SVG

Scalable Vector Graphics

Gestaltung einer interaktiven Karte

Studienarbeit ALEXANDER BERGER

Technische Universität Dresden

Institut für Kartographie

Dresden, 3. Juli 2002

Technische Universität Dresden
Institut für Kartographie

Dresden, 30.Januar 2002

Aufgabenstellung zur Studienarbeit

für Herrn ALEXANDER BERGER

Thema: Entwurf und Gestaltung einer interaktiven Karte im Format SVG (Scalable Vector Graphics)

Aufbauend auf der Studienarbeit von VOIGT, in der wesentliche Eigenschaften von SVG zusammengestellt wurden, sollen in dieser Studienarbeit die Möglichkeit der Einbindung von Interaktivität in eine Karte des SVG – Formates aufgezeigt werden. Dabei sind sowohl die erforderlichen SVG – Parameter als auch ein effektiver Weg zur Herstellung in Form eines Studienhilfsmittels zu dokumentieren.

Am Beispiel der Karte des Campus der Technischen Universität Dresden sind diese Untersuchungen durchzuführen. Die Karte soll die Homepage des Instituts für Kartographie erweitern und u.a. Auskunft zu Gebäuden, Bibliotheken und Cafeterias geben.

Die Ergebnisse der Studienarbeit sind kritisch zu werten.

Prof. Dr. Ingeborg Wilfert

Inhaltsverzeichnis

1 Einleitung

1.1 Interaktivität in Internetkarten

Mit der rasanten Entwicklung des **W**orld **W**ide **W**eb (WWW) ist die Zahl der Internetseiten, die Karten als Informationsmedium nutzen, ebenso rasant gestiegen. An die 40 Millionen Karten werden nach Schätzungen an jedem Tag von Internetnutzern weltweit aufgerufen [DICKMANN, 2001]. Dabei handelt es sich nicht mehr ausschließlich um rein statische Karten, also Karten deren Originale gescannt wurden, um sie als Rastergraphik am Bildschirm zu visualisieren. Allerdings bildet diese Visualisierungsform derzeit den größten Anteil an Internetkarten.

Die technischen Möglichkeiten des Internets sind mittlerweile so vielfältig, dass auch interaktive Karten dargestellt werden können. Unter Interaktivität versteht man dabei die Möglichkeit des Nutzers, durch Eingaben, sei es über die Tastatur oder mit Hilfe der Maus, mit dem Browser zu kommunizieren, sodass im weitesten Sinne eine Veränderung der Kartendarstellung erfolgt. Bei diesen interaktiven Karten kann es sich um Rastergraphiken handeln, die durch HTML (**H**yper**T**ext **M**arkup **L**anguage) – Elemente oder durch Skript-sprachen, wie JavaScript, manipuliert werden („imagemaps“, „clickable maps“). Die Manipulation kann sich beispielsweise auf das Einblenden weiterer Informationsebenen oder zusätzlicher Fenster beziehen, wodurch man allerdings schnell an die Grenzen der Möglichkeiten an Interaktionen stößt. Des weiteren können es aber auch interaktive Karten sein, die der Nutzer individuell mit Hilfe von Datenbankabfragen selbst generiert („maps on demand“) oder aufwendig gestaltete Web – GIS mit einer Vielzahl von Funktionen. Eine genaue Beschreibung aller derzeit aktuellen Techniken kann in der Veröffentlichung von F. Dickmann [DICKMANN, 2001] nachgelesen werden.

Allen gemein ist diesen gebräuchlichsten Formen der interaktiven Karte die Nutzung von Rastergraphiken, wodurch sie oft zu groß, in der Qualität der Wiedergabe zu gering, zu kompliziert zu erstellen und nur beschränkt interaktiv sind. Eine Ausnahme bilden Karten im Flash – Format der Firma Macromedia, das wohl als das am weitesten verbreitetste Vektorformat zählt. Zwar können mit Macromedia Flash ein hoher Grad an Interaktivität erreicht und einige der oben genannten Probleme gelöst werden, doch lieferte dieses Format gleich neue Probleme hinzu, da es zum einen kein offizieller W3C (**W**orld **W**ide **W**eb **C**onsortium) – Standard ist und zum anderen in binärer Form vorliegt und somit nicht außerhalb der Entwicklungsumgebung editiert werden kann [SCHNABEL, 2002].

Mit der Veröffentlichung der SVG (**S**calable **V**ector **G**raphics) – Spezifikation Version 1.1 am 30.April 2002 steht nun allerdings schon die zweite Version des Vektorformates zur Verfügung, welches vom W3C als offizieller Standard festgelegt wurde. SVG besitzt dabei nicht nur ähnliche Funktionalitäten wie das Format Flash, sondern übertrifft diese sogar und kann somit

als Alternative für interaktive Karten genutzt werden. Einen ausführlichen Vergleich findet man in der Arbeit von O. Schnabel [SCHNABEL, 2002].

1.2 Warum SVG?

Da diese Arbeit auf die Studienarbeit von Y. Voigt [VOIGT, 2001] aufbaut und auch weitere Veröffentlichungen existieren, welche die Eigenschaften von SVG abhandeln, zu erwähnen sei hier die Diplomarbeit von A. Winter [WINTER, 2000], wird in diesem Abschnitt auf eine ausführliche Beschreibung verzichtet. Vielmehr sollen kurz die Vorteile des Vektor – Formates SVG gegenüber anderen Formaten erörtert werden, wobei auf einige Punkte im späteren Verlauf der Arbeit noch einmal näher eingegangen wird.
Bei SVG handelt es sich um eine Sprache zur Beschreibung zweidimensionaler Graphiken in XML (e**X**tensible **M**arkup **L**anguage). Genau genommen ist es keine eigene Sprache, sondern ein „Dialekt“ von XML, eine XML – Applikation. Der Name SVG steht dabei übersetzt für skalierbare Vektor – Graphik und enthält daher schon drei Vorteile in sich:
Als erstes wäre der Vorteil der **Skalierbar**keit zu nennen, wodurch die Graphik in jeder beliebigen Größe dargestellt werden kann und Ein- und Auszoomen ohne Qualitätsverlust möglich ist. Diese Möglichkeit beruht auf den zweiten entscheidenden Vorteil, nämlich dem **Vektor** – Format von SVG, bei dem die einzelnen Objekte mathematisch beschrieben werden. Dieser Vorteil liefert allerdings nicht nur die Skalierbarkeit, sondern auch eine geringe Dateigröße als Ergebnis, die bekanntermaßen von großer Bedeutung bei der Einarbeitung von Graphiken in Internetseiten ist. Der dritte Vorteil verbirgt sich im Wort **Graphik**. Wird diese mit Hilfe von XML beschrieben, ist der Inhalt der Datei voll zugänglich und kann mit jedem einfachen Texteditor bearbeitet werden, da es sich bei XML um eine textbasierte Sprache handelt.
In der Abstammung von XML verbergen sich weitere Vorteile des SVG – Formates, die vor allem für Interaktionen von Bedeutung sind. XML – Dokumente sind hochstrukturierte Dokumente mit einer eindeutigen Objekthierarchie. Diese Objekthierarchie entspricht den Bestimmungen des DOM (**D**ocument **O**bject **M**odel), welches ebenfalls festlegt, in welcher Weise auf ein Dokument zugegriffen bzw. ein Objekt verändert werden kann (siehe Kapitel 2.3). In SVG ist das DOM ebenfalls implementiert, allerdings in der Version 2.0. Somit ist es möglich mit Hilfe einer Scriptsprache jedes einzelne Element innerhalb der SVG-Graphik anzusprechen, zu klonen, zu entfernen oder Eigenschaften des Elementes zu ändern, um nur einige Möglichkeiten zu nennen. Dieses Konzept ist der Schlüssel für die Dynamik in SVG – Graphiken bzw. in allen XML – Dokumenten [SALATHÉ, 2001]. Unter Dynamik wird also im weiteren Verlauf der Arbeit die Möglichkeit verstanden, den in einem Browser – Fenster angezeigten Inhalt, in diesem Fall die Internetkarte, nachträglich zu

manipulieren, sei es zum Beispiel die Veränderung einer Eigenschaft, wie die Farbe, oder eine Veränderung der Anzahl bestimmter Elemente innerhalb der Karte.
Abschließend soll noch ein letzter Vorteil genannt werden der ebenfalls auf die XML – Basiertheit zielt. SVG stellt eine Reihe von Elementen zur Erzeugung von Animationen zur Verfügung. Unter Animation versteht man hierbei die zeitliche Veränderung einer Graphik bzw. eines Objektes innerhalb der Graphik, dabei kann es sich zum Beispiel um fließende Bewegungen oder stufenlose Veränderungen von Objekten handeln [BOLLMANN; KOCH, 2001]. Diese Animationselemente basieren zum größten Teil auf Animationselemente der XML – Applikation SMIL (**S**ynchronized **M**ultimedia **I**ntegration **L**anguage), welche zur Entwicklung von multimedialen Präsentationen im WWW dient. Die Übertragung der Animationselemente auf SVG verdeutlicht außerdem die Kompatibilität der XML – Applikationen untereinander und kann als weiterer Vorteil gesehen werden.

1.3 Zielstellung

Ziel dieser Arbeit ist es, den Weg von einer statischen Karte, welche mit Hilfe von Macromedia Freehand erstellt wurde, zu einer interaktiven Internetkarte im SVG – Format zu beschreiben. Dabei werden im ersten Teil die Grundlagen für Interaktivitäten in SVG näher erläutert. Im zweiten Teil werden diese genutzt, um eine interaktive Karte des Campus der Technischen Universität Dresden als Prototyp zu erstellen. Allerdings sollen auch grundlegende Überlegungen zum Einsatz von Interaktivitäten aufgeführt werden.
Wert gelegt wurde bei der Erstellung des Prototyps auf eine möglichst klare Strukturierung und Formulierung der einzelnen Funktionen und auf eine gewisse Allgemeinhaltung im Sinne der Übertragbarkeit auf andere SVG – Internetkarten.

2 Interaktivitäten mit SVG

Im folgenden Kapitel sollen die Grundlagen bzw. Eigenschaften erörtert werden, die es ermöglichen, Interaktionen des Nutzers innerhalb der SVG – Karte zu erlauben. Zum besseren Verständnis werden dabei auch SVG – Codefragmente, also einzelne Quelltextabschnitte aufgeführt, die diese Eigenschaften beschreiben. An dieser Stelle sollen deshalb noch einige wichtige Festlegungen zur Ausdrucksweise innerhalb der Arbeit folgen. Die grundlegendste Einheit bei der Schreibweise von SVG ist das Element, welches in der Form von Tags immer in spitzen Klammern (<...>) geschrieben wird. Ein Element besteht immer aus einem Starttag und optional aus einem Endtag, je nachdem, ob ein Inhalt zum Element gehört. Ist dies der Fall, wird nach dem Inhalt (z.B. ein weiteres Element oder Text) ein Endtag mit einem Schrägstrich vor dem Elementname platziert, z.B. *<text> Inhalt </text>*. Ist dies nicht der Fall, wird der Schrägstrich vor die spitze Klammer am Ende des Starttags gesetzt, z.B. *<circle/>*. Innerhalb des Elementes können verschiedene Attribute aufgeführt werden, welche sich in einen Attributnamen und einen Attributwert gliedern. Der Attributwert kann wiederum aus einer Zahl oder aus einer wählbaren Eigenschaft bestehen. Konkret könnte dies wie folgt aussehen.

```
<text style="font-family:'Arial'">Inhalt des Elementes.</text>
```

Man spricht also vom Element „text“ mit einem Attribut namens „style“. Der Wert des Attributes ist „font-family“ (Schriftart), deren Eigenschaft wiederum „Arial“ ist. Zwischen Start- und Endtag befindet sich der Inhalt, also der Satz „Inhalt des Elementes.“.
Diese Grammatikregeln treffen im übrigen nicht nur auf SVG zu, sondern auch auf alle anderen XML – basierten Formate.

2.1 Animationen

Animationen bilden zwar nicht direkt eine Grundlage für Interaktivitäten, sie werden aber oft durch diese ausgelöst und sollen deshalb hier mit aufgeführt werden. SVG besitzt, wie schon im Kapitel 1.2 erwähnt, eine Vielzahl von Elementen und Attributen zur Festlegung von Animationen, an dieser Stelle alle zu erläutern würde allerdings zu weit führen. Aus diesem Grund wurden 2 Elemente ausgewählt, die für die Erstellung der interaktiven Karte von Bedeutung sind.
Das erste Element und wohl auch das am meisten genutzte, ist das Element *<animate>*. Mit ihm lassen sich Attributwerte und Eigenschaften anderer Elemente ändern, also z.B. die Breite eines Rechtecks oder die Größe eines Schriftzuges. Dazu müssen aber noch verschiedene Attribute hinzugefügt werden. Eine Auswahl dieser wären:

- das Attribut *xlink:href="#idname"* zur Referenzierung des Elementes, dessen Attribute oder Eigenschaften animiert werden sollen (entfällt, wenn Animationselement als Inhalt innerhalb des zu animierenden Elementes steht);
- das Attribut *attributeName="Name"* zur Referenzierung des Attributes, welches animiert werden soll;
- die Attribute *beginn="Zeit1", dur="Zeit2"* und *end="Zeit3"*, die angeben, wann eine Animation starten, wie lange sie dauern und wann sie enden soll (Standardzeiteinheit: Sekunde);
- das Attribut *repeatCount="Anzahl"* zur Angabe der Anzahl der Wiederholungen oder *repeatDur="Gesamtdauer"* zur Angabe der Gesamtdauer der Wiederholungen (wenn Attributwert auf „indefinite" gesetzt wird, läuft Animation bis zum Schließen des Dokumentes oder bis zum Ende der Animation durch Interaktion des Nutzers);
- das Attribut *fill="freeze"* oder *"remove"*, um festzulegen, ob das Ergebnis der Animation „eingefroren" (freeze) oder verschwinden (remove) soll;
- die Attribute *from="Wert1", to="Wert2"* und *by="Wert3"*, wenn die Animation einen Übergang von einem zum anderen Wert darstellen soll, wobei der Anfangswert um den Wert des Attributes „by" geändert wird;
- die Attribute *values="Wert1;Wert2;Wert3;..."*, zur Angabe mehrerer Übergangswerte und *keytimes="Zeitpunkt1; Zeitpunkt2; Zeitpunkt3;..."*, wenn diese Übergangswerte jeweils zu einem bestimmten Zeitpunkt erreicht werden sollen.

Die Vielzahl der möglichen Attribute zeigt, wie differenziert einzelne Animationen in SVG gestaltet werden können. Hinzu kommt dabei, dass jede Animation ihre eigene Zeitlinie besitzt, da kein zentrale Zeitlinie existiert [WINTER, 2000].
Ein konkretes Beispiel für eine Textanimation könnte also folgendermaßen aussehen.

```
<text id="info">Information</text>
<animate xlink:href="#info" attributeName="opacity" begin="3" dur="6"
keyTimes="0;0.4;0.6;1" values="0;1;1;0" repeatCount="2"/>
```

Bei diesem Beispiel wird das Attribut „opacity" (Deckfähigkeit) des Textes animiert, dessen „id" – Wert „info" lautet. Drei Sekunden nach dem Laden der Datei wird der Wert des Attributes „opacity" von 0 auf 1 (sichtbar) gesetzt, verweilt einen kurzen Moment und geht wieder von 1 auf 0 (unsichtbar) zurück. Die gesamte Animation dauert sechs Sekunden und wird zweimal wiederholt. Sie bewirkt ein zweimaliges aufblinken des Textes „Information".

Viel bedeutender für kartographische Anwendungen mit SVG ist die Möglichkeit, Bewegungen von Objekten entlang Pfaden darzustellen. Dies kann mit Hilfe des Animationselementes *<animateMotion>* realisiert werden. Wie schon im oberen Beispiel müssen diesem Element noch

verschiedene Attribute zugewiesen werden, diese entsprechen jedoch größtenteils den oben aufgeführten. Um eine Bewegung darstellen zu können, werden allerdings Punkte benötigt, die Anfangs- und Endpunkt der Bewegung markieren. Deshalb erhalten die Attribute *from="x1,y1"*, *to="x2,y2"* oder *value="x1,y1;x2,y2;x3,y3;..."* Koordinatenpaare als Attributwerte. Oft ist es aber notwendig, Bewegungen entlang längeren Pfade darzustellen. Dies kann erreicht werden, indem man in das Animationselement das Element *<mpath>* einfügt und darin mit Hilfe des Attributes „*xlink:href*" auf einen schon vorhanden Pfad verweist. Dem Element *<animateMotion>* kann außerdem noch

- das Attribut *rotate=„auto"* oder *„auto-reverse"* hinzugefügt werden, welches die Lage des zu bewegenden Objektes zum Pfad bestimmt. Ist der Attributwert auf „auto" gesetzt, erfolgt eine automatische Drehung des Objektes je nach Steigung des Pfades, an dem es entlang geführt wird. Der Wert „auto-reverse" führt zum gleichen Ergebnis, dreht das Objekt allerdings noch um 180°.

Wieder soll am Beispiel des Textes das Element *<animateMotion>* verdeutlicht werden.

```
<text id="info">Information</text>
<path id="Pfad" d="M-7324 118l3195 2229c0,0 134,73 247,59l292 –74"/>
<animateMotion xlink:href="#info" begin="0" dur="10" rotate="auto" repeatCount="5">
  <mpath xlink:href="#Pfad"/>
</animateMotion>
```

Diese Animation ergibt eine Bewegung des Textes mit dem „id" – Wert „info" entlang des Pfades mit dem „id" – Wert „Pfad". Die Animation beginnt null Sekunden nach dem Laden der Datei, eine vollständige Bewegung dauert zehn Sekunden und wird fünfmal wiederholt. Das Wort Information bewegt sich also innerhalb von zehn Sekunden vom Anfangspunkt zum Endpunkt des Pfades. Dabei dreht es sich automatisch je nach Krümmung des Pfades, da das Attribute „rotate" auf „auto" gesetzt wurde.

Mit dieser Erläuterung der beiden Elemente ist aber noch lange nicht der Vorrat an Möglichkeiten, Animationen mit SVG zu erstellen, ausgeschöpft. Weitere interessante Animationselemente wären z.B. das Element *<animateColor>* zur Farbänderung innerhalb eines bestimmten Zeitraumes sowie das Element *<animateTransform>*, welches animierte Verzerrungen, Rotationen oder Vergrößerungen erlaubt.

2.2 Scriptsprachen

Die erste wirkliche Grundlage für Interaktivitäten in SVG sind Scriptsprachen, denn sie bilden die Schnittstelle zwischen der Interaktion des Nutzers und dem Browser. Unter einer Scriptsprache versteht man eine objekt – orientierte Programmiersprache, die eingesetzt wird, um die Einrichtung eines existierenden Systems zu verändern, anzupassen und zu

automatisieren. Dabei läuft eine Scriptsprache immer in einer bestimmten Umgebung ab (in diesem Fall der Web – Browser) und verwendet die von dieser Umgebung bereitgestellten Funktionalitäten [SEEBOERGER-W., 2001]. SVG stellt die Möglichkeit zur Verfügung, Scripte mit Hilfe des einleitenden *<script>* und abschließenden *</script>* – Elementes direkt in den SVG – Quelltext einzubinden oder aus einer externen Scriptdatei auf des SVG – Dokument und dessen Elemente zuzugreifen.
Bleibt die Frage, mit welcher Scriptsprache man arbeiten soll, da erstens eine Vielzahl dieser vorhanden und zweitens die Kompatibilität und Anwendbarkeit sehr verworren ist. Eine Standardisierung existiert zwar mit dem ECMAScript (**E**uropean **C**omputer **M**anufacturers **A**ssociation), das 1998 als ISO – Standard vom gleichnamigen ECMA – Konsortium festgelegt wurde, doch sind andere Scriptsprachen nicht vollständig kompatibel dazu.
Im folgenden sollen drei Vertreter kurz vorgestellt werden: JavaScript, Jscript und VBScript, wobei vorneweg gesagt werden muss, dass JavaScript am weitesten verbreitet ist und auch in dieser Arbeit zur Anwendung kam.

2.2.1 JavaScript

JavaScript ist eine Gemeinschaftsentwicklung der Firmen Netscape und Sun und wurde 1995 mit der Herausgabe des Web – Browser „Netscape Navigator 2“ das erste Mal veröffentlicht. Es lehnt sich an die Programmiersprache Java an und ist plattformunabhängig. Ordentlich programmierte Scripte dürften also ohne Probleme im Netscape – Browser unter Windows, Linux oder Apple Macintosh laufen. JavaScript erfreut sich großer Beliebtheit unter den Programmierern von Web – Seiten und existiert mittlerweile in der Version 1.3.
Das große Dilemma beginnt allerdings, wenn man JavaScript von Netscape als Konkurrent der Microsoft – Produkte Jscript und VBScript betrachtet. Kann man im Netscape Navigator JavaScript je nach Version uneingeschränkt nutzen, so ist durchaus mit Problemen bei dem von Microsoft entwickelten Browser „Internet Explorer“ zu rechnen, da dieser nicht alle Eigenschaften und Methoden unterstützt.

2.2.2 Jscript

Jscript ist eine von Microsoft entwickelte Scriptsprache, die in den Internet Explorer integriert ist. Jscript enthält alle Funktionalitäten wie JavaScript und ist deshalb zu dieser Sprache kompatibel. Allerdings geht man mit Jscript noch einen Schritt weiter und ermöglicht Zugriffe auf das Betriebssystem Windows und das Dateisystem, was die Sprache viel mächtiger macht als JavaScript [MÜNZ, 2001]. Alles in allem sind sich aber Jscript und JavaScript sehr ähnlich und man versuchte nicht zuletzt deshalb beide Sprachen durch den ECMA – Standard unter einen Hut zu bringen.

2.2.3 VBScript

VBScript steht für Visual Basic Script und ist eine zweite Entwicklung von Microsoft. Sie basiert auf der Programmiersprache Basic und ist nicht kompatibel zu JavaScript. Auch wird diese Sprache nur vom Microsoft Internet Explorer interpretiert. Dadurch kann man im Internet nur wenig Seiten finden, in die VBScript integriert ist. Diese Tatsache liegt aber auch darin begründet, dass VBScript noch mehr Möglichkeiten besitzt, tiefgehende Eingriffe in das Betriebssystem Windows vorzunehmen, die wiederum nicht ganz ungefährlich sind. In VBScript programmierte Vieren, welche großen Schaden anrichteten, haben dies bewiesen.

2.3 Document Object Model (DOM)

Wie schon in den einleitenden Worten in Kapitel 1.2 aufgeführt wurde, bilden die Festlegungen des DOM eine Grundlage für die Dynamik des SVG – Dokumentes. Da Dynamik durch eine Interaktion des Nutzers ausgelöst werden kann, werden diese Festlegungen auch als Grundlage für Interaktivitäten angesehen und sollen deshalb im folgenden Abschnitt näher erläutert werden.

2.3.1 Allgemeines

Mit der Einführung von Scriptsprachen zur Optimierung bzw. Manipulierung von Web – Seiten entstand ein Machtkampf der beiden größten Anbieter Netscape und Microsoft (siehe Kapitel 2.2). Beide Konkurrenten bieten ihren eigenen Web – Browser sowie ihr eigene Scriptsprache an, sodass dem Programmierer nur die Wahl blieb, sich für eine Variante zu entscheiden oder doppelt zu programmieren. Hinzu kommt die Misere der verschiedenen Browser – Versionen, die je nach Version Scriptsprachen unterschiedlich unterstützen. Aus dieser Not heraus wurde das W3C beauftragt, eine allgemeine Sprachnorm zu entwickeln. Das W3C erarbeitete jedoch keine allgemeingültige Scriptsprache, sondern ein allgemeines Modell für Objekte und Dokumente zur Festlegung einer Objekthierarchie, welches außerdem definiert, in welcher Weise diese Objekte angesprochen und verändert werden können. Das DOM Level 1.0 war entstanden und wurde 1998 eine offizielle W3C – Empfehlung, der im Herbst 2000 das DOM Level 2.0 folgte. Scriptsprachen, die sich als Ergänzungssprache zu HTML bzw. XML – Applikationen verstehen, sollten deshalb das DOM vollständig unterstützen. Dieses ist aber noch nicht der Fall, was wiederum zu Problemen führt und die Programmierung erschwert.
Verdeutlichen lässt sich eine DOM – Objekthierarchie am besten mit Hilfe einer Baumstruktur. Die einzelnen Bestandteile einer solchen Baumstruktur werden als Knoten (node) bezeichnet, die man wiederum in Elementknoten, Attributknoten und Textknoten unterscheiden kann. Um jedoch zu einem hierarchischen Aufbau zu gelangen, erfolgt noch eine weitere Namensgebung.

Übergeordnete Knoten werden dabei immer als Elternknoten (parentnode), untergeordnete Knoten als Kindknoten (childnode) bezeichnet. Handelt es sich bei einem untergeordneten Knoten jedoch um einen Attributknoten, so spricht man von einem assoziierten Knoten. Anhand dieser Beschreibung lässt sich erahnen, dass ein Dokument schnell sehr verschachtelt und tiefgründig sein kann. Deshalb soll der Aufbau einer solchen Baumstruktur mit Hilfe eines Beispieles in Kapitel 2.3.2 veranschaulicht werden.
Diese Hierarchie der einzelnen Objekte ist die Grundlage dafür, um mit den Eigenschaften und Methoden, die das DOM außerdem zur Verfügung stellt, einzelne Elemente zu referenzieren und deren Attribute bzw. Attributwerte zu ändern. Einige dieser Eigenschaften und Methoden sollen ebenfalls in Kapitel 2.3.2 erläutert werden.

2.3.2 SVG – DOM

Wie schon erwähnt, basiert das in SVG implementierten DOM auf der DOM Level 2 Spezifikation des W3C. Es geht aber teilweise auch darüber hinaus, um spezielle SVG – Eigenschaften zu ermöglichen [SALATHÉ, 2001]. Im Anschluss soll an einem Beispiel die Objekthierarchie innerhalb eines SVG – Dokumentes verdeutlicht werden. Dazu zuerst der Quelltext.

```
<svg width="12cm" height="4cm" viewBox="0 0 100 35">
  <rect x="2" y="2" width="96" height="31" style="fill:none; stroke:black; stroke width:1"/>
  <g id="linien" style="fill:darkgreen; stroke:darkgreen">
      <line  x1="5" y1="28" x2="20" y2="7" style="stroke-width:0.5"/>
      <line  x1="20" y1="28" x2="35" y2="7" style="stroke-width:1.2"/>
      <line  x1="35" y1="28" x2="50" y2="7" style="stroke-width:2"/>
  </g>
  <polygon style="fill:red; stroke:black; stroke-width:0.5" points="55,15 65,15 65,22 55,22"/>
  <text x="75" y="19" style="font-family:Arial;font-size:6">SVG</text>
</svg>
```

Das aufgeführte Codefragment umschreibt ein SVG – Dokument mit folgendem Aussehen.

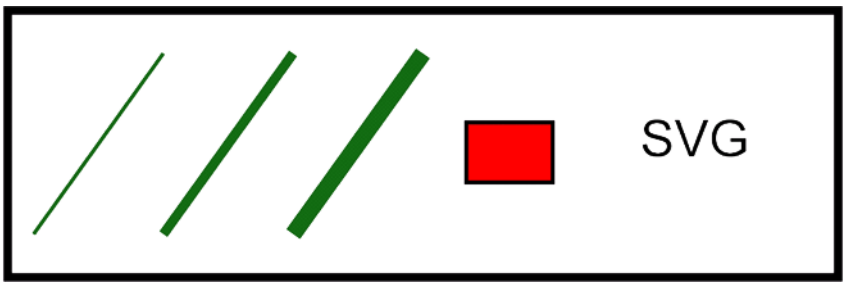

Abbildung 1: SVG – Graphik.

Wie man schon durch die Gliederung des Quelltextes erkennen kann, besteht die Graphik nach dem einleitenden Elementknoten (*<svg>*) und dessen drei Attributknoten aus vier weiteren Elementknoten. Diese wären im einzelnen das umrahmenden Rechteck (*<rect>*), eine Gruppe von drei unterschiedlichen Linien (*<g>*), das rot gefüllte Polygon (*<polygon>*) und der Schriftzug (*<text>*).

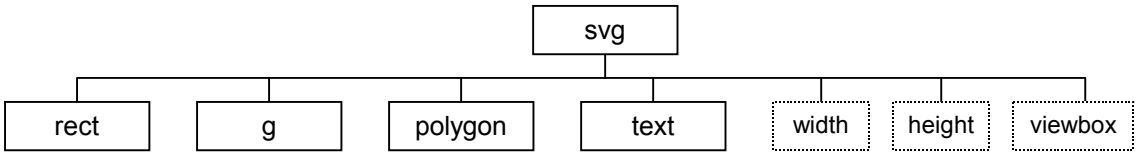

Abbildung 2: ein Elementknoten (Elternknoten) mit vier untergeordneten Elementknoten (Kindknoten) und drei Attributknoten (assoziierte Knoten).

Vom SVG – Element aus gesehen handelt es sich also bei den vier Elementknoten in der Ausdrucksweise des DOM um vier Kindknoten. Diese vier Kindknoten stellen jedoch gleichzeitig für die untergeordneten Knoten vier Elternknoten dar. Zwei der so gesehenen Elternknoten sind nur noch Attributknoten (z.B. „style“) untergeordnet, nämlich dem Polygon und dem Rechteck. Man spricht dabei also von assoziierten Knoten.

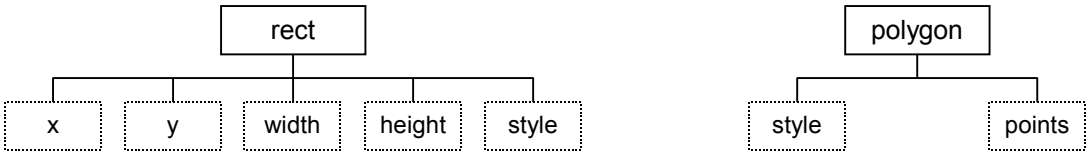

Abbildung 3: je zwei Elementknoten (Elternknoten) mit fünf bzw. zwei Attributknoten (assoziierte Knoten).

Dem Elementknoten für das Textelement folgen ebenfalls Attributknoten, jedoch auch ein Textknoten, der in DOM – Sichtweise in diesem Fall einen Kindknoten darstellt.

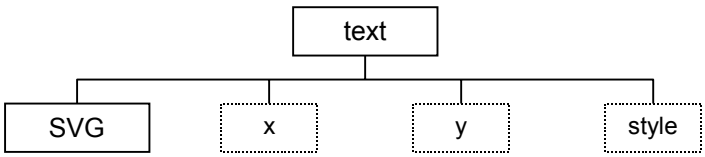

Abbildung 4: ein Elementknoten (Elternknoten) mit einem Textknoten (Kindknoten) und drei Attributknoten (assoziierte Knoten).

Abschließend noch die Betrachtung des Elementknoten der Gruppe. Diesem folgen Attributknoten und weitere Elementknoten für die einzelnen Linien. Die Elementknoten der Linien gelten als Kindknoten, denen wiederum Attributknoten angehangen sind.

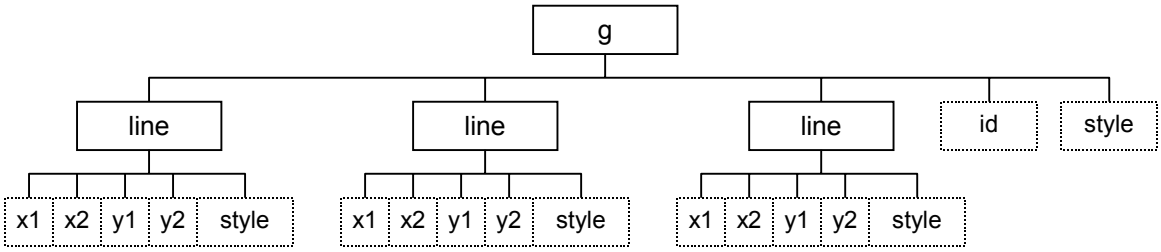

Abbildung 5: ein Elementknoten (Elternknoten) mit zwei Attributknoten (assoziierte Knoten) und drei untergeordneten Elementknoten (Kindknoten) mit wiederum fünf Attributknoten (assoziierte Knoten).

Aus Platzgründen wird auf eine gesamte Darstellung der Baumstruktur an dieser Stelle verzichtet. Trotzdem machen diese einzelnen Abbildungen schnell deutlich, wie komplex und verschachtelt schon ein so kleines SVG – Dokument sein kann. Deshalb ist es notwendig, einfache Methoden zu besitzen, um auf die einzelnen Knoten zugreifen zu können. Mit der Implementierung des DOM in das SVG – Format werden entsprechende Eigenschaften und darauf anzuwendende Methoden bereitgestellt. Einige sollen nun kurz vorgestellt werden. Wichtig ist dabei allerdings noch zu wissen, dass diese Methoden nur mit Hilfe einer Scriptsprache (in diesem Fall JavaScript) umgewandelt werden können. Sozusagen liefert das DOM die Eigenschaften und Methoden, die mit Hilfe von JavaScript dem Browser verständlich gemacht werden.

Jeder Zugriff auf einzelne Knoten innerhalb eines SVG – Dokumentes erfordert immer zu aller erst eine Referenz auf das Dokument selbst, die wiederum in einer Variablen gespeichert wird (z.B. *var SVGKarte*). Zur Referenzierung steht das *document* – Objekt zur Verfügung. Es bezieht sich auf den Inhalt des Browser – Fenster und steht innerhalb der DOM – Hierarchie nach dem *window* – Objekt an zweiter Stelle. Die entscheidende Methode für den Zugriff lautet dann *getSVGDocument( )*, wobei zwischen dem *document* – Objekt und der Methode der Name aufgeführt wird, der bei der Einbettung der SVG-Graphik in die HTML – Datei vergeben wurde. Der vollständige Ausdruck müsste also lauten:

```
var SVGKarte = document.name.getSVGDocument();
```

Nun steht einer Bearbeitung der einzelnen Knoten nichts mehr im Wege, denn der Zugriff auf diese erfolgt mittels einer der Methoden

- *getElementById( )*, um über das vergebene Attribut „id“ den Knoten anzusprechen

  ```
  var Knoten = SVGKarte.getElementById(„idname“);
  ```

- *getElementesByTagName( )*, um über das einleitende Element den Knoten anzusprechen

```
var Knoten = SVGKarte.getElementByTagName(„elementname“);
```

Nach dem Zugriff auf den gewünschten Knoten kann man verschiedene Möglichkeiten anwenden, um die Eigenschaften desselben zu ermitteln. Eine Auswahl dieser bilden die folgenden Methoden.

- *getNodeType( )*, um den Typ des Knotens zu erhalten, wobei der Rückgabewert 1 für einen Elementknoten, der Rückgabewert 2 für einen Attributknoten und der Rückgabewert 3 für einen Textknoten steht.

  ```
  var Knotentyp = Knoten.getNodeTyp( );
  ```

- *getNodeName( )*, um den Namen des Knotens, z.B. „circle“, zu erhalten.

  ```
  var Knotenname = Knoten.getNodeName( );
  ```

- *getChildNodes( )*, um eine Liste der Kinderknoten des angesprochenen Knotens zu erhalten. Die Anzahl der Kinderknoten kann dann mit Hilfe der Methode *getLength( )* ermittelt werden.

  ```
  var Kinderknoten = Knoten.getChildNodes( );
  ```

  Besitzt ein Element nur einen Kinderknoten, kann dieser durch den Ausdruck *getFirstChild( )* angesprochen werden.

- *getParentNode( )*, um den Elternknoten des angesprochenen Knotens zu referenzieren.

  ```
  var Eltern = Knoten.getParentNode( );
  ```

- *getAttribute( )*, um den Wert eines Attributes des angesprochenen Knotens zu ermitteln. Handelt es sich z.B. um ein Rechteck, kann die x-Position abgefragt werden.

  ```
  var Attributwert = Knoten.getAttribute(„x“);
  ```

- *getStyle( )*, um einzelne Eigenschaften des Attributes „style“ zu erhalten, z.B. die Farbe der Füllung eines Rechteckes oder die Breite der umrandenden Linie.

  ```
  var Eigenschaften = Knoten.getStyle( );
  ```

  Da das Attribut „style“ aber oft mehrere Eigenschaften beinhalten kann, muss eine dieser angesprochen werden. Dies geschieht mit Hilfe der Methode *item( )*.

  ```
  var Eigenschaft1 = Eigenschaften.item(1);
  ```

  Den eigentlichen Wert dieser Eigenschaft erhält man dann mit Hilfe von *getPropertyValue( )*.

  ```
  Eigenschaft1 = Eigenschaft1.getPropertyValue( );
  ```

- *getData( )*, um den Inhalt eines Textknotens zu erhalten. Auf den Textknoten wurde allerdings noch nicht zugegriffen. Er bildet einen Kinderknoten des oben angesprochenen Elementknotens und kann über *getFirstChild( )* referenziert werden.

```
var text = Knoten.getFirstChild( );
var inhalt = text.getData( );
```

Ähnlich viele Möglichkeiten existieren, um Eigenschaften von Knoten zu ändern. Sie bilden jeweils ein Paar mit der entsprechenden, oben aufgeführten Methode.

- *setAttribute( )*, um den Wert eines Attributes des angesprochenen Knotens zu ändern. Handelt es sich z.B. um einen Kreis, kann der Radius geändert werden.

```
Knoten.setAttribute( „r“ , „20“);
```

- *setProperty( )*, um eine Eigenschaft des „style“ – Attributes zu ändern. Dazu müssen jedoch die Eigenschaften erst angesprochen werden.

```
var Eigenschaften = Knoten.getStyle( );
Eigenschaft = setProperty(„fill“ , „red“);
```

  Handelt es sich beim Knoten um ein Rechteck, so wird dieses rot gefüllt.

- *setData( )*, um den Inhalt eines Textknotens zu ändern. Dieser muss aber, wie schon im vorangehenden Beispiel, erst angesprochen werden.

```
var text = Knoten.getFirstChild( );
text.setData(„Hallo“);
```

Somit ist es ohne Probleme möglich, Eigenschaften von Elementen oder Texte innerhalb der SVG – Karte zu ändern.
Nun fehlt noch eine der wichtigsten Eigenschaften des DOM, nämlich die Möglichkeit, einzelne Objekte dynamisch zu erzeugen oder zu kopieren. Für kartographische Anwendungen ist diese von besonderer Bedeutung, da mit ihr dynamische Prozesse sehr gut dargestellt werden können oder der Nutzer die Möglichkeit hat, Elemente im Kartenbild hinzu zufügen. Die dazu bereitgestellten Methoden werden in dieser Arbeit z.B. verwendet, um dem Nutzer zu erlauben, einen von ihm festgelegten Weg innerhalb der Karte darzustellen. Zur Schaffung eines neuen Elementes stehen dabei zwei Methoden zur Verfügung. Diese wären zum Ersten die Methode

- *createElement( )*, um ein neues Element zu generieren, z.B. einen Kreis,

```
var neuerKreis = SVGKarte.createElement(„circle“);
```

und zum Zweiten die Methode

- *cloneNode( )*, um ein bereits vorhandenes Objekt zu klonen, welches vorher mit *getElementById( )* referenziert wurde.

  var neuerKnoten = Knoten.cloneNode(true oder *false);*

  Wie aus dem Beispiel ersichtlich wird, erfordert diese Methode den Parameter „true“ oder „false“. Dabei legt „true“ fest, dass alle Kinderknoten des geklonten Elementes mit geklont werden. „false“ bestimmt dagegen, dass nur der Knoten selbst geklont wird.

Stellt sich die Frage, wie das neu geschaffene Element in die Graphik eingefügt werden kann. Im ersten Schritt muss dabei das Element referenziert werden, welches in unmittelbarer Umgebung zum neuen Element innerhalb der DOM – Hierarchie stehen soll. Eingefügt wird das neue Element dann mit Hilfe der Methode

- *appendChild( )*, welche dieses hinter den letzten Kinderknoten des referenzierten Elementes einordnet,

  var Eltern = SVGKarte.getElementById(„Elternknoten“);
  Eltern.appendChild(neuerKnoten);

oder durch die Methode

- *insertBefore( )*, die das neue Element vor den referenzierten Knoten einordnet.

  var Nachbar = SVGKarte.getElementById(„Nachbarknoten“);
  Nachbar.insertBefore(neuerKnoten);

All diese Methoden können genutzt werden, um Dynamik innerhalb des SVG – Dokumentes zu erzeugen. Dabei werden die Methoden in Form von Funktionen in der Scriptdatei verarbeitet. Die Funktionen werden wiederum durch Interaktion des Nutzers ausgelöst. Dazu benötigt man aber sozusagen „Sensoren“, die eine Interaktion des Nutzers registrieren und weiterverarbeiten. Diese „Sensoren“ bezeichnet man üblicherweise als Event – Handler. Mit deren Hilfe werden „Events“, also Ereignisse, wie z.B. ein vom Nutzer ausgelöster Mausklick, registriert. Diese registrierten Ereignisse können dann Funktionen auslösen, die mit den oben aufgeführten Methoden arbeiten und z.B. die Farbe eines Objektes ändern. Ein interaktives SVG – Dokument ist also entstanden.
Einige dieser Ereignisse, die in SVG integriert sind, sollen im folgenden erläutert werden. Dabei wird aber nicht mehr von Ereignissen gesprochen, sondern von der englischen Form, dem Event, da dies die gebräuchlichere Ausdrucksweise ist.

2.4 Events

2.4.1 Animation – Events

Animation – Events stellen eine Möglichkeit dar, Animationen durch Interaktionen des Nutzers auszulösen. Wie in Kapitel 2.1 beschrieben, kann ein Animationselement die Attribute „begin“, „dur“ und „end“ enthalten. Diesen Attributen kann ein Sekundenwerte als Attributwerte zugewiesen werden, doch ist dadurch keine Interaktionsmöglichkeit geschaffen. Animation – Events schaffen aber Interaktivität, da sie erlauben, eine Animation z.B. durch einen Mausklick zu starten. Dies geschieht durch die Zuweisung des gewünschten Events als Attributwert. Eine Auswahl an Animation – Events sieht wie folgt aus.

- *activate* reagiert, wenn irgend eine Taste über dem Objekt gedrückt und wieder gelöst wird (Maus oder Tastatur).
- *click* reagiert, wenn über dem Objekt die Maustaste gedrückt und wieder gelöst wird.
- *mouseover* reagiert, wenn der Mauszeiger in ein Objekt hineinbewegt wird bzw. sich über einem Objekt befindet.
- *mouseout* reagiert, wenn der Mauszeiger aus einem Objekt hinausbewegt wird bzw. sich nicht mehr über einem Objekt befindet.
- *mousemove* reagiert, wenn der Mauszeiger über einem Objekt bewegt wird.
- *mousedown* reagiert, wenn die Maustaste über einem Objekt gedrückt gehalten wird.
- *mouseup* reagiert, wenn die gedrückte Maustaste über einem Objekt losgelassen wird.

Eine vollständige Auflistung kann unter der SVG – Spezifikation [W3C, 2002] nachgelesen werden. Die hier aufgeführten Events zählen aber zu den gebräuchlichsten.
Soll eine Animation durch einen Mausklick auf das zu animierende Element ausgelöst werden, würde man das Attribut „begin“ also folgendermaßen belegen: *begin = „click“*. Möchte man die Animation durch einen Mausklick auf ein anderes Element (z.B. einem Button) starten, muss diesem anderem Element das Attribut „id“ mit einem Attributwert (z.B. *id=“Startbutton“*) vergeben werden. Die Animation könnte man dann starten, in dem man das „begin“ – Attribut so formuliert.

begin = „Startbutton.click“

Somit ist durch die Animation – Events wieder ein Stück mehr Interaktivität möglich.

2.4.2 Event – Attribute

Event – Attribute gleichen den Event – Handlern, wie sie aus HTML bekannt sind. Wie der Name schon sagt, können in diesem Fall Events als Attribute eines Elementes verarbeitet werden. Mit Hilfe dieser Attribute werden dann Scriptfunktionen gestartet. Auch bei Event – Attributen existieren mehr als hier aufgeführt werden, die bekanntesten sind allerdings:

- *onactivate*
- *onclick*
- *onmouseover*
- *onmouseout*
- *onmousemove*
- *onmousedown*
- *onmouseup*
- *onload*, reagiert beim Laden der SVG – Datei.

Die unkommentierten Events reagieren in der gleichen Situation, wie die oben aufgeführten Animation – Events. Man könnte also vereinbaren, dass der Name eines Objektes mit Hilfe einer Scriptfunktion angezeigt wird, wenn man sich mit der Maus über dieses Objekt bewegt. Würde es sich bei dem Objekt um einen Kreis handeln, könnte dies in SVG wie folgt aussehen.

```
<circle cx="10" cy="10" r="5" onmouseover = "Scriptfunktion"/>
```

Um auf den Beginn, die Weiderholung oder das Ende einer Animation reagieren zu können, werden in SVG zusätzlich drei spezielle Event – Attribute bereit gestellt:

- *onbegin*
- *onrepeat*
- *onend.*

Sie erlauben es, eine Scriptfunktion in dem Moment aufzurufen, wenn eine Animation startet (*onbegin*), wenn sie wiederholt wird (*onrepeat*) oder wenn sie zu Ende geht (*onend*).

2.4.3 DOM2 – Events

Die dritte Art von Events bilden die DOM2 – Events, die in das DOM Level2 Event Model integriert sind. Da, wie bereits beschrieben, das DOM Level 2 in SVG zur Anwendung kommt, können somit auch diese Events genutzt werden. Ein großer Vorteil liegt dabei in der Möglichkeit, auch ohne dem Attribut „id“ ein Element ausfindig zu machen. In Kapitel 2.3.2 wurde beschrieben, dass man eine Referenz auf ein Element mit Hilfe der Methode *getElementById( )* erstellen kann. Wenn z.B. eine Funktion durch das Event – Attribut *onclick* ausgelöst wird, müsste man den Wert des Attributes „id“ als Parameter der Funktion übergeben, um das Element im nächsten Schritt innerhalb der Scriptfunktion ansprechen zu können. Die

Vergabe des Attributes „id“ kann aber, vor allem bei sehr großen SVG – Dokumenten, zu einer aufwendigen Sache werden, ist aber auch wiederum oft unvermeidlich.
Um dies zu vereinfachen, steht mit dem DOM Level 2 Event Model das Objekt *event* zur Verfügung, welches die Methode *getTarget( )* besitzt. Mit ihr ist es möglich, eine Referenz zu dem Objekt zu erstellen, über das die Scriptfunktion ausgelöst wurde. Dazu wird der Funktion der Parameter „evt“ übergeben. Wird z.B. über das Event – Attribut *onclick* beim klicken auf einen Kreis eine Scriptfunktion ausgelöst, sieht das wie folgt aus.

```
<circle cx="10" cy="10" r="5" onclick = "Funktionsname(evt)"/>
```

Der Parameter „evt“ wird an eine Variable der Scriptfunktion übergeben, mit der es dann in Verbindung mit der Methode *getTarget( )* möglich ist, das entsprechende Element zu referenzieren. Dies könnte innerhalb der Scriptdatei folgendermaßen aussehen.

```
Funktionssname(evt) { ...
var Element = evt.getTarget( );
... }
```

Diese Methode offenbart viele Möglichkeiten und kommt im weiteren Verlauf der Arbeit noch häufiger zum Einsatz.
Ein weiteres nützliches Objekt innerhalb des DOM2 Event Model bildet das Objekt *mouseevent*. Es beinhaltet Methoden, um einzelne Informationen einer Anwenderaktion zu ermitteln und zu verarbeiten. Bei diesen Methoden handelt es sich z.B. um

- *getClientX( ), getClientY( ),* um die Mausposition, von der linken oberen Ecke des SVG – Dokumentes aus gesehen, zu erhalten;
- *getScreenX( ), getScreenY( ),* um die Mausposition, von der linken oberen Ecke des Bildschirmbereiches aus gesehen, zu ermitteln.

Im letzten Abschnitt dieses Kapitels soll noch eine weitere Möglichkeit beschrieben werden, die im DOM2 Event Model integriert ist. Sie erlaubt es, nachträglich Events innerhalb eines Elementes einzufügen oder zu entfernen. Dies erfolgt mit den Methoden

- *addEventListener( )*, um ein Event – Attribut hinzuzufügen;
- *removeEventListener( )*, um ein Event – Attribut zu entfernen.

Dabei müssen allerdings innerhalb der Klammer drei Parameter übergeben werden. Diese wären: die Art des Events (z.B. *click*), der Name der auszuführenden Scriptfunktion und ein Parameter, der über die Möglichkeit der Nutzung des Event – Capturing entscheidet. Event – Capturing organisiert den Ablauf von Events innerhalb einer hierarchischen Struktur [SALATHÉ, 2001]. Der entsprechende Parameter kann mit „true“ oder „false“ belegt werden.

Ist das Element, dem das Event hinzugefügt werden soll erst einmal referenziert (z.B. ein Rechteck), könnte der Ausdruck in der Scriptdatei beispielsweise so aussehen.

Rechteck.addEventListener(„click“ , Funktionsname , false);

Diese Methoden verbergen wieder eine Vielzahl von Möglichkeiten. Man könnte z.B. einen Button erst funktionsfähig machen, wenn vorher auf einen anderen Button geklickt wurde.

2.5 Zusammenspiel

Nach der Aufführung der Grundlagen für Interaktivitäten innerhalb eines SVG – Dokumentes soll noch eine kurze Bemerkungen zum Zusammenspiel dieser einzelnen Möglichkeiten folgen, denn nur dieses erlaubt einen reibungslosen Ablauf nach dem erfolgten Anwenderereignis. Es reicht also bei weitem nicht aus, sich auf eine dieser Grundlagen festzulegen. Entscheidend ist immer noch die Kenntnis einer Scriptsprache, in diesem Falle also JavaScript, denn erst mit ihr ist es möglich, die Methoden anzuwenden, die z.B. über das DOM auf einzelne Elemente zugreifen und sie ändern. Die Kenntnisse von JavaScript und dem DOM reichen aber wiederum alleine nicht aus, da man wissen bzw. festlegen muss, mit welchem Ereignis der Nutzer eingreifen darf. Die Kenntnisse der verschiedenen Events werden benötigt. Dieses Zusammenspiel der verschiedenen Grundlagen kann bei der Beschreibung der einzelnen Interaktionsmöglichkeiten im späteren Verlauf der Arbeit noch genauer betrachtet werden.

3 Interaktionsmöglichkeiten in Internetkarten

3.1 Vorüberlegungen

Will man eine interaktive Karte für das Internet gestalten, sollten schon zu Beginn der Arbeit einige Vorüberlegungen durchgeführt werden. Mit der Festlegung des Formates, also SVG, werden dem Nutzer schon einige Möglichkeiten von Haus aus geboten (siehe [VOIGT, 2001]). Diese müssen unbedingt bei den Vorüberlegungen mit in Betracht gezogen werden, damit es später nicht zu Konflikten zwischen den SVG – spezifischen und den programmierten Interaktionsmöglichkeiten kommt. Besonderes Augenmerk wird dabei auf die Möglichkeiten des Ein- und Auszoomens sowie der Verschiebung des Ausschnittes („ZoomAndPan“ – Funktion) gelegt. Oft werden diese zu Gunsten eigener „Zoom“ und „Pan“ – Möglichkeiten außer Kraft gesetzt, was aber keinen Rückgang an Funktionalität bedeutet.
Mit der Vielzahl an Möglichkeiten, die in Kapitel 2 aufgeführt wurden, wird schnell deutlich, dass man den Nutzer durchaus auch zur Verzweiflung bringen kann. Wie bei der Gestaltung von HTML – Seiten gilt auch hier nicht unbedingt die Redewendung „viel hilft viel“. Vielmehr sollten wenige Interaktionsmöglichkeiten zur Veränderung der Informationsdichte dienen und dem Nutzer bei der Orientierung und dem Verständnis des Inhaltes helfen. Man muss sich also über Sinn und Unsinn einer Interaktion im klaren sein, gleichzeitig aber auch Aufwand und Nutzen abwägen.

3.2 Ausgewählte Beispiele

3.2.1 „mouseover“ – Effekte

Der Begriff „mouseover“ wurde bereits in Kapitel 2.4.1 erläutert. Hinter ihm verbergen sich verschieden Möglichkeiten eine Informationserweiterung direkt auf eine Nutzeraktion (dem Anfahren des Elementes mit der Maus) folgen zu lassen. Dies kann zum Beispiel eine Farbveränderung zur besseren Hervorhebung eines Objektes oder die Anzeige des Namens desselben sein. Gerade für kartographische Anwendungen ist diese Möglichkeit äußerst sinnvoll, da mit ihr Übersichtlichkeit gewonnen werden kann. Es müssen z.B. nicht mehr alle Namen gleichzeitig in der Karte aufgeführt werden, wie es bei konventionellen Papierkarten der Fall ist, sondern der Nutzer entscheidet selbst, welcher Name angezeigt werden soll, indem er den Mauszeiger z.B. auf das gewünschte Gebäude bewegt. Das bedeutet aber auch wiederum, dass ein bloßes Betrachten der Karte nicht ausreicht, um alle Informationen zu erfassen und dass dem Anwender verständlich gemacht werden muss, den Mauszeiger über die Karte zu bewegen. „mouseover“ – Effekte wurden in der anschließenden Erstellung eines Prototyps mehrfach umgesetzt.

3.2.2 Ebenensteuerung

Die Interaktionsmöglichkeit der Ebenensteuerung ist eine weit verbreitete Anwendung, wenn es darum geht, die Informationsdichte im Kartenbild zu verändern. Die Möglichkeit, die Sichtbarkeit einzelner Informationsebenen zu steuern, ist von digitalen Karten im Rasterformat her ebenso bekannt wie von Papierkarten, bei denen mit Hilfe von auflegbaren Folien eine Veränderung der Informationsdichte bewirkt wurde. Auch innerhalb einer Karte im SVG – Format ist dies ohne Probleme möglich. Dazu werden innerhalb des SVG – Dokumentes die Objekte, die eine Informationsebene darstellen, zu einer Gruppe zusammengefasst. Mit Hilfe der im Kapitel 2.3.2 aufgeführten Methoden kann dann die Sichtbarkeit dieser Gruppe gesteuert werden. Mit der damit erreichten Interaktivität kann man z.B. das Gewässernetz oder eine bestimmte Gruppe von Signaturen ein- bzw. ausblenden.

3.2.3 Verlinkung

Wie auch in HTML, können in ein SVG – Dokument Hyperlinks, also Verweise zu anderen Internetseiten eingearbeitet werden. Dies bedarf keiner aufwendigen Programmierung, sondern einfach nur der Nutzung des Elementes **<a>**. Diesem Element kann das Attribut „href“ zugewiesen werden, welches wiederum als Attributwert die gewünschte URL (**U**niform **R**esource **L**ocators) erhält. Jedes beliebige Objekt kann innerhalb des SVG – Quelltextes mit dem <a> Element umschlossen werden (<a> Objekt) und ist somit verlinkbar. Es könnten z.B. weiterführende Verweise mit Gebäuden oder Straßenbahnhaltestellen innerhalb der Karte verknüpft werden und dadurch für eine Informationserweiterung sorgen.

3.2.4 Navigation

Dem Nutzer zu erlauben, nach seinen Vorstellungen einen Kartenausschnitt zu wählen und den Maßstab der Darstellung zu verändern, gehört wohl zu den grundlegendsten Interaktions-möglichkeiten einer interaktiven Internetkarte. Gerade in diesem Punkt ist ein gut durchdachtes Konzept erforderlich, vor allem, wenn die Arbeit mit einer zusätzlichen Übersichtskarte erwünscht wird. In dieser könnte dann der Kartenausschnitt durch das Verschieben eines Rechteckes, welches den aktuellen Ausschnitt wiederspiegelt, gewählt werden. Aber auch über Richtungspfeile am Kartenrand sollte eine Bewegung innerhalb der Karte möglich sein.
Ein- und Auszoomen bilden die zweite Navigationsmöglichkeit. Sie sollte allerdings beschränkt sein, da sonst die Karte bis zur Unkenntlichkeit verkleinert oder bis zum Verlust jeglicher Überschaubarkeit vergrößert werden könnte. Um den Wechsel von der Originalansicht, also der Darstellung der gesamten Karte, zur Detailansicht zu erlauben, wären z.B. Zoomstufen von 200% und 400% sinnvoll.

4 Vorarbeiten zur Erstellung eines Prototyps

Wie schon in den Aussagen zur Zielstellung beschrieben, sollen die zu Beginn der Arbeit erläuterten Eigenschaften und Möglichkeiten von SVG genutzt werden, um diese zu einem Prototypen einer interaktiven Karte, im speziellen einer interaktiven Karte des Campus der Technischen Universität Dresden, zu verarbeiten. In diesem Kapitel sollen die vorbereitenden Arbeitsschritte bis zur Einarbeitung der Interaktionsmöglichkeiten beschrieben werden.

4.1 Verwendete Software

Die einzelnen Arbeitsschritte erfordern die Nutzung verschiedener Software. Für die Arbeiten an der Ausgangskarte wurde das Vektorgraphikprogramm Macromedia Freehand 9 genutzt. Für den leider immer noch umständlichen Weg des Exports in das SVG – Format kam es zur Anwendung unterschiedlicher Programme mit ebenso unterschiedlichen, aber für die weitere Arbeit bedeutsamen Ergebnissen. Es handelt sich dabei zum einen um das Vektorgraphik-programm Corel Draw 10 und zum anderen um das Vektorgraphikprogramm Adobe Illustrator 9. Ein kurzer Vergleich der Exportergebnisse soll im Abschnitt 4.3.2 vorgenommen werden, weiterführende Aussagen findet man aber auch in der Arbeit von Y. Voigt [VOIGT, 2001]. Für die weitere Bearbeitung und Erstellung der für die Internetpräsentation notwendigen Dateien wurde der Texteditor „Ulli Meybohm's HTML Editor Phase 5“ genutzt. Er zeichnet sich durch eine Vielzahl von Automatisierungen aus und ist zu dem im WWW kostenlos erhältlich. Für zusätzliche Layout – Arbeiten kam außerdem das Rastergraphikprogramm Adobe Photoshop 6 zum Einsatz.

4.2 Vor dem Export

4.2.1 Ausgangskarte

Bei der Ausgangskarte handelte es sich um ein Karte des Campus der TU Dresden, deren Erstellung mit Hilfe von Macromedia Freehand erfolgte. Sie wurde inhaltlich überarbeitet und aktualisiert. Vor allem durch die Möglichkeit des sehr weiten Einzoomens, die SVG liefert, musste eine Vielzahl von Gebäuden hinsichtlich des Detaillierungsgrades und der Darstellungsgenauigkeit überarbeitet werden. Außerdem ist, wie von Y. Voigt [VOIGT, 2001] beschrieben, eine saubere Ebenen- und Formatnutzung innerhalb des Programms Freehand von immenser Bedeutung. Auch in Bezug auf diesen Punkt wurde die Ausgangskarte überarbeitet. Natürlich sind solche Arbeiten nicht erforderlich, wenn von vornherein eine Internetpräsentation geplant ist und dies schon bei der Erstellung der Karte berücksichtigt werden kann. Auch ist es in diesem Falle nicht notwendig, umfangreiche Gruppierungen vorzunehmen oder Schriftfreistellungen einzuarbeiten, da diese beim Export in SVG nicht berücksichtigt werden.

4.2.2 Das Schriftproblem

Üblicherweise werden bei der Erstellung von Karten mittels Macromedia Freehand die einzelnen Straßennamen an Pfade gebunden. Das Problem, das daraus beim Export entsteht, wurde schon von Y. Voigt [VOIGT, 2001] beschrieben und trat auch in dieser Arbeit auf. Der Straßenname wurde also nicht als ein Textelement im Ganzen exportiert, sondern jeder Buchstabe einzeln, um ihn bei einem gebogenen Schriftverlauf mittels einer Transformationsmatrix zu rotieren. Dies verdeutlicht das folgende Codefragment für den gebogenen Schriftzug „weg“.

```
<g transform="matrix(0.945522 0.325559 -0.325559 0.945522 -1546.23 4031.12)">
  <text x="0" y="8016" class="fil9 fnt1">w</text></g>
<g transform="matrix(0.93358 0.358368 -0.358368 0.93358 -1193.8 4158.63)">
  <text x="0" y="8016" class="fil9 fnt1">e</text></g>
<g transform="matrix(0.927185 0.374603 -0.374603 0.927185 -994.746 4236.59)">
  <text x="0" y="8016" class="fil9 fnt1">g</text></g>
```

Um dies zu umgehen, wurden alle Straßennamen vom Pfad getrennt und durch eine Rotation des gesamten Textes schon in der Freehand – Karte in die entsprechende Lage gebracht. Dies konnte natürlich nur erfolgen, da es der geradlinige Straßenverlauf innerhalb der Karte zuließ.
Man kann aber auch genauso in SVG Schriften an Pfade orientieren, allerdings ist dies erst nach dem Export möglich. Dabei ist es aber wiederum sehr müßig, den entsprechenden Pfad innerhalb des SVG – Quelltextes zu lokalisieren. Diesem muss dann das Attribut „id“ gegeben werden, sodass im entsprechenden Textelement auf den Pfad verwiesen werden kann. Konkret könnte dies wie folgt aussehen.

```
<path id="Textpfad" style="stroke:white; stroke-width:1" d="M-5525 11333c695,-37
1760,221 2152,638"/>
<text class="fil9 fnt1">
  <textPath xlink:href="#Textpfad" startOffset="20%">
    <tspan dy="80">weg</tspan>
  </textPath>
</text>
```

Im Element *<textPath>* wird also auf den Pfad „Textpfad“ verwiesen, der Schriftzug „weg“ verläuft entlang des Pfades.
Offensichtlich kann der Umgang mit gekrümmter Schrift sehr aufwendig werden insbesondere, wenn es sich bei der Karte um einen Stadtplan handelt. Auf jeden Fall kann man nicht über den Export in Einzelbuchstaben hinwegsehen, da damit auch die in SVG integrierte Textsuche keine sinnigen Ergebnisse hervorbringt.

4.2.3 Spezielle Bearbeitung für animierte Darstellungen

Sollen in der Internetkarte mit Hilfe von Interaktionen des Anwenders Animationen ausgelöst werden, so müssen schon in der Ausgangskarte einige Grundlagen dafür geschaffen werden. Dies ist insbesondere für Pfad – Animationen der Fall. Während der Bearbeitung der Karte mit Macromedia Freehand wird oft sehr nachlässig mit übereinander verlaufenden Liniensignaturen umgegangen. Tritt eine Linie für einen kurzen Abschnitt in den Hintergrund und ist somit nicht mehr sichtbar, so wird oft im verdeckten Bereich nicht durchgezeichnet, sondern abgesetzt und neu begonnen. Der Pfad ist somit unterbrochen, was wiederum eine Animation entlang desselben Pfades ebenfalls unterbrechen bzw. abbrechen lassen würde.
Konkret kann dies am Beispiel der Darstellung der Buslinie 61 erläutert werden. Sie wird in der linken Hälfte der Karte in beide Fahrtrichtungen durch eine Linie dargestellt. Erst durch den späteren mehrspurigen Verlauf der Straße kommen Linien für beide Fahrtrichtungen zum Einsatz. Für eine Fahrtrichtung endet die Linie also beim Zusammenlaufen dieser, da sie nicht übereinander weitergeführt wurden sind. Dies hat aber verheerende Folgen, will man eine Animation entlang dieser Buslinien programmieren. Sie würde auf der Hälfte der Strecke abbrechen. Im ungünstigsten Fall kann es außerdem passieren, dass die Animation auf der jeweiligen Straßenseite in die verkehrte Richtung verläuft, denn die Richtung in der die Liniensignatur gezeichnet wurde, ergibt auch die Animationsrichtung. Die aus diesen Eigenarten ergründeten Bearbeitungen wurden ebenfalls an der Ausgangskarte durchgeführt.
Mit diesem Hintergrundwissen kann allerdings schon bei der Erstellung der Karte auf eine durchgehende Linienführung geachtet werden. Es verdeutlicht aber auch, dass von Anfang an klar sein muss, was animiert werden soll und in welche Richtung!

4.3 Erstellung der SVG – Datei

4.3.1 PDF als Zwischenspeicher

Beim Export der Ausgangskarte aus dem Programm Freehand wurde wie in der Arbeit von Y. Voigt [VOIGT, 2001] vorgegangen. Leider muss dazu immer noch das Format PDF (**P**ortable **D**ocument **F**ormat) in Anspruch genommen werden, um die aus dem Export entstandene Datei in eines der beiden, im nächsten Kapitel beschriebenen, Programme zu importieren und wiederum in das SVG – Format zu exportieren.
Ein direkter Export aus Freehand in SVG ist immer noch nicht wirklich möglich. Zwar existiert ein Art Druckertreiber (SVGmaker) mit dem man über die Druckfunktion eine SVG – Datei erstellen kann, die Ergebnisse sind allerdings nicht sehr zufrieden stellend.

4.3.2 Adobe Illustrator 9 vs. Corel Draw 10

Wie schon in Kapitel 4.1 angedeutet, stehen im weiteren Verlauf des Exports die beiden Programme Adobe Illustrator 9 und Corel Draw 10 zur Verfügung. Wenn man die Wahl hat, ist es also wichtig zu wissen, welches Programm das beste Ergebnis liefert. Grundsätzlich sind beide SVG – Karten im Aussehen ähnlich, jedoch unterscheidet sich der Quelltext sehr stark und dies sowohl im Aussehen als auch im Inhalt. Der große Vorteil von Adobe Illustrator 9 liegt in der Beibehaltung der einzelnen Ebenen in Form von Gruppen, welche beim Export über Corel Draw 10 verloren gehen. Zwei Codefragmente sollen einige Unterschiede der beiden Exportvarianten verdeutlichen, beide stehen für den farbigen Hintergrund der Karte.

- Export aus Adobe Illustrator 9:

```
<g id="Hintergrund" class="st47">
  <g>
    <path class="st29" d="M0.597,4.116h567.951v378.369H0.597V4.116z"/>
    <path class="st29" d="M284.573,193.301"/>
  </g>
</g>
```

- Export aus Corel Draw 10:

```
<path id="13409868" class="fil0 str0" d="M-10039 17l20036 0 0 13348 -20036 0 0 -13348z"/>
```

Adobe Illustrator 9 nutzt für diese Festlegung sechs Elemente, CorelDraw nur ein Element, allerdings ist auch keine Gruppeneinteilung vorhanden. Bei Adobe Illustrator 9 treten außerdem immer wieder Redundanzen bzw. völlig überflüssige Elemente auf, so z.B. das zweite *<path>* – Element. Eine aufwendige Überarbeitung ist also erforderlich.
Viel entscheidender für die weitere Arbeit ist die Eigenart von Corel Draw 10, jedem Element das Attribut „id“ automatisch einzufügen und mit einer Zahl zu belegen. Diese Tatsache ist von sehr hohem Nutzen, da dadurch leicht jedes Element ausfindig gemacht werden kann. Im Verlauf der Arbeit ist mittlerweile schon mehrmals erwähnt wurden, dass sehr viele Methoden zur Veränderung eines Objektes nur zum Einsatz kommen können, wenn eine Referenz zum entsprechenden Objekt erstellt wurde. Da dies oft über das Attribut „id“ erfolgt, ist es also von großer Wichtigkeit. Es galt jedoch immer als sehr aufwendig, einzelne Objekte innerhalb des mehrere Seiten langen Quelltextes zu lokalisieren, um ihnen dann dieses Attribut zu geben.
Dieser Vorgang kann aber mit Hilfe eines kurzen JavaScript – Programms und den von Corel Draw 10 automatisch vergebenen Werten entscheidend vereinfacht werden. Dabei braucht der Bearbeiter nur noch auf das gesuchte Objekt innerhalb der Karte zu klicken und er erhält den von Corel Draw 10 vergebenen Wert. Da in jedem Texteditor eine Suchfunktion integriert ist,

muss nur noch dieser Wert in die Suche eingegeben und gesucht werden, um das Objekt auch innerhalb des Quelltextes zu lokalisieren. Dies funktioniert aber nur, wenn man dem *<svg>* – Element am Anfang der SVG – Datei das Event – Attribut *onclick* mit dem Aufruf des Namens der Scriptfunktion einfügt. Somit kann ein, gerade für die Erstellung interaktiver Internetkarten, wichtiger Arbeitsschritt effektiver durchgeführt werden, was den Entschluss hervorbringt, bei einer Auswahlmöglichkeit Corel Draw 10 für den Export zu nutzen.
Der erläuterte Quelltext der Hilfsfunktion *ids( )* kann im Anhang nachgelesen werden.

4.4 Grundlegender Aufbau der Internetpräsentation

Es versteht sich von selbst, dass zur Darstellung der SVG – Karte im Internet eine bloße SVG – Datei nicht ausreicht. Vielmehr bildet ein Zusammenspiel mehrerer unterschiedlicher Dateien die vollständige Internetpräsentation. Diese unterschiedlichen Dateien müssen im Vorfeld angelegt werden.

4.4.1 Die HTML – Datei (*campus.html)*

Die HTML – Datei bildet neben der SVG – Datei die grundlegendste Voraussetzung für die Darstellung einer Internetkarte im WWW. Sie ist nach den üblichen Regeln aufgebaut, wobei der genutzt HTML – Editor automatisiert die Grundstruktur vorgibt. Allerdings ist es wichtig, innerhalb des *<head>* – Elementes auf die entsprechende CSS – Datei (**C**ascading **S**tyle **S**heets) und die JavaScript – Datei zu verweisen, um ein Zusammenarbeit dieser zu erlauben.

```
<link rel=stylesheet type="text/css" href="formate.css">
<script language="JavaScript1.2" src="script.js" type="text/javascript"></script>
```

Somit kann schon im einleitenden *<body>* – Element mit dem Event – Handler *onLoad* beim Laden der Datei die Funktion *initMap( )* aufgerufen werden. Sie erstellt dann Referenzen zu allen SVG – Dokumenten, die in die HTML – Datei mit dem *<embed>* – Element eingebettet wurden.

```
<embed name="karte" src="svg/campus.svg" type="image/svg-xml">
```

In die HTML – Datei sind fünf SVG – Dokumente eingearbeitet. Es handelt sich dabei um die Karte an sich (*campus.svg*), die Maßstabsleiste (*massstab.svg*), die Übersichtskarte (*klein.svg*), den Menüteil in der rechten Ecke (*menue.svg*) und um die Überschrift (*titel.svg*). Zum gestalterischen Arrangieren der verschiedenen SVG – Dokumente kam es zur Anwendung mehrerer, ineinander verschachtelter Tabellen. Außerdem wurden über das *<form>* – Element Formulare in der Art von Pulldown – Menüs angelegt, die eine Auswahl des zu suchenden Gebäudes oder Studienganges und die Zoomstufenauswahl erlauben. Ein Überblick zur Anordnung der einzelnen Dokumente gibt folgende Abbildung.

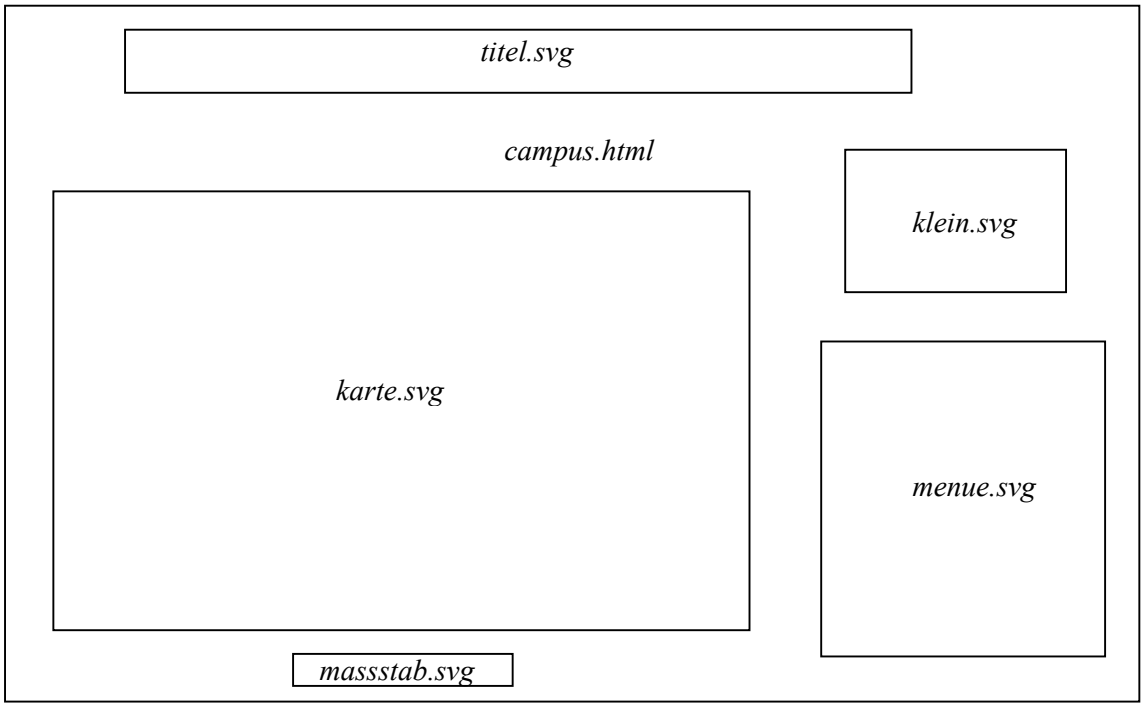

Abbildung 6: die, in die Datei *campus.html,* eingebetteten SVG – Dokumente.

4.4.2 Die JavaScript – Datei (*script.js*)

In der JavaScript – Datei werden alle Funktionen zusammengestellt, die für die Verarbeitung der Interaktionen des Anwenders notwendig sind. Außerdem enthält sie aber auch Funktionen, wie z.B. die in Kapitel 4.4.1 erwähnte Funktion *initMap( )*, die dazu dienen, eine gewisse Ausgangssituation wieder bzw. zum ersten Mal herzustellen. Der Aufbau dieser Datei bedarf keiner besonderen Regeln, allerdings muss natürlich den Vorgaben gefolgt werden, die eine Programmiersprache mit sich bringt. In der Regel werden am Anfang der Datei eine Vielzahl von Variablen mit Hilfe des Schlüsselwortes *var* festgelegt, die im Laufe der Arbeit immer wieder ergänzt werden. Bei diesen Variablen handelt es sich um so genannte globale Variablen. Sie kommen innerhalb der JavaScript – Datei immer wieder in verschiedenen Funktionen zum Einsatz. Im Anschluss daran werden die einzelnen Funktionen aufgeführt. Sie stellen einen Block aus Anweisungen dar, der beim Aufruf der Funktion durch den Interpreter des Web – Browsers verarbeitet wird. Die innerhalb der Funktion deklarierten Variablen werden im übrigen als lokale Variablen bezeichnet.

Die Scriptdatei *script.js* wurde außerdem genutzt, um Zusatzinformationen für z.B. Bibliotheken oder Cafeterias abzulegen. Dies geschah mit Hilfe von Arrays, also Feldern, in denen die Daten abgespeichert werden können. Diese wurden an den Anfang der Scriptdatei platziert.

4.4.3 Die CSS – Datei *(formate.css)*

Die CSS – Datei stellt im Prinzip einen Katalog der unterschiedlichen Formate innerhalb der Internetseite dar, also Farben, Schriftgrößen usw.. CSS ist eine Ergänzungssprache von HTML zur Definition von Formateigenschaften einzelner HTML – Elemente [MÜNZ, 2001]. Die Festlegung von Formaten in dieser Form ist nicht zwingend erforderlich, jedoch wird sie vom W3C empfohlen. Um z.B. dem Hintergrund eine bestimmte Farbe zu geben, müsste deshalb innerhalb der CSS – Datei folgendes Format festgelegt werden.

```
body { background-color:#FFFFCC; }
```

In der Datei *formate.css* wurden allerdings (außer für den Hintergrund) nur Formate für verschiedene Schriftarten bestimmt.

4.5 Anmerkungen zur Bildschirmauflösung

Bei der Erstellung der Internetseite wurde entgegen den Empfehlungen von O. Schnabel [SCHNABEL, 2002] mit einer Bildschirmauflösung von 1024 x 768 Pixel gearbeitet. Damit aber bei geringeren Auflösungen nicht Teile der Seite außerhalb des sichtbaren Bereiches dargestellt werden, oder bei höheren Auflösungen nicht zuviel Freiraum entsteht, wurden innerhalb der Funktion *initMap( )* in der Scriptdatei Anweisungen getroffen, welche die Größen der eingebetteten SVG – Dokumente in Abhängigkeit der Bildschirmauflösung berechnen. Somit kann bei jeder beliebigen Auflösung eine optimale Darstellung gewährleistet werden.

4.6 Überarbeitung der SVG – Datei

Nach dem erfolgten Export der Karte aus dem Programm Freehand über PDF und das Programm Corel Draw 10 müssen noch einige Überarbeitungen des SVG – Dokumentes vorgenommen werden. Sie werden in den folgenden Abschnitten beschrieben. Sind diese nachträglichen Überarbeitungen erfolgt, steht einer Einarbeitung der Interaktivitäten nichts mehr im Wege.

4.6.1 Gruppenbildung

Wie schon beim Vergleich der beiden Exportmöglichkeiten in Kapitel 4.3.2 angesprochen, gehen die einzelnen Ebenen beim Export über Corel Draw 10 verloren. Diese Ebenen können aber ohne Problem in Gestalt von Gruppen wieder eingefügt werden, da die Reihenfolge der einzelnen Elemente innerhalb der SVG – Datei der Zeichenreihenfolge innerhalb der Ausgangskarte in Freehand entspricht. Das heißt also, dass der zuerst gezeichnete Hintergrund in der untersten Ebene der Freehand – Karte im SVG – Quelltext an erster Stelle steht. Da Freehand die Möglichkeit bietet, die Anzahl der Elemente einer Ebene anzeigen zu lassen, ist es

dann nur noch eine Frage der „Zählkunst“ innerhalb von SVG, um die Ebenen bzw. Gruppen wieder einzufügen. Dies wird erleichtert, da jedes Objekt genau eine Zeile einnimmt und der verwendete Texteditor eine Durchnummerierung der Zeilen anbietet.

4.6.2 ID – Verteilung

Die Wichtigkeit des Attributes „id“ wurde schon mehrmals angedeutet. Mit Hilfe der in Kapitel 4.3.2 erläuterten Methode konnte jedes Objekt, welches für die spätere Einarbeitung der Interaktivitäten von Bedeutung ist, innerhalb des SVG – Quelltextes lokalisiert werden. Diesem wurde dann ein „id“ – Wert übergeben. Zum Beispiel war es notwendig, alle Objekte, die zur Darstellung der Buslinie 61 dienen, zu gruppieren und dieser Gruppe einen „id“ – Wert zu vergeben. Dies sieht folgendermaßen aus.

```
<g id="Buslinie61">...</g>
```

Einen „id“ – Wert erhielten außerdem alle Hauptgruppen (Nutzfläche, Wiese, TU – Gebäude,...) sowie alle *<path>* – Elemente, die ein TU – Gebäude oder ein anderes Gebäude von Bedeutung abbilden (z.B. *id=„tu01“* für Georg – Schumann – Bau).
Man kann natürlich auch die von Corel Draw 10 automatisch vergebenen „id“ – Werte beibehalten, allerdings sind diese weniger selbsterklärend und verwirren eher.

4.6.3 Signaturenfestlegung

SVG erlaubt es, innerhalb des *<defs>* – Elementes, am Anfang der Datei eine Signatur mit Hilfe des *<symbol>* – Elementes festzulegen, sodass diese durch den Aufruf des „id“ – Wertes beliebig oft genutzt werden kann. Dies erspart gerade bei sehr oft vorkommenden Signaturen (z.B. einer Baumsignatur) eine Menge Quelltext und somit Speicherplatz. Die Festlegung und der Aufruf sehen dann z.B. so aus.

```
<defs>
  <symbol id="baum" overflow="visible" class="fil2 str4">
    <circle cx="0" cy="0" r="42"/>
  </symbol>
</defs>

<use x="-8468" y="1020" xlink:href="#baum"/>
```

Dieses separate Anlegen der Signaturen wird aber beim Export nicht unterstützt, jede Signatur wird als Pfad beschrieben. Die gleiche Signatur hat nach dem Export folgendes Aussehen.

```
<path id="13522264" class="fil2 str4" d="M-8510 1020c0,-24 19,-43 43,-43 24,0 43,19
43,43 0,24 -19,43 -43,43 -24,0 -43,-19 -43,-43z"/>
```

Dabei werden auch Kreise nicht mit Hilfe des *<circle>* – Elementes beschrieben, sondern durch das <path> – Element.

Nun ist es nicht zwingend erforderlich alle Signaturen zu überarbeiten, soll jedoch eine Interaktion erfolgen, z.B. ein Vergrößern der Signatur beim Überfahren mit dem Mauszeiger, so ist eine allgemeine Festlegung ratsam. Mit ein wenig Aufwand, der hier aber nicht näher erläutert werden soll, können nachträglich diese allgemeinen Signaturenbeschreibungen erarbeitet werden. Die Datei *campus.svg* enthält insgesamt zehn allgemein beschriebene Signaturen. Diese dienen zur Darstellung der Bäume, aller Signaturen für zusätzliche Informationen und zur Darstellung der Straßenbahnen, Busse und der Haltestellen für eine Animation.

5 Erzeugung des interaktiven Prototyps

Voranstellend soll bemerkt werden, dass es nicht Ziel dieser Ausführungen ist, die kompletten JavaScript – Funktionen darzustellen. Diese kann man im Anhang ausführlich mit Kommentaren betrachten. Vielmehr sollen an dieser Stelle die grundlegendsten Funktionsprinzipien mit kurzen Auszügen der Funktion zur Geltung kommen.

5.1 Voraussetzungen

Nach der Ausführung der Arbeitsschritte, die in Kapitel 4 beschrieben wurden, müssten nun vier Dateien vorhanden sein. Diese wären die HTML – Datei (*campus.html*), in die das SVG – Dokument (*campus.svg*) mit Hilfe des <embed> – Elementes eingebettet worden ist, sowie die JavaScript – Datei (*script.js*) und die CSS – Datei (*formate.css*). Ist das SVG – Dokument außerdem überarbeitet, kann innerhalb der Scriptdatei mit der Schaffung von Funktionen begonnen werden, die auf Anwenderereignisse reagieren.
Auch die schon in Kapitel 4.4.1 erwähnte Funktion *initMap( )* dient dafür als Voraussetzung. Sie erstellt Referenzen zu den in der HTML – Datei eingebetteten SVG – Dokumenten über das Attribut „name“ im <embed> – Element. Konkret wurde die SVG – Karte wie folgt eingefügt.

```
<embed name="karte" src="svg/campus.svg" type="image/svg-xml">
```

Die Anweisung für die Referenz innerhalb der Funktion *initMap( )* ist dann so beschrieben.

```
var svgkarte = document.karte.getSVGDocument();
```

Des weiteren werden in dieser Funktion verschiedene andere Funktionen aufgerufen. Diese kontrollieren erstens die Auswahllisten zur Gebäude- und Studiengangsuche und setzten diese auf den Ausgangswert (*ausgangsuche( )*, *ausgangsuche2( )*). Zweitens kommt die Funktion *aufloesung( )* zum Einsatz, welche die jeweiligen Bildschirmauflösungen abfragt und dazu die Größen der SVG – Dokumente berechnet und übergibt.
Mittlerweile erfolgte schon des öfteren die Erwähnung von mehreren SVG – Dokumenten, obwohl nur die Schaffung der SVG – Karte näher erläutert wurde. Die anderen Graphiken, also der Titel, die Übersichtskarte, das Menü und die Maßstabsleiste entstanden eigenständig „per Hand“, d.h. ohne Zuhilfenahme von speziellen Programmen. Die Übersichtskarte wurde z.B. durch eine Verkleinerung und Ausdünnung der Originalkarte generiert. Auch die Erstellung dieser weiteren SVG – Dokumente gilt als Voraussetzung für die Schaffung des Prototypen, da Interaktionen des Anwenders z.B. auch über den Menüteil ausgelöst werden können.

5.2 „mouseover" – Effekte

5.2.1 Namensanzeige

Die Namensanzeige dient zur Darstellung der Gebäude- und Signaturnamen im Menüteil der Internetseite. Dazu wurde die Funktion *name( )* geschaffen, die durch die Event – Attribute *onmouseover* und *onmouseout* ausgelöst wird. Der Aufruf, z.B. an einem bestimmten Gebäude der TU, wurde direkt im entsprechenden <path> – Element vorgenommen und sieht folgendermaßen aus.

```
<path id="tu07" onmouseover="name('Zeuner-Bau')" onmouseout="name(' ')“ ... />
```

Beim ersten Aufruf der Funktion wird als Parameter der Name des Gebäudes übergeben. Dieser wird dann innerhalb der Funktion *name( )* mit Hilfe der Methode *setData( )* an ein Textelement im SVG – Dokument *menue.svg* weiter gereicht und kommt dort zur Anzeige. Beim zweiten Aufruf der Funktion werden die gleichen Anweisungen durchlaufen, nur das es sich bei dem übergebenen Parameter nicht um den Namen des Gebäudes handelt, sondern um ein leeres Textfeld, was die Anzeige keines Namens verursacht.

Abbildung 7: Namensanzeige und Farbänderung beim Überfahren des Gebäudes mit dem Mauszeiger (Ausschnitt eines Screenshot der Internetseite).

5.2.2 Farbänderung

Um beim Überfahren des Mauszeigers über ein TU – Gebäude einen Farbwechsel zur besseren Hervorhebung zu erzielen, wurde die Funktion *farbe( )* angelegt. Sie wird mit den selben Event – Attributen wie die Funktion *name( )* ausgelöst.

```
<path id="tu07" onmouseover="name('ZeunerBau');farbe('tu07');"
onmouseout="name('');farbe('tu07')" ... />
```

Durch den Aufruf der Funktion wird ein Parameter an diese übergeben, der dem Wert des Attributes „id“ entspricht. Deshalb kann innerhalb der Funktion durch die Methode

getElementById( ) eine Referenz zum entsprechenden Objekt hergestellt werden. Beim erstmaligen Durchlaufen der Anweisungen wird durch den Einsatz der Methode *getAttribute( )* die aktuelle Farbe in einer Variablen gespeichert und durch *setAttribute( )* eine neue Farbe zugewiesen. Nach dem zweiten Aufruf der Funktion wird dem Objekt wieder die alte Farbe mit *setAttribute( )* übergeben.

5.3 Ebenensteuerung

Hinter dem Begriff Ebenensteuerung verbirgt sich nichts anderes, als das Setzen der Eigenschaft Sichtbarkeit („visibility“) einer Gruppe von Objekten auf den Wert sichtbar („visible“) oder auf den Wert unsichtbar („hidden“). Dies erfolgt mit der Funktion *anzeigen( )*, welche über das Event – Attribut *onclick* aufgerufen wird. Das Steuern der einzelnen Ebenen bzw. Gruppen wird jedoch über das SVG – Dokument *menue.svg* in der rechten unteren Hälfte der Internetseite ermöglicht, sodass der Aufruf der Funktion aus diesem Teil erfolgen muss. Genauer gesagt wird der Eintrag in dem entsprechenden *<use>* – Element vorgenommen, welches zur Anzeige einer festgelegten Signatur dient (wie in Kapitel 4.6.3 beschrieben).

```
<use id="s4" x="600" y="1100" onclick="anzeigen(evt,'1')" xlink:href="#bibo"/>
```

Klickt der Nutzer also auf die gewünschte Signatur im Menü, startet die Funktion *anzeigen( )*. Dabei werden zwei Parameter übergeben. Der erste Parameter dient in Verbindung mit den Methoden *getTarget( )* und *getAttribute( )* zur Ermittlung des Objektes und dessen „id“ – Wertes, auf das geklickt wurde. Der zweite Parameter sagt an, ob die Gruppe sichtbar (Wert = 1) oder unsichtbar (Wert = 2) werden soll. Nachdem der „id“ – Wert des angeklickten Objektes ermittelt ist, kann dieser Wert genutzt werden, um eine Referenz zur Gruppe der Signaturen dieser Art in der SVG – Karte zu erstellen. Im obigen Beispiel wäre das die Gruppe im Dokument *campus.svg* mit dem „id“ – Wert „s4“, welche alle Signaturen der Bibliotheken innerhalb der Karte enthält. Nun muss nur noch mit Hilfe der Methoden *getStyle( )* und *setProperty( )* die Eigenschaft „visibility“ je nach Wert des zweiten Parameters auf „visible“ oder „hidden“ gesetzt werden.

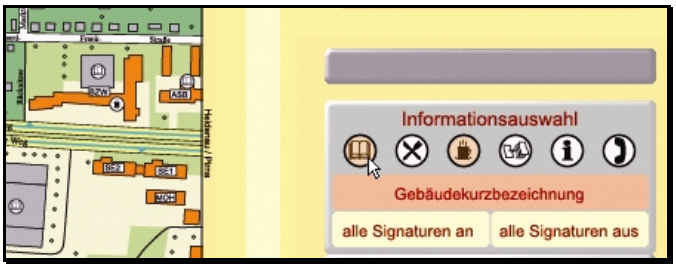

Abbildung 8: Steuerung der Signaturebenen im Menüteil (Ausschnitt eines Screenshot der Internetseite).

Mit der Funktion *anzeigen( )* ist dem Anwender eine Möglichkeit geschaffen worden, die ihm erlaubt, selbständig über die Inhaltsdichte des Kartenbildes zu entscheiden. Er kann außerdem alle im Menü angebotenen Signaturen gleichzeitig ein- oder ausschalten. Dies gestattet die Funktion *alle( )*, die im Grundprinzip gleich abläuft und auch über das Event – Attribut *onclick* gestartet wird.

5.4 Navigation

Ziel dieser Arbeit war es unter anderem, dem Nutzer verschiedene Möglichkeiten der Navigation zu erlauben. Speziell wurde dabei an eine Auswahl des Kartenausschnittes über Ein- und Auszoomen, über die Verschiebung eines Rechteckes in einer Übersichtskarte und über Richtungspfeile gedacht.

5.4.1 Zoomfunktion

Die Möglichkeit des Ein- und Auszoomens wird über ein Pulldown – Menü in der HTML – Datei und nicht über die vom SVG – Viewer gegebene Anwendung gesteuert. Dabei kann der Nutzer zwischen vier verschiedenen Zoomstufen wählen. Hat dieser sich entschieden, wird die Funktion *zoom( )* aufgerufen, die zuerst durch die Anweisung:

```
zoomVal = parseFloat(selectZoomVal.Zoomstufe.value);
```

den Wert ermittelt, der innerhalb des Pulldown – Menüs im HTML – Quelltext zu jeder Zoomstufe festgelegt wurde und den Zoomfaktor darstellt. Die Methode *parseFloat( )* wandelt dabei den Wert des Typs „String“ in eine Gleitkommazahl um. Im weiteren Verlauf der Funktion werden dann die Werte des Attributes „viewBox“ im <svg> – Element der Karte berechnet, da diese den angezeigten Kartenausschnitt festlegen. Bei den Werten handelt es sich um die Eckkoordinate der linken oberen Ecke sowie um die Breite und Höhe des Ausschnittes. Eine Besonderheit der Berechnung ist allerdings, dass anfangs nur mit dem teilweise transparenten Rechteck der Übersichtskarte gerechnet wird. Dabei wird außerdem die Funktion *kontrolle( )* aufgerufen, in der in mehreren „if“ – Schleifen überprüft wird, ob die Werte des neuen Ausschnittes über den Kartenrand hinaus gehen. Ist dies der Fall, wird der Ausschnitt so gesetzt, dass die äußere Kante genau am Kartenrand liegt. Erst nach diesen Kontrollen und den daraus mit der Methode *setAttribute( )* getroffenen Festlegungen der Position des Rechteckes in der Übersichtskarte erfolgt die Übergabe der neuen Werte an das Attribut „viewbox“ in der SVG – Karte mit Hilfe der Funktion *PanEnde( )*, auf die im nächsten Kapitel noch näher eingegangen wird. Durch diese Vorgehensweise wird also gleichzeitig der neue Kartenausschnitt in Form des teilweise transparenten Rechteckes in der Übersichtskarte verdeutlicht und die Darstellung der eigentlichen Karte angepasst.

Da die Funktion *zoom( )* aber auch in anderen Funktionen aufgerufen wird (z.B. aus der Legendenfunktion) und somit nicht über die Auswahl, wird beim Aufruf dieser ein Parameter übergeben. Dieser gibt an, ob der Wert der gewünschten Zoomstufe aus der Auswahl genommen werden soll oder ob dieser festgelegt ist. Der Aufruf von *zoom( )* in der Legendenfunktion heißt z.B. *zoom(2)*, wobei die Zahl 2 für eine Zoomstufe von 100 % steht und somit der Originalansicht entspricht. Außerdem werden innerhalb dieser Funktion weitere Funktionen aufgerufen, die zum einen die Anzeige der Maßstabsleiste zur Zoomstufe anpassen und zum anderen die Anzeige des Hinweises unterhalb der Übersichtskarte steuern.
Wie schon beschrieben, stehen dem Anwender im Prototyp vier Zoomstufen zur Verfügung. Bei diesen handelt es sich konkret um die Originalansicht (100%) sowie um die Stufen 200%, 400% und 600%. Ein weiteres Auszoomen, z.B. auf 50%, erschien nicht sinnvoll, da in diesem Fall die einzelnen Objekte nur noch schwer erkennbar sind und zum Teil ineinander verlaufen. Eine weitere Vergrößerung erschien ebenfalls als ungünstig, da sie weder mehr Details hervorbringt noch der Überschaubarkeit dient.

5.4.2 Bewegungsfunktionen

Bei den Funktionen, die das Verschieben des Kartenausschnittes erlauben, muss man zwei Möglichkeiten unterscheiden. Dies wäre erstens die Variante, den Kartenausschnitt durch anklicken der entsprechenden Richtungspfeile zu verschieben. Zweitens kann aber auch eine Verschiebung erfolgen, indem das teilweise transparente Rechteck der Übersichtskarte durch anklicken und gedrückt lassen der Maustaste an die gewünschte Position geschoben wird.

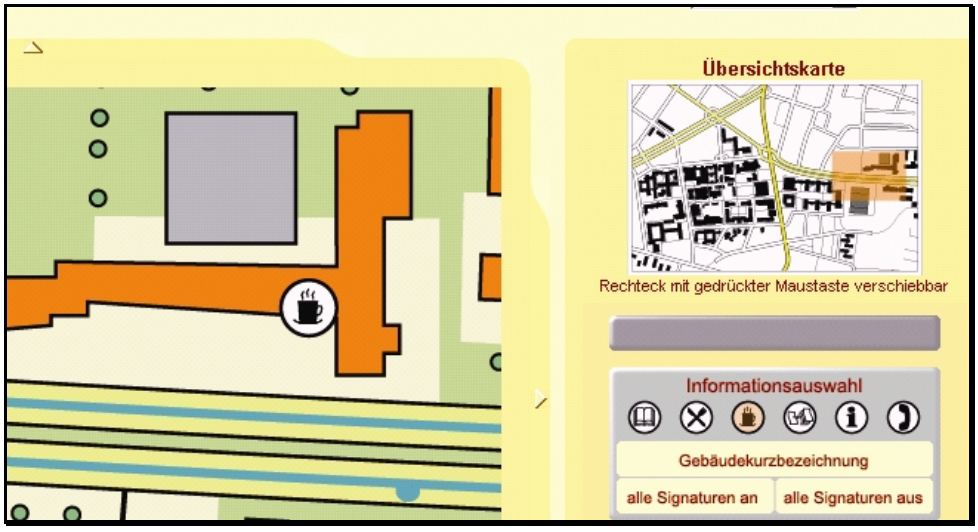

Abbildung 9: Bestimmung des Kartenausschnitt mittels Richtungspfeile oder Übersichtskarte (Ausschnitt eines Screenshot der Internetseite).

Um über die Richtungspfeile navigieren zu können, wurde die Funktion *bewegen( )* geschaffen. Da es sich bei den Pfeilen um GIF – Graphiken (**G**raphics **I**nterchange **F**ormat) in der HTML – Datei handelt, wird diese Funktion durch den Event – Handler *onClick* im HTML – Dokument ausgelöst. Als Parameter werden dabei die Verschiebungswerte in x – und in y – Richtung übergeben. Die Berechnung des neuen Kartenausschnittes erfolgt durch Addieren der Verschiebungsrichtungen zu den Eckkoordinaten das alten Ausschnittes. Wie schon im vorherigen Kapitel wird aber anfangs erst das Rechteck der Übersichtskarte neu gesetzt und die Funktion *kontrolle( )* aufgerufen, um ein Bewegen über den Kartenrand hinaus zu vermeiden. Erst danach erfolgt die Übergabe der Werte an die Karte mittels der Funktion *PanEnde( )*.
Das Navigieren über die Übersichtskarte wird im wesentlichen durch drei Funktionen ermöglicht, unter anderem durch die schon erwähnte Funktion *PanEnde( )*. Es sind drei Funktionen erforderlich, da auf drei verschiedene Nutzerhandlungen reagiert werden muss. Die erste Handlung des Nutzers ist das Anklicken des Rechteckes in der Übersichtskarte, welches durch das Event – Attribut *onmousedown* registriert wird und die Funktion *PanStart( )* auslöst. Die zweite Handlung ist das Verschieben des Rechteckes mit gedrückter Maustaste an die gewünschte Position. Dieses Ereignis wird durch das Event – Attribut *onmousemove* registriert und löst die Funktion *PanVerlauf( )* aus. Das letzte Ereignis registriert das Loslassen der Maustaste und somit das Ende der Verschiebung mit Hilfe des Event – Attributes *onmouseup*. Dabei wird die Funktion *PanEnde( )* aufgerufen. Da die ganze Zeit mit dem Rechteck der Übersichtskarte gearbeitet wird, werden die genannten Event – Attribute innerhalb des *<rect>* – Elementes im SVG – Dokument *klein.svg* notiert.

```
<rect ... onmousedown="PanStart(evt)" onmousemove="PanVerlauf(evt)"
onmouseup="PanEnde()"/>
```

Der Ablauf der Funktionen ist vom Prinzip her einfach, da nur mit Hilfe der Methoden *getAttribute( )* und *setAttribute( )* immer wieder auf die Attribute „x“, „y“, „width“ und „height“ des Rechteckes der Übersichtskarte zugegriffen wird. Die Abfrage der Attribute geschieht in der Funktion *PanStart( )* zum ersten Mal. Aus dem Codefragment wird allerdings auch ersichtlich, dass die ersten beiden Funktionen den Parameter „evt“ übergeben. Durch diesen Parameter und den Methoden *getClientX( )* und *getClientY( )* kann im nächsten Schritt die Position des Mauszeigers über der Übersichtskarte bestimmt werden. In der Funktion *PanVerlauf( )* erfolgt diese Positionsabfrage zu Beginn erneut. Aus der Verschiebung des Rechtecks ergeben sich dann Differenzen der Zeigerposition zur vorherigen Position. Diese Differenzen werden zur Eckkoordinate des Rechteckes, also den Attributen „x“ und „y“ addiert, wobei mit der Funktion *kontrolle( )* das Erreichen des Kartenrandes geprüft wird und die neuen Werte an das Rechteck übergeben werden. Am Ende der Funktion wird wieder die neue Position des Mauszeigers abgefragt. Da die Funktion *PanVerlauf( )* während der Verschiebung wiederholt durchläuft,

bewegt sich das Rechteck mit dem Mauszeiger über die Übersichtskarte. Hat der Anwender die gewünschte Position erreicht, lässt er die Maustaste los und die Funktion *PanEnde( )* wird gestartet. Dabei kommt es noch einmal zur Abfrage der eingangs erwähnten Attribute und schließlich zur Übergabe der Werte an das Attribut „viewbox“ der SVG – Karte.

```
eckex = parseFloat(svgrechteck.getAttribute("x"));
eckey = parseFloat(svgrechteck.getAttribute("y"));
width = parseFloat(svgrechteck.getAttribute("width"));
height = parseFloat(svgrechteck.getAttribute("height"));
viewBoxneu = eckex + " " + eckey + " " + width + " " + height;
svgMainViewport.setAttribute("viewBox",viewBoxneu);
```

Damit wird eine Anzeige des vom Rechteck in der Übersichtskarte überdeckten Bereiches im eigentlichen Kartenbild erreicht.

5.5 Suchfunktionen

Mit den Suchfunktionen stehen dem Anwender zwei weitere Interaktionsmöglichkeiten zur Verfügung. Zum einen kann nach einem Gebäude der TU gesucht werden, zum anderen nach dem Standort der Studienberatung eines bestimmten Studienganges. Die Auswahl erfolgt dabei wieder über Pulldown – Menüs in der HTML – Datei. Hat der Anwender eine Auswahl getroffen, wird durch den Event – Handler *onChange* die Funktion *suchen( )* gestartet. Da beide Suchmöglichkeiten mit der gleichen Funktion arbeiten, gibt ein übergebener Parameter Auskunft darüber, durch welche Suchmöglichkeit die Funktion ausgelöst wurde. Mit der Anweisung:

```
var id = document.suche.suchid.value;
```

kann dann der Wert ermittelt werden, der innerhalb des Pulldown – Menüs im HTML – Quelltext zu jedem Gebäude oder Studiengang festgelegt wurde. Für den Zeuner – Bau heißt der Wert z.B. "tu07zeu". Die Schwierigkeit bei der Funktion *suchen( )* besteht darin, dass zwei Objekte innerhalb der SVG – Datei ermittelt werden müssen. Deshalb besteht der ermittelte Wert auch aus zwei Bestandteilen, die durch die Methode *substr( )* zu zwei Variablen geteilt werden. Für den Zeuner – Bau ergibt das eine Variable mit dem Wert „tu07“ und eine mit dem Wert „zeu“. Mit Hilfe der ersten Variablen wird über die Methode *getElementById( )* das gesuchte Gebäude im SVG – Dokument ausfindig gemacht und farbig betont. Die Anweisungen entsprechen dabei denen der Funktion *farbe( )*. Es erfolgt aber nicht nur eine farbliche Hervorhebung des gesuchten Objektes, sondern auch eine eingezoomte Darstellung desselben im Zentrum des Kartenausschnittes. Das Einzoomen erfolgt automatisch durch den Aufruf der Funktion *zoom( )*. Für die zentrierte Darstellung sind allerdings Koordinaten notwendig, die

einen Punkt im Zentrum des Gebäudes charakterisieren. Diese Koordinaten liefert die Schrift der Signaturen für die Gebäudekurzbezeichnungen, da sich diese in der Regel im Zentrum des jeweiligen Hauses befinden. Durch die zweite Variable und der Methode *getElementById( )* wird also die dazu gehörige Signaturschrift referenziert, welche für das Beispiel folgendermaßen aussieht.

```
<text id="zeu" x="-5907" y="6493" class="fil9 fnt0">ZEU</text>
```

Im Anschluss daran, kann über die Methode *getAttribute( )* die x – und y – Koordinate abgefragt werden. Sie bilden die Grundlage für die Festlegung des Kartenausschnittes nach dem Prinzip der Bewegungsfunktionen. Somit wird dem Nutzer das von ihm gesuchte Gebäude gleichzeitig farbig betont und vergrößert in der Mitte des Kartenausschnittes dargestellt.

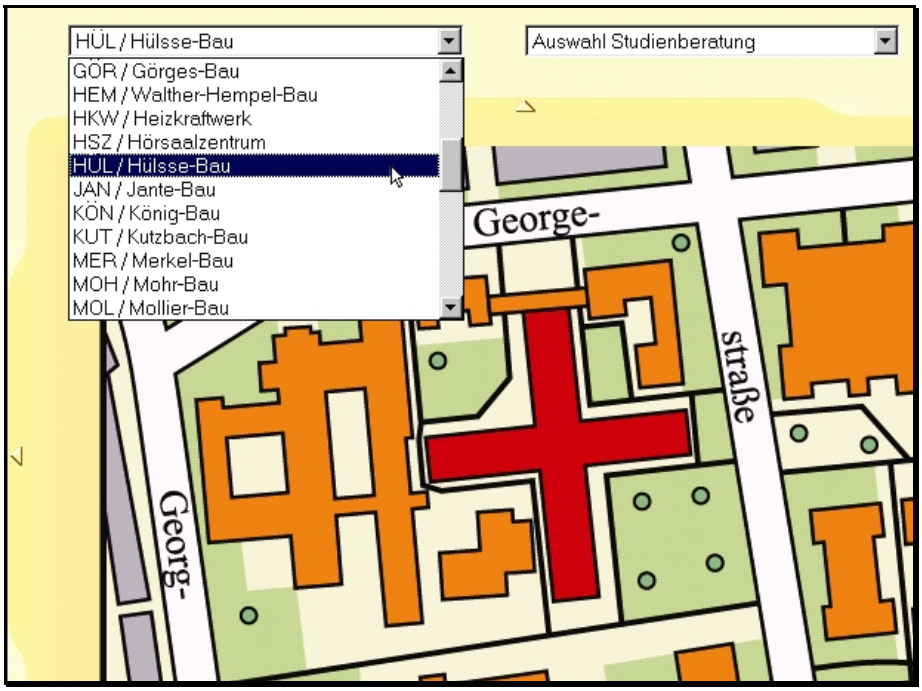

Abbildung 10: Auswahl und Anzeige eines gesuchten Gebäudes (Ausschnitt eines Screenshot der Internetseite).

Um Suchergebnisse wieder aufzuheben, wurde außerdem die Funktion *ausgang( )* programmiert, die zum einen dem ehemals gesuchten Gebäude die alte Farbe wieder zuweist und die zum anderen den im Pulldown – Menü aufgeführten Text wieder in die Ausgangslage bringt.

5.6 „onclick" – Effekte / Verlinkung

In Kapitel 3.2.3 wurde bereits die herkömmliche Verlinkung von SVG – Objekten mit anderen Internetseiten besprochen. Im Prinzip handelt es sich dort auch um einen „onclick" – Effekt, da die neue Seite durch das Anklicken des Objektes aufgerufen wird. Es kommt jedoch nicht zum Einsatz des Event – Attributes *onclick*, welches allerdings in der folgenden Interaktionsmöglichkeit verwendet wird. Bei dieser Anwendung wird nach dem Anklicken eines Gebäudes (z.B. Hülsse – Bau) oder einer Einrichtung (z.B. Cafeteria) ein separates Fenster geöffnet. Dieses enthält entweder eine Auflistung der Studiengänge, deren Studienberatung sich im Gebäude befindet und Verweise zu den Instituten im Gebäude oder das Fenster enthält nur Informationen zu einer Einrichtung, z.B. die Öffnungszeiten einer Cafeteria.

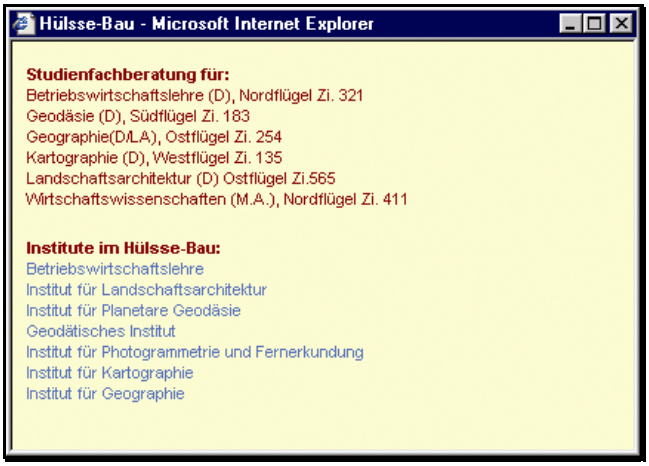

Abbildung 11: das nach dem Klick auf ein Gebäude geöffnete Informationsfenster.

Für die Darstellung ist dabei entweder die Funktion *linkanzeige( )* oder *infoanzeige( )* verantwortlich. Beide Funktionen enthalten in etwa die gleichen Anweisungen und werden innerhalb der Karte durch das Event – Attribut *onclick* ausgelöst. Sie unterscheiden sich nur in der Übergabe von Parametern und im Aufbau des neuen Fensters.
Beim Aufruf der Funktion *linkanzeige( )* werden zwei Parameter übergeben. Der erste dient zur Erstellung einer Referenz zum angeklickten Gebäude und zur Kontrolle, ob dieses gerade ein Suchergebnis darstellt. Damit soll erreicht werden, dass nach einer Suche ein noch aktuelles Ergebnis aufgehoben wird, wenn nicht auf dieses Gebäude geklickt wurde. Der zweite Parameter stellt den Namen des Arrays dar, dass die Informationen und Links zum Gebäude enthält. Mit Hilfe der Anweisung:

```
var win=window.open("","Links","width=420,height=275,left=100,top=100, ... ");
```

kann dann ein Fenster mit einer vorgeschriebenen Größe und Position geöffnet werden. Da nicht zu jedem Gebäude extra eine HTML – Datei existieren soll, wird diese innerhalb der JavaScript – Datei für die Dauer der Anzeige geschaffen. Dazu kommt die Anweisung:

```
win.document.write( );
```

zum Einsatz, in der die notwendigen HTML – Elemente notiert werden. Gleichzeitig liefert eine Schleife die Inhalte des Arrays, um sie ebenfalls über die zuletzt genannte Anweisung darzustellen. Der Unterschied zur Funktion *infoanzeige( )* besteht nun darin, dass erstens nur der zweite Parameter beim Aufruf übergeben werden muss, da eine Einrichtung kein Ergebnis einer Suche sein kann. Zweitens sind andere HTML – Elemente erforderlich, da diese Funktion nur einfache Informationen und keine Hyperlinks darstellt.

5.7 Entfernungsmessung

Die Möglichkeit, innerhalb der Karte Entfernungen zu messen, stellt die letzte Interaktivität dar, die hier näher erläutert werden soll. Sie erlaubt dem Anwender in der Karte einen Weg durch anklicken der einzelnen Wegpunkte zu markieren, wobei gleichzeitig die Entfernung zum Ausgangspunkt angezeigt wird. Dazu wurden die vier Funktionen *entfernung( )*, *koord( )*, *punktweg( )* und *aufheben( )* geschaffen. Zur Entfernungsmessung an sich sind allerdings nur die ersten beiden Funktionen zuständig. Die Funktion *punktweg( )* erlaubt die Löschung des letzten gesetzten Wegpunktes, die Funktion *aufheben( )* löscht den gesamten Weg.
Mit Hilfe von *entfernung( )* wird zu Beginn die Entfernungsmessung durch das Anklicken des entsprechenden Buttons im Menüteil aktiviert. Dabei wird automatisch auf die Originalansicht gezoomt und nach dem Prinzip der Ebenensteuerung die Gruppe „Messung“ sichtbar gemacht. Sie enthält eine „versteckte“ Kreissignatur, auf die später noch eingegangen wird und ein völlig transparentes Rechteck, welches das gesamte Kartenbild überdeckt. Das hat zur Folge, dass alle anderen Funktionen in der Karte nicht mehr ausgelöst werden können. Der wichtigste Teil der Funktion besteht darin, dass der Wert der Variablen „messstatus“ auf eins gesetzt wird. Dies ist deshalb von großer Wichtigkeit, da die Funktion *koord( )* mit Hilfe des Event – Attributes *onclick* im *<svg>* – Element der Karte ausgelöst wird, also bei jedem Klick auf das Kartenbild. Die Variable „messstatus“ verhindert aber, dass die Anweisungen von *koord( )* auch abgearbeitet werden, wenn der Nutzer für zusätzliche Informationen ein Gebäude angeklickt hat. Nur wenn *messstatus = 1;* angewiesen wurde, kommt es zum vollständigen Durchlauf der Funktion. Mit dem Aufruf von *koord( )* wird der Parameter „evt“ übergeben. Durch diesen und die Methoden *getClientX( )* und *getClientY( )* wird die Position des Mauszeigers über der Karte bestimmt. An diese Position muss nun ein Wegpunkt gesetzt werden. Dazu wurde bereits eine

Kreissignatur festgelegt, die sich allerdings außerhalb des Kartenbildes befindet, sozusagen versteckt ist. Mit den Anweisungen:

```
var element = svgkarte.getElementById("messpunkt");
var neuPunkt = element.cloneNode(false);
```

wird diese versteckte Signatur im SVG – Quelltext lokalisiert und geklont. Danach erfolgt die Zuweisung der ermittelten Zeigerposition mit der Methode *setAttribute( )*. Durch:

```
var eltern = svgkarte.getElementById("Messung");
eltern.appendChild(neuPunkt);
```

wird der neue Punkt in die Gruppe „Messung“ eingefügt und somit sichtbar gemacht. Auf diese Weise wird mit jedem Klick des Nutzers auf die Karte mit dem Setzten eines Wegpunktes reagiert, wobei immer die Koordinaten des neuen und des vorhergehenden Punktes bekannt sind. Deshalb können nach dem Setzen des zweiten Punktes, die beiden Punkte mit einer farbigen Linie verbunden werden. Sie entsteht durch die Anweisung:

```
var wegstueck = svgkarte.createElement('line');
```

und erhält ihre Anfangs- und Endpunktkoordinaten über die Methode *setAttribute( )*. Durch das Wissen der Koordinaten kann außerdem die Länge der Strecke in Bildpunkten berechnet werden, die wiederum über eine Referenzzahl in Meter umgerechnet wird. Die Länge der Gesamtstrecke ergibt sich mit der Addition der einzelnen Streckenlängen.

Abbildung 12: in der Karte dargestellter und gemessener Weg (Ausschnitt eines Screenshot der Internetseite).

Wie schon angesprochen, dienen die Funktionen *punktweg( )* und *aufheben( )* zum Löschen einzelner Wegpunkte oder zum Löschen der gesamten Strecke. Die entscheidende Anweisung bildet dabei die Methode *removeChild( )*. Wie der Namen schon sagt, können mit ihr einzelne Elemente für die Dauer der Darstellung aus dem Quelltext entfernt werden.

5.8 Weitere Interaktivitäten

Nach diesen Erläuterungen einzelner Interaktivitäten und deren Umsetzung sollen im folgenden Abschnitt noch weitere Funktionen erwähnt werden, die dem Anwender eine Interaktion erlauben. Jedoch wird auf eine ausführliche Erklärung der Funktionsweise verzichtet.
Zur Darstellung der Legende kommt es bei dieser Karte zur Anwendung einer völlig neuen Variante. Dabei wurde auf eine separate Legendenanzeige im traditionellen Sinne verzichtet. Der Anwender entscheidet vielmehr selbst, zu welchem Objekt er sich die Legenden-bezeichnung anzeigen lassen möchte. Dazu fährt er mit dem Mauszeiger über die Karte und erhält in einem transparenten Rechteck, welches sich mit dem Zeiger bewegt, die jeweilige Bezeichnung. Die dafür notwendigen Funktionen heißen *legende( )* und *laufen( )*. Die Legendenfunktion wird im Menüteil der Internetseite durch das Klicken auf den entsprechenden Button ausgelöst.
Überfährt der Anwender eine Signatur in der Karte, so wird diese für diesen Moment vergrößert dargestellt. Diese Interaktion erfolgt aber nur in den Zoomstufe „Originalansicht“ und „200%“. Für diese Möglichkeit wurde die Funktion groesser( ) geschaffen, deren Grundprinzip einem Beispiel der Carto:net – Internetseite [Carto:net, 2002] entnommen werden konnte. Dadurch ist es selbst in den kleineren Maßstäben möglich, ohne Probleme die Signaturen zu erkennen.
Weiterhin wurden Funktionen erstellt, die dem Anwender zu den meisten Aktionen ein entsprechendes Hinweistextfeld im Menüteil liefern. Die Steuerung der Informationsanzeige erfolgt dabei durch die Funktionen *hinweisan( )* und *hinweisaus( )*. Diese Hilfe soll dem Nutzer in kurzen Stichworten die Möglichkeiten der Internetseite näher bringen.

5.9 Kopplung von Interaktivität, Dynamik und Animation zur Detaildarstellung

Die Idee dieser Interaktionsmöglichkeit war es, eine bessere Verdeutlichung der verschiedenen Bus- und Straßenbahnlinien sowie der dazu gehörigen Haltestellen zu erreichen. Dies sollte durch eine Hervorhebung der Liniensignatur, durch spezielle Haltestellensignaturen und durch eine Animation entlang eines Pfades erreicht werden. Mit der Umwandlung dieser Zielsetzung wurde schnell deutlich, dass dafür eine dynamische Änderung des Kartenbildes, eine mit SVG erstellte Animation und eine Interaktion des Nutzers notwendig sind.
Der erste Schritt erforderte eine Vielzahl von Arbeiten innerhalb des SVG – Quelltextes, unter anderem das Lokalisieren und Gruppieren aller Objekte einer Bus- oder Straßenbahnlinie sowie das Einfügen einer Gruppe am Ende des Dokumentes zur Darstellung der Details. Ausgelöst wird die für die Detaildarstellung verantwortliche Funktion *detail( )* mit Hilfe des Event – Attributes *onclick*, welches im Gruppenelement jeder Linie notiert wurde.

```
<g id="Buslinie61" onclick="detail(evt,61)" ... >
```

Mit dem Aufruf wird die am Ende des Quelltextes stehende Gruppe „liniendetail“ sichtbar gemacht. Sie enthält ein teilweise transparentes Rechteck, welches das gesamte Kartenbild überdeckt, die Festlegungen für den Button zum Ausblenden der Details und weitere Gruppen, von denen je eine zu einer Bus- oder Straßenbahnlinie gehört.

```
<g id="liniendetail" style="visibility:hidden">
...
  <g id="61" style="visibility:hidden"> ... </g>
  <g id="3" style="visibility:hidden"> ... </g>
  <g id="72" style="visibility:hidden"> ... </g>
  <g id="8" style="visibility:hidden"> ... </g>
</g>
```

Durch den ersten übergebenen Parameter wird im Anschluss die Gruppe referenziert, die alle Elemente der angeklickten Linie enthält, im Beispiel also die Gruppe mit dem „id“ – Wert „Buslinie61“. Diese Gruppe wird mit der Methode *cloneNode( )* geklont und in die Gruppe eingefügt, deren „id“ – Wert dem des zweiten übergebenen Parameter entspricht. Im Beispiel also die Gruppe mit dem „id“ – Wert „61“. Somit erscheint eine Kopie der darunter liegenden Linie über dem teilweise transparenten Rechteck, welches aber das gesamte Umfeld farblich zurück setzt.

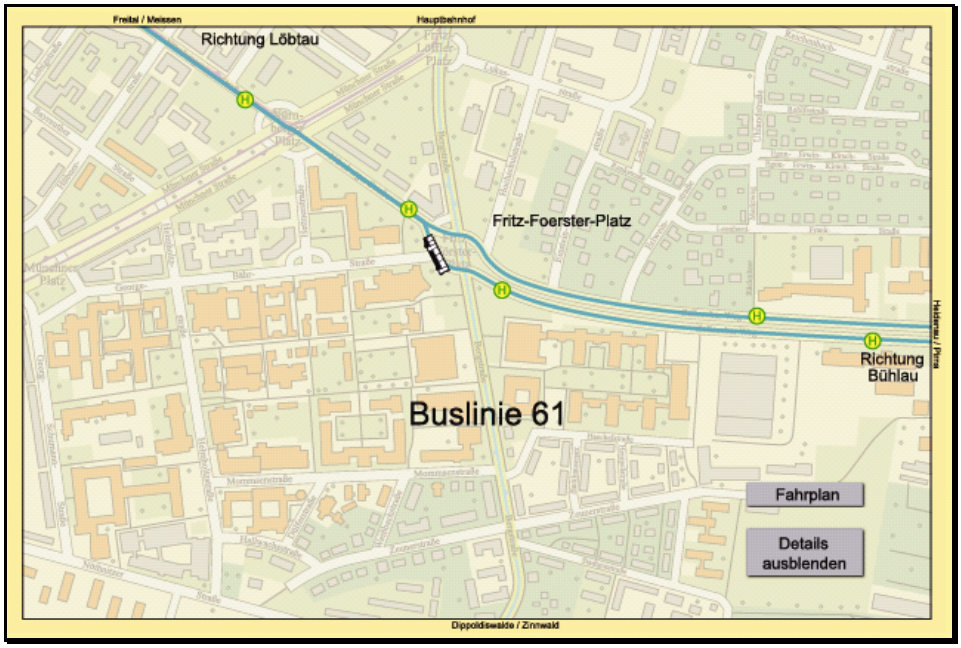

Abbildung 13: Darstellung und Animation einer Buslinie (Ausschnitt eines Screenshot der Internetseite).

Das „style“ – Attribut in der Untergruppe zeigt aber an, dass diese noch nicht sichtbar ist. Deshalb kommt noch die Methode *setProperty( )* zum Einsatz. Durch das Sichtbarmachen der jeweiligen Untergruppe kommen gleichzeitig die Haltestellensignaturen und das animierte Fahrzeug zum Vorschein, da diese Objekte in der Gruppe festgelegt sind.
Die Animation selbst beginnt durch das Animation – Event *click* mit dem Klick des Anwenders auf die entsprechende Linie. Durch das Anklicken des entsprechend beschrifteten Buttons kann dann die Animation über die Funktion *detailaus( )* abgebrochen werden. Wird dieser Button nicht gedrückt, endet sie nach der fünften Wiederholung. Dabei kommt das speziell für Animationen vorhandene Event – Attribut *onend* zum Einsatz, welches ebenfalls *detailaus( )* auslöst. Diese Funktion setzt dann alle zu Beginn sichtbar gemachten Gruppen wieder auf die „style“ – Eigenschaft „hidden“, also unsichtbar und beendet damit die Detailansicht.
Anhand dieses Beispiels wird sehr gut das schon in Kapitel 2.5 beschriebene Zusammenspiel der einzelnen Interaktionsgrundlagen deutlich.

6 Nachträgliche Arbeiten

6.1 Fallbehandlungen in der JavaScript – Datei

Bei der Vielzahl, der im vorangehenden Kapitel erläuterten Funktionen, wird schnell klar, dass immer wieder Abfragen in der JavaScript – Datei gestartet werden müssen, die prüfen, ob bereits eine andere Funktion in der Verwendung ist. Dies hat zur Folge, dass Funktionen teilweise nur gestartet werden können, wenn eine andere Anwendung beendet wurde. So kann z.B. keine Entfernungsmessung gestartet werden, wenn die Legendenfunktion noch aktiv ist. Die Funktionen *blinken( )* und *warnung( )* wurden deshalb geschaffen, um den Anwender durch ein Aufblinken des entsprechenden Buttons auf diese Tatsache hinzuweisen. Die Abfragen zur Kontrolle wurden durch „if“ – Anweisungen realisiert, wobei die meisten Funktionen eine „Statusvariable“ erhielten, deren Wert den Zustand der Funktion (aktiv oder inaktiv) beschreibt. Viel Zeit und Mühe musste dabei in die verschiedenen Fallbehandlungen investiert werden. Um dies zu verdeutlichen ein Beispiel:
Hat der Anwender eine Suche durchgeführt, kann er, was das Suchergebnis aufheben würde, auf ein anderes Gebäude oder eine Signatur klicken, eine neue Suche durchführen, den Kartenausschnitt verschieben, die Legendenfunktion oder Entfernungsmessung aktivieren. Oder er kann, was das Suchergebnis nicht aufheben würde, auf das gesuchte Gebäude klicken, auszoomen, Signaturen ein- und ausschalten oder die Vorstufe der Detailanzeige für Bus- oder Straßenbahnlinien im Menüteil aktivieren, welche nur die verschiedenen Linien aufblinken lässt. Klickt er allerdings auf eine Linie im Kartenbild, wird das Suchergebnis aufgehoben.
Für all diese Fälle sind zu Beginn der einzelnen Funktionen „if“ – Anweisungen notiert, auf die aber auch nicht verzichtet werden kann. Man muss jedoch bedenken, dass eine gewisse Logik dahinter stehen sollte, um den Anwender nicht völlig zu verwirren.

6.2 Schriftfreistellung

Das Kapitel Schriftfreistellung beschreibt einen Arbeitsschritt, der eigentlich nicht mit der Einarbeitung von Interaktivitäten zusammen hängt. Er soll aber hier der Vollständigkeit halber erläutert werden, da die Schriftfreistellung der Originaldatei beim Export verloren geht und dieses Problem noch nicht besprochen wurde.
SVG stellt eine Vielzahl von Filtereffekten zur Verfügung, mit denen das Aussehen von Objekten und somit auch die Schrift manipuliert werden kann. Der für die Freistellung interessante Filter ist dabei der Typ *<feMorphology>*. Er enthält das Attribut „operator“, welches entweder den Wert „erode“ (zur Verdünnung) oder „dilate“ (zur Verdickung) besitzen kann und sieht folgendermaßen aus.

```
<filter id="filter1">
  <feMorphology in="SourceGraphic" operator="dilate" radius="20"/>
</filter>
```

Auf eine Kopie des Schriftzuges wird dieser Filter angewandt. Wichtig ist dabei, dass die Kopie im SVG – Quelltext vor dem Originalschriftzug platziert wird. Somit entsteht ein dilatierter, also vergrößerter Schriftzug, der sich aber genau unter dem Original befindet. Wird diesem nun die gewünschte Farbe zugewiesen, erfolgt eine Freistellung des Originaltextes. Wirklich sinnvoll ist diese Art der Freistellung allerdings nur, wenn das ganze Wort in der gleichen Farbe freigestellt werden soll. Eine Freistellung einzelner Buchstaben hätte die Zerlegung des Wortes zur Folge und wäre sehr aufwendig. Außerdem würde dadurch die von SVG ermöglichte Textsuche zu keinem sinnvollen Ergebnis kommen.
Die Schriftfreistellung ganzer Wörter wurde, wie hier beschrieben, an den Platznamen der SVG – Karte angewandt. Als Freistellungsfarbe ist dabei die Farbe des Kartenhintergrundes festgelegt.

6.3 Nachbearbeitung der SVG – Datei

Die letzten Arbeiten am Prototyp beziehen sich nur noch auf die Verringerung des Speicherplatzes. Da jedes einzelne Textzeichen und somit auch Leerzeichen Speicherplatz benötigt, ist es sinnvoll, die Strukturierung der SVG – Dokumente zu „begradigen“. Dies erfolgte auch in dieser Arbeit. Allerdings wurde die Strukturierung im Quelltext des Anhangs zum besseren Verständnis beibehalten. Wie bereits mehrfach angesprochen, erhält jedes SVG – Element durch den Export über Corel Draw 10 ein Attribut „id“. Dies kann ebenfalls zur Verringerung des Speicherplatzes gelöscht werden, wenn es im Verlauf der Arbeit nicht benötigt wurde.
Gerade bei solchen Arbeiten wird gerne mit einer Vielzahl von Kommentaren im Quelltext gearbeitet, um das Verständnis zu erhöhen. Diese benötigen aber auch Speicherplatz. Somit wurde sich für eine kommentierte Version und für eine Web – Version entschieden, welche keine Kommentare enthält.
Zu guter Letzt können die SVG – Dokumente, wie in den Arbeiten von Y. Voigt [VOIGT, 2001] und O. Schnabel [SCHNABEL, 2002] beschrieben, komprimiert werden. Dieser letzte Arbeitsschritt bildet die effektivste Art, Speicherbedarf zu senken und somit die Ladezeit beim Aufruf der Internetseite zu minimieren. Genutzt wurde dazu das Programm WinGZ. Dabei konnte z.B. die Dateigröße von *campus.svg* von 118 auf 31 KB komprimiert werden.

7 Zusammenfassung

7.1 Funktionen der Internetseite des Campus im Überblick

Die Abbildung in diesem Kapitel soll noch einmal die verschiedenen Funktionen verdeutlichen. Dabei sind ein Großteil der Funktionen aufgeführt, die der Nutzer direkt durch eine Aktion auslösen kann. Funktionen die innerhalb einer Funktion aufgerufen werden oder Hilfs- und Warnfunktionen (z.B. *punktweg( )* oder *blinken( )*) wurden nicht mit abgebildet. Die Pfeile zeigen von wo aus der Aufruf der Funktion erfolgt. Die Nummern geben an, welches Event die Funktion auslöst.

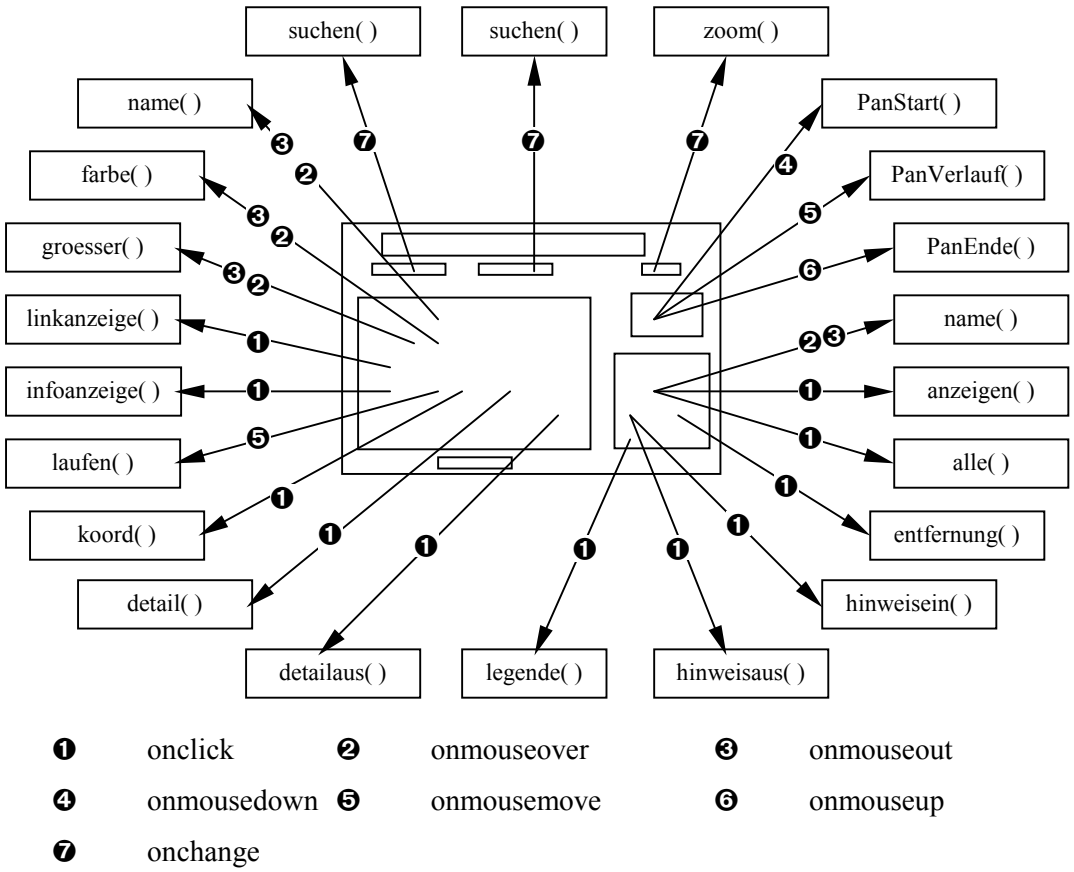

Abbildung 14: Orte und Auslösungsereignisse des Großteils der Funktionen der Internetseite.

7.2 Beurteilung des Ergebnisses und Ausblick

Mit der Erstellung der interaktiven Karte bzw. der interaktiven Internetseite des Campus der TU Dresden wurde eine, bis dahin nur mit dem Format Flash erreichbare Form der Kartendarstellung erarbeitet. Die zu Beginn der Arbeit aufgeführten Ansprüche konnten durchaus mit SVG umgesetzt werden. Natürlich kann man die Möglichkeiten des Einbaus von Interaktivitäten noch weiter ausreizen, doch muss sich dann die Frage gestellt werden, ob der

erreichte Nutzen überhaupt den Aufwand rechtfertigt, gerade bei einem so schnelllebigen Medium wie dem Internet. Zu bedenken ist außerdem, dass es sich bei der Karte um einen relativ kleinen Ausschnitt handelt. Die Darstellung eines gesamten interaktiven Stadtplanes würde wohl die Erstellung durch mehrere Bearbeiter erfordern, um in einem akzeptablen Zeitrahmen zu bleiben.

Völlig andere Überlegungen sind notwendig, wenn man bedenkt, dass eine ursprünglich für den Druck hergestellte Karte für die Internetpräsentation genutzt wurde. Abgesehen von den Überlegungen zu den verwendeten Farben und der damit verbundenen Problematik der Web – Farben müssen sich auch Gedanken gemacht werden, ob denn eigentlich noch bildhafte Signaturen notwendig sind, wenn über eine Funktion die Signaturenbezeichnung angezeigt werden kann. Außerdem stellt sich die Frage, welche Rolle dann die traditionell separat angeordnete Legende spielt. In dieser Arbeit wurde deshalb eine interaktive Legendenform zur Anwendung gebracht. Ob diese wirklich sinnvoll ist, ist zu prüfen, denn genauso könnte auch ständig die Bezeichnung des gerade überfahrenen Kartenelementes angezeigt werden.

Mit dem Grad an Interaktivität steigt natürlich auch der Grad an Komplexität der Internetseite und somit die Notwendigkeit einer klar strukturierten und mit Hinweisanzeigen unterstützen Präsentation. Tests, die während der Erstellung durchgeführt wurden, zeigten, dass viele Anwender ohne Hinweisanzeige nur einen Bruchteil der zur Verfügung gestellten Interaktionsmöglichkeiten nutzten. Deshalb wurde diese Anzeige in die Arbeit eingearbeitet, was aber gerade in der Wortwahl auf Grund des beschränkten Platzes nicht immer leicht fiel. Auf jeden Fall sollte aber nicht darauf verzichtet werden.

Wie sich die Entwicklung von Werkzeugen zur Erstellung einer interaktiven SVG – Karte darstellt, ist schwer abzuschätzen. Ein Programm, das die in dieser Arbeit ausgeführten Schritte unterstützt, gibt es mit großer Wahrscheinlichkeit nicht. Mittlerweile sind allerdings eine Vielzahl von Programmen auf dem Markt, mit denen sich SVG – Graphiken erstellen und bearbeiten lassen. Eine Liste der Programme kann auf der offiziellen SVG – Seite des W3C [SVG-W3C, 2002] nachgelesen werden. Ein Beispiel ist das von der Firma Jasc Software entwickelte Programm Jasc WebDraw, welches, ähnlich wie HTML – Editoren, eine Ansicht im Arbeitsmodus, im Quelltextmodus und im Vorschaumodus erlaubt. Jedoch führte die Anzeige eines aus Adobe Illustrator 9 oder Corel Draw 10 exportierten SVG – Dokumentes im Arbeitsmodus zu keinem sinnigen Ergebnis. Es gelang auch nicht, die überarbeitete Endversion der SVG – Karte mit diesem Programm zu betrachten. Die Schwierigkeiten liegen dabei wohl in unsauber formulierten Ausdrücken und in nicht verwendeten Attributen. Andere Programme scheitern wiederum an den eingefügten Event – Attributen. Ein Austesten dieser Programme in Hinsicht auf die Einarbeitung von Interaktivitäten wäre daher wünschenswert und könnte Thema einer anschließenden Studienarbeit sein.

8 Anhang

8.1 Verzeichnisse

8.1.1 Abkürzungen

CSS	**C**ascading **S**tyle **S**heets
DOM	**D**ocument **O**bject **M**odel
ECMA	**E**uropean **C**omputer **M**anufacturers **A**ssociation
GIF	**G**raphics **I**nterchange **F**ormat
HTML	**H**yper**T**ext **M**arkup **L**anguage
PDF	**P**ortable **D**ocument **F**ormat
SMIL	**S**ynchronized **M**ultimedia **I**ntegration **L**anguage
SVG	**S**calable **V**ector **G**raphics
URL	**U**niform **R**esource **L**ocators
WWW	**W**orld **W**ide **W**eb
W3C	**W**orld **W**ide **W**eb **C**onsortium
XML	e**X**tensible **M**arkup **L**anguage

8.1.2 Abbildungen

8.1.3 Quellen

BOLLMANN, J.; KOCH, W.G. (2001): Lexikon der Kartographie und Geomatik;
Heidelberg, Berlin: Spektrum, Akademischer Verlag.

Carto:net (2002): http://www.carto.net/papers/svg

DICKMANN, F. (2001):Web-Mapping und Web-GIS;
Braunschweig: Westermann Schulbuch Verlag GmbH.

MÜNZ, S. (2001): Selfhtml 8.0 – HTML-Dateien selbst erstellen;
http://selfhtml.teamone.de

SALATHÉ, M. (2001): SVG - Scalable Vector Graphics;
München: Markt + Technik Verlag.

SCHNABEL, O. (2002): Konzeption eines Online-Nationalatlas Bundesrepublik Deutschland;
Diplomarbeit, Technische Universität Dresden, Institut für Kartographie.

SEEBOERGER-WEICHSELBAUM, M. (2001): JavaScript – Das bhv Taschenbuch;
Landsberg: verlag moderne industrie Buch AG & Co. KG.

SVG-W3C (2002): Scalable Vector Graphics
http://www.w3.org/Graphics/SVG

VOIGT, Y. (2001): Scalable Vector Graphics (SVG) für kartographische Internet-Anwendungen;
Studienarbeit, Technische Universität Dresden, Institut für Kartographie.

WINTER, A.M. (2000): Internetkartographie mit SVG;
Diplomarbeit, Universität Wien.

W3C (2002): SVG 1.1 Specifikation
http://www.w3.org/TR/SVG

8.2 Quelltext (*script.js*)

Auf den nachfolgenden Seiten soll der Quelltext der JavaScript – Datei (*script.js*) aufgeführt werden. Dabei wird aus Platzgründen auf eine Darstellung der Arrays zur Speicherung der Informationen für die separaten Informationsfenster verzichtet.

```
var svgkarte
var svgueberblick
var svgrechteck
var svgmenue
var kartedx
var kartedy

// für Funktion farbe()
var tempstyle
var farbstatus = 0
var wechselfarbe = "fill:#E00000"

// für Funktionen suche() und ausgang()
var tempstyle2
var suchfarbe = "fill:#FF0000"
var suchstatus = 0
var w1

// für Funktionen bewegen() und ~pan() und zoom()
var eckex
var eckey
var eckexneu
var eckeyneu
var cursorx
var cursory
var cursorxneu
var cursoryneu
var svgMainViewport
var width
var height
var ueberbreite = 200
var ueberhoehe = 132.43
var kartenbreite = 20207
var kartenhoehe = 13380
var loEckex = -10060
var loEckey = 0
var status = 0

//für Zoomauswahl bei Suchergebnis
var vorher

// für Legendenfunktionen
var legendestatus = 0
var kasten
var text
var inhalt
var tempx
var tempy
var ttx
var tty
var temptext
var kastenbreite = 3600
var kastenhoehe = 1000

// für Blinken bei unerlaubter Handlung
var color
var aktiv
var durchlauf = 0
var blinker
var part

// für Änderung der Richtungspfeile
var norden = 1
var sueden = 1
var osten = 1
var westen = 1
```

```
//für Detailanzeige
var liniennummer
var detailstatus = 0
var textstatus = 0

// für Menüblock
var menuestatus = 0

// für Messfunktion
var messstatus = 0
var punktstatus = 0
var altabstand = 0
var abstand = 0
var altx = 0
var alty = 0
var punktzahl = 0
var verhaeltnis = 0.071115317

//-----------------------------------------------------------------------------------------------------------------------------------------
// zur Initialisierung der eingebetteten Karte

function initMap() {
// Erstellung der Referenz auf die SVG-Dokumente, die später noch angesprochen werden
   svgkarte = document.karte.getSVGDocument();
   svgmenue = document.menue.getSVGDocument();
   svgmassstab = document.massstab.getSVGDocument();
   svgueberblick = document.ueberblick.getSVGDocument();
// Erstellung der Referenz auf das transparente Rechteck in Übersichtskarte und auf Viewport der
// Hauptkarte
   svgrechteck = svgueberblick.getElementById("rechteck");
   svgMainViewport = svgkarte.getElementById("campus");
// Gebäude- und Studienberatung werden auf Ausgangswert gesetzt
   ausgangsuche();
   ausgangsuche2();
// Zoomstufe wird auf Originalgröße gesetzt
   for(i=0; i<document.selectZoomVal.Zoomstufe.length; i++)
   if(document.selectZoomVal.Zoomstufe.options[i].defaultSelected == true)
   document.selectZoomVal.Zoomstufe.options[i].selected=true;
// Berechnung der Größe der SVG – Elemente je nach Auflösung des Monitors
   aufloesung();
  }

//-----------------------------------------------------------------------------------------------------------------------------------------
//Berechnung der Größen der SVG – Elemente je nach Auflösung des Monitors
//Angabe in Pixel nötig, da Elemente in Tabelle
//Prozentabgaben würden nur für Tabellezelle zutreffen

function aufloesung() {
// Abfrage der Monitorauflösung
   var breite = screen.width;
   var hoehe = screen.height;
// Berechnung der Größen und Rückgabe an embed – Element in html – Datei
// Karte
   kartedx = breite / 100 * 61.94;
   kartedy = kartedx / 1.51;
   document.karte.width = kartedx;
   document.karte.height = kartedy;
   document.getElementById("fenster").width = kartedx;
   document.getElementById("fenster").height = kartedy;
// Maßstab
   var massdx = Math.round((breite / 100 * 21.53));
   var massdy = Math.round((hoehe / 100 * 2.73));
   document.massstab.width = massdx;
   document.massstab.height = massdy;
// Menue
   var steuerdx = Math.round((breite / 100 * 25.39));
   var steuerdy = Math.round((hoehe / 100 * 37.76));
```

```
    document.menue.width = steuerdx;
    document.menue.height = steuerdy;
// Übersichtskarte
    var ueberdx = Math.round((breite / 100 * 19.53));
    var ueberdy = Math.round((hoehe / 100 * 17.24));
    document.ueberblick.width = ueberdx;
    document.ueberblick.height = ueberdy;
// Titel
    var titeldx = Math.round((breite / 100 * 78.13));
    var titeldy = Math.round((hoehe / 100 * 6.51));
    document.titel.width = titeldx;
    document.titel.height = titeldy;
   }

//-----------------------------------------------------------------------------------------------------------------------------------------------
//um Suchauswahl wieder auf Anfangswerte zu setzen

function ausgangsuche() {
// setzt Gebäudesuche wieder auf Ausgangswert
    for(i=0; i<document.suche.suchid.length; i++)
    if(document.suche.suchid.options[i].defaultSelected == true)
    document.suche.suchid.options[i].selected=true;
   }

function ausgangsuche2() {
// setzt Studiengangsuche wieder auf Ausgangswert
    for(i=0; i<document.suche2.suchid2.length; i++)
    if(document.suche2.suchid2.options[i].defaultSelected == true)
    document.suche2.suchid2.options[i].selected=true;
   }

//-----------------------------------------------------------------------------------------------------------------------------------------------
// zur Anzeige des Namens beim Überfahren mit der Maus

function name(n) {
// Abbruch wenn Legende aktiviert
    if (legendestatus == 1) return;
// Referenz auf Textelement mit id=anzeige im Menue
    var text = svgmenue.getElementById("anzeige");
    text = text.getFirstChild();
// wenn Name ungleich 0 dann Namensanzeige, ansonsten leere Anzeige
    if (n != 0) text.setData(n);
    else text.setData("");
   }

//-----------------------------------------------------------------------------------------------------------------------------------------------
//zur Änderung der Farbe beim Überfahren mit der Maus

function farbe(f) {
// Abbruch wenn Legende an
    if (legendestatus == 1) return;
// Abbruch wenn Gebäude durch Suche bereits rot gefärbt
    if (w1 == f && suchstatus == 1) return;
// Referenz zum Objekt, das Farbe ändern soll
    var element = svgkarte.getElementById(f);
// Farbwechsel
    if (farbstatus == 0) {
// Speicherung der alten Farbe und Änderung zur neuen
     tempstyle = element.getAttribute("style");
     element.setAttribute("style",wechselfarbe);
     farbstatus = 1;
    }
    else {
// Rückgabe der alten Farbe
     element.setAttribute("style", tempstyle);
     farbstatus = 0;
    }
```

```
  }

//-----------------------------------------------------------------------------------------------------------------------------------------------------------
// zur Steuerung der Signaturenanzeige

// zum Aus- oder Einschalten alle Signaturen
function alle(wert) {
// Abbruch wenn Legende an
   if(legendestatus == 1){
    blinken(1);
    return;
   }
// Schleife bis alle Signaturen abgearbeitet sind
   for (var z = 1; z <= 7; z++) {
//Referenzen zu Signaturen und farbigen Deckern
    var sigebene = "s" + z;
    var e1 = svgkarte.getElementById(sigebene);
    var sbk1 = e1.getStyle();
    var decker = sigebene + "d";
    var e2 = svgmenue.getElementById(decker);
    var sbk2 = e2.getStyle();
// je nach Wert werden alle sichtbar oder alle unsichtbar
     if (wert == 1){
      sbk1.setProperty('visibility','visible');
      sbk2.setProperty('visibility','visible');
     }
     else {
      sbk1.setProperty('visibility','hidden');
      sbk2.setProperty('visibility','hidden');
     }
    }
   }

// zum Aus- und Einschalten einzelner Signaturen
function anzeigen(evt,wert) {
// Abbruch wenn Legende an
   if(legendestatus == 1){
    blinken(1);
    return;
   }
// Referenz zum angeklickten Objekt und dessen id
   var element = evt.getTarget();
   var id = element.getAttribute("id");
// wenn Objekte ausgeschaltet werden sollen wird auf farbigen Decker geklickt
// deshalb nur ersten zwei Werte der id nehmen
   if (wert == 2) id = id.substr(0,2);
   var objekt = svgkarte.getElementById(id);
   var sichtbarkeit = objekt.getStyle();
// je nach Wert sichtbar oder unsichtbar machen der Signaturen und des farbigen Deckers
   if (wert == 1){
    sichtbarkeit.setProperty('visibility','visible');
    var decker = id + "d";
    objekt = svgmenue.getElementById(decker);
    sichtbarkeit = objekt.getStyle();
    sichtbarkeit.setProperty('visibility','visible');
   }
   else {
    sichtbarkeit.setProperty('visibility','hidden');
    var decker = id + "d";
    objekt = svgmenue.getElementById(decker);
    sichtbarkeit = objekt.getStyle();
    sichtbarkeit.setProperty('visibility','hidden');
   }
  }
//-----------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
// zur Vergrößerung der Signaturen

function groesser(evt,wert) {
// um unterschiedliche Vergrößerungen je nach Zoomstufe zu erhalten
// wenn Wert = 1, dann zurück zu Ausgangsgröße
   switch (wert) {
    case 1: factor = 1;
    break;
// wenn Wert = 2, dann je nach aktueller Zoomstufe unterschiedliche Vergrößerungen oder keine
    case 2: zoomVal = parseFloat(selectZoomVal.Zoomstufe.value);
     switch (zoomVal) {
      case 100: factor = 2.5;
      break;
      case 200: factor = 1.7;
      break;
      case 400: factor = 1;
      break;
      case 600: factor = 1;
      break;
     }
     break;
    }
// Refernez zum Objekt auf dem sich Maus befindet
   var element = evt.getTarget();
// Referenz zu Werten des Attributes transform
   var trafo1 = element.getAttribute("transform");
   trafo1 = new String(trafo1);
// Festlegung eines Regulären Ausdruckes zur Transformation
// (... just copy it, I copied it either ...)  ;o)
    var translateRegExp=/translate\(([-+]?\d+)(\s*[\s,]\s*)([-+]?\d+)\)\s*/;
// Anwendung des RegExp auf trafo1 und Extraktion der Werte
   if (trafo1.length != 0) {
    var result = trafo1.match(translateRegExp);
    if (result == null || result.index == -1) {
      oldTranslateX = 0;
      oldTranslateY = 0;
     }
     else {
      oldTranslateX = result[1];
      oldTranslateY = result[3];
     }
// Festlegung der neune Transformation mit Vergrößerungsfaktor
    var trafo2 = "translate(" + oldTranslateX + " " + oldTranslateY + ") " + "scale(" + factor + ")";
   }
// Übergabe an Element
   element.setAttribute('transform', trafo2);
  }

//-----------------------------------------------------------------------------------------------------------------------------------------
// zum Ein- und Auszoomen

function zoom(wert) {
//Abbruch wenn Legende an
   if (legendestatus == 1){
    document.selectZoomVal.Zoomstufe.options[0].selected=true;
    blinken(1);
    return;
   }
//Abbruch wenn gemessen wird
   if (messstatus == 1){
    document.selectZoomVal.Zoomstufe.options[0].selected=true;
    blinken(2);
    return;
   }
// Fallbehandlung je nach Wert
   switch (wert) {
// Aufruf über Zoomstufenauswahl
```

```
    case 0: zoomVal = parseFloat(selectZoomVal.Zoomstufe.value);
    break;
// Aufruf durch Suchfunktion
    case 1:
     zoomVal = 400;
     vorher = 610;
    break;
// Aufruf wenn Legende Messen oder DVB-Detail an
    case 2: zoomVal = 100;
    break;
   }
// je nach Zoomstufe wird entsprechende Maßstabsleiste sichtbar
   switch (zoomVal) {
    case 100: massstab(100,200,400,600);
    break;
    case 200: massstab(200,100,400,600);
    break;
    case 400: massstab(400,200,100,600);
    break;
    case 600: massstab(600,200,400,100);
    break;
   }
// zum Aufheben des Suchergebnis beim reinzoomen
   if (suchstatus == 1){
    var aktuell = zoomVal;
// wenn neue Zoomstufe größer als Vorherige ist, wird Suchergebnis aufgehoben
    if (aktuell > vorher) ausgang();
// Übergabe der neuen Zoomstufe
   vorher = aktuell;
  }
// Zuweisung der Eckkoord. und Größe des transparenten Rechtecks in Übersichtskarte
   eckex = parseFloat(svgrechteck.getAttribute("x"));
   eckey = parseFloat(svgrechteck.getAttribute("y"));
   width = parseFloat(svgrechteck.getAttribute("width"));
   height = parseFloat(svgrechteck.getAttribute("height"));
// Berechnung der Zentrumskoord
   var zentrumx = eckex + width / 2;
   var zentrumy = eckey + height / 2;
//Berechnung der neuen Zentrumskoordinaten
   eckexneu = zentrumx - kartenbreite / 2 * (100/zoomVal);
   eckeyneu = zentrumy - kartenhoehe / 2 * (100/zoomVal)
// Berechnung der neune Größe
   width = kartenbreite * (100/zoomVal);
   height = kartenhoehe * (100/zoomVal);
// Kontrolle der neuen Koordinaten und Anpassung der Bewegungspfeile am Kartenrand
   kontrolle();
// wenn zoomen auf Originalansicht
   if (zoomVal == 100) {
    document.nord.src='button/norden3.gif';
    document.sued.src='button/sueden3.gif';
    document.ost.src='button/osten3.gif';
    document.west.src='button/westen3.gif';
    norden = 1; sueden = 1; osten = 1; westen = 1;
    textanzeige(2);
   }
   else textanzeige(1);
// Rückgabe der neuen Breite und Hoehe an transparentes Rechteck
   svgrechteck.setAttribute("width",width);
   svgrechteck.setAttribute("height",height);
// Abfrage der in kontrolle neu gesetzten Eckwerte und
// Übergabe der neuen Werte an Viewbox der Campuskarte
   PanEnde();
  }
```

```
// zur Anzeige des Hilfstextes wenn Zoomstufe ungleich Originalgröße
function textanzeige(wert) {
    if ( wert == 1 && textstatus == 0 ) {
     document.all["maushilfe"].innerText = "Rechteck mit gedrückter Maustaste verschiebbar";
     textstatus = 1
    }
    if (wert == 2 ) {
     document.all["maushilfe"].innerText = " ";
     textstatus = 0
    }
   }

//-----------------------------------------------------------------------------------------------------------------------------------------
// zur Veränderung des Kartenausschnitt (Verschieben des Rechtecks im Überblicksfenster)

function PanStart(evt) {
// wenn vorher Suche, wird Suchergebnis aufgehoben
    if (suchstatus == 1) ausgang();
// Zuweisung der Eckkoord. und Größe des transparenten Rechtecks in Übersichtskarte
    eckex = parseFloat(svgrechteck.getAttribute("x"));
    eckey = parseFloat(svgrechteck.getAttribute("y"));
    width = parseFloat(svgrechteck.getAttribute("width"));
    height = parseFloat(svgrechteck.getAttribute("height"));
// Zuweisung der aktuellen Pixelkoord. des Cursors
    cursorx = parseFloat(evt.getClientX());
    cursory = parseFloat(evt.getClientY());

    status = 1;
   }

function PanVerlauf(evt) {
// solang Maustaste gedrückt bleibt
    if (status == 1) {
// Zuweisung der neuen Cursorposition nach Bewegung
    cursorxneu = parseFloat(evt.getClientX());
    cursoryneu = parseFloat(evt.getClientY());
// Berechnung der neuen Werte nach Bewegung
    eckexneu = eckex + (cursorxneu - cursorx) * kartenbreite / ueberbreite;
    eckeyneu = eckey + (cursoryneu - cursory) * kartenhoehe / ueberhoehe;
// Kontrolle und Abfang, wenn Rechteck an Kartenrand; Anpassung der Bewegungspfeile
// Rückgabe der neuen Werte an Rechteck
    kontrolle();
// Übergabe der alten Pixelwerte des Cursors
    cursorx = cursorxneu;
    cursory = cursoryneu;
// Zuweisung der neuen Eckkoord. des Rechtecks
    eckex = parseFloat(svgrechteck.getAttribute("x"));
    eckey = parseFloat(svgrechteck.getAttribute("y"));
    }
   }

function PanEnde() {
// Zuweisung der Eckkoord. und Größe des transparenten Rechtecks in Übersichtskarte
    eckex = parseFloat(svgrechteck.getAttribute("x"));
    eckey = parseFloat(svgrechteck.getAttribute("y"));
    width = parseFloat(svgrechteck.getAttribute("width"));
    height = parseFloat(svgrechteck.getAttribute("height"));
// Zusammensetzung dieser Werte für die neue Viewbox der Karte
    viewBoxneu = eckex + " " + eckey + " " + width + " " + height;
    svgMainViewport.setAttribute("viewBox",viewBoxneu);

    status = 0;
   }
//-----------------------------------------------------------------------------------------------------------------------------------------
```

```
// zur Bewegung über Anklicken der Richtungspfeile

function bewegen(xrichtung,yrichtung) {
// wenn vorher Suche, wird Suchergebnis aufgehoben
    if (suchstatus == 1) ausgang();
// Zuweisung der Eckkoord. und Größe des transparenten Rechtecks in Übersichtskarte
    eckex = parseFloat(svgrechteck.getAttribute("x"));
    eckey = parseFloat(svgrechteck.getAttribute("y"));
    width = parseFloat(svgrechteck.getAttribute("width"));
    height = parseFloat(svgrechteck.getAttribute("height"));
// Berechnung der neuen Werte nach Bewegung
    eckexneu = eckex + xrichtung * kartenbreite / ueberbreite;
    eckeyneu = eckey + yrichtung * kartenhoehe / ueberhoehe;
// Kontrolle und Abfang, wenn Rechteck an Kartenrand; Anpassung der Bewegungspfeile
// Rückgabe der neuen Werte an Rechteck und viewport
    kontrolle();
    PanEnde();
   }

// zur Kontrolle und Abfang, wenn Rechteck an Kartenrand; Anpassung der Bewegungspfeile
function kontrolle(){
// Rechteck am linken Rand
    if (eckexneu <= loEckex) {
     svgrechteck.setAttribute("x",loEckex);
     document.west.src='button/westen3.gif';
     westen = 1;
    }
    else
// Rechteck am rechten Rand
     if ((eckexneu + width) >= (loEckex + kartenbreite)) {
      svgrechteck.setAttribute("x",loEckex + kartenbreite - width);
      document.ost.src='button/osten3.gif';
      osten = 1;
     }
// Rechteck an keinem Rand
     else {
      svgrechteck.setAttribute("x",eckexneu);
      document.ost.src='button/osten2.gif';
      document.west.src='button/westen2.gif';
      osten = 0; westen = 0;
     }
// Rechteck am oberen Rand
    if (eckeyneu <= loEckey) {
     svgrechteck.setAttribute("y",loEckey);
     document.nord.src='button/norden3.gif';
     norden = 1;
    }
    else
// Rechteck am unteren Rand
     if ((eckeyneu + height) >= (loEckey + kartenhoehe)) {
      svgrechteck.setAttribute("y",loEckey + kartenhoehe - height);
      document.sued.src='button/sueden3.gif';
      sueden = 1;
     }
// Rechteck an keinem Rand
     else {
      svgrechteck.setAttribute("y",eckeyneu);
      document.nord.src='button/norden2.gif';
      document.sued.src='button/sueden2.gif';
      norden = 0; sueden = 0;
     }
    }
//-----------------------------------------------------------------------------------------------------------------------------------------------
```

```
// zur Anzeige des gesuchten Gebäudes für Gebäudeauswahl oder Studiengangauswahl

function suchen(wert) {
// Suche ist nicht möglich, wenn DVB-Details oder Legende angezeigt werden oder gemessen wird
   if(legendestatus == 1){
    ausgangsuche();
    ausgangsuche2();
    blinken(1);
    return;
   }
   if(messstatus == 1){
    ausgangsuche();
    ausgangsuche2();
    blinken(2);
    return;
   }
   if(menuestatus == 1){
    ausgangsuche();
    ausgangsuche2();
    blinken(3);
    return;
   }
// wenn vorher Suche, erhält altes Suchergebnis wieder normale Farbe
   if (suchstatus == 1){
    var element = svgkarte.getElementById(w1);
    element.setAttribute('style', tempstyle2);
    suchstatus = 0;
   }
// je nach wert wird die ID von unterschiedliche Listen übernommen
// wert=1 -->Gebäudeauswahl, wert=2 -->Studiengangauswahl
// jeweils andere Auswahl wird in Ausgangssit. gesetzt
   if(wert == 1){
    var id = document.suche.suchid.value;
    ausgangsuche2();
   }
   else {
    var id = document.suche2.suchid2.value;
    ausgangsuche();
   }
// wenn id=nix hat nutzer das Wort Gebäudeauswahl oder Studiengangauswahl angeklickt
   if(id == "nix") return;
// je nach id erfolgt Suche oder Ausgabe Hinweisfenster
   if(id == 0) {
    alert("Gebäude liegt ausserhalb des Kartenausschnittes...\n \t --> Weberplatz 5");
    ausgangsuche2();return;
   }
   if(id == 1) {
    alert("Gebäude liegt ausserhalb des Kartenausschnittes...\n \t --> Hans-Grundig-Straße 25");
    ausgangsuche2();return;
   }
   if(id == 2) {
    alert("Gebäude liegt ausserhalb des Kartenausschnittes...\n \t --> Tharandt");
    ausgangsuche2();return;
   }
   if(id == 3) {
    alert("Gebäude liegt ausserhalb des Kartenausschnittes...\n \t --> August-Bebel-Straße");
    ausgangsuche2();return;
   }
   if(id == 4) {
    alert("Gebäude liegt ausserhalb des Kartenausschnittes...");
    ausgangsuche2();return;
   }
// values aus Gebäudelisteliste werden in zwei Werte zerlegt
// erster Wert Gebäude, zweiter Wert id der Gebäudekurzbezeichnung (für Koord.)
   w1 = id.substr(0,4);
   var w2 = id.substr(4,3);
// Koord. aus Gebäudekurzbezeichnung
```

```
    var element = svgkarte.getElementById(w2);
    var xkoord = parseFloat(element.getAttribute("x"));
    var ykoord = parseFloat(element.getAttribute("y"));
// automatisches Einzoomen und Anzeigen von 400 % in der Zoomstufenauswahl
    zoom(1);
    document.selectZoomVal.Zoomstufe.options[2].selected=true;
// Zuweisung der Größe des transparenten Rechtecks in Übersichtskarte
    width = parseFloat(svgrechteck.getAttribute("width"));
    height = parseFloat(svgrechteck.getAttribute("height"));
    eckexneu = xkoord - width / 2;
    eckeyneu = ykoord - height / 2;
// Kontrolle und Abfang, wenn Rechteck an Kartenrand; Anpassung der Bewegungspfeile
// Rückgabe der neuen Werte an Rechteck und viewport
    kontrolle();
    PanEnde();
// ändert Farbe des gesuchten Gebäudes
    element =  svgkarte.getElementById(w1);
    tempstyle2 = element.getAttribute("style");
    element.setAttribute("style", suchfarbe);
// anzeigen des Textes im Menü
    var objekt = svgmenue.getElementById("tsuche");
    var sichtbarkeit = objekt.getStyle();
    sichtbarkeit.setProperty('visibility','visible');
    suchstatus = 1;
   }

// um nach Suche wieder in Ausgangslage zurück zu gehen
function ausgang() {
    ausgangsuche();
    ausgangsuche2();
    var element = svgkarte.getElementById(w1);
    element.setAttribute('style', tempstyle2);
    var objekt = svgmenue.getElementById("tsuche");
    var sichtbarkeit = objekt.getStyle();
    sichtbarkeit.setProperty('visibility','hidden');
    suchstatus = 0;
   }

//-----------------------------------------------------------------------------------------------------------------------------------------------------
// zum Wechsel der Richtungspfeile

function wechsel1(nr) {
    switch (nr) {
     case 1:
      if (norden == 0) document.nord.src='button/norden1.gif';
     break;
     case 2:
      if (westen == 0) document.west.src='button/westen1.gif';
     break;
     case 3:
      if (osten == 0) document.ost.src='button/osten1.gif';
     break;
     case 4:
      if (sueden == 0) document.sued.src='button/sueden1.gif';
     break;
    }
   }

function wechsel2(nr) {
    switch (nr) {
     case 1:
      if (norden == 0) document.nord.src='button/norden2.gif';
     break;
     case 2:
      if (westen == 0) document.west.src='button/westen2.gif';
     break;
     case 3:
```

```
     if (osten == 0) document.ost.src='button/osten2.gif';
     break;
     case 4:
      if (sueden == 0) document.sued.src='button/sueden2.gif';
     break;
    }
   }

//-----------------------------------------------------------------------------------------------------------------------------------------------
// zum Anzeigen der Links bei Anklicken der Gebäude

function linkanzeige(evt,id) {
// Abbruch wenn Legende oder DVB an
    if (legendestatus == 1){
     blinken(1);
     return;
    }
    if (menuestatus == 1){
     blinken(3);
     return;
    }
// wenn id = 1, dann keine Infos vorhanden
    if (id == 1) {
     alert("Keine Informationen vorhanden");
     return;
    }
// wenn davor Suche erfolgt ist und klick erfolgt nicht auf gesuchtes (rotes) gebäude
// wird Suchergebnis aufgehoben, ansonsten bleibt Suchergebnis bestehen
    if (suchstatus == 1) {
     var element = evt.getTarget();
     var farbe = element.getAttribute("style");
     if (farbe != suchfarbe) ausgang();
    }
// Öffnung neues Fenster und einlesen der Daten aus Array
    var haus = id;
    var anzahl = haus.length - 1;
    var win;
    win=window.open("","Links","width=420,height=275,left=100,top=100,resizable=no,menubar=no");
    win.focus();
    win.document.write("<HTML><HEAD><TITLE>" + haus[0]);
    win.document.write("</TITLE><link rel=stylesheet type='text/css' href='formate.css'>
   </HEAD><BODY>");
    if (haus[1] != 0) {
    win.document.write("<p><span class='kdick'>Studienfachberatung f&uuml;r:</span><br>");
    win.document.write("<span class='klein'>"+ haus[1] +"</span></p>");
    }
    if (haus[2] != 0) {
    win.document.write("<p><span class='kdick'>Institute im " + haus[0] +":</span><br>");
    for(i=2; i<anzahl; i=i+2)
    win.document.write("<a href="+ haus[i+1] +" target='_blank'>" + haus[i] + "</a><br>");
    }
    win.document.write("</p></BODY></HTML>");
    win.document.close();
    }

//-----------------------------------------------------------------------------------------------------------------------------------------------
// zum Anzeigen der Infos bei Anklicken der Signaturen oder Mensen

function infoanzeige(id) {
// zum öffnen der Hilfe
    if (id == 100) {
     window.open ("hilfe.html","Hilfe","width=500,height=300,left=130,top=130, resizable=no,
     menubar=no,scrollbars=yes");
     return;
    }
```

```
// zum öffnen des Infofenster
    if (id == 111) {
     window.open("info.html","Info","width=500,height=300,left=130,top=130,resizable=no,
     menubar=no,scrollbars=no");
     return;
    }

// wenn vorher Suche, wird Suchergebnis aufgehoben
    if (suchstatus == 1) ausgang();
// Abbruch wenn Legende oder DVB an
    if (legendestatus == 1){
     blinken(1);
     return;
    }
    if (menuestatus == 1){
     blinken(3);
     return;
    }
// Öffnung neues Fenster und einlesen der Daten aus Array
    var info = id;
    var anzahl = info.length - 1;
    var win;
    win=window.open("","Infos","width=420,height=200,top=130,left=130,resizable=no,menubar=no");
    win.focus();
    win.document.write("<HTML><HEAD><TITLE>Infoanzeige</TITLE>");
    win.document.write("<link rel=stylesheet type='text/css' href='formate.css'>
    </HEAD><BODY onclick='window.close()'>");
    win.document.write("<p class='dick'>"+ info[0] +"</p><p>");
    for(i=1; i<=anzahl; i++)
    win.document.write(info[i] +"<br>");
    win.document.write("</p></BODY></HTML>");
    win.document.close();
   }

//-----------------------------------------------------------------------------------------------------------------------------------------
// zum Anzeigen der Massstabsleiste je nach Zoomstufe

function massstab(id,w1,w2,w3){
// je nach Zoomstufe wird Ebene mit entsprechender Leiste sichtbar
    var ebene
    var sichtbarkeit
    ebene = svgmassstab.getElementById(id);
    sichtbarkeit = ebene.getStyle();
    sichtbarkeit.setProperty('visibility', 'visible');
    ebene = svgmassstab.getElementById(w1);
    sichtbarkeit = ebene.getStyle();
    sichtbarkeit.setProperty('visibility', 'hidden');
    ebene = svgmassstab.getElementById(w2);
    sichtbarkeit = ebene.getStyle();
    sichtbarkeit.setProperty('visibility', 'hidden');
    ebene = svgmassstab.getElementById(w3);
    sichtbarkeit = ebene.getStyle();
    sichtbarkeit.setProperty('visibility', 'hidden');
   }

//-----------------------------------------------------------------------------------------------------------------------------------------
// zur Anzeige der DVB-Details

function detail(evt,zahl) {
// Abbruch wenn Legende oder DVB an
    if (legendestatus == 1){
     blinken(1);
     return;
    }
// wenn vorher Suche, wird Suchergebnis aufgehoben
    if (suchstatus == 1) ausgang();
```

```
// automatisches Zoomen auf Originalansicht
    zoom(2);
    document.selectZoomVal.Zoomstufe.options[0].selected=true;

    liniennummer = zahl;
// transparentes Rechteck sichtbar machen
    var hintergrund = svgkarte.getElementById("liniendetail");
    var zustand = hintergrund.getStyle();
    zustand.setProperty('visibility','visible');
// entsprechende Linienebene sichtbar machen
    var ebene = svgkarte.getElementById(liniennummer);
    var sichtbarkeit = ebene.getStyle();
    sichtbarkeit.setProperty('visibility','visible');
// klonen der angeklickten Linie und einfügen über transparentes Rechteck
    var element = evt.getTarget();
    var gruppe = element.parentNode;
    var neueGruppe = gruppe.cloneNode(true);
    neueGruppe.setAttribute('style','stroke-width:80');
    var nummer = "l" + liniennummer;
    var eltern = svgkarte.getElementById(nummer);
    eltern.appendChild(neueGruppe);
    detailstatus = 1;
   }

// zum ausschalten der DVB-Angaben
function detailaus() {
    var hintergrund = svgkarte.getElementById("liniendetail");
    var zustand = hintergrund.getStyle();
    zustand.setProperty('visibility','hidden');
    var ebene = svgkarte.getElementById(liniennummer);
    var sichtbarkeit = ebene.getStyle();
    sichtbarkeit.setProperty('visibility','hidden');
    hinweisaus('tdvb');
    detailstatus = 0;
   }

//-----------------------------------------------------------------------------------------------------------------------------------------------------
// Funktionen für Legende

function legende() {
// Abbruch und Warnung wenn bereits andere Funktion aktiv
    if (messstatus == 1){
     blinken(2);
     return;
    }
    if (menuestatus == 1){
     blinken(3);
     return;
    }
// Referenz auf Textfenster und Text
    kasten = svgkarte.getElementById("textfenster");
    text = svgkarte.getElementById("legtext");
    inhalt = text.getFirstChild();
// Referenz auf Legendenbutton im Menü
    var text2 = svgmenue.getElementById("legtext");
    var inhalt2 = text2.getFirstChild();
    var decker = svgmenue.getElementById("legdecker");
    var deckerwert = decker.getStyle();
    var element = svgkarte.getElementById("Legende");
    var sichtbarkeit = element.getStyle();
// zum Ausblenden der Legende --> Ende
    if (legendestatus == 1){
// wenn Button gerade blinkt, dann Abbruch
     if(durchlauf != 0) return;
// Rücksetzen aller Werte auf Ausgangssituation
     sichtbarkeit.setProperty('visibility','hidden');
     inhalt2.setData("Legende ein");
```

```
    kasten.setAttribute("x",tempx);
    kasten.setAttribute("y",tempy);
    text.setAttribute("x",ttx);
    text.setAttribute("y",tty);
    inhalt.setData(temptext);
    deckerwert.setProperty('visibility','hidden');
    hinweisaus('tleg');
    legendestatus = 0;
   }
// zum Einblenden der Legende --> Start
   else {
// wenn Suchergebnis angezeigt, dann Aufhebung
    if (suchstatus == 1) ausgang();
// wenn DVB-Details angezeigt werden, dann Abstellung
    if (detailstatus == 1) detailaus();
// alle Signaturen aus
    alle(2);
// zoomen auf Originalgröße
    zoom(2);
    document.selectZoomVal.Zoomstufe.options[0].selected=true;
// Ebene Legende in SVG sichtbar machen
    sichtbarkeit.setProperty('visibility','visible');
//Ausgangsposition von Rechteck und Text speichern
    tempx = kasten.getAttribute("x");
    tempy = kasten.getAttribute("y");
    ttx = text.getAttribute("x");
    tty = text.getAttribute("y");
    temptext = inhalt.getData();
    inhalt2.setData("Legende aus");
    deckerwert.setProperty('visibility','visible');
// Hinweisfenster im Menü aktivieren
    hinweisan('tleg');
    legendestatus = 1;
   }
  }

function laufen(evt) {
// zur Bewegung des Legendenkastens und Aktualisierug der Textanzeige
   if (legendestatus ==1) {
// Abruf der Mausposition und Umrechnung in Pixel der Karte
    var x = parseFloat(evt.getClientX());
    var y = parseFloat(evt.getClientY());
    var kastenx = x * kartenbreite / kartedx + loEckex + 500;
    var kasteny = y * kartenhoehe / kartedy + 500;
    var textx = Math.round(kastenx) + 200;
    var texty = Math.round(kasteny) + 600;
// abfangen des Kasten am linken und unteren Rand
    var schwelle1 = kastenx + kastenbreite;
    var schwelle2 = kasteny + kastenhoehe;
    if (schwelle1 >= kartenbreite + loEckex) {
     kastenx = kartenbreite + loEckex - kastenbreite;
     textx = kartenbreite + loEckex - kastenbreite + 200;
    }
    if (schwelle2 >= kartenhoehe) {
     kasteny = kartenhoehe - kastenhoehe;
     texty = kartenhoehe - kastenhoehe + 500;
    }
// Rückgabe der neuen Position an Rechteck und Text
    kasten.setAttribute("x",kastenx);
    kasten.setAttribute("y",kasteny);
    text.setAttribute("x",textx);
    text.setAttribute("y",texty);
// Bestimmung des Elements, über dem sich Mauszeiger befindet
    var element = evt.getTarget();
// Aufstieg zum Elternelement, also der Gruppe und Abruf der Id
    var eltern = element.parentNode;
    var id = eltern.getAttribute("id");
```

```
// Aufstieg zum nächsten Elternelement, wenn keine Id vorhanden
     if (id == 0){
      var eltern2 = eltern.parentNode;
      var id2 = eltern2.getAttribute("id");
// Übergabe des Id-Wertes an Text im Legendenkasten
       inhalt.setData(id2);
     }
     else inhalt.setData(id);
    }
   }

//-----------------------------------------------------------------------------------------------------------------------------------------------------
// Funktionen zum warnenden Aufblinken der Button
// Funktion blinken() ruft 8x aller halbe Sekunde Funktion warnung() auf

function blinken(wert) {
// wenn bereits in Ausführung, Abbruch
    if(durchlauf != 0) return;
// wenn Suchergebnis vorhanden, Aufhebung
    if(suchstatus == 1) ausgang();
// Speicherung des übergebenen Parameter
    color = 1;
    blinker = wert;
// je nach Parameter blinkt entsprechendes Objekt
    if (blinker == 1) {
     part = svgmenue;
     blinker = "legdecker";
    }
    else
     if (blinker == 2) {
      part = svgmenue;
      blinker = "messdecker";
     }
     else
      if (blinker == 3) {
       part = svgmenue;
       blinker = "dvbdecker";
      }
      else {
// wenn keiner der ersten Fälle dann blinken der DVB-Linien
       part = svgkarte;
       zoom(2);
       document.selectZoomVal.Zoomstufe.options[0].selected=true;
      }
// startet Funktion warnung() aller halbe Sek.
    aktiv = window.setInterval("warnung()",300);
}

function warnung() {
// Referenz zum zu blinkenden Objekt
    var decker = part.getElementById(blinker);
    var deckerwert = decker.getStyle();
// je nach Wert von color wird Objekt sichtbar oder unsichtbar
    if(color==1){
     deckerwert.setProperty('visibility','hidden');
     color=2;  }
    else {
     deckerwert.setProperty('visibility','visible');
     color=1;
    }
// wenn Zählvariable=8, Ende
    durchlauf = durchlauf + 1;
    if(durchlauf == 8){
     window.clearInterval(aktiv);
     durchlauf = 0;
    }
   }
```

```
//-----------------------------------------------------------------------------------------------------------------------------------------------
//Funktionen zur Anzeige des Hinweistextes im Menü

function hinweisan(text) {
// Abbruch und Warnung wenn bereits andere Funktion aktiv
    if (legendestatus == 1){
     blinken(1);
     return;
    }
    if (messstatus == 1){
     blinken(2);
     return;
    }
// Referenz auf Hinweisfenster und Sichtbarmachung
    var objekt = svgmenue.getElementById(text);
    var objektwert = objekt.getStyle();
    objektwert.setProperty('visibility','visible');
    if (text == 'tdvb'){
     objekt = svgmenue.getElementById('dvbdecker');
     objektwert = objekt.getStyle();
     objektwert.setProperty('visibility','visible');
     var dvbtext = svgmenue.getElementById("dvbtext");
     var dvbinhalt = dvbtext.getFirstChild();
     dvbinhalt.setData("DVB aus");
     menuestatus = 1;
    }
   }

function hinweisaus(text) {
// Abbruch, wenn blinken() aktiviert
    if(durchlauf != 0) return;
// Referenz auf Hinweisfenster und Unsichtbarmachung
    var objekt = svgmenue.getElementById(text);
    var objektwert = objekt.getStyle();
    objektwert.setProperty('visibility','hidden');
    if (text == 'tdvb'){
     objekt = svgmenue.getElementById('dvbdecker');
     objektwert = objekt.getStyle();
     objektwert.setProperty('visibility','hidden');
     var dvbtext = svgmenue.getElementById("dvbtext");
     var dvbinhalt = dvbtext.getFirstChild();
     dvbinhalt.setData("DVB");
     menuestatus = 0;
    }
   }

//-----------------------------------------------------------------------------------------------------------------------------------------------
// Funktionen zur Entfernungsmessung

function entfernung() {
// Abbruch und Warnung wenn bereits andere Funktion aktiv
    if (legendestatus == 1){
     blinken(1);
     return;
    }
    if (menuestatus == 1){
     blinken(3);
     return;
    }
// Referenz auf Enfernungsanzeige, roten Decker für Button
// und transparenten Rechteck über Karte
    var text2 = svgmenue.getElementById("messtext");
    var inhalt2 = text2.getFirstChild();
    var decker = svgmenue.getElementById("messdecker");
    var deckerwert = decker.getStyle();
    var element = svgkarte.getElementById("Messung");
    var sichtbarkeit = element.getStyle();
```

```
// zum Ausblenden --> Ende
    if (messstatus == 1){
// wenn gerade blinken () aktiviert, Abbruch
     if(durchlauf != 0) return;
// Wiederherstellung der Ausgangssituation
     sichtbarkeit.setProperty('visibility','hidden');
     inhalt2.setData("Entfernung");
     deckerwert.setProperty('visibility','hidden');
// Entfernung der Hinweisanzeige im Menü
     hinweisaus('tmess');
     aufheben();
     messstatus = 0;
    }
// zum Einblenden --> Start
    else {
// wenn andere Funktionen aktiviert, Aufhebung
     if (suchstatus == 1) ausgang();
     if (detailstatus == 1) detailaus();
// zoomen auf Originalgröße
     zoom(2);
     document.selectZoomVal.Zoomstufe.options[0].selected=true;
// Sichtbarmachung des transparenten Rechteckes über der Karte,
// des Hinweisfenster, des roten Deckers über dem Button und
// Anpassung des Textes im Button
     sichtbarkeit.setProperty('visibility','visible');
     hinweisan('tmess');
     deckerwert.setProperty('visibility','visible');
     inhalt2.setData("Entfernung aus");
     messstatus = 1;
    }
   }

function koord(e) {
// zum Setzen der Wegpunkte
    if (messstatus == 1) {
// Abfrage der Mausposition über der Karte und Umrechnung
     var mausx = e.getClientX();
     var mausy = e.getClientY();
     var neux = Math.round((mausx * kartenbreite / kartedx - 10060));
     var neuy = Math.round((mausy * kartenhoehe / kartedy));
// bei erstem Punkt wird Abstand auf null gesetzt
     if (punktzahl == 0) {
      abstand = 0;
      punktzahl = 1;
     }
// bei nächsten Punkten Aufsummierung und Umrechnung in Naturmaß
     else {
      abstand = ((altx - neux)*(altx - neux))+((alty - neuy)*(alty - neuy));
      abstand = Math.round((Math.sqrt(abstand)) * verhaeltnis);
     }
// Signatur für Wegpunkt referenzieren, clonen und an Zeigerposition setzen
     var element = svgkarte.getElementById("messpunkt");
     var neuPunkt = element.cloneNode(false);
     neuPunkt.setAttribute('cx',neux);
     neuPunkt.setAttribute('cy',neuy);
     var id = "punkt" + punktzahl;
     neuPunkt.setAttribute('id',id);
     var eltern = svgkarte.getElementById("Messung");
     eltern.appendChild(neuPunkt);
// Punkte mit Linie verbinden
     if (punktzahl != 1) {
      var wegstueck = svgkarte.createElement('line');
      wegstueck.setAttribute('x1',altx);
      wegstueck.setAttribute('y1',alty);
      wegstueck.setAttribute('x2',neux);
      wegstueck.setAttribute('y2',neuy);
      id = "linie" + punktzahl;
```

```
      wegstueck.setAttribute('id',id);
      wegstueck.setAttribute('style',"stroke:#FF0000;stroke-width:50");
      eltern.appendChild(wegstueck);
     }
// Speicherung der verwendeten Koordinaten für Abstandsberechnung mit nächsten Punkt
     altabstand = abstand + altabstand;
     altx = neux;
     alty = neuy;
// Übergabe der Entfernung an Textanzeige im Menü
     var strecke = altabstand + " Meter";
     name(strecke);
     punktzahl = punktzahl + 1;
     punktstatus = 0;
    }
   }

function punktweg() {
// zum Entfernen des letzen Punktes
    if(punktstatus == 1) return;
    if(punktzahl <= 2) return;
    punktzahl = punktzahl - 1;
// Referenzierung des letzten Punktes
    var id = "punkt" + punktzahl;
    var punkt = svgkarte.getElementById(id);
    var eltern = punkt.getParentNode();
// löschen
    eltern.removeChild(punkt);
// Entfernung des letzten Linienelementes
    var id = "linie" + punktzahl;
    var linie = svgkarte.getElementById(id);
    var eltern = linie.getParentNode();
    eltern.removeChild(linie);
// Referenzierung des vorletzen Punktes
    punktzahl = punktzahl - 1;
    id = "punkt" + punktzahl;
    punkt = svgkarte.getElementById(id);
    altx = punkt.getAttribute('cx');
    alty = punkt.getAttribute('cy');
// Neuberechnung der gesamten Entfernung und Anzeige
    altabstand = altabstand - abstand;
    var strecke = altabstand + " Meter";
    name(strecke);
    punktzahl = punktzahl + 1;
    punktstatus = 1;
   }

function aufheben() {
// zum Löschen des gesamten Weges
    name('');
    abstand = 0;
    altabstand = 0;
    altx = 0;
    alty = 0;
// entfernen der gesetzen Punkte
    for (var z = 1; z < punktzahl; z++) {
     var id = "punkt" + z;
     var punkt = svgkarte.getElementById(id);
     var eltern = punkt.getParentNode();
     eltern.removeChild(punkt);
    }
// entfernen der Linien
   for (var i = 2; i < punktzahl; i++) {
    var id = "linie" + i;
    var linie = svgkarte.getElementById(id);
    var eltern = linie.getParentNode();
    eltern.removeChild(linie);
   }
```

```
  punktzahl = 0;
 }



//-------------------------------------------------------------------------------------------------------------------------------------------------
//HILFSFUNKTION
//-------------------------------------------------------------------------------------------------------------------------------------------------
//zur Ermittlund der von CorelDraw  automatisch vergebenen Id
//zur Aktivierung muss innerhalb des <svg> - Elementes in der
//Datei campus.svg folgender Aufruf festgelegt werden:
//  onclick="ids(evt)"

function ids (elem){
// Referenz zum angeklickten Objekt
    var element = elem.getTarget();
    var id = element.getAttribute("id");
    alert("ID = "+id);
    return;
   }
```