Informatik

STK 2012

7. Juli 2013

Inhaltsverzeichnis

1 Vorlesung 12.10.12

1. **Was ist Informatik?**
   Definition: Informatik ist die Wissenschaft von der systematischen und automatischen Verarbeitung von Informationen, insbesondere mit Hilfe von Computern.
   Der Computer ist nur Hilfsmittel/Werkzeug
   Informatik: Information + Automaten (/Technik/Mathematik); engl. Computer Science

2. **Bereiche der Informatik:**

   - Theoretische Informatik:
     math. Modelle, Entscheidbarkeit u. Komplexität von Problemen, Sprachen
   - Praktische Informatik:
     Techniken der Programmierung zur Lösungs von Informationverarbeitung(Software); Sprachen, Algorithmen, Datenbanken-/strukturen, Betriebssysteme
   - Technische Informatik:
     Aufbau, Struktur und Funktonsweise von Hardware, Elektro- und Schalttechnik, Speicher, Prozessoren, Bussysteme
   - Angewandte Informatik:
     Entwickelung und Verwendung von Software für bestimmte Anwendungen: Wirtschaftsinfo, Ingenieurinfo, Bioinfo, Geoinfo

3. **Warum Info. für Ingenieure?**
   Grundlage und Ergebnis ihrer Arbeit sind Informationen (Entwickelung, Simulation, Test)
   Computer ist Hautarbeitsmittel!
   Informatiker und Ingenieure müssen in Kooperation arbeiten um Probleme zu lösen, Austausch zwischen Spezialisten, Grundlegendes Verständnis notwendig, Effizienter Einsatz, Spezialisten mit Fachwissen aus beiden Bereichen, spezielle Verabeitungsschritte über Erweiterungsschnittstelle ergänzen.

2 Vorlesung 19.10.12

1. **Geschichte:**
   Abakus,Al-Chwarizmi (820, Algorithmen), Riese (Rechenvorschriften), Leibniz (1673, Rechenmaschine, Dualsystem), George Boole (1854, Boolsche Logik), Turing-Maschine (1936), Von-Neumann-Architektur (1946), seit 1950 sprunghafte Entwickelung, Grundlagen seit Jahrzehnten stabil

2. **Von Neumann Architektur:**
   John von Neumann entwirft Architektur für Aufbau von speicherprogrammierbaren Rechnersystemen.
   Prinzip des Stored Program, d.h. Programm u. Daten zusammen im Speicher flexible Programmierung für verschiedene Anwendungen.

3. **Dualsystem:** Gottfried Wilhelm Leibniz beschreibt Dual/- Binärsystem mit ziffern 1 u. 0
   Heute Grundlage für elektr. Rechner - verschiedene Schaltzustände (Spannung)
   $1110 = 0 * 2^0 + 1 * 2^1 + 1 * 2^2 + 1 * 2^3 = 14$

4. **Compiler u. Interpreter:**
   Programm wird in Hoch/-Programmiersprache verfasst, muss aber zur Verarbeitung am Prozessor in Maschinensprache übersetzt werden.
   Der **Compiler** tut dies vor der Ausführung und testet auf syntaktische Korrektheit. Dadurch ergibt sich eine schnellere Ausführung und der Code ist nicht rekonstruierbar (Schutz). Oft bei Betriebssystemen usw..
   Der **Interpreter** übersetzt simultan zur Ausführungszeit. Dadurch ergibt sich eine langsamere Ausführung und es sind Syntaxfehler während der Laufzeit möglich. Der Quelltext bleibt sichtbar und das Programm ist plattformunabhängig (überall wo interpreter läuft). Einsatz im Web.
   Es gibt auch Mischformen (Java Sun)

5. **Algorithmus:**
   Ist eine eindeutige Beschreibung eines in mehreren Schritten durchzuführenden Vorgangs zu Lösung einer bestimmten Klasse von Problemen.
   Hat In- und Output. Ist wie ein Rezept.
   Der Algorithmus ist ein Ergebnis der Problemanalyse.

3 Vorlesung 26.10.12

1. C ist eine imperative Sprache (Sequenz hintereinander auszuführender Befehle).
   C ist prozedural (Programm kann in Prozeduren zerlegt werden).
   C++ ist eine objektorientierte Programmiersprache (Daten und zugehörige Funktionen können in Klassen zusammengefasst werden) und ist eine häufig genutzte Sprache zur Entwickelung von Betriebssystemen und Anwendungsprogrammen.
   C++ konzipiert als kompatible Erweiterung von C. Teils problematisch.
   C++ hat alternative objekt-orientierte Bibiliotheken

2. **Header-Dateien** enthalten Deklaration von Funktionen, Variabeln, Konstanten und Klassendefinitionen, jedoch nicht Definition (Implementierung, Funktionsweise).
   Code-Dateien(auch Quelldateien) enthalten Definition der Funktion(bzw. Methode) und ggf. main als Einstiegspunkt.

4 Vorlesung 02.11.12

1. Darstellung von **Programmabläufen**: Pseudocode, Strukturgramm (Blockdiagramm), Programmablaufpläne (PAP)

2. **Bedingte Ausführung** (Selektion, Verzweigung) mit if/ else, switch/case.
   if $\cong wenn; else \cong andernfalls$
   $switch \cong schalten/springen; break \cong rausgehen/unterbrechen$

3. **Schleifen** (Loops, Iterationen, Zyklen):
   for: Zählschleife, abweisend; Endlosschleife bei Betreibssystemen usw.
   while: kopfgesteuert, Bedingung vor Programmteil, abweisend
   do-while: fußgesteuert, Bedingung nach Programmteil, nicht abweisend
   (siehe Übung 4)

5 Vorlesung 09.11.12

1. **Steueranweisungen in Schleifen**:

   - break: Abbruch der Schleifenausführung, Programm wird nach dem Schleifentext fortgesetzt.
   - continue: Abbruch des aktuellen Schleifendurchlaufsn d.h. Code bis zum Ende wird ignoriert und wieder am Anfang der Schleife begonnen

   Automatische Verarbeitung von Informationen erfordert Darstellung in geeigneter Form (Daten) und deren Umwandelung (Kodierung)

3. Def. **Information**:
   Sind übertragene Muster von Energie und Materie, die von einem Empfänger verarbeitet und genutzt werden können. (aus der informationstheorie)
   Hat dynamischen Aspekt, Information kann zur Veränderung führen (Entstehen von Wissen), Keine Rücksicht auf konkrete Anwendbarkeit, Potenieller Gehalt in Signalen.

4. Def. **Daten**:
   Sind durch eine festgelegte Struktur (Syntax) repräsentiert und durch Computer interpretierbare Informationen mit einer bestimmten Bedeutung (Semantik).
   Syntax: legt erlaubte Ausdrücke fest (ist ein Teil der Grammatik)
   Semantik: Bedeutung der Daten, abhängig vom Anwendungskontext

5. Def. **Kodierung**:
   Ist Abbildung von Wörtern in Worte eines anderen/desselben Alphabetes zum Zweck der Übertragung/Verarbeitung.
   Alphabet: geordneter Zeichenvorrat
   Wort: Zeichenfolge
   Sprache: Teilmenge der möglichen Wörtern über einem Alphabet
   Ist die Abbildungsvorschrift (Algorithmen, Tabellen, Formeln) umkehrbar, so ist der Code entzifferbar.

6. **Binärkodierung**: (siehe Übung 3)
   Darstellung jeder natürlichen Zahl mit Ziffern 1 und 0, Abbildungsregeln für negative und Gleitkommazahlen, Abbildungstabelle für Alphanumerische Daten (ASCII-Code).
   Bit: kleinste Einheit mit Zustand 1/0, mit n bits ergeben sich 2^n Zustände
   Byte: Sequenz von 8 Bit (ergibt 256 Einzelzustände)
   Maßeinheit $2^{10}Byte = 1Kilobyte, 2^{20}Byte = 1Megabyte$

6 Vorlesung 16.11.12

1. **Zweierkomplement** ist die bessere Alternative als das Einerkompliment(2x die Zahl 0 und Spezialbehandelung bei Rechenperationen) zur Darstellung negativer Zahlen.
   Bildungsvorschrift: Negiere alle Stellen und addiere 1
   (siehe Übung 6)

2. **Kodierung von Gleitkommazahlem**:
   Da zu gehören rationale und reelle Zahlen.
   Grenzen der Darstellbarkeit: Obere und untere Grenze durch feste Anzahl von Bytes; Präzision: dargestellte Gleitkommazahl ist nur Annäherung an Realweltzahl.
   Basiert auf Halblogarithmischen Darstellung von Zahlen mit Vorzeichen, Mantisse m, Basis b und Exponet e.
   In Programmiersprachen: 0.0072 = 7.2E -3
   Basiert auf normierte Halblogarithmischer Darstellung mit b = 2 als Basis.
   Bestimmte Anzahl an Bits wird aufgeteilt für verschiedene Komponeten:
   1 Bit für Vorzeichen (0 pos., 1 neg.)
   m bits für Mantisse (Nachkommastellen der nomierten Darstellung)
   c Bits für Charakteristik: Darstellung des Exponenten als immer positive Zahl durch Addition eines konstanten Wertes k in der Mitte des möglichen Wertebereiches.
   Die Gesamtzahl n der Bits ergibt sich also durch: $n = 1 + m + c$
   (siehe Übung im 2. Semester)

3. **Kodierung Alphanumerischer Daten:**
   Kodierung unserer Schriftsprache durch eineindeutige Kodierungstabelle.
   Z.B. ASCII (American Standart Code mit 8 bit for Information Interchange für einfache Textdateien), Unicode mit 16 (für Internationalisierung chinesisch, kyrillisch)
   ASCII: 0-31 Steuerzeichen, Sonderzeichen usw. 32-64, Großbuchstaben 65-90, Buchstaben 97-122

7 Vorlesung 23.11.12

1. **Funktionen:**

   - **Definition**:
     Zusammenfassung von Befehlssequenzen mit Schnittstelle (Signatur); erlaubt Dekomposition von Programmen
   - **Syntax**: [Speicherklasse] Typ Funktionsname Parameterliste Anweisung
     Speicherklassen: static(nur innerhalb Quelldatei benutzbar), extern(auch von anderen Quelldateien nutzbar), inline (optimierung beim Compilieren, direkt eingesetzt)
   - **Rückgabetyp**: z.B. void, wenn kein return
   - **Deklaration**(Signatur, Prototyp):
     Angabe der Signatur ohne Implementierung - Bekanntmachung. Erlaubt Verwendung der Funktion im Quelltext, auch wenn noch keine

Implementierung angegeben wurde.
Anwendungen: Vorabdeklaration (erforderlich, wenn Funktionen sich gegenseitig aufrufen), Deklaration von Methoden (Klassenfunktionen) in Headerdateien

- **Parameterübergabe**:
  Pass by Value: Übergabe durch reinkopieren der Werte in die Funktion, dadurch keine Modifikation (Veränderung) der ursprünglichen Variablen
  Pass by Reference: Übergabe einer Referenz auf die Ursprungsvariable/zum Speicherort der Ursprungsvariabele, dadurch kann Ursprungsvariable verändert werden.

  C/C++: Übergabe als Zeiger
  C++: Übergabe als spezielles Referenzparameter d.h. Variablen haben in Funktion anderen Namen
  Overloading: Funktionen mit gleichen Namen aber unterschiedlichen Parametern
  Defaultwerte für Parameter bei Funktionsdefinition angeben, können somit bei aufruf weggelassen werden.

2. **Felder** (Arrays):

   - Zusammenfassung von Variablen des gleichen Typs in ein- oder mehrdimensionaler Anordung fester Größe, Zugriff über deren Position.
     Anwendung: Messwerte, Vektor, Matrix.

   - **Eindimensional**:
     Syntax: datentyp feldname [elementanzahl/Feldgröße]
     Positionen: Feldgröße n, 0 bis $n-1$, darüber Zugriff: feld[2]= 5
     Oft Feldgröße über define feldgröße n geregelt.
     Feldgröße nicht dynamisch veränderbar.
     float feld [7]; sizeof(feld) = 7 * sizeof(float)
     Initialisierung:
     feld[4] = 2,5,6,4
     feld[4] = 2,5 , letzten beiden Werte werden mit Nullen aufgefüllt.
     feld[4] = 0, alles Nullen feld[4] = , Werte zufällig
   - **Mehrdimensional**:
     Syntax: datentyp feldname [dim1][dim2] ...
     Initialisierung:
     name [2][2]=2,56,7
     name [2][2]=2,5,6,7

3. **Zeichenketten**:
   Als **char-Array**: Ein char-Wert ist ein 1 Byte groß und codiert ein ASCII-Zeichen. Länge l der Zeichenketten wird durch Feldgröße bestimmt:
   $l = Feldgröße - 1$, da das Ende gekennzeichnet wird.
   Initialisierung:
   char feld[6]='h','a','l','l','o','

char feld[] = "Hallo";
Als **string**: Ist eine Funktion von C++ mit eigener Bibliothek, hat viele Funktionalitäten wie size() und compare().
Initialisierung:
string wort("hallo");
string wort[3] = "bla","bi","bu
; (siehe Übung)

8 Vorlesung 30.11.12

1. **Zeiger**(Pointer) speichern statt Wert des Datentyps Adresse als Verweis auf den Speicherbereich, wo Daten dieses Typs stehen. Nutzung für flexiblere Manipulation von Daten. Einsatzfelder: pass by reference, Arbeiten mit Feldern, Dynamische Speicherverwaltung auf dem Heap, Datenstrukturen ... .
Deklaration: datentyp* zeigername[=initialisierung];
Initialisierung:
int*zeigername - nicht initialisiert, verweist auf zufällige Stelle im Hauptspeicher, gefährlich
int* zn = NULL - gilt als nicht definiert
int* zn = i - über Referenzierung gesetzter Zeiger verweist auf Folge von sizeof(int) Bytes und/bzw. auf die Speicherstelle der Variablen i.
Operatoren:
* : Derefernzierungsoperator liefert eigentlichen Wert der referenzierten Speicherstelle
"Kaufmanns-Und" : Referenzierungsoperator liefert Speicheradresse der Variablen
Zeigerarithmentik: z.B. float* p
$p++$ setzt Zeiger auf nächsten float-Wert vor, d.h. um 4 Byte
$p--$ zurück
p+=3 setzt Zeiger p um 4 float-Werte vor, d.h. 12 Byte
p-=4 zurück

9 Vorlesung 7.12.12

1. **Von-Neumann-Architektur** (1945) bis heute Grundlage Elektronischer Datenverarbeitung (EDV, umfasst alle computergestützten Techniken zur Erfassung u. Bearbeitung von Information)

2. **Datenverarbeitungsschritte** und **Architekturkomponenten**; tech. **Umsetzung**(genauere Beschreibung: Folie 4, ab S. 4):

   - Ein-/Ausgabe(Input, Output) u. Ein-/Ausgabe; Maus, Tastatur/Monitor, Drucker, Sound; Netzwerk(als Spezialform externer Bussysteme)
   - Speicherung (Storage) u. Speicherwerk; im engerem Sinne nur der Hauptspeicher als Random Access Memory (RAM) also nicht sequentiell wie Magnetbänder, Sekundärspeicher (Festplatte) gehört zu I/O

- Transport (Transfer) u. Bus(-system)
- Verarbeitung (Processing) u. Rechenwerk(Arithmetic Logic Unit - Alu) in der Zentralen Verarbeitungseinheit (Central Processing Unit - CPU)
- Steuerung (Control) u. Steuerwerk(Control Unit) in der Zentralen Verarbeitungseinheit

3. **Speicherhierarchie**:
   Prozessorregister mit Prozessor-Cache, Arbeitsspeicher (Hauptspeicher) mit Festplatten-Cache, Festplatte, Sicherungsmedien;
   hier: Kapazität vs. Geschwindigkeit
   Lesen: Übertragen auf schnelleres Medium
   Schreiben: Übertragen auf sicheres Medium
   Cache: schneller Pufferspeicher zur Optimierung von Lesezugriffen

4. **Bussystem**:

   - **Leitungssystem** zur Datenübertragung zwischen Hardwarekomponenten, wird über Protokoll gesteuert(wer, wann, welche Leitung)
     Lokaler, interner Bus zur Verbind von CPU, Speicherwerk usw. = Systembus
   - **Adressbus**: unidirektional, bestimmt Anzahl der zur Verfügung stehenden Leitungen/Größe des adressierbaren Speichers
     Datenbus: bidirektional, Datenübertragung
     Steuerbus: steuert nach dem Busprotokoll Zugriffsberechtigung, Richtung, Taktung, Status etc.
   - **Synchron**(fester Takt) vs. **Asynchron**(beliebiger Zeitung zu Datenübertragung), aktuell semisynchron
     Serieller(in Reihe in einer Leitung) vs. **Paralleler**(parallel in mehren Leitung zum gleichen Takt)
     Intern(im Rechner) vs. **Extern**(mit Peripherie) (in Vorlesung genauer beschrieben)
   - **Warum wichtig für Ingenieure?**
     Feldbussystem zur Verbindung von Aktoren, Sensoren und Anlagen
     Eingebettete Systeme kommunizieren über Bussysteme

10 Vorlesung 14.12.12

1. **Programme**:
   Prozessoren verstehen nur binärcodierte Maschinensprache, daher vorher Übersetzung notwendig.
   Prozessoren verarbeiten Maschinensprachbefehle als sequentielle Folge mit dem Von-Neumann-Zyklus

2. **Betriebssysteme**:

- Umfasst alle Systemprogramme, welche die Ausführung von Anwendungsprogrammensteuern und alle dafür notwendigen Betriebsmittel(Hard-/Softwar) verwalten und betreiben.
  Systemprogramme: alle Programme, die unabhängig von einer bestimmten Anwendung sind

- **Hardware Abstraktion**: Nutzer-Anwendungsprogramm-Betriebssystem-Hardware

- **Aufgaben**:
  Hochfahren des Betriebssystems durch Boot-Loader
  Prozessverwaltung:
  Starten/Beenden von Prozessen, Laden in den Hauptspeicher, Zuteilung von Resourcen an Prozesse z.B. Rechenzeit der CPU, kommuniktion,Unterstützung von Multi-Tasking/-Threading: Scheduling(berücksichtig: Priorität, Aktivität, etc. für effiziente Auslastung), Prozesszustände: RUNNING, READY, BLOCKED
  Speicherverwaltung:
  Anforderung u. Freigabe von Speicher für Prozesse, physicher Adressraum wird augeteilt u. zugeordnet, Virtuelle Speicherverwaltung: Gleichzeitig Ausführung sprengt physischen Adressraum - Prozesse kriegen virtuelle Adressräume zugeteilt - aktuell nicht benötigte Ausschnitte des Adressraumes werden auf Sekundärspeicher(Festplatte) ausgelagert
  Dateiverwaltung:
  Datei: auf nicht-flüchtigen Speichermedium gespeicherte Sequenz von inhaltlich zusammenhängenden Daten(Bild,Programm etc.), die durch festgelegte Struktur(Format, Dateityp) interpretiert werden können;
  Dateisystem: Organisation von Dateien durch Zuordnung von Speicherbereichen, Freispeicherverwaltung, Zugriffsrechte, Nachvollziehbarkeit von Änderungen und Bereitstellung von Zugriffsoperatoren; heute: hierarchische Struktur von Verzeichissen, eindeutige Identifikation der Datei durch Verzeichnispfad und Namen, Suffix gibt Dateityp an z.B. .pdf
  Geräteverwaltung:
  Geräte zum Teil nur über andere Hardware zugreifbar(Monitor über Grafikkarte), koordiniert Geräte, die direkt verwendet werden(Grafikkarte, Festplatte) durch Treiber, Treiber: Systemprogramme, die als Software-Schnittstelle auf Hardware-Schnittstelle zugreifen, sind spezifisch für Gerät u. Betriebsystem

3. **Zusammenfassung**:
   Von-Neumann-Architektur als Beschreibung der Funktionseinheiten eines Rechners.
   Entwickelung von Anwendungsprogrammen als Umsetzung von Algorithmen zur Lösung von Problemklassen.
   Betriebssystem als Abstraktionsschicht zwischen Anwendungsprogrammen und Hardware.

4. **Computergrafik**:

Teilgebiet der Informatik, das sich mit der Erzeugung von Bildern u. Animationen aus Daten beschäftig.

- **Abgrenzung**:
  Daten - *Computergrafik* - Bild
  Bild - *Bildbearbeitung* - Bild
  Bild - *Bildanalyse* - Daten

- **Teilbereiche**:
  Geometrische Modellierung:
  interaktive Methoden u. Strukturen zur Abbildung von Daten zu 2 oder mehrdimensionalen Sachverhalten für bildhafte Darstellung bzw. computergestützte Beschreibung der Form geometrischer Objekte
  Bildsynthese(Rendering):
  automatische Projektion geometrischer Modelle auf Ausgabemedium (Monitor, rasterbasiert(Pixel)/Grafikdateien) bzw. Erzeugung eines Bildes aus Rohdaten
- **Anwendungen**: Mensch-Computer-Interaktion, Bearbeitung geometrischer Realweltdaten(CAD, Simulation,...), Spezialeffekte, Computerspiele
- **Warum wichtig für Ingenieure?**
  arbeiten oft mit grafischer Anwendungssoftware, CAD, Produktmodelle

11 Vorlesung 21.12.12

1. **Rastergrafik**:

   - Bild fester Größe aus in Zeilen/Spalten organisierten Pixeln
   - Pixel(Picture Element) tragen Frabinfo für Bildpunkt
   - Farbtiefe bestimmt den Wertebereich für Farben und damit Kodierungsaufwand je Pixel in Bit (oft: 24 Bit =True Colour)
   - In der Comuptergrafik Ergebnis der Bildsynthese aus Geometriedaten
   - Nachteil: nicht beliebig skalierbar - Detailverlust - Treppeneffekt
   - Formate: JPG, GIF, TIFF PNG
   - Anwendungen: Digitale Fotografie, www, Bildschirmausgabe

2. **Vektorgrafik**:

   - Allgemeiner Begriff für Speicherung modellierter Geometriedaten (für 3D Grafiken ehr der Begriff Szenen/-Geometrieformate)
   - Bilddaten bestehen aus zusammengesetzten Primitiven(Linien, Flächen, Körper) und deren Eigenschaften (Position, Farbe, Textur, etc.)
   - Nachteil: müssen für Darstellung/Ausgabe immer neu gerendert werden (Bildsynthese)

- Vorteil: beliebig skalierbar
- Formate: viele spezialisierte Dateiformate für Anwendungen
- Anwendungen: CAD, Geoinformationssysteme, Desktop Publishing

3. Def. **Geometrische Modellierung**: bezeichent sowohl Vorgang als auch Methoden u. Strukturen zur Beschreibung der Form geometrischer Objekte als Dateien in Computeranwendungen (Anwendungen: Bildsynthese, Interaktive Bearbeitung in CAD, Simulation)
Der interaktive Vorgang basiert auf 3 **Grundprinzipien**:
Grafische Primitive
Kombination von Primitiven
Transformation

4. **Grafische Primitive** werden in 2/3-dimensionalen Raum verwendet als: Eigenständiges Primitiv, Begrenzung für Linien u. Flächen u. Körper, koordinaten für Position anderer Primitive, Punkte von Freiformkurven -/flächen
Bsp. 2-dim. Primitive: Linien, linienzüge, Polygone(Dreiecke, Vierecke,etc.), Freiformkurven
Bsp. 3-dim. Primitive: alle 2-dim. Primitive, Grundkörper(Kugel, Quader, Torus, Pyramide usw.), Freiformflächen

5. **Kombination von Primitiven** zur Bildung komplexer Geometrien.
Hierarchische Konstruktion: Punkte begrenzen Kanten, diese begrenzen Flächen, diese Begrenzen Körper
Mengenoperationnen auf duch Primitive repräsentierte Punktmenge (Fläche/Volumen): Vereinigung der Objekte, Schnittmenge, Differenzen

6. **Transformationen** (Abb. auf neue Koordinaten) enthält auch Bildsynthese(Projektion auf Bildeben) und Animation(Folge von Transformationen)
Klassische Transformationen: Identitätstransformation(keine Veränderung), Translation(Verschiebung), Skalierung(Vergrößerung, Spiegelung), Rotation und Scherung(Verformung)

12 Vorlesung 21.12.12

1. Def. **Geometrische Modellierungsmethoden** (auch geometrische Repräsentation) fassen verschiedene Beschreibungsmittel(Datenstrukturen,Operationen usw.) für geo. Objekte zusammen.

   - Kriterien für Methoden: Ausdrucksfähigkeit, Korrektheit und Effizienz
   - Beispiele für Modellierungsmethoden: Kantenmodelle, Boundary Representation(BREP), Constructive Solid Geometry(CSG), Polygonnetz, Voxelgitter und Octrees
   - Klassifizierungsmethoden der Methoden: nach Anzahl der unterstützen Dimensionen, nach verwendeten Primitiven, nach unterstützenden Operationen und nach Verwendung

2. **Boundary Repräsentation (BREP)**
   ehemals nur Flächenmodellierung, jetzt auch Körper durch abeschließende Flächen
   - Verwendung vor allem in CAD (gute Bibliothek)
   - Datenstruktur: in Listen u. Tabellen (Knotenliste, kantenliste, Flächenliste, Volumenliste)
   - Konstruktionsmethoden: nur aus Punkten sehr aufwändig - deshalb spezielle Methoden: Konstruktion von Körpern aus Flächen(Sweeping), Mengenoperationen
   - Vorteile: komplexes Modell mit vielen Möglichkeiten, intuitive Modellierung, Datenstrukturen leicht in Datenmodellen Darstellbar(Tabellen, objekt-orientiert)
   - Nachteile: hoher Speicheraufwand, aufwändige Prüfungs zur korrektheit der Geometrien, Bildsynthese erfordert Umwandelung in anderes Modell
3. **BREP: Aufbau einer Geometrie**
   Hierarxhische Grundstruktur zum Aufbau eines Körpers
   - Schale(Shell): Menge an Flächen, die Körper begrenzen
   - Flächen(Face, Facettes): begrenzt durch Konturen
   - Konturen(Loops): abgeschlossene Folge von Kanten
   - Kanten(Edges): von Punkten geführte/begrenzte Kurven/Linien
   - Punkte(Vertexes, Vertices): haben Koordinaten im 3-dim Raum
4. **Polygonnetze**
   Repräsentation eines Köper durch deine Oberfläche (Teilmenge von BREP)
   - Oberfläche als Netz(Mesh) von Polygonen; diese sind einfach, meist Dreiecke(TraingleMeshes), dies führt zu einfacher Datenstruktur und eingeschränkten Operationen, Kompromiss
   - Approximation von Oberflächen: keine gekrümmten Flächen und Freiformflächen - nur annähernde Darstellung, Qualität über Dreiecksgröße steuerbar, Kompromiss zwischen Aufwand(Datenvolumen, Berechung) und Qualität
   - Vorteile: besser geeignet für Bildsynthese(effizienter Algorithmen), daher oft Umwandelung in Triangle Meshes, vermehrt für Interaktion genutzt
   - Nachteile: Semantisch armes Modell, wenig Primitive, Verlust Bedeutung der Geometrie(diese Flächen = Körper?); hohes Datenvolumen - für Abspeicherung ungünstig

13 Vorlesung 17.01.13

1. Die **Bildsynthese** bildet ein Geometrisches Modell(Datenstruktur, zur Verknüpfung geo. Primitive, welche Gesamtszene beschreiben) auf ein Pixelbasiertes Modell(bestehend aus Rasterpunkten für Ausgabe auf Monitor, Drucker, Speicher) ab.

Grundprobleme in 2D-und 3D-Computergrafik:

- **Linien, Kurven**: Algorithmen müssen jeden beliebigen Anstieg zeichnen können (auch vertikal) und auch Freiformkurven, ebenso Kreise(Kurve muss geschlossen sein)
  Grundprinzip: Ablaufen der Kurve
  Effizienz wünschenswert
- **Flächen**: Algorithmen müssen Polygone u. beliebig begrenzte Flächen mit Zeichenfarbe oder Muster füllen können
  Effizienz sehr wichtig, da viele Pixel betroffen
- **Antialiasing**: Bei Rasterung werden mehrere Primitve auf einen Pixel abgebildet (z.B.Linie vor Hintergrund) - Entscheidung für einen der Farbwerte führt zu Treppeneffekt - Lösung: gewichteter Mischwert (z.B. je nach Überdeckung
- **Clipping**: Objekte können innerhalb, außerhalb u. überlappend in abgebildete Bereich/Projektionsbereich/Viewport liegen - spezielle Algorithmen entscheiden welche Geometrien ausgeschlossen werden (definierende punkte außerhalb Viewports, Raytracing)

Grundprobleme in 3D-Computergrafik:

- Zur **Projektion** gibt es 2 klassische Methoden:
  Perspektivische Projektion(ZentralProjektion): Projektionsstrahlen treffen sich im Projektionspunkt, verzerrt Größenverhältnisse
  Definition: Kameraposition, Blickrichtung, Up-Vector, Öffnungswinkel, far and near clipping plane
  Parallelprojektion(Orthogonalprojektion): Projektionsstrahlen senkrecht zur Bildebene, verzerrt nur Kanten nicht parallel zur Bildebene , Anwendung: technisches Zeichnen
  Definiton:Richtungsvektor, Up-vector, Höhe u. Breite
- **Verdeckung** von Objekten: Bei Projektion geht Info zur räumlichen Lage verloren (welches Objekt verdeckt welches?) - Tiefenpuffer/z-Buffer speicher Tiefeninformation - kommt bei Rasterung u. Transparenz zum Einsatz
  Backface Culling: Rückseiten von Flächen werden ignoriert

2. **Echtzeitrendering** bringt durch Projektion 3D-Grafik auf Bildebene:
   - sehr schnell, aber kein Fotorealismus - nähert sich aber heutzutage dem an
   - erlaubt interaktive Manipulation in Echtzeit
   - Anwendung: CAD, Simulation, Spiele
   - Grafikpipeline: Transformation(Standpunkt) - Beleuchtung(Beleuchtungsmodell fürs Shading) - Projektion(mit z-Buffer) - Clipping u. Culling - Rasterung u. Shading
3. **Fotorealistisches Rendering** erzeugt 3D-Grafik durch komplexe Verfahren:
   - je nach Anspruch extrem berechnungsaufwändig - deshalb nicht interaktiv

- Raytracing(Strahlenoptik) u. Radiosity(Lichtverteilung)
- kann gleiches geo. Modell als Eingabe haben wie Echtzeitrendering
- Anwendung: Spezialeffekte im Film, Computerkunst

4. Zur **Grafikprogrammierung** braucht man Unterstützung für die Geometrische Modellierung und die Bildsynthese.
Populäre Grafikbibliotheken fürs Echtzeitrendering:
OpenGL (plattform- und programmiersprachenunabhängig)
DirectX (nur MS Windows)

5. **OpenGL**(Open Graphics Libray) wird bei CAD, Simulation, Spielen usw. eingesetzt und bietet 3D-Echtzeitrendering, Primitive zur Flächenmodellierung und effiziente Bildsynthese

   - **OpenGL** ist: programmiersprachenunabhängig, eine funktionale Schittstelle, plattformunabhängig, zustandsbasiert, matrixbasiert, Hardwarenah, erweiterbar, kein GUI-System
   OpenGL definiert eigene Datentypen und Funktionen
   - **Primitive** basieren auf Eckpunkten(Vertexes)
   Syntax:

   ```
   glBegin(GL_Lines);
   ... dazwischen können Punkte und Zeichenparameter gesetzt werden ...
   glColor3f(0,0,0);
   glVertex3f(2,4,5);
   glEND();
   ```

   Primitivarten:

   ```
   GL_POINTS, Gl_LINES, GL_STRIP, GL_LINE_LOOP, GL_POLYGON, GL_QUADS,
   GL_QUAD_STRIP, GL-TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN
   ```

14 Vorlesung 12.04.13

1. **Objektorientierung** ist ein Ansatz aus der Softwareentwickelung, der die Zusammenfassung von Daten und den dazugehörigen Funktionen in Klassen von Realweltobjekten unterstützt. Ziel ist eine erhöhte Wiederverwendbarkeit(Nutzung vordefinierten Klassen aus Bibos, eigene Definition von Klassen, Spezialisierung) und bessere Verständlichkeit und Wartbarkeit.

2. Klassen fassen Objekte mit gleichen Eigenschaften und auf sie anwendbare Funktionen zusammen.
Begriffe:

   - Attribute: Eigenschaften der Objekte einer Klasse
   - Methoden: Funktionen, die auf Objekte einer Klasse angewendet werden können
   - Typ(Klassen): Gesamtheit von Attributen und Methoden
   - Instanzen: ein Objekt ist eine Instanz einer Klasse/Typs (Erzeugung eines Objektes einer Klasse = Instantiierung)

3. **Kapselung** ist ein Programmierkonzept. Es unterteilt die Methoden und Attribute einer Klasse in externe (von außen sichtbar) und interne (Implementierung dr Klasse)

4. Mit **Vererbung** bzw. Spezialisierung ist die Weitergabe von Attributen und Methoden von einer Oberklasse an eine Unterklasse gemeint. Durch das Ableiten von Unterklassen entsteht eine Vererbungshierarchie, in der jedes Objekt der Unterklasse auch als Objekt der Oberklasse behandelt werden kann (Polymorphie)

5. **Konstruktoren** sind spezielle Methoden, die beim Erzeugen von Objekten benutzt werden. Das Objekt kann durch Parameter modifiziert werden, zufällig erzeugt werden und eine Kopie sein.

6. **Destruktoren** sind spezielle Methoden, die beim Löschen eines Objektes aufgerufen werden. Es kann nur das Objekt selber aber auch die mit ihm in Verbindung stehenden Speicher löschen

15 Vorlesung 26.04.13

1. **Vererbung**(Inheritance): alle Attribute(Eigenschaften) und Methoden(darauf anzuwendende Operatoren) der Oberklasse werden übernommen.

2. **Spezialisierung**: in der Unterklasse werden zusätzliche/speziellere Attribute und Methoden hinzugefügt.

3. **Polymorphie**: überall, wo ein Objekt der Oberklasse verwendet werden kann, kann auch ein Objekt der Unterklasse verwendet werden - uneingeschränkt nur bei Public Inheritance.

4. Überschreiben von Methoden(**Overriding**): in der Unterklasse kann eine speziellere Implementierung für eine bereits in der Oberklasse implementierte Methode angegeben werden (z.B. normale Addition + zu Vekoraddition +)

(zur Implementierung siehe Übungen)

16 Vorlesung 10.05.13

1. **Terminiertheit**: Der Algorithmus, bricht nach endlichen vielen Schritten ab, bei erlaubten Parameterwert. Terminiertheit ist nicht immer erwünscht (z.B. Betriebssysteme)

2. **Determiniertheit**: Der Algorithmus liefert die gleichen Ergebnisse, bei gleichen Startbedingungen und Eingaben. (Alternative: zufallsbasiert)

3. Die **Laufzeit** ist kein ausreichendes Maß für die Effizienz, weil auch die Resourcennutzung von Speicher, Strom usw. mit einbezogen werden muss. Laufzeiteffizienz drückt optimales zeitliches Verhalten bei gegeben Resourcen aus. In der Realität müssen aber auch die tatsächliche Hardware, die Programmiersprache und andere parallele Prozesse beachtet werden, deshalb ist Laufzeitkomplexität das besssere Maß zur Abschätzung der Effizienz

4. Mit der **Laufzeitkomplexität** wird die Abhängigkeit der durchzuführenden Berechnungsschritte von der Größe der Eingabe abgeschätzt. Die Landau-Notation beschreibt dabei die Wachstumsgeschwindigkeit. Z.B.:
O(1)
O(log(n))
O(n)
O(n*log(n))
$O(n^2)$
$O(2^n)$

Es gibt drei Unterteilungen:

- Best Case-Komplexität: Wieviele Berechnungsschritte im günstigsten Fall nötig?
- Average Case-Komplexität: Wieviele im durchschnittlichen Fall
- Worst Case-Komplexität: Wieviele im schlechtesten Fall

17 Vorlesung 24.05.13

1. Sortieralgorithmen: Bubblesort, Mergesort (siehe Übung)
2. **Rucksackproblem**: Man nur begrenzte Kapazität (Gewicht) und muss möchte möglichst viel "Nutzen"mitnehmen. Die Auswahl besteht aus Gegenständen mit unterschiedlichen Nutzen und Gewicht.
   - **Greedy-Strategie**: Gegenstände mit besten Kosten/Nutzen-Verhältnis werden zuerst reingepackt bis der Rucksack voll ist. Es werden lokal optimale Entscheidungen getroffen, aber nicht die global optimale Lösung erreicht. Dafür sehr geringer Aufwand für hinreichend gute Lösung: O(n)
     Als einfache Klassen umgesetzt.
   - **Backtracking**(Zurückverfolgen): Alle möglichen Kombinationen beim Packen werden systematisch getestet. Dies garantiert die global optimale Lösung, allerdings bei sehr hohem Aufwand: $O(2^n)$
     Optimierung durch z.B. abschneiden von Suchpfaden bei Rucksack voll und wiederverwendung von ähnlichen Zweigen.
     Mit Rekursion, die eine baumartige Aufrufstruktur erzeugt, umgesetzt.
3. Genetische Algorithmen (zufallsbasiert)

18 Vorlesung 7.06.13

Datenstrukturen sind Anordnungsvorschriften zur Organisation und Speicherung von Daten, die für bestimmte Anwendungen einen effizienten Zugriff ermöglichen.
Beispiele: Kantenlisten, Menge von Gegenständen im Inventar, Verzeichnisbäume,

Warteschlangen und der Programmstack.
Datenstrukturen sind komplex (werden aus einfacheren Strukturen zusammengesetzt z.B. Zeiger, Klassen), dynamisch (können ihre Größe/Ausprägung während der Laufzeit verändern) und wiederverwendbar (Einsatz für viele verschiedene Anwendungen).

1. **Kollektion** ist ein Oberbegriff für Datenstrukturen, die eine Sammelung an gleichartigen Objektten verwalten.
   Mögliche Eigenschaften: Duplikate(es könne gleiche Objekte in der Struktur auftreten), Ordnung(Reihenfolge der Elemente ist wichtig), Positionaler Zugriff (Zugriff über Position in der Struktur), Assoziativer Zugriff (Zugriff über Eigenschaft/Wert), Iterativer Zugriff (Durchlaufen aller Elemente, für alle Strukturen möglich)

Kollektionstyp	Dynamisch	Duplikate	Ordnung	Zugriff
Array/Feld	nein	ja	ja	Position
Set/Menge	ja	nein	nein	-
Bag/Multimenge	ja	ja	nein	-
List/Liste	ja	ja	ja	Position
Map, Hash Table	ja	ja	nein	Assoziativ

   Funktionen für Kollektionen: Erzeugen einer leeren Kollektion, Suchen, Löschen u. Einfügen eines Elementes.
   Speziell Liste: Element an bestimmte Position anzeigen, einfügen und löschen, Sortieren nach einem Kriterium
   Speziell Maps/Hash: Suchen/Einfügen anhand Zugriffsschlüssels

2. **Queues**(Warteschlange) sind sind einfache Listen bei denen das FiFo-Prinzip(First In, First Out) gilt.
   Operationen: enqueue(einreihen) und dequeue(auslesen)
   Verwendung: bei Betreibssystemen u. Protokollen, bei Synchronisation (Zugriffe auf beschränkte Resource), bei asynchroner Kommunikation, Simulation von Transport u. Produktions und bei Lastverteilung

3. **Stacks**(stapelspeicher) sind Listen bei denen das LiFo-Prinzip(First In, First Out) gilt.
   Operationen: push(ablegen) und pop(entnehmen)
   Verwendung: bei Mikroprozessoren, bei Speicherverwaltung von Programmen und bei Syntaktischer Analyse von Sätzen mit aus Regeln bestehender hierarchischer Struktur

19 Vorlesung 21.06.13

Mit Hilfe von **Bäumen** und **Hash-Tabellen** wird der assoziative Zugriff mittels Schlüsselwort auf Daten optimiert. Bäume sind eine grundlegende Datenstruktur zur Abbildung einer Hierarchie. Es gilt "Teile und Herrsche die Datenmenge wird immer feiner zerlegt. Die root bildet die Gesamtheit und führt über die branches zu den leaves, der kleinsten Einheit.

Höhe des Baumes: Anzahl der Knotenebenen z.B. nur root u. 2 leaves - h=2
Verzweigungsgrad: maximale Anzahl der Kindknoten, an einer Verzweigung

1. Der **Binärbaum** hat den Verzweigungsgrad 2. Jeder Knoten speichert einen Suchschlüssel, passend zu gesuchten Datei, und ermöglicht damit drei Entscheidungen während des Suchprozesses:
   - gesuchter Wert = Suchschlüssel - gefunden!
   - gesuchter Wert ist kleiner als Suchschlüssel - gehe zum linken Kindknoten
   - gesuchter Wert ist größer als Suchschlüssel - gehe zum rechtem Kindknoten
2. Einfügen u. Suchen sind ähnliche Operationen, deren Aufwand maßgeblich von der Höhe des Baumes an jener Stelle abhängt. In einem **balancierten** Baum sind die Wege voon der root zu den leaves überall möglichst gleich. Dann beträgt die Höhe h bei einem Baum mit $n = 2^k$ Elemeten ca. $h = k = \log_2(n)$. Somit beträgt der durchschnittliche Aufwand zum Suchen und Einfügen O(log n).
3. Die **Entartung** eines Baumes entsteht durch die Einfügereihenfolge und kann dazu führen, dass dieser einfach nur eine Liste gleicht. Damit ist der Baum dann das Gegenteil zu einem balancierten Baum. Der Entartung kann mit Balancekriterien u. Reorganisation, Verweis auf Beiche zwischen 2 Kindknoten, mehr als ein Schlüssel pro Knoten und Anpassung an die Speicherstrukturen entgegen gewirkt werden
4. In **Hash-Tabellen** wird aus dem Schlüsselwort die Position der gesuchten Datei im Speicher berechnet. Somit kann bei günstigen Bedingungen ein konstanter Aufwand O(1) erreicht werden d.h. egal wie groß die Datenmenge, Daten werden immer gleich schnell gefunden.
   Probleme: Es muss eine überdimensionierter Speicher vorreserviert werden; hoher Aufwand für dynamische Speicherreservierung; es kann zu Kollisionen kommen da Hash-Funktion nicht injektiv; Überläufe sind möglich

20 Vorlesung 05.07.13

1. Datenbank (Sammelung strukturierter Daten, dauerhaft gespeichert für eine bestimmmte Anwendung) + Datenbankmangementsystem (Sammelung von Programmen, die die Zugriffe auf die Datenbank umsetzt) = **Datenbanksystem**
   Datenbankmangementsysteme(DBMS) in **SQL** (Structured Query Language) umgesetzt
2. Ein **Relationales Datenbanksystem** (RDBMS) kann man sich als eine Sammlung von Tabellen (Relationen) vorstellen, in welchen Datensätze abgespeichert sind. Jede Zeile (Tupel) in einer Tabelle ist ein Datensatz (record). Jedes Tupel besteht aus einer Reihe von Attributwerten, den Spalten der Tabelle.

3. Der **Primärschlüssel** ist im Grunde das Erkennungsmerkmal jedes Tupels Einzelne Spalte oder Komnination derer können Primärschlüssel sein. Eine extra für den Primärschlüssel erzeugt Spalte heißt Surrogatschlüssel.

4. Beim **Fremdschlüssel** verweist ein Attribut im Tupel/Zeile auf eine andere Zeile (z.B. in einer anderen Tabelle), dadurch können komplexe Beziehungen zwischen Daten hergestellt werden.
   N:1 (beliebige Menge N an Datensätzen beziehen sich auf einen Datensatz (z.B. viele Studenten auf einen Studiengang))
   N:M (beliebig viele Datensätze N beziehen sich auf beliebig viele Datensätze M (z.B. viele Studenten auf viele Vorlesungen))

5. **Operationen** auf Tabellen:
   Selektion (Auswahl von Zeilen/Tupel durch Auswahlbedingung)
   Projektion (Auswahl von Spalten/Attributen durch Angabe ihre Namen)
   Verbundoperationen (Zusammenführen von Tupeln verschiedener Tabellen/Relationen durch Fremdschlüssel)
   Mengenoperationen (z.B: Vereinigung oder Schnittmengen von Relationen)