Reihe: Telekommunikation @ Mediendienste · Band 6
Herausgegeben von Norbert Szyperski, Udo Winand, Dietrich Seibt, Rainer Kuhlen und Rudolf Pospischil

Martin Engelien/Jens Homann (Hrsg.)

Virtuelle Organisation und Neue Medien

Workshop GeNeMe99
Gemeinschaften in Neuen Medien

TU Dresden, 28./29.10.1999

JOSEF EUL VERLAG
Lohmar · Köln

Reihe: Telekommunikation @ Mediendienste · Band 6

Herausgegeben von Prof. Dr. Dr. h. c. Norbert Szyperski, Köln, Prof. Dr. Udo Winand, Kassel, Prof. Dr. Dietrich Seibt, Köln, Prof. Dr. Rainer Kuhlen, Konstanz, und Dr. Rudolf Pospischil, Brüssel

PD Dr.-Ing. habil. Martin Engelien
Dipl.-Inform. (FH) Jens Homann (Hrsg.)

Virtuelle Organisation und Neue Medien

Workshop GeNeMe99
Gemeinschaften in Neuen Medien

TU Dresden, 28./29.10.1999

JOSEF EUL VERLAG
Lohmar · Köln

Die Deutsche Bibliothek – CIP-Einheitsaufnahme

GeNeMe <1999 Dresden> :
GeNeMe 99 : Gemeinschaften in neuen Medien ; Dresden, 28./29.10.1999, an der Fakultät Informatik der Technischen Universität Dresden / Technische Universität Dresden, Fakultät Informatik, Institut für Informationssysteme, Forschungsgruppe "Entwurfsmethoden und Werkzeuge für Anwendungssysteme". Martin Engelien ; Jens Homann (Hrsg.). – Lohmar ; Köln : Eul, 1999
(Reihe: Telekommunikation @ Mediendienste ; Bd. 6)
ISBN 3-89012-710-X

Josef Eul Verlag GmbH
Brandsberg 6
53797 Lohmar
Tel.: 0 22 05 / 91 08 91
Fax: 0 22 05 / 91 08 92
http://www.eul-verlag.de
eul.verlag.gmbh@t-online.de

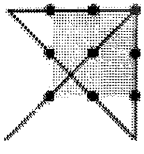
Printed in Germany
Druck: Rosch-Buch, Scheßlitz

Gedruckt auf säurefreiem, 100% chlorfrei gebleichtem, alterungsbeständigem Papier nach DIN 6738

Technische Universität Dresden
Fakultät Informatik • Institut für Informationssysteme
Forschungsgruppe „Entwurfsmethoden und Werkzeuge für Anwendungssysteme"

PD Dr.-Ing. habil. Martin Engelien
Dipl.-Inform. (FH) Jens Homann
(Hrsg.)

an der
Fakultät Informatik der Technischen Universität Dresden



Gefördert von der Klaus Tschira Stiftung,
gemeinnützige Gesellschaft mit beschränkter Haftung

sowie unter Mitwirkung der
GI-Regionalgruppe Dresden

am 28./29.10.1999
in Dresden

E. Lernen in virtuellen Gemeinschaften

E.1. Ein Web-basierter Computergraphik-Kurs im Baukastensystem

F. Hanisch
Dr. R. Klein
Prof. Dr. W. Straßer
Universität Tübingen

Zusammenfassung

Dieses Beispiel eines interaktiven Online-Kurses zeigt, wie virtuelle Experimente die traditionellen Lehrmethoden im Bereich der Computergraphik sinnvoll ergänzen. Das Zusammenspiel von Java und dem World-Wide-Web erlaubt die einheitliche Integration von hypertextuellen Vorlesungstexten, interaktiven Visualisierungen, virtuellen Experimenten, Programmierübungen und Programmierschnittstellen in eine unbeschränkt nutzbare virtuelle Lernumgebung.
Es wird aufgezeigt, wie eine komponenten-basierte Konzeption eines solchen Kurses nicht nur das strukturelle Denken des Schülers fördert, sondern dem Lehrenden auch die Modifikation des Kurses in einfachster Weise ermöglicht. Schließlich wird der Aufwand zur Adaption an weitere Fachgebiete kurz vorgestellt - konkret der Aufbau eines virtuellen Kurses für die Bildverarbeitung.

1 Java und das World-Wide-Web

Einleitend sollen die Motive für die Erstellung des hier beschriebenen Kurses vorgestellt werden.
Einer Vorlesung über Computergraphik mangelt es insbesonders bei der Lehre von Algorithmen oft an Möglichkeiten zur Visualisierung. Umgekehrt gehen in den Übungen oft die Bezüge zur Theorie verloren. Erst mit dem Aufblühen des World-Wide-Webs und der Programmiersprache Java wurde eine **einheitliche Lernumgebung** möglich, die Vorlesungstexte, virtuelle Experimente, Programmierschnittstellen und Programmierübungen untereinander verknüpft und parallel zum theoretischen auch einen spielerischen oder technischen Zugang zur Thematik ermöglicht.
Ein gelungenes **virtuelles Experiment** in der Computergraphik *visualisiert* komplexe Algorithmen, abstrakte Begriffe und Strukturen und erlaubt die *Interaktion* mit ihnen. Aktives Experimentieren mit dem Lerninhalt erleichtert erheblich das Verständnis

komplexer abstrakter Vorgänge [1] und schafft eine motivierende und intuitive Grundlage für vertiefende Literatur.

Basierend auf Java und dem World-Wide-Web ist die gesamte hier vorgestellte Lernumgebung **unbeschränkt** für das Selbststudium und die Vertiefung des Wissens verwendbar: sie ist kostenlos, plattform-unabhängig und anspruchslos hinsichtlich der Software- und Hardware-Basis. Vorausgesetzt wird lediglich ein Browser mit Java2-Plug-in; ein aktiver Internet-Zugang ist nicht notwendig.

2 Der Kurs „Computergraphik spielend lernen"

Der Kurs [2, 3] wird hier in einem Rundgang vorgestellt. Innerhalb des Kursrahmens gliedern sich die **Hypertexte** in die vier genannten, untereinander verknüpften Bereiche: in Vorlesungstexte für das Studium der Theorie, in Informationsseiten zu den Experimenten, in Seiten bezüglich der Programmierschnittstelle und in reine Programm-Quelltexte. Die Informationsseiten geben eine kurze Anleitung zur Verwendung des jeweiligen Experiments und enthalten eine Art elektronischen Lehrer, der schrittweise in die Thematik einführt und theoretische Fragestellungen bespricht.

Die virtuellen **Experimente** sind eingebettet in eine Hypertext-Seite, die wiederum Querverweise zum theoretischen Hintergrund in den Vorlesungstexten und zu den Experiment-spezifischen Informationsseiten mit Hinweisen zur verwendeten Programmierschnittstelle sowie den Quelltexten des Experiments bereitstellt (Abb. 1). Infolge gemeinsamer Basis-Komponenten für die graphische Oberfläche besitzen alle Experimente ein einheitliches Erscheinungsbild und Benutzerverhalten. Die sichtbare Information ist klar strukturiert, damit der Betrachter die wesentlichen Schlüsselkonzepte und ihre Relation zueinander schnell in sein mentales Bild integrieren kann. Eingesetzt wird neben Gestaltgesetzen wie betitelte Rahmen oder Abstände vor allem eine simultane Darstellung der Genese, beispielsweise der schrittweisen Konstruktion eines geometrischen Objektes.

Wichtig für ein vertieftes Verständnis graphischer Algorithmen ist deren **Programmierung.** Für alle Objekte des gesamten Programmpakets steht der Quelltext zur Verfügung, worin Sonderfälle und Fallstricke sichtbar werden, die in der Theorie und im virtuellen Experiment oft nicht erläutert werden. Aus diesem Grunde enthalten die Informationsseiten der virtuellen Experimente Architekturhinweise, die sich näher mit der Programmierung des Experiments beschäftigen und Querverweise auf die entsprechende Dokumentation der Programmierschnittstelle und auf die Quelltexte anbieten.

Praktische **Übungsaufgaben** entstehen einerseits aus vorhandenen virtuellen Experimenten, indem Teile des Quelltextes ausgeschnitten und dem Übungsteilnehmer

sozusagen als Lückentext gegeben werden, und anderseits aus der fortschreitenden Erweiterung des bestehenden Pakets, bei der neue Algorithmen als Implementierungsaufgabe gestellt werden. Auch hier zahlt sich aus, daß die Quelltexte zu jedem Detail einer Graphik-Engine frei zugänglich sind.

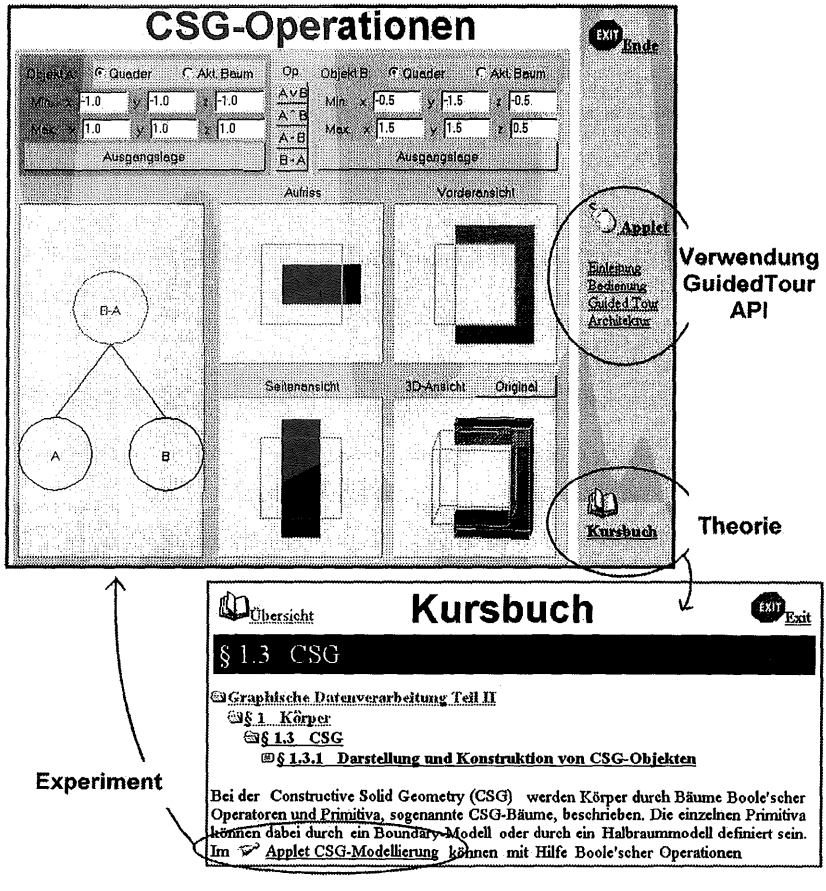


Abbildung 1: Gegenseitige Verknüpfung von Theorie und Experiment mit Experiment-spezifischen Hintergrund-Informationen zur Bedienung und Programmierung.

Virtuelle Experimente, Programmierschnittstelle und Übungsaufgaben verfügen jeweils über ein Inhaltsverzeichnis, so daß schnell auf die gewünschte Information zugegriffen werden kann.

Auf diese Weise ergibt sich je nach Ausprägung des Leserverhaltens ein eher theoretischer, spielerischer oder technischer **Zugang zum Kurs**:
Der Teilnehmer beginnt das Studium mit der Theorie und folgt den Querverweisen in den Vorlesungstexten, die ihn zu geeigneten Experimenten und Übungen führen, wo er sein Wissen praktisch vervollständigt.
Das Interesse eines Teilnehmeres wird durch ein virtuelles Experiment geweckt und er erhält ein intuitives Grundwissen über das Thema, welches ihm beim Verständnis der beiliegenden Vorlesungstexten, Programmierschnittstellen und Quelltexten hilft.
Der Teilnehmer vertieft sein Verständnis durch das Studium eines Experiments und den zugehörigen Informationsseiten, worin ihm mit dem elektronischen Lehrer eine „Learning by Example"-Methode angeboten wird; erst bei Bedarf benutzt er die obigen verfügbaren Querverweise.
Eine weitere Möglichkeit ist das „Learning by Doing", bei dem der Teilnehmer mit den Übungsaufgaben oder selbstständigem Programmieren beginnt und bei Bedarf auf ein Beispiel in den Quelltexten oder auf den theoretischen Hintergrund in den Vorlesungstexten zurückgreift.

3 Bausteine des Computergraphik-Kurses

Ohne Automatisierung ist die Erstellung der hypertextuellen Struktur und der gegenseitigen Verknüpfung zwischen Theorieteil, Experiment, Programmierschnittstelle und Quelltext nicht denkbar. Da die Vorlesungstexte als LaTeX-Quellen vorlagen, wurden sie mithilfe mehrerer Perl-Skripte hypertextuell überarbeitet - insbesonders dem Medium entsprechend in einzelne Textbausteine gegliedert, strukturiert, obige Verweise eingefügt und Ankerstellen für Verweise auf die einzelnen Kapitel eingefügt. Diese und alle weiteren **Textbausteine** und Daten werden zentral in einer **Datenbank** gehalten und über eine Java-Applikation verwaltet, die auch die konkrete Seitenhierachie und Lesepfade gestaltet. Das endgültige Design der Seiten wird anhand von Schablonen festgelegt; sollen beispielsweise alle Verweise auf Experimente neu durch ein Piktogramm gestaltet werden, so genügt eine kleine Veränderung an dieser Schablone.
Ein solches Kernstück eines **Autorensystems** konnte mit den GUI-Elementen des Java-Pakets *Swing* [4] unkompliziert und innerhalb weniger Mannwochen entwickelt werden: entfächerbare Baumstrukturen visualisieren die Seitenhierarchie, der Seiteninhalt ist als Quelltext, als endgültiger Hypertext oder in einem frei definierbaren Designmodus modifizierbar und die in die Seite eingebetteten Daten werden graphisch aufgelistet (Abb. 2). Der letztgenannte Designmodus stellt den Seiteninhalt in einer benutzerfreundlichen Weise dar, indem HTML-Schlüsselworte und selbstdefinierte Schlüsselworte für Schablonen oder einzufügende Datenbank-Inhalte symbolisch

dargestellt werden. Aufgrund des verwendeten Java-Pakets *JDBC*, das eine standardisierte Interaktion über SQL mit einer relationalen Datenbank bereitstellt, bleibt die zu Grunde liegende Datenbank austauschbar.

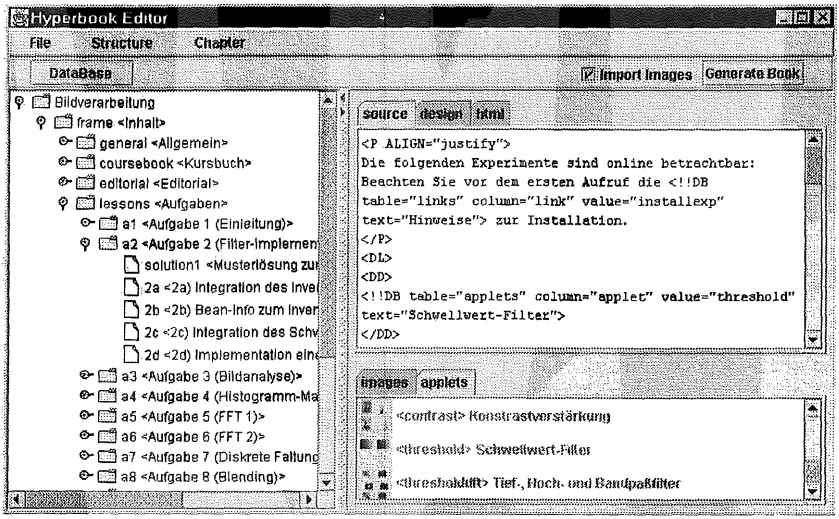


Abbildung 2: Java-Applikation zur Generierung der Hypertexte mit editierbarer Seitenhierarchie (links), Seiteninhalt (oben) und eingebetteten Daten (unten); mit nur einem Knopfdruck wird die gesamte Kursumgebung automatisch generiert.

Eine rein objektorientierte Programmierung bleibt zeitaufwendig und erfordert Programmierfahrung und Kenntnisse der Programmierschnittstelle. Deshalb wurde die Implementierung von **wiederverwendbaren Komponenten** (Java-Beans, [5]) mit einer standardisierten Schnittstelle begonnen, die sich visuell in einem Buildertool zu neuen Applikationen zusammensetzen lassen und die je nach Funktion eine sichtbare Benutzerschnittstelle besitzen oder rein logische Aufgaben erfüllen.

Hier ein Überblick über die wichtigsten Komponenten:

GUI-Komponenten, die die Java-Pakete AWT und Swing erweitern, wie beispielsweise ein Eingabefeld für eine Fließkommazahl, deren Zahl der Nachkommastellen frei definierbar ist, und das keine Rundungsfehler, wie sie üblicherweise bei den Standardtypen (float, double) in Java auftreten, aufweist. Bei der Entwicklung der Komponenten wurde besonders auf eine intuitive Eingabe und ein einheitliches Design geachtet; so besitzt das genannte Eingabefeld zwei Knöpfe, die bei gepreßter Maustaste die aktuelle Zahl langsam inkrementiert bzw. dekrementiert (in Abb. 3 ersichtlich) und falls zusätzlich der Parameterbereich eingeschränkt wird,

erscheinen im Layout automatisch die anwählbaren Grenzen. Ein weiteres Beispiel, das auf diesem Eingabefeld aufbaut, ist eine Matrix beliebiger Dimension, das sich wahlweise, um das Layout nicht zu überfrachten, in einem eigenen Fenster öffnen läßt.

Mathematische und geometrische Komponenten wie Vektoren, Matrizen, Dreiecke, usf. Diese Objekte besitzen eine rein logische Funktion, das heißt, sie sind für den Betrachter nicht sichtbar und müssen, falls sie im Layout erscheinen sollen, zur Darstellung eine der GUI-Komponenten verwenden.

2D- und 3D-Zeichenflächen, die auf den Java GUI-Komponenten aufbauen, aber zusätzliche Eigenschaften besitzen; beispielsweise werden Standard-Aktionen für Tastatur- und Mauseingaben definiert.

2D- und 3D-Szenengraph für hierarchische Szenenbeschreibungen, wie sie für komplexe und interaktive graphische Szenen benötigt werden. Neben den üblichen Komponenten wie Gruppenknoten, Objekte und Attribute finden sich hier auch spezielle Knoten, deren Kinder jeweils einen Einzelschritt eines graphischen Algorithmus erläutern und die somit mit einer weiteren Komponente, die genau ein Kind dieses Knotens auswählt und darstellt, eine in beliebige Richtung animierbare Visualisierung des Algorithmus realisiert.

Komponenten, die ein einheitliches **Design** bereitstellen, beispielsweise Gruppierungselemente für die obigen GUI-Komponenten, die einen betitelten Rahmen um das Objekt zeichnen oder verschiedene Kontexte durch einen zusätzlichen Leerraum verdeutlichen.

High-Level-Komponenten, die aus den genannten Basiskomponenten zusammengesetzt wurden und die Aufgaben, die mehrmals in ähnlicher Weise wiederkehren, übernehmen, beispielsweise ein Funktionsparser, editierbare Kurven, vordefinierte Graphikobjekte, usf.

Da die gewählte Programmiersprache **Java** einer ständigen Weiterentwicklung unterliegt, befindet sich der Entwurf dieser Komponenten mittlerweile in der dritten Generation. Der Kern der Objekte wurde ursprünglich objekt-orientiert in *C++* programmiert und nach *Java 1.0* portiert, wobei die Konvertierung geradlinig und problemlos ablief. Der Schritt von *Java 1.0* auf *Java 1.1* war mit einer Neudefinition des Event-Modells und des Umstiegs auf das Komponenten-Modell verbunden, und mit den zahlreichen Erweiterungen, wie sie mit der aktuellen Version *Java 2* verbunden sind, werden einige der entwickelten Komponenten wiederum obsolet. Insbesonders die instituts-eigenen 3D-Pakete und einige der mathematischen und geometrischen Komponenten gewinnen durch die Standard-Erweiterung *Java3D* an Effizienz bzw. können zum Teil durch deren Elemente ersetzt werden. Zur Zeit hat dies allerdings noch zwei Nachteile: Das Paket *Java3D* wird im Gegensatz zu den bisherigen Standard-Java-

Paketen nicht im Quelltext verteilt und da es zu seinem Einsatz *Java 2* voraussetzt, muß der Betrachter ein Browser-Plug-in installieren und mehrere Jar-Pakete des *Java3D* in Systemverzeichnisse kopieren. Erfahrungsgemäß stellt eine solche zusätzliche Installation bei vielen Anwendern ein Hindernis dar. Aus diesem Grund werden momentan noch zwei Versionen der 3D-Komponenten gepflegt.

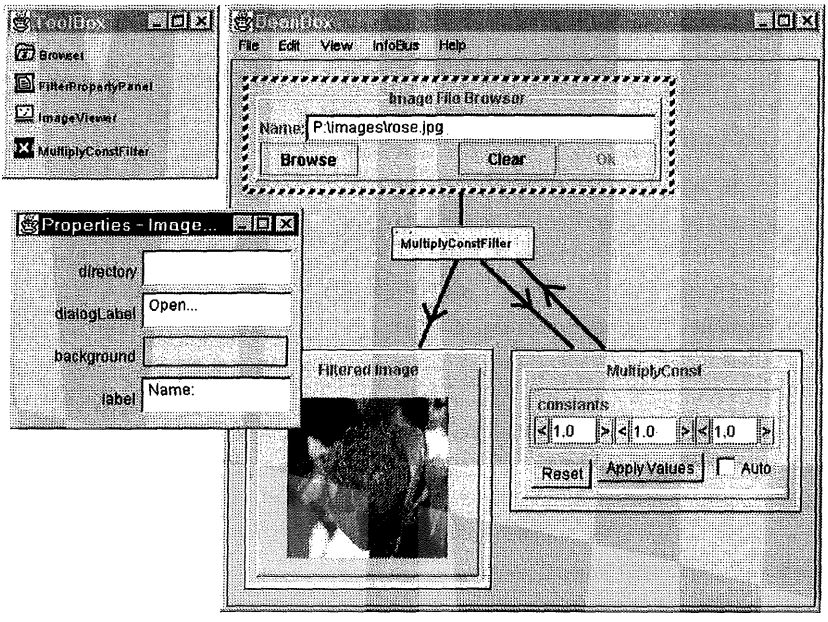


Abbildung 3: Ein visuelles Buildertool zur Verknüpfung von Basiskomponenten; hier wird ein virtuelles Experiment zur Modifikation der Farbkanäle eines Bildes erzeugt.

Wie schon die Text- und Datenmodule werden die vorhandenen Komponenten visuell in einem Buildertool zu neuen Komponenten oder virtuellen Experimenten zusammensetzt. Es existieren bereits eine Vielzahl solcher visueller Buildertools für *JavaBeans*, doch die von Sun mitgelieferte **BeanBox** hat ihnen etwas voraus: sie ist kostenlos und im Quelltext verfügbar, zudem läuft sie mit allen aktuellen Java-Versionen. Prinzipiell enthält ein solches Buildertool eine Palette aus Basiskomponenten, ein Designfenster, in dem diese Komponenten miteinander verknüpft werden, ein Layoutfenster, in dem die Lage und relative Größe der Komponenten festgelegt wird, und ein Dialogfenster, in dem die Parameter der Komponenten definiert werden. Aus den verknüpften Komponenten kann auf Wunsch eine vollständiges Applet oder eine neue Komponente generiert werden. In Abb. 3

wurde ein einfaches virtuelles Experiment dadurch erzeugt, daß ein Bildlader, ein Bildfilter und ein Bildbetrachter sequentiell aneinandergehängt wurden. Da der Bildfilter als rein logische Komponente selbst keine graphische Oberfläche besitzt, wurde er mit einer Komponente zur Darstellung von allgemeinen Bildfiltern gekoppelt. Zu beachten ist hier, daß die BeanBox Design- und Layoutfenster in einem Fenster vereinigt und die Verknüpfungslinien leider nicht dargestellt, weshalb sie in der Abbildung von Hand eingetragen wurden (eine leichte Modifikation am Quelltext der BeanBox behebt allerdings diese Schwäche).

4 Förderung des strukturellen Denkens

Die visuelle Programmierung eröffnet den Weg für eine **struktur-betonte Lernform**: Übungsaufgaben können nun als visuelle Datenfluß-Aufgaben gestellt werden, ohne daß der Teilnehmer zeitintensive Vorarbeiten zu leisten hat; und der eigentlichen Low-Level-Programmierung, die in der Computergraphik auch eingeübt werden soll, kann zunächst eine strukturelle Übersicht über die wesentlichen Begriffe und den Beziehungen zwischen ihnen vorangestellt werden [6, 7].

Daß ein Verständnis über die implizite Struktur eines Themas auf visuellem Wege schneller erreichbar ist als auf textliche Weise, soll an folgendem Beispiel verdeutlicht werden. Im vorigen Abschnitt (Abb. 3) wurde der Aufbau eines einfachen virtuellen Experiments beschrieben; das schlußendliche Ergebnis ist in Abb. 4 rechts dargestellt, gemeinsam mit dem bestmöglichsten Quelltext, der zum selben Ziel führt. Zwar ist der Quelltext aufgrund der komponenten-basierten Programmierung kurz und übersichtlich, doch muß der Teilnehmer bei der textlichen Darstellung mehr Zeit investieren, bis er sich eine ähnliche Struktur wie in der visuellen Programmierung erarbeitet hat. Außerdem stößt er vermutlich an mehreren Stellen im Quelltext auf ihm unbekannte Ausdrücke, die ihn vom eigentlichen Lernziel ablenken.

Allerdings bleibt zu bemerken, daß eine rein visuelle Darstellung erfahrungsgemäß nur oberflächlich aufgenommen wird. Der Teilnehmer fühlt sich imstande, die Aufgabe zu lösen, jedoch fehlen ihm die Begrifflichkeiten und die notwendige Programmierpraxis, die Lösung real durchzuführen. Deshalb wurde im vorliegenden Kurs auf die Konfrontation mit dem Quelltext und auf die selbstständige Programmierung nicht verzichtet.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import grdev.gui.*;
import grdev.filter.*;

public class MainClass extends JApplet {
    private ImageViewer imgViewer = new ImageViewer();
    private ImageBrowser imgBrowser = new ImageBrowser();
    private MultiplyConstFilter filter = new MultiplyConstFilter();
    private FilterPropertyPanel fpanel = new FilterPropertyPanel();

    public void init() {
        imgBrowser.addPropertyChangeListener(filter);
        filter.addPropertyChangeListener(imgViewer);

        filter.addFilterListener(fpanel);
        fpanel.addPropertyChangeListener(filter);

        getContentPane().setLayout(new BorderLayout());
        getContentPane().add("North", imgBrowser);
        getContentPane().add("Center", imgViewer);
        getContentPane().add("South", filterPanel);
    }
}
```

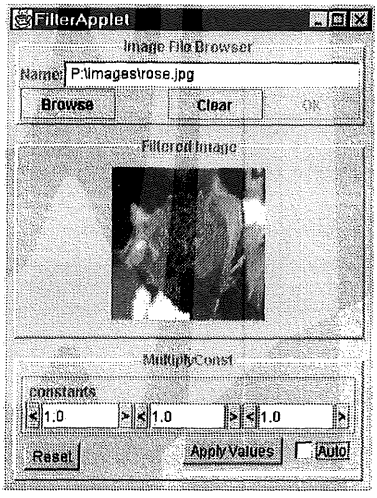


Abbildung 4: Ein einfaches virtuelles Experiment mit zugehörigem Quelltext.

Beim Entwurf virtueller Experimente stellt sich die wichtigste Frage im Kontext der Visualisierung: *Wie wird ein graphischer Algorithmus visualisiert?* Die Programmierung eines Themas bedingt nicht seine gelungene Umsetzung in ein virtuelles Experiment. **Genetische Prozesse** werden in der Computergraphik entweder iterativ oder rekursiv beschrieben und wurden in den Experimenten des Kurses in der Regel dadurch visualisert, daß dem Betrachter die Möglichkeit gegeben wurde, die Einzelschritte eines Algorithmus sequentiell mit frei wählbaren Startpunkten zu betrachten. Falls hilfreich, wurden zudem mehrere Folgeschritte parallel dargestellt. Räumliche **Bewegungen** wurden auch als solche visualisiert, beispielsweise wirkte sich beim Thema *Transformationen*, bei dem die Auswirkung einer frei definierbaren affinen Abbildung auf ein geometrisches Objekt dargestellt wurde, eine räumliche Animation vom ursprünglichen zum Bildobjekt positiv auf das Verständnis aus. Hier zeigt sich, daß in eine Visualisierung auch Elemente aufgenommen werden können, die nicht Teil des eigentlichen Algorithmus sind.

Hinsichtlich der **Interaktion** wurde bei den implementierten Experimenten darauf geachtet, daß alle wichtigen Parameter eines Algorithmus frei modifizierbar sind, und zwar nach Möglichkeit direkt am Ort ihrer visuellen Darstellung. Um den Betrachter nicht abzulenken, sollte die Art der Interaktion so intuitiv wie nur möglich und über die gesamte Lernumgebung hinweg konsistent sein. Doppelte Mausklicks wurden dem aktuellen Trend entsprechend gemieden.

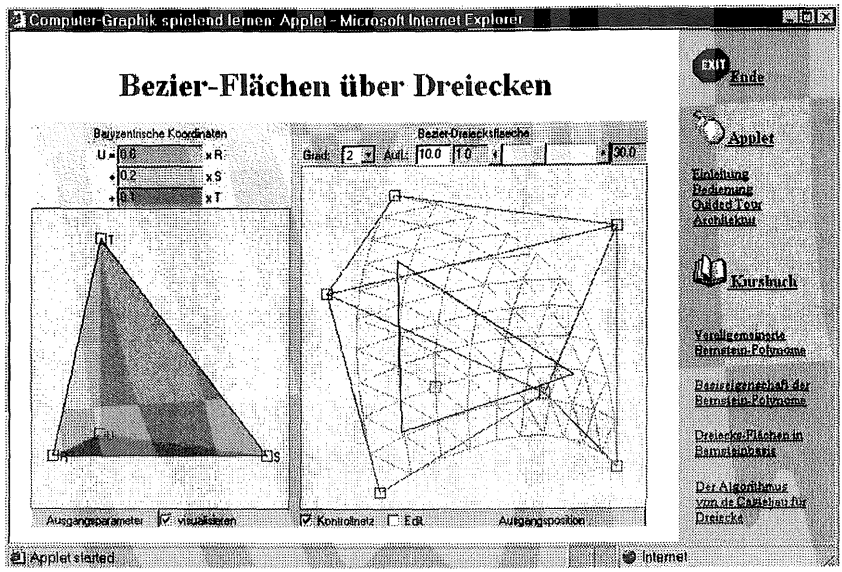


Abbildung 5: Visualisung eines graphischen Algorithmus für Bézier-Flächen.

Zur Visualisierung eines graphischen Algorithmus genügen allerdings diese Merkregeln nicht, sondern das dazu notwendige Verständnis über die Zusammenhänge muß mit visualisiert werden, wie Abb. 5 exemplarisch verdeutlicht: Derartige *Bézier-Flächen* können mit Hilfe des Algorithmus von *de Casteljau* iterativ konstruiert werden. Implizit fließt dabei die Idee der *baryzentrischen Koordinaten* ein, die deshalb mit visualisiert wird. Dem Betrachter wird die Möglichkeit geben, alle wichtigen Parameter (Kontrollnetz, baryzentrische Koordinaten, Grad der Fläche) direkt zu modifizieren. Alle Iterationen des Algorithmus werden parallel visualisiert und bei jeder Modifikation aktualisiert. Mit dem Einsatz von Farbe, Titeln, Gruppierung und Umrandung werden die inhaltlichen Zusammenhänge verdeutlicht.

5 Modifikation des Kurses

An der *Universität Tübingen* wird die Lernumgebung zur Vertiefung der Vorlesung angeboten und als Grundlage für die Übungen zur Vorlesung verwendet. Der zeitliche Umfang (jeweils zwei WS Vorlesung und Praxis) und die Voraussetzungen der Studenten (Grundvorlesungen der Informatik, dabei nur wenig Programmiererfahrung) schränkt den zeitlichen Rahmen für die **Teilnehmer** zur Bearbeitung der naturgemäß ein. Zwar kann die Vervollständigung noch nicht vorhandener oder nur rudimentär entwickelter Komponenten durchaus als Übungsaufgabe gestellt werden. Trotzdem liegt

der Schwerpunkt in den Übungen darauf, Basis-Algorithmen innerhalb eines vorgegebenen Rahmens zu implementieren. Erfreulich ist deshalb um so mehr, daß wiederholt freiwillige Ergänzungen und neue Applikationen beigesteuert wurden, die nicht Teil der Aufgabenstellung waren. Thematisch an die Vorlesung anschließende Studien- und Diplomarbeiten flossen ebenfalls gewinnbringend ein.

Den Teilnehmern wie auch dem Dozenten fallen Modifikationen am Quelltext der Komponenten und der virtuellen Experimente hauptsächlich deshalb schwer, da zur Low-Level-Programmierung ein konkretes Vorwissen über den Aufbau der Komponenten, über die Programmiersprache und nicht zuletzt über die Programmierumgebung vonnöten ist. Der Ansatz der visuellen Programmierung ermöglicht es ihnen nun, kurzfristig und ohne dieses Detailwissen an den vorhandenen Experimenten visuell einzelne Parameter zu modifizieren oder aus den implementierten Basiskomponenten neue Experimente zu kreieren.

Für den **Programmierer** bedeutet die Programmierung von Komponenten zunächst einen Mehraufwand, da die dazu notwendige Funktionalität nach wie vor low-level implementiert wird. Zusätzliche Informationen (*BeanInfo*) sind zu spezifizieren und benutzerdefinierten Objekten, deren Parametern visuell modifizierbar sein sollen, ist ein dementsprechender Editor mitzugeben. Die Benutzerschnittstelle, also die sichtbaren Parameter und Methoden des Objektes und ihre Bezeichung, kann zwar automatisch bestimmt werden (*Introspection*), ist jedoch für gewöhnlich anzupassen.

Diese Mehrarbeit macht sich freilich sofort wieder bezahlt: durch die Verknüpfung vorhandener Komponenten entstehen rasch neue Komponenten und durch die komponenten-basierte Programmierung entsteht automatisch eine übersichtliche Objektstruktur mit austauschbaren Elementen. Beispielsweise entstand aus einem Datei-Browser, einem Bildlader und einem Bildbetrachter eine neue Komponente. Weiter ließe sich in einfachster Weise ein beliebiger Bildfilter dazwischenlegen. Wird von einer anderen Arbeitsgruppe ein leistungsfähigerer Browser entwickelt, kann er ohne weitere Modifikationen einfach substituiert werden. Hierbei sei bemerkt, daß im World-Wide-Web mittlerweile Komponenten zu allen Arten von Fachbereichen veröffentlicht werden, die in eigene Projekte integriert werden können [8].

Selbst bei Komponenten mit Fähigkeiten, die keine der vorhandenen Komponenten implementieren und die nach wie vor low-level programmiert werden, können vorhandene Komponenten für Teilaufgaben genutzt werden. Auf diese Weise wurden über 70 Experimente erstellt, die das gesamte Gebiet der Computergraphik abdecken.

6 Adaption an neue Inhalte

Auf die Veröffentlichung der Standard-Erweiterung *Java Advanced Imaging (JAI, [9])* konnte durch diesen komponenten-basierten Ansatz schnell reagiert und in wenigen Wochen ein **vollständiger neuer Kurs** für die Bildverarbeitung [10] erstellt werden, der sich in die bestehenden Java-Komponenten einfügt.

Die **hypertextuelle Kursumgebung** wurde mit Hilfe des beschriebenen Authoring-Tools entworfen und aufgebaut, wobei das Seitendesign, das durch die Schablonen festgelegt wird, leicht modifiziert wurde; insbesonders wurden Farben und Piktogramme neu definiert. Vereinzelt konnten Textbausteine der Rahmenseiten übernommen werden, in der Regel wurden die Inhalte jedoch neu verfaßt. Nachdem die Seitenhierarchie definiert war, wurde die eigentliche Vorlesung, die aus Powerpoint-Folien bestand, nach Hypertext konvertiert und in den Kurs integriert.

Der Kurs umfaßt hinsichtlich der **virtuellen Experimente** den folgenden Inhalt: Nach einer Einleitung in die Programmierung von *Java 2* und die Verwendung von *JavaBeans* werden zunächst die Grundtechniken der verwendeten Filterklassen vorgestellt. Danach wird die Implementierung eines JAI-Filters und eines eigenen Filters besprochen und Bildanalyse betrieben, indem relative und kumulative Histogramme von Bildern betrachtet und eine Kontrastverstärkung implementiert wird. Fortgeschrittene Techniken zur Histogramm-Manipulation und die Verwendung einer Lookup-Table zur Histogramm-Einebnung oder Falschfarben-Darstellung schließen sich an. Daraufhin wird die Fourier-Transformation und ihre Anwendung in der Bildverarbeitung vorgestellt (Abb. 6) und die diskrete Faltung mit einem Filterkern näher erläutert. Das Blending zweier Bilder wird vorgestellt, verschiedene Verfahren zur Bildkorrektur werden implementiert und verwackelte Bilder rekonstruiert. Neben Bildkodierung und Dekodierung wird letztlich die Kantendetektion, affine Bildtransformationen und polynomielles Warping erläutert.

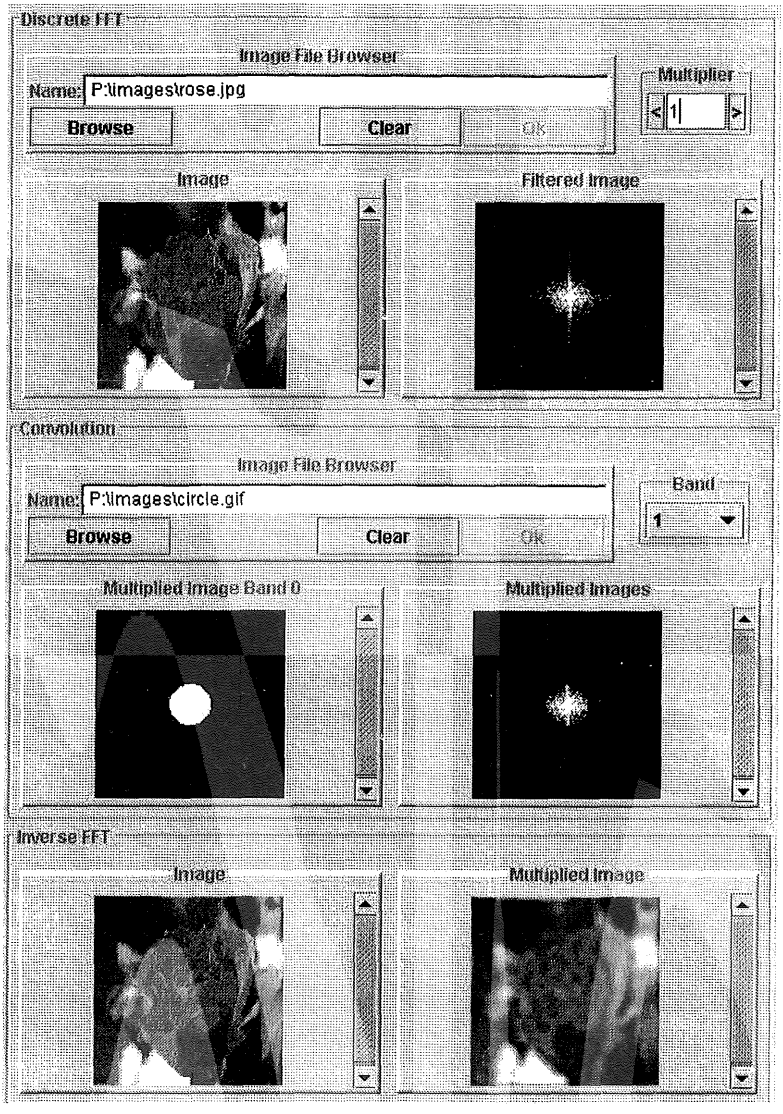


Abbildung 6: Anwendung der diskreten Fouriertransformation als Tiefpaßfilter.

Mit der bisherigen Funktionalität von Java war Bildverarbeitung in diesem Umfang komplex und ineffizient. Das Paket *JAI* begegnet diesen Nachteilen, indem es ein einfaches, flexibles und erweiterbares Programmier-Modell zur Bildverarbeitung spezifiziert und eine Vielzahl von effizienten Bildoperatoren zur Verfügung stellt.

Bei der Integration in die bereits vorhandenen Komponenten waren die folgenden Erweiterungen notwendig:

Alle JAI-Operatoren wurden durch Wrapper-Klassen in **Filterkomponenten**, d.h. *JavaBeans*, mit einem einheitlichen Datenfluß-Konzept gewandelt.

Bei Bedarf wurden mehrere JAI-Operatoren zu einer **Filterkette** verknüpft und als neue Filterkomponente implementiert, wie beispielsweise bei der Normalisierung einer Farbverteilung eines Bildes, bei dem zunächst die Extremwerte bestimmt werden und daraufhin der Farbbereich auf den maximalen Bereich gestreckt wird.

Es wurden **GUI-Komponenten** zur Darstellung von Bildern und Histogrammen implementiert, wie auch eine Filter-Oberflächenkomponente, die für eine beliebige Filterkomponente ein Layout automatisch generiert. Dazu mußten für komplexere Filterparameter die entsprechenden Editoren entworfen werden. Bildlade- und Speicherkomponenten konnten durch Rückgriff auf den Dateibrowser ohne Aufwand bereitgestellt werden.

In JAI **unbehandelte Operationen** wurden selbst programmiert, wodurch die Entwicklung eigener Bildoperationen demonstriert wird, wie beispielsweise eine Rotationsfiltermaske oder die Bildkodierung und Dekodierung in einem benutzereigenen Format.

Das Paket *JAI* erfordert die Verwendung von *Java 2*, weshalb für die virtuellen Experimente das zugehörige Browser-Plug-in installiert werden muß. Außerdem müssen dazu in der aktuellen Version von JAI mehrere Jar-Pakete von JAI in ein Systemverzeichnis kopiert werden und die Java-Sicherheitsregeln *(Java-Policy)* modifiziert werden. Diese Voraussetzungen erzeugten bei den Teilnehmern des Kurses eine gewisse Hemmschwelle, es ist allerdings zu erwarten, daß dieses Hindernis mit einer zukünftigen Browser-Generation oder einer Weiterentwicklung des Java-Plug-ins behoben wird.

Der Kurs wurde im SS 1999 im Rahmen der Lehrveranstaltung „Bildverarbeitung I" an der *Universität Tübingen* eingesetzt und durch das *Deutsche Institut für Fernstudienforschung (DIFF)* evaluiert. Da die Auswertung noch nicht abgeschlossen ist, läßt sich hier nur sagen, daß sich die Teilnehmer im Vergleich zu früheren Veranstaltungen aktiver mit dem Quelltext auseinandersetzten und positiv auf die Integration neuer Java-Entwicklungen in die Lehre reagierten.

7 Zusammenfassung

Es wurde demonstriert, wie ein komponenten-basiertes Konzept einer objekt-orientierten Lernumgebung neue strukturelle Fähigkeiten und Flexibilität verleiht. Auf der Basis eines vorhandenen Web-Kurses zur Computergraphik konnte mit wenig

Aufwand auf aktuelle Trends in der Java-Entwicklung eingegangen und ein neuer Kurs zur Bildverarbeitung ausgearbeitet werden.

8 Danksagung

Wir danken G. Rößner, R. Schwering und H. Müller für ihren unermüdlichen Einsatz bei der Realisierung der beschriebenen Programmpakete.

Literatur

[1] G. S. Owen. *Teaching Computer Graphics as an Experimental Science*. In Eurographics Workshop on Graphics and Visualization Education (GVE), Oslo, Norway, 10-11 September. Eurographics, 1994

[2] Web-Kurs *Computergraphik spielend lernen.* URL: http://www.gris.uni-tuebingen.de/gris/grdev/java, Universität Tübingen, Graphisch-Interaktive Systeme (WSI/GRIS), 1998.

[3] R. Klein, F. Hanisch. *Web-basierter Unterricht in der Computergraphik: Konzepte und Realisierung von interaktiven Online-Kursen.* In: Volker Claus. Informatik und Ausbildung GI-Fachtagung Stuttgart, Springer-Verlag, 1998.

[4] Graham Hamilton. *The JavaBeans API Specification.* Sun Microsystems, Inc., Juli 1997.

[5] Robert Eckstein, Marc Loy und Dave Wood. *Java Swing.* O'Reilly & Associates, Inc., 1998.

[6] Avi C. Naiman. *Interactive teaching modules for computer graphics.* j-COMP-GRAPHICS, 30(3):33-35, August 1996.

[7] E. Wernert. *A unified environment for presenting, developing and analyzing graphics algorithms* Computer Graphics, 31(3):26-28, August 1997.

[8] Sun. *The JavaBeans Directory.* URL: http://beans.cuesta.com, Sun Microsystems, Inc., 1998

[9] Sun. *Java Advanced Imaging API White Paper (Beta),* Sun Microsystems, Inc., April 1999.

[10] Web-Kurs *Bildverarbeitung.* URL: http://www.gris.uni-tuebingen.de/study/bv/ss99, Universität Tübingen, Graphisch-Interaktive Systeme (WSI/GRIS), April 1999.