkel um die die einzelnen Richtungsvektoren gedreht werden sollen, weshalb eben diese Winkeldeltas als zu optimierende Variablen verwendet werden. Um dazu sowohl einen globalen Energieterm als auch den Gradienten-Vektor der Variablen angeben zu können, wird wieder das Homogenitätskriterium verwendet. Aus Gründen der Berechnungskomplexität jedoch nur die einfachere Variante, sprich das Kriterium für nachbarschaftliche Spannungen. Darüber lässt sich nun unter Einbeziehung der Winkeldeltas bestimmen, wie gross die Spannung an den einzelnen Knoten ist und die Summe aller Spannungen ergibt den globalen Energieterm. Die jeweilige Komponente des Gradienten-Vektors lässt sich über die Ableitung des zuvor angegebenen Kosinusterms (siehe Formel 3.1) bestimmen. Die Optimierungsfunktion versucht dann eine Konstellation der Winkeldeltas zu finden in der der Energieterm minimal wird. Danach ist es notwendig, die einzelnen Richtungsvektoren um eben genau die von der Optimierungsfunktion bestimmten Winkeldeltas zu drehen. Danach ist das Feld sehr homogen (Abb. 3.6).

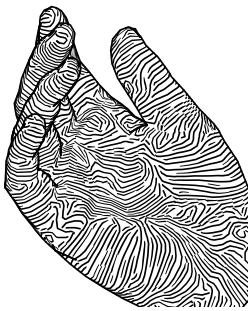


Abbildung 3.4: Richtungsfeld gänzlich ohne Optimierung.

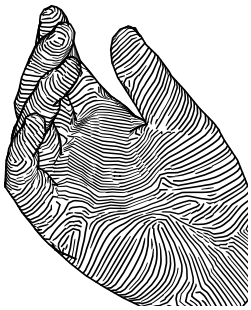


Abbildung 3.5: Richtungsfeld mit lokaler Relaxierung.

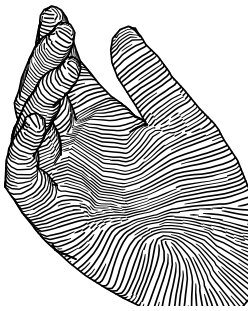


Abbildung 3.6: Richtungsfeld mit globaler Optimierung.

4. Stromlinien

Nachdem ein möglichst homogenes Richtungsfeld hergestellt wurde, müssen anhand der gegebenen Richtungsinformation Linien erzeugt werden, die auf der Modell-Oberfläche dem Richtungsfeld folgen. Da dieser Prozess relativ aufwändig ist und es möglich sein soll, interaktiv das Modell zu begutachten, wurden die Linien nicht im Bildraum, sondern im Objektraum plaziert. Dadurch ergeben sich zwar Artefakte bei der Projektion, die gesondert behandelt werden müssen, aber es hat den Vorteil, dass die Linienrepräsentation vollständig blickrichtungsinvariant ist und nur nach einer Veränderung des Modells neu berechnet werden muss.

Bei der Generierung der Linien ist darauf zu achten, dass die komplette Objektoberfläche abgedeckt werden muss, damit keine unbedeckten Flächen entstehen und die Linien dabei möglichst gleichmässige Abstände zueinander aufweisen sollten. Da es nicht praktikabel ist, überall exakt gleiche Abstände der Linien zu fordern, werden nur minimale und maximale Abstände definiert.

Als Grundlage für die Linienplatzierung dient ein Algorithmus zur Visualisierung von 2D Vektorfeldern (siehe [Jobard & Lefer, 1997]), der jedoch erst an die dritte Dimension adaptiert werden musste.

Um ein Feld zu visualisieren, werden von einem Startpunkt aus Stromlinien integriert und dabei neue mögliche Startpunkte für folgende Stromlinien in einer Queue abgelegt. Eine Stromlinie wächst dabei zugleich an beiden Enden und das solange, bis in beiden Richtungen ein Abbruchkriterium ein weiteres Wachstum verhindert. Um sicherzustellen, dass wirklich die komplette Oberfläche des Modells mit Linien bedeckt werden kann, wird immer, falls kein neuer möglicher Startpunkt mehr in der Queue vorhanden ist, ein Dreiecksschwerpunkt als möglicher neuer Startpunkt verwendet. Der Algorithmus terminiert, sobald die Queue der möglichen Startpunkte leer ist und alle Dreiecksschwerpunkte getestet wurden. Um neue mögliche Startpunkte zu finden, werden, während eine Linie integriert wird in festen Abständen orthogonal zur Laufrichtung zwei Suchen gestartet. Diese versuchen, sich von der Linie so weit zu entfernen, wie der gewünschte maximale Linienabstand beträgt. Ist dort noch keine Linie in einem Umkreis des minimalen Linienabstands, so wird dieser Punkt als möglicher neuer Startpunkt vermerkt.

4.1 Integration

Für die Integration selbst wird als ODE-Solver Runge-Kutta vierter Ordnung verwendet (siehe [Press, 1992]). Es wurde vorher anstelle dessen ein einfacher Euler-Integrator eingesetzt, doch waren die Ergebnisse nicht zufrieden stellend. Durch einen besseren Integrator kann die Schrittweite grösser gehalten werden, ohne der Stabilität des Integrationsverfahrens zu schaden. Letztlich macht sich dies nicht nur in einer bedeutend verkürzten Wartezeit für die Integration bemerkbar, sondern senkt gleichzeitig die Anzahl der resultierenden Liniensegmente. Dadurch erhöht sich letztlich auch die Framerate bei der interaktiven Darstellung merkbar.

Da die Vektoren im Richtungsfeld nur den einzelnen Knoten des Modells zugeordnet sind, ist es erforderlich, für dazwischenliegende Punkte einen Richtungsvektor zu interpolieren. Dazu werden die baryzentrischen Koordinaten eines Punktes innerhalb des umschließenden Dreiecks bestimmt, die dann verwendet werden, um per sphärisch-linearer Interpolation aus den den Knoten des Dreiecks zugeordneten Richtungsvektoren eine für diesen Punkt entsprechende Richtung zu bestimmen. Da es für die Stabilität und Güte der Interpolation wichtig ist, dass die Vektoren möglichst in eine ähnliche Richtung zeigen, werden die Vektoren vor der Interpolation unter Umständen um 180° gedreht. Als Referenz wird dabei die letzte Orientierung verwendet. Per Skalarprodukt wird getestet, ob eine der Richtungen davon mehr als 90° abweicht und diese gegebenenfalls negiert. Hat man noch keine letzte Orientierung, was nur für Startpunkte gilt, so wird als Referenz der Richtungsvektor der dem Punkt am nächsten liegenden Dreiecksknoten verwendet.

Neben der Bestimmung eines Richtungsvektors an einem Punkt ist es für die Integration jedoch auch erforderlich, einen Richtungsvektor zu ermitteln, der von einem gegebenen Punkt aus in einer bestimmten Laufrichtung und Distanz liegt. Dazu wird von dem Punkt aus in der Laufrichtung solange über die Objektoberfläche gelaufen, bis die gewünschte Distanz überwunden wurde. Wird dabei eine Dreieckskante überschritten, so wird die Laufrichtung angepasst, indem sie entlang der Kante gedreht wird, so dass sie in der Ebene des neuen Dreiecks liegt. Wurde die gewünschte Distanz bewältigt, so wird dort, wie gehabt, der Richtungsvektor durch Interpolation bestimmt und danach die bei der Überschreitung der Kanten erfolgten Transformationen umgekehrt ausgeführt, um den bestimmten Richtungsvektor wieder in das ursprüngliche Koordinatensystem zu übertragen.

Damit hat man die wichtigsten Teilstücke beisammen, die nötig sind um den ODE-Solver zu implementieren. Aus einer Gewichtung mehrerer Richtungsvektoren wird, abhängig von der Schrittgrösse des Integrators, ein neuer Richtungsvektor bestimmt. Dieser wird verwendet, um sich einen Schritt weit über die Oberfläche des Modells zu bewegen und einen neuen Punkt in den Linienzug einzufügen. Wird dabei eine Dreiecksgrenze überschritten, so wird dort auch ein neuer Punkt eingefügt, damit die Linie die Oberfläche des Modells nicht verlässt.

4.2 Abbruchkriterien

Ein Linienzug wird dabei stets weitergeführt, bis ein Abbruchkriterium erfüllt wird. Davon gibt es in dieser Implementierung mehrere, die vom Benutzer ein- und ausgeschaltet werden können, um unterschiedliche Effekte zu erzielen. Es kann auch in seltenen Fällen passieren, dass Linien abgebrochen werden müssen, weil die Integration nicht weitergeführt werden kann. Dies tritt auf, da die Algorithmen auf Fliesskomma-Arithmetik beruhen, wodurch in ungünstigen Situation Floatingpoint-Underflows auftreten können. In diesem Fall kann ein Linienzug dort nicht sinnvoll weiter integriert werden, weshalb diese Situation ebenfalls als Abbruchkriterium aufgefasst wird.

Während der Integration einer Linie wird festgehalten, wie stark die Linie verbogen wird, also durch den Einfluss des Richtungsfeldes von einem geradlinigen Pfad abweicht. Auch hier lässt sich ein Grenzwert angeben, bei dessen Überschreiten die Linie nicht weiterverfolgt wird. Das ist notwendig um zu starke Knicke der Linie zu verhindern, was besonders bei weniger homogenen Richtungsfeldern oder in der Nähe von Singularitäten von Bedeutung ist.

Um scharfe Knicke in der Oberfläche selbst besser herausarbeiten zu können, wird die Grenze zwischen zwei Dreiecken nur dann überschritten, wenn der Winkel zwischen den Normalen der Flächen kleiner ist als ein vom Benutzer definierter Schwellenwert. Dadurch entsteht entlang von stärkeren Kanten eine Diskontinuität im Fluss der Linien. Dies findet sich auch bei handgemachten Illustrationen (vergleiche Abb. 1.1 und 1.2).

Zu lange Linienzüge haben oft die Eigenheit, dass sie sich überdeutlich als Struktur herausbilden. Um das zu verhindern, kann eine maximale Länge für einen Linienzug spezifiziert werden. Würde durch einen Integrations-Schritt diese Länge überschritten werden, so wird nur noch ein Teil des Schrittes ausgeführt, sodass die Länge genau erreicht wird.

Das wichtigste Kriterium ist jedoch die Nähe zu bereits bestehenden Liniensegmenten. Da es nicht erwünscht ist, dass sich Linien überschneiden, wird eine Linie abgebrochen, sobald der minimale Linienabstand unterschritten wird.

Zusätzlich wird, nachdem die Integration eines Linienzuges abgebrochen wurde, überprüft, ob eine minimale Linienlänge erreicht wurde. Ist dies nicht der Fall, so wird der komplette Linienzug verworfen. Dadurch ist es möglich, kleine Fragmente auszuschliessen und einen insgesamt ruhigeres Linienbild zu erhalten.

4.3 Liniennähe

Wie bereits erwähnt ist es wichtig, die Distanz zu anderen auf der Oberfläche liegenden Linienzügen messen zu können, um Kollisionen und Überschneidungen zu vermeiden sowie um Startpunkte zu verwerfen, die im Inneren eines schon bestehenden Linienzugs liegen. Hierfür existieren mehrere Varianten:

In der ersten Variante wurde auf 3D-Grid-Hashing zurückgegriffen. Ein Segment wird dabei in eine Folge von Punkten zerlegt, die in einem 3D-Raster gespeichert werden. Da der Platzbedarf für eine 3D-Matrix unter Umständen sehr gross werden kann und trotzdem die meisten Zellen nicht belegt werden, bietet es sich an, eine "Sparse-Matrix" zu verwenden. Diese wurde implementiert, indem aus den Zellenkoordinaten ein Hashwert erstellt wird, mit dessen Hilfe auf eine Liste der dort enthaltenen Zellen zugegriffen werden kann. Die Größe der einzelnen Rasterzellen richtet sich dabei nach dem minimalen Linienabstand. Um die Umgebung eines Test-Punktes zu prüfen, wird ermittelt, in welcher Zelle er liegt und es werden alle Punkte aus der Zelle und den acht Nachbarzellen ermittelt und der minimale Abstand zwischen den Punkten und dem Test-Punkt bestimmt. Das Verfahren ist zwar schnell und die durch

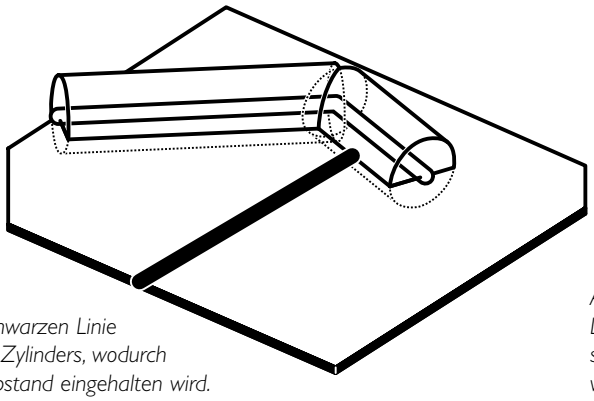


Abbildung 4.1:
Die Integration der schwarzen Linie endet am Körper des Zylinders, wodurch ein minimaler Linienabstand eingehalten wird.

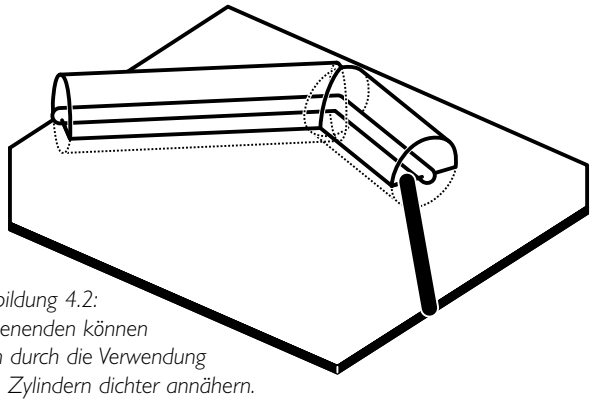


Abbildung 4.2:
Linienenden können sich durch die Verwendung von Zylindern dichter annähern.

die Quantisierung der Segmente entstehenden Ungenauigkeiten sind in der Praxis zu vernachlässigen, aber leider ist auch der Wirkungsbereich nicht durch die Geometrie des Objekts eingeschränkt. Das macht sich besonders an sehr dünnen Strukturen bemerkbar, bei denen die Geometrie dünner ist als der minimale Linienabstand, wie es zum Beispiel bei Blütenblättern der Fall ist. Hier wirkt sich die Existenz eines Segments auch auf die Rückseite aus, wodurch eine Linie auf der Oberseite eines Blütenblattes Linien auf der Unterseite unterbindet.

Daher wurde letztlich ein anderes Verfahren entwickelt. Linienzüge werden in eine Folge von Segmenten zerlegt, die wiederum mit Zylindern umschrieben werden, deren Radius dem minimalen Linienabstand entspricht (Abb. 4.1). Diese werden in den Flächen gespeichert, die sie schneiden und von der Fläche in der das Segment liegt, erreichbar sind. Dazu werden von der Ursprungsfläche ausgehenden rekursiv die Kanten, die ja Transitionen zu den Nachbarn darstellen, untersucht und ein Nachbar besucht, falls er noch nicht besucht wurde, die Kante den Zylinder schneidet und der Knick zwischen den Flächen nicht zu gross ist. Dadurch wirkt sich ein Zylinder nur auf seine lokale Umgebung aus. Solange die Dreiecke nicht zu gross werden, ist der Zugriff auch schnell, denn um die Frage der Nähe für einen Punkt zu klären, muss nur die Liste der der Fläche zugeordneten Zylinder befragt werden und solange die Größe der Dreiecke kleiner ist als der minimale Linienabstand, so befindet sich dort auch höchstens ein Zylinder, der getestet werden muss. Ein weiterer Vorteil des Verfahrens ist, dass sich so Linien entlang ihrer Enden sehr viel dichter kommen können (Abb. 4.2) als bei Verfahren, die den tatsächlichen Abstand zu einem Segment bestimmen. Ansonsten entsteht zwischen den Enden zweier nicht zusammenhängender Linienzüge immer mindestens eine Lücke die dem minimalen Linienabstand entspricht (Abb. 4.3). Zu beachten ist jedoch, dass an den Knickstellen zwischen zwei Zylindern Kerben auftreten. Diese werden mit Zylindern überdeckt (Abb. 4.4).

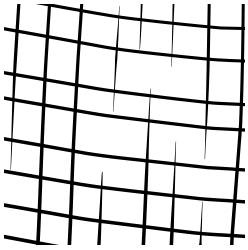


Abbildung 4.3:
Detailauschnitt eines Feldes von [Herzmann & Zorin, 2000] zeigt sehr deutliche Lücken zwischen den Linienenden.

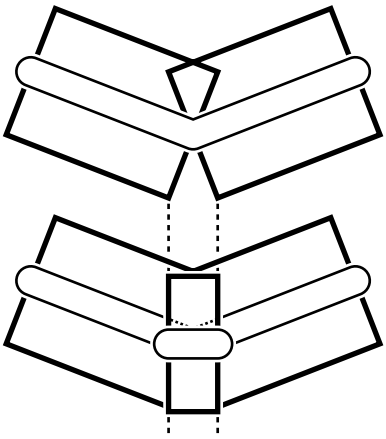


Abbildung 4.4:
Die Zylinder an einem aus zwei Segmenten bestehenden Linienzug müssen am Knick mit einem Zylinder überdeckt werden um eine Lücke zu vermeiden.

4.4 Implementierungs-Details

Da für die bisher beschriebenen Algorithmen viel auf der Oberfläche des Modells gearbeitet wird und Information über die Nachbarschaft einer Fläche äußerst wichtig ist, ist es sinnvoll, dabei eine für diese Operationen förderliche Modell-Beschreibung zu wählen. Da eine 2-Mannigfaltigkeit für diese Arbeit ausreicht, wurde die Half-Edge Datenstruktur (siehe [Mäntylä, 1988]) verwendet. Dabei wird eine Kante des Modells über je zwei gerichtete Kanten dargestellt, die jeweils zu einer der an die Kante angrenzenden Flächen gehören und Information über die Nachbarschaft halten (siehe Abb. 4.5). Trotz der sehr einfachen Datenstruktur lassen sich die wichtigsten Fragen über angrenzende Flächen in adäquater Laufzeit beantworten:

```
struct HalfEdge {
   Vertex*   vert;
   HalfEdge* next;
   HalfEdge* loop;
   HalfEdge* pair;
   Triangle* tri;
};
```

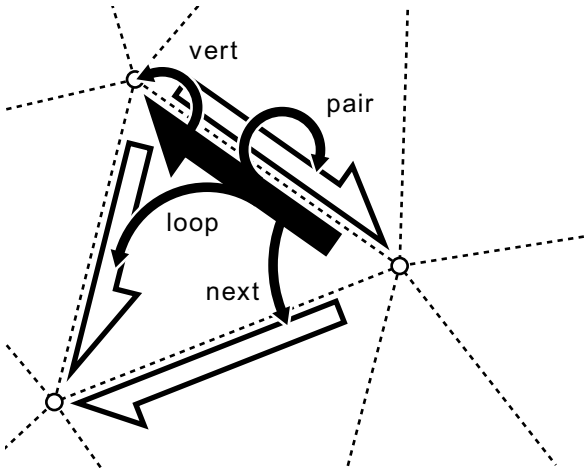


Abbildung 4.5: Visualisierung einer Half-Edge mit ihren direkt zugänglichen Nachbarn.

Eine Operation, die auch sehr häufig ausgeführt wird, ist das Umklappen eines Vektors entlang einer Kante, um ihn von einer Fläche auf eine benachbarte zu übertragen. Für eine naive Implementierung dreht man dabei den Vektor um die Kante und zwar entsprechend des Winkels zwischen den beiden Normalenvektoren der angrenzenden Flächen. Das ist jedoch relativ aufwändig und es existiert eine viel einfachere geometrische Herleitung für diese Operation. Projiziert man das Problem in die Ebene, so werden aus den Ebenen zwei Graden, der zu drehende Vektor liegt auf einer dieser Graden und der gewünschte auf der anderen. Da die Rotation längentreu ist sind der ursprüngliche und der gesuchte Vektor gleich lang und bilden zusammen ein gleichschenkliges Dreieck. Zerlegt man es an der Winkelhalbierenden, so entstehen zwei kongruente rechtwinklige Dreiecke, deren Hypotenusen dem gegebenen und gesuchten Vektor entsprechen. Die gemeinsame Strecke entlang der Winkelhalbierenden entspricht jedoch der Projektion einer der Vektoren auf die Winkelhalbierende und aus dem gegebenen Vektor und der Projektion lässt sich leicht der gesuchte Vektor bestimmen. Das ist nun direkt auf den ursprünglichen Fall übertragbar (Abb. 4.6):

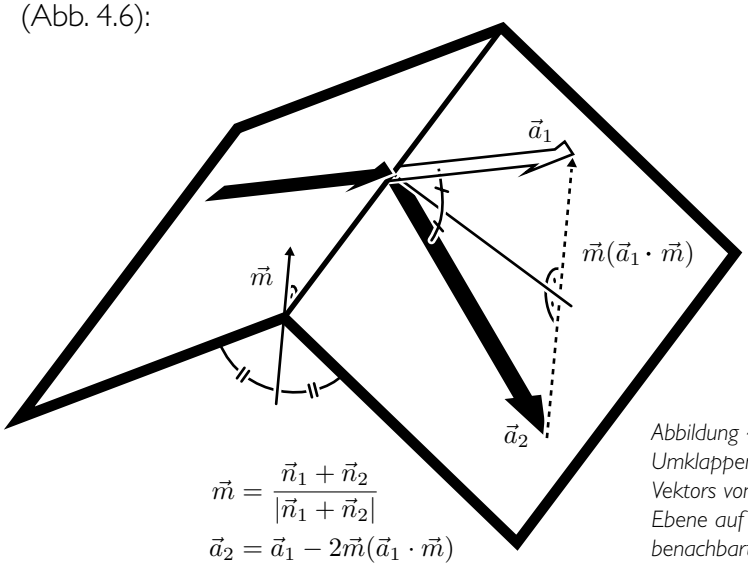


Abbildung 4.6: Umklappen eines Vektors von einer Ebene auf eine benachbarte.

5. Visualisierung

Nach der Vorverarbeitung muss aus den zuvor generierten Daten letztlich ein sichtbares Bild generiert werden. Die dazu benötigten Schritte werden im Folgenden erläutert.

5.1 Verdeckte Linien

Da nur Linien und keine verdeckenden Flächen gezeichnet werden ist es wichtig, dass die Bereiche von Linienzügen entfernt werden die nicht gesehen werden können, weil sie von davor liegenden Teilen des Modells verdeckt werden. Dafür existiert in OpenNPAR bereits ein Modul, das jedoch ursprünglich für die Verarbeitung von Silhouetten ausgelegt wurde (siehe [Isenberg, 2002]). Linienfragmente werden hier beseitigt, indem als Grundlage eine z-Buffer-Darstellung des Modells verwendet wird. Die Liniensegmente werden rasterisiert und falls Bereiche z-Koordinaten aufweisen, die weiter entfernt sind als die Referenzwerte des Modells im z-Buffer, so werden sie herausgeschnitten. Da die Linienzüge auf der Oberfläche des Modells liegen, funktioniert das auch mit diesen. Es mussten nur einige der zusätzlichen Daten die von den Linienzügen benutzt werden, auch diesem Modul vermittelt werden, damit sie beim Beseitigen eines Linienfragments mit herausgeschnitten werden können und so konsistent bleiben.

Zusätzlich wurden einige dieser Daten benutzt, um vor dem eigentlichen Zerschneiden Backface-Culling durchzuführen, wodurch nur noch in etwa die Hälfte der ursprünglichen Segmente verarbeitet werden müssen. Der daraus resultierende Geschwindigkeits Gewinn ist deutlich merkbar. Ein zusätzlicher Vorteil des Backface-Cullings ist, dass Linien, die Silhouetten-Kanten kreuzen, gewöhnlich durch eine kleine Ungenauigkeit der z-Buffer basierten Methode nicht exakt geschnitten werden, sondern oft ein kleines Teilstück übrig bleibt, das eigentlich über die nicht sichtbaren abgewandten Flächen verläuft und entsprechend ein kleiner Haken übrig bleibt (Abb. 5.1). Dieses Problem wird ebenfalls dadurch vermieden, dass von vornherein alle Segmente entfernt werden, die auf vom Betrachter abgewandten Flächen liegen.

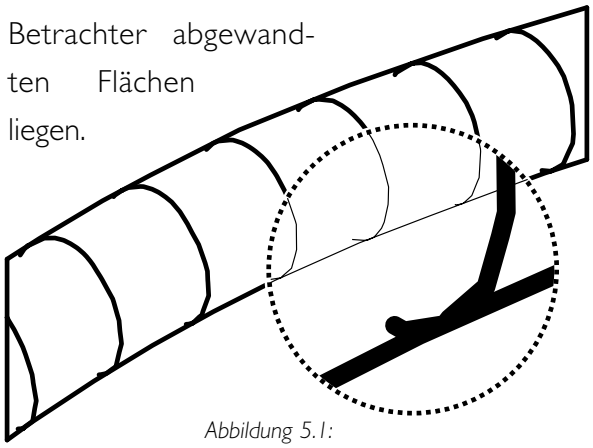


Abbildung 5.1:
Durch Toleranzen des z-Buffer basierten Verfahrens wird nicht exakt am Übergang der Sichtbarkeit geschnitten, sondern es entstehen entlang der Objektkontur kleine Häkchen.

5.2 Schattierung

Bevor ein Objekt schattiert wird, ist es nötig, den wahrgenommenen Grauwert des nicht schattierten Modells zu homogenisieren. Hat man dies erreicht, so ist für das eigentlich Schattieren nur eine Modulation der Linienstärken notwendig, da sich diese dann direkt proportional auf den wahrgenommenen Grauwert auswirkt. Somit wird es möglich, das Schattierungsmodell auf eine rein lokale Betrachtung eines Knotens zu vereinfachen.

Ein Faktor, der Inhomogenitäten verursacht, ist die Projektion der Linien vom Objekt- in den Bildraum, denn je nach Neigung einer Fläche und dem Verlauf von parallelen Linien auf selbiger ist der finale Abstand in der Bildebene sehr viel geringer als der ursprünglich spezifizierte Linienabstand auf der Oberfläche. Wird die Linienstärke selbst nicht durch die Projektion beeinflusst, so ändern sich nur die Abstände der Linien, wodurch sich je nach Orientierung der Fläche ein unterschiedliches Verhältnis aus Untergrund und Linie ergibt, was sich letztlich im wahrgenommenen Grauwert widerspiegelt (Abb. 5.2).

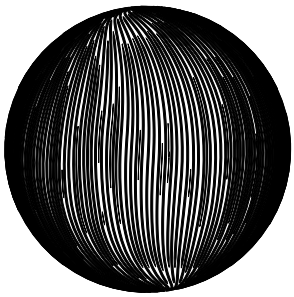
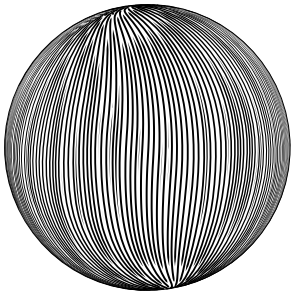


Abbildung 5.2:
Unterschiede im wahrgenommenen Grauwerte, die durch das Plazieren der Linienzüge im Objektraum auftreten.

Um dem entgegenzuwirken wird mittels einer Heuristik versucht abzuschätzen, wie weit Linien nach der Projektion voneinander entfernt sein werden. Dazu wird ein Einheitsvektor konstruiert, der orthogonal zur mittleren Laufrichtung verläuft und in der Tangentialebene eines Knotens liegt. Die Länge der Projektion dieses Vektors auf die Bildebene entspricht einer Abschätzung der tatsächlichen Distanzen der Linien. Wird die Linienbreite mit diesem Wert moduliert, so verbessert sich die Homogenität des wahrgenommenen Grauwerts deutlich (Abb. 5.3). Für eine weitere Verbesserung der Qualität müsste jedoch zusätzlich Information gehalten werden. Eine Möglichkeit wäre zum Beispiel, anstelle der Abschätzung benachbarter Linien direkt zwei Vektoren zu halten, die von einem Knoten die Distanz zu den tatsächlichen Nachbarlinien markieren.

Abbildung 5.3:
Wie Abb. 5.2, jedoch mit angewendetem Korrekturfaktor.



Ein anderer Faktor für Inhomogenitäten entsteht durch die nicht exakt gleichen Abstände der Linienzüge. Diese können zwischen dem minimalen und maximalen Linienabstand schwanken. So ist es nötig, diesbezüglich einen visuellen Ausgleich zu schaffen. Dieser kann in einem Vorverarbeitungsschritt für jeden Knoten eines Linienzuges berechnet werden, nachdem alle Linien plaziert wurden. Der Faktor verjüngt Linien, die sich näher kommen, indem ein Quotient aus der Distanz zum dichtesten Segment in der Nachbarschaft und der Differenz des maximalen und minimalen Linienabstands gebildet wird. Entsprechend des zylinderbasierten Nähe-Kriteriums wird jedoch auch hier die Distanz nicht omnidirektional (Abb. 5.4), sondern parallel zur Laufrichtung des Linienzugs berechnet (Abb. 5.5), indem für einen Knoten eine gemittelte Laufrichtung bestimmt wird, die in Form eines Normalenvektors die Grundlage bildet, um eine Ebene aufzuspannen. Nun werden nur die Durchstosspunkte benachbarter Liniensegmente betrachtet und zu diesen der minimale Abstand ermittelt. Ein Vorteil dieser Methode ist, dass sich Liniensegmente mit ihren Enden sehr viel näher kommen können, ohne das die Linienstärke vorzeitig abnimmt und somit grosse sichtbare Lücken entstehen.

Abbildung 5.4
Verjüngung auf Grundlage eines omnidirektionalen Nähe-Kriteriums.

Abbildung 5.5
Verjüngung auf Grundlage eines Laufrichtungs abhängigen Nähe-Kriteriums.

Das Schattierungsmodell selbst besteht aus zwei Ausdrücken, über die zum einen die Linienbreite moduliert werden kann, aber auch Einfluss auf die Liniendichte möglich wird, wodurch eine Linie von einem festen Zustand in Punkte aufgelöst werden kann (Abb. 5.6). Diese Ausdrücke werden für jeden Knoten eines Linienzuges ausgewertet und bilden so für jeden Knoten zwei zusätzliche Parameter, die zu einem späteren Zeitpunkt vom Liniendarstellungs-Modul verwendet werden, um das tatsächliche Aussehen einer Linie zu bestimmen.

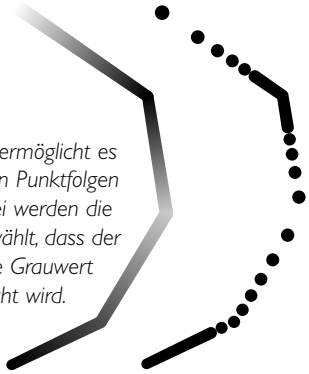


Abbildung 5.6:
Die Liniendichte ermöglicht es einen Linienzug in Punktfolgen aufzulösen. Dabei werden die Abstände so gewählt, dass der wahrgenommene Grauwert annähernd erreicht wird.

5.3 Virtuelle Maschine

Für die Auswertung der Schattierungs-Ausdrücke wird eine spezialisierte virtuelle Maschine eingesetzt. Das hat den Vorteil, dass die Ausdrücke nicht zur Kompilierungszeit fest in ein Programm aufgenommen werden müssen, sondern sich auch noch zur Laufzeit frei verändern lassen. Dadurch wird es erheblich einfacher, spielerisch ein Gefühl für die Ausdrücke zu entwickeln und einen reichhaltigen Fundus neuer Effekte aufzubauen.

Ein Ausdruck besteht dabei aus einer festgelegten Menge von Bezeichnern sowie Operatoren. Diese werden in einem Übersetzungs-Schritt in einen Byte-Code übertragen, wobei gleichzeitig eine Infix-Darstellung in Umgekehrte-Polnische-Notation konvertiert wird. Das befreit den Ausdruck sowohl von Klammern als auch von der Präzedens der Operatoren.

Soll solch ein in Byte-Code übersetzter Ausdruck für einen Knoten ausgewertet werden, so wird zuerst für jeden im Ausdruck auftauchenden Bezeichner der entsprechende Wert berechnet. Taucht zum Beispiel der Bezeichner "`LIGHT`" im Ausdruck auf, so wird die diffuse Beleuchtungs-Komponente für diesen Knoten ermittelt. Der Byte-Code wird dann mit Hilfe eines Stack-basierten Interpreters abgearbeitet, wobei beim Auftreten eines Bezeichners dieser durch den zuvor berechnet Wert ersetzt wird.

Neben den üblichen Rechenoperatoren +, -, *, / und ^ stehen folgenden Bezeichner zu verfügung:

`LIGHT`
Der Wert des diffusen Beleuchtungsterms. Die dafür benötigte Normale wird per Phong-Interpolation bestimmt, wobei die Lichtquelle fest von links-oben beleuchtet.

`RIM`
Entspricht `LIGHT`, nur wird die Lichtquelle hier zwischen Objekt und Betrachter plaziert, um Rim-Shadows zu simulieren.

`DEPTH`
Der Abstand zur Bildeben für Depth-Cueing.

`CURVATURE`
Die mittlere Krümmung für Curvature-Lighting.

6. Vektor-Linien

In dieser Arbeit spielt die Linie eine zentrale Rolle, weshalb es überaus wichtig ist, sich umfassend mit dieser Thematik zu beschäftigen. Andere Ansätze dazu finden sich bei [Vehmeier, 2002] und für die Implementierung ist auch insbesondere der Quelltext von [LibArt] interessant.

6.1 OpenGL

Bei der Verwendung von OpenGL ist eine naheliegende Variante `glLineWidth` zu benutzen und dann wie gewohnt Linien zu zeichnen. Der Nachteil ist jedoch, dass die Liniendarstellung von OpenGL mehr Wert auf Geschwindigkeit legt als auf die Darstellungsqualität, denn Linien bestehen in OpenGL aus gescherten Rechtecken. Dadurch ändert sich die Stärke der Linie je nach Verlauf. Für horizontale oder vertikale Linien ist diese noch richtig, aber je dichter sich der Winkel an die Diagonalen annähert, desto grösser wird der Fehler. An den Diagonalen sind es nur noch etwa 70% der eigentlich gewünschten Linienstärke (Abb. 6.1).

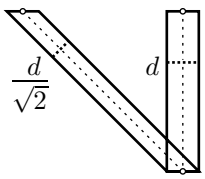


Abbildung 6.1: OpenGL-Linien ändern ihre Dicke je nach Laufrichtung.

Ein weiteres Problem sind die Übergänge der Liniensegmente. Falls zwei aufeinander folgende Liniensegmente in unterschiedlichen Quadranten liegen, kann es passieren, dass der Knick durch kleine quadratische Einkerbungen sehr unästhetisch wirkt (Abb. 6.2).

Abbildung 6.2: Artefakte an Knicken in OpenGL-Linien.

Selbes gilt für die Enden. Verlaufen die Linien entlang der Diagonalen, so sind die Enden eher spitz und entlang der Koordinaten-Achsen stumpf.

Diese Artefakte werden auch besonders sichtbar in Animationen, da sich hier ein Knick von einem Bild zum nächsten verändern kann oder Linien während einer Transformation ihre Linienstärke ändern.

6.2 Vierecke

Eine weitere Möglichkeit ist es, entlang der Linien zusammenhängende Vierecke aufzuspannen. Dies hat den Vorteil, dass es so möglich wird, die Linie mit Hardware-beschleunigten texturierten Polygonen darzustellen, wodurch sich viele Effekte auf die Linie anwenden lassen.

Für die einfachste Variante berechnet man an den einzelnen Knoten der Liniensegmente, die Winkelhalbierende und setzt darauf im Abstand der Linienstärke die einzelnen Knoten eines `GL_TRIANGLE_STRIP`. Leider hat man dadurch ähnliche Probleme wie mit dicken OpenGL-Linien, da sich auch hier die Linienstärke ändert: je spitzer der Winkel, desto stärker wird die Linie zusammengedrückt (Abb. 6.3). Dieser Effekt lässt sich jedoch leicht kompensieren, wenn man sich die Geometrie an der Spitze einer richtig durchgeführten Gehrung betrachtet: Die Winkelhalbierende bildet die Hypotenuse eines rechtwinkligen Dreiecks (Abb. 6.4). Wir wissen, dass die Ankathete kürzer ist als die Hypotenuse. Der Faktor entspricht dabei genau dem Kosinus

Abbildung 6.3: Obwohl der Querschnitt am Knick dem Liniendurchmesser entspricht, ist dieser nicht gleichmässig.

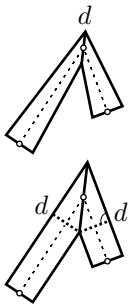


Abbildung 6.4: Rechtwinklige Dreiecke an einer korrekten Gehrung.

des Winkels zwischen Ankathete und Hypotenuse. Es reicht also, den Abstand durch diesen Wert zu teilen, damit die Linie überall mit gleicher Stärke verläuft.

6.3 Gehrungsgrenze

Leider ist auch dieses Verfahren nicht völlig problemfrei, denn je grösser der Winkel wird (sprich je spitzer die Ecke), desto kleiner wird der Kosinus, was letztlich in einem unendlichen langen Spitze enden würde. Daher muss man ab einer bestimmten Grenze (der *Gehrungsgrenze*) die Strategie ändern und die Spitze abschneiden. Dazu verbindet man die äußeren Eckpunkte der Rechtecke der einzelnen Liniensegmente. Man konstruiert also für die beiden an einem Knoten anliegenden Richtungsvektoren orthogonale Vektoren die halb so lang sind wie die Liniestärke sein soll, achtet darauf, dass sie auch nach aussen zeigen und verbindet sie (Abb. 6.5).

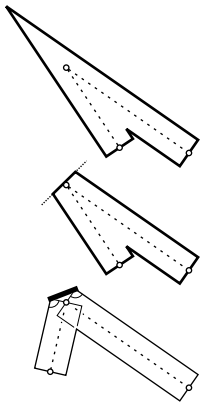


Abbildung 6.5: Wird die Gehrungsgrenze überschritten, so wird die Spitze abgeschnitten.

6.4 Runde Linien

Ein anderer Linientyp ist besonders in der klassischen technischen Illustration sehr beliebt: die runde Linie. Hier sind sowohl Anfangs- und Endpunkt als auch die Knicke rund.

Um dies zu erreichen, geht man ähnlich vor wie im letzten Abschnitt, nur verbindet man die Eckpunkte mit Kreisbögen und schließt die Linie an beiden Enden noch jeweils mit einem Halbkreis ab (Abb. 6.6). Für Linien mit fester Linienbreite ist die Konstruktion sehr einfach, jedoch ist es erstrebenswert anstelle einer festen Linienstärke an jedem Knoten des Linienzuges auch unterschiedliche Radien zu erlauben und so Variationen der Linienstärke zu erzeugen. Die Konstruktion der Konturlinie wird dadurch leider etwas komplizierter als im einfachen Fall, da nun die Konturlinien nicht mehr parallel zu den Segmenten verlaufen müssen. Um an den einzelnen Radien nahtlosse Übergänge zwischen den Konturlinien und Kreisbögen zu erreichen, müssen die Linien tangential zu den Kreisen verlaufen. Der Kosinus der Winkel entspricht dabei dem Verhältnis aus der Größendifferenz zweier Radien und dem Abstand ihrer Mittelpunkte (Abb. 6.7).

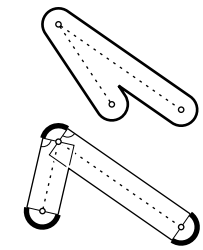


Abbildung 6.6: Kreisbögen an Knicken sowie Linienenden.

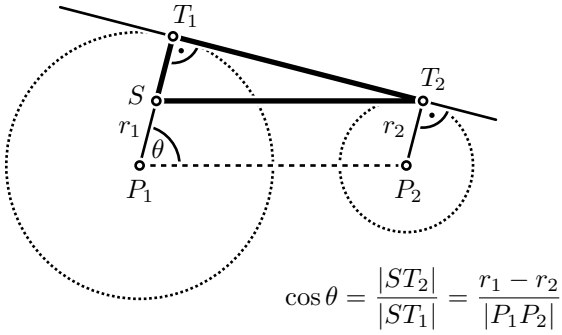


Abbildung 6.7: Berechnung des Winkels zum Berührpunkt einer äusseren Tangente zweier Kreise.

6.5 Implementierungs-Details

Zuerst eine Betrachtung des einfachen Falls mit konstanter Linienbreite. Um den Algorithmus möglichst einfach zu halten, entfernt man in einem Vorbereitungsschritt zum einen Segmente, deren Länge 0 ist, und sammelt zusätzlich alle Daten die später zum Zeichnen benötigt werden. Das sind insbesondere die Verbindungsvektoren sowie jeweils ein normalisierter, orthogonaler Vektor dazu.

Man bekommt es mit drei Hauptfällen zu tun, die sich jeweils noch in zwei Unterfälle auftei-

len. Es gibt Rechts- und Linksknicke sowie die Kollinearität. Diese lassen sich recht einfach über einen Knicktest unterscheiden. Für die Fälle Rechts- und Linksknick muss man jedoch prüfen, ob der Schnittpunkt der beiden inneren Aussenkanten überhaupt gültig ist, sprich innerhalb der Aussenkanten liegt. Für den Fall der Kolinearität gibt es die Möglichkeit, dass es an der Stelle überhaupt keinen Knick gibt, sondern es sich nur um eine Unterteilung eines graden Segments handelt. Dann kann darüber hinweggesprungen werden, es ist jedoch auch möglich, dass es sich um eine 180°-Wende handelt.

Der Zeichenvorgang selbst behandelt einen Knoten nach dem anderen, klassifiziert den Knick und zerlegt diesen entsprechend in einen Triangle-Strip, um so die gesamte Linie zu triangulieren. Da zum Teil mehrfach auf Knoten zurückgegriffen werden muss, sollten Vertex-Arrays benutzt werden, um Lücken in der Darstellung zu vermeiden. So hat der OpenGL-Treiber die Möglichkeit, während des Triangle-Setups degenerierte Dreiecke zu verwerfen. Diese werden nämlich benötigt, um trotz Triangle-Strips die Kurven an den Knicken mit einem Triangle-Fan darstellen zu können. Dies entspricht einer Situation, in der die Dreiecke an der Innenseite eines Knicks so stark zusammengedrückt werden, dass alle auf einen Punkt fallen (Abb. 6.8).

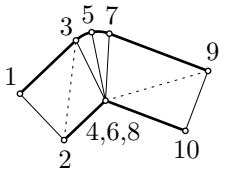


Abbildung 6.8: Ein Triangle-Strips mit degenerierten Dreiecken ermöglicht die Simulation von Triangle-Fans.

Falls es aber keinen Knoten gibt, an dem sie sich verdichten lassen (falls der innere Schnittpunkt nicht auf beiden Graden liegt, oder es sich um eine 180°-Wende handelt), so muss man an dieser Stelle mit der Triangulation absetzen und neu anfangen. Dies erfolgt wieder durch Einfügen von degenerierten Dreiecken (Abb. 6.9), so vermeidet man die Aufrufskosten zum Zeichnen eines zusätzlichen Vertex-Arrays.

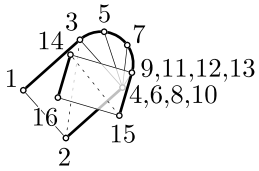


Abbildung 6.9: Unterbrechung in einem Triangle-Strip Mithilfe mehrerer degenerierter Dreiecke.

Für den komplizierteren Fall mit variabler Linienbreite gleicht der algorithmische Aufbau der vorhergehenden einfachen Version. Es entstehen jedoch durch die veränderten Rahmenbedingungen weitaus mehr mögliche Fälle (Abb. 6.10), die sich auch nicht mehr so einfach durch einen Knicktest unterscheiden lassen. Anstelle dessen wurde direkt mit den Winkeln der vier Tangenten-Berührpunkte gearbeitet. Für jeden einzelnen Winkel wird geprüft, ob er innerhalb des Winkelbereichs liegt, der von dem anderen Winkelpaar aufgespannt wird. Daraus ergeben sich vier Vergleiche, die in einem 4-Bit Ausdruck zusammengefasst werden, der wiederum als Index in eine Tabelle dient, in der der entsprechende Fall nachgeschlagen werden kann. Wie sich in der Praxis zeigt, ist diese Implementierung nicht nur sehr schnell, sondern auch überaus robust.

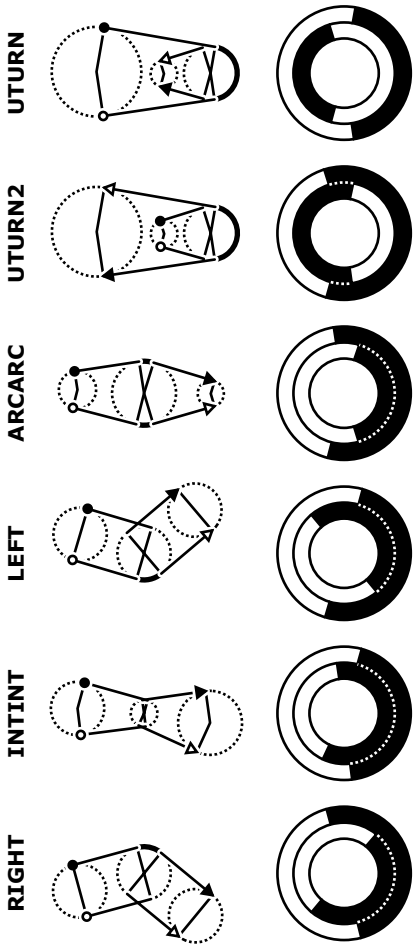


Abbildung 6.10: Die sechs Fälle der Knick-Klassifikation und die jeweiligen überlappenden Winkelbereiche.

6.6 Negative Linienstärke

Durch die gewählte Parametrisierung der Linienstärke sind Änderungen nur an den einzelnen Knoten einer Linie möglich, wodurch zwangsläufig Linienenden innerhalb auch nur auf eben diesen Knoten liegen können und zwar genau dann, wenn dort die Linienstärke 0 beträgt. Um diese Beschränkung aufzuweichen, wurde die Möglichkeit eingeführt negative Linienstärken zu verwenden. Segmente, bei denen die Vorzeichen der angrenzenden Linienstärken unterschiedlich sind, werden unterteilt. Dabei wird ein Segment an genau der Stelle unterteilt, an dem sich die Tangenten zu den beiden angrenzenden Radien im inneren schneiden würden (Abb. 6.11). Somit wird die Plazierung der Linienenden unabhängig von den Knoten eines Linienzuges.

Abbildung 6.11: Effekt einer schrittweisen verminderung des Radius an einem Knoten.

6.7 Punktierte Linien

Betrachtet man handgemachte Strichzeichnungen, so fällt einem neben der Variation der Linienstärke ein weiteres oft verwendetes Stilmittel auf: Punktieren. Dies wird bevorzugt benutzt, um auch unter der Verwendung von rein schwarzer Tinte und bei gleichbleibender Linienstärke unterschiedliche Helligkeitsabstufungen simulieren zu können. Dies sieht man entweder flächig, zur Darstellung von Schattierung und Oberflächendetails oder aber auch entlang von Linien, z.B. um Enden weicher auslaufen zu lassen. Punktieren entlang von Linien wurde auch in den Linien-Renderer eingeführt. So ist es möglich, die Dichte von Linien zu variieren und sie so in Folgen von Punkten aufzubrechen (siehe Abb. 5.6).

6.8 Punkt-Abstände

Um die Abstände zwischen den Punkten zu berechnen, wird zuerst der umgekehrte Fall untersucht; die wahrgenommene Helligkeit zweier benachbarter Punkte. Diese lässt sich über das Verhältnis aus schwarzer und weisser Fläche innerhalb eines Vierecks, dass zwischen den Punkten aufgespannt wird, abschätzen. Für eine vereinfachte Herleitung betrachtet man dies o.E.d.A. zwischen Einheitskreisen. Dabei gibt es zwei Fälle: entweder die Kreise sind mindestens zwei Einheiten auseinander und berühren sich höchstens tangential (Abb. 6.12) oder es gibt eine Überlappung (Abb. 6.13), wodurch man verhindern muss, dass der gemeinsame Teil doppelt gezählt wird. Für diese Fälle lassen sich leicht jeweils Formeln herleiten, die einem für einen gegeben Abstand d zwischen Einheitskreisen eine geschätze Helligkeit $I(d)$ liefern:

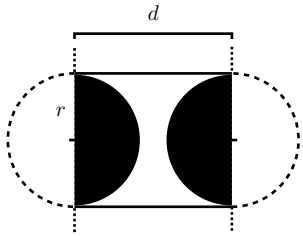


Abbildung 6.12: Abschätzung der wahrgenommenen Helligkeit zwischen zwei Punkten gleicher Größe für den Fall: d >= 2r.

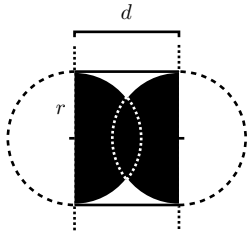


Abbildung 6.13: Abschätzung der wahrgenommenen Helligkeit zwischen zwei Punkten gleicher Größe für den Fall: d < 2r.

$$f(b) = \int_0^b \sqrt{1-x^2}dx = b\sqrt{1-b^2} + \arcsin b$$

$$I(d) = \begin{cases} \text{für } d \geq 2r & \frac{\pi r^2}{2dr} & = \frac{\pi r}{2d} \\ \text{sonst} & f(\frac{d}{2r})\frac{2r^2}{2dr} & = f(\frac{d}{2r})\frac{r}{d} \end{cases}$$

Eigentlich ist jedoch der umgekehrte Weg von Interesse. Fällt es für den ersten Fall noch leicht eine Umkehrfunktion anzugeben, so hat man mit dem zweiten erhebliche Probleme. Da aber nur praktische Genauigkeit und Geschwindigkeit wichtig sind, ist eine Näherungslösung völlig ausreichend und lässt sich leicht über die Verwendung einer Tabelle erzielen. Zu beachten ist, dass für die Umkehr-Tabelle die Funktionswerte gleichmässige Abstände haben sollten. Dazu wird die gegebene Funktion mit einem gedachten Raster entsprechend der gewünschten Anzahl von Sample-Punkten überzogen und von ihrem Minimum aus mit einer adaptiven Schrittweite abgetastet, die so angepasst wird, dass je ein Funktionswert zwischen den horizontalen Gitterlinien liegt. Der genau Schnittpunkt mit einer Gitterlinie wird über lineare Interpolation mit je einem Funktionswert über- und unterhalb ermittelt (Abb. 6.14).

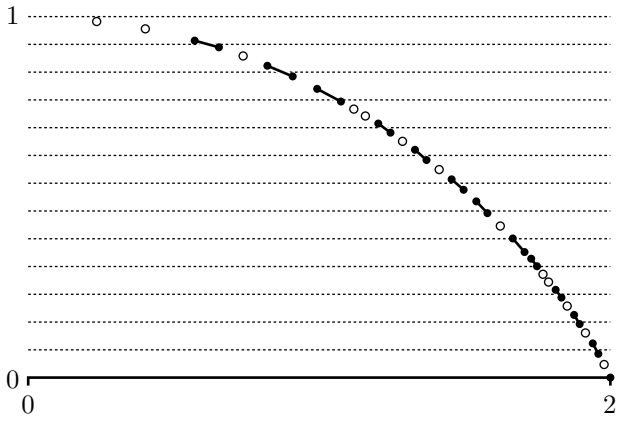


Abbildung 6.14:
Sampling über die Funktion, von der die Inverse gebildet werden soll, mit Segmenten, die die Gitterlinien schneiden.

Um mit den Abstandsinformationen die Punkte entlang eines Linienzuges zu plazieren, wird wie folgt vorgegangen: zuerst wird ein Linienzug gegebenenfalls in mehrere Teile zerlegt. Dies erfolgt immer dann, wenn ein Segment Knoten mit minimaler Dichte enthält, da dies nur mit unendlichen Abständen erreicht werden könnte, was nicht praktikabel ist. Danach wird jeweils vom Knoten maximaler Dichte begonnen, die Punkte auf dem Linienzug zu verteilen. Für jeden Punkt wird zuerst die Dichte linear interpoliert und damit der Abstand zum nächsten Punkt geschätzt. Dies ist jedoch nur eine erste Approximation des Abstands, da ja die Dichte für den Bereich zwischen zwei Punkten angenähert werden soll und nicht die Dichte direkt an einem Punkt. Darum wird dieser Wert benutzt, um die Dichte in diesem Bereich des Linienzuges zu mitteln und erst mit dem Ergebnis den endgültigen Abstand zu bestimmen. Das ist natürlich auch nur eine Annäherung an den tatsächlichen Wert, aber in der Praxis hat sich gezeigt, dass dies einen guten Kompromiss aus Genauigkeit, Geschwindigkeit und Ausgeglichenheit der Plazierung ergibt.

6.9 Minimale Punkt-Abstände

In den meisten Fällen ist die Ästhetik des Übergang zwischen vollen und punktieren Liniensegmenten wichtiger als eine perfekte Annäherung der Dichte in allen Regionen. Eine Beobachtung, die man bei handgemachten Illustrationen macht, ist, dass die Punkte oft einen kleinen Abstand zu den vollen Liniensegmenten haben. Als Äquivalent ist es möglich, einen minimalen Abstand für die Punkte anzugeben (Abb. 6.15). Um diesen zu erreichen, werden Liniensegmente an genau den Stellen unterteilt, an denen der Dichtegrad genau dem gewünschten Abstand entspricht. Die Teilhälfte

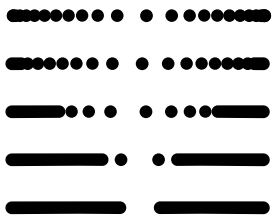


Abbildung 6.15:
Unterschiedliche minimale Abstände im Bereich von [0,4].

bei der der Abstand zu klein wäre, wird als volles Segment gezeichnet und die andere entsprechend punktiert.

Ein Problem, dass sich noch durch die schrittweise Plazierung der Punkte entlang eines Linienzuges ergibt, sind die Enden. Konnte am Anfang noch auf den richtigen Abstand geachtet werden, endet die Plazierung der Punkte sobald ein neuer Punkt dichter am Ende sitzen würde als erlaubt, wodurch meist eine überaus unschöne Lücke entsteht. Daher werden die Punkte in zwei Schritten plaziert. Zuerst werden nur ihre relativen Positionen vermerkt. Wurde dies für alle Punkte eines Abschnitts abgeschlossen, so wird der komplette Abschnitt soweit gestreckt, dass der minimale Abstand auch für das Ende eingehalten wird (Abb. 6.16).

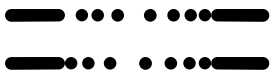


Abbildung 6.16: Punktplazierung ohne und mit Ausgleich.

6.10 Ausgabe

Für OpenNPAR wurden zwei Ausgabe-Module entwickelt, die sehr unterschiedliche Ziele haben. Zum einen eine WYSIWYG-Vorschau für den Benutzer. Diese ermöglicht es, das Modell mit interaktiven Bildraten zu betrachten, den Blickwinkel oder das Shading-Modell zu verändern, um die einzelnen Parameter optimal wählen zu können. Dies wird mit Hilfe eines auf Geschwindigkeit optimierten Linienrenderers gelöst, der, wie in den Abschnitten zuvor beschrieben, die Linienzüge in Triangle-Strips und Punkte in Triangle-Fans zerlegt. Dabei werden Kreisbögen mit einer durch den Benutzer gegebenen Auflösung diskretisiert und Linien noch leicht vereinfacht. So lässt sich die Qualität je nach Bedarf anpassen.

Für die Darstellung von monochromen Linien auf einem niedrig aufgelösten Rasterschirm ist es sehr wichtig, Maßnahmen gegen Aliasing-Probleme vorzunehmen. Auf aktuellen Graphikkarten bietet sich da im besonderen *Full-Scene-Anti-Aliasing* an, was jedoch mit der vorhandenen Implementierung leider noch nicht möglich ist, da es einen Konflikt mit dem Hidden-Line-Remover-Modul von OpenNPAR gibt. Bei der Darstellung eines durchschnittlich komplexen Modells mit etwa 900 Stromlinien, die aus insgesamt etwa 12700 Einzel-Segmenten bestehen, führt dies zu dramatischen Geschwindigkeitseinbußen: Wird das HLR-Modul deaktiviert, so ist eine Darstellung mit etwa 20 FPS möglich, doch schaltet man es ein, so sinkt die Framerate auf nicht akzeptable 2 FPS (Auf einem Dual Athlon 1600+ und Geforce2 MX400). Bis zur Fertigstellung des Praktikums konnte die genau Ursache im HLR-Modul leider nicht mehr identifiziert werden, wodurch die Qualität der Vorschau eingeschränkt bleibt.

Neben der WYSIWYG-Vorschau gibt es jedoch noch ein zweites Ausgabe-Modul, mit dem Vektorpfade in eine PDF-Datei exportiert werden können. Dabei werden Kreisbögen natürlich nicht mehr diskretisiert, sondern in ihrer nativen Form belassen. Diese Dateien lassen sich dann nicht nur am Rechner mit maximaler Darstellungs-Qualität betrachten, sondern auch weiterverarbeiten, was sich insbesondere für die hochqualitative Reproduktion im Print-Bereich anbietet.

Da das PDF-Format inzwischen in weiten Teilen des Desktop-Publishing-Sektors Einzug gehalten hat und von den gängigen Programmen unterstützt wird, ist es auch kein

Problem, diese Dateien ohne weitere Konvertierung direkt weiterzuverarbeiten. Daraus ergeben sich vielfältige Möglichkeiten; so lassen sich mehrere Schraffurebenen erzeugen, indem das komplette Richtungsfeld leicht gedreht und das Schattierungs-Modell entsprechend angepasst wird (Abb. 6.17 und 6.18) . Dies lässt sich natürlich weiterführen und so lassen sich auch ganze Collagen zusammensetzen und Flächen einfärben (Abb. 6.19). Man ist also nicht mehr nur auf schwarz/weiß-Graphiken angewiesen und hat weiterhin auch den Vorteil, dass die Farbwahl nicht auf Prozessfarben beschränkt ist, sondern anstelle dessen Volltonfarben verwenden werden können. Dadurch können farbige Bilder später rasterfrei gedruckt werden, was natürlich der Darstellungsform von Strichzeichnungen zugute kommt.

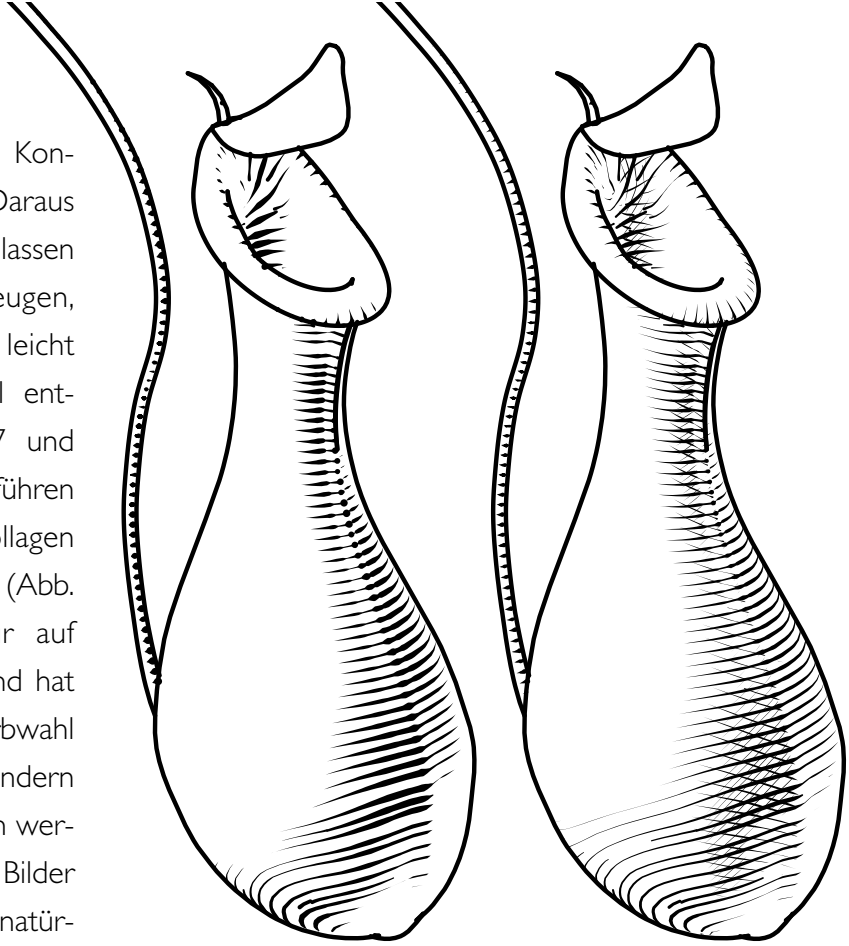


Abbildung 6.17: Kannenpflanze mit einfachen Oberflächen-Linie, sowie einer Komposition aus drei Einzelbildern, deren Richtungsfelder jeweils leicht gedreht wurden.

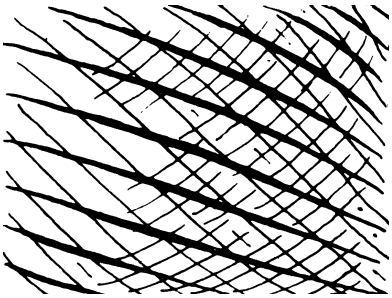


Abbildung 6.18a: [Hodges, 1989, Abb. 115] Schraffur in einer handgemachten Illustration.

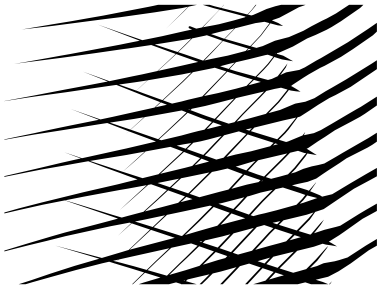


Abbildung 6.18b: Detailausschnitt aus Abb. 6.17.

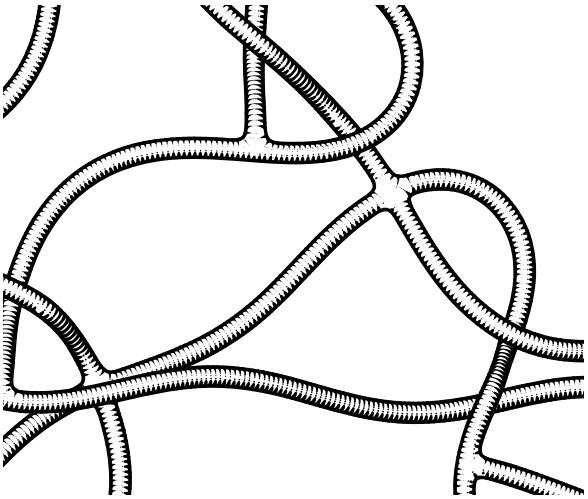


Abbildung 6.19a: Prozedurale Geometrie, versehen mit einer Linien-Schattierung.

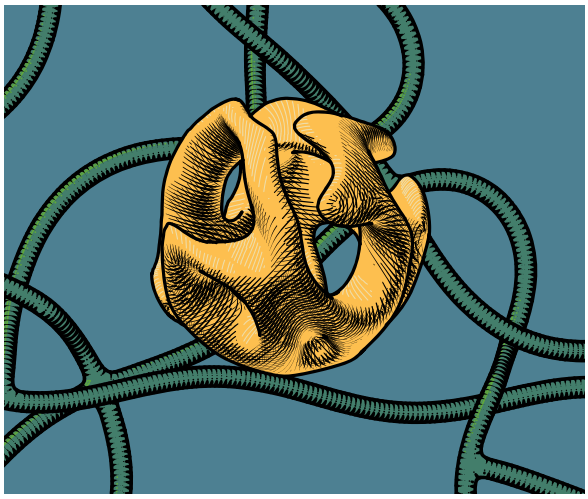


Abbildung 6.19b: Kolorierte Komposition.

7. OpenNPAR

Bei OpenNPAR handelt es sich um ein NPR-Framework, das am Institut für Simulation und Graphik der Otto-von-Guericke-Universität Magdeburg entwickelt wurde. Als Grundlage dient dabei OpenInventor, dessen Szenengraph verwendet wird um die Kommunikation mehrerer Graphen-Knoten untereinander zu koordinieren. Dabei realisiert jeder Graphen-Knoten einen kleinen Teil des gesammten Prozesses und erst durch den Verbund mehrerer Knoten in einer Pipeline entsteht das eigentliche Programm.

Im folgenden wird die Einbettung dieser Arbeit in OpenNPAR an einer leicht vereinfachten Abfolge der Knoten in der OpenNPAR-Pipeline und grob der generelle Informationsfluss zwischen den einzelnen Knoten angerissen:

SoGenerate-WingedEdge

Dieser Knoten erzeugt, falls nötig, aus den Rohdaten eines Modells eine *WingedEdge*-Repräsentation.

SoModify-WingedEdge

Auf die zuvor erstellte *WingedEdge*-Struktur werden bei Bedarf Operationen wie Glättung oder Subdivision-Surfaces angewendet.

SoGenerate-HeModelHQ

Die *WingedEdge*-Struktur wird für die weitere Verwendung in eine *HalfEdge*-Struktur umgewandelt. Falls die zugrundeliegden Flächen mehr als drei Kanten haben, so werden die Flächen trianguliert.

SoGenerate-CurvatureHQ

Ausgehend von der *HalfEdge*-Struktur wird für jeden Knoten des Modells *Curvature*-Information erzeugt, die minimale- und maximale Krümmung, sowie die Krümmungsrichtungen enthält.

SoGenerate-FlowFieldHQ

Aus der *Curvature*- und *HalfEdge*-Struktur werden für jeden Knoten möglichst passende Richtungsvektoren gewählt und in einem *FlowField* gespeichert. Bei Bedarf kann dieses Feld noch vom Benutzer optimiert werden.

SoWingedEdge

Stellt eine *WingedEdge*-Struktur dar, wobei nach Voreinstellung nichts angezeigt wird, sondern diese nur in den z-Buffer gezeichnet wird.

SoGenerate-Silhouette

Aus der *WingedEdge* Struktur werden die Silhouetten-Kanten extrahiert und in Form von *Coordinates* und *LineCoordinateIndices* verfügbar gemacht. Das wird im folgenden als *Stroke* bezeichnet.

SoLineHidden-LineRemover

Ein Silhouetten-*Stroke* wird mit Hilfe des zuvor geschriebenen z-Buffers von allen verdeckten Bereichen befreit.

SoLineThickener

Für einen *Stroke* werden zusätzlich noch *Line-Thickness*-Informationen angelegt um festzulegen, wie dick die Linie an welchen Stellen werden soll. Für die Silhouetten-Kanten wurde jedoch eine konstante Breite verwendet.

SoLineMergerHQ

Alle bisherigen *Stroke*-Daten werden zwischen gespeichert. Für nachfolgende Knoten wird diese Information vorerst aus der Pipeline genommen.

SoStreamlinerHQ

Es wird mittels einer *HalfEdge*-Struktur ein *FlowField* integriert. Die dabei entstehenden Stromlinien werden in Form von *Strokes* gespeichert. Zusätzlich wird noch Information über die Dreiecke, zu denen ein Knoten einer Stromline gehört (*LineTriangle*), sowie für jeden Knoten ein Verjüngungs-Faktor (*LineTaper*) beigefügt.

SoLineHidden-LineRemover

Auch an dieser Stelle werden nicht sichtbare Bereiche der *Strokes* entfernt. Durch den Bezug zwischen Linie und zugrundeliegendem Dreieck können jedoch auch alle auf vom Betrachter abgewendeten Dreiecken befindliche Strokes direkt entfernt werden. Beim Beseitigen von Teilen eines *Strokes* muss jedoch auch der Dreiecksbezug, sowie die Verjüngungs-Faktoren entsprechend behandelt werden.

SoLine-ThickenerHQ

Ähnlich des normalen *LineThickeners* wird auch hier jedem Knoten eines *Strokes* eine Dicke zugewiesen. Durch den Dreiecksbezug ist es jedoch möglich, die Oberflächennormale an der entsprechenden Stelle durch Interpolation zu ermitteln, wodurch die Liniendicke auch mit Licht geformt werden kann.

SoLine-DensifierHQ

Dieses Modul ist identisch zum *SoLineThickenerHQ*, wobei jedoch anstelle der Dicke die *LineDensity* verändert werden kann. Dadurch lässt sich für einen *Stroke* die Liniendichte (Abb. 5.6) beeinflussen.

SoLineMergerHQ

Die zuvor gespeicherten Daten der Silhouetten-Kanten werden mit den neu erstellten *Strokes* verbunden und dann als eine Menge von Strokes den nachfolgenden Knoten in der Pipeline zur Verfügung gestellt.

SoLineSettingsHQ

Da es mehrere Renderer gibt, die mit den gleichen Einstellungen arbeiten, werden diese in einem Knoten in Form einer *LineSetting* gespeichert.

SoLineOutput-TgaHQ

Ein Stroke-Renderer, der mit Hilfe der Library [LibArt] einen "Screenshot" in Form eines TGA-Bildes mit vollem Antialiasing erzeugen kann.

SoLineOutput-PdfHQ

Analog zum vorherigen Knoten wird ein "Screenshot" mit Hilfe der Library [ClibPDF] erzeugt. Dabei werden direkt die Vektor-Daten in einer PDF-Datei ablegt.

SoLineShapeHQ

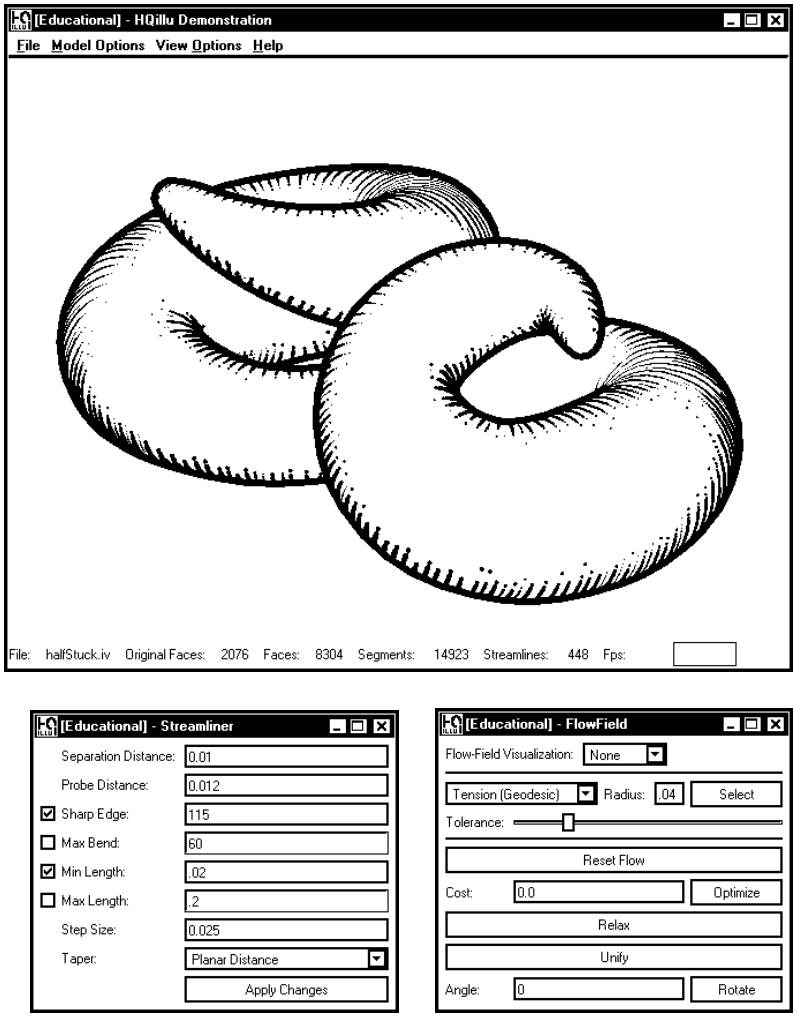
Ein interaktive Stroke-Renderer, der die Strokes trianguliert und in Form von OpenGL-Vertex-Arrays auf den Bildschirm bringt.

Rahmen-Legende:

- ——— *neuentwickelte Knoten*
- - - - - *modifizierte Knoten*
- ..... *vorhandene Knoten*

8. Beispiele

Die graphische Benutzeroberfläche der Anwendung wurde, aufbauend auf einer Beispielanwendung für OpenNPAR in QT realisiert (Abb. 8.1).



Für die Manipulation der wichtigsten OpenNPAR-Knoten stehen dem Benutzer eine Reihe nicht-modaler Dialogfelder zu Verfügung. In diesen lassen sich die Parameter der Knoten ändern und sich deren Auswirkungen direkt im Hauptfenster verfolgen. Bei Operationen, die für ihre Bearbeitung eine gewisse Zeit benötigen, wie z.B. das Verfolgen von Stromlinien, wird am unteren Rand des Hauptfensters ein Fortschritts-Balken angezeigt. Dort finden sich auch einige Statusinformationen wie Dateiname des aktuell geladenen Modells, sowie Anzahl der Flächen, Stromlinien und daraus resultierenden Segmente.

Über das File-Menü lässt sich ein "Bildschirmfoto" in Form einer PDF-Datei abspeichern (Abb. 8.2). Das Papierformat wird dabei so gewählt, dass zwar die Proportionen des Fensters und die Plazierung der Graphik innerhalb des Fensters übernommen werden, aber sichergestellt wird, dass das Format bestmöglich auf eine DIN-A4 Seite passt, also mindestens eine der Seitenkanten dem Format einer DIN-A4 Seite entspricht.

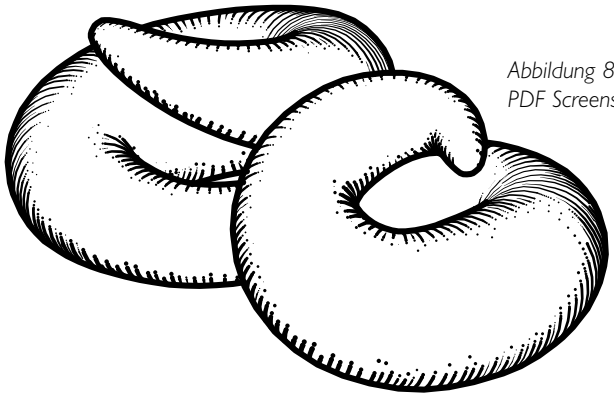


Abbildung 8.2: PDF Screenshot.

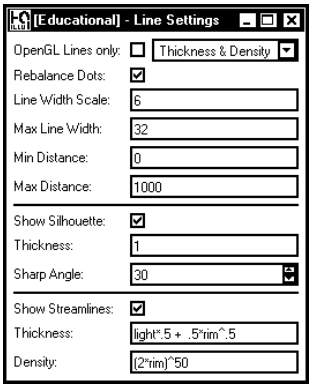


Abbildung 8.1: Das Hauptfenster und drei der wichtigsten Dialogfelder der Anwendung.

Um zu verdeutlichen, wie mit den Ausdrücken für das Beleuchtungsmodell gearbeitet werden kann, folgt ein kleiner Überblick über den Verlauf einer Beispielsitzung:

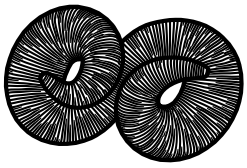


Abbildung 8.3:
thickness = 1
density = 1

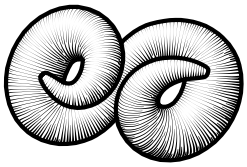


Abbildung 8.4:
thickness = light
density = 1

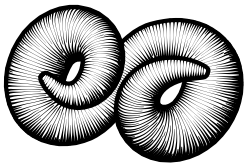


Abbildung 8.5:
thickness = light+rim
density = 1

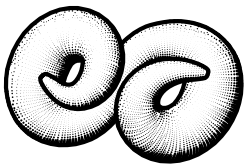


Abbildung 8.6:
thickness = light+rim
density = light

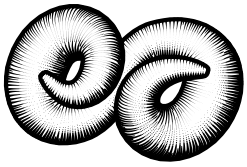


Abbildung 8.7:
thickness = light+rim
density = 5rim

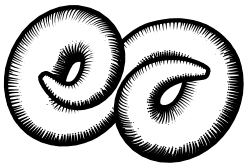


Abbildung 8.8:
thickness = light+rim
density = (5rim)[50]

Mit konstanten Faktoren für die Linienbreite und -dichte werden die Linien nur noch durch Verjüngungseffekte sowie den Korrekturfaktor beeinflusst (Abb. 8.3). Dadurch entsteht der Eindruck eines halbwegs konstanten Tonwerts. Moduliert man die Linienbreite dagegen über die Beleuchtung, so entsteht ein Eindruck von räumlicher Tiefe (Abb. 8.4). Fügt man dem Term noch Rim-Shadows hinzu, so wird dieser Effekt durch stärkere Abdunklungen entlang der Objektkonturen noch deutlicher (Abb. 8.5). Benutzt man die Beleuchtung zusätzlich, um die Liniendichte zu verringern, so hellt sich das Bild auf und es entsteht ein Raster ähnlicher Effekt (Abb. 8.6). Um diesen auf einen kleineren Bereich einzuschränken reicht es, den Term mit einer Konstante zu multiplizieren (Abb. 8.7). Möchte man den Bereich andererseits von der entgegengesetzten Seite her einschränken, um mehr Weissraum zu schaffen, so erreicht man das z.B. durch das Hinzufügen einer Potenz (Abb. 8.8). Dadurch entsteht ein Effekt, der einem Glanzlicht nahe kommt.

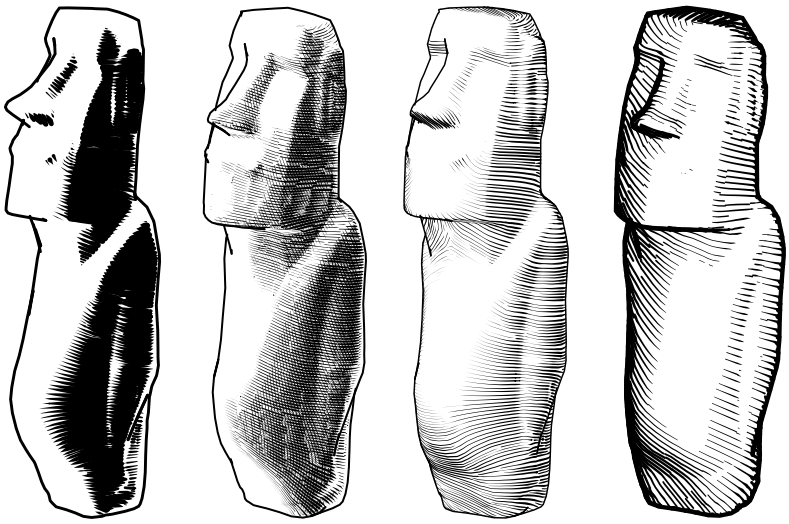
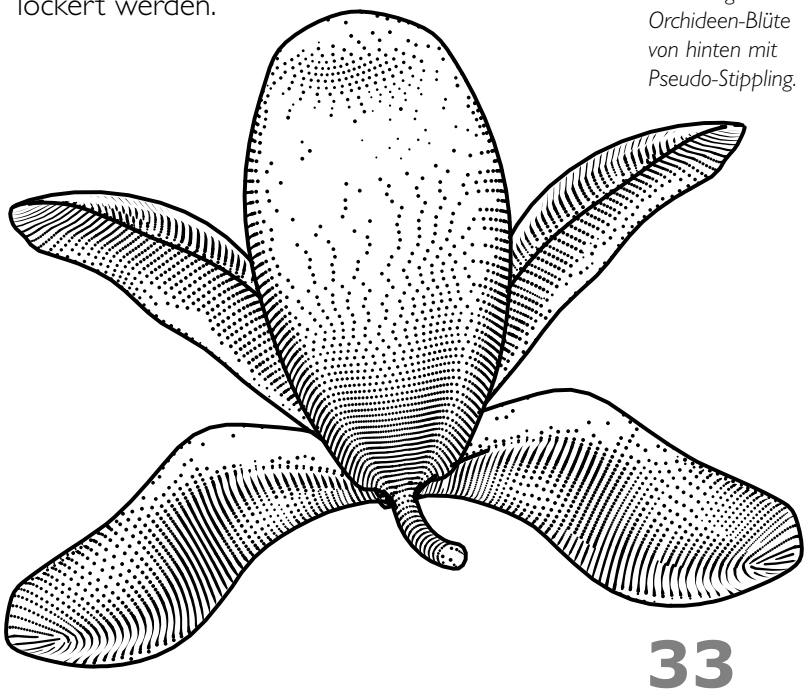


Abbildung 8.9: Darstellung einer Moai-Statur mit unterschiedlichen Effekten.

Abb. 8.9 zeigt dagegen nicht nur die Bedeutung der Schattierungsausdrücke, sondern zusätzlich auch den Einfluss unterschiedlicher Linienabstände für den Eindruck des vermittelten Details, sowie die Überlagerung von mehreren Schraffurebenen für eine deutlichere Schattierung.

Indem die Linienbreite grösstenteils konstant gehalten wird und nur die Liniendichte verwendet wird um Schattierung zu erzielen, entsteht ein Effekt, der dem Punktieren sehr nahe kommt (Abb. 8.10). Für eine bessere Qualität des Effekts sollte jedoch die deterministische Punktplazierung durch Zufallseinflüsse aufgelockert werden.

Abbildung 8.10: Orchideen-Blüte von hinten mit Pseudo-Stippling.



9. Fazit

Wie an den Beispielen ersichtlich wird, eignet sich der gewählte Ansatz sehr gut um Illustrationen in hoher Qualität erstellen zu können. Es bedarf jedoch noch weiterer Arbeit, um dies einfacher zu gestalten.

Ein grosses offenes Problem ist zum Beispiel die einfache Erstellung eines Richtungsfeldes. So sollte der eher technische Ansatz zwar den Benutzer entlasten, aber am Ende bleibt dadurch nicht genügend Einflussmöglichkeit auf das Feld. Oft wünscht man sich die Möglichkeit, von Hand einzelne Bereiche verändern zu können, weil der vom Programm gewählte Fluss eher unnatürlich ist und nicht so verläuft, wie man es intuitiv für richtig hält. Hier wäre ein Prozess, der den Benutzer stärker einbindet, sicher vorteilhaft. Auch die Möglichkeit, mehrere Algorithmen auf unterschiedlichen Bereichen des Modells anwenden zu können, würde sicher gut in solch einen Arbeitsablauf passen. In diesem Zusammenhang schwebt mir zum Beispiel die Möglichkeit vor, in einer Art 3D-Malprogramm nicht nur direkt auf dem Modell das Richtungsfeld "kämmen" zu können, sondern auch "malend" Teilbereiche auszuwählen, auf denen man bestimmte Algorithmen anwenden möchte und das nicht mit einer binären Logik, sondern der Möglichkeit die Stärke des Einflusses unterschiedlich stark wichten zu können. Daraus ergibt sich eher ein Arbeitsablauf, wie er auch ähnlich in Bildbearbeitungsprogrammen zum Einsatz kommt, wo ja auch von Hand eine weiche Maske gemalt werden kann, die dann benutzt wird, um einen Filter nur auf einzelne Teilbereiche des Bildes anzuwenden.

Andererseits gibt es auch noch einige Probleme mit der Erstellung der Stromlinien. Der gewählte Algorithmus neigt dazu, Artefakte zu produzieren, die durch ihre Deutlichkeit stören (Abb. 9.1). Das ist schon in der ursprünglichen 2D-Form des Algorithmus der Fall und ist wohl eher ein grundsätzliches Problem.

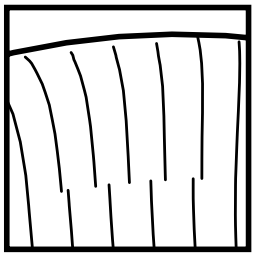


Abbildung 9.1: Stromlinien Artefakte.

Dabei kollidiert eine Stromlinie frühzeitig mit der Stromlinie von der sie erzeugt wurde und verursacht dadurch eine Blockade für folgende Stromlinien, wodurch sich der Fehler auf einen grösseren Bereich ausdehnt. Um das zu verhindern schweben mir zwei unterschiedliche Wege vor. Eine Möglichkeit wäre, mehr Sorgfalt bei der Wahl eines Startpunktes zu treffen. Eine einfache aber auch teure Strategie wäre es also, erst alle Kinder einer Stromlinie zu erzeugen und von diesen die längste zu wählen. Da dann jedoch nicht nur der Reihe nach die n möglichen Startpunkte für Kinder in Betracht gezogen werden müssen, sondern jedesmal wieder alle verbleibenden Startpunkte getestet werden müssen, würde der Aufwand beim Erzeugen von $O(n)$ auf $O(n^2)$ steigen, was vermutlich zu einer enormen Verlangsammung führen würde. Alternativ könnte man eine einfachere Heuristik testen bei der man die Startpunkte, die eine Stromlinie erzeugt nach ihrer "Tangentialität" zur Ursprungs-Stromlinie sortiert, denn solange das Feld in der Nachbarschaft auf die Stromlinie zu geht und nicht vollkommen tangential dazu verläuft, ist es sehr wahrscheinlich, dass eine dort initiierte Stromlinie mit der ursprünglichen kollidieren wird und so eine zu kurze Stromlinie erzeugt wird.

Ein ganz anderer Ansatz wäre es, die Stromlinien hierarchisch zu erzeugen, also den Abstand der Linien jedesmal zu halbieren, wenn sich keine neuen Stromlinien mehr einfügen lassen und das solange, bis der endgültige minimale Abstand unterschritten wird. Diese hätte auch gleichzeitig den Vorteil, dass man die dabei entstehende Linien-Hierarchie benutzen kann, um bei der Darstellung je nach Distanz zum Betrachter die Anzahl der Linien anzupassen.

Ein anderes Problem hängt letztlich damit zusammen. Dadurch, dass das Verfahren im Objektraum arbeitet, sind die Linienabstände nicht so gleichmässig, wie es bei einem ähnlich Verfahren im Bildraum der Fall wäre. Hier wäre es möglicherweise besser, einen hybriden Ansatz zu wählen, der für die interaktive Darstellung arbeitet wie gehabt, aber auch auf Wunsch Bilder mit höherer Qualität erzeugen kann. Da würde es sich zum Beispiel anbieten, einen ähnlichen Weg wie [Hertzmann & Zorin, 2000] zu wählen, also Linien erst nach der Projektion zu erzeugen.

Literatur

[Deussen, 1999]
O, Deussen, J, Hamel, A, Raab, S, Schlechtweg und T, Strothotte: ***An illustration technique using hardware-based intersections and skeletons.*** In Proceedings of Graphics Interface '99 (1999), Morgan Kaufmann Publishers Inc., pp. 175-182

[Girshick, 2000]
A. Girshick, V. Interrante, S. Haker, T. Lemoine: ***Line Direction Matters: An Argument For The Use Of Principal Directions In 3D Line Drawings.*** In Proceedings NPAR 2000 (New York, 2000), ACM Press, pp. 43-52

[Goldfeather, 2001]
J. Goldfeather: ***Understanding Errors in approximating principal direction vectors.***
Technical Report 01-006, University of Minnesota, 2001

[Hertzmann & Zorin, 2000]
A. Hertzmann und D. Zorin: ***Illustrating smooth surfaces.***
In Proceedings of SIGGRAPH 2000 (New York, 2000), ACM Press, pp. 517-526

[Interrante, 1997]
V. Interrante: ***Illustrating Surface Shape in Volume Data via Principal Direction-Driven 3D Line Integral Convolution.*** In Proceedings of SIGGRAPH'97 (New York, 1997), pp. 109-116

[Isenberg, 2002]
T. Isenberg, N. Halper und T. Strothotte: ***Stylizing Silhouettes at Interactive Rates: From Silhouette Edges to Silhouette Strokes.*** Computer Graphics Forum 21-3 (September 2002), pp. 249-258.

[Hsu & Lee 1994]
S. C. Hsu und I. H. H. Lee: ***Drawing and Animation Using Skeletal Strokes.***
In Proceedings of SIGGRAPH'94 (Orlando, July 1997), pp. 109-118

[Jobard & Lefer, 1997]
B. Jobard und W. Lefer: ***Creating Evenly-Spaced Streamlines of Arbitrary Density.***
In Proceedings of the 8th Eurographics Workshop on Visualization in Scientific Computing (1997), pp. 45-55

[Kalnins, 2003]
R. D. Kalnins, P. L. Davidson, L. Markosian und A. Finkelstein: ***Coherent Stylized Silhouettes.***
In Proceedings of SIGGRAPH 2003 (New York, 2003), ACM Press, pp. 856-861

[Leister, 1994]
W. Leister: ***Computer Generated Copper Plates.***
Computer Graphics Forum 13-1 (Mar. 1994), pp. 69–77

[Mäntylä, 1988]
M. Mäntylä: ***An Introduction to Solid Modeling. Principles of Computer Science.***
Computer Science Press, Maryland, U.S.A, 1988, pp. 401

[Oustromoukhov, 1999]
V. Oustromoukhov: ***Digital Facial Engraving.***
In Proceedings of SIGGRAPH'99 (New York, 1999), ACM Press, pp. 417–424

[Pnueli & Bruckstein, 1994]
Y. Pnueli und A. M. Bruckstein: ***DigiDürer - A Digital Engraving System.***
The Visual Computer 10-5 (Apr. 1994), 277–292

[Press, 1992]
W. H. Press, S. A. Teukolsky, W.T. Vetterling und B. P. Flannery: ***Numerical Recipes in C.***
Cambridge University Press, New York, 1992, Second Edition

[Praun, 2001]
E. Praun, H. Hoppe, M. Webb und A. Finkelstein: ***Real-Time Hatching.***
In Proceedings of SIGGRAPH 2001 (New York, 2001), ACM Press, pp. 581–586

[Rössl, 1999]
C. Rössl: ***Semi-Automatische Methoden für die Rekonstruktion von CAD-Modellen aus Punktdaten.***
Diplomarbeit, Friedrich-Alexander-Universität Erlangen-Nürnberg, 1999

[Rössl & Kobbelt, 2000]
C. Rössl und L. Kobbelt: ***Line-Art Rendering of 3D-Models.***
In Proceedings of WSGC'2000 (2000), The University of West Bohemia, Plzen, Czech Republic, pp. 168–175.

[Salisbury, 1994]
M. P. Salisbury, S. E. Anderson, R. Barzel und D. H. Salesin: ***Interactive pen-and-ink illustration.***
In Proceedings of SIGGRAPH'94 (New York, 1994), ACM Press, pp. 101–108

[Strothotte & Schlechtweg, 2002]
T. Strothotte und S. Schlechtweg: ***Non-Photorealistic Computer Graphics: Modeling, Rendering, and Animation.***
Morgan Kaufmann, San Francisco, 2002

[Vehmeier, 2002]
B. Vehmeier: ***Qualität in Liniendarstellungen durch lokale Informationen.***
Diplomarbeit, Otto-von-Guericke-Universität Magdeburg, 2002

[Winkenbach & Salesin, 1994]
G. Winkenbach und D. H. Salesin: ***Computer-Generated Pen-and-Ink Illustration.***
In Proceedings of SIGGRAPH'94 (New York, 1994), ACM Press, pp. 91–100.

[Winkenbach & Salesin, 1996]
G. Winkenbach und D. H. Salesin: ***Rendering Parametric Surfaces in Pen and Ink.***
In Proceedings of SIGGRAPH'96 (New York, 1996), ACM Press, pp. 469–476.

[Zhu & Byrd, 1997]
C. Zhu, R. H. Byrd, P. Lu und J. Nocedal: ***L-BFGS-B - Fortran Subroutines for Large-Scale Bound Constrained Optimization.*** ACM Trans. Math. Software 23-4 (1997), pp. 550--560

Bild-Referenzen

[Brockhaus, 1894]
Brockhaus' Konversations-Lexikon
Verlag F. A. Brockhaus, Leipzig • Berlin • Wien
14. Auflage 1894

[Durst, 1991]
Rohen Durst: ***Chirurgische Operationslehre***
Schattauer-Verlag, Stuttgart • New York, 1991

[Haeckel, 1998]
Ernst Haeckel: ***Kunstformen der Natur***
Prestel Verlag, München • New York, 1998

[Hodges, 1989]
Elaine R. S. Hodges: ***The Guild Handbook of Scientific Illustration***
Verlag Van Nostrand Reinhold
New York, 1989

[Kristahn, 1983]
Heinz-Jürgen Kristahn: ***Japanische Plakate***
Vier-Türme-Verlag, Münsterschwarzach, 1983

[Merian,1999]
Maria Sibylla Merian: ***Neues Blumenbuch***
Prestel Verlag, München • London • New York, 1999

[Rogers, 1992]
A. W. Rogers: ***Anatomy***
Verlag Churchill Livingston
Edinburgh • Madrid • Melbourne • New York • Tokyo, 1992

[Saegesser, 1987]
M. Saegesser: ***Spezielle chirurgische Therapie***
Verlag Hans Huber
Bern • Stuttgart • Toronto
10. Auflage, 1987

[Tortora, 1997]
Gerhard J. Tortora: ***Introduction to the Human Body***
Verlag Benjamin Cummings
Menlo Park, California • Reading, Massachusetts • New York, 1997
Fourth Edition

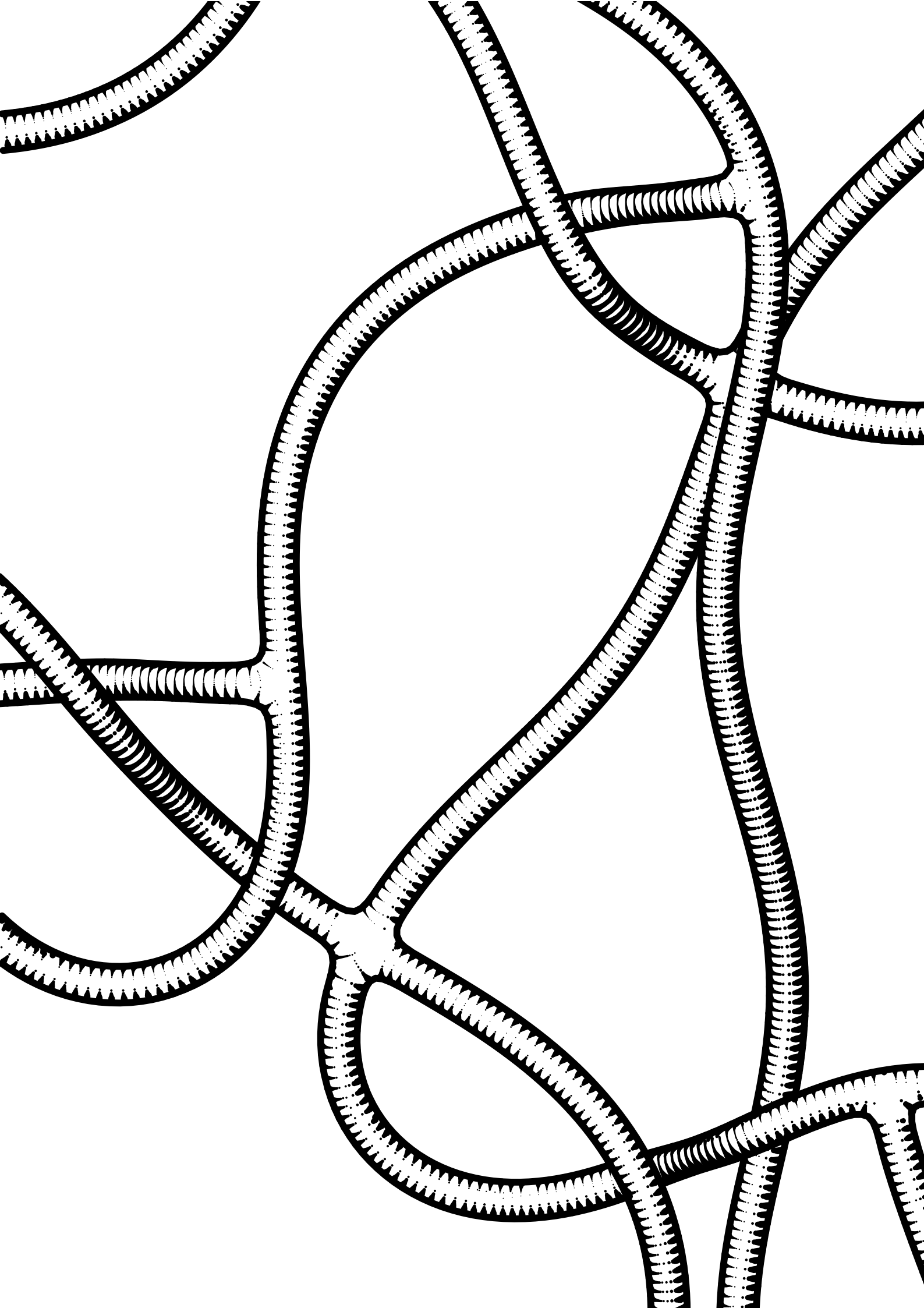
[Wood, 1994]
Phyllis Wood: ***Scientific Illustration***
Verlag John Wiley & Sons
New York • Chichester • Weinheim • Brisbane - Singapore • Toronto, 1994
Second Edition

WWW-Referenzen

[ClibPDF]
http://www.fastio.com/

[F2C]
http://www.netlib.org/f2c/

[LibArt]
http://www.levien.com/libart/

[OpenNPAR]
http://opennpar.org/

[Terdiman]
http://www.codercorner.com/Strips.htm