INTRODUCTION TO COMPUTER GRAPHICS

(bachelor/master scheikunde, natuurkunde, informatica)

maart 2008

Radboud Universiteit Nijmegen

Faculteit der Natuurwetenschappen, Wiskunde en Informatica
P.F. Klok
A.J. Dammers

$\Longrightarrow$ *Deze aanduiding wordt gebruikt om extra aandacht voor een tekstgedeelte te vragen.*

Contents

List of Figures

Hoofdstuk 1

Inleiding

1.1 Algemeen

Inleiding Computer Graphics bestaat uit een college met bijbehorend practicum. Bedoeling is om op het college een inleiding tot de basisbegrippen en -technieken op het gebied van **computer graphics** te geven, aangevuld met het opdoen van praktijkervaring tijdens het practicum. Daarnaast zal, zowel op het college als tijdens het practicum, aandacht aan het onderwerp **beeldverwerking** besteed worden.

Voor het kunnen volgen van Inleiding Computer Graphics is geen speciale **voorkennis** vereist, maar het is wel belangrijk om de nodige **programmeervaardigheid** te hebben om de opdrachten van het practicum snel genoeg te kunnen maken.

Tijdens het **college** wordt het boek "**Introduction to Computer Graphics**" van Foley et al. (zie [1]) gebruikt als tekstboek voor de theorie en het boek "**Computer Graphics Using OpenGL**" (Second Edition) van F.S. Hill, Jr. (zie [3]) als handboek bij het practicum.
Voor beeldverwerking wordt het boek "**Digital Image Processing**" van Gonzalez en Woods (zie [10]) aanbevolen.
Deze **handleiding** [2] is een samenvatting van de verschillende onderwerpen die op het college behandeld zullen worden. De bij dit vak behorende website zal steeds de meest "verse" informatie geven, het adres daarvan is

`http://www.hef.ru.nl/~pfk/teaching/cg/`

Voor het maken van de opdrachten van het **practicum** worden grafische werkstations gebruikt met Unix als besturingssysteem, met een op X-Windows gebaseerd gebruikersinterface, met C als programmeertaal en met OpenGL als grafische software.

Aan het eind van het semester wordt een schriftelijk **tentamen** afgenomen. Het eindcijfer van Inleiding Computer Graphics wordt bepaald door beide resultaten, waarbij dat van het practicum drie maal zwaarder telt, met als bijkomende voorwaarde dat het practicumcijfer voldoende moet zijn.

Zie — naast de website — de appendices B, C en D van deze handleiding voor informatie betreffende benodigde software en hardware.

1.2 Computer Graphics

Computer graphics omvat alle aspecten van het met behulp van een computer genereren van grafische gegevens.

Binnen computer graphics kunnen de volgende **ontwikkelingsgebieden** onderscheiden worden:
- het **ontwerpen van grafische hardware**, enerzijds voor het tonen van beelden, anderzijds voor het snel genereren van die beelden,
- het **opstellen van algoritmen** om de benodigde grafische technieken snel en efficiënt uit te kunnen voeren,
- het **ontwikkelen van software** voor grafische systemen en voor grafische toepassingen voor gebruikers, waarbij van de beide vorige punten gebruik wordt gemaakt.

Tijdens college en practicum zullen het tweede en derde gebied het meest aan bod komen.

Toepassingen van computer graphics kunnen globaal verdeeld worden in:
- al dan niet interactieve **presentatie van informatie** (grafieken, afbeeldingen, kartografie, XV, WWW),
- **interactief ontwerp**, CAD of Computer Aided Design (mechanisch ontwerp, electronische schakelingen, architectuur, electronic publishing, Coreldraw, Xpaint),
- **simulatie** en **animatie** (vluchtsimulatie, tekenfilms, video-spelletjes),
- **gebruikersinterface** (windowsystemen, WWW browsers),
- **computer art** (kan iets of niets van elke vorige toepassing hebben).

Vrijwel al deze toepassingen kunnen op elk willekeurig gebied aangewend worden (waarbij de vraag is of dat altijd nuttig is).

De **grafische technieken** die voor de verschillende toepassingen gebruikt worden, zijn gebaseerd op:
- 2D lijntekeningen (gebruik van lijnen),
- 2D beelden (gebruik van vlakken),
- 3D lijntekeningen (lijnen, wire frames met hidden line removal),
- 3D beelden (vlakken, solid modeling met hidden surface removal).

Binnen deze groepen zijn er nog grote verschillen mogelijk:
- er kan monochroom, met grijstinten of met kleuren gewerkt worden,
- er kan met intensiteitsverschillen gewerkt worden,
- er kan met egale of met niet-egale vlakken gewerkt worden.

De geschiedenis van computer graphics begint met de **beeldbuis** (**CRT** of **Cathode Ray Tube**). Hierop werden de eerste tekeningen, uit lijnen of **vectoren** opgebouwd, gemaakt. De aanmerkelijk goedkopere **vector refresh storage tube** (**DVST** of **Direct Vision Storage Tube**) was door een veel groter publiek te gebruiken. Door het gebruik van **display processor**, **raster scan CRT**'s en **frame buffers** werd de ontwikkeling verder voortgezet. Met het beschikbaar komen van **chips** met steeds grotere dichtheden kwam de ontwikkeling in een enorme stroomversnelling, met als resultaat het gebruik van graphics in **gebruikersinterfaces** (**HCI** of **Human Computer Interface** en **GUI** of **Graphical User Interface**), de ontwikkeling van moderne **grafische softwaresystemen**, **animatie**, **solid modeling**, enz. Dit alles gecombineerd leverde **grafische werkstations**.

Wat is er nodig aan hardware om graphics te bedrijven? In principe kan alles met de basisstenen van een gewone computer (processor, geheugen, invoer- en uitvoerapparatuur) gedaan worden. Het resultaat zal — zoals te verwachten is — niet erg aanspreken. Want om het grote oplossend vermogen en de grote snelheden van geavanceerde grafische apparatuur te kunnen halen, zijn er een aantal grote ontwikkelingen in de computerarchitectuur nodig geweest. Hierover later.

Zie [1], hoofdstuk 1, voor een uitgebreide verwoording van de hierboven behandelde stof.

1.3 Beeldverwerking

Beeldverwerking houdt in het met behulp van een computer bewerken van bestaande grafische gegevens.

Op het gebied van beeldverwerking (of beeldbewerking) kunnen onder andere de volgende onderwerpen onderscheiden worden:
- **reconstructie**: het reconstrueren van beelden uit een aantal projecties daarvan,
- **beeldverbetering**: het verbeteren van beelden (en **niet** het verrijken van beelden!),
- **datacompressie**: technieken om grote hoeveelheden gegevens te verminderen zonder informatie te verliezen.

De op het college te behandelen onderwerpen staan beschreven in de hoofdstukken 11-13 van deze handleiding. Een overzicht met uitgebreide behandeling van de verschillende onderwerpen is bijvoorbeeld te vinden in [10].

Hoofdstuk 2

Basisbegrippen

Figuren kunnen worden opgebouwd uit een aantal basiselementen of **primitieven**. Bij het maken van een eenvoudige tweedimensionale grafiek, bijvoorbeeld, zijn er maar twee primitieven nodig: lijn en tekst. Maar voordat de grafiek op beeldscherm of plotter getekend is, zijn er een aantal dingen gebeurd: de coördinaatparen die de waarden van de grafiek bepalen, zijn aangepast aan de ruimte die er op scherm of plotter beschikbaar is, de grootte van de teksten is aangepast, primitieven zijn samengevoegd tot deelfiguren, daar zijn mogelijk bewerkingen op uitgevoerd (vergroten, verplaatsen, draaien), en uiteindelijk zijn de deelfiguren tot één figuur samengevoegd.

Bij het afbeelden van **driedimensionale beelden** is er een grote overeenkomst tussen het moderne computer graphics en een camera of een menselijk oog. Bij de laatste wordt de afbeelding achter het projectiecentrum (de lens) op het netvlies gemaakt. Maar de afbeelding kan ook op een vlak tussen het beeld en het projectiecentrum gemaakt worden, er wordt als het ware door een **venster** of **window** gekeken, waarbij de afbeelding in het venstervlak gemaakt wordt. Door het venster — met het beeld- of projectievlak — te verschuiven, wordt een ander deel van de ruimte afgebeeld. Merk op dat beelden in die ruimte ook kunnen bewegen en dat het projectiecentrum ook in de ruimte kan bewegen. Op deze wijze wordt er een camera nagedaan: de analogie van de **synthetische camera**. Het kijken door een venster dat verschoven kan worden, bepaalt wat er gezien en dus afgebeeld wordt.

Met de synthetische camera valt het probleem van afbeelden uiteen in twee delen:
- de specificatie van **afmeting, positie en oriëntatie van objecten in de ruimte**,
- de specificatie van **viewing informatie** (vanuit welke positie, in welke richting en door welk venster wordt er naar de objecten gekeken).

Vóór het verschijnen van grafische systemen moesten de bewerkingen, die zorgen voor een juiste afbeelding (de **viewing informatie**), door de programmeur zelf gemaakt worden. Door deze bewerkingen te formaliseren tot "standaard"-bewerkingen en deze in een aantal procedures of functies van een grafisch systeem te verpakken, is het voor een gebruiker niet meer nodig het basisbewerkingen zelf te doen.

Een andersoortig probleem dat opgelost moet worden, is het feit dat verschillende uitvoer- en invoerapparaten vrijwel gegarandeerd verschillende **coördinaatsystemen** hebben. Het is gewenst om **device-onafhankelijkheid** te hebben bij het gebruik van grafische software, zodat de gebruiker niet steeds aanpassingen moet doen bij het gebruik van een ander in- of uitvoerapparaat (device). Dit laatste wordt meestal opgelost door in een grafisch systeem verschillende coördinaatsystemen te gebruiken:
- voor specificatie van objecten door de gebruiker,
- voor intern gebruik in het grafische systeem,
- voor een specifiek in- of uitvoerapparaat.

De **viewing informatie** wordt geïmplementeerd door het gebruik van **windows** en **viewports**. Windows zijn in WC en geven een venster aan. Viewports zijn in NDC en geven aan welk gedeelte van het device gebruikt wordt om het venster op af te beelden. Een window wordt dus via een viewport op een device afgebeeld.

Bij het afbeelden zal er vroeg of laat een deel van een beeld binnen en een deel buiten het viewport vallen: het beeld moet op de rand afgesneden worden. Dit wordt met de term **clipping** aangeduid. Algoritmen om efficiënt te clippen zijn zeer belangrijk om snel te kunnen werken.

Het tekenen van beelden wordt meestal gedaan met rechte lijnstukken, die **vectoren** genoemd worden. Deze

hebben een richting, een lengte, een begin- en een eindpunt. Bij het tekenen van beelden met behulp van vectoren kan er op twee manieren gewerkt worden:

- door middel van **absoluut adresseren**,
- door middel van **relatief adresseren**.

Het beschrijven van krommen kan op drie manieren gedaan worden:

- in **expliciete vorm** door $y = f(x)$ (2D) en door $y = f(x)$ en $z = g(x)$ (3D),
- in **impliciete vorm** door $f(x, y) = 0$ (2D) en door $f(x, y, z) = 0$ (3D),
- in **parametrische vorm** door $x = x(t)$ en $y = y(t)$ (2D) en door $x = x(t)$, $y = y(t)$ en $z = z(t)$ (3D).

Het hangt van het doel af welk van deze manieren gebruikt wordt. Er zijn bijvoorbeeld problemen bij de expliciete vorm en met horizontale en vertikale lijnen en met de definitie van cirkel (er wordt maar een halve cirkel beschreven). Er zijn ook problemen bij de impliciete vorm bij het beschrijven van een halve cirkel en bij het continu aan elkaar laten sluiten van verschillende krommen. Daarom zal meestal de parametrische vorm gebruikt worden voor het beschrijven van krommen (zie hoofdstuk 10).

Hoofdstuk 3

Grafische pakketten

3.1 Algemeen

Analoog aan de hogere programmeertalen zijn er voor grafische toepassingen standaarden ontwikkeld, die het voordeel hebben algemeen — d.w.z. op meerdere typen computers — gebruikt te kunnen worden. Voorbeelden van dergelijke grafische standaarden zijn **GKS** (Graphical Kernel System), **PHIGS** (Programmer's Hierarchical Interactive Graphics System), **PEX** (PHIGS Extension for X-windows) en **OpenGL**. Kenmerken van dergelijke systemen zijn:

- er zijn meerdere **logische** werkstations te gebruiken (die mogelijk hetzelfde **fysieke** werkstation gebruiken),
- figuren kunnen uit **deelfiguren** opgebouwd worden (**segmentatie**),
- figuren kunnen op een aantal manieren **getransformeerd** worden,
- het is mogelijk om grafische informatie **op te slaan** in een bestand en om dergelijke opgeslagen informatie later weer te **gebruiken**,
- de gebruiker kan de **wijze van foutafhandeling** zelf regelen.

Een gebruiker zal dus met één of meer logische in- en uitvoerapparaten werken, die uiteindelijk met één of meer fysieke werkstations blijken te corresponderen. Voor een gebruiker is ook belangrijk welke procedures of functies aangeroepen kunnen worden, welke parameters die hebben en wat de typen van die parameters zijn. De precieze beschrijving van deze namen en aanroepregels voor een bepaalde computertaal wordt de **language binding** genoemd. Daarnaast zijn er **device drivers** nodig voor de communicatie tussen de grafische software en de hardware. Dit laatste is echter volledig transparant voor de gebruiker.

3.2 OpenGL

Tijdens het practicum wordt gebruik gemaakt van het grafische pakket **OpenGL** en de programmeertaal **C**. Informatie over (het gebruik van) OpenGL is te vinden via de bij het college horende webpagina

`http://www.hef.ru.nl/~pfk/teaching/icg/`

Een korte samenvatting van C staat in appendix C. Verwijzingen naar een aantal C-cursussen staan vermeld op de bovengenoemde webpagina van dit vak.

Hoofdstuk 4

Interactive Graphics

Bij **interactieve grafische toepassingen** zijn o.a. de volgende aspecten van belang:
- **interactie** met de gebruiker,
- **invoer** van gegevens om het maken van de gewenste figuren te sturen,
- **manipulatie** van (deel)figuren,
- **vormgeving** van complexe figuren.

Hierna volgt een opsomming van **elementen** en **technieken** voor interactieve grafische toepassingen:

- Het gebruik van **windows** en **viewports** voor het scheiden van gebieden waarin verschillende functies van het programma actief zijn. Er kunnen bijvoorbeeld aparte windows of specifieke gedeelten van een window of van het scherm voor uitvoer en voor invoer gebruikt worden.
- Het gebruik van **grafische objecten** als basiselementen die gemanipuleerd kunnen worden. Zo kunnen iconen als grafische objecten bijvoorbeeld gebruikt worden om een keuze te maken.
- Afhankelijk van de snelheid van de grafische hardware kunnen veranderingen op het scherm automatisch getoond worden (**implicit screen regeneration**) of kan er door de gebruiker aangegeven worden wanneer het scherm opnieuw getekend moet worden (**explicit screen regeneration**).
- Er zijn een aantal **structuurattributen** die — in tegenstelling tot de attributen van primitieven — tijdens display van de structuren waarop ze van toepassing zijn, veranderd kunnen worden en daarmee ook actief worden. Dergelijke attributen zijn bijvoorbeeld:
  - **visibility**,
  - **priority**,
  - **highlighting**,
  - **detectability**,
  - **transformation**.
- Voor de invoermogelijkheden voor grafische systemen is er onderscheid gemaakt tussen **logische** en **fysieke invoer**. In een programma wordt van de logische invoer gebruikt gemaakt, terwijl het grafische systeem zorgt voor de koppeling met de fysieke invoer. Dit heeft als voordeel dat programma's onafhankelijk van de hardware zijn (en blijven).
- De **logische invoer** is verdeeld in de volgende **invoerklassen**:
  - **locator**, voor een positie,
  - **string**, voor een tekst,
  - **pick**, voor een grafisch object,
  - **choice**, voor een keuze uit een aantal mogelijkheden,
  - **stroke**, voor een aantal posities,
  - **valuator**, voor een waarde.
- **Invoer** kan algemeen beschreven worden door middel van een **triggerproces** en een **uitleesproces**. De **invoerprocedure** zorgt voor een prompt om aan te geven dat er op invoer gewacht wordt, start het uitleesproces en geeft na afloop het resultaat en **status**informatie terug. De wijze van invoer wordt verdeeld in de volgende **invoermodes**:
  - **request**, wacht op trigger en lees uit,
  - **sample**, lees uit,
  - **event**, wacht op triggers van verschillende soorten invoer en zet het resultaat van de invoer in een wachtrij of **event queue** om uitgelezen te worden.
- De fysieke **invoerapparatuur** kan verdeeld worden in:

  - **toetsenbord**, te gebruiken als string, pick device en locator,
  - **lichtpen**, te gebruiken als pick device en locator,
  - **joystick**, te gebruiken als pick device en locator,
  - **trackball** en **muis**, te gebruiken als pick device en locator,
  - **data tablet**, te gebruiken als locator,
  - **grafische invoerapparatuur** van bovenstaande typen, te implementeren via beeldscherm en pick device.
- Afhankelijk van invoerklasse en invoermode zijn er een aantal **invoertechnieken**. Door de initialisatie wordt de wijze van uitlezen bepaald. Belangrijk bij elke invoertechniek is de **return status**, omdat deze gebruikt kan worden om informatie over het uitgelezene aan de gebruiker te geven. Invoertechnieken met de daarvoor belangrijke aspecten zijn:
  - **picking**, met gebruik van pick identifier in plaats van segment identifier,
  - **locator**, met het gebruik van de juiste transformatie,
  - **string invoer**, met het opvragen van informatie over de grootte van de ruimte waarin tekst weergegeven kan worden (implementatie- en apparatuurafhankelijk),
  - **event-driven invoer**, met gebruik van een time-out om het gebrek aan gebruikersactiviteit op te vangen.

Aan de hand van een voorbeeldprogramma (shape layout program, bestand slp.c) wordt een en ander duidelijk gemaakt.

Zie [1], hoofdstuk 8.

Hoofdstuk 5

User Interface

Het eenvoudigste — op sommige systemen nog gebruikte — **user interface** of **gebruikersinterface** bestaat uit een **prompt** (bijvoorbeeld een dollarteken), waarachter een commando ingegeven kan worden. Na uitvoering van het commando verschijnt er weer een prompt. Is er een tikfout in het commando gemaakt of is er een niet-bestaand commando uitgegeven, dan verschijnt er een **foutboodschap** (bijvoorbeeld een vraagteken), gevolgd door een nieuwe prompt.

Een "modern" gebruikersinterface bestaat uit iets meer elementen van verschillende aard:

- **Windows**, die elk een toepassing bevatten, bijvoorbeeld een terminal sessie, een tekenprogramma, een editor, een analoge klok. Had je vroeger maar één window, het terminalscherm van 24 bij 80 tekens, nu heb je een bijna willekeurig aantal schermen met niet alleen tekst- maar ook met grafische mogelijkheden.
- **Menu's**, deze hebben het voordeel dat niet de commando's (die hele dikke onbegrijpelijke manual), maar alleen de werking van het menu aan de gebruiker bekend moet zijn, want via de verschillende keuzemogelijkheden van het menu komt de gebruiker aan de weet wat er gedaan kan worden. Naast "gewone" menu's — een rijtje mogelijkheden — zijn er, afhankelijk van de beschikbare grafische mogelijkheden, ook **pull down menu**'s en **pop-up menu**'s die zo mogelijk **hiërarchisch** georganiseerd zijn.
- **Iconen**, deze symbolen zijn — indien goed ontworpen — internationaal bruikbaar. Ze worden bijvoorbeeld gebruikt om de windows die gesloten (niet op het scherm zichtbaar) zijn, aan te duiden. Aan de vorm van het icoon is dan te zien welke toepassing in het window draait. De cursor kan ook door middel van iconen op verschillende manieren afgebeeld worden om toestanden van het systeem aan te geven.
- **Interactieve helpfuncties**, om bijvoorbeeld informatie over de werking van een toepassing waar je mee bezig bent, op te vragen.
- **Terugkoppeling** of **feedback**, om fouten of de wijze van corrigeren daarvan aan te geven, of om informatie over of mogelijkheden van het huidige menu te geven.
- Extra **hulpmiddelen**, deze kunnen gebruikt worden om grotere nauwkeurigheid te krijgen bij het positioneren op het scherm of bij het geven van invoer. Voorbeelden zijn maatlatten langs windows, crosshairs of roosters in windows en het gebruik van zooming.
- **Layout**, belangrijk om de informatie en de windows op het beeldscherm beter te presenteren aan de gebruiker. Windows kunnen bijvoorbeeld wel of niet overlappen. Het is belangrijk dat de layout aan persoonlijke wensen aangepast kan worden.
- **Kleur**, hiermee kan het op het beeldscherm gepresenteerde duidelijker gemaakt worden en kunnen er accenten gelegd worden. Vaak is het mogelijk een eigen kleurenpalet te kiezen of te maken, waardoor, eventueel ook met een zelfgekozen achtergrond, aan het scherm een persoonlijke tint gegeven kan worden. Het is daarom handig om eerst even naar het scherm van een gebruiker te kijken: met die kleuren kan ik die lastige vraag nu beter niet stellen...

Dat de genoemde elementen niet voor niets gebruikt worden, spreekt vanzelf. Een gebruikersinterface dat intensief van deze elementen gebruik maakt, zal veel ruimte innemen op disk en zal veel tijd kosten om op te starten en te werken. Ofwel, er is erg veel **overhead** bij een dergelijk systeem. Maar door de steeds grotere snelheid en de grotere (èn goedkopere) geheugens zijn betere gebruikersinterfaces, gebaseerd op bovengenoemde elementen, meestal zonder problemen te gebruiken.

De Apple Macintosh was de eerste computer met een windowsysteem. Daarna heeft het zeker een tiental jaren geduurd voordat windowsystemen gemeengoed werden, maar nu zijn ze beschikbaar voor vrijwel alle systemen.

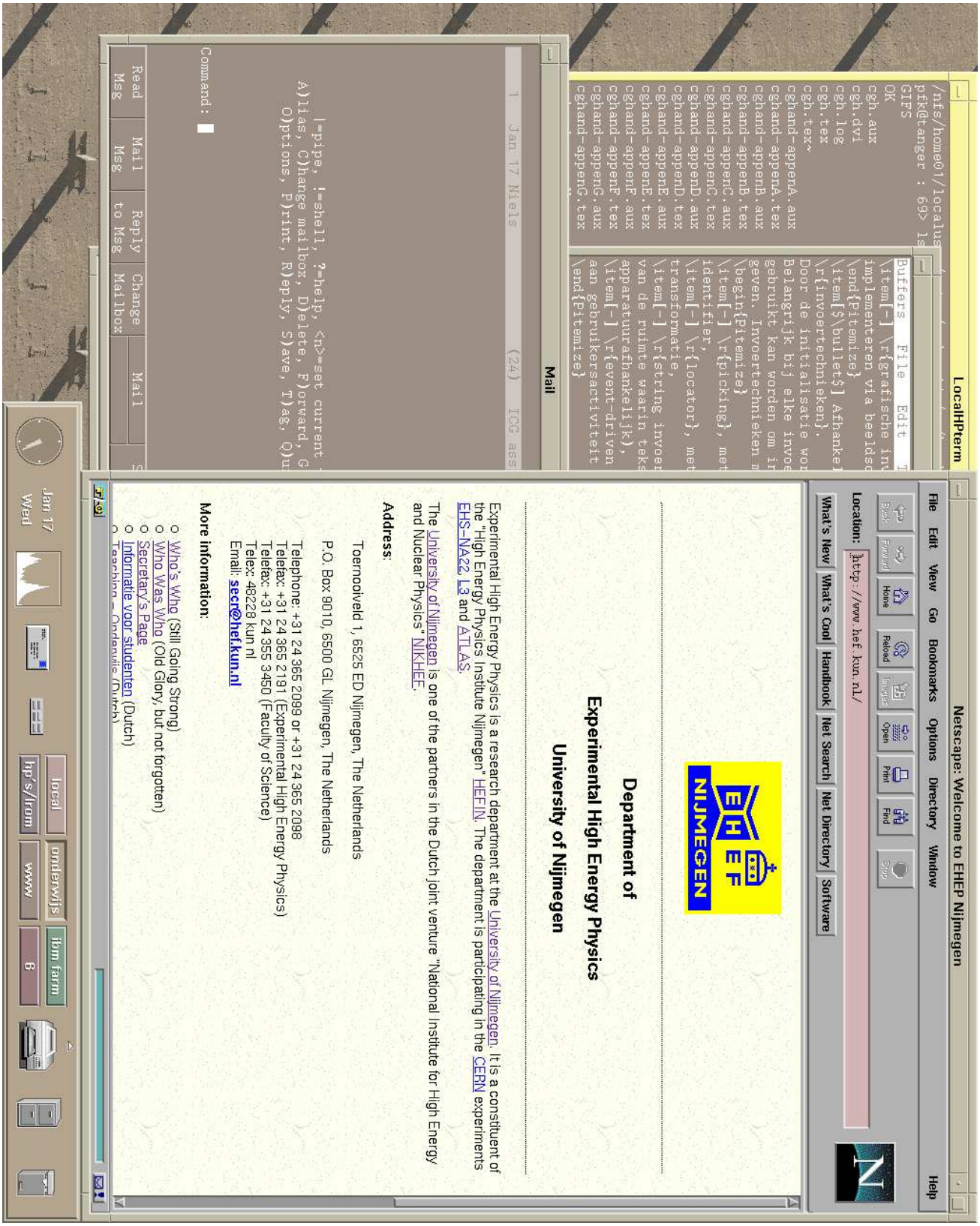


Figure 5.1: *Voorbeeld van een user interface gebaseerd op X-windows.*

Hoofdstuk 6

2D transformaties

6.1 Affine transformaties

Manipuleren van objecten in computer graphics is meestal terug te voeren tot het afbeelden van (delen van) rechte lijnen, waarbij de volgende punten gelden:
- lijnstukken zijn bepaald door een **begin**- en een **eindpunt**,
- lijnen **gaan** bij afbeelding weer **over in** lijnen.

Transformaties die dergelijke afbeeldingen — waarbij lijnen in lijnen overgaan — beschrijven, heten **affine transformaties**. Tot de afbeeldingen die met affine transformaties uitgevoerd kunnen worden, behoren:

- **Translatie**.
  De afbeelding van een punt $\mathbf{p} = \begin{bmatrix} x \\ y \end{bmatrix}$ op een punt $\mathbf{p}' = \begin{bmatrix} x' \\ y' \end{bmatrix}$ wordt gegeven door een transformatie (met d_x als verplaatsing in de x-richting en d_y als verplaatsing in de y-richting) $\mathbf{t} = \begin{bmatrix} d_x \\ d_y \end{bmatrix}$ en kan geschreven worden als $\mathbf{p}' = \mathbf{p} + \mathbf{t}$.

- **Rotatie** om de oorsprong.
  De afbeelding van een punt $\mathbf{p} = \begin{bmatrix} x \\ y \end{bmatrix}$ op een punt $\mathbf{p}' = \begin{bmatrix} x' \\ y' \end{bmatrix}$ wordt gegeven door een transformatie (met θ als de hoek waarover gedraaid wordt) $\mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$ en kan geschreven worden als $\mathbf{p}' = \mathbf{R} \cdot \mathbf{p}$.

- **Schaalverandering** (vergroting of verkleining).
  De afbeelding van een punt $\mathbf{p} = \begin{bmatrix} x \\ y \end{bmatrix}$ op een punt $\mathbf{p}' = \begin{bmatrix} x' \\ y' \end{bmatrix}$ wordt gegeven door een transformatie (met schaalfactor s_x in de x-richting en schaalfactor s_y in de y-richting) $\mathbf{S} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$ en kan geschreven worden als $\mathbf{p}' = \mathbf{S} \cdot \mathbf{p}$.

- **Shearing** (of proportioneel verschuiven langs een as).
  Voor shearing langs de x-as wordt de afbeelding van een punt $\mathbf{p} = \begin{bmatrix} x \\ y \end{bmatrix}$ op een punt $\mathbf{p}' = \begin{bmatrix} x' \\ y' \end{bmatrix}$ gegeven door een transformatie (met a als de verschuivingsfactor langs de x-as) $\mathbf{H}_x = \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix}$ en kan geschreven worden als $\mathbf{p}' = \mathbf{H}_x \cdot \mathbf{p}$.
  Voor shearing langs de y-as wordt de afbeelding van een punt $\mathbf{p} = \begin{bmatrix} x \\ y \end{bmatrix}$ op een punt $\mathbf{p}' = \begin{bmatrix} x' \\ y' \end{bmatrix}$ gegeven door een transformatie (met b als de verschuivingsfactor langs de y-as) $\mathbf{H}_y = \begin{bmatrix} 1 & 0 \\ b & 1 \end{bmatrix}$ en kan geschreven worden als $\mathbf{p}' = \mathbf{H}_y \cdot \mathbf{p}$.

Een belangrijke **eigenschap** van affine transformaties is de volgende: omdat lijnen behouden blijven, kunnen twee opeenvolgende affine transformaties vervangen worden door één affine transformatie, die dezelfde afbeelding geeft. Omgekeerd kan één affine transformatie opgesplitst worden in meerdere affine transformaties, die na elkaar uitgevoerd dezelfde afbeelding geven. Het na elkaar uitvoeren van een aantal transformaties wordt **concatenatie** genoemd.

6.2 Homogene coördinaten

Het concateneren van transformaties kan gebruikt worden om ingewikkelde afbeeldingen op te breken in eenvoudige afbeeldingen, die uiteindelijk hetzelfde resultaat leveren. Een voorbeeld hiervan is het roteren om een willekeurig punt. Het is mogelijk om dit in één keer te doen, maar een dergelijke rotatie kan ook opgesplitst worden in een translatie naar de oorsprong, een rotatie om de oorsprong en een translatie terug naar het oorspronkelijke rotatiepunt.

Een probleem bij dit voorbeeld is, dat de eerste translatie een optelling van vectoren is, dat de rotatie een vermenigvuldiging van vector met matrix is en dat de tweede translatie weer een optelling van vectoren is. Om deze verschillende bewerkingen te kunnen combineren, wordt er gebruik gemaakt van **homogene coördinaten**.

Een tweedimensionaal punt $\begin{bmatrix} x \\ y \end{bmatrix}$ wordt in homogene coördinaten weergegeven door een driedimensionaal punt $\begin{bmatrix} wx \\ wy \\ w \end{bmatrix}$ waarbij w niet gelijk aan nul mag zijn. Als $w = 1$ genomen wordt, kan een translatie $\mathbf{T}(d_x, d_y)$ van $\mathbf{p} = \begin{bmatrix} x \\ y \end{bmatrix}$ naar $\mathbf{p}' = \begin{bmatrix} x' \\ y' \end{bmatrix}$ weergegeven worden door:

$$\mathbf{p}' = \mathbf{T}(d_x, d_y) \cdot \mathbf{p} \quad \text{ofwel} \quad \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Op dezelfde wijze kunnen rotatie, schaalverandering en shearing weergegeven worden:

$$\mathbf{p}' = \mathbf{R}(\theta) \cdot \mathbf{p} \qquad \text{met} \qquad \mathbf{R}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{p}' = \mathbf{S}(s_x, s_y) \cdot \mathbf{p} \qquad \text{met} \qquad \mathbf{S}(s_x, s_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{p}' = \mathbf{H}_x(a) \cdot \mathbf{p} \qquad \text{met} \qquad \mathbf{H}_x(a) = \begin{bmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{p}' = \mathbf{H}_y(b) \cdot \mathbf{p} \qquad \text{met} \qquad \mathbf{H}_y(b) = \begin{bmatrix} 1 & 0 & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

6.3 Concatenatie van transformaties

Door het invoeren van homogene coördinaten is het **concateneren** van verschillende affine transformaties nu teruggebracht tot het **vermenigvuldigen** van matrices.

$\Longrightarrow$ *Merk op dat bij matrixvermenigvuldiging* $\mathbf{A} \cdot \mathbf{B} = \mathbf{B} \cdot \mathbf{A}$ *niet altijd geldt! De volgorde waarin de vermenigvuldigingen uitgevoerd worden is dus erg belangrijk.*

$\Longrightarrow$ *Merk ook op dat de volgorde van uitvoering van de vermenigvuldiging van rechts naar links is. Dit kan verholpen worden door getransponeerde matrices en vectoren te gebruiken, met als resultaat dat er van links naar rechts vermenigvuldigd wordt:*

$$\mathbf{p'}^T = \mathbf{p}^T \cdot \mathbf{A}^T \cdot \mathbf{B}^T \cdot \mathbf{C}^T = \mathbf{p}^T \cdot \mathbf{M}^T$$

Voor de vier basisafbeeldingen translatie, rotatie, schaalverandering en shearing is het altijd mogelijk een **inverse matrix** op te stellen (in het geval van schaalverandering met de beperking dat de schaalfactor niet gelijk aan nul mag zijn). De inverse transformatiematrices van de genoemde afbeeldingen zijn:

$$\begin{aligned}
\mathbf{T}^{-1}(d_x, d_y) &= \mathbf{T}(-d_x, -d_y) \\
\mathbf{R}^{-1}(\theta) &= \mathbf{R}(-\theta) \\
\mathbf{S}^{-1}(s_x, s_y) &= \mathbf{S}(\frac{1}{s_x}, \frac{1}{s_y}) \\
\mathbf{H_x}^{-1}(a) &= \mathbf{H_x}(-a) \\
\mathbf{H_y}^{-1}(b) &= \mathbf{H_y}(-b)
\end{aligned}$$

6.4 Voorbeeld van concatenatie

Als voorbeeld van het uitvoeren van een complexe transformatie, opgebouwd uit een aantal "basis"-transformaties, wordt hier de rotatie over hoek θ rond een punt P buiten de oorsprong O genomen.

Om deze rotatie uit te voeren kunnen bijvoorbeeld de volgende transformaties gebruikt worden:
- transleer het rotatiepunt P naar de oorsprong (zie figuur 6.1: van 6.1A naar 6.1B),
- roteer over hoek θ rond de oorsprong O (zie figuur 6.1: van 6.1B naar 6.1C),
- transleer van de oorsprong naar het rotatiepunt P (zie figuur 6.1: van 6.1C naar 6.1D).

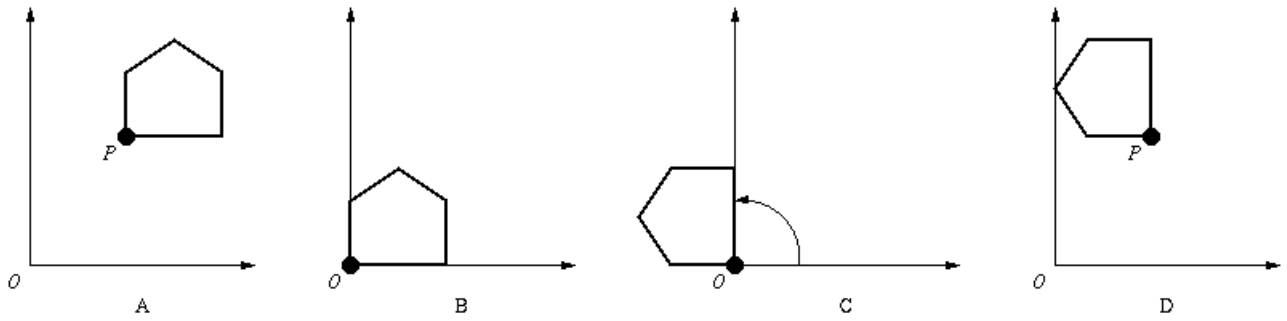


Figure 6.1: *Rotatie rond een punt buiten de oorsprong.*

Als P de coördinaten (x_1, y_1) heeft, dan levert de bovengenoemde concatenatie de volgende transformatie $\mathbf{M}$ op die een punt $\mathbf{p}$ in een punt $\mathbf{p'}$ over doet gaan:

$$\mathbf{p'} = \mathbf{T^{-1}} \cdot (\mathbf{R} \cdot (\mathbf{T} \cdot \mathbf{p}))$$

ofwel

$$\mathbf{p'} = \mathbf{M} \cdot \mathbf{p}$$

waarbij

$$\mathbf{M} = \mathbf{T^{-1}} \cdot \mathbf{R} \cdot \mathbf{T}$$

De totale transformatiematrix $\mathbf{M}$ wordt dan:

$$\begin{aligned}
\mathbf{M} &= \mathbf{T}(x_1, y_1) \cdot \mathbf{R}(\theta) \cdot \mathbf{T}(-x_1, -y_1) \\
&= \begin{bmatrix} 1 & 0 & x_1 \\ 0 & 1 & y_1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_1 \\ 0 & 1 & -y_1 \\ 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} \cos\theta & -\sin\theta & x_1(1-\cos\theta) + y_1 \sin\theta \\ \sin\theta & \cos\theta & y_1(1-\cos\theta) - x_1 \sin\theta \\ 0 & 0 & 1 \end{bmatrix}
\end{aligned}$$

Zie [1], hoofdstuk 5.

Hoofdstuk 7

3D transformaties

7.1 2D versus 3D

Er zijn twee belangrijke verschillen tussen 2D en 3D graphics. Bij het afbeelden van een 3D afbeelding op een 2D uitvoerapparaat moet er een extra stap, de **projectie** van 3D op 2D, uitgevoerd worden. Verder is 3D graphics door de extra dimensie veel **rekenintensiever**, waardoor de **efficiëntie van algoritmen** nog belangrijker wordt.

Bij het werken in een driedimensionale ruimte kan de z-as twee verschillende oriëntaties ten opzichte van het xy-assenstelsel hebben, namelijk als **rechts-** of als **linkshandig coördinaatsysteem**. In grafische systemen worden beide gebruikt, soms binnen hetzelfde systeem. Daarnaast moet vastgelegd worden in welke richting de **positieve rotatie** om de verschillende assen loopt.

Krommen kunnen in 3D alleen in **parametrische vorm** weergegeven worden, **vlakken** in **expiciete** en in **impliciete vorm**. De **normaal van een vlak** (een vector loodrecht op het vlak met lengte gelijk aan de eenheidsvector) is erg belangrijk bij het efficiënt uitvoeren van bewerkingen op vlakken (zoals hidden surface removal). Door een vlak te beschrijven met

$$ax + by + cz + d = 0$$

waarbij als voorwaarde

$$a^2 + b^2 + c^2 = 1$$

gesteld kan worden, is $\mathbf{n} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$ de normaal op het vlak en is d de afstand van het vlak tot de oorsprong.

Om van een tweedimensionale versie een uitbreiding naar 3D te maken, is er in het geval van een polylijn alleen een derde coördinaat nodig. Een structuur van punten die elk drie coördinaten bevatten, kan gebruikt worden in een functie om driedimensionale polylijnen af te beelden. Het projecteren van 3D objecten op een 2D scherm wordt dan, transparant voor de gebruiker, door deze functie gedaan.
Voor de text en fill area primitieven is het nodig om naast een 3D positie ook een oriëntatie in de 3D ruimte als argument mee te geven aan functies die deze primitieven afbeelden.

7.2 3D transformaties

Affine transformaties in 3D worden, analoog aan affine transformaties in 2D, in **homogene**, in dit geval vierdimensionale vorm weergegeven. Uitgaande van een punt $\mathbf{p} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$ dat door een affine transformatie overgaat in een punt $\mathbf{p}' = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}$, kunnen er transformatiematrices opgesteld worden voor translatie, rotatie,

schaalverandering en shearing (voor de x-as in y- en z-richting en voor de x- en y-as in de z-richting):

$$\mathbf{p}' = \mathbf{T}(d_x, d_y, d_z) \cdot \mathbf{p} \qquad \text{met} \qquad \mathbf{T}(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{p}' = \mathbf{R}_z(\theta) \cdot \mathbf{p} \qquad \text{met} \qquad \mathbf{R}_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{p}' = \mathbf{R}_x(\theta) \cdot \mathbf{p} \qquad \text{met} \qquad \mathbf{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{p}' = \mathbf{R}_y(\theta) \cdot \mathbf{p} \qquad \text{met} \qquad \mathbf{R}_y(\theta) = \begin{bmatrix} cos\theta & 0 & -sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{p}' = \mathbf{S}(s_x, s_y, s_z) \cdot \mathbf{p} \qquad \text{met} \qquad \mathbf{S}(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{p}' = \mathbf{H}_x(a, b) \cdot \mathbf{p} \qquad \text{met} \qquad \mathbf{H}_x(a, b) = \begin{bmatrix} 1 & a & b & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{p}' = \mathbf{H}_{xy}(b, d) \cdot \mathbf{p} \qquad \text{met} \qquad \mathbf{H}_{xy}(b, d) = \begin{bmatrix} 1 & 0 & b & 0 \\ 0 & 1 & d & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Merk op dat er verschillende mogelijkheden voor shearing zijn. Er kan shearing langs één as uitgevoerd worden (voor één of voor beide andere assen) en er kan shearing uitgevoerd worden langs twee assen (voor de derde as).

Zie [1], hoofdstuk 5.

7.3 Boomstructuren

Uitgaande van een aantal voorgedefinieerde grafische objecten of **symbolen**, kunnen er door samenvoegen van deze symbolen **complexe figuren** opgebouwd worden, waarbij op elk te gebruiken symbool een aantal transformaties uitgevoerd wordt om het symbool de gewenste grootte, oriëntatie en positie te geven. De transformatiematrix of de variabelen die de verschillende op het symbool uit te voeren transformaties beschrijven, kunnen dan bijvoorbeeld als parameters aan de symboolgenererende functie toegevoegd worden.
Op deze manier wordt een figuur in de praktijk opgebouwd door voor elk symbool de symboolgenererende functie met de juiste parameters aan te roepen, waardoor **absolute** grootte, oriëntatie en positie bepaald worden.

Figuren die opgebouwd zijn uit symbolen waartussen relaties bestaan die in de vorm van "**relatieve**" transformaties kunnen worden beschreven, kunnen op de volgende wijze beschreven en afgebeeld worden.
De gehele figuur, ofwel de relaties tussen de symbolen en de beschrijving van de symbolen, kan op **hiërarchische wijze** in de knooppunten of **nodes** van een **boomstructuur** vastgelegd worden, waarbij de verschillende nodes op de wijze van een **linked list** met elkaar verbonden worden. Door deze boomstructuur na te lopen, worden de benodigde transformatiematrices voor de symbolen opgesteld en vervolgens gebruikt om de volledige figuur op de juiste wijze en plaats af te beelden.
Door de parameters die de relaties tussen de symbolen beschrijven stapsgewijs te veranderen, kan een serie afbeeldingen gegenereerd worden. En van hier is het niet ver meer naar het weergeven van bewegende beelden.

In dit geval wordt een figuur in de praktijk opgebouwd door het aanroepen van een functie die bijvoorbeeld op **recursieve wijze** de boomstructuur naloopt en voor elke node de benodigde transformatiematrix opstelt en daarmee het symbool afbeeldt.

Bij een andere uitvoering van de boomstructuur (de **left child – right sibling** structuur) wordt per ouder-node maar één link naar een kind-node gelegd. Zijn er meerdere kinderen, dan worden deze onderling door middel van kind-naar-kind links gekoppeld. Per node zijn er dus maar twee links, één voor een kind en één voor een broer/zuster. Voordeel van deze structuur is, dat de datastructuur die gebruikt wordt om de inhoud van een node weer te geven, altijd eenzelfde aantal elementen heeft.
In dit geval wordt een figuur in de praktijk op dezelfde wijze als bij een "normale" boomstructuur opgebouwd.

$\Longrightarrow$ *Merk op dat het aflopen van een boomstructuur op verschillende wijzen gedaan kan worden, waardoor er in bepaalde gevallen geheel verschillende afbeeldingen kunnen ontstaan.*

Een andere wijze om grafische gegevens op te slaan, is met behulp van een bepaald type graaf, de **directed acyclic graphs** of **gerichte a-cyclische grafen**. Hierbij wordt er een structuur van elementen opgebouwd waarbij geen loops zijn toegestaan, maar waarbij het wel mogelijk is dat bepaalde elementen meermalen vanuit andere elementen gerefereerd worden. In een element worden transformaties, attributen en beschrijving van een object of symbool aangegeven. Ook worden referenties naar andere elementen — voorafgegaan door de relatieve transformaties daarvan ten opzichte van het aanroepende element — aangegeven.
Aangezien in dit geval dezelfde technieken gebruikt kunnen worden als bij de eerder genoemde boomstructuren, kan het afbeelden in de praktijk op dezelfde wijze geschieden.

Bij het grafische systeem **PHIGS** wordt van de laatste methode gebruik gemaakt. Er kunnen structuren gedefinieerd worden, die in een grafische database (de **Central Structure Store** of **CSS**) opgeslagen worden. Met een speciaal commando kunnen structuren uit de CSS op een werkstation actief gemaakt worden (d.w.z. afgebeeld worden), door ze te "posten" op het betreffende werkstation. Tijdens het afbeelden is het mogelijk om de inhoud van structuren in de CSS te veranderen (te "editten"), waardoor er veel dynamischer dan in **GKS** gewerkt kan worden.

Zie [1], hoofdstuk 7.

7.4 Toepassing

Om, uitgaande van een eenheidskubus met hoekpunt in de oorsprong als symbool, een willekeurig parallelepipedum in de driedimensionale ruimte als afbeelding (of **instance**) te maken, moet er geschaald, geroteerd en getransleerd worden. De verschillende stappen die voor dit afbeelden (of **instancing**) in het algemeen nodig zijn, worden hierna besproken. Als $\mathbf{I}$ de transformatiematrix voor de afbeelding is, dan bestaat deze dus uit

$$\mathbf{I} = \mathbf{T} \cdot \mathbf{R} \cdot \mathbf{S}$$

waarbij de matrices $\mathbf{T}$ de translatie, $\mathbf{R}$ de rotatie en $\mathbf{S}$ de schaalverandering aangeven. Hiervan bestaat $\mathbf{R}$ uit drie componenten, namelijk de rotaties om de verschillende hoofdassen in een vastgestelde volgorde

$$\mathbf{R} = \mathbf{R}_x \cdot \mathbf{R}_y \cdot \mathbf{R}_z$$

Deze rotaties kunnen als volgt beschreven worden: eerst een rotatie om de z-as, daarna een rotatie om de y-as en tenslotte een rotatie om de x-as waarna het object de gewenste oriëntatie heeft.

In de praktijk wordt er echter meestal niet van het symbool maar van de afbeelding uitgegaan, waarbij de oriëntatie van de afbeelding gegeven wordt door een richtingsvector $\mathbf{v}$ en een rotatie θ om de as die door deze richtingsvector gegeven wordt. Door de rotatie eerst in omgekeerde volgorde "aan te vatten", bestaat de rotatiematrix uit

$$\mathbf{R}' = \mathbf{R}'_z \cdot \mathbf{R}'_y \cdot \mathbf{R}'_x$$

waarbij dan geldt

$$\mathbf{R} = \mathbf{R}'^{-1} = \mathbf{R}'^{-1}_x \cdot \mathbf{R}'^{-1}_y \cdot \mathbf{R}'^{-1}_z = \mathbf{R}_x \cdot \mathbf{R}_y \cdot \mathbf{R}_z$$

met

$$\mathbf{R}_x^{-1}(\theta) = \mathbf{R}_x(-\theta) = \mathbf{R}_x^T(\theta)$$

Als de lengte van de richtingsvector $\mathbf{v}$ gelijk is aan de eenheidsvector, dan geldt

$$\mathbf{v} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

met

$$a^2 + b^2 + c^2 = 1$$

Zijn in dit geval α, β en γ de hoeken die $\mathbf{v}$ met de x-, y- en z-as maakt, dan geldt bovendien dat

$$\begin{aligned} cos(\alpha) &= a \\ cos(\beta) &= b \\ cos(\gamma) &= c \end{aligned}$$

De waarden a, b en c worden de **direction cosines** genoemd. Zijn twee van de drie direction cosines van $\mathbf{v}$ en de hoek θ bekend, dan kunnen de transformatiematrices (in omgekeerde volgorde) opgesteld worden. Is ϕ de rotatiehoek om de x-as en is ψ de rotatiehoek om de y-as, dan zijn de bijbehorende rotatiematrices:

$$\begin{aligned} \mathbf{R'}_z(\theta) &= \begin{bmatrix} cos\theta & -sin\theta & 0 & 0 \\ sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ \mathbf{R'}_x(\phi) &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\phi & -sin\phi & 0 \\ 0 & sin\phi & cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ \mathbf{R'}_y(\psi) &= \begin{bmatrix} cos\psi & 0 & -sin\psi & 0 \\ 0 & 1 & 0 & 0 \\ sin\psi & 0 & cos\psi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

Tenslotte geldt dan dat

$$\mathbf{R} = \mathbf{R'}_z(-\phi) \cdot \mathbf{R'}_y(-\psi) \cdot \mathbf{R'}_x(-\theta)$$

Uitgaande van het hiervoor afgeleide, kan een rotatie van een object om een **willekeurige** as als volgt beschreven worden

$$\mathbf{R} = \mathbf{T}^{-1} \cdot \mathbf{R}_x^{-1} \cdot \mathbf{R}_y^{-1} \cdot \mathbf{R}_z \cdot \mathbf{R}_y \cdot \mathbf{R}_x \cdot \mathbf{T}$$

ofwel: translatie naar oorsprong, rotatie naar xz-vlak, rotatie naar z-as, de gewenste rotatie (nu om de z-as), rotatie terug naar xy-vlak, rotatie terug naar oorspronkelijke richting van rotatie-as, translatie terug naar oorspronkelijke positie.

Zie [1], hoofdstuk 5.

Hoofdstuk 8

3D viewing

8.1 Synthetische camera

Bij het afbeelden van **driedimensionale beelden** is er een grote overeenkomst tussen het moderne computer graphics en een camera of een menselijk oog. Bij de laatste wordt de afbeelding achter het projectiecentrum (de lens) op het netvlies gemaakt. Maar de afbeelding kan ook op een vlek tussen het beeld en het projectiecentrum gemaakt worden, er wordt als het ware door een **venster** of **window** gekeken, waarbij de afbeelding in het venstervlak gemaakt wordt. Door het venster — met het beeld- of projectievlak — te verschuiven, wordt een ander deel van de ruimte afgebeeld. Merk op dat beelden in die ruimte ook kunnen bewegen en dat het projectiecentrum ook in de ruimte kan bewegen. Op deze wijze wordt er als het ware door een camera gekeken: de analogie van de **synthetische camera**. Het kijken door een venster dat verplaatst kan worden, bepaalt wat er gezien en dus afgebeeld wordt.

Met de synthetische camera valt het probleem van afbeelden uiteen in twee delen:

- de specificatie van **afmeting, positie en oriëntatie van objecten** in de ruimte,
- de specificatie van **viewing informatie** (vanuit welke positie, in welke richting en door welk venster wordt er naar de objecten gekeken).

Vóór het verschijnen van grafische systemen moesten de bewerkingen, die zorgen voor een juiste afbeelding (de **viewing informatie**), door de programmeur zelf gemaakt worden. Door deze bewerkingen te formaliseren tot "standaard"-bewerkingen en en deze in een aantal procedures of functies van een grafisch systeem te verpakken, is het voor een gebruiker niet meer nodig dit zelf te doen.

8.2 Grafische pijplijn

De weg die een grafisch object aflegt langs de verschillende transformaties van het grafische systeem om uiteindelijk op het gewenste uitvoerapparaat afgebeeld te worden, wordt de "**grafische pijplijn**" genoemd.

De transformaties die in **PHIGS** op de coördinaten van een 3D grafisch object uitgevoerd worden om tot een afbeelding te komen, vormen, samen met clipping operaties, de volgende **3D pijplijn**:

- van **MC** naar **WC**
  Het door de gebruiker in **modeling coordinates** opgestelde object wordt door **globale** en **lokale transformaties** omgezet naar **world coordinates**.
  Het in **WC** gedefinieerde object wordt gecontroleerd op het overschrijden van de viewportgrenzen en wordt indien nodig op de grens van het viewport als het ware afgeknipt. Dit proces wordt **clipping** genoemd.
- van **WC** naar **VRC**
  Het in **WC** gedefinieerde object (of wat daarvan over is na clipping) wordt met behulp van de gebruikte **view orientation transformatie** (die door **view reference point**, normaal op het **view plane** en **up vector** bepaald wordt) afgebeeld in **view reference coordinates**.
- van **VRC** naar **NPC**
  Door een **view mapping** of **projection transformatie** wordt het object in **VRC** op **normalized pro-**

jection coordinates afgebeeld, waarbij ook weer **clipping** uitgevoerd wordt.
- van **NPC** naar **DC**
  Tenslotte wordt het object via een **workstation transformatie** op **device coordinates** afgebeeld.
  Omdat een uitvoerapparaat heel specifiek kan zijn, is er meestal voor elk uitvoerapparaat een speciaal programmadeel dat het weergeven van informatie op dat uitvoerapparaat controleert en uitvoert, de **device driver**.
  Bij veel uitvoerapparatuur wordt gebruik gemaakt van een **frame buffer** (zie hoofdstuk 9), waarin de grafische objecten in de vorm van pixelinformatie opgeslagen worden. Het omzetten van lijnen in pixels voor een frame buffer wordt **scan conversion** genoemd.

Het projectievlak waarop een 3D object afgebeeld wordt, is een deel van het view plane. Dit laatste kan bepaald worden door een **normaalvector** n op dat vlak en door een punt in dat vlak, het **view reference point**. Door middel van een window in het view plane wordt het projectievlak aangegeven. Aangezien het projectievlak een willekeurige oriëntatie kan hebben, is het efficiënter om alle coördinaten om te zetten naar een coördinaatsysteem dat op het projectievlak gebaseerd is, het **u,v,n-systeem**. Hiervoor zorgt de **viewing transformatie**.
Het u, v, n-systeem wordt bepaald door de normaal op het viewing plane (de n-as), door het view reference point (de oorsprong) en door de projectie op het viewing plane van een vector die niet in het viewing plane ligt (de u-as). De v-as ligt hiermee vast als de orientatie van de assen ten opzichte van elkaar vastgesteld is.

Clipping in 3D wordt uitgevoerd voor een volume dat aangegeven wordt door onder- en bovengrenscoördinaten voor u, v en n. In de u- en v-richtingen wordt dit volume begrensd door vier vlakken die elkaar snijden in vier projectoren en in de n-richting door twee vlakken evenwijdig aan het projectievlak.

8.3 Projecties

Er zijn verschillende mogelijkheden om te projecteren. De lijnen waarlangs geprojecteerd wordt, worden **projectors** of **projectoren** genoemd. Als de projectoren evenwijdig aan elkaar lopen, is er sprake van **parallelle projectie**, als de projectoren samenkomen in een in de eindige 3D ruimte gelegen **projection reference point** (**center of projection** of **verdwijnpunt**), dan spreken we van **perspectiefprojectie**.

Afhankelijk van de richting van de projectoren kunnen er verschillende soorten projecties onderscheiden worden:
- Bij **orthografische** of **orthogonale projectie** lopen de projectoren evenwijdig aan elkaar en staan ze loodrecht op het projectievlak.
  - Worden er drie projecties gemaakt die voor-, zij- en bovenaanzicht weergeven, dan wordt dit **multi-view orthografic projection** genoemd.
  - Heeft het projectievlak een willekeurige oriëntatie ten opzichte van het object, dan wordt er van **axonometrische projectie** gesproken.
- Bij **oblique projection** lopen de projectoren evenwijdig aan elkaar, maar kunnen ze een willekeurige hoek met het projectievlak maken.
- Bij **perspectiefprojectie** kunnen er één, twee of drie verdwijnpunten zijn waar de projectoren op uitkomen.

Zie [1], hoofdstuk 6.

Hoofdstuk 9

Raster Graphics

9.1 Algemeen

In dit hoofdstuk worden de mogelijkheden en een aantal technieken voor weergave op **raster displays** behandeld. Vergeleken met **vector displays** hebben raster displays een aantal extra mogelijkheden:

- het vullen van vlakken,
- het maken van realistische afbeeldingen.

Daarnaast kan een bewerking als **hidden-surface removal** bijvoorbeeld veel efficiënter op rastersystemen uitgevoerd worden.

Een belangrijk onderdeel van een rastersysteem is de **frame buffer**. Deze bevat de informatie van de beeldpunten op het scherm, de "picture elements" of **pixels**. Een frame buffer kan beschouwd worden als een matrix of een rij van elementen (voor elk pixel een element), waarin de informatie opgeslagen is. Afhankelijk van de mogelijkheden (bijvoorbeeld monochroom of kleur), bevat een element één of meer bits. In het geval van meerdere bits kan de frame buffer beschouwd worden als opgebouwd zijnde uit een blok of uit meerdere vlakken (voor elk bit een vlak).

Een frame buffer kan gezien worden als een soort doorgeefluik: aan de ene kant is het grafische systeem of het gebruikersprogramma bezig met het aanpassen van de informatie (het opbouwen van afbeeldingen) en aan de andere kant is de display processor bezig de informatie op het scherm te zetten. Het doorgeefluik zorgt er in dit geval voor dat er een nette scheiding tussen deze functies is aangebracht.

De bewerkingen die op de pixels in een frame buffer uitgevoerd worden, kunnen verdeeld worden in het veranderen van informatie **onafhankelijk** van en **afhankelijk** van de reeds in de frame buffer opgeslagen informatie. In het eerste geval wordt de informatie vervangen, in het tweede geval wordt de informatie — afhankelijk van de in de pixels aanwezige informatie — al of niet veranderd.
Een bewerking die ook veel in de frame buffer gedaan wordt, is het **verwisselen** van de informatie. Het beeld dat in de frame buffer staat, moet in dit geval verwisseld worden met een beeld dat in een ander deel van het geheugen staat. Het verwisselen van de pixels kan heel efficiënt gedaan worden door driemaal een **XOR** (exclusive OR) operatie op de overeenkomstige pixels van de twee beelden uit te voeren.

De XOR operatie is ook te gebruiken bij het tijdelijk afbeelden van een figuur en het daarna weer verwijderen van die figuur, waarbij de oorspronkelijk toestand weer wordt hersteld. Dit wordt bijvoorbeeld toegepast bij het weergeven van een cursor en bij rubberbanding. Een andere toepassing waarbij van de XOR gebruik gemaakt kan worden, is het lijn voor lijn vullen van vlakken.

In het algemeen worden de genoemde operaties niet op één pixel, maar op een rij of blok met pixels uitgevoerd. Deze operaties worden **bit block transfers** of **BitBlt**'s genoemd. In het geval van tekens uit een bepaald font — die elk als een blok pixels gedefinieerd zijn — wordt van **character block transfer** of **CharBlt** gesproken.

Veelhoeken of "gevulde" polylijnen worden veel gebruikt in computer graphics. Er zijn drie belangrijke aspecten die besproken worden:

- **Weergave**.
  Een veelhoek wordt bepaald door een aantal hoekpunten die een vaste volgorde hebben. Om de gegevens

van een veelhoek vast te leggen, kan er bijvoorbeeld een datastructuur opgesteld worden die bestaat uit een linked list van alle hoekpunten van die veelhoek. Om een complex grafisch object, opgebouwd uit meerdere veelhoeken (een **netstructuur**), te beschrijven, wordt de datastructuur uitgebreid met een linked list van alle veelhoeken in de netstructuur.
Aangezien er een groot aantal hoekpunten meerdere malen in een op deze manier opgezette datastructuur voor zal komen, is het efficiënter om een lijst van hoekpunten te maken, en hier vanuit de datastructuur naar te verwijzen. Het veranderen van de coördinaten van een gemeenschappelijk hoekpunt hoeft dan maar op één plaats te gebeuren.
In een dergelijke datastructuur is de **topologie** of de structuur van het netwerk gescheiden van de **geometrie** of de positie van de hoekpunten.
Omdat er nu nog gemeenschappelijke zijden zijn in het netwerk, kunnen deze op hun beurt in een linked list gezet worden, waardoor veranderingen in de geometrie voor gemeenschappelijke zijden in één keer uitgevoerd zijn. Hiermee zijn echter nog niet alle problemen opgelost.

- **Clipping**.
  Hierbij is het begrip **convexe veelhoek** van belang. Voor convexe veelhoeken geldt, dat elke lijn tussen twee punten binnen die veelhoek, ook geheel binnen die veelhoek ligt. Door uit te gaan van objecten bestaande uit convexe veelhoeken, zijn de algoritmen om dergelijke objecten te clippen veel eenvoudiger. **Reentrant clipping** is hierbij ook mogelijk.
- **Scan conversie**.
  Het tekenen van een veelhoek in de frame buffer bestaat uit het genereren van de pixels binnen het door de hoekpunten beschreven vlak: ligt een pixel binnen de veelhoek, dan moet het pixel met de kleur van de veelhoek gevuld worden.
  Voor het opvullen van veelhoeken zijn er verschillende algoritmen. Bij het opvullen per scanlijn (lijn met pixels) wordt er bijvoorbeeld gebruik gemaakt van het feit dat er, langs de scanlijn gaande, na een oneven aantal grenspixels gevuld moet worden (een grenspixel ligt op een grens of zijde van de veelhoek), terwijl er na een even aantal niet gevuld moet worden. Een probleem hierbij wordt gevormd door eindpunten die precies op een grenspixel liggen. Dit probleem kan verholpen worden door de coördinaten van de eindpunten altijd **tussen** de coördinaten van de pixels te laten vallen.
  Een ander probleem wordt gevormd doordat er verschillende veelhoeken over elkaar heen vallen. Hiervoor zijn verschillende oplossingen mogelijk. Eén daarvan maakt gebruik van een methode analoog aan schilderen (het **painter's algorithm**), namelijk door de vlakken die onderop liggen het eerst te vullen. Deze worden dan later overschreven door de erop liggende vlakken (de vlakken met hogere prioriteit).
  Een geheel andere, algemene methode om vlakken te vullen, de **flood fill** methode, begint met een punt binnen de veelhoek en vult het vlak op rekursieve wijze door de punten rond dat eerste punt te bekijken. Vallen ze binnen de veelhoek en zijn ze nog niet gevuld, dan worden ze gevuld en vervolgens als nieuw eerste punt genomen. Nadeel van deze methode is het extra werk dat gedaan wordt doordat veel al gevulde pixels nog een aantal malen onnodig bekeken worden.

Zie [1], hoofdstuk 3.

9.2 Praktische implementatie

Het tweemaal voorkomen van **clipping** in de grafische pijplijn zorgt voor het verwijderen van die objecten of objectdelen die buiten het viewport vallen en dus verder niet meegenomen hoeven te worden in berekeningen bij volgende delen van de pijplijn. Bovendien wordt voorkomen dat er vreemde effecten bij het afbeelden ontstaan, die veroorzaakt worden door objectdelen die buiten het viewport uitsteken. Clipping wordt in het eerste geval (in WC) in software gedaan en in het tweede geval (in DC) meestal in hardware. Hierbij kunnen dezelfde clipping-algoritmen gebruikt worden.

Omdat er veel van clipping gebruik gemaakt wordt, is het belangrijk om **snel** en **efficiënt** werkende **algoritmen** te gebruiken. Afhankelijk van het soort object worden er dan ook verschillende algoritmen gebruikt. Zo wordt er onderscheid gemaakt tussen rechte lijnen, krommen en tekst. Bij tekst wordt er onderscheid gemaakt tussen het clippen van gehele woorden (**string precision**), van letters (**character precision**) of van letterdelen (**stroke precision**).

Bij het clippen van **rechte lijnen** moet onderscheid gemaakt worden tussen de gevallen waarin het lijnsegment geheel binnen, geheel buiten of gedeeltelijk binnen het viewport valt. Een efficiënt clipping-algoritme voor rechte lijnen is het **Cohen-Sutherland algoritme**. Om nog sneller te clippen kan er gebruik gemaakt worden van het

feit dat clippen langs de vier zijden van een viewport, afgezien van oriëntatie, hetzelfde is: **reentrant clipping**, ofwel het parallel uitvoeren van delen van het clipping algoritme in hardware.

Device drivers zorgen voor de aanpassing van een grafisch systeem aan een specifiek uitvoerapparaat. Afhankelijk van de hardware-mogelijkheden van dat apparaat (bijvoorbeeld hardware clipping, hardware generatie van tekens) moet de device driver meer of minder uitvoeren. Naast uitvoerapparatuur van een bepaald merk zijn er echter ook algemeen opgezette "talen" waarmee apparatuur van verschillende merken aangestuurd kunnen worden.

Bij het gebruik van een **frame buffer** moet elk object omgezet worden in een aantal pixels. Hiervoor zijn weer verschillende algoritmen ontwikkeld, waardoor dat omzetten snel en efficiënt gedaan wordt. Eén van deze algoritmen voor het omzetten van rechte lijnen is het **algoritme van Bresenham**.

In een aantal gevallen, bijvoorbeeld als er maar één afbeelding gemaakt hoeft te worden, is de snelheid waarmee uitvoer op een grafisch device gegenereerd wordt niet enorm belangrijk. Maar veel toepassingen genereren niet één plaatje, maar een continue serie plaatjes, bijvoorbeeld om een beweging weer te geven. En deze beweging moet het liefst met natuurlijke snelheid, ofwel in **real time** uitgevoerd worden. Om dit te kunnen doen, zijn er computers met een hierop toegesneden computerarchitectuur ontwikkeld. Deze grafische werkstations kunnen, onder andere met behulp van speciale processoren (**display processor** en **display controller**), de gewenste snelheid halen. Bovendien zullen zoveel mogelijk standaardbewerkingen — eventueel parallel — in hardware uitgevoerd worden.

Zie [1], hoofdstuk 3.

9.3 Kleur

De gevoeligheid van het menselijk oog voor de frequenties van het zichtbare licht is afhankelijk van die frequenties. Een standaardverdeling hiervoor, het **CIE-model**, is opgesteld door de **Commission Internationale de L'Eclairage** (**C.I.E.**) en wordt de **Standard Observer Curve** genoemd. Bij het tonen van kleur (op een beeldscherm bijvoorbeeld) kan de intensiteit van de frequenties waarvoor het menselijk oog minder gevoelig is, verhoogd worden, waardoor bereikt wordt dat de helderheid van verschillende kleuren voor het oog ongeveer gelijk is.

Kleuren kunnen samengesteld worden uit een combinatie van de drie hoofdkleuren rood, groen en blauw. Er kunnen gevoeligheidskrommen $R(\lambda)$, $G(\lambda)$ en $B(\lambda)$ voor de hoofdkleuren opgesteld worden, die dan bijvoorbeeld specifiek zijn voor het menselijk oog. Voor een bepaald soort kleurenfilm zullen deze gevoeligheidskrommen er weer anders uitzien. Uitgaande van de intensiteiten die gegeven worden door een functie $C(\lambda)$, kunnen er drie **tristimulus**-waarden opgesteld worden:

$$T_1 = \int C(\lambda)R(\lambda)d\lambda$$

$$T_2 = \int C(\lambda)G(\lambda)d\lambda$$

$$T_3 = \int C(\lambda)B(\lambda)d\lambda$$

Hiervoor geldt in de praktijk dat deze integralen een positieve uitkomst leveren. Wordt het maximum voor elk van de tristimuli op 1 genormaliseerd, dan kan een kleur $\mathbf{C}$ weergegeven worden door een punt in de driedimensionale kleurruimte, dat binnen een eenheidskubus (de **color solid**) ligt

$$\mathbf{C} = (T_1, T_2, T_3)$$

Worden er drie eenheidsvectoren $\mathbf{R}$, $\mathbf{G}$ en $\mathbf{B}$ voor de drie hoofdkleuren gebruikt (het **RGB-model**), dan kan een kleur voorgesteld worden als een driedimensionale vector

$$\mathbf{C} = T_1\mathbf{R} + T_2\mathbf{G} + T_3\mathbf{B}$$

Alle kleuren waarvoor geldt

$$T_1 + T_2 + T_3 = 1$$

liggen in een driehoek die de uiteinden van de drie eenheidsvectoren als hoekpunten heeft, de **driehoek van Maxwell**.

Door de verschillende hoofdkleuren in verschillende verhoudingen bij elkaar op te tellen, krijgen we **additieve kleuren**. Door een witte lichtbron met filters, die de complementaire hoofdkleuren (cyaan, magenta en geel) geheel of gedeeltelijk tegenhouden, af te schermen krijgen we **subtractieve kleuren**. De eerste techniek wordt bij beeldschermen toegepast.

Verschillende kleursensoren hebben verschillende gevoeligheid voor de hoofdkleuren. Voor een bepaalde kleur zal voor het menselijk oog (met (RGB)-systeem) gelden dat

$$\mathbf{C} = T_1\mathbf{R} + T_2\mathbf{G} + T_3\mathbf{B}$$

en voor een beeldscherm (met (R', G', B')-systeem) zal voor diezelfde kleur gelden dat

$$\mathbf{C} = T_1'\mathbf{R}' + T_2'\mathbf{G}' + T_3'\mathbf{B}'$$

Nu kan elk van de drie hoofdkleurcomponenten van het ene systeem uitgedrukt worden in het andere systeem, bijvoorbeeld

$$\mathbf{R}' = r_r\mathbf{R} + r_g\mathbf{G} + r_b\mathbf{B}$$

waarbij (r_r, r_g, r_b) de tristimuluswaarden van het menselijke oog zijn. Wordt dit ook voor de andere hoofdkleuren gedaan, dan kunnen de verkregen vergelijkingen geschreven worden als

$$\begin{bmatrix} T_1' \\ T_2' \\ T_3' \end{bmatrix} = \begin{bmatrix} r_r & r_g & r_b \\ g_r & g_g & g_b \\ b_r & b_g & b_b \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \end{bmatrix} = \mathbf{M} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \end{bmatrix}$$

Door een matrix met de tristimuluswaarden als elementen kan er dus een transformatie aangegeven worden van het ene in het andere kleurgevoelige systeem.

Daarnaast is er nog het **HLS-model** met de begrippen kleurtint (**hue**), kleurhelderheid (**lightness**) en kleurverzadiging (**saturation**). Deze kunnen ook in een driedimensionale HLS-ruimte met behulp van poolcoördinaten weergegeven worden. De L is de lengte van een vector langs de as die diagonaal van de oorsprong naar het punt $(1, 1, 1)$ loopt, de S is de lengte van een vector in een vlak loodrecht op de L-as en de H geeft de hoek aan van de vector in dat vlak.

Uitgaande van meerdere bits per pixel kunnen de verschillende hoofdkleurintensiteiten weergegeven worden. Door bijvoorbeeld acht bits voor elke hoofdkleur te gebruiken, zijn er drie bytes per pixel nodig, waarmee ongeveer 16M kleuren gegenereerd kunnen worden. Dit kost veel geheugen en daarom wordt er gebruik gemaakt van kleurtabellen waarin een kleurpalet gedefinieerd wordt. Er is nu maar een beperkt aantal kleuren mogelijk ("maar" 1M bijvoorbeeld), maar dit zal in de praktijk nauwelijks een probleem zijn. Bijkomend voordeel is de mogelijkheid om het kleurpalet aan te passen zonder de beschrijving van de afbeeldingen te veranderen.

Met meerdere bits per pixel is het ook mogelijk om **anti-aliasing** toe te passen. Hierbij worden de door de diskrete pixels veroorzaakte abrupte overgangen — het zogenaamde **aliasing** effect, dat vooral in lijnen voorkomt die bijna evenwijdig aan x- of y-as lopen — wat minder abrupt gemaakt. Dit gebeurt door naastgelegen pixels die "gedeeltelijk" tot de lijn behoren maar door het algoritme buiten de lijn vielen, toch met de lijn mee te laten doen maar met een intensiteit die het tot het pixel behorende deel van de lijn weergeeft. De intensiteit van de pixels die wel tot de lijn behoren wordt op dezelfde wijze aangepast.

Zie [1], hoofdstuk 11.

Hoofdstuk 10

Afbeelden in 3D

10.1 Algemeen

Bij het weergeven van 3D afbeeldingen moet er een keus gemaakt worden voor de te gebruiken **primitieven**. In een eenvoudig grafisch systeem zullen polylijnen voldoen, maar bij echte 3D toepassingen, is het handiger om polygonen of zelfs 3D volumes (zoals een kubus) als primitieven te kunnen gebruiken.

Het weergeven van objecten wordt **rendering** genoemd. Als er alleen lijnen gebruikt worden om een 3D object weer te geven, dan wordt er van een **wire frame model** gesproken. Hierbij zijn in het eenvoudigste geval alle lijnen van een afgebeeld object zichtbaar, omdat het object geheel doorzichtig is. Door lijnen, die niet-zichtbare delen van een niet-doorzichtig object aangeven, weg te laten, wordt er een meer realistische afbeelding gemaakt. De techniek van het verwijderen van deze niet-zichtbare lijnen of vlakken wordt **hidden-line removal** of **hidden-surface removal** genoemd.

De volgende stap naar realistische afbeeldingen bestaat uit het weergeven van een **ondoorzichtig** object, door de vlakken, waaruit het object opgebouwd is, te vullen. Hierbij wordt bij het afbeelden ook weer **hidden-surface removal** toegepast. Met de term **solid modeling** wordt het weergeven van "massieve" objecten bedoeld.

Tenslotte kunnen er **eigenschappen** aan de verschillende vlakken toegekend worden, bijvoorbeeld om doffe of spiegelende oppervlakken te verkrijgen. Door het object vervolgens met één of meer **lichtbronnen** te verlichten worden er echt realistische afbeeldingen gemaakt. De technieken die gebruikt worden om dit laatste uit te voeren worden — afhankelijk van de gebruikte methode — met **ray tracing** en met **ray casting** aangeduid.

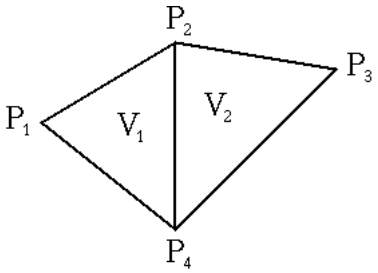
10.2 Netstructuren

Objecten kunnen eenvoudig beschreven worden door **polygonen**, die zelf weer beschreven kunnen worden door de **hoekpunten** en door de **normaal** van het vlak waarin het polygoon ligt. Hierbij wordt aangenomen dat alle hoekpunten van een polygoon in hetzelfde vlak liggen. Uitgaande van drie hoekpunten kan de normaal bij een bepaald polygoon bepaald worden. Het feit dat de normaal de oriëntatie van het polygoon bepaald, kan dan bijvoorbeeld gebruikt worden bij hidden surface removal.

De polygoonstructuren die voor het beschrijven van objecten worden gebruikt, worden **netstructuren** of **polygon meshes** genoemd. In de praktijk bestaat zo'n netstructuur bijvoorbeeld uit een aantal vlakken, die elk opgebouwd zijn uit een aantal zijden, die elk opgebouwd zijn uit twee eindpunten. Hiermee kunnen wire-frame images opgebouwd worden en kan er **hidden surface removal** uitgevoerd worden. Afhankelijk van het object zijn er **open** en **gesloten netstructuren**. Zo kan een bol bijvoorbeeld door een gesloten netstructuur weergegeven worden.

De handigste vorm voor een polygoon is een **driehoek**, onder andere omdat de drie hoekpunten altijd in één vlak liggen. De enige beperking is dat de punten niet op één rechte lijn mogen liggen. Vierhoeken in netstructuren worden gemakkelijk in driehoeken omgezet door elke vierhoek door een diagonaal in twee driehoeken te verdelen.

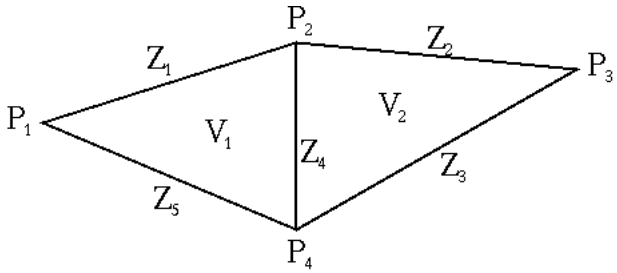
Een netstructuur kan op verschillende wijzen opgebouwd worden. Een veelvlak uit de structuur kan bijvoorbeeld opgeslagen worden als een pointerlijst van hoekpunten (zie figuur 10.1) of als een pointerlijst van zijden (zie figuur 10.2).



$V_1 = (P_1, P_2, P_4)$
$V_2 = (P_4, P_2, P_3)$

$P_1 = (x1, y1, z1)$
$P_2 = (x2, y2, z2)$
$P_3 = (x3, y3, z3)$
$P_4 = (x4, y4, z4)$

Figure 10.1: *Netstructuur met veelvlak als pointerlijst van hoekpunten.*



$V_1 = (Z_1, Z_4, Z_5)$
$V_2 = (Z_2, Z_3, Z_4)$

$Z_1 = (P_1, P_2, V_1, 0)$
$Z_2 = (P_2, P_3, V_2, 0)$
$Z_3 = (P_3, P_4, V_2, 0)$
$Z_4 = (P_2, P_4, V_1, V_2)$
$Z_5 = (P_1, P_4, V_1, 0)$

$P_1 = (x_1, y_1, z_1)$
$P_2 = (x_2, y_2, z_2)$
$P_3 = (x_3, y_3, z_3)$
$P_4 = (x_4, y_4, z_4)$

Figure 10.2: *Netstructuur met veelvlak als pointerlijst van zijden.*

Het op realistische wijze weergeven van objecten, opgebouwd met behulp van netstructuren bestaande uit rechte lijnen, wordt beperkt doordat een erg fijnmazige netstructuur (om niet-gewenste hoekige vormen te voorkomen) te veel rekentijd vergt. Een andere manier van weergeven gaat uit van het gebruik van een grove netstructuur opgebouwd uit **krommen** om de bollingen van een object op de gewenste wijze te benaderen.

10.3 Krommen

Voor driedimensionale krommen is de weergave in **parametrische vorm** in het algemeen de beste keuze:

$$\begin{aligned} x &= x(t) \\ y &= y(t) \\ z &= z(t) \end{aligned}$$

De **expliciete vorm**, aangegeven met

$$\begin{aligned} y &= f(x) \\ z &= g(y) \end{aligned}$$

geeft, net als in het tweedimensionale geval, problemen met het weergeven van een aantal rechten, terwijl de **impliciete vorm**

$$\begin{aligned} G(x,y,z) &= 0 \\ H(x,y,z) &= 0 \end{aligned}$$

niet goed bruikbaar is om de punten van een kromme voor afbeelding te genereren.

Krommen kunnen met behulp van **polynomen** beschreven worden. Door de graad daarvan niet al te hoog te nemen, wordt de benodigde rekentijd binnen de perken gehouden. In de praktijk worden polynomen van de derde graad, **cubic polynomials**, gebruikt.

Door vier controlepunten op een segment van de te berekenen kromme te geven, kunnen de coëfficienten van de polynomen die dat segment benaderen, berekend worden. Wordt de afstand tussen de controlepunten gelijk gehouden, dan kan een **interpolating geometry matrix** berekend worden, die voor elk segment van de kromme gelijk is. **Blending polynomials** geven de bijdrage van elk controlepunt voor de punten van het segment van de te maken kromme aan.

Een probleem is de aansluiting van de verschillende segmenten, die de benadering vormen, op elkaar. Een goede aansluiting eist dat de eerste en tweede afgeleiden van twee op elkaar aansluitende segmenten gelijk zijn. Met behulp van **Hermite polynomen** (zie figuur 10.3), **Bézier polynomen** (zie figuur 10.4) en **B-splines** (zie figuur 10.5) worden betere tot goede aansluitingen verkregen.

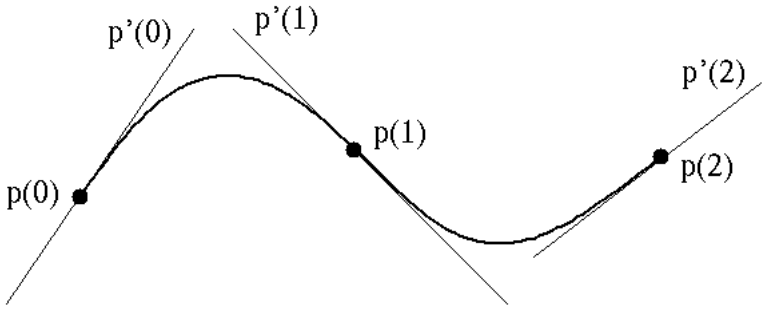


Figure 10.3: *Kromme bestaande uit twee Hermite segmenten. Per segment zijn er twee controlepunten en twee afgeleiden in de controlepunten gedefinieerd: P(0), P(1), P'(0) en P'(1) voor het eerste segment en P(1), P(2), P'(1), P'(2) voor het tweede.*

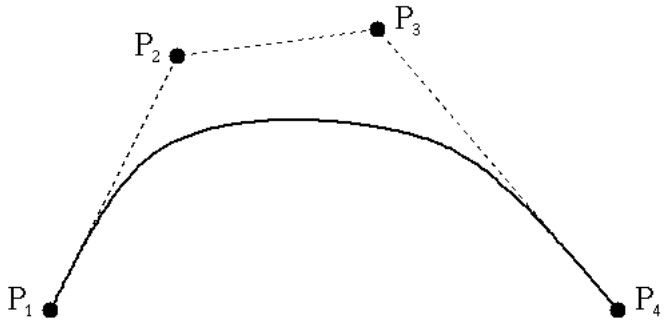


Figure 10.4: *Kromme gemaakt met een Bézier polynoom. Er zijn vier controlepunten per segment: twee controlepunten vormen begin- en eindpunt van het segment en met de andere twee controlepunten wordt een benadering van de afgeleiden in begin- en eindpunt gegeven.*

Na het bepalen van de coëfficiënten van de voor de benadering gebruikte segmenten, moet er nog **scanconversie** uitgevoerd worden om de pixelposities te berekenen. Veel methoden die hierbij gebruikt worden, zijn gebaseerd op

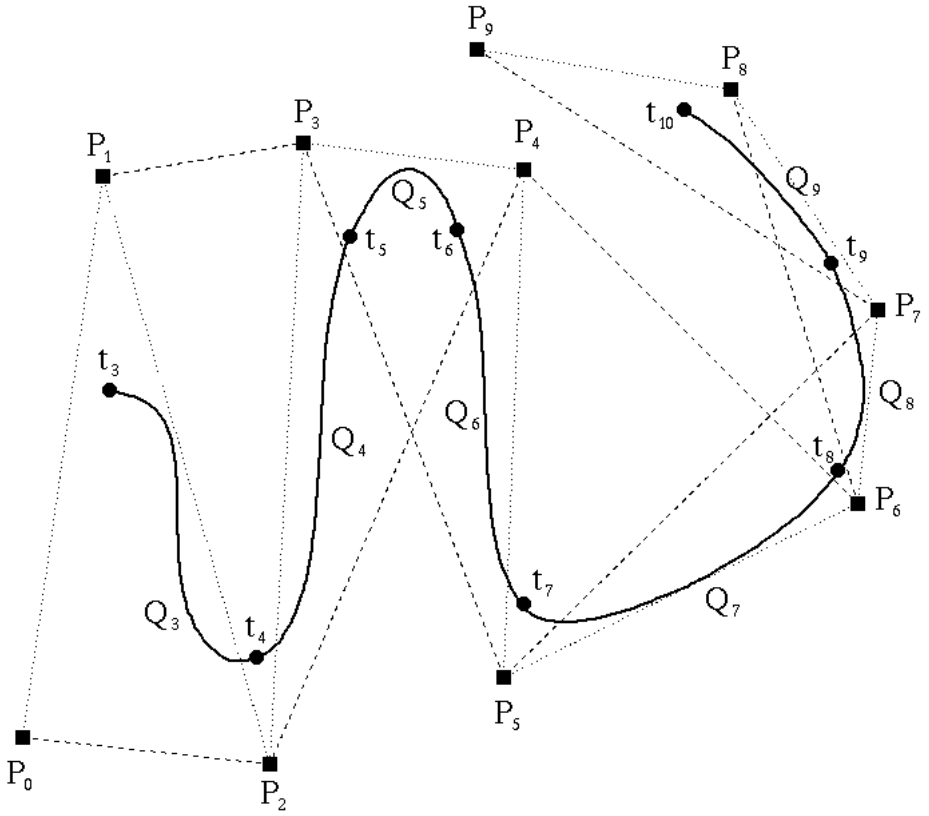


Figure 10.5: *Kromme bestaande uit een aantal B-spline segmenten. Als $M \geq 3$, zijn er M+1 controlepunten (P_0 t/m P_M), M-2 segmenten (Q_3 t/m Q_M) en M-1 knopen (t_3 t/m t_{M+1}).*

het gebruik van **forward differences**, de afwijkingen die optreden bij opeenvolgende punten. Hierdoor kunnen volgende punten berekend worden door drie optellingen uit te voeren.

10.4 Oppervlakken

Oppervlakken worden in **parametrische vorm** weergegeven door

$$x = x(s,t)$$
$$y = y(s,t)$$
$$z = z(s,t)$$

Voor een oppervlak beschreven door een netstructuur kunnen de mazen of **patches** afzonderlijk benaderd worden door **bicubic polynomials**

$$p(s,t) = \sum_{i=0}^{3}\sum_{j=0}^{3} c_{ij} s^i t^j$$

Bézier patches geven tamelijk goed op elkaar aansluitende mazen; **splines** geven een beter resultaat, maar kosten over het algemeen te veel rekentijd.

Om uiteindelijk een echt realistische afbeelding te maken, moeten er nog **hidden-surface removal** en **ray tracing** uitgevoerd worden.

Afhankelijk van de toepassing waarvoor de afbeelding van grafische objecten gebruikt wordt, zijn er verschillende soorten primitieven, soorten datastructuren en soorten bewerkingen nodig. Primitieven kunnen bijvoorbeeld opgebouwd worden door, uitgaande van een tweedimensionaal primitief, dit primitief langs een as in de derde

dimensie uit te breiden (bij een cirkel ontstaat zo een cylinder), of door een driedimensionale rotatie om een as uit te voeren (een cirkel kan zo in een bol of in een torus overgaan). Bij het construeren van voorwerpen met behulp van primitieven (**constructive solid geometry** of **CSG**), wordt er bijvoorbeeld van set-algebra gebruik gemaakt on voorwerpen te construeren.

Zie [1], hoofdstuk 9.

10.5 Hidden surface removal

Omdat algoritmen voor hidden-surface removal voor gebruik in frame buffers veel efficiënter zijn dan algoritmen voor hidden-line removal, wordt er hier alleen op de eerste dieper ingegaan. In het algemeen moet er bepaald worden wat de volgorde van de vlakken langs verschillende projectoren is. Door de snijpunten van de vlakken met een projector te sorteren, wordt het dichtstbijzijnde snijpunt gevonden. En dit bepaalt dan de kleur en/of de intensiteit van het bijbehorende pixel. Omdat er verschillende methoden zijn om niet-zichtbare (delen van) vlakken te vinden, zijn er meestal verschillende algoritmen beschikbaar. Van geval tot geval wordt dan bekeken welk het meest efficiënt is.

Er zijn twee globale wijzen waarop hidden-surface removal aangepakt kan worden; deze worden met **object-space** en met **image-space** aangeduid:

- Bij **object-space** methoden is er een eenvoudige relatie (wat betreft positie ten opzichte van elkaar) tussen de verschillende objecten. Voorbeelden hiervan zijn algoritmen waarbij de oriëntatie van de normaal van de vlakken van een object gebruikt wordt en **depth sort algoritmen**.
- Bij **image-space** methoden wordt de afbeelding via projectoren op het projectievlak gebruikt. Voor elke projector wordt bepaald wat het dichtstbijzijnde vlak is dat doorkruist wordt. Voorbeelden hiervan zijn het **z-buffer algoritme** en het **scan-line algoritme**.
  Bij het z-buffer algoritme wordt gebruik gemaakt van een **z-buffer**, een buffer in matrixvorm, waarin per pixel de afstand van view reference point tot een object bijgehouden kan worden.

10.6 Shading

Het gebruik van één of meer lichtbronnen die de verschillende afbeeldingen verlichten, kan op twee manieren gerealiseerd worden:

- Bij de eerste manier, **ray tracing**, worden de lichtstralen van een lichtbron nagelopen om te bepalen of deze, eventueel via contact met oppervlakken — waarbij licht gereflecteerd en geabsorbeerd kan worden — op het projectievlak terecht komen. Nadeel hierbij is, dat er veel lichtstralen voor niets nagelopen worden, omdat deze in andere richtingen verdwijnen.
- Bij de tweede manier, **ray casting**, worden alleen die lichtstralen die het projectievlak bereiken, nagelopen. Dit gebeurt in tegengestelde richting, dus beginnend bij het projectievlak. Voordeel hierbij is het beperkte aantal lichtstralen, bijvoorbeeld één per pixel. Nadeel is dat er kleine objecten gemist kunnen worden (dit wordt **aliasing** genoemd). De kans hierop kan verkleind worden door meerdere lichtstralen per pixel te nemen.

Bij het uitvoeren van ray casting zijn er verschillende mogelijkheden om meer of minder dicht bij de fysieke waarheid te blijven: het gebruik van **shadow rays**, het toelaten van **spiegelende oppervlakken** (waardoor er meerdere lichtbronnen ontstaan, die voor de nodige extra berekeningen zorgen), het negeren van **ondoorzichtige objecten** waar een gevolgde lichtstraal doorheen gaat.

Als toevoeging aan ray tracing of -casting kan de **verstrooiing** van het licht aan oppervlakken van objecten aangegeven worden: **volkomen reflectie**, **voorwaartse verstrooiing**, **verstrooiing in alle richtingen**, **absorptie**. Deze toevoeging zorgt echter weer voor het nodige rekenwerk!

Om veel rekenen te voorkomen, kan er, in plaats van één of meer lichtbronnen, van **omgevingslicht** of **ambient light** gebruik gemaakt worden. Hierbij wordt voor elke richting een vaste lichtintensiteit aangenomen, die bij contact met oppervlakken gebruikt wordt en daarbij, afhankelijk van de eigenschappen van het oppervlak, een lichtopbrengst geeft.

Voldoende spiegelende oppervlakken tonen bij de juiste invalshoek ook **reflecties** van lichtbronnen, zogenaamde **specular reflections**. Om dit zonder al te veel rekenen weer te kunnen geven, zijn er benaderingen van de fysieke werkelijkheid die een goede weergave opleveren, zoals het **Phong model**.

Met het hiervoor behandelde zijn er realistische afbeeldingen van objecten, die uit polygonen opgebouwd zijn, te maken. Op de grenzen tussen de polygonen kunnen er echter abrupte overgangen zichtbaar zijn, doordat de vlakken onder de gebruikte lichtval een te grote hoek met elkaar maken. Methoden om vloeiende overgangen te krijgen, zijn **Gouraud**- en **Phong shading**. Bij de eerste wordt er geïnterpoleerd tussen de intensiteiten van punten van het oppervlak en bij de tweede wordt er geïnterpoleerd tussen de normalen van het oppervlak.

Voor kleur moeten alle bewerkingen voor de drie hoofdkleuren apart uitgevoerd worden. Het samenvoegen van de drie lichtopbrengsten voor elk pixel geeft dan de afbeelding in kleur.

Zie [1], hoofdstuk 9, 10, 12, 13 en 14.

Hoofdstuk 11

Fouriertransformatie

11.1 Continue fouriertransformatie

11.1.1 Basisbegrippen: 1D fouriertransformatie

Stel dat $f(x)$ een continue functie is van een reële variabele x. De **fouriergetransformeerde** van $f(x)$, aangegeven met $\mathcal{F}\{f(x)\}$, wordt gedefinieerd door de vergelijking

$$\mathcal{F}\{f(x)\} = F(u) = \int_{-\infty}^{\infty} f(x)\, e^{-j2\pi ux} dx \tag{11.1}$$

Uit $F(u)$ kan via **inverse fouriertransformatie** $f(x)$ worden verkregen volgens

$$\mathcal{F}^{-1}\{F(u)\} = f(x) = \int_{-\infty}^{\infty} F(u)\, e^{j2\pi ux} du \tag{11.2}$$

De vergelijkingen (11.1) en (11.2) vormen een **paar** van fouriergetransformeerden (FT-paar) en er kan worden aangetoond dat deze bestaan als $f(x)$ continu en integreerbaar en $F(u)$ integreerbaar is. In de praktijk wordt vrijwel altijd aan deze voorwaarden voldaan.
Een verkorte notatie voor een FT-paar is

$$f(x) \Leftrightarrow F(u) \tag{11.3}$$

Bij onze toepassing hiervan op beeldverwerking hebben we te maken met functies $f(x)$ (of eigenlijk $f(x, y)$) die reëel zijn. De **fouriergetransformeerde** van een reële functie is echter complex. We schrijven daarom

$$F(u) = R(u) + jI(u) \tag{11.4}$$

of in modulus-argument notatie

$$F(u) = |F(u)|\, e^{j\phi(u)} \tag{11.5}$$

met het **fourierspectrum** (amplitude) van $f(x)$

$$|F(u)| = [R^2(u) + I^2(u)]^{1/2} \tag{11.6}$$

en de **fasehoek** van $f(x)$

$$\phi(u) = arctg\left[\frac{I(u)}{R(u)}\right] \tag{11.7}$$

Het **vermogensspectrum**, ook vaak aangeduid met **spectrale dichtheid**, is het kwadraat van het fourierspectrum:

$$P(u) = |F(u)|^2 = R^2(u) + I^2(u) \tag{11.8}$$

Als we de integraal (11.1) interpreteren als een limiet-sommatie van discrete termen, en we schrijven

$$e^{-j2\pi ux} = cos\ 2\pi ux - j\ sin\ 2\pi ux \tag{11.9}$$

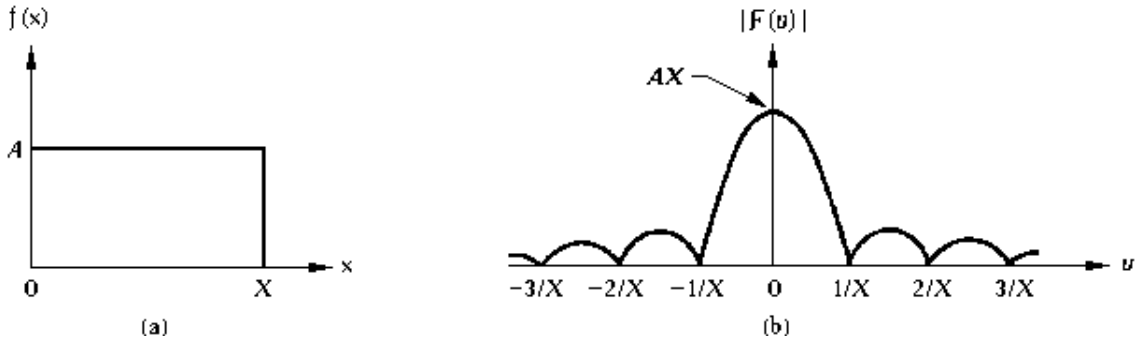


Figure 11.1: *Fouriertransformatie van een blok.*

dan is het duidelijk dat F(u) is samengesteld uit een oneindige som van sinus en cosinus termen, en dat iedere waarde van u de **frequentie** van het corresponderende sinus-cosinuspaar bepaalt. Daarom wordt u de **frequentievariabele** genoemd.

Opgave: Definieer een functie $f(x)$ als

$$f(x) = \begin{cases} 0 & x < 0 \\ A & 0 \leq x \leq X \\ 0 & x > X \end{cases} \tag{11.10}$$

Bewijs dat de fouriergetransformeerde hiervan wordt gegeven door

$$\mathcal{F}\{f(x)\} = F(u) = \frac{A}{\pi u}\, sin(\pi u X)\, e^{-j\pi u X} \tag{11.11}$$

Het fourierspectrum van deze functie is

$$|F(u)| = AX \left| \frac{sin(\pi u X)}{\pi u X} \right| \tag{11.12}$$

De functie $[sin(\pi u X)/(\pi u X)]$ wordt vaak met $sinc(\pi u X)$ aangeduid:

$$sinc(\xi) = \frac{sin\xi}{\xi} \tag{11.13}$$

□

11.1.2 Uitbreiding: 2D fouriertransformatie

De fouriertransformatie kan eenvoudig worden uitgebreid naar twee dimensies. Aan de ruimtelijke variabelen (x, y) worden nu frequentievariabelen (u, v) toegevoegd. Als $f(x, y)$ continu en integreerbaar is, en $F(u, v)$ is integreerbaar, dan bestaat het volgende paar fouriergetransformeerden:

$$\mathcal{F}\{f(x,y)\} = F(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\, e^{-j2\pi(ux+vy)} dxdy \tag{11.14}$$

$$\mathcal{F}^{-1}\{F(u,v)\} = f(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(u,v)\, e^{j2\pi(ux+vy)} dudv \tag{11.15}$$

Op analoge wijze als in het ééndimensionale geval kunnen we weer fourierspectrum, fase en spectrale dichtheid definiëren:

$$|F(u,v)| = [R^2(u,v) + I^2(u,v)]^{1/2} \tag{11.16}$$

$$\phi(u,v) = arctg\left[\frac{I(u,v)}{R(u,v)}\right] \tag{11.17}$$

$$P(u,v) = |F(u,v)|^2 = R^2(u,v) + I^2(u,v) \tag{11.18}$$

11.2 Eigenschappen van fouriergetransformeerden

Uit de definitie van fouriertransformatie kunnen eenvoudig een aantal eigenschappen worden afgeleid.

11.2.1 Separeerbaarheid

Een tweedimensionale fouriertransformatie kan worden geschreven als een herhaalde toepassing van een ééndimensionale transformatie in de twee richtingen afzonderlijk:

$$\begin{aligned}\mathcal{F}\{f(x,y)\} &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\, e^{-j2\pi(ux+vy)}dxdy \\ &= \int_{-\infty}^{\infty}\left\{\int_{-\infty}^{\infty} f(x,y)\, e^{-j2\pi ux}dx\right\} e^{-j2\pi vy}dy \end{aligned} \tag{11.19}$$

11.2.2 Lineariteit

$$af_1(x,y) + bf_2(x,y) \Leftrightarrow aF_1(u,v) + bF_2(u,v) \tag{11.20}$$

11.2.3 Translatie

$$f(x-x_0, y-y_0) \Leftrightarrow e^{-j2\pi(ux_0+vy_0)}\, F(u,v) \tag{11.21a}$$

en

$$e^{j2\pi(u_0x+v_0y)} f(x,y) \Leftrightarrow F(u-u_0, v-v_0) \tag{11.21b}$$

11.2.4 Rotatie

Introduceer poolcoördinaten

$$\begin{aligned} x &= r cos\phi \;\;,\;\; y = r sin\phi \\ u &= \omega cos\theta \;\;,\;\; v = \omega sin\theta \end{aligned}$$

Uitgedrukt in de nieuwe variabelen is een FT-paar nu

$$f(r,\phi) \Leftrightarrow F(\omega,\theta) \tag{11.22}$$

met

$$\begin{aligned} F(\omega,\theta) = \mathcal{F}\{f(r,\phi)\} &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(r,\phi)\, e^{-j2\pi r\omega\,(cos\phi cos\theta + sin\phi sin\theta)}\, r dr d\phi \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(r,\phi)\, e^{-j2\pi r\omega\, cos(\phi-\theta)}\, r dr d\phi \end{aligned} \tag{11.23}$$

en analoog

$$f(r,\phi) = \mathcal{F}^{-1}\{F(\omega,\theta)\} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(\omega,\theta)\, e^{j2\pi r\omega\, cos(\phi-\theta)}\, w dw d\theta \tag{11.24}$$

Uit deze formules kan eenvoudig worden afgeleid, dat als $f(x,y)$ wordt geroteerd over een hoek ϕ_0, dan geldt dit ook voor $F(u,v)$ (en vice versa):

$$f(r,\phi+\phi_0) \Leftrightarrow F(\omega,\theta+\phi_0) \tag{11.25}$$

11.2.5 Convolutie

Een eigenschap die een belangrijke rol speelt bij het verkrijgen van inzicht in de effecten van discretisatie, maar ook voor de practische (numerieke) toepassing van fouriertransformaties, is het **convolutietheorema**.

Functies van één variabele

De **convolutie** van twee continue functies $f(x)$ en $g(x)$ wordt gedefinieerd door

$$f(x) \otimes g(x) \equiv \int_{-\infty}^{\infty} f(\alpha)\, g(x-\alpha)\, d\alpha \tag{11.26}$$

We kunnen deze uitdrukking als volgt interpreteren. Beschouw de integraal als limiet van een sommatie van termen $f(\alpha)\, g(x-\alpha)$. Deze bijdrage aan de convolutie komt als volgt tot stand. **Verschuif** $g(x)$ langs de x-as over een afstand α. Dit levert een copie $g'(x) = g(x-\alpha)$, identiek aan $g(x)$, maar met zijn oorsprong in $x = \alpha$. Vermenigvuldig $g'(x)$ nu met de waarde van f in dit punt, dus $f(\alpha)$. Dit levert een **verschoven** én **geschaalde** copie van $g(x)$: $f(\alpha)g(x-\alpha)$. Deze term levert in principe een bijdrage aan de convolutie over de gehele x-as, afhankelijk van de breedte van $g(x)$. Eén punt van $f(x)$ is dus als het ware door het convolutieproces **uitgesmeerd**.

Als $F(u)$ en $G(u)$ de fouriergetransformeerden zijn van resp. $f(x)$ en $g(x)$, dan zegt het **convolutietheorema** dat $F(u)G(u)$ en $f(x) \otimes g(x)$ een FT-paar vormen:

$$f(x) \otimes g(x) \Leftrightarrow F(u)G(u) \tag{11.27}$$

Dus de convolutie $f(x) \otimes g(x)$ in het x-domein kan worden berekend door de inverse fouriergetransformeerde te nemen van het product $F(u)G(u)$. Een analoog verband, dat aangeeft dat convolutie in het frequentiedomein correspondeert met vermenigvuldiging in het x-domein is

$$f(x)g(x) \Leftrightarrow F(u) \otimes G(u) \tag{11.28}$$

We bewijzen de geldigheid van (11.27). Het bewijs van (11.28) loopt analoog.

Bewijs:

$$\mathcal{F}\{f(x) \otimes g(x)\} = \int_{-\infty}^{\infty} e^{-j2\pi ux} \left\{ \int_{-\infty}^{\infty} f(\alpha)\, g(x-\alpha)\, d\alpha \right\} dx \tag{11.29a}$$

$$= \int_{-\infty}^{\infty} e^{-j2\pi u\alpha} f(\alpha) \left\{ \int_{-\infty}^{\infty} e^{-j2\pi u(x-\alpha)}\, g(x-\alpha)\, dx \right\} d\alpha \tag{11.29b}$$

$$= \underbrace{\left\{ \int_{-\infty}^{\infty} e^{-j2\pi u\alpha} f(\alpha)\, d\alpha \right\}}_{F(u)} \underbrace{\left\{ \int_{-\infty}^{\infty} e^{-j2\pi ux'}\, g(x')\, dx' \right\}}_{G(u)} \tag{11.29c}$$

Functies van twee variabelen

Voor functies van twee variabelen gelden analoge uitdrukkingen. Als

$$f(x,y) \otimes g(x,y) \equiv \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(\alpha,\beta)\, g(x-\alpha, y-\beta)\, d\alpha\, d\beta \tag{11.30}$$

dan geldt

$$f(x,y) \otimes g(x,y) \Leftrightarrow F(u,v)G(u,v) \tag{11.31}$$

en

$$f(x,y)g(x,y) \Leftrightarrow F(u,v) \otimes G(u,v) \tag{11.32}$$

Het bewijs hiervan loopt analoog aan dat voor functies van één variabele.

11.2.6 Correlatie

Een grootheid die ook een belangrijke rol speelt is de **correlatie** van twee functies.

Functies van één variabele

De **correlatie** van twee continue functies $f(x)$ en $g(x)$ wordt gedefinieerd door

$$f(x) \odot g(x) \equiv \int_{-\infty}^{\infty} f^*(\alpha)\, g(\alpha + x)\, d\alpha \tag{11.33}$$

Hierin betekent "$*$" complex toegevoegde.
Analoog aan het convolutietheorema kan $f(x) \odot g(x)$ worden gerelateerd aan de fouriergetransformeerden $F(u)$ en $G(u)$, en wel op de volgende wijze:

$$f(x) \odot g(x) \Leftrightarrow F^*(u)G(u) \tag{11.34}$$

en

$$f^*(x)g(x) \Leftrightarrow F(u) \odot G(u) \tag{11.35}$$

We bewijzen (11.34).

$$\begin{aligned}
\mathcal{F}\{f(x) \odot g(x)\} &= \int_{-\infty}^{\infty} e^{-j2\pi ux} \left\{ \int_{-\infty}^{\infty} f^*(\alpha)\, g(x+\alpha)\, d\alpha \right\} dx && \text{(11.36a)} \\
&= \int_{-\infty}^{\infty} e^{j2\pi u\alpha} f^*(\alpha) \left\{ \int_{-\infty}^{\infty} e^{-j2\pi u(x+\alpha)}\, g(x+\alpha)\, dx \right\} d\alpha && \text{(11.36b)} \\
&= \underbrace{\left\{ \int_{-\infty}^{\infty} e^{+j2\pi u\alpha} f^*(\alpha)\, d\alpha \right\}}_{F^*(u)} \underbrace{\left\{ \int_{-\infty}^{\infty} e^{-j2\pi ux'}\, g(x')\, dx' \right\}}_{G(u)} && \text{(11.36c)}
\end{aligned}$$

Het bewijs van (11.35) verloopt analoog.

Functies van twee variabelen

Voor functies van twee variabelen gelden analoge uitdrukkingen. Als

$$f(x,y) \odot g(x,y) \equiv \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f^*(\alpha,\beta)\, g(\alpha + x, \beta + y)\, d\alpha d\beta \tag{11.37}$$

dan geldt

$$f(x,y) \odot g(x,y) \Leftrightarrow F^*(u,v)G(u,v) \tag{11.38}$$

en

$$f^*(x,y)g(x,y) \Leftrightarrow F(u,v) \odot G(u,v) \tag{11.39}$$

Ook nu is het bewijs weer een directe generalisering van dat voor functies van één variabele.

11.3 Discrete fouriertransformatie

11.3.1 Functies van één variabele (1D fouriertransformatie)

We beschouwen een continue functie $f(x)$ die is gediscretiseerd tot een reeks $\{\ f(x_0),\ f(x_0 + \Delta x),\ f(x_0 + 2\Delta x),\ \ldots,\ f(x_0 + [N-1]\Delta x)\ \}$ door bemonstering (**sampling**) in N punten op onderlinge afstand Δx.
We gebruiken nu de notatie

$$f(x) \equiv f(x_0 + x\Delta x) \tag{11.40}$$

met $x = 0, 1, 2, \ldots, N-1$, dus x wordt gebruikt als **discrete** (tel)variabele. M.a.w. de reeks $\{\ f(0),\ f(1),\ f(2),\ \ldots,\ f(N-1)\ \}$ zal worden gebruikt om een **willekeurige** set van N equidistante samples van een corresponderende

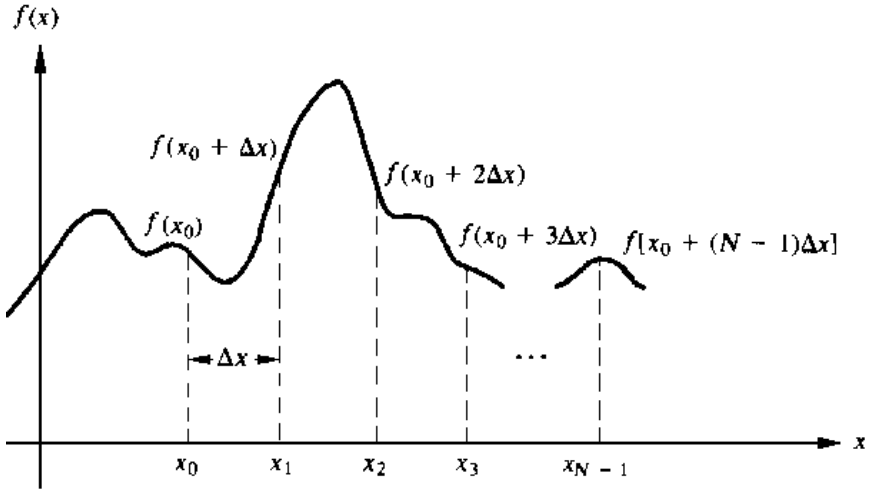


Figure 11.2: *Bemonstering van een continue functie $f(x)$.*

continue functie $f(x)$ aan te geven. Of eventueel op een bepaalde plaats de oorspronkelijke continue definitie van x wordt bedoeld zal uit de context duidelijk zijn.
Met deze notatie definiëren we als tegenhangers van (11.1) en (11.2) het **discrete** FT-paar

$$F(u) = \frac{1}{N} \sum_{x=0}^{N-1} f(x)\, e^{-j2\pi ux/N} \tag{11.41}$$

voor $u = 0, 1, \ldots, N-1$, en

$$f(x) = \sum_{u=0}^{N-1} F(u)\, e^{j2\pi ux/N} \tag{11.42}$$

voor $x = 0, 1, \ldots, N-1$.
De waarden $u = 0, 1, \ldots, N-1$, in de discrete fouriertransformatie (11.41) corresponderen met samples van de **continue** transformatie voor de waarden $u = 0, \Delta u, 2\Delta u, \ldots, (N-1)\Delta u$. M.a.w. we representeren $F(u\Delta u)$ door $F(u)$. Deze notatie is gelijk aan die voor de discrete $f(x)$, met het verschil echter, dat de samples van $F(u)$ in de oorsprong van de frequentie-as beginnen.

11.3.2 Functies van twee variabelen (2D fouriertransformatie)

Voor twee dimensies i.p.v. één kunnen we analoge uitdrukkingen opschrijven. $f(x, y)$ wordt bemonsterd op een rechthoekig rooster, met M punten in de x-richting en N punten in de y-richting, op onderlinge afstand Δx resp. Δy. Op dezelfde wijze als boven representeert $f(x, y)$ een sample $f(x_0 + x\Delta x, y_0 + y\Delta y)$ van de **continue** functie. Eenzelfde analogie geldt voor $F(u, v)$. Het $2D$ FT-paar is dan

$$F(u,v) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y)\, e^{-j2\pi(ux/M + vy/N)} \tag{11.43}$$

en

$$f(x,y) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u,v)\, e^{j2\pi(ux/M + vy/N)} \tag{11.44}$$

Vaak worden beelden bemonsterd op een vierkant rooster, dus $M = N$. Het FT-paar kan dan worden geschreven als

$$F(u,v) = \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y)\, e^{-j2\pi(ux+vy)/N} \tag{11.45}$$

en

$$f(x,y) = \frac{1}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} F(u,v)\, e^{j2\pi(ux+vy)/N} \tag{11.46}$$

Merk op dat de voorfactor $1/N^2$ nu symmetrisch verdeeld is als factoren $1/N$ in $F(u,v)$ én $f(x,y)$. Een dergelijke verdeling is arbitrair, omdat slechts de consistentie van $F(u,v)$ en $f(x,y)$ als FT-**paar** van belang is.
In het nu volgende deel wordt ervan uitgegaan, dat met een vierkant $N \times N$ wordt gewerkt.

11.3.3 Separeerbaarheid

We kunnen 11.45 en 11.46 respectievelijk schrijven als

$$F(u,v) = \frac{1}{N} \sum_{x=0}^{N-1} e^{-j2\pi ux/N} \sum_{y=0}^{N-1} f(x,y)\, e^{-j2\pi vy/N} \tag{11.47}$$

en

$$f(x,y) = \frac{1}{N} \sum_{u=0}^{N-1} e^{j2\pi ux/N} \sum_{v=0}^{N-1} F(u,v)\, e^{j2\pi vy/N} \tag{11.48}$$

Schrijf 11.47 als

$$F(u,v) = \frac{1}{N} \sum_{x=0}^{N-1} F(x,\nu)\, e^{-j2\pi ux/N} \tag{11.49}$$

met

$$F(x,v) = N \left[\frac{1}{N} \sum_{x=0}^{N-1} f(x,y)\, e^{-j2\pi vy/N} \right] \tag{11.50}$$

Voor **iedere** waarde van x is de uitdrukking tussen de rechte haken een ééndimensionale transformatie met frequentiewaarden $v = 0, 1, \ldots, N-1$. Daarom kan de tweedimensionale functie $F(x,v)$ verkregen worden door een transformatie te nemen langs **iedere** rij van $f(x,y)$ en het resultaat met N te vermenigvuldigen. Het gewenste eindresultaat $F(u,v)$ wordt dan verkregen door een transformatie te nemen langs iedere **kolom** van $F(x,v)$. Deze procedure is weergegeven in figuur 11.3.

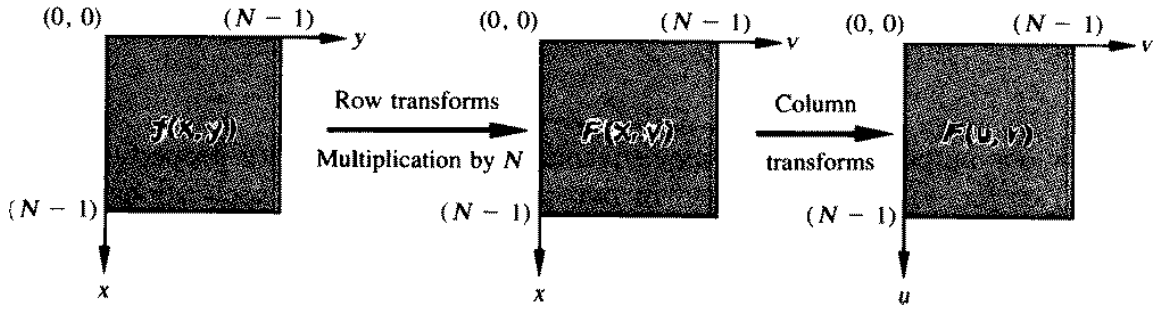


Figure 11.3: *Berekening van de tweedimensionale fouriertransformatie als een reeks ééndimensionale transformaties.*

11.3.3.1 Periodiciteit en conjugatiesymmetrie

De discrete fouriergetransformeerde en zijn inverse zijn **periodiek** met periode N

$$F(u,v) = F(u+N,v) = F(u,v+N) = F(u+N,v+N) \tag{11.51}$$

Als f(x,y) reëel is, dan vertoont de fouriergetransformeerde ook conjugatiesymmetrie:

$$F(u,v) = F^*(-u,-v) \tag{11.52}$$

Hieruit volgt ook

$$|F(u,v)| = |F(-u,-v)| \tag{11.53}$$

Dit impliceert dat $|F(u,v)|$ spiegelsymmetrie bezit t.o.v. de u-as en de v-as.

11.3.3.2 Translatie

De discrete equivalenten van (11.21) zijn

$$f(x-x_0, y-y_0) \Leftrightarrow e^{-j2\pi(ux_0+vy_0)/N}\, F(u,v) \tag{11.54a}$$

en

$$e^{j2\pi(u_0x+v_0y)/N} f(x,y) \Leftrightarrow F(u-u_0, v-v_0) \tag{11.54b}$$

Een interessant speciaal geval is translatie over $u_0 = v_0 = N/2$. Dan is

$$e^{j2\pi(u_0x+v_0y)/N} = e^{j\pi(x+y)} = (-1)^{x+y}$$

zodat

$$(-1)^{x+y} f(x,y) \Leftrightarrow F(u-N/2, v-N/2) \tag{11.55}$$

Dus de oorsprong van de fouriergetransformeerde van $f(x,y)$ kan verschoven worden naar het centrum van het corresponderende $N \times N$ frequentie vierkant door $f(x,y)$ eenvoudig te vermenigvuldigen met $(-1)^{x+y}$.

11.4 Sampling

We hebben de formules gegeven waarmee de discrete fouriertransformatie en de inverse hiervan gedefinieerd worden. We gaan nu in op de relatie tussen de continue en discrete versie. In het bijzonder zal dit gericht zijn op de vraag of het sampling proces zodanig gekozen kan worden dat geen informatie verloren gaat, en als tegenhanger hiervan de vraag hoe de aanwezigheid van onvoldoende samples tot uitdrukking komt.

11.4.1 1D sampling

Beschouw een functie $f(x)$ die continu is en loopt van $-\infty$ tot ∞. Bemonsteren (zie figuur 11.4) voeren we mathematisch uit door $f(x)$ te vermenigvuldigen met een **pulstrein** $s(x)$, een functie die bestaat uit equidistante δ-pulsen op onderlinge afstand Δx

$$s(x) = \sum_{k=-\infty}^{\infty} \delta(x - k\Delta x) \tag{11.56}$$

De fouriergetransformeerde hiervan is

$$\mathcal{F}\{s(x)\} \equiv S(u) = \frac{1}{\Delta x} \sum_{k=-\infty}^{\infty} \delta\left(u - \frac{k}{\Delta x}\right) \tag{11.57}$$

dus weer een reeks pulsen, maar nu in het frequentiedomein en op onderlinge afstand $1/\Delta x$. Aangezien vermenigvuldiging in het plaatsdomein correspondeert met convolutie in het frequentiedomein (zie (11.28)), betekent dit, dat $F(u)$ periodiek wordt herhaald met periode $1/\Delta x$. Dit impliceert dat alle relevante informatie aanwezig is in een interval met een breedte $1/\Delta x$. Echter, als Δx te groot is, overlappen opeenvolgende copieën van $F(u)$ elkaar en is exacte reconstructie van $f(x)$ niet mogelijk. Als $f(x)$ een in **bandbreedte begrensde** functie is, d.i. $F(u) = 0$ voor $|u| > W$, dan wordt dit overlap probleem opgelost als de sampling afstand Δx zodanig wordt gekozen, dat

$$\Delta x \leq \frac{1}{2W} \tag{11.58}$$

Dit wordt wel het **Nyquist** criterium genoemd en $1/\Delta x$ heet de **Nyquist frequentie**. De periodiek voortgezette copieën van $F(u)$ verschijnen dan geïsoleerd van elkaar op de u-as. Het belang hiervan ligt in het feit, dat we nu $F(u)$ volledig kunnen isoleren door $S(u) \otimes F(u)$ te vermenigvuldigen met de functie

$$G(u) = \begin{cases} 0 & u < -W \\ 1 & -W \leq u \leq W \\ 0 & u > W \end{cases} \tag{11.59}$$

De inverse fouriergetransformeerde reproduceert dan de **volledige continue** functie

$$f(x) = \mathcal{F}^{-1}\{G(u)[F(u) \otimes S(u)]\} \tag{11.60}$$

Het feit dat een in bandbreedte begrensde functie volledig gereconstrueerd kan worden uit samples op onderlinge afstand Δx, mits deze voldoet aan (11.58), staat bekend als het **bemonsterings theorema van Shannon**[1].

[1] Vaak ziet men hier de namen **Kotelnikov** en/of **Whittaker** nog aan toegevoegd.

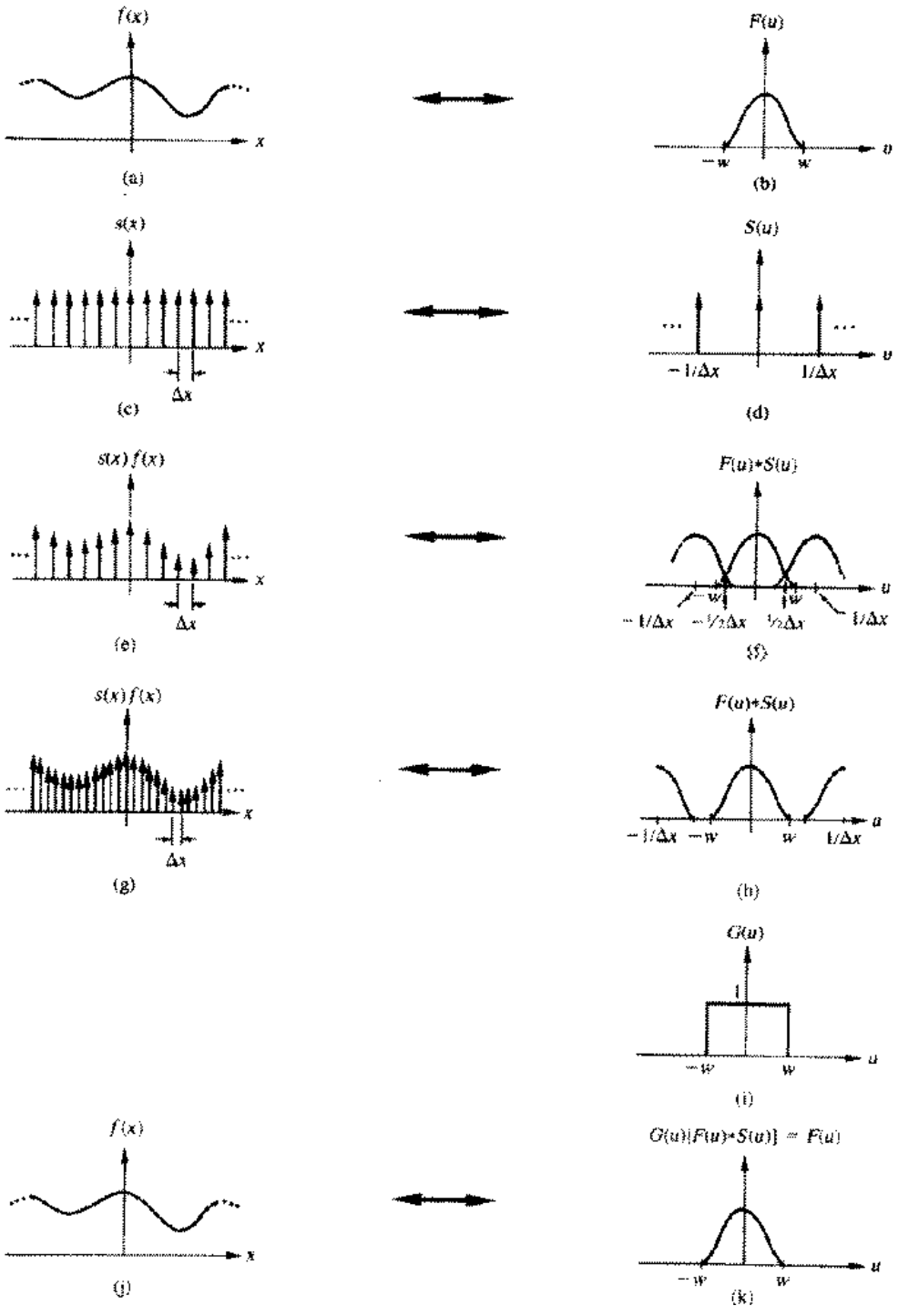


Figure 11.4: *Grafische weergave van sampling concepten.*

11.4.1.1 Samples te ver uit elkaar

Als **niet** aan (11.58) wordt voldaan, dus als de x-samples te ver van elkaar liggen, dan gaat informatie in de randintervallen $[-W, -1/2\Delta x]$ en $[1/2\Delta x, W]$ verloren. In combinatie met de overlap met aangrenzende copieën van $F(u)$ kan dit gezien worden als **terugvouwen** ("**foldover**") van het spectrum om $u = -1/\Delta x$ respectievelijk $u = 1/\Delta x$. Dit leidt tot laagfrequente artefacten in het gereconstrueerde beeld, ofwel "**aliasing**". Vooral bij beelden die van zichzelf al een sterke periodiciteit bezitten, zoals bijvoorbeeld de atomaire patronen die met Scanning Tunneling Microscopie zichtbaar gemaakt kunnen worden, worden zo niet-aanwezige periodiciteiten geïntroduceerd!

11.4.1.2 Het aantal samples is eindig

Het Shannon theorema refereert aan functies $f(x)$ die een onbegrensd x-bereik hebben. In de praktijk heeft men echter altijd met een **eindig** sampling interval $[0, X]$ te maken (zie figuur 11.5). Dit kan mathematisch worden gerealiseerd, door $f(x)$ te vermenigvuldigen met een **window**

$$h(x) = \begin{cases} 0 & x < 0 \\ 1 & 0 \leq x \leq X \\ 0 & x > X \end{cases} \tag{11.61}$$

In het frequentiedomein correspondeert dit met convolutie van $S(u) \otimes F(u)$ met $H(u)$, de fouriergetransformeerde van de windowfunctie $h(x)$. Aangezien $H(u)$ frequentiecomponenten heeft die tot oneindig doorlopen, zij het met afnemende amplitude (zie (11.10) e.v.), leidt het toepassen van de de windowfunctie tot vervorming van het spectrum. Immers, het is nu niet langer mogelijk om het **volledige** fourierspectrum $F(u)$ te isoleren. Bij terugtransformatie verschijnt dan een functie, die niet meer begrensd is op het interval $[0, X]$, maar zich uitstrekt van $-\infty$ tot ∞. Dit kan worden samengevat in de volgende uitspraken:

- Een functie $f(x)$ met een eindige duur kan niet bandgelimiteerd zijn.
- Een functie $f(x)$ die bandgelimiteerd is moet zich uitstrekken van $-\infty$ tot ∞ in het x-domein.

Een uitzondering hierop is een functie die bandgelimiteerd is én periodiek met een periode X, dus exact de breedte van het window.

11.4.1.3 Periodiciteit van de discrete fouriergetransformeerde

Er is afgeleid dat bemonsteren van een functie $f(x)$ in punten op onderlinge afstand Δx leidt tot een periodieke continue functie $F(u)$ in het frequentiedomein met periode $1/\Delta x$. Om een discrete fouriertransformatie te verkrijgen moeten we ook in het frequentiedomein sampling toepassen. Hiertoe vermenigvuldigen we $F(u)$ met een reeks pulsen op onderlinge afstand Δu, waarvoor we weer de notatie $S(u)$ gebruiken. $\mathcal{F}^{-1}\{S(u)\} = s(x)$ is een pulsreeks met periode $1/\Delta u$. De equivalente operatie in het x-domein is convolutie van $f(x)$ met $s(x)$. Dus de oorspronkelijke bemonsterde functie wordt nu (na reconstructie) periodiek herhaald met periode $1/\Delta u$. Als N equidistante samples van $f(x)$ en $F(u)$ worden genomen, zodanig dat in ieder domein precies één periode gedekt wordt, dan hebben we $X = N\Delta x = 1/\Delta u$ in het x-domein en $N\Delta u = 1/\Delta x$ in het frequentiedomein. Hieruit volgt direct

$$\Delta u = \frac{1}{N\Delta x} \tag{11.62}$$

11.4.1.4 Discrete convolutie

Het mechanisme van discrete convolutie is in essentie hetzelfde als dat voor continue convolutie. Het enige verschil is, dat de verschuivingen (zie toelichting onder (11.26)) nu discreet zijn. Maar er doet zich een complicatie voor. We bespreken dit aan de hand van functies van één variabele. Stel dat $f(x)$ en $g(x)$ zijn gediscretiseerd in arrays met respectievelijk lengte A en B: $\{f(0), f(1), f(2), \ldots, f(A-1)\}$ en $\{g(0), g(1), g(2), \ldots, g(B-1)\}$. Het convolutieproces vermengt bijdragen van periodieke herhalingen van de functies met elkaar. Om dit te voorkomen

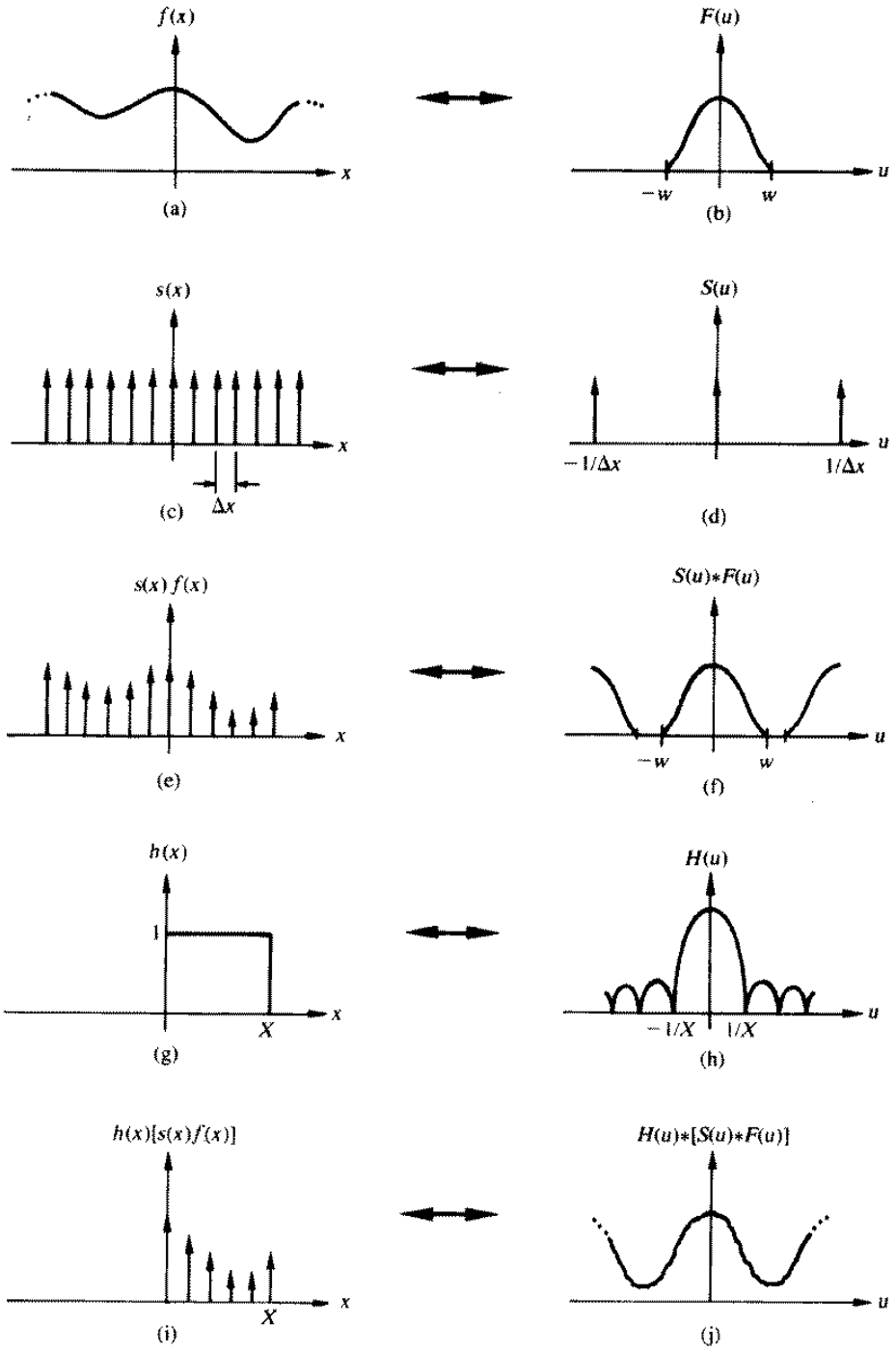


Figure 11.5: *Sampling van $f(x)$ over een eindig interval $[0, X]$ correspondeert met convolutie van $F(u)$ met een functie $H(u)$ in het frequentiedomein.*

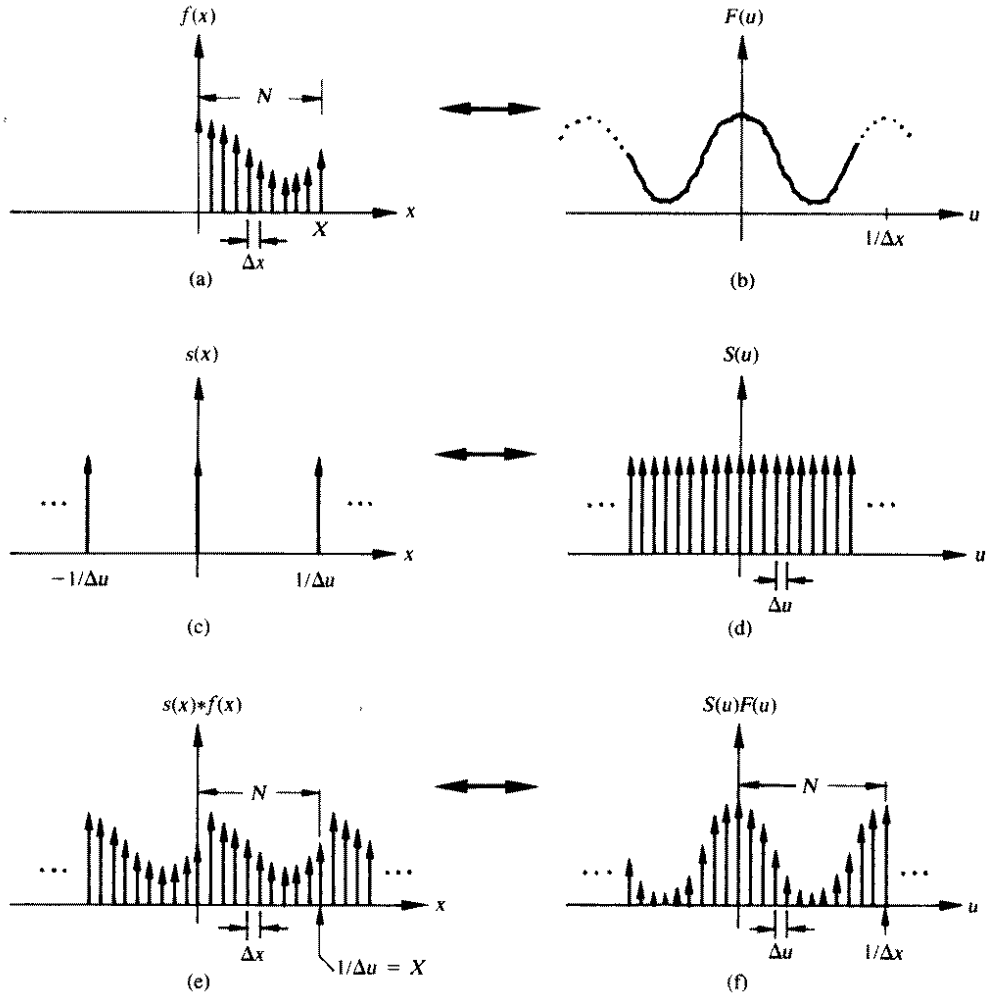


Figure 11.6: *Sampling van een functie $f(x)$ in een eindig aantal punten N op onderlinge afstand Δx levert in het frequentiedomein een continue functie. Als deze op zijn beurt wordt bemonsterd met pulsen op onderlinge afstand Δu, dan is het gereconstrueerde signaal in het x-domein $s(x) \otimes f(x)$ een periodieke herhaling van de oorspronkelijke bemonsterde functie $f(x)$, met periode $1/\Delta u$.*

wordt het volgende gedaan. Allereerst stellen we vast, dat we slechts één copie van f met één copie van g willen convolueren. Dit wordt bereikt door beide arrays tot lengte M uit te breiden door het toevoegen van nullen. Dit is toegestaan, omdat deze extra termen geen bijdrage zullen leveren (vermenigvuldiging met nul levert nul). De resulterende convolutie zal dan óók periodiek zijn met deze periode M. Er kan worden aangetoond dat **tenzij** we kiezen

$$M \geq A + B + 1 \tag{11.63}$$

de individuele perioden van de convolutie elkaar zullen overlappen (**wraparound error**). Als aan deze voorwaarde wordt voldaan, dan wordt slechts één copie van iedere functie in het convolutieproces betrokken. De optimale keus voor M (minimum aantal termen consistent met (11.63)) is $A + B + 1$. Dit levert de verlengde (**extended**) reeksen:

$$f_e(x) = \begin{cases} f(x) & 0 \leq x \leq A-1 \\ 0 & A \leq x \leq M-1 \end{cases} \tag{11.64}$$

en

$$g_e(x) = \begin{cases} g(x) & 0 \leq x \leq B-1 \\ 0 & B \leq x \leq M-1 \end{cases} \tag{11.65}$$

Hiermee wordt de **discrete convolutie** van $f_e(x)$ en $g_e(x)$ gedefinieerd als

$$f_e(x) \otimes g_e(x) \equiv \sum_{m=0}^{M-1} f_e(m) g_e(x-m) \tag{11.66}$$

voor $x = 0, 1, 2, \ldots, M-1$. Dit convolutieproces levert dus M termen, die precies één periode van $f_e(x) \otimes g_e(x)$, d.i. $x = 0, 1, 2, \ldots, M-1$, beslaan.
Het discrete convolutietheorema kunnen we nu analoog aan (11.27) en (11.28) formuleren, waarbij echter de **extended** versies $f_e(x)$ en $g_e(x)$ van de gediscretiseerde functies gebruikt moeten worden. De discrete variablen x en u hebben hierin allebei waarden in het bereik $0, 1, 2, \ldots, M-1$.

11.4.2 2D sampling

Een geheel analoge beschrijving kan worden gegeven voor functies $f(x, y)$ van twee variabelen. De bemonsterings functie is nu

$$s(x, y) = \sum_{k=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} \delta(x - k\Delta x)\delta(y - m\Delta y) \tag{11.67}$$

In essentie wordt dan het voorgaande betoog herhaald voor zowel x als y. Er komen geen wezenlijk nieuwe resultaten te voorschijn. De generaliseringen zullen duidelijk zijn aan de hand van figuur 11.7 en 11.8.

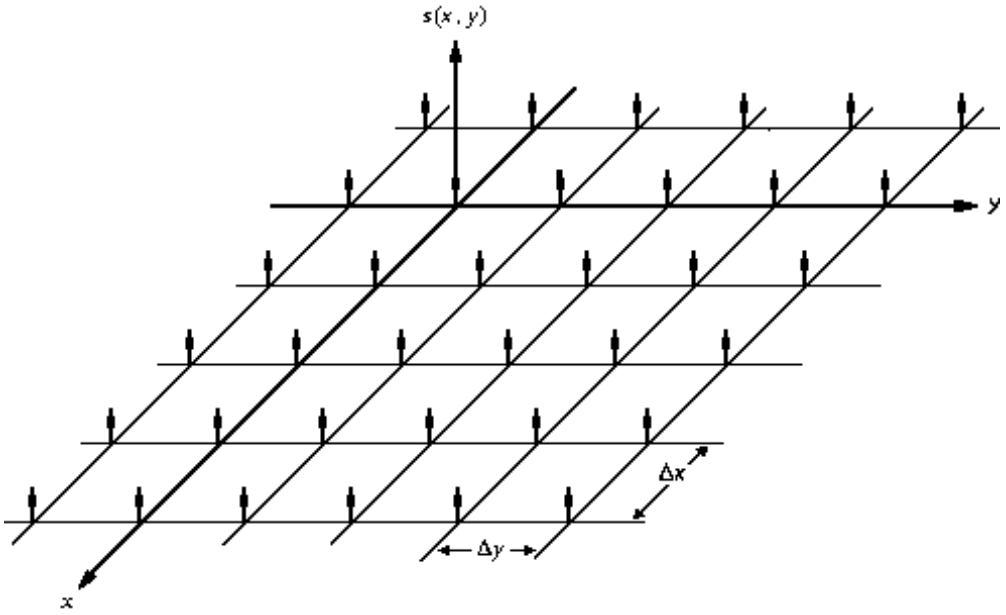


Figure 11.7: *Een tweedimensionale sampling functie* $s(x, y)$.

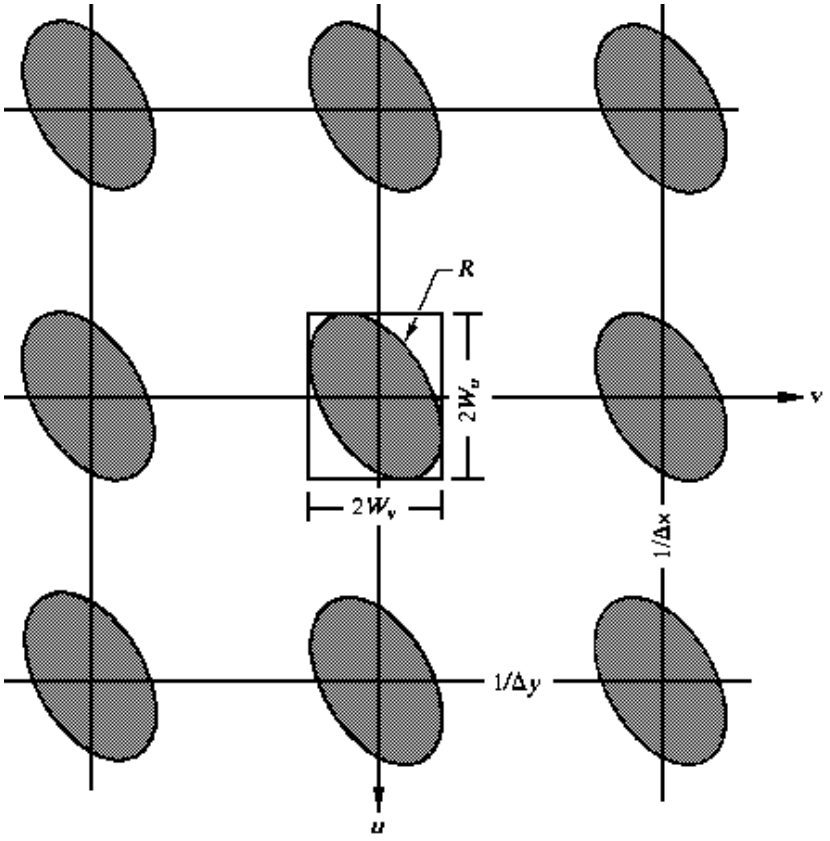


Figure 11.8: *Representatie in het frequentiedomein van een bemonsterde tweedimensionale, band-gelimiteerde functie.*

Hoofdstuk 12

Afbeeldingstechnieken en reconstructie

12.1 Inleiding

12.1.1 Afbeeldingstechnieken: enkele voorbeelden

Licht stelt ons in staat objecten in onze omgeving direct waar te nemen. Omdat licht in het algemeen een gering doordringend vermogen heeft kijkt men in essentie naar reflectie-eigenschappen van het **oppervlak** van een voorwerp. Wil men informatie over het inwendige verkrijgen, dan zal in het algemeen fysiek een doorsnede gemaakt moeten worden. De ontdekking van de röntgenstraling (1895) heeft echter deze beperkingen aanzienlijk verlegd en werd al snel onderkend als van groot belang in de medische diagnostiek. Deze straling blijkt een tamelijk groot doordringend vermogen in het menselijk lichaam te hebben, waarbij de absorptie afhangt van het type weefsel dat wordt gepasseerd. Plaatst men achter het af te beelden object (deel van de patiënt) een fotografische plaat, dan ontstaat een schaduwbeeld, waarbij de stralingsintensiteit op een bepaalde positie wordt bepaald door de absorptie die onderweg is opgetreden.

Bij exploratieseismologie van aardlagen (opsporen van olie, gas, steenkool) maakt men gebruik van mechanische trillingen, welke aan het aardoppervlak worden gegenereerd. De trillingen (geluidsgolven) planten zich voort in de bodem en worden gereflecteerd in gebieden waar de akoestische impedantie, dus de bodemsamenstelling, verandert. Een netwerk van trillingsopnemers op het oppervlak detecteert deze gereflecteerde golven en via computer-reconstructietechnieken wordt op deze wijze de bodemsamenstelling in kaart gebracht.

Radio-isotopen zijn atomen, welke door radioactief verval α-, β-, γ- of röntgenstraling uitzenden, afhankelijk van het type isotoop. Om de fysiologische functie van een bepaald orgaan in een lichaam in kaart te brengen wordt een dergelijk isotoop als **label**, als deel van een groter molecuul dat specifiek bij de processen in het orgaan betrokken is, in het lichaam ingebracht. De geëmitteerde γ- of röntgenstraling (α- en β-deeltjes hebben in weefsel een te korte dracht), wordt extern gedetecteerd. Dit levert een beeld van de verdeling van de toegediende tracer.

Deze voorbeelden maken deel uit van een grote klasse afbeeldingstechnieken, waarbij één of andere vorm van straling wordt gebruikt:
- Electromagnetische straling (radiofrequent-, zichtbare-, infrarood-, röntgen-, γ-straling).
- Elementaire deeltjes (positron annihilatie, electronenbundel).
- Mechanische trillingen (geluidsgolven, ultrageluid).

Deze straling is a.h.w. gecodeerd met informatie over het object waarvan het afkomstig is. De wijze waarop deze codering tot stand komt maakt een volgende indeling mogelijk.
- Reflectie (lichtmicroscopie, fotografie, holografie, ultrageluid-echografie, doppler-echografie, electronenmicroscopie (SEM)).
- Transmissie (lichtmicroscopie, röntgendiagnostiek, computer aided tomography (CT), electronenmicroscopie (TEM), ultrageluidtomografie).
- Emissie (IR fotografie, γ-camera, positron annihilatie- of emissie, nuclear magnetic resonance (NMR), veldemissiemicroscopie).
- Secundaire emissie (scanning auger (electronen), microprobe (X-ray), SIMS (ionen)).

Bij bovenstaande technieken kan de afbeelding tot stand komen door het scannen van bron en/of detector, waarbij

vaak ook een parallelle verwerking (array detectoren) van de signalen wordt toegepast. Soms is het mogelijk een directe afbeelding te maken, bijvoorbeeld door middel van lenzen (optische voor licht of magnetische voor electronen). Meestal wordt echter met behulp van een computer een reconstructie geconstrueerd.

12.1.2 Grenzen van de beeldkwaliteit

De keuze voor een bepaalde afbeeldingstechniek wordt primair bepaald door de eigenschap van het object die men in kaart wil brengen. Men zal in ieder geval zoveel mogelijk details willen zien. Twee fysische basisgrootheden die de kwaliteit van een afbeeldingstechniek karakteriseren zijn **contrast** en **oplossend vermogen**.

12.1.2.1 Contrast

Als we met I_{min} en I_{max} respectievelijk de minimale en maximale intensiteit van de straling afkomstig van aangrenzende delen aangeven, dan kan het contrast C worden gedefinieerd als

$$C = \frac{I_{max} - I_{min}}{I_{max} + I_{min}} \tag{12.1}$$

Hoe groter C is, hoe meer details zichtbaar zullen zijn. Een voorbeeld hiervan is röntgenstraling. Harde röntgenstraling wordt door de verschillend soorten weefsel in het menselijk lichaam nauwelijks geabsorbeerd. In dit geval is $I_{min} \simeq I_{max} \simeq 1$, zodat $C \simeq 0$. Anderzijds wordt te zachte röntgenstraling zeer sterk geabsorbeerd, dus dan is $I_{min} \simeq I_{max} \simeq 0$, hetgeen opnieuw leidt tot $C \simeq 0$. Men zal daarom een frequentie in het middengebied moeten kiezen.

12.1.2.2 Oplossend vermogen

Hoe klein de details zijn welke nog waargenomen kunnen worden, hangt af van factoren, welke voor de diverse afbeeldingstechnieken verschillen. Een zeer fundamentele grens wordt opgelegd door de golflengte van de gebruikte straling. In het algemeen zullen details kleiner dan de golflengte niet meer zichtbaar zijn. Bij optische microscopen is de golflengte van zichtbaar licht de grens, dus typisch $\mathcal{O}(500\,nm)$. Geluid heeft in water een snelheid van 1500 ms^{-1}, dus bij een frequentie f is de het kleinst waarneembare detail $\lambda = 1500/f$. Bij een frequentie van $15\,kHz$ is de ondergrens dan $0.1\,m$, bij $1.5\,MHz$ is dit $1\,mm$.
Voor de niet-klassieke afbeeldingstechnieken (d.w.z. die niet met lenzen of schaduwbeelden werken), waar we in dit hoofdstuk nader op in gaan, is de fijnheid van de reconstructie afhankelijk van de beperktheid van filters.

12.2 Beeldreconstructie

12.2.1 Computer assisted tomography

In de inleiding hebben we een aantal afbeeldingstechnieken genoemd. Laten we ons, om de gedachten te bepalen, nu richten op één eenvoudig scansysteem voor zogenaamde transaxiale röntgentomografie. Hierbij wordt de transmissie van röntgenstraling door een object gebruikt om informatie over het inwendige te verkrijgen. Deze methode valt binnen een klasse afbeeldingstechnieken, die met de term **computer assisted tomography** (CAT of CT) wordt aangeduid. De beeldvorming komt hier tot stand via computerbewerking van de gedetecteerde signalen. Definieer een assenstelsel $\{x, y, z\}$. Een smalle bundel röntgenstralen, met een richting gekarakteriseerd door de hoek ϕ in een vlak $z = z_0 = constant$, doorsnijdt het voorwerp en wordt na transmissie door een detector opgevangen. Bron en detector voeren simultaan een translatie uit in het xy-vlak (t-richting) loodrecht op de bundel (s-richting). Het st-stelsel is geroteerd over een hoek ϕ t.o.v. het xy-stelsel. De röntgenstralingsbundel maakt zo een doorsnede ("slice") van het voorwerp.
Wat is de informatie die op deze wijze door de detector als functie van zijn positie t wordt opgevangen? Tijdens de voortgang van de bundel gaan fotonen verloren door verstrooiïng of absorptie. In ieder punt wordt dit verlies gekarakteriseerd door een lineaire verzwakkingscoëfficient $\mu(x, y)$ (de constante z-coördinaat z_0 is als argument weggelaten). Als we met $I(x, y)$ de intensiteit van de bundel op positie (x, y) aangeven, dan is

$$\frac{dI(x,y)}{ds} = -\mu(x,y)I(x,y) \tag{12.2}$$

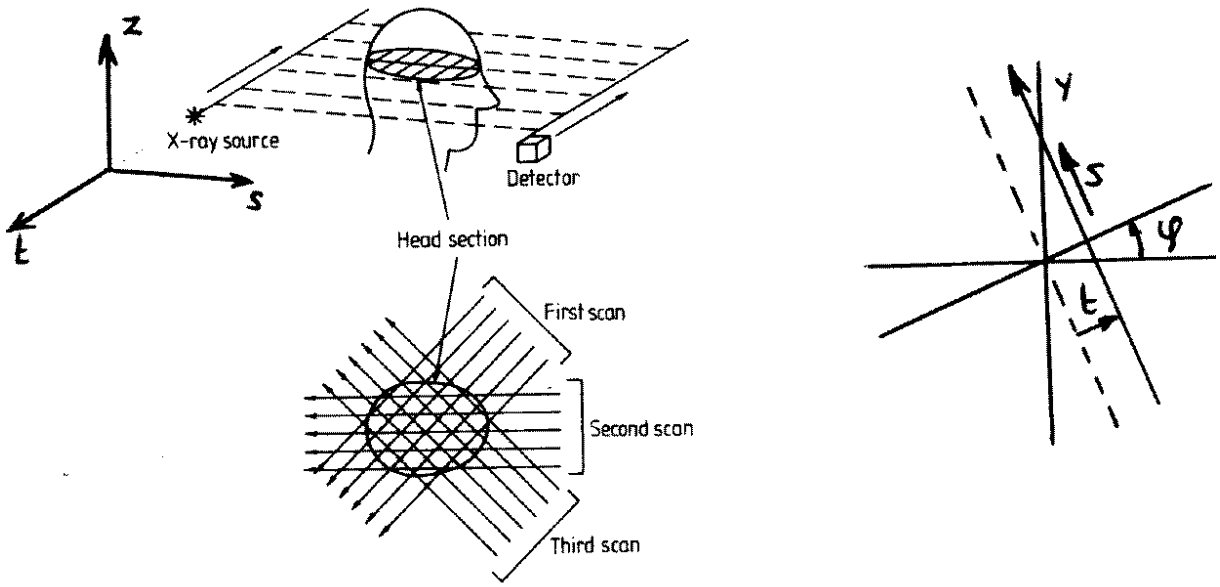


Figure 12.1: *Computer assisted tomography.*

waarbij s de coördinaat langs de bundel is. Integratie geeft

$$I_{uit} = I_{in}\, exp\left(-\int_{bundel} \mu(x,y)ds\right) \tag{12.3}$$

I_{in} en I_{uit} de intensiteiten zijn van de inkomende, respectievelijk doorgelaten bundel. Hiermee definieren we de **projectie** $p(\phi,t)$ als

$$p(\phi,t) \equiv ln\left(\frac{I_{in}}{I_{uit}}\right) = \int_{bundel} \mu(x,y)ds \tag{12.4}$$

In woorden: $p(\phi,t)$ is de lijnintegraal van $\mu(x,y)$ langs de bundel. Bij vaste ϕ ontstaat een intensiteitsprofiel als functie van t, die de **parallelle projectie** van $\mu(x,y)$ wordt genoemd. De vraag is nu, of het mogelijk is om uit dergelijke projecties (voor verschillende waarden van ϕ) de oorspronkelijke functie $\mu(x,y)$ te reconstrueren. Dit is niet á priori duidelijk, immers er gaat bij projectie informatie verloren, omdat voor iedere bron/detector positie slechts de absorptie **geïntegreerd langs de bundel** wordt gemeten.
Dit probleem werd lang voordat toepassingen voor tomografie in zicht waren al bestudeerd. De oudste publicatie over dit probleem verscheen al in 1917, geïnspireerd door een sterrenkundige vraagstelling[1]. Met de komst van computers is dit onderwerp van zeer groot belang geworden, omdat deze practische implementatie mogelijk hebben gemaakt.

In de volgende paragraaf zullen we (voor het geval van parallelle projectie) aantonen, dat inderdaad in theorie het inverse proces $p(\phi,t) \rightarrow \mu(x,y)$ mogelijk is. Daarna zullen we enkele aspecten van de practische implementatie hiervan laten zien.

12.2.2 Het fourier slice theorema

Het reconstructieprobleem kan wiskundig als volgt geformuleerd worden. Laat $f(x,y)$ een twee-dimensionale functie representeren. Een lijn die door $f(x,y)$ loopt noemen we een **straal**. Deze wordt gekarakteriseerd door een richting ϕ en de loodrechte afstand t tot de oorsprong van het xy-stelsel. De vergelijking van deze lijn is

$$xcos\phi + ysin\phi = t \tag{12.5}$$

De integraal van $f(x,y)$ langs deze lijn is

$$p(\phi,t) = \int_{straal} f(x,y)ds = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\,\delta(xcos\phi + ysin\phi - t)\,dxdy \tag{12.6}$$

[1] J. Radon, *Uber die Bestimmung von Funktionen durch ihre Integralwerte langs gewisser Mannigfaltigkeiten, Ber. Verh. Sachs. Akad. Wiss. Leipzig Math. Phys. Kl.* **69**, 262-77 (1917).

De ééndimensionale integraal over s (langs de straal) is hier omgevormd tot een tweedimensionale integraal over x en y: de δ-functie selecteert juiste die waarden die voldoen aan (12.5). $p(\phi, t)$ als functie van t voor vaste ϕ definiëert de **parallelle projectie**[2] van $f(x, y)$. De tweedimensionale functie $p(\phi, t)$ wordt ook wel de **radontransformatie** van $f(x, y)$ genoemd.
Ga nu over van het plaats (t-)domein over naar het frequentie (w-)domein via fouriertransformatie:

$$P(\phi, w) = \int_{-\infty}^{\infty} e^{-j2\pi wt}\, p(\phi, t)\, dt \tag{12.7}$$

Als we hier (12.6) invullen, dan blijkt dat de term $\delta(xcos\phi + ysin\phi - t)$ deze integraal reduceert tot een factor $\exp(-j2\pi w(xcos\phi + ysin\phi))$.
We vinden zo

$$P(\phi, w) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-j2\pi w(xcos\phi+ysin\phi)} f(x, y)\, dxdy \tag{12.8}$$

en dit kunnen we schrijven als

$$P(\phi, w) = F(u, v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-j2\pi(ux+vy)} f(x, y)\, dudv \tag{12.9}$$

met $u = wcos\phi$ en $v = wsin\phi$. Dit definieert in het uv-vlak een lijn door de oorsprong, die een hoek ϕ met de u-as maakt en waarlangs w varieert. Dus, de eendimensionale fouriergetransformeerde van de **projectie** van $f(x, y)$ in de richting ϕ levert de tweedimensionale fouriergetransformeerde van $f(x, y)$ zèlf langs een radiële lijn in het uv-vlak. Door dit proces te herhalen voor een aantal waarden van ϕ kan dus $F(u, v)$ geconstrueerd worden en daarna kan in principe door inverse fouriertransformatie f(x,y) worden bepaald met

$$f(x, y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{j2\pi(ux+vy)} F(u, v)\, dudv \tag{12.10}$$

Dit resultaat staat bekend als het **fourier slice theorema**.

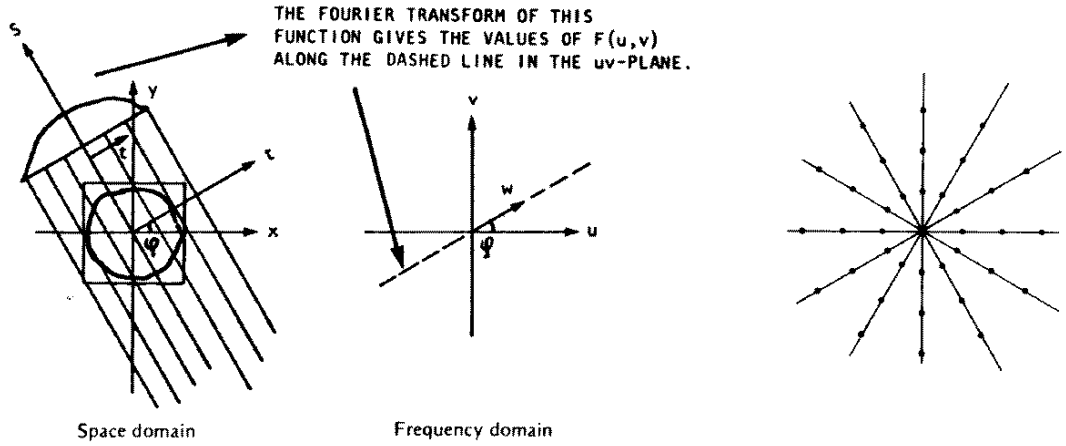


Figure 12.2: *Het fourier slice theorema.*

12.2.2.1 Bemonsteringsproblemen bij practische implementatie

Als een meting slechts bij een eindig aantal detectorposities (ϕ, t) wordt uitgevoerd, dan kan ook $f(x, y)$ slechts in een eindig aantal punten worden gereconstrueerd. Als we $f(x, y)$ willen bepalen op een $N \times N$ vierkant rooster, dan zijn N^2 punten in het uv-vlak, óók op een vierkant rooster, nodig. De terugtransformatie kan dan numeriek zeer snel worden uitgevoerd met een **fast fouriertransformatie** (FFT).

Het is in principe mogelijk om $F(u, v)$ in deze punten te vinden uit N^2 waarden op de lijnen door de oorsprong, maar dit vergt het oplossen van een groot stelsel vergelijkingen, hetgeen rekenintensief is en met instabiliteiteiten

[2]Dit is zeker niet de enige mogelijkheid. Zo worden bijvoorbeeld ook waaiervormige bundels gebruikt bij röntgentomografie.

gepaard kan gaan. Een minder veeleisende methode is bepaling van de gewenste waarden door interpolatie tussen de bekende waarden van $F(u,v)$. Aangezien de dichtheid van deze radiële punten afneemt naarmate men verder van de oorsprong komt, zal de door deze interpolatie in de reconstructie van $f(x,y)$ geïntroduceerde fout groter zijn voor de hoogfrequente dan de laagfrequente componenten. Dit leidt tot degradatie van het beeld.

12.2.3 Gefilterde terugprojectie: theorie

Schrijf (12.10) in poolcoördinaten. Dit impliceert de substituties $u \rightarrow wcos\phi$, $v \rightarrow wsin\phi$ en $dudv \rightarrow |w|dwd\phi$, zodat

$$f(x,y) = \int_0^{\pi} d\phi \int_{-\infty}^{\infty} e^{j2\pi w(xcos\phi + ysin\phi)} P(\phi,w)\,|w|\,dw \tag{12.11}$$

We introduceren nu de functie

$$q(\phi,t) = \int_{-\infty}^{\infty} e^{j2\pi wt} P(\phi,w)\,|w|\,dw \tag{12.12}$$

Hiermee kan (12.11), met $t = xcos\phi + ysin\phi$, worden geschreven als

$$f(x,y) = \int_0^{\pi} q(\phi,t)\,d\phi \tag{12.13}$$

Aan deze uitdrukking kan een interessante interpretatie worden gegeven.
De fouriergetransformeerde van $q(\phi,t)$ is

$$Q(\phi,w) = \int_{-\infty}^{\infty} e^{-j2\pi wt}\,q(\phi,t)\,dt \tag{12.14}$$

Met de definite van $P(\phi,w)$ (12.7) volgt dan

$$Q(\phi,w) = |w|\,P(\phi,w) \tag{12.15}$$

dus $q(\phi,t)$ ontstaat uit de projectie $p(\phi,t)$ door **filtering** met een filter, waarvan de frequentieresponsie wordt gegeven door $H(w) = |w|$. Bij ieder waarde van ϕ wordt dus een gefilterde projectie homogeen verdeeld over de punten die oorspronkelijk aan de projectie $p(\phi,t)$ hebben bijgedragen. Men noemt dit wel **terugprojectie** (**backprojection**).

Hiermee is de term **gefilterde terugprojectie** (**filtered backprojection**) verklaard. Deze zal later in ruimere zin worden gebruikt voor technieken waar andere filters, dus $H(w) \neq |w|$, worden toegepast.

12.2.3.1 Algoritme

Noem $\tilde{f}(x,y)$ de reconstructie van $f(x,y)$ (in de praktijk zijn x en y discreet). Verwerking van de voorgaande formules kan dan schematisch als volgt worden weergegeven:

1. Initialiseer $\tilde{f}(x,y) = 0$
2. FOR n_ϕ hoeken ϕ DO
3. Scan object en maak projectie voor alle (x,y): $f(x,y) \rightarrow p(\phi,t)$
   met index $t = x * cos(\phi) + y * sin(\phi)$
4. Filter projectie: $p(\phi,t) \rightarrow q(\phi,t)$
5. Voer terugprojectie uit:
   FOR alle (x,y) DO
   $\tilde{f}(x,y) = \tilde{f}(x,y) + q(\phi, x * cos\phi + y * sin\phi)$
   END DO
   END DO
6. Normeer: $\tilde{f}(x,y) = \frac{1}{n_\phi}\tilde{f}(x,y)$

12.2.4 Terugprojectie algoritmes

De filterstap neemt een sleutelpositie in bij de implementatie van gefilterde terugprojectie als methode voor beeldreconstructie. Dit wordt vooral bepaald door afwegingen m.b.t. rekentijd en acceptabele beeldvervorming. In deze paragraaf zullen we enkele van de meest gangbare methodes behandelen.

12.2.4.1 Ongefilterde terugprojectie

Een snelle methode, waarbij de filterstap wordt overgeslagen, is directe terugprojectie van $p(\phi, t)$ i.p.v. $q(\phi, t)$. Dit betekent in het frequentiedomein dat we $P(\phi, w)$ i.p.v. $|w|P(\phi, w)$ gebruiken (zie (12.15)), dus effectief is het beeld dan gefilterd met $|w|^{-1}$. De hoogfrequente componenten worden dus onderdrukt, m.a.w. het beeld vervaagt ("blurring").

12.2.4.2 Iteratieve terugprojectie

Een variant op de ongefilterde terugprojectie is de iteratieve reconstructiemethode. Deze hoort eigenlijk thuis in een klasse van **algebraïsche iteratieve reconstructie** algoritmen. Een eenvoudige representant is de volgende. Voor iedere projectie (dus waarde van ϕ) wordt een **voorspelling** $\tilde{p}(\phi, t)$ op basis van het reeds gedeeltelijk gereconstrueerde beeld $\tilde{f}(x, y)$ berekend. Vervolgens wordt het verschil $\{p(\phi, t) - \tilde{p}(\phi, t)\}$ teruggeprojecteerd. Het zal duidelijk zijn dat dit proces stopt als $p(\phi, t) = \tilde{p}(\phi, t)$. Over de **convergentie** hiervan doet deze summiere beschrijving echter geen uitspraak.

12.2.4.3 Gefilterde terugprojectie

In §12.2.3 hebben we laten zien dat $f(x, y)$ eenduidig wordt bepaald door de gefilterde projecties $q(\phi, t)$. We schrijven dit filterproces (12.15) nu algemener op als

$$Q(\phi, w) = H(w)\, P(\phi, w) \tag{12.16}$$

waarbij $H(w)$ niet noodzakelijkerwijs gelijk is aan $|w|$. In het plaatsdomein correspondeert dit product met een convolutie-integraal

$$q(\phi, t) = h(t) \otimes p(\phi, t) \equiv \int_{-\infty}^{+\infty} h(t - \tau)\, p(\phi, \tau)\, d\tau \tag{12.17}$$

met

$$h(t) = \int_{-\infty}^{\infty} e^{j2\pi wt} H(w)\, dw \tag{12.18}$$

Er staan nu twee wegen open voor de berekening van $q(\phi, t)$, namelijk via inverse fouriertransformatie van $Q(\phi, w)$ of via convolutie.

12.2.4.4 Gefilterde terugprojectie: Convolutiemethode

Om $q(\phi, t)$ via het convolutievoorschrift (12.17) te kunnen berekenen moet $h(t) = \mathcal{F}^{-1}\{H(w)\}$ (12.18) bekend zijn. Omdat $\mathcal{F}^{-1}\{|w|\}$ niet bestaat, moet een andere filterfunctie $H(w)$ worden gekozen. Dit kan op de volgende wijze. Als $f(x, y)$ geen frequentiecomponenten heeft buiten het interval $[-W, W]$, dan levert (12.16) **nul** voor $|w| > W$, onafhankelijk van de vorm van $H(w)$. We kunnen dan zonder verlies van informatie het volgende filter gebruiken:

$$\begin{aligned} H(w) &= |w| \quad |w| \le W \\ &= 0 \quad |w| > W \end{aligned} \tag{12.19}$$

$\mathcal{F}^{-1}\{H(w)\}$ kan nu wèl worden berekend. Omdat geldt $H(w) = H(-w)$, levert de sinusterm van $e^{j2\pi wt} = cos2\pi wt + jsin2\pi wt$ in (12.18) nul, zodat

$$h(t) = 2 \int_0^W w \cos w\, dw \tag{12.20}$$

Partiële integratie leidt tot de volgende uitdrukking:

$$h(t) = W^2 \left[\, 2\, sinc(2\pi Wt) - sinc^2(\pi Wt) \right] \tag{12.21}$$

met

$$sinc(\xi) \equiv \frac{sin\xi}{\xi} \tag{12.22}$$

Het bemonsteringstheorema van Shannon zegt ons dat, omdat $p(\phi, t)$ een beperkte bandbreedte heeft in het w-domein, deze functie **volledig** gereconstrueerd kan worden uit samples in het t-domein op een onderlinge afstand die niet groter is dan $1/2W$. We kiezen deze kritische waarde

$$\Delta = \frac{1}{2W} \tag{12.23}$$

als stapgrootte in de discrete versie van de convolutie-integraal (12.17)

$$q(\phi, k\Delta) = \sum_{n=0}^{N-1} h((\mid k-n \mid)\Delta)\; p(\phi, n\Delta) \tag{12.24}$$

Hierin is N het totale aantal monsterpunten.
De filterfunctie $h(t)$ (12.21) in de monsterpunten heeft nu drie mogelijke waarden:

$$\begin{array}{rcll} h(n\Delta) & = & \frac{1}{4\Delta^2} & n = 0 \\ & = & 0 & n = even \\ & = & -\frac{1}{n^2\pi^2\Delta^2} & n = oneven \end{array} \tag{12.25}$$

De N filterwaarden $h(n\Delta)$ hoeven voor het volledige reconstructieproces slechts één maal berekend te worden.

Hiermee hebben we de ingrediënten voor een computerimplementatie van de convolutiemethode geformuleerd. Voor het uitrekenen van $q(\phi, k\Delta)$ met (12.24) zijn N vermenigvuldigingen nodig, en dit moeten we uitvoeren voor $k = 1, \ldots, N$. De rekentijd van deze methode schaalt dus met N^2.

12.2.4.5 Gefilterde terugprojectie: Fouriermethode

De berekening van $q(\phi, t)$ uit $Q(\phi, w)$ via (12.16) vereist als tijdsbepalende stappen fouriertransformatie van $p(\phi, t)$, en inverse transformatie van $Q(\phi, w)$. We beschrijven hier beknopt hoe dit geïmplementeerd kan worden. Met de hierboven geintroduceerde bemonsterde waarden van $h(t)$, $p(\phi, t)$ en $q(\phi, t)$

$$h_k \equiv h(k\Delta) \tag{12.26}$$
$$p_k \equiv p(\phi, k\Delta) \quad k = 0, \ldots, N-1 \tag{12.27}$$
$$q_k \equiv q(\phi, k\Delta) \tag{12.28}$$

corresponderen equidistante waarden van hun fouriergetransformeerden in het frequentiedomein

$$H_n \equiv H(n/N\Delta) \tag{12.29}$$
$$P_n \equiv P(\phi, n/N\Delta) \quad n = -N/2, \ldots, N/2 \tag{12.30}$$
$$Q_n \equiv Q(\phi, n/N\Delta) \tag{12.31}$$

Dit bestrijkt de volledige relevante frequentieband

$$[-W\ ,\ W] = [-1/2\Delta\ ,\ 1/2\Delta] \tag{12.32}$$

Hiermee kan als volgt een discrete fouriertransformatie geformuleerd worden:

$$h(t) = \int_{-\infty}^{\infty} e^{j2\pi wt} H(w)\, dw \implies h_k = \sum_{n=0}^{N-1} H_n\, e^{j2\pi kn/N} \tag{12.33}$$

$$H(w) = \int_{-\infty}^{\infty} e^{-j2\pi wt} h(t)\, dt \implies H_n = \frac{1}{N} \sum_{k=0}^{N-1} h_k\, e^{-j2\pi kn/N} \tag{12.34}$$

en analoog voor de andere functies.
Omdat

$$e^{-j2\pi kn/N} = \left(e^{-j2\pi/N}\right)^{kn} \qquad k, n = 0, \ldots, N-1 \tag{12.35}$$

lijkt deze procedure opnieuw N^2 vermenigvuldigingen te vragen. Het blijkt echter mogelijk te zijn om uit de array $[h_0, \ldots, h_{N-1}]$ de array $[H_0, \ldots, H_{N-1}]$ (en omgekeerd) te bepalen met $Nlog_2N$ vermenigvuldigingen. Deze methode, waarvan verschillende practische implementaties bestaan, draagt de naam **fast fourier transform** (FFT). We zullen hier niet op het **FFT algoritme** ingaan, maar een dergelijke routine als "black box" gebruiken tijdens het practicum.
De filterstap in het gefilterde terugprojectie algoritme (§12.2.3) kan schematisch als volgt worden weergegeven.

1. $[p_0, \ldots, p_{N-1}] \xrightarrow{FFT} [P_0, \ldots, P_{N-1}]$
2. $Q_n = H_n\, P_n \quad (n = 0, \ldots, N-1)$
3. $[Q_0, \ldots, Q_{N-1}] \xrightarrow{FFT^{-1}} [q_0, \ldots, q_{N-1}]$

We kunnen concluderen dat door de beschikbaarheid van FFT de fouriermethode voor het berekenen van $q(\phi, t)$ superieur is t.o.v. directe convolutie, omdat de rekentijd globaal evenredig is met $Nlog_2N \ll N^2$.

Hoofdstuk 13

Visualiseren

13.1 Inleiding

In het vorige hoofdstuk hebben we kennis gemaakt met gefilterde terugprojectie, een typische representant van methoden voor beeldreconstructie uit projecties. Vooral het principe is naar voren gehaald, met als opmerkelijk centraal thema dat in theorie inderdaad een volledige reconstructie uit projecties mogelijk is. Ook is over practische implementatie gesproken, maar nabewerking van het beeld (om onvolkomenheden te corrigeren) en visualiseren zijn echter nauwelijks ter sprake gekomen. Het beeld komt tot stand door filteren van de projecties in een aantal richtingen. Dit levert beeldinformatie langs radiële lijnen in het xy-vlak, dus op niet-uniforme wijze verdeeld over het vierkante rooster van pixels. Dit maakt, dat beeldbewerkingstechnieken toegepast op op deze wijze gereconstrueerde beelden een minder duidelijke samenhang met de fysische realiteit te zien geven dan het geval kan zijn bij directe afbeeldingsmethoden.

We introduceren daarom e.e.a. tegen de achtergrond van een directere beeldvormingstechniek: **scanning tunneling microscopie** (STM). [1] Een dunne naald voert over het oppervlak van een stuk materiaal een translatiebeweging uit. De afstand van de naald tot het oppervlak wordt tijdens zo'n scan constant gehouden. Op deze wijze wordt via de naaldhoogte $z(x, y)$ de topografie van het oppervlak in kaart gebracht, met een resolutie loodrecht op het oppervlak welke van de orde is van typisch 1% van een atomaire diameter.

We merken hier direct met nadruk op, dat de beeldbewerkingsmethodes die we in dit hoofdstuk bespreken in principe onafhankelijk zijn van de wijze waarop het beeld tot stand is gekomen: STM, röntgentomografie, optische microscopie, etc... Men zal zich in de praktijk echter moeten laten leiden door de specifieke kenmerken van de afbeeldingstechniek, omdat deze uiteindelijk bepalen welk type bewerking zinvol is. Het "blind" uitvoeren van beeldbewerking is zeer gevaarlijk en kan artefacten opleveren, die aanleiding kunnen geven tot een onjuiste interpretatie van de meetgegevens.

13.2 Scanning tunneling microscopie

13.2.1 Fysisch principe

De onderliggende fysische basis van STM is electron tunneling. Dit treedt op tussen twee geleiders (electroden) gescheiden door een zeer dunne, maar eindige, isolerende laag, in fysische termen een potentiaalbarrière. Dit kan een lege ruimte zijn of een laagje isolerend materiaal. De tunnelstroom I is een maat voor de overlap van de golffuncies van de twee electroden in deze scheidingslaag. Een veel voorkomende **tunneljunctie** is een vlakke geleider-isolator-geleider-"sandwich", waarin de isolator gewoonlijk een oxidelaag is, gevormd door oxidate van één van de electrodes. Als ϕ de hoogte is van de potentiaalbarrière tussen de electrodes, dan is bij een kleine

[1] De afkorting STM wordt zowel voor scanning tunneling microscopie (de techniek) als de scanning tunneling microscoop (het apparaat) gebruikt. Welke wordt bedoeld zal duidelijk zijn uit de context.

opgelegde spanning $V \ll \phi$ de stroomdichtheid gelijk aan

$$j = \left(\frac{e^2}{\hbar}\right) \left(\frac{\kappa_0}{4\pi^2 s}\right) V\, e^{-2\kappa_0 s} \tag{13.1}$$

met s de effectieve tunnelafstand in **A**, κ_0 de inverse vervallengte (**decay length**) van de golffunctiedichtheid buiten het oppervlak, V de opgelegde spanning in Volt en $\left(e^2/\hbar\right) = 2.44 \cdot 10^{-4}\,\Omega^{-1}$. κ_0 wordt gegeven door de uitdrukking $2\kappa_0\ (\mathbf{A}) = 1.025\sqrt{\phi(eV)}$.

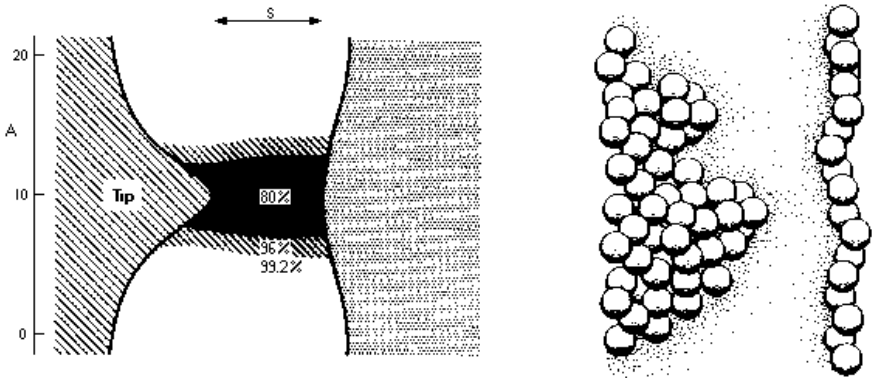


Figure 13.1: *Scanning tunneling microscopie.*

Bij STM is één van de electrodes het te onderzoeken materiaaloppervlak, terwijl de andere een scherpe naald (**tip**) is. Hoewel deze geometrie complexer is dan in de "sandwich" situatie, wordt de essentie van het tunneling proces nog steeds door (13.1) gegeven: s is nu de afstand van de tip tot het oppervlak. Deze komt zeer sterk tot uiting in de tunnelstroom: een verandering van s met 1 **A** resulteert al in een verandering van j met een **orde van grootte**. Dit is de basis voor de grote verticale resolutie van de STM, welke van de orde van 0.01**A** kan zijn. De tunnelstroom loopt door een smal "kanaal" met een doorsnede L_{eff}, die zeer klein kan worden voor een scherpe tip (tot atomaire afmeting). Opnieuw komt dit voort uit de sterke s-gevoeligheid van j (13.1). Dus als de tip nauwkeurig gepositioneerd kan worden, dan is in principe een **laterale** resolutie (xy-vlak) van de orde van atomaire afmetingen mogelijk. Deze resolutie is minder dan in verticale richting, omdat de tunnelstroom afkomstig is van een eindig oppervlakte element $\sim L_{eff}^2$.

Op basis van het hier beschreven principe kan het oppervlak in kaart gebracht worden. Een typische methode is die, waarbij gedurende laterale scans de tunnelstroom, dus ook s, constant wordt gehouden door de tippositie in verticale richting te corrigeren. Dit geeft dan het oppervlakte profiel $z(x, y)$. [2]

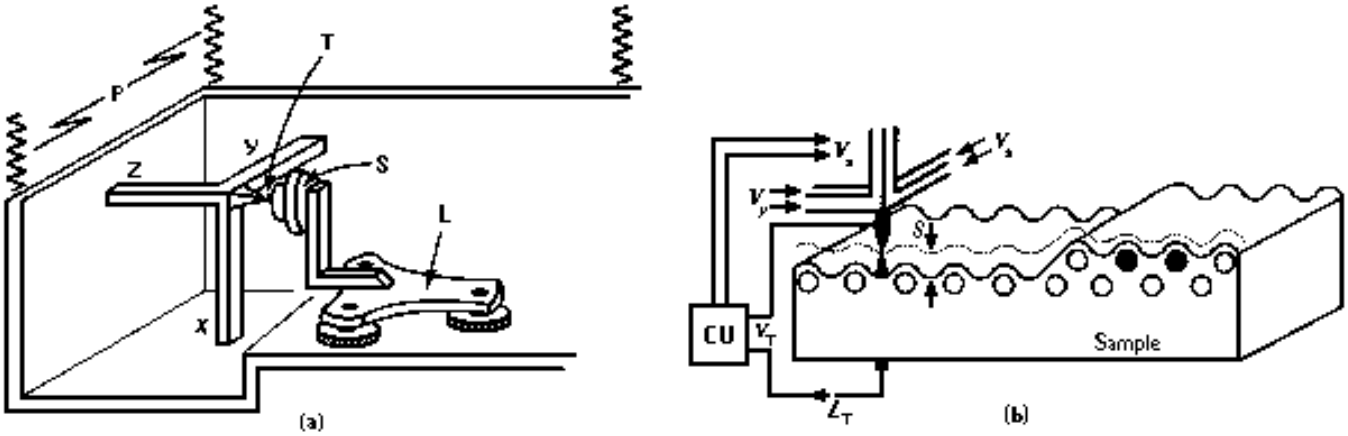


Figure 13.2: *Scanning tunneling microscoop.*

[2] Dit is de **constant tunneling current mode**. Het is ook mogelijk de verticale tippositie constant te houden en dan tijdens de scan de stroom te meten (**current imaging mode**). Deze methode is relatief snel (geen terugkoppeling van de tunnelstroom), maar is alleen toepasbaar bij zeer gladde oppervlakken.

13.2.2 Practische uitvoering

In figuur 13.2 is een scanning tunneling microscoop schematisch weergegeven. Dit maakt het principe duidelijk, maar er zijn vele varianten in omloop en de ontwikkelingen op dit gebied zijn nog steeds in volle gang:

(a) De tip T van de microscoop wordt over het oppervlak van het sample S bewogen met een piezo-electrische driepoot (X, Y, Z). De grove positieregelaar L brengt het sample binnen het bereik van de driepoot. Een vibratie-filtersysteem P schermt het instrument af van externe trillingen.

(b) Door de regeleenheid (**control unit**) CU wordt een spanning V_z opgelegd aan het Z piezo-electrische element, zodanig dat de tunnelstroom constant blijft terwijl de tip door verandering van V_x en V_y over het oppervlak beweegt. Het spoor van de tip, een y-scan, lijkt in het algemeen op de oppervlaktetopografie (stippellijn). Electronische onzuiverheden leveren ook structuur in het tipspoor, zoals in het rechterdeel is geïllustreerd boven twee atomen met negatieve lading.

13.3 Beeldweergave

13.3.1 Overzicht van STM visualiseringstechnieken

STM-data worden verzameld door parallel aan de scanrichting de naaldpositie met stapjes Δy te verplaatsen, en op ieder van deze posities wordt $z(x, y)$ bepaald (zie boven). Dergelijke scans worden herhaald op onderlinge afstand Δx. Dit levert uiteindelijk een verzameling z-waarden op een discreet rechthoekig rooster. Er zijn diverse methodes om $z(x, y)$ weer te geven.

Lijnplot (pseudo 3D): Punten verzameld tijdens één y-scan bij vaste x worden met elkaar verbonden. Dit levert een doorsnede van het oppervlak met een vlak x=constant. Door dergelijke scans verschoven t.o.v. elkaar te plaatsen (stacked plots) ontstaat pseudoperspectief. Hidden line removal maakt het beeld realistischer.

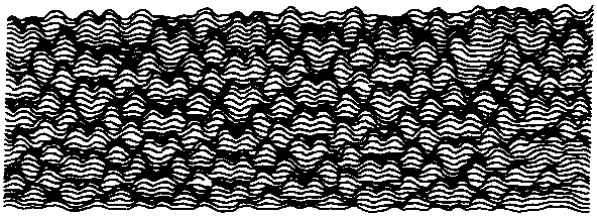


Figure 13.3: *STM lijnplot.*

Topview (2D): Bovenaanzicht van het oppervlak levert een rechthoekig netwerk van pixels. Aan iedere waarde $z(x, y)$, dus aan iedere pixel, wordt een grijswaarde of kleur (eventueel met variabele intensiteit) toegekend. Een alternatieve wijze van weergeven is met **hoogtelijnen**, waarbij punten met eenzelfde waarde van $z(x, y)$ met elkaar verbonden worden.

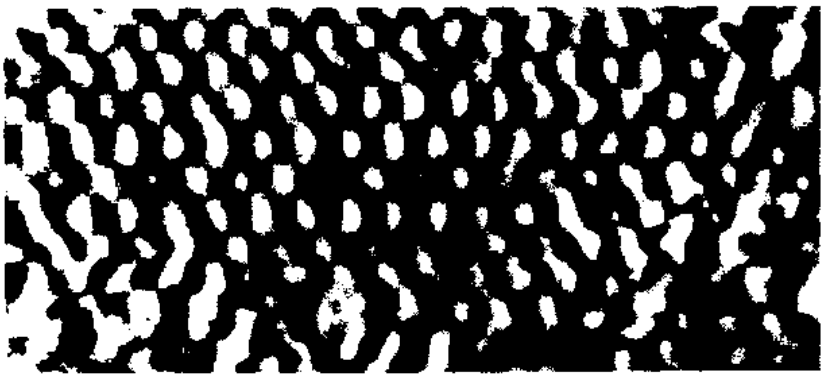


Figure 13.4: *STM topview.*

3D: Door de samplepunten zowel in x- als in y-richting met elkaar te verbinden ontstaat een **wire frame**. Dit kan worden geïnterpreteerd als een netwerk van veelvlakken op een 3D oppervlak. Typische grafische technieken die hierop kunnen worden toegepast zijn:
- Hidden surface removal.
- Kleur en intensiteit van het oppervlak als functie van z.
- Belichting; Gouraud of Phong shading.

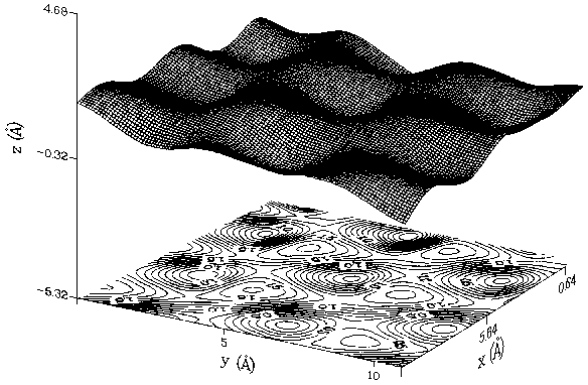


Figure 13.5: *STM wireframe inclusief hoogtelijnen.*

13.3.2 Toekenning van grijswaarden en kleuren

Uit de voorgaande beknopte opsomming lichten we één aspect voor een nadere beschouwing: het associeren van een grijswaarde of kleur met een bepaalde waarde van $z(x, y)$ [3]. Dit is vooral van belang voor **topview**, maar kan ook bij een 3D weergave een rol spelen. Hiertoe introduceren we de weergavefunctie $f(x, y)$, welke een bepaalde grijswaarde of kleur representeert:

$$z(x, y) \rightarrow f(x, y)$$

Vaste drempelwaarden

Als voorbeeld stellen we, dat $z(x, y)$ is gedigitaliseerd op 256 niveau's ($0 \rightarrow 255$), dus dit vraagt aan geheugen één byte (8 bits) per pixel.

a. Zwart-wit: Kies een drempelwaarde Z, bijvoorbeeld het midden van het bereik van z ($Z = 127$), en ken aan $f(x, y)$ een waarde toe, afhankelijk van de waarde van $z(x, y)$ t.o.v. deze drempel:

$$\begin{array}{ll} z(x,y) \leq Z \rightarrow f(x,y) = 0 & \text{Zwart} \\ z(x,y) > Z \rightarrow f(x,y) = 1 & \text{Wit} \end{array}$$

b. 4 grijswaarden of kleuren: Deel het bereik van z op in vier gebieden. Leg de grenzen bijvoorbeeld op vaste afstanden van elkaar. We definiëren zo vier segmenten, ieder gekarakteriseerd door een grijswaarde of kleur:

$$\begin{array}{rlll} z(x,y) < 64 & \rightarrow f(x,y) = 0 & \text{Zwart} & \text{Zwart} \\ 64 \leq z(x,y) < 128 & \rightarrow f(x,y) = 1 & \text{Donkergrijs} & \text{Rood} \\ 128 \leq z(x,y) < 192 & \rightarrow f(x,y) = 2 & \text{Lichtgrijs} & \text{Groen} \\ 192 \leq z(x,y) \leq 255 & \rightarrow f(x,y) = 3 & \text{Wit} & \text{Geel} \end{array}$$

Een dergelijke weergave met vaste drempelwaarden heeft als nadeel, dat vooral bij contrastarme beelden veel informatie verloren gaat. Nuanceverschillen *binnen* een segment verdwijnen hier volledig! Een oplossing voor dit probleem is het werken met variabele (random) drempelwaarden.

[3] De illustraties zijn geen STM-plaatjes

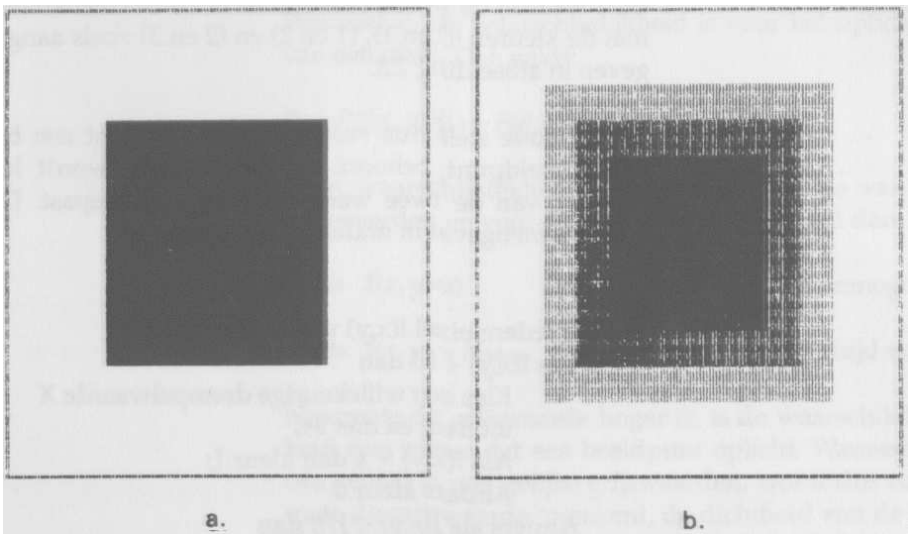


Figure 13.6: *Vaste drempelwaarden.*

Random drempelwaarden

De segmentatie in grijswaarden of kleuren welke hierboven beschreven is, blijft gehandhaafd, maar een pixel dat op grond van de waarde $z(x, y)$ recht heeft op een bepaald waarde van $f(x, y)$ krijgt deze waarde slechts toegekend met een waarschijnlijkheid, welke groter is naarmate $z(x, y)$ hoger in het segment ligt. Geef met $f_0, f_1, \ldots$ grijs-/kleurwaarden aan in opeenvolgende segmenten. We kunnen deze procedure dan alsvolgt weergeven:

1. Stel vast in welk segment $z(x, y)$ thuishoort: $z(x, y) \in (Z_{min}, Z_{max}) \equiv$ segment k.
2. Kies een random waarde $Z_R \in (Z_{min}, Z_{max})$.
3. IF $z(x, y) > Z_R$ THEN
   $f(x, y) = f_k$
   ELSE
   IF $k = 0$ THEN
   $f(x, y) = f_0$
   ELSE
   $f(x, y) = f_{k-1}$
   ENDIF
   ENDIF

Voorbeeld (4 kleuren): Stel $z(x, y) = 144$. Deze waarde correspondeert met het groene segment en ligt op 25% van de segmentlengte boven de ondergrens ($Z_{min} = 128$). Z_R zal dus met 75% waarschijnlijkheid **groter** zijn dan $z(x, y)$. Als dit laatste het geval is, wordt de kleur van één segment lager (dus rood) toegekend. Zoniet, dan krijgt de pixel de kleur groen.

Bovenstaande figuur geeft de random drempel varianten van (a) ($\rightarrow$ c) en (b) ($\rightarrow$ d).

Enerzijds leidt deze manier van weergeven tot een rijker geschakeerd beeld, maar anderzijds zullen grenzen tussen domeinen met een groot verschil van $z(x, y)$, welke aanvankelijk scherp waren, vervagen.

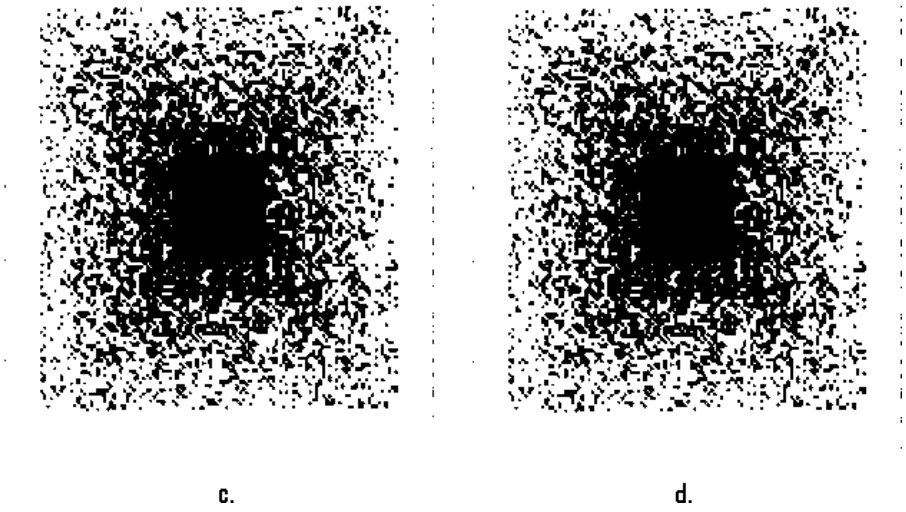


Figure 13.7: *Random drempelwaarden.*

Hoofdstuk 14

Beeldverbetering

In dit hoofdstuk worden een aantal technieken beschreven om de kwaliteit van een beeld in verschillende opzichten te verbeteren. De algemene kwaliteit van een beeld kan bijvoorbeeld verbeterd worden door vergroten van contrast of door verminderen van ruis. Maar het kan ook gewenst zijn om specifieke details van een beeld naar voren te halen, zoals bijvoorbeeld contouren. Of er kunnen achteraf correcties op toegepaste beeldbewerkingstechnieken toegepast worden om het effect van artefacten daarvan te verminderen.

14.1 Vergroten van contrast d.m.v. histogramtechnieken

In het vorige hoofdstuk zijn enkele elementaire methodes besproken om de waarde $z(x, y)$ om te zetten in visuele informatie (grijswaarde, kleur). Om geringe verschillen in z beter tot uitdrukking te brengen kan het **contrast** van het beeld worden vergroot door de toegekende grijswaarden/kleuren te transformeren naar andere waarden.

14.1.1 Lookup Table (LUT)

Bij gebruik van een **lookup table** worden de oorspronkelijke grijswaarden of kleuren afgebeeld op nieuwe waarden, waardoor het contrast verhoogd wordt. Deze techniek wordt hier besproken aan de hand van *grijswaarden*. Als notatie wordt met index B een oorspronkelijke beginwaarde en met index E een nieuwe eindwaarde aangeduid.

G_B = verzameling grijswaarden vóór transformatie $[g_B = 0, 1, \ldots, N_B]$
G_E = verzameling grijswaarden ná transformatie $[g_E = 0, 1, \ldots, N_E]$

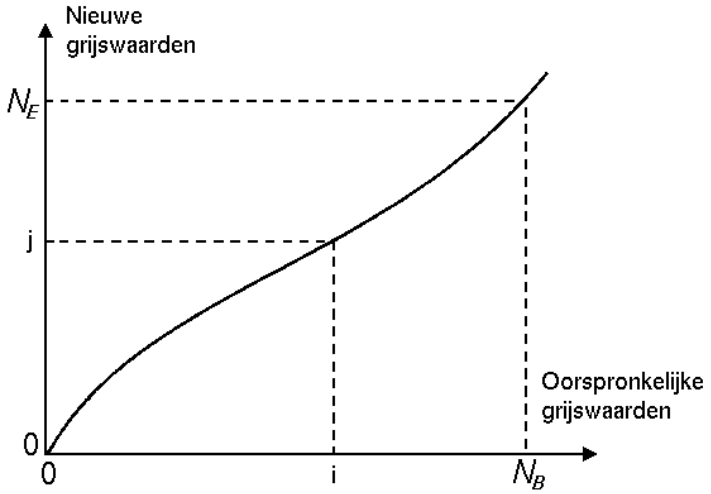


Figure 14.1: *Lookup table.*

Een grijswaarde g_B wordt afgebeeld op een nieuwe grijswaarde g_E volgens een voorschrift $\mathcal{L}$. De toekenning van een grijswaarde aan een z-waarde verloopt dus alsvolgt:

$$z \longrightarrow g_B \xrightarrow{\mathcal{L}} g_E$$

Aangezien g_B slechts een discreet aantal mogelijke waarden heeft, kan deze afbeeldingsinformatie worden opgeslagen in een tabel (array) L. $L[i] = j$ impliceert, dat grijswaarde $g_B = i$ wordt vertaald naar een grijswaarde $g_E = j$. De oorspronkelijke waarde g_B is dus een **verwijzing** naar een werkelijke waarde in de tabel L. Daarom spreekt men van een **Lookup Table** (LUT), of in geval van kleuren soms van een **Colour Lookup Table** (CLUT). L kan gevuld worden door voor $\mathcal{L}$ een functie te kiezen zoals *exp*, *log*, een polynoom etc., maar de toekenning kan ook op andere wijze tot stand komen.

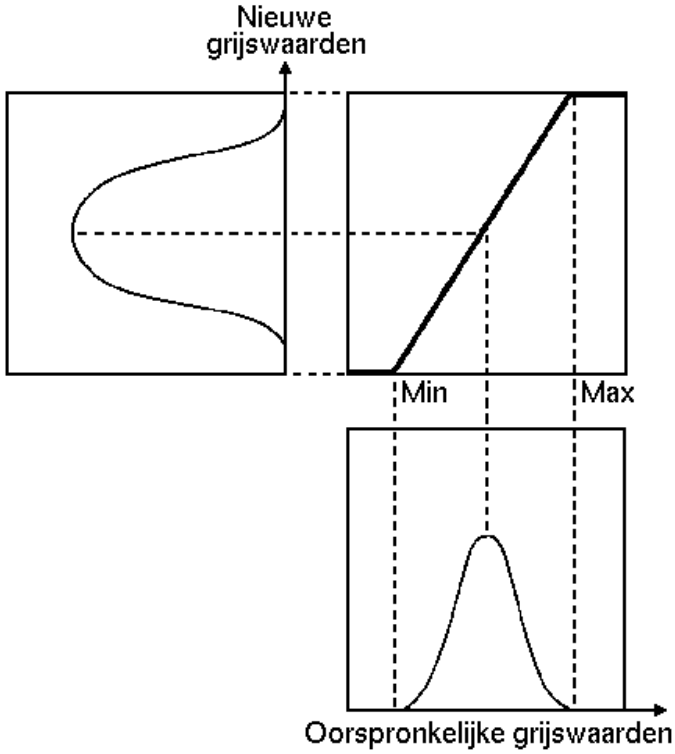


Figure 14.2: *Dynamiek vergroten met behulp van een lookup table.*

Een belangrijke toepassing van de LUT-methode is het vergroten van de dynamiek van een afbeelding. Als een beeld weinig dynamiek heeft (te licht, te donker, te weinig contrast), dan is dat in een **histogram** van de grijswaardenverdeling zichtbaar als een sterke concentratie in een beperkt deel van het totale bereik van de grijswaarden. Via een geschikt gekozen LUT kan deze piek in het histogram verbreed en/of verplaatst worden.[1]

14.1.2 Histogramegalisatie

Bij **histogramegalisatie** worden weinig voorkomende en wat waarde betreft dicht bij elkaar liggende kleuren samen genomen waardoor de verschillende waarden per waarde ongeveer in gelijk aantal in de afbeelding voorkomen.

Beschouw een beeld dat bestaat uit n_p pixels en dat is gecodeerd met N_B grijswaarden. De verdeling hiervan kan worden gerepresenteerd door een histogram met N_B categorieën. Definieer een tweede histogram met N_E categorieën ($N_E < N_B$).

In het oude histogram worden nu **cumulatieve** categoriën gevormd met een aantal pixels dat zo dicht mogelijk ligt bij n_p/N_E: het **gemiddelde** aantal per **nieuwe** categorie. Aan alle pixels binnen één cumulatieve categorie wordt nu eenzelfde nieuwe grijswaarde toegekend. De pixels worden dus gelijkmatig over N_E grijswaarden

[1] De keuze van het afdrukpapier (hard/zacht) in zwart-wit fotografie heeft hetzelfde doel, en kan in feite ook als een specifieke keuze van $\mathcal{L}$ worden gezien.

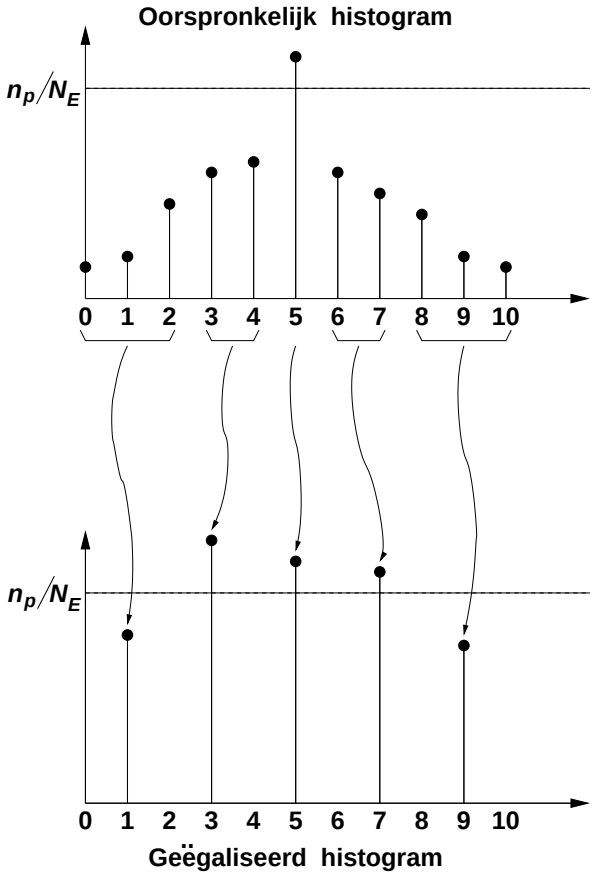


Figure 14.3: *Histogramegalisatie.*

verdeeld, m.a.w. in het nieuwe beeld is iedere grijswaarde (ongeveer) even sterk vertegenwoordigd, met als prijs dat het **aantal** grijswaarden N_E **kleiner** is geworden dan in het oorspronkelijke beeld. Hieronder is het effect op een beeld geïllustreerd.

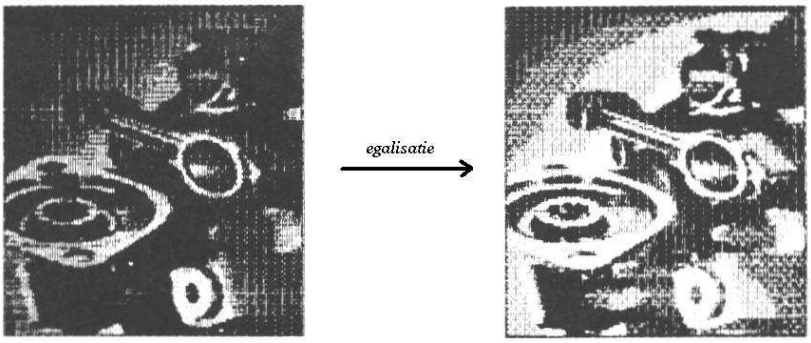


Figure 14.4: *Voorbeeld van histogramegalisatie.*

14.2 Verminderen van ruis d.m.v. filters

In de voorgaande paragraaf zijn technieken besproken waarbij de waarde van ieder pixel afzonderlijk wordt getransformeerd, onafhankelijk van de waarden van de buurpixels. Dergelijke methoden zullen echter weinig effect op ruis in het beeld hebben. Een locale uitschieter in intensiteit, die het gevolg is van een storing tijdens de beeldvorming, zal op deze wijze aanwezig blijven. Om dergelijke fluctuaties te verzwakken kunnen we het volgende argument gebruiken. Een beeld is een ruimtelijke verdeling van pixels die een zekere mate van coherentie

vertonen. Aan elkaar grenzende pixels zullen daarom vaak nauwelijks in waarde van elkaar verschillen, behalve natuurlijk bij scherpe overgangen in intensiteit. Het ligt daarom voor de hand om voor het verminderen van ruis de informatie in naburige pixels met elkaar te combineren. Dit wordt gedaan door toepassen van discrete convolutie in de vorm van **filters** in het plaatsdomein. Een filter geeft een relatie aan tussen een bepaald pixel en de naburige pixels die een nieuwe waarde voor het bepaalde pixel oplevert. Er kunnen twee soorten filters onderscheiden worden, **lineaire filters** en **niet-lineaire filters**.

In het frequentiedomein corresponderen scherpe pieken met hoogfrequente componenten. Het proces dat we op het beeld willen toepassen zal daarom filteren met een **laagdoorlaatfilter** moeten zijn. Een negatief effect van een dergelijk filterproces is dat er bij het wegwerken van de ruis ook details verloren gaan. We zullen daarom óók ingaan op methodes waarbij dit bijverschijnsel wordt verminderd, of nog sterker: waar juist scherpe intensiteitsovergangen benadrukt worden.

14.2.1 Lineaire filters

Bij lineaire filters wordt de nieuwe waarde van een pixel berekend uit een lineaire combinatie van eventueel gewogen waarden van omringende pixels.

Gebruik weer de indices B (begin) en E (eind) voor de intensiteitsverdeling vóór en ná de bewerking. We kunnen dan het volgende filterproces in ruimtelijke coördinaten definiëren:

$$I_E = H \otimes I_B \tag{14.1}$$

Dit is een convolutie-operatie: de waarden I_B in het oorspronkelijke beeld worden via het filter H (*masker*) verdeeld over pixels in de omgeving. Als dit masker $(n+1) \times (n+1)$ groot is, met n een even getal, dan kan dit convolutieproces als volgt worden weergegeven:

$$I_E(x,y) = \sum_{i=-n/2}^{n/2} \sum_{j=-n/2}^{n/2} H(i+\frac{n}{2}, j+\frac{n}{2})\, I_B(x+i, y+j) \tag{14.2}$$

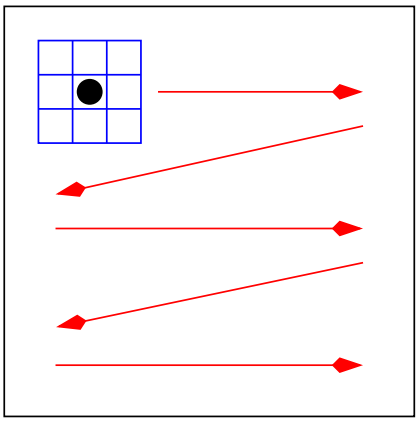


Figure 14.5: *Toepassen van een filter op een beeld.*

14.2.1.1 Voorbeelden van lineaire filters

Een eenvoudig voorbeeld hiervan is een 3×3 filter waarbij voor elk punt van het beeld het gemiddelde van de centrale pixel met de 8 buurpixels wordt genomen (**middelingfilter**). Het masker kan dan worden gerepresenteerd door de volgende matrix:

$$H_1 = \frac{1}{9} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \tag{14.3}$$

Een filter dat een groter gewicht aan het centrale pixel dan aan de buurpixels geeft – en daardoor een minder sterk resultaat oplevert dan dat met H_1 – is

$$H_2 = \frac{1}{10}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{pmatrix} \tag{14.4}$$

Meer gewicht aan de pixels in de x- en y-richting wordt gegeven door het filter

$$H_3 = \frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix} \tag{14.5}$$

14.2.1.2 Rand van het beeld

Maar wat gebeurt er met de rand? Want een $(n+1) \times (n+1)$ filter is langs de rand van het beeld in een kader met een breedte $n/2$ in eerste instantie niet toepasbaar. Stel dat het beeld $p \times p$ pixels groot is, dan zijn er verschillende oplossingen:

- Filter alleen het centrale deel van het beeld van $n/2$ tot $(p - n/2)$ in zowel x- als y-richting. Er blijft dan een bewerkt beeld met afmetingen $(p-n) \times (p-n)$ over met daaromheen een onbewerkte rand.
- Voeg om het beeld een rand toe met breedte $n/2$ en spiegel het beeld vervolgens t.o.v. de oorspronkelijke begrenzing (waarbij de hoekpunten wat extra aandacht vereisen). Het uiteindelijke beeld is dan $p \times p$ groot.
- Voeg om het beeld een rand toe met breedte $n/2$ en vul deze met nullen.

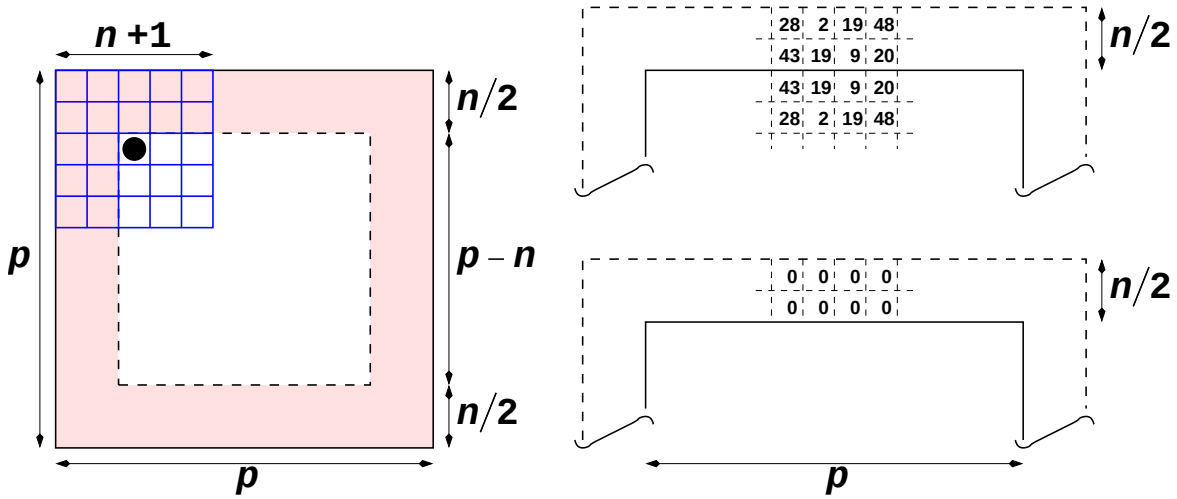


Figure 14.6: *Verschillende oplossingen voor het filteren van de rand van een beeld.*

14.2.1.3 Gaussfilter

We bespreken nu een Gaussfilter

$$\mathcal{G}_{xy}(x, y; \sigma) = \frac{1}{2\pi\sigma}\, exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \tag{14.6}$$

waarmee de filteroperatie wordt uitgevoerd volgens (14.1):

$$I_E = \mathcal{G}_{xy} \otimes I_B \tag{14.7}$$

Als we een $(n+1) \times (n+1)$ masker vullen met de (discrete) waarden van deze functie, dan dient n voldoende groot te zijn. Als n_σ het aantal pixels op een lengte σ is, dan is een typische waarde $n > 4n_\sigma$, zodat (minstens) 95 % onder de Gausscurve wordt meegenomen. Voor $n > 6n_\sigma$ is dit al 99.9 %.
Omdat de fouriergetransformeerde van een Gausscurve wéér een Gausscurve is, kan eenvoudig worden aangetoond, dat (14.7) kan worden geschreven als

$$I_E = \mathcal{G}_x \otimes \mathcal{G}_y \otimes I_B \tag{14.8}$$

met

$$\mathcal{G}_k(k;\sigma) = \frac{1}{\sqrt{2\pi}\sigma}\, exp\left(-\frac{k^2}{2\sigma^2}\right) \qquad k = x, y \tag{14.9}$$

De consequentie hiervan is, dat de oorspronkelijk convolutie met een $N \times N$ filter kan worden vervangen door twéé convoluties met een $N \times 1$ filter, respectievelijk in x- en y-richting. Het aantal vermenigvuldigingen en optellingen bij convolutie van een $p \times p$ beeld met een $N \times N$ filter is $\sim N^2\, p^2$. De splitsing van $\mathcal{G}_{xy}$ in $\mathcal{G}_x$ en $\mathcal{G}_y$ reduceert dit getal tot $2Np^2$, dus de winst is vermindering van de rekentijd met een factor $2/N$.

14.2.2 Niet-lineaire filters

Bij niet-lineair filteren is de nieuwe waarde van het centrale pixel dus geen lineaire combinatie van de waarden van de omringende pixels. Als voorbeeld wordthet mediaanfilter

14.2.2.1 Mediaanfilter

Lineair filteren heeft als nadeel, dat door het middelingsproces scherpe details verloren gaan. Een ander probleem is, dat als een enkele pixel door een storing zeer veel in intensiteit verschilt van zijn omgeving ("shot noise"), deze storing verspreid wordt, zodat in feite het beeld verslechtert. Een methode die dit soort problemen vermindert is **mediaanfiltering**. Hiertoe wordt als boven weer een masker toegepast, maar alle pixels binnen het masker worden nu **gesorteerd** op intensiteit. De waarde van de pixel die de **middelste** positie (de mediaan) van deze reeks inneemt wordt toegekend aan de centrale pixel waar het masker is gepositioneerd. Bijvoorbeeld in een 3×3 omgeving is het de 5^{de} pixel in de reeks, in een 5×5 omgeving de 13^{de}, enz.

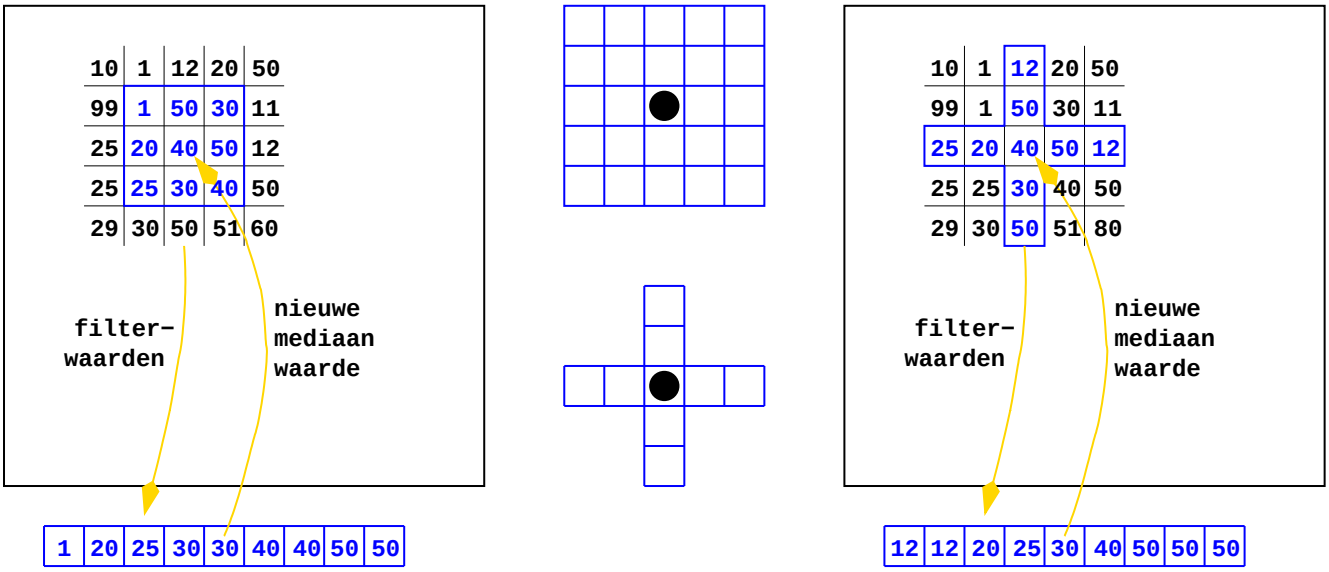


Figure 14.7: *Mediaanfilter met voorbeelden van toegepaste filters.*

14.2.3 Laagdoorlaatfilters

14.2.3.1 Butterworth laagdoorlaatfilter

We hebben tot nu toe het verwijderen van ongewenste scherpe details in het beeld besproken in termen van operaties op pixels in het plaatsdomein. We bespreken nu procedures, die in het frequentiedomein worden geformuleerd. We willen in principe een ideaal laagdoorlaatfilter, waarvan de karakteristiek wordt gegeven door

$$H(u,v) = \begin{cases} 0 & D(u,v) \le D_0 \\ 1 & D(u,v) > D_0 \end{cases} \tag{14.10}$$

Hierin zijn (u, v) de frequentievariabelen en $D(u, v)$ is de afstand van het punt (u, v) tot de oorsprong

$$D(u, v) = \left(u^2 + v^2\right)^{1/2} \tag{14.11}$$

We weten dat een dergelijke functie in het frequentiedomein correspondeert met een sinc-functie (d.i. van de vorm $sin\xi/\xi$). Hierdoor zal een punt in het plaatsdomein omgeven worden door concentrische cirkels ("ringing"). Bovendien zal de frequentie-inhoud van een werkelijk beeld niet abrupt bij D_0 endigen, zodat informatie verloren gaat. Een klasse filters dat een wat milder gedrag heeft rond de afsnijfrequentie zijn **Butterworth laagdoorlaatfilters** (Butterworth low pass filter of BLPF). Een BLPF van de orde n wordt gedefinieerd door

$$H(u, v) = \frac{1}{1 + [D(u, v)/D_0]^{2n}} \tag{14.12}$$

De steilheid van de afval van dit filter rond D_0 neemt toe met stijgende n.

14.3 Benadrukken van contouren d.m.v. filters

Soms is men geïnteresseerd in het benadrukken van de contouren van een beeld, ofwel de **overgangen** tussen gebieden met verschillende intensiteit. Dit kan zeer nuttig zijn voor het herkennen van structuren, en het speelt bijvoorbeeld een belangrijke rol bij **patroonherkennen** met behulp van computers.

Middeling van aangrenzende pixels in het plaatsdomein, zoals hierboven is besproken, kan worden gezien als **integratie** en dit werkt als **laagdoorlaatfilter**. Complementair hieraan gedraagt **differentiatie** zich juist als **hoogdoorlaatfilter**. We zullen nu enkele filters H formuleren die in deze klasse vallen.

14.3.1 Differentiërende filters

Uitgangspunt is een taylorontwikkeling van een functie $f(x)$:

$$f(x \pm \Delta x) = f(x) \pm \Delta x \frac{df}{dx} + \frac{(\Delta x)^2}{2} \frac{d^2 f}{dx^2} \pm \frac{(\Delta x)^3}{3!} \frac{d^3 f}{dx^3} + \frac{(\Delta x)^3}{4!} \frac{d^4 f}{dx^4} \pm \ldots \tag{14.13}$$

waarbij alle afgeleiden in het punt x moeten worden genomen. We nemen voor Δx de afmeting van een pixel. In de limiet voor zeer kleine Δx is dan

$$\frac{df}{dx} \approx \frac{f(x + \Delta x) - f(x)}{\Delta x} \quad \stackrel{notatie}{\rightarrow} \quad f(x + 1) - f(x) \tag{14.14}$$

dus het verschil van de waarde van f in twee opeenvolgende pixels. De tweede afgeleide verbindt drie pixels met elkaar volgens

$$\frac{d^2 f}{dx^2} \approx \frac{f(x + \Delta x) - 2f(x) + f(x - \Delta x)}{(\Delta x)^2} \quad \stackrel{notatie}{\rightarrow} \quad f(x + 1) - 2f(x) + f(x - 1) \tag{14.15}$$

Als $f(x, y)$ een functie van x en y is, dan volgt eenvoudig, dat nu de volgende uitdrukkingen van toepassing zijn:

$$\frac{\partial f(x, y)}{\partial x} \quad \rightarrow \quad f(x + 1, y) - f(x, y) \tag{14.16}$$

$$\frac{\partial f(x, y)}{\partial y} \quad \rightarrow \quad f(x, y + 1) - f(x, y) \tag{14.17}$$

$$\frac{\partial^2 f(x, y)}{\partial x^2} \quad \rightarrow \quad f(x + 1, y) - 2f(x, y) + f(x - 1, y) \tag{14.18}$$

$$\frac{\partial^2 f(x, y)}{\partial y^2} \quad \rightarrow \quad f(x, y + 1) - 2f(x, y) + f(x, y - 1) \tag{14.19}$$

Dit schrijven we in de vorm van filtermaskers (14.1).

14.3.1.1 Gradiëntfilters

Bij gradiëntfilters worden contouren in een specifieke richting benadrukt.

$$\frac{\partial}{\partial x} \rightarrow H_x^{(1)} = \begin{pmatrix} -1 & 1 \end{pmatrix} \tag{14.20}$$

$$\frac{\partial}{\partial y} \rightarrow H_y^{(1)} = \begin{pmatrix} -1 \\ 1 \end{pmatrix} \tag{14.21}$$

of, in matrixvorm

$$\frac{\partial}{\partial x} \rightarrow H_x^{(1)} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \tag{14.22}$$

$$\frac{\partial}{\partial y} \rightarrow H_y^{(1)} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \end{pmatrix} \tag{14.23}$$

14.3.1.2 Laplace filter

$$\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \rightarrow H^{(2)} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix} \tag{14.24}$$

Verscherping van de contouren met behoud van informatie buiten de verscherpte delen wordt bereikt door het Laplace gefilterde beeld van het oorspronkelijke af te trekken. Dit correspondeert met het volgende filtermasker:

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} - \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} \tag{14.25}$$

14.3.2 Hoogdoorlaatfilters

Een alternatief voor de hierboven beschreven verscherpingstechnieken is het gebruik van een **hoogdoorlaatfilter** in het frequentiedomein. Complementair aan hetgeen in paragraaf (14.2.3) is besproken introduceren we een ideaal hoogdoorlaatfilter

$$H(u,v) = \begin{cases} 1 & D(u,v) \leq D_0 \\ 0 & D(u,v) > D_0 \end{cases} \tag{14.26}$$

en als minder rigoureus alternatief een **Butterworth hoogdoorlaat filter** (Butterworth high pass filter; BHPF). Een BHPF van de orde n wordt gedefinieerd door

$$H(u,v) = \frac{1}{1 + [D_0/D(u,v)]^{2n}} \tag{14.27}$$

14.4 Beeldcorrectie

14.4.1 Onvolkomenheden bij STM beeldvorming

Het afbeelden van een oppervlakteprofiel met STM komt neer op het controleren/vastleggen van drie electrische spanningen, V_x, V_y en V_z, die via piezo-electrische elementen de ruimtelijke positie van de tip bepalen. In meer of mindere mate zullen in de electronische circuits **ruis**bijdragen δV_x, δV_y en δV_z aan deze spanningen worden toegevoegd. Hierin zijn twee typen met een sterk verschillend karakter te onderkennen: witte ruis en $1/f$-ruis (f is de frequentie). Witte ruis heeft een spectrum dat min of meer vlak is, terwijl $1/f$-ruis een frequentieafhankelijkheid van de vorm $1/f^\beta$, met typische waarden $\beta = 1.4 \pm 0.2$. $1/f$-ruis heeft dus vooral sterke **laagfrequente** componenten. Bij hoge frequenties is witte ruis van belang.

V_x en V_y zijn een maat voor de laterale positie (dus x en y) van de sampletip. Als de door een spanningsbron aangeboden spanningen V_x en V_y zijn, dan zal uiteindelijk het piezo-electrische element gestuurd worden door spanningen $(V_x + \delta V_x)$ en $(V_y + \delta V_y)$, hetgeen een foutieve positionering $(x + \delta x, y + \delta y)$ tot gevolg heeft. Als we $f(x, y)$ de **correcte** beeldinformatie noemen voor pixel (x,y) (dus in afwezigheid van storende invloeden), en $g(x, y)$ het verstoorde beeld, dan vindt dus ten onrechte de volgende toekenning plaats:

$$g(x, y) = f(x + \delta x, y + \delta y)$$

Als de fluctuaties in de spanning langzaam zijn t.o.v. de tijd dat de tip op een bepaalde positie staat, dan wordt het oppervlak dus afgetast langs sporen die **niet** parallel en equidistant zijn, maar enigszins vervormd. Naarmate de spanningsfluctuaties sneller zijn vindt **middeling** plaats van het oppervlakteprofiel over een gebied met typische afmeting $\delta x \times \delta y$, dus het werkelijke profiel wordt a.h.w. **uitgesmeerd** over een aantal pixels.
V_z is een maat voor de hoogte van de tip boven het oppervlak en deze wordt gecorrigeerd naar rato van de afwijking van de gemeten tunnelstroom t.o.v. een ingestelde waarde (constant current mode). Afwijkingen resulteren nu in een onjuiste toekenning

$$g(x, y) = f(x, y) + \eta(x, y]$$

waarbij de notatie $\eta(x, y)$ verwijst naar **noise**. Naarmate de ruis hoogfrequenter is, zal $n(x, y]$ meer uitgemiddeld worden ($\rightarrow 0$), maar dit is middeling **binnen** hetzelfde pixel.

Mechanische trillingen hebben vergelijkbare effecten op de beeldvorming. De technologie rond STM-trillingsdemping is echter inmiddels zover gevorderd dat dit (beslist niet-triviale probleem) van secundair belang is.

Een systematische bron van fouten komt voort uit de geometrie van de combinatie tip $\leftrightarrow$ sampleoppervlak.
• Als de tip zich boven een "dal" van het oppervlak bevindt, dan zullen ook de wanden hiervan een bijdrage aan de tunnelstroom leveren. Op een top van het oppervlak is de tunnelstroom veel sterker gelocaliseerd. Er vindt dus vooral uitsmering plaats van het oppervlakteprofiel in de dalen en dit effect is sterker naarmate het profiel scherpere variaties vertoont. M.a.w. dit effect spreekt vooral door in de hoogfrequente (ruimtelijke) componenten van het beeld.
• Omdat de punt van de tip een eindige straal r_t heeft is ook voor een volkomen vlak oppervlak de tunnelstroom afkomstig van een eindig gebied. Opnieuw wordt dus het werkelijke oppervlakteprofiel uitgesmeerd.
Voor een ruimtelijke fouriercomponent van het oppervlakteprofiel met golflengte a kan de fout Δs in de tip-sample afstand t.g.v. deze effecten worden samengevat in de formule

$$\frac{\Delta s}{h_s} = exp\left[-\frac{\pi^2(r_t + s)}{\kappa_0 a^2}\right] \tag{14.28}$$

waarbij Δs de verstoring van de tip-oppervlakte afstand is.

We hebben diverse factoren genoemd, die er toe leiden dat de beeldinformatie $f(x, y)$ wordt verstoord. Additieve ruis $\eta(x, y)$ leidt ertoe, dat de informatie binnen een pixel onjuist is, maar dit is onafhankelijk van informatie in de buurpixels. Ook vindt uitsmering van het oppervlakte profiel over diverse pixels plaats, met een zuiver statische (tip - oppervlakte geometrie) of dynamische (tipbeweging) oorzaak. We kunnen dit mathematische als volgt weergeven:

$$g(x, y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\alpha, \beta)\, h(x, \alpha, y, \beta)\, d\alpha d\beta \; + \eta(x, y) \tag{14.29}$$

$h(x, \alpha, y, \beta)$ wordt de *punt spreid functie* (PSF) genoemd. Deze naam wordt duidelijk, als we een beeld bestaand uit één punt, een δ-puls, beschouwen (stel $\eta(x, y) = 0$) :

$$g(x, y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \delta(x - \alpha, y - \beta)\, h(x, \alpha, y, \beta)\, d\alpha d\beta = h(x, x, y, y) \tag{14.30}$$

Dus $h(x, \alpha, y, \beta)$ geeft de bijdrage van een *punt* op positie (α, β) aan het beeld op positie (x, y). We nemen aan dat de PSF onafhankelijk is van de absolute positie (x, y), maar dat uitsluitend relatieve posities $(x - \alpha, y - \beta)$ van belang zijn. Dan kunnen we (14.29) schrijven als

$$g(x, y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\alpha, \beta)\, h(x - \alpha, y - \beta)\, d\alpha d\beta \; + \eta(x, y) \tag{14.31}$$

Hierin herkennen we de convolutie van f met h, dus we kunnen dit compact schrijven als

$$g(x,y) = f(x,y) \otimes h(x,y) + \eta(x,y) \tag{14.32}$$

In het frequentiedomein wordt dit

$$G(u,v) = F(u,v)H(u,v) + N(u,v) \tag{14.33}$$

Het probleem waar we voor staan is $f(x,y)$ te bepalen uit het gemeten beeld $g(x,y)$. Hierbij moet men zich realiseren dat $h(x,y)$ en $\eta(x,y)$ slechts bij benadering bekend zijn. Er staan twee wegen open: **direct** in het plaatsdomein via (14.32) of **indirect** door een fouriertransformatie uit te voeren naar het frequentiedomein en dan (14.33) te gebruiken.

Appendix A

Literatuurlijst

1. Foley, van Dam, Feiner, Hughes, Phillips,
   Introduction to Computer Graphics,
   Addison Wesley, 1993, ISBN 0-201-60921-5.

2. P.F. Klok en A.J. Dammers,
   Handleiding Computer Graphics,
   zie webpagina `http://www.hef.ru.nl/ pfk/teaching/icg/`.

3. F.S. Hill, Jr.,
   Computer Graphics Using OpenGL,
   Prentice Hall, 2000, ISBN 0-02-354856-8.

4. Foley, van Dam, Feiner, Hughes,
   Computer Graphics, Principles and Practice, second edition in C,
   Addison Wesley, 1996, ISBN 0-201-84840-6.

5. David F. Rogers, J. Alan Adams,
   Mathematical Elements for Computer Graphics, second edition,
   McGraw-Hill, 1990, ISBN 0-07-100289-8.

6. Ron Sommeling, Jos Alsters, Peter van Campen en Willem-Jan Karman,
   Gebruikershandleiding voor het MIT X venstersysteem,
   KU Nijmegen, 1990.

7. Brian W. Kernigan, Dennis M. Ritchie,
   The C Programming Language, 2nd ed.,
   Prentice Hall, 1988, ISBN 0-13-110362-8.

8. Al Keller en Ira Pohl,
   De Programmeertaal C, Grondbeginselen en Toepassingen, 2^e editie,
   Addison Wesley, 1990, ISBN 90 6789 207 6.

9. Tomasz Müldner en Peter W. Steele,
   C as a Second Language, For Native Speakers of Pascal,
   Addison Wesley, 1988, ISBN 0-201-19210-1.

10. Rafael C. Gonzalez en Richard E. Woods,
    Digital Image Processing, third edition,
    Addison Wesley, 1993, ISBN 0-201-50803-6.

11. Don Pearson, editor,
    Image Processing,
    McGraw-Hill, 1991, ISBN 0-07-707323-1.

12. J. Toumazet,
    Image Processing,
    Sybex, 1991, ISBN 90-5160-315-0.

13. W.H. Press et al.,
"Numerical Recipes in C, The Art of Scientific Computing",
Cambridge University Press, 1988, ISBN 0-521-35465-X.

14. C.P.F. Terheggen,
"UNIX voor Theoretische Hoge-energiefysica", 2e editie,
C&CZ, 1992.

Appendix B

Software en hardware

B.1 Algemeen

Informatie over facultaire ICT-voorzieningen (zoals hoe te gebruiken, waar te vinden, wie te benaderen bij problemen) is te vinden op de website

`http://www.hef.ru.nl/ic-hand/`

B.2 Software en hardware

Benodigde software om de opdrachten uit te kunnen voeren bestaat uit het grafische pakket OpenGL en de computertaal C.

Beide zijn als open source beschikbaar voor Unix/Linux systemen. OpenGL is gratis beschikbaar voor MS-Windows systemen.

Op de centrale studentencomputer "solo" (met Unix) kunnen de opdrachten gemaakt worden met behulp van van een Makefile die via de website te verkrijgen is.

Daarnaast kunnen pc's met Linux gebruikt worden. De programmeertaal C is daarop standaard aanwezig en een open source versie van OpenGL (Mesa) kan daarop geïnstalleerd worden.

Ook kunnen pc's met een MS-Windows systeem gebruikt worden, maar het ontbreken van een open source C compiler kan een probleem zijn. De OpenGL software is wel gratis beschikbaar.

Zie de bij dit vak behorende website voor meer informatie:

`http://www.hef.ru.nl/~pfk/teaching/icg/`

B.3 MS-Windows

Als OpenGL en C op de pc aanwezig zijn, kunnen de opdrachten op een pc met MS-Windows gemaakt worden. Zaak is wel om het gebruik van specifieke MS-Windows bibliotheken te vermijden, omdat de opdrachten onder Linux nagekeken worden.

B.4 Unix

Er zijn verschillende Unix systemen in gebruik, die allemaal een grote gemeenschappelijke basis hebben voor uitvoerbare commando's. Linux is een volwaardig Unix systeem dat o.a. op pc's draait.

Unix commando's bestaan uit de naam van het commando, eventueel gevolgd door vlaggen of switches, eventueel gevolgd door argumenten. De switches worden aangegeven door een min-teken gevolgd door de letter van de switch. In tegenstelling met de meeste andere besturingssystemen is Unix **case sensitive**, dat wil zeggen dat er onderscheid gemaakt wordt tussen hoofd- en kleine letters.

Hieronder volgt een lijstje met een aantal Unix commando's:

- informatie over een Unix commando vragen:
  man *commandonaam*
- informatie over Unix commando's vragen via een trefwoord:
  man -k *trefwoord*
- copiëren van bestanden:
  cp *oude_bestandsnaam nieuwe_bestandsnaam*
- verplaatsen van bestanden:
  mv *oude_bestandsnaam nieuwe_bestandsnaam*
- weggooien van bestanden:
  rm *bestandsnaam*
- lijst van bestanden in directory maken:
  ls
  ls -l
- inhoud van een bestand in een window afdrukken:
  cat *filenaam*
  more *filenaam*
- maken van een subdirectory:
  mkdir *subdirectorynaam*
- weggooien van een subdirectory:
  rmdir *subdirectorynaam*
- naar een (sub)directory gaan:
  cd *subdirectoryspecificatie*
- tijdelijk stoppen van een programma (met de bedoeling later weer door te gaan):
  CONTROL z
- voorgoed stoppen van een programma:
  CONTROL c
- verder gaan met een programma dat met CONTROL z gestopt is:
  fg
- herhalen van het laatstgegeven commando:
  !!
- herhalen van een eerder gegeven commando:
  !*beginletter(s) van commando*
- compileren en linken (binden) van een C-programma:
  make *programmanaam*

Appendix C

Korte beschrijving van C statements

Dit appendix bevat een zeer korte samenvatting van verschillende statements en constructies van de taal C. Voorafgaande hieraan een paar opmerkingen.

Een C-programma heeft de volgende structuur:

```
#include ...

#define ...

globale variabelen

functies

  type functienaam (parameters)
    deklaratie van parameters
  {
    lokale variabelen
    statements
  }

main ()
{
  lokale variabelen
  statements
}
```

De index van een array in C begint altijd bij 0. In een array van tien elementen loopt de index dus van 0 tot en met 9 en niet van 1 tot en met 10 zoals bij de meeste andere talen.

In een printf-statement wordt een nieuwe regel aangegeven met `\n` (een backslash gevolgd door een kleine letter n).

Het compileren en linken van een C programma met PHIGS-statements wordt gedaan met het volgende commando:
`make` *programmanaam*

Hieronder volgt een korte beschrijving van de verschillende constructies van de taal C. Meer informatie is bijvoorbeeld te vinden in [6], [7] en [8].

Constructie
: *C-equivalent*

Identifiers
: Sequence of letters and digits starting with a letter or underscore; case sensitive.

Comments
: `/* ... */`

Data types

```
int
long int
short int
int
char
float
double
```

Declaration of variables

```
int i;
char c, d;
float x, y;
```

Assignment

value `=` *expression*

```
i = 2;
c = 'a';
x = x + 2 * x;
```

Main program

```
main()
{
   int i, j;
   i = 1;
   j = 3;
   i = i + j;
}
```

Addition

```
+
```

Subtraction

```
-
```

Multiplication

```
*
```

Real division

```
/
```

Unary +

not defined

Unary –

```
-
```

Integer division

```
/
```

Modulo

```
%
```

Reading an integer value

```
scanf ("%d", &i);
```

Writing an integer value

```
printf ("%d", i);
```

Less than

```
<
```

Less than or equal to

```
<=
```

Greater than

```
>
```

Greater than or equal to

```
>=
```

Equality

```
==
```

Inequality

```
!=
```

Logical and

```
&&
```

Logical or

```
||
```

Negation

```
!
```

If statement

`if` (*expression*)
statement

`if` (*expression*)
statement1
`else`
statement2

```
if (x < 1)
   y = 2

if (x < 1)
   y = 2;
else
   y = 3
```

While statement

```
while (expression) statement
```

```
while (x < 1) x++
```

Repeat statement

```
do statement
while (condition)
```

```
do x++;
while (x <= 0)
```

For statement

```
for (expression1; expression2; expression3)
   statement
```

```
for (i=1; 1<=10; i++)
   printf ("%d", i)
```

Case statement

```
switch (expression)  {
case l1:  ... case lk:
       statementl; break;
   ...
case m1:  ... case mn:
       statementm; break;
default:  statements; break;
}
```

```
switch (c = getchar())  {
case 'a':  case 'b':
case 'c':  putchar ('1'); break;
case 'd':  putchar ('2'); break;
default :  putchar ('3');
}
```

Open file for reading

```
f = fopen ("TEST", "r");
```

Open file for writing

```
f = fopen ("TEST", "w");
```

Testing for end-of-line

```
while ((c = getchar()) != '\n')
```

Testing for end-of-file

```
while ((c = getchar()) != EOF)
```

Function definition

```
int max (a, b)
int a, b;
{
   return (a>b?a:b);
}
```

Procedure definition

```
void test ()
{
   int i;
   i = 2;
   ...
}
```

Pointer variable definition

```
type *identifier;
int *p;
```

Pointer type

```
typedef int * pint;
pint p;
```

Dereferencing

```
*p
```

Nil pointer

```
NULL
```

Memory allocation

```
type *identifier;
identifier = (type*) malloc (sizeof(type));
```

Memory deallocation

```
free ((char*)identifier);
```

Array

type array-identifier`[`*size*`];`

Array type

`typedef` *knownID newID*`[`*size*`];`

```
typedef int VEC5[5];
typedef int VEC3[3];
typedef float FVEC[10];

VEC5 v5;
VEC3 v3;
FVEC f;
```

Two-dimensional array

```
int x[2][3];
```

Two-dimensional array type

```
typedef int TWO[3][4];
typedef int SINGLE[5];
typedef SINGLE TWO[4];
```

Record variable

```
struct {
   components;
} identifier;

struct {
   int i;
   float f;
} a;
```

Record type

```
typedef struct type {
   components;
} type;

typedef struct student {
       char name[30];
       int StudNum;
} STUDENT;

typedef STUDENT CLASS[100];

CLASS Stats;
```

Field access using pointers

pointer `->` *field*

Free unions

```
typedef union type {
   variant1;
   ...
   variantk;
} type;

typedef union {
   struct {
       int i, j;
   } v1;
   struct {
       float k, l;
   } v2;
} VARIANTS;
```

Enumeration type

```
typedef enum {
   blue, red, green
} color;
```

Appendix D

Fast fourier fransform (FFT)

D.1 FFT programma

Voor het berekenen van de discrete fouriergetransformeerde van een functie $f(x, y)$ is de functie `fourn.c` [zie 14] beschikbaar:

```
void fourn(data,nn,ndim,isign)
float data[];
int nn[],ndim,isign;
```

- De functie f(x,y) is bemonsterd op een vierkant rooster, met N_1 samples in de x-richting (afstand Δ_1) en N_2 samples in de y-richting (afstand Δ_2). Zowel N_1 als N_2 *moeten* machten van 2 zijn! Zonodig moeten nullen worden toegevoegd om de array te completeren.
- `fourn` verwacht *complexe* data. Als de functie $f(x, y)$ reëel is, zoals bij beeldinformatie waar slechts intensiteiten beschikbaar zijn, moeten imaginaire delen = 0 worden toegevoegd.

Betekenis van de parameters:

`data[1...(2`N_1N_2`)]` Data array (input) welke vervangen wordt door zijn fouriergetransformeerde.

`ndim` Dimensie van de fouriertransformatie, dus
1 : $f(x) \Leftrightarrow F(u)$
2 : $f(x, y) \Leftrightarrow F(u, v)$.

`nn[1...ndim]` Array welke de lengte van iedere dimensie (aantal complexe getallen) bevat. Als `ndim=2` is dus `nn[1]=`N_1 en `nn[2]=`N_2.

`isign` Bepaalt of $\mathcal{F}$ of $\mathcal{F}^{-\infty}$ wordt berekend

1 : $\mathcal{F}$
-1 : $\mathcal{F}^{-\infty}$

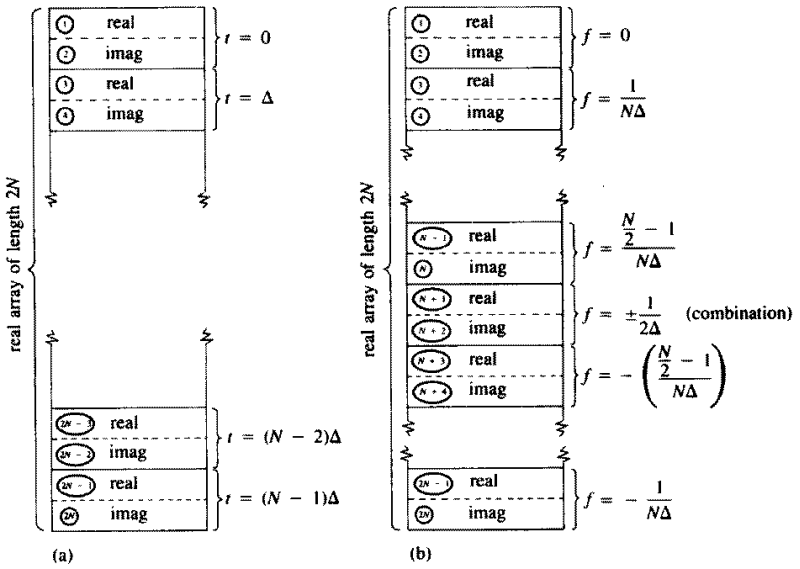


Figure D.1: *Input en output arrays voor ééndimensionale FFT: (a) De input bestaat uit N (een macht van 2) complexe ruimtelijke samples in punten t ($\equiv x$), opgeslagen in een array van lengte $2N$, waarin reële en imaginaire delen van de samples elkaar afwisselen, (b) Het output array bevat het complexe fourierspectrum voor N complexe waarden van de frequentie f ($\equiv u$).*

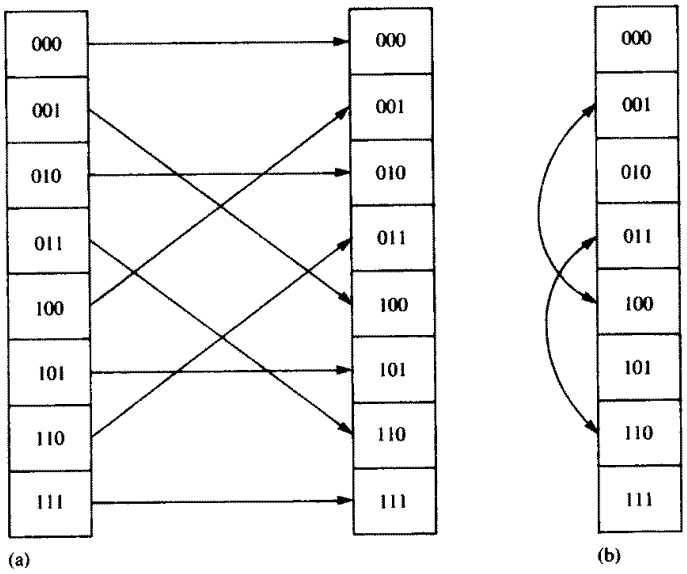


Figure D.2: *Opslag van een 2D FFT (output) in de array* `data[]`. *Iedere rij correspondeert met een vaste waarde van f_1 ($\equiv u$). Binnen een rij zijn de frequenties f_2 ($\equiv v$) georganiseerd zoals is weergegeven in Fig.D.1. De input array* `data[]` *bevat de complexe samples opgeslagen in opeenvolgende rijen (x=constant) van de array $f(x, y)$.*

```
#include <math.h>
#define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr
void fourn(data,nn,ndim,isign)
float data[];
int nn[],ndim,isign;
{
  int i1,i2,i3,i2rev,i3rev,ip1,ip2,ip3,ifp1,ifp2;
  int ibit,idim,k1,k2,n,nprev,nrem,ntot;
  float tempi,tempr;
  double theta,wi,wpi,wpr,wr,wtemp;
  ntot=1;
  for (idim=1;idim<=ndim;idim++)
    ntot *= nn[idim];
  nprev=1;
  for (idim=ndim;idim>=1;idim--) {
    n=nn[idim];
    nrem=ntot/(n*nprev);
    ip1=nprev << 1;
    ip2=ip1*n;
    ip3=ip2*nrem;
    i2rev=1;
    for (i2=1;i2<=ip2;i2+=ip1) {
      if (i2 < i2rev) {
        for (i1=i2;i1<=i2+ip1-2;i1+=2) {
          for (i3=i1;i3<=ip3;i3+=ip2) {
            i3rev=i2rev+i3-i2;
            SWAP(data[i3],data[i3rev]);
            SWAP(data[i3+1],data[i3rev+1]);
          }
        }
      }
      ibit=ip2 >> 1;
      while (ibit >= ip1 && i2rev > ibit) {
        i2rev -= ibit;
        ibit >>= 1;
      }
      i2rev += ibit;
    }
    ifp1=ip1;
    while (ifp1 < ip2) {
      ifp2=ifp1 << 1;
      theta=isign*6.28318530717959/(ifp2/ip1);
      wtemp=sin(0.5*theta);
      wpr = -2.0*wtemp*wtemp;
      wpi=sin(theta);
      wr=1.0;
      wi=0.0;
      for (i3=1;i3<=ifp1;i3+=ip1) {
        for (i1=i3;i1<=i3+ip1-2;i1+=2) {
          for (i2=i1;i2<=ip3;i2+=ifp2) {
            k1=i2;
            k2=k1+ifp1;
            tempr=wr*data[k2]-wi*data[k2+1];
            tempi=wr*data[k2+1]+wi*data[k2];
            data[k2]=data[k1]-tempr;
            data[k2+1]=data[k1+1]-tempi;
            data[k1] += tempr;
            data[k1+1] += tempi;
          }
        }
        wr=(wtemp=wr)*wpr-wi*wpi+wr;
        wi=wi*wpr+wtemp*wpi+wi;
      }
      ifp1=ifp2;
    }
    nprev *= n;
  }
}
#undef SWAP
```

Appendix E

Opdrachten

1. **Grafieken**

   [met OpenGL omgaan, window, voorbeeldprogramma]

   Maak een venster waarin een aantal grafieken getekend worden (bv. sinus, cosinus, tangens). Teken ook een assenstelsel door de oorsprong.

2. **Muis-interactie**

   [interactie, viewport, gebruikers- en venstercoordinaten, voorbeeldprogramma]

   Maak een venster dat een tekengebied en een gebied met knoppen bevat voor het selecteren van figuur (lijn, rechthoek), kleur (rood, groen, blauw) en functie (maak tekengebied schoon, stop programma).
   Met de muis kunnen de knoppen gebruikt worden om een tekening in het tekengebied te maken.

3. **Transformaties**

   [transformaties, viewport, voorbeeldprogramma]

   Pas een aantal transformaties toe door in vier kwadranten het volgende te demonstreren:

   - Translatie van een object van linksonder naar rechtsboven in het kwadrant.
   - Schaalverandering van een object met als centrum eerst de linkeronderhoek en daarna het midden van het kwadrant.
   - Rotatie van een object rondom het midden van het kwadrant.
   - Een combinatie van translatie, schaalverandering en rotatie, naar eigen fantasie.

4. **Bresenham**

   [scan conversie algoritme, transformaties, voorbeeld]

   Maak een functie die een cirkel genereert met behulp van het "Midpoint Circle Algorithm" van Bresenham. De functie heeft als argumenten de positie van het middelpunt en de grootte van de straal.
   De cirkel kan getekend worden door de berekende "rasterpunten" met lijnen te verbinden, zodat er niet direct in de frame buffer veranderd hoeft te worden.

   Gebruik deze cirkel om een bol te maken door lengte- en breedtecirkels te tekenen.

   Maak een programma dat één van de volgende visualisaties laat zien:

   - Laat een dergelijke bol in een kubusvormige ruimte bewegen, waarbij de bol steeds van de wanden teruggekaatst wordt.
   - Maak een miniatuur sterrenstelsel, dat minimaal een zon, een planeet en een maan bevat.
     De maan draait om de planeet, de planeet draait om de zon en de zon staat stil in het centrum. Zon, maan en planeet draaien ook om hun eigen as.

   Aanwijzingen:

- **Midpoint Circle Algorithm**:
  Beschouw het cirkelsegment van 45^o van $x = 0$ tot $x = y = \frac{R}{\sqrt{2}}$, waarbij R de straal van de cirkel is en $(0, 0)$ het middelpunt van de cirkel. Beginnend bij het punt $(0, R)$ en in de positieve richting van de X-as gaande, worden de pixels gegenereerd door x steeds met één op te hogen en door y, indien nodig, met één te verlagen. Hierbij wordt aangenomen dat x, y en R integers zijn. Door de windowgrenzen groot genoeg te definiëren, is dit geen probleem.
  De functie $F(x, y) = x^2 + y^2 - R^2$ is gelijk aan nul op de cirkel, groter dan nul buiten de cirkel en kleiner dan nul binnen de cirkel. Deze functie wordt gebruikt om, uitgaande van een pixel in (x_p, y_p), de bij $x_p + 1$ behorende y-waarde te bepalen. Als d de waarde van de functie in het punt tussen $(x_p + 1, y_p)$ en $(x_p + 1, y_p - 1)$ is (het midpoint ofwel het punt $(x_p + 1, y_p - \frac{1}{2})$), dan geldt:

$$d_{old} = F(x_p + 1, y_p - \frac{1}{2}) = (x_p + 1)^2 + (y_p - \frac{1}{2})^2 - R^2$$

  Als $d_{old} < 0$, dan ligt het midpoint binnen de cirkel (of met andere woorden: loopt de cirkel buiten het midpoint om). De waarde van y_p wordt niet verlaagd en het volgende pixel op de cirkel heeft $(x_p + 1, y_p)$ als coördinaten. Voor het daarop volgende midpoint $(x_p + 2, y_p - \frac{1}{2})$ geldt dan:

$$d_{new} = F(x_p + 2, y_p - \frac{1}{2}) = (x_p + 2)^2 + (y_p - \frac{1}{2})^2 - R^2 = d_{old} + (2x_p + 3) = d_{old} + \Delta_{noincy}$$

  Als $d_{old} >= 0$, dan ligt het midpoint buiten of op de cirkel. De waarde van y_p wordt nu wel verlaagd en het volgende pixel heeft $(x_p + 1, y_p - 1)$ als coördinaten. Voor het daarop volgende midpoint $(x_p + 2, y_p - \frac{3}{2})$ geldt dan:

$$d_{new} = F(x_p + 2, y_p - \frac{3}{2}) = (x_p + 2)^2 + (y_p - \frac{3}{2})^2 - R^2 = d_{old} + (2x_p - 2y_p + 5) = d_{old} + \Delta_{incy}$$

  Afhankelijk van de waarde van d wordt de waarde van y al of niet met één verlaagd en kan de waarde van Δ_{incy} of Δ_{noincy} aan d toegevoegd worden om de waarde van het volgende midpoint te berekenen. De beginwaarde van d tenslotte, kan berekend worden door uit te gaan van het punt $(0, R)$. Het volgende midpoint ligt op $(1, R - \frac{1}{2})$ en dit levert

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

  Omdat ervan uitgegaan is dat x en y integers zijn, is d de enige floating point variabele in deze berekening. Om deze ook als integer te kunnen gebruiken voor snellere uitvoering van het algoritme, doen we het volgende. Door $h = d - \frac{1}{4}$ te definiëren en $h + \frac{1}{4}$ voor d te substitueren, krijgen we als initialisatie $h = 1 - R$ en wordt de vergelijking $d < 0$ vervangen door $h < \frac{1}{4}$. Maar omdat h nu ook als integer gebruikt kan worden, en omdat er integer waarden Δ_{incy} en Δ_{noincy} bij opgeteld worden, kan het gehele algoritme met behulp van integer variabelen uitgevoerd worden.
- Elk pixel dat in het segment van 45^o gegenereerd is, kan door de veelvoudige symmetrie van een cirkel nog zeven maal op eenvoudige wijze op overeenkomstige plaatsen in de overige segmenten geplaatst worden. Dus als een pixel berekend is, kan dit door middel van een functie op acht plaatsen getekend worden.
- Aangezien er nog niet direct in de frame buffer gewerkt wordt, kan er een polylijn gebruikt worden om alle pixels in op te slaan.

5. **Projecties**

   [projecties, synthetische camera, viewport]

   Basisfiguur is een assenstelsel (X-, Y- en Z-as in verschillende kleuren) met een object (kubus, theepot) dat ligt binnen een kubus begrensd door de oorsprong en ribben langs de assen van lengte 10. Daarnaast is er een punt P buiten de kubus.

   Toon vier kwadranten waarin het volgende te zien is:

   - Een orthogonale projectie gezien vanaf P (doel: oefenen projecties).
   - Een perspectiefprojectie gezien vanaf P (doel: oefenen projecties).

- Zoom in en uit op het object vanaf punt P (doel: synthetische camera).
- Draai in een cirkel om het object heen, kijkend naar het object, beginnend in punt P, met de oorsprong als middelpunt van de cirkel (doel: synthetische camera).

6. **3D viewer**

[projecties, synthetische camera, voorbeeldprogramma]

Maak een driedimensionale viewer waarmee het mogelijk is objecten te bekijken. Minimale functionaliteit van de viewer bestaat uit het kunnen roteren van het object rond X-, Y- en Z-as en het kunnen in- en uitzoomen.

Mogelijkheid 1: event viewer.

In de hoge-energiefysica worden versnellers gebruikt om kleine deeltjes (bv. elektronen, positronen, protonen) te versnellen met het doel deze te laten botsen en de bij de botsing ontstane brokstukken te bestuderen om zo meer te weten te komen over de bouw van materie en de krachten die op materiedeeltjes werken.
De versnelde deeltjes kunnen op een vast doel geschoten worden (fixed target) of twee bundels met versnelde deeltjes kunnen op elkaar geschoten worden. Dit laatste gebeurt meestal in een cirkelvormige opstelling, waarin de twee bundels tegen elkaar in langs de cirkelomtrek vliegen (collider).
Een collider bevat een aantal vaste punten waar de bundels kunnen botsen en op deze punten worden detectoren gebruikt om informatie over de tijdens een botsing ontstane deeltjes te registreren.

Een dergelijke detector is de *L3*-detector in de LEP-ring van het CERN-laboratorium bij Genève.
Deze detector bestaat in vereenvoudigde vorm uit de volgende subdetectoren:

- TEC (Time Expansion Chamber), dradenkamer, cylinder met de as gelijk aan de as waarlangs de deeltjesbundels zich voor een botsing bewegen, buitendiameter 50 cm, lengte 50 cm,
- EC (Electromagnetic Calorimeter), BGO kristallen, bol, binnendiameter 75 cm, buitendiameter 100 cm,
- HC (Hadron Calorimeter), dradenkamers, cylinder met de as als bij de TEC, binnendiameter 110 cm, buitendiameter 200 cm, lengte 200 cm,
- MU (Muon Chambers), dradenkamers, drie cylinders (eigenlijk heeft elk daarvan geen cirkel maar een achthoek als doorsnede) met de assen als bij de TEC, binnendiameter resp. 210, 360 en 510 cm, buitendiameter resp. 240, 390 en 540 cm, lengte resp. 200, 350 en 500 cm.

De meetgegevens van de verschillende detectoren bestaan uit:

- TEC: θ, ϕ, R
- EC: θ, ϕ, E
- HC: θ, ϕ, E
- MU: θ_1, ϕ_1, θ_2, ϕ_2, θ_3, ϕ_3

waarbij:

- θ = hoek met de z-as in een vlak door de z-as (0^o in de positieve z-richting),
- ϕ = hoek in een vlak loodrecht op de z-as,
- R = straal van cirkel die afbuiging van deeltje aangeeft,
- E = energie.

Het botsingspunt ligt in de oorsprong en valt samen met het middelpunt van de detectoren.

Gegevens van detector en van event worden uit een database ingelezen.

Mogelijkheid 2: viewer voor moluculen.

Bedoeling is een viewer te maken waarmee moleculen bekeken kunnen worden. Tot de mogelijkheden van de viewer moeten in ieder geval behoren:

- kiezen van een molecuul uit een lijst van beschikbare moleculen,
- tonen van het gekozen molecuul, d.w.z. de verschillende atomen waaruit het molecuul is opgebouwd en de bindingen tussen de atomen,
- in- en uitzoomen op het molecuul,

– roteren om X-, Y- en/of Z-as van het molecuul.

De beschrijvingen van de atomen waaruit moleculen opgebouwd kunnen zijn, kunnen uit een file ingelezen worden, net als de beschrijvingen van specifieke moleculen.

Voorbeeld van een file met beschrijvingen van atomen. De eerste twee regels van deze file zijn commentaar en elke daarop volgende regel bevat de beschrijving van een atoom. Deze beschrijving bestaat uit het referentienummer van het atoom, de diameter (er vanuitgaande dat het atoom als een bolletje weergegeven wordt) en de drie kleurcomponenten rood, groen en blauw om het bolletje een eigen kleur te geven.

Voorbeelden van files met beschrijvingen van moleculen: C_2H_2, ethaan, SO_2, NH_4 (met dank aan Michiel Alsters).
In deze files is de eerste regel commentaar, de tweede regel bevat het aantal atomen waaruit het molecuul is opgebouwd, gevolgd door regels met gegevens over de atomen (één regel per atoom), dan een regel met het aantal bindingen tussen de verschillende atomen, gevolgd door de gegevens over de bindingen (één regel per binding).
Een atoom wordt beschreven door het referentienummer en de X-, Y- en Z-ordinaten die de positie van het middelpunt in het molecuul aangeven.
Een binding wordt beschreven door de nummers van twee te verbinden atomen, deze nummers zijn het volgnummer van de atomen binnen de molecuulbeschrijving.

7. **NURBS**

[NURBS, interactie, voorbeeldprogramma]

Maak een programma waarin een kromme, gemaakt met behulp van een "non-uniform rational B-spline" (NURBS) in een venster getoond wordt, tezamen met de controlepunten die de kromme "opspannen".
Zorg dat elk controlepunt met de muis verschoven kan worden, waarbij de kromme meteen mee verandert.

Als "extra" bij de opdracht kunnen de waarden van alle knopen getoond worden, met de mogelijkheid om die waarden te veranderen, waarbij de nieuwe kromme weer getekend wordt.

Als "extra-extra" kan de orde veranderd worden (en dus het aantal knopen bij gelijkblijvend aantal controlepunten).

8. **Beeldreconstructie**

Maak een programma dat een gegeven tweedimensionaal object vanuit verschillende richtingen projecteert en uit de projecties een reconstructie van het oorspronkelijke object maakt. Hierbij kan een keuze uit verschillende objecten en reconstructiemethoden gemaakt worden.

De objecten waaruit gekozen kan worden, zijn:

(a) vierkant,

(b) cirkel,

(c) ellips,

(d) twee rechthoeken,

(e) eigen ontwerp.

De reconstructiemethoden waaruit gekozen kan worden, zijn:

(a) ongefilterde terugprojectie,

(b) iteratieve terugprojectie,

(c) gefilterde terugprojectie (convolutiemethode),

(d) gefilterde terugprojectie (fouriermethode).

Breng de keuze van objecten en terugprojectiemethode onder in menu's.

De uitvoer van het programma bestaat uit:

(1) Het originele object (dus slechts 1 slice).

(2) De opeenvolgende reconstructies (zie beneden): 8 slices.

Aanwijzingen:

- Projectierichtingen ϕ liggen tussen 0 en π.
- Verwerk series projecties in opeenvolgende graad van verfijning. Neem per serie als richtingen de hypothenusa's van de hoeken tussen de vorige richtingen:
  Stap 1: $\phi = 0, \pi/2$ $\rightarrow$ slice 1
  Stap 2: $\phi = \pi/4, 3\pi/4$ $\rightarrow$ slice 2
  Stap 3: $\phi = \pi/8, 3\pi/8, 5\pi/8, 7\pi/8$ $\rightarrow$ slice 3
  $\vdots$
  Stap k: $\phi = \pi/(2^k), 3\pi/(2^k), 5\pi/(2^k), \ldots$ $\rightarrow$ slice k
  Om rekentijd acceptabel te houden: ga tot $k = 8$ (dus 8 slices).
- Gebruik een rooster van 64×64 pixels om de objecten in weer te geven.
- Meer toelichting:
  Het origineel (een tweedimensionaal object) sla je op in een NxN matrix ORI, bijvoorbeeld een vierkant in 10x10, waarbij 0=niets en 1=gevuld aangeeft:

  0000000000
  0111111110
  0100000010
  0100000010
  0100000010
  0100000010
  0100000010
  0100000010
  0111111110
  0000000000

  Dit is het oorspronkelijke 2-D object, dat je onder verschillende hoeken doorlicht (scannen), wat een 1-D projectie onder elke hoek geeft.

  Als phi de hoek is waaronder gescand wordt, kan je elk element (x,y) van de matrix ORI dat groter dan nul is, projecteren op het projectie-array PRO, dat N*sqrt(2.0) elementen heeft (maximaal aantal elementen bij projectie onder bijvoorbeeld 45 graden):
  index in PRO = midden van projectie-array + x*cos(phi) + y*sin(phi)
  PRO[index] += ORI[x][y]

  Op deze manier kan je de gewenste 1-D projecties maken, elke projectie bij een specifieke hoek phi.
  (stap 3 van algoritme op http://www.hef.ru.nl/ pfk/teaching/icg/icg-12-2-3.html)

  De filterstap die hierna gedaan wordt (stap 4 van het genoemde algoritme) kan dan overgeslagen worden (ongefilterd), of d.m.v. iteratieve terugprojectie, convolutie- of fouriermethode gedaan worden.

  Stap 5 van het algoritme is dan min of meer het omgekeerde van stap 3, nu wordt de gereconstrueerde figuur opgebouwd in een matrix REC van dezelfde dimensies als de matrix ORI.

  De eerste keer, slice 1, wordt dit voor projecties onder twee hoeken (0 en pi/2) gedaan.
  De tweede keer, slice 2, worden aan REC de projecties onder de hoeken pi/4 en 3*pi/4 toegevoegd (dit spaart rekenwerk!).
  Enzovoort...
  Let er wel op om de normalisatie (stap 6 van het algoritme) steeds op de juiste plaats te doen.

  Voor de fouriermethode kan fourn.c gebruikt worden. Invoer wordt gevormd door de 1-D projecties, alleen moeten hier imaginaire delen (gelijk aan 0.0) voor alle waarden in het array aan toegevoegd worden. Ook moet het aantal waarden een macht van 2 zijn.
  (zie http://www.hef.ru.nl/ pfk/teaching/icg/icg-AF.html)

Convolutie- en fouriermethode zelf zijn beschreven in
http://www.hef.ru.nl/ pfk/teaching/icg/icg-12-2-4.html
Aan het eind van de ze pagina staat het algoritme voor de fouriermethode, stap 1 is heen projecteren met fourn.c, stap 2 is filteren, stap 3 is terug projecteren met fourn.c.

9. **Viewer voor data array**

   [visualisatie]

   Maak viewer voor een NxN data array, waarbij de gegevens volgens verschillende methoden bekeken worden:

   - topview m.b.v. grijstinten,
   - topview m.b.v. false color (SHV),
   - 3D wire grid,
   - 3D solid model m.b.v. kleur (3D solid model met Gouraud shading).

   Data files die gebruikt kunnen worden (grafiet1.pgm, ltgraf18.pgm, ltgraf22.pgm) en korte beschrijving van het formaat van deze files.

10. **Filteren**

    [visualisatie]

    Maak een programma dat de volgende filtertechnieken bevat:

    - middelingfilter,
    - mediaafiltern,
    - gradiëntfilter,
    - Laplacefilter,
    - Butterworth laagdoorlaatfilter.

    Data file die gebruikt kan worden (bnoise.pgm) en korte beschrijving van het formaat van deze file.

11. **Texture mapping**

    [voorbeeldprogramma met image file mickey.ppm, voorbeeldprogramma met mipmaps]

    Maak een kamer met aan de wanden een aantal schilderijen en volg daarin een pad om naar de schilderijen te kijken.

12. **Beeldverbetering**

    [visualisatie]

    Verbeter een gegeven afbeelding, die met "sneeuw" vervuild is, door middel van mediaanfiltering.

    Experimenteer met verschillende afmetingen van het filter (in ieder geval met 3x3 en 5x5).

Appendix F

Register