Visualisierung von Graphenalgorithmen

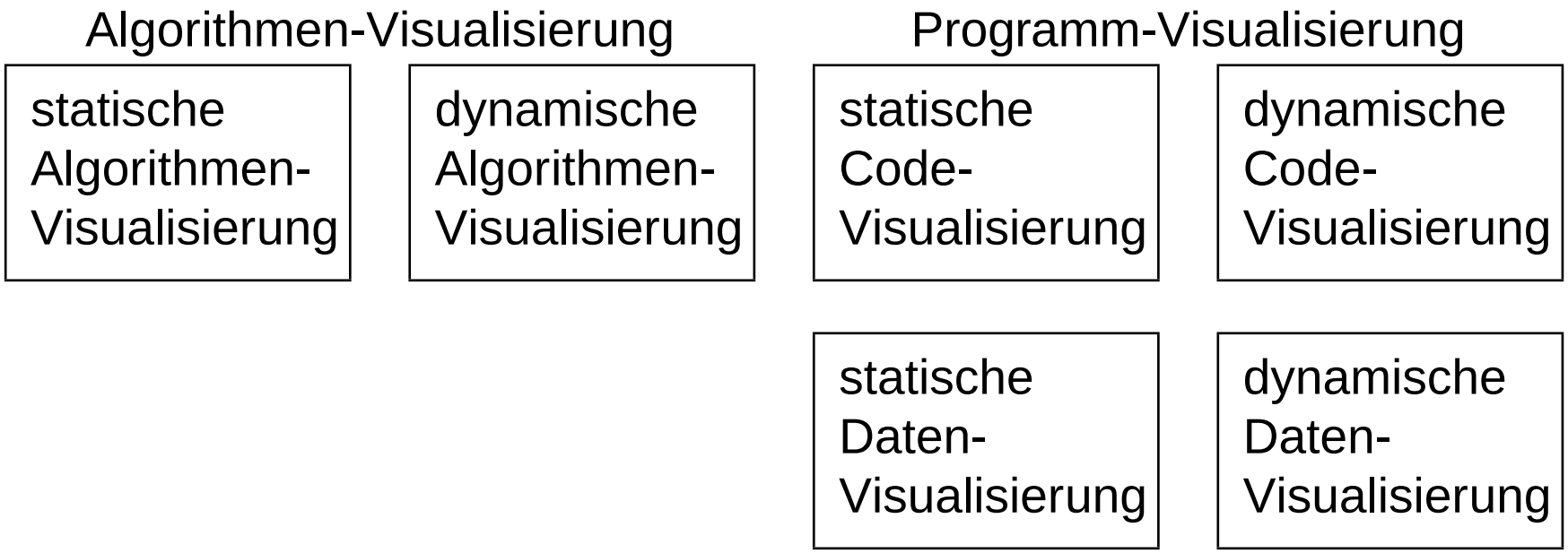
Philipp Matthias Hahn
Technische Universität Darmstadt
FB Informatik, FG Systemprogrammierung
pmhahn@informatik.tu-darmstadt.de

15. August 2002

Warum Visualisierung?

- Der Veranstalter
  - Generieren von Beispielen
  - Validieren von Implementierungen
- Der Zuhörer
  - Verstehen der Beispiele
  - Experimentieren

Software Visualisierung



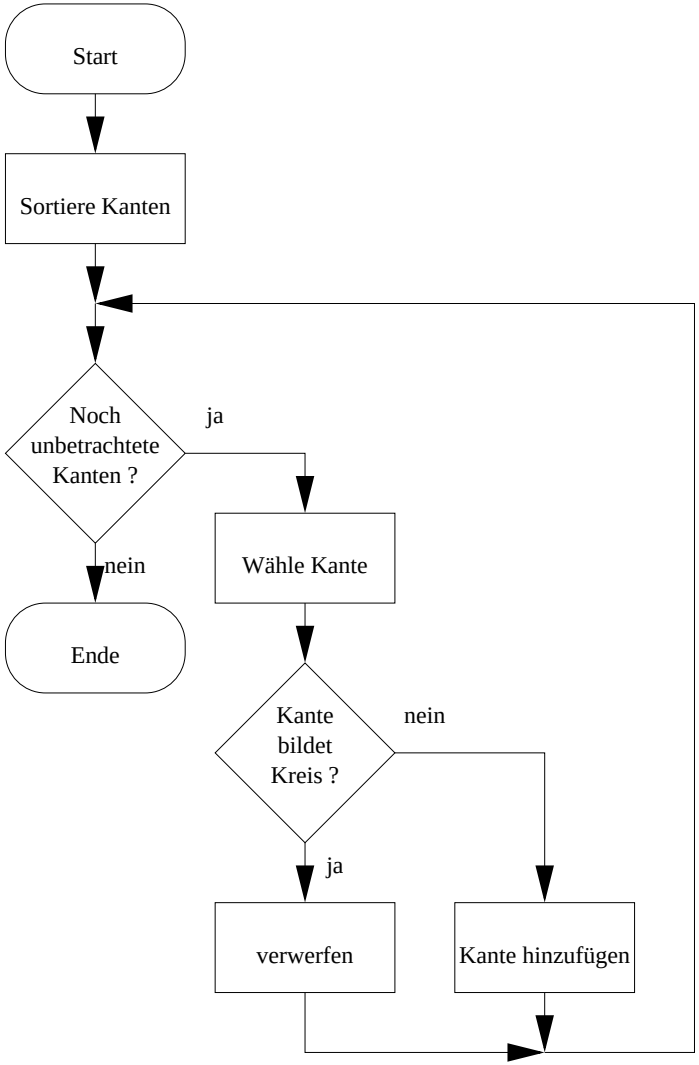
Algorithmus, Programme

Algorithmen sind exakt formulierte Rechenvorschriften, die auf Computern als ausführbare Programme implementiert und auf diese Art für die Lösung von Aufgaben angewandt werden können.

Ein Algorithmus besteht somit aus einer wohldefinierten endlichen Folge von elementaren Rechenoperationen und Entscheidungen, um aus einer bestimmten Menge von Eingabegrößen [. . .] das gewünschte Resultat [. . .] zu erzeugen.

Algorithmen-Visualisierung

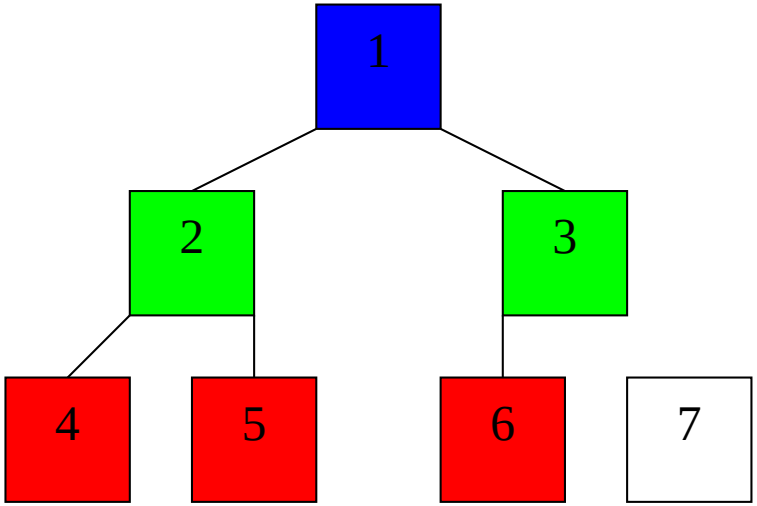
1. Bearbeite die Kanten sortiert nach ihrem Kantengewicht.
2. Wähle die nächste Kante mit dem kleinsten Kantengewicht.
3. Verwerfe sie, wenn sie zusammen mit den bereits ausgewählten Kanten einen Kreis bildet.
4. Anderenfalls füge sie der Menge der ausgewählten Kanten hinzu.
5. Wiederhole alle Schritte ab 2.



Programm-Visualisierung

```
def kruskal( E, fE ):
    E.sort( fE )
    Tree  = [ ]
    for  e in  E:
==>      if  hasCycle( Tree+[e ,] ):
             pass
         else:
             Tree.append( e )
    return Tree
```

Daten-Visualisierung



Was angezeigt werden soll

- Quelltexte
- Eingangsgraph
- „Hilfsgraphen“
- Interne Datenstrukturen
- Traces von Variablen
- Rekursionen

Zwei Arten der Visualisierung

- Ereignisgesteuert
- Datengesteuert

Ereignisgesteuert

- Das **Programm** steuert die Animation

```
for e in E:
    display.colorize( e,Red )
    if hasCycle( Tree+[e ,] ):
        display.colorize( e,Gray )
    else:
        Tree.append( e )
        display.colorize( e,Blue )
```

- Zum Beispiel *LEDA*
- „Wo ist der Algorithmus?“

Externe Darstellungen

- AspectJ
  - Erfordert *Java* als Programmiersprache
    $\Rightarrow$ Typecasts
  - Basiert auf Methodenaufrufen
- `tie`
  - Visualisierung wird erst zur Übersetzungszeit mit dem Programm verwoben
  - Wie wird die Konsistenz sichergestellt?
  - Wie kompliziert ist die Adaption für andere Implementierungen?

Datengesteuert

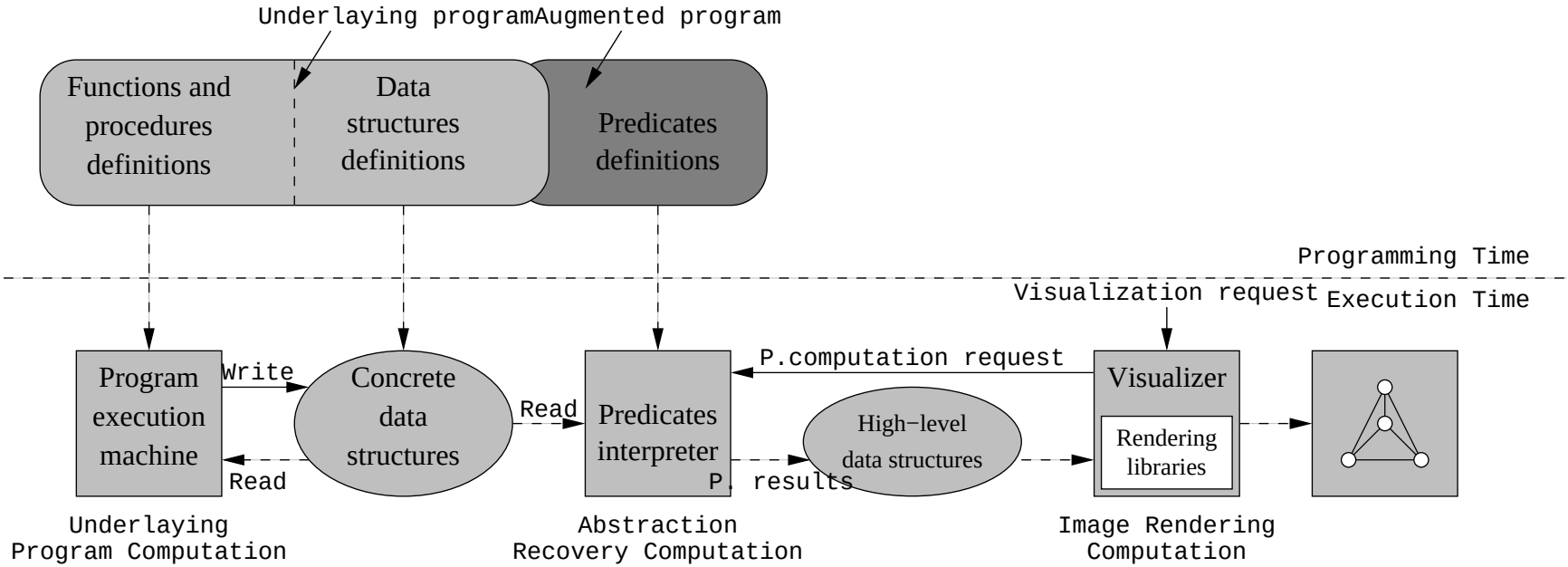
- Die **Datenstrukturen** steuern die Animation

```
E = AnimatedList( E )
Tree = AnimatedSet()
for e in E:
    if not hasCycle( Tree+[e ,] ):
        Tree.append( e )
```

- Zum Beispiel *GATO*

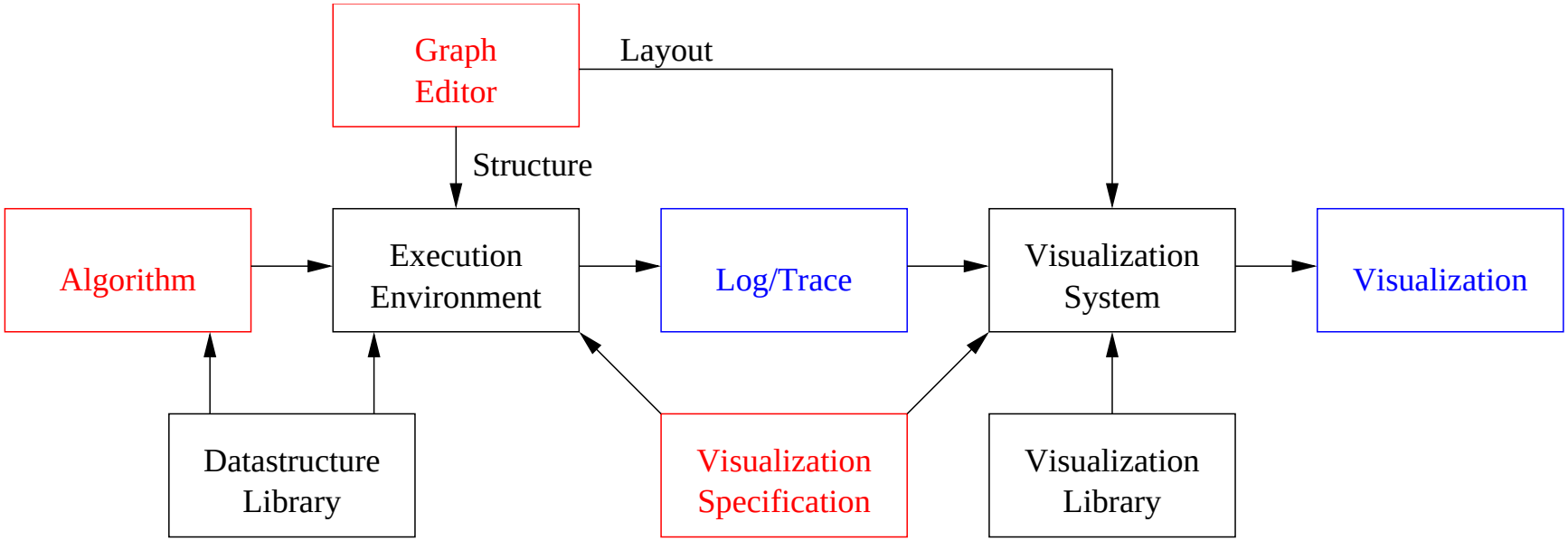
- „Wo und wie wird e nun eingefärbt?“

Leonardo



- *C*-Compiler
- Virtuelle Maschine
- “Undo”-Funktion
- Visualisierungssprache *Alpha*
- Entwicklungsumgebung
- Grapheneditor

Datenfluß



- Eingabespezifikation
- Visualisierungssystem
- Programmausgabe

Interpreter als Virtuelle Maschine

- Interpreter liefert virtuelle Maschine gratis
- In wie weit lassen sich die interessanten Informationen automatisch aus dem Ablauf extrahieren?

- Model: Die Datenstrukturen
- View: Sicht auf die Daten
- Controller: Der Algorithmus

Python

- Inerpretiert $\Rightarrow$ Erlaubt das Experimentieren
- Untypisiert $\Rightarrow$ Keine störenden Typumwandlungen
- Objektorientiert $\Rightarrow$ Zusätzlicher Freiheitsgrad
- Minimalistisch $\Rightarrow$ Leicht zu lernen
- Platformunabhängig $\Rightarrow$ Leicht austauschbar
- Sandbox, Debugger, Parser, Jython, . . .

Python Wrapper

- Erlaubt das Protokollieren aller Veränderungen
- 72 Methodenaufrufe:
  - Numerische Typen
  - Listen, Tupel, assoziative Felder
  - Klassen, Instanzen
  - Methoden, Funktionen
- Problem: Wertzuweisung

Python Tracer

- Nutzt Python Debugger-Schnittstelle
- Protokolliert wichtige Ereignisse:
  - Funktionsaufrufe inklusive aller Parameter
  - Rückgabewerte
  - Ausnahmebedingungen
  - Programmzeilenausführungen
- Problem: Programmzeilen schlecht auswertbar

Python Parser *

- Syntaktische Analyse der Programmzeilen
- Zugriff auf abstrakten Syntaxbaum zur Laufzeit
- Ermöglicht das Aufzeichnen von Strukturinformationen
  - Schleifenkopf
  - Schleifenkörper
  - Testbedingungen
- Diese können der Informationsrückgewinnung dienen

XML Mapping

- Dynamischer Programmablaufbaum
  - Wertzuweisungen, Änderungen
  - Wertabfragen, Vergleiche
  - Strukturinformationen
- Export als XML Struktur
- Navigation über XPath mit Standardprogrammen
- Transformation mit XSLT in Ausgabeformate

Was wurde (nicht) betrachtet

- Eingabe von Graphen
- Verwendung von Algorithmenbibliotheken
- Datenstruktursammlungen
- Generierung der Animationen
- Didaktische Gesichtspunkte

Was bleibt zu tun

- Implementierung des Parsers
- Vergleich der Ergebnisse mit existierenden Lösungen
- Extraktion der Informationen für Animationen
- Weitere Datenstrukturen und Algorithmen implementieren
- Die „anderen“ beobachten