XML-basierte dreidimensionale Animation von Algorithmen und Datenstrukturen

Ashraf Abu Baker, Dirk Grunwald, Stefan Kappes
Institut für Graphische Datenverarbeitung
Johann Wolfgang Goethe-Universität Frankfurt am Main
Robert-Mayer-Straße 10

60435 Frankfurt
baker@gdv.cs.uni-frankfurt.de
grunwald.dirk@gmail.com
s_kappes@cs.uni-frankfurt.de

Abstract: Der zunehmende Gewinn an Bedeutung und Akzeptanz von Algorithmenvisualisierungssystemen (AVS) – sowohl bei Lernenden als auch bei Lehrenden – führt zu deren verstärkten Einsatz in der Lehre als moderne E-Learning-Instrumente. Während für viele Algorithmen eine zweidimensionale Visualisierung aus didaktischen Gründen vollkommen ausreicht, existieren zahlreiche Algorithmen, z. B. aus den Bereichen der Computergrafik und Bioinformatik, deren wesentliche Eigenschaften erst bei einer Visualisierung im Dreidimensionalen zur Geltung kommen. Da fast alle derzeit existierenden AVS sich jedoch auf die zweidimensionale Visualisierung von konventionellen Algorithmen und Datenstrukturen beschränken, entwickelten wir an unserem Institut einen XML-Standard (Auszeichnungssprache) [Th07, Ka05] zur dreidimensionalen Animation von Algorithmen und Datenstrukturen und stellen diesen in dieser Arbeit vor. Unser Hauptaugenmerk richtet sich dabei auf die Anwendbarkeit für rechenintensive und $\mathcal{NP}$-vollständige Algorithmen [We03].

Nach einer kurzen Einführung in die Thematik und Motivation unserer Arbeit, gehen wir im zweiten Abschnitt auf die Ergebnisse unser Anforderungs- und State-of-the-Art-Analysen ein. Im darauffolgenden Abschnitt stellen wir die Struktur, den Aufbau und die Bestandteile sowie einige Features der von uns entwickelten Auszeichnungssprache (bzw. Standard) xml3DVis vor. Anschließend erfolgt eine Beschreibung des von uns implementierten Interpreters, mit dem in xml3DVis erstellte Skripte geparst, interpretiert und visuell dargestellt werden können. Im vorletzten Abschnitt zeigen wir anhand eines rechts aufwendigen Beispiels, wie wir mithilfe unserer Sprache das $\mathcal{NP}$-vollständige Travelling-Salesman-Problem animiert haben. Abschließend evaluieren wir die Sprache und zeigen ein überraschendes Ergebnis.

1 Einführung und Motivation

In den vergangenen Jahren wurden immer mehr Studien [BCS96, LBS94] durchgeführt, die bestehende Algorithmenvisualisierungssysteme in Bezug auf ihre Lerneffektivität untersuchen, mit dem Ziel, die didaktische Qualität der darunterliegenden Algorithmenvisualisierungen zu verbessern und den Lerneffekt zu steigern. Algorithmenvisualisierungs-

systeme [Rö02, Br91, Ba98] gewinnen aus diesem Grund immer mehr an Bedeutung und Akzeptanz [As06], sowohl bei Lehrenden als auch bei Lernenden. Dementsprechend steigen die Erwartungen an Algorithmenvisualisierungen kontinuierlich und somit wachsen auch die Anforderungen an neue Algorithmenvisualisierungssysteme [Fa02]; beispielsweise gelten statische Animationen[1] als nicht besonders lernfördernd. Eine dieser Anforderung ist die Visualisierung von Algorithmen im Dreidimensionalen [BN93]. Obwohl bei vielen Algorithmen aus den Bereichen der Bioinformatik und der Computergrafik die wesentlichen Eigenschaften erst bei einer 3D-Visualisierung angemessen zur Geltung kommen und einige Studien die Relevanz der dritten Dimension für eine hohe Lerneffektivität bereits gezeigt haben [Fa02], beschränken sich die meisten der bisherigen Algorithmenvisualisierungsssysteme auf die Visualisierung in 2D. Im Rahmen unserer Forschungsarbeiten gelang die fast vollständig automatisierte Erstellung echter visueller 3D-Simulationen für diverse Algorithmen und Datenstrukturen, darunter einige aus den Bereichen der Bioinformatik und der Computergrafik, die ein besonders hohes Maß an Anforderungen erfüllen. Ein Problem stellen dabei die Versuche, Algorithmen zur Lösung $\mathcal{NP}$-vollständiger [We03] bzw. rechenintensiver Probleme zu simulieren, dar. Bei der Simulation eines $\mathcal{NP}$-vollständigen Algorithmus mit entsprechend großer Eingabemenge werden zur Berechnung des Endresultats exponentiell viele Zwischenergebnisse berechnet. Um ein späteres Rückgängigmachen eines Schrittes [PK94] zu ermöglichen, muss für jeden dieser durchgeführten Schritte die Speicherung aller Informationen und Daten (Zustände) erfolgen, was zu einem Speicherüberlauf führen kann.

Somit eignet sich der Ansatz der visuellen Simulation von Algorithmen in Echtzeit zwar hervorragend für alle Algorithmen mit angemessener Laufzeit; für rechenintensive Algorithmen ist er vollkommen ungeeignet. Aufgrund der beiden zuvor genannten Probleme bleibt dem Animationsautor zur Visualisierung rechenintensiver Algorithmen nur noch der Ansatz der statischen Animation, wenngleich diese Art der Darstellung als nicht besonders lernfördernd gilt. Das Ziel unserer Arbeit ist daher die Entwicklung eines Algorithmenanimationsstandards (Algorithmenanimationsskriptsprache) für Animationen beliebiger Algorithmen in 3D. Dieser Standard soll insbesondere auch für die Animation von $\mathcal{NP}$-vollständigen Algorithmen in der Lehre eingesetzt werden.

[1] Wir unterscheiden in unserer Arbeit zwischen zwei Arten von Algorithmenvisualisierungen: Statische Animationen und visuelle dynamische Simulationen.

Eine statische Animation ist eine immer gleich aussehende Abfolge von Frames, die keine Art der Benutzerinteraktion unterstützt und das Verhalten eines Algorithmus bzw. einer Datenstruktur für eine vom Animationsautor zum Zeitpunkt der Animationserstellung festgelegte Eingabefolge visualisiert [Rö02]. Auf den Lernenden wirkt die Animation wie ein Film, den er nur vor- und rückwärts spulen kann. Es bestehen keine weiteren Möglichkeiten, den Ablauf der Animation zu beeinflussen. Das Lernen erfolgt durch passives Beobachten des Animationsfilms.

Visuelle Algorithmensimulationen hingegen sind dynamische Animationen, denen eine echte Simulation des Algorithmus bzw. der Datenstruktur zugrunde liegt. Der Lernende kann aktiv ins Geschehen eingreifen, z. B. indem er eine Simulation zu jedem Zeitpunkt abbrechen und mit einer neuen Eingabemenge erneut starten kann. Außerdem hat der Benutzer die Möglichkeit, während des Ablaufs der Simulation mit dem Algorithmus bzw. mit der Datenstruktur zu interagieren, z. B. Elemente in eine Datenstruktur einzufügen bzw. zu entfernen; dies alles ist bei statischen Animationen nicht möglich.

2 „State-of-the-Art“- und Anforderungsanalyse

Wir erstellten zunächst eine Liste von Anforderungen, die ein derartiger Standard erfüllen muss. Eine wesentliche Eigenschaft des neuen Standards, ist seine Fähigkeit, beliebige Algorithmen in 3D zu animieren, die allen bislang im Rahmen von Studien erstellten Anforderungen genügen [BCS96, Rö02a]. Die Sprache soll die Modellierung und Verwendung von Datenstrukturen und ihrer zugehörigen Operationen auf möglichst komfortable Art und Weise unterstützen. Dies impliziert, dass komplexere Datenstrukturen (Bäume, Graphen, mehrdimensionale Arrays etc.) verfügbar oder zumindest leicht zu konstruieren sind. Darüber hinaus soll die Sprache leicht erlernbar sein. Anwendern ohne bzw. mit geringen Programmierkenntnissen soll damit die Möglichkeit gegeben werden, Animationen ohne großen zusätzlichen Lernaufwand zu erstellen. Einzelne Schritte der Animation sollen durch Keyframes festgelegt werden; optionale Vergabe von Keyframe-IDs ermöglicht desweiteren die direkte Ansteuerung bestimmter Keyframes durch den Animationsinterpreter. Für die Syntax der Sprache soll ein plattformunabhängiges, portables Format verwendet werden, dass nicht nur die manuelle, sondern auch die tool-basierte Skripterstellung unterstützt [Rö02a]. Weiterhin muß die Möglichkeit bestehen, Skripte leicht parsen, interpretieren, austauschen und konvertieren zu können [Cl99]. Der weit verbreitete XML-Standard [Th07] unterstützt die zuletzt genannten Anforderungen; er liegt deshalb der Syntax unserer neuen Sprache zugrunde.

Unsere „State-of-the-Art“-Analyse ergab, dass es keinen Standard speziell für die 3D-Animation von Algorithmen gegeben hat, als wir mit unserer Arbeit begonnen haben und nach unserem Wissen bis heute nicht gibt. Ein jedoch bereits existierender XML-Standard, der die Erstellung von allgemeinen Animationen unterstützt, ist der bekannte X3D-Standard [Br05]. X3D ist eine XML-basierte 3D-Beschreibungssprache für interaktive 3D-Inhalte, die auch für die Animation von Algorithmen verwendet werden könnte.

Bevor wir mit der Erstellung eines neuen Standards begonnen haben, zogen wir den Einsatz von X3D in Betracht. Hierfür analysierten wir das Schema und stellten fest, dass er aus vielen Gründen, die wir hier nur verkürzt aufführen wollen, für den Einsatz als Algorithmenanimationsstandard aus unserer Sicht nicht geeignet ist. X3D ist zu mächtig. Um es anwenden zu können, muss erst das ganze Schema erlernt werden. Dies widerspricht unserer Forderung nach einer leicht zu erlernenden Animationssprache. Werden komplexere Datenstrukturen und Methoden benötigt, so müssen unter X3D zunächst eine Vielzahl von graphischen Primitiven angelegt und daraus die entsprechenden komplexen Datenstrukturen zusammengesetzt werden. Algorithmenanimationsspezifische Konzepte wie Quell-Code-Anzeige, Code-Highlighting, Syntax-Coloring, Quiz, Anzeige von Narratives und Dokumentation etc. müssen unter X3D manuell kodiert werden. Die Integration eines bestehenden X3D-Interpreters in ein bestehendes Algorithmenanimationssystem wäre schwierig. Die komplette Neuprogrammierung eines X3D-Interpreters hingegen wäre nicht zuletzt wegen der Implementierung der Undo/Redo-Funktionalität [PK94] aufgrund der sehr vielfältigen Transformationsmöglichkeiten von X3D sowie der Integration in ein bestehendes Algorithmenanimationssystems zu aufwendig.

3 Struktur und Aufbau von xml3DVis

Da eine komplette Beschreibung aller xml3DVis-Sprachelemente den Rahmen dieser Arbeit sprengen würde, beschränken wir uns hier auf eine kurze Vorstellung der wichtigsten Sprachbestandteile.

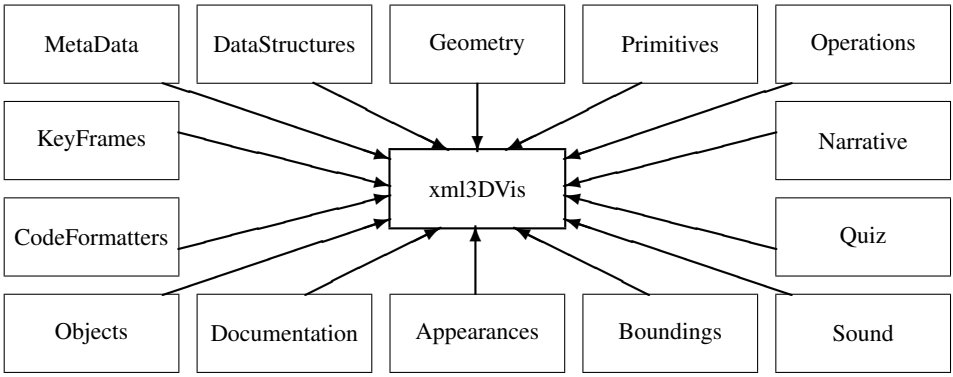

Abbildung 1: Logische Aufteilung aller xml3DVis-Sprachelemente in 14 Kategorien

Eine xml3DVis-Datei könnte die folgende Struktur aufweisen:

```
<xml3DVis>
  <metadata>        ... </metadata>
  <boundings>       ... </boundings>
  <objects>         ... </objects>
  <scene>           ... </scene>
  <sounds>          ... </sounds>
  <operations>      ... </operations>
  <codeFormatters>  ... </codeFormatters>
  <code>            ... </code>
  <keyframes>       ... </keyframes>
  <documentation>   ... </documentation>
</xml3DVis>
```

Im Abschnitt **MetaData** können nicht nur Informationen über die Datendatei (wie Autor, Version, etc.) gespeichert, sondern auch allgemeine Informationen festgelegt werden, die die Animation und ihr Abspielverhalten betreffen. **Boundings**-Elemente werden zur Definition des Wirkungsbereichs von Sichtbarkeit und Objektaktivierung verwendet. Objekte, die unter Umständen zu Beginn der Animation noch nicht im Szenegraphen enthalten sind, werden im Abschnitt **Objects** eingeführt. Die Positionierung der zuvor eingeführten Objekte in den Szenengraphen erfolgt im Abschnitt Operations – sowohl zu Beginn als auch im weiteren Verlauf der Animation. Eigenschaften, die das Erscheinungsbild von graphischen Objekten bestimmen (z. B. Farbe, Textur, Beleuchtungsparameter etc.) werden diesen mithilfe des **Appearance**-Elementes zugewiesen. Das Wurzelelement für die 3D-Szene innerhalb der Datendatei ist das obligatorische Element **Scene** [Bo08] mit

den beiden optionalen Attributen *defaultBounding* und *viewerDistance*. Das erste gibt dem Animationsautor die Möglichkeit, die ID eines Boundings zu referenzieren und mit dem zweiten wird die Entfernung zwischen dem Betrachter und dem Zentrum der Szene zu Beginn der Animation festgelegt.

xml3DVis unterstützt 21 graphische Primitive und geometrische Strukturen; aus diesen einfachen graphischen Geometrien und Grundformen können weitere komplexere Objekte zusammengesetzt bzw. komplexere Datenstrukturen definiert werden. Zusammen mit der Unterstützung für konventionelle Datenstrukturen (wie ein- und mehrdimensionale Arrays, Bäume, Graphen etc.) [OW93] erfüllt xml3DVis eine der Hauptanforderungen, die wir zu Beginn an einen solchen Standard gestellt haben.

```
<graph>
  <defaultInsertElement>
    <sphere ID="sphere1" radius="0.2" divisions="70">
      <appearance> <material diffusecolor="blue"/> </appearance>
    </sphere>
  </defaultInsertElement>
  <defaultEdge directed="true" radius="0.02">
    <appearance> <material diffusecolor="orange"/> </appearance>
  </defaultEdge>
  <nodes>
    <node ID="node1"> <offset x="-1" y="0" z="1"/> </node>
    <node ID="node1"> <offset x=" 1" y="0" z="1"/> </node>
    ...
  </nodes>
  <edges>
    <edge from="node1" to="node2"/>
    <edge from="node2" to="node3"/>
    ...
  </edges>
</graph>
```

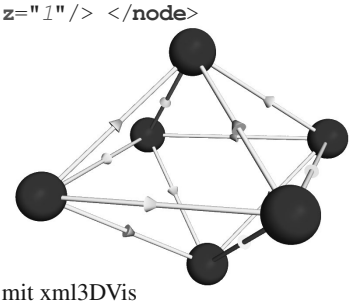

Abbildung 2: Darstellung eines Graphen mit xml3DVis

Die von xml3DVis unterstützten **Operationen** zur Anwendung auf die zuvor definierten Objekte kann man in die folgenden drei Kategorien einteilen:

1. Animationsoperationen zur visuellen Animation von graphischen Primitiven
2. Ereignisabhängige Operationen, die am Anfang oder am Ende eines Keyframes ausgeführt werden können, wenn ein bestimmtes Ereignis eingetreten ist (*playSound*, *showQuiz*, *setAppearance*, *setText* etc.) xml3DVis unterstützt insbesondere die Anzeige von Testaufgaben in Form von Single- und Multiple-Choice-Fragen (**Quiz**), sowie Erzähltexten (**Narratives**).
3. Datenstrukturoperationen zur Manipulation von Datenstrukturen (*add*, *remove*, *search* etc.)

Im Abschnitt **Code** werden die Codelistings definiert. Während der Inhalt des Codes bereits zu Beginn einer Animation festgelegt wird. Im Verlauf der Animation können

allerdings unterschiedliche Zeilen hervorgehoben werden (Code-Highlighting). Dadurch wird der Betrachter in einer Algorithmenanimation darüber informiert, welcher Teil des Codes (Anweisungen) gerade abgespielt wird. **Code-Formatierer** werden zur Syntax-Formatierung des Codes der zu animierenden Algorithmen bzw. Datenstrukturen verwendet; hiermit kann die einheitliche Darstellung von Schlüsselwörtern, Kommentaren, Methodenaufrufen etc bezüglich Farbe und Schriftstil sowohl im PseudoCode als auch in den gängigen Programmiersprachen gewährleistet werden.

Animationen in xml3DVis werden prinzipiell in zeitliche Abschnitte, die sogenannten **Keyframes** [Wa06], unterteilt. Ihre Nummerierung beginnt bei 1 und wird als Framenummer bezeichnet. Neben dieser Nummer enthält ein Keyframe zusätzliche Informationen darüber, welche Operationen am Anfang, Ende oder während eines Keyframes ausgeführt werden. Desweiteren erfolgt hier die Steuerung von Erzähltext- und Codelisting-Ausgabe.

```
<keyframe  frame="1"  label="first frame">
  <operation IDRef="rot360"  duration="1000"  start="1000"/>
  <event     IDRef="playSound1"  type="onStart"/>
  <codeview>
    <highlight partRef="javaPart">       <line>line1</line>
                                         <line>line4</line> </highlight>
    <highlight partRef="pseudoCodePart"> <line>line1</line> </highlight>
  </codeview>
  <narrative>
    <value> Das ist der <![CDATA[<Erzähltext>]]> zu Keyframe 1 </value>
    <contentType> text </contentType>
    <codeFormatterRef> javaFormatter </codeFormatterRef>
    <codeFormatter>
      <wordFormat  color="blue"> <word> Keyframe </word> </wordFormat>
    </codeFormatter>
  </narrative>
</keyframe>
```

Zu Anfang jedes Keyframes kann ein neuer Erklärungstext im Textbereich des Tabellenreiters Narratives angezeigt werden. Die Definition eines Erklärungstextes wird durch das Unterelement **Narrative** eingeleitet. Für das bessere Verständnis des zu animierenden Sachverhalts bietet xml3DVis dem Animationsautor die Möglichkeit, die Animation um eine Anzeige tutorieller Inhalte (wie z. B. Einführung in den animierten Algorithmus) in Form von statischen Texten oder **Dokumentation** zu erweitern.

Mit der Unterstützung der o. g. Features erfüllt xml3DVis die wesentlichen Anforderungen, die im Rahmen von zahlreichen Studien gestellt wurden und ermöglicht damit die Animation einer großen Auswahl an Algorithmen und Datenstrukturen. Wir werden jedoch im Abschnitt 6 zeigen, dass es auch Algorithmen gibt, die jede Algorithmenanimationssprache an ihre Grenzen stoßen lässt.

4 Implementierung

Damit in xml3DVis erstellte Animationen in einem AVS abgespielt werden können, muss zuerst ein Interpreter implementiert werden, der v3d-Dateien[2] liest, parst, interpretiert und visualisiert. Unabhängig von dieser Arbeit haben wir an unserem Institut eine Algorithmenvisualisierungslernplattform namens 3D-VISIAN (*Three Dimensional Visual Simulation and Animation System for Algorithms and Data Structures*) entwickelt, mit der dreidimensionale visuelle Simulationen und statische Animationen geladen und in Echtzeit abgespielt werden können. Im Rahmen dieser Arbeit haben wir nun einen solchen Interpreter für v3d-Dateien entwickelt und in 3D-VISIAN integriert.

Der xml3DVis-Interpreter besteht aus drei Hauptkomponenten: einem XML-Parser [Ap01] (zum Parsen von v3d-Dokumenten), einer Datenmodellierungsengine (zum Abbilden der geparsten XML-Elemente auf ein Datenmodell) und einer Visualisierungsengine. Letztgenannte ist verantwortlich für die Analyse der im Datenmodell vorliegenden Daten und Umwandlung in (3D)-graphische Primitive sowie für die Interpretation und Ausführung aller vorliegenden Operationen.

Ein wesentlicher Aspekt bei der Entwicklung eines für Animationen geeigneten Interpreters ist die Sicherstellung der unbegrenzten linearen Undo/Redo-Funktionalität [PK94]. Animationen, deren Schritte nicht rückgängig gemacht werden können, gelten als nicht lernfördernd [Rö02a]. Die Tatsache, dass die Bewerkstelligung dieser Aufgabe bei unserer Implementierung nicht zum Zuständigkeitsbereich der Skriptsprache gehört, sondern vollständig beim Interpreter angesiedelt wird, erleichtert dem Animationsautor die Erstellung von Animationen — ein wesentlicher Schritt zur Erfüllung der Anforderung an die leichte Erlernbarkeit des Standards.

5 Anwendungsbeispiele

Als Anwendungsbeispiel für die Implementierung unseres Standards und zur Demonstration der Fähigkeiten von xml3DVis stellen wir nun die Visualisierung des Travelling-Sales-Man-Problems [Nä06] für acht Städte vor. Wir haben aus drei Gründen dieses Problem in einer dreidimensionalen Szene visualisiert, obwohl dessen Visualisierung auch in einem zweidimensionalen Raum erfolgen kann:

- Um das Problem realitätsnah zu präsentieren.
- Um zu zeigen, wie man komplexe Animationen erstellen kann.
- Und um einen Eindruck davon zu geben, wie mächtig xml3DVis ist.

Das Problem gilt als $\mathcal{NP}$-vollständig [We03], ist also vermutlich durch keinen Algorithmus in polynomieller Zeit lösbar. Alle bisher hierfür bekannten Algorithmen benötigen exponentielle Laufzeit. Daher eignet sich dieses Problem besonders für die Visualisierung mittels einer Animation.

[2] xml3DVis-Animationsdateien werden mit der Erweiterung v3d gespeichert.

Abbildung 3: Animation von TSP mithilfe von xml3DVis unter 3D-VISIAN

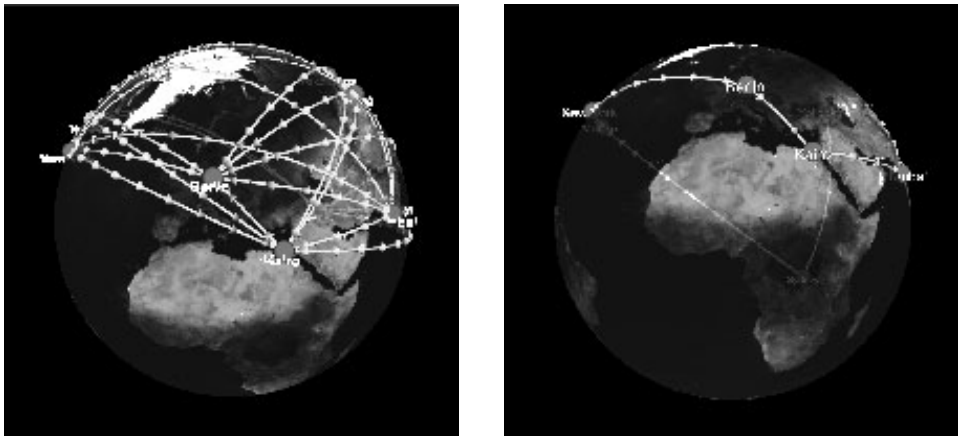

Abbildung 4: Das Travelling-Sales-Man-Problem auf der Erdkugel

Bei der Simulation dieses Problems stößt man schon für relativ kleine Knotenzahlen an die Grenze dessen, was mit Computern berechenbar ist — sowohl bezüglich Laufzeit- als auch Speicherkomplexität: Nach [Nä06] erreicht man mit der Bruteforce-Methode, also dem Ausprobieren aller möglichen Wege, für acht Knoten noch Rechenzeiten von weni-

gen Sekunden; bei 16 Knoten hingegen muss man bereits ca. 20 Jahre auf ein Ergebnis warten. Auch durch die Verwendung besserer Entwurfsmethoden kann man das Komplexitätsproblem nicht in den Griff bekommen. So kommt es unter Anwendung Dynamischer Programmierung [CLRS01] für 16 Knoten bereits zu Laufzeiten, die sich in der Größenordnung von Stunden bewegen. Zur Verdeutlichung: Eine Simulation mit acht Städten benötigt 1145 Schritte (mit neun Städten wären es bereits 5623 Schritte). Abbildung 4 zeigt links alle möglichen Verbindungen zwischen den acht Städten; auf der rechten Seite wird die optimale Route angezeigt.

Zur Vermeidung dieser gigantischen Rechenzeiten, setzt man zur Visualisierung statische Animationen ein, da diese die Berechnungen der optimalen Lösung nicht zur Laufzeit durchführen, sondern lediglich eine vorbereitete Animation abspielen. Vorab mussten jedoch zwei Programme erstellt werden. Die Aufgabe des ersten bestand in der Berechnung der optimalen Lösung, die des zweiten war die Erstellung der Datendatei für xml3DVis.

Bei acht Städten wäre es natürlich auch noch möglich gewesen, den Algorithmus zu simulieren. Der Nachteil der Simulation liegt darin, dass es für Lernzwecke weder sinnvoll ist, gleichartige Schrittfolgen in einem Algorithmus immer und immer wieder zu sehen, noch so lange zu warten, bis die Abarbeitung des Algorithmus die nächste „interessante“ Schrittfolge erreicht. Viel mehr ist es in solch einem Fall nützlicher, einige ausgewählte, pädagogisch wertvolle Schritte auszusuchen, die den Weg zum Ziel weisen und diese zuvor zu berechnen.

6 Evaluation der Arbeit und Fazit

Trotz aller Nachteile, die eine statische Animation eines Algorithmus oder einer Datenstruktur mit sich bringt, bleibt diese Art der graphischen Darstellung die einzige Möglichkeit, rechenintensive Algorithmen zu visualisieren. Nicht rechenintensive Algorithmen sollten nur durch visuelle Echtzeit-Simulationen graphisch umgesetzt werden, da diese wesentlich lerneffektiver sind als statische Animationen. Der Einsatz von statischen Algorithmenanimationen sollte daher nur für die Visualisierung von rechenintensiven Algorithmen in Betracht gezogen werden.

Statische Algorithmenanimationen können allerdings nicht nur mittels Animationsskriptsprachen, sondern auch mithilfe von höheren Programmiersprachen erstellt werden. Beide Alternativen haben jedoch Vor- und Nachteile. Der Einsatz von Animationsskriptsprachen hat den Vorteil, dass hierfür keine Programmierkenntnisse vorausgesetzt werden müssen. So können z. B. auch Designer professionelle und visuell ansprechende Animationen, die vielen Gestaltungsregeln unterliegen [SM00], ohne jegliche Programmierkenntnisse erstellen. Der Einsatz von Programmiersprachen hingegen erfordert sowohl Design- als auch Programmiererfahrung. Und im Gegensatz zu Designern legen Programmierer häufig weniger Wert auf die Gestaltung der Animation. Andererseits sind höhere Programmiersprachen mächtiger als Algorithmenanimationssprachen, weil diese in der Regel weder Schleifen noch bedingte Anweisungen unterstützen.

Nun stellt sich jedoch die Frage, ob wegen dem oben genannten Vorteil, den Animationsskriptsprachen gegenüber den gängigen höheren Programmiersprachen aufweisen, statische Animationen mittels Animationsskriptsprachen (und nicht in einer höheren Programmiersprache) erstellt werden sollen. Um diese Frage beantworten zu können, wollen wir hier nun ein weiteres Beispiel eines rechenintensiven und graphisch anspruchsvollen Algorithmus aus dem Bereich der Bioinformatik betrachten.

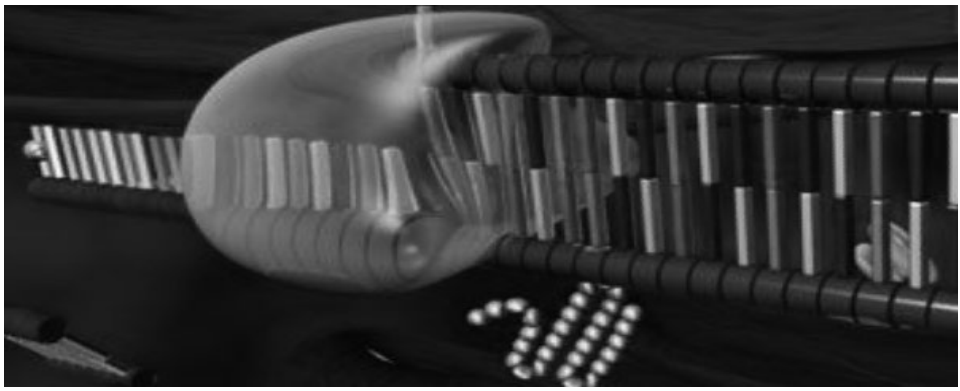

Abbildung 5: Ausschnitt aus einer RNA-Animation

Der Iterated-Loop-Matching-Algorithmus (ILMA) [JXS04] ist ein Algorithmus, der die RNA-Sekundärstruktur inklusive Pseudoknoten vorhersagen kann. Er basiert auf dem Loop-Matching-Algorithmus von Nussinov [NJ80], verwendet die Entwurfsmethode der Dynamischen Programmierung [CLRS01] und nutzt thermodynamische und vergleichende (Kovarianz-)Informationen aus. Der Algorithmus kann jeden Typ von Pseudoknoten vorhersagen, in justierten (ausgerichteten) und einzelnen Sequenzen. Die worst-case-Laufzeit des ILM-Algorithmus beträgt $\mathcal{O}(n^4)$; hierbei ist n die Länge der Eingabesequenz.

Abbildung 5 zeigt einen Ausschnitt einer in einer höheren Programmiersprache erstellten Animation des ILM-Algorithmus. Die Komplexität der Visualisierung lässt erahnen, wie schnell eine Animationsskriptsprache an ihre Grenzen stoßen kann. Theoretisch lässt sich der Algorithmus in einer mächtigen 3D-Algorithmenanimationssprache erstellen. Der Algorithmus ist jedoch graphisch so anspruchsvoll, dass eine angemessene Erstellung nur noch in einer höheren Programmiersprache erfolgen soll.

Da die Erstellung von rechenintensiven Algorithmen in einer Algorithmenanimationssprache mindestens genau so aufwendig sein kann wie in einer höheren Programmiersprache und in bestimmten Fällen wie z. B. beim TSP-Problem fast genau soviel Programmiererfahrung erfordert[3], halten wir den Ansatz, rechenintensive Algorithmen in einer Algorithmenanimationssprache zu erstellen, für nicht geeignet. Selbst wenn Werkzeuge wie z. B. XML-Editoren zur Erstellung der 1873 Zeilen in unserem TSP-Beispiel eingesetzt werden, kommt man an der programmiertechnischen Vorabberechnung der Zwischenschritte (für acht Städte waren es wie bereits erwähnt 1145) nicht vorbei.

[3] Die v3d-Datei für die Animation von TSP besteht aus 1873 Zeilen.

Daher sollte sowohl für statische Animationen als auch für dynamische Simulationen von Algorithmen und Datenstrukturen eine hierzu geeignete höhere Programmiersprache herangezogen werden. Die Entwicklung von neuen Algorithmenanimationsskriptsprachen führt zu keinen nennenswerten Vorteilen.

Literatur

[Ap01] Apache, *Sample dom.DOMAddLines*. Internetressource, November 2001 http://xerces.apache.org/xerces2-j/samples-dom.html#DOMAddLines/

[As06] P. Aschenbrenner, *Generierung interaktiver Lerneinheiten aus visuellen Spezifikationen*. Bundeswehruniversität, 2006.

[Ba98] R. Baecker, *Sorting out Sorting – A Case Study of Software Visualization for Teaching Computer Science* http://www.dgp.toronto.edu/people/RMB/papers/p25.pdf

[BCS96] M. Byrne, R. Catrambone, J. Stasko, *Do Algorithmen Animation aid Learning?* GIT-GVU-96-18, 1996

[BN93] M. H. Brown, M. A. Najork, *Algorithm Animation Using 3D Interactive Graphics*, 1993

[Bo08] D. J. Bouvier, *Java 3D API Tutorial*, 2008. http://java.sun.com/developer/onlineTraining/java3d/

[Br91] M. H. Brown, *Zeus: A System for Algorithm Animation and Multiview Editing*, 1991

[Br05] D. Brutzman, *X3D Specifications, Encodings and Language Bindings*. Internetressource, Nov. 2005

[Cl99] J. Clark, *XSL Transformations (XSLT) Version 1.0*. Internetressource, Nov. 1999. http://www.w3.org/TR/xslt

[CLRS01] T. Cormen, C. Leiserson, R. Rivest, C. Stein, *Introduction to Algorithms*, N.Y., 2001.

[Fa02] N. Faltin, *Strukturiertes aktives Lernen von Algorithmen mit interaktiven Visualisierungen*, Dissertation an der Universität Oldenburg, 2002

[JXS04] Y. Ji, X. Xu, and G. D. Stormo, *A graph theoretical approach for predicting common RNA secondary structure motifs including pseudoknots in unaligned sequences*, 2004, Bioinformatics, 20(10), 1591–1602.

[Ka05] V. Karavirta, *XAAL*, 2005. http://www.cs.hut.fi/Research/SVG/XAAL/

[LBS94] A. Lawrence, A. Badre, and J. T. Stasko, *Empirically Evaluating the use of Animations to teach Algorithms*, IEEE 1994, S. 48–54, St. Louis, MO 1994.

[Na03] T. Naps et al., *Exploring the role of visualization and engagement in computer science education*, 2003.

[Na05] T. Naps et al., *Development of XML-based tools to support user interaction with algorithm visualization*, ACM SIGCSE Bulletin archive Volume 37, Issue 4, p. 123–138, New York, NY, USA, 2005.

[Nä06] S. Näher, *Das Travelling Salesman Problem.* Internetressource, Okt.2006. http://www-i1.informatik.rwth-aachen.de/~algorithmus/algo40.php

[NJ80] R. Nussinov, and A. B. Jacobson, *Fast algorithm for predicting the secondary structure of single-stranded RNA*. Proc. Nat. Acad. Sci., USA 77,11 (1980)

[OW93] T. Ottmann, P. Widmayer, *Algorithmen und Datenstrukturen*, 2. Auflage. Mannheim, Leipzig, Wien, Zürich: Wissenschaftsverlag, 1993

[PK94] A. Prakash, and M. J. Knister, *A framework for undoing actions in collaborative systems*. ACM Transactions on Computer Human Interaction, 1(4):295–330, 1994

[Rö02] G. Rößling, *Animal-Farm: an extensible framework for algorithm visualization*, Dissertation an der Universität Siegen, 2002

[Rö02a] G. Rößling et al., *A testbed for pedagogical requirements in algorithm visualizations*, ACM SIGCSE Bulletin archive, Volume 34, Issue 3, p. 96–100, New York 2002.

[Rö03] G. Rößling, *Key Decisions in Adopting Algorithm Animations for Teaching*, 2003.

[SM00] H. Schumann, W. Müller, *Visualisierung: Grundlagen und einfache Methoden*, Springer Verlag, 2000

[Th07] H. S. Thompson, *W3C XML Schema Definition Language (XSDL) 1.1 Part 1: Structures*. `http://www.w3.org/TR/xmlschema11-1/`

[Wa06] D. Waston, *Keyframe Animation*, 1996–2006 `http://www.cadtutor.net/dd/bryce/anim/anim.html`

[We03] I. Wegener, *Theoretische Informatik. Eine algorithmische Einführung*, Heidelberg 2003