Carsten Görg

Algorithmenanimation

Samba und empirische Untersuchung

- Byrne, Catrambone, Stasko:
  "Do algorithm animations aid learning?"

- Stasko: "Using student-built algorithm animations as learning aids"

- School of Psychology, College of Computing, Georgia Institute of Technology, Atlanta

- http://www.cc.gatech.edu/gvu/softviz/algoanim/

Helfen Algorithmenanimationen beim Lernen ?

- Bilder und Visualisierungen sind als pädagogische Hilfe akzeptiert

- Animationen gehen einen Schritt weiter: statische Darstellungen können nur zeigen wie etwas aufgebaut ist, Animationen können einen dynamischen Prozeß darstellen

- Algorithmenanimation ist eine dynamische Visualisierung der Daten und Operationen eines Algorithmus

- Algorithmenanimation ist Teilgebiet der Softwarevisualisierung: mit Hilfe von Bildern, Graphiken und Animationen wird gezeigt, wie Algorithmen, Programme und Prozesse arbeiten

- Algorithmenanimation wurde auf der SIGGRAPH'81 durch das Video "Sorting Out Sorting" eingeführt

- elementare Ziele der Algorithmenanimation:

  1. konkrete Darstellung der Abstraktionen und Operationen eines Algorithmus

  2. Darstellung der Dynamik des sich zeitlich entwickelnden Prozesses

Wie können Animationen beim Lernen helfen?

- Information, die durch eine Algorithmenanimation übermittelt wird, kann in zwei Bereiche aufgeteilt werden:

  1. prozedural: die einzelnen Schritte, die der Algorithmus ausführt

  2. konzeptuell: die Bedeutung und Auswirkungen des Algorithmus (z. B. Effizienz, Laufzeit, ...)

- Animation bietet eine Übersicht des Algorithmus, die auf dem Codelevel nicht zu erkennen ist

- Animationen motivieren, sich mit dem Algorithmus zu beschäftigen

- Hypothesenbildung wird gefördert (aktives Lernen)

- Hypothesenbildung = stoppe Algorithmus und sage nächsten Schritt / Operation voraus

- schnelles Feedback zur Hypothesenbildung

- Animation ist einprägsamer als eine Folge von statischen Bildern (man kann sich den Algorithmus besser merken)

- Animation kann ermutigen, zu versuchen sich das Verhalten des Algorithmus selbst zu erklären

Experiment 1

- Algorithmus: Depth first search
- 88 Teilnehmer (keine Informatikstudenten)
- Lernmaterial
  - sechsminütiges Video einer aufgenommenen Vorlesung
  - dreiseitige Textbeschreibung von DFS (aus Cormen, Leiserson, Rivest)
- alternative Lernphasen
- Abschlußtest (keine Zeitbeschränkung)

Lernphase

Einteilung der Teilnehmer in 4 Gruppen:

1. "Animation / Hypothesenbildung": 5 Minuten zum Lesen des Textes, danach schrittweise Animation mit Hypothesenbildung

2. "Animation / keine Hypothesenbildung": 5 Minuten zum Lesen des Textes, danach Animation ohne explizite Aufforderung zur Hypothesenbildung

3. "keine Animation / Hypothesenbildung": 5 Minuten zum Lesen des Textes, danach Hypothesenbildung an 2 gedruckten Graphen auf dem Papier

4. "keine Animation / keine Hypothesenbildung": 10 Minuten zum Lesen des Textes

Abschlußtest

1. leichte Fragen: ein einzelner Schritt des Algorithmus oder eine komplette Suche auf einem Graphen, der dem Beispielgraphen ähnlich ist

2. schwere Fragen: komplette Suche auf neuem Graphen

Ergebnisse

1. während der Lernphase

   - Auswirkung der Animation auf Hypothesenbildung: von 42 Hypothesenbildungen wurden mit Animation im Durchschnitt 0.46 falsch abgegeben, ohne Animation 1.19

2. beim Abschlußtest

   - die Grundlagen von DFS wurden von allen Testteilnehmern verstanden: 95% der leichten Fragen wurden richtig beantwortet

   - schwierige Fragen: von 7 Problemen wurden im Durchschnitt 5.18 richtig gelöst (74%). Ein Problem gilt als richtig gelöst, wenn es ohne Fehler gelöst wird.

3. Beobachtung der Teilnehmer

   - Vermutung bestätigt: Animation ermutigt zur Hypothesenbildung (das hat sich dadurch gezeigt, daß die Teilnehmer in der Gruppe "Animation / keine Hypothesenbildung" für sich doch Hypothesenbildungen machten, obwohl sie nicht explizit dazu aufgefordert wurden)

Statistik

	Animation	keine Animation	
Hyp.bildung	5.73	5.24	5.49
keine Hyp.bildung	5.27	4.50	4.89
	5.50	4.86	

- es ist nicht zu erkennen, daß die Animation gegenüber der Hypothesenbildung an statischen Graphen Vorteile hat

- die Gruppe keine "Animation / keine Hypothesenbildung" hat gegenüber den anderen 3 Gruppen schlechter abgeschnitten (im Durchschnitt eine Aufgabe von 7 weniger gelöst)

Experiment 2

- komplexerer Algorithmus: Binomial heap
- 62 Teilnehmer (Informatikstudenten im Hauptstudium)
- elfminütige Videovorlesung
- zwölfseitige Textbeschreibung der Binomial Heaps (aus Cormen, Leiserson Rivest)
- Abschlußtest (25 Minuten)
  1. prozedurale Fragen: Ausführen von Operationen auf Binomial Heaps
  2. konzeptuelle Fragen: abstrakte Eigenschaften der Binomial Heaps

Ergebnisse

1. Lernphase

   - positive Auswirkung der Animation auf Hypothesenbildung hat sich **nicht** bestätigt: mit Animation wurden bei der Hypothesenbildung im Schnitt 2.77 Fehler gemacht (von 12) und ohne Animation 2.54

2. Abschlußtest

   - kein Unterschied in den Gruppen bei den konzeptuellen Fragen

	Animation	keine Animation	
Hyp.bildung	9.93	10.94	10.45
keine Hyp.bildung	10.80	9.94	10.35
	10.37	10.44	

- bei den prozeduralen Fragen ist das Ergebnis ähnlich wie bei Experiment 1

	Animation	keine Animation	
Hyp.bildung	5.13	4.94	5.03
keine Hyp.bildung	5.00	4.00	4.48
	5.07	4.47	

- insgesamt wurden 63% aller Fragen richtig beantwortet

- es gibt zwei Gruppen von Teilnehmern: eine Gruppe hat die Fragen sehr gut gelöst und die andere hat nur wenig verstanden

- die Spaltung in diese zwei Gruppen ist bei allen vier Testgruppen gleichermaßen vorhanden

Programmieren von Algorithmenanimationen als Lernhilfe

- Alternative zum klassischen Einsatz der Algorithmenanimation: Studenten schauen sich nicht vorbereitete Animationen an, sondern programmieren selbst Algorithmenanimationen

- durch die Entwicklung der Animation sollen die Studenten die wichtigsten Eigenschaften des Algorithmus lernen

- Anforderungen an ein Algorithmenanimationssystem für diesen Zweck:

  1. sehr leicht zu erlernen und zu benutzen

  2. Entwicklung der Animation muß sehr stark mit dem Algorithmus und seinen Operationen verbunden sein

Samba: ein Algorithmen-Animations-Tool

Polka

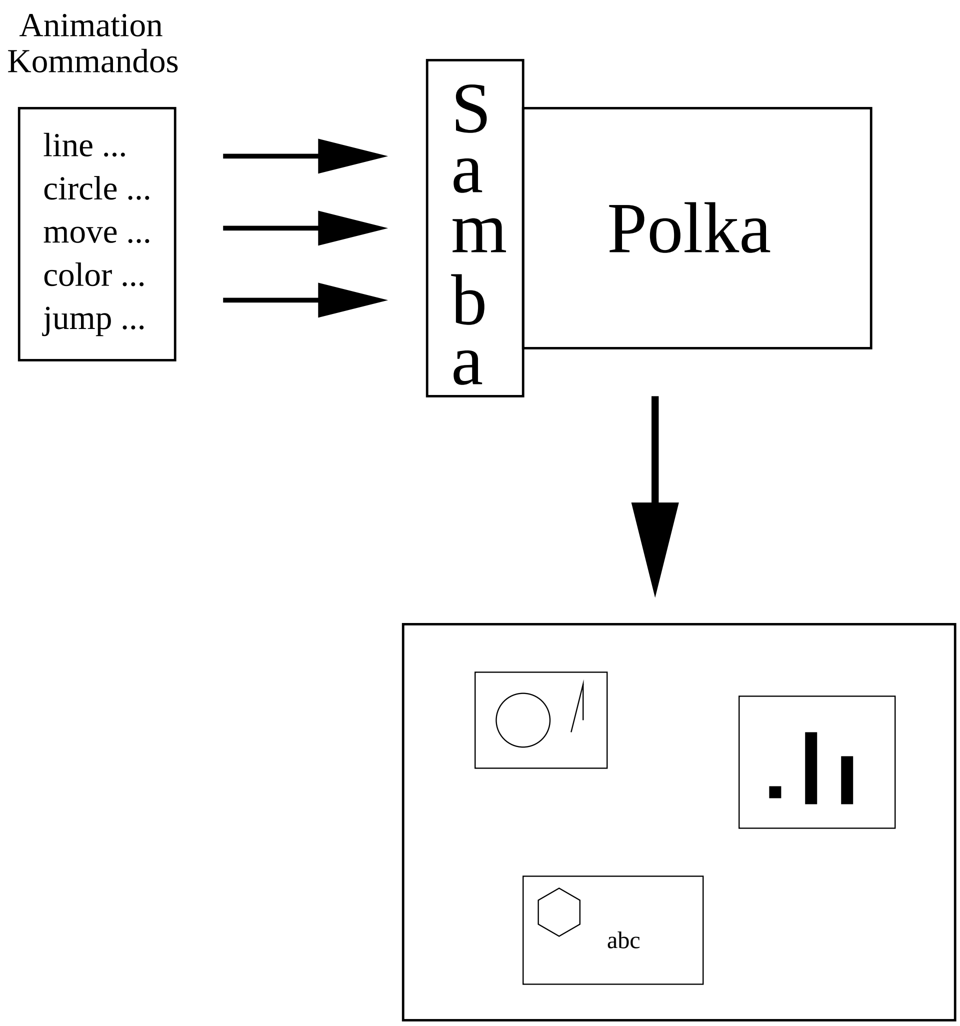
- allgemeines Animationssystem
- besonders gut geeignet für Animationen von Algorithmen und parallelen Berechnungen
- leicht zu benutzen und dennoch sehr mächtig
- man muß kein Graphiker sein, um eine Animation zu erstellen
- man muß in C++ programmieren

Samba

- Frontend zu Polka:
  - leichter und verständlicher als Polka
  - unterstützt 90% der Basisfunktionalität von Polka
- interaktiver Animationsinterpreter
- kann mit **jeder** Programmiersprache benutzt werden
- es gibt Versionen für Unix, Windows und ein Java-Applet
- kann auch im Batchmodus benutzt werden

- Eingabeformat ist sehr leicht zu erlernen und zu benutzen

- liest als Eingabe eine Reihe von ASCII Kommandos, ein Kommando pro Zeile, und führt entsprechende Animation aus, wobei es Routinen von Polka aufruft



Die Architektur von Samba

Kommandos in Samba

- jedes Kommando besteht aus mehreren Feldern

- das erste Feld legt den Typ fest und bestimmt eindeutig die Anzahl der folgenden Felder

- es gibt zwei Gruppen von Kommandos

  1. Kommandos zum Erzeugen graphischer Objekte

     – zum Beispiel

     ```
     line l3 0.1 0.1 0.2 0.2 green thin
     circle 27 0.8 0.7 0.1 red solid
     ```

     – das zweite Feld ist der Bezeichner des Objekts (im Beispiel l3 bzw. 27)

  - die folgenden Felder bestimmen die Attribute der Objekte

2. Kommandos zum Verändern existierender Objekte (Aktionen der Animation)
   - zum Beispiel

     ```
     move 13 0.5 0.6
     color 27 blue
     jump 27 0.3 0.4
     ```

   - Objektbewegungen können gleitende Bewegungen oder diskrete Sprünge sein

- Samba erlaubt mehrere Views: mit **`viewdef`** wird ein neues Animationsfenster erstellt und mit **`view`** werden die folgenden Kommandos auf dieses Fenster geleitet

- mit { und } können Animationskommandos synchronisiert werden: alle Kommandos in diesem Block beginnen zur gleichen Zeit

Entwicklung von Animationen mit Samba

- Festlegen der Interesting Events
- Annotation durch `print` Statements
- bei der Ausführung des Programms wird ein Trace erstellt
- dieser Trace wird an Samba weitergeleitet

  1. interaktiv:

     ```
     % algoanim | samba
     ```

  2. der Trace wird zuerst in eine Datei gespeichert

     ```
     % algoanim > tracer
     % samba tracer
     ```

- das Userinterface von Samba erlaubt schrittweise durch die Animation zu laufen

- wichtige Eigenschaft von Samba: Benutzung von ASCII-Kommandos

  1. es muß keine komplexe Sprache gelernt werden

  2. Unabhängigkeit von der Programmiersprache, in der der Algorithmus implementiert wird

Beispiel: Insertion-Sort

```
int A[10];
double width, stdheight;

init()
{
  int i;
  double xpos, ypos, xsize, ysize;

  A[0]=5; A[1]=2; A[2]=7;
  A[3]=9; A[4]=4; A[5]=8;
  A[6]=3; A[7]=1; A[8]=10; A[9]=6;

  width=1/12.0;
  stdheight=1/24.0;
  ypos = stdheight;
  xsize = width-0.01;
  for (i=0; i<10; i++)
    {
      xpos = width*(i+1)+0.005;
      ysize = stdheight*(A[i]);
      printf("rectangle %d %f %f %f %f blue solid\n",
             i,xpos,ypos,xsize,ysize);
    }
  printf("triangle tri %f %f %f %f %f %f green solid\n",
         width*1.25,stdheight/3,width*1.75,stdheight/3,
         width*1.5,stdheight/3*2);
  printf("pointline line %f %f %f %f red thin\n",
         width*2,0.0,width*2,stdheight*12);
}
```

```
main()
{
  int i,k,x;

  init();
  for (k=1;k<10;k++)
    {
      x=A[k];
      i=k-1;
      printf("moverelative line %f %f\n",width,0.0);
      printf("moverelative %d %f %f\n",k,0.0,0.5);
      printf("move tri %f %f\n",(i+1.5)*width,stdheight/2);
      while (i>=0 && A[i]>x)
        {
          A[i+1]=A[i];
          printf("moverelative %d %f %f\n",i,width,0.0);
          printf("swapid %d %d\n",i,i+1);
          i--;
          printf("moverelative tri %f %f\n",-width,0.0);
        }
      A[i+1]=x;
      printf("move %d %f %f\n",i+1,
     width*(i+2)+0.005,stdheight);
    }
}
```

Benutzung von Samba in einer Algorithmenvorlesung

- drei Programmieraufgaben während des Semesters:

  1. "Flying Logos": zum Kennenlernen des Animationssystems (Animation der Namen der Studenten)

  2. Quicksort: Entwicklung einer Algorithmenanimation mit deren Hilfe der Algorithmus erklärt werden soll

  3. Minimum spanning tree (MST): Entwicklung einer Algorithmenanimation die entweder Kruskal's oder Prim's Algorithmus benutzt

- Beobachtungen
  - bei Quicksort haben alle Studenten zuerst den Algorithmus implementiert und dann die Animation dazugefügt
  - bei MST haben einige Studenten die Animation parallel zum Algorithmus implementiert und als Debugging benutzt

- Feedback der Studenten:
  Skala von 1 (nicht zutreffend) bis 5 (zutreffend)

	WS 95	WS 96
1. Animation war eine wichtige Lernerfahrung	4.34	4.05
2. Animation war für das Verständnis hilfreich	4.21	4.05
3. Animation hat Spaß gemacht	4.21	4.00
4. Programmierung der Animation war schwierig	2.83	2.55

Fazit

- 1. Studie:
  relativ aufwendig, mit Beteiligung von Psychologen, doch kein relevanter Vorteil für Algorithmenanimationen

- 2. Studie:
  nur Erfahrungsbericht, Studenten empfinden subjektiv einen Lernerfolg (kein Vergleich)

- offene Frage bleibt: erhöht Einsatz von Algorithmenanimation den Lernerfolg
  - wie muß ein Experiment aussehen (Lernphase, Tests, Teilnehmerauswahl)
  - waren die Algorithmen gut ausgewählt

- waren die Animationen gut programmiert
- Vorwissen der Teilnehmer
- Aufgabenstellung: waren die Aufgaben besonders zum Vor- / Nachteil von Algorithmenanimation ausgelegt
- wie beeinflußt das Bewußtsein, Teilnehmer einer Studie zu sein, das Testverhalten der Studenten (Manipulation ?)