Algorithmen

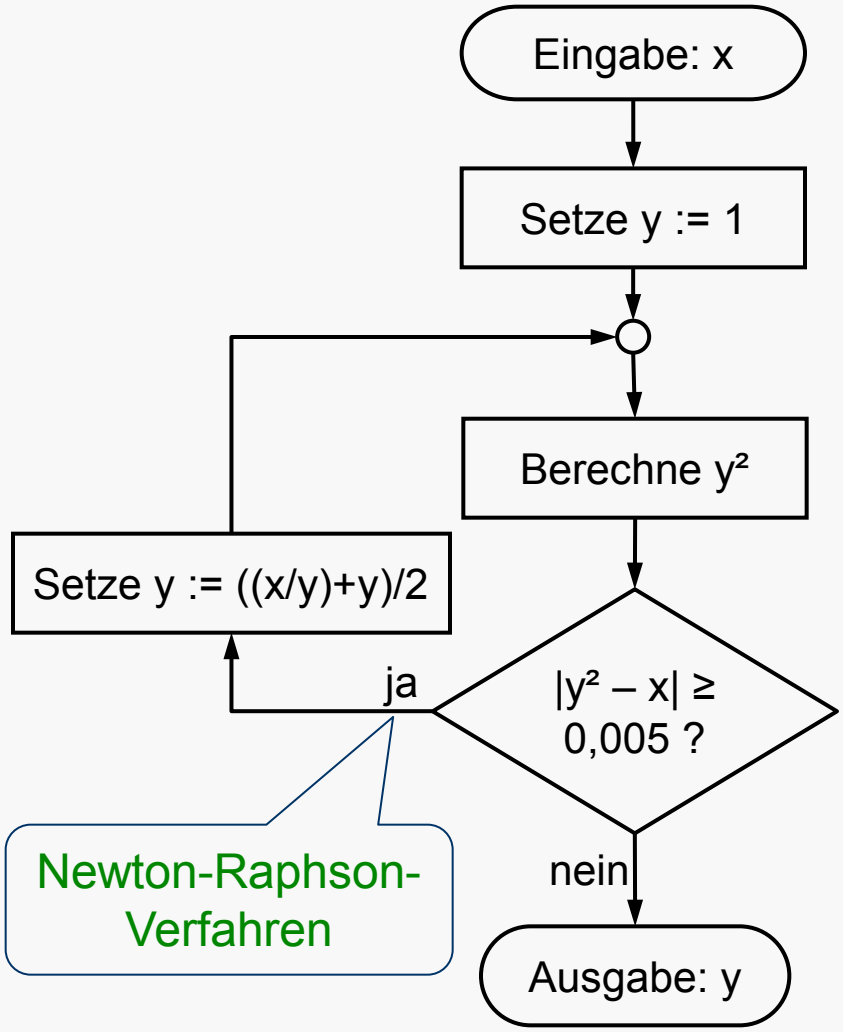
Unabhängig von Darstellung: Wirksamkeit des Lösungswegs

Beispiel:

Ermittlung von $y=|\sqrt{x}|$ $(x \geq 0)$
auf 2 Stellen hinterm Komma

… fragwürdige Methode

Eingabe: x	
Wiederhole	Wähle y
	Berechne y^2
solange	$(\lvert y^2 - x\rvert \geq 0{,}005)$
Ausgabe: y	



Algorithmen

Ein Algorithmus ist eine Folge v.Aktionen, die ein gegebenes Problem mit einer gegebenen technischen Anordnung löst.

Wichtige Kennzeichen eines Algorithmus:

- Die Problemlösung besteht aus einer endlichen Anzahl von Aktionen.
- Es gibt eine eindeutige erste Aktion.
- Zu jeder Aktion kann die Folgeaktion eindeutig ermittelt werden.
- Die Aktionsfolge endet entweder mit der Lösung des Problems, oder mit der Aussage, daß die gegebene Aufgabenstellung so nicht lösbar ist.

Algorithmen

Hinweis:

Quadratwurzel-Beispiel als Algorithmus unvollständig („semi-algorithmisch“);

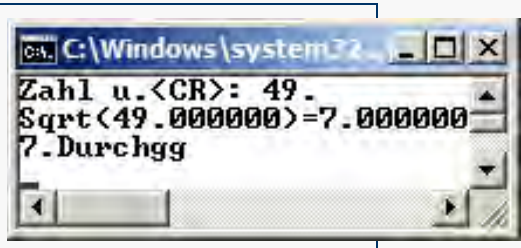
es fehlen:

- die Möglichkeit, die Unlösbarkeit anzuzeigen,
- der Bezug auf geeignete technische Anordnungen.

Hierzu wird meist Code oder Pseudocode eingesetzt.

nicht genormt

Beispiel: `SquareRoot.c`
(Programmiersprache C)




```
int main (void)
{ float x=0,y=0;
  int   i=0;
  char  c=' ';
  do
  { printf("Zahl u.<CR>: ");
    scanf ("%f", &x);
    if (x < 0.) return(0);
    for (i=1, y=1;
         fabs(y*y-x) >= .005;
         i++)
    { y = ((x/y) + y) / 2; }
    printf("Sqrt(%f)=%f\n\r",
            x, y);
    printf("%d.Durchgg\n",i);
  } while (c=_getch()!= 27);
  return(1);
}
```

SquareRoot.exe

Algorithmen

Grundelemente der Algorithmenbeschreibung

- **Variablen** sind Speicherplätze, in denen Werte eines Datentyps abgelegt werden können.
- **Literale** sind unveränderbare Werte wie Buchstaben, Zahlen oder reservierte Zeichenfolgen (z.B.: **`NULL`**).
- **Ausdrücke** stellen Werte dar: **einfache** bestehen aus einer Variablen oder einem Literal, **zusammengesetzte** verknüpfen mehrere von ihnen durch Operatoren zu einem Wert.
- Es gibt **mathematische**, (**+**, **–**, *****, **/** u.a.), **relationale** (**==**, **!=**, **>**, **>=** etc.) und **logische Operatoren** (**&&**, **||** u.a.); mit ihnen gebildete **Operationen** verknüpfen **Operanden** (d.h. Variablen und/oder Literale) zu Werten.
- Eine **Zuweisung** belegt eine Variable mit einem Wert.

Algorithmen

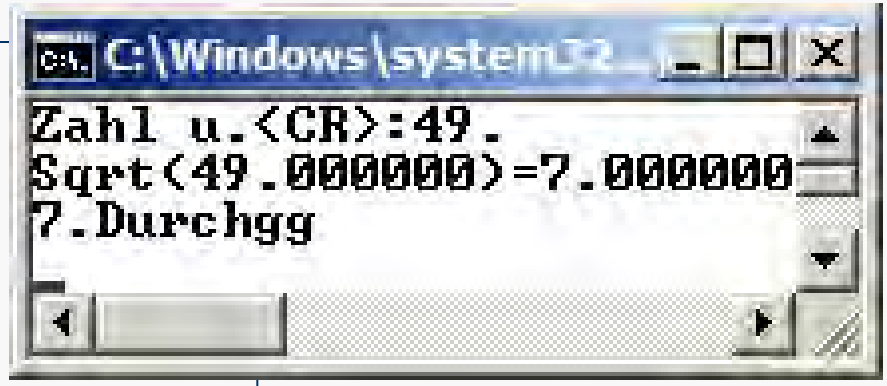
Grundelemente der Algorithmenbeschreibung (Forts.):

Kontrollstrukturen steuern den Programmablauf:

- **Sequenzen** sind Folgen von Anweisungen, die der Reihe nach ausgeführt werden.
- **Verzweigungen** sind zusammengesetzte Anweisungen, die eine Abweichung von der sequentiellen Ausführung der Programmanweisungen veranlassen.

  Dies kann geschehen aufgrund von
  - bedingten („**`if`**- bzw. „**`if-else`**-“) Anweisungen oder von
  - Selektionsanweisungen
- **Schleifen** sind identische oder parametergesteuerte Wiederholungen von Codeabschnitten; die Anzahl der Wiederholungen kann konstant sein oder von der Erfüllung von Bedingungen abhängen.

Algorithmen




```
int main (void)
{ float x=0,y=0;
  int   i=0;
  char  c=' ';
  do
  { printf("Zahl u.<CR>: ");
    scanf ("%f", &x);
    if (x < 0.) return(0);
    for (i=1, y=1;
         fabs(y*y-x) >= .005;
         i++)
    { y = ((x/y) + y) / 2; }
    printf("Sqrt(%f)=%f \n",
            x, y);
    printf("%d.Durchgg\n",i);
  } while (c=_getch()!= 27);
  return(1);
}
```

Variablen

Literal

Verzweigung

math. Ausdruck

Zuweisung

Schleife

relat. Operation

Schleife

Sequenz

Algorithmen

Jedes Programm, das immer terminiert, ist ein Algorithmus (unabhängig davon, ob/was es löst).

Jedes Programm ist mindestens semi-algorithmisch.

Geglückte Wahl und geschickte Implementierung von Algorithmen sind wichtig für die Qualität von IT-Produkten mit

- Wartbarkeit

  (vgl. Umstellung DM / €, 1999/2000: Y2K)
- Robustheit

  (vgl. Panik-Verkäufe an Börse, Konvergenz v.Newton-Raphson)
- Modularität

  (vgl. zunehmende Globalisierung: Währungen, Sprachen)
- Intuition

  (vgl. Vorbeugung v. Fehlbedienung: Medizin, Tschernobyl 1986)

Algorithmen

Verbrauch der Hauptressourcen „Rechenzeit“ und „Speicher“ insg. abhängig von:

- Umfang der geforderten Verarbeitung
  (z.B.: Gesichtserkennung vs. Identitätsermittlung auf Foto)
- Menge der Eingabedaten
  (z.B.: Kompression von Testsignalen vs. Symphonien)
- Hard- und Software
  (z.B. Co-Prozessor, optimierte Übersetzung in Maschinencode)
- Genauigkeitsanforderungen
  (z.B. Anzahl der Nachkommastellen beim Wurzelziehen, π)
- Güte der Implementierung
  (z.B. Ermittlung von Pixeln mit Gleitkomma-Operationen)
- Wahl des Algorithmus
  (z.B. Raten vs. Newton-Raphson)

Algorithmen

Gütekriterien zur Klassifizierung von Algorithmen:

- Zeitbedarf
- Speicherbedarf
- Genauigkeit
- Universalität (Allgemeingültigkeit)

Extrem-Beispiel:

Die möglichen Spielsituationen und Züge eines Schachspiels sind abzählbar und endlich: Man braucht sie „lediglich“ zu erfassen und vor jedem Zug entsprechend abzurufen.

Schätzungen ergeben, daß Computer heutiger Technologie für eine Schach-Partie Jahr-Milliarden benötigen würden, mit einem Speicher, der kaum adressierbar wäre.

Algorithmen

Gesucht: Hilfsmittel zur Klassifizierung von Algorithmen nach Laufzeitkomplexität mit Abstraktion / Nichtberücksichtigung…

- … von der Art und Struktur der Daten
  (ASCII, ganze/gebrochene Zahlen, Pixel, Variablen-Bits,…)
- … von der technischen Ausrüstung zur Problemlösung
  (Rechnermodell, Programmiersprache/Compiler, Prozessor,…)
- … von unvermeidbaren Operationen / Overhead
  (Dialoge/Menüs zur Ein-/Ausgabe, Wertzuweisung, Einrichtung von Schleifen-Anweisungen, **`if`**-Abfrage (Verzweigung),…)

in Abhängigkeit vom Parameter, der den Rechenzeitbedarf dominiert – d.h. i.d.R.: v.d.Anzahl n der Eingabedaten ($n \in \mathbb{N}$)

[Unabhängig davon gilt weiterhin die alte Faustregel „90:10“:
90% der Rechenzeit werden in 10% des Codes verbraucht.]

Algorithmen

Es erweist sich oft als kaum leistbar, den durchschnittlichen Rechenzeitbedarf eines Algorithmus zu ermitteln:

- Algorithmen können Anweisungen enthalten, für die sich Rechner unterschiedlich gut eignen

  (Pixelsetzen, **`float`**-Addition oft abhängig von Zusatzhardware)
- Der „durchschnittliche“ Einsatz eines Algorithmus ist oft zu komplex, als daß schlüssige Aussagen möglich wären.

  (Was macht ein Börsen-Computer „im Durchschnitt“?)
- Es gibt häufig nicht einmal schlüssige Datenmodelle für den durchschnittlichen Fall.

  (Was für Daten fallen bei Verarbeitung natürlicher Sprache an? Wie ungeordnet sind „meist“ Telefonnummern im Telefonbuch?)

⇨ Konzentration auf den ungünstigsten Fall („worst case“) u. auf „große“ Beanspruchung ≈ Datenmengen (vgl.: „$n \to \infty$“)

TECHNISCHE HOCHSCHULE MITTELHESSEN
Prof. Dr. A. Christidis • SS 2015

„O-Notation“:

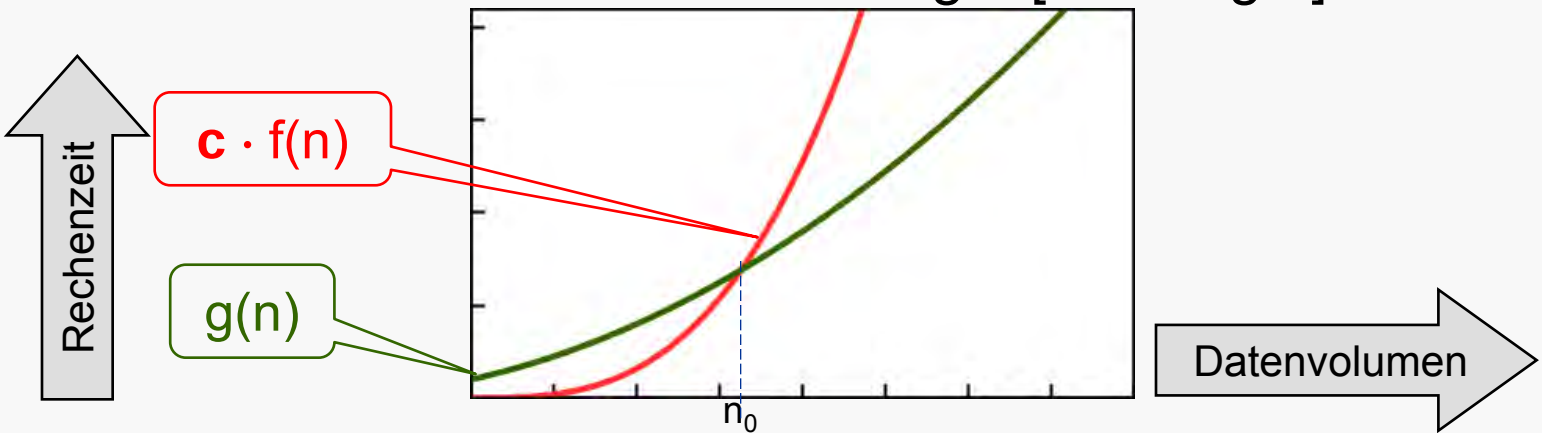
- Ermittelt man eine math. Funktion f(n) der Anzahl n der Eingabedaten u. zwei Konstanten c und n_0 so, daß für ein Datenvolumen $n \geq n_0$ für d. Laufzeit g(n) eines Algorithmus $g(n) \leq c \cdot f(n)$ gilt, so bezeichnet man g(n) als O(f(n)).

  Man sagt, die Laufzeit g(n) sei „von der Ordnung f(n)“.

  [„O“ kennzeichnet dabei „Big-Oh“: die Größen-**O**rdnung der erforderlichen Rechenzeit im ungünstigsten Fall.]

- Ist etwa f(n)=n [bzw. f(n) = log n], so sagt man auch: „Der Algorithmus hat eine Laufzeit der Ordnung n [bzw. log n]“.

z.B.:



Algorithmen

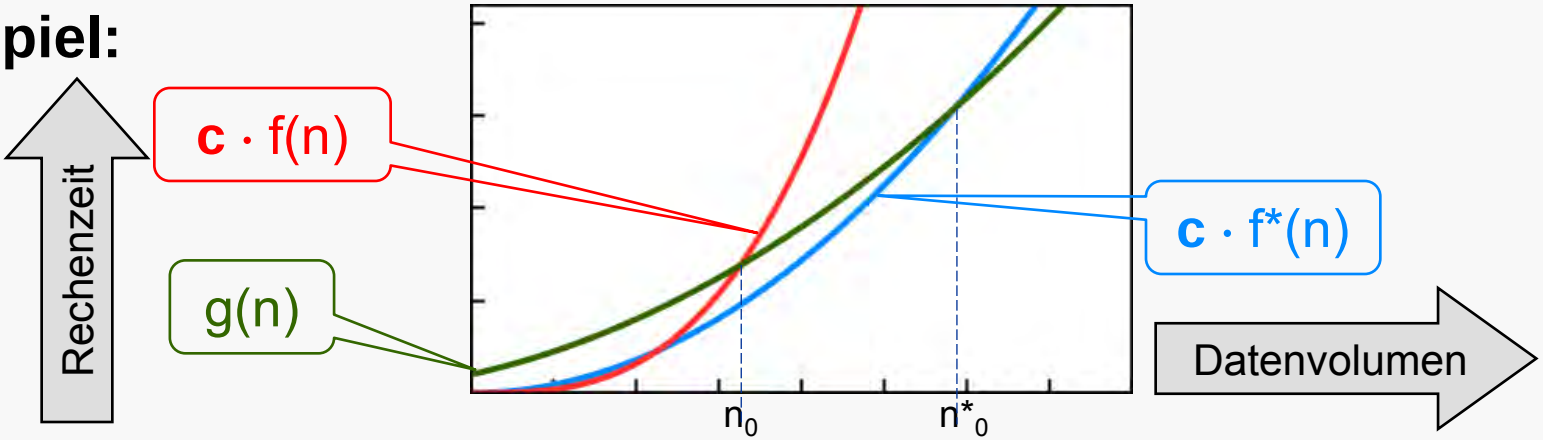
Anmerkungen zur O-Notation (auch: „O-Schreibweise“):

- O(f(n)) ist eine der (vielen möglichen) Obergrenzen des Rechenzeitbedarfs – auch im ungünstigsten Fall.

  Hat ein Algorithmus Laufzeit O(n), so hat er auch O(n²),…
- Die Ermittlung einer „optimalen“ (möglichst knappen) Obergrenze O() ist oft umständlich, ihre Kenntnis hilft aber bei der Optimierung von Algorithmen – gesucht dabei ist:

  f*(n) so, daß g(n) auch O(f*(n)) ist, aber f*(n) nicht asymptotisch zu f(n) verläuft, d.h.: $f^*(n)/f(n) \to 0$ für $n \to \infty$

Beispiel:



Anmerkungen zur O-Notation (Forts.):

- Da es sich bei O() um die Kennzeichnung einer Menge von Funktionen handelt (nämlich jener, die Obergrenzen für g(n) bilden), wird in der Literatur die Schreibweise „g(n) = O(f(n))“ vermieden:

  $O(f(n)) = \{ g: \mathbb{N} \rightarrow \mathbb{N} \mid \exists\, c>0 \wedge \exists\, n_0>0: \forall\, n>n_0 : g(n) \leq c \cdot f(n) \}^*$

- Die tatsächliche Rechenzeit g(n) kann weit unterhalb von f(n) liegen: O() ist nur eines von mehreren Wahlkriterien:
  - Der „ungünstigste Fall“ kann unrealistisch sein.
  - c und n_0 können sehr groß sein: Ein Algorithmus, der n^2 Nanosekunden benötigt, ist weiterhin vorzuziehen einem Algorithmus mit log n Jahrhunderte braucht.

* O(f(n) ist die Menge („{}“) der Abbildungen g der natürlichen Zahlen auf die natürlichen Zahlen „mit der Eigenschaft“ („|“), daß es ein c>0 und ein n_0>0 gibt, für die „gilt: “ („:“) für jedes n>n_0 „gilt: “g(n)≤c·f(n).

Algorithmen

Typische Algorithmen-Laufzeiten in der O-Notation:

O(1)	konstante Laufzeit: einmaliger Durchlauf d. Anweisungen (z.B.: `BitTest.exe`)
O(log n)	Programm wird mit wachsender Belastung allmählich, unterproportional, in fast konstanten Schritten langsamer (häufig bei Reduktion größerer Probleme auf mehrere kleinere). Logarithmus-Basis belanglos (Faktor c)! (z.B.: Suche in Binärbäumen)
O(n)	mehrmalige Anwendung desselben Verfahrens (z.B.: Schattierung/Texturierung einer geraden Linie)
O(n·logn)	überproportionale Verlangsamung bei wachsender Last (z.B. Sortieralgorithmen)
$O(n^2)$	typischer Laufzeit-Anstieg bei flächigen Anwendungen (z.B. Bildverarbeitung)
$O(n^3)$	üblich bei Anwendungen mit 3 verschachtelten Schleifen (z.B. Matrizen-Multiplikation, Ear-Clipping-Algorithmen)
$O(2^n)$	selten annehmbar als Lösung (z.B. „brute-force-“ oder „quick&dirty-“Ansätze)

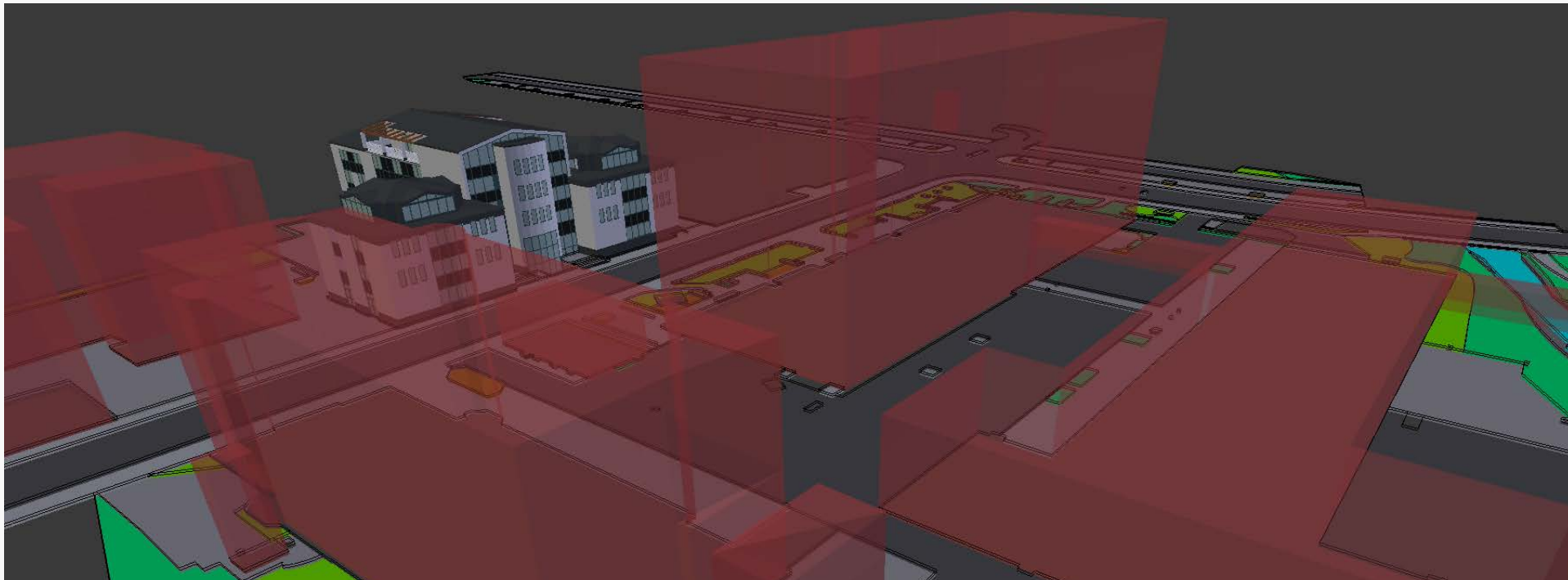
Algorithmen

Unter „Komplexität“ wird meist Zeitkomplexität verstanden.

Praxis-typisch: Abwägung („trade-off“) zw. Zeit- und Speicherbedarf

Beispiele:

- sin-/cos-Tabellen vs. sin-/cos-Reihenentwicklung
- z-Buffer vs. Verdeckungsalgorithmen



- Die O-Notation ist auch auf die anderen Gütekriterien, insb. auf den Speicherbedarf von Algorithmen anwendbar.

  Weniger relevant in der Praxis der O-Notation sind die Kriterien Genauigkeit und Universalität, weil sie entweder unabänderbar vorgegeben oder „pragmatisch“ entschieden werden.
- Parameter n der O-Notation können (neben der Anzahl der Datenelemente) alle Elemente eines Algorithmus sein, die Ressourcen beanspruchen und für das betrachtete System / Projekt kritisch werden können:
  - Grad eines Polynoms,
  - Anzahl zu sortierender Zeichen,
  - Anzahl der Knoten eines Graphen,
  - Anzahl oder Größe zu verarbeitender Dateien,
  - Personen-Zuordnung aufgenommener Stimmen,
  - u.v.m.