Sammlung und Erklärungen

zu

Computergraphik

Thomas Neurauter (92 55 135)

Inhaltsverzeichnis

1 Einführung

1.1 Motivation

"Ein Bild sagt mehr als 1000 Worte."
(→ größere Aussagekraft)

1.2 Definition

Die Computergraphik besteht heute aus 3 Teilgebieten:

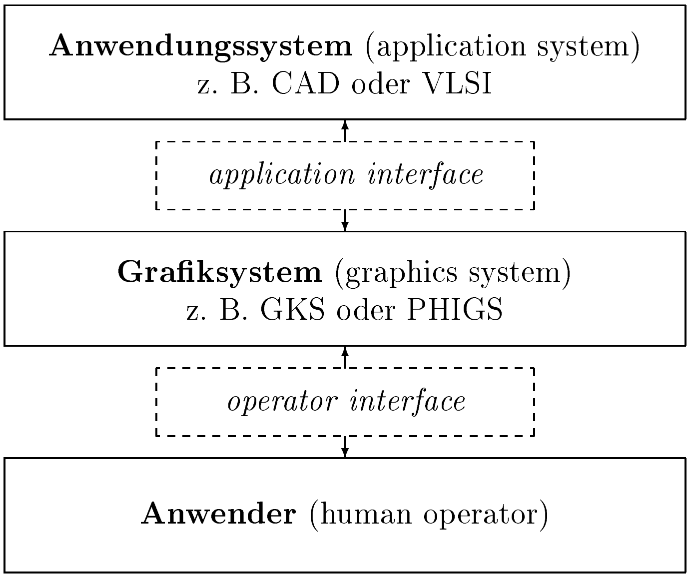
- generative Computer Graphik *(computer graphics)*
- Bildverarbeitung *(image processing)*
- Bildanalyse *(picture analysis)*

1.3 Hauptanwendungsgebiete der Computer Graphik

- CAD
- Kartographie
- Business Graphik
- Echtzeit-Bilddarstellung bei der Simulation wirklichkeitsnaher Szenarien
- Visualisierung technisch-naturwissenschaftlicher Vorgänge
- sonstige Visualisierungen
- Prozeßüberwachung
- Computerkunst
- Computerspiele

generative CG	**Bildanalyse**	**Bildverarbeitung**
	Eingabe	
formale Beschreibung Programme- & Datenstrukturen	unstrukturiertes Bild Photographie/Video Menge der Pixels	unstruriertes Bild (Pixels)
	Ausgabe	
Bild, Zeichnung, Tabelle, Pläne	formale Beschreibung Daten & -strukturen Zusammenhangsbe- schreibung	Bild
	graphische Typen	
Linien Flächen Texte	erzeugte oder abge- tastete Bilder	abgetastete Bilder
	formale Typen	
Segmente Objekte Primitives	Menge von Farbpunkten	Menge schlecht unterscheidbarer Farbpunkte
	Zweck	
Bilderstellung Bildausgabe Bildmanipulation	Bildstrukturierung Bilderkennung	Bildverbesserung Konturenerkennung
	Vorteile für den Bediener	
Sichtbarmachung von erstellten Ergebnissen, Dialoge auf Bild- und Symbolebene	Manipulation vormals unstrukturierter Bil- der, Identifikation von Bildern	Bilder erkennen

1.4 Modellkonfiguration graphischer Systeme



Auswahlkriterien:

für *Anwendungssystem:*

- Anwendungsproblematik z. B. Darstellen, Zeichnen und Entwerfen

für das *graphische System:*

- technische Aspekte, z. B. 2D-, 3D-Darstellungsverfahren, Interaktion, Sprachenbindung, graphische Darstellungselemente

für die *Auswahl der graphischen Geräte:*

- Auflösung
- Darstellungsgeschwindigkeit
- Qualität der Darstellung
- Farbgebung
- Größe des Ausgabemediums
- logische und physikalische Schnittstellen

1.5 Graphikstandards

GDDM, PLOT-10, CALCOMP, POSTSCRIPT, REGIS, DI-3000, DISSPLA, UNIRAS, GKS, PHIGS, ...

Ziele von GKS:

- Festlegung einer einheitlichen Schnittstelle zwischen Anwendungen und graphischen Systemen
- Definition einer anwendungsunabhängigen und effizient realisierbaren Funktionsbibliothek
- Abdeckung möglichst vieler Anforderungen aus dem graphischen Problembereich
- Trennung von grundlegenden graphischen Kernfunktionen und höherwertigen Modellierungsfunktionen

1.6 GKS - PHIGS

PHIGS	Programmers Hierachical Interactive Graphics System
GKS	Graphical Kernel System

1.6.1 Unterschiede GKS - PHIGS

Unterschiede:	GKS-3D	PHIGS	PHIGS++
Segmente	JA	JA	JA
Hierachische Datenbasis	Nein	JA	JA
Editieren von Modelldaten	Nein	JA	JA
Hiddenline - Surface Removal	Nein	Nein	JA
Beleuchtungsmodelle	Nein	Nein	JA
Schattierungen	Nein	Nein	JA

2 Graphische Ein- und Ausgabe

2.1 Ausgabegeräte

2.1.1 Überblick über optische Ausgabegeräte

1. Kathodenstrahlröhre *(Cathode Ray Tube)*

   Besteht aus einer *Kathode*, die gemeinsam mit der Anode den Elektronenstrahl erzeugt, einem *Ablenkungssystem*, das die Steuerung der Elektronen durchführt und aus der auf der Innenseite *phosphorbeschichteten Schirmoberfläche* (der Anode). Die Anzeige erfolgt durch laufendes Aussenden von Elektronen, die durch eine hohe elektrische Spannung beschleunigt werden, fokussiert und abgelenkt auf die Schirmoberfläche treffen und dort ein Aufleuchten des Phosphors bewirken (Fluoreszenz).

2. *Beam Penetration Color CRT*

   Sie verwenden zwei verschiedenfarbige Phosphorschichten mit unterschiedlichen Erregungsschwellen. Durch Modulation des Beschleunigungspotentials der Elektronen können etwa mit einer roten und einer grünen Phosphorschicht die Farben rot, orange, gelb und grün erzeugt werden. Um ein Flackern des Bildes durch dauerndes und zeitraubendes Verändern des Potentials ($20 - 200\mu$sek) zu verhindern, müssen die Elemente nach Farben sortiert angezeigt werden.

3. Speicherröhre *(Direct View Storage Tube)*

   Vom Prinzip her handelt er sich auch hier um eine Elektronenstrahlröhre, deren Phosphor allerdings extrem lange nachleuchtet. Dieser Effekt wird durch die Speicherung des Bildes auf einem direkt unter der Phosphorschicht liegenden Gitter erreicht. Der Elektronenstrahl reduziert durch die hohe kinetische Energie beim Auftreffen auf dieses nichtleitende Gitter die Anzahl der dort befindlichen Elektronen und hinterläßt somit eine positive Ladung. Erst die zweite Elektronenquelle, die andauernd eine Flut von Elektronen aussendet, macht das gespeicherte Bild sichtbar, da die Elektronen an den positiv geladenen Gitterpunkten bis zum Phosphorschirm vordringen können.

4. Plasmabildschirm

   Er besteht im wesentlichen aus zwei Glasplatten, auf denen horizontale und vertikale Leiterbahnen aufgebracht sind, und kleinen Zellen an den Kreuzungspunkten der Leiterbahnen, in denen Neongas eingeschlossen ist. Mit einer entsprechenden elektrischen Spannung, die zur Aufrechterhaltung des momentanen Zustands dient, befinden sich diese Zellen in einem von zwei stabilen Zusänden - erleuchtet oder dunkel. Die Spannungsdifferenz bringt das Gas zum Leuchten.

5. LED-Schirme *(light emitting diodes)*

6. LCD-Schirme *(liquid crystal displays)*

7. Laser-Displays

   Der Laser schreibt auf ein Filmmedium, und die dadurch entstandene "Spur" wird durch eine eigene Lichtquelle auf einen Großschirm projiziert.

8. Farbbildschirm *(Shadow Mask Color CRT)*

   Ähnlich den Farbfernsehgeräten ist die Schirmoberfläche mit Dreier-Gruppen von roten, grünen und blauen Phosphorpunkten beschichtet. Weiters werden drei Kathoden (für jede Phosphorart) verwendet. Dadurch kann eine fast unbegrenzte Zahl von Farben dargestellt werden.

2.1.2 Raster- und Vektorschirme

Eine weitere Möglichkeit, Sichtgeräte zu charakterisieren, liegt in der Unterscheidung, ob die Anzeige *vektororientiert (random scan)* oder *rasterorientiert (raster scan)* erfolgt.

- *Random Scan:* Der Elektronenstrahl wird frei am Bildschirm "bewegt". Gezeichnete Linien werden direkt — ohne Approximation — dargestellt.

  Die Informationen über das darzustellende Bild sind in einem sogenannten *Display-File* abgelegt, wo sie von der Kontrolleinheit *(Display Controller)* gelesen und in Steuerungsinformationen für den Bildschirm umgesetzt werden. Außer bei Speicherbildschirmen muß refreshed werden (mind. 25mal/Sekunde). ⇒ Bei zu komplexen Bildern (Display-Files) kann es zu Bildschirmflimmern kommen.

  Die Vorteile liegen in der hohen Dynamik der Anzeige und in der, verglichen mit Rastersystemen, höheren Auflösung. Um Bildteile zu löschen bzw. hinzuzufügen, muß man nur deren Definition aus dem Display-File entfernen bzw. hineinschreiben.

- *Raster Scan:* Der Refresh erfolgt zeilenweise von oben nach unten. Linien werden approximiert.

  Die eigentliche Bildinformation liegt Pixel für Pixel in einem eigenen Speicher, dem *Bildwiederholspeicher (frame buffer, refresh buffer)*.

  – Gefüllte Flächen können dargestellt werden.
  – Vereinfachte Steuerung des Elektronenstrahls (nur horizontal und vertikal). Dabei wird der Übergang von einer Zeile in die nächste (horizontaler Refresh) durch das horizontale Synchronisierungssignal und der Übergang von rechts unten nach links oben (vertikaler Rücklauf) durch das vertikale Synchronisierungssignal gesteuert.
  – Keinerlei Einschränkungen bezüglich der Bildkomplexität.

  Sollte das System nicht imstande sein, das gesamte Bild mit der geforderten Refresh-Rate darzustellen ⇒ *Interlacing.*

2.1.3 Hardcopy-Geräte

- *Plotter:*
  - Trommelplotter (drum plotter)
  - Tischplotter (flatbed plotter)
  - COM-Plotter (Computer Output on Microfilm): Laser $\rightarrow$ Microfilm
- *Drucker:*
  - Elektrostatische Plotter
  - Tintenstrahl-Drucker (ink-jet-printer)
  - Laserdrucker
  - Thermotransferdrucker

2.2 Eingabegeräte

1. Tastatur
2. Paddles
3. Joystick
4. Track-Ball
5. Maus
6. Tablett (Digizer)
7. Lichtgriffel
8. Berührungsempfindlicher Bildschirm *(Touch-Panel)*
9. Video-Digitalisierungssystem *(Frame-Grabber)*

3 Grundelemente der grafischen Programmierung

3.1 Koordinatensysteme, -transformationen

Generell verwendet wir die *kartesischen* Koordinaten, obwohl Polarkoordinaten häufig günstiger wären.
In *GKS* unterscheidet man folgende Systeme:

- **Weltkoordinatensystem** *(world coordinates)*

  Der mit den *WC* abgedeckte Problembereich sind *Welten*, indem sich der Benutzer sein eigenes Koordinatensystem definiert.

- **Normiertes Koordinatensystem** *(normalized device coordinates)*

  Mit *NCD* werden Teile aus $[0,1] \times [0,1]$ auf Geräten dargestellt, um arbeitsplatzspezifische und arbeitsplatzunabhängige Bildteile zu trennen.

- **Geräte-Koordinaten** *(device coordinates)*

  DC sind naturgemäß begrenzt.

Definition:

> Fenster *(window)* und Darstellungsfeld *(viewport)* sind Ausschnitte, die achsparallel zu zugehörigen Koordinatensystemen sind, wobei das sich das window auf einer höheren Ebene (world coordinates) befindet als der viewport (normalized device coordinates).

3.2 Darstellungselemente

3.2.1 Überblick

Grundprinzipien der Definition von graphischen Objekten sind die **Typenangabe** (Linie, Rechteck, ...) und die **Angabe von Bestimmungspunkten** (*absolute* Koordinatenangabe bzw. *relativ* zu einem anderen Punkt (z. B. Chaincode)).

3.2.2 Polymarker

In GKS hat der Benutzer die Möglichkeit, über sogenannte Attribute aus den verschiedensten "Formen" eines Punktes wie $*, \circ, \times, +$ usw. auszuwählen.

3.2.3 Text

Punkt- oder Vektororientiert.

3.2.4 Linie

Aufgabe:

Darstellung einer durch Anfangs- und Endpunkt gegebenen Linie mittels einer Folge von Rasterpunkten.

Nebenbedingungen:

- optisch möglichst gut
- effizienter Algorithmus, d. h. möglichst ohne Division und möglichst Integer-Operationen $\Longrightarrow$ ein sog. **DDA** (Digital Differential Analyzer)

Bresenham-Algorithmus:

Die beste Approximation der wahren Linie ist gegeben durch jene Punkte, die den geringsten Abstand von der wahren Linie haben. Es sei t_i der Abstand zu den Punkten, die direkt über der wahren Linie liegen, der Abstand zu denen direkt darunter sei s_i. Bresenhams Algorithmus benutzt als Entscheidungsgröße (= Fehler) $e_i = s_i - t_i$. Ist e_i kleiner Null, dann ist der Punkt unterhalb der wahren Linie das nächstgelegene im Raster. Im anderen Fall, wenn e_i kleiner oder gleich Null ist, dann ist der Punkt direkt darüber der wahren Linie am nächsten.

Zur Realisierung des Algorithmus muß man dann nur noch die e_i-Werte wie folgt berechnen und aktualisieren:

Anfangs setze $e_1 = 2 \cdot \Delta y - \Delta x$, wobei $\Delta x = x_2 - x_1$ und $\Delta y = y_2 - y_1$.

Danach: Ist $e_i > 0$, dann werden x und y erhöht, also $x_{i+1} = x_i + 1, y_{i+1} = y_i + 1$ und $e_{i+1} = e_i + 2 \cdot (\Delta y - \Delta x)$.

Ist jedoch $e_i < 0$, dann wird nur x erhöht: $x_{i+1} = x_i + 1$ und $e_{i+1} = e_i + 2\Delta y$.

Diese Technik bietet eine effiziente Methode zur Rasterkonvertierung von Linien, deren Steigung weniger als 45^0 beträgt. Für Geraden mit Steigung größer als 45^0 und weniger als 90^0 müssen die Koordinaten x und y vor Aufruf des Unterprogramms vertauscht und später vor dem Plotten des Punktes wieder zurückgetauscht werden.

Problem der dünnen Linien:

Das Problem besteht darin, daß hellere Linien dicker erscheinen. Umgangen kann das werden, indem erstens horizontale und vertikale Linien mit weniger Intensität gezeichnet werden und zweitens Nachbarpunkte berücksichtigt werden.

3.2.5 Kreis

$$x^2 + y^2 = r^2$$

Die beste Annäherung ist dann gefunden, wenn

$$\underbrace{\left|x^2 + y^2 - r^2\right|}_{\text{euklid'scher Abstand}} \qquad = \text{minimal.}$$

Bresenham-Algorithmus:

Wird die *Acht-Wege-Symmetrie des Kreises* ausgenutzt, dann müssen nur die Punkte in einem 45^0-Bereich erzeugt werden. Werden dann nocht die Punkte des Bereiches zwischen 90^0 und 45^0 erzeugt, so werden die Bewegungen nur in der $+x$ und $-y$ Richtung stattfinden.

Die beste Annäherung des wahren Kreises ist durch jene Punkte im Raster gegeben, die den geringsten Abstand vom wahren Kreis besitzen. Werden die Punkte im Vereich zwischen 90^0 und 45^0 erzeugt, so kann jeder neue dem Kreis am nächsten gelegene Punkt durch eine von zwei Aktionen gefunden werden:

1. Man gehe eine Einheit in der x-Richtung oder
2. man gehe eine Einheit in der x-Richtung und eine Einheit in der negativen y-Richtung.

Um die Punkte zu finden, die dem Kreis am nächsten liegen, benötigt man also nur eine Methode zur Auswahl einer dieser Aktionen.

Zur Abschätzung des Fehlers verwenden wir die Variablen d_1 und d_2.

$$\begin{array}{rclcrcl} d_1 & = & y_i^2 - y^2 & \qquad & d_2 & = & y^2 - (y_i - 1)^2 \\ & = & y_i^2 - r^2 + (x_i + 1)^2 & & & = & r^2 - (x_i + 1)^2 - (y_i - 1)^2 \end{array}$$

Die Differenz e_i dieser beiden Variablen entscheidet durch das Vorzeichen über die Auswahl von y_i oder $y_i - 1$. Für $e_i < 0$ wird y_i und für $e_i \geq 0$ der y-Wert $y_i - 1$ gewählt.

In eine iterative Form gebracht lautet die Gleichung

$$\begin{array}{rcl} e_{i+1} & = & e_i + 4x_i + 6 + 2(y_{i+1}^2 - y_i^2) - 2(y_{i+1} - y_i) \qquad \text{mit} \\ e_1 & = & 3 - 2r \end{array}$$

Da im 2. Oktanten y_{i+1} entweder y_i oder $y_i - 1$ ist, kann e_{i+1} in Abhängigkeit des Vorzeichens von e_i weiter vereinfacht werden.

$$\begin{array}{lcl} e_i < 0 & \Rightarrow & e_{i+1} = e_i + 4(x_i + 1) + 2 \\ e_i \geq 0 & \Rightarrow & e_{i+1} = e_i + 4(x_i + 1) + 2 - 4(y_i - 1) \end{array}$$

3.2.6 Füllgebiete

In GKS ist ein Füllgebiet als beliebiges Polygon mit n Punkten definiert, wobei zwei aufeinanderfolgende Punkte eine Polygonkante definieren. Die schließende Kante wird durch Verbindung des ersten und letzten Punktes erzeugt.
Ein mit dem Attribut *ungefüllt* gefülltes Füllgebiet entspricht einem Polygonzug.

3.2.7 Zellmatrix

Speziell zur Wiedergabe von grafischen Daten, die über Scanner oder Videokameras gewonnen wurden, bietet GKS eine Zellmatrix als Ausgabeobjekt an. Die Zellmatrix ist als achsenparalleles Rechteck durch zwei gegenüberliegende Eckpunkte P und Q definiert. Zusätzlich geben die Parameter Δx und Δy die logische "Auflösung" des Rechtecks in x- und y-Richtung an.

3.3 Attribute der Darstellungselemente

Die im vorangegangenen Kapitel besprochenen Darstellungselemente können mir einer Vielzahl von Attributen versehen werden. In GKS werden die zur Verfügung stehenden Attribute in drei Gruppen eingeteilt:

1. *geometrische* Attribute,

   Sie beeinflußen die Form oder Größe des ganzen Elements (z. B. die *Zeichenhöhe* oder *Schreibrichtung*).

2. *nichtgeometrische* Attribute und

   Sie steuern nur das Aussehen (z. B. den *Linientyp*) oder die Form und Größe des entsprechenden Teils des Darstellungselements (z. B. *Vergrößerungsfaktor* bei Polymarker).

3. *Identifizierungs*-Attribute

   Das einzige Identifizierungs-Attribut in GKS ist die *Pickerkennzeichnung*, die zur Identifizierung von Darstellungselementen bzw. einer Gruppe von Darstellungselementen in einem Segment dient, wenn dieses Segment vom Benutzer gepickt wurde.

GKS und PHIGS führen eine *Zustandsliste* für Attribute, die gegebenenfalls manipuliert wird.

3.3.1 Marker-Attribute

Für Marker sind nur folgende **nichtgeometrische Attribute** vorgesehen:

- der *Markertyp* (marker type), der das Auzssehen des Markers festlegt (Typen 1 - 5 sind: $\cdot, +, *, \circ, \times$),
- der *Markervergrößerungsfaktor* (marker size scale factor) und
- der *Markerfarbindex* (polymarker color index)

3.3.2 Text-Attribute

Dieses Darstellungselement weist in GKS am meisten Attribute auf.

Die **nichtgeometrischen Attribute** sind

- Der *Zeichenbreitefaktor* (character expansion factor), der die Abweichung vom im Font vorgegebenen Höhen-zu-Breitenverhältnis angibt.
- Der *Zeichenabstand* (character spacing), der festlegt, wieviel zusätzlicher Abstand zwischen zwei benachbarten Zeichenkörpern einzufügen ist.

- Die *Schriftart* und *-qualität* (text font and precision) zur Auswahl eines Fonts und zur Festlegung, wie genau ein grafischer Arbeitsplatz die geräteunabhängigen Textattribute sowie aktuelle Transformationen und das Abschneiden am Fensterrand (Clipping) berücksichtigen muß.
- der *Textfarbindex* (text color index).

An **geometrischen Attributen** stehen zur Verfügung:

- Die *Zeichenhöhe* (character height) zur Festlegung der nominalen Höhe eines Großbuchstabens.
- Der *Zeichanaufwärtsvektor* (character up vector), der als im Ursprung beginnender Vektor die Aufwärtsrichtung eines Zeichens angibt.
- Die *Schreibrichtung* (text path), die angibt, ob nach rechts, links, oben oder unten geschrieben wird.
- Die *Textausrichtung* (text alignment), die die Positionierung des Textausdehnungsrechteckes in Relation zur Textposition festlegt.

Zum Verständnis der Textausrichtung muß man wissen, daß die Attribute Zeichenhöhe, Zeichenbreitefaktor, Schreibrichtung und Zeichenabstand sowie die Länge des Textes ein imaginäres, achsenparalleles Rechteck definieren, das *Textausdehnungsrechteck* genannt wird.

3.3.3 Linien-Attribute

Linien besitzen wir Polymarker nur **nichtgeometrische Attribute**, nämlich

- den *Linientyp* (line type) zur Auswahl, ob die Linie durchgezogen, strichliert, punktiert oder strichpunktiert sein soll, um nur einige Beispiele zu nennen,
- den *Linienbreitefaktor* (line width scale factor) zum Steuern der Strichstärke von Linien und jenen Darstellungsobjekten, die als Linienobjekte behandelt werden (z. B. Splines) und
- den *Linienzugfarbindex* (polyline color index).

Die Berücksichtigung der unterschiedlichen Linientypen kann durch einen Zähler realisiert werden, der das Verhältnis von angezeigten zu freigelassenen Punkten der Linie steuert. Punktierte Linien könnten z. B. durch Anzeige jedes dritten berechneten Punktes erzeugt werden.
Die Darstellung von Kurven mit einer Strichstärke größer eins – auch *Brushing* genannt – ist etwas schwierig. Stellt man jedes Pixel der Kurve als gefülltes Quadrat mit der aus dem Linienbreitefaktor resultierenden Kantenlänge dar, so ist dies zwar relativ einfach

zu realisieren, bietet aber den Nachteil, daß die Strichstärke einer solcherart gezeichneten Kurve vom Anstieg abhängt.
Eine andere Realisierung des Brushing besteht in der Berechnung der Kontur der Kurve, die dann mit geeigneten Füllalgorithmen gefüllt wird.

3.3.4 Füllgebiet-Attribute

Füllgebiete besitzen folgende **nichtgeometrischen Attribute**, nämlich

- die *Füllgebietsausfüllung* (interior style), die festlegt, wie die Fläche gefüllt wird (hohl, leer (nur PHIGS), voll, schraffiert und Muster),
- der *Füllgebietsausfüllungsindex* (interior style index), der für die Fälle der Schraffur und des Musters die Entsprechung auswählt und
- der *Füllgebietsfarbindex* (interior color index).

Weiters existieren folgende **geometrische Attribute**:

- die *Musterreferenz* (pattern reference point und reference vectors) zur Ausrichtung des Musters und
- die *Mustergröße* (pattern size)

Ganz allgemein bieten sich zur Füllung beliebiger Objekte zwei Ansätze an:

- *Scan-Line*-Verfahren, die eine *geometrische Beschreibung der Begrenzungskurve* voraussetzt, und
- Verfahren die die *Zusammenhangseigenschaften der Pixel* im Inneren eines zu füllenden Objektes ausnützen – *boundary fill* und *interior fill*.

3.3.5 Scan-Line-Algorithmus

Der *Scan-Line*-Algorithmus verwendet die Schnittpunkte zwischen der Begrenzungskurve und der Scan-Line, um die Pixel im Inneren des Objekts zu berechnen. Um die Berechnung der Schnittpunkte möglichst zu vereinfachen, wird als Scan-Line eine zur x- oder y-Achse parallele Gerade gewählt. Für den folgenden Algorithmus wird die Scan-Line als zur x-Achse parallel angenommen. Außerdem beschränkt sich der Algorithmus auf das Füllen von Polygonen.
Im wesentlichen besteht der Algorithmus aus den folgenden Schritten:

- Sortierung der Polygonkanten nach den größten y-Werten.
- Beginnend mit dem größten y-Wert bewegt sich die Scan-Line zum kleinsten y-Wert.

- Für jede Position der Scan-Line
  - wird die Liste der zu betrachtenden Polygonkanten aktualisiert,
  - werden die Schnittpunkte berechnet und nach den x-Werten sortiert und
  - jene Teile der Scan-Line, die im Inneren der Polygons liegen, angezeigt.

Es können einige *Sonderfälle* auftreten, falls ein Endpunkt des Polygons auf der Scan-Line liegt. Allgemein gilt, daß jeder Punkt im Inneren des Polygons liegt, wenn ein von ihm ausgehender Halbstrahl die Polygonkanten ungeradzahlig oft schneidet.
Die Korrektur erfolgt durch eine Sonderbehandlung von Endpunkten, indem jene Endpunkte, deren angrenzende Kanten beide oberhalb oder beide unterhalb des Halbstrahls liegen, nicht als Schnittpunkte gezählt werden. Liegt eine Kante oberhalb und die andere unterhalb des Halbstrahls, so wird dieser Endpunkt einfach gezählt.

Sind die Punkte $P_i = (x_i, y_i)$ und $P_{i+1} = (x_{i+1}, y_{i+1})$ Schnittpunkte einer Kante mit den Scan-Lines i und $i+1$, so läßt sich der Anstieg der Kante durch $s = \frac{y_{i+1}-y_i}{x_{i+1}-x_i}$ beschreiben. Da die Abstände der Scan-Lines gleich Eins sind, ergibt sich $x_{i+1} = x_i - \frac{1}{s}$. Beginnend mit dem Startpunkt der Kante berechnen sich aufeinanderfolgende Schnittpunkte mit der Scan-Line durch Subtarktion des invertierten Anstiegs.

3.3.6 Boundary-Fill und Interior-Fill

Sie beruhen auf den Zusammenhangseigenschaften von Pixeln und führen den Füllprozeß kreisförmig von einem vorgegebenen Punkt aus durch.

Beim *Boundary-Fill* wird dem Algorithmus ein Startpunkt (x, y), die Farbe der Begrenzungskurve(n) und eine Füllfarbe (bzw. Muster) übergeben. Von dem Punkt (x, y) ausgehend werden solange alle benachbarten Pixel $(x \pm 1, y \pm 1)$ umgefärbt, bis ein Pixel mit der Farbe der Begrenzungskurve(n) oder der Rand der Zeichenfläche erreicht wird. An dieser Stelle bricht die Rekursion ab. Die Rekursion bedingt aber auch den Nachteil, daß Pixels meistens mehrmals auf dem Stapel liegen.

Im Gegensatz zum Boundary-Fill wird beim *Interior-Fill* dem Algorithmus ein Startpunkt, die Farbe der umzufärbenden Pixel und eine Füllfarbe (bzw. Muster) übergeben. Es werden also nicht alle Punkte innerhalb einer einfarbigen Begrenzungskurve gefärbt, sondern von einem Startpunkt ausgehend alle Punkte gleicher Farbe umgefärbt. Die Begrenzung der "inneren" Fläche ist beim Interior-Fill die Menge aller Pixel, deren Farbe ungleich der zu ersetzenden Farbe ist und die 4-way-stepping-Nachbarn der umzufärbenden Pixel sind. Problematisch wird es aber genau bei diesen 4- oder 8-Nachbarschaften. Um dieses Problem zu umgehen, muß man den Rand des zu füllenden Objektes zuerst dichtmachen.

3.3.7 Gebündelte Attribute

Soll ein Anwenderprogramm mit verschiedenen Ausgabegeräten betrieben werden, von denen unter Umständen noch nicht bekannt ist, welche Darstellungsfunktionen sie anbieten, so ist die Verwendung von den vorher beschriebenen *individuellen* Attributen nicht zielführend. Somit führte man die *gebündelten* Attribute ein, die man als einen Satz aller *nichtgeometrischen* Attribute eines Darstellungselementes versteht.

Beispiel für gebündelte Attribute:

- **Linienelemente:** Linientyp, Linienbreitefaktor, Linienzugfarbindex

Um die Flexibilität bei der Attributsauswahl zu erhöhen, stehen dem Programmierer noch *Aspektanzeiger* (aspect source flags – ASF) zur Verfügung, die für jedes Darstellungselement festlegen, welche Attribute der entsprechenden Kategorie durch die Bündeltabelle definiert werden und welche Attribute die individuell gesetzten Werte erhalten.

3.3.8 Tabelle der Attribute

Marker	Typ Vergrößerungsfaktor Farbe
Text	Zeichenbreitefaktor Zeichenabstand Schriftart und -qualität Farbe Zeichenhöhe Zeichenaufwärtsvektor Schreibrichtung Textausrichtung
Linie	Typ Breitefaktor Farbe
Füllgebiet	Ausfüllung Ausfüllungsindex Farbe Musterreferenzpunkt Mustergröße

3.4 Segmente und Strukturen

3.4.1 GKS

Hinter dem Konzept des Segments verbirgt sich die Gruppierung von Darstellungselementen zu Gebilden, die als Gesamtheit mit Attributen versehen, verwaltet, manipuliert und angezeigt werden können. Zum Beispiel im CAD-Bereich erfolgt dann die Anzeige über die Referenz "Tisch", "Badewanne", "Sitzgruppe" usw. ohne daß diese Objekte jedesmal explizit aus ihren Darstellungselementen aufgebaut werden müssen.
Darüberhinaus sind in GKS auch andere Funktionen eines Grafiksystems wie das Picken oder die 2D-Transformation von Objekten mit dem Konzept des Segments verbunden. Die wesentlichsten Punkte dabei sind:

- bei der Regenerierung von Bildern auf einem grafischen Ausgabegerät (z. B. nach der Modifikation von Objekten) werden nur die in Segmenten gespeicherten Darstellungselemente angezeigt;
- beliebige 2D-Transformationen können nur auf Darstellungselemente in Segmenten angewendet werden (die achsenparallelen Koordinatentransformationen wirken sich natürlich auf alle Ausgabeelemente aus);
- nur in Segmenten gespeicherte Darstellungselemente können gepickt werden.

Für die Verwendung von Segmenten sind in GKS zusätzlich die beiden folgenden Regeln zu beachten:

- Die Attribute der Darstellungselemente werden zum Zeitpunkt der Segmenterstellung fix an die Elemente gebunden.
- Es gibt *keine hierarchische Struktur von Segmenten*, sondern nur eine einstufige Nebeneinanderstellung.

Aus diesen beiden Gründen wurde bei PHIGS die *dynamische Attributzuordnung* und die *Segment-Hierarchien* von Anfang an berücksichtigt.

3.4.2 PHIGS

Segmente heißen Strukturen.

3.5 Eingabeprogramierung

3.5.1 Logische Klassifizierung der Eingabegeräte

- **Lokalisierer** (Locator) liefern eine Position (x, y).

  Tablett, Maus, Joystick, Trackball, Thumb Wheels, Tastatur mit Cursor-Tasten

- **Liniengeber** (Stroke) liefert eine Folge von Positionen.

  Tablett, Maus, eventuell Joystick

- **Wertgeber** (Valuator) liefert einen skalaren Wert (reelle Zahl).

  Paddles, Thumb Wheels, eventuell Tastatur

- **Auswähler** (Choice) liefert die Auswahl aus einer Anzahl von Möglichkeiten (nicht-negative ganze Zahl).

  Berührungsempfindlicher Schirm, Lichtgriffel, Tablett, Maus, Tastatur, Joystick

- **Picker** (Pick) liefert die Auswahl aus am Schirm dargestellten und selektierbaren Bildelementen.

  Tablett, Maus, Lichtgriffel, Joystick

- **Textgeber** (String) liefert eine Zeichenfolge.

  Tastatur

3.5.2 Eingabemodi

In GKS und in PHIGS unterscheidet man pro logischem Eingabegerät

- *Anforderungs*-Modus (request mode) wartet auf eine Eingabe
- *Abfrage*-Modus (sample mode) für Geräte, die ständig Werte liefern
- *Ereignis*-Modus (event mode) buffert die Daten in einer *Eingabewarteschlange* (event queue) und ermöglicht so die parallele Verwendung der Modi

3.5.3 Picking

Picking beschreibt die Objekterfassung innerhalb eines Bildes.

Man trifft folgende Unterscheidung für die verschiedenen Anzeigearten:

- *Vektorrefreshsysteme:* Interrupt durch Lichtgriffel ausgelöst
- *Rastersystem:* Koordinatenabfrage vom *Picker* und anschließende Feststellung des Objekts

Generelle Probleme beim Picking:

1. Benutzer muß nicht exakt auf ein Objekt zeigen (v. a. wegen hoher Auflösung notwendig).
2. Gefüllte Objekte, bei denen ein innerer Punkt weit vom Rand entfernt ist.
3. Verdeckte Objekte sollten auswählbar sein (GKS: ja, PHIGS: nein).

Häufigste Implementierungsart:

- Rechteck um jedes Objekt
- gepicktes Pixel innerhalb eines einzigen Rechteckes $\Rightarrow$ klar
- Bei mehreren Rechtecken ermittelt man das näheste Objekt mittels Abstandsfunktion.

In **GKS** kann ein Segment (Darstellungselement) eine *Pickerkennzeichnung* haben, welches das einzige Identifizierungsattribut in GKS darstellt.

- Elemente außerhalb von Segmenten sind nicht ansprechbar.
- Nur wenn ein Segment sichtbar gesetzt und ansprechbar ist, kann es gepickt werden.
- Die *PickId* kann nur bei Segmenten gesetzt werden.
- Gruppen können gleiche *PickId* aufweisen.
- Bei überlappenden Darstellungselementen aus verschiedenen Segmenten mit unterschiedlicher Priorität wird das mit der höchsten Priorität identifiziert.
- Was bei gleicher Priorität erfolgt, ist abhängig von der Implementierung (z. B. können alle Objekte in der Umgebung geliefert werden).

In **PHIGS** können nur auf dem Bildschirm sichtbare und pickbar gesetzte Elemente gepickt werden. In PHIGS wird die Fähigkeit, gepickt werden zu können, über *Namen* gesteuert.

Der *Picker* liefert den *Pick-Pfad*, welcher die gesamte Strukturhierarchie angibt, in der das gepickte Element sich befindet.

3.5.4 Eingabe-Techniken

- **Objekte** können entweder mittels
  - *Pick & Place* oder
  - *direkt*

  positioniert werden.

- **Linien** können folgendermaßen gezeichnet werden:
  - *frei*,
  - mit *horizontaler Beschränkung*,
  - mit *vertikaler Ausrichtung*,
  - mittels einer *Mischform* der beiden vorgenannten,
  - unter Zuhilfenahme eines *Gitters* oder
  - durch Anwendung der *Rubber-Band*-Technik.

4 Darstellung allgemeiner Kurven

Durch **Interpolation** entstandene Kurven verlaufen zwischen vorgegebenen Stützstellen (*Splines*).

Durch **Approximation** erzeugt man interaktive Formen, die vor allem auch ästhetischen Gesichtspunkten genügen müssen (hs. *Bézier-Approximation*).

4.1 Interpolation von Kurven

Mit einem algebraischen Ansatz läßt sich das Problem für n Punkte durch ein Polynom $(n-1)$-ten Grades (mit nicht zu vernachläßigendem Rechanaufwand) lösen:

$$y = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1 x + a_0$$

Der Nachteil dieser Lösung liegt in der resultierenden Kurve, die zwar durch die Stützpunkte verläuft, aber zwischen den Stützpunkten meistens oszilliert. Zusätzlich treten bei der Computerberechnung durch die hohen Potenzen enorme Rechenungenauigkeiten auf. Ein zufriedenstellendes Ergebnis erhält man durch *stückweise polynomielle Kurven*, den sogenannten *Splines*.
Wählt man zwischen je zwei Punkten Kurven erster Ordnung (d. h. Geradenstücke), so erhält man den bereits bekannten Linienzug.
Glattere Kurven entstehen durch Kurven zweiter Ordnung (Parabeln). Allerdings ergibt sich das Problem, daß schon die ersten Ableitungen an den Intervallgrenzen im allgemeinen nicht stetig sind. Deshalb verewendet man meistens Kurven dritter Ordnung mit stetigen ersten und zweiten Ableitungen an den Intervallgrenzen.

4.1.1 Kubische Splines

Für die Interpolation einer Kurve $f(t)$, die durch n vorgegebene Punkte verlaufen soll, wird die Funktion in $n-1$ Intervalle zerlegt und in jedem Teilintervall $t \in [t_i, t_{i+1}]$ durch eine eigene Funktion $f_i(t), i = 1, \ldots, n-1$ dargestellt. Die n Stützpunkte der zu interpolierenden Kurve f liegen in den Punkten $(t_i, f(t_i)), i = 1, \ldots, n$.
Da es sich bei der Annäherung um *kubische Splines* handelt, hat jede Funktion $f_i, i = 1, \ldots, n-1$ die Form

$$\begin{aligned} f_i(t) &= f_i(t_i) + a_i \cdot (t - t_i) + b_i \cdot (t - t_i)^2 + c_i \cdot (t - t_i)^3, \\ &\quad t_i \leq t \leq t_{i+1}, i = 1, \ldots, n-1 \end{aligned}$$

Für t_{i+1} folgt

$$f_i(t_{i+1}) = f_i(t_i) + a_i \cdot (t_{i+1} - t_i) + b_i \cdot (t_{i+1} - t_i)^2 + c_i \cdot (t_{i+1} - t_i)^3$$

bzw. für $\Delta t_i = t_{i+1} - t_i, i = 1, \ldots, n-1$

$$f_i(t_{i+1}) = f_i(t_i) + a_i \cdot \Delta t_i + b_i \cdot \Delta t_i^2 + c_i \cdot \Delta t_i^3$$

In die Form der *geteilten Differenzen* D_i gebracht, gilt

$$D_i = \frac{f_i(t_{i+1}) - f_i(t_i)}{\Delta t_i} = a_i + b_i \cdot \Delta t_i + c_i \cdot \Delta t_i^2 \tag{1}$$

Aus der Forderung der stetigen ersten Ableitungen an den Intervallgrenzen folgt

$$f_i'(t_{i+1}) = f_{i+1}'(t_{i+1})$$

oder

$$a_i + 2b_i \cdot \Delta t_i + 3c_i \cdot \Delta t_i^2 = a_{i+1}$$

Durch Einsetzen von Gleichung (1) ergibt sich

$$c_i = \frac{a_i + a_{i+1} - 2D_i}{\Delta t_i^2} \tag{2}$$

Da auch die zweite Ableitung an den Intervallgrenzen stetig sind, gilt

$$f_i''(t_{i+1}) = f_{i+1}''(t_{i+1})$$

oder

$$b_i + 3c_i \cdot \Delta t_i = b_{i+1}$$

Daraus ergibt sich durch Gleichung (1)

$$\frac{D_i - a_i}{\Delta t_i} + 2c_i \cdot \Delta t_i = \frac{D_{i+1} - a_{i+1}}{\Delta t_{i+1}} - c_{i+1} \cdot \Delta t_{i+1}$$

Die Verwendung von Gleichung (2) führt zu

$$\begin{aligned} a_i \Delta t_{i+1} + 2a_{i+1}(\Delta t_i + \Delta t_{i+1}) + a_{i+2}\Delta t_i &= 3(D_i \Delta t_{i+1} + D_{i+1}\Delta t_i) \\ & i = 1, \ldots, n-2 \end{aligned} \tag{3}$$

Unter der Annahme, daß die beiden Parameter a_1 und a_n (Anstieg im ersten und letzten Stützpunkt) entsprechend gewählt wurden, ergibt sich aus Gleichung (3) ein lineares Gleichungssystem mit $n-2$ Gleichungen für $n-2$ Unbekannte.

Somit ist nur noch die Berechnung der Werte a_1 und a_n durchzuführen. Ein häufig angewendetes Verfahren, welches auch *natürliche Spline-Interpolation* genannt wird, beruht auf dem Ansatz $f''(t_1) = f''(t_n) = 0$, der außer für den Fall, daß die 2. Ableitung der angenäherten Kurve wirklich gleich Null sind, einen Fehler von $O(t^2)$ an den Enden aufweist.

Für den Fall, daß die Ableitungen in den Endpunkten nicht bekannt sind, erweist es sich als vorteilhaft, die Funktionen $f_1 = f_2$ und $f_{n-2} = f_{n-1}$ zu setzen. Dadurch wird der zweite und vorletzte Punkt nicht als Stützpunkt verwendet, obwohl die erzeugte Kurve durch diese Punkte verläuft. Das erste und letzte Spline-Stück wird dadurch über 3 Punkte und eine Tangente anstatt über 2 Punkte und 2 Tangenten berechnet. Für a_1 und a_n ergeben sich die Gleichungen

$$a_i \Delta t_2 + a_2(t_3 - t_1) = \frac{\Delta t_2(\Delta t_1 + 2(t_3 -_1))D_1 + \Delta t_1^2 D_2}{t_3 - t_1}$$

$$\text{und} \tag{4}$$

$$a_n \Delta t_{n-2} + a_{n-1}(t_n - t_{n-2}) = \frac{\Delta t_{n-2}(\Delta t_{n-1} + 2(t_n - t_{n-2}))D_{n-1} + \Delta t_{n-1}^2 D_{n-2}}{t_n - t_{n-2}}$$

Konkret sieht für $n = 4$ das durch (3) und (4) festgelegte Gleichungssystem folgendermaßen aus:

$$\begin{bmatrix} \Delta t_2 & t_3 - t_1 & 0 & 0 \\ \Delta t_2 & 2(\Delta t_1 + \Delta t_2) & \Delta t_1 & 0 \\ 0 & \Delta t_3 & 2(\Delta t_2 + \Delta t_3) & \Delta t_2 \\ 0 & 0 & t_4 - t_2 & \Delta t_2 \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} K_1 \\ 3(D_1 \Delta t_2 + D_2 \Delta t_1) \\ 3(D_2 \Delta t_3 + D_3 \Delta t_2) \\ K_4 \end{bmatrix}$$

wobei K_1 und K_4 die konstanten Terme aus den Gleichungen (4) sind.

Zur Wahl der t-Werte reicht eine streng monoton steigende Folge, damit die Reihenfolge der Punkte entlang der Kurve der Eingabefolge entspricht.
Besser eignet sich der *euklid'sche Abstand* zwischen zwei aufeinanderfolgenden Punkten

$$d_i = d(P_{i+1}, P_i) = +\sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}, \qquad i = 1, \ldots, n-1$$

Beginnend mit dem Wert $t_i = 0$ wird $t_{i+1} = t_i + d_i, \quad i = 1, \ldots, n-1$ gesetzt.

4.2 Approximation von Kurven

4.2.1 Bézier-Kurven

Die dabei verwendeten Kurven $\mathcal{P}(t)$ — Bézier-Kurven vom Grad n — werden über $n+1$ Stützpunkte P_i in Vektorform durch die Gleichung

$$\mathcal{P}(t) = \sum_{i=0}^{n} B_{i,n}(t) \cdot P_i, \qquad 0 \leq t \leq 1$$

bestimmt. $\mathcal{P}(t)$ ist ein d-dimensionaler Vektor und $B_{i,n}(t)$ sind die nach dem Mathematiker Sergei N. Bernstein benannten *Bernsteinpolynome* der Form

$$B_{i,n}(t) = \frac{n!}{i! \cdot (n-i)!} \cdot t^i \cdot (1-t)^{n-i}$$

Durch diese Polynome wird der Einfluß der einzelnen Stützpunkte P_i auf den Kurvenverlauf festgelegt.

Die wichtigsten Eigenschaften der Bernsteinpolynome sind:

- Alle Bernsteinpolynome sind positiv
- Für ein beliebiges, aber fixes t gilt $\sum_{i=0}^{n} B_{i,n}(t) = 1$
- $B_{i,n}(t) = B_{n-i,n}(1-t)$

Wesentliche Eigenschaften der Bézier-Kurve sind:

- Die Anzahl der Stützpunkte bestimmt den Grad der Kurve.
- Die Kurve geht durch P_0 und P_n.
- Sie beginnt im Stützpunkt P_0 tangential zur Geraden $\overline{P_0P_1}$, endet im Stützpunkt P_n tangential zur Geraden $\overline{P_{n-1}P_n}$.
- Außerdem verläuft sie innerhalb der konvexen Hülle der Stützpunkte.

Die *Berechnung* erfolgt mit dem Algorithmus von **De Casteljau**, der sich aus der mathematischen Definition der Kurve herleiten läßt und zur Erkenntnis gelangt, daß sich eine Bézier-Kurve vom Grad n durch zwei Bézier-Kurven vom Grad $n-1$ darstellen läßt.
Eine Möglichkeit, die Effizienz des Algorithmus zu verbessern, liegt in der Vorausberechnung der Binomialkoeffizienten und der Anwendung des Horner-Schemas zur Berechnung der Terme t^i bzw. $(1-t)^j$.

Vorteile:

- leichte Formgebung durch die Polygondefinition
- keine Angabe von Tangentenvektoren notwendig
- glättende Wirkung

Nachteile:

- höhere Stellenanzahl, daher steigt der Grad und damit die Rechenzeit
- Änderung einer Stelle bewirkt die Änderung der gesamten Kurve, sprich es gibt keine lokalen Änderungen.

Ziel:

Vermeidung der Nachteile unter Wahrung der Vorteile.

Erreicht wird dies durch:

- segmentweise definiertes Polynom
- gute Stetigkeit an den Segmentübergängen
- Erhöhung des Freiheitsgrades ohne Erhöhung des Polynomgrades

Aus diesen Gründen verwendet man das Verfahren der *B-Splines*.

4.2.2 B-Splines

Sie sind eine Verallgemeinerung von Bézier-Kurven.

Der wesentlichste Unterschied ist, daß

- die Stützpunkte nur einen lokalen Einfluß ausüben und
- daß der Grad der Polynome unabhängig von der Anzahl der Punkte ist.

Daraus folgt, daß sich Änderungen eines Stützpunktes nur lokal auf die B-Splinekurve bzw. -fläche auswirken.

Die Definition der *uniformen B-Splinekurve* $\mathcal{P}(t)$ vom Grad $k-1$ erfolgt über $n+1$ Stützpunkte $P_i, \quad 0 \leq i \leq n$, und einen Vektor von ganzen Zahlen, den *Knotenvektor* $T = (t_0, \ldots, t_n + k), \quad t_j \leq t_{j+1}$, in der Parameterform durch

$$\mathcal{P}(t) = \sum_{i=0}^{n} N_{i,k}(t) \cdot P_i$$

Hier ist $N_{i,k}(t)$ ein **normierter** (bzw. Intervall-)B-Spline der Ordnung k (Grad $k-1$), wenn gilt:

1. $N_{i,k}(t) = 1$ für $t \in [t_{k-1}, t_{n+1}]$

2. $N_{i,k}(t)$ ist Polynom vom Grad $k-1$ in jedem Intervall $[t_i, t_{i+1}]$

3. $N_{i,k}(t) \in C^{k-2}$ in $[t_0, t_n]$

4. $N_{i,k}(t) = 0$ für $t < t_i$ und $t \geq t_{i+k}$

Die *Basis* des B-Splines sind die Polynome $N_{i,k}(t)$ vom Grad $k-1$, die durch folgende Rekursionsformel berechnet werden:

$$\begin{aligned} N_{i,1}(t) &= \begin{cases} 1 & \text{falls} \quad t_i < t_{i+1} \quad \text{und} \quad t_i \leq t \leq t_{i+1} \\ 0 & \text{sonst} \end{cases} \\ N_{i,k}(t) &= \frac{t - t_i}{t_{i+k-1} - t_i} \cdot N_{i,k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} \cdot N_{i+1,k-1}(t), \quad k > 1 \end{aligned}$$

Man unterscheidet *zwei* Arten von B-Splines:

- Für *uniforme* B-Splines gilt $t_{i+1} = t_i$ oder $t_{i+1} = t_i + 1$ (einheitlicher Intervall-Abstand).

- Bei *nonuniformen* B-Splines muß das nicht gelten.

4.3 Flächendarstellung

4.3.1 Analytisch gegebene Flächen

z. B. Beschreibung einer *Kugel*:

$$\begin{aligned} x(s,t) &= x_z + r \cdot \sin(\pi s) \cdot \cos(\pi t) \\ y(s,t) &= y_z + r \cdot \cos(\pi s) \\ z(s,t) &= z_z + r \cdot \sin(\pi s) \cdot \sin(\pi t) \\ & \text{für} \quad s,t \in [0,1] \end{aligned}$$

4.3.2 Freiformflächen

Polygonflächen: werden durch ebene Flächenstücke dargestellt, die durch Polygonzüge umrandet sind (= Polygon). Man verwendet dazu bevorzugt *Dreiecke*.

Vorteile:

- gute Approximation möglich

- Feinheiten können durch Verkleinern der Polygone sichtbar gemacht werden

Nachteile:

- starke Vermehrung der Flächen und damit der zu speichernden Daten

- Problematik äquivalent zu der bei der Approximation von Kurven durch Polygonzüge.

4.4 Fraktale

Nach *Mandelbrot* lassen sich fraktale Objekte als Mengen nichtganzzahliger *Hausdorff-Dimension* charakterisieren:

> Ist X eine Menge im m-dimensionalen Raum und $n(\varepsilon)$ die Anzahl der m-Dimensionalen Kugeln mit dem Durchmesser ε, die ausreichen, um X zu überdecken, dann hat X die *Hausdorff-Dimension* D, wenn für $\varepsilon \to 0$ die Anzahl $n(\varepsilon)$ proportional zu ε^{-D} ist.

$$D = \frac{\lg(N)}{\lg(1/s)}$$

N	...	Anzahl der Linien im erzeugten Linienzug
s	...	Skalierungsfaktor

Im Falle einer *quadratischen* Koch-Kurve entspricht N gleich 8 erzeugten Linien und s gleich 4 Einheiten kleiner, wodurch sich eine Hausdorff-Dimension von

$$D = \frac{\lg(8)}{\lg(1/4)} = 1.5$$

ergibt.

5 3D-Modelle und deren Transformationen

5.1 3D-Grafik

Ein 3D-Grafikpaket besteht aus

- einem 2D-Paket erweitert um
- 3D-Darstellungselemente (versch. Projektionen, Entfernung verdeckter Linien oder Flächen ...)

Die Hauptanwendungsgebiete umfaßen

- die *Darstellung existierender Objekte*, bei denen die geforderte Genauigkeit Punkt für Punkt die originale Oberfläche beträgt (Dies wären zu viele Punkte, daher geht man zu Modellen über — sprich ausreichend viele Punkte — und ergänzt diese um *Interpolationsvorschriften* zur Berechnung weiterer Punkte.) und
- *CAD* zur Erzeugung neuer, relativ einfacher Objekte durch
  - geeignete Darstellungs- und Konstruktionselemente und
  - interaktive Funktionen.

5.2 Darstellung von Objekten (interne Repräsentation)

5.2.1 Randdarstellung

Bei der *Randdarstellung* unterscheidet man folgende *zwei* Arten:

- **Drahtmodelle** haben den Nachteil, daß sie u. U. nicht eindeutig sind und daher für Modelle nicht geeignet sind.
- **Flächenmodelle** ergänzen die *Kanteninformation* um die *Flächeninformation*, wodurch aber
  - die *Konsistenzprüfung* und
  - die *Eingabe komplexer Gebilde*

  sehr aufwendig ausfallen.

  Der Vorteil liegt darin, daß eine *eindeutige Objektdarstellung* erreicht werden kann.

5.2.2 Volumendarstellung

Bei der *Volumendarstellung* unterscheidet man folgende *zwei* Arten:

- Die **Voxeldarstellung** (voxel $\equiv$ pixel im dreidimensionalen Raum) hat den Vorteil der *Eindeutigkeit* aber die Nachteile, daß sie aus *anderen Darstellungen* gewonnen werden muß und daß die *Oberflächenbestimmung* schwierig ist.

  Zur Speicherung bei Voxeldarstellung bietet sich die *Octree*- (entwickelt aus der *Quadtree*-)Struktur an.

- Die **Konstruktion mit Raumprimitiven** baut ein Objekt aus gegebenen Raumprimitiven unter Verwendung *mengentheoretischer Operationen* auf. Dies bietet den Vorteil, daß die Eingabe relativ einfach ist, zieht aber die Nachteile mit sich, daß *komplexe Strukturen* sehr aufwendig zu modellieren sind und daß *Oberflächenberechnungen* sehr komplex sind.

5.3 3D-Koordinatensysteme

Es werden meist *kartesische Koordinaten* verwendet.

- **rechtshändiges** und
- **linkshändiges** System

5.4 Anzeigetechniken

Hiermit verfolgt man das **Ziel** der *Darstellung einer 3D-Szene auf einem Ausgabemedium* mit dem **Hauptproblem** der *räumlichen Tiefe*.

Methoden:

- *Projektionen:*
  - *Parallel*-Projektion
  - *perspektivische* Projektion
- *Entfernung unsichtbarer Teile:*
  - *Hidden-Line-Removal:*
    - ganz entfernen
    - stricheln
  - *Hidden-Surface-Removal*
- *Intensitätsabstufung*

- *Schattierungsmodelle*
- *Stereobilder*
- *Explosionszeichnungen*

5.5 Transformationen

Unter *Transformationen* versteht man Operationen auf allen Punkten eines (Teil-)Modelles bzw. -Bildes.

Da im Bereich von *CAD, Grafik-Editoren ...* Manipulationen von Darstellungen notwendig sind, verwendet man die im Folgenden behandelten *Transformationen* unter Zuhilfenahme von *homogenen Koordinaten* (hier: $(x, y, z, 1)$, um das Problem des Ursprunges zu umgehen).

In Grafik-Paketen werden gewissen Objekten Transformationsmatrizen zugeordnet, welche mit der *Einheitsmatrix* initialisiert werden.

Vorteil:

- Transformationen werden akkumuliert erst bei der Ausgabe durchgeführt und nicht einzeln

5.5.1 Elementare Transformationen

- **Translation** ist die Verschiebung eines Objektes um einen Translationsvektor T, d. h. jeder Punkt des Objektes wird um den Vektor $T = (t_x, t_y, t_z)$ verschoben:

$$T(t_x, t_y, t_z) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- **Skalierung:** Ein mit den Skalierungswerten (s_x, s_y, s_z) skalierter Punkt P hat die Koordinaten $P' = (x', y', z') = (x \cdot s_x, y \cdot s_y, z \cdot s_z)$, wenn der Fixpunkt der Skalierung im Ursprung liegt. Die daraus resultierende Transformationsmatrix lautet:

$$S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- **Rotation:** Wie im zweidimensionalen Fall, der in einem rechtshändigen Koordinatensystem als *Rotation um die z-Achse* gesehen werden kann, bewirkt ein positiver Winkel eine Drehung gegen den Uhrzeigersinn. Die Koordinaten des um die z-Achse gedrehten Punktes $P = (x, y, z)$ sind somit für den Drehwinkel δ

  $$\begin{aligned} x' &= x \cdot \cos(\delta) - y \cdot \sin(\delta) \\ y' &= x \cdot \sin(\delta) + y \cdot \cos(\delta) \\ z' &= z \end{aligned}$$

  Die zugehörige Transformationsmatrix für die homogenen Koordinaten lautet

  $$R_z(\delta) = \begin{bmatrix} \cos(\delta) & \sin(\delta) & 0 & 0 \\ -\sin(\delta) & \cos(\delta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  Bei einer Drehung um die x-Achse ergibt sich

  $$R_x(\delta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\delta) & \sin(\delta) & 0 \\ 0 & -\sin(\delta) & \cos(\delta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  Bei einer Drehung um die y-Achse ergibt sich

  $$R_y(\delta) = \begin{bmatrix} \cos(\delta) & 0 & -\sin(\delta) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\delta) & 0\,\cos(\delta) & 0 & \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  Für den Fall einer Rotation *um eine beliebige Achse* geht man folgendermaßen vor:

  1. Translation der Rotationsachse (und des zu drehenden Objektes), sodaß die Achse durch den Ursprung verläuft.
  2. Rotation der Drehachse um die x-Achse in die xz-Ebene.
  3. Rotation der Drehachse um die y-Achse in die z-Achse.
  4. Rotation des Objekts um die z-Achse (diese entspricht jetzt der Drehachse) mit dem Winkel δ.
  5. Rücktransformation der Drehachse (sowie des gedrehten Objektes) in ihre ursprüngliche Lage durch Anwendung der inversen Transformationen der Schritte (3), (2) und (1).

5.5.2 Allgemeine Transformationen (affine Abbildungen)

Dazu zählen

- die **Spiegelung** mittels

$$SP_{xy} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  und

- die **Scherung** z. B. an der x-Achse im dreidimensionalen Fall mittels

$$SH_x = \begin{bmatrix} 1 & a & b & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  oder im zweidimensionalen Fall um zwei Einheiten z. B. mittels

$$\begin{bmatrix} x+2 \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

5.5.3 Zusammengesetzte Transformationen

Die *zusammengesetzte Transformation* erfolgt durch *Hintereinanderschalten* einer Folge von Transformationen. Bei gegebenen Matrizen ergibt sich die zusammengesetzte Transformationsmatrix als Matrixprodukt.

Beispiele:

- *Rotation um eine beliebige Achse*
- *Spiegelung*

5.5.4 Rastertransformationen (bit block transfers)

Darunter versteht man die *direkte Manipulation* der Pixel im Bildwiederholspeicher.

Beispiele:

- *Kopieren* eines rechteckigen Pixel-Bereichs
- *Rotation*
- *Vergrößern*
- *Invertieren*

5.5.5 Klippen

Darunter versteht man die Entfernung bzw. die Weglassung von jenen Objekten, die ganz oder teilweise außerhalb des abzubildenden Fensters liegen.

Um dies zu erreichen, verwendet man folgende Algorithmen:

Klippen von Polymarkern:

Für jeden Punkt P_i wird der Test bezüglich der Fenstergrenzen durchgeführt

$$x_{min} \leq x_i \leq x_{max} \qquad y_{min} \leq y_i \leq y_{max}.$$

Klippen von Linien:

- **Cohen-Sutherland-Algorithmus:** Dieser Algorithmus besteht aus zwei Schritten:
    (a) Anfangspunkt P und Endpunkt Q wird folgender Code zugeordnet:
        Bit 1: links vom Fenster
        Bit 2: rechts vom Fenster
        Bit 3: unter dem Fenster
        Bit 4: über dem Fenster
    (b) Falls eine der folgenden Bedingungen erfüllt ist, ist der Vektor fertig bearbeitet. Und zwar ist der Vektor
        - *innerhalb* des Fensters wenn $\mathbf{Code}(P)$ *and* $\mathbf{Code}(Q) \neq \mathbf{0}$ oder
        - *außerhalb* des Fensters wenn $\mathbf{Code}(P)$ *or* $\mathbf{Code}(Q) = \mathbf{0}$.

    1. Eliminiere einen Teil des Vektors, den Rest behandle weiter mit Schritt 1.
        (a) eliminiere Vektorteil links vom Fenster
        (b) eliminiere Vektorteil rechts vom Fenster
        (c) eliminiere Vektorteil unterhalb vom Fenster
        (d) eliminiere Vektorteil oberhalb vom Fenster

- **Mittelpunktsalgorithmus:** vermeidet die teure Schnittpunkgsberechnung von Cohen-Sutherland dadurch, daß *Schritt 2 durch Halbieren der Strecke und anschließender separater Behandlung der Hälften* ersetzt wird.

Im *dreidimensionalen* Raum funktioniert der Algorithmus mit *6-stelligem Code.*

Klippen von Polygonen:

- **1. Methode:** *Als Menge von Vektoren bearbeiten.* Dabei ergibt sich das Problem, daß der Zusammenhang verlorengeht (Füllgebiete!)

- **2. Methode:** *Polygon wird als Folge von Vektoren bearbeitet.* Hier werden die Austrittspunkte mit den Eintrittspunkten durch zusätzliche Vektoren verbunden, wodurch sich nur mehr das Problem in den *Ecken* ergibt.

- **3. Methode:** *Sutherland-Hodgman-Algorithmus.* Es wird *zuerst* am *rechten* Rand, dann am *linken* Rand, dann am *unteren* Rand und schließlich am *oberen* Rand geklippt.

Klippen von Text:

Zur Textdarstellung gibt es folgende Möglichkeiten:

- wenn Textausdehnungsrechteck ganz innerhalb vom Fenster ist

- nur wenn der Zeichenkörper völlig innerhalb ist

- Schneiden des Zeichens an der Fenstergrenze

5.5.6 Projektionen

Allgemein wird unter einer *Projektion* die Abbildung eines Vektors in einem n-dimensionalen Vektorraum X auf einen Vektor in einem m-dimensionalen Unterraum U für $m < n$ verstanden.
Die Festlegung der Projektion erfolgt durch

- Angabe des *Projektionszentrums* (Augpunkt) und

- einer *Projektionsebene* (Bildebene).

Man unterscheidet folgende Arten der Projektion:

- *Perspektivische Projektion* (Zentralprojektion): Das Projektionszentrum befindet sich hier im Endlichen. Sie weist folgende Eigenschaften auf:

  - entfernt liegende Objekte werden proportional zur Entfernung von Bildebenen verkleinert. Distanzen zwischen verschiedenen Punkten werden in der Abbildung unterschiedlich verzerrt.
  - Winkel zwischen zwei Geraden werden nur dann treu abgebildet, wenn die Ebene, die beide Geraden enthält, parallel zur Bildebene liegt.
  - Parallele Geraden, die nicht parallel zur Bildebene sind, laufen in einem *Verschwindungspunkt* zusammen.
  - Wenn die Bildebene die Hauptachse schneidet, schneiden alle zu dieser Achse parallelen Geraden diese Hauptachse in einem speziellen Verschwindungspunkt, dem *Fluchtpunkt*.

Die Anzahl der *Fluchtpunkte* (Anzahl der Hauptachsen, die die Bildebene Schneiden) wird zur Charakterisierung der Projektion herangezogen (1-Punkt-Perspektive, 2-Punkt-Perspektive, ...).

- *Parallelprojektion:* Das Projektionszentrum liegt hier im *Fernpunkt* — sprich im Unendlichen — und die *Projektionsstrahlen* sind zueinander parallel.

  Stehen die Sehstrahlen normal zur Bildebene, so handelt es sich um **orthogonale** Projektionen oder **Normalprojektionen**, andernfalls spricht man von **schiefen (oblique)** Projektionen.

  - *orthogonale* Parallelprojektion: Eine der Hauptachsen steht senkrecht zur Bildebene.
  - *axonometrische* Parallelprojektion: Keine der Hauptachsen steht senkrecht zur Bildebene.
    - *isometrisch:* alle Hauptachsenabschnitte sind gleich lang.
    - *dimetrisch:* zwei Hauptachsenabschnitte sind gleich lang.
    - *trimetrisch:* alle Hauptachsenabschnitte sind verschieden lang.
  - *oblique* Parallelprojektion (Schrägriß):
    - *Kavalierprojektion:* Der Winkel α beträgt 45^0 und der *Verkürzungsfaktor* d beträgt 1.
    - *Kabinettprojektion:* Der Winkel α beträgt 30^0 und der *Verkürzungsfaktor* d beträgt $\frac{1}{2}$.

5.6 Die Viewingpipe von PHIGS

Modellkoordinaten	
↓	Zusammensetzung der Modellteile
Weltkoordinaten (einheitliches System für das Modell)	
↓	view orientation
View Reference Coordinates	
↓	perspektivische Transformation & Klippen
Normalisierte Projektionskoordinaten (Einheitswürfel)	
↓	Gerätetransformation
Gerätekoordinaten	

6 Farbmodelle

6.1 Eigenschaften von Licht und Auge

Die *Spektralfarben* rot, orange, gelb, grün, blau, indigo und violett decken den Bereich des sichtbaren Lichts von $4.3 \cdot 10^{14}$ Hz (rot) bis $7.5 \cdot 10^{14}$ Hz (violett) bzw. von ca. 780 nm bis 380 nm ab, wobei das Auge ungefähr 350 000 verschiedene Farben in diesem Bereich unterscheiden kann.

Eine Methode zur Darstellung einer Farbe ist zum Beispiel

- *Methode der dominante Wellenlänge*

Die Kombination von verschiedenfarbigen Lichtquellen wird ganz gezielt zur Erzeugung von **Farbpaletten** eingesetzt. Dabei bezeichnet man als **Komplementärfarben** zwei verschiedenfarbige Lichtquellen, die in Kombination weißes Licht ergeben.

6.2 Standardfarbmodelle für Rasterbildschirme

7 Erzeugung realistisch wirkender Bilder

Für dieses Anwendungsgebiet der Computergraphik sind folgende Kriterien zu erfüllen:

- Genaue Beschreibung der darzustellenden Objekte
- Entfernung nicht sichtbarer Teile
- Möglichst realistisches Beleuchtungsmodell
- Anbringung von Texturen

7.1 Entfernung verdeckter Kanten und Flächen

Die Methoden der Entfernung verdeckter Kanten und Flächen ist ein wesentliches Kriterium zur Erzeugung wirklichkeitsnaher Abbildungen, da hiermit alle sichtbaren Teile einer dreidimensionalen Szene von einem gegebenen Blickpunkt aus erfolgt.
Da heute *hohe Interaktionsraten* durch den Benutzer und meist auch *Echtzeitausgabe* gefordert sind, werden meist kleine Fehler in Kauf genommen, um die erforderlichen Bildraten gewährleisten zu können, da diese Forderungen häufig im Gegensatz zur verfügbaren Rechenkapazität der Rechner stehen.

Die eingesetzten Verfahren werden in folgende zwei Familien geräteunabhängiger Verfahren eingeteilt:

- Bei **Objektraumverfahren** liegt die Priorität auf *Rechengenauigkeit,*
- bei **Bildraumverfahren** auf *Darstellungsgenauigkeit.*

Einige Bildraumverfahren wären

- **Methode des gleitenden Horizonts:** Zur Darstellung von dreidimensionaler Flächen, definiert durch $F(x, y, z) = 0$.
- **z-Puffer-Algorithmus:** Zur Darstellung beliebiger Flächen.
- **Watkins' Algorithmus (Spanning Scan-Line-Algorithmus):** Zur Darstellung von Polygonflächen.
- **Warlock's Unterleitungsalgorithmus:** Ebenfalls zur Darstellung von Polygonflächen.

7.2 Beleuchtung

Die verwendeten Beleuchtungsmodelle dienen dazu, die Realität möglichst genau anzunähern.
Beleuchtungsmodelle werden von folgenden Größen beeinflußt:

- **Lichtquellen:** Lage, Größe, Stärke und spektrale Zusammensetzung
- **Beleuchtungsbedingungen:** Blickpunktlage und Blickrichtung
- **Oberflächenbeschaffenheit:** Geometrie, Reflexionseigenschaften wie Reflexion, Absorption und Transmission

Beleuchtungsmodelle sollen *einfach, schnell berechenbar* und mit *akzeptablen Ergebnissen* sein.

Die Hauptprobleme stellen *Multireflexion, Nichtlinearität der Ausgabegeräte* usw. dar.