Algorithmen lernen mit interaktiven Visualisierungen

Nils Faltin

C. v. Ossietzky Universität Oldenburg
Abt. Computer Graphics & Software-Ergonomie
D – 26111 Oldenburg
Faltin@Informatik.Uni-Oldenburg.de

Abstract: Es werden drei Formen des interaktiven visuellen Lernens von Algorithmen behandelt: Betrachten einer Animation, interaktives Üben der Schritte und Experimentieren mit einer Simulation. Für sie werden aus didaktischer und software- ergonomischer Sicht wünschenswerte Merkmale angegeben und Beispiele aus webbasierten Lernprogrammen gezeigt. Diese Systematik soll es dem Lehrer erleichtern, aus der Fülle der im Internet verfügbaren Visualisierungen die für seinen Unterricht passenden auszuwählen.

Einführung

Algorithmen und Datenstrukturen sind ein wichtiger Teil der Informatik. Um sie zu vermitteln werden von je her grafische Hilfsmittel verwendet. So wird z.B. in Lehrbüchern die Datenstruktur und der Ablauf eines Algorithmus in mehreren einzelnen Bildern dargestellt. Es bietet sich an, auch beim Lernen mit dem Medium Computer auf visuelle Darstellungen zu setzen. Gegenüber dem Buch kommen beim Lernmedium Computer aber noch bewegte Bilder, Töne und Interaktion als Ausdrucksmittel hinzu.
Der Computer kann vom Lehrer verwendet werden, um die Datenstruktur und den Ablauf des Algorithmus den Schülern zu zeigen. Die Schüler können aber auch selbst mit interaktiven Visualisierungen eines Algorithmus arbeiten. Durch die zunehmende Ausstattung der Schulen mit modernen multimedialen Computersystemen wird diese Form der Vermittlung an immer mehr Schulen möglich. Auch die nötige Software ist nun einfacher erhältlich. Über den Internet-Anschluss der Schule kann man auf hunderte im Internet verfügbare interaktive Lernmedien zu Algorithmen zugreifen. Dabei handelt es sich zumeist um Algorithmen-Animationen. Dies sind kurze Filmsequenzen, die den Ablauf des Algorithmus durch die Veränderung der Daten in der Datenstruktur zeigen. Zumeist stehen sie als Java-Applet zur Verfügung, dass direkt im Webbrowser ausgeführt werden kann.
Typischerweise fällt dem Lehrer die Aufgabe zu, für einen zu lehrenden Algorithmus Algorithmen-Visualisierungen zu finden, zu bewerten und auszuwählen. Eine erste Übersicht über verfügbare Algorithmen-Visualisierungen können die folgenden Quellen bieten:

1. *Ressourcen-Datenbank des Deutschen Bildungsservers*
   (http://dbs.schule.de/db/listen.html). Eine Anfrage im Januar 2001 mit (Ressourcenkategorie = Lernmittel UND Sachgebiet = Algorithmen) ergab 24 Treffer.

2. *Datenbankgestütztes Verzeichnis von Algorithmen-Animationen*. Verzeichnet Animationen in deutscher und englischer Sprache (http://www.animal.ahrgr.de/Anims/animations.php3?all=1). Eine Anfrage im Mai 2001 ergab 147 Treffer.
3. „*Bestandsaufnahme von Algorithmenanimationen*“ (http://www-cg-hci.informatik.uni-oldenburg.de/~da/peters/Kalvin/Start.htm), ein Verzeichnis mit 200 Einträgen.

Dieser Artikel soll helfen, die bei der Recherche gefundenen interaktiven Visualisierungen nach didaktischen und software-ergonomischen Kriterien zu bewerten, so dass der Lehrer je nach Zielen und Rahmenbedingungen eine Auswahl treffen kann. Im Folgenden werden drei Arten von interaktiven Algorithmen-Visualisierungen unterschieden:

1. *Algorithmen-Animationen* werden vom Schüler betrachtet und analysiert.
2. *Algorithmen-Übungen* fordern vom Schüler, zuvor gesehene Schritte eines Algorithmus selbst auszuführen.
3. *Algorithmen-Simulationen* erlauben es dem Schüler, sich durch Experimentieren an die Arbeitsweise eines Algorithmus heranzutasten.

Algorithmen-Animationen

Bei Animationen ist der Ablauf des Algorithmus fest vorgegeben. Vom Schüler wird im wesentlichen erwartet, dass er die Schritte des Algorithmus in einer Visualisierung betrachtet. Es wird nicht erwartet, dass er Schritte voraussagt oder selbst Teile des Algorithmus entwickelt. Von den drei Arten der interaktiven Visualisierung ist Animation die mit Abstand verbreitetste. Einen guten Überblick über Forschungsarbeiten zu Algorithmen- Animation gibt das Werk von Stasko et al [St89].

Wünschenswerte Merkmale

Aus didaktischer und software-ergonomischer Sicht möchte ich nun einige wünschenswerte Merkmale für Animationen angeben, die für die Lernform „Betrachten“ entwickelt wurden. Peter Gloor hat eine ähnliche Liste mit Schwerpunkt auf software-ergonomischen Kriterien entwickelt [Gl98]. Die Animation sollte möglichst viele der Kriterien erfüllen.

Funktionale Struktur. Der Algorithmus wird in Funktionen aufgeteilt. Jede nichttriviale Funktion wird in einer eigenen Animation erklärt. Bereits gelernte Funktionen können in einer Animation als Bausteine (Funktionsaufrufe) verwendet werden und werden nur noch als ein Schritt angezeigt.

Lehrtext. Mit der Animation ist ein Lehrtext verbunden, der in der Art eines Lehrbuchs mit Text und Bildern den Algorithmus erklärt. Alternativ kann dies auch durch einen vorherigen Vortrag vermittelt werden.

Anzeige des Pseudocode. Zusätzlich zu den Datenstrukturen wird der Pseudocode des Algorithmus angezeigt. Die aktuelle Befehlszeile wird hervorgehoben. So kann der

Lerner den Pseudocode und die Aktionen auf den Datenstrukturen miteinander in Beziehung setzen.

Erklärung der Schritte. Die Schritte des Algorithmus werden textuell oder per Sprachausgabe kommentiert, damit der Lerner versteht, was der Algorithmus gerade tut.

Interessante Eingabedaten. Viele Algorithmen zeigen bei bestimmten Eingabedaten ein besonderes Verhalten. Der Sortieralgorithmus Quicksort zeigt z. B. bei vorsortierten Eingabedaten ein anderes Laufzeitverhalten als bei zufällig gemischten Eingabedaten. Dem Lerner werden die verschiedenen Sätze von Eingabedaten zur Auswahl gestellt.

Kleine Datenmenge. Die Animation arbeitet auf einer kleinen Zahl von Datenobjekten. Die Anzahl ist gerade so gewählt, dass das wesentliche am Algorithmus noch zu erkennen ist. Große Datenmengen erschweren die Wahrnehmung und das Verständnis. Dabei bringen sie im Normalfall keine zusätzlichen Erkenntnisse. Kognitive Psychologen haben in Tests ermittelt, dass Menschen typischerweise 7-9 Einheiten („chunks“) im Arbeitsgedächtnis halten können. Deshalb sollten nicht mehr als 7 Objekte an einem Algorithmusschritt aktiv beteiligt sein.

Wenige Schritte. Analog werden nur so viele Wiederholungen von Schleifendurchläufen gezeigt, wie für Wahrnehmung und Verständnis nötig sind. Andernfalls wird der Lerner schnell gelangweilt und seine Aufmerksamkeitskapazität unnötig verbraucht.

Flexible Ablaufsteuerung. Die Animation des Algorithmus kann ähnlich wie bei einem Videorecorder gesteuert werden: Einzelschritte vor und zurück, Abspielen, Pause, Stopp, Sprung an Anfang und Ende. Aus didaktischer Sicht ist es besonders wichtig in Einzelschritten vor und zurück schalten zu können. Gerade Anfänger, die den Algorithmus neu lernen sind mit einem durchlaufenden Film meist überfordert. Da normale Programme nicht „rückwärts“ laufen können stellt dies besondere Anforderungen an die Software-Architektur der interaktiven Visualisierung.

Interessante Ereignisse. Ein Schritt umfasst dabei ein „interessantes Ereignis“ des Algorithmus. Das kann z. B. eine Aktion des Algorithmus oder das Erreichen wichtiger Zwischenergebnisse sein. Im Gegensatz zu diesem Prinzip machen manche SW-Visualisierungs-Systeme grafische Operationen zu Schritten. Das Konzept der „interesting events“ wurde von Marc Brown entwickelt [Br87].

Fokussierung der Aufmerksamkeit. Der Blick des Lerners wird auf die Teile der Darstellung gelenkt, die sich als nächstes verändern werden. Dazu kann der Autor diese Objekte z.B. blinken oder ihre Größe ändern lassen. So wird verhindert, dass der Betrachter wichtige Schritte nur als diffuse Bewegung wahrnimmt.

Weiche Übergänge. Veränderungen auf den Datenstrukturen werden möglichst mit weichen, fließenden Bewegungen gezeigt. Werden z.B. zwei Elemente eines Array vertauscht, so können sie sich jeweils auf einem Halbkreisbogen bewegen und so „Plätze tauschen“. Dies erleichtert die Wahrnehmung des Vorgangs. Damit beim Lerner nicht der Eindruck entsteht, in einem Computer würden sich die Daten „fließend bewegen“ sollte im Begleittext darauf hingewiesen werden, wie die Veränderung der Daten implementiert wird. Vertauschung wird z. B. über „plötzliche“ Kopiervorgänge im Dreieckstausch realisiert. Es war eines der wesentlichen Ziele bei

der Entwicklung des SW-Visualisierungssystems Tango, Animationen mit weichen Übergängen zu ermöglichen [St98b].

Beispiel

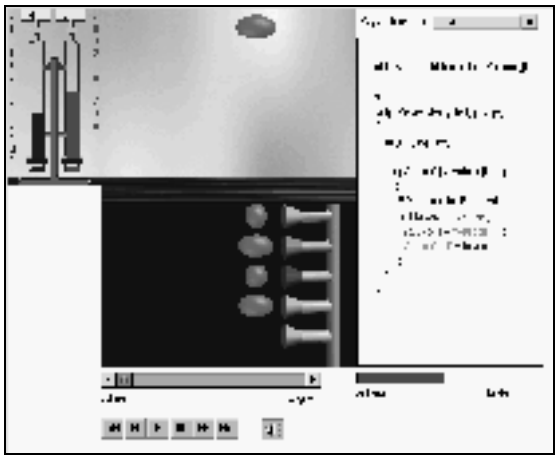

Abb. 1 Lernen durch Betrachten: Bubblesort-Animation

Mir ist keine Algorithmen-Animation bekannt, die alle genannten Prinzipien umsetzt. Die Autoren der Animation des Bubblesort-Algorithmus (Abb. 1) wollten eine ästhetische Animation entwickeln, die die Metapher „Luftblasen“ (Bubbles) aufgreift [Ba98]. Sie haben dabei die folgenden Prinzipien umgesetzt:

- Weiche Übergänge
- Flexible Ablaufsteuerung
- Anzeige des Pseudocodes
- Kleine Datenmenge
- Lehrtext

Algorithmen-Übungen

Nachdem ein Algorithmus durch Betrachten einer Animation gelernt worden ist, sollte das Wissen durch Übung überprüft und vertieft werden. Bei einer *Übung* soll der Schüler mehrfach den nächsten Schritt des Algorithmus voraussagen. Es gibt dabei nur einen „richtigen“ nächsten Schritt, und zwar den, den der Standardalgorithmus als nächstes ausführen würde. Dazu interagiert der Schüler mit einer Visualisierung der Datenstruktur über eine grafische Benutzungsoberfläche. Es scheint nur wenige Projekte zu geben die solche interaktive Visualisierungen entwickelt haben ([SSN99], [Bl99], [Fa00]).

Wünschenswerte Merkmale

Die wünschenswerten Merkmale für Animationen, die ich für das Lernen durch „Betrachten“ gegeben habe, gelten bis auf den Punkt „Anzeige des Pseudocode“ auch für das Lernen mit Algorithmen-Übungen. Zusätzlich sollten noch die folgenden Merkmale vorhanden sein.

Kein Pseudocode. Da der Schüler den nächsten Schritt eigenständig voraussagen soll, wird kein Pseudocode angezeigt. Insbesonders der Charakter der Überprüfung des Wissens würde sonst unterlaufen.

GUI nutzen. Die Möglichkeiten einer grafischen Benutzungsoberfläche (GUI) werden soweit möglich und sinnvoll für die Interaktion genutzt. Insbesondere kann der Schüler Datenobjekte in der Grafik der Visualisierung direkt anwählen (z.B. mit der Maus anklicken). Ein Schritt eines Algorithmus besteht oft im Aufruf einer Funktion. Die Funktionen können z.B. durch Schaltflächen (Buttons) dargestellt werden. Der Schüler ruft eine Funktion auf, indem er erst die Objekte (aktuelle Parameter) in der Grafik anklickt und dann die Schaltfläche der Funktion.

Bedienung vereinfachen. Die Bedienung der Aufgabe ist möglichst einfach gestaltet. Es ist einfach erkennbar, welche Funktionen angeboten sind und welche Objekte anwählbar sind. Der Schüler muss also Objekte und Funktionen nur auswählen. Bedienungsfehler sollten als solche gemeldet werden. Die Bedienung sollte beschrieben sein (z.B. als Online-Hilfe oder im Begleittext).

Lösung erschweren. Es werden z. B. mehr Funktionen angeboten, als tatsächlich benötigt werden. Auch sind möglichst alle Objekte der Visualisierung anwählbar. Dadurch wird es erschwert, den nächsten Schritt zu erraten.

Fehlermeldungen und mehrstufige Hilfe. Wählt der Schüler eine falsche Funktion oder falsche Parameter, so kann es sinnvoll sein zu begründen, warum dieser Funktionsaufruf jetzt keinen Sinn macht. Meist wird es aber reichen zu melden, dass der Algorithmus einen anderen Schritt ausführen würde. Es sollte aber eine mehrstufige Hilfe abrufbar sein, die einen Hinweis gibt, was zu tun ist und als nächste Stufe den Schritt direkt angibt.

Beispiel

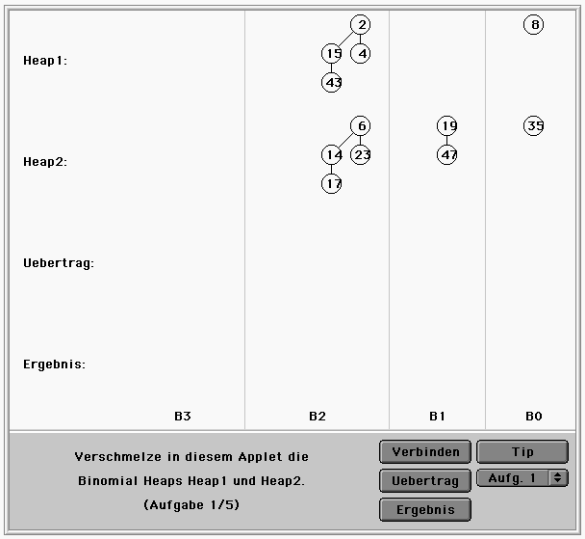

Abb. 2 Lernen durch Üben: Verschmelzen-Funktion

Ein Beispiel für *Lernen durch Üben* ist das Applet aus der Lernsoftware zum Binomial Heap Algorithmus (Abb. 2) [Bl99]. In ihm sollen zwei Listen von Binomialbäumen zu einer Ergebnisliste verschmolzen werden. Dies wird ähnlich dem schriftlichen Addieren durchgeführt. Der Schüler muss genau die Schritte des Standardalgorithmus einhalten, kann aber über die Schaltfläche „Tip“ eine einstufige Hilfe abrufen. Die oben genannten Kriterien sind im Wesentlichen erfüllt.

Algorithmen-Simulationen

Algorithmen-Simulationen erlauben ein freies Experimentieren mit Funktionen und Datenobjekten. Dies soll es dem Schüler ermöglichen den Algorithmus „nachzuentdecken“. Der Algorithmus wird dem Schüler nicht als ein fertiges Werk vermittelt, sondern dieser soll einen Teil der Arbeit der Entwicklung und Entdeckung wiederholen. Er wird dadurch in die Rolle eines Algorithmenentwicklers versetzt.
Die Lernform „Experiment“ muss, wie andere Lernformen auch, in einen größeren didaktischen Kontext eingebettet werden. Für die Gestaltung von Lernsoftware zu Algorithmen habe ich hierzu das didaktisches Konzept „Strukturiertes aktives Lernen von Algorithmen (SALA)“ entwickelt [Fa99].

Nach meinem Verständnis besteht ein Algorithmus aus den folgenden Strukturen:

1. Statische und dynamische *Struktur der Daten* (z.B. Listen, Verzeigerung, Records/ Structs)

2. Mathematische und logische Bedingungen zwischen Datenobjekten, den sog. *Invarianten*
3. Funktionale Struktur
4. *Pseudocode* der einzelnen Funktionen

Die *funktionale Struktur* eines Algorithmus gibt an, in welche Funktionen der Algorithmus aufgeteilt ist und welche Funktionen einander aufrufen. Für jede Funktion muss angegeben sein:

- *Zweck* der Funktion
- *formale Parameter* (Typ und Rolle)
- *Start-Invarianten*, die bei Aufruf der Funktion gelten müssen, damit die Funktion arbeiten kann.
- *Arbeits-Invarianten*, die während der Arbeit der Funktion beachtet werden müssen. Teils drückt sich darin die „Strategie“ aus, nach der die Funktion arbeitet.
- *Ziel-Invarianten*, die beim Ende der Funktion gelten müssen. Diese leiten sich aus dem Zweck der Funktion ab und aus den Voraussetzungen (zeitlich) nachfolgender Funktionen.

Man kann nicht erwarten, dass ein Schüler den Algorithmus selbst entwickelt (nachentdeckt), wenn er nicht zuvor wesentliche Teile vermittelt bekommt. Ich schlage vor, die Punkte 1 bis 3 (Struktur der Daten, Invarianten und funktionale Struktur) zu vermitteln und den Schüler den Pseudocode selbst entwickeln zu lassen.
Der Pseudocode bestimmt, welche Schritte der Algorithmus auf einem konkreten Satz Daten ausführt. Umgekehrt kann sich der Schüler durch Ausprobieren von Schritten auf Daten zum allgemein funktionierenden Pseudocode vortasten. Dabei ist zu vermuten, dass ihm ein zielloses Probieren wenig weiterhilft, sondern dass er sich eine Strategie überlegen muss, die er durch Experimentieren erproben und verfeinern kann.
Verglichen mit dem Lernen durch „Üben“ arbeitet der Schüler beim „Experimentieren“ mit einer echten Simulation, bei der aus einer Situation heraus in der Regel alternative Schritte möglich sind. Es kann durchaus mehrere Wege zum Ziel geben. Der Fortschritt der Bearbeitung sollte sich in der Visualisierung zeigen, so dass der Schüler sieht wo „die Arbeit erledigt ist“ und wie nah er dem Ziel gekommen ist.
Die Simulation ermöglicht dem Schüler zu experimentieren. Dazu stellt sie mehrere Funktionen zur Verfügung, die auf den Datenobjekten angewendet werden können. Sie führt aber einen Funktionsaufruf nur dann aus, wenn die Arbeits- und Start-Invarianten dies erlauben.

Wünschenswerte Merkmale

Die Benutzungsoberfläche der Algorithmen-Simulation ähnelt der Oberfläche der Algorithmen-Übung. Zusätzlich zu den dort genannten Kriterien sollten die folgenden Merkmale erfüllt sein:

Inhaltsbezogene Fehlermeldungen. Versucht der Schüler eine Funktion aufzurufen, deren Start-Invarianten nicht erfüllt sind oder würde dadurch eine Arbeits-Invariante

verletzt, so muss die interaktive Visualisierung dies mit einer Fehlermeldung abweisen. Die Fehlermeldung sollte den Grund in Bezug auf die Invarianten angeben.

Fortschrittsanzeiger. Der Schüler kann erkennen, wie nah er dem Ziel gekommen ist. Die Visualisierung kann dazu z.B. anzeigen, welcher Anteil der Daten die Ziel-Invariante bereits erfüllt.

Undo. Es kann durchaus sein, dass der Schüler Schritte tätigt, die zwar die Invarianten nicht verletzen, aber nicht zum Ziel führen. Über einen Undo-Mechanismus kann der Schüler einzelne Schritte zurücknehmen.

Beispiel

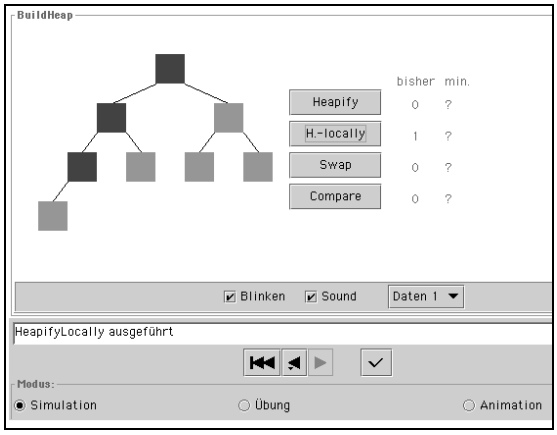

Abb. 3 Lernen durch Experimentieren: "Build-Heap"-Funktion

Abbildung 3 zeigt einen Screenshot einer Simulation der „Build-Heap“-Funktion aus dem Heapsort-Algorithmus [Fa00]. Der Schüler soll einen Knoten im Baum wählen und dann dort eine der Funktionen aus der Buttonleiste starten. Für die richtige Lösung müssen die ungeordneten Knoten bottom-up mit Heapify geordnet werden. Diese Simulation bietet Fehlermeldungen, Fortschrittsanzeiger und Undo. Das Applet kann auch in die Modi „Üben“ und „Animation“ umgeschaltet werden.

Zusammenfassung und Ausblick

Es wurden drei Grundtypen von Algorithmen-Visualisierungen vorgestellt: Animation, Übung und Simulation. Während es viele Veröffentlichungen zu technischen Fragen rund um die Animation von Algorithmen gibt, werden didaktische Fragestellungen nur selten behandelt. Hier ist sicher noch Forschungsbedarf gegeben.
Neue Lernformen wie „Üben“ und „Experimentieren mit Simulationen“ müssen sich in der Praxis bewähren. Dazu braucht es zunächst noch mehr Lernmaterial (z.B. Lernsoftware), die nach diesen Methoden gestaltet ist. Anschließend ist der Lernerfolg dieser Vermittlungsmethoden zu evaluieren.

Zur weiteren Lektüre kann ich die Dissertation von Hundhausen empfehlen [Hu99]. Sie gibt einen Überblick über Evaluationen zur Effektivität von Algorithmen-Animationen (S. 27-31). Allein der Faktor „Beteiligung der Lerner“ (learner involvement) zeigte durchgängig einen signifikanten Effekt auf den Lernerfolg. Damit ist gemeint: Eingabedaten selbst erstellen, Vorhersage des nächsten Schritts, Algorithmus programmieren und Animation programmieren (S. 18). Daraufhin entwickelte Hundhausen eine neue Lehrmethode. Er ließ Studenten Prototypen von Algorithmen-Animationen (Storyboard) aus einfachen Materialien (z.B. Pappe) erstellen und damit den Algorithmus vortragen. Er beobachtete eine hohe Motivation und fachlich tiefgehende Gespräche über die Algorithmen (Kap. 4).
Abschließend möchte ich die Leser ermutigen, sich anhand der genannten Lernsoftware ([Fa00],[Ba98], [Bl99]) ein eigenes Bild davon zu machen, was die neuen Lernformen leisten können. Die Lernsoftware kann mit normalen Webbrowsern aus dem Internet geladen und ausgeführt werden.

Quellen

[Ba98] A. Barbu, M. Dromowicz, X. Gao, M. Köster und C. Wolf: *Lernsoftware „Sortieren mit Bubblesort und Quicksort“*. 1998. Abrufbar über [Fa01].

[Bl99] Karsten Block: *"Lernsoftware zum Binomial Heap Algorithmus"*. 1999.Abrufbar über [Fa01].

[Br87] Marc Brown: *"Algorithm Animation"*. 1987. MIT Press, Cambridge.

[Fa00] Nils Faltin: „Heapsort-SALA - Ein Lernprogramm zum Heapsort-Algorithmus“. 2000. Abrufbar über [Fa01].

[Fa01] Nils Faltin: „Oldenburger Sammlung von Lernprogrammen zur Informatik“. 2001. http://www-cg-hci.informatik.uni-oldenburg.de/~da/Lernprogramme_Informatik/katalog.html.

[Fa99] Nils Faltin: *"Gestaltung von Lernprogrammen zu Algorithmen für aktives Lernen mit virtuellen Brettspielen"*. 1999. Tagungsband der Tagung "Informatiktage 1999" der Gesellschaft für Informatik, 12. - 13. November 1999 in Bad Schussenried. Konradin Verlag. Preprint-Version: http://www-cg-hci.informatik.uni-oldenburg.de/~musik/Gest_LP

[GF00] Peter Gorny und Nils Faltin: *"Das Projekt Medienunterstütztes Studium der Informatik (MuSIK)"*. 2000. http://www-cg-hci.informatik.uni-oldenburg.de/~musik/.

[Gl98] Peter Gloor: „User Interface Issues for Algorithm Animation“ in: , S. 145-152.

[Hu99] Christopher Hundhausen: "Toward Effective Algorithm Visualization Artifacts. Designing for Participation and Communication in an Undergraduate Algorithms Course". Dissertation. June 1999. CIS-TR-99-07. Dept. of Comp. and. Inf. Science, University of Oregon, Eugene, USA. http://lilt.ics.hawaii.edu/~hundhaus/dis/.

[SSN99] Linda Stern, Harald Sondergaard and Lee Naish: *"A Strategy for Managing Content Complexity in Algorithm Animation"*. PP 127-130 in: Bill Manaris (Ed.), Proceedings of the 4th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education - ITiCSE '99. June 1999. ACM Press, New York.

[St89] John T. Stasko, John B. Domingue, Marc H. Brown und Blaine A. Price (Hrsg.): *"Software Visualization"*. 1998. MIT Press, Cambridge, Massachusets.

[St89b] John Stasko: „Smooth Continous Animation for Portraying Algorithms and Processes“. S. 103-118 in [St89].