Graphische Systeme
Zusammenfassung

Michael Gregorius

15. November 2002

http://www.michaelgregorius.de/

Vorwort

Diese Zusammenfassung wurde im Rahmen der Vorbereitung auf die zweite Hauptdiplomsprüfung (Inf2) im Studienfach Kerninformatik an der Universität Dortmund erstellt. Die Prüfung erstreckte sich über die Vorlesungen *„Graphische Systeme“* und *„Betriebssysteme“* und wurde am 21. 10. 2002 erfolgreich bei Prof. Dr. Heinrich Müller abgelegt.

Für die Erstellung einer eigenen Zusammenfassung in LaTeX sprachen mehrere Gründe. Zum einen hatte ich schon für einige Prüfungen des Grundstudiums Zusammenfassungen geschrieben und damit sehr gute Erfahrungen gemacht. Denn schon beim Schreiben der Texte lernt man einen großen Teil des Stoffes und da alle Zusammenfassungen, so wie auch diese, in Frage-Antwort-Form geschrieben wurden, wusste man in der Prüfung auch direkt, wie man ein Thema bzw. Themengebiet gut erklären kann. Zum anderen bestand das „Skript“ [Mül00] zur Vorlesung lediglich aus Folienkopien, anhand derer allein der Stoff nicht wirklich gut gelernt werden konnte. Daher lag es nahe, den Stoff der Folien und aus weiterführender Literatur ([Fel92] und [Fol94]) in einer eigenen Zusammenfassung zu konzentrieren, mit welcher der Stoff im Anschluss wiederholt und gelernt werden konnte.

Zur Vorlesung *„Betriebssysteme“* wurde ebenfalls eine Zusammenfassung geschrieben, welche, wie die etwas weiter oben schon erwähnten anderen Zusammenfassungen sowie dem Prüfungsprotokoll zu dieser Prüfung, auf meiner Homepage unter `http://www.michaelgregorius.de/` heruntergeladen werden kann. Auf dieser Seite findet sich auch eine Email-Adresse, unter der man mich erreichen kann. Über Kommentare, Meinungen und sonstige Rückmeldungen würde ich mich freuen.

Dortmund, den 11. 11. 2002 *Michael Gregorius*

Inhaltsverzeichnis

1 Graphische Hardware

1.1 Sichtgeräte

Frage 1.1: Wie ist eine Kathodenstrahlröhre aufgebaut?
Die Kathodenstrahlröhre wurde von Ferdinand Braun entwickelt. Sie besteht grob aus den folgenden Komponenten:

Röhre: Die Röhre ist luftdicht abgeschlossen und ist auf 10^{-6} Torr evakuiert. Aufgrund des Drucks muss die Frontscheibe gewölbt sein.

Glühkathode: Die Glühkathode erzeugt mittels des glühelektrischen Effekts Elektronen.

Wehnelt-Zylinder: Zwischen der Glühkathode und dem Wehnelt-Zylinder liegt eine Spannung an, durch die die Menge der abgestrahlten Elektronen gesteuert wird. Dadurch wird die Helligkeit der Bilddarstellung beeinflusst.

Fokussier-System: Damit die ausgestrahlten Elektronen nicht divergieren, werden sie mit Hilfe des Fokussier-Systems zu einem Strahl gebündelt.

Beschleunigungsbereich: In diesem Bereich werden die emittierten Elektronen mit Hilfe eines elektrischen Feldes (15000-20000V), welches zwischen Steuergitter und Anode anliegt, beschleunigt.

Ablenkungseinrichtung: Sie positioniert den Elektronenstrahl auf dem Bildschirm.

Phosphorschicht: Trifft der Elektronenstrahl auf die Phosphorschicht, leuchtet der getroffene Punkt kurz auf, ein Effekt, der **Floureszenz** genannt wird. Der Effekt, das nach dem Treffer noch Licht emittiert wird, wird **Phosphoreszenz** genannt. Verschiedene Phosphore können durch Farbton und Nachleuchtdauer klassifiziert werden.

Frage 1.2: Welche Betriebsarten gibt es bei einer CRT?
Man kann zwei Betriebsarten unterscheiden:

Vektorscan: Beim Vektorscan-Verfahren wird der Elektronenstrahl entsprechend des Bildinhaltes über den Bildschirm bewegt. Um zum Beispiel eine Linie zu zeichnen, würde der Strahl auf einen Punkt gerichtet, angeschaltet und dann zum Zielpunkt hin bewegt werden. Beginnt die nächste zu zeichnende Linie woanders, würde er ausgeschaltet und dorthin bewegt werden.

Bei diesem Verfahren handelt es sich um ein veraltetes Verfahren, welches einige Nachteile hat. Zum einen können keine gefüllten Flächen gezeichnet werden und zum anderen sinkt die Bildwiederholrate mit steigender Bildkomplexität.

Rasterscan: Bei diesem Verfahren wird das Bild auf dem Bildschirm zeilenweise aufgebaut. Eine Zeile setzt sich dabei aus einzelnen Bildpunkten zusammen. Um das Bild aufzubauen, können zwei Techniken verwendet werden: interlaced und non-interlaced.

Frage 1.3: Was ist der Unterschied zwischen „interlaced“ und „non-interlaced“?

Wird ein Bild „interlaced“ aufgebaut, werden zwei Halbbilder gezeichnet, d.h. es werden erst alle gerade Zeilen gezeichnet und dann alle ungeraden. Benötigt ein Refresh-Zyklus dabei $\frac{1}{60}$ einer Sekunde benötigt ein kompletter Zyklus also $\frac{1}{30}$ einer Sekunde.

Der Zweck dieser Vorgehensweise besteht darin, dass im ganzen Bildschirmbereich neue Information mit einer Freuquenz von 60 Hz angezeigt werden. Dies funktioniert jedoch nur gut, wenn die Informationen in benachbarten Zeilen ähnlich sind.

Im Modus „non-interlaced“ werden die Zeilen einfach eine nach der anderen aufgebaut.

Frage 1.4: Wie kann mit Hilfe einer Kathodenstrahlröhre Farbe dargestellt werden?

Um z.B. auf einem Farbmonitor Farbe (siehe Kapitel 2) darzustellen, werden drei Elektronenröhren benutzt, anstatt einer einzelnen. Auf der Innenseite der Darstellungsfläche werden rote, grüne und blaue Phosphorpunkte zu Tripeln zusammengefasst, welche wiederum nebeneinander liegen. Jede der drei Elektronenröhren ist nun für die Anregung eines bestimmten Phosphors zuständig.

Damit jede Röhre auch wirklich nur den Phosphor treffen kann, für den sie zuständig ist, wird vor den Tripel eine sog. Lochmaske montiert. Das Auge nimmt das von einem Tripel abgestrahlte Licht als Mischung der drei Farben wahr.

Die Anordnung der Kathoden und der Bildpunkte, sowie das Aussehen der Maske kann dabei variiert werden:

Delta-Delta CRT: Kathoden und Phosphorpunkte sind jeweils dreiecksförmig angeordnet. Es wird eine normale Lochmaske verwendet.

PIL-Delta: Die Kathoden sind linear angeordnet und die Phosphorpunkte dreiecksförmig (Zeichnung im Skript anders, aber warum sollte es sonst so heißen?). Normale Lochmaske.

PIL mit Schlitzmasken-CRT: Die Kathoden sind linear angeordnet und der Phosphor in Streifen. Es wird eine Schlitzmaske verwendet, die den Vorteil besitzt, dass weniger Elektronen ausgeblendet werden.

Frage 1.5: Wie funktioniert ein Rasterscan-Terminal?

Eine wichtige Komponente eines Rasterscan-Terminal ist der Bildwiederholspeicher. In diesem Speicher (RAM), wird das Bild gespeichert, welches auf dem Monitor ausgegeben werden soll. Dazu wird für jeden Bildpunkt die **Intensitäts-** bzw. **Farbinformation** gespeichert. Diese Informationen können vom Rechner per **random access** manipuliert und beschrieben werden. Der Bildwiederholspeicher wird zyklisch ausgelesen, um das anzuzeigende Bild zu generieren.

Wie wird ein Bild nun zeilenweise auf dem Monitor ausgegeben bzw. wie wird der Monitor durch die Grafikkarte angesprochen? Eine Rasterscan-Terminal besitzt dafür zwei Adresszähler: den x-Adresszähler und den y-Adresszähler.

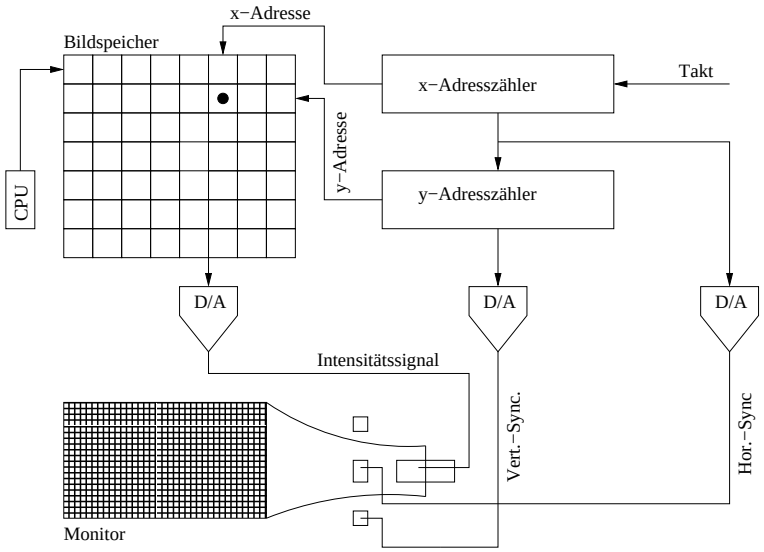


Abbildung 1: Funktionsweise eines Rasterscan-Terminals.

Weiter wird nun davon ausgegangen, dass wir ein Bild mit einer Auflösung von $n \times m$ darstellen möchten, d.h. das Bild ist n Pixel (**picture element**) breit und m Pixel hoch. Der x-Adresszähler wird auf 0 initialisiert und der y-Zähler auf $(m-1)$. Um die erste Zeile zu zeichnen zählt der x-Adresszähler nun von 0 bis $(n-1)$, während der y-Adresszähler auf $(m-1)$ verharrt. Dabei wird davon ausgegangen, dass ein karthesisches Koordinatensystem benutzt wird, d.h. dass der linke, untere Punkt des Bildschirms der Punkt $(0,0)$ ist. Ist der x-Adresszähler nun bei $(n-1)$ angelangt, wird der y-Adresszähler um eins dekrementiert und der x-Zähler springt wieder auf 0. Ist der y-Zähler schließlich bei 0 angelangt, springt er wieder auf $(m-1)$.

Frage 1.6: Wie funktioniert ein Plasmabildschirm?

Ein Plasmabildschirm besteht im Prinzip aus drei Komponenten. Die Hauptkomponente, die dem ganzen System auch den Namen gibt, ist eine Glasplatte mit plasmagefüllten Zellen. Wird an diese Zellen eine ausreichend hohe Spannung angelegt, so emmitieren sie Licht. Vor dieser Glasplatte befinden sich nun Elektroden, die in x-Richtung ausgerichtet sind, dahinter Elektroden, die in y-Richtung ausgerichtet sind.

Soll nun der Punkt (i,j) zum Leuchten gebracht werden. so wird an die i-te x-Elektrode eine Spannung von $-V$ Volt angelegt und an die j-te y-Elektrode eine Spannung von V Volt. Durch die Spannungsdifferenz am Kreuzungspunkt (i,j) wird das Plasma zum Leuchten angeregt. An allen anderen Punkten, an denen entweder nur die negative oder nur die positive Spannung anliegt, reicht die Spannungsdifferenz nicht aus, um das Plasma zum Leuchten zu bringen.

Frage 1.7: Welche Arten von LCD-Technologien kennst du?

LCD steht für **Liquid Crystal Display** und man kann zwischen zwei Techniken unterscheiden: **passiv** und **aktiv**. LCD-Displays emittieren selbst kein Licht, weshalb sie auch weniger Energie als andere Techniken benötigen.

Frage 1.8: Wie funktionieren die passiven LCD-Technologien?
Die passive LCD-Technologie benötigt vier verschiedene Komponenten, welche in sechs Schichten übereinander angeordnet sind:

Lichtpolarisationsfilter: Davon gibt es zwei: einen vertikalen und einen horizontalen Polarisationsfilter.

Glasplatten mit dünnen Drähten: Auch hiervon gibt es zwei. In einem Fall sind die Drähte horizontal ausgerichtet, im anderen vertikal.

Flüssigkristallschicht: Die Flüssigkristallschicht kann mit Hilfe eines elektrischen Feldes in zwei verschiedene Zustände gebracht werden. In einem wird die Polarisation von eintreffendem Licht um 90° gedreht, im anderen nicht.

Reflektor: Der Reflektor reflektiert eintreffendes Licht und ist ganz am Ende der Schichten angebracht.

Zuerst wird ein vertikaler Lichtpolarisationsfilter benutzt, um nur noch vertikal polarisiertes Licht durchzulassen. Dieses passiert dann in jedem Fall die beiden mit dünnen Drähten bestückten Glasplatten, welche die Flüssigkristallschicht zwischen sich einschliessen. Liegt aufgrund einer Spannungsdifferenz zwischen x- und y-Elektroden ein elektrisches Feld an der Flüssigkristallschicht an, so werden alle Kristalle in dieselbe Richtung ausgerichtet. Die Polarisationsrichtung des eintreffenden Lichts wird dadurch nicht geändert. Liegt keine Spannung an, wird die Polarisation des Lichtes um 90° gedreht. In diesem Fall kann es den horizontalen Polarisationsfilter passieren, wird reflektiert und kann die Schichten wieder komplett durchwandern. Dadurch entsteht also ein heller Punkt auf dem Bildschirm. Liegt eine Spannung an der Flüssigkristallschicht an, wird die Polarisation des Lichtes nicht geändert, wodurch es den horizontalen Polarisationsfilter nicht durchqueren kann. An der Stelle entsteht also ein schwarzer Punkt.

Frage 1.9: Wie funktionieren die aktiven LCD-Technologien?
Bei der aktiven LCD-Technologie wird an jeden Gitterpunkt ein (Thin Film-) Transistor angebracht. Dieser kann den Kristallzustand an dieser Stelle sehr schnell ändern, wobei der Grad der Zustandsveränderung ebenfalls steuerbar ist. Letzteres ermöglicht die Darstellung von Grautonbildern. Durch die Verwendung von rot-, grün und blaugefärbten Kristallen lassen sich farbige LCD-Displays realisieren.

Frage 1.10: Welche Möglichkeiten gibt es zur 3D-Darstellung?
Um ein Bild dreidimensional wirken zu lassen, gibt es mehrere Möglichkeiten:

Rot-Grün-Stereo: Hierfür wird lediglich eine Rot-Grün-Brille benötigt. Es werden getrennte Bilder für beide Augen berechnet, wobei das eine über den roten Farbkanal eines RGB-orientierten Darstellungsmedium geschickt wird, das andere über den grünen. Der blaue Kanal bleibt ungenutzt.

Polarisationsstereo: Auch bei diesem Verfahren werden Bilder für beide Augen berechnet. Diese werden abwechseln, mit einer möglichst hohen Frequenz,

dargestellt. Mit Hilfe eines Polarisationsfilters wird dafür gesorgt, dass jedes Bild abwechselnd auf jeweils nur einem Auge sichtbar ist.

Getrennte Displays: Wieder werden für das linke und das rechte Auge getrennt Bilder berechnet. Diese beiden Bilder können nun entweder mit Hilfe von Minibildschirmen direkt an die Augen herangeführt werden oder mit Hilfe eines Spiegelstereoskop betrachtet werden.

Weitere Möglichkeiten sind **3D LCD-Displays**, ein **multifokaler Spiegel** oder eine **TI-Spirale**.

1.2 Hardcopy-Geräte

Frage 1.11: Wofür werden Hardcopy-Geräte benutzt?
Während Sichtgeräte dazu benutzt werden, einen aktuellen Zustand darzustellen (wobei sich dieser schnell ändern kann), werden Hardcopy-Geräte zur dauerhaften Wiedergabe benutzt. Dazu kann zum Beispiel Papier, Film oder Magnetband verwendet werden.

Frage 1.12: Wie funktioniert ein Stiftplotter und welche Arten von Stiftplottern kennst du?
Ein Stiftplotter zeichnet mit Hilfe eines Stiftes auf Papier. Die Stifte sind oft aus einem Magazin auswechselbar, so dass unterschiedliche Farben und Strichdicken möglich sind. Der Stift kann mit Hilfe eines Servomotors in x- und in y-Richtung bewegt werden. Wie dies realisiert wird, hängt von der Art des Stiftplotters ab. Man kann zwischen **Tisch-** und **Trommelplottern** unterscheiden.

Ein Stiftplotter besitzt Vor- und Nachteile. Zu den Vorteilen zählt, dass eine hohe Auflösung erreicht werden kann und er daher z.B. für technische Zeichnungen sehr geeignet ist. Außerdem entfällt der Schritt der Aufrasterung. Hierin liegt jedoch zugleich der Nachteil. Rastergrafiken können mit einem Stiftplotter nämlich nur sehr schwer ausgegeben werden. Desweiteren sind die Geräte meist sehr teuer.

Frage 1.13: Wie funktioniert ein Tischplotter? Wie ein Trommelplotter?
Bei einem Tischplotter ist das Papier fest auf einem Tisch eingespannt und der Stift ist in x- und y-Richtung frei beweglich. Bei einem Trommelplotter ist der Stift nur in der y-Richtung frei beweglich und die x-Richtung wird realisiert, indem das Papier mit Hilfe einer Trommel unter dem Stift in x-Richtung hin- und herbewegt wird.

Frage 1.14: Wie funktioniert ein Matrixdrucker?
Bei einem Matrixdrucker besteht der Schreibkopf aus 9 bis 24 individuell ansteuerbaren Nadeln. Dieser wird horizontal über das Papier geführt, welches über eine Trommel vor und zurück bewegt werden kann. Zwischen den Nadeln und dem Papier befindet sich ein Farbband. Indem eine Nadel das Farbband auf das Papier drückt, kann so ein entsprechend farbiger Punkt gedruckt werden. Dabei unterscheidet man zwischen zwei Techniken, um die Nadeln auf das Papier zu drücken:

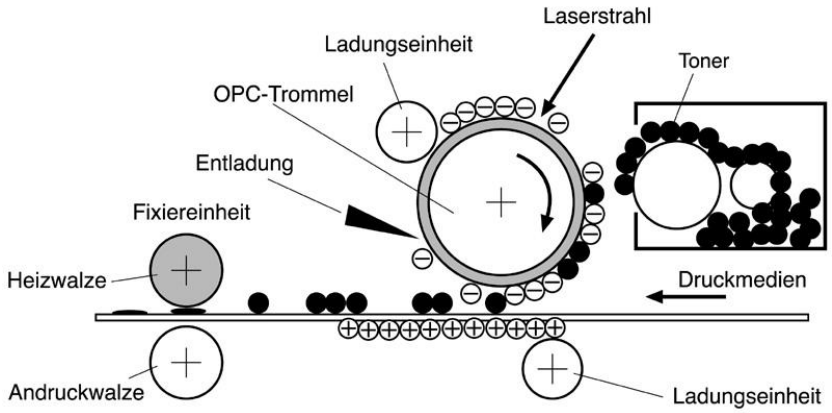


Abbildung 2: Funktionsweise eines Laserdruckers.

aktiv: Die Nadeln werden mit Elektromagneten auf das Papier geschossen und mit Hilfe von Federn zurück geholt.

passiv: Die Nadeln werden mit Hilfe von Federn auf das Papier gedrückt und mit Hilfe von Elektromotoren zurück geholt.

Matrixdrucker bieten eine Auflösung von bis zu maximal 360 dpi. Zu den Vorteilen von Matrixdruckern zählt, dass sie billig in der Anschaffung und Betrieb sind und dass Durchschläge möglich sind. Jedoch sind sie auch relativ laut und für Farbwiedergabe nur bedingt geeignet.

Frage 1.15: Wie funktionieren elektrostatische Drucker?
Bei einem elektrostatischen Drucker wird das Papier mit Hilfe einer Elektrode an den Stellen negativ geladen, die später schwarz sein sollen. Anschließend wird schwarzer Toner auf das Papier aufgetragen, der an den negativ aufgeladenen Stellen haften bleibt. Mit Hilfe von Druck und Hitze wird der Toner abschließend fixiert.

Farbdruck ist ebenfalls möglich. Dabei wird das CMY-Farbmodell (siehe Kapitel 2) benutzt und die Farbe in einem Mehrpassverfahren aufgetragen. Die typische Auflösung eines solchen Systems liegt zwischen 100 und 400 dpi.

Frage 1.16: Wie funktioniert ein Laserdrucker?
Ein Laserdrucker arbeitet, ähnlich wie ein elektrostatischer Drucker, mit Hilfe von elektrischer Ladung. In Abbildung 2 ist die Funktionsweise eines Laserdruckers dargestellt. Zunächst werden auf eine positiv geladene Trommel negative Ladungen aufgebracht. Die Stellen, die später schwarz sein sollen, werden mit Hilfe eines Laserstrahls neutralisiert. Anschließend wird die Trommel am Toner vorbeigeführt, der nur an den Stellen haften bleibt, die zuvor neutralisiert wurden. Das Papier, auf dem gedruckt werden soll, wird ebenfalls elektrisch aufgeladen und zieht wiederum die Tonerpartikel an. Abschließend wird der Toner mit Hilfe von Druck und Hitze fixiert.

Da die Laserdiode fixiert angebracht ist und sich daher nicht bewegen kann, muss der Laser mit Hilfe eines meist sechs- oder achteckigen Drehspiegels um-

gelenkt werden. Dieser Spiegel dreht sich sehr schnell und sorgt dafür, dass sich der Laser jeweils über die Trommel bewegt.

Eine Farbwiedergabe ist ebenfalls möglich, indem für jede Grundfarbe eine Druckstation verwendet wird. Die typische Auflösung eines Laserdruckers liegt bei 300–600 dpi. Ein Vorteil von Laserdruckern ist, dass sie sehr leise sind. Jedoch sind sie auch sehr teuer und produzieren Ozon.

Frage 1.17: Wie funktioniert ein Tintenstrahldrucker?

Der Schreibkopf eines Tintenstrahldruckers besteht aus bis zu 30 Tintendüsen und wird horizontal über das Papier geführt. Die Tintendüsen können feine Tintenpartikel auf das Papier sprühen, wobei es zwei Ansteuerungsmöglichkeiten gibt:

Einzelimpulse: Hierbei schießt die Tintendüse einen Tintenstrahl nur dann, wenn auch einer gebraucht wird.

Kontinuierlicher Tintenfluss: Die Düse gibt kontinuierlich Tinte ab, wobei mit Hilfe eines elektrischen Feldes gesteuert wird, ob sie das Papier trifft oder aufgefangen wird.

Tintenstrahldrucker erreichen eine Auflösung von bis zu 300 dpi und bieten eine gute Farbwiedergabe. Zudem sind sie relativ leise. Es sind jedoch keine Durchschläge möglich und für eine gute Ausgabe ist Spezialpapier nötig. Allgemein sind die Betriebskosten für Tintenstrahldrucker relativ hoch.

Frage 1.18: Wie funktioniert ein Thermotransferdrucker? Wie ein Farbsublimationsdrucker?

Bei einem Thermotransferdrucker überträgt ein Schreibkopf, der aus eng benachbarten Heizstäbchen besteht, Pigmente von einem mit Wachs beschichteten Farbband auf das Druckpapier. Farbdarstellung ist durch die Verwendung von Farbbändern unterschiedlicher Farbe möglich.

Ein Farbsublimationsdrucker arbeitet ähnlich wie ein Thermotransferdrucker, nur ist hier die Farbintensität eines Punktes steuerbar. Üblicherweise kann zwischen 256 verschiedenen Stufen unterschieden werden. Die Farbe geht bei diesem Verfahren durch Erhitzung direkt vom festen in den gasförmigen Zustand über, wobei sie sich mit den anderen Grundfarben mischt und von einem Spezialpapier aufgenommen wird. Die Farbanteile lassen sich durch individuelle Regelung der Heizelemente steuern.

Beide Arten von Druckern bieten eine Auflösung von bis zu 300 dpi, wobei ein Farbsublimationsdrucker schon eine fotoähnliche Wiedergabequalität besitzt. Nachteilig wirkt sich aus, dass Spezialpapier mit Thermofarbbändern benötigt wird und dass die Druckgeschwindigkeit langsam ist.

Frage 1.19: Wie können Bilder auf Film festgehalten werden?

Um Bilder auf Fotos bzw. auf Video festzuhalten gibt es mehrere Möglichkeiten. Die einfachste ist die **Direktfotografie vom Bildschirm** in einem abgedunkelten Raum mit einer Spiegelreflexkamera, deren Belichtungszeit zwischen einer halben und einer Sekunde liegt. Diese Methode ist sehr kostengünstig, jedoch kann es, da das Bild direkt von einem normalen Monitor abfotografiert wird, zu Verzerrungen kommen.

Alternativ dazu kann ein **Filmbelichter** benutzt werden. Dabei werden drei Farbauszüge (Rot, Grün und Blau) getrennt wiedergegeben und aufgenommen. Es wird ein flacher, monochromer Bildschirm verwendet, vor den ein umschaltbarer Farbfilter angebracht ist. Bei dieser Vorgehensweise sind hohe Auflösungen möglich, da bei einem Monochrombildschirm keine Lochmaske benötigt wird und desweiteren keine Zeitbedingungen einzuhalten sind.

2 Farbmodelle

2.1 Grundlagen

Frage 2.1: Warum müssen Farben standardisiert werden?
Farben müssen standardisiert werden, da Farbwahrnehmung eine sehr subjektive Angelegenheit ist. Zudem gibt es, auch aufgrund der alltäglichen eigenen Erfahrung mit Farbe, unterschiedliche Begriffe für verschiedene Empfindungen, die mit der Farbwahrnehmung zusammenhängen. Im alltäglichen Gebrauch wird zwischen Farbe, Sättigung und Helligkeit unterschieden. Der Begriff der Farbe unterscheidet zwischen Farben wie rot, blau, lila. Die Sättigung einer Farbe gibt an, wie weit die empfundene Farbe von einem Grauwert gleicher Intensität entfernt ist. Pures Rot ist stark gesättigt, während z.B. rosa relativ ungesättigt ist. Mit der Helligkeit wird die wahrgenommene Intensität eines reflektierenden Lichts beschrieben, also quasi wieviel Energie das Licht enthält.

Da die subjektive Beschreibung von Farbe jedoch zu ungenau sind, werden objektive, messbare Möglichkeiten zur Beschreibung von Farbe benötigt. Das Teilgebiet der Physik, welches sich genau damit beschäftigt, heißt Kolorimetrie. In der Kolorimetrie wird eine Farbe über das Tripel **dominante Wellenlänge**, **Reinheit** und **Luminanz** beschrieben. Diese drei Begriffe haben ihre Entsprechungen zu den Begriffen der subjektiven Wahrnehmung, welche in Tabelle 1 aufgelistet sind.

Wahrnehmung	Kolorimetrie
Farbe	Dominante Wellenlänge
Sättigung	Reinheit
Helligkeit	Luminanz

Tabelle 1: Zusammenhang zwischen den Begriffen der Wahrnehmung und Kolorimetrie

Die dominante Wellenlänge beschreibt die wahrgenommene Farbe. Ein reflektiertes Licht besteht in Wirklichkeit aus einem ganzen Spektrum von Wellen bestimmter Wellenlängen und Energien. Dies kann durch eine Funktion $P(\lambda)$ beschrieben werden, wobei λ die Wellenlänge der Welle ist und $P(\lambda)$ die Energie mit der diese Welle im Spektrum vorkommt. Eine solche spektrale Energieverteilung ist in Abbildung 3 dargestellt. Die dominante Wellenlänge ist anscheinend die Frequenz, die aus diesem Spektrum am meisten heraussticht. Es ist nun so, dass verschiedene spektrale Energieverteilungen den gleichen Farbeindruck hervorrufen können und dementsprechend auch durch das gleiche Tripel beschrieben werden können.

Frage 2.2: Was besagt die Tristimulustheorie?
Die Tristimuslustheorie besagt, dass es im Auge drei unterschiedliche Arten von Zellen gibt, welche auf Licht unterschiedlicher Wellenlänge reagieren. Diese Zellen, Zäpfchen genannt, reagieren dabei auf die Farben Rot, Grün und Blau, wobei sie für Grün am empfindlichsten und für Blau am wenigsten empfindlich sind. Trifft also Licht ins Auge, dann entsteht der Farbeindruck dadurch, dass diese Zellen unterschiedlich angeregt werden. Licht besteht, wie in Abbildung

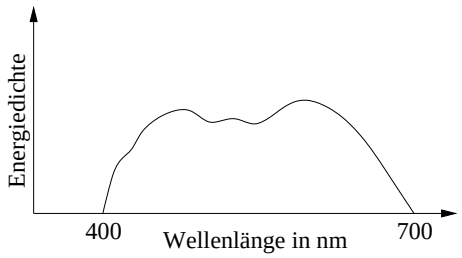


Abbildung 3: Spektrale Energieverteilung $P(\lambda)$ eines bestimmten Lichts

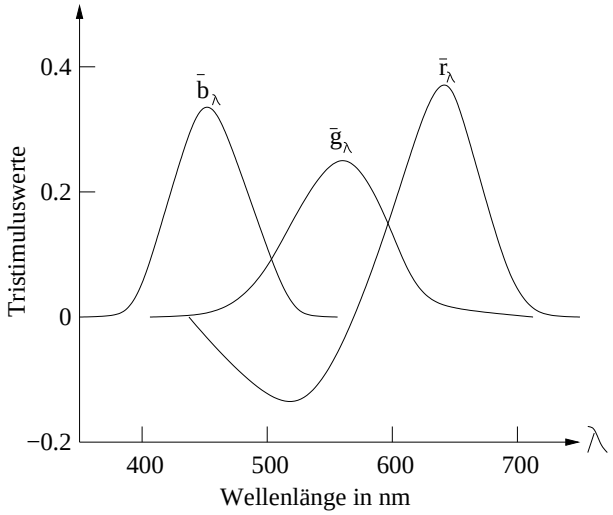


Abbildung 4: Die Hilfsfunktionen $\overline{r}(\lambda)$, $\overline{g}(\lambda)$ und $\overline{b}(\lambda)$

3 dargestellt, aus einem Spektrum von Wellen unterschiedlicher Wellenlängen, welche in unterschiedlichen Intensitäten auftreten. Es können nun drei Hilfsfunktionen $\overline{r}(\lambda)$, $\overline{g}(\lambda)$ und $\overline{b}(\lambda)$ angegeben werden, welche bestimmen, in welchen Verhältnissen die Primärfarben Rot, Grün und Blau gemischt werden müssen, damit im Auge der gewünschte Farbeindruck entsteht. Die Gewichtung r der Farbe rot ergibt sich zu einem gegebenen Energiespektrum $P(\lambda)$ z.B. als:

$$r = k \cdot \int \overline{r}(\lambda) \cdot P(\lambda)\ d\lambda$$

Entsprechend werden die Gewichte für Grün und Blau bestimmt.

2.2 Das CIE-Farbmodell

Frage 2.3: Was ist das CIE-Farbmodell?

Aufgrund der Tristimulustheorie erscheint es sinnvoll, Farben als Mischung von drei Primärfarben zu definieren. Allerdings sind die negativen Gewichte (siehe Abbildung 4), die bei der Verwendung der Primärfarben Rot, Grün und Blau entstehen, sehr ungünstig. Aus diesem Grund hat die *Commision Internationale de l'Éclairage* (CIE) 1931 die Primärfarben **X**, **Y** und **Z** definiert, welche

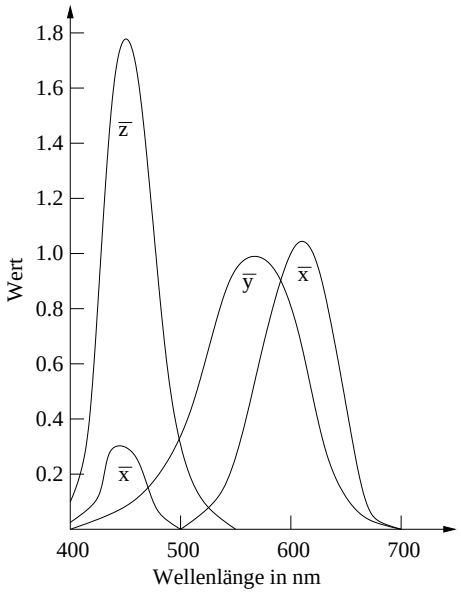


Abbildung 5: Die Spektralwertkurven $\overline{x}(\lambda)$, $\overline{y}(\lambda)$ und $\overline{z}(\lambda)$ für die Primärfarben **X**, **Y** und **Z** nach CIE 1931

ohne negative Gewichte auskommen. Dabei entspricht die Farbfunktion $\overline{y}_\lambda$ der Primärfarbe **Y** der Luminanzempfindlichkeit des menschlichen Auges.

Ist nun eine spektrale Energieverteilung $P(\lambda)$ gegeben, so errechnen sich die Gewichtungen X, Y und Z der Primärfarben **X**, **Y** und **Z** wie folgt:

$$X = k \int P(\lambda)\overline{x}(\lambda)\ d\lambda \quad Y = k \int P(\lambda)\overline{y}(\lambda)\ d\lambda \quad Z = k \int P(\lambda)\overline{z}(\lambda)\ d\lambda$$

Wurde also für eine Farbe **C** die Gewichtungen X, Y und Z ermittelt, so ergibt sich die Farbe als $\mathbf{C} = X \cdot \mathbf{X} + Y \cdot \mathbf{Y} + Z \cdot \mathbf{Z}$. Werden nun die Werte für alle sichtbaren Farben ermittelt, so ergibt sich der Kegel in Abbildung 6. Werden die Werte X, Y und Z nun bezüglich $X+Y+Z$ normalisiert, so entstehen Farbwerte, welche nur von der dominanten Wellenlänge und der Sättigung abhängen, nicht jedoch von der Lichtenergie bzw. Luminanz. Es ergeben sich dadurch x, y und z:

$$x = \frac{X}{X+Y+Z} \quad y = \frac{Y}{X+Y+Z} \quad z = \frac{Z}{X+Y+Z}$$

Durch die Normalisierung gilt nun $x + y + z = 1$ und die Tripel der x, y und z liegen alle in der Ebene $X + Y + Z = 1$. Mit diesem Wissen wird z quasi redundant, da es sich für gegebene x- und y-Werte als $z = 1 - x - y$ ergibt. Wird für jeden Punkt der Ebene $z = 0$ gesetzt, entspricht dies der Projektion der Ebene in die x-y-Ebene und es entsteht das in Abbildung 7 dargestellte Chromatizitätsdiagramm. Da es sich dabei allerdings um eine Projektion handelt, reichen die x- und y-Werte allein nicht aus, um eine Farbe zu spezifizieren. Es fehlt eine dritte Komponente, welche meist in Form der Luminanzinformation

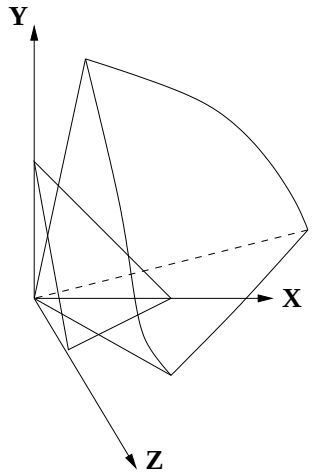


Abbildung 6: Der Kegel der sichtbaren Farben im CIE-Farbraum

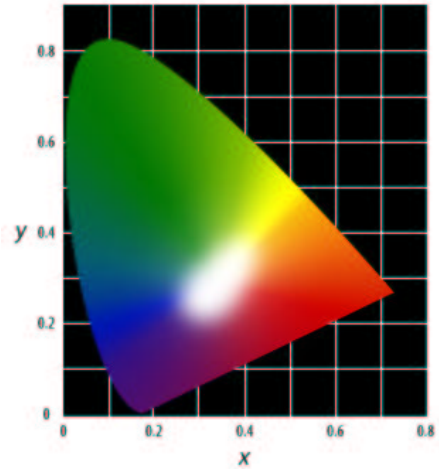


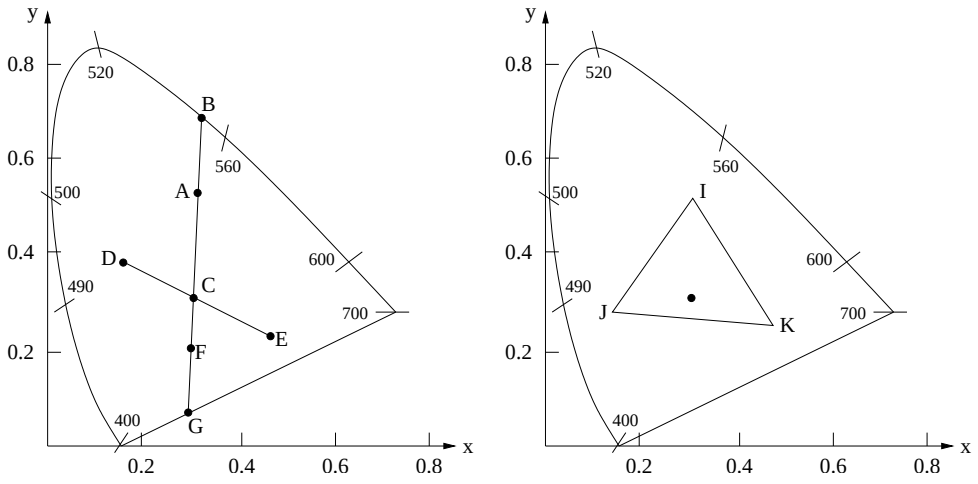
Abbildung 7: Chromatizitätsdiagramm (Projektion in x-y-Ebene)

Y gegeben wird. Dann lässt sich X, Y und Z wie folgt zurückgewinnen:

$$X = \frac{x}{y}Y \quad Y = Y \quad Z = \frac{1-x-y}{y}Y$$

Die Korrektheit dieser Formeln lässt sich leicht durch Einsetzen der Definitionen von x, y und z überprüfen.

Die sichtbaren Farben liegen im Inneren und auf dem Rand des Diagramms, wobei auf dem Rand die zu 100% reinen Farben liegen. Abbildung 8(a) zeigt dies schematisch. Der Punkt C entspricht dabei einem standardisierten weißen Licht, welches das Sonnenlicht approximiert. Für einen Punkt A im Diagramm ist seine dominante Wellenlänge B gegeben, als der Schnitt des Rands mit der Geraden, welche durch A und C verläuft. Farben, welche sich auf dieser Geraden bezüglich C gegenüber liegen, sind Komplementärfarben, d.h. sie mischen sich zu Weiß. In der Abbildung sind dies die Farben A und F oder auch D und E. Die Reinheit einer Farbe lässt sich ebenfalls aus dem Diagramm ablesen. Für



(a) Farben im chromatischen Diagramm

(b) Mischen von Farben

Abbildung 8: Verwendungen des Chromatizitätsdiagramms

die Farbe A ist ihre Reinheit gegeben als der Quotient der Längen von AC und BC.

Frage 2.4: Was ist ein Gamut?

Eine weitere Verwendungsmöglichkeit des Chromatizitätsdiagramm ist die Definition eines Farbgamuts bzw. einer Farbpalette. Dies ist in Abbildung 8(b) dargestellt. Sind zum Beispiel die Farben I und J gegeben, so lassen sich alle Farben auf der Strecke zwischen I und J erzeugen, indem die beiden Farben in entsprechenden Anteilen gemischt werden. Wird noch ein dritter Punkt K hinzugenommen, so ergibt sich ein Dreieck, in dem alle mischbaren Farben liegen. Für einen bestimmten RGB-Monitor können nun z.B. die Punkte für den roten, grünen und blauen Phosphor eingezeichnet werden. Das durch diese drei Punkte aufgespannte Dreieck enthält dann alle durch den Monitor darstellbaren Farben. Dies erklärt auch, warum es nicht möglich ist, *alle* sichtbaren Farben durch eine Mischung von drei Farben zu erzeugen. Es gibt kein Dreieck, welches vollständig im Chromatizitätsdiagramm liegt und dies komplett abdeckt.

2.3 Das RGB-Farbmodell

Frage 2.5: Was ist das RGB-Farbmodell?

Das RGB-Farbmodell ist ein **additives** Farbmodell, welches insbesondere bei Farbbildschirmen Verwendung findet. Additiv bedeutet dabei, dass die Primärfarben Rot, Grün und Blau in bestimmten Verhältnissen zu Schwarz hinzuaddiert werden, um die Farben des Farbraums zu erzeugen. Es basiert auf einem kartesischen Koordinatensystem und wird, wie in Abbildung 9 gezeigt, in Form eines Einheitswürfels dargestellt. Im Ursprung des RGB-Würfels liegt Schwarz und entlang jeder Koordinatenachse erstreckt sich eine der Primärfarben Rot,

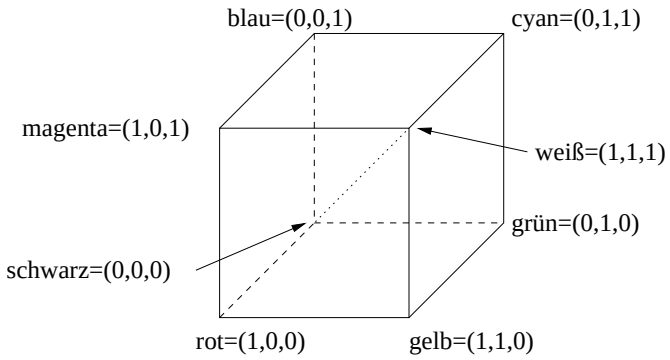


Abbildung 9: Der RGB-Würfel

Grün und Blau. An der Position $(1, 1, 1)$ findet sich Weiß, welches sich als Mischung aller Primärfarben ergibt. Entlang der Geraden zwischen Schwarz und Weiß befinden sich die Grautöne.

2.4 Das CMY-Farbmodell

Frage 2.6: Was ist das CMY-Farbmodell?

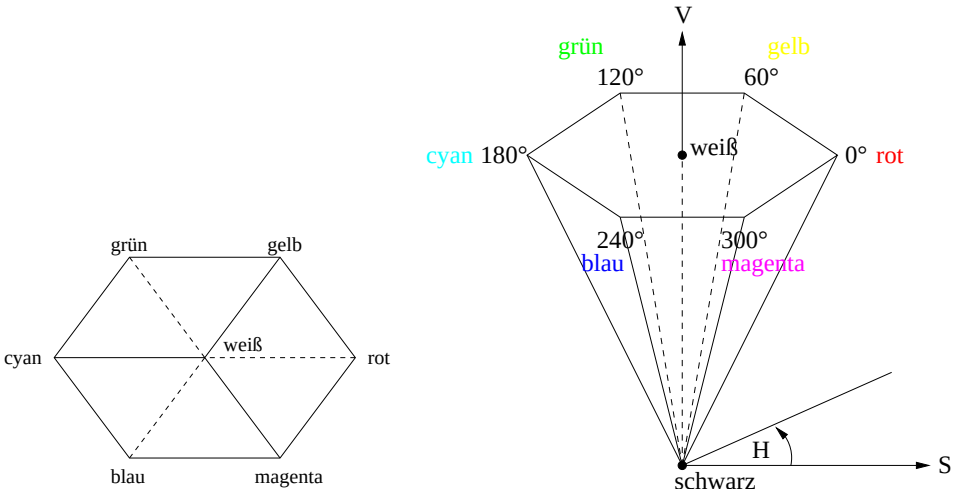
Das CMY-Farbmodell ist ein **subtraktives** Farbmodell und findet überall dort Verwendung, wo das Licht nicht direkt in das Auge des Betrachters gelangt, wie es z.B. bei einem Farbmonitor der Fall ist, sondern erst reflektiert wird. Dies ist zum Beispiel bei Druckern bzw. allgemein bei Druckverfahren der Fall. Die Primärfarben im CMY-Modell sind die Farben Cyan, Magenta und Gelb (Yellow). Jede kann als ein Filter für eine der Primärfarben des RGB-Modells aufgefasst werden. Cyan filtert die Farbe Rot, Magenta filtert Grün und Gelb filtert Blau. Dies wird auch aus Abbildung 9 ersichtlich. Denn Cyan ergibt sich z.B. aus Grün und Blau. Oder anders interpretiert: Cyan ist ein Weiß, dem der Rotanteil vollständig fehlt. Grün würde also erzeugt werden, indem Cyan und Gelb addiert werden, denn in diesem Fall würden dem Weiß die Farben Rot und Blau entzogen. Übrig bleibt Grün.

Das CMY-Modell wird, genau wie das RGB-Modell mit einem Einheitswürfel dargestellt. Nur dass in diesem Fall Weiß im Ursprung liegt und Schwarz sich als Mischung aller Primärfarben ergibt. Die Umrechnung zwischen den Modellen ist recht simpel:

$$\begin{pmatrix} C \\ M \\ Y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} R \\ G \\ B \end{pmatrix} \text{ bzw. } \begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} C \\ M \\ Y \end{pmatrix}$$

Eine Erweiterung des CMY-Modells ist das CMYK-Modell, welches als vierte Farbe zusätzlich schwarz verwendet. Dieses Modell wird bei Druckverfahren häufig benutzt, damit Schwarz nicht aus den drei anderen Primärfarben gemischt werden muss. Ist eine Farbe in Form ihrer CMY-Werte gegeben, ergeben sich die CMYK-Werte wie folgt:

$$K = \min(C, M, Y),\ C = C - K,\ M = M - K,\ Y = Y - K$$



(a) Projektion des RGB-Würfels

(b) Die Pyramide des HSV-Farbmodell

Abbildung 10: Das HSV-Farbmodell

Das Wert von K kann im CMY-Würfel so interpretiert werden, dass, um eine Farbe im Würfel zu erreichen, erst entlang der Diagonalen zwischen Weiß und Schwarz abgekürzt wird und der restliche Weg mit den anderen Farben „abgelaufen“ wird.

Frage 2.7: Was ist das HSV-Farbmodell?

Die bisher vorgestellten Farbmodelle waren eher technisch orientiert und für Menschen nicht unbedingt sehr intuitiv. Das HSV-Modell (Hue, Saturation, Value) benutzt die von Malern benutzte, relativ intuitive Mischungsmethode der „Tints, Shades und Tones“. Dabei wird davon ausgegangen, dass eine reine Farbe gegeben ist. Wird dieser nun Weiß hinzugefügt, entsteht ein *Tint*, d.h. eine weniger gesättigte Farbe. Das hinzufügen von Schwarz zu einer reinen Farbe bewirkt, dass ein *Shade* entsteht. Die Farbe wird abgedunkelt und erscheint weniger hell. Durch Hinzufügen von Schwarz und Weiß entsteht schließlich ein *Tone*.

Der Farbraum des HSV-Modells entsteht durch die Projektion des RGB-Würfels entlang der Achse, die durch Weiß und Schwarz verläuft. Dadurch entsteht ein Sechseck, welches in Abbildung 10(a) dargestellt ist. Der Farbe rot wird ein Winkel von 0° zugeordnet. Abbildung 10(b) zeigt, wie der Farbraum aufgebaut ist. Über den Winkel H wird die Farbe festgelegt. Die Sättigung S gibt, wie der Name schon sagt, die Sättigung der Farbe an und über V wird die Helligkeit eingestellt. Die S-Komponente spielt in diesem Modell also die Rolle der Tints und die V-Komponente die der Shades. Die reinen Farben liegen auf dem Rand des Sechsecks.

Frage 2.8: Wie sollten Intensitätsstufen ausgewählt werden?

Viele Geräte erlauben es, unterschiedliche Intensitäten darzustellen, um so z.B. Graustufen zu erzeugen. Im folgenden werden diesen unterschiedlichen Inten-

sitätstufen nun Werte zwischen 0 und 1 zugeordnet, wobei 0 die kleinstmögliche darstellbare Intensität sei und 1 die größtmögliche.

Soll nun eine bestimmte Anzahl von Intensitätsstufen angezeigt werden können, so muss überlegt werden, wie diese über das Intervall der verschiedenen Intensitäten verteilt werden. Der einfachste Ansatz wäre wahrscheinlich, die verschiedenen Intensitäten gleichmäßig über das Intervall zu verteilen. Sollen z.B. 256 verschiedene Stufen möglich sein, so würden diese in der Form $(0 = \frac{0}{255}, \frac{1}{255}, \ldots, \frac{255}{255} = 1)$ verteilt werden. Allerdings würde sich dieser Ansatz nicht mit einer wichtigen Eigenschaft des menschlichen Auges vertragen: Es nimmt keine absoluten Intensitäten wahr, sondern Quotienten von Intensitätsstufen. So wird zum Beispiel der Sprung von 0.10 nach 0.11 als gleich stark empfunden wie ein Sprung von 0.50 nach 0.55. Aus diesem Grund müssen die verschiedenen Intensitätsstufen logarithmisch anstatt linear angeordnet werden, um gleichmäßige Helligkeitsstufen zu erhalten. Dies wird nun an einem Beispiel verdeutlicht, bei dem davon ausgegangen wird, dass 256 Intensitätsstufen erzeugt werden sollen, wobei jede Intensität dem r-fachen der Vorgängerintensität entspricht. Es gilt also:

$$I_0 = I_0,\ I_1 = r \cdot I_0,\ I_2 = r \cdot I_1 = r^2 \cdot I_0,\ I_3 = r^3 \cdot I_0,\ \ldots,\ I_{255} = r^{255} \cdot I_0 = 1$$

Letztere Formel wird nun nach r aufgelöst und das Ergebnis in die Formel für I_j eingesetzt:

$$r = \left(\frac{1}{I_0}\right)^{\frac{1}{255}},\ I_j = r^j \cdot I_0 = \left(\frac{1}{I_0}\right)^{\frac{j}{255}} \cdot I_0 = I_0^{-\frac{j}{255}} \cdot I_0^{\frac{255}{255}} = I_0^{\frac{255-j}{255}} \text{ für } 0 \leq j \leq 255$$

Allgemein gilt also für $n+1$ Intensitätsstufen:

$$r = \left(\frac{1}{I_0}\right)^{\frac{1}{n}},\ I_j = I_0^{\frac{n-j}{n}} \text{ für } 0 \leq j \leq n$$

Das Verhältnis von maximaler Intensität $I_n = 1$ zu minimaler Intensität I_0 wird Dynamik genannt. Beträgt die minimale Intensität eines CRT-Bildschirm zum Beispiel ein Fünfzigstel der maximalen Intensität, so beträgt die Dynamik $\frac{1}{0.02} = 50$. Von der Dynamik des Ausgabemediums hängt auch ab, wieviele Intensitätsstufen benötigt werden, damit Farbübergänge als fließend empfunden werden. Dies ist der Fall, wenn r kleiner oder gleich dem Wert 1.01 ist. Um also herauszufinden, wieviele Intensitätsstufen nötig sind, muss r in der Formel von etwas weiter oben auf den Wert 1.01 gesetzt werden und die Formel nach n aufgelöst werden. Es ergibt sich:

$$1.01 = \left(\frac{1}{I_0}\right)^{\frac{1}{n}} \Leftrightarrow 1.01^n = \frac{1}{I_0} \Leftrightarrow n = \log_{1.01}\left(\frac{1}{I_0}\right)$$

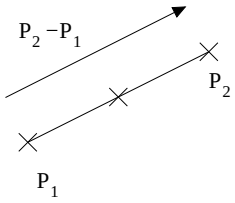


Abbildung 11: Darstellung einer Strecke im $\mathbb{R}^2$

3 Darstellung von Kurven

Zuerst werden nun einige grundlegende geometrische Objekte eingeführt, die später teilweise verwendet werden.

3.1 Grundlagen

Frage 3.1: Wie ist ein Punkt bzw. Markierer definiert?
Ein zwei-dimensionaler Punkt p ist wie folgt definiert:

$$p = \begin{pmatrix} p_x \\ p_y \end{pmatrix}, p_x, p_y \in \mathbb{R}$$

Frage 3.2: Wie ist eine Strecke definiert?
Eine Strecke wie in Abbildung 11 im $\mathbb{R}^d$ ist wie folgt definiert: Gegeben seien zwei Punkte $p_1, p_2 \in \mathbb{R}^d$.

$$s[p_1, p_2] = \{p : p = p_1 + \lambda \cdot (p_2 - p_1), \lambda \in [0, 1]\}$$

Frage 3.3: Welche Möglichkeiten kennst du Kreise darzustellen?
Zwei Möglichkeiten Kreise darzustellen sind die **Mittelpunkt-Radius-Darstellung** und die **3-Punkt-Darstellung**.

Bei der Mittelpunkt-Radius-Darstellung ist der Mittelpunkt $m \in \mathbb{R}^2$ und der Radius $r \in \mathbb{R}$ gegeben. Der Kreis ergibt sich dann als:

$$k[m, r] = \{p : (p_x - m_x)^2 + (p_y - m_y)^2 = r^2\}$$

Im Prinzip ist dies nichts andere als der Pythagoras, was man sich anhand von Abbildung 12 klarmachen kann. Bei der 3-Punkt-Darstellung sind, wie der Name schon sagt, drei Punkte gegeben, anhand derer man den Kreis vollständig rekonstruieren kann. KOMMT NOCH!

Frage 3.4: Was ist ein Polygonzug?
Gegeben seien m Punkte $p_1, p_2, \ldots, p_m$. Dann ist ein **offener Polygonzug** $p[p_1, p_2, \ldots, p_m]$ definiert als die Folge von Strecken $s[p_i, p_{i+1}]$ mit $i = 1, \ldots, m-1$. Handelt es sich um einen **geschlossenen Polygonzug**, dann kommt zusätzlich noch die Strecke $s[p_m, p_1]$ hinzu.

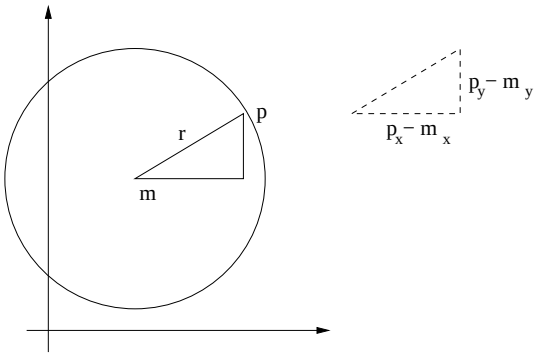


Abbildung 12: Kreis in Mittelpunkt-Radius-Darstellung

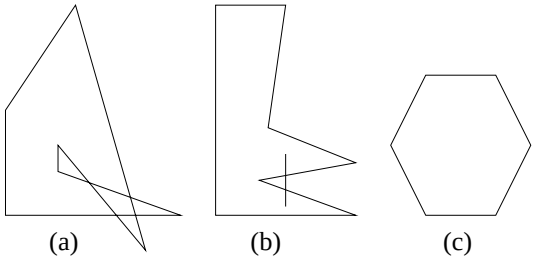


Abbildung 13: Verschiedene Polygone

Frage 3.5: Was ist ein Polygon? Was ist ein einfache Polygon? Was ein konvexes?

Gegeben sei ein geschlossener Polygonzug $p[p_1, p_2, \ldots, p_m]$. Dann ist das dazugehörige Polygon die Menge aller Punkte, für die ein dort ausgehender Strahl, der durch keinen der Punkte p_i geht, den Polygonzug in einer ungeraden Anzahl von Punkten schneidet.

Ein einfaches Polygon ist ein Polygon bei dem sich der Polygonzug nicht selbst schneidet. Ein konvexes Polygon ist ein einfaches Polygon, bei dem zu je zwei seiner Punkte auch die auf der Verbindungsstrecke liegenden Punkte im Polygon liegen. Abbildung 13 zeigt unter (a) ein Beispiel für ein nichteinfaches, nichtkonvexes Polygon, unter (b) ein einfaches, jedoch nichtkonvexes Polygon und unter (c) ein konvexes und damit auch einfaches Polygon.

Frage 3.6: Wie funktioniert die Parameterdarstellung einer Kurve im $\mathbb{R}^d$?

Bei dieser Darstellung ergeben sich die d Koordinaten eines Punkts der Kurve als Funktionen über einen eindimensionalen Parameterbereich $P \subseteq \mathbb{R}$:

$$p = k(t) = \begin{pmatrix} f_1(t) \\ f_2(t) \\ \vdots \\ f_d(t) \end{pmatrix}, k : P \to \mathbb{R}^d, t \in D \subseteq \mathbb{R}$$

So lässt sich z.B. ein Kreis in der x-y-Ebene mit dem Radius r um den Mittel-

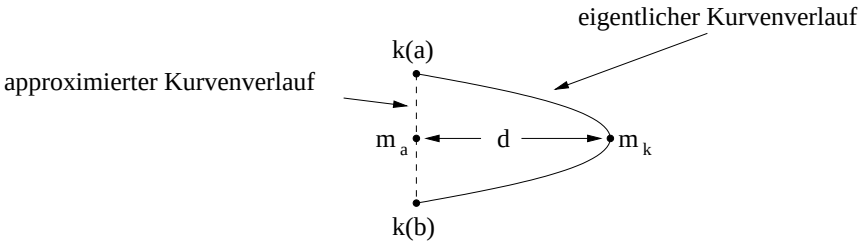


Abbildung 14: Approximation einer Kurve in Parameterdarstellung

punkt $m = (m_x, m_y)$ wie folgt darstellen:

$$p = k(t) = \begin{pmatrix} f_1(t) = m_x + r \cdot \cos t \\ f_2(t) = m_y + r \cdot \sin t \end{pmatrix},\ t \in [0, 2\pi]$$

Ähnlich lässt sich eine Schraubenlinie im Raum darstellen:

$$p = k(t) = \begin{pmatrix} f_1(t) = \sin t \\ f_2(t) = \cos t \\ f_3(t) = t \end{pmatrix},\ t \in \mathbb{R}$$

Frage 3.7: Wie lässt sich eine Kurve in Parameterdarstellung durch einen Polygonzug approximieren?
Hier kann grob zwischen zwei Verfahren unterschieden werden:

Äquidistante Unterteilung: Das Parameterintervall P wird äquidistant unterteilt und die Kurvenpunkte an diesen Unterteilungen als Eckpunkte des approximierenden Polygonzugs verwendet.

Unterteilung mit Schranke: Die Kurve wird durch fortschreitende Unterteilung der Parameterintervalle sukzessive in jeweils zwei Teilkurven zerlegt. Dies wird solange gemacht, bis die Strecke zwischen zwei Intervallendpunkten eine gute Approximation der Kurve darstellt.

Um die Qualität der Approximation zu beurteilen, kann zum Beispiel die folgende Heuristik Verwendung finden: Die Kurve sei, wie in Abbildung 14 dargestellt, im Parameterintervall $[a, b] \subseteq P$ durch die Strecke zwischen den Kurvenpunkten $k(a)$ und $k(b)$ approximiert. Es wird nun der Abstand zwischen den Punkten m_a und m_k gemessen. Ist dieser kleiner als eine vorgegebene Schranke ε, wird das Intervall nicht weiter unterteilt. Die beiden Punkte berechnen sich dabei wie folgt:

$$m_a = \frac{k(a) - k(b)}{2} \text{ und } m_k = k\left(\frac{a+b}{2}\right)$$

Frage 3.8: Wie funktioniert die implizite Darstellung einer Kurve im $\mathbb{R}^2$?
Bei der impliziten Darstellung ist die Kurve als Nullstellenmenge einer Funktion F gegeben:

$$F(x, y) = 0 \text{ mit } F : \mathbb{R}^2 \to \mathbb{R}$$

So kann ein Kreis in der x-y-Ebene mit dem Radius r um den Mittelpunkt $m = (m_x, m_y)$ wie folgt dargestellt werden:

$$(x - m_x)^2 + (y - m_y)^2 - r^2 = 0$$

Diese implizite Darstellung wird bei der Verrasterung von Kreisen sehr nützlich werden, da sie erlaubt, für einen Punkt zu bestimmen, ob dieser innerhalb, außerhalb oder auf dem Kreis liegt.

Eine weitere Klasse von Kurven sind die sogenannten Kegelschnittkurven, welche ebenfalls in einer impliziten Form gegeben sind. Die allgemeine Form lautet:

$$ax^2 + by^2 + cxy + dx + ey + f = 0$$

Mit Hilfe dieser Kegelschnittkurven lassen sich zum Beispiel Ellipsen, Hyperbeln und Parabeln im Raum darstellen.

Frage 3.9: Was ist ein Polygonkurvensegment?

Ein Polynomkurvensegment im $\mathbb{R}^d$ ist wie folgt gegeben:

$$p(t) = \sum_{i=0}^{n} a_i t^i, \ t \in [0,1], \ a_i \in \mathbb{R}^d$$

Im Prinzip kann ein Polynomkurvensegment als eine Parameterdarstellung aufgefasst werden, bei der die $f_i(t)$ alle Polynome vom Grad n sind, deren Koeffizienten durch die Punkte a_i bestimmt werden. Damit wird das Aussehen der Kurve also durch die Koeffizienten bestimmt. Problematisch ist hierbei jedoch, dass der Zusammenhang zwischen Kurve und Koeffizienten nicht sehr intuitiv ist. Außerdem tendieren Polynomkurven dazu zu oszillieren. Es wurde daher nach einer anderen Klasse von Kurven gesucht, die ein besseres Verhalten aufweisen, was zu den Bezier-Kurven führte. Praktischerweise lassen sich Polynomkurvensegmente auch mit Hilfe von Bezier-Kurven darstellen, so dass bei der Verwendung dieser Kurven also nichts verloren geht.

3.2 Bezier-Kurven

Frage 3.10: Was ist eine Bezier-Kurve?

Bezier-Kurven werden hier über den Algorithmus von de Casteljau eingeführt. Später wird auf den Zusammenhang mit Bernstein-Polynomen eingegangen. Wir beginnen mit der Definition einer **linearen** Bezier-Kurve. Eine lineare Bezierkurve wird über zwei Punkte definiert und entspricht im Prinzip einer Strecke, wie sie in diesem Kapitel zuvor eingeführt wurde. Nur dass die Formel im Fall der Bezier-Kurven etwas umgestellt wird. Eine Strecke war definiert als:

$$s[p_1, p_2] = \{p : p = p_1 + \lambda \cdot (p_2 - p_1), \lambda \in [0,1]\}$$

Statt λ benutzen wird im folgenden die Variable t welche stets im Intervall $[0,1]$ liegt. Punkte werden mit b benannt. Also wird aus der Definition:

$$b_1 + t \cdot (b_2 - b_1) = b_1 + t \cdot b_2 - t \cdot b_1 = (1 - t) \cdot b_1 + t \cdot b_2$$

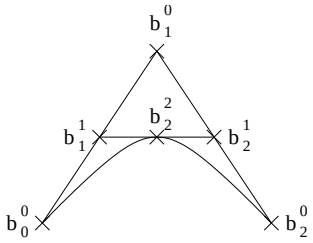


Abbildung 15: Quadratische Bezier-Kurve, die durch die Punkte b_0^0, b_1^0, b_2^0 gegeben ist

Im Prinzip wird durch diese Formel, welche ein lineares Polynom ist, ein Punkt auf der Strecke zwischen b_1 und b_2 interpoliert. Um zu verstehen, wie sich eine Bezierkurve ergibt, die aus mehr als zwei Bezier-Kontrollpunkten besteht, wird nun erst einmal gezeigt, in welchem Zusammenhang **quadratische** Bezier-Kurven mit linearen stehen.

Eine quadratische Bezier-Kurve ist durch drei Punkte b_1, b_2, b_3 gegeben. Abbildung 15 zeigt eine solche Kurve. Wie berechnet sich zu einem gegebenen t nun ein Punkt auf dieser Kurve? Dazu kann der **Algorithmus von de Casteljau** benutzt werden. In der Abbildung wurde ein Punkt für $t = 0.5$ berechnet. Er berechnet sich wie folgt: Es wird eine Interpolation zu $t = 0.5$ zwischen den Punkten b_0^0 und b_1^0 gemacht, woraus sich der Punkt b_1^1 ergibt. Ebenso interpoliert man zu $t = 0.5$ zwischen den Punkten b_1^0 und b_2^0, woraus sich b_2^1 ergibt. Zwischen diesen beiden Ergebnispunkten interpoliert man nun wieder mit $t = 0.5$ und erhält schließlich b_2^2.

Diesen Vorgang kann nun für quadratische Bezierkurven auch allgemeiner aufgeschrieben werden. Es ergibt sich dann:

$$b_1^1 = (1-t) \cdot b_0^0 + t \cdot b_1^0 \text{ und } b_2^1 = (1-t) \cdot b_1^0 + t \cdot b_2^0$$

Da gilt $b_2^2 = (1-t) \cdot b_1^1 + t \cdot b_2^1$ ergibt eine Einsetzung der obigen beiden Ergebnisse schließlich:

$$\begin{aligned} b_2^2 &= (1-t) \cdot \left((1-t) \cdot b_0^0 + t \cdot b_1^0\right) + t \cdot \left((1-t) \cdot b_1^0 + t \cdot b_2^0\right) \\ &= (1-t)^2 \cdot b_0^0 + 2t(1-t) \cdot b_1^0 + t^2 \cdot b_2^0 \end{aligned}$$

Oder anders interpretiert, ergibt sich ein Punkt auf der Kurve zu einem gegebenen t als eine gewichtete Addition der drei Kontrollpunkte. Die Gewichtung ergibt sich bei Bezier-Kurven beliebigen Grades durch eine ganz bestimmte Art von Polynomen, den **Bernstein-Polynomen**. Dies bedeutet, dass sich die Gewichtungen der Kontrollpunkte nach einem mathematischen System ergeben. Allgemein gilt für einen Punkt $b(t)$ auf der Kurve, dass er sich ergibt aus:

$$b(t) = \sum_{i=0}^{n} B_i^n(t) \cdot b_i; t \in [0,1], b_i \in \mathbb{R}^d$$

Dabei sind die $B_i^n(t)$ die sogenannten Bernstein-Polynome. Bei den Kontrollpunkten wurde der obere Index weggelassen, da bei dieser Formel keine Zwischenpunkte mehr berechnet werden müssen.

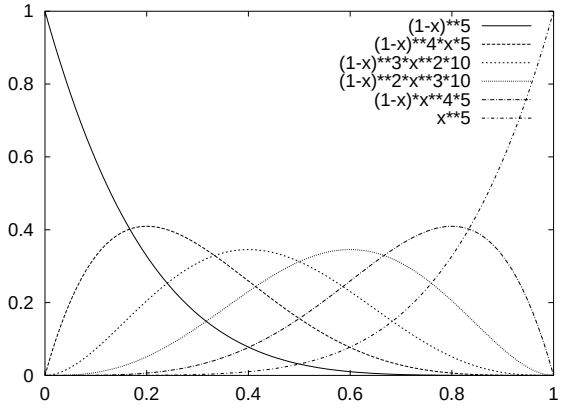


Abbildung 16: Die Bernstein-Polynome vom Grad 5

Frage 3.11: Wie sind die Bernstein-Polynome definiert?
Bernstein-Polynome vom Grad n sind definiert als:

$$B_i^n(x) = \binom{n}{i} x^i (1-x)^{n-i}, i \in \{0, \ldots, n\}$$

So ergeben zum Beispiel die Bernstein-Polynome vom Grad 2 tatsächlich, die etwas weiter ober berechneten Polynome:

$$B_0^2(t) = \binom{2}{0} t^0 (1-t)^2 - 0 = (1-t)^2, B_1^2 = 2t(1-t), B_2^2 = t^2$$

Abbildung 16 zeigt die Bernsteinpolynome vom Grad 5.

Frage 3.12: Welche Eigenschaften haben die Bernstein-Polynome?
Bernstein-Polynome haben unter anderem die folgenden 7 Eigenschaften.

Partition der 1: Für jedes $x \in [0, 1]$ ergibt die Summe der $B_i^n(x)$ gleich 1:

$$\sum_{i=0}^{n} B_i^n(x) = \sum_{i=0}^{n} \binom{n}{i} x^i (1-x)^{n-i} = (x + 1 - x)^n = 1^n = 1$$

Dies gilt einfach wegen der folgenden Beziehung auf die man unter anderem im Zusammenhang mit Binomialkoeffizienten stößt:

$$(a+b)^n = \sum_{i=0}^{n} \binom{n}{i} a^i b^{n-i}$$

Positivität: Bernstein-Polynome sind über dem Einheitsintervall $[0, 1]$ positiv, d.h.:

$$B_i^n(x) \geq 0, i \in \{0, \ldots, n\}, t \in [0, 1]$$

Dies kann man sich relativ schnell klar machen, wenn man sich die einzelnen Komponenten in der Definition der Bernstein-Polynome anschaut, die alle stets größer oder gleich 0 werden.

Maxima: Das Bernstein-Polynom $B_i^n(x)$ nimmt über $[0,1]$ sein Maximum an der Stelle $\frac{i}{n}$ an, für alle $i \in \{0,\ldots,n\}$. Dies bedeutet, dass ein Kontrollpunkt b_i des Kontrollpolygonzuges seinen größten Einfluss bei $t = \frac{i}{n}$ hat.

Wir benutzen den Satz für die Differentation von Bernsteinpolynomen und setzen die Ableitung von $B_i^n(x)$ gleich Null:

$$n \cdot \left(B_{i-1}^{n-1}(x) - B_i^{n-1}(x)\right) = 0$$

Es muss also die rechte Klammer gleich Null sein, d.h.:

$$\frac{(n-1)!}{(i-1)!(n-i)!}x^{i-1}(1-x)^{n-i} - \frac{(n-1)!}{i!(n-1-i)!}x^i(1-x)^{(n-1)-i} = 0$$

Jetzt wird erst einmal jede Menge ausgeklammert:

$$\frac{(n-1)!}{(i-1)!((n-i)-1)!} \cdot t^{i-1}(1-t)^{(n-i)-1} \cdot \left(\frac{1}{n-i} \cdot (1-t) - \frac{1}{i} \cdot t\right) = 0$$

Wieder reicht es, wenn die rechte Klammer Null wird:

$$\begin{aligned}
\frac{1}{n-i} \cdot (1-t) - \frac{1}{i} \cdot t &= 0 \\
\frac{1}{n-i} - \frac{1}{n-i} \cdot t - \frac{1}{i} \cdot t &= 0 \\
\frac{1}{n-i} - t \cdot \left(\frac{1}{n-i} + \frac{1}{i}\right) &= 0 \\
\frac{1}{n-i} - t \cdot \left(\frac{i+n-i}{(n-i)\cdot i}\right) &= 0 \\
\frac{1}{n-i} - t \cdot \frac{n}{(n-i)\cdot i} &= 0 \\
t \cdot \frac{n}{(n-i)\cdot i} &= \frac{1}{n-i} \\
t &= \frac{(n-i)\cdot i}{(n-i)\cdot n} \\
t &= \frac{i}{n}
\end{aligned}$$

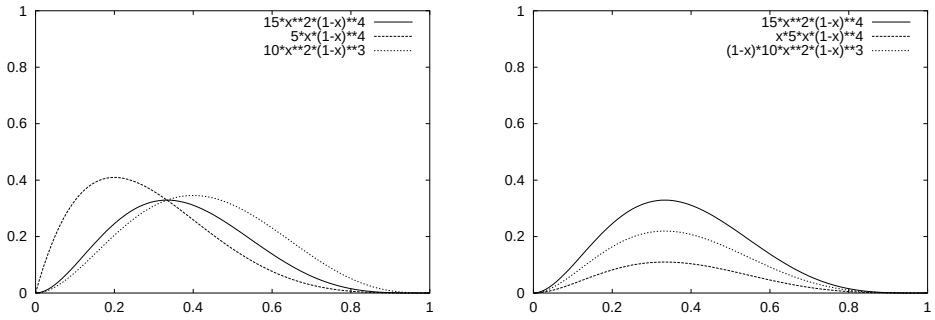
Somit nimmt das Bernsteinpolynom $B_i^n(x)$ an der Stelle $\frac{i}{n}$ sein Maximum an.

Summation: Der folgenden Satz war Teil einer Übungsaufgabe:

$$\sum_{i=0}^{n} i \cdot B_i^n(x) = n \cdot x$$

Symmetrie: Wie man in Abbildung 16 erkennen kann, gibt es eine Symmetrie bei den Bernstein-Polynomen. Und zwar der Form:

$$B_i^n(x) = B_{n-i}^n(1-x), i \in \{0,\ldots,n\}$$



(a) $B_2^6(x), B_1^5(x), B_2^5(x)$ (b) $B_2^6(x), t \cdot B_1^5(x), (1-t) \cdot B_2^5(x)$

Abbildung 17: Beispiel für Rekursion: $B_2^6(x) = t \cdot B_1^5(x) + (1-x) \cdot B_2^5(x)$

Dies ergibt sich unter anderem aus der Symmetrie der Binomialkoeffizienten: $\binom{n}{i} = \binom{n}{n-i}$. Denn damit gilt:

$$\begin{aligned} B_i^n(x) &= \binom{n}{i} x^i (1-x)^{n-i} = \binom{n}{n-i} (1-(1-x))^{n-(n-i)} (1-x)^{n-i} \\ &= \binom{n}{n-i} (1-x)^{n-i} (1-(1-x))^{n-(n-i)} = B_{n-i}^n (1-x) \end{aligned}$$

Rekursion: Für $i \in \{0, \ldots, n-1\}$ gilt:

$$B_{i+1}^{n+1}(x) = x \cdot B_i^n(x) + (1-x) \cdot B_{i+1}^n(x)$$

Diese Formel bedeutet also, dass ein Bernsteinpolynom vom Grad $n+1$ durch die Konvexkombination zweier Bernsteinpolynome vom Grad n dargestellt werden kann. Abbildung 17 zeigt z.B. wie $B_2^6(x)$ durch die beiden Bernstein-Polynome $B_1^5(x)$ und $B_2^5(x)$ dargestellt werden kann.

Differentation: Für die Ableitung eines Bernstein-Polynoms $B_i^n(x)$ gilt:

$$(B_i^n(x))' = n \cdot \left(B_{i-1}^{n-1}(x) - B_i^{n-1}(x)\right)$$

Im folgenden Beweis wird $n! = n \cdot (n-1)!$, sowie die Produktregel der Differentation genutzt. Diese lautet:

$$(f(x) \cdot g(x))' = f'(x) \cdot g(x) + f(x) \cdot g'(x)$$

$$\begin{aligned}
\frac{d}{dx}B_i^n(x) &= \frac{d}{dx}\binom{n}{i}x^i(1-x)^{n-i} \\
&= \frac{n!}{i!(n-i)!}\cdot\big(i\cdot x^{i-1}(1-x)^{n-i} \\
&\quad -(n-i)\cdot x^i(1-x)^{(n-i)-1}\big) \\
&= n\cdot\left(\frac{(n-1)!}{i!(n-i)!}\cdot i\cdot x^{i-1}(1-x)^{n-i}\right. \\
&\quad \left.-\frac{(n-1)!}{i!(n-i)!}\cdot(n-i)\cdot x^i(1-x)^{(n-i)-1}\right) \\
&= n\cdot\left(\frac{(n-1)!}{(i-1)!(n-i)!}\cdot x^{i-1}(1-x)^{n-i}\right. \\
&\quad \left.-\frac{(n-1)!}{i!((n-i)-1)!}\cdot x^i(1-x)^{(n-i)-1}\right) \\
&= n\cdot\big(B_{i-1}^{n-1}(x)-B_i^{n-1}(x)\big)
\end{aligned}$$

Das Minus in der Klammer kommt durch die Ableitung von $(1-x)^{n-i}$ zustande, da hier die Regel „innere Ableitung mal äußere Ableitung“ angewandt werden muss.

Frage 3.13: Wie ist ein Bezier-Kurvensegment im $\mathbb{R}^d$ definiert?
Im Prinzip genau so, wie einige Seiten zuvor eingeführt:

$$b(t)=\sum_{i=0}^{n} b_i B_i^n(t), t\in[0,1], b_i\in\mathbb{R}^d$$

Frage 3.14: Wie kann man ein Polynomkurvensegment mit Hilfe von Bezier-Kurvensegmenten darstellen? Wieso geht dies? Was bedeutet dies?
Gegeben sei ein Polynomkurvensegment $p(t)=\sum_{i=0}^{n} a_i t^i$. Dann lässt sich dieses durch das Bezier-Kurvensegment $b(t)=\sum_{i=0}^{n} b_i B_i^n(t)$ darstellen, wobei sich die einzelnen b_i aus den a_i ergeben. Und zwar als:

$$b_i=\sum_{k=0}^{i}\binom{i}{k}\binom{n}{k}^{-1}a_k$$

Wieso geht dies nun? Die Klasse aller Polynome vom Grad n über den reellen Zahlen bilden einen Vektorraum der Dimension $n+1$. Dies bedeutet, dass sich jedes Polynom als eine Linearkombination der folgenden Gestalt darstellen lässt:

$$\sum_{i=0}^{n}\lambda_i e_i(x)$$

Dabei sind die λ_i Skalare und die Menge $\{e_o(x),\ldots,e_n(x)\}$ bildet eine Basis des Vektorraums. Eine bekannte Basis für diesen Vektorraum bilden die Monome $e_i(x)=x^i$. Eine weitere Basis ist jedoch die Menge der Bernsteinpolynome, d.h. $e_i(x)=B_i^n(x)$.

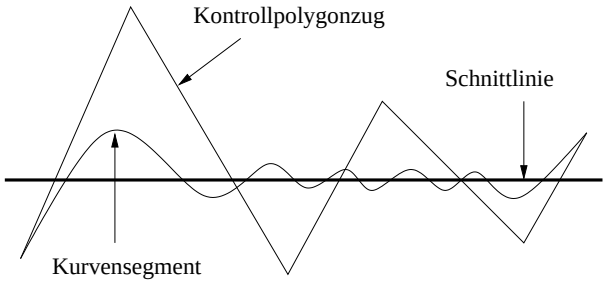


Abbildung 18: Beschränkte Schwankung: So etwas kann nicht passieren

Dementsprechend handelt es sich bei der oben genannten Umrechnung lediglich um eine Basistransformation. Dies bedeutet, dass man alle Kurven, welche durch Polynomkurvensegmente darstellbar sind, auch durch Bezier-Kurvensegmente darstellen kann und man dementsprechend also nichts verliert, wenn man statt Polynomkurvensegmenten Bezier-Kurvensegmente verwendet.

Frage 3.15: Welche Eigenschaften haben Bezier-Kurvensegmente?
Bezier-Kurvensegmente besitzen die folgenden Eigenschaften:

Verhalten in den Endpunkten: Das Kontrollpolygon und das Kurvensegment stimmen in Anfangs- und Endpunkt überein. Das erste und das letzte Segment des Kontrollpolygonzugs ist dort tangential zur Kurve.

Der Beweis hierzu ergibt sich direkt, wenn man $t = 0$ bzw. $t = 1$ setzt. Man kann dies aber auch alternativ direkt aus den Graphen für die Bernsteinpolynome ablesen.

Konvexe-Hülle-Eigenschaft: Das Kurvensegment liegt in der konvexen Hülle der Bezierpunkte. Somit liegt es auch in der konvexen Hülle des Kontrollpolygonzugs.

Diese Eigenschaft ergibt sich direkt aus dem Algorithmus von de Casteljau.

Einfluss von Kontrollpunkten: Der Kontrollpunkt b_i hat den größten Einfluss auf die Kurve $b(t)$ bei $t = \frac{i}{n}$.

Der Einfluss eines Punktes ist für das t am größten, für das sein Bernsteinpolynom sein Maximum annimmt. Das Bernsteinpolynom $B_i^n(t)$ wird maximal an der Stelle $t = \frac{i}{n}$.

Beschränkte Schwankung: Keine Gerade schneidet ein Beziersegment öfter als den dazu gehörigen Kontrollpolygonzug. Das Segment schwankt also nicht stärker als der Polygonzug. D.h. so etwas wie in Abbildung 18 kann also nicht passieren, bzw. der eingezeichnete Kontrollpolygonzug kann nicht zur Kurve gehören.

Frage 3.16: Wie unterteilt ein Punkt auf einer Bezierkurve diese?

Ein Punkt $b(t), t \in [0,1]$ unterteilt ein Beziersegment in zwei Teilkurven. Diese Teilkurven sind wieder Bezier-Segmente gleichen Grades. Die Bezier-Punkte dieser beiden Segmente sind gegeben durch die Hilfspunkte $b_0^0, b_1^1, \ldots, b_n^n$ bzw. $b_n^n, b_n^{n-1}, \ldots, b_n^0$ des Algorithmus von de Casteljau an der Stelle t.

Frage 3.17: Welche Laufzeit hat der Algorithmus von de Casteljau?

Der Zeitaufwand des Algorithmus von de Casteljau beträgt bei einer Kurve mit $n+1$ Kontrollpunkten $O(n^2)$. Zuerst werden n neue Punkte berechnet, welche jeweils auf den n Strecken des Kontrollpolygonzuges liegen. Diese werden zu $n-1$ neuen Strecken verbunden und dann wiederum $n-1$ neue Punkte berechnet. Dies geht so lange, bis man nur noch zwei Punkte zu einer Strecke verbindet und auf dieser einen Punkt ausrechnet. Insgesamt berechnet man also die folgende Anzahl von Punkten:

$$n + (n-1) + (n-2) + \cdots + 2 + 1 = \sum_{i=1}^{n} i = \sum_{i=0}^{n} i = \frac{n(n+1)}{2} \in O(n^2)$$

Frage 3.18: Wie können Kurvenpunkte eines Bezier-Segments effizient ausgerechnet werden?

Es gibt noch einen effizienteren Algorithmus als den Algorithmus von de Casteljau. Der effizientere Algorithmus benötigt lineare Zeit ($O(n)$), um einen Punkt auf der Kurve zu berechnen. Dieser Algorithmus benutzt dazu das Horner-Schema und wird daher im folgenden Bezier-Horner genannt. Das Horner-Schema wird hier anhand eines Beispiel erläutert. Angenommen, es ist ein Polynom vom Grad n gegeben, in diesem Beispiel vom Grad 3:

$$p(x) = a_3 \cdot x^3 + a_2 \cdot x^2 + a_1 \cdot x + a_0$$

Rechnet man dieses Polynom ohne viel nachzudenken aus, beträgt der Aufwand dafür $O(n^2)$. Denn zur Berechnung von $a_n \cdot x^n$ benötigt man n Multiplikationen. Insgesamt ergbt dies $\sum_{i=0}^{n} i = O(n^2)$ Multiplikationen zuzüglich n Additionen. Allerdings ist diese Vorgehenweise sehr dumm, da man viele Werte mehrfach berechnet. Zur Berechnung von x^3 zum Beispiel berechnet man zwischendurch wieder x^2, wenn man die Monome nach aufsteigendem Grad berechnet. Man kann sich diese Zwischenergebnisse also merken und muss dann zur Berechnung des nächsthöheren Monoms lediglich eine Multiplikation durchführen. Insgesamt ergibt dies $n-1$ Multiplikationen. Zusätzlich werden noch n Multiplikationen für die Koeffizienten $a_i, i \in \{1, \ldots, n\}$ benötigt und wie auch oben n Additionen. Insgesamt also $2n + (n-1) = 3n - 1$ Operationen. Es geht jedoch noch cleverer, indem man das Polynom $p(x)$ wie folgt umschreibt:

$$((a_3 x + a_2)x + a_1)x + a_0$$

Wie man schnell einsieht, müssen hier nur 3 Multiplikationen und Additionen durchgeführt werden. Bei einem Polynom vom Grad n also insgesamt $2n$ Operationen.

Frage 3.19: Wie funktioniert die Graderhöhung bei Bezierkurven?

Algorithmus 1 Das Bezier-Horner-Schema

if $t^* \leq \frac{1}{2}$ **then**
  $b = b_n$;
  for $i = 1$ **to** n **do**
    $b = b \cdot \frac{i}{n-i+1} \cdot \frac{t^*}{(1-t^*)} + b_{n-i}$;
    $b = (1-t^*)^n \cdot b$;
  end for
else
  $b = b_0$;
  for $i = 1$ **to** n **do**
    $b = b \cdot \frac{i}{n-i+1} \cdot \frac{(1-t^*)}{t^*} + b_i$;
    $b = (t^*)^n \cdot b$;
  end for
end if
$b(t^*) = b$;

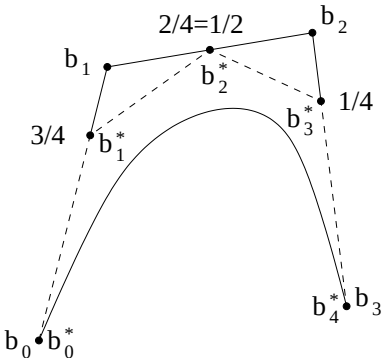


Abbildung 19: Graderhöhung bei Bezierkurven

Gegeben sei eine Bezierkurve mit $n+1$ Kontrollpunkten $b_0, \ldots, b_n \in \mathbb{R}^d$. Es soll nun der Grad der Kurve erhöht werden, d.h. dieselbe Kurve soll mit $n+2$ Kontrollpunkten $b_0^*, \ldots, b_{n+1}^* \in \mathbb{R}^d$ dargestellt werden. Dies wird wie folgt erreicht, wobei der neue Anfangs- und Endpunkt mit dem alten übereinstimmt. Es müssen also nur die Punkte in der Mitte neu berechnet werden:

$$\begin{aligned} b_0^* &= b_0 \\ b_i^* &= \alpha_i b_{i-1} + (1-\alpha_i) b_i, \text{ mit } \alpha_i = \frac{i}{n+1} \text{ für } i = 1, \ldots, n \\ b_{n+1}^* &= b_n \end{aligned}$$

Abbildung 19 zeigt das Ergebnis dieses Algorithmus für eine Kurve mit ursprünglich vier Kontrollpunkten. Wie kann man sich diese Formel nun leicht merken? Der Bruch $i/n+1$ erinnert an die Maxima der Bernsteinpolynome.
Frage 3.20: Wie wird die Eigenschaft der beschränkten Schwankung bewiesen?

Die Eigenschaft der beschränkten Schwankung wird mit Hilfe des Algorithmus für die Graderhöhung bewiesen. Gegeben ist dabei die folgende Eigenschaft: Wählt man auf einer gegebenen Kurve eine Folge von Stützstellen und verbindet diese zu einem Polygonzug, dann wird dieser Polygonzug von einer beliebigen vorgegebenen Gerade nicht öfter geschnitten als die Kurve.

Der Trick am Beweis ist nun, dass man dieses Argument quasi umdrehen kann. Gegeben sei nun ein Kontrollpolygonzug einer Bezierkurve. Dieser wird nun als eine vorgegebene Kurve betrachtet. Führt man nun den Graderhöhungsalgorithmus durch, dann kann man dies so interpretieren, dass auf dem Kontrollpolygonzug (also unserer „Kurve“) eine Folge von Stützstellen gewählt wird. Der neue Kontrollpolygonzug wird also nicht öfter geschnitten als der ursprüngliche. Wird der Graderhöhungsalgorithmus nun unendlich oft durchgeführt, konvergiert der Kontrollpolygonzug gegen die Kurve. Daraus folgt, dass eine Kurve, die durch einen Kontrollpolygonzug gegeben ist, nicht öfter geschnitten wird als eben dieser.

3.3 B-Spline-Kurven

Frage 3.21: Was sind die Vorteile von B-Splines gegenüber Bezier-Kurven?
Bezier-Kurven haben zwei Nachteile, die bei B-Splines nicht gegeben sind:

Keine lokale Kontrollierbarkeit: Die Veränderung eines Bezier-Punktes ändert das Aussehen der ganzen Kurve.

Hoher Polynomgrad: Komplexere Formen benötigen viele Bezier-Punkte, womit jedoch auch der Grad des Polynoms sehr hoch wird. Dies führt zu erhöhtem Rechenaufwand und zu numerischen Ungenauigkeiten.

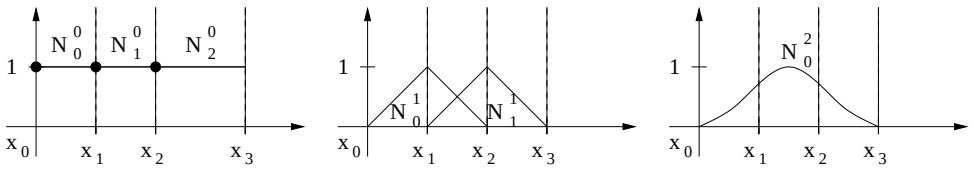
Die Idee bei Spline-Kurven besteht nun darin, komplexe Formen aus mehreren einfachen Formen zusammen zu setzen.

Frage 3.22: Wann ist eine Funktion eine Spline-Funktion?
Seien $x_0 \leq x_1 \leq \cdots \leq x_k, x_i \in \mathbb{R}$ gegeben. Eine Funktion S heißt Spline-Funktion vom Grad d bzw. von der Ordnung $d+1$, wenn die folgenden beiden Bedingungen erfüllt sind:

1. S ist ein Polynom vom Grad d in jedem Teilintervall $[x_i, x_{i+1}], 0 \leq i < k-1$.
2. $S \in C^{(d-1)}[x_0, x_k]$.

$\xi = (x_0, x_1, \ldots, x_k)$ heißt **Knotenvektor** der Spline-Funktion.

Frage 3.23: Wie sind die B-Spline-Funktionen definiert?
Bei der B-Spline-Technik benutzt man anstatt der Bernstein-Polynome die sogenannten B-Spline-Funktionen als Gewichtungsfunktion für die Kontrollpunkte. Die B-Spline-Funktionen N_i^n sind rekursiv definiert.



(a) Die B-Splines $N_i^0(x)$ (b) Die B-Splines $N_i^1(x)$ (c) Die B-Spline $N_0^2(x)$

Abbildung 20: Die B-Splines $N_i^0(x)$, $N_i^1(x)$ und $N_i^2(x)$ für einen Knotenvektor $\xi = (x_0, x_1, x_2, x_3)$

Gegeben sei ein Knotenvektor $\xi = (x_0, x_1, \ldots, x_k)$ mit $k+1$ Elementen. Dann sind die B-Spline-Funktionen wie folgt definiert:

$$\begin{aligned} N_i^0(x) &:= \begin{cases} 1 & \text{für } x \in [x_i, x_{i+1}[\\ 0 & \text{sonst} \end{cases} \quad \text{für } i \in \{0, \ldots, k-1\} \\ N_i^n(x) &:= \frac{x - x_i}{x_{i+n} - x_i} \cdot N_i^{n-1}(x) + \frac{x_{i+n+1} - x}{x_{i+n+1} - x_{i+1}} \cdot N_{i+1}^{n-1}(x) \\ &\quad \text{mit } 0 \leq i \leq k-n-1, 1 \leq n \leq k-1 \end{aligned}$$

Dabei ist jede Division durch Null (u.a. $\frac{0}{0}$) als Null definiert. Abbildung 20(a) zeigt die B-Spline-Funktionen $N_i^0(x)$ für einen Knotenvektor $\xi = (x_0, x_1, x_2, x_3)$.

Frage 3.24: Wie setzen sich die B-Spline-Funktionen (rekursiv) zusammen?

Um einen Einblick in die „Funktionsweise“ der B-Spline-Funktionen zu bekommen, ist es sinnvoll, sich erst einmal für einen äquidistanten Knotenvektor $\xi = \{x_0, x_1, \ldots, x_k\}$ anzuschauen, wie sich die N_i^n aus den N_i^{n-1} bzw. N_{i+1}^{n-1} ergeben. Äquidistant bedeutet, dass der Abstand zwischen allen x_i und $x_{i+1}, i \in \{0, \ldots, k-1\}$ gleich ist. Also zum Beispiel $\xi = \{0, 1, 2, 3, 4, 5\}$ oder auch $\xi = \{0, 3, 6, 9, 12\}$.

Abbildung 20(a) zeigt die einzelnen N_i^0, in welchen die Rekursion quasi endet. In dieser Abbildung ist der Knotenvektor jedoch nicht äquidistant gewählt worden. Aus diesen N_i^0 setzen sich nun jeweils die N_i^1 zusammen und zwar der Form:

$$N_i^1(x) = \frac{x - x_i}{x_{i+1} - x_i} \cdot N_i^0(x) + \frac{x_{i+2} - x}{x_{i+2} - x_{i+1}} \cdot N_{i+1}^0(x)$$

Oder etwas konkreter, ergibt sich z.B. N_0^1 als:

$$N_0^1(x) = \frac{x - x_0}{x_1 - x_0} \cdot N_0^0(x) + \frac{x_2 - x}{x_2 - x_1} \cdot N_1^0(x)$$

N_0^1 setzt sich also aus N_0^0 und N_1^0 zusammen, welche entsprechend mit $(x - x_0)/(x_1 - x_0)$ bzw. $(x_2 - x)/(x_2 - x_1)$ gewichtet werden.

Als nächstes sollte man sich überlegen, für welche x die B-Spline-Funktion $N_0^1(x)$ überhaupt ein von Null verschiedenes Ergebnis annimmt, d.h. für welche x die beiden Brüche überhaupt erst „greifen“ können. Es gilt $N_0^0(x) \neq 0$, wenn $x \in [x_0, x_1[$ und $N_1^0(x) \neq 0$, wenn $x \in [x_1, x_2[$. Die ganze Funktion ist also im

Intervall $]x_0, x_2[$ von Null verschieden, wobei sie im Intervall $[x_0, x_1[$ linear von 0 bis 1 läuft und anschließend im Intervall $[x_1, x_2[$ wieder von 1 bis 0 (ebenfalls linear). Das Intervall ist bei x_0 offen, da $N_0^1(0) = 0$ gilt.

Abbildung 20(b) zeigt die „Kurven“, die dadurch entstehen. In diesem konkreten Beispiel besteht der Knotenvektor aus vier Elementen. Aufgrund der Definition von $N_i^0(x)$ ergeben sich daraus drei konstante Funktionen. Und aufgrund der Definition der Rekursion ergeben sich daraus wiederum nur zwei $N_i^1(x)$ (nämlich $N_0^1(x)$ und $N_1^1(x)$). Die liegt daran, dass wenn man ein $N_2^1(x)$ berechnen wollte, dieses von $N_2^0(x)$ und $N_3^0(x)$ abhängen würde. Letzteres würde jedoch nur für ein x zwischen x_3 und x_4 ungleich 0 sein. Der Knotenvektor geht jedoch nur von x_0 bis x_3. Entsprechend ergibt sich für $N_0^2(x)$ nur eine Kurve über das Intervall $[x_0, x_3[$. Dies hängt mit den Trägerintervallen zusammen, auf die später eingegangen wird.

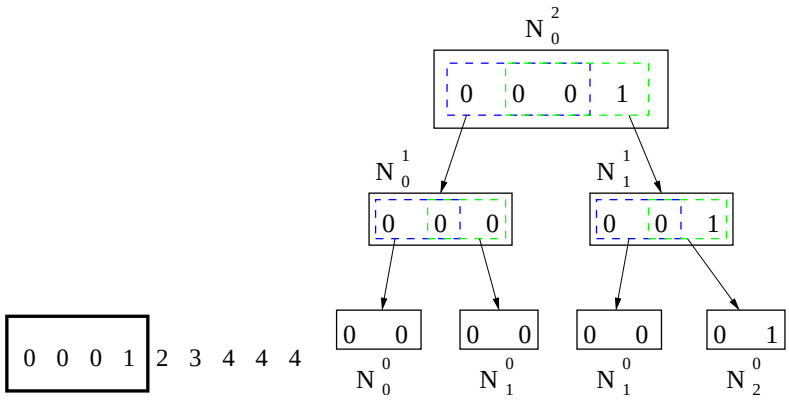
Die Trägerintervalle sind auch das Argument dafür, dass die Brüche in der rekursiven Definition der B-Spline-Funktion nur Werte zwischen 0 und 1 annehmen. Wie wir später sehen werden, ist das Trägerintervall der Funktion $N_i^n(x)$ das Intervall $[x_i, x_{i+n+1}[$, d.h. für $x \in [x_i, x_{i+n+1}[$ ist $N_i^n(x)$ von 0 verschieden. Betrachten wir nun den linken Summand in der rekursiven Definition. Er ist von $N_i^{n-1}(x)$ abhängig, dessen Trägerintervall das Intervall $[x_i, x_{i+n}[$ ist. Dies hat jedoch zur Konsequenz, dass das x im Bruch auch nur Werte zwischen x_i und x_{i+n} annimmt. Lässt man in

$$\frac{x - x_i}{x_{i+n} - x_i}$$

das x nun von x_i nach x_{i+n} wandern, so sieht man schnell, dass der Bruch Werte zwischen 0 und 1 annimmt und zwar linear. Dasselbe Argument gilt entsprechend für den rechten Summanden, nur dass der Bruch hier linear von 1 nach 0 abfällt.

Frage 3.25: Wie entstehen die Polynome der B-Spline-Funktionen?
Wie schon gesehen, sind die B-Spline-Funktionen rekursiv definiert. Die Formel für $N_i^n(x)$ besteht aus zwei Summanden, welche wiederum B-Spline-Funktionen eines Grades niedrigeren Grades beinhalten, nämlich $N_i^{n-1}(x)$ und $N_{i+1}^{n-1}(x)$. An diese B-Spline-Funktionen niedrigeren Grades werden nun lineare Funktionen multipliziert, welche über dem Trägerintervall der B-Spline-Funktionen niedrigeren Grades von Null verschieden sind. Geht man in der Rekursion nun noch einen Schritt weiter, sofern man nicht schon bei $N_i^0(x)$ angelangt ist, multipliziert man diese lineare Funktion unter Umständen wieder mit einer linearen Funktion, so dass insgesamt eine quadratische Funktion entsteht. In Anhängigkeit davon, wie oft man diesen Schritt wiederholt, bis man bei $N_i^0(x)$ angelangt ist, entstehen Polynome höheren Grades.

Frage 3.26: Erläutere anschaulich den Zusammenhang zwischen dem Knotenvektor und den B-Spline-Funktionen!
Es ist möglich die Berechnung von $N_i^n(x)$ etwas anschaulicher zu betrachten als dies bei der rekursiven Definition von $N_i^n(x)$ der Fall ist. Bei dieser weiteren Betrachtungsweise sieht man auch recht schnell ein, was passiert, wenn Knoten im Knotenvektor ξ öfter als einmal vorkommen, wie zum Beispiel in



(a) Fenster für $N_0^2(x)$ (b) Fenster in Fenster für $N_0^2(x)$

Abbildung 21: Betrachtungsweise als Fenster für $N_0^2(x)$

$\xi = \{0, 0, 0, 1, 2, 3, 4, 4, 4\}$. Im folgenden wird nun genau dieser Knotenvektor betrachtet.

Eine weitere Interpretation von $N_i^n(x)$ besteht darin, dieses als eine Art Fenster zu sehen, welches über den Knotenvektor geschoben werden kann. Dabei gibt i die Startposition des Fensters an und n die Breite des Fensters, welche genau $n + 2$ beträgt. Abbildung 21(a) zeigt, wie das Fenster für $N_0^2(x)$ für den oben erwähnten Knotenvektor ξ aussieht. Diesem Fenster kann man z.B. auch sofort entnehmen, über welchem Intervall die jeweilige B-Spline-Funktion ungleich 0 ist. In diesem Fall ist dies das Intervall $[0, 1[$

Eine Betrachtung von Abbildung 21(b) zeigt nun auch, was passiert, wenn der Knotenvektor (welcher nur monoton und *nicht* streng monoton sein muss) gleiche Einträge enthält.

Frage 3.27: Welche Rolle spielen die Differenzen zwischen den Elementen des Knotenvektors?

Frage 3.28: Wie lassen sich die Bernstein-Polynome mit Hilfe von B-Spline-Funktionen darstellen? Was hat dies für Konsequenzen?

Um das Bernstein-Polynom B_i^n darzustellen, benötigt man einen Knotenvektor mit $2 \cdot (n + 1)$ Elementen. Die ersten $(n + 1)$ Elemente dieses Knotenvektors sind lauter Nullen, die restlichen $(n + 1)$ lauter Einsen. Der Knotenvektor hat also die folgende Form:

$$\xi = (\underbrace{0, 0, \ldots, 0}_{(n+1)\text{-mal}}, \underbrace{1, \ldots, 1, 1}_{(n+1)\text{-mal}})$$

In diesem Fall stimmen die B-Spline-Funktionen im Intervall $[0, 1]$ mit den Bernstein-Polynomen überein, d.h. es gilt:

$$N_i^n = B_i^n$$

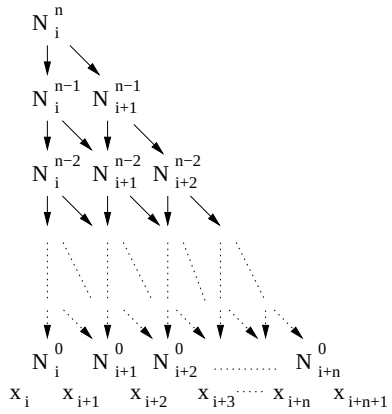


Abbildung 22: Das Trägerintervall von $N_i^n(x)$

Um eine Vorstellung davon zu bekommen, warum dies funktioniert, muss man sich überlegen, wie sich das „Fenster“ für N_i^n über diesen Knotenvektor bewegt. N_0^n beginnt am Anfang des Knotenvektors und erstreckt sich insgesamt über $n+2$ Elemente. Dies bedeutet aber, dass das Fenster alle Nullen enthält und die erste Eins. Für N_0^2 würde es also aussehen, wie in Abbildung 21(b). Es setzen sich nur die abfallenden Funktionen durch, so dass die Kurve am Ende so aussieht, wie die Kurve von B_0^2. Eine quadratische Funktion, die für $x = 0$ den Wert 1 annimmt und dann auf 0 abfällt. Das Fenster wandert dann über den Knotenvektor, wobei immer mehr Einsen und immer weniger Nullen an Einfluß gewinnen und sich so alle anderen Polynome bilden. Zum Schluss beinhaltet das Fenster die letzte Null und alle Einsen, woraus die Kurve für B_i^n entsteht.

Die Konsequenz daraus ist, dass man also alle Kurven, welche mit Bezierkurven darstellbar sind, auch mit B-Spline-Funktionen darstellen kann. Wie wir im Abschnitt über B-Spline-Kurven sehen werden, ist es dementsprechend relativ simpel, eine Bezier-Kurve in eine B-Spline-Kurve zu überführen.

Frage 3.29: Welche Eigenschaften besitzen B-Spline-Funktionen?
Die B-Spline-Funktionen besitzen unter anderem die folgenden fünf Eigenschaften:

Trägerintervall: Das Trägerintervall, d.h. der Bereich in dem die Funktion von 0 verschieden ist, ist $[x_i, x_{i+n+1}[$. Dies kann man sich recht schnell anhand von Abbildung 22 klarmachen. Auf jeder Stufe, die man sich von $N_i^n(x)$ nach unten bewegt, wächst i um eins. Dies macht man so lange, bis man bei N_{i+n}^0 angekommen ist. Und dieses ist genau für ein x zwischen x_{i+n} und x_{i+n+1} von 0 verschieden.

Partition der 1: Diese Eigenschaft wird später bei den B-Spline-Kurven benutzt, um die einzelnen Kontrollpunkte gleichmäßig zu gewichten. Die Formel lautet:

$$\sum_{i=0}^{k-n-1} N_i^n(x) = 1, x \in [x_n, x_{k-n}[$$

Es ist wichtig zu beachten, dass diese Eigenschaft also nicht für alle x gilt.

Für welche x diese Eigenschaft gilt, hängt, wie in der Formel zu ersehen ist, vom Grad der B-Spline-Funktionen ab.

Äquidistante Knotenvektoren: Ist der Knotenvektor äquidistant, wie z.B. im Fall $\xi = (0, 1, \ldots, k)$, sind die $N_i^n(x)$ bis auf eine Verschiebung entlang der Parameterachse identisch. Für den oben genannten Knotenvektor ξ gilt dann also:

$$N_i^n(x) = N_{i+1}^n(x+1)$$

Differenzierbarkeit: Sind die Knoten x_i des Knotenvektors paarweise verschieden, so ist $N_i^n(x)$ $(n-1)$-mal stetig differenzierbar. Es gilt also $N_i^n(x) \in C^{(n-1)}$. Tritt ein Knoten x_i mehrfach auf, so vermindert sich der Grad der Differenzierbarkeit in diesem Punkt. Tritt ein Knoten mit der Vielfachheit l_j auf, gilt nur noch $N_i^n(x) \in C^{n-l_j}$. Diese Formel kann man sich recht schnell merken, wenn man sich klar macht, dass im oben genannten Fall jeder Knoten nur in der Vielfachheit 1 auftaucht.

Frage 3.30: Wie ist eine offene B-Spline-Kurve definiert?

B-Spline-Kurven funktionieren so ähnlich wie Bezier-Kurven. Auch hier werden Kontrollpunkte gewichtet aufsummiert, um einen Punkt der Kurve zu berechnen. Im Falle der B-Spline-Kurven sind diese Gewichtungsfunktionen, wie der Name schon sagt, die B-Spline-Funktionen. Zusätzlich zu den Kontrollpunkten muss also noch ein Knotenvektor definiert werden. Wie dieser aussieht, hängt vom Grad n der verwendeten B-Spline-Funktionen N_i^n und von der Anzahl m der Kontrollpunkte ab.

Nun zur Definition einer B-Spline-Kurve im $\mathbb{R}^d$. Seien $(m+1)$ Kontrollpunkte $d_0, d_1, \ldots, d_m \in \mathbb{R}^d$ gegeben. Diese Kontrollpunkte werden auch de-Boor-Punkte genannt. Dann wird durch

$$b(t) = \sum_{i=0}^{m} d_i \cdot N_i^n(t), t \in [t_0, t_{m+1}[$$

eine B-Spline-Kurve gegeben. Dabei sieht der Kontrollvektor ξ wie folgt aus:

$$\xi = (t_0, t_1, \ldots, t_{n+m+1}) \text{ mit } t_0 = \cdots = t_n \text{ und } t_{m+1} = \cdots = t_{n+m+1}$$

Die Kurve ist für t aus dem Intervall $[t_0, t_{m+1}[$ definiert, weil die letzten $(n+1)$ Werte des Knotenvektors ja gleich sein sollen. Man könnte genau so gut das Intervall $[t_0, t_{n+m+1}[$ nehmen oder auch $[t_n, t_{m+1}[$.

Wie kann man sich die ganzen Indizes nun leicht merken? Wie das Aussehen des Knotenvektors? Wir haben $(m+1)$ Kontrollpunkte $d_0, \ldots, d_m \in \mathbb{R}^d$ gegeben. Jeder davon soll mit einer B-Spline-Funktion vom Grad n gewichtet werden. Also wird ein Knotenvektor benötigt, der so lang ist, dass ein Fenster der Größe $(n+2)$ (s.o.), genau an $(m+1)$ verschiedene Positionen geschoben werden kann. Zu Beginn steht das Fenster auf dem Anfang des Knotenvektors und braucht dementsprechend dafür schon $(n+2)$ Elemente. Dieses Fenster ist für die Gewichtung des ersten Punktes d_0 zuständig. Für die restlichen m Kontrollpunkte wird das Fenster jeweils um eine Position nach rechts geschoben

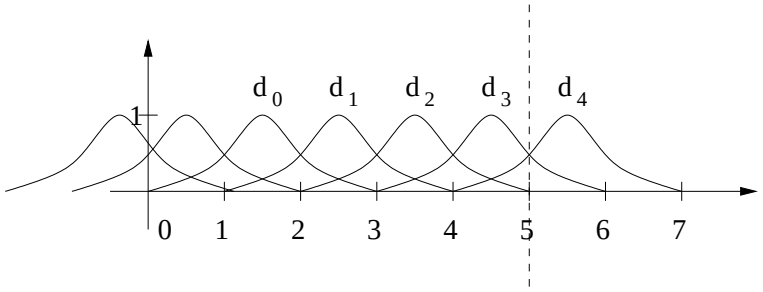


Abbildung 23: Geschlossene B-Spline-Funktionen

und bei jedem mal kommt ein neuer Punkt hinzu. Insgesamt muss der Knotenvektor also $(n+2)+m = n+m+2$ Werte enthalten. Da bei t_0 begonnen wird, enthält der Knotenvektor also Werte von t_0 bis $t_{(n+m+2)-1} = t_{n+m+1}$.

Frage 3.31: Warum setzt man in der obigen Definition einer offenen B-Spline-Kurve die ersten bzw. letzten (n+1) Knotenvektoreinträge auf gleiche Werte?

Bei der obigen Definition handelt es sich um eine **nicht-uniforme** B-Spline-Kurve. Das bedeutet, dass die Einträge des Knotenvektors keinen gleichmäßigen Abstand voneinander haben. Dies liegt einfach daran, dass einige Einträge gleich sind und dementsprechend die Differenz zwischen diesen 0 beträgt.

Der Grund für diese Wahl ist der, dass die Kurve in den Anfangs- und Endpunkten beginnen soll. Um dies zu erreichen, muss dafür gesorgt werden, dass $N_0^n(t_0) = 1$ und $N_i^n(0) = 0$ für $i = 1, \ldots, m$ bzw. $N_m^n(t_{m+1}) = 1$ und $N_i^n(t_{m+1}) = 0$ für $i = 0, \ldots, m-1$. Wieder einmal verdeutlicht Abbildung 21(b) auf Seite 35, warum dies mit einem solchen Knotenvektor klappt. Für $N_0^n(t)$ ist das Trägerintervall $[x_0, x_{n+1}[$. Die Einträge $x_0, \ldots, x_n$ sind jedoch alle gleich gewählt, so dass sich nur abfallende Fuktionen durchsetzen und durchmultiplizieren. Insgesamt handelt es sich also um eine abfallende Funktion im Intervall $[x_0, x_{n+1}[$, welche bei 1 beginnt. Verschiebt man das Fenster nun, vermischen sich ansteigende und abfallende Funktionen, bis man schließlich für $N_m^n(t)$ den Fall erreicht, wo sich nur noch ansteigende Funktionen durchsetzen.

Frage 3.32: Wie ist eine geschlossene B-Spline-Kurve definiert?

Die Konstruktion einer geschlossenen B-Spline-Kurve wird zuerst allgemein angegeben und dann anhand eines Beispiels verdeutlicht. Im Beispiel wird davon ausgegangen, dass 5 Kontrollpunkte verwendet werden ($m = 4$), die B-Spline-Funktionen den Grad $n = 2$ besitzen und der Knotenvektor $\xi = (0, 1, 2, 3, 4, 5, 6, 7)$ ist. Abbildung 23 zeigt die entsprechenden B-Spline-Polynome, wobei über jedem Polynom der Punkte steht, den es gewichtet. Der Punkt d_4 übt zum Beispiel nur im Intervall $[4, 7[$ Einfluss aus. Oder anders gesagt, der Punkt d_4 wirkt sich nur auf das Ende der Kurve aus.

Soll die B-Spline-Kurve nun geschlossen sein, muss dafür gesorgt werden, dass der letzte Punkt (in diesem Fall d_4) und einige Punkte davor (abhängig vom verwendeten Grad n) auch Einfluss auf den Anfang der Kurve bekommen. Denn dies wäre das Verhalten, welches intuitiv von einer geschlossenen B-Spline-Kurve erwartet werden würde. Es wird nun die Definition einer geschlossenen

B-Spline-Kurve angegeben, wobei an jeder Stelle erläutert wird, warum sie so definiert ist. Eine geschlossenen B-Spline-Kurve ist definiert durch:

$$b(t) = \sum_{i=0}^{m} d_i \cdot \overline{N_i^n}(t), t \in [t_0, t_{m+1}[$$

Wir haben es also fast mit der Definition einer offenen B-Spline-Kurve zu tun. Die etwas anders definierte Gewichtungsfunktion $\overline{N_i^n}(t)$ wird benutzt, da eine Fallunterscheidung gemacht werden muss. Desweiteren muss der Knotenvektor, welcher anders definiert ist als bei den offenen Kurven, auch noch ein ganz bestimmtes Aussehen haben. Doch zuerst zu $\overline{N_i^n}(t)$. Es ist wie folgt definiert:

$$\overline{N_i^n}(t) = \begin{cases} N_i^n(t) & \text{, für } t \in [t_i, t_{m+1}[\\ N_i^n(t - t_0 + t_{m+1}) & \text{, für } t \in [t_0, t_i[\end{cases}$$

Zuerst einmal sieht man, dass die Definition für jedes $t \in [t_0, t_{m+1}[$ gültig ist. Desweiteren sollte man sich klar machen, dass $N_i^n(t-t_0+t_{m+1})$ auch geschrieben werden kann als $N_i^n(t + (t_{m+1} - t_0))$.

Man sieht ebenfalls, dass nicht das ganze Intervall $[t_0, t_{m+n+1}[$ auf die Kurve abgebildet wird, sondern lediglich das Intervall $[t_0, t_{m+1}[$. Allerdings werden durch die Definition von $\overline{N_i^n}(t)$ trotzdem $N_i^n(t)$ mit $t \in [t_{m+1}, t_{n+m+1}[$ berechnet. Dies geschieht genau dann, wenn für ein $\overline{N_i^n}(t)$ das t aus dem Intervall $[t_0, t_i[$ kommt.

Das Ganze nun noch einmal etwas konkreter anhand des Beispiels. Für dieses ergibt sich die Kurve wie folgt:

$$b(t) = \sum_{i=0}^{4} d_i \cdot \overline{N_i^2}(t), t \in [0, 5[$$

Dabei ist $\overline{N_i^2}(t)$ entsprechend definiert als:

$$\overline{N_i^2}(t) = \begin{cases} N_i^2(t) & \text{, für } t \in [t_i, 5[\\ N_i^2(t - 0 + 5) = N_i^2(t + 5) & \text{, für } t \in [0, t_i[\end{cases}$$

Die Intervallgrenze t_i, an der diese Fallunterscheidung vorgenommen wird, ist dabei entsprechend vom Punkt d_i abhängig. Sie liegt immer am Anfang des jeweiligen Trägerintervalls. Für den Punkt d_4 ist dies z.B. der Wert 4.

Durch den unteren Fall in der Fallunterscheidung wird nun dafür gesorgt, dass ein Punkt, welcher am Ende der Kurve liegt, auch Einfluss auf den Anfang der Kurve hat. Betrachten wir dazu noch einmal den Punkt d_4. Liegt das t im Intervall $[0, 4[$, so wird nicht $N_4^2(t)$ ausgerechnet, sondern $N_4^2(t + 5)$, d.h. der Parameterwert t wird quasi auf die rechte Seite von t_{m+1} verschoben (im Beispiel ist $t_{m+1} = t_5 = 5$), so dass u.a. der Punkt d_m (im Beispiel d_4) auf einmal Einfluss für diesen Parameterwert bekommt.

Wie muss der Knotenvektor nun aussehen? Wenn der Parameterwert in einigen Fällen verschoben wird, muss dafür gesorgt werden, dass die Polynome nicht auf einmal radikal anders aussehen, als dies am Anfang des Knotenvektors der Fall ist. Dies bedeutet, dass die Kurve z.B. stetig bleiben soll. Dies wird

dadurch erreicht, dass die Differenz zwischen den Knotenwerten t_{m+1} und t_{m+2} genau so groß gemacht wird, wie die zwischen t_0 und t_1, die zwischen t_{m+2} und t_{m+3} genau so groß, wie die zwischen t_1 und t_2, usw. Oder etwas formaler:

$$\xi = (t_0, \ldots, t_{m+n+1}), t_{m+2} = t_{m+1} + (t_1 - t_0), \ldots, t_{m+n+1} = t_{m+n} - (t_n - t_{n-1})$$

Das dies das oben Geforderte erfüllt, ist schnell einzusehen:

$$t_{m+2} = t_{m+1} + (t_1 - t_0) \Leftrightarrow t_{m+2} - t_{m+1} = t_1 - t_0$$

Die Werte t_0 bis t_{m+1} können dabei beliebig gewählt werden.

Frage 3.33: Welche Eigenschaften besitzen B-Spline-Kurven?
B-Spline-Kurven beitzen u.a. die folgenden sechs Eigenschaften, wovon einige schon von den Bezier-Kurven bekannt sein dürften:

Verhalten in den Endpunkten: Bei einer offenen B-Spline-Kurve wird durch die Wahl des Knotenvektors dafür gesorgt, dass die Kurve in den Anfangs- und Endpunkten mit dem Kontrollpunktpolygonzug übereinstimmt.

Konvexe Hülle: Jede B-Spline-Kurve liegt in der konvexen Hülle der de-Boor-Punkte. Diese Eigenschaft ergibt sich aus dem Algorithmus von de-Boor.

Beschränkte Schwankung: Keine Gerade schneidet eine B-Spline-Kurve öfter als den Kontrollpolygonzug.

Lokalität: Bei Veränderung eines Kontrollpunktes wird die Kurve nur lokal geändert. Wird der Punkt d_i manipuliert, so ändert sich die Kurve nur für $t \in [t_i, t_{i+n+1}[$, also über dem Trägerintervall. Dies ist auch leicht einzusehen, da die zu diesem Punkt gehörende B-Spline-Funktion nur über diesem Intervall von Null verschieden ist.

Mehrfachstützstellen: Mehrfachstützstellen ziehen die Kurve an. Die Differenzierbarkeit in solchen Stellen nimmt ab. Mit genügend Kontrollpunkten an einer Stelle ist es sogar möglich, Spitzen zu erzeugen.

Geradenstücke: Geradenstücke können durch kollineare Stützstellen erzeugt werden.

Frage 3.34: Wie funktioniert der Algorithmus von de-Boor?
Der Algorithmus von de-Boor (Algorithmus 2) funktioniert so ähnlich wie der Algorithmus von de-Casteljau bei Bezier-Kurven. Auch hier werden Strecken zwischen verschiedenen Punkten immer wieder unterteilt, so dass neue Punkte entstehen, zwischen welchen wiederum unterteilt wird, bis der endgültige Punkt berechnet ist. Jedoch ist es bei B-Spline-Kurven so, dass nicht mehr alle Kontrollpunkte Einfluss auf den Verlauf der Kurve nehmen. Daher muss zuerst ermittelt werden, welche der Kontrollpunkte überhaupt Einfluss auf die Kurven ausüben. Sind diese Punkte ermittelt, wird das Unterteilungsverfahren angewandt, bis der Punkt errechnet wurde.

Was wird bei diesem Algorithmus nun genau gemacht? Um den Algorithmus besser zu verstehen, kann es sehr hilfreich sein, Abbildung 24 zu betrachten. Zu

Algorithmus 2 Der Algorithmus von de-Boor für offene B-Spline-Kurven

berechne den Index i, so dass $t_i \leq t^* < t_{i+1}, 0 \leq i \leq m-1$
for $j = 0$ **to** n **do**
  for $l = i - n + j$ **to** i **do**
    if $j = 0$ **then**
      $d_l^0 = d_l$ {nehme sofort die Punkte}
    else
      $t_l^j = (t^* - t_l)/(t_{l+n+1} - t_l)$;
      $d_l^j = (1 - t_l^j) \cdot d_{l-1}^{j-1} + t_l^j \cdot d_l^{j-1}$
    end if
  end for
end for
$b(t^*) = d_i^n$

Beginn wird geschaut, wo im Knotenvektor sich das t^* des zu berechnenden Punktes befindet. Es wird das größte i gesucht, so dass der Eintrag t_i kleiner oder gleich dem t^* ist und der nächste Eintrag t_{i+1} größer. Ist dieses i gefunden, wird den Großteil des Algorithmus zwei **for**-Schleifen durchlaufen.

Da das j bei 0 startet, kann man sich schnell klarmachen, dass in diesem Fall der obere Teil der **if**-Schleife ausgeführt wird. Es werden also einige d_l^0 gleich d_l gesetzt. In diesem Schritt wird also quasi eine Startmenge von Punkten initialisiert, auf denen später gearbeitet wird. Dies sind $n+1$ Punkte, da das l im Falle von $j = 0$ die Werte $i - n$ bis i durchläuft.

Frage 3.35: Welche Laufzeit hat der Algorithmus von de-Boor?
Die Laufzeit des Algorithmus von de-Boor beträgt $O(n^2 + m)$. Zuerst muss der Knotenvektor nach dem Index i durchsucht werden, so dass $t_i \leq t^* < t_{i+1}$ für $0 \leq i \leq m-1$ gilt. Das Durchlaufen des Knotenvektors benötigt Zeit $O(m)$, obwohl dies wahrscheinlich mit Hilfe der binären Suche auf $O(\log_2 m)$ gedrückt werden könnte. Der Grund dafür, dass das Durchsuchen des Knotenvektors mit $n + m + 2$ Elementen nur $O(m)$ dauert und nicht $O(n + m)$ ist, dass die ersten und letzten $n + 1$ Einträge identisch sind. Anschließend werden $n + 1$ Knoten initialisiert und aus diesen in jeder Runde neue Knoten berechnet, wobei die Anzahl dieser Knoten jedes Mal um Eins abnimmt, bis schließlich nur noch ein Knoten übrig ist. Insgesamt also:

$$\sum_{i=1}^{n+1} i = \frac{(n+1)(n+2)}{2} = O(n^2)$$

3.4 Geometrische Transformation

Frage 3.36: Wie ist eine affine Abbildung definiert?
Eine affine Abbildung für einen Punkt p ist wie folgt definiert, wobei $\overline{p}$ die affine Abbildung des Punktes ist:

$$\overline{p} = A \cdot p + t, \text{ mit } A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, a_{ij} \in \mathbb{R} \text{ und } t = \begin{pmatrix} t_1 \\ t_2 \end{pmatrix}, t_i \in \mathbb{R}$$

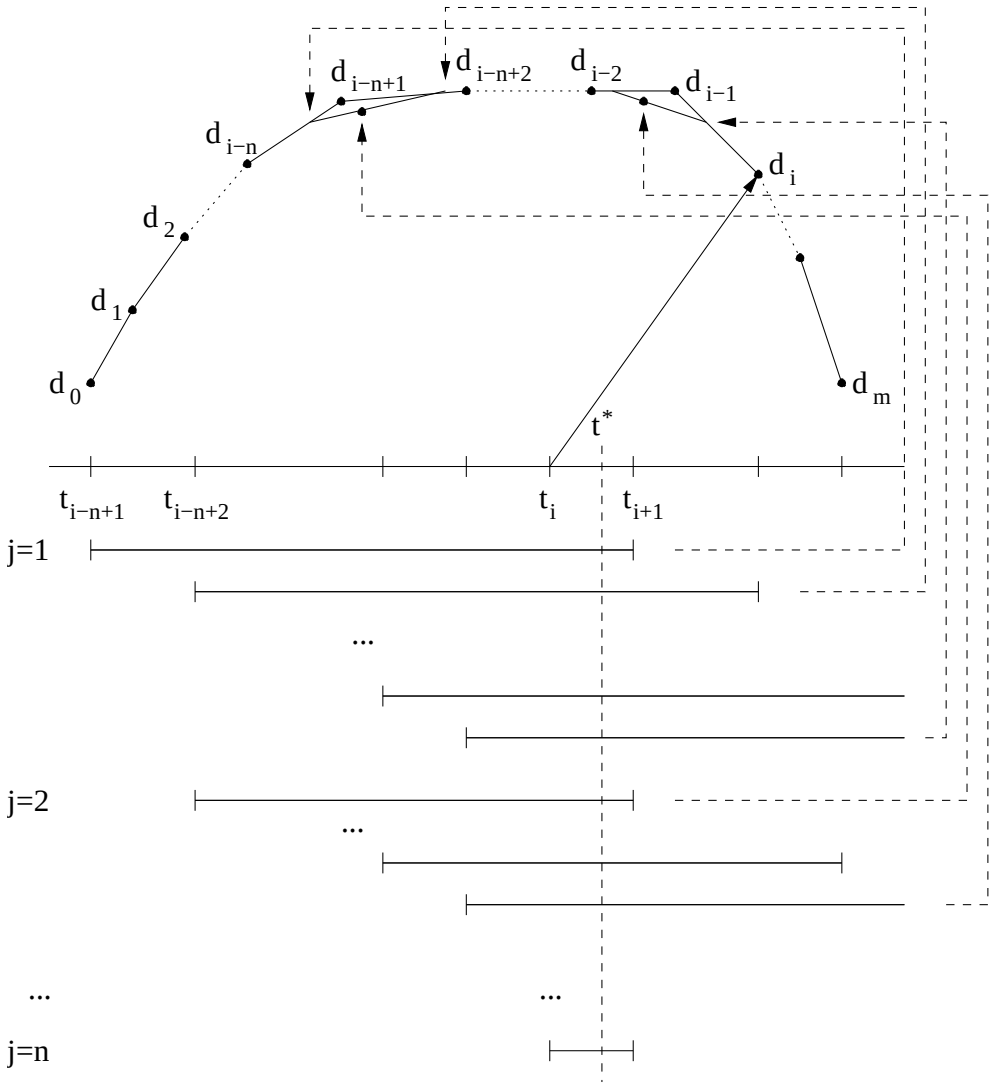


Abbildung 24: Graphische Darstellung des de-Boor Algorithmus

Frage 3.37: Welche Spezialfälle von affinen Abbildungen kennst du?
Unter anderem gibt es die folgenden fünf Spezialfälle:

Translation: Bei der Translation wird ein Punkt lediglich um einen Vektor t verschoben:

$$\overline{p} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \cdot p + t$$

Skalierung: Mit Hilfe der Skalierung können Objekte in x und y-Richtung skaliert werden:

$$\overline{p} = \begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix} \cdot p = \begin{pmatrix} a \cdot p_x \\ b \cdot p_y \end{pmatrix}, a, b \geq 0$$

Rotation: Mit Hilfe dieser Abbildung werden Punkte um den Ursprung rotiert:

$$\overline{p} = \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix} \cdot p$$

Spiegelung: Man kann drei Arten von Spiegelungen unterscheiden: an der waggerechten Achse, an der senkrechten Achse und am Ursprung (welche auch als Spiegelung an der waagerechten Achse, gefolgt von einer Spiegelung an der senkrechten Achse, interpretiert werden kann).

waagerechte Achse: $\overline{p} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \cdot p = \begin{pmatrix} p_x \\ -p_y \end{pmatrix}$

senkrechte Achse: $\overline{p} = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \cdot p = \begin{pmatrix} -p_x \\ p_y \end{pmatrix}$

Ursprung: $\overline{p} = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix} \cdot p = \begin{pmatrix} -p_x \\ -p_y \end{pmatrix}$

Scherung: Eine Scherung kann entlang der waagerechten oder senkrechten Achse geschehen:

waagerechte Achse: $\overline{p} = \begin{pmatrix} 1 & s \\ 0 & 1 \end{pmatrix} \cdot p = \begin{pmatrix} p_x + s \cdot p_y \\ p_y \end{pmatrix}$

senkrechte Achse: $\overline{p} = \begin{pmatrix} 1 & 0 \\ s & 1 \end{pmatrix} \cdot p = \begin{pmatrix} p_x \\ s \cdot p_x + p_y \end{pmatrix}$

Abbildung 25 zeigt eine Schwerung entlang der waagerechten Achse. Die y-Koordinate eines Punktes bleibt unverändert. Die x-Koordinate eines Punktes ändert sich dagegen um so stärker, je weiter der Punkt von der x-Achse entfernt ist.

Frage 3.38: Wie sieht das affine Bild einer Kurve aus? Wie kann die affine Abbildung eines Bezier-Kurvensegments berechnet werden?
Sei eine Kurve $k(t)$ und eine affine Abbildung $\overline{p} = A \cdot p + t$ gegeben. Dann ist das affine Bild der Kurve $\overline{k}(t)$ gegeben als:

$$\overline{k}(t) = A \cdot k(t) + t$$

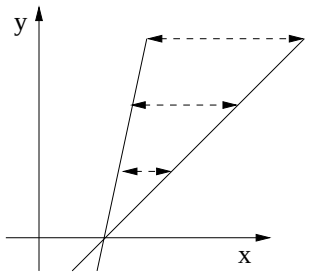


Abbildung 25: Scherung entlang der waagerechten Achse

Es wird also der Punkt der Kurve berechnet und dieser abgebildet. Im Falle der Bezierkurven kann das affine Bild der Kurve jedoch sehr leicht berechnet werden. Es reicht nämlich, das affine Bild des Kontrollpolygonzugs zu berechnen. Sei die Bezierkurve $b(t) = \sum_{i=0}^{n} b_i B_i^n(t)$ und eine affine Abbildung $\overline{p} = A \cdot p + t$ gegeben. Dann ist das affine Bild $\overline{b}(t)$ der Kurve gegeben als:

$$\overline{b}(t) = \sum_{i=0}^{n} \overline{b_i} \cdot B_i^n(t) \text{ mit } \overline{b_i} = A \cdot b_i + t$$

Frage 3.39: Wie lautet die homogene Darstellung affiner Abbildungen?
Die affinen Abbildungen, so wie sie etwas weiter oben definiert wurden, haben den Nachteil, dass zur Ausführung der Translation eine Vektoraddition durchgeführt werden muss, während alle anderen Abbildungen durch Matrixmultiplikationen realisiert werden. Es wäre angenehmer, wenn alle diese Operationen einheitlich dargestellt werden könnten. Damit dies möglich ist, wird die homogene Darstellung eingeführt. Die homogene Darstellung einer affinen Abbildungen $\overline{p} = A \cdot p + t$ lautet:

$$\overline{p}^* = A^* \cdot p^* \text{ mit } \overline{p}^* = \begin{pmatrix} \overline{p} \\ 1 \end{pmatrix}, p^* = \begin{pmatrix} p \\ 1 \end{pmatrix}, A^* = \begin{pmatrix} A & t \\ 0 & 1 \end{pmatrix}$$

Im Prinzip bedeutet dies also:

$$\begin{pmatrix} A & t \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} p \\ 1 \end{pmatrix} = \begin{pmatrix} A \cdot p + t \\ 1 \end{pmatrix}$$

Frage 3.40: Worin besteht der Vorteil der homogenen Darstellung?
Der erste Vorteil besteht darin, dass durch die homogene Darstellung nun alle affinen Abbildungen, einschließlich der Translation, durch eine Matrixmultiplikation dargestellt werden. Die Multiplikation zweier Matrizen kann dabei effizient in Hardware realisiert werden. Ein weiterer Vorteil besteht darin, dass die Hintereinanderausführung mehrerer affiner Abbildungen durch eine Matrixmultiplikation bewerkstelligt werden kann. Sollen mehrere Punkte einer Folge von affinen Abbildungen unterworfen werden, so werden zuerst die Matrizen der affinen Abbildungen multipliziert und anschließend jeder Punkt mit der Ergebnismatrix multipliziert. Denn für zwei affine Abbildungen

$$\begin{aligned} T_1 &: \overline{p} = A_1 \cdot p + t_1 \\ T_2 &: \overline{p} = A_2 \cdot p + t_2 \end{aligned}$$

gilt, dass ihre Hintereinanderausführung wie folgt aussieht:

$$\begin{aligned} T_3 := T_2 \circ T_1 : \overline{p} &= A_2 \cdot (A_1 \cdot p + t_1) + t_2 \\ &= \underbrace{A_2 \cdot A_1}_{A_3} \cdot p + \underbrace{A_2 \cdot t_1 + t_2}_{t_3} \\ &= A_3 \cdot p + t_3 \end{aligned}$$

Dasselbe Ergebnis liefert die Matrizenmultiplikation der Matrizen der homogenen Darstellung der Abbildungen, denn hier gilt:

$$\begin{aligned} \begin{pmatrix} A_2 & t_2 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} A_1 & t_1 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} p \\ 1 \end{pmatrix} &= \begin{pmatrix} A_2 \cdot A_1 & A_2 \cdot t_1 + t_2 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} p \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} A_3 & t_3 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} p \\ 1 \end{pmatrix} \end{aligned}$$

3.5 Kompositionsverfahren

Frage 3.41: Welche Arten von Kompositionsverfahren kennst du?

- Menge aus Elementarbestandteilen
- Hierarchische Gruppierung
- Syntaktische Beschreibung

Frage 3.42: Wie funktioniert die Szenendarstellung als Menge von Elementarbestandteilen?

Bei diesem Verfahren besteht die Szene einfach aus einer Menge von Graphikelementen, die dem Graphiksystem bekannt sind (z.B. Punkte, Strecke, Polygonzug, etc.). Es kann also in der folgenden Form beschrieben werden:

$$\begin{aligned} \textit{Szene} &:= \{\textit{Graphikelement}, \ldots, \textit{Graphikelement}\} \\ \textit{Graphikelement} &:= \text{Strecke|Polygonzug|Markierung|} \ldots \end{aligned}$$

Frage 3.43: Wie funktioniert die Szenendarstellung als hierarchische Gruppierung?

Bei diesem Verfahren besteht die Szene aus einer Gruppe, die einer (affinen) Transformation unterworfen wird. Eine Gruppe wiederum besteht aus mehreren Bestandteilen, wobei ein Bestandteil entweder wieder eine Gruppe ist, die einer (affinen) Transformation unterworfen wird oder aber ein Graphikelement, welches dem Graphiksystem bekannt ist. Als Grammatik hingeschrieben sieht dies wie folgt aus:

$$\begin{aligned} \textit{Szene} &:= \textit{TransformationGruppe} \\ \textit{Gruppe} &:= \{\textit{Bestandteil}, \ldots, \textit{Bestandteil}\} \\ \textit{Bestandteil} &:= \textit{TransformationGruppe} | \textit{Graphikelement} \\ \textit{Transformation} &:= \text{eine affine Transformation} \\ \textit{Graphikelement} &:= \text{Strecke|Polygonzug|Markierung|} \ldots \end{aligned}$$

Durch die Verwendung von Gruppen ist es möglich komplexe Szenen relativ einfach zu beschreiben. Um zum Beispiel einen Roboter mit zwei gleichartigen Armen zu konstruieren, reicht es aus, den Arm einmal zu modellieren und zu einer Gruppe zusammenzufassen. Der zweite Arm kann dann einfach erzeugt werden, indem der ursprüngliche Arm durch eine affine Transformation, in Form einer Spiegelung entlang der Körperachse, modelliert wird.

Frage 3.44: Was versteht man unter CSG?
CSG steht für „Constructive Solid Geometry“ und dient zur Modellierung von Festkörpern. Festkörper sind Körper, bei denen man für jeden Punkt eindeutig sagen kann, ob er im Inneren, auf der Oberfläche oder außerhalb des Körpers liegt.

Beim Verfahren des CSG geht man von einer Menge von **regulären** Grundkörpern aus, welche durch Anwendung von **regularisierten** Mengenoperationen zu komplexeren Körpern kombiniert werden können.

Frage 3.45: Wann heißt ein Körper regulär?
Ein Körper K heißt regulär, wenn er mit dem Abschluss $(K^o)^-$ seines offenen Kerns K^o übereinstimmt.

Frage 3.46: Was ist der offene Kern eines Körpers?
Der offene Kern K^o eines Körpers ist die Menge aller Punkte, die eine zur d-dimensionalen Einheitskugel äquivalente Umgebung besitzen, die ebenfalls im Körper liegt. Abbildung 26 deutet den offenen Kern eines Kreises an.

Frage 3.47: Was ist der Abschluss einer Punktmenge?
Der Abschluss M^- einer Punktmenge M im $\mathbb{R}^d$ besteht in der Hinzunahme der Randpunkte. Randpunkte sind alle Punkte im $\mathbb{R}^d$, die nicht zum offenen Kern der betrachteten Punktmenge gehören, für die jedoch jede zur d-dimensionalen Einheitskugel äquivalente Umgebung Körperpunkte enthält.

Den Abschluss kann man zum Beispiel anhand eines Kreises erklären, der als die Menge aller Punkte definiert ist, für deren Abstand a zum Mittelpunkt gilt: $0 \leq a < r$. Würde man diesen Kreis mit einem unendlich dünnen Stift und einem Radius von r zeichnen, so würde die gezeichnete Linie nicht mehr zum offenen Kern des Kreises gehören. Würde man jetzt jedoch den Abschluss des Kreises bilden, so würde auch dieser Rand hinzukommen.

Frage 3.48: Was sind regularisierte Mengenoperationen?
Regularisierte Mengenoperationen funktionieren so ähnlich wie die „normalen“ Mengenoperationen. Mit ihrer Hilfe wird nur dafür gesorgt, dass das Ergebnis einer Mengenoperation auf Festkörpern wieder einen Festkörper ergibt.

Wenn man zum Beispiel den Schnitt zweier Würfel, die nur einen Eckpunkt gemeinsam haben, berechnet, so ergibt die normale Operation einen Punkt. Das Problem hierbei ist jedoch, dass ein Punkt kein Festkörper ist, da man für ihn nicht sagen kann, ob er innen, auf der Oberfläche oder außen von Irgendetwas liegt. Aus diesem Grund benutzt man bei der CSG die regularisierte Mengenoperationen $\cup^*, \cap^*$ und $-^*$. Diese werden mit Hilfe der normalen Mengenoperationen $\cup, \cap$ und $-$ definiert:

regularisierte Vereinigung: $A \cup^* B := ((A \cup B)^o)^-$

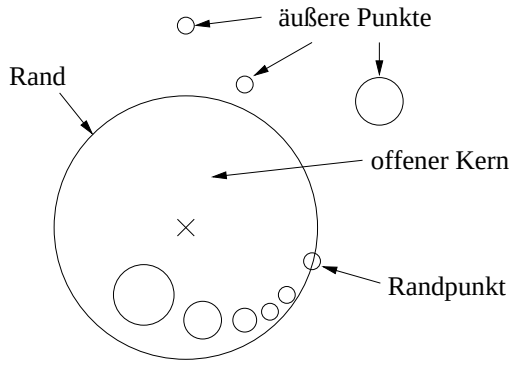
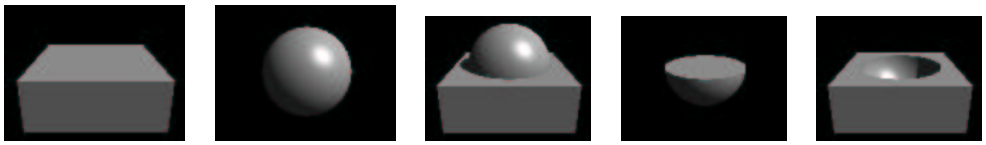


Abbildung 26: Offener Kern und Rand anhand eines Kreises



(a) Quader Q (b) Kugel K (c) $Q \cup K$ (d) $Q \cap K$ (e) $Q - K$

Abbildung 27: Die verschiedenen regularisierten Mengenoperationen

regularisierter Durchschnitt: $A \cap^* B := ((A \cap B)^o)^-$

regularisierte Differenz: $A -^* B := ((A - B)^o)^-$

Dabei ist M^o das Innere der Menge M und M^- der Abschluss der Menge M. Abbildung 27 zeigt die verschiedenen regularisierten Mengenoperationen anhand des Beispiel eines Quaders Q und einer Kugel K.

Frage 3.49: Wie werden CSG-Objekte abgespeichert?
CSG-Objekte werden als ein CSG-Baum gespeichert.

Abbildung 28: Eine typische Graphik-Pipeline

4 Clipping und Picking

4.1 Clipping

Frage 4.1: Wie sieht eine typische Graphik-Pipeline aus?
Abbildung 28 zeigt, wie eine typische Graphik-Pipeline in etwa aussieht. Zuerst werden die Szenedaten einer Transformation unterworfen, etwa wenn sich bestimmte Teile der Szene bewegt haben oder wenn die Kameraposition sich geändert hat. Anschließend erfolgt das Clipping, welches in diesem Kapitel behandelt wird. Clipping dient dazu, die Teile einer Szene auszuschneiden, die außerhalb eines Bereiches befinden. In den meisten Fällen ist dies der Bereichs des sichtbaren Bildes. Anschließend werden die Primitive, welche noch sichtbar sind verrastert und auf dem Bildschirm angezeigt?

Frage 4.2: Wie können Punkte gegen ein Rechteck geclippt werden?
Gegeben sind hier drei Punkte. Zwei von diesen bestimmen das Rechteck, gegen das der dritte Punkt geclippt werden soll. Das Rechteck ist also in Form der Punkte (x_{min}, y_{min}) und (x_{max}, y_{max})gegeben. Damit der dritte Punkt $p = (x, y)$ in diesem Rechteck liegt, müssen die folgenden vier Ungleichungen gelten:

$$x_{min} \leq x \leq x_{max} \text{ und } y_{min} \leq y \leq y_{max}$$

Frage 4.3: Welche Algorithmen kennst du, um Strecken gegen ein Rechteck zu clippen?

- Algorithmus von Cohen-Sutherland
- Algorithmus von Liang und Barsky

Frage 4.4: Wie funktioniert der Algorithmus von Cohen-Sutherland?
Soll eine Strecke gegen ein Rechteck geclippt werden, so kann zuerst geschaut werden, ob einer von zwei Sonderfällen zutrifft. In einem Fall wird die Strecke trivialerweise akzeptiert, weil beide Endpunkte sich im Inneren des Clipping-Rechtecks befinden. Im anderen Fall wird sie trivialerweise abgeleht, da sich die beiden Endpunkte beide in einer der vier durch die Rechteckskanten aufgespannten Halbebenen befinden.

Abbildung 29 zeigt die verschiedenen Fälle, die auftreten können. So kann die Strecke $[A, B]$ zum Beispiel trivialerweise akzeptiert werden, da sich die Punkte A und B im Inneren des Clipping-Rechtecks befinden, während die Strecke $[C, D]$ trivialerweise abgelehnt werden kann. Denn ihre beiden Punkte C und D liegen beide in der durch die linke Kante des Clipping-Rechtecks aufgespannten Halbebene. Die restlichen Strecken müssen näher betrachtet werden.

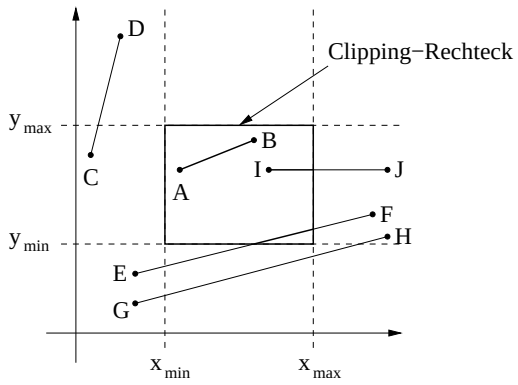


Abbildung 29: Die verschiedenen Möglichkeiten eines Schnittes

Der Algorithmus von Cohen-Sutherland teilt die potentiellen Positionen der Endpunkte in neun verschiedene Bereiche ein, welche in Abbildung 30 dargestellt sind. Diese Bereiche werden mit Hilfe von vier Bits $b_3b_2b_1b_0$ codiert, wobei jedes Bit angibt, ob ein Punkt in einer der vier Halbebenen liegt. So codiert b_3 die obere Halbebene, b_2 die untere, b_1 die rechte und b_0 die linke Halbebene. Liegt ein Punkt also innerhalb des Clipping-Rechtecks, ist keines seiner Bits gesetzt und er bekommt den Code 0000 zugewiesen. Ansonsten ist mindestens eines der Bits gesetzt.

Soll eine Strecke nun geclippt werden, so werden zuerst die Codes für die beiden Endpunkte berechnet. Wann kann ein Punkt nun trivialerweise akzeptiert werden? Wenn beide Punkte den Code 0000 zugewiesen bekommen haben. Dies lässt sich mit einer einfachen Oder-Verknüpfung der Codes der beiden Punkte überprüfen. Ist C_1 der Code des ersten Punktes und C_2 der Code des zweiten, so muss also $C_1 \vee C_2 = 0000$ gelten. Ebenso leicht kann überprüft werden, wann eine Strecke trivialerweise abgelehnt werden kann. Dies ist dann der Fall, wenn beide Endpunkte in einer der Halbebenen liegen. Oder anders gesagt, das Bit, welches die Halbebene codiert muss bei beiden gesetzt sein. Dies bekommt man mit eine Und-Verknüpfung heraus. Wenn also gilt $C_1 \wedge C_2 \neq 0000$, kann die Strecke trivialerweise verworfen werden. Kann die Strecke nicht trivialerweise akzeptiert oder verworfen werden, muss sie näher untersucht werden. In diesem Fall muss die Strecke auch mindestens eine der vier Halbebenen aufspannenden Geraden schneiden. Welche der Geraden geschnitten wird, kann wieder durch eine Betrachtung der Bereichscodes für die Punkte herausgefunden werden. Ist ein Bit bei einem der beiden Punkte gesetzt, so muss die entsprechende Gerade geschnitten werden. Ansonsten wäre die Strecke ja von vornherein abgelehnt worden. Es werden nun also, in dieser Reihenfolge, die obere, untere, rechte und linke Gerade betrachtet und der erste Punkt, bei dem eines der entsprechenden Bits gesetzt ist, durch den Schnittpunkt mit der Geraden ersetzt. Dann wird die dabei entstehende neue Gerade aufs neue betrachtet, bis eine Entscheidung getroffen werden kann.

Frage 4.5: Wie kann der Code eines Punktes berechnet werden?
Bei Verwendung des Vorzeichenbits kann der Code eines Punktes relativ einfach

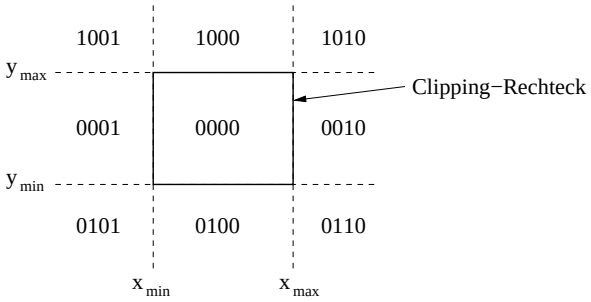


Abbildung 30: Die Kodierung der verschiedenen Bereiche

Codebit	Differenz
1. Bit	$y_{max} - y$
2. Bit	$y - y_{min}$
3. Bit	$x_{max} - x$
4. Bit	$x - x_{min}$

Tabelle 2: Berechnung der Bits des Bereichscodes

berechnet werden. Wird davon ausgegangen, dass bei einer negativen Zahl das Vorzeichenbit auf 1 gesetzt ist, kann der entsprechende Code jeweils durch die in Tabelle 2 aufgeführten Differenzen berechnet werden. Liegt zum Beispiel ein Punkt in der oberen Ebene, so ist sein y-Wert größer als y_{max} und somit wird die Differenz $y_{max} - y$ negativ. Dementsprechend wird das Vorzeichenbit gesetzt und für den Code das richtige Ergebnis errechnet.

Frage 4.6: Wie können die Schnittpunkte mit den Geraden berechnet werden?

Es wird hier nun davon ausgegangen, dass der Schnittpunkt mit der Gerade berechnet werden soll, die die linke Halbebene aufspannt. Diese ist durch x_{min} gegeben. Alle anderen Fällen funktionieren analog, nur dass entsprechend x_{max}, y_{min} bzw. y_{max} benutzt wird. Die Gerade sei durch die beiden Punkte $p_1 = (x_1, y_1)$ und $p_2 = (x_2, y_2)$ gegeben, wobei davon ausgegangen wird, dass p_1 der Punkt in der linken Halbebene ist. Die Steigung dieser Geraden berechnet sich als:

$$\Delta = \frac{y_2 - y_1}{x_2 - x_1}$$

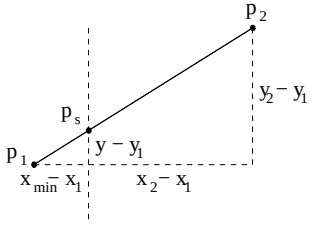


Abbildung 31: Schnittpunktberechnung beim Cohen-Sutherland Algorithmus

Diese Steigung gilt für alle Punktpaare der Geraden. Somit gilt sie auch für den Punkt in der linken Halbebene und den gesuchten Schnittpunkt $p_s(x_{min}, y)$:

$$\frac{y - y_1}{x_{min} - x_1} = \Delta \Leftrightarrow y = \Delta(x_{min} - x_1) + y_1$$

Alle Werte bis auf y sind gegeben und müssen dementsprechend nur noch eingesetzt werden. Abbildung 31 verdeutlicht die Zusammenhänge.

Frage 4.7: Wie funktioniert der Algorithmus von Liang und Barsky?
Der Algorithmus von Liang und Barsky benutzt die Darstellung einer Strecke $s[P_1, P_2]$ in Parameterform. Sind die beiden Punkte $P_1 = (x_1, y_1)$ und $P_2 = (x_2, y_2)$ gegeben, dann ist die Strecke für $t \in [0, 1]$ gegeben als:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \left[\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} - \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \right] \cdot t$$

Dies kann aufgeteilt werden in:

$$\begin{aligned} x &= x_1 + \Delta x \cdot t \quad \text{mit} \quad \Delta x = x_2 - x_1 \\ y &= y_1 + \Delta y \cdot t \quad \text{mit} \quad \Delta y = y_2 - y_1 \end{aligned}$$

Für $t \in [0, 1]$ wird genau die Strecke beschrieben und für $t \in]-\infty, \infty[$ eine Gerade durch die beiden Punkte P_1 und P_2. Für welche t liegt diese Gerade nun im durch x_{min}, x_{max} und y_{min}, y_{max} begrenzten Fenster? Es wird dabei davon ausgegangen, dass die vier Kanten des Clipping-Rechtecks die Ebene jeweils in zwei Halbebenen teilen. Die Halbebene, in der sich das Clipping-Rechteck befindet, wird dabei als sichtbare Halbebene bezüglich einer der Kanten bezeichnet. Aus dem Durchschnitt dieser vier Halbebenen wiederum ergibt sich das Clipping-Rechteck. Damit die Gerade durch die Punkte P_1 und P_2 bezüglich einer Kante in der sichtbaren Halbebene liegt, müssen folgende Ungleichungen erfüllt sein:

$$\begin{aligned} x_1 + \Delta x \cdot t \geq x_{min} &\Leftrightarrow -\Delta x \cdot t \leq x_1 - x_{min} \\ x_1 + \Delta x \cdot t \leq x_{max} &\Leftrightarrow \Delta x \cdot t \leq x_{max} - x_1 \\ y_1 + \Delta y \cdot t \geq y_{min} &\Leftrightarrow -\Delta y \cdot t \leq y_1 - y_{min} \\ y_1 + \Delta y \cdot t \leq y_{max} &\Leftrightarrow \Delta y \cdot t \leq y_{max} - y_1 \end{aligned}$$

Allgemein handelt es sich also um Gleichungen der Form $p_i \cdot t \leq q_i, i = 1, \ldots 4$. Dabei sind die einzelnen p_i bzw. q_i gegeben als:

$$\begin{aligned} p_1 &= -\Delta x & \qquad q_1 &= x_1 - x_{min} \\ p_2 &= \Delta x & \qquad q_2 &= x_{max} - x_1 \\ p_3 &= -\Delta y & \qquad q_3 &= y_1 - y_{min} \\ p_4 &= \Delta y & \qquad q_4 &= y_{max} - y_1 \end{aligned}$$

Das t für eine Kante lässt sich als q_i/p_i berechnen. Dabei muss jedoch beachtet werden, dass $p_i \neq 0$ gilt, d.h. die Strecke darf nicht parallel zur oberen und unteren bzw. zur linken und rechten Begrenzungskante des Clipping-Rechtecks liegen. Aus diesen einzelnen p_i und q_i lassen sich nun aber noch einige weitere

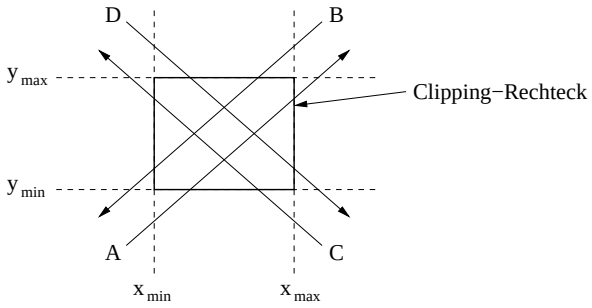


Abbildung 32: Die verschiedenen Orientierungsmöglichkeiten einer Strecke

Informationen über das Aussehen der Strecke bzw. der Gerade gewinnen. So gibt das p_i zum Beispiel an, ob die entsprechende Kante des Clipping-Rechteck ein potentieller Eintritts- oder Austrittspunkt ist. Für $p_i < 0$ ist die Kante ein potentieller Eintrittspunkt und für $p_i > 0$ ein potentieller Austrittspunkt. Dies hängt mit der Orientierung der Strecke zusammen, welche dadurch gegeben ist, wie P_1 und P_2 zueinander liegen. Liegt z.B. P_2 bezüglich der y-Achse oberhalb von P_1 so verläuft die Strecke von unten nach oben. Dies ist z.B. bei Strecke A und C in Abbildung 32 der Fall. Die beiden Strecken A und C unterscheiden sich in ihrer Orientierung weiterhin nur noch darin, ob sie von links nach rechts oder anderherum orientiert sind. Wie man sich anhand der Abbildung ebenfalls klar machen kann, ist durch die vollständige Orientierung auch eindeutig festgelegt, welche Kanten Eintritts- und Austrittspunkte sein können. Betrachtet man zum Beispiel Strecke A, so können nur die linke oder die untere Kante Eintrittspunkte sein, während die beiden anderen Kanten nur Austrittspunkte sein können. Und mathematisch schlägt sich dies halt im p_i nieder.

Das q_i gibt an, ob sich der Punkt P_1 im sichtbaren oder nicht sichtbaren Bereich der entsprechenden Halbebene befindet. Gilt $q_i < 0$, so befindet sich der Punkt im der nicht sichtbaren Halbebene der entsprechenden Kante, ansonsten ($q_i \geq 0$) im sichtbaren Bereich. Das Ziel ist es nun, zwei Parameterwerte t_1 und t_2 zu berechnen, mit denen der Eintrittspunkt (t_1) und der Austrittspunkt (t_2) der Gerade berechnet werden kann. Mit den oben gemachten Beobachtungen ergeben sich diese beiden Punkte als:

$$\begin{aligned} t_1 &= \max\left(\{q_i/p_i | p_i < 0, i = 1, \ldots, 4\} \cup \{0\}\right) \\ t_2 &= \min\left(\{q_i/p_i | p_i > 0, i = 1, \ldots, 4\} \cup \{1\}\right) \end{aligned}$$

Wieso leisten die beiden Bedingungen das Geforderte? Betrachten wir die Bedingung für t_1, welches den Parameter des Eintrittspunktes angibt. Durch die Bedingung $p_i < 0$ wird dafür gesorgt, dass tatsächlich nur die t der Kanten berechnet werden, die auch tatsächlich potentielle Eintrittspunkte darstellen. Damit ist aber auch der Nenner des Bruchs q_i/p_i negativ. Ist das entsprechende q_i nun positiv, d.h. liegt der Punkt P_1 im sichtbaren Bereich bezüglich der betrachteten Kante, so wird der Bruch negativ und die Null setzt sich bei der Berechnung des Maximums durch. Dies ist auch korrekt, denn der Punkt P_1, den man für $t = 0$ erhält, befindet sich sich ja im sichtbaren Bereich. Ansonsten

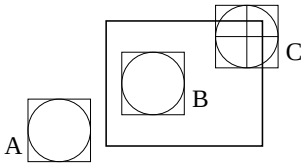


Abbildung 33: Clipping von Kreisen

wird der Bruch positiv und damit auch das berechnete t, welches sich nun gegen die Null durchsetzt. Dies ist ebenfalls korrekt, da in diesem Fall der Punkt P_1 außerhalb des sichtbaren Bereiches liegt und der Schnittpunkt höchstens mitten auf der Strecke liegen kann.

Die Berechnung des Minimums bzw. Maximums ist ebenfalls korrekt, wie man sich anhand der Zeichnung klarmachen kann. Wenn zum Beispiel eine Strecke vom Typ A ihren Eintrittpunkt auf der linken Kante des Clipping-Rechtecks hat, muss sie vorher die untere Kante (bzw. deren Verlängerung) kreuzen. Liegt der Eintrittspunkt auf der unteren Kante, so muss vorher die linke Kante gekreuzt werden.

Entsprechende Argumente können für t_2 gegeben werden. Gilt nach der Berechnung $t_1 > t_2$, so liegt die Strecke komplett außerhalb des Clippingrechtecks.

Frage 4.8: Wie können Kreise geclippt werden?

Beim Clipping von Kreisen wird zuerst einmal geschaut, ob der Kreis trivial akzeptiert oder verworfen werden kann. Dies wird gemacht, indem das achsenparallele Hüllquadrat des Kreise auf einen Schnitt mit dem Clipping-Rechteck getestet wird. Dies kann zum Beispiel mit dem Algorithmus zum Clippen von Polygonen gemacht werden. In Abbildung 33 kann Kreis A z.B. trivial abgeleht und Kreis B trivial akzeptiert werden.

Wird, wie bei Kreis C, ein Schnitt festgestellt, so wird der Kreis in vier Quadranten geteilt und für jeden einzelnen Quadranten getestet, ob er das Clipping-Rechteck schneidet. Wird für einen der Quadranten ein Schnitt festgestellt, so wird dieser berechnet und ermittelt, welche Kreisabschnitte im Inneren des Clipping-Rechtecks liegen.

Frage 4.9: Wie können Polygone an einem Rechteck geclippt werden?

Um ein Polygon an einem Rechteck zu clippen, könnte man auf die Idee kommen, die einzelnen Polygonkanten sequentiell am Clipping-Rechteck zu clippen. Bei einem solchen Vorgehen würden sich jedoch mehrere Probleme ergeben. Zum einen wären die so entstehenden Kanten nicht mehr zusammenhängend und zum anderen kann ein nicht konvexes Polygon in mehrere nicht konvexe Polygone zerfallen. Um ein Polygon also mit diesem Verfahren zu clippen, müsste man sich also zumindest eine ganze Menge merken.

Der Algorithmus von Sutherland-Hodgman geht aus diesem Grund anders vor. Anstatt für jede einzelne Polygonkante zu prüfen, an welcher Kante des Clipping-Rechtecks sie geclippt werden muss, werden die Punkte des Polygons sequentiell betrachtet und der Reihe nach gegen alle vier Kanten des Clipping-Rechtecks geclippt. Das heißt, das Polygon wird zum Beispiel erst gegen die

Fall	Ausgabe
(a)	p
(b)	i
(c)	-
(d)	i,p

Tabelle 3: Die Ausgabepunkte der vier verschienden Fälle (s. Abb. 34)

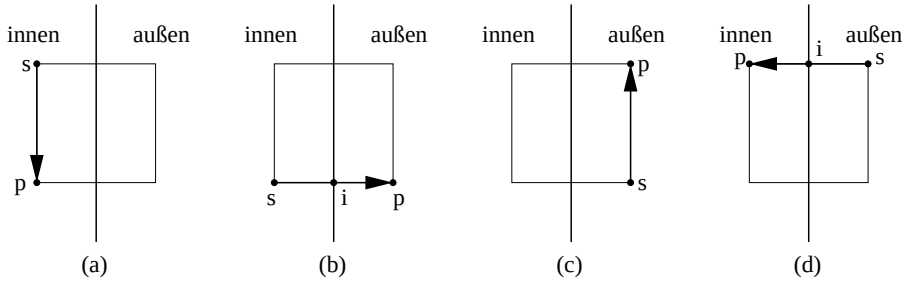


Abbildung 34: Die vier Fälle im Algorithmus von Sutherland-Hodgman

untere Kante geclippt, das Ergebnis dann gegen die rechte, usw. Es wird dabei im folgenden davon ausgegangen, dass das ursprüngliche Polygon durch die n Punkte $P_1, \ldots, P_n$ gegeben ist. Dabei gibt es $n-1$ Polygonkanten von P_i nach P_{i+1} für $i = 1, \ldots, n-1$ und eine Kante von P_n nach P_1. Der Algorithmus beginnt nun mit dem Punkt P_n und läuft über P_1 wieder bis zu P_n. Währenddessen werden die Punkte ausgegeben (oder in eine Liste gehängt), die das Ergebnispolygon repräsentieren.

In jedem Schritt wird nun ein Paar von Punkten betrachtet, wobei der erste mit s bezeichnet wird und von dem aus zum nächsten Punkt p gelaufen wird. Hierbei können nun vier Fälle unterschieden werden, in Abhängigkeit davon, wo sich die Punkte bezüglich der Clipping-Kante befinden. Wie in Abbildung 34 ersichtlich ist, können s und p entweder beide im Inneren des Clipping-Bereichs liegen, s innerhalb und p außerhalb, beide außerhalb oder s außerhalb und p innerhalb. Tabelle 3 fasst zusammen, welche Punkte in welchem der Fälle ausgegeben werden und damit zum Ergebnispolygon gehören.

Da im Prinzip viermal dasselbe gemacht wird, nur mit unterschiedlichen Eingabepunkten kann der Algorithmus als eine Pipeline realisiert werden. Dies bedeutet, dass sobald zwei neue Punkte in einer der drei ersten Phasen berechnet wurden, diese schon in der nächsten Phase gegen die nächste Kante geclippt werden können. In der vierten Phase wird in diesem Fall das Endergebnis berechnet.

4.2 Picking

Frage 4.10: Worum geht es beim Picking und wie kann es realisiert werden?
Beim Picking geht es darum, ein geometrisches Objekt durch die Identifizierung eines seiner Punkte aufzufinden. Ein Beispiel hierfür ist, wenn ein Benutzer mit

der Maus auf dem Bildschirm klickt und herausgefunden werden soll, welches Objekt er selektiert hat (z.B. in einem Zeichenprogramm).

Oft wird dies so gemacht, dass um den Bereich, in dem mit der Maus geklickt wurde, ein kleines Clipping-Rechteck gelegt wird und erst einmal alle Objekte dagegen geclippt werden. Die Objekte, welche im Clipping-Rechteck liegen bzw. davon geschnitten werden, werden dann näher untersucht.

Frage 4.11: Welche Schwierigkeiten können beim Picking auftreten?
Es kann unter anderem zu den folgenden beiden Schwierigkeiten kommen:

1. Welches Objekt wird identifiziert, wenn es sich um „dünne“ Objekte, wie zum Beispiel Linien und Punkte handelt, welche unter Umständen nicht exakt „getroffen“ werden können?

2. Wie kann festgestellt werden, ob ein Punkt zu einem Objekt, z.B. ein Polygon, gehört?

Frage 4.12: Wie können „dünne“ Objekte identifiziert werden?
Ein Objekt wird dann identifiziert, wenn es hinreichend nah am Auswahlpunkt liegt. Gibt es mehrere Objekte, wird das Objekt identifiziert, welches dem Auswahlpunkt am nächsten ist. Um die Nähe zweier Punkte zueinander festzustellen, können verschiedene Entfernungsmaße benutzt werden, welche unterschiedlich aufwändig sind.

Um die Entfernung zweier Punkte P und Q in einer Ebene zueinander zu messen, kann z.B. euklidische Abstand verwendet werden, welcher im Grunde genommen nichts anderes ist, als eine Anwendung des Pythagoras. Die Formel hierfür lautet:

$$d = \sqrt{(q_x - p_x)^2 - (q_y - p_y)^2}$$

Das ziehen der Wurzel ist jedoch schon relativ aufwändig. Möchte man nur die relativen Abstände von Punkten berechnen, kann auch auf das Ziehen der Wurzel verzichtet werden. Denn wenn für zwei Abstände $d_1 < d_2$ gilt, so wird auch $d_1^2 < d_2^2$ gelten. Alternativ kann der Abstand auch durch die folgende Formel approximiert werden:

$$d = \frac{(|q_x - p_x| + |q_y - p_y| + 2 \cdot \max(|q_x - p_x|, |q_y - p_y|))}{3}$$

Frage 4.13: Wie kann festgestellt werden, ob sich ein Punkt im Inneren eines Polygons befindet?
Ein Punkt liegt genau dann im Inneren eines Polygons, wenn ein beliebiger Strahl von diesem Punkt aus, das Polygon in ungerader Anzahl schneidet. Die algorithmische Umsetzung sieht dabei wie folgt aus:

1. Verschiebe das Polygon so, dass der Testpunkt im Ursprung liegt.

2. Wähle die positive x-Achse als Strahl.

3. Durchlaufe die Kanten und finde anhand der y-Koordinaten der Endpunkte heraus, ob sie den Strahl schneiden.

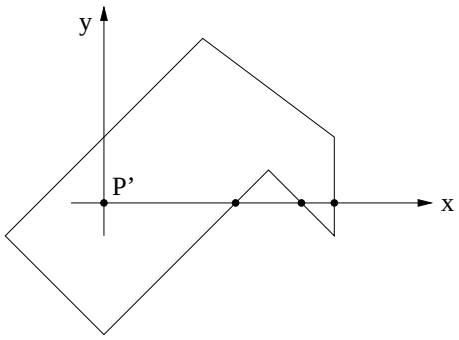


Abbildung 35: Überprüfung ob ein Punkt P sich im Inneren eines Polygons befindet

4. Sonderfälle werden dadurch vermieden, dass für Punkte, die auf dem Strahl liegen, angenommen wird, dass sie oberhalb des Strahls liegen.

Dieser Algorithmus ist in Abbildung 35 graphisch dargestellt.

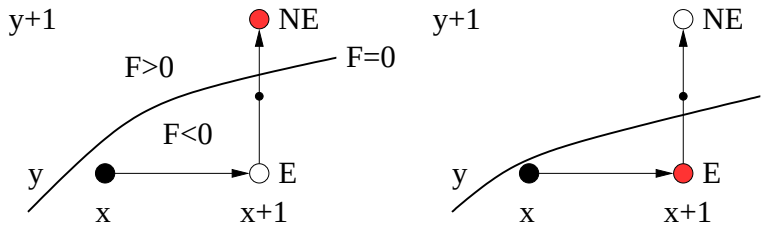


Abbildung 36: Mittelpunktschema für Kurven in impliziter Darstellung

5 Rasterdarstellung

5.1 Verrasterung

Frage 5.1: Worum geht es bei der Verrasterung?
Bei der Verrasterung ist ein kontinuierliches geometrisches Objekt O gegeben. Gesucht ist nun eine Approximation von O durch Rasterpunkt, die bei hinreichend hoher Auflösung wie O aussieht.

Frage 5.2: Wie funktioniert das Prinzip der Verrasterung durch Mittelpunktvergleich?
Bei diesem Prinzip wird davon ausgegangen, dass eine Kurve in impliziter Darstellung gegeben ist, deren Steigung zwischen 0 und 1 liegt. In diesem Fall wird auf dem Raster in x-Richtung fortgeschritten und die Punkte jeweils an den Rasterpunkt gesetzt, der den kleinsten Abstand zur idealen Kurve besitzt. Im gleich folgenden Algorithmus ist dies immer entweder der Punkt rechts vom zuletzt gesetzten Punkt (E) oder der Punkt rechts oben davon (NE). Bei einer Steigung, die größer als 1 oder kleiner als 0 ist, würde dies nicht mehr funktionieren. Dies kann man sich auch klarmachen, wenn man davon ausgeht, dass die Kurve, die verrastert werden soll, parallel zur y-Achse verläuft. Dann würde ein Entlanglaufen in x-Richtung ja nichts bringen, da die Pixel, die gesetzt werden müssen, entlang der y-Achse verlaufen. In einem solchen Fall wird daher auch tatsächlich entlang der y-Achse gelaufen. Im folgenden wird jedoch davon ausgegangen, dass die Steigung zwischen 0 und 1 liegt.

Wie schon erwähnt, ist die Kurve in impliziter Darstellung gegeben, d.h. alle Punkte (x, y), für die die Bedingung

$$F(x, y) = 0, F : \mathbb{R}^2 \to \mathbb{R}$$

gilt, liegen auf der Kurve. Bei impliziten Kurven ist es so, dass alle Punkte (x, y) für die $F(x, y) < 0$ gilt, unterhalb der Kurve liegen, alle Punkte für die $F(x, y) > 0$ gilt oberhalb. Die Entscheidung, welcher Punkt als nächster Punkt gewählt werden soll, wird derart gefällt, dass geschaut wird, ob der Mittelpunkt zwischen den beiden Punkten oberhalb oder unterhalb der Kurve liegt. Es ist offensichtlich, dass der obere Punkt näher an der Kurve liegt, wenn der Mittelpunkt unterhalb der Kurve liegt und im anderen Fall der untere Punkt. Abbildung 36 verdeutlicht dies nochmal. Ist das Ergebnis genau 0 wird irgendeiner der beiden Punkte gewählt.

Mathematisch gesprochen wird also $F(x+1, y+\frac{1}{2})$ berechnet. Ist das Ergebnis kleiner als 0 wird als nächster Punkt $(x+1, y+1)$ gesetzt und von diesem Punkt aus weitergemacht, ansonsten wird $(x+1, y)$ gesetzt und von diesem Punkt aus weitergemacht.

Frage 5.3: Wie können Kurven in Parameterdarstellung verrastert werden?
Die Verrasterung einer Kurve in Parameterdarstellung erfolgt analog zur Approximation durch einen Polygonzug. Nur wird die Schrittweite in diesem Fall so gewählt, dass die Punktfolge Pixelabstand oder weniger hat.

Frage 5.4: Wie werden Strecken verrastert?
Um Strecken zu verrastern, kann ebenfalls das Mittelpunktschema für implizite Kurven angewandt werden, da Strecken als spezielle Kurven betrachtet werden können. Es wird jedoch nicht der oben beschriebene Algorithmus benutzt, sondern eine modifizierte Variante. Zuerst einmal muss eine implizite Darstellung für Geraden gefunden werden. Diese wird die Form

$$F(x, y) = a \cdot x + b \cdot y + c = 0$$

haben. Doch wie bekommt man diese Darstellung? Die „normale“ Geradegleichung

$$y = m \cdot x + B, m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{dy}{dx}$$

kann umgestellt werden, so dass wir die oben geforderte Darstellung erreichen.

$$\begin{aligned} y = \frac{dy}{dx} \cdot x + B \quad &\Leftrightarrow \quad dx \cdot y = dy \cdot x + dx \cdot B \\ &\Leftrightarrow \quad dy \cdot x - dx \cdot y + dx \cdot B = 0 \end{aligned}$$

Es gilt also $a = dy, b = -dx$ und $c = dx \cdot B$. Im Gegensatz zu den Kurven ist es bei dieser Darstellung für Geraden jedoch so, dass Punkte, die oberhalb der Geraden liegen, negative Werte liefern und Punke, die unterhalb liegen, positive Werte. Dies kann man sich wie folgt klarmachen: Wird ein Punkt (x, y), der auf der Kurve liegt (es gilt also $F(x, y) = 0$), nach oben verschoben, vergrößert sich der y-Wert. Das y geht in die Formel als $-dx \cdot y$ ein, es wird in diesem Fall also mehr abgezogen als für $F(x, y)$, womit der Wert der Funktion negativ wird. Entsprechend kann für Punkte unterhalb der Geraden argumentiert werden.

Die Entscheidung, ob das Pixel E oder NE gesetzt wird, hängt davon ab, welchen Wert $F(x, y)$ für den Punkt $(x+1, y+\frac{1}{2})$ liefert. Es wird daher im folgenden eine Entscheidungsfuktion d definiert, welche uns sagt, ob der Mittelpunkt zwischen E und NE oberhalb oder unterhalb der Geraden liegt und damit also entschieden wird, ob E oder NE als nächster Punkt gewählt wird. Die Entscheidungsfunktion d ist also definiert als:

$$d(x, y) = a \cdot (x+1) + b \cdot \left(y + \frac{1}{2}\right) + c$$

Gilt $d > 0$ wird der Punkt NE als nächstes gewählt, ansonsten für $d \leq 0$ der Punkt E (für $d = 0$ ist es egal, welcher Punkt als nächstes gewählt wird). Als nächstes wird nun betrachtet, was an der nächsten Gitterlinie $x + 2$ passiert.

Dies hängt natürlich davon ab, ob der Punkt E oder NE gewählt wurde. Wurde der Punkt E gewählt, ergibt sich d_{neu} als:

$$d_{neu} = F\left(x+2, y+\frac{1}{2}\right) = a\cdot(x+2) + b\cdot\left(y+\frac{1}{2}\right) + c$$

Wie unterscheidet sich d_{neu} vom vorherigen d?

$$\begin{aligned} d_{neu} - d &= a\cdot(x+2) + b\cdot\left(y+\frac{1}{2}\right) + c - \left(a\cdot(x+1) + b\cdot\left(y+\frac{1}{2}\right) + c\right) \\ &= a\cdot(x+2) - a\cdot(x-1) = a \end{aligned}$$

Es gilt also $d_{neu} - d = a$, was umgeformt werden kann zu $d_{neu} = d + a$. Das neue d ergibt sich also aus dem alten durch Addition von $a = dy$. Wurde der Punkt NE gewählt sieht die Situation ähnlich aus. In diesem Fall ergibt sich d_{neu} als:

$$d_{neu} = F\left(x+2, y+\frac{3}{2}\right) = a\cdot(x+2) + b\cdot\left(y+\frac{3}{2}\right) + c$$

Wieder wird $d_{neu} - d$ berechnet:

$$\begin{aligned} d_{neu} - d &= a\cdot(x+2) + b\cdot\left(y+\frac{3}{2}\right) + c - \left(a\cdot(x+1) + b\cdot\left(y+\frac{1}{2}\right) + c\right) \\ &= a\cdot(x+2) - a\cdot(x-1) + b\cdot\left(y+\frac{3}{2}\right) - b\cdot\left(y+\frac{1}{2}\right) = a + b \end{aligned}$$

Es gilt also $d_{neu} = d + a + b = d_{neu} = d + dy - dx$. Damit kann das neue d in beiden Fällen inkrementell durch das alte d berechnet werden. Was jetzt noch etwas unangenehm erscheinen könnte, ist das Vorkommen von $c = dx \cdot B$. Es sieht also so aus, als müsste das B aus den beiden gegebenen Punkten P_0 und P_1 berechnet werden. Dies ist jedoch aufgrund der Tatsache, dass der Startpunkt (x_0, y_0) genau auf der Linie liegt und somit $F(x_0, y_0) = 0$ gilt, nicht der Fall. Das erste d ergibt sich dann nämlich als:

$$\begin{aligned} d_{start} &= F\left(x_0+1, y_0+\frac{1}{2}\right) = a\cdot(x_0+1) + b\cdot\left(y_0+\frac{1}{2}\right) + c \\ &= ax_0 + by_0 + c + a + \frac{b}{2} = \underbrace{F(x_0, y_0)}_{0} + a + \frac{b}{2} = a + \frac{b}{2} \end{aligned}$$

Andererseits ist nun ärgerlich, dass d_{start} einen Bruch enthält, denn bisher sind keine Divisionen oder Multiplikationen aufgetreten. Dieses Problem kann jedoch leicht behoben werden, indem die ursprüngliche Funktion $F(x, y)$ mit zwei multipliziert wird: $F(x, y) = 2\cdot(ax + by + c)$. Dies ist zulässig, da es im Endeffekt immer nur um das Vorzeichen der Entscheidungsvariable d ging und sich dies durch diese Operation nicht ändert.

Frage 5.5: Wie können Kreise verrastert werden?

Bei Kreisen wird im Prinzip wieder so ähnlich vorgegangen, wie beim Mittel-

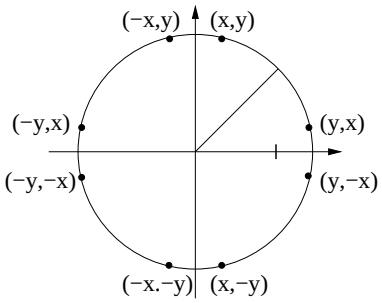


Abbildung 37: Achtfache Symmetrie eines Kreises

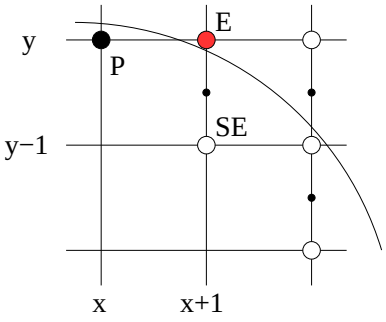


Abbildung 38: Mittelpunktschema für Kreise

punktschema für Linien. Jedoch werden hier wieder einige Eigenschaften ausgenutzt, wie z.B. die achtfache Symmetrie des Kreises, welche in Abbildung 37 dargestellt ist. Dies wird gemacht, da es ansonsten aufgrund von Rundungsfehlern passieren kann, dass der berechnete Endpunkt nicht mit dem Startpunkt übereinstimmt und der Kreis nicht richtig schließt. Aus diesem Grund wird im gleich folgenden Algorithmus nur der zweite Oktant, der sich von $x = 0, y = R$ bis $x = y = \frac{R}{\sqrt{2}}$ erstreckt. Wird auf diesem Kreisbogen ein (x, y) berechnet so werden die sieben anderen symmetrischen Punkte (siehe Abbildung 37) ebenfalls gesetzt. Die implizite Darstellung für einen Kreis lautet:

$$F(x, y) = x^2 + y^2 - R^2 = 0$$

Für Punkte außerhalb des Kreises nimmt diese Funktion positive Werte an und für Punkte innerhalb negative Werte. Wieder wird eine Entscheidungsfunktion definiert, die angibt, welcher Punkt als nächstes gesetzt werden soll. Da jedoch der zweite Oktant gezeichnet werden soll, können dies diesmal die Punkte E oder SE sein. Dieser Sachverhalt wird in Abbildung 38 verdeutlicht. Auch hier geht es also wieder darum, welchen Wert eine Entscheidungsfunktion d für einen Punkt annimmt. Da wir uns jedoch „nach unten“ bewegen, ist dieser Punkt diesmal der Punkt $(x + 1, y - \frac{1}{2})$. Es ergibt sich d also als:

$$d(x, y) = F\left(x + 1, y - \frac{1}{2}\right) = (x + 1)^2 + \left(y - \frac{1}{2}\right)^2 - R^2$$

Ist $d < 0$ wird der Punkt E gewählt und d_{neu} ergibt sich wie folgt:

$$d_{neu} = F\left(x+2, y-\frac{1}{2}\right) = (x+2)^2 + \left(y-\frac{1}{2}\right)^2 - R^2$$

Genau wie beim Linienalgorithmus wird wieder die Differenz zwischen d_{neu} und d berechnet, welche diesmal jedoch keine Konstante ergibt, sondern eine lineare Funktion, die vom Punkt P abhängig ist. Es gilt:

$$\begin{aligned} d_{neu} - d &= (x+2)^2 + \left(y-\frac{1}{2}\right)^2 - R^2 - \left((x+1)^2 + \left(y-\frac{1}{2}\right)^2 - R^2\right) \\ &= (x+2)^2 - (x+1)^2 = x^2 + 4x + 4 - (x^2 + 2x + 1) \\ &= 2x + 3 \end{aligned}$$

Genauso wird vorgegangen, wenn der Punkt SE gewählt wurde. Dann ist d_{neu}:

$$d_{neu} = F\left(x+2, y-\frac{3}{2}\right) = (x+2)^2 + \left(y-\frac{3}{2}\right)^2 - R^2$$

Die Differenz ergibt:

$$\begin{aligned} d_{neu} - d &= (x+2)^2 + \left(y-\frac{3}{2}\right)^2 - R^2 - \left((x+1)^2 + \left(y-\frac{1}{2}\right)^2 - R^2\right) \\ &= (x+2)^2 - (x+1)^2 + \left(y-\frac{3}{2}\right)^2 - \left(y-\frac{1}{2}\right)^2 \\ &= x^2 + 4x + 4 - (x^2 + 2x + 1) + y^2 - 3y + \frac{9}{4} - (y^2 - y + \frac{1}{4}) \\ &= 2x - 2y + 5 \end{aligned}$$

Auch hier ergibt sich also eine lineare Funktion, d.h. der nächste „Fehler“ ergibt sich also in Abhängigkeit davon, ob der Punkt E oder SE gewählt wurde und an welchem Punkt man sich gerade befindet, denn von diesem gehen die x und y ein. Der Startwert d_{start} ergibt sich, da wir im Punkt $(0, R)$ starten, wie folgt:

$$d_{start} = F(1, R - \frac{1}{2}) = 1^2 + \left(R-\frac{1}{2}\right)^2 - R^2 = 1 + R^2 - R + \frac{1}{4} - R^2 = \frac{5}{4} - R$$

Auch hier stört jedoch wieder der Bruch. Allerding kann d_{start} einfach zu $d_{start} = 1 - R$ abgeändert werden. KOMMT NOCH!

Frage 5.6: Wie werden Polygone verrastert?

Es wird davon ausgegangen, dass ein Polygon, dessen Eckpunkte auf einem ganzzahligen Raster liegen, gegeben ist. Gesucht werden nun die Punkte des Raster, die in P oder auf eine Kante liegen. Grundsätzlich kann zwischen zwei verschiedenen Ansätzen unterschieden werden, um Polygon zu verrastern bzw. zu füllen:

Scan-Line-Verfahren: Bei diesem Verfahren wird davon ausgegangen, dass eine geometrische Beschreibung der Begrenzungskurven des Polygons vorliegt.

Rand- und innenbezogenes Füllen: Diese beiden Verfahren nutzen die Zusammenhangseigenschaft der Pixel im Inneren eines zu füllenden Objektes (z.B. ein Polygon) aus.

Frage 5.7: Wie funktioniert das Scan-Line-Verfahren zum Füllen von Polygonen?

Beim Scan-Line-Verfahren werden die Schnittpunkte zwischen einer Scan-Line und den Polygon-Kanten berechnet, um für eine Rasterzeile herauszufinden, welche Pixel sich im Inneren des Polygons befindet. Die Scan-Line verläuft dabei im allgemeinen parallel zur x-Achse, obwohl man auch eine Scan-Line parallel zur y-Achse benutzen könnte. Die Scan-Line wird dabei nicht über den gesamten Bildschirm bewegt, sondern nur von der maximalen bis zur minimalen y-Koordinate eines Polygons. Damit nicht in jedem Schritt alle Kanten betrachtet werden müssen, werden diese nach den größten y-Werten sortiert. Grob arbeitet der Algorithmus wie folgt:

- Sortiere die Kanten nach den größten y-Werten.
- Laufe mit der Scan-Line vom größten zum kleinsten y-Wert.
- Mache das folgende für jede Scan-Line:
  - Aktualisiere die Liste mit den für diese Scan-Line zu betrachtenden Kanten.
  - Berechne die Schnittpunkte und sortiere diese nach den x-Werten.
  - Zeige die Teile der Scan-Line, welche im Inneren des Polygons liegen an.

Diese Schritte zerfallen in einige Unterprobleme, welche es möglichst effizient zu lösen gilt. Zuerst kann man sich fragen, wann eine Kante in die Liste der zu betrachtenden Kanten eingefügt wird bzw. wann sie wieder aus dieser heraus genommen wird. Da sich die Scan-Line von oben nach unten bewegt, wird eine Kante dann eingefügt, wenn die Scan-Line ihren Endpunkt mit der größeren y-Koordinate überstreicht und herausgenommen, wenn der andere Endpunkt überstrichen wird.

Die Berechnung der Schnittpunkte kann aufgrund der Kohärenz-Eigenschaften eines Polygons relativ effizient gestaltet werden. Dies liegt daran, dass benachbarte Schnittpunkt eine gewisse Nähe zueinander aufweisen, die im Prinzip von der Steigung der Kante abhängt. Angenommen die Scan-Lines der Zeile i und $i+1$ schneiden eine Kante in den beiden Punkten $P_i = (x_i, y_i)$ und $P_{i+1} = (x_{i+1}, y_{i+1})$. Zwischen den beiden Punkten verläuft die Kante mit der Steigung $m = \frac{y_{i+1}-y_i}{x_{i+1}-x_i}$. Da sich die y-Koordinaten der beiden Punkte jedoch nur um 1 unterscheiden, ergibt sich:

$$m = \frac{\overbrace{y_{i+1} - y_i}^{1}}{x_{i+1} - x_i} \Leftrightarrow m \cdot (x_{i+1} - x_i) = 1 \Leftrightarrow x_{i+1} = x_i + \frac{1}{m}$$

Dies bedeutet, dass der Schnittpunkt einer Kante mit einer Scan-Line stets aus dem Schnittpunkt mit der vorhergehenden Rasterzeile berechnet werden kann.

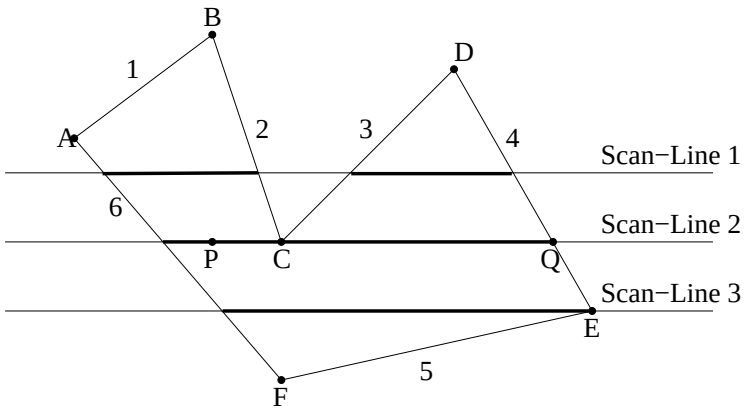


Abbildung 39: Prinzip des Scan-Line-Verfahrens

Ein weiteres Problem kann bei der Berechnung des Inneren eines Polygons auftreten. Wie schon in Kapitel 4.2 erläutert wurde, liegt ein Punkt im Inneren eines Polygons, wenn ein Strahl, der von ihm ausgeht, die Polygonkanten in ungerader Anzahl schneidet. Betrachtet man nun jedoch Abbildung 39, so sieht man, dass der Sonderfall eines auf der Rasterlinie liegenden Punktes auftreten kann. In der Abbildung sind dies die Punkte C und E. Um für den Punkt P das richtige Ergebnis zu bekommen, nämlich dass er innerhalb des Polygons liegt, darf C nicht als Schnitt gezählt werden. Für einen Punkt im Inneren des Polygons auf der Scan-Line 3 müsste der Punkt E jedoch als ein Schnitt gewertet werden, um ein richtiges Ergebnis zu bekommen. Was ist nun richtig? Die Punkte C und E unterscheiden sich darin, auf welcher Seite der Scan-Line die anderen Endpunkte liegen. Für Punkt C liegen die Punkte B und D z.B. auf derselben Seite der Scan-Line. Für E ist dies nicht der Fall. Also kann folgende Regel benutzt werden: Liegen die beiden anderen Punkte auf verschiedenen Seiten der Scan-Linie, so wird der Schnitt gezählt, ansonsten nicht.

Frage 5.8: Welche Datenstrukturen können zur Implementierung des Scan-Line-Verfahren verwendet werden?

Zur Implementierung des Scan-Line-Verfahren werden verschiedene Datenstrukturen benötigt. Zuerst einmal müssen die Kanten des Polygons irgendwie abgespeichert werden. Dazu reicht es, die maximale y-Koordinate, die x-Koordinate des unteren Punktes und das x-Inkrement zu speichern. Desweiteren werden zwei Datenstrukturen benötigt, um die Kanten zu verwalten. In der sogenannten X-Datenstruktur werden die für die momentane Scan-Line aktiven Kante sortiert gespeichert. Die noch nicht betrachteten Kanten werden nach abfallenden y-Werten sortiert in der Y-Datenstruktur verwaltet.

Frage 5.9: Was versteht man unter einer 4- bzw. 8-zusammenhängenden Pixelkurve?

Eine 4-zusammenhängende Kurve ist eine Folge von Pixeln, wobei das nächste Pixel durch waagerechtes oder senkrechtes Fortschreiten erreichbar ist. Bei einer 8-zusammenhängenden Kurve können die Pixel auch noch diagonal erreichbar

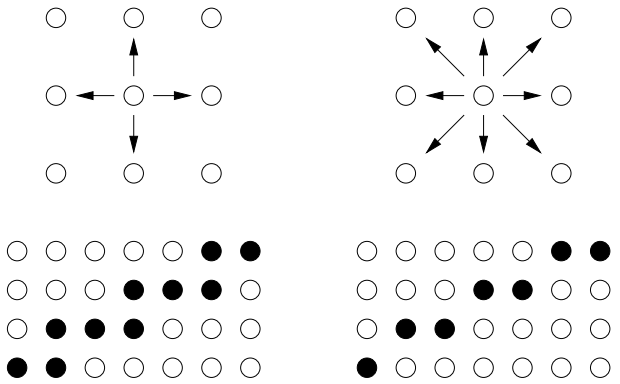


Abbildung 40: 4- und 8-zusammenhängende Pixelkurven

sein. Der Unterschied zwischen den beiden Definitionen wird in Abbildung 40 verdeutlicht.

Frage 5.10: Wie funktioniert randbezogenes Füllen?
Bei diesem Verfahren wird eine geschlossene, 4-zusammenhängende (geht nicht auch 8-zusammenhängend?) Pixelkurve, deren Pixel auf den Wert RW gesetzt seien, als gegeben angenommen. Desweiteren wird ein Punkt (x_S, y_S) im Inneren der Kurve als Startpunkt vorgegeben. Von diesem Punkt aus sollen nun alle anderen Pixel im Inneren der Kurve auf einen neuen Wert NW gesetzt werden. Dazu wird zuerst überprüft, ob das momentan betrachtete Pixel nicht schon auf den Wert NW oder RW gesetzt ist. Ist dies nicht der Fall, wird es auf NW gesetzt und dasselbe mit den 4-zusammenhängenden Nachbarn des Pixels gemacht. Der Ablauf ist in Algorithmus 3 als Pseudo-Code beschrieben. Ein großer Nachteil dieses Algorithmus ist der große Speicherbedarf, da aufgrund der vielen Rekursionen der Stack schnell wächst.

Algorithmus 3 Randbezogenes Füllen

```
Randfüllen(x,y,RW,NW):
if Pixel(x, y) ≠RW und Pixel(x, y) ≠NW then
  Pixel(x, y) =NW
  Randfüllen(x + 1,y,RW,NW)
  Randfüllen(x − 1,y,RW,NW)
  Randfüllen(x,y + 1,RW,NW)
  Randfüllen(x,y − 1,RW,NW)
end if
```

Frage 5.11: Wie funktioniert innenbezogenes Füllen?
Das innenbezogene Füllen funktioniert ähnlich wie das randbezogene Füllen. Allerdings wird hier ein Startpunkt gegeben, welcher in einem 4-zusammenhängenden Gebiet liegt, dessen Pixel alle einen gemeinsamen Wert IW besitzen. Von diesem Startpixel aus sollen nun alle anderen Pixel auf den neuen Wert NW gesetzt werden. Algorithmus 4 beschreibt das Vorgehen in Form von Pseudo-Code.

Algorithmus 4 Innenbezogenes Füllen

```
Innenfüllen(x,y,IW,NW):
if Pixel(x, y) =IW then
  Pixel(x, y) =NW
  Innenfüllen(x + 1,y,IW,NW)
  Innenfüllen(x − 1,y,IW,NW)
  Innenfüllen(x,y + 1,IW,NW)
  Innenfüllen(x,y − 1,IW,NW)
end if
```

Frage 5.12: Wie funktioniert der Span-Fill-Algorithmus?
Die Algorithmen für das rand- bzw. innenbezogene Füllen sind, bedingt durch die vielen Rekursionen, häufig sehr speicherintensiv und damit ineffizient. Der Span-Fill-Algorithmus vermeidet diese häufigen Rekursionen, indem ein eigener Stack verwaltet wird, auf dem die Pixel gelegt werden, von denen aus als nächstes gefüllt wird.

Der Algorithmus funktioniert wie folgt: Von einem Pixel aus wird eine komplette Zeile gefüllt, d.h. jedes Pixel bekommt den neuen Wert, bis man auf den Rand trifft. Während des Durchlaufs durch eine Zeile werden die beiden benachbarten Zeilen betrachtet und jedes Pixel, welches an einer Intervallgrenze liegt, auf den Stack gelegt.

5.2 Farbreduktion

Frage 5.13: Welche Möglichkeiten gibt es. die Darstellungsqualität eines Rasterbildes zu ändern?

- Farbbild → Grauwertbild
- Grauwertbild → Monochrombild
- Farbbild mit vielen Farben → Farbbild mit Farbtabelle

Frage 5.14: Wie werden Farbbilder auf Grauwertbildern reduziert?
Um ein Farbbild, bei dem ein Pixel in drei Byte kodiert ist, auf ein Grauwertbild, welche nur ein Byte für die Intensität benutzt, zu reduzieren, werden die Rot-, Grün- und Blauanteile gewichtet aufsummiert, so dass sich ein neuer Wert zwischen 0 und 255 ergibt. Dies leistet die folgende Formel:

$$I = 0.3 \cdot I_{Rot} + 0.59 \cdot I_{Grün} + 0.11 \cdot I_{Blau}$$

Die Gewichtungen kommen über die unterschiedliche Empfindlichkeit des Auges gegenüber bestimmten Farben zustande.

Frage 5.15: Wie wird ein Grauwertbild auf ein Monochrombild reduziert?
Gegeben sei ein Grauwertbild, welches als eine Funktion $f(x, y)$ dargestellt wird, die jedem Pixel einen Intensitätswert zuordnet. Gesucht ist ein neues Bild f', so dass gilt $f'(x, y) \rightarrow \{0, 1\}$, d.h. das neue Bild besteht nur aus dem

Werten Schwarz und Weiss. Es wird davon ausgegangen, dass das Bild n Pixel hoch ist und m Pixel breit. Pixel werden bei 0 beginnend gezählt und laufen entsprechend bis $n-1$ bzw. $m-1$. Um ein Grauwertbild auf ein Monochrombild zu reduzieren gibt es mehrere Möglichkeiten:

Konstanter Schwellwert: Dies ist das einfachste Verfahren. Für jedes Pixel wird die Intensität überprüft. Liegt sie über einem gewissen Schwellwert T, wird das Ergebnispixel auf den Wert 1 gesetzt, ansonsten auf 0.

Konstanter Schwellwert mit Fehlerübertragung: Dieses Verfahren wurde von Floyd und Steinberg entwickelt und arbeitet ebenfalls mit einem konstanten Schwellwert. Jedoch wird hier versucht, den Fehler, den man bei der Modifikation eines Pixels macht, auf einige der benachbarten Pixel zu übertragen. Wenn zum Beispiel ein Pixel (x, y) gegeben ist, dessen Originalwert $f(x, y) = 0.7$ beträgt, wird dies bei einem Schwellwert von $T = 0.5$ auf den Wert $f'(x, y) = 1$ gesetzt. Der Fehler, der dabei gemacht wird, ist $f'(x, y) - f(x, y) = 0.3$. Das Pixel wird also heller gemacht, als es eigentlich war. Aus diesem Grund werden einige Pixel in der Umgebung dieses Pixels etwas dunkler gemacht, indem ihnen ein gewisser Bruchteil dieses Fehlers abgezogen wird. Algorithmus 5 zeigt das Verfahren als Pseudo-Code und Abbildung 41 zeigt, wie der Fehler eines Pixels auf die benachbarten Pixel verteilt wird. Die Gewichtungen sind dabei nach Floyd-Steinberg gesetzt, es können auch andere gewählt werden.

Algorithmus 5 Konstanter Schwellwert mit Fehlerübertragung

for $i = 0$ **to** $n - 1$ **do**
  for $j = 0$ **to** $m - 1$ **do**
    if $f(i, j) > T$ **then**
      $f'(i, y) = 1$
    else
      $f'(i, y) = 0$
    end if
    $E = f(i, j) - f'(i, j) \cdot Q$ {Fehlerberechnung}
    $f(i, j + 1) = f(i, j + 1) + \frac{7}{16} \cdot E$
    $f(i + 1, j + 1) = f(i + 1, j + 1) + \frac{1}{16} \cdot E$
    $f(i + 1, j) = f(i + 1, j) + \frac{5}{16} \cdot E$
    $f(i + 1, j - 1) = f(i + 1, j - 1) + \frac{3}{16} \cdot E$
  end for
end for

Dither-Matrix: Bei den beiden Verfahren, die einen konstanten Schwellwert benutzen, gibt es nur einen einzigen Schwellwert und der neue Wert eines Pixels hängt nur von seinem alten Wert ab. Bei der Verwendung einer Dither-Matrix ist dies anders. Hier wird eine Matrix benutzt, in der mehrere unterschiedliche Schwellwerte stehen, wobei es nun auch von der Position eines betrachteten Pixels abhängt, welcher dieser Schwellwerte benutzt wird. Sei zum Beispiel die Matrix M gegeben als:

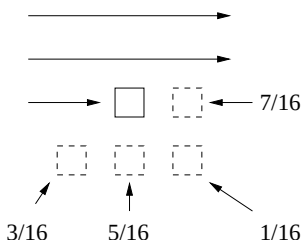


Abbildung 41: Konstanter Schwellwert mit Fehlerübertragung

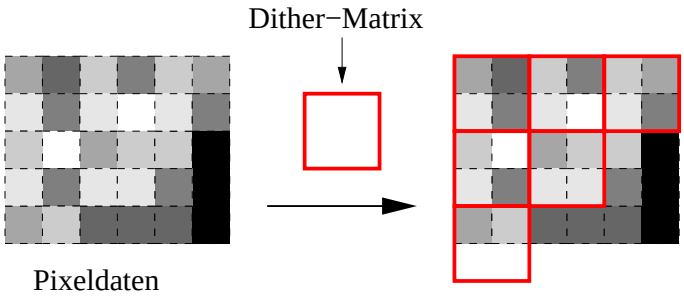


Abbildung 42: Prinzip des Dithering

$$M = \begin{pmatrix} 32 & 160 \\ 224 & 96 \end{pmatrix}$$

Beim Dithering wird diese nun, so oft wie dies möglich ist, wiederholt über das Bild gelegt. Abbildung 42 zeigt das Prinzip. Für jedes Pixel gibt es dann in der Matrix einen korrespondierenden Schwellwert. Pixel, welche z.B. von der linken oberen Ecke der 2×2-Matrix M überdeckt werden, bekommen den Schwellwert 32 zugewiesen.

Anhand von Bildern, deren Pixelwerte alle in einem bestimmten Intervall liegen, kann man sich ganz gut klarmachen, was hierbei passiert. Liegen alle Pixelwerte z.B. zwischen 0 und 31, das Bild ist also recht dunkel, werden alle Pixel auf den neuen Wert 0 gesetzt. Dieser Fall ist in Abbildung 43 ganz links gezeigt. Im Prinzip müsste in diesem Fall alles schwarz sein, aber aufgrund der besseren Sichtbarkeit wird in der Abbildung quasi alles invers dargestellt. Liegen dagegen alle Werte zwischen 32 und 95, so kann sich stets nur das Pixel oben links in der Matrix durchsetzen und es entsteht eine Aneinanderreihung von 2×2-Bildern bei denen nur das linke obere Pixel gesetzt ist. Insgesamt wird das Bild also heller. Für jedes Intervall werden nun immer mehr Pixel „einschaltet“, bis für helle Bilder, deren Werte alle zwischen 224 und 255 liegen schließlich alle Pixel gezeichnet werden.

Wie können nun größere Dithermatrizen entworfen werden? Ein Verfahren hierzu wurde von Judice, Jarvis und Ninke vorgestellt. Bei diesem wird aus einer kleineren, gegebenen $n \times n$-Matrix eine $2n \times 2n$-Matrix erzeugt.

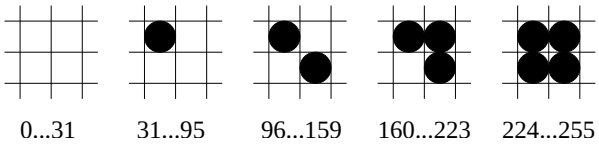


Abbildung 43: „Einschalten“ der einzelnen Pixel beim Dithering

Sei die $n \times n$-Matrix $M_{c,n}$ gegeben und U_n eine $n \times n$-Matrix, welche nur mit Einsen gefüllt ist. Dann ergibt sich die Matrix $M_{c,2n}$ wie folgt:

$$M_{c,2n} = \begin{pmatrix} 4M_{c,n} & 4M_{c,n} + 2U_n \\ 4M_{c,n} + 3U_n & 4M_{c,n} + U_n \end{pmatrix}$$

Dies wird nun anhand eines Beispiels verdeutlicht:

$$M_{c,2} = \begin{pmatrix} 0 & 2 \\ 3 & 1 \end{pmatrix} \qquad M_{c,4} = \begin{pmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{pmatrix}$$

Der Vorteil der gegebenen Matrix $M_{c,2}$ ist der, dass die neue Matrix alle Werte zwischen 0 und $n^2 - 1$ enthält. Wie errechnen sich nun die Schwellwerte in der entgültigen Dithermatrix? Für eine Matrix $M_{c,n}$ errechnet sie sich wie folgt:

$$M(i,j) = M_{c,n}(i,j) \cdot \Delta + \frac{\Delta}{2}, \Delta = \frac{\overbrace{256}^{Q}}{n^2}$$

Dabei ist Q die Anzahl aller Intensitäten. Für die oben errechnete Matrix $M_{c,4}$ ergibt sich damit folgende Schwellwertmatrix M:

$$M = \begin{pmatrix} 8 & 136 & 40 & 168 \\ 200 & 72 & 232 & 104 \\ 56 & 184 & 24 & 152 \\ 248 & 120 & 216 & 88 \end{pmatrix}$$

Algorithmus 6 zeigt, wie das Nebeneinanderlegen der Dithermatrix über das Bild mathematisch funktioniert.

Frage 5.16: Worum geht es bei der Farbreduktion? In welche Teilprobleme zerfällt das Problem?

Bei der Farbreduktion geht es darum ein Bild mit sehr vielen Farben, in ein Bild mit relativ wenig Farben aus einer Farbtabelle umzurechnen. Das Problem zerfällt in zwei Teilprobleme:

1. Berechnung einer Farbtabelle
2. Ersetzung der alten Farben des Bildes durch die neuen in der Farbtabelle

Algorithmus 6 Dithering

```
for i = 0 to n − 1 do
  for j = 0 to m − 1 do
    if f(i, j) > M(i mod n, j mod n) then
      f′(i, y) = 1
    else
      f′(i, y) = 0
    end if
  end for
end for
```

Frage 5.17: Welche Möglichkeiten kennst du zur Berechnung einer Farbtabelle?

- Gleichmäßige Farbraumzerlegung
- Popularitätsverfahren *(popularity approach)*
- Octree-Quantisierung
- Medianschnitt *(median cut)*

Frage 5.18: Wie funktioniert die gleichmäßige Farbraumzerlegung?
Die gleichmäßige Farbraumzerlegung wird hier anhand des Beispiels der Erstellung einer Farbtabelle mit 256 unterschiedlichen Farben gezeigt. Dazu wird der RGB-Würfel entlang seiner R- und G-Achse in acht gleichmäßige Intervalle unterteilt und entlang der B-Achse in vier Intervalle. Die Unterteilung wird dabei in dieser Art und Weise vorgenommen, da das menschliche Auge auf Blau nicht so intensiv reagiert und da $\sqrt[3]{256}$ nicht ganzzahlig ist. Der Nachteil bei diesem Verfahren ist, dass die Zerlegung bildunabhängig geschieht.

Frage 5.19: Wie funktioniert der Popularitätsansatz?
Angenommen die gesuchte Farbtabelle soll k Farben enthalten. Dann werden beim Popularitätsansatz die k am häufigsten im Bild auftretenden Farben in diese übernommen. Es werden also zuerst alle Farben gezählt und anschließend nach Häufigkeit sortiert. Die k häufigsten Farben werden in die Farbtabelle übernommen. Das Zählen muss dabei nicht ganz so genau sein, d.h. dass die letzten drei Bit jeder Farbe z.B. nicht so genau betrachtet werden müssen.

Der Nachteil dieses Verfahrens ist, dass die Bildqualität bei Bildern schlecht sein kann, in denen selten auftretende Farben von Bedeutung sind. Desweiteren kann die Bestimmung der am häufigsten auftretenden Farben sehr zeitaufwändig sein.

Frage 5.20: Wie funktioniert die Octree-Quantisierung?
Bei diesem Verfahren wird, wie der Name schon sagt, eine Datenstruktur namens Octree benutzt. Ein Octree entsteht durch fortlaufende Unterteilung eines Würfels entlang seiner Achsen. Wird ein Würfel, in diesem Fall der RGB-Würfel, entlang seiner drei Achsen halbiert, entstehen dadurch acht neue, gleich

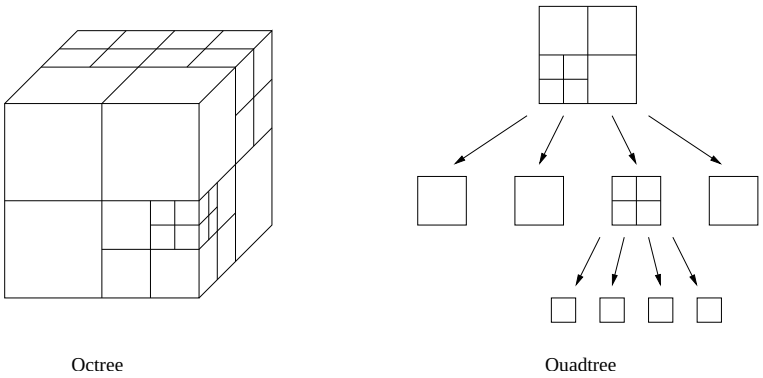


Abbildung 44: Octree und Quadtree

große Würfel, mit denen man unter Umständen dasselbe machen kann. Die linke Seite von Abbildung 44 gibt eine Beispiel für eine Zerlegung, die auf diese Art und Weise entstehen kann. Gezeigt wird ein Würfel, dessen auf diese Art und Weise entstehenden Unterwürfel teilweise wieder geteilt wurden.

Auf der rechten Seite der Abbildung wird die eigentliche Datenstruktur dargestellt, jedoch anhand des Beispiels eines Quadtrees, welcher im zwei-dimensionalen Fall entsteht. Würde gar keine Unterteilung vorgenommen werden, so würde der Baum nur aus der Wurzel bestehen und quasi das ganze Quadrat repräsentieren. Jedoch wurde das Quadrat einmal unterteilt. Daher werden vier Kinder an die Wurzel gehängt, wobei jedes Kind für eines der so entstehenden vier kleineren Quadrate steht. Das Quadrat links unten wurde noch einmal unterteilt, so dass unter diesen Knoten im Quadtree-Baum wieder vier Kinder hängen.

Bei der Octree-Quantisierung läuft das Verfahren jedoch genau anders herum. Es wird nicht ein Baum immer feiner unterteilt, sondern eine feine Unterteilung wird nach und nach immer mehr vergröbert. Der Algorithmus läuft wie folgt, wobei wieder davon ausgegangen wird, dass ein Farbtabelle mit k Farben erzeugt werden soll. Die ersten k verschiedenen Farben werden in den Octree eingefügt, wobei sie an die am weitesten unten liegenden Blätter gesetzt werden. Das heißt, sie laufen also bis ganz unten in dem Baum. Kommen dabei einige Farben mehrfach vor, so wird ein Zähler, der für jede Farbe mitverwaltet wird, für die entsprechende Farbe hochgesetzt. Die $k+1$-te Farbe wird ebenfalls in den Baum eingefügt. Da sich nun eine Farbe zuviel im Baum befindet, werden nun die Blätter gesucht, die am nächsten beieinander liegen, um sie zu einem neuen Blatt zu vereinigen. In einem Octree sind dies die Blätter, die am weitesten unten liegen und einen gemeinsamen Elter besitzen. Die Vereinigung einer z_1-mal vorkommenden Farbe F_1 mit einer z_2-mal vorkommenden Farbe F_2 erfolgt dabei nach der folgenden Formel:

$$F_{neu} = \frac{z_1 \cdot F_1 + z_2 \cdot F_2}{z_1 + z_2}$$

Im Beispiel in Abbildung 45 werden die Farben allerdings nicht sofort zu einer neuen Farbe gemischt, sondern sich nur die Ebenen des Octrees gemerkt, in

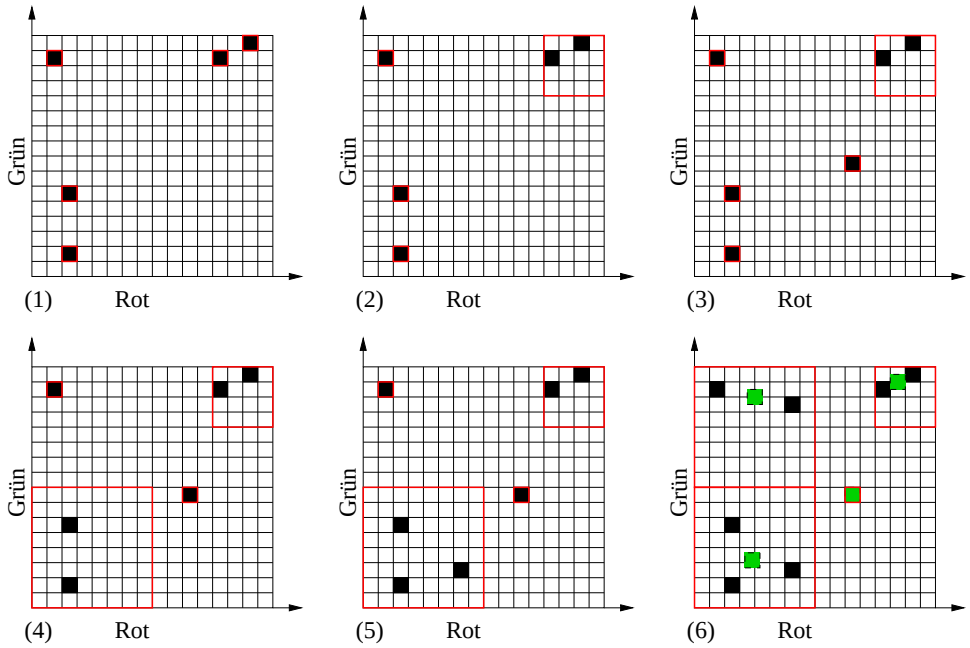


Abbildung 45: Octree-Quantisierung

der sich die zusammengefassten Farben befinden. Da in dem Beispiel jedoch von einem zweidimensionalen Farbraum mit den Komponenten Rot und Grün ausgegangen wird, handelt es sich in diesem Fall um einen Quadtree. Im Beispiel gibt es 8 verschiedene Farben, welche zu $k = 4$ neuen Farben vereinigt werden sollen.

In (1) ist der Punkt erreicht, an dem $k + 1 = 5$ verschiedene Farben in den Quadtree eingefügt wurden. Jede dieser Farben ist durch ein rotes Rechteck markiert, welches zeigt, auf welcher Ebene des Quadtree diese Farbe repräsentiert wird. Momentan ist dies noch für jede Farbe das kleinstmögliche Rechteck (siehe auch Abbildung 44), da jede Farbe so tief wie möglich im Quadtree liegt. Es werden nun die beiden am nächsten beieinander liegenden Farben ermittelt, welches die beiden Farben oben rechts sind. In (2) werden diese zu einem neuen 4×4-Rechteck zusammengefasst, d.h. sie werden auf der drittletzten Ebene des Quadtree repräsentiert. Hätten die Farben beide in einem 2×2-Rechteck in einer der Ecken des roten Rechtecks gelegen, so wäre das rote Rechteck auch nur eben dieses 2×2-Rechteck geworden. Da die Farben dazu jedoch zu weit auseinander liegen, musste das 4×4-Rechteck gewählt werden. In (3) wird der nächste Punkt eingefügt. Da er nicht in einem der schon erzeugten Rechtecke liegt, müssen also wieder Farben zusammengefasst werden. In (4) ist dies geschehen, die beiden Farben links unten wurden zu einem 8×8-Rechteck zusammengefasst. In dieses neue Rechteck wird in (5) eine neue Farbe eingefügt. Daher muss nicht weiter zusammengefasst werden. Nachdem in (6) die letzte Farbe eingefügt wurde und entsprechend vereinigt wurde, werden die gewichteten Mittel für jedes Rechteck ausgerechnet, welche in der Abbildung grün dargestellt sind.

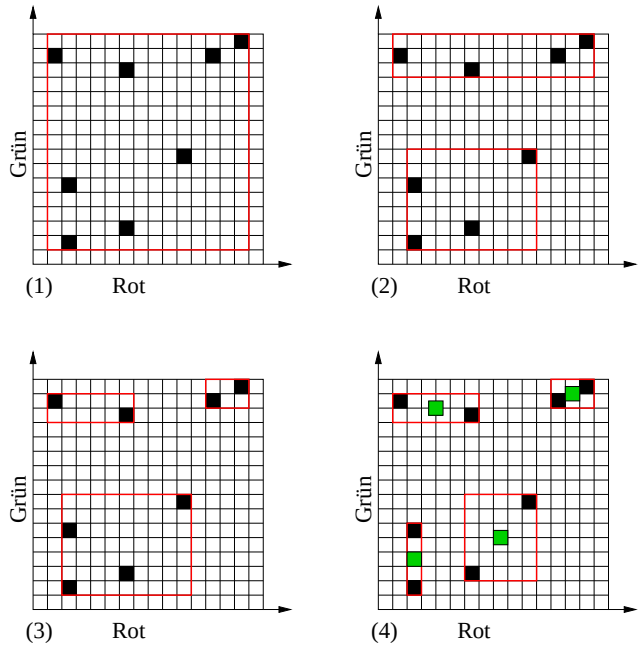


Abbildung 46: Das Medianschnitt-Verfahren

Frage 5.21: Wie funktioniert das Medianschnitt-Verfahren?

Es wird im Folgenden davon ausgegangen, dass eine Farbtabelle mit k Farben erstellt werden soll. Die Datenstruktur, auf der der Algorithmus im Prinzip arbeitet, ist eine Liste von Boxen. Initialisiert wird diese Liste mit der kleinsten Bounding Box, die alle Farben des Bildes enthält. Aus diesem Grund müssen zu Beginn quasi alle Farben bzw. Pixel des Bildes betrachten werden. Ist die Liste initialisiert, arbeitet der Algorithmus die folgenden Schritte wieder und wieder ab, bis sich insgesamt k Boxen in der Liste befinden:

- Entferne die Box mit den meisten Farben aus der Liste.
- Unterteile die entfernte Box am Median ihrer längsten Achse in zwei Teilmengen.
- Füge die Bounding Boxen der so entstandenen beiden Teilmengen in die Liste ein.

Nachdem auf diese Art und Weise k Boxen entstanden sind, wird für jede Box das gewichtete Mittel aller in ihr enthaltenen Farben in die Farbtabelle eingetragen. Eine Teilung entlang dem Median entspricht einer Aufteilung, so dass links und rechts von Medianelement gleich viele Farben liegen.

Man kann sich nun noch fragen, was gemacht werden kann, wenn alle Boxen gleich viele Farben enthalten. In diesem Fall wäre es wahrscheinlich am intelligientesten, die Box zu teilen, die entlang einer aller Achsen am längsten ist.

Abbildung 46 stellt das Verfahren anhand eines Beispiels graphisch dar. Der Einfachheit halber, wird in diesem Beispiel ein zweidimensionaler Farbraum

verwendet, der nur aus den Komponenten Rot und Grün besteht. In diesem Farbraum sind acht verschiedene Farben eingetragen, welche auf vier Farben reduziert werden sollen. Zu Beginn, in Schritt (1), liegt eine Bounding Box um alle Farben. Da diese Bounding Box auch die einzige in der Liste ist, wird sie entlang der Grün-Achse in zwei neue Bounding Boxen geteilt. Dabei entstehen zwei neue Boxen mit vier Elementen, d.h. es tritt der etwas weiter oben erwähnte Fall auf, dass alle Boxen in der Liste gleich viele Elemente besitzen. Da die obere Box entlang der roten Achse länger ist als die untere, wird die obere Box weiter geteilt, so dass (3) entsteht. Die untere Box enthält nun wieder die meisten Elemente, so dass sie noch einmal geteilt wird. Die Liste enthält nun die in (4) eingezeichneten vier Boxen, so dass nur noch das gewichtete Mittel jeder Box berechnet werden muss. In Abbildung 46 sind diese in Form der grünen Punkte dargestellt.

Frage 5.22: Wie werden die alten Farben durch die neuen ersetzt?
Ist die neue Farbtabelle erstellt, müssen die alten Farben des Bildes durch die neuen Farben ersetzt werden. Dabei wird eine Farbe durch die Farbe in der Tabelle ersetzt, die ihr am nächsten ist. Zur Messung des Abstandes zweier Farben können verschiedene Metriken benutzt werden, so z.B. die Manhattan-Metrik oder die „normale“ euklidische Metrik. Sei eine Bildfarbe $F_b = (r_b, g_b, b_b)$ und eine Tabellenfarbe $F_t = (r_t, g_t, b_t)$ gegeben. Dann beträgt der Abstand zwischen den beiden nach Manhattan-Metrik:

$$d(F_b, F_t) = |r_b - r_t| + |g_b - g_t| + |b_b - b_t|$$

Der euklidische Abstand, welcher zeitlich aufwändiger, aber auch genauer als die Manhattan-Metrik ist, berechnet sich wie folgt:

$$d(F_b, F_t) = \sqrt{(r_b - r_t)^2 + (g_b - g_t)^2 + (b_b - b_t)^2}$$

6 Dreidimensionale Darstellung

Frage 6.1: Wie lassen sich Strecken und Kurven im dreidimensionalen Raum beschreiben?
Strecken lassen sich wie in Kapitel 3.1 beschrieben darstellen. Im dreidimensionalen Fall gilt für den Start- und Endpunkt einer Strecke: $p_1, p_2 \in \mathbb{R}^3$.

Kurven lassen sich auch im dreidimensionalen Raum wieder in der Paramterdarstellung und in der impliziten Form darstellen. Die Parameterdarstellung entspricht dabei den in Kapitel 3 vorgestellten Konzepten, wie z.B. Bezierkurven (Kapitel 3.2) und B-Spline-Kurven (Kapitel 3.3).

Frage 6.2: Wie lautet die implizite Darstellung einer Raumkurve?
Eine Raumkurve in impliziter Darstellung ergibt sich als Schnitt zweier implizit definierter Flächen:

$$F(x, y, z) = 0,\ G(x, y, z) = 0 \text{ mit } F, G : \mathbb{R}^3 \to \mathbb{R}$$

6.1 Darstellung von Flächen

Frage 6.3: Welche Möglichkeiten gibt es, um Flächen im $\mathbb{R}^3$ darzustellen?

- Polygone
- Mathematische Darstellung (Parameter, explizit und implizit)
- Bezier-Flächen
- B-Spline-Flächen

Frage 6.4: Wie kann eine Fläche im $\mathbb{R}^3$ durch Polygone dargestellt werden?
Eine Fläche im $\mathbb{R}^3$ kann durch Polygone dargestellt bzw. approximiert werden. Dabei müssen jedoch die Eckpunkte eines jeden Polygons in einer gemeinsamen Ebene liegen, da es sonst schwierig wird, das Innere eines Polygons zu definieren. Aus diesem Grund sind Dreiecke sehr beliebt, da bei diesen die Punkte immer in einer Ebene liegen, wenn diese nicht gerade kollinear sind. Haben die Polygone nicht diese Form, können sie trianguliert, d.h. in einzelne Dreiecke zerlegt werden.

Frage 6.5: Wie lautet die Parameterdarstellung einer Fläche im $\mathbb{R}^3$?
Bei der Parameterdarstellung einer Fläche im $\mathbb{R}^3$ wird zur Berechnung eines Punktes p ein zweidimensionaler Parameterbereich in den $\mathbb{R}^3$ abgebildet. Diese Abbildung ist dabei wie folgt definiert:

$$p = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{pmatrix} = f(u,v) \text{ mit } (u,v) \in P \subseteq \mathbb{R}^2$$

Die Funktion f bildet also einen Punkt (u, v) aus dem Parameterbereich P in den $\mathbb{R}^3$ ab, indem für den Punkt jeweils über die Funktionen $x(u, v)$, $y(u, v)$

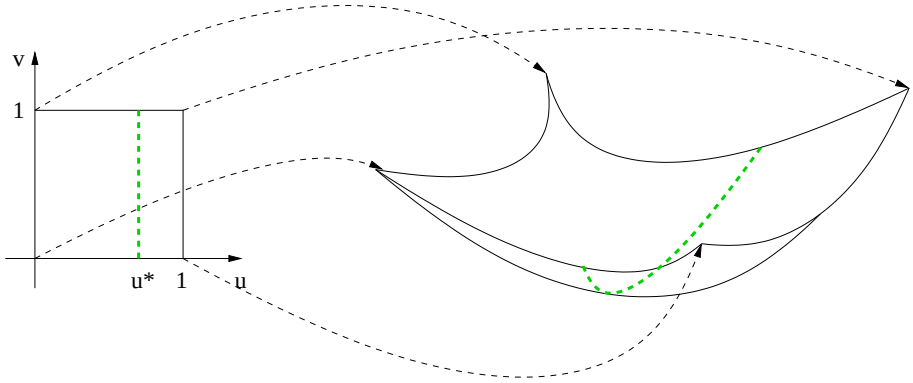


Abbildung 47: Prinzip der Parameterdarstellung einer Fläche

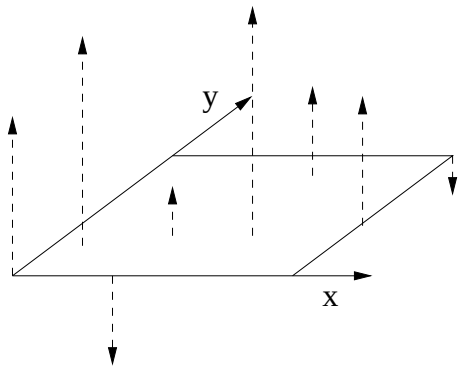


Abbildung 48: Explizite Darstellung einer Fläche

und $z(u, v)$ eine neue x-, y- und z-Koordinate berechnet wird. Abbildung 47 verdeutlicht die Idee dahinter.

In dieser Abbildung wird auch deutlich, was passiert, wenn ein Parameter, z.B. $u = u^*$ festgehalten wird. In diesem Fall definiert $f(u^*, v)$ eine Kurve im Raum. Die Fläche wiederum ergibt sich aus der Schar von Kurven, die sich für verschiedene u^* ergeben. Analog kann auch v festgehalten werden.

Frage 6.6: Wie funktioniert die explizite Darstellung einer Fläche im $\mathbb{R}^3$?
Bei der expliziten Darstellung für Flächen ergibt sich die z-Koordinate einer Fläche als eine Funktion der x- und y-Komponente:

$$z = f(x, y), \quad f : P \to \mathbb{R}, \quad P \subseteq \mathbb{R}^2$$

Abbildung 48 zeigt dabei das grundlegende Prinzip. Ein Punkt auf der Fläche ergibt sich als $p = \begin{pmatrix} x & y & f(x,y) \end{pmatrix}^T$. Hier einige simple Beispiele:

- Sinuswelle im Raum: $z = \sin x$
- x-y-Ebene im Raum: $z = 0$

Frage 6.7: Wie funktioniert die implizite Darstellung einer Fläche im $\mathbb{R}^3$?

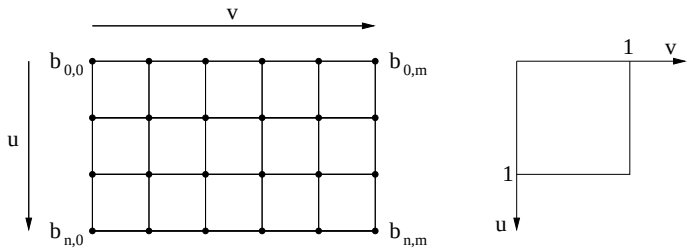


Abbildung 49: Das Gitter von Kontrollpolygonzügen

Die implizite Darstellung einer Fläche im Raum funktioniert ähnlich wie die implizite Darstellung einer Kurve in der Ebene. In diesem Fall ist die Fläche als die Nullstellenmenge einer Funktion, die von drei Variablen abhängt, gegeben:

$$F(x, y, z) = 0 \text{ mit } F : \mathbb{R}^3 \to \mathbb{R}$$

Ein relativ intuitives Beispiel ist die implizite Darstellung einer Kugel um den Mittelpunkt (m_x, m_y, m_z):

$$(x - m_x)^2 + (y - m_y)^2 + (z - m_z)^2 - r^2 = 0$$

Die Klasse der Quadriken wird beschrieben durch:

$$ax^2 + by^2 + cz^2 + dxy + exz + fyz + gx + hy + iz + j = 0$$

Ein Beispiel für eine Quadrik ist ein Ellipsoid, welcher durch die folgende Formel gegeben ist:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1$$

Frage 6.8: Wie sind Bezier-Flächen definiert? Wie funktioniert der Algorithmus von de Casteljau für Bezier-Flächen?

Im Gegensatz zu den Bezierkurven wird bei den Bezier-Flächen nicht nur ein Kontrollpolygonzug sondern ein Gitter bzw. Netz von Kontrollpolygonzügen benutzt. Dabei sind $(n + 1) \cdot (m + 1)$ Kontrollpunkte $b_{i,j}$ mit $i \in \{0, \ldots n\}$ und $j \in \{0, \ldots, m\}$ gegeben. Abbildung 49 zeigt ein solches Kontrollnetz. Für ein solches Netz ist durch folgende Formel ein Vierecks-Bezier-Segment gegeben:

$$b(u, v) = \sum_{i=0}^{n} \left(\sum_{j=0}^{m} b_{i,j} \cdot B_j^m(v) \right) \cdot B_i^n(u) \text{ mit } u, v \in [0, 1], b_{i,j} \in \mathbb{R}^3$$

Wird diese Formel etwas systematischer aufgeschrieben, wird recht schnell klar, wie der Algorithmus von de Casteljau in diesem Fall angewandt wird. Zuerst einmal wird der Punkt $c_i, i \in \{0, \ldots, n\}$ definiert als:

$$c_i = \sum_{j=0}^{m} b_{i,j} \cdot B_j^m(v)$$

Der Punkt c_i ist also der Punkt, der sich in der i-ten Zeile ergibt, wenn der Kontrollpolygonzug der i-ten Zeile des Kontrollnetzes für einen Wert $v \in [0,1]$ ausgewertet wird. Für die zweite Zeile, wäre der Kontrollpolygonzug also gegeben durch die Punkte $b_{2,0}, b_{2,1}, \ldots, b_{2,m}$. Da es $(n+1)$ Zeilen gibt, können also $(n+1)$ verschiedene solcher neuen Punkte c_i ausgerechnet werden. Diese $(n+1)$ Punkte werden nun als ein neues Kontrollpolygon interpretiert, welches sich in diesem Fall quasi entlang einer Spalte erstreckt. Obige Formel kann wie folgt umgeschrieben werden:

$$b(u,v) = \sum_{i=0}^{n} c_i \cdot B_i^n(u)$$

Der Punkt (u,v) ergibt sich also, indem zuerst $(n+1)$ Zeilen-Kontrollpolygonzüge ausgewertet werden und deren Ergebnisse als ein Spalten-Kontrollpolygonzug zum Parameter u ausgewertet wird.

Frage 6.9: Welche Laufzeit hat der Algorithmus von de Casteljau für Bezier-Flächen?

Die Laufzeit beträgt $O(m^2 \cdot n + n^2)$. Für eine Bezier-Kurve mit $m+1$ Kontrollpunkten beträgt die Laufzeit zur Berechnung eines Punktes auf der Kurve $O(m^2)$. Es müssen $(n+1)$ solcher Punkte berechnet werden $(O(m^2 \cdot m))$ und anschließend eine Bezierkurve mit $(n+1)$ Kontrollpunkten ausgewertet werden, was $O(n^2)$ dauert. Der Zeitaufwand mit dem Horner-Schema beträgt $O(m \cdot n + n)$.

Frage 6.10: Nenne einige Eigenschaften von Vierecks-Bezier-Segmenten!

Vierecks-Bezier-Segmente besitzen unter anderem die folgenden Eigenschaften, die hier nicht bewiesen werden, da sie aus den Eigenschaften für Bezier-Kurven und dem Vorgehen bei Bezier-Flächen resultieren.

Konvexe-Hülle-Eigenschaft: Das Flächenstück liegt in der konvexen Hülle des Kontrollnetzes.

Einfluss von Kontrollpunkten: Der Bezierpunkt $b_{i,j}$ hat bei $(u,v) = (\frac{i}{n}, \frac{j}{m})$ den größten Einfluss auf die Fläche.

Verhalten in den Eckpunkten: Die Eckpunkte des Kontrollnetzes und die Eckpunkte des Flächenstücks stimmen überein.

Verhalten in den Randkurven: Die Randpunkte des Kontrollnetzes sind die Bezierpunkte der Randkurve des Flächensegmentes.

Planarität: Eine Bezier-Fläche ist genau dann eben, wenn das Bezier-Kontrollnetz ein ebenes Raster bildet.

Frage 6.11: Wie funktioniert die Graderhöhung bei Bezier-Flächen?

Die Graderhöhung bei Bezier-Flächen benutzt das Verfahren der Graderhöhung von Bezier-Kurven. Angenommen eine Bezier-Fläche mit $(n+1) \cdot (m+1)$ Kontrollpunkten $b_{i,j}, i \in \{0, \ldots, n\}$ und $j \in \{0, \ldots, m\}$ ist, wie weiter oben definiert, gegeben. Das Netz soll nun entlang der Zeilen auf $m+2$ Kontrollpunkte pro

Zeile verfeinert werden. Dann wird in jeder Zeile der Graderhöhungsalgorithmus durchgeführt. Die neue Fläche ist also definiert als:

$$b^*(u,v) = \sum_{i=0}^{n} \left(\sum_{j=0}^{m+1} b^*_{i,j} \cdot B_j^{m+1}(v) \right) \cdot B_i^n(u)$$

Dabei werden die einzelnen $b_{i,j}$ wie folgt ersetzt:

$$\begin{aligned} b^*_{i,0} &= b_{i,0} \\ b^*_{i,j} &= \alpha_j b_{i,j-1} + (1-\alpha_j) b_{i,j}, \text{ mit } \alpha_j = \frac{j}{m+1} \text{ für } j = 1, \ldots, m \\ b^*_{i,m+1} &= b_{i,m} \end{aligned}$$

Frage 6.12: Wie sind B-Spline-Flächen definiert? Wie funktioniert dabei der Algorithmus von de-Boor?

Bei B-Spline-Flächen wird im Prinzip das Gleiche gemacht, wie bei den Bezier-Flächen. Anstatt eines Kontrollpolygonzuges gibt es auch hier ein Netz, wobei dieses zur Unterscheidung gegenüber den Bezier-Flächen aus $(k+1) \cdot (l+1)$ Kontrollpunkten (de-Boor-Punkte) $d_{i,j}$ mit $i \in \{0, \ldots, k\}$ und $j \in \{0, \ldots, l\}$ besteht. Im Prinzip ist die Benennung der Variablen natürlich egal.

Weiterhin sind zwei Grade m und n gegeben, sowie zwei Knotenvektoren $\mu = (u_0, \ldots, u_{k+m+1})$ und $\nu = (v_0, \ldots, v_{l+n+1})$, die denselben Kriterien wie bei offenen bzw. geschlossenen B-Spline-Kurven genügen. Eine B-Spline-Fläche ist dann wie folgt definiert:

$$b(u,v) = \sum_{i=0}^{k} \left(\sum_{j=0}^{l} d_{i,j} \cdot N_j^n(v) \right) \cdot N_i^m(u) \text{ mit } (u,v) \in [u_0, u_{k+1}[\times [v_0, v_{l+1}[$$

Das Prinzip des Algorithmus von de-Boor ist ähnlich wie das des Algorithmus von de Casteljau für Bezier-Flächen. Zuerst werden $(k+1)$ neue Punkte zum Parameterwert v ausgerechnet. Anschliessend werden diese zum Parameter u ausgewertet.

Frage 6.13: Welche Laufzeit hat der Algorithmus von de-Boor für B-Spline-Flächen?

Der Zeitaufwand des Algorithmus von de-Boor beträgt $O(n \cdot m^2 + n^2 + k + l)$. Zuerst muss für das v, welches berechnet werden soll, der entsprechende Index im Knotenvektor der Länge l gefunden werden. Dies dauert $O(l)$, muss jedoch nur einmal gemacht werden, da der Wert später n-mal verwendet werden kann. Anschliessend müssen n neue Punkte in B-Spline-Kurven vom Grad m ausgerechnet werden. Dies dauert $O(n \cdot m^2)$. Nun muss für das u der Index im Knotenvektor gefunden werden ($O(k)$) und der Punkt auf einer Kurve des Grades n berechnet werden. Der letzte Schritt ergibt $O(n^2)$ und alles zusammen die weiter oben behauptete Laufzeit.

Frage 6.14: Welche Eigenschaften besitzen B-Spline-Flächen?

B-Spline-Flächen haben unter anderem die folgenden Eigenschaften:

Parameterlinien: Die Parameterlinien ($u = const$ bzw. $v = const$) sind B-Spline-Kurven mit den de-Boor-Punkten

$$d_j = \sum_{i=0}^{k} d_{i,j} \cdot N_i^m(u) \text{ bzw. } d_i = \sum_{j=0}^{l} d_{i,j} \cdot N_j^n(v).$$

Lokalität: Eine Änderung des de-Boor-Punktes $d_{i,j}$ beeinflusst die Fläche nur im Intervall $[u_i, u_{i+m+1}[\times[v_j, v_{j+n+1}[$. Dies liegt wie auch schon bei den Kurven an den Trägerintervallen des Punktes $d_{i,j}$.

Verfeinerung des Kontrollnetzes: Neue Knotenlinien u^* oder v^* können erzeugt werden, indem der Algorithmus zum Einfügen neuer Knoten in Kurven auf jede Zeile bzw. Spalte des de-Boor-Netzes angesetzt wird.

Konvergenz gegen die Fläche: Bei Iteration des Algorithmus zum Einfügen neuer Knoten werden de-Boor-Netze generiert, die gegen die Fläche konvergierern.

6.2 Volumina

Frage 6.15: Wie sind Volumina definiert? Nenne Beispiele!

Ein Volumen ist eine Menge $V \subset \mathbb{R}^3$, für die gilt, dass ihre abgeschlossene Hülle V^- mit der abgeschlossenen Hülle ihres Kerns V^o (siehe Kapitel 3.5) übereinstimmt. Es gilt also:

$$V^- = (V^o)^-$$

Es gibt unter anderem die folgenden Beispiele:

- Kugel mit Mittelpunkt (m_x, m_y, m_z) und Radius r:

$$\{(x, y, z) | (x - m_x)^2 + (y - m_y)^2 + (z - m_z)^2 \leq r^2\}$$

- Zylinder mit Radius r und Höhe h um die z-Achse:

$$\{(x, y, z) | x^2 + y^2 \leq r^2 \text{ und } 0 \leq z \leq h\}$$

- Quader, der durch $a = (a_x, a_y, a_z)$ und $b = (b_x, b_y, b_z)$ definiert ist:

$$\{(x, y, z) | a_x \leq x \leq b_x, a_y \leq y \leq b_y, a_z \leq z \leq b_z\}$$

Frage 6.16: Welche Möglichkeiten kennst du, Volumen darzustellen?

- Parameterdarstellung
- Quader-Bezier-Segment
- (Polyedrische) Zellzerlegung

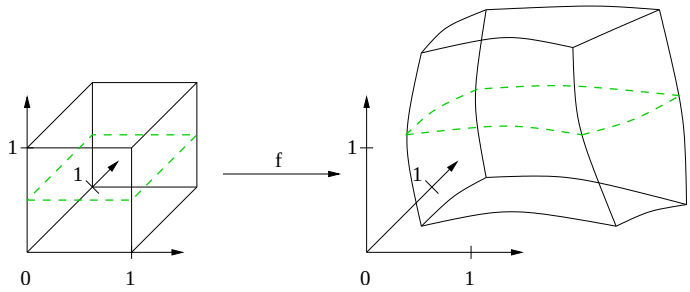


Abbildung 50: Parameterdarstellung von Volumina

Frage 6.17: Wie funktioniert die Parameterdarstellung zur Darstellung von Volumina?

Die Parameterdarstellung eines Volumens in $\mathbb{R}^3$ funktioniert im Prinzip wie die Parameterdarstellung für Kurven und Flächen. Nur ist der Parameterbereich P in diesem Fall eine Teilmenge des $\mathbb{R}^3$, also: $P \subseteq \mathbb{R}^3$. Ein Punkt p ergibt sich wie folgt:

$$p = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x(u,v,w) \\ y(u,v,w) \\ z(u,v,w) \end{pmatrix} = f(u,v,w) \text{ mit } (u,v,w) \in P \subseteq \mathbb{R}^3$$

Einem Punkt (u, v, w) werden also durch die Funktionen $x(u, v, w), y(u, v, w)$ und $z(u, v, w)$ neue x-,y- und z-Koordinaten zugeordnet. Dabei gilt $x, y, z : P \to \mathbb{R}$ und $f : P \to \mathbb{R}^3$. Der Parameterbereich kann zum Beispiel der Einheitswürfel $[0, 1] \times [0, 1] \times [0, 1]$ sein, so wie in Abbildung 50 gezeigt.

Frage 6.18: Wie funktionieren Bezier-Quader-Segmente?

Bezier-Quader-Segmente funktionieren ähnlich wie Bezier-Flächen, nur dass hierbei nun ein Kontrollknoten-Quader benutzt wird. Dieser ist gegeben durch $(l+1)\cdot(m+1)\cdot(n+1)$ Bezier-Kontrollpunkte $b_{i,j,k}$. Dies entspricht also im Prinzip dem Übereinanderlegen von $(n + 1)$ Flächen, die jeweils durch $(l + 1) \cdot (m + 1)$ Knoten gegeben sind. Soll nun ein Punkt des Volumens zum Parameterwert (u, v, w) berechnet werden, so werden zuerst $(n + 1)$ Punkte ausgerechnet, indem für jede Fläche der Punkt zum Parameterwert (u, v) berechnet wird (siehe Kapitel 6.1). Diese $(n + 1)$ Punkte werden wiederum als Kontrollpunkte interpretiert, für welche der Punkt zum Parameter w errechnet wird.

Frage 6.19: Was ist ein Polyeder?

Ein Polyeder ist eine Punktmenge im $\mathbb{R}^d$, die durch eine d-dimensionale simpliziale Zerlegung beschrieben werden kann. Die Darstellung erfolgt dabei als eine Hierarchie von Seitenpolyedern. Dabei setzt sich der Rand aus endlich vielen Seitenpolyedern zusammen, deren Rand wiederum aus Seitenpolyedern und so weiter, bis schließlich die Eckpunkte erreicht sind. Eckpunkte sind nulldimensionale Polyeder.

Frage 6.20: Was ist eine simpliziale Zerlegung im $\mathbb{R}^d$?

Eine simpliziale Zerlegung im $\mathbb{R}^d$ ist eine Menge von Simplexen im $\mathbb{R}^d$, so dass folgendes gilt:

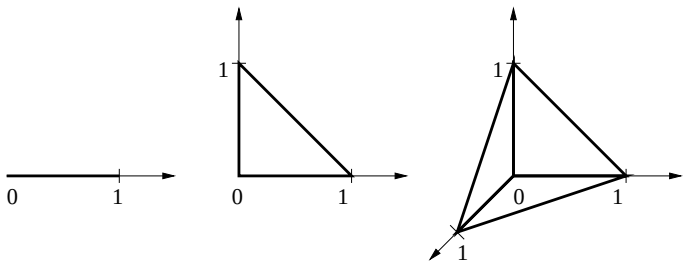


Abbildung 51: Polyeder in $\mathbb{R}$, $\mathbb{R}^2$ und $\mathbb{R}^3$

- Der Durchschnitt von je zwei Simplexen ist entweder leer oder ein Seitensimplex beider Simplexe.
- Mit jedem Simplex gehören auch alle seine Seitensimplexe zur gegebenen Simplexmenge.

Frage 6.21: Was ist ein d-dimensionales Simplex?
Ein d-dimensionales Simplex im $\mathbb{R}^d$ ist eine Punktmenge, die zur Menge S_d affin äquivalent ist, d.h. die beiden Mengen gehen durch eine affine Abbildung (Translation, Verzerrung, etc.) auseinander hervor. Dabei ist S_d definiert als:

$$S_d = \{p | p = (p_1, \ldots, p_d), 0 \leq p_1, \ldots, p_n \leq 1, p_1 + \cdots + p_n \leq 1\}$$

Ein eindimensionales Simplex ist eine Strecke zwischen 0 und 1. Ein zweidimensionales Simplex ist ein Dreieck und ein dreidimensionales Simplex ein **Tetraeder**. Abbildung 51 zeigt diese Simplexe. Simplexe bilden quasi eine Hierarchie. So ist ein Tetraeder durch Dreiecke umrandet, welche zweidimensionale Simplexe sind. Diese wiederum sind durch Kanten umrandet, welche durch zwei Punkte umrandet werden. Diese sukzessive Zerlegung gilt für beliebige Dimensionen. Allgemein gilt, dass ein d-dimensonales Simplex durch $d+1$ $(d-1)$-dimensionale Simplexe umrandet wird. Diese wiederum sind durch d $(d-2)$-dimensionale Simplexe umrandet (für ein festes d) und so fort.

Frage 6.22: Wie funktioniert die polyedrische Zellzerlegung?
Eine polyedrische Zellzerlegung ist eine Menge von Polyedern im $\mathbb{R}^d$ mit den folgenden Eigenschaften:

- Der Durchschnitt von je zwei Polyedern ist leer oder ein Seitenpolyeder beider Polyeder.
- Mit jedem Polyeder gehören auch alle seine Seitenpolyeder zur gegebenen Polyedermenge.

Frage 6.23: Wie können Zellzerlegungen repräsentiert bzw. gespeichert werden?
Zellzerlegungen können mit Hilfe von Zellinzidenzgraphen abgespeichert werden. Ein Zellinzidenzgraph ist ein Graph $I = (V, E)$. Dabei ist die Menge der Knoten V eine disjunkte Vereinigung der Teilmengen V_i für $i = -1, \ldots, d+1$.

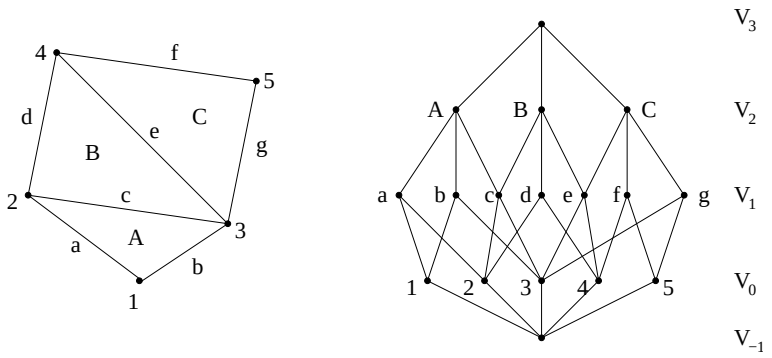


Abbildung 52: Beispiel für einen Zellinzidenzgraphen

Die Knotenmengen V_{-1} und V_{d+1} enthalten dabei nur einen Knoten und dienen dabei nur als Einstieg, bzw. Endpunkt der Datenstruktur und enthalten sonst keine Informationen. In den Knotenmengen $V_i, i = 0, \ldots d$ werden die i-dimensionalen Zellen gespeichert, d.h. in V_0 werden zum Beispiel alle Punkte gespeichert, in V_1 alle Kanten, usw.

Die Menge E ist ebenfalls eine disjunkte Vereinigung von Teilmengen E_i mit $i = -1, \ldots, d$. Die Kantenmenge E_i enthält dabei Kanten zwischen den Knotenmengen V_i und V_{i+1}. Dabei ist ein Knoten $v \in V_i$ mit einem Knoten $w \in V_{i+1}$ verbunden, wenn v eine Randzelle von w ist. Der Knoten in V_{-1} ist mit allen Knoten in V_0 verbunden und alle Knoten in V_d sind mit allen Knoten in V_{d+1} verbunden.

Frage 6.24: Wie können Polyeder gespeichert werden?

Polyeder können mir der Winged-Edge Datenstruktur abgespeichert werden. Ein dreidimensionaler Polyeder besteht aus Knoten, Kanten und Flächen. Die Flächen werden durch einige Kanten begrenzt, welche wiederum durch Knoten begrenzt werden. Bei dieser Datenstruktur werden drei Tabellen benutzt: die Kantentabelle, welche die wichtigste der drei Tabellen ist, sowie die Knoten- und Flächentabelle.

In der Kantentabelle werden für jede Kante einige Informationen gespeichert. So zum Beispiel durch welche Knoten sie begrenzt ist und welche beiden Flächen an ihr angrenzen. Wird die Kante als gerichtet und der Polyeder von außen betrachtet, so steht in diesem Fall eindeutig fest, welche der beiden Flächen links und welche rechts von der Kante liegt. In Abbildung 53 ist die Kante a als eine Kante von X nach Y definiert (siehe auch Tabelle 4). Von außen betrachtet liegt daher die Fläche 1 links von ihr und die Fläche 2 rechts. Diese beiden Flächen können nun, jeweils bei a beginnend, entlang ihrer Kanten abgelaufen werden. Auch hier wird wieder die äußere Seite einer Fläche betrachtet und im Uhrzeigersinn abgelaufen. Für jede Fläche gibt es dann eine Kante, welche abgelaufen wird, bevor über die Kante a gelaufen wird und eine Kante, die nach a abgelaufen wird. Für jede Fläche wird, wie auch in Tabelle 53 geschehen, dieser Vorgänger und Nachfolger ebenfalls gemerkt.

Die beiden anderen Tabellen enthalten jeweils einen Eintrag für jeden Knoten und für jede Fläche. In der Knotentabelle wird für jeden Knoten eine Kante

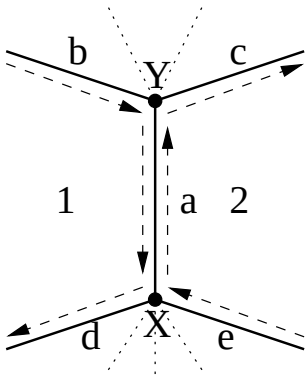


Abbildung 53: Beispiel für die Winged-Edge Datenstruktur

Kante	Punkte		Flächen		Links		Rechts	
Name	Anfang	Ende	Links	Rechts	V	N	V	N
a	X	Y	1	2	b	d	e	c

Tabelle 4: Kantentabelleneintrag für die Kante a aus Abbildung 53

eingetragen, die zu diesem inzident ist. In Abbildung 53 könnte der Eintrag für X z.B. die Kante c sein. Und in der Flächentabelle wird für jede Fläche genau eine Kante vermerkt, welche diese begrenzt.

Mit Hilfe dieser drei Datenstrukturen ist es nun möglich, ziemlich viele Anfragen relativ effizient zu beantworten, wie z.B. welche Kanten alle zu einer Fläche adjazent sind. Einige Anfragen können sogar in konsanter Zeit beantwortet werden.

Frage 6.25: Was ist eine Triangulierung? Wo wird sie benötigt?

Die Triangulierung einer Punktmenge $p_1, \ldots, p_n \in \mathbb{R}^d$ ist eine simpliziale Zerlegung der konvexen Hülle der Punkte $p_1, \ldots, p_n$, die genau diese gegebenen Punkte als Eckpunkte (nulldimensionale Simplexe) enthält.

Diese Technik wird zum Beispiel benötigt, wenn ein Objekt mit einer Laserscanner abgetastet wurde und aus dieser „Punktwolke“ an Informationen eine Darstellung erzeugt werden soll.

Frage 6.26: Was ist eine Delaunay-Triangulierung? Was ist in diesem Zusammenhang ein Delaunay-Simplex?

Eine Delaunay-Triangulierung ist eine Triangulierung, welche nur aus Delaunay-Simplexen bezüglich der gegebenen Punktmenge besteht. Ein Delaunay-Simplex im $\mathbb{R}^d$ ist ein Simplex mit k von n Punkten als Eckpunkten, wobei $1 \leq k \leq d+1$ gilt, welches bezüglich der gegebenen Punkten die Umkugelbedingung erfüllt. Dies bedeutet, dass sich keiner der anderen Punkte innerhalb der von den maximal $d+1$ Eckpunkten des Simplexes erzeugten Hyperkugel befindet.

Im $\mathbb{R}^2$ ist ein Delaunay-Simplex ein Dreieck, in dessen Umkreis sich keiner der anderen Punkte befindet. Ein Delaunay-Simplex im $\mathbb{R}^2$ hat höchsten 3 Eckpunkte und eine zweidimensionale Hyperkugel ist ein Kreis. Abbildung 54(a) zeigt eine gültige Delaunay-Triangulierung im $\mathbb{R}^2$. Die vier Punkte der Ebene

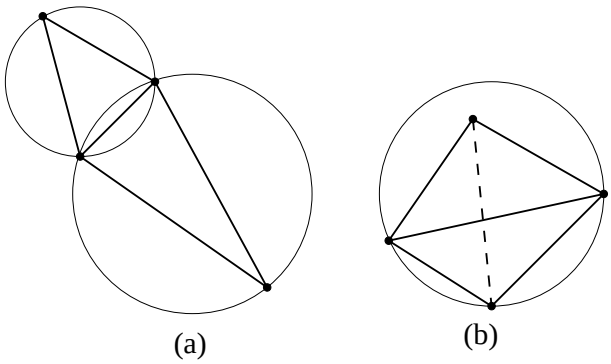


Abbildung 54: Eine gültige (a) und eine ungültige (b) Delaunay-Triangulierung

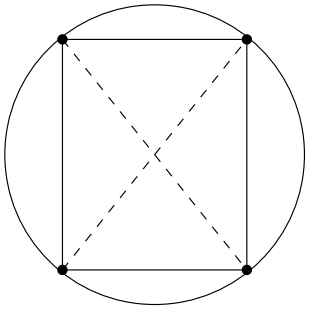


Abbildung 55: Eine nicht eindeutige Delaunay-Triangulierung

wurden in zwei Dreiecke trianguliert, welche beide Delaunay-Simplexe sind, der sich der jeweilige vierte Punkt nicht im Umkreis der Dreiecke befindet. Abbildung 54(b) ist hingegen keine gültige Delaunay-Triangulierung, da sich der vierte Punkt im Umkreis der anderen drei Punkte befindet. Eine Aufteilung der Dreiecke entlang der gestrichelten Linie würde in einer Delaunay-Triangulierung resultieren.

Frage 6.27: Welche Eigenschaften hat die Delaunay-Triangulierung?
Die Delaunay-Triangulierung hat die folgenden zwei Eigenschaften:

Existenz: Zu jeder endlichen Punktmenge im $\mathbb{R}^d$ gibt es eine Delaunay-Triangulierung.

Eindeutigkeit: Sind keine $d+2$ der gegebenen Punkte kosphärisch, dann ist die Delaunay-Triangulierung eindeutig. Punkte sind kosphärisch, wenn eine Hyperkugel existiert, auf der sie alle liegen. Im $\mathbb{R}^2$ ist die Triangulierung also nicht eindeutig, wenn vier Punkte auf einem Kreis liegen. Dies kann man sich recht schnell an Abbildung 55 klarmachen. Die Punkte liegen alle auf dem Umkreis und daher besteht die Wahlmöglichkeit, entlang welcher der beiden gestrichelten Linien trianguliert wird.

Frage 6.28: Wie kann die Delaunay-Triangulierung berechnet werden?
Die Delaunay-Triangulierung in einer beliebigen Dimension kann durch eine Berechnung der konvexen Hüllen einer um eine Dimension höheren Punktmenge

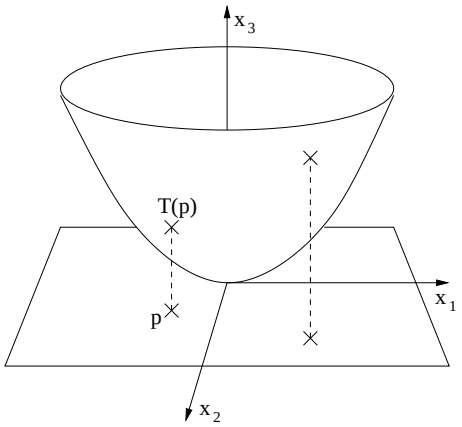


Abbildung 56: Transformation einer Delaunay-Triangulierung im $\mathbb{R}^2$ ($d = 2$)

berechnet werden. Zuerst müssen die Punkte $p_1, \ldots, p_n$ mit Hilfe einer Abbildung (oder Transformation) in einer höhere Dimension abgebildet werden. Dies geschieht wie folgt:

$$T(p) = \begin{pmatrix} p \\ \sum_{i=1}^{d} p_i^2 \end{pmatrix}, p = \begin{pmatrix} p_1 \\ \vdots \\ p_d \end{pmatrix}$$

$\sum_{i=1}^{d} p_i$ ist dabei nichts anderes als ein Abstandmessung zum Ursprung, wobei jedoch nicht die Wurzel gezogen wird. Trotzdem reicht diese Formel, um relative Abstände zum Ursprung zu messen. Je weiter ein Punkt also vom Ursprung entfernt ist, desto größer wird dieser Wert. Abbildung 56 zeigt eine solche Transformation für den Fall $d = 2$. Ein Punkt in der Ebene wird per $T(p)$ auf die „Schüssel“ projeziert. Diese Schüssel ergibt sich, da alle Punkte der Ebene, welche den gleichen Abstand zum Ursprung haben, auf die gleiche Höhe abgebildet werden.

Ist diese Transformation für alle Punkte vollzogen, so wird die konvexe Hülle der Bildpunkte $T(p_i), i = 1, \ldots n$ berechnet. Im $\mathbb{R}^3$ würde diese konvexe Hülle aus Dreiecken bestehen. Die Delaunay-Triangulierung ist dann kombinatorisch äquivalent zu der Zellzerlegung, die von den d-dimensionalen Zellen induziert wird, deren äußerer Normalenvektor eine negative $(d+1)$-Komponenten hat. Auf deutsch: Während der Berechnung der konvexen Hülle entstehen d-dimensionale Zellen, d.h. im Beispiel Dreiecke. Es werden nun alle die Dreiecke betrachtet, welche von der Ebene, die von x_1 und x_2 aufgespannt wird (siehe Abbildung 56), sichtbar sind. Wenn also eine Zelle bestehend aus den Punkten $T(p_1)$, $T(p_2)$ und $T(p_3)$ sichtbar ist, so bilden das Dreieck der Punkte p_1, p_2 und p_3 ein Delaunay-Simplex.

Frage 6.29: Wie kann die Delaunay-Triangulierung im $\mathbb{R}^2$ berechnet werden?

Im zweidimensionalen Fall einer Delaunay-Triangulierung kann ein inkrementeller Algorithmus verwendet werden. Gegeben seien die Punkte $P_1, \ldots, P_n$. Dann läuft der Algorithmus in folgenden Schritten ab:

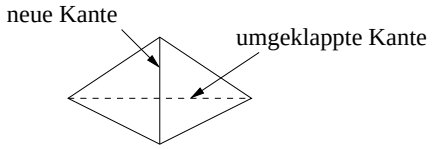


Abbildung 57: Umklappen einer Kante

1. Trianguliere die ersten drei Punkte P_1, P_2 und P_3.
2. Für die restlichen Punkte $P_i,\ i \in \{P_4, \ldots, P_n\}$:
   - Setze den Punkt P_i und schaue, in welches Gebiet er fällt.
   - Verbinde P_i mit allen möglichen Eckpunkten des Gebiets zu einer Triangulierung.
   - So lange es eine Kante e gibt, die P_i gegenüberliegt und keine Delaunay-Kante ist, klappe diese Kante um.

Eine Delaunay-Kante ist eine Kante, für die ein Umkreis exisitert, so dass keiner der anderen Punkte im Inneren dieses Umkreis liegt. Eine Kante e wird umgeklappt, indem die Diagonale des Vierecks, welches durch die beiden zur Kante e inzidenten Dreiecke definiert ist, durch die andere Viereckdiagonale ersetzt wird.

6.3 Reguläre Zellzerlegung

Frage 6.30: Was ist ein Rastermodell?

Ein d-dimensionales Rastermodell ist ein $n_1 \times \cdots \times n_d$ Feld mit $R[i_1 \ldots i_d] \in \mathbb{R}_0^+,\ i_k = 0, \ldots n_k - 1$ für $k = 1, \ldots, d$. Dies bedeutet das die n_i jeweils die Auflösung in eine Richtung angeben und $R[i_1 \ldots i_d]$ jeweils den Wert des Pixels an der Stelle $(i_1 \ldots i_d)$, wobei dieser Wert größer oder gleich Null sein muss. Die Koordinaten laufen jeweils von 0 bis $n_i - 1$.

Bestehen die Einträge nur aus endlich vielen Werten, z.B. $\{0, \ldots, m-1\}$, so spricht man von einem quantisierten Rastermodell. m heißt in diesem Zusammenhang **Quantisierungsauflösung**. Ein Spezialfall hiervon ist ein binäres Rastermodell, bei dem die Einträge aus der Menge $\{0, 1\}$ kommen.

Ein zweidimensionales binäres Rastermodell wird auch binäres Pixelmodell genannt, wobei die Rasterelement als **Pixel** („picture element“) bezeichnet werden. Ein digitales Geländemodell kann mit Hilfe eines zweidimensionalen Rastermodells realisiert werden, bei dem die Elemente als Höhenwerte interpretiert werden. Ein **Voxelmodell** ist ein dreidimensionales Rastermodell, bei dem die einzelnen Rasterelemente als **Voxel** bezeichnet werden.

Frage 6.31: Was ist das „Marching-Cubes“-Verfahren und wofür wird es benötigt?

Das „Marching-Cubes“-Verfahren dient dazu, aus einem gegebenen Voxelmodell eine Oberflächenapproximation, d.h. ein Darstellung in Form von Polygonen zu berechnen. Ein Voxelmodell besteht aus einem dreidimensionalen Koordinatengitter, an dessen Schnittpunkten sich die Bildelemente, die sogenannten

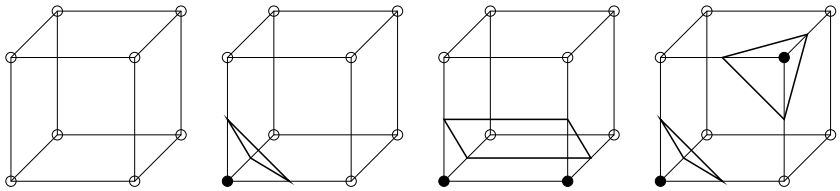


Abbildung 58: Beispiele für elementare Konfigurationen des „Marching-Cubes“-Verfahren

Voxel, befinden. Jedes Voxel speichert einen Intensitätswert, z.B. zwischen 0 und 255. Es ist nun möglich, je acht Voxel zu einem Würfel zusammenzufassen, der an den Eckpunkten verschiedene Intensitätswerte haben kann. Beim „Marching Cubes“-Verfahren wird das Voxelmodell nun mit diesen Würfeln abgelaufen und für jedes Voxel geschaut, ob sein Wert über oder unter einem gewissen Schwellwert liegt. Liegt ein Voxel unter einem Schwellwert, so gilt es als außen liegend, ansonsten liegt es innen. In Abbildung 58 sind Voxel, die innen liegen, schwarz gefärbt und Voxel, die außen liegen, weiß. Da jedes der acht Voxel an den Eckpunkten des Würfels entweder innen oder außen liegt, gibt es $2^8 = 256$ verschiedene Möglichkeiten oder Konfigurationen, von denen einige jedoch symmetrisch sind. Für jede dieser Konfigurationen werden nun Polygone entsprechend der Oberflächenstruktur an diesem Würfel gezeichnet. Abbildung 58 zeigt einige Beispiele. Beim zweiten Würfel von links wird z.B. nur ein Dreick an der Ecke gezeichnet, die innen liegt.

7 Dreidimensionale Transformation und Projektion

Frage 7.1: Wie funktionieren die Transformationen in drei Dimensionen?
Die dreidimensionalen Transformationen funktionieren im Prinzip genau wie die Transformationen in zwei Dimensionen, nur dass eine zusätzliche Komponente hinzu kommt. Die Rotationen ist ebenfalls etwas trickreicher, da nun die Achse angegeben werden muss, um die rotiert werden soll. Es wird sofort die homogene Darstellung angegeben, welche die gleichen Vorteile wie in zwei Dimensionen bietet.

Translation: Eine Translation um den Vektor t: $\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Skalierung: Skalierung um s_x, s_y, s_z: $\begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Rotation: Im Raum muss zusätzlich angegeben werden, um welche Achse rotiert wird, wodurch sich für jede Achse eine andere Rotationsmatrix ergibt, die sich jedoch recht ähnlich sehen. Es wird hier jedoch nur die nicht homogene Matrix angegeben:

$$\underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix}}_{\text{Rotation um } x\text{-Achse}}, \underbrace{\begin{pmatrix} \cos\phi & 0 & \sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{pmatrix}}_{\text{Rotation um } y\text{-Achse}}, \underbrace{\begin{pmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{Rotation um } z\text{-Achse}}$$

Frage 7.2: Was ist eine Projektion? Was ist der Unterschied zwischen Perspektiv- und Parallelprojektion?
Eine Projektion ist die Abbildung von Punkten aus einem n-dimensionalen Koordinatensystem in ein $(n-1)$-dimensionales Koordinatensystem. Hier wird jedoch nur die Projektion von drei auf zwei Dimensionen besprochen. In diesem Fall wird die Projektion eines dreidimensionalen Objekts durch die sogenannten **Projektoren** bestimmt. Dies sind gerade Projektionsstrahlen, die ihren Ursprung in einem **Projektionszentrum** haben. Sie laufen durch jeden zu projizierenden Punkt und schneiden schließlich eine **Projektionsebene**, wodurch die Projektion erzeugt wird.

Bei der Perspektivprojektion ist der Abstand zwischen Projektionszentrum und Projektionsebene endlich. In einigen Fällen kann es jedoch praktisch sein, anzunehmen, dass der Abstand zwischen Projektionszentrum und Projektionsebene unendlich weit ist. In diesem Fall verlaufen die Projektoren alle parallel zueinander, was der Parallelprojektion ihren Namen gibt.

Frage 7.3: Wie funktioniert die perspektivische Projektion?
Bei der perspektivischen Projektion wird ein Augenpunkt a angegeben, in

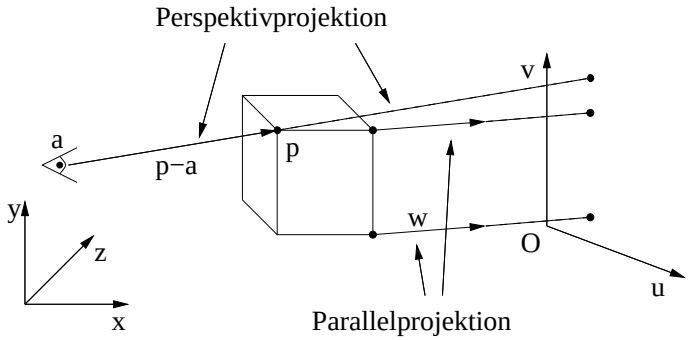


Abbildung 59: Perspektiv- und Parallelprojektion

dem die Projektoren zusammentreffen. Die Projektion ist also abhängig von der Lage des zu projizierenden Punktes p relativ zum Augenpunkt. Die Ebene ist durch einen Ursprung O gegeben und zwei Vektoren u und v, welche die Ebene aufspannen, auf die projiziert werden soll. Der Schnitt mit der Ebene wird über folgende Gleichung gelöst:

$$a + \overline{p}_z \cdot (p - a) = O + \overline{p}_x \cdot u + \overline{p}_y \cdot v$$

Abbildung 59 verdeutlicht, wie es zu dieser Gleichung kommt. Der Differenzvektor zwischen dem Punkt p und dem Augenpunkt a gibt an, in welche Richtung projiziert werden soll. Dieser Vektor wird so lange verlängert ($\overline{p}_z$), bis er die Ebene schneidet. Dabei geben die Werte $\overline{p}_x$ und $\overline{p}_y$ an, wie die Vektoren u und v vom Ursprung O aus verlängert werden müssen, damit eben dieser Schnittpunkt gegeben ist. Die Gleichung kann weiter umgeformt werden, so dass sich ein lineares Gleichungssystem mit drei Gleichungen und drei Unbekannten ergibt:

$$\overline{p}_x \cdot u + \overline{p}_y \cdot v + \overline{p}_z \cdot (a - p) = a - O$$

Das resultierende lineare Gleichungssystem sieht in Matrix-Vektor-Schreibweise wie folgt aus:

$$\begin{pmatrix} u_x & v_x & (a-p)_x \\ u_y & v_y & (a-p)_y \\ u_z & v_z & (a-p)_z \end{pmatrix} \begin{pmatrix} \overline{p}_x \\ \overline{p}_y \\ \overline{p}_z \end{pmatrix} = a - O$$

Es kann die Cramer'schen Regel benutzt werden, um dieses Gleichungssystem zu lösen. Diese besagt, dass bei einem Gleichungssystem $A \cdot x = b$ mit n Gleichungen und n Unbekannten und einer regulären Koeffizientenmatrix A, sich die k-te Komponente des Lösungsvektors x ergibt als:

$$x_k = \frac{\det A_k(b)}{\det A}$$

Die Matrix $A_k(b)$ ergibt sich dadurch, dass in der Matrix A die k-te Spalte durch den Ergebnisvektor b ersetzt wird. Im Falle des Gleichungssystem der

perspektivischen Projektion ergibt sich also:

$$\overline{p}_x = \frac{\det\begin{pmatrix} (a-O) & v & (a-p) \end{pmatrix}}{\det\begin{pmatrix} u & v & (a-p) \end{pmatrix}}$$
$$\overline{p}_y = \frac{\det\begin{pmatrix} u & (a-O) & (a-p) \end{pmatrix}}{\det\begin{pmatrix} u & v & (a-p) \end{pmatrix}}$$
$$\overline{p}_z = \frac{\det\begin{pmatrix} u & v & (a-O) \end{pmatrix}}{\det\begin{pmatrix} u & v & (a-p) \end{pmatrix}}$$

Die Werte $\overline{p}_x$ und $\overline{p}_y$ sind die Koordinaten des projizierten Punktes p im u-v-System. Der Wert $\overline{p}_z$ enthält Tiefeninformation. Denn von den Punkten, die alle auf der durch $a + \overline{p}_z \cdot (p - a)$ definierten Geraden liegen, liefern die näher an der Projektionsebene liegenden Punkte kleinere $\overline{p}_z$-Werte als die Punkte, welche weiter weg liegen.

Frage 7.4: Wie funktioniert die Parallelprojektion?

Die Parallelprojektion funktioniert so ähnlich wie die Perspektivprojektion. Nur muss kein Differenzvektor zwischen dem zu projizierenden Punkt und einem Augenpunkt errechnet werden, da die Projektionsrichtung für alle Punkte durch den Richtungsvektor w vorgegeben ist (siehe Abbildung 59). Dadurch ergibt sich folgende Formel:

$$p + \overline{p}_z \cdot w = O + \overline{p}_x \cdot u + \overline{p}_y \cdot v$$

Diese Formel wird, wie schon bei der perspektivischen Projektion, umgestellt:

$$\overline{p}_x \cdot u + \overline{p}_y \cdot v - \overline{p}_z \cdot w = p - O$$

Auch in diesem Fall wird die Cramer'sche Regel angewendet und es ergibt sich:

$$\overline{p}_x = \frac{\det\begin{pmatrix} (p-O) & v & -w \end{pmatrix}}{\det\begin{pmatrix} u & v & -w \end{pmatrix}}$$
$$\overline{p}_y = \frac{\det\begin{pmatrix} u & (p-O) & -w \end{pmatrix}}{\det\begin{pmatrix} u & v & -w \end{pmatrix}}$$
$$\overline{p}_z = \frac{\det\begin{pmatrix} u & v & (p-O) \end{pmatrix}}{\det\begin{pmatrix} u & v & -w \end{pmatrix}}$$

8 Entfernung verdeckter Kurven und Flächen

Frage 8.1: Worum geht es beim Entfernen verdeckter Flächen? Welche zwei Ansätze kann man unterscheiden?

Bei der Ermittlung sichtbarer Kurven und Flächen geht es darum, nur die Kanten und Flächen darzustellen, die bezüglich des Projektionszentrums bzw. der Projektionsrichtung sichtbar sind. Kanten werden also als Begrenzung von nicht durchsichtigen Flächen betrachtet, welche andere Kanten verdecken können, wenn diese weiter vom Betrachter entfernt sind. Die Lösung dieses Problems kann sehr rechenintensiv sein, weshalb auch sehr viel in diese Richtung geforscht wurde.

Grob kann zwischen zwei Ansätzen unterschieden werden. Gegeben seien n Objekte, welche aus mehreren Polygonen bestehen können. Es kann nun für jedes Pixel entschieden werden, welches Objekt in ihm sichtbar ist. Das heißt, es wird für jedes Pixel das Objekt ermittelt, welches am nächsten am Betrachter liegt und vom Projektor durch das entsprechende Pixel getroffen wird. Solche Ansätze werden als **Algorithmen mit Bildgenauigkeit** bezeichnet. Wird der Algorithmus für p Pixel ausgeführt, wobei diese Anzahl in die Millionen gehen kann, liegt die Laufzeit also ungefähr bei $O(np)$.

Beim anderen Ansatz werden die Objekte untereinander verglichen und für jedes Objekt die Teile bestimmt, welche nicht durch andere Teile des Objekts oder durch andere Objekte verdeckt werden. Bei einem solchen Vorgehen wird von **Algorithmen mit Objektgenauigkeit** gesprochen. Die Laufzeit beträgt $O(n^2)$. Auf den ersten Blick scheinen die Algorithmen mit Objektgenauigkeit für $n < p$ effizienter zu sein, jedoch sind die einzelnen Schritte für diese Algorithmen oft sehr komplex und rechenintensiv.

Frage 8.2: Wie kann für Flächen in Parameterdarstellung eine Sichtbarkeitsberechnung durchgeführt werden?

Ähnlich wie bei Kurven in Parameterdarstellung.

Frage 8.3: Wie können komplexe Polygone für die Sichtbarkeitsberechnung trianguliert werden?

Neben der Delaunay-Triangulation gibt es noch weitere Möglichkeiten ein Polygon zu triangulieren. Ist ein einfaches Polygon gegeben, so kann dies trianguliert werden, indem ihm sukzessive die Ohren abgeschnitten werden. Ein Ohr ist ein von drei aufeinander folgeden Eckpunkten gebildetes Dreieck, welches im Inneren des Polygons liegt und keinen Eckpunkt enthält.

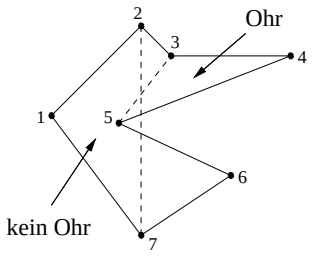


Abbildung 60: Ohren eines Polygons

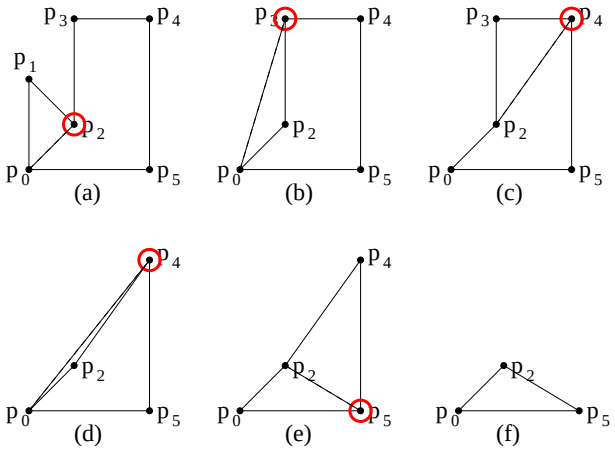


Abbildung 61: Beispiel für den Algorithmus zum Entfernen von Ohren

Abbildung 60 gibt Beispiele dafür, was ein Ohr ist und was nicht. Die Punkte 3, 4 und 5 bilden ein Ohr, da das Dreieck, welches sie bilden, im Inneren des Polygons liegt und sich kein anderer Eckpunkt innerhalb dieses Dreiecks befindet. Bei dem durch die Punkte 1, 2 und 7 gebildeten Dreieck ist die nicht der Fall, da sich der nichtkonvexe Punkt 5 innerhalb dieses Dreiecks befindet.

Wie schon erwähnt, kann ein Polygon trianguliert werden, indem ihm sukzessive die Ohren abgeschnitten werden. Der Algorithmus dazu funktioniert wie folgt. Gegeben sei ein einfaches Polygon mit den Eckpunkten $p_0, \ldots, p_n$. Es werden die folgenden Datenstrukturen benutzt.

- Eine Menge D von Diagonalen, welche am Ende des Algorithmus die Triangulierung definieren werden. Zu Beginn ist diese Menge leer.
- Die Menge R der nichtkonvexen Eckpunkte des Polygons, welche während des Algorithmus ständig aktualisiert wird.
- Eine Menge P, in welcher die restlichen Punkte, des bisher noch nicht triangulierten Restpolygons gemerkt werden. Zu Beginn besteht diese Menge aus dem ganzen Polygon.

Während des Algorithmus wird immer ein aktueller Punkt p aus P betrachtet, für den geschaut wird, ob er an einem Ohr beteiligt ist. Zu Beginn ist dieser Punkt p der Punkt p_2. Solange der aktuelle Punkt nicht p_0 ist, wird folgendes gemacht:

- Wenn das restliche Polygon nicht nur ein Dreieck ist, schau nach, ob der aktuelle Punkt p mit seinem Vor- und Vorvorgänger ein Ohr bildet.
- Ist dies der Fall mache folgendes:
  - Speichere die Diagonale zwischen dem aktuellen Punkt und seinem Vorvorgänger in D.

  - Entferne den Vorgänger des aktuellen Punkt aus P. In diesem Schritt wird also das Ohr abgeschnitten. Desweiteren bekommt der aktuell betrachtete Punkt dadurch einen neuen Vorgänger, nämlich den alten Vorvorgänger.
  - Schaue nach, ob sich die Konvexeigenschaften für den Punkt p und seinen (neuen) Vorgänger geändert haben. Aktualisiere R entsprechend.
  - Wenn der Vorgänger des aktuellen Punktes nun der Punkt p_0 ist, setze den Nachfolger des aktuellen Punktes als neuen aktuellen Punkt, ansonsten nicht.

- Bildet der aktuelle Punkt kein Ohr mit seinen Vorgängern, setze den nächsten Punkt als aktuellen Punkt.

Abbildung 61 zeigt die Funktionsweise des Algorithmus anhand eines Beispiels. Der aktuelle Punkt ist dabei jeweils eingekreist. Begonnen wird mit Punkt p_2, der mit seinen Vorgängern ein Ohr bildet. Dieses wird abgeschnitten, indem p_1 entfernt wird. Sein neuer Vorgänger ist nun p_0, weshalb sein Nachfolger p_3 der neue aktuelle Punkt wird. Da dieser mit seinen Vorgängern kein Ohr bildet (das Dreieck liegt nicht im Inneren), wird p_4 zum neuen aktuellen Punkt. Dieser bildet mit seinen Vorgängern ein Ohr, welches abgeschnitten wird. Der neue Vorgänger von p_4 ist nicht p_0, also bleibt er der aktuelle Punkt. Im nächsten Schritt bildet er jedoch kein Ohr mit seinen Vorgängern, weshalb p_5 zum aktuellen Punkt wird. Dieser bildet ein Ohr, welches abgeschnitten wird und nur noch ein Dreieck übrig lässt, womit der Algorithmus terminiert.

Frage 8.4: Welche Verfahren kennst du, um die Sichtbarkeit konvexer Polygone zu berechnen?

- Brute-Force
- Tiefenpuffer (Z-Buffer)
- Scan-Line
- Bereichsunterteilung
- Prioritätslisten

8.1 Brute-Force-Verfahren

Frage 8.5: Wie funktioniert das Brute-Force-Verfahren?
Das Brute-Force-Verfahren ist ein Algorithmus mit Objektgenauigkeit. Er ist jedoch, wie viele Brute-Force-Verfahren, nicht sehr intelligent und damit auch nicht besonders effizient. Die Grundidee besteht darin, jedes Polygon gegen alle anderen Polygone der Szene zu clippen, um zu sehen, welche Teile von ihm bezüglich der anderen Polygone sichtbar sind. Dabei wird davon ausgegangen, dass die Polygone konvex sind. Sind sie es nicht, können sie durch Triangulierung konvex gemacht werden (siehe oben). Wenn ein Polygon gegen ein anderes

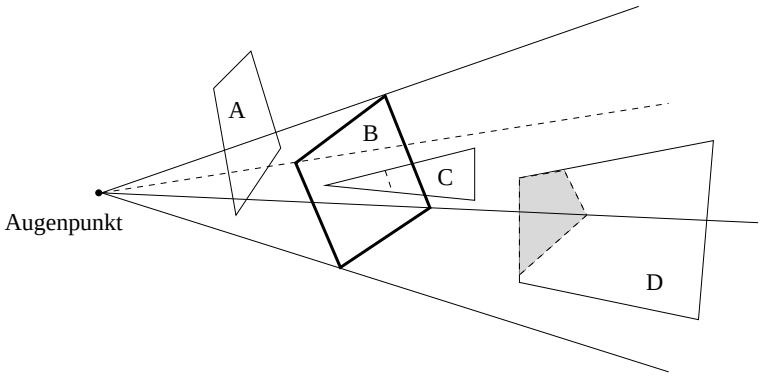


Abbildung 62: Brute-Force-Algorithmus zur Ermittlung sichtbarer Flächen

geclippt wird, kann es unter Umständen in mehrere Teilpolygone zerfallen. Der Algorithmus arbeitet nun so, dass er ein Polygon nimmt und dieses gegen alle anderen Polygone der Szene clippt. Zerfällt das Polygon dabei, so werden sich die verschiedenen Teile gemerkt und jeder Teil weiter gegen die restlichen, noch zu betrachtenden Polygone geclippt. Der Algorithmus, der ein Polygon gegen ein anderes clippt, wird gleich angegeben. Sind die sichtbaren Teile eines Polygones ermittelt worden, so werden diese gezeichnet und das nächste Polygon bearbeitet, also wieder gegen alle anderen geclippt.

Abbildung 62 zeigt, welche Fälle auftreten können, wenn ein Polygon gegen ein anderes geclippt wird. Das dicker gezeichnete Polygon B ist in diesem Fall das Polygon, gegen welches geclippt wird. Es verdeckt die anderen Polygone also unter Umständen. Polygon A ist ein Polygon, welches überhaupt nicht von B verdeckt wird und daher bezüglich B komplett sichtbar ist. Polygon C wird teilweise von B verdeckt. Da es B jedoch durchdringt, sind einige Teile von ihm auch sichtbar. Das Polygon D liegt hinter B und durchdringt dieses auch nicht. Einige Teile (grau gezeichnet) werden von B verdeckt, d.h. das Polygon D ist nicht komplett sichtbar. Alle diese Fälle werden im gleich folgenden Algorithmus überprüft, wobei erst geschaut wird, ob ein Polygon überhaupt verdeckt werden kann. Ist dies der Fall, werden die noch sichtbaren Teile des Polygons berechnet und abschließend überprüft, ob das Polygon eventuell durchdringend ist.

Nun zum Clippen eines Polygons gegen ein anderes. Im folgenden wird das Polygon, gegen welches geclippt wird, Verdeckpolygon genannt und besteht aus den k Eckpunkten $p_1, \ldots, p_k$. Damit hat es auch k Kanten. Das Polygon, welches eventuell vom Verdeckpolygon verdeckt wird, wird Schnittpolygon genannt. Es besteht aus den l Eckpunkten $q_1, \ldots, q_l$. Zu Beginn des Algorithmus werden einige Ebenen berechnet. Und zwar k Ebenen $E_1, \ldots, E_k$, wobei sich die Ebene E_i aus dem Augenpunkt und den Punkten p_i und p_{i+1} des Verdeckpolygons ergibt. Desweiteren wird die Ebene E berechnet, die sich aus den k Punkten $p_1, \ldots, p_k$ ergibt. Gegen letztere Ebene wird auch der erste Test durchgeführt. Es wird geschaut, ob sich alle Punkte des Schnittpolygons auf der Seite der Ebene E befinden, auf der sich auch der Augenpunkt befindet. Ist dies der Fall, kann das Schnittpolygon nicht vom Verdeckpolygon verdeckt werden. Es kann komplett zurückgegeben werden und gegen das nächste Polygon geclippt

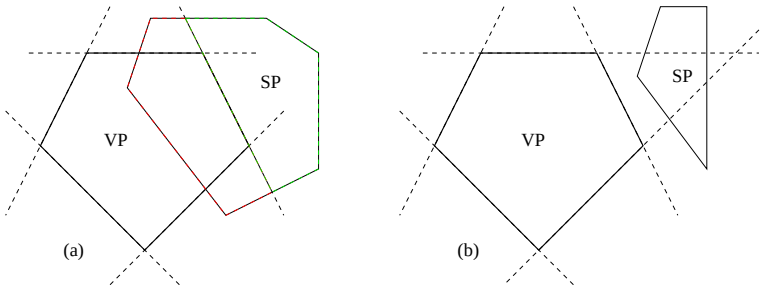


Abbildung 63: Einige Fälle des Brute-Force-Algorithmus (projizierte Szene)

werden. Die wäre in Abbildung 62 für das Polygon A der Fall. Ansonsten wird das Schnittpolygon sequentiell gegen die Ebenen $E_1, \ldots, E_k$ geclippt. Dabei wird das Schnittpolygon im i-ten Schritt unter Umständen in zwei Teile geteilt. Ein Teil, der bezüglich der Ebene E_i sichtbar ist und einer der nicht sichtbar ist. Der nicht sichtbare Teil liegt auf der Seite der Ebene, in der auch alle anderen Punkte des Verdeckpolygons liegen. In Abbildung 63(a) ist der sichtbare Teil grün und der nicht sichtbare Teil rot eingezeichnet. Der sichtbare Teil wird sich gemerkt, da der sichtbare Teile des ganzen Polygons die Vereinigung der sichtbaren Teile jeder Ebene ist. In jedem Schritt wird dann mit dem Rest, also dem nicht sichtbaren Teil weitergemacht. Es kann jedoch auch vorkommen, dass das Schnittpolygon in einem der Schritte bezüglich einer der Ebenen komplett im sichtbaren Bereich liegt und daher auch komplett sichtbar ist. In diesem Fall wird es wieder komplett zurückgegeben. Abbildung 63(b) zeigt diesen Fall. Obwohl es von einigen Ebenen geschnitten wird, ist es komplett sichtbar. Tritt dieser Fall nicht ein, wird zuletzt geschaut, ob der Rest des Polygons bezüglich der Ebene E durchdringend ist.

Frage 8.6: Wie groß ist die Laufzeit des Brute-Force-Algorithmus?
Sind n Polygone gegeben, so beträgt die Laufzeit $O(n^2)$. Denn jedes Polygon muss gegen $n-1$ andere Polygone geclippt werden. Wie oft jedoch der Algorithmus zum Clippen eines Polygons gegen ein anderes aufgerufen wird, hängt von der Szene ab. Dies liegt daran, dass Polygone in mehrere Teilpolygone zerfallen können, welche alle einzeln weiter geclippt werden müssen. Bei einer Szene, in der viele Polygone zerfallen, muss also auch öfter geclippt werden.

Frage 8.7: Wie kann die Anzahl der Tests beim Brute-Force-Verfahren verringert werden, wenn die Szene aus vielen kleinen Polygonen besteht?
In diesem Fall kann der Algorithmus durch einen Vorverarbeitungsschritt beschleunigt werden. Dabei wird ein gleichmäßiges Gitter über die Projektion der Polygone gelegt. Soll ein bestimmtes Polygon geclippt werden, dann wird es nur gegen die Polygone geclippt, die sich im gleichen Bereich des Gitters befinden. Abbildung 64 verdeutlicht diese Idee. Das dunkelgraue Polygon soll geclippt werden. Es erstreckt sich über zwei Gitterbereiche und wird daher gegen alle hellgrauen Polygone geclippt, die sich ebenfalls in diesen Bereichen befinden.

Frage 8.8: Was ist Backface Culling?
In dem Fall, dass die Polygone die Seitenflächen eines Polyeders darstellen, kann

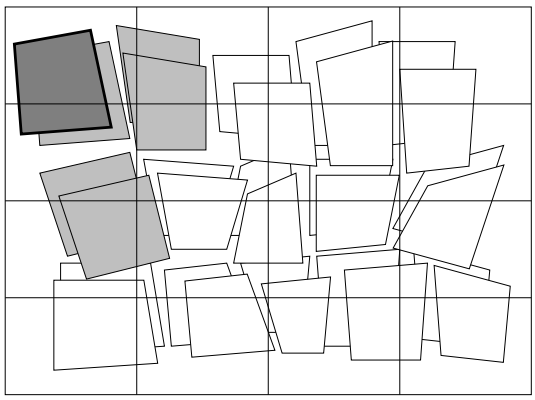


Abbildung 64: Beschleunigung des Brute-Force-Verfahrens

es sinnvoll sein, zuerst die Flächen zu entfernen, die überhaupt nicht sichtbar sein können. Dies sind alle Flächen, deren Normalenvektor bezüglich der Blickrichtung mindestens eine negative Komponente hat. Der Normalenvektor eines Polygons kann mit Hilfe des Kreuzprodukts zweier seiner Kanten berechnet werden. Die beiden Kanten können dabei durch drei nicht kolineare Eckunkte gegeben sein.

8.2 Tiefenpuffer-Verfahren

Frage 8.9: Wie funktioniert das Tiefenpuffer-Verfahren (Z-Buffer)?
Das Tiefenpuffer-Verfahren wurde von Catmull entwickelt und ist einer der einfachsten Algorithmen mit Bildgenauigkeit. Er kann in Hardware und in Software implementiert werden. Zusätzlich zum Bildwiederholspeicher (**Frame Buffer**) wird ein Tiefenpuffer benutzt, der genau so viele Einträge speichert, wie der Bildwiederholspeicher Pixel enthält. Diese Einträge können entweder Ganzzahlen oder Gleitkomma-Zahlen sein.

Der Algorithmus funktioniert wie folgt: Der vorderen und hinteren Kappungsebene werden Werte zugeordnet, wobei die hintere Ebene den Wert Null erhält und die vordere Ebene den größten Wert, der im Tiefenpuffer gespeichert werden kann. Objektpunkte, die während des Algorithmus gezeichnet werden sollen, erhalten später einen Wert zwischen diesen beiden Extremwerten. Ein Punkt, der näher am Betrachter liegt als ein anderer, erhält dabei auch einen größeren Wert. Nachdem der Bildwiederholspeicher mit der Hintergrundfarbe und der Tiefenpuffer mit Null, also dem Wert der hinteren Kappungsebene, initialisiert wurde, werden die Polygone nun der Reihe nach gezeichnet. Dazu kann zum Beispiel der Scan-Line-Algorithmus aus Kapitel 5.1 verwendet werden. Dabei wird für jeden Punkt, der gezeichnet werden soll, geschaut, ob er näher am Betrachter liegt als der Punkt, dessen Farbe und Tiefe momentan in den Puffern steht. Ist dies der Fall, wird der Punkt gezeichnet, also seine Farbe in den Bildwiederholspeicher geschrieben, und sein Tiefenwert im Tiefenpuffer eingetragen. Abbildung 65 veranschaulicht diese Vorgehensweise des Algorithmus. Die Zahlen entsprechen den Einträgen im Tiefenpuffer, die Farben

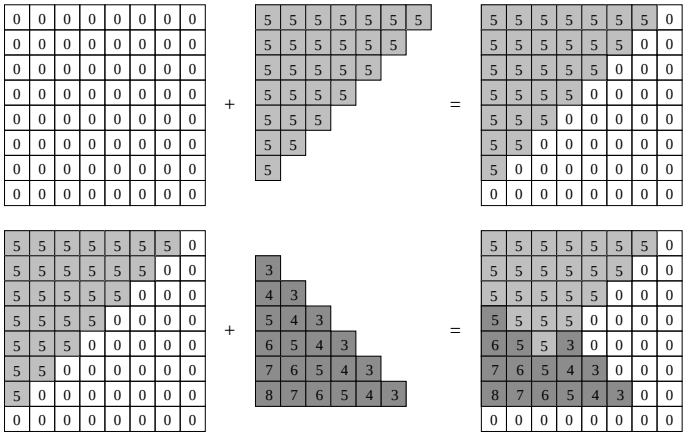


Abbildung 65: Das Tiefenpuffer-Verfahren

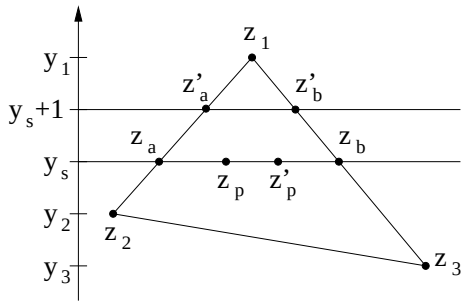


Abbildung 66: Inkrementelle Berechung der Tiefenwerte

den Einträgen im Bildwiederholspeicher. Zuerst wird ein graues Polygon, dessen z-Werte alle 5 betragen (es liegt also parallel zur Projektionsebene), in den frisch initialisierten Bildwiederholspeicher und den Tiefenpuffer geschrieben. Anschliessend wird ein weiteres, schräg im Raum liegendes Polygon, welches sich mit dem zuerst gezeichneten Polygon schneidet, gezeichnet. Wird für ein Pixel ein Tiefenwert errechnet, welcher dem im Tiefenpuffer entspricht, wird das neuere Pixel gezeichnet. Daher setzen sich in der Abbildung auch die Fünfen bzw. die Farbe des später gezeichneten Polygons durch.

Frage 8.10: Wie können die Tiefenwerte eines Polygons effizient berechnet werden?

Wird ein Polygon beim Tiefenpuffer-Verfahren mit Hilfe des Scan-Line-Algorithmus verrastert, dann können seine Tiefenwerte effizient inkrementell berechnet werden. Abbildung 66 verdeutlicht die Idee dabei. Die Tiefen z_1, z_2 und z_3 für die drei das Polygon definierenden Punkte seien gegeben bzw. berechnet worden. Dann ergibt sich z.B. die Tiefe am Punkt z_a, der auf der momentan betrachteten Scanline liegt, wie folgt:

$$z_a = z_1 + (z_2 - z_1) \cdot \frac{y_s - y_1}{y_2 - y_1}$$

Für den Punkt z'_a ergibt sich eine ähnliche Formel:

$$z'_a = z_1 + (z_2 - z_1) \cdot \frac{(y_s + 1) - y_1}{y_2 - y_1}$$

Die Differenz zwischen den beiden Werten ist jedoch eine Konstante, denn es gilt:

$$\begin{aligned} z'_a - z_a &= \left(z_1 + (z_2 - z_1) \cdot \frac{(y_s + 1) - y_1}{y_2 - y_1}\right) - \left(z_1 + (z_2 - z_1) \cdot \frac{y_s - y_1}{y_2 - y_1}\right) \\ &= (z_2 - z_1) \cdot \frac{((y_s + 1) - y_1) - (y_s - y_1)}{y_2 - y_1} \\ &= \frac{z_2 - z_1}{y_2 - y_1} \end{aligned}$$

Ähnlich ergibt sich die Differenz von z_b und z'_b:

$$z'_b - z_b = \frac{z_3 - z_1}{y_3 - y_1}$$

Der Punkt z_p und sein Nachfolger z'_p auf der Scanline bei y_s ergeben sich als Interpolation zwischen z_a und z_b:

$$z_p = z_a + (z_b - z_a)\frac{x_p - x_a}{x_b - x_a} \text{ und } z'_p = z_a + (z_b - z_a)\frac{(x_p + 1) - x_a)}{x_b - x_a}$$

Auch in diesem Fall ist die Differenz zwischen z'_p und z_p wieder eine Konstante:

$$z'_p - z_p = \frac{z_b - z_a}{x_b - x_a}$$

8.3 Scan-Line-Verfahren

Frage 8.11: Wie funktioniert das Scan-Line-Verfahren?

8.4 Bereichsunterteilungsverfahren

Frage 8.12: Wie funktioniert das Bereichsunterteilungsverfahren?

Der **Algorithmus von Warnock** ist ein **Divide-and-Conquer**-Algorithmus und geht so vor, dass die Projektion der Szene solange rekursiv in vier quadratische Teilbilder zerlegt wird, bis die Sichtbarkeitsberechnung sehr einfach ist.

Dabei kann zwischen vier verschiedene Möglichkeiten unterschieden werden, wie ein Polygon zu einem Teilbereich stehen kann. Diese sind in Abbildung 67 dargestellt.

Umgebend: Das Polygon umgibt den Bereich vollständig, d.h. ein Schnitt zwischen Polygon und dem Bereich gibt wieder den ganzen Bereich.

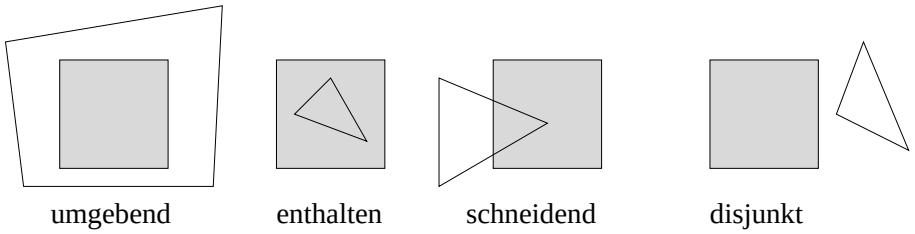


Abbildung 67: Die vier Fälle beim Warnock-Algorithmus

Enthalten: Das Polygon ist vollständig im Bereich enthalten. Der Schnitt zwischen Bereich und Polygon gibt das ganze Polygon.

Schneidend: Das Polygon schneidet den Bereich. Ein Schnitt ergibt einen Teil des Polygons.

Disjunkt: Das Polygon liegt vollkommen außerhalb des Bereichs. Der Schnitt zwischen beiden ist leer.

Es gibt nun vier Fälle, bei denen eine Entscheidung leicht fällt:

1. Alle Polygone sind disjunkt zum Bereich. Der Bereich wird mit der Hintergrundfarbe angezeigt.
2. Es gibt nur ein enthaltendes oder schneidendes Polygon in dem Bereich. Es wird erst der Hintergrund gezeichnet und dann das Polygon bzw. sein sichtbarer Teil gerastert.
3. Es gibt ein einziges umgebendes Polygon, aber kein schneidendes oder enthaltenes Polygon. Der Bereich wird in diesem Fall vollständig in der Farbe des Polygons gezeichnet.
4. Es gibt mehr als ein schneidendes, enthaltenes oder umgebendes Polygon, jedoch ist darunter ein umgebendes Polygon, welches vor allen anderen liegt, d.h. diese also verdeckt. In diesem Fall wird der ganze Bereich in der Farbe des Polygons gefüllt.

Tritt keiner dieser vier Fälle ein, wird der entsprechende Bereich erneut unterteilt. Die maximale Unterteilung liegt dann vor, wenn die Unterteilung der Pixelebene entspricht. Kann dann immer noch keine eindeutige Entscheidung getroffen werden, wird der Mischwert aller in diesem Bereich befindlichen Polygone gezeichnet.

Wie kann der vierte Fall nun erkannt werden? Wird von einem rechthändigen Koordinatensystem ausgegangen, dann liegt ein umgebendes Polygon vor allen anderen Polygonen, wenn sein minimaler Tiefenwert an den Eckpunkten größer ist als alle maximalen Tiefenwerte der anderen Polygone.

Der Algorithmus von Warnock ist eine Mischung zwischen einem objektgenauen und einem bildgenauen Algorithmus. Die Aufteilung und Entscheidung der Bereiche geschieht objektgenau. Die Abbruchbedingung, d.h. wann die Pixelgröße erreicht ist, hängt jedoch von der verwendeten Auflösung ab und ist damit pixelgenau.

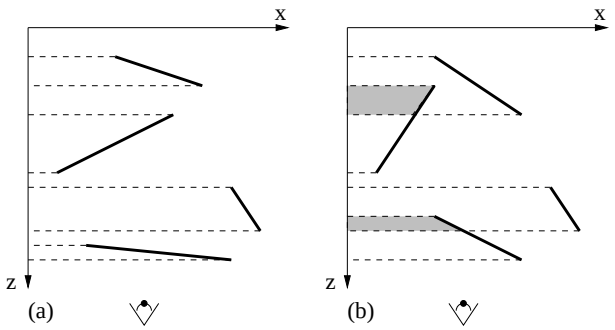


Abbildung 68: Nichtüberlappende und überlappende Polygone beim Painter's Algorithmus

8.5 Prioritätslisten-Verfahren

Frage 8.13: Welche Prioritätslisten-Verfahren kennst du?

- Painter‘s Algorithmus (Newell, Newell und Sancha)
- BSP-Bäume

8.5.1 Painter's Algorithmus

Frage 8.14: Wie funktionieren der Painter's Algorithmus?

Der Painter's Algorithmus ist eine Mischung zwischen einem Objektraum- und einem Bildraumverfahren. Die Grundidee besteht darin, die Elemente der Szene im Objektraum nach abnehmender Entfernung zum Betrachter zu sortieren. Anschließend werden sie in dieser Reihenfolge auf den Bildschirm gezeichnet. Da die am weitesten entfernt liegenden Polygone zuerst gezeichnet werden, können sie unter Umständen von weiter vorne liegenden Polygonen überzeichnet werden. Aufgrund dieser Vorgehensweise hat der Algorithmus auch seinen Namen bekommen, denn Ölgemälde werden auf die gleiche Art und Weise gemalt. Durch dieses Vorgehen wird im Bildraum das Verdeckungsproblem gelöst, wobei dies jedoch nicht in jedem Fall bzw. für jede Szene funktioniert. So kann es zum Beispiel vorkommen, dass sich Polygone wechselseitig überlappen und eine Reihenfolge daher nicht mehr eindeutig ist. Abbildung 68(a) zeigt einen günstigen Fall, bei dem sofort gezeichnet werden kann, da sich die Polygone bezüglich der z-Achse, welche die Entfernung zum Betrachter angibt, nicht überlappen. Die Szene aus Abbildung 68(b) kann dagegen nicht ohne weiteres gezeichnet werden, da sich einige Polygone in den grau gezeichneten Intervallbereichen überlappen. In diesem Fall müssen einige zusätzliche Tests durchgeführt werden.

Der Algorithmus läuft nun in den folgenden Schritten ab, wobei von einem rechthändigen Koordinatensystem ausgegangen wird:

- Sortiere die Polygone nach den kleinsten z-Werten jeder Fläche in eine Prioritätsliste.

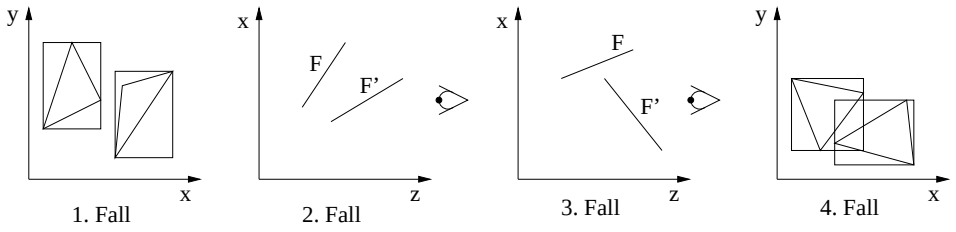


Abbildung 69: Die vier Tests des Painters Algorithmus

- Beginne mit der entferntesten Fläche wird nun für jede Fläche F überprüft, ob sie von anderen Flächen bezüglich der z-Achse überlappt wird. Ist dies der Fall, werden die folgenden vier Tests durchgeführt, um zu schauen, ob die Flächen umgeordnet werden müssen.

  1. Die Bounding Boxes der Projektion der beiden Flächen überlappen sich nicht.
  2. Alle Eckpunkte von F liegen hinter der von F' aufgespannten Ebene.
  3. Alle Eckpunkte von F' liegen vor der von F aufgespannten Ebene.
  4. Die Projektion der beiden Flächen auf die Projektionsebene schneiden sich nicht.

  - Ist eine der Bedingung erfüllt muss keine Umordnung durchgeführt werden und F kann gezeichnet werden.
  - Trifft keine der Bedingungen zu, so wird die Fläche F, sofern sie nicht schon markiert ist, in der Liste mit der Fläche F' vertauscht. Es wird dann mit F' weitergearbeitet, wobei die Tests 1 und 4 bezüglich F nicht mehr durchgeführt werden müssen.
  - Andernfalls liegt eine zyklische Selbstüberlappung vor und F wird entlang der Überlappungsgrenze mit F' geteilt. Anschließend werden die Tiefenvergleiche mit dem entfernteren Teil von F erneut gestartet.

- Eine Fläche F kann angezeigt werden, wenn für alle Flächen F' eine der vier Bedingungen erfüllt ist.

8.5.2 BSP-Bäume

Frage 8.15: Wie funktionieren BSP-Bäume?
BSP-Bäume eignen sich besonders gut für statische Szenen, in denen sich nichts ändert. Der Grund hierfür besteht darin, dass in diesem Fall in einem Vorverarbeitungsschritt eine Datenstruktur erstellt werden kann, mit der in linearer Zeit das Verdeckungsproblem gelöst werden kann. Da diese Vorverarbeitungsphase etwas aufwändig sein kann, darf sich die Szene danach nicht mehr ändern, da diese Phase ansonsten erneut durchgeführt werden müsste. Die Datenstruktur, die in diesem Schritt erstellt wird, ist ein BSP-Baum.

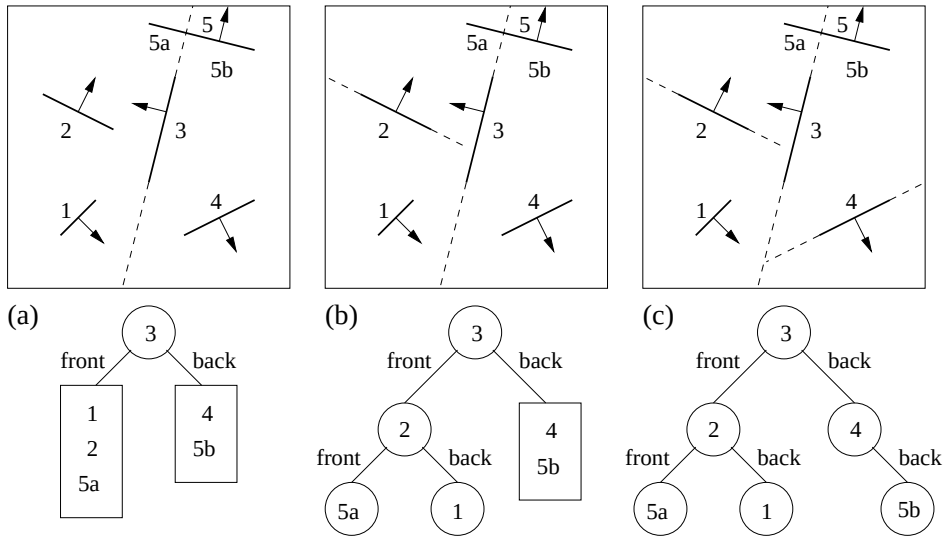


Abbildung 70: Aufbau des BSP-Baumes anhand eines zweidimensionalen Beispiels

Die Grundidee besteht darin, eine Raumzerlegung durchzuführen und diese in einem Suchbaum, dem sogenannten BSP-Baum zu speichern. Allgemein wird ein n-dimensionaler Raum rekursiv entlang $(n-1)$-dimensionaler Objekte in zwei n-dimensionale Halbräume geteilt. Objekte, die in beiden Halbräumen liegen, werden geteilt. Abbildung 70 verdeutlicht diese Idee wahrscheinlich besser. Sie zeigt eine Ebene, also einen zweidimensionalen Raum, die rekursiv entlang eindimensionaler Objekte, also Geraden, geteilt wird. Zuerst wird entlang Objekt 3 geteilt, wobei Objekt 5 geschnitten wird und deshalb in die Teilobjekte $5a$ und $5b$ zerfällt. Die Pfeile in der Abbildung geben dabei die Vorderseite eines Objektes an. Im zweidimensionalen Raum scheint dies nicht so viel Sinn zu machen, aber im dreidimensionalen Raum ist dies schon sinnvoller. Denn dann haben Objekte im allgemeinen auch ein Innen und ein Außen. Wie man sieht, gibt es nun einige Objekte, die in der vorderen Halbebene liegen und einige, die in der hinteren Halbebene liegen. In der vorderen Halbebene liegen z.B. die Objekte 1, 2 und $5a$. Dies wird im Baum jeweils vermerkt und dann rekursiv weitergeteilt. Im Beispiel wird zuerst die vordere Halbebene entlang Objekt 2 weitergeteilt. Danach kann nicht mehr weitergeteilt werden, da sich in jeder Halbebene nur noch ein Objekt befindet. Nachdem Objekt 4 geteilt wurde, ist dies auch in der hinteren Halbebene von Objekt 3 der Fall und der BSP-Baum komplett aufgebaut.

Wie kann dieser Baum nun benutzt werden, um das Verdeckungsproblem zu lösen? Allgemein besteht die Idee darin, wenn die Szene gezeichnet werden soll, erst einmal alles in korrekter Weise darzustellen, was sich vom Betrachter aus hinter einem bestimmten Polygon befindet. Denn diese Elemente befinden sich alle komplett in der hinteren Halbebene des Polygons und können deshalb potentiell von diesem Polygon verdeckt werden. Anschließend wird das Polygon selbst gezeichnet und danach alles, was sich davor befindet. Wie wird nun

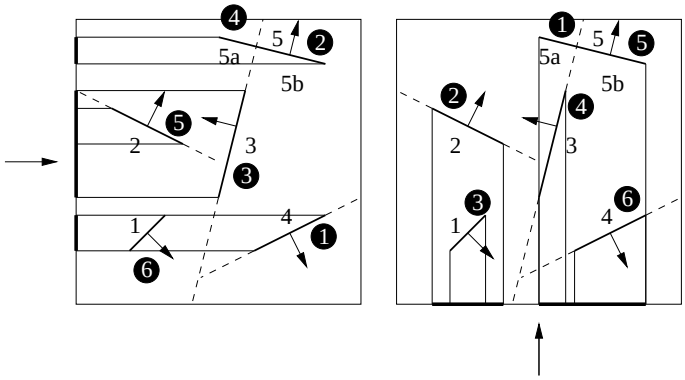


Abbildung 71: Beispiel für zwei Projektionen

sicher gestellt, dass vor und hinter dem Polygon alles richtig gezeichnet wird? Ganz einfach, indem das Problem dort rekursiv, mit derselben Methode gelöst wird. Der Baum wird daher, beginnend bei der Wurzel, in Abhängigkeit von der Betrachterposition nach einem bestimmten Durchlaufschema durchlaufen. Wenn sich ein Betrachter irgendwo in der Szene befindet, beginnt man an der Wurzel und zeichnet erst einmal die Objekte, die sich bezüglich des Wurzelobjekts nicht im selben Halbraum wie der Betrachter befinden. Dann wird das Objekt selbst gezeichnet und dann alles, was sich im selben Halbraum wie der Betrachter befindet. Abbildung 71 zeigt zwei Beispiele, wobei der Pfeil grob die Betrachterposition und Blickrichtung angibt.

9 Realistische Darstellung

Frage 9.1: Wozu wird realistische Darstellung benötigt?

Mit den in den vorherigen Kapiteln beschriebenen Techniken ist es möglich, dreidimensionale Szenen darzustellen. Es wurden jedoch noch viele Aspekte weggelassen, die eine Szene erst realisitisch erscheinen lassen. Dazu gehört zum Beispiel die Beleuchtung einer Szene. Bis jetzt hatte jedes Polygon eine Farbe, in der es gezeichnet wurde. Jedoch reagieren Menschen auf Helligkeitsunterschiede empfindlicher als auf Farbunterschiede. Um eine ansprechende Darstellung zu erreichen, ist ein gutes Beleuchtungsmodell also unverzichtbar. Dabei wird ein Modell benutzt werden, welches zwar nicht der Realität entspricht, jedoch schon gute Ergebnisse erzielt.

Mit diesem Verfahren wird es jedoch nicht so leicht sein, Schatten, Spiegelungen und Brechungen darzustellen. Daher wird in diesem Kapitel desweiteren auf das Raytracing-Verfahren eingegangen, welches diese Effekte ermöglicht, dafür jedoch auch relativ zeitaufwändig ist. Zu guter Letzt wird auf das Radiosity-Verfahren eingegangen, welches es ermöglicht, eine noch realistischere Beleuchtung zu erzielen, indem berücksichtigt wird, dass helle, große Flächen in einer Szene Licht reflektieren und auf diese Weise gewissermaßen ebenfalls eine Lichtquelle darstellen.

9.1 Beleuchtungsmodell

Frage 9.2: Welche Annahmen werden im Beleuchtungsmodell von Phong gemacht?

Das Beleuchtungsmodell von Phong geht davon aus, dass es eine Reihe von Objekten gibt, die von verschiedenen, im Raum positionierten, punktförmigen Lichtquellen angestrahlt werden. Dabei wird davon ausgegangen, dass die Lichtquellen sogenannte **Lambert-Strahler** sind, d.h. sie geben das Licht gleichmäßig in alle Richtungen ab. Diese Lichtquellen sind auch die einzigen Objekte, die in diesem Modell Licht emittieren. Es wird also davon ausgegangen, dass die anderen Objekte der Szene kein Licht emittieren. Im folgenden wird das Beleuchtungsmodell nun anhand eines Objekts und einer Lichtquelle erklärt.

Die wahrgenommene Beleuchtung in einem Punkt eines Objekts hängt im allgemeinen von mindestens drei Faktoren ab: der Position des Beobachters, der Position und der Eigenschaften der Lichtquelle (z.B. Intensität) und der Position, Orientierung und der Eigenschaften (z.B. Materialeigenschaften) des Objekts selbst. Diese drei Faktoren können als eine Menge von Vektoren repräsentiert werden, welche wiederum in bestimmten Verhältnissen zueinander stehen. Diese Vektoren sind:

Normalenvektor N: Der Normalenvektor N steht senkrecht auf der Tangentialebene der Oberfläche im betrachteten Punkt und zeigt von der Objektoberfläche weg nach außen.

Lichtvektor L: Der Lichtvektor zeigt vom betrachteten Punkt aus in Richtung der Lichtquelle.

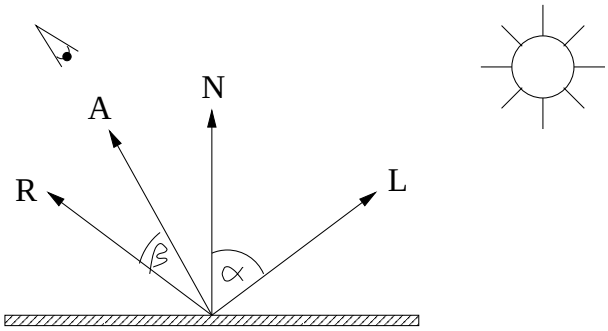


Abbildung 72: Beleuchtungsmodell nach Phong

Augenvektor *A*: Der Augenvektor zeigt vom betrachteten Punkt aus in Richtung des Augenpunktes.

Auf die beiden Winkel α und β wird an entsprechender Stelle eingegangen. Die Intensität eines Punktes auf der Oberfläche ergibt sich nun als die Summe der **ambienten**, **diffusen** und **spiegelnden** Reflexion, die jeweils von unterschiedlichen Faktoren abhängen und unterschiedliche Gegebenheiten der realen Welt simulieren. Die grobe Formel lautet also:

$$I(A) = I_a + I_d + I_g$$

Die ambiente Beleuchtung ist nötig, um eine gewissen Grundbeleuchtung zu simulieren. Wie schon erwähnt wurde, wird im Phong-Modell davon ausgegangen, dass die Objekte der Szene selbst kein Licht emittieren. Dies bedeutet zum Beispiel, dass eine helle weiße Wand, die von einer Lichtquelle angestrahlt wird, dieses Licht nicht wieder in die Szene zurückwirft und so Objekte in ihrer Nähe beleuchtet. Dies wird mit Hilfe der ambienten Beleuchtung kompensiert, da ansonsten nur die Objekte der Szene sichtbar wären, die direkt von einer Lichtquelle angestrahlt werden.

Die diffuse Beleuchtung ist vom Winkel zwischen der Lichtquelle L und Normalenvektor N anhängig. Für sie gilt das **Cosinus-Gesetz von Lambert**. Dieses besagt, dass ein Punkt auf der Oberfläche am hellsten erscheint, wenn das Licht senkrecht auf ihn einfällt, wenn also der Winkel zwischen L und N genau 0° beträgt. Je größer dieser Winkel nun wird, desto mehr nimmt die Lichtintensität in diesem Punkt ab, bis das Objekt bei 90° schließlich nicht mehr direkt beleuchtet wird. Damit bietet sich der Cosinus geradezu an, um dieses Verhalten zu modellieren. Da bei der diffusen Beleuchtung davon ausgegangen wird, dass das Objekt einfallendes Licht gleichmäßig reflektiert, ist sie also betrachterunabhängig.

Die spiegelnde Beleuchtung ist im Gegensatz zur diffusen Beleuchtung betrachterabhängig. Ein spiegelndes Objekt wirft einfallendes Licht nach der Regel „Einfallswinkel gleich Ausfallswinkel“ zurück. Dabei kann es nun vorkommen, dass das zurückgeworfene Licht direkt in das Auge das Betrachters fällt, wodurch der entsprechende Punkt am Objekt als sehr hell wahrgenommen wird. Da die meisten Objekte keine perfekten Spiegel sind, wird auch ein gewisser

Teil der Umgebung des Punktes noch als hell wahrgenommen. Es wird später darauf eingegangen, wie dies im Phong-Modell modelliert wird.

Frage 9.3: Wie berechnet sich die ambiente Beleuchtung?

Die ambiente Beleuchtung ist für die ganze Szene gegeben und ist damit ein konstanter Ausdruck, der bei jeder Intensitätsberechnung in einem Punkt hinzu addiert wird. Es gilt also:

$$I_a = k_a \cdot I_{amb}$$

Dabei ist k_a ein materialabhängiger Dämpfungsfaktor, der **ambiente Reflexionskoeffizient**, der zwischen 0 und 1 liegt. Wie man sieht, ist die ambiente Beleuchtung von den Lichtquelle unabhängig, d.h. es geht kein I_i in die Formel ein.

Frage 9.4: Wie berechnet sich die diffuse Beleuchtung?

Es wird nun nach und nach die Formel für die diffuse Beleuchtung aufgebaut. Auch bei der diffusen Beleuchtung gibt es einen materialabhängigen Dämpfungsfaktor, den **diffusen Reflexionskoeffizienten** k_d, der ebenfalls zwischen 0 und 1 liegt. Die Intensität einer Lichtquelle L_i ergibt sich, wie schon erwähnt, durch den Cosinus zwischen dem Normalenvektor des Objekts und des Richtungsvektors zur Lichtquelle L_i. Beträgt der Winkel 0°, so geht die Lichtquelle L_i mit der vollen Intensität I_i ein, ansonsten nimmt die Intensität langsam ab, bis sie bei 90° schließlich 0 wird. Es ergibt sich also erst einmal die folgende Formel:

$$I_d = k_d \cdot \cos\alpha \cdot I_i$$

Wenn davon ausgegangen wird, dass die Vektoren des Modells alle normalisiert sind, dann kann $\cos\alpha$ auch als das Skalarprodukt der beiden Vektoren ausgedrückt werden. Denn für zwei Vektoren u und v, die den Winkel α einschließen, gilt:

$$\cos\alpha = \frac{(u * v)}{||u|| \cdot ||v||}$$

Dabei ist $(u * v)$ das Skalarprodukt zwischen den Vektoren u und v. Sind die Vektoren normiert, gilt also $\cos\alpha = (u * v)$. In diesem Fall ergibt sich für die Formel der diffusen Reflexion:

$$I_d = k_d \cdot (N * L_i) \cdot I_i$$

Ganz realistisch wird dieses Modell jedoch immer noch nicht wirken, da verschieden weit entfernte Lichtquellen, die im gleichen Winkel auf die Oberfläche strahlen, die gleiche Intensität in diesem Punkt liefern. Aus der Realität ist man jedoch ein anderes Verhalten gewöhnt. Hier nimmt die Intensität mit dem Quadrat der Entfernung ab. Aus diesem Grund wird eine Funktion $f(d_i)$, die eine Funktion des Abstandes der Lichtquelle L_i zum Objektpunkt ist, eingebaut. Es ergibt sich also insgesamt:

$$I_d = k_d \cdot \frac{(N * L_i) \cdot I_i}{f(d_i)}$$

In der Realität entspricht $f(d_i)$ dem Quadrat der Entfernung, jedoch sieht dies im Phong-Modell nicht so gut aus. Dies liegt daran, dass die Unterschiede für

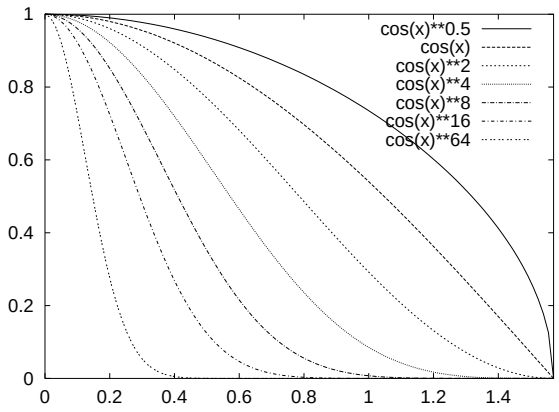


Abbildung 73: Auswirkung verschiedener Werte für γ im Phong-Modell

Objekte, die sich nah an der Lichtquelle befinden, größer sind, also für Objekte, die sich weiter entfernt davon befinden. Daher werden oft andere Funktionen benutzt, die bessere Ergebnisse erzielen.

Frage 9.5: Wie berechnet sich die spiegelnde Beleuchtung?

Die Formel für die spiegelnde Beleuchtung sieht der Formel für die diffuse Beleuchtung recht ähnlich. Auch sie enthält einen **spiegelnden Reflexionskoeffizienten** k_g. Ein weiterer Unterschied ist, dass nun der Winkel zwischen dem Reflexionsvektor R und dem Augenvektor A eine Rolle spielt. Die spiegelnde Reflexion in einem Punkt ist also die einzige Komponente die sich ändert, wenn sich der Betrachter bewegt. Zwischen diesen beiden Vektoren R und A geht wieder der Cosinus ein, denn wenn der reflektierte Lichtstrahl direkt das Auge trifft, ist die Intensität am höchsten. Zusätzlich wird jedoch ein Dämpfungsfaktor $\gamma > 0$ eingeführt, der modelliert, wie perfekt die Spiegelung ist. Die Formel lautet:

$$I_g = k_g \cdot \frac{(A * R_i)^\gamma \cdot I_i}{f(d_i)}$$

Je höher der Wert von γ nun ist, desto perfekter ist die Spiegelung. Abbildung 73 verdeutlicht dies. Bei 0° wird immer der Wert 1 erreicht. Wächst der Winkel nun bis 90° an, nimmt die Funktion um so schneller ab, je größer γ ist.

Frage 9.6: Wie sieht das Beleuchtungsmodell von Phong für mehrere Lichtquellen aus?

Gibt es mehrere Lichtquellen in einer Szene, so ergibt sich die Intensität in einem Punkt als die Summe der Intensitäten, die von den einzelnen Lichtquellen verursacht werden. Die Formel lautet also:

$$I(A) = k_a \cdot I_{amb} + k_d \cdot \sum_{i=0}^{l} \frac{(N * L_i) \cdot I_i}{f(d_i)} + k_g \cdot \sum_{i=0}^{l} \frac{(A * R_i)^\gamma \cdot I_i}{f(d_i)}$$

Das ambiente Licht ist für eine Szene gegeben und unabhängig von den Lichtquellen. Alle anderen Faktoren sind, bis auf die materialabhängigen Reflexionskoeffizienten, abhängig von der Beschaffenheit der Lichtquellen und gehen summiert in die Formel ein.

Frage 9.7: Wie geht das RGB-Modell in das Beleuchtungsmodell ein?
Bis jetzt wurde davon ausgegangen, dass die Lichtquellen mit einer Intensität I_i scheinen. Besteht I_i nur aus einem Wert, so kann dies also für $I_i = 1$ als helles weißes Licht interpretiert werden und für $I_i = 0$ als schwarzes, also nicht vorhandenes Licht. In der Realität gibt es jedoch auch farbiges Licht, von daher liegt es nah, I_i zu einem Vektor zu erweitern, welcher für jedes RGB-Komponente die Intensität angibt. Ähnlich können die Reflexionskoeffizienten so modelliert werden, dass sie die unterschiedlichen Komponenten des Lichts unterschiedlich absorbieren. Es ergibt sich also:

$$I_i = \begin{pmatrix} I_{i_r} \\ I_{i_g} \\ I_{i_b} \end{pmatrix} \text{ mit } i \in \{amb, 0, \ldots, l\} \text{ und } k_x = \begin{pmatrix} k_{x_r} \\ k_{x_g} \\ k_{x_b} \end{pmatrix} \text{ mit } x \in \{a, d, g\}$$

9.2 Schattierungsmethoden

Frage 9.8: An welcher Stelle der Grafik-Pipeline kann die Schattierung der Objekte vorgenommen werden?
Eine Möglichkeit besteht darin, die Beleuchtungssimulation mit der Sichtbarkeitsberechnung zu kombinieren. Wird ein Polygon z.B. mit Hilfe des Tiefenpuffer- oder Scanline-Verfahren verrastert, dann wird der Farbwert des Pixels gleich entsprechend des Beleuchtungsmodells berechnet. Bei der Verwendung des Tiefenpuffer-Verfahrens kann es daher sinnvoll sein, die Polygone zuerst grob nach fallender Entfernung zum Betrachter zu sortieren und diese von vorne nach hinten abzuarbeiten. Denn in diesem Fall werden relativ wenige unnütze Berechnungen für die Beleuchtung gemacht. Andernfalls könnte es nämlich passieren, dass viele Beleuchtungsberechnungen für Pixel gemacht werden, die kurze Zeit später wieder überschrieben werden.

Frage 9.9: Wie funktioniert das Gouraud-Shading? Welche Probleme gibt es dabei?
Beim Gouraud-Shading werden die korrekten Farbwerte nur an den Eckpunkten des Polygons errechnet. Die restlichen Pixel des Polygons werden interpoliert. Dabei wird im Prinzip genau so vorgegangen, wie bei der inkrementellen Berechnung der Tiefenwerte beim Z-Buffer-Verfahren. Es wird also erst entlang der Kanten interpoliert und diese Werte werden entlang der Scan-Line nach innen interpoliert. Dies ist in Abbildung 74 dargestellt. Es ergeben sich also I_a und I_b aus:

$$I_a = I_1 + (I_2 - I_1) \cdot \frac{y_s - y_1}{y_2 - y_1} \text{ und } I_b = I_1 + (I_3 - I_1) \cdot \frac{y_s - y_1}{y_3 - y_1}$$

Daraus ergibt sich der Punkt I_p:

$$I_p = I_a + (I_b - I_a) \cdot \frac{x_p - x_a}{x_b - x_a}$$

Auch hier wird es wahrscheinlich wieder möglich sein, die Werte inkrementell zu berechnen, was das Verfahren relativ schnell macht.

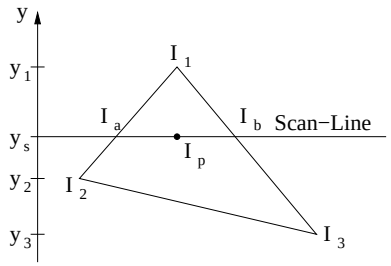


Abbildung 74: Interpolation der Farbwerte beim Gouraud-Shading

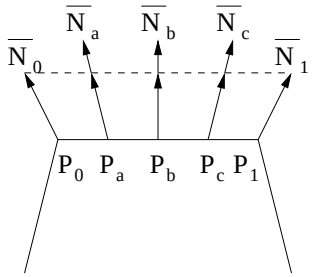


Abbildung 75: Interpolation der Normalenvektoren beim Phong-Shading

Ein Problem des Gouraud-Shading ist, dass der Mach-Band-Effekt noch relativ stark auftreten kann. Das heißt, dass Intensitätsübergänge, die nicht stetig differenzierbar sind, heller bzw. dunkler erscheinen können, als es den Intensitätswerten entspricht. Dadurch werden diese verstärkt wahrgenommen.

Frage 9.10: Wie funktioniert das Phong-Shading?
Das Phong-Shading ist aufwändiger als das Gouraud-Shading, liefert dafür jedoch auch im allgemeinen die besseren Ergebnisse. Es ist deshalb aufwändiger, da die Farbwerte an den Pixeln nun nicht mehr interpoliert werden, sondern für jedes Pixel eine Berechnung nach dem Phong-Modell durchgeführt wird. Da für jedes Pixel ein Normalenvektor benötigt wird, diese jedoch meist nur an den Eckpunkten des Polygons bzw. des Objekts bekannt sind, werden nun die Normalenvektoren interpoliert. Auch hier wird nach demselben System wie beim Gouraud-Shading interpoliert. Die Normalenvektoren der Eckpunkte, werden entlang der Kanten interpoliert und diese entlang einer Scan-Line. Die Interpolation zwischen einem Vektor $\overline{N}_0$ und einem Vektor $\overline{N}_1$ geschieht dabei nach der Formel:

$$(1 - \lambda) \cdot \overline{N}_0 + \lambda \cdot \overline{N}_1 \text{ mit } \lambda \in [0, 1]$$

Bei jeder Interpolation muss der errechnete Normalenvektor jedoch erst noch normalisiert werden, was in Abbildung 75 durch die gestrichelte Linie dargestellt ist.

Frage 9.11: Was kann gemacht werden, wenn die Normalenvektoren an den Eckpunkten nicht bekannt sind?
Ist eine polyedrische Fläche gegeben, deren Normalenvektoren an den Eckpunkten nicht bekannt sind, können diese durch eine Schätzung bestimmt werden.

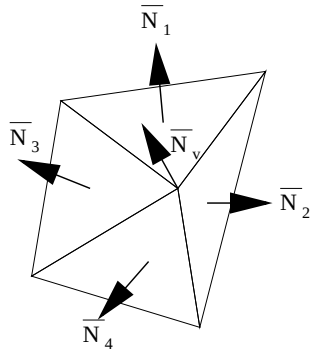


Abbildung 76: Schätzung des Normalenvektors $\overline{N}_v$ aus den Vektoren $\overline{N}_1$, $\overline{N}_2$, $\overline{N}_3$ und $\overline{N}_4$

Dabei wird, wie in Abbildung 76 dargestellt, das gewichtete Mittel der Normalenvektoren der zu einem Eckpunkt inzidenten Polygone berechnet. Dabei können verschiedene Gewichtungen benutzt werden:

- Gleichgewichtung
- $\frac{\text{Polygonfläche}}{\text{Summe der Polygonflächen}}$
- $\frac{\text{Polygonwinkel am Eckpunkt}}{\text{Summe aller Polygonwinkel am Eckpunkt}}$

Frage 9.12: In welchen Fällen ist Phong-Shading genauer als Gouraud-Shading?
Es gibt zwei Fälle, bei denen die Ungenauigkeit des Gouraud-Shadings besonders auffällt:

1. Ein Glanzlicht fällt auf einen der Eckpunkte. Beim Gouraud-Shading wird das Glanzlicht über das ganze Polygon interpoliert, während es beim Phong-Shading nur in die betroffene Ecke gezeichnet wird.

2. Es liegt ein Glanzlicht im Inneren des Polygons. Beim Phong-Shading wird dies, da die interpolierten Normalenvektoren bekannt sind, berechnet und angezeigt. Beim Gouraud-Shading ist dies nicht der Fall, da das Innere des Polygons aufgrund der Interpolation nicht heller werden kann als die Eckpunkte.

9.3 Raytracing

Frage 9.13: Wie funktioniert Raytracing?
Raytracing ist ein Verfahren, welches Effekte wie Schlagschatten, Spiegelung und Brechung ermöglicht. Die Szene sei, wie in Abbildung 77 gezeigt, durch die folgenden Objekte gegeben:

- Augenpunkt a

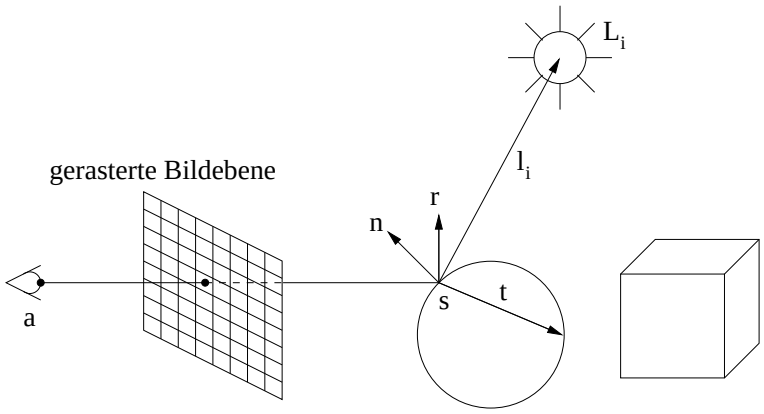


Abbildung 77: Prinzip des Raytracing

- Projektionsebene
- Geometrische Objekte
- l Lichtquellen $L_i, i = 1, \ldots, l$

Zuerst wird die Projektionsebene in $m \times n$ gleichmäßige Rechtecke aufgeteilt, wobei jedes Rechteck einem Pixel des späteren Ergebnisbildes entspricht. Dieses hat dementsprechend also eine Auflösung von $n \cdot m$. Nun wird ein Strahl vom Augenpunkt a aus durch den Mittelpunkt eines jeden Rechtecks geschossen. Für jeden dieser Strahlen wird getestet, welche Objekte der Szene er schneidet. Wird mindestens ein Objekt geschnitten, so wird das Objekt genommen, welches dem Augenpunkt am nächsten ist. Vom Schnittpunkt s dieses Objektes aus werden nun weitere Strahlen in die Szene geworfen, um die Farbe des Objektes an diesem Punkt zu bestimmen.

Zuerst wird geschaut, ob das Objekt im Schatten eines anderen Objekts liegt. Dazu wird vom Schnittpunkt s aus ein Strahl l_i zu jeder Lichtquelle L_i in der Szene geworfen. Wird dabei ein Objekt geschnitten, so liegt dieses zwischen der entsprechenden Lichtquelle und dem Objekt, für welches die Farbe bestimmt werden soll. In diesem Fall ist die Lichtquelle also nicht an der Farbgebung des Objektes beteiligt und wird in der Beleuchtugsberechnung ignoriert. Ansonsten wird, entsprechend des Beleuchtungsmodells, eine Beleuchtungsberechnung an dem Punkt durchgeführt.

Ist das geschnittene Objekt spiegelnd, wird nach der Regel „Einfallswinkel gleich Ausfallswinkel“ ein Reflexionsstrahl r berechnet, welcher erneut in die Szene geschossen wird. Wieder wird geschaut, ob ein Objekt geschnitten wird und welches davon dem Punkt s am nächsten liegt. Wird dabei ein Objekt in einem Punkt p geschnitten, wird für diesen auf die gleiche Art und Weise wie für s der Farbwert ermittelt. Dies bedeutet, dass von p aus eventuell wieder Strahlen in die Szene geschossen werden. Ist der Wert für p ermittelt, wird dieser in einem gewissen Verhältnis zu s hinzugerechnet, denn der Punkt s wirft den Punkt p ja zu einem gewissen Grad in das Auge des Betrachters, da er spiegelnd ist.

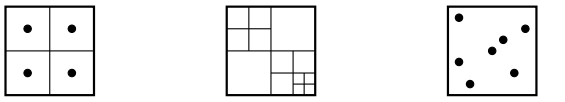


Abbildung 78: Drei Strategien des Anti-Aliasing durch Super-Sampling

Weiterhin kann das getroffene Objekt transparent sein, so dass an s nach dem Brechungsgesetz von Snellius ein Strahl t berechnet wird, welcher erneut in die Szene geschossen wird. Auch das Ergebnis dieser Berechnung geht zu einem Teil in den Farbwert des Punktes s ein.

Die rekursiven Aufrufe, die bei der Berechung der Spiegelung und Brechung entstehen, können in Form eines Strahlenbaums dargestellt werden. Die Anzahl der Knoten dieses Baumes wächst dabei exponentiell an. Damit nicht allzu viele Strahlen berechnet werden bzw. die Berechnung unter Umständen sogar überhaupt nicht mehr stoppt, wird die Anzahl der rekursiven Aufrufe der Strahlberechnungen begrenzt. Oder anders gesagt, es wird dafür gesorgt, dass der Strahlenbaum nicht über eine gewisse Tiefe hinauswächst.

Frage 9.14: Welche Methoden des Anti-Aliasing gibt es beim Raytracing?
Eine allgemeine Methode des Anti-Aliasing ist die des Super-Sampling. Hierbei werden pro Pixel mehrere Strahlen losgeschickt, um den Wert des Pixels zu ermitteln. Wie leicht ersichtlich ist, steigt der Rechenaufwand durch dieses Vorgehen unter Umständen stark an. Es gibt drei Strategien, wie ein Pixel zerlegt werden kann. Diese sind in Abbildung 78 noch einmal dargestellt.

nicht adaptiv: Das Pixel wird in eine feste Anzahl von Unterpixeln unterteilt, deren Intensitäten berechnet werden. Die verschiedenen Intensitäten werden anschließend gewichtet aufsummiert.

adaptiv: Das Pixel wird nach Bedarf unterteilt. Dies kann z.B. so geschehen, dass zuerst die vier Eckpunkte eines Pixels ausgewertet werden. Sind die Unterschiede zwischen diesen gering, werden sie gemittelt und ihr Wert dem Pixel zugewiesen. Andernfalls wird das Pixel in vier gleich große Unterpixel aufgeteilt mit denen auf die gleiche Art und Weise verfahren wird.

stochastisch: Die Strahlen werden zufällig durch das Pixel geschossen und gleichgewichtet aufsummiert.

Frage 9.15: Wieso ist eine effiziente Strahlverfolgung nötig? Welche beiden Möglichkeiten gibt es zur Beschleunigung?
Der Zeitbedarf für eine primitive ad-hoc-Lösung beim Raytracing ist sehr hoch. Denn bei einer primitiven Lösung wird jeder Strahl mit jedem Objekt auf einen Schnitt getestet. Wird zum Beispiel davon ausgegangen, dass eine Szene mit 10000 Dreiecken in einer Auflösung von ca. einer Millionen Pixel (z.B. 1024×1024) berechnet werden soll, wobei eine Schnittberechnung ungefähr 50 Operationen benötigt, dann ergibt dies insgesamt $5 \cdot 10^{11}$ Operationen. Wird

nun weiter davon ausgegangen, dass eine dieser Operationen in 10^{-6} Sekunden durchgeführt werden kann, dann dauert die Berechnung des Bildes 10^5 Sekunden, was einer Zeit von fast 6 Tagen entspricht. Dies ist jedoch inakzeptabel. Es gibt nun zwei Ansatzpunkte für eine Beschleuningung:

- Beschleunigung des Schnitttests selbst.
- Reduktion der Anzahl der Schnittteste. Es muss also dafür gesorgt werden, dass nicht mehr jedes Objekt der Szene auf einen Schnitt mit einem Strahl getestet wird.

Frage 9.16: Welche Möglichkeit gibt es, den Schnitttest für ein Objekt zu beschleunigen?

Für komplexe Objekte ist es sinnvoll, erst in einem Vorverarbeitungsschritt die achsenparallelen Hüllkörper dieser Objekte zu berechnen. Ein Schnitttest mit einem der Objekte wird dann zuerst mit dem Hüllkörper des Objekts durchgeführt, da ein solcher Test im allgemeinen schneller ist als ein Schnitt mit dem Objekt selbst. Wird kein Schnitt festgestellt, wird viel Rechenzeit gespart, wenn die Berechnung eines Schnitts mit dem Objekt selbst sehr zeitaufwändig ist. Ansonsten wird der Schnitttest noch einmal mit dem Objekt selbst durchgeführt. Abbildung 79 zeigt ein zweidimensionales Beispiel. Strahl a schneidet den Hüllquader des linken Dreiecks. Ein genauerer Test mit dem Objekt selbst ergibt jedoch, dass das Dreieck selbst nicht geschnitten wird. Strahl b schneidet beide Hüllquader sowie die Dreiecke selbst. Bei Strahl c wird relativ schnell festgestellt, dass er die Dreiecke nicht schneiden kann.

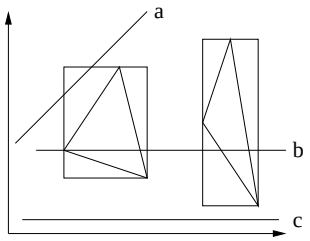


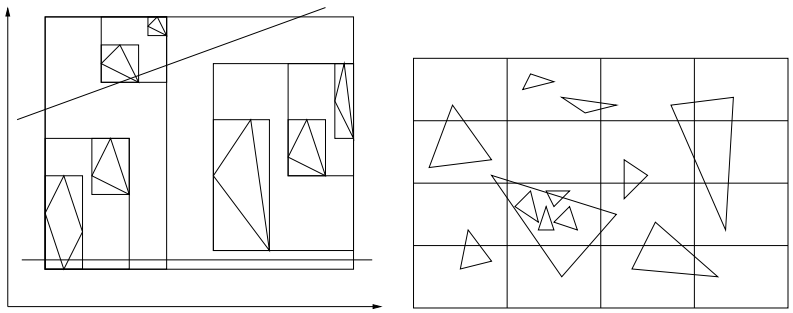
Abbildung 79: Verwendung von Hüllquadern beim Raytracing

Frage 9.17: Wie kann die Anzahl der Schnitttests für eine Szene reduziert werden?

Hier gibt es verschiedene Möglichkeiten:

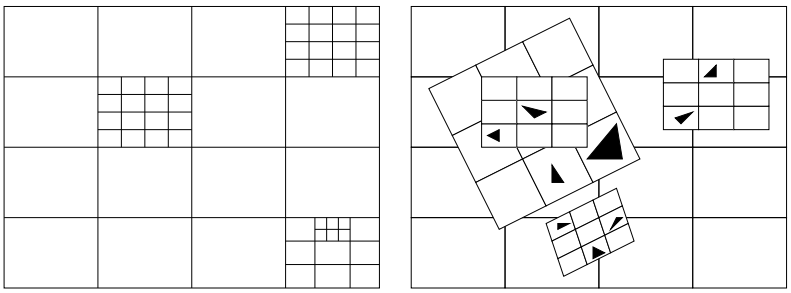
- Geschachtelte Hüllquader
- Reguläres Gitter
- Hierarchisches Gitter
- Geschachtelte Gitter

Frage 9.18: Wie funktionieren geschachtelte Hüllquader?



(a) Geschachtelter Hüllquader

(b) Reguläres Gitter



(c) Hierarchisches Gitter

(d) Geschachtelte Gitter

Abbildung 80: Verschiedene Strategien zur Reduktion von Schnittberechnungen

Bei diesem Verfahren wird zu Beginn ein Hüllquader für die gesamte Szene berechnet. Dieser wird in zwei Hüllquader von Teilszenen von benachbarten Objekten aufgeteilt, z.B. indem die Szene entlang eines Median-Objekts in eine linke und eine rechte Teilszene geteilt wird. Dies wird solange gemacht, bis sich jedes Objekt in einem eigenen Hüllquader befindet. Auf diese Art und Weise entsteht ein binärer Baum von Hüllquadern, an dessen Wurzel der Hüllquader für die ganze Szene steht und an dessen Blättern sich die Hüllquader der einzelnen Objekte befinden. Soll nun ein Schnitt mit einem Strahl ermittelt werden, wird der Hüllquader an der Wurzel des Baumes auf einen Schnitt mit dem Strahl getestet. Findet ein Schnitt statt, werden auch seine beiden Kinder getestet, ansonsten nicht. Wird also die Wurzel eines Teilbaumes nicht geschnitten, so brauchen die Objekte des gesamten Teilbaumes nicht mehr betrachtet werden. Abbildung 80(a) zeigt ein zweidimensionales Beispiel für dieses Vorgehen.

Das Median-Objekt kann zum Beispiel bestimmt werden, indem die Szene auf die x-y-Ebene projiziert wird und entlang der x-Achse das Median-Objekt gesucht wird. Das Median-Objekt ist das Objekt, von dem sich links genau so viele Objekte befinden wie rechts.

Frage 9.19: Wie funktionieren reguläre Gitter?

Bei diesem Verfahren wird der Hüllquader der gegebenen Szene durch eine äquidistante Unterteilung in gleichmäßig große Quaderzellen zerlegt. Für jede Zelle werden die Szenenelemente gespeichert, die diese schneiden. Dazu kann zum Beispiel ein dreidimensionales Array verwendet werden, welches Listen von Szenenelementen speichert. Wird nun ein Strahl in die Szene geschossen, werden die von ihm durchlaufenen Zellen ermittelt und die in diesen enthaltenen Objekte gegen den Strahl getestet. Die Zellen, die von dem Strahl getroffen werden, können effizient durch inkrementelle Addition ermittelt werden. Dieses Vorgehen bringt jedoch nur eine Beschleunigung, wenn sich die Szenenelemente nicht in einigen Gitterzellen häufen. Abbildung 80(b) zeigt diese Methode.

Frage 9.20: Wie funktionieren hierarchische Gitter?

Hierarchische Gitter versuchen, der Anhäufung von Szenenelementen in einer Gitterzelle dadurch entgegen zu treten, dass solche Zellen wieder durch ein reguläres Gitter unterteilt werden. Abbildung 80(c) zeigt ein solches Gitter.

Frage 9.21: Wie funktionieren geschachtelte Gitter?

Geschachtelte Gitter lassen sich auf natürliche Art und Weise bei hierarchisch aus Teilszenen zusammengesetzten Szenen anwenden. Dabei wird eine Teilszene durch ein reguläres Gitter unterteilt und in ihrer „Oberszene“, d.h. der Szene, in der sie eingebettet ist, durch ihren Hüllquader repräsentiert. Wird bei einem Schnitttest nun ein Gitter durchlaufen, so wird ein Schnitttest mit dem Hüllquader der Szene gemacht. Wird dieser geschnitten, wird mit dem regulären Gitter der Teilszene weitergetestet und danach wieder im ursprünglichen Gitter fortgefahren. Abbildung 80(d) verdeutlicht dieses Vorgehen.

9.4 Radiosity-Verfahren

Frage 9.22: Welche Vorteile besitzt das Radiosity-Verfahren?

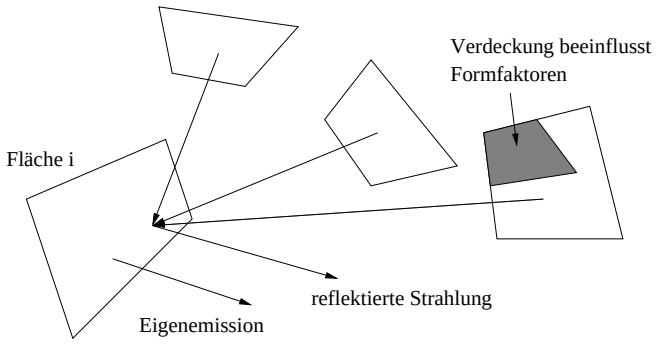


Abbildung 81: Zusammensetzung der abgegebenen Energie einer Fläche i

Das Raytracing-Verfahren hat zwei Nachteile, die beim Radiosity-Verfahren nicht gegeben sind. Zum einen gibt es beim Raytracing keine diffuse Interreflexion zwischen Szenenelementen. In der Realität beleuchtet eine große weiße Wand, die von einer oder mehreren Lichtquellen angestrahlt wird, Gegenstände in ihrer Nähe, indem sie einen großen Teil des Lichts in die Szene zurück wirft. Dadurch werden auch Gegenstände beleuchtet, die nicht direkt von einer Lichtquelle getroffen werden. Die meisten anderen Verfahren gleichen dies durch eine ambiente Beleuchtung aus, was jedoch nicht sehr realisitsch wirkt. Zum anderen gibt es beim Raytracing nur Punktlichtquellen, d.h. es sind keine ausgedehnten Lichtquellen möglich.

Das Radiosity-Verfahren löst diese beiden Probleme, indem es den Lichttransport zwischen den verschiedenen Flächenelementen einer Szene simuliert. Die Grundlage hierfür ist die **Strahlungsgleichung**.

Frage 9.23: Wie lautet die Strahlungsgleichung?

Es wird davon ausgegangen, dass die Szene aus n Flächenelementen besteht. Die von einer Fläche F_i ausgestrahlte Strahlung B_i hängt im verwendeten Modell von zwei Faktoren ab: Der Eigenemission an Strahlung und dem Anteil an reflektierter Strahlung. Die Eigenemission ist der Anteil an Strahlung, den das Flächenelement von sich aus abgibt. Dadurch ist es möglich, ausgedehnte Lichtquellen zu modellieren. Die reflektierte Strahlung ist der Anteil an Strahlung, der von den anderen Flächenelemente eingeht und zu einem gewissen Bruchteil, je nachdem wie gut die Fläche reflektiert, wieder zurück in die Szene geworfen wird. Bei der weiter oben erwähnten weißen Wand wäre der Anteil der zurückgeworfenen Strahlung wahrscheinlich relativ hoch. Wieviel der Strahlungsenergie von einer Fläche bei einer anderen ankommt, hängt von der Orientierung der Flächen zueinander und ihrer Größe ab. Diese Faktoren werden in Form der sogenannten Formfaktoren F_{ij} in die Strahlungsgleichung eingehen. Die Strahlungsgleichung lautet:

$$B_i = E_i + \rho_i \sum_{j=1}^{n} B_j \cdot F_{ij}$$

Dabei ist B_i die Strahlung, die von der Fläche i abgegeben wird. Sie wird in Energie pro Zeiteinheit und Flächeneinheit gemessen. Energie pro Zeiteinheit entspricht der physikalischen Größe der Leistung, welche in Watt W gemessen

wird. Also kann B_i in der Einheit $\frac{W}{m^2}$ gemessen werden. Oder anders gesagt, gibt B_i an, wieviel Energie eine Einheitsfläche innerhalb einer Sekunde abgibt. E_i wird in derselben Einheit gemessen.

Doch wie kommt man auf die obige Formel? B_i ist die Energie, die von der Einheitsfläche des i-ten Elements abgestrahlt wird. Diese setzt sich zusammen aus der abgestrahlten Energie E_i pro Einheitsfläche und der reflektierten einfallenden Energie der anderen Elemente pro Einheitsfläche. Die insgesamt vom einem Flächenelement j abgestrahlte Energie beträgt:

$$B_j A_j$$

Also die Energie der Einheitsfläche multipliziert mit der Größe der Fläche. Der Formfaktor F_{ji} bestimmt, welcher Anteil der Energie pro Einheitsfläche überhaupt bei der Fläche A_i ankommt. Dies ergibt:

$$B_j A_j F_{ji}$$

Die Einheitsfläche erhält davon wiederum den Anteil

$$\frac{B_j A_j F_{ji}}{A_i}$$

wobei davon der folgende Anteil reflektiert wird:

$$\rho_i \cdot \frac{B_j A_j F_{ji}}{A_i}$$

Es ergibt sich also insgesamt:

$$B_i = E_i + \rho_i \cdot \sum_{j=1}^{n} \frac{B_j A_j F_{ji}}{A_i}$$

Eine Multiplikation mit A_i liefert:

$$B_i \cdot A_i = E_i \cdot A_i + \rho_i \cdot \sum_{j=1}^{n} B_j A_j F_{ji}$$

Wie sich bei der Definition der Formfaktoren nachher herausstellen wird, gilt $A_j F_{ji} = A_i F_{ij}$, weshalb wir wie folgt ersetzen können:

$$B_i \cdot A_i = E_i \cdot A_i + \rho_i \cdot \sum_{j=1}^{n} B_j A_i F_{ij}$$

Nun kann wieder durch A_i dividiert werden, um es loszuwerden:

$$B_i = E_i + \rho_i \cdot \sum_{j=1}^{n} B_j F_{ij}$$

Diese Formel kann wie folgt umgestellt werden, wodurch sich ein Gleichungssystem mit n Gleichungen und n unbekannten ergibt:

$$B_i - \rho_i \sum_{j=1}^{n} B_j F_{ij} = E_i$$

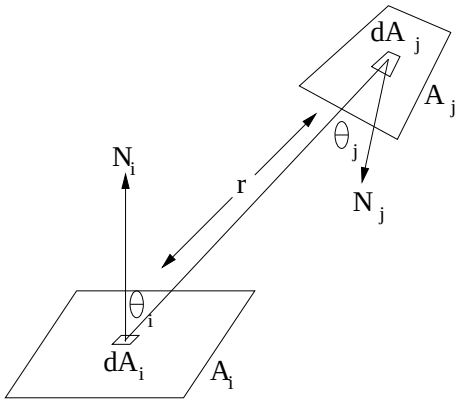


Abbildung 82: Herleitung der Formfaktoren

Denn wenn diese Formel für alle i hingeschrieben wird, ergibt sich:

$$\begin{aligned} B_1 - \rho_1 B_1 F_{11} - \rho_1 B_2 F_{12} - \rho_1 B_3 F_{13} - \ldots - \rho_1 B_n F_{1n} &= E_1 \\ B_2 - \rho_2 B_1 F_{21} - \rho_2 B_2 F_{22} - \rho_2 B_3 F_{23} - \ldots - \rho_2 B_n F_{2n} &= E_2 \\ \vdots & \vdots \ \vdots \\ B_n - \rho_n B_1 F_{n1} - \rho_n B_2 F_{n2} - \rho_n B_3 F_{n3} - \ldots - \rho_n B_n F_{nn} &= E_n \end{aligned}$$

In jeder Zeile kann ein $B_n - \rho_n B_n F_{nn}$ zu $(1 - \rho_n F_{nn}) \cdot B_n$ zusammengefasst werden, wodurch sich die folgende Matrix-Vektor-Schreibweise ergibt:

$$\begin{pmatrix} 1 - \rho_1 F_{11} & -\rho_1 F_{12} & \ldots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1 - \rho_2 F_{22} & \ldots & -\rho_2 F_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \ldots & 1 - \rho_n F_{nn} \end{pmatrix} \cdot \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{pmatrix}$$

Auf die Berechnung der Formfaktoren F_{ij} wird später eingegangen.

Frage 9.24: Wie sind die Formfaktoren definiert?
Der Formfaktor von A_i nach A_j ergibt sich als:

$$F_{ij} = \frac{1}{A_i} \cdot \int_{A_i} \int_{A_j} \frac{\cos\theta_i \cos\theta_j}{\pi r_{ij}^2} \cdot H_{ij} \quad dA_j dA_i$$

Abbildung 82 zeigt, wie es zu dieser Formel kommt. Im Prinzip geht man zuerst von zwei differentiellen Flächen dA_i und dA_j aus. Also von zwei Punkten auf den beiden Flächen. Der Anteil der Energie, die zwischen den beiden differentiellen Flächen transportiert werden kann, wird durch die folgende Formel bestimmt:

$$F_{di,dj} = \frac{\cos\theta_i \cos\theta_j}{\pi r^2} \cdot H_{ij}$$

H_{ij} bestimmt, ob überhaupt ein Energietransport zwischen den beiden Punkten stattfindet. Es könnte ja sein, dass sich ein weiteres Objekt zwischen den beiden Punkten befindet. Ist dies der Fall, sind die beiden Punkte nicht voneinander

sichtbar und es gilt daher $H_{ij} = 0$. Ansonsten gilt $H_{ij} = 1$. Wenn die beiden Punkte nun voneinander sichtbar sind, hängt der Anteil der Energie von den Winkeln der Flächen zueinander sowie von der Entfernung der beiden Flächen ab. Was das π in dieser Formel soll, weiß nur der liebe Gott oder die Leute von der Cornell University.

Als nächster Schritt wird der Formfaktor einer differentiellen Fläche zu einer endlichen Fläche berechnet. Im Prinzip wird also der Punkt auf der Fläche A_i festgehalten und der Wert für alle Punkte der Fläche A_j berechnet. Die gesamte Energie, die vom Punkt auf der Fläche A_i zur gesamten Fläche A_j transportiert wird, ergibt sich dann als Integral über diese Fläche:

$$F_{di,j} = \int_{A_j} \frac{\cos\theta_i \cos\theta_j}{\pi r^2} \cdot H_{ij} \quad dA_j$$

Nun besteht die Fläche A_i jedoch nicht nur aus diesem einen Punkt. Gesucht ist der Anteil der Energie der pro Einheitsfläche von A_i nach A_j transportiert wird. Aus diesem Grund muss noch einmal über die Fläche A_i integriert und anschließend durch die Fläche A_i geteilt werden:

$$F_{ij} = \frac{1}{A_i} \cdot \int_{A_i} \int_{A_j} \frac{\cos\theta_i \cos\theta_j}{\pi r_{ij}^2} \cdot H_{ij} \quad dA_j dA_i$$

Anschaulich werden also die Werte für die unendlich vielen Punkte der Fläche A_i bestimmt und dann durch die Fläche geteilt.

Frage 9.25: Wie können die Formfaktoren praktisch berechnet werden?

Die Formel für die Formfaktoren enthält ein Doppelintegral, was eine Berechnung sehr unangenehm macht. Zudem enthält sie die Funktion H_{ij}, welche von der Geometrie abhängig ist. Aus diesem Grund wurden zwei weitere Verfahren entwickelt, welche ohne die Integrale und die Funktion H_{ij} auskommen. Das äußere Integral fällt weg, da bei beiden Verfahren einfach davon ausgegangen wird, dass es einen Punkt auf der Fläche gibt, der repräsentativ für die anderen Punkte bzw. für die Fläche selbst ist. Dies kann zum Beispiel der Mittelpunkt oder der Schwerpunkt einer Fläche sein. Die beiden Verfahren sind die **Halbkugeldarstellung** und die **Halbwürfelapproximation**.

Frage 9.26: Was ist die Halbkugeldarstellung?

Bei der Halbkugeldarstellung handelt es sich um eine andere Interpretation der Formfaktorberechnung, deren Grundidee in Abbildung 83 verdeutlicht wird. Wie weiter oben schon erwähnt, wird davon ausgegangen, dass ein Punkt der Fläche A_i für diese repräsentativ ist, wodurch das äußere Integral wegfällt. Um diesen repräsentativen Punkt dA_i wird nun eine Halbkugel mit dem Radius 1 aufgespannt. Soll nun der Anteil der Energie zwischen zwei Punkten berechnet werden, kann dies geometrisch wie folgt interpretiert werden, wobei das zweite Integral wegfällt, indem diese Schritte einfach mit allen von dA_i aus sichtbaren Punkten der Fläche durchführt werden:

- Der Punkt dA_j wird auf die Halbkugel projiziert. Dies entspricht einem Faktor von $\frac{\cos\theta_j}{r^2}$.

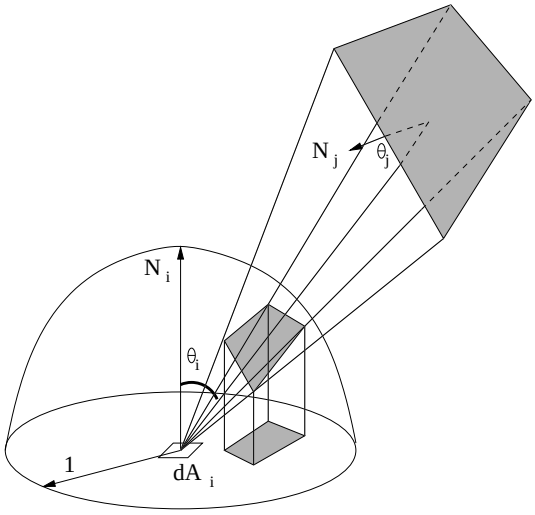


Abbildung 83: Halbkugeldarstellung des Radiosity-Verfahrens

- Anschließend wird der projizierte Punkt erneut projiziert und zwar orthogonal auf die kreisförmige Grundfläche der Halbkugel. Die entspricht einem Faktor von $\cos\theta_i$.

Um die vollständige Formel $(\cos\theta_j \cos\theta_i)/\pi r^2$ zu erhalten, fehlt nur noch der Faktor $\frac{1}{\pi}$. Das Ergebnis der Projektion der gesamten Fläche ist ein Teil der Grundfläche der Halbkugel. Da die Halbkugel den Radius $r = 1$ besitzt, beträgt der Flächeninhalt der Grundfläche $\pi \cdot r^2 = \pi \cdot 1^2 = \pi$. Wird nun durch π dividiert, wird also der Flächeninhalt der Doppelprojektion zur ganzen Grundfläche in Relation gesetzt. Die Formel ist somit komplett, was bedeutet, dass der Formfaktor das Verhältnis zwischen den beiden Flächeninhalten ist.

Frage 9.27: Wie funktioniert die Halbwürfelapproximation?

Da die Projektion einer Fläche auf eine Halbkugel sehr aufwändig sein kann, wird zur Annäherung statt einer Halbkugel ein Halbwürfel („hemi-cube“) benutzt. Wie in Abbildung 84 dargestellt, werden die fünf Seiten des Halbwürfels in gleich große quadratische Zellen aufgeteilt. Die Projektion wird nun für jede dieser fünf Seiten einzeln vollzogen. Dazu wird die Szene zuerst gegen den Pyramidenstumpf geclippt, der sich durch den Mittelpunkt dA_i und die vier Ecken einer jeden Würfelseite ergibt. Die verbleibenden Flächen werden anschließend auf die Würfelseite projiziert, wobei jede Zelle als ein Pixel aufgefasst wird. Zur Projektion kann das Tiefenpuffer-Verfahren (Z-Buffer) verwendet werden, welches den Vorteil hat, dass auf diese Weise auch gleich die verdeckten Flächenteile eliminiert werden. Die verschiedenen Flächen werden also quasi auf die Würfelseite gerastert.

Wie ergeben sich nun daraus die Formfaktoren? Jeder Zelle des Halbwürfels kann ein sogenannter Delta-Formfaktor zugewiesen werden, der sich aus der Position der Zelle ergibt. Im Prinzip wird die Zelle als eine quadratische Fläche aufgefasst, deren Formfaktor bezüglich dA_i berechnet wird. Da die Fläche quadratisch ist, kann dieser Formfaktor anscheinend relativ leicht berechnet werden. Der gesamte Formfaktor einer Fläche ergibt sich als die Summe aller

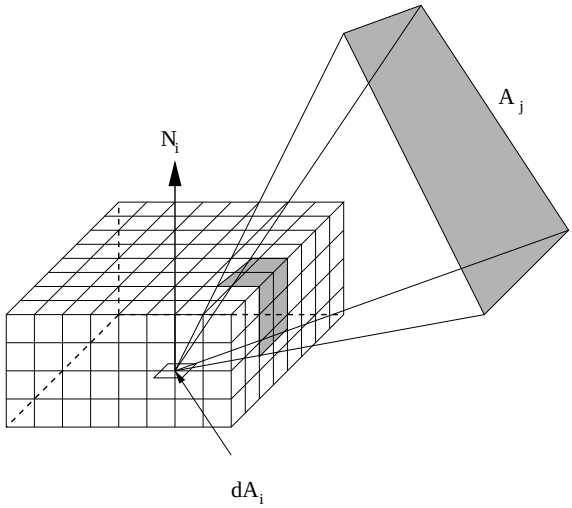


Abbildung 84: Verwendung eines Halbwürfels

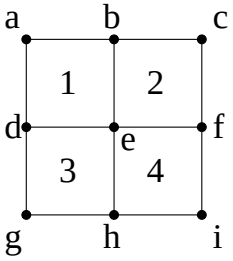


Abbildung 85: Berechnung der Strahlung der Eckpunkte

Delta-Formfaktoren der von der Projektion einer Fläche überdeckten Zellen. Der Delta-Formfaktor für eine Zelle p ergibt sich dabei als:

$$\Delta F_p = \frac{\cos\theta_i \cos\theta_p}{\pi r^2} \Delta A$$

Frage 9.28: Wie wird die Szene dargestellt, nachdem die Formfaktoren berechnet und das Gleichungssystem gelöst wurde?

Nach dem Lösen des Gleichungssystems liegen nur Strahlungswerte für ganze Flächenelemente vor. Es wäre jedoch wünschenswert, wenn jeder Eckpunkt einer Fläche einen Strahlungswerte hätte, denn dann könnten mit Hilfe des Gouraud-Shading glatte Übergänge erzeugt werden. Es sei ein Flächenverband wie in Abbildung 85 gegeben. Es wird zwischen Randecken und inneren Ecken unterschieden, wobei sich die Strahlung in den Punkten wie folgt ergibt:

Innere Ecke: Für eine innere Ecke ergibt sich ihre Strahlung als der Mittelwert aller zu ihr inzidenten Flächen.

Randecke: Für diese Ecken wird eine Gleichung aufgestellt. Es wird gefordert, dass der Mittelwert der Strahlung der Randecke und der zu ihr nächsten inneren Ecke gleich dem Mittelwert der Strahlung der zur Randecke inzidenten Ecken ist.

Das Beispiel aus Abbildung 85 hat nur eine innere Ecke, nämlich die Ecke e. Für diese ergibt sich die Strahlung B_e als:

$$B_e = \frac{B_1 + B_2 + B_3 + B_4}{4}$$

Für die Ecke b ergibt sich die Strahlung wie folgt, wobei die Strahlung der Ecken d, f und h analog ergibt:

$$\frac{B_b + B_e}{2} = \frac{B_1 + B_2}{2} \Leftrightarrow B_b = B_1 + B_2 - B_e$$

Und für die Ecke a ergibt sich (c, g und i analog):

$$\frac{B_a + B_e}{2} = B_1 \Leftrightarrow B_a = 2 \cdot B_1 - B_e$$

9.5 Darstellung von Voxelmodellen

Frage 9.29: Welche Verfahren zur Darstellung von Voxelmodellen kennst du?

Es gibt zwei Ansätze, die verfolgt werden können:

- Die Daten werden erst mit einem Verfahren, z.B. dem Marching-Cubes-Verfahren, in Primitive umgewandelt, welche die Oberfläche beschreiben. Diese Primitive können direkt gezeichnet werden.
- Es wird ein Verfahren der direkten Projektion benutzt, wovon es zwei gibt:
  - Bildordnungsstrategien
  - Volumenordnungsstrategien

Frage 9.30: Wie funktionieren die Bildordnungsstrategien?

Bei den Bildordnungsstrategien wird für jeden Bildpunkt berechnet, welche Voxel zur Intensität des Bildpunktes beitragen. Dazu wird von jedem Bildpunkt ein Strahl in die Szene geschickt und geschaut, welche Voxel geschnitten werden (bzw. in der Nähe liegen). Diese Strategie wird daher auch oft als **Ray-Casting** bezeichnet.

Frage 9.31: Wie funktionieren die Volumenordnungsstrategien?

Bei der Volumenordnungsstrategie wird von jedem Voxel aus gestartet. Es wird geschaut, zu welchen Bildpunkten es beiträgt und jeweils die Intensität an diesen berechnet. Dabei können zwei Strategien zur Abarbeitung der Voxel benutzt werden:

- Die Voxel werden nach fallender Entfernung zum Betrachter abgearbeitet. Diese Strategie heißt daher **Back-to-Front**. Für den Fall, dass nur die Oberfläche eines Voxelmodells gezeichnet werden soll, kann es passieren, dass viele Voxel gezeichnet werden, die später eh überzeichnet werden.

- Die Voxel werden nach steigender Entfernung zum Betrachter abgearbeitet. Diese Strategie heißt daher **Front-to-Back**. Soll nur die Oberfläche eines Voxelmodells gezeichnet werden, kann daher viel Arbeit gespart werden, wenn man sich vermerkt, welche Bildpunkte schon gezeichnet wurden.

Frage 9.32: Welche Möglichkeiten gibt es, den Beitrag eines Voxels zu einem Pixel zu bestimmen?

Es kann grob zwischen vier Verfahren unterschieden werden:

Oberflächendarstellung: Hier werden die Voxel bestimmt, die sich an der Oberfläche des zu zeichnenden Objektes befinden. Dies sind die Voxel, an denen die Intensitätsdifferenz zu ihren Nachbarvoxeln einen gewissen Grenzwert überschreitet.

Schnittebenendarstellung: Es werden die Voxel projiziert, die auf einer vorgegebenen Schnittebene liegen.

Integralprojektion: Die Intensitätswerte, welche auf dem durch das Pixel gehenden Strahl liegen, werden aufsummiert. Dabei werden die Intensitäten als Transparenz interpretiert. Diese Darstellung entspricht der eines Röntgenbilds.

Projektion der maximalen Intensität: Es wird der maximale Intensitätswert des auf dem durch das Pixel gehenden Strahls gezeichnet.

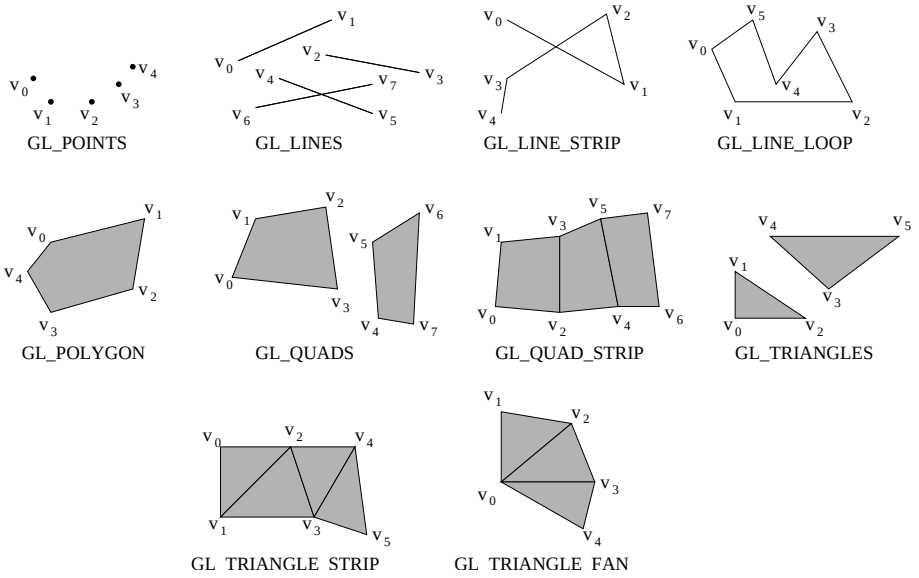


Abbildung 86: Die verschiedenen Zeichenmodi von OpenGL

10 OpenGL

OpenGL ist eine herstellerunabhängige Graphikschnittstelle, welche auf der von Silicon Graphics entwickelten Graphikbibliothek GL basiert. Sie ist als Programmbibliothek für verschiedene Programmiersprachen erhältlich.

Frage 10.1: Beschreibe das Grundkonzept von OpenGL!
Das Grundkonzept von OpenGL ist das einer Zustandsmaschine. Das bedeutet, dass der Kontext von Operationen und damit deren Wirkung durch den aktuellen Zustand von dieser Zustandsmaschine gegeben ist. Beispiele für Zustände sind die Farbe oder die Linienbreite mit der aktuell gezeichnet wird. Beim Starten von OpenGL werden diese Zustände auf sinnvolle Default-Werte gesetzt. Der aktuelle Wert einer Zustandsvariable kann stets abgefragt und manipuliert werden.

Frage 10.2: Welche Zeichenmodi gibt es bei OpenGL?
OpenGL bietet unter anderem die Möglichkeit Punkte, Linien und Polygone zu zeichnen. Welches dieser Elemente gezeichnet werden soll, kann über den Aufruf `glBegin(GLenum mode)` festgelegt werden. Anschließend folgt eine Liste von Punkten, die entsprechend des gewählten Modus interpretiert wird. Danach folgt ein Aufruf von `glEnd()`. Die verschiedenen möglichen Zeichenmodi sind in Abbildung 86 dargestellt.

Ein Punkt kann per `glVertex[234][sifd][v](TYPE coords)` spezifiziert werden. Dabei steht `[234]` für die Dimension des Punktes, d.h. ob dieser zwei- oder dreidimensional ist bzw. ob er in homogenen Koordinaten angegeben wird. Mit `[sifd]` wird der verwendete Datentyp angegeben, also Short, Integer, Float oder Double. Zu guter Letzt kann per `[v]` spezifiziert werden, ob die Koordinaten des Punktes in einem Array (Vector) übergeben werden.

Weiterhin können die Attribute von Punkten und Linien geändert werden.

Per `glPointSize()` kann die Größe eines gezeichneten Punktes in Pixeln eingestellt werden. Bei Linien kann die Linienbreite (`glLineWidth()`) und die Linienart (`glLineStipple()`) eingestellt werden.

Natürlich können auch die Attribute von Polygonen beeinflusst werden. Da bei Polygonen zwischen Vorder- und Hinterseite unterschieden wird, kann für beide unabhängig eingestellt werden, wie diese dargestellt werden sollen. Dies geht mit dem Befehl `glPolygonMode(GLenum face, GLenum mode)`. Dabei gibt der Parameter `face` an, ob die Einstellung für die Vorder- und/oder Hinterseite des Polygons gelten soll und `mode` bestimmt, ob das Polygon als Menge von Punkten, als Linienzug oder gefüllt dargestellt werden soll. Um Rechenzeit zu sparen, kann per `glCullFace(GLenum mode)` eingestellt werden, welche Polygone von vornherein verworfen werden sollen. Der Parameter `mode` gibt an, ob die Polygone, die zum Betrachter hinzeigen, die die wegzeigen oder beide weggeworfen werden sollen. Da für Operationen der Normalenvektor in einem Punkt benötigt wird, kann auch dieser per `glNormal()` angegeben werden.

Für jeden Punkt kann die Zeichenfarbe eingestellt werden. Dazu kann eine Farbe entweder direkt als RGB(A)-Wert angegeben werden oder ein Index in eine Farbtabelle benutzt werden. Ersteres geschieht bei Benutzung des Befehls `glColor()`. Mit `glClearColor()` wird die Hintergrundfarbe eingestellt und mit `glClear()` wird der Bildschirm gelöscht, d.h. mit der per `glClearColor()` spezifizierten Farbe gefüllt. Ein `glFlush()` bewirkt eine Ausführung des Zeichnens. Die Wahl des Schattierungsmodells geschieht mittels Aufruf der Funktion `glShadeModel(GLenum mode)`. Gültig Optionen sind Flat-Shading (`GL_FLAT`) oder Gouraud-Shading (`GL_SMOOTH`).

Frage 10.3: Wie werden Rasterdaten in OpenGL behandelt?

In OpenGL ist es ebenfalls möglich direkt auf den Bildwiederholspeicher (Frame Buffer) zuzugreifen. Es wird zwischen Bitmaps und Bildern unterschieden. Eine Bitmap ist ein rechteckiges Feld in dem nur Nullen und Einsen stehen. Sie kann z.B. als Zeichnungsmaske für einen Bildschirmbereich dienen, wobei in diesem Fall ein Pixel im Bildwiederholspeicher geändert wird, wenn es mit einer Eins maskiert ist. Die Position des Ursprungs der Bitmap wird mit `glRasterPos[234][sifd][v](coords T)` festgelegt. Die zu zeichnende Bitmap wird mittels `glBitmap()` an die letzte per `glRasterPos` gesetzte Position gezeichnet. Dies kann zum Beispiel benutzt werden, um Zeichensätze zu definieren und zu zeichnen.

Ein Bild ist ein rechteckiges Feld aus Werten. Diese können als Farbwerte, Tiefenwerte oder beliebige andere Werte interpretiert werden. Im Groben kann zwischen drei Operationen unterschieden werden:

glReadPixels: Diese Operation entspricht dem Kopieren von Daten aus dem Bildwiederholspeicher an eine andere Speicherposition außerhalb des Bildwiederholspeichers. Der Ausschnitt des Bildwiederholspeichers, der kopiert werden soll, wird in Form eines Rechtecks an die Funktion übergeben. Dieses Rechteck ist durch seinen Ursprung sowie seiner Breite und Höhe angegeben.

glDrawPixels: Hier werden Bilddaten aus einem Speicherbereich in den Bild-

wiederholspeicher kopiert.

glCopyPixels: Kopiert einen rechteckigen Bereich aus dem Bildwiederholspeicher an die zuletzt per `glRasterPos()` gesetzte Position.

Mittels der Funktion `glBlendFunc()` ist es möglich, Überblendungseffekte zu realisieren. Diese erhält als Parameter die Quell- und Zielüberblendungsfaktoren. Sind zwei Pixel $(R_q, G_q.B_q, A_q)$ und $(R_z, G_z.B_z, A_z)$ sowie die Überblendfaktoren (Q_r, Q_g, Q_b, Q_a) und (Z_r, Z_g, Z_b, Z_a) gegeben, ergibt sich daraus das neue Pixel als:

$$(R_qQ_r + R_zZ_r,\ G_qQ_g + G_zZ_g,\ B_qQ_b + B_zZ_b,\ A_qQ_a + A_zZ_a)$$

Frage 10.4: Wie können komplizierte Objekte effizient mehrfach gezeichnet werden?

Mit Hilfe von Display-Lists können komplizierte Objekte durch einen Aufruf gezeichnet werden. Dazu wird per `glNewList(GLuint list, GLenum mode)` eine neue Liste angelegt. Anschließend wird das Objekt, wie beim „normalen“ Zeichnen, durch die entsprechenden Aufrufe beschrieben. Nach der Erzeugung wird `glEndList()` aufgerufen. Der Aufruf von `glCallList(GLuint list)` an einer Stelle im Code bewirkt dann das Zeichnen des Objektes als ob die Befehle erneut eingegeben werden. Weitere praktisch Funktionen sind das Erstellen von n Display-Listen, das Löschen einer Liste und die Überprüfung der Existenz einer Liste. Es ist ebenfalls möglich, Listen hierarchisch zu verschlachteln.

Frage 10.5: Welche Arten von Transformationen gibt es in OpenGL?

Transformationen werden in OpenGL durch verschiedene Matrizen dargestellt. Es kann zwischen der Modelview-, der Projektions- und der Texturmatrix unterschieden werden. Auf diesen Matrizen sind viele der üblichen Matrixoperationen möglich. Auf welche der Matrizen sich eine Operation auswirkt, wird durch die Funktion `glMatrixMode(GLenum mode)` angegeben. Dabei ist `mode` entweder `GL_MODELVIEW`, `GL_PROJECTION` oder `GL_TEXTURE`. Mit der Funktion `glLoadMatrix[fd](const TYPE *m)` kann eine Matrix geladen werden. Soll dies die Einheitsmatrix sein, kann die Funktion `glLoadIdentity()` benutzt werden. Desweiteren ist es möglich, die ausgewählte Matrix mit einer anderen Matrix zu multiplizieren. Dazu wird die Funktion `glMultMatrix[fd](*m)` benutzt. Dies entspricht, wie schon in Kapitel 3.4 beschrieben, in vielen Fällen der Hintereinanderausführung von Abbildungen.

Für jede der verschiedenen Matrizenarten wird ein Stack verwaltet auf dem Matrizen abgelegt und runtergenommen werden können. Dies ist mit Hilfe der Funktionen `glPushMatrix()` und `glPopMatrix()` möglich. Stacks können z.B. gut für relative Transformationen und Hierarchien benutzt werden.

Man kann bei OpenGL zwischen vier verschiedenen Transformationen unterscheiden. Diese werden hier nun kurz vorgestellt, wobei jedesmal das Äquivalent der realen Welt aufgezeigt wird:

Blicktransformation: Diese wird auf der Projektionsmatrix ausgeführt und entspricht der Positionierung einer Kamera in einer Szene. In OpenGL wird

dies mit Hilfe des Aufrufs `gluLookAt()` realisiert. Diesem werden insgesamt neun Koordinaten übergeben, wovon jeweils drei Koordinaten einen Punkt im Raum bzw. eine Richtung beschreiben. Es wird der Standpunkt und der Blickpunkt der Kamera und die Richtung nach Oben angegeben.

Modelltransformation: Diese Transformation entspricht der Positionierung der Objekte in der Szene. Wird bei OpenGL per `glVertex()` ein Punkt in der Szene gesetzt, so wird dieser erst mit der Modelview-Matrix multipliziert. Die Modelview-Matrix entspricht also einer affinen Abbildung, die auf dem Punkt angewendet wird. Wie schon in Kapitel 3.4 beschrieben, können dies z.B. Translation, Rotation und Skalierung sein. Die entsprechenden Funktionen sind `glTranslate()`, `glRotate()` und `glScale()`.

Projektionstransformation: Diese entspricht der Einstellung der Linse einer Kamera. Es kann zwischen Perspektiv- und Zentralprojektion unterschieden werden. Die Perspektivprojektion kann entweder mit Hilfe der Funktion `glFrustrum()` oder `gluPerspective()` angegeben werden. Für die Parallelprojektion stehen `glOrtho()` bzw. `glOrtho2D()` zur Verfügung.

Bildausschnittstransformation: Dieser Schritt entspricht quasi dem Drücken des Auslösers. Der Funktion `glViewport()` werden die folgenden Parameter übergeben: die linke untere Ecke der rechteckigen Bildausschnitts, sowie dessen Höhe und Breite in Pixeln.

Frage 10.6: Auf welche Art und Weise werden Licht und Materialeigenschaften in OpenGL berücksichtigt?

Die Beleuchtungsberechnungen in OpenGL werden anhand des Beleuchtungsmodell von Phong vorgenommen, welches in Kapitel 9.1 vorgestellt wurde.

Abbildungsverzeichnis

Tabellenverzeichnis

Liste der Algorithmen

Literatur

[Fel92] Fellner, Wolf D.: *Computergrafik*. Zweite Auflage. BI Wissenschaftsverlag, 1992

[Fol94] Foley, James D.: *Grundlagen der Computergrafik*. Erste Auflage. Addison-Wesley, 1994

[Mül00] Müller, Heinrich. *Skript Graphische Systeme*. 2000

Index