Martin Wantke
17840

Bachelorarbeit

Erzeugung von interaktiven Grafiken im Rahmen der Web 2.0 Programmierung

www.hs-merseburg.de/~0mwantke/

Inhaltsverzeichnis

1. Einleitung

In unserer Gesellschaft ist die zeitnahe Kommunikation über das Internet zu einem unabdingbaren Gut geworden. Webseiten dienen nicht nur als Informationsquelle, sie dienen auch dazu, Nutzer direkt und zeitnah miteinander zu vernetzen. Die Bedienoberfläche im Web unterscheidet sich nicht mehr von der klassischen Bedienoberfläche einer Desktop-Anwendung. Zunehmend werden auf mobilen Geräten neue Web-Standards eingesetzt. Tendenziell sind maschinennahe Anwendungen rückläufig.

In dieser Arbeit wird der Web-Standard HTML5 genutzt. Obwohl dieser noch in Entwicklung ist, sind viele Funktionen schon heute nutzbar. Diese neuen Funktionen kommen hier zum Einsatz, um eine Bibliothek mit verschiedenen Diagrammtypen zu entwickeln. Die Diagramme werden erst zur Laufzeit dynamisch generiert. Fertige Grafikdateien kommen nicht zum Einsatz.

Mit Hilfe dieser Bibliothek können Webentwickler auf möglichst einfachere Art und Weise eigene Diagramme in einer Seite einbinden. Der Inhalt der Diagramme wird nicht nur statisch vorgegeben. Im Rahmen der Web 2.0 Programmierung soll der Betrachter der Seite die Möglichkeit haben, die Daten des Diagramms zu ändern. So kann er ein Diagramm im Browser von Grund auf neu gestalten. Er hat auch die Möglichkeit interaktiv auf die Darstellung einzugreifen. So kann er die Ansicht vergrößern, verschieben oder das Diagramm drehen.
Für die Lauffähigkeit sind keine weiteren Installationen nötig. Die Bibliothek ist damit *plattformunabhängig*.

Im Grundlagenteil dieser Arbeit (Abschnitt 2.1) wird zunächst geklärt was sich hinter HTML verbirgt. Es werden die Ansprüche an Hard- und Software geklärt sowie der Aufbau einer Webseite beschrieben. Danach widmen wir uns im Abschnitt 2.2 den Prinzipien des Web 2.0. Was verbirgt sich hinter diesem Begriff, welche Auswirkung hat es und welche technischen Ansprüche werden gestellt?

Im Abschnitt 2.3 geht es um die neuen Möglichkeiten von HTML5. Thema ist unter anderem die direkte Kommunikation zwischen zwei Endpunkten, das Erzeugen und Verarbeiten von multimedialen Inhalten oder das Erfassen von Geräteinformationen.

Im zweiten Teil beschäftigen wir uns mit der Umsetzung der bereitgestellten Bibliothek. Die zuvor beschriebenen Techniken und Standards bilden die Grundlage für die Umsetzung. Abschnitt 3.1 gibt zunächst eine Übersicht zu den Diagrammtypen, die hier in der Bibliothek umgesetzt wurden, bevor im Abschnitt 3.2 der grundlegende Aufbau der Bibliothek beschrieben wird. Danach wird im Abschnitt 3.3 anhand eines ausgewählten Diagrammtyps der Weg von der Eingabe bis hin zur Darstellung Schritt für Schritt verfolgt. Es werden die Speicherung von Daten, die Darstellung der Grafiken und das Zusammenspiel der einzelnen Komponenten eine vorgestellt.

Zum Abschluss wird auf eine Grafikschnittstelle eingegangen, die Darstellung von 3D Grafiken in einer Webseite ermöglicht und die ursprünglich im Web-Standard nicht vorgesehen war. Mit Hilfe dieser Schnittstelle wurde das Oberflächendiagramm umgesetzt.

2. Stand der Technik

2.1 Grundlagen

Im ersten Teil dieser Arbeit beschäftigen wir uns mit dem Basiswissen der Webprogrammierung. Dieses ist Voraussetzung für eine eigene Anwendung die im zweiten Teil umgesetzt wird. Es werden nicht nur grundlegende Begrifflichkeiten geklärt, auch Teilaspekte wie die Sicherheit oder geltende Rechtsnormen spielen eine Rolle.

2.1.1 Rechnernetze

In der Netzwerkkommunikation werden netzwerkfähige Computerprogramme eingesetzt, die Daten austauschen und verarbeiten. Je nachdem welche Aufgabe sie übernehmen, unterscheidet man zwischen „Server“ und „Client“, auch dann wenn beide auf derselben Plattform laufen.
In der Regel greifen mehrere Clients auf einem Server zu. Der am häufigste genutzte Kommunikationskanal zwischen Client und Server ist das Internet. Ein Client kann somit weltweit auf einen entfernten Server zugreifen.

Ein Server bietet abstrakt gesehen immer einen Dienst an. Im Web ist es die Aufgabe des Servers eine Webseite in maschinenlesbarer Form auszuliefern. Diese kann sich statisch auf einen Datenträger befinden oder sie wird erst bei der Anfrage dynamisch erzeugt. Gegebenenfalls wartet der Server nach dem Ausliefern auf weitere Anfragen des Clients. Grundlage bei einer Client-Server-Architektur ist immer das Prinzip von Anfrage und Antwort.

Ein Server kann auch die Dienste anderer Server in Anspruch nehmen. Dies ist meist bei einer Webseite mit einer Datenbank-Anbindung der Fall. Greift ein Nutzer über ein Web-Interface auf seine gespeicherten Daten zu, holt sich der Web-Server die Daten von einem Datenbank-Server und pflegt diese in die Webseite ein bevor er sie ausliefert.

Die meiste Zeit läuft der Server für seine Aufgabe im Leerlauf. Er kann nicht wissen, wann die nächste Anfrage ansteht. Diese soll möglichst schnell im Millisekunden Bereich beantwortet werden. Ein erneutes Hochfahren des Betriebssystems kommt daher nicht in Frage. Server sind daher meist für eine Spitzenlast ausgelegt, die selten eintritt. Bei einem Server fallen hauptsächlich Kosten für die Stromversorgung im Leerlauf an.

Eine Lösung für das Dilemma sind so genannte Cloud-Dienste. Dabei wird die Ausführung des Dienstes immer auf die Hardware verlagert, bei denen momentan die wenigsten Kosten entstehen. Da dies auch andere Länder sein können ist ein strengerer Datenschutz nötig.

Eine andere Variante, das Problem zu mindern sind so genannte V-Server. Hier laufen mehrere Betriebssysteme mit deren Software auf derselben Hardware. Die Chance, dass alle Anwendungen auf einem V-Server schlafen ist somit geringer. Viele kleinere Webseiten können so kostengünstiger für den jeweiligen Webseitenbetreiber angeboten werden.

Ein Client nimmt Dienste des Servers in Anspruch, ohne einen Dienst selbst zur Verfügung zu stellen. Für den Anwender der eine Internetseite aufruft, ist das die Software die Rohdaten vom Server nimmt und sie zu einer Darstellung rendert, der so genannten *Browser*. Eine nicht zu unterschätzende Aufgabe.

Es werden immer neue Webseiten-Elemente mit der Zeit standardisiert. Diese werden nicht gleich oder nur unvollständig von Browsern umgesetzt oder variiert in der Darstellung je nach verwendeten Browser und Versionsnummer. Es kann einige Zeit in Anspruch nehmen bevor ein Element von jedem Browser unterstützt wird. Es gibt auch Elemente die als „*veraltet*“ deklariert werden. Diese sind dann nur noch für die Abwärtskompatibilität verfügbar. Bei neueren Browser fehlen diese jedoch.

Das Client-Server-Prinzip ist nicht die einzige Möglichkeit ein Netz aufzubauen. Es existieren auch Netze in denen nur die Clients untereinander kommunizieren, ohne Server. Diese werden als *Peer-To-Peer (P2P)* Netze bezeichnet. Solche Netze können zur besseren Lastverteilung genutzt werden. Dafür existieren eigene Protokolle wie *torrent*.
Jeder Traffic kostet für den Server-Administrator Geld. Wird eine sehr große Datei angeboten fallen diese bei jeden Download der ein Client anstößt für die ganze Datei an. Daher sind manche Anbieter dazu übergegangen als Zweitoption eine kleine torrent-Datei anzubieten, welche nur ein Verweis innerhalb des P2P Netzes speichert. Damit wird nahezu vollständig die Bandbreite der verbundenen Clients im P2P-Netz genutzt, indem jeder Nutzer seinen Teil im Upload-Bereich anbietet.

Damit ein neu angemeldeter Client möglichst schnell andere Clients findet, ist er auf Hilfe eines Servers angewiesen, egal in welcher Form. Dieser hält eine Adress-Liste von Clients vor. Bei diesen Bootstrapping-Verfahren, also das anstoßen von einem komplexen Vorgang, spielt der Server eine untergeordnete Rolle. Sobald sich zwei Clients neu verbunden haben tauschen sich diese ihre Liste von aktiven Clients untereinander aus, es bildet sich ein Schwarm. Der Server hat in dieser Situation bereits die Verbindung getrennt.

Die Auswirkung solcher dezentralen Schwärme wird an Hand von virtuellen Währungen wie Bitcoin oder Litecoin deutlich. Diese bauen vollständig auf Peer-To-Peer Netzen auf. Alle Transaktionen werden von anderen Clients überwacht und sind irreversibel. Jede Geldeinheit ist fälschungssicher und Kredite gibt es nicht. Staaten oder Zentralbanken können keinen Einfluss auf so eine Währung nehmen was wiederum zu Kursschwankung führt. Besitzt man ein Bitcoin, kann anhand einer global geführten Datenbank, die von allen Clients geteilt wird, deren Weg bis hin zu ihrer Entstehung zurückverfolgen. Im Einzelhandel wird bei einer virtuellen Währung keine neue teure Hardware benötigt, es reicht eine App oder ein QR-Code. Prinzipiell ist das Geld sofort verfügbar. In Krisenstaaten werden virtuelle Währungen genutzt um gegen eine starke Geldentwertung vorzugehen. Für den Schutz gegen eine Inflation ist die Anzahl an Einheiten begrenzt. Whistleblower-Seiten, die aufgrund ihrer Enthüllungen und durch Druck der Regierung keine Zahlungen mehr abwickeln können, weichen auf virtuelle Währungen aus. In Thailand sind Transaktionen mit Bitcoins inzwischen verboten[1]. Dagegen ist in Deutschland Bitcoins als „privates Geld“ anerkannt [2].

Wer ein Internetauftritt auf einen Server betreibt, muss sich an bestimmte Vorgaben halten wie die Angabe eines Impressums oder bei einem Kaufvertrag eine vom Verbraucherschutz geforderte Button-Lösung. Bei P2P-Netze existieren solche Bedingungen nicht, sie arbeiten über Ländergrenzen hinweg. Die technischen Anforderungen ein dezentrales P2P-Netz zu

---

[1] "Bitcoin-Verbot in Thailand", Heise Zeitschriften Verlag, abgerufen am 22.9.2013
http://www.heise.de/newsticker/meldung/Bitcoin-Verbot-in-Thailand-1926521.html

[2] Kathrin Gotthold und Daniel Eckert,
"Deutschland erkennt Bitcoin als privates Geld an", Axel Springer AG,
abgerufen am 22.9.2013
http://www.welt.de/finanzen/geldanlage/article119086297/Deutschland-erkennt-Bitcoin-als-privates-Geld-an.html

regulieren sind hoch. Die meisten P2P-Anwendungen sind darauf programmiert Sperren wie das heimische NAT im Router zu umgehen [3]. China, das Land mit den meisten Internetsperren, versucht inzwischen P2P-Anwendungen zu blocken [4]. Das ist über die Portnummer im IP-Protokoll möglich. Diese Blockierung kann jedoch ein wirtschaftlicher Schaden darstellen, wenn Unternehmen auf Internet (Video-)Telefonie setzen. Es gibt für diesen Fall Clients die ihre Kommunikation über verschlüsselte Web-Protokolle verschleiern. Diese Portnummer wird bei den meisten Firewalls durchgelassen.

Bietet ein Client Daten an, haftet zunächst derjenige der die Anwendung betreibt. In manchen Fällen kann das eine Gefahr darstellen. Kryptografie ermöglicht ein *anonymes Peer-To-Peer* Netz. Die Daten werden über mehreren Knotenpunkten (so genannte *Hops*) immer mit unterschiedlichen Routen durch das Netz geleitet und mit anderen Datenströmen vermischt. Bei jedem Hop wird das Datenpaket entweder teilweise ver- oder entschlüsselt. Auf dem Ziel-Client werden die Daten von den Nachbarknoten gesammelt und dort zum Klartext entschlüsselt. Von wem die Daten stammen ist somit nicht mehr nachprüfbar, egal welchen Kanal man abhört. Eine 100-prozentige Sicherheit gibt es in der Kryptografie nicht. Die Wahrscheinlichkeit, dass ein Angreifer eine Verschlüsselung knackt, kann nur extrem unwahrscheinlich mit zunehmender Schlüssellänge werden. Bei verschlüsselten Verbindungen im Web wird unter anderem der *Advanced Encryption Standard* (AES) eingesetzt. Bei einem 256-bittigen AES ist die Wahrscheinlichkeit 1 zu $1{,}17 \cdot 10^{77}$, dass eine Angreifer im ersten Versuch den Schlüssel errät.

Die P2P-Technik hält inzwischen auch bei Webseiten Einzug, mit all seinen Vor- und Nachteilen. Möchte ein neuer Peer-Knoten eine Verbindung zum Browser aufbauen, muss der Nutzer den Verbindungswunsch erst akzeptieren.

Wie bereits erwähnt greifen in der Regel mehrere Clients auf einen Server zu. Die einzelnen Clients sehen nicht die Aktivität der anderen Teilnehmer. Bei Social-Media Anwendungen wo sich Nutzer direkt untereinander austauschen, stößt diese Netzwerktopologie an ihre Grenzen. Bei der Video-Telefonie muss der komplette Traffic von jedem Nutzer über einen zentralen Server geleitet werden. Allein bei einem Video-Stream fallen im Vergleich zu einem bloßen Seitenabruf viele Nutzdaten an.

Nutzt man hingegen Peer-To-Peer werden die Daten direkt zwischen den Teilnehmern ausgetauscht und die unnötige Last fällt auf dem Server fällt weg. Der Server dient in diesen Fall nur dazu die Verbindung zwischen zwei Clients herzustellen. Die Grundlage für diese Technik bildet der offene Standard „WebRTC“, was übersetzt für „Web-Echtzeitkommunikation“ steht. Es nutzt das mit HTML5 neu eingeführte *JavaScript Session Establishment* Protokoll, welches eine Direktverbindung zwischen den Clients auszuhandeln.

Dieses Protokoll ist nur eins von vielen Neuerungen die derzeit standardisiert werden. Im Kapitel HTML5 wird näher darauf eingegangen.

---

[3] "Filesharing blockieren", Heise Zeitschriften Verlag, abgerufen am 22.9.2013 http://www.heise.de/ct/hotline/Filesharing-blockieren-1436278.html

[4] "China will Skype blockieren", IDG Tech Media GmbH, abgerufen am 22.9.2013 http://www.pcwelt.de/news/China-will-Skype-blockieren-326658.html

2.1.2 Was passiert bei einem Seitenaufruf?

Die Client-Server-Architektur beschreibt nur grob die Funktionsweise des Webs. Wird eine Webseite vom Browser abgerufen, geschehen im Hintergrund so einige Dinge, die dem Nutzer verborgen bleiben. Bei einer Webseite wie www.hs-merseburg.de, wird zunächst diese Adresse aufgelöst. Eine solche Adresse wird *Domain* genannt. Sie bildet bei Netzadressen das Bindeglied zwischen einer Menschen lesbaren und einer Maschinen lesbaren Form. Ein Mensch kann sich eine Domain durch richtige Wörter besser merken. In der Informatik nutzt man dagegen *IP-Adressen*. Bei der Auflösung einer Adresse wird die Domain www.hs-merseburg.de in die IP-Adresse 149.205.5.70 umgewandelt.

Die Auflösung der Domain geschieht von hinten nach vorn. Zuerst wird die Top-Level-Domain „de“ aufgelöst und als letztes das „www“. Die letzte Auflösung wird meist als Load-Balancer genutzt. Wenn zu viele Anfragen auf einmal auftreten, wird die Last verteilt indem die Auflösung immer auf verschieden gespiegelte Server zeigt.
Durch die Domain Auflösung erhält der Browser die für die Kommunikation nötige IP-Adresse.

Die eigentliche Kommunikation findet mit Hilfe von Protokollen statt. Diese stellt eine feste Verbindung zwischen Client und Server her. Im Web wird das „Transmission Control Protocol“ (kurz TCP) genutzt. Es sorgt für eine verlustfreie Übertragung. Geht ein Datenpaket verloren wird es erneut gesendet. Das kann durchaus passieren wenn bei einem Internetknoten die Hardware, der so genannte *Router*, überlastet ist. Jedes Paket kann über einen unterschiedlichen Weg und in unterschiedlich langer Zeit zu seinem Ziel gelangen. Das Protokoll sorgt dafür, dass alle Pakete in richtiger Reihenfolge zum Ziel gelangen. Alle über das Protokoll befindlichen Schichten müssen sich somit nicht mehr um eine saubere Datenübertragung kümmern.

Eine Webseite wird über das HTTP Übertragungsprotokoll übertragen. Dieses nutzt TCP für seine Kommunikation. HTTP ist nicht auf Webseiten beschränkt. Vielmehr dient es zur Übertragung von beliebigen Daten. Wird bei einer Webseite per Rechtsklick ein eingebettetes Bild auf den eigenen Rechner gespeichert, wird dasselbe Protokoll genutzt wie bei der Auslieferung einer Webseite. Nur ist in diesen Fall das Format der Daten anders.

Damit Browser und Server erkennen können wie mit den gesendet oder empfangenen Daten umzugehen ist, existiert eine Medientyp-Vereinbarung. Man spricht von so genannten *MIME-Typen*, auch *Internet Media Type* genannt. Der MIME Typ von einer Webseiten ist `text/html` und der von einen Bild im JPG Format `image/jpeg`, um nur zwei zu nennen. Neben den MIME Typ werden bei HTTP noch weitere Meta-Daten angegeben. Dieser Bereich in dem diese Informationen stehen wird als „HTTP-Header“ bezeichnet. Dies ist ein ganz normaler Fließtext aus mehreren Zeilen. In jeder Zeile steht ein Schlüssel-Wert-Paar. Zuerst wird ein Schlüssel, zum Beispiel der MIME-Type, angegeben. Danach folge ein Doppelpunkt mit dem dazugehörigen Wert. Die Reihenfolge spielt dabei keine Rolle.

Bei einem Aufruf einer Seite kann der HTTP-Header wie folgt aussehen:

```
GET / HTTP/1.1
Host: www.hs-merseburg.de
User-Agent: Mozilla/5.0 (Windows NT 5.1; rv:22.0) Gecko/20100101
Firefox/22.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: de-de,de;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

Jede Zeile ist entweder eine Information über den Funktionsumfang des Browsers oder eine Anweisung an den Server. „`Connection: keep-alive`“ sorgt beispielsweise dafür, dass zwischen zwei Webseiten Aufrufe von der selben Domain die TCP-Verbindung nicht neu ab- und aufgebaut wird. Sowohl bei einem Verbindungsaufbau als auch Verbindungsabbau wird der Hin- und Rückkanal dreimal verwendet. Dieser Vorgang heißt *TCP-Handshake* oder auch „Drei-Wege-Handschlag“. Nach dem Verbindungsaufbau wird bei eine simplen Anfrage dagegen der Kanal nur zweimal genutzt.

Eine leere Zeile schließt den HTTP-Header ab. Danach kommen, falls vorhanden, die Nutzdaten. Wird eine Seite vom Browser angefordert, sind diese Daten nicht vorhanden. Es liegen nur Meta-Daten vor. Bei der Antwort vom Server befindet sich hingegen an dieser Stelle der Inhalt der Seite. Wie groß diese Daten sind, wird mit den Eintrag „`Content-Length`: “ im HTTP-Header angegeben.

Das HTTP-Protokoll ist ein zustandsloses Protokoll. Bei der Entstehung des Protokolls im Jahr 1989 war das Abspeichern von Zuständen über mehrere Anfragen hinweg nicht nötig. Es war auch nicht absehbar welche enorme Verbreitung das „World Wide Web“ haben wird. Das sieht man bereits an der Spezifikation der IP-Adresse die nur 2^{32}, also 4294967296 Endpunkte zulässt. Diese sind bereits verbraucht. Das machte eine neue Spezifikation nötig, das so genannte *IPv6*, das 2^{256} Adressen zulässt.

Damit der Server bei den ganzen unterschiedlichen Anfragen einen Nutzer identifizieren und von den anderen unterscheiden kann ist ein Workaround nötig. Auf der untersten Netzwerk-Ebene kann das über die Verbindungsdaten erfolgen (IP-Adresse, Portnummer, usw.). In der Regel werden bei Diensten, die zunächst ihre Leistungen kostenlos anbieten auf diese Weise Einschränkungen wie Drosslungen vorgenommen, um den Kunden mit dem Kauf eines Premium-Diensts zu locken. Das Filtern einer Dienstleistung an Hand der IP-Adresse wird als „Geoblocking“ bezeichnet. Jedoch können sich hinter einer IP-Adresse mehrere Clients oder Benutzer befinden. Zudem wird bei IPv6 für die Identifizierung ganze Adressbereiche genutzt. Bei 340 Sextillionen Adressen ist es unmöglich alle Zustände gleichzeitig zu speichern.

Auf dieser Grundlage ist es also nicht möglich einen einzelnen Nutzer sicher zu identifizieren. In der Reger werden dafür *Cookies* eingesetzt. Das sind kleine Dateien die der Browser durch die Befehle vom Server auf der Festplatte des Nutzers hinterlässt und bei späterem Webseitenaufruf wieder ausliest. Damit lassen sich ebenfalls Schlüssel-Wert-Paare erzeugen. Aus technischer Sicht wird bei statischen Seiten diese Dateien durch das gesetzte Attribut „Cookie: “ im HTTP-Header realisiert. Sowohl bei der Anfrage als auch bei der Antwort vom Server wird dieser Wert übertragen, wenn ein Cookie gesetzt ist.

Rein statische Webseiten werden eher seltener angeboten. Die Mehrzahl der heutigen Seiten sind dynamisch aufgebaut. Das sind Seiten deren Inhalt sich nachträglich ändert, ohne dass eine neue Seite vom Server angefordert wird. Der Server hat dazu einen Quelltext mit Anweisung mit in der Seite eingebettet, der nachträglich im Browser ausgeführt wird. Diese Anweisungen können ebenfalls, ganz ohne HTTP-Header, Cookies setzten oder auslesen.

Cookies stehen in der Kritik weil sie von der Werbe-Industrie eingesetzt werden, um das Verhalten von Nutzern auszuspionieren. In Cookies können auch sensible Daten stehen. Um ein Missbrauch von Cookies vorzubeugen sollte ein Nutzer diese regelmäßig löschen.

Neben Cookies werden auch so genannte *„hidden-fields“* eingesetzt, um Zustände über mehrere Anfragen hinweg zu speichern. Das sind kleine Webseiten-Elemente die der Server in die Webseite einbettet, die der Nutzer aber nicht sehen kann. Die Seite ist so programmiert, dass der Browser dieses Feld bei der nächsten Interaktion zurück sendet. Der Vorteil besteht darin, dass keine Dateien beim Nutzer dauerhaft hinterlegt werden. Jedoch hat das Verfahren auch einen Nachteil. Schließt der Nutzer diese Seite oder öffnet er einen neuen Tab geht die Informationen im Feld verloren.

Das HTTP-Protokoll, die verschiedenen MIME-Typen und die Cookies bilden die Grundlage für Webseiten. Was fehlt ist eine Beschreibungssprache für die Darstellung einer Seite. Dies ist in einem Web-Standard beschrieben und ist Gegenstand des nächsten Kapitels.

2.1.3 Web-Standards

Das übersenden von Daten mit den MIME-Type `text/html` reicht nicht aus um eine Webseite zu beschreiben. Sie muss in einen Format vorliegen, welches der Browser versteht. Weltweit gesehen brauchen Webentwickler ein Format welches standardisiert und für Menschen lesbar ist.

Dafür ist die Beschreibungssprache *HTML* („Hypertext Markup Language“) zu ständig. Es ist eine Unterform von XML, eine Beschreibungssprache welche die Grundlage für eine gemeinsam lesbare Sprache zwischen Mensch und Computer bildet. Sie ist hierarchisch aufgebaut. HTML spezialisiert diese Sprache mit Webseiten-Elementen.

Ein HTML Webseiten-Element wird über eine Zeichenfolge beschrieben, die als *Tag* bezeichnet wird (englische Aussprache: [tæg]). Das sind kleine Kürzel in Spitzen klammern. Es existiert ein einleitendes und ein abschließende Tag, die gemeinsam einen umschlossenen Bereich bilden. Das abschließende Tag hat zusätzlich einem Schrägstrich vor dem Kürzel innerhalb der eckigen Klammern. Je nach verwendetem Kürzel sind die Auswirkungen auf den angegebenen Bereich unterschiedlich. Beispielsweise erzeugt das <h1>-Tag eine große Überschrift:

```
<h1>Das ist eine Überschrift</h1>
```

Für die bloße Textgestaltung gibt es Tags wie <b> für Fettschrift, <i> für kursive Schrift oder <u> für einen unterstrichenen Schriftzug. Nicht jeder Tag muss abgeschlossen werden. Der Tag
 für den Zeilenumbruch ist so ein Beispiel.

Tags können geschachtelt werden. Innerhalb eines Block-Elements können so weitere Blöcke definiert werden. Ein Block wird durch das -Tag eingeleitet und dient zur visuellen Unterteilung der Webseite.

```
<div>
      <div>
      </div>
      <div>
      </div>
</div>
```

Ein Tag kann ein oder mehrere *HTML-Attribute* besitzen:

```
<canvas id="graph" width="100" height="100">
    Das Element ist nicht darstellbar.
</canvas>
```

In diesem Beispiel wird eine Leinwand erzeugt. Der Zugriff auf die Fläche geschieht über die ID „`graph`“ und die Leinwand ist 100 mal 100 Pixel groß. Wird die Leinwand vom Browser nicht unterstützt, werden die <canvas>-Tags ignoriert. Das hat zur Folge, dass der von dem Tag umgeschossene Bereich unverändert dargestellt wird. In diesen Fall die Fehlermeldung „Das Element ist nicht darstellbar“.

Jedes HTML-Element ist mit seinen Funktionen im Web-Standard beschrieben. Er sorgt für eine möglichst geringe Inkompatibilität zwischen den verschiedenen Browsern.
Aber wer gibt diese Normen vor?

Zuständig dafür ist das „World Wide Web Consortium“ Gremium oder kurz *W3C*. Es wurde von Tim Berners-Lee gegründet. Ursprünglich ging es im WWW nur darum den Zugang zu wissenschaftlichen Texten zu vereinfachen. Inzwischen bauen ganze Anwendungen auf die W3C-Standards auf.

Die Standardisierung von HTML ist ein langwieriger mehrstufiger Prozess. Darin sind zwischen 300 bis 350 Mitgliedsorganisationen beteiligt. Im Grunde kann jede Organisation Mitglied werden und Vorschläge einreichen, solange diese einen jährlichen Mitgliedsbeitrag zahlt. Ein Vorschlag wird zunächst nur intern diskutiert. Danach wird eine Arbeitsgruppe gegründet und es entsteht ein Arbeitsentwurf. Diskutiert wird in dieser Phase öffentlich. Man spricht auch von einem „*Working Draft*“, also einen unfertigen Standard. Ist auch dieser Schritt abgeschlossen wird geprüft ob der Standard vollständig umsetzbar ist und allen technischen Anforderungen genügt. Bei der letzen Phase ist der Standard fertig und wird vom W3C empfohlen.

Eine Standardisierung kann tausende von Seiten umfassen und Jahre in Anspruch nehmen. Die Beschreibung muss zu 100% interpretier frei sein. An den neuen HML5 Standard wird beispielsweise seit 2004 gearbeitet und wird voraussichtlich 2014 verabschiedet (Stand 2013).

Auch wenn ein fertiger Standard vorliegt, heißt dies nicht, dass dieser von einem Browser eins zu eins umgesetzt wird. Dies wird besonders in den Anfangszeiten des Internets deutlich. Dort tobte ein Browserkrieg (*„browser war“*). Es ging um die Vorherschaft, den Microsoft durch Ausnutzung seiner Monopolstellung durch sein Betriebssystem gewann. Das war für die Webentwickler keine leichte Zeit, weil sie zwingend eine Webseite mehrspurig entwickeln mussten, damit diese nicht nur von einem Browser unterstützt wird. In der jetzigen Zeit herrscht zwar wieder ein (zweiter) Browserkrieg, jedoch bekommt der Nutzer beim Ansurfen einer Seite wenig davon mit. Der Kern des Standards wird größtenteils eingehalten. In diesen Krieg geht es mehr um den Marktanteil.

Obwohl heute sich die Browser stärker an Normen halten, variieren manche Elemente je nach Browser in der Darstellung oder werden nicht oder erst sehr spät umgesetzt. Es kommt auch vor, dass Browser Hersteller ihre eigenen Funktionen implementieren. Eine Abfrage des Mausrads erfordert zum Beispiel für jeden Browser einen eigenen Quelltext. Kommen bestimmte Funktionen bei den Nutzern gut an, werden sie meist von anderen Browsern übernommen.

Anders herum kann ein Browser schon Elemente unterstützten bevor ein Standard fertig vorliegt. So können Entwickler „schon mal testen“ und der Hersteller ist der Konkurrenz ein Stückchen voraus. Für die Gestaltung einer Webseite haben die Hersteller speziell dafür eigene CSS *Vendor Präfixe* eingefügt. Darüber kann die browserspezifische Implementierung einer unfertigen CSS Eigenschaft abgefragt und genutzt werden. Dazu wird vor der entsprechenden Eigenschaft das Browser-Kürzel -webkit-, -moz-, -ms- oder -o- vorangestellt. Ein Effekt ist dadurch früher verfügbar, jedoch ist die Implementierung aufwändiger und im ungünstigsten Fall wird ein Vorschlag wieder verworfen.

Browser Hersteller können daher nur die Funktionen früher umsetzen, von denen sie wissen, dass sie sich bis zur Finalisierung nicht ändern. Ansonsten entstehen unnötige Entwicklungskosten und Webentwickler kämpfen mit Inkompatibilitäten.

Der hier thematisierte unfertige HTML5 Standard ist so ein Fall. Darin wird der <input>-Tag um weitere Funktionen erweitert. Eine clientseitige programmierte Validierung der Eingabe wird dadurch überflüssig. Wird der Typ über ein HTML-Attribut auf „`color`“ gesetzt, erzeugt das eine Farbauswahlbox des Betriebssystems. Nicht jeder Browser hat diese Funktion bereits umgesetzt.

Während im Chrome in der Version 28 ein Button erscheint, der das Popup-Fenster öffnet:

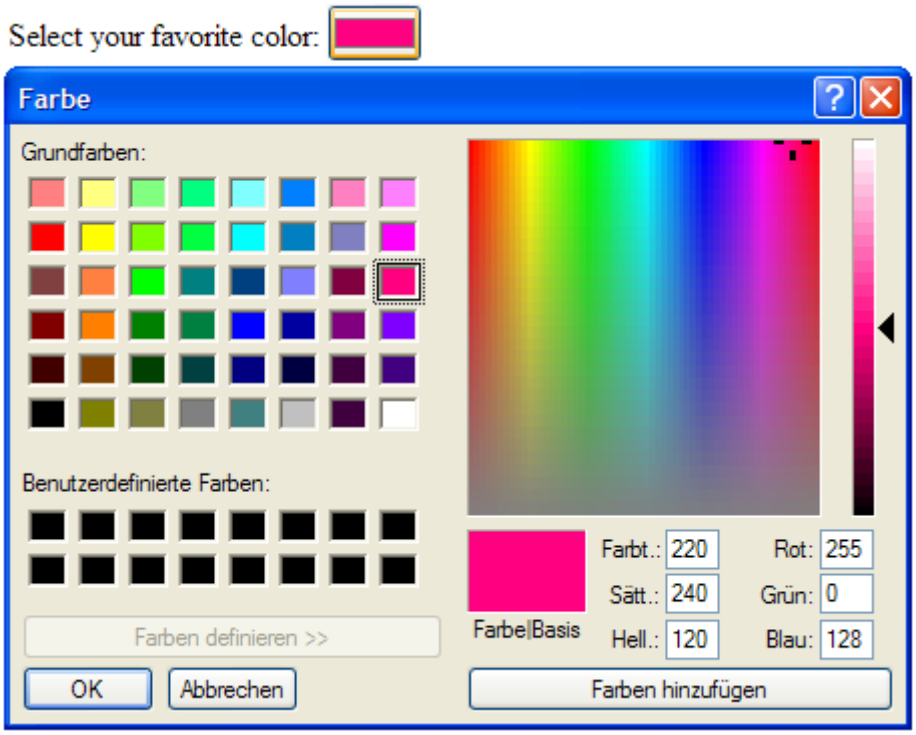
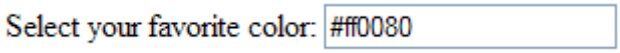


Abb. 1: Farbauswahlbox unter Windows XP

Wird in Firefox in der Version 22 nur ein Textfeld angezeigt, indem der Nutzer den CSS-Wert manuell einträgt:



Für Laien ist diese Variante schwieriger in der Bedienung.
Eine Lösung wäre eine Browserweiche die auf einen selbst erstellten Colorpicker zurückgreift. Dafür existieren fertige *Frameworks*. Ein Framework ist ein Packet mit vorgefertigten Funktionen die man nur einbinden braucht, um diese zu nutzen. Jede Anwendung, auch der Browser selbst oder ein Server, ist auf vorgefertigten Funktionen angewiesen. Hier in dieser Arbeit wird jedoch ein eigener Colorpicker entwickelt.

2.1.4 Aufbau einer Webseiten

Der Browser bekommt eine Webseite in Form von HTML. Der Funktionsumfang wird durch einen Standard vorgegeben. Aber wie ist eine Seite konkret aufgebaut?

Nach dem aktuellen Stand sieht der grundlegende Aufbau wie folgt aus:

```
<!DOCTYPE html>
<html>
    <head>
        <style type="text/css"></style>
        <script type="text/javascript"></script>
    </head>
    <body>
    </body>
</html>
```

Die erste Zeile signalisiert dem Browser das im Dokument HTML5 genutzt wird.
Bei diesem Beispiel fehlen Informationen wie Titel oder Zeichensatz. Diese werden in den <head>-Bereich geschrieben:

```
<title>Titel der Seite</title>
<meta charset="UTF-8">
```

Der <meta>-Tag dient dazu weitere Informationen über die Seite anzugeben, wie Autor oder Hinweise für Mobile-Geräte. Über „UTF-8“ nutzt der Browser deutsche Umlaute.
Es gibt auch Angaben die dafür sorgen, dass die Seite nicht in den Suchtreffern einer Suchmaschine landet:

```
<meta name="robots" content="nofollow">
```

Im <body>-Bereich wird der Inhalt eingetragen, der den Nutzer letztendlich zu Gesicht bekommt. Reine HTML-Elemente sind in ihren Gestaltungsmöglichkeiten begrenzt. Dafür gibt es *Cascading Style Sheets*, kurz CSS. Das sind eine Folge von Kürzel die bestimmte HTML-Elemente *selektieren* und ihnen visuelle Eigenschaften über Schlüssel-Werte-Paare zuweisen. Eingeleitet werden diese mit den <style>-Tag in der Regel im <head>-Bereich. Entweder werden diese direkt in der HTML-Datei geschrieben oder es wird ein Verweis zu einer ausgelagerten Datei angegeben.

CSS Werte können auch per HTML über das style-Attribut gesetzt werden. Dies ist jedoch unüblich da diese eingetragenen Werte nachträglich nur schwer zu überschreiben sind.
In dieser Arbeit wird CSS eingesetzt. Daher seien die 3 wichtigsten Selektoren genannt:

1. Wird ein Bezeichner angegeben, der nur aus Buchstaben des englischen Alphabets besteht, so wird ein bestimmter Tag selektiert und zwar jeder der im Dokument vorkommt.
2. Das Raute-Zeichen »#« symbolisiert vor einem Bezeichner eine bestimmte ID. Damit ist das Element gemeint, bei dem das Attribut „id“ mit dem entsprechendem Attributwert gesetzt ist. Eine ID kann nur einmal vergeben werden.
3. Beginnt ein Bezeichner mit einen Punkt ».« so ist eine Klasse gemeint. Ähnlich wie bei Punkt 2 wird nach dem Attribut „class“ gesucht. Hier werden jedoch *alle* Elemente selektiert wo der Klassenname mit dem Bezeichner übereinstimmt. Bei Klassen ist eine Mehrfachnennung möglich.

Nach dem Selektor-Ausdruck folgt ein offene und geschlossene geschweifter Klammer. In diesen stehen die Eigenschaften die vergeben werden soll. Das folgende Beispiel soll den Einsatz von CSS verdeutlichen:

```
<!DOCTYPE html>
<html lang="de">
<head>
	<title>2 Bereiche</title>
	<meta charset="utf-8">
	<style type="text/css">
		html, body { height: 100%; }
		div { padding: 16px; }
		#header { border: 6px double black; text-align: center; }
		.sidebar { height: 50%; float: left; background: black; color: white; }
	</style>
</head>
<body>
	<div id="header">Überschrift</div>
	<div class="sidebar">Navigation</div>
</body>
</html>
```

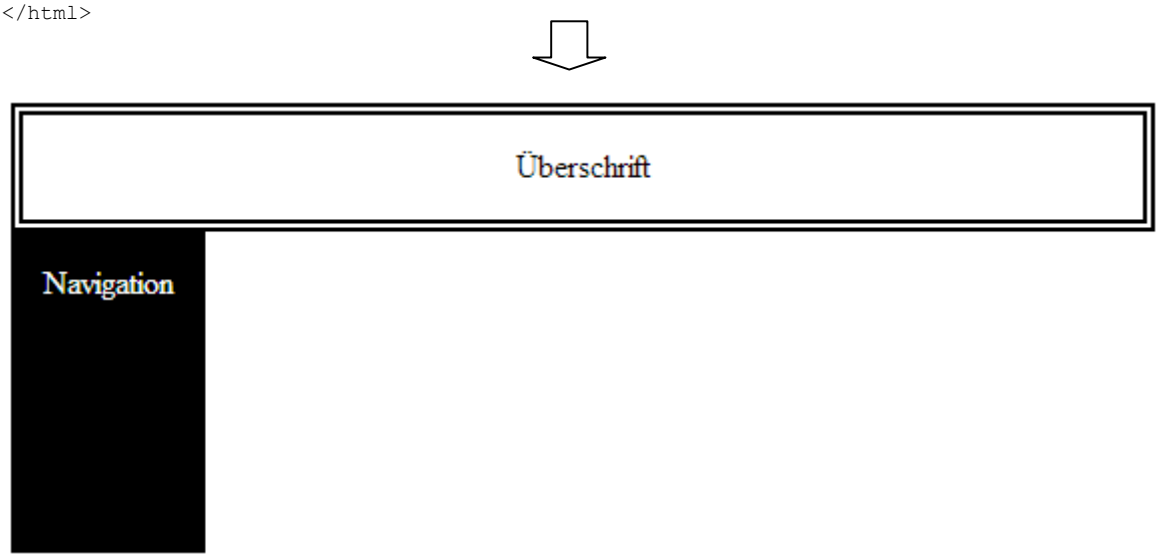


Abb. 2: Einsatz von CSS am Beispiel von zwei DIV-Containern

Der <script>-Bereich ist aus heutiger Sicht nicht mehr wegzudenkender. Auch hier kann man den Inhalt in einer separaten Datei ausgelagert. Es handelt sich um **JavaScript**. Eine Skriptsprache die 1995 entstand und die innerhalb des Browser ausgeführt wird. Sie dient als Grundlage für dynamische Webseiten. In den Anfangszeiten wurde sie für kleinere Animationen eingesetzt. Heute bauen komplette Anwendungen auf dieser Sprache auf.

Es existieren zahlreiche JavaScript Techniken die sowohl Vorteile als auch Nachteile mit sich bringen. Wird JavaScript direkt im <body>-Bereich eingebunden wird der Quelltext sofort ausgeführt. Bei nachgeladenen Skripten kann das von Vorteil sein. Ein höheres Risiko besteht dann, wenn das Skript aus einer nicht vertrauten Quelle stammt.
JavaScript Anweisungen lassen sich direkt in einen Link einbetten. Dazu wird in der URL „javascript:“ vorangestellt. Das erleichtert die Webprogrammierung. Es erleichtert jedoch auch das Unterschieben eines Schadcodes.

Mit der Zeit wurde JavaScript um einige Funktionen erweitert. Die wichtigste Funktion ist das Ändern des HTML-Inhaltes. Der sichtbare Bereich im <body> bildet wie schon erwähnt eine Baumstruktur. Diesen zu verändern ist ohne weitere Hilfsmittel umständlich. Um den Zugriff zu vereinfachen wurde eine einheitliche Schnittstelle geschaffen. Diese Schnittstelle zusammen mit der Beschreibung des Baums nennt sich **Document Object Model**.

Es vereinfacht den Umgang mit der HTML Darstellung. Das Parsen von XML über JavaScript fällt damit weg. Im Folgenden wird nur noch von **DOM** die Rede sein.

Die zweite wichtige Erweiterung ist AJAX. Es ist die Grundlage für Web 2.0 Anwendungen. Es ermöglicht Ressourcen zu einen späteren Zeitpunkt vom Server anzufordern (Asynchronous JavaScript and XML). Zunächst wurde XML als Datenformat genutzt, inzwischen werden auch andere Formate und verschiedene Protokolle unterstützt.

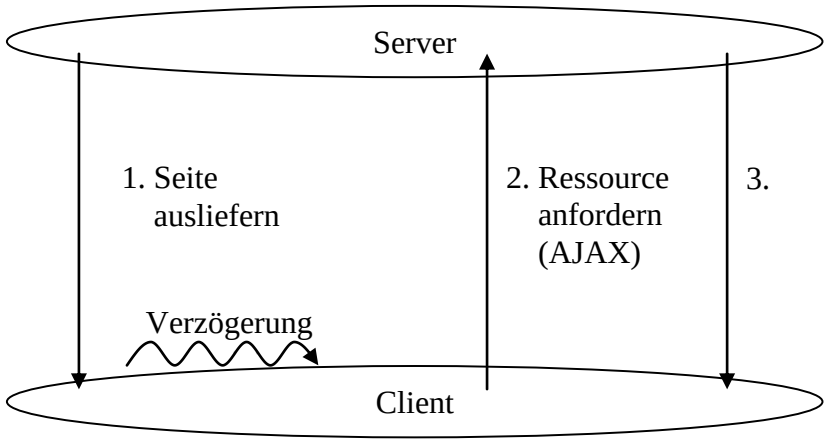


Abb. 3: Ablauf einer AJAX-Anfrage

Zusammen mit DOM werden kleine Inhalte vom Server nachgeladen und in der Seite dargestellt, ohne dass der komplette Verbindungsaufbau von vorne beginnt. Weil sich die Seite sich nicht neu aufbaut und die Nutzeroberfläche nicht blockiert, bekommt der Nutzer das Nachladen der Ressource nicht mit. Ein typisches Anwendungsbeispiel ist das Anzeigen von Vorschlägen in einem Suchfeld.

2.2 Webdienste und Web 2.0

Webanwendungen sind Anwendungen die innerhalb des Browsers laufen. Die Anwendungslogik wird vom Server angeboten. Wird die Anwendung erweitert oder gewartet geschieht das zentral einmalig auf dem Server. Die Nutzer brauchen sich nicht um die technischen Details kümmern.

Es gibt verschiedene Typen von Webanwendungen. *Rich Internet Application* sind Webanwendungen, wo der größte Teil an Berechnungen auf dem Client ausgelagert ist. Nachdem der Server die Anwendung verschickt hat, werden kaum noch Berechnungen für den Client übernommen. Die hier zu entwickelnde Bibliothek ist so ein Typ.

In Laufe der Zeit hat sich der Begriff »Web 2.0« herauskristallisiert. Was zunächst ein Modewort war, ist inzwischen fester Bestandteil von Webanwendungen geworden. Damit ist die veränderte Nutzung des Internets gemeint. Dabei spielen zwei Aspekte eine Rolle: Die Einbeziehung von Nutzeraktivitäten und die technische Voraussetzung dafür. In den Anfangszeiten des Internets hat ein Anbieter eine statische Seite mit Informationen eingerichtet und der Internetgemeinde zur Verfügung gestellt. Der Nutzer hat im klassischen Web nur die Möglichkeit Inhalte zu konsumieren. Ein Feedback zur Informationsquelle oder der direkte Austausch zwischen anderen Nutzern gab es nicht. Die Informationsquelle ist der Seitenbetreiber.

Im Web 2.0 Zeitalter ist jedoch der Nutzer einer Seite derjenige, der den Inhalt erzeugt. Der Seitenbetreiber stellt den Nutzern nur die dafür nötige Dienstleistung zur Verfügung. Durch starke Verknüpfung zwischen den Nutzern untereinander entsteht eine „kollektive Intelligenz“. Soziale Netzwerke, Wiki’s oder Blog’s mit einer Kommentar-Funktion sind typische Beispiele für eine Web 2.0Anwendung.

Es kommt zu einer Verschiebung bei der Produktion neuer Inhalte. Anstatt wenig einzelne Personen viel Inhalt für die breite Masse veröffentlichen, erzeugt im Web 2.0 jeder Nutzer wenig Inhalt. Diese werden über einen gewissen Zeitraum massiv parallel erzeugt und ausgetauscht. So entsteht der gleiche Umfang an Inhalt bei geringerem Arbeitsaufwand des Einzelnen.

Die kollektive Intelligenz erfordert eine andere technische Voraussetzung. Nur mit statischen Seiten ist die Einbeziehung der Gemeinde nicht möglich. Es werden *dynamische* Internetseiten benötigt. Bevor die Seite zum Client verschickt wird, ist eine dynamische Anpassung an den Nutzer erforderlich. Ist die Seite übertragen und im Browser dargestellt, braucht es eine Technik, die auf Nutzeraktivität reagiert. Die Darstellung wird clientseitig an den Interaktionen des Nutzers anpasst, ohne eine komplett neue Seite vom Server anzufordern. Dafür wird JavaScript und AJAX eingesetzt.

Um dynamische Seiten auf den Server zu erzeugen wird in den meisten Fällen die Sprache PHP eingesetzt. JavaScript und PHP werden interpretiert, sind quelloffen und schwach typisiert. Das heißt der Quelltext liegt bei der Ausführung unverändert im Klartext vor und welchen Datentyp eine Variable zur Laufzeit hat lässt sich schwer herleiten. Erst zur Laufzeit, wenn die Seite auf dem Server generiert wird oder wenn der Client den JavaScript-Quelltext erhält, wird dieser geparst und dadurch lauffähig gemacht. Auf dem Server braucht das nur einmal zu geschehen, weil sich dort der Quelltext nur selten ändert. Alle Clients sollen dieselbe Dienstleistung bekommen.

Auf dem Client wird der JavaScript Quelltext meist immer neu interpretiert. Ein nativer Code, also ein Quelltext der Maschinebefehle enthält, hat im Vergleich zu interpretierten Quelltext einen Geschwindigkeitsvorteil. Browserhersteller versuchen daher per Just-in-time Kompilierung die JavaScript Anweisungen in plattformnahen Code umzuwandeln.

Dabei müssen sie bei ihrer JavaScript-Engine abwägen. Entweder soll der JavaScript Quelltext ausgiebig optimiert werden, dann laufen nachträgliche ausgeführte Anweisungen innerhalb der Seite auf dem Client schneller und die Ladezeit verlängert sich. Oder der JavaScript Quelltext bleibt unoptimiert. Die Seite wird dann schneller geladen und dargestellt, alle weiteren Interaktionen laufen dann aber langsamer.

Eine echte Alternative zu JavaScript für clientseitige Berechnungen ohne Plug-Ins gibt es nicht. Es gibt zwar andere Sprachen wie Visual Basic Script oder die vom Unternehmen Google vorgestellte Alternative „Dart“. Diese laufen jedoch nur auf bestimmten Browsern oder müssen nachinstalliert werden. Einige stellen ein Sicherheitsproblem dar. JavaScript ist die einzige Sprache die jeder Browser versteht und die in der Standardeinstellung bereits aktiviert ist. Dazu kommt der hohe Verbreitungsgrad bei bestehenden Webseiten.

Webanwendungen sind im Vergleich zu Desktopanwendung plattformunabhängig und brauchen kein bestimmtes Betriebssystem. Für eine Web 2.0 Anwendung sind keine nachträglichen Installationen für deren Lauffähigkeit nötig. Egal welchen Browser genutzt wird, prinzipiell reicht das Ansteuern einer URL aus um eine Webanwendung zu nutzen. Der Funktionsumfang ist auf den vorhandenen Standard eingeschränkt. Desktop-Anwendungen sind dagegen auf einer Zielplattform zugeschnitten und müssen erst installiert werden. Sie sind nicht an einen Standard gebunden sondern liegen in Maschinensprache vor und können so direkt den höheren Funktionsumfang des Betriebssystems nutzen. Die Oberfläche wird ohne Umwege mit speziellen Grafikbefehlen erzeugt und die für eine Netzwerkkommunikation nötigen Sockets sind bereits integriert. Im Browser wird eine Bedienoberfläche durch HTML Elementen nachgebildet, der Funktionsumfang ist somit fest vorgegeben. Dem gegenüber steht die Tatsache, dass eine Webanwendung ohne Hürden weltweit zur Verfügung steht und genutzt werden kann.

Webseiten können mit weiteren Objekten und mit neuen Funktionen nachgerüstet werden. Das sind Erweiterungen, also Plug-Ins, die innerhalb des Browsers installiert werden. Genau genommen bekommen der Browser und das Betriebssystem die neuen Funktionen. Solche Plug-Ins gehören nicht zum Web-Standard und für jeden Browser existiert für dieselbe Erweiterung eine unterschiedliche Installationsroutine. Plug-Ins können den Zugriff auf einen Proxy Server erleichtern, einen Video-Stream verschlüsseln, auf die Grafikkarte zugreifen oder bieten eine direkte Netzwerkkommunikation an. Bei einem Browser Plug-In hängt die Wartung und Umsetzung allein vom Hersteller des Plug-Ins ab. Nutzer müssen gegebenenfalls selber neue Updates einspielen. Somit ist man vom Hersteller abhängig. Bei einem Web-Standard hingegen ist die Nutzung der bereitgestellten Funktionen immer kostenlos. Jedoch ist der Entwickler hier auf die richtige Umsetzung durch den Browserhersteller angewiesen.

Wird ein Plug-In dafür genutzt um neue Elemente im Browser anzuzeigen, ist nach der Installation ein neuer Dateitypen verfügbar der über den allgemeinen <object>-Tag eingebunden wird. Ursprünglich wurde dafür spezielle Tags verwendet, wie das <applet>-Tag für ein Java Applet. Prinzipiell können so beliebige Dokumente in die Webseite eingebunden werden. Es ist auch möglich eine weitere Webseite mit einer anderen URL einzubinden.

Ein Browser Plug-Ins kann für den Nutzer ein erhebliches Sicherheitsrisiko darstellen. Ist auf dem Client das Plug-In „Adobe Flash“ installiert, können *Evercookies* gesetzt werden. Das sind Cookies die über mehrere Seiten hinweg gelesen und gesetzt werden können. Eine Löschung über den Browser ist jedoch nicht möglich.

Die gefährlichste Sicherheitslücke ist 2012 über das „Java“ Plug-In aufgetreten, die bis heute nicht ausreichend geschlossen wurde. Das bloße Ansurfen einer präparierten Seite reicht aus, um aus der Java Sandbox auszubrechen. Danach kann ein Angreifer beliebe Anweisungen auf dem Client ausführen. Der Nutzer bekommt davon nichts mit. Der BKA-Trojaner, der den Computer mit einer drohenden Falschmeldung sperrt, wurde auf dieser Weise verteilt.

Je nach Plug-In werden unterschiedliche Dateiformate genutzt. Bei Flash ist es die Dateiendung „*.swf“ und bei Java „*.class“. Will ein Entwickler so ein Format erstellen, muss er die Lizenzbedingungen vom jeweiligen Hersteller akzeptieren. Manche Formate sind kostenlos, bei anderen Formaten fallen unter Umständen Lizenzgebühren an oder die dazugehörige Entwicklungsumgebung ist kostenpflichtig. Bei jedem neuem Format ist der Quelltext ohne weiteres nicht einsehbar.

Bei Desktop-Anwendungen ist es oft verboten den kompilierten Maschinencode einer Bibliothek zu dekompilieren, um an den Quellcode zu gelangen [5]. Sei es um das Verhalten der Anwendung zu ändern oder nur für ein Selbststudium. In Lizenzverträgen wird dies explizit deutlich gemacht.

Bei Webanwendungen ist das anders. Im Web liegt der Quelltext einer Anwendung unverändert vor und ist vom jeden frei änderbar. Die quelloffene serverseitige Skriptsprache PHP ist lizenzfrei nutzbar. Auf dem Client reicht ein Rechtsklick im Browser aus, um zu sehen wie eine Seite aufgebaut und programmiert ist. Der Browser ist für den Entwickler Darstellung, Laufzeitumgebung, Entwicklungswerkzeug und Debugger in einem. Sowohl für CSS als auch für JavaScript sind genügend Werkzeuge in allen gängigen Browsern vorhanden um Änderungen an einer Seite zur Laufzeit durchzuführen. Für den Nutzer hat das den Vorteil, dass er bestehende Funktionen für seine Anwendung ohne Hürden weiterverwenden kann.

Ein Programmierer erstellt nicht jedes Mal eine Anwendung von Grund auf neu, das ist oft zu Zeit aufwändig und umständlich. Er ist auf eine fertige Programmbibliothek angewiesen. Eine Anwendung, ein Betriebssystem oder eine Programmiersprache selbst bietet von Haus aus ein Satz von fertigen Funktionen in Form einer Bibliothek an. Der Web-Browser oder der Server selbst nutzt solche Betriebssystemfunktionen um auf Basis von Netzwerk Sockets eine Verbindung herzustellen.

---

[5] Prof. Dr. Bernhard Bergmans und Prof. Dr. Ralf-Michael Marquardt,
"Schutz vor Reverse Engineering im deutschen Recht", Logos Verlag Berlin, abgerufen am 22.9.2013
http://www.logos-verlag.de/ReWirPDF/ReWir-11-2012.pdf

Im Web werden auf Open-Source Basis gezielt Bibliotheken für Weiterentwicklungen angeboten. Als Beispiel soll die meistverwendete JavaScript Bibliothek „jQuery“ dienen. Jede zweite Seite verwendet diese Bibliothek [6][7] (Stand 2013). Sie wird hauptsächlich dazu eingesetzt DOM-Manipulationen durch einen CSS-Stil zu vereinfachen. Mit ihr können auch JavaScript Animationen erstellt werden. Sie beinhaltet zudem eine AJAX-Bibliothek.

jQuery bietet die Möglichkeit an, innerhalb der Bibliothek den Funktionsumfang zu erweitern. Das Grundgerüst sieht wie folgt aus:

```
(function ($)
{
    $.fn.MeinPlugin = function ()
    {
        // Funktion hier implementieren
        return this;
    };
}(jQuery));
```

Diese Plug-Ins können dann zwischen den Entwicklern ausgetauscht, weiter entwickelt und veröffentlicht werden.

Nicht nur clientseitig werden Nutzer über solche Konzepte mit einbezogen. Auf Servern werden Content Management Systeme (CMS) eingesetzt. Sie dienen dazu Darstellung, Programmierlogik und Funktionsumfang sauber zu trennen. Funktionen können auch dort über Plug-Ins nachinstalliert werden. Nimmt man das auf PHP aufbauende CMS „Wordpress“, so sind derzeit rund 25000 Plug-Ins von Drittanbieter für dieses System verfügbar [8].

Das im Web alles offen liegt und leicht abänderbar ist hat auch seine Nachteile. Plug-Ins wie auch ein CMS selber können Sicherheitslücken beinhalten. Kostenlose Webanwendungen, wie zum Beispiel Dienste für Geodaten oder Adress-Validierung, die auf JavaScript aufbauen können nur schwer vor einen Missbrauch oder einer Weitergabe geschützt werden. Video-Stream Anbieter können de facto nicht nachprüfen ob die übermittelten Daten dauerhaft beim Nutzer gespeichert werden oder nicht. Dies stellt für das Urheberrecht ein Problem dar. Nach derzeitiger Rechtsprechung sind nur flüchtige Kopien zulässig. Wer einen „wirksamen Kopierschutz umgeht“ (§ 95a Abs. 2 UrhG) begibt eine Straftat. Baut eine Software auf Open-Source mit mehreren losen Entwicklern auf, ist die genaue Abgrenzung der Haftung nicht abschließend geklärt.

---

[6] "Usage of JavaScript libraries for websites", W3Techs, abgerufen am 22.9.2013 http://w3techs.com/technologies/overview/javascript_library/all

[7] "jQuery now runs on every second website", W3Techs, abgerufen am 22.9.2013 http://w3techs.com/blog/entry/jquery_now_runs_on_every_second_website

[8] "Wordpress Plugin Directory", WordPress.org, abgerufen am 22.9.2013 http://wordpress.org/plugins/

Das W3C liegt inzwischen ein Vorschlag für ein digitales Rechtekontrollmanagement (DRM) mit einer Ende-Zu-Ende Verschlüsselung vor [9]. Das enthaltene Protokoll „Encrypted Media Extensions“ kann dazu eingesetzt werden die Rechte des Urhebers besser zu schützen. Es ermöglicht das Abspielen von geschützten DRM Inhalten und es bietet den Anbietern die Möglichkeit mehr Einfluss auf das Endgerät zu nehmen.

Ein anderer Aspekt der sich aus der Nutzung von offenem Quelltext ergibt, ist die Sicherheit. Wird ein manipulierter Link angeklickt, kann der Nutzer zwar die gewünschten Seite erhalten und der Browser zeigt eine identische URL an. Jedoch kann die dargestellte Seite gefälscht sein („Link-Spoofing“). Ein normaler Nutzer findet keinen Unterschied. Das machen sich Betrüger zu nutzen und verschicken massenhaft Links zu Phishing-Seiten per E-Mail.

Ein Nutzer kann durch einen falschen Klick unbemerkt einen eingeschleusten Code ausführen. Durch das Anklicken eines Links kann er (unwissentlich) sich an einer DDoS Attacke beteiligen, was in manchen Ländern strafbar ist. Nach dem Landgericht Düsseldorf handelt es sich um Computersabotage[10].

Bei dieser Attacke überfluten mehrere Clients einen Server mit Anfragen, um ihn lahm zu legen. Technisch wird dafür die DOM-Manipulation genutzt. Es wird wiederholt die zu attackierende Seite des Servers per <iframe>-Tag in den DOM-Baum des Nutzers eingebettet und zwar, sodass der Nutzer diesen Bereich nicht sieht. Das kann durch den CSS-Eigenschaft „`display: none;`“ geschehen.

Eine andere Variante nutzt eine statische Seite mit mehreren identischen i-Frames die sich selbst über die Meta-Information `<meta http-equiv="refresh" content="1">` immer wieder neu lädt. Diese Technik wird „iframe attack“ genannt. Eine effektivere Attacke erlangt man über die der neuen HTML5 Techniken „Cross Origin Requests“ und „WebWorkers“.

---

[9] David Dorwin, Adrian Bateman und Mark Watson,
"Encrypted Media Extensions - W3C Editor's Draft 17 September 2013",
World Wide Web Consortium, abgerufen am 22.9.2013
https://dvcs.w3.org/hg/html-media/raw-file/tip/encrypted-media/encrypted-media.html

[10] "Strafgesetzbuch - §303b Computersabotage", dejure.org, abgerufen am 22.9.2013
http://dejure.org/gesetze/StGB/303b.html http://openjur.de/u/165558.html

„Denial of Service“-Attacken können in unterschiedlichen Formen auftreten. Ein einziger schwacher Client kann ausreichen, um einen starken Server offline zu nehmen. Eine Methode zielt darauf ab einen Verbindungsaufbau nur vorzutäuschen. Jeder Verbindungsaufbau im TCP-Protokoll erfolgt über den 3-Wege-Handshake:

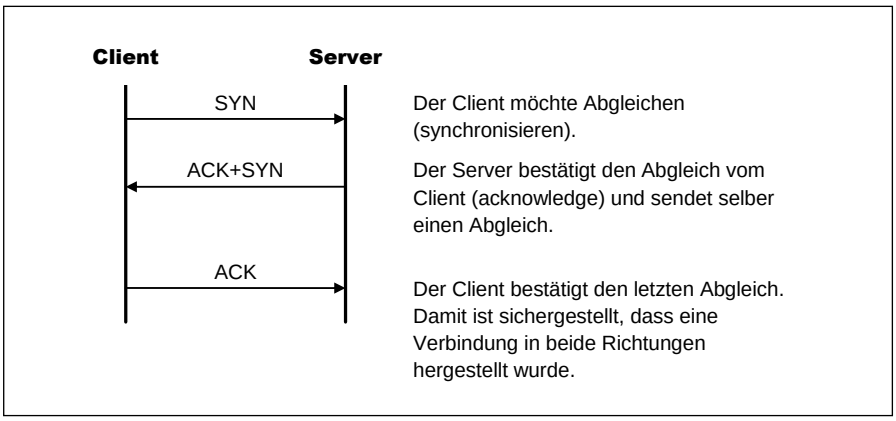


Abb. 4: TCP Drei-Wege-Handschlag

Der Angreifer führt den ersten Schritt wiederholt aus und ignoriert alle ankommenden Pakete vom Server. Der Server geht davon aus, dass sich mehrere Clients verbinden wollen und wartet auf die jeweiligen Bestätigungen. Dadurch läuft sein Speicher-Puffer über. Dieser einfache DoS-Angriff wird als „SYN-Flooding“ bezeichnet, weil der Server mit TCP-Paketen überflutet wird, bei denen das SYN-Flag gesetzt ist. Diese Attacke kann nur dann funktionieren, wenn der Server ungünstig konfiguriert ist.

Neben DDos-Attacken stellt das Einschleusen von Schadcode ein Sicherheitsproblem von Webanwendungen dar.
Eine weit verbreitete Methode um im Web einen Quelltext einzuschleusen ist das *Cross-Site-Scripting*, kurz XSS. Werden Benutzer-Eingaben vom Server nicht ausreichend geprüft und landet dieser unformatiert als Ausgabe auf einer Webseite, so liegt XSS vor.
Wird beispielsweise die Zeichenkette „`<script></script>`“ eingeschleust und landet dieser Text eins zu eins in den <body>-Bereich der Seite, so werden die Befehle innerhalb des <script>-Tags vom Browser als JavaScript-Code erkannt und ausgeführt. Mit Hilfe von XSS können so Passwörter, Cookies oder Session-Keys entwendet werden.

Wie leicht dies möglich ist, können Sie selbst testen. Loggen Sie sich auf eine beliebige Internetseite ein und geben Sie in der Browser-Konsole den JavaScript Befehl **alert(document.cookie)** ein.
Die Konsole ist beim Mozilla Firefox über Extras->Web-Entwickler->Web-Konsole und bei Google Chrome über Tools->JavaScript-Konsole erreichbar.
Noch kürzer geht das Abfangen durch die Eingabe **javascript:alert(document.cookie)** in die Adressleiste des Browsers.

Besitzt der Angreifer eine fremde Session, kann er unter fremden Namen Online-Geschäfte nachgehen. Ende 2010 waren auf 17 Banking-Seiten eine XSS-Lücke aktiv. In CMS-Plug-Ins als auch in den entsprechenden CMS selber können solche Lücken auftreten.

Es gibt weitere Sicherheitsaspekte (Same-Origin-Policy, Frame-Spoofing, die `event.preventDefault()`-Funktion, usw.) die in Verbindung mit einigen JavaScript Techniken ein Sicherheitsproblem darstellen. Nichtsdestotrotz ist die Laufzeitumgebung selbst solide aufgebaut.

JavaScript läuft in einer *Sandbox* ab. Sie sorgt dafür, dass die Anweisungen innerhalb des Browsers ausgeführt werden, und nicht in der Umgebung des Betriebssystems. Wäre dies möglich, könnte ein Webserver beliebe Anweisungen auf dem Rechner außerhalb des Browsers ausführen. Damit wäre es per JavaScript prinzipiell möglich auf Speichermedien zu zugreifen oder das Betriebssystem mit einem Virus zu infizieren.

In der Geschichte des Webs sind nur selten solch gravierende Ausbrüche bekannt geworden. In Verbindung mit Microsofts eigener Sprache JScript und in Verbindung mit ActiveX können solche Einschränkungen umgangen werden. Beide sind ausschließlich im Internet Explorer verfügbar.

Ist der verwendete Browser nicht auf den aktuellen Stand und somit die Sandbox nicht hundertprozentig abgedichtet, kann dies schon ausreichen um mehr über den Nutzer zu erfahren. So wurden 2013 über eine bereits geschlossene Sicherheitslücke Nutzer in einem anonymen Netzwerks identifiziert, in dem Hardwareinformationen ausgelesen wurden [11].

---

[11] "Tor-Nutzer über Firefox-Lücke verfolgt", Heise Zeitschriften Verlag, abgerufen am 22.9.2013 http://www.heise.de/newsticker/meldung/Tor-Nutzer-ueber-Firefox-Luecke-verfolgt-1930154.html

2.3 HTML5

Die bisher beschriebenen Techniken bilden bis heute unverändert die Grundlage für Webseiten. Der Funktionsumfang ist beschränkt und wird meist durch den Einsatz von Browser Plug-Ins erweitert. Diese Situation kann sich in nächster Zeit ändern. Momentan wird die HTML Spezifikation Schritt für Schritt um neue Funktionalitäten erweitert. Diese Neuerungen sind in diesen Kapitel Gegenstand.

Lange Zeit stand die Entwicklung von HTML still, bis 2006 Tim Berners-Lee, Leiter des W3C eine neue Arbeitsgruppe ankündigte. Daraus ist *HTML5* entstanden. Ursprünglich war 2004 noch von dem Begriff „Web Applications 1.0“ die Rede und dieser Vorschlag stammte auch nicht von der W3C selbst, sondern von der Arbeitsgruppe WHATWG. Diese wird von Unternehmen wie Mozilla Foundation betrieben und wurde gegründet weil die W3C nur langsam Standards entwickelt. Beide Arbeitsgruppen gehen unterschiedliche Wege.

Während reines HTML als Auszeichnungssprache für die Darstellung einer Seite zuständig ist, wird bei der HTML5 Spezifikation zusätzlich JavaScript benötigt um alle enthaltenen Funktionen zu nutzen. Es handelt sich um einen unfertigen Standard der voraussichtlich 2014 endgültig verabschiedet wird. Man spricht dann von einer *W3C Recommendation*. Einige HTML5 Elemente wurden in den gängigsten Browsern bereits umgesetzt.

Für eine bessere Strukturierung einer Webseite wurden neue Tags hinzugefügt. Bei Betrachtung der am häufigsten aufgerufenen Webseiten zeichnet ein Muster ab. Häufig werden für -Elemente die ID’s wie „`header`“, „`footer`“ oder „`article`“ verwendet. Diese werden dann für eine bessere Selektierung per CSS genutzt. Suchmaschinen können anhand solcher ID’s den Inhalt besser auswerten. In HTML5 gibt es speziell dafür eigene Tags wie <article> oder <header>. Ein <article> Element beschreibt zum Beispiel einen Bereich mit eigenständigem Inhalt. Dieser ist von dem Inhalt der restlichen Seite unabhängig und es ist in der Regel eine Auflistung davon vorhanden.

Benutzereingaben werden über <input>-Feldern entgegen genommen. Oft wird die Eingabe per JavaScript validiert, zum Beispiel um zu prüfen ob eine erforderliche Angabe fehlt oder ob eine E-Mail Adresse ein gültiges Format hat. Mit Hilfe von weiteren <input>-Attributen die im HTML5 definiert sind kann nahezu vollständig auf JavaScript verzichtet werden. Eine Angabe wie `type="number"` oder `required` im HTML-Element reicht aus.

Reines HTML dient dazu Text zu formatieren, Bilder und Hyperlinks einzubinden oder die Seite in Bereiche zu unterteilen. Der Funktionsumfang ist allein für eine statische Darstellung beschränkt. Multimedia Inhalte sind nur über zusätzliche Plug-Ins möglich. Diese Einschränkungen wurden in HTML5 aufgehoben. Per <audio> oder <video> Tag in Kombination mit JavaScript ist es möglich einen eigener Player zu entwerfen. Aussehen und Funktionalität ist beliebig anpassbar. Jedoch fehlt im Standard eine klare Vorgabe über die unterstützten Formate.

Über einen Video Stream kann mit Hilfe das <canvas>-Element ein Filter gelegt werden. Auch einzelne Pixel lassen manipulieren. Das <canvas>-Element ist eine Zeichenfläche welche verschiedene Grafikoperationen unterstützt. Die Pixelinformationen werden dabei immer wieder neu überschrieben.

Neben dem <canvas>-Element existiert noch das <svg>-Element um Grafiken (dynamisch) darzustellen. Dieser Grafiktyp ist im Vergleich zur <canvas>-Element skalierbar, das heißt sie können nachträglich vergrößert oder verkleinert werden, ohne dass es zu Qualitätsverlusten kommt. HTML5 erlaubt es direkt SVG Grafiken zu beschreiben und per DOM zu verändern. Bisher war ein Verweis zu einer bereits vorhandenen Grafikdatei nötig. Die Grafik wird von der hintersten Ebene bis hin zur vordersten durch Flächen, so genannte *Shapes*, beschrieben. Bereits vorhandene Shapes, auch die verdeckt sind, können nachträglich bearbeitet werden oder auf Ereignisse wie das Bewegen der Maus reagieren. Bei einer höheren Anzahl an Shapes wird automatisch der Overhead beim Zeichnen größer. Dies ist beim <canvas>-Element anders, weil es sich um eine Pixelgrafik handelt.

Um mathematische Ausdrücke darzustellen existieren spezielle Zeichensätze[12]. Für komplizierte Ausdrücke sind diese jedoch zu unflexibel. Bisher wurden dafür ebenfalls fertige Grafiken verwendet. HTML5 unterstützt die Darstellung von mathematischen Ausdrücken über der Beschreibungssprache MathML, eine Unterform von XML. So ein Ausdruck wird über das <math>-Tag eingeleitet.

$$A = \begin{bmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \end{bmatrix}$$

Um diese Matrix in einer HTML5-Webseite darzustellen recht ein MathML-Ausdruck aus:

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
    <mrow>
        <mi>A</mi><mo>=</mo>
        <mfenced open="[" close="]">
            <mtable>
                <mtr>
                    <mtd><mn>11</mn></mtd>
                    <mtd><mn>12</mn></mtd>
                    <mtd><mn>13</mn></mtd>
                </mtr>
                <mtr>
                    <mtd><mn>21</mn></mtd>
                    <mtd><mn>22</mn></mtd>
                    <mtd><mn>23</mn></mtd>
                </mtr>
            </mtable>
        </mfenced>
    </mrow>
</math>
```

Für bessere Gestaltungsmöglichkeiten einer Webseite existieren neue CSS Eigenschaften. Ein Teil davon befindet sich im Working-Draft. Über die Eigenschaft `transform` ist es möglich HTML-Elemente beliebig zu Drehen oder zu Transformieren, auch im dreidimensionalem Raum. Per `keyframes` können Animationen erstellt werden und per `transition` weiche Übergänge bei Wert-Änderungen. Für Beides wurden bisher die JavaScript Zeitfunktionen wie *setInterval()* genutzt. Diese haben den Nachteil, dass man damit nicht die richtige Bildwiederholungsrate des Monitors nutzt.

---

[12] Jürgen Kohlenberg, "Mathematische Zeichen", code-knacker.de, abgerufen am 22.9.2013 http://www.code-knacker.de/mathe.htm

Reicht `transition` nicht aus, weil beispielsweise die Animation erst berechnet werden muss, stellt HTML5 genau für diesen Zweck die Funktion *requestAnimationFrame()* zur Verfügung. Diese passt sich der Bildwiederholungsrate des Monitors an.

Für die Darstellung werden nun Farbverläufe (genannt „Gradient“) oder Schatten unterstützt. Bei jedem Farbwert ist Angabe einer Transparenz, also einem Alpha Kanal, möglich. Der HSL Farbraum wird ebenfalls unterstützt. Neu hinzugekommen sind das Umrahmen eines Textes (*text-stroke-width* und *text-stroke-color*) und das Umrahmen eines Elements mit Hilfe eines Bildes (*border-image*). Eigenschaften können für jede Ecke und jede Kante unabhängig voneinander vergeben werden. Es existieren neue Selektoren und eigene Schriftarten können über `@font-face` in einer Seite eingebunden werden. Auch eine automatische Silbentrennung ist möglich. Die Liste lässt sich so weiter führen.

Innerhalb des Standards wurde die JavaScript Schnittstelle erweitert, die wichtigsten seien hier genannt. Für DOM Manipulationen existiert die Funktion document.querySelectorAll(). Damit lassen sich Elemente genauso selektieren, wie es in CSS üblich ist. Ohne diese Schnittstelle muss der Entwickler diese Funktion selber entwickeln oder auf eine JavaScript Bibliothek ausweichen.

Ein Handelsüblicher Computer hat heutzutage mehr als ein Prozessor Kern. Mit Hilfe von „WebWorker“ ist es im Browser möglich diese für umfangreiche Berechnungen mit einzubeziehen. Das Laden einer Seite, die Darstellung und das Ausführen von clientseitigen Code läuft in einen Hauptprozess statt. In der Informatik nennt man so etwas einen *Thread*. Mit Hilfe der JavaScript WebWorker-Klasse können weitere Nebenthread erzeugt werden. Diese laufen parallel zur selben Zeit mit dem Hauptthread ab. Sind diese abgearbeitet wird ein Ereignis im Hauptthread ausgelöst. So werden die nötigen Daten zwischen den Threads ausgetauscht.

Eine effektivere, wenn auch aufwendigere Methode Berechnungen zu beschleunigen, ist es diese auf die Grafikkarte auszulagern. Dieses Verfahren wird als GRGPU bezeichnen. Es steht für „universelle Berechnungen auf den Grafikprozessor“. Die Einschränkungen sind höher, jedoch ist der Leistungsgewinn im Vergleich zu interpretierten JavaScript enorm. Dies liegt an den Umstand, dass eine Grafikkarte ungleich mehr Kerne besitzt als ein Computer-Prozessor. Alle Kerne laufen parallel.

Um im Browser die Grafikkarte direkt anzusteuern wird WebGL genutzt. Dazu ist ein <canvas>-Element nötig das in JavaScript selektiert wird und auf dem der Befehl `canvas.getContext("webgl")` ausgeführt wird. Hauptsächlich wird mit WebGL 3D Grafik erzeugt. In einer Desktop-Umgebung haben sich die Schnittstellen OpenGL und DirectX durchgesetzt um Grafikkarten anzusteuern. Während OpenGL kostenlos offen nutzbar und gut dokumentiert ist, wird DirectX nur für Windows Systeme und meist bei grafisch anspruchsvollen Spielen eingesetzt. WebGL nutzt OpenGL ES 2.0, eine kompaktere Form von OpenGL welches für Plattform mit eher wenig Ressourcen gedacht ist. Da es sich bei OpenGL ES 2.0 um eine Low Level API handelt ist der Entwicklungsaufwand entsprechend hoch. Daher existieren zahlreiche JavaScript Bibliotheken wie GLGE, three.js oder X3DOM.

HTML5 bietet eine bessere Unterstützung für Offline Anwendungen. Über „Web Storage“, erreichbar über das *window.localStorage*-Objekt können Daten in den Offline-Speicher dauerhaft geschrieben werden. Im Vergleich zu Cookies laufen diese nicht ab, bieten pro Domain deutlich mehr Speicherplatz und können nur vom Client selber gesetzt werden.

Das Anlegen einer Datenbank oder das zusätzliche setzen von Indices an einer Spalten ist prinzipiell möglich, jedoch wurde die Arbeit an den dafür erstellten Standard „Web SQL Database“ eingestellt [13]. Die Browser-Hersteller haben jedoch diese Technik aus dem Browser nicht entfernt.

Eine von den Browser bisher kaum umgesetzte Technik ist die FileSystem-API. Damit kann der Browser innerhalb der Sandbox Dateien schreiben, verschieben, löschen oder Ordner anlegen wie bei einer richtigen Anwendung. Die File-API ist dagegen schon weiter verbreitet. Sie vereinheitlicht die Interaktion mit Dateien, egal woher sie stammt. Dafür wurde ein Datentyp eingeführt der die Bezeichnung „*Blob*“ trägt. Mit Hilfe dieses Typs kann beispielsweise der MIME-Typ oder die Dateigröße bestimmt werden.

Mit Hilfe der File-API und dem Blob-Datentyp kann ein Vorschaubild angezeigt werden, nachdem der Nutzer per „Drag and Drop“ eine Datei in den Browser gezogen hat. Bevor Daten den Client verlassen können diese verschlüsselt werden. Es ist auch möglich eine Datei asynchron zu lesen. Das bedeutet, dass während eine Datei auf dem Server hoch geladen wird, der Nutzer noch eingreifen kann. Während dem Upload blockiert der Browser nicht und der Nutzer muss nicht warten.

Bei Uploads die über AJAX realisiert sind wird das XMLHttpRequest-Objekt genutzt. Dieses wurde überarbeitet, was auch als *XMLHttpRequest2* bezeichnet wird. Der Objekt Name ist in JavaScript gleich geblieben. Wie aus den Namen ersichtlich wird, war das Objekt ursprünglich nur für die Übertragung im XML Format ausgelegt. Diese Einschränkung gibt es nicht mehr. Ohne einen Workaround können binär Daten zwischen Client und Server ausgetauscht werden. XMLHttpRequest2 ermöglicht eine Ursprungsübergreifende Anforderung. Während der Übertragung sind Informationen über den Fortschritt abrufbar.

Bei der Portierung von Spielen, besonders bei Ego-Shootern auf WebGL Basis, kommt das klassische Event-Handling der Maus an ihre Grenzen. Stößt die Maus an den Bildschirmrand können weitere Bewegungen nicht richtig erfasst werden. Die Webseite erkennt nur die Bildschirmkoordinaten des Mauszeigers über der Webseite. Es ist jedoch nötig alle Werte der Maus direkt zu lesen. Das geschieht über die „Pointer Lock-API“. Über die Variable *document.pointerLockElement* kann geprüft werden ob der Browser diese unterstützt. Das Sperren der Maus geschieht über die Funktion *requestPointerLock()*. Der Nutzer wird dann vom Browser gefragt ob er damit einverstanden ist die Maus zu sperren. Danach werden die Werte der Maus per JavaScript direkt abgefragt.

So ähnlich funktioniert die Geolokalisierung. Internetfähige Smartphones besitzen meist ein GPS-Empfänger mit denen Längen und Breitegrad genau bestimmt werden. Diese liefern einen viel genaueren Wert, als über die Lokalisierung über die IP-Adresse. Auch ohne GPS ist eine Lokalisierung anhand der Mobilfunkzelle möglich. Per HTML5 lassen sich diese Werte über *navigator.geolocation.getCurrentPosition()* auslesen. Wie bei Blockieren der Maus muss auch hier der Nutzer erst einwilligen.

---

[13] Jens Ihlenfeld, "Web SQL vor dem Aus", Computec Media AG, abgerufen am 22.9.2013 http://www.golem.de/1011/79548.html

Der neue Standard ermöglicht nicht nur auf GPS Informationen, Bewegungssensor oder auf die Maus zu zugreifen. Über *navigator.getUserMedia()* lassen sich auch Kamera oder Mikrofon in eine Web-Anwendung einbinden und auslesen. Das spielt für Live-Streams eine wichtige Rolle. Dabei brauchen die Daten nicht über den Server zu laufen. Das W3C arbeitet an ein Protokoll das eine direkte Kommunikation zwischen zwei Browsern ermöglicht[14].

HTML5 ist in der Lage Audio und Video zu streamen. Wie bereits beschrieben kann das Video-Signal über das <canvas>-Element bearbeitet werden. Was bei einer interaktive Webseite noch fehlt, ist das Erzeugen und Bearbeiten eines Audiosignals, wie es bei einer normalen Anwendung möglich ist.
Dafür ist die WebAudio-API zuständig. Kernpunkt ist eine Instanz der `AudioContext`-Klasse die im window-Objekt zu finden ist. Zum Funktionsumfang gehören das dynamische Abspielen von erstellten oder nachgeladenen Sounds, der Einsatz von Filtern wie Tiefpass, Hochpass, Hall oder die Ausgabe auf unterschiedlichen Kanälen. Das Abmischen oder der Übergang (Cross-fading) zwischen zwei Quellen wird von der API unterstützt.

Wie bereits erwähnt handelt sich bei dem im Web verwendeten HTTP Protokoll um ein zustandsloses Protokoll. Eine Kommunikation zwischen Client und Server, nachdem die Seite angefordert wurde, geschieht in der Regel über Ajax. Jedoch bleibt auch dort das Konzept gleich, das der Client etwas anfordert und daraufhin der Server antwortet. Der Server kann von sich aus keine Daten an den Client senden. Abhilfe schafft die in HTML5 eingeführte JavaScript WebSocket-Klasse. Diese baut eine persistente Verbindung zwischen Client und Server auf, wie es in der Netzwerkprogrammierung üblich ist. Ein Socket beschreibt ein Kommunikationsendpunkt bei dem sowohl Daten empfangt als auch gesendet werden.

Betrachtet man die Möglichkeiten von HTML5 als Ganzes wird deutlich, dass in naher Zukunft im Grunde jede Anwendung in das Web portierbar ist, ohne zusätzliche Plug-Ins und auf jeder Plattform. Ein Haken ist die Geschwindigkeit von JavaScript. Da diese Sprache interpretiert wird, schwach typisiert ist und in einer Sandbox läuft ist diese deutlich langsamer als Sprachen die in einen nativer Code kompiliert wurden. Abhilfe soll das Modul „asm.js“ schaffen, dass im JavaScript Code einen Satz von vorgefertigten Vereinbarungen trifft. So wird zum Beispiel der Typ einer Variablen strikt angegeben. Ist die JavaScript Engine im Browser darauf vorbereitet kann er besser Optimieren. Im Idealfall erhält man auf der Client Seite wieder einen nativen Code, der etwa halb so schnell läuft wie ein Programm das in C/C++ geschrieben wurde.

---

[14] Adam Bergkvist, Daniel C. Burnett, Cullen Jennings und Anant Narayanan, "WebRTC 1.0: Real-time Communication Between Browsers", World Wide Web Consortium, abgerufen am 22.9.2013 http://dev.w3.org/2011/webrtc/editor/webrtc.html

3. Konzeption und Umsetzung

HTML5 ist die Grundlage der zu erstellenden JavaScript-Bibliothek. Im folgenden Kapitel werden die enthaltenen Diagrammtypen näher erläutert.

3.1 Diagrammtypen

3.1.1 Liniendiagramm

Dieser Diagrammtyp wird wohl am meisten genutzt. Gegeben ist eine *Funktion*. Diese stellt einen Zusammenhang zwischen den Variablen x und y her. Bei einen Liniendiagramm wird in der Regel eine explizite Funktion genutzt. Es handelt sich um eine Funktionsgleichung der Form ***y = f(x)***. Das bedeutet x ist eine unabhängige Variable die als Funktionsargument in die Funktionsgleichung eingesetzt wird. Zu jeden x-Wert ist immer genau einen y-Wert zugeordnet. Die Variable y ist somit abhängig. Es können keine zwei y-Werte demselben x-Wert zugeordnet sein.

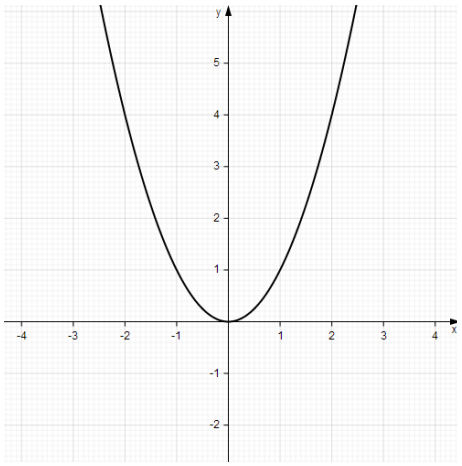


Abb. 5: Liniendiagramm

Das Liniendiagramm stellt diesen Zusammenhang dar. Es besteht aus zwei Achsen für die verschiedenen x beziehungsweise y-Werte. Zusammen bilden sie ein kartesisches Koordinatensystem.
In die Funktionsgleichung werden sichtbare x-Werte eingesetzt, das Ergebnis ist der zugeordnete y-Wert. Sie bilden ein Werte-Paar. Im Koordinatensystem werde sie durch einen Punkt repräsentiert. Alle so erzeugten Punkte werden zu einer Linie verbunden.

Ist die Funktionsgleichung weder nach x, noch y aufgelöst erhält man eine implizite Funktion. Sie hat die Form ***F(x, y) = 0***.
Bei dieser Darstellung können mehrere y-Werte demselben x-Wert zugeordnet sein, nämlich die Werte y die sich bei vorgegebenem x als Lösung der Gleichung ergeben. Die grafische Darstellung wird dann als *implizites Liniendiagramm* bezeichnet.

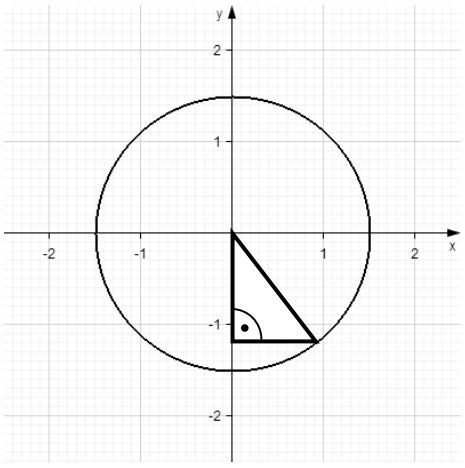


Abb. 6: Darstellung einer impliziten Funktion

Als Beispiel wird die Darstellung eines Kreises mit dem Radius 1.5 betrachtet:

$$(1.5)^2 = x^2 + y^2$$

Diese Gleichung ergibt sich nach dem Satz des Pythagoras im Dreieck, welches in Abbildung 6 eingezeichnet ist.
Durch Umformung erhält man die implizite Darstellung:

$$0 = x^2 + y^2 - 2.25$$

In diesem Beispiel wird die y-Achse zweimal geschnitten. Es sind also zwei verschiedene y-Werte den denselben x-Wert zugeordnet.

3.1.2 Streudiagramm

Im Streudiagramm werden die Wertpaare aus x und y als Punkte deutlich sichtbar eingetragen. Diese werden nicht zu einer zusammengeschlossenen Linie verbunden.

Hier steht der durch die Punkte repräsentierte Informationsgehalt im Vordergrund. Eine Funktionsgleichung, die einen unmittelbaren Zusammenhang zwischen x und y herstellt, existiert nicht.

Abb. 7: Streudiagramm

Streudiagramme dienen zur besseren Visualisierung von statistischen Daten, um beispielsweise Muster zu erkennen.

Wenn es sinnvoll ist, wird eine Ausgleichsgerade in das Diagramm eingezeichnet (lineare Regression). Auf diese Weise ist ein Trend besser erkennbar.

Die Ausgleichsgerade beschreibt eine sichtbare lineare Abhängigkeit zwischen x und y in der Form **y = f(x) = mx + n**. Der Anstieg **m** bildet in der Statistik eine wichtige Rolle und soll daher mit ausgegeben werden. Man spricht hier auch von einer *Korrelation*. Die Wertpaare „Körpergröße“ und „Körpergewicht“ ist so eine Korrelation.

Alternativ kann in die Punktwolke zur Beschreibung auch eine Kurve gelegt werden (nichtlineare Regression). Die Kurve wird durch ein kubisches Polynom beschrieben.
Zur Verdeutlichung soll ein Beispiel aus der Wirtschaft dienen. Es soll die Frage nachgegangen werden, wie häufig ein Produkt zu welchem Preis deutschlandweit verkauft wird. Der x-Wert gibt den Verkaufswert an und der y-Wert die Häufigkeit der verkauften Ware. An der eingezeichneten Ausgleichskurve lässt sich erkennen, welcher Verkaufspreis am wirtschaftlichsten ist.

Die Ausgangsgleichung bzw. -kurve kann zur Vorhersage genutzt werden. Für beliebig zu wählendes x bekommt man über die Gerade/Kurve den Vorhersagewert für y.

Bei bestimmten Mustern kann es hilfreich sein das Diagramm in Intervalle zu unterteilen. Diese Intervalle müssen zunächst richtig erkannt werden.

Streudiagramm haben auch Nachteile. So lässt sich schwerer erkennen wenn ein Wertpaar mehrfach existiert und das Analysieren der Daten ist zeitaufwändiger.

3.1.3 Kreisdiagramm

Das Kreisdiagramm (auch Tortendiagramm) dient zur grafischen Darstellung von Anteilen, die den einzelnen Merkmalsausprägungen eines kategorischen Merkmals in Bezug auf ein stetiges Zielmerkmal zukommen.

Im Diagramm wird ein Kreis in einzelne Sektoren aufgeteilt. Die Größe des betroffenen Sektors ist proportional zu dem Anteil der zugeordneten Merkmalsausprägung. Es ist ein Teilwert des Ganzen. Der Kreis entspricht der Summe aller Teilwerte. Die Größe eines einzelnen Sektors wird in Bogenlänge, Flächeninhalt oder Volumen angegeben.

Der Winkel für die jeweilige Sektorengröße ergibt sich aus der Gleichung:

$$Winkel = \frac{360° \times Teilwert}{Gesamtwert}$$

Als Beispiel soll eine Umfrage dienen. Gefragt werden 100 Personen nach ihrem genutzten Betriebssystem. Dann ergeben sich folgende statistische Variablen:

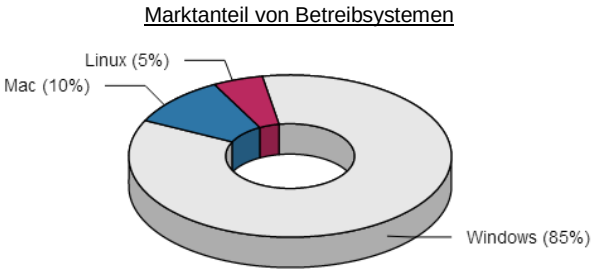


Abb. 8: Kreisdiagramm.

Grundgesamtheit: Nutzer
Kategorisches Merkmal: Betriebssystem
Merkmalsausprägungen: [Windows, Mac, Linux, …]
Stetiges Zielmerkmal: Anzahl Personen die das betreffende Betriebssystem nutzen.

Das Ergebnis sei: 85 Personen nutzen Windows, 10 Personen Mac und 5 Personen Linux. Der Gesamtwert ist somit **100**.

Somit ergeben sich für die Darstellung eines Kreisdiagramms folgende relevante Informationen:

Merkmalsausprägung	Stetiges Zielmerkmal	Winkel
Windows	85	306°
Mac	10	36°
Linux	5	18°

Die Darstellung kann variieren. So muss es sich nicht um einen Vollkreis handeln. Einzelne Sektoren können eine unterschiedliche Position oder einen unterschiedlichen Radius besitzen.

Eine Unterform des Kreisdiagramms ist das *Ringdiagramm*. Es besitzt mehrere Innenkreise mit unterschiedlicher Dicke.

3.1.4 Säulendiagramm / Balkendiagramm

Das Säulendiagramm dient zur grafischen Darstellung von Werten eines stetigen Zielmerkmals für die einzelnen Kategorien eines kategorischen Merkmals.

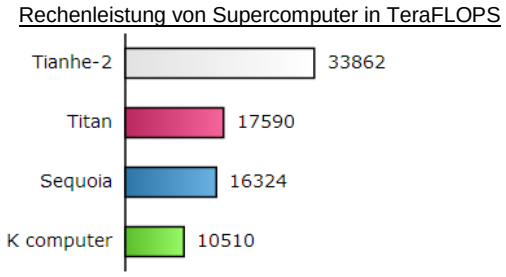


Abb. 10: Balkendiagramm.

Nebeneinander angeordnete Rechtecke repräsentieren die Werte des stetigen Zielmerkmals. Werden die Balken senkrecht dargestellt, dann spricht man von einen *Säulendiagramm*, bei waagerechten Balken von einen *Balkendiagramm*.

Ein Säulen- beziehungsweise Balkendiagramm ist zu bevorzugen, wenn wenige Merkmalsausprägungen vorliegen.

Beim Säulendiagramm existieren verschiedene Varianten. Wenn die stetigen Zielmerkmale negativ sind, können sich die Säulen bzw. Balken auch unterhalb der Achse befinden. Kommt es mehr auf einen Kurvenverlauf an, dann ist ein Liniendiagramm zu bevorzugen. Mehrere Rechtecke können übereinander stehen, dann spricht man von einem *gestapelten Säulendiagramm*. Es spricht prinzipiell nichts dagegen entsprechende Rechtecke in einer Gruppe zusammenzufassen (*gruppiertes Säulendiagramm*).

Einige Bemerkungen zur Umsetzung der Bibliothek.
In dieser Arbeit geht es um die Darstellung von einem normalen Balkendiagramm. Eine vertikale Linie trennt die links befindenden Beschriftungen (Merkmale) von den Werten des Zielmerkmals. Der Platz für die Beschriftung wird automatisch angepasst. Das heißt bei kürzeren Beschriftungen befindet sich diese Linie weiter links. Die restliche horizontale Breite dient für die Rechtecke für die einzelnen Werte. Standardmäßig solle diese automatisch normiert werden. Das heißt alle Werte zusammen summiert ergibt die vollständige Breite. Wenn eine automatische Skalierung unerwünscht ist, soll der Nutzer die Möglichkeit haben diese mit dem Mausrad einzustellen.

Auch vertikal wird eine automatische Anpassung vorgenommen. Je mehr Merkmale desto schmaler die Balken, bis zu einen minimal Wert. Ist dieser erreicht und werden mehr Merkmale dargestellt als auf die Zeichenfläche passt, soll es die Möglichkeit geben das Diagramm mit gedrückter Maustaste zu verschieben.

Ein Gradient für einen linearen Farbverlauf rundet die Darstellung ab.

3.1.5 Histogramm

Ein Histogramm stellt eine Häufigkeitsverteilung von statistischen Daten dar.

Für die Darstellung werden Rechtecke genutzt, die direkt nebeneinander liegen. Jedem Rechteck ist ein bestimmter Wertebereich zugeordnet. Bestimmt wird dieser durch die Breite und die Position der Rechtecke.
Durch die im Histogramm dargestellten Rechtecke wird ein allgemeiner Kurvenverlauf deutlich.

IQ-Punkte:

88	99	103
91	100	104
93	100	105
95	101	106
97	101	106
97	102	106
98	102	117
99	102	

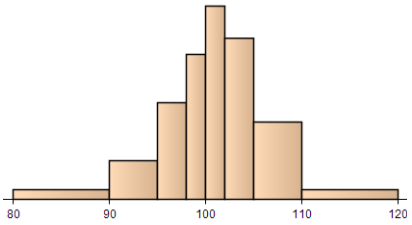


Abb. 10: Gemessene IQ-Punkte, dargestellt durch ein Histogramm.

Anders als beim Säulendiagramm sind bei den Rechtecken sowohl Breite als auch Höhe von Bedeutung. Die Kernaussage ist der *Flächeninhalt*, welcher entweder die relative oder die absolute Häufigkeit darstellt. Der Flächeninhalt ist proportional zur Häufigkeit.

Die Einteilung des Wertebereichs der zu untersuchenden Daten nennt man *Klassen*. Die Breite des Intervalls ist die *Klassenbreite*. Die *Klassenhäufigkeit* gibt an, wie viele Datenwerte im Intervall der Klasse liegen.

Alle Klassen haben entweder eine konstante oder eine variable Klassenbreite. Werden die Intervalle zu schmal gewählt, dann schwankt die Höhe der Rechtecke zu stark, was nicht wünschenswert ist. Bei sehr großen Intervallen liefert das Diagramm keine detaillierte Aussage. Ziel ist es ein vernünftigen Mittelwert zu finden.

Bei gleich großen Intervallen ist auch die Höhe der Balken proportional zum Flächeninhalt.

Sind die Klassen festgelegt, muss für jede einzelne Klasse die Höhe der Rechtecke bestimmt werden, welche die Häufigkeitsdichte repräsentiert. Dazu wird die Klassenhäufigkeit bestimmt und durch die Klassenbreite dividiert:

$$\textit{Häufigkeitsdichte} = \frac{\textit{relative Klassenhäufigkeit}}{\textit{Klassenbreite}}$$

Die Klassenhäufigkeit ist proportional zum Flächeninhalt und die Häufigkeitsdichte proportional zur Höhe. Die Fläche eines Rechtecks ist durch das Produkt von Höhe und Breite definiert, sodass der Zusammenhang durch Umformung deutlicher wird:

$$\textit{Fläche} = \textit{Höhe} \times \textit{Breite} \quad \triangleq \quad \textit{rel. Klassenhäufigkeit} = \textit{Häufigkeitsdichte} \times \textit{Klassenbreite}$$

Ein Histogramm ist normiert, wenn die Summe aller Flächen eins ist. In diesen Fall wird die relative Häufigkeitsdichte der Klassen dargestellt. Fehlt diese Normierung zeigen die Klassen eine absolute Häufigkeit an. Hier soll es nur um die Darstellung der absoluten Häufigkeitsdichten gehen.

3.1.6 Oberflächendiagramm

Beim Oberflächendiagramm wird die Abhängigkeit zwischen zwei unabhängigen Variablen x und y und einer abhängigen Variable z durch eine Funktion dargestellt:

z = f(x, y)

Der x-Wert sowie der y-Wert dient als Funktionsargument, das Ergebnis z ist der Funktionswert. Der Wertebereich der Funktionsargumente liegt in der x-y-Ebene.

Zu jeden Paar aus x- und y-Wert wird über die Funktion ein z-Wert zugeordnet. Die einzelnen Punkte (x; y; z) ergeben die Fläche.

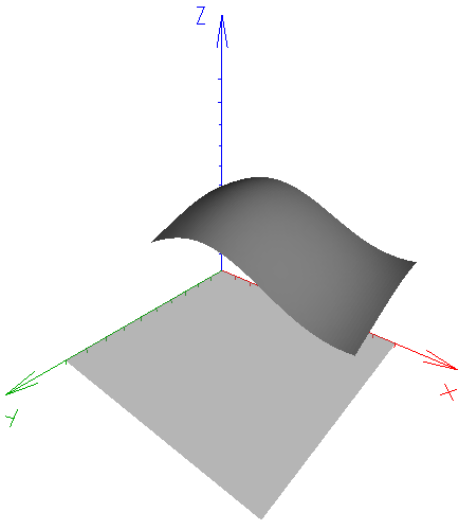


Abb. 11: Oberflächendiagramm

Eine Wölbung wird nur bei einer richtigen Belichtung sichtbar. In jedem Bildpunkt auf der Oberfläche wird ein Winkel α bestimmt. α ist der Winkel zwischen der Neigungstangente der Fläche in Betrachtung und der Verbindungsstrecke vom Auge zum Oberflächenpunkt. Aus dem Winkel α wird der Helligkeitswert bestimmt. Würde jeder Pixel den gleichen Farbwert bekommen, so ist keine Tiefe erkennbar und die Fläche erscheint zweidimensional.

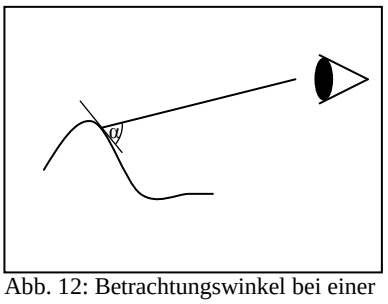


Abb. 12: Betrachtungswinkel bei einer Oberfläche

Oft wird für die Darstellung eine axonometrische Projektion genutzt, man spricht auch von einer orthogonalen Darstellung. Jede der 3 Achsen wird in die Zeichenfläche gezeichnet wie bei einer zweidimensionalen Darstellung, nur die Achsenrichtungen sind verschieden. Hier gibt es durch die verwendete Parallelprojektion kein Nah und Fern. Der Nutzer kann bei zwei Objekten nicht erkennen, welches das nähere ist.

Bei der Umsetzung soll daher auf die orthogonale Darstellung verzichtet werden. Eine korrekte Fotoperspektive soll einen Tiefeneindruck vermitteln. Das Mausrad dient dazu den Abstand zur Oberfläche zu verändern. Bei gedrückter Maustaste kann der Nutzer die Kameraposition und so die Ansicht zur Oberfläche ändern.

Diese Funktion soll auch bei den anderen Diagrammen zum Einsatz kommen. Das Mausrad vergrößert oder verkleinert die Ansicht. Mit Hilfe der gedrückten Maustaste kann die Position verschoben werden.

Das Oberflächendiagramm ist nicht mit dem einfachen Flächendiagramm zu verwechseln, welches ein Liniendiagramm mit mehreren Linien beschreibt, bei dem die einzelnen Flächen ausgefüllt sind.

3.2 Architektur

In diesem Kapitel geht es um den inneren Aufbau der zu erstellenden Bibliothek. Einzelne Komponenten lösen jeweils ein Teilproblem und stehen in Beziehung zueinander.

Die Aufgaben lassen sich in folgende Schritte unterteilen:

- DOM auswerten und überwachen
- Objekte für die Diagramme erzeugen
- Grafik generieren

Zunächst wird die Grafikausgabe beschrieben.

HTML5 bietet zum Zeichnen von Grafiken zwei Techniken an: *Vektorgrafik* und *Pixelgrafik*. Die entsprechenden Tags lauten <*svg*> und <*canvas*>. In dieser Arbeit werden beide Techniken vereinigt. Unabhängig davon welcher Grafiktyp zum Zeichnen genutzt wird, die Ausgabe soll identisch sein. Dazu wird eine Schnittstelle erstellt, hinter der sich zwei unterschiedliche Programmier-Techniken befinden:

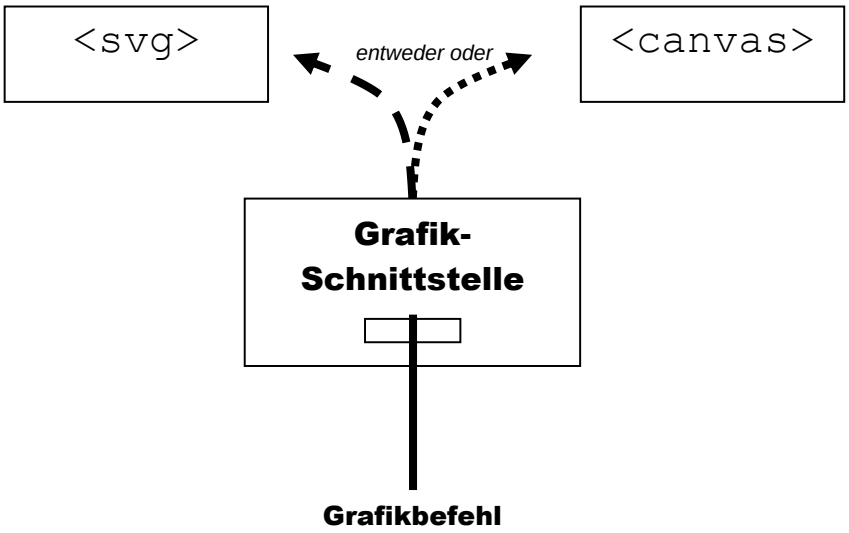


Abb. 13: Grafikschnittstelle

Mit dieser Herangehensweise kann der Nutzer der Bibliothek selbst entscheiden, ob er lieber eine skalierbare oder eine ressourcenschonende Grafik haben möchte.

Das Setzen von zusätzlichen HTML-Attributen im <svg> oder im <canvas>-Tag soll ausreichen, um ein eigenes Diagramm zu erstellen. Um beispielsweise ein Balkendiagramm mit 3 Balken zu erzeugen, soll es ausreichen das Attribut „barchart“ zu setzen. Je nach Diagrammtyp wird ein anderes Schlüsselattribut verwendet. Die erforderlichen Daten stehen im Attributwert. Für weitere Optionen dienen weitere Attribute:

```
<svg barchart="60 25 15" width="200" height="150"></svg>
```

Jegliche Beschreibung für ein Diagramm findet in diesen Tag-Abschnitt statt. Das unterscheidet sich von bereits bestehenden Lösungen, bei denen die Zeichenfläche, das Eingabeformulare und die Programmierlogik fest miteinander verzahnt sind. Bei dieser Bibliothek wurde der Sprachumfang von HTML erweitert. Man benötigt für die Darstellung kein JavaScript und auch keine Änderungen an dem bestehenden Quelltext, weil dieser verborgen ist. Der Nutzer der Bibliothek arbeitet sozusagen ausschließlich mit HTML. Eigene Diagramme können so leicht, flexibel und unabhängig voneinander erzeugt werden. Wie bereits erwähnt ändert sich der Grafiktyp, wenn an Stelle des <svg>-Tags ein <canvas>-Tag genutzt wird. Die Darstellung bleibt jedoch identisch.

Das Setzten von beliebigen Attributen ist im Übrigen nicht ganz HTML5 konform. Es kann zu Inkompatibilität kommen, wenn ein Bezeichner für eine andere Funktionen im Standard reserviert ist. Um dies zu umgehen existiert das Präfix „data-“. Dieses sollte man bei jedem eigenem Attributnamen voran stellen[15]. Damit der Nutzer beim Setzen der Attribute es möglichst einfach hat, wurde auf diese Konvention verzichtet.

Beim Laden der Seite muss für eine korrekte Darstellung zunächst der DOM-Baum nach den passenden Attributen durchsucht werden. Diese Herangehensweise eignet sich nur für Diagramme mit statischem Inhalt. Bei Seiten wo sich der HTML-Inhalt dynamisch ändert, reicht dieses Vorgehen nicht aus.
Damit ein Diagramm neu gezeichnet wird, reicht es aus das entsprechende HTML-Attribut neu zusetzen. Dazu ist es nötig das entsprechende HTML-Element auf Attributänderungen zu überwachen.

Zu jedem gültigem HTML-Attribut wird ein eigenständiges *Objekt* erstellt. Dazu werden bei der objektorientierten Programmierung *Klassen* eingesetzt. Das ist eine Art Bauplan für Objekte die man beliebig häufig aus der Klassendefinition erzeugen kann. Wird in einem Tag ein gültiges HTML-Attribut erkannt, erzeugt die Bibliothek daraus ein Objekt.

Aus dem oben genannten Beispiel wird so aus dem <svg>- Tag mit den Attribut „`barchart`“ ein richtiges Objekt mit eigenen Eigenschaften und Funktionen erzeugt. Die inneren Eigenschaften des Objekts werden aus den HTML-Attributen bestimmt. Also 3 Säulen mit den Werten 60, 25 und 15. Die Funktionen nennt man in der objektorientierten Programmierung „*Methoden*“. Sie dienen dazu den Zustand eines Objekts zu ändern. Die Interaktion mit einem Objekt findet über solche Methoden statt.

Bei Objekten mit einem ähnlichem Funktionsumfang ist es ungünstig denselben Quelltext erneut zu implementieren. Dazu wird eine Programmier-Technik eingesetzt die sich *Vererbung* nennt. Eine *Oberklasse* (auch *Basisklasse*) dient als Vorlage für verschiedene Unterklassen. Diese *erben* alle relevanten Eigenschaften und Methoden der Oberklasse.

---

[15] Denis Potschien, "HTML5: Eigene Attribute für Elemente erstellen", yeebase media GmbH, abgerufen am 22.9.2013 http://t3n.de/news/html5-eigene-attribute-elemente-erstellen-338743/

Jedes Objekt das ein Diagramm repräsentiert liest die entsprechenden HTML-Attribute aus und reagiert auf deren Änderungen. Für das Speichern und das Überwachen der Attribute empfiehlt es sich eine Oberklasse zu schreiben:

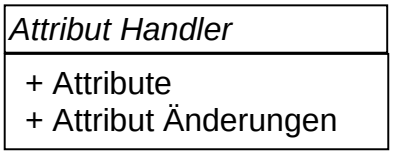


Abb. 14: Klassedefinition für HTML-Attribute

Jedes Objekt das ein Diagramm repräsentiert erbt von dieser Klasse, um das Objekt mit entsprechenden Funktionalitäten auszustatten:

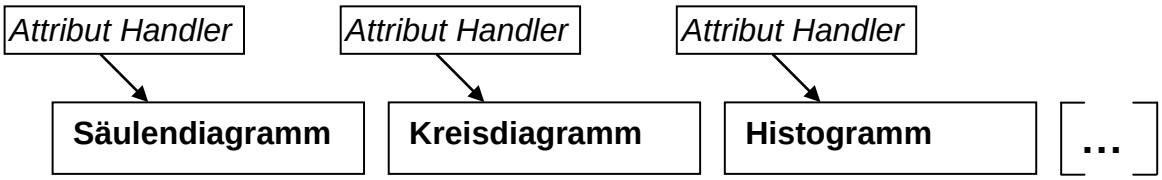


Abb. 15: Vererbungshierarchie für Diagramme ohne einem Koordinatensystem

Eine weitere wichtige Aufgabe ist das Zeichnen des Koordinatensystems. Je nach Diagramm-Typ ist eins erforderlich. Der Betrachter der Seite soll die Möglichkeit bekommen mittels gedrückter Maustaste diese zu verschieben oder mit dem Mausrad die Ansicht zu vergrößern.

Vom Koordinatensystem soll nur das gezeichnet werden was auch tatsächlich im Darstellungsbereich liegt. Das dargestellte Koordinatensystem stimmt nicht mit den Koordinaten der Bildschirmpunkte überein. Je nach Ansichtspunkt, Vergrößerungsfaktor sowie Höhe und Breite des Darstellungsbereichs ist die Beziehung immer unterschiedlich**.** Eine Hilfsklasse soll das Umrechnen zwischen beiden Ansichten vereinfachen.

Diese Hilfsklasse ist Bestandteil der Oberklasse für ein *interaktives Diagramm* mit einem Koordinatensystem. Die Implementierung für die Maus, der Umgang mit dem Koordinatensystem und die Grafikausgabe finden in dieser Oberklasse statt.

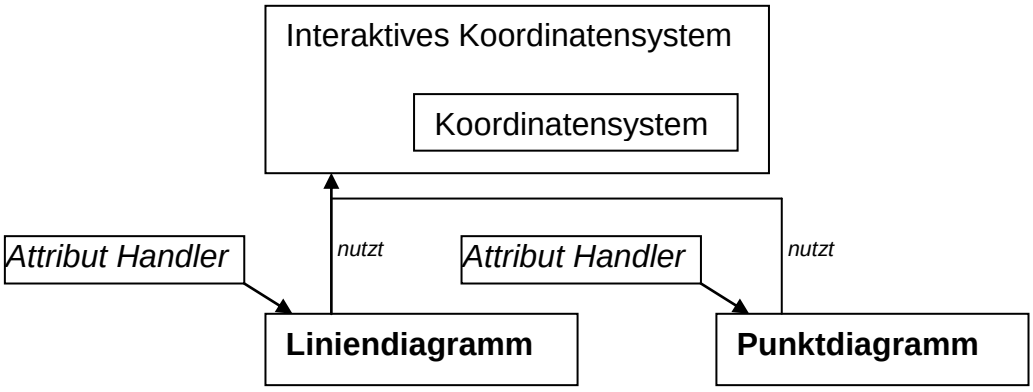


Abb. 16: Vererbungshierarchie für Diagramme mit einem Koordinatensystem

Bei Betrachtung der bisher beschriebenen Komponente (Abbildung 16) sei darauf hingewiesen, dass die Grafikschnittstelle der Bibliothek das ursprünglich ausgelöste HTML-Element besitzen muss, um die Grafik zeichnen zu können. Das Element wird also durchgereicht.

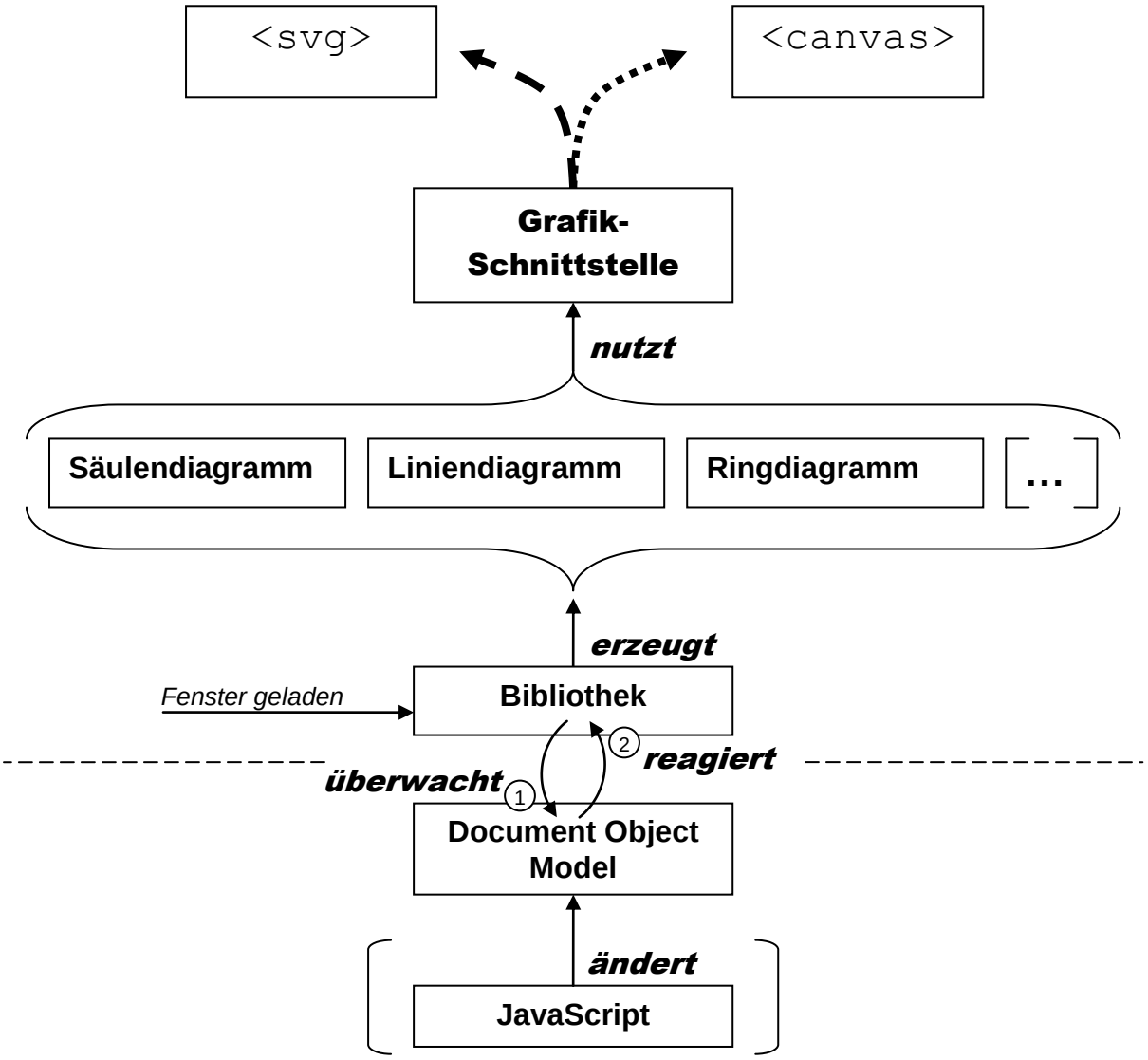


Abb. 17: Architektur

Die Bibliothek mit seinen Klassen ist nach außen hin nicht sichtbar. Das Zeichnen der Diagramme richtet sich allein an den gesetzten HTML-Attributen.

Die wichtigsten Klassen für die Diagrammtypen sind damit untergebracht. Für den vollständigen Funktionsumfang sind noch weitere nötig. Diese werden im nächsten Kapitel näher beschrieben.

Ein Punkt soll noch hervorgehoben werden: Die Eingabe des Nutzers. Diese wird üblicherweise bei HTML über ein <input>-Feld entgegen genommen. Nicht immer reicht dieser Funktionsumfang aus, auch nicht mit HTML5. Genau wie bei <svg> und <canvas> soll hierfür dieselbe Technik eingesetzt werden, um den Funktionsumfang der <input>-Felder zu erweitern. Ist beispielsweise ein mathematischer Ausdruck mit zwei unbekannten *a* und *b* verlangt, soll es ausreichen ein zusätzliches HTML-Attribut zu setzen:
`<input term="a b">`

3.3 Implementierung

Als nächstes geht es um die Umsetzung. Die Diagramme werden auf dem Client erzeugt und der Nutzer kann mit ihnen interagieren. Darum kommt hauptsächlich JavaScript zum Einsatz. Das Skript greift auf den DOM-Baum zu, ändert dynamisch die Darstellung, nutzt HTML5 und setzt gegebenenfalls CSS Werte. Die folgende Übersicht soll ein Überblick über die wichtigsten Syntax-Regeln geben.

Anweisungen werden durch Semikolon getrennt:

```
anweisung1; anweisung2; anweisung3;
anweisung4;
```

Geschweifte Klammern bilden einen Anweisungsblock. Diese können verschachtelt sein:

```
{
      anweisung1;
      { anweisung2; anweisung3; }
}
```

Verzweigungen werden über die `if` Anweisung realisiert. Ist die in runden Klammern angegebene Bedingung erfüllt, so wird der erste Anweisungsblock ausgeführt. Ist sie nicht erfüllt wird der `else`-Zweig ausgeführt:

```
if(Bedingung)  { /* wahr */ }
else { /* falsch */ }
```

Bedingungen werden über boolesche Ausdrücke angegeben:

```
a == b; // Gleichheit
a || b; // Oder
a && b; // Und
```

Die `while`-Schleife dient dazu ein Anweisungsblock mehrmals zu durchlaufen, solange wie die angegebene Bedingung wahr ist:

```
while(Bedingung){/*anweisungen*/}
```

Wiederkehrende Aufgaben werden in Funktionen ausgelagert:

```
function quadrat(a){ return a*a; }
```

Der Aufruf erfolgt über den Funktionsnamen:

```
var q = quadrat(4); // q = 16
```

Variablen werden durch das Schlüsselwort `var` deklariert. Danach folgt der Variablenname. Die Zuweisung geschieht über das Gleichheitszeichen. Der Typ (Zeichenkette, Ganzzahl, Objekt, usw.) spielt keine Rolle und kann sich zur Laufzeit ändern:

```
var varible1, variable2 = "abc";
varible1 = 64; variable2 = true;
var variable3 = new Object();
```

Es existieren 5 arithmetische Operationen:

```
var a = 1 + 1; // Addition
var b = 5 - 1; // Subtraktion
var c = 2 * 3; // Multiplikation
var d = 1 / 4; // Division
var e = 15 % 4; // Rest
console.log(a, b, c, d, e);
// Ausgabe 2, 4, 6, 0.25, 3
```

Objekte können über ein Objekt-Literal definiert werden. JavaScript ermöglicht es Eigenschaften auch nachträglich zu ändern:

```
var objekt= {prop1: 8, prop2: "X"}
console.log(objekt.prop2); // 8
objekt.prop3 = new Array(5);
```

Funktionen werden wie ein Datentyp behandelt:

```
var func = function() {}
```

Um eine Aufzählung oder ein Objekt zu durchlaufen, bietet sich die `for`-Schleife an:

```
var a = [1, 1, 2, 3, 5, 8];
for(var key in a){ var i=a[key]; }
```

Der DOM-Zugriff und das setzen von Attributen geschieht über „`document`“

```
var g =
document.getElementById("graph");
g.setAttribute("width", "96px");
```

3.3.1 Klassendefinition

Eine saubere Programmierung setzt eine Abstraktion voraus. Eine schwierige Aufgabe wird in einfache unterteilt und zusammengehörige Daten werden zu einem Objekt zusammengefasst.

Die einzelnen Diagramme bilden so einzelne Objekte. Der Bauplan wird über eine *Klasse* definiert. Anhand einer gegebenen Klasse werden so beliebig viele Objekte erzeugt. Das Erzeugen von Objekten aus einer Klasse heraus nennt man *Instanziierung*.

Gegeben sei eine fertige Klasse für ein Liniendiagramm. Diese sind alle vom selben Typ, können aber unterschiedliche Zustände besitzen. Auf derselben Webseite können sich mehrere Liniendiagramme befinden die unterschiedlich groß sind, einen anderen Wertebereich zeigen oder eine unterschiedliche Linie beinhalten. Alle diese Parameter sind im jeweiligen Objekt gespeichert.

Es gibt unterschiedliche Möglichkeiten Objekte in JavaScript zu erzeugen. Um eine Klasse zu definieren ist eine Funktionsdeklaration die gängigste Methode. Diese Funktion wird jedoch nicht auf den herkömmlichen Weg aufgerufen, abgearbeitet und daraufhin die Ressourcen wieder freigegeben. Soll eine Funktion als Klasse fungieren, wird sie über das Schlüsselwort `new` aufgerufen. Das bewirkt, dass der innere Zustand bei der Abarbeitung der Funktion dauerhaft gespeichert wird. Es wird kein Funktionswert zurück gegeben. Durch das Schlüsselwort `new` wird ein *Objekt* zurückgegeben, welches die inneren Zustände enthält.

```
function LineChart(element) // Klassendefinition
{
      this.GetElement = function() { return element; }
}

var element = document.getElementById("graph");

var chart = new LineChart(element) // Instanziierung

/* Die private Variable "element" ist im Objekt "chart" gespeichert. */
console.log(chart.GetElement().tagName); // Objekt-Zugriff
```

Die Prozedur die nötig ist, um ein Objekt zu erzeugen nennt man *Konstruktor*. Dem Konstruktor können bei der Instanziierung Parameter übergeben werden, in diesen Fall `element`.

In der objektorientierten Programmierung besitzen Objekte *Attribute* und *Methoden*. Die Eigenschaften (wie Höhe und Breite eines Diagramms) wird in Variablen gespeichert und bilden die Attribute eines Objekts. Diese allein reichen nicht aus. Eine Funktionalität ist erst dann gegeben wenn Algorithmen die Attribute nutzen und verändern. Das geschieht über die Methoden, sie werden durch einfache Funktionen realisiert.

Eine Klasse dient nicht nur dazu zusammengehörige Variablen (Attribute) und deren Operationen (Methoden) zu vereinen. Ein wichtiger Aspekt ist die Kapselung. Wäre alles nach außen sichtbar, so ist die Verwendung der instanziierten Klasse fehleranfällig und nicht robust. Um dies zu vermeiden gibt es unterschiedliche Sichtbarkeitsstufen. Bei der öffentlichen Sichtbarkeit handelt es sich meist um Methoden die dazu dienen um auf das Objekt zuzugreifen. Sie dienen als Schnittstelle nach draußen. `GetElement()` im obigen Beispiel ist so eine öffentliche Methode.

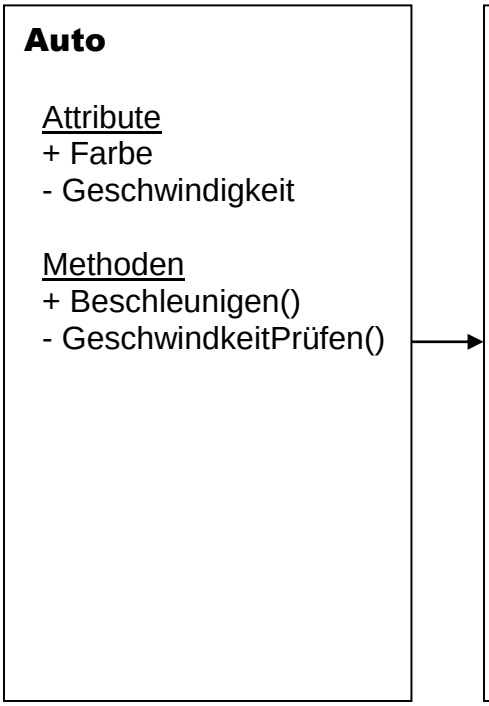
Attribute können ebenfalls öffentlich sein. Ein besseres Herangehensweise ist es jedoch öffentliche Attribute privat zu deklarieren und nur über öffentliche Methoden zu schützen, damit diese keine ungültige Werte annehmen. Die Kombination aus privaten Attributen und öffentlich-schützende Methoden bilden die Eigenschaften eines Objekts. *Felder* ist meist ein Synonym für „*private Attribute*“.

Die private Sichtbarkeit ist das Gegenstück zur öffentlichen Sichtbarkeit. Private Methoden und Attributen können nur von der Klasse selbst genutzt werden. Sie sind von außen nicht sichtbar.

Wird das Konzept der Datenkapselung richtig umgesetzt, kann das Objekt keinen ungültigen inneren Zustand annehmen.

Zur Veranschaulichung der unterschiedlichen Sichtbarkeitsstufen soll ein einfaches Beispiel reichen. Gegeben ist eine Klasse „Auto“. Das Auto hat die Attribute „Farbe“ und „Geschwindigkeit“. Die Farbe soll in diesen Beispiel ein öffentliches Attribut sein (+ Zeichen) und die Geschwindigkeit privat (- Zeichen). Mit Hilfe der öffentlichen Methode „Beschleunigen“ soll der innere Zustand geändert werden. Eine private Methode überprüft den inneren Zustand, sprich die Geschwindigkeit.

Die entsprechende Klasse sieht in JavaScript etwa wie folgt aus:




```
function Auto(farbe)
{
    this.Farbe = farbe;
    var geschwindikeit = 0;

    this.Beschleunigen = function(wert)
    {
        geschwindikeit += wert;
        GeschwindigkeitPruefen();
    }
    function geschwindigkeitPruefen()
    {
        if(geschwindikeit < 0)
        {
            geschwindikeit = 0;
        }
        else if(geschwindikeit > 100)
        {
            geschwindikeit = 100;
        }
    }
}
```

Öffentliche Attribute und Werte werden mit dem Schlüsselwort `this` deklariert. Bei privaten Attributen und Werten ergeben sich keine Änderungen. Der Zugriff auf öffentliche Member innerhalb der Klasse erfolgt ebenfalls über `this`.

Es ist mit `this` jedoch nicht möglich, innerhalb einer privaten Methode auf öffentliche Member zuzugreifen. Der Trick besteht nun darin das Objekt selber, welches durch `this` referenziert wird, in einem Feld zu speichern. Meist wird dafür die Bezeichnung `that` verwendet.

```
function A()
{
      var that = this;
      function privat()        { that.X = 456; }
      this.public = function() { this.X = 789; }
      this.X = 123;
}
```

Der Grund für diese Einschränkung lässt sich logisch herleiten. Gegeben sei eine Klassendeklaration A, die eine private Methode B besitzt. Innerhalb dieser Methode wird eine Hilfsfunktion f deklariert. Käme in dieser Funktion `this` zum Einsatz, dann kann der Interpreter nicht bestimmen ob die Funktion A oder die Funktion B als Klasse eingesetzt wird. Beide haben dieselbe Signatur und der Aufruf beider Funktionen kann sowohl mit, als auch ohne `new` erfolgen.

Bei öffentlichen Methoden ist der richtige Geltungsbereich schnell gefunden. Dieser liegt in dem Geltungsbereich, bei dem die Funktion mit `this` deklariert wurde:

```
function A()
{
    function B()
    {
        function f()
        {
            // Soll this auf
            // A oder B zeigen?
        }
    }
}
```

```
function a()
{
    this.b = function()
    {
        function c()
        {
            var x = this;
        }
    }
}
```

`this` zeigt auf die erzeugte Instanz der Klasse a

3.3.2 Vererbung

Alle Diagramm-Objekte sind an einem <svg> oder <canvas>-Element gebunden. Die Parameter für die Darstellung bezieht das Objekt von zusätzlich gesetzten HTML-Attributen. Werden Attribute geändert muss darauf reagiert werden, weil der Nutzer auf den Inhalt der Grafik Einfluss nehmen kann. Jede Klasse die ein Diagramm repräsentiert besitzt diese Funktionen. Damit nicht in jede Klasse derselbe Algorithmus implementiert wird, soll eine Oberklasse diese Aufgabe übernehmen. Die Klassen erben die Funktionen der Oberklasse.

Eine richtige Vererbung gibt es in JavaScript nicht. Stattdessen wird das Konzept eingesetzt das sich „prototypenbasierte Vererbung“ nennt [16]. Hier dient eine Klasse als Vorlage, der Prototyp. Dieser wird geklont und mit der abgleitenden Klasse verbunden. Dieser Vorgang findet erst zur Laufzeit statt. Wird auf ein Member zugegriffen, wird zuerst in der abgleitenden Klasse gesucht. Findet der Interpreter dort nichts, wird im Prototyp nachgeschaut. In JavaScript können auf dieser Weise nur *öffentliche* Member vererbt werden.

Mit Hilfe der prototypenbasierten Vererbung und einer Oberklasse für die HTML-Attribute werden die verschiedenen Unterklassen mit zwei wichtigen Eigenschaften ausgestattet. Diese sind über öffentliche Attribute realisiert, weil private Attribute nicht vererbt werden können. Das erste Objekt-Attribut ist eine Liste aus Schlüssel-Werten-Paaren für jedes HTML-Attribut. Das zweite ist ebenfalls eine Liste bei dem für jeden HTML-Attribut einen Eintrag existiert. Als Wert beinhaltet dieser Eintrag jedoch eine Funktion die als Ereignis fungiert. Dieses Ereignis wird ausgelöst sobald sich das entsprechende HTML-Attribut ändert. Übergeben wird der neue und der alte Wert.
Eine drittes Objekt-Attribut ist ein Ereignis, dass auf jede Änderung im HTML-Element reagiert. Zunächst sind alle drei Listen leer.

```
this.Attributes = {};        // Werte für alle HTML-Attribute
this.AttributesChange = {}; // Wird ausgelöst, sobald sich ein bestimmtes
                            // HTML-Attribut ändert

this.AttributeOnChange = function() { }; // Wird ausgelöst, sobald sich ein
                                         // beliebiges HTML-Attribut ändert
```

Es wird beispielsweise ein Balkendiagramm initialisiert, dafür wird ein Objekt angelegt. Für die Initialisierung wird der HTML Knoten übergeben und ausgewertet. Sei dieser wie folgt definiert:

```
<svg width="150" height="100" bar="75 20 5" moveable></svg>
```

Die Oberklasse sorgt nun dafür, dass im Objekt folgende öffentliche Attribute gesetzt sind:

```
this.Attributes.WIDTH = "150"
this.Attributes.HEIGHT = "100"
this.Attributes.BAR = "75 20 5"
this.Attributes.MOVEABLE = ""
```

---

[16] "Prototypenbasierte Programmierung", Wikimedia Foundation Inc., abgerufen am 22.9.2013 http://de.wikipedia.org/wiki/Prototypenbasierte_Programmierung

Damit es nicht zu Verwechselung oder Tippfehlern bei der Groß- und Kleinschreibung kommt, werden die Einträge mit Großbuchstaben abgespeichert. Während in HTML das Minuszeichen »-« im Bezeichner erlaubt ist, unterstützt JavaScript nur das Unterstrichzeichen »_«. Diese wurden beim Abspeichern ausgetauscht.

Werden Daten im Balkendiagramm über ein neues HTML-Attribut geändert, reicht es aus das entsprechende Ereignis der Oberklasse abzufangen, wenn darauf reagiert werden soll:

```
this.AttributesChange.BARCHART = function(neuerWert, alterWert)
{
}
```

Das Erfassen von Änderungen bei den HTML-Attributen ist in dieser Arbeit von entscheidender Rolle. Ist dies nicht möglich, kann der Betrachter der Seite auch keine Änderungen am Diagramm vornehmen.

Ursprünglich gab es speziell für diese Aufgabe im Browser das Ereignis „`DOMAttrModified`“ welches man nur abonnieren brauchte. Dieses ist inzwischen als *deprecated*, also veraltet deklariert. Im Firefox existiert dieses Ereignis noch, während im Chrome kein Ereignis ausgelöst wird.

Abhilfe schafft das `MutationObserver` Objekt:

```
var observer = new MutationObserver(function(mutations)
{
      that.AttributeOnChange();
      mutations.forEach(function(mutation)
      {
            var key = mutation.attributeName.toUpperCase(),
                  previousValue = that.Attributes[key],
                  newValue = mutation.target.getAttribute
                  (
                        mutation.attributeName
                  );
            if(typeof previousValue == "undefined") { previousValue = ""; }
            that.Attributes[key] = newValue;
            if(newValue === null)
            {
                  delete that.Attributes[key];
            }
            if(typeof that.AttributesChange[key] == "function")
            {
                  that.AttributesChange[key](newValue, previousValue);
            }
      });
});

observer.observe(element, { attributes: true }); // Im HTML-Knoten
                                                 // "element" soll auf
                                                 // Attribut-Änderungen
                                                 // reagiert werden
```

Ein Objekt, zum Beispiel ein Liniendiagramm, erhält die Funktion der Basisklasse, indem man die Basisklasse in seinen Prototyp einträgt:

```
LineChart.prototype = new AttributeSetter();
function LineChart(graphics)
{
      // Implementierung
}
```

Beim Laden einer neuen Seite fehlen zunächst die Diagramm-Objekte. Um sie zu Erzeugen muss der DOM-Baum zunächst nach relevanten Tags durchsucht werden, in denen entscheidende HTML-Attribute gesetzt sind. In dem zuletzt genannten Beispiel wird also nach <svg>-Elementen gesucht, bei dem das Attribut „`bar`“ gesetzt ist. Diese Aufgabe soll die Funktion `registerNewNode` übernehmen. Dieser wird ein HTML Knoten übergeben und durchsucht ihn vollständig nach relevanten Tags.

Weil beim Laden der Seite der komplette Baum zu durchsuchen ist, wird der Funktion zu Beginn der komplette DOM-Baum übergeben:

```
registerNewNode(document);
```

Wird nachträglich ein unbekannter Baum in den DOM-Baum eingefügt, muss dieser ebenfalls durchsucht werden. Dafür existiert im Browser das Ereignis `DOMNodeInserted`:

```
window.addEventListener("DOMNodeInserted", function(event)
{
      registerNewNode(event.target);
}, false);
```

Somit werden die Objekte für die einzelnen Diagramme automatisch erzeugt. Die Objekte sind durch die Oberklasse in der Lage auf sicheren Weg die Attribute zu lesen und auf deren Änderungen zu reagieren.

Kommen wir nun zur Grafikausgabe über einer definierten Schnittstelle.

3.3.3 Grafikschnittstelle

Nachdem geklärt ist wie die einzelnen Objekte erzeugt werden und welche Rolle sie in der Vererbungshierarchie einnehmen, soll Anhand des Liniendiagramms grob der Weg bis zur fertigen Darstellung verfolgt werden.

Zunächst betrachten wir die Grafikschnittstelle welche im jedem Diagramm-Objekt Verwendung findet. Das einzelne Diagramm-Objekt soll keinen Unterschied erkennen können, ob er gerade eine Vektorgrafik oder Pixelgrafik erstellt.

Dazu wird ein Objekt definiert, dass sowohl für <svg> als auch für <canvas> dieselben Schnittstellen bereitstellt. Zu jedem unterstütztem Tag, welches für eine Grafikausgabe genutzt wird, existiert ein Eintrag im Objekt mit demselben Namen. Daher ist die richte Schreibweise wichtig. Nachträglich können so neue Tags hinzugefügt werden, ohne die Programmlogik zu ändern

```
var Graphics =
{
	svg: function(tag)
	{
		this.Schnittstelle1 = function() { }
		this.Schnittstelle2 = function() { }
		this.Schnittstelle3 = function() { }

	},
	canvas: function(tag)
	{
		this.Schnittstelle1 = function() { }
		this.Schnittstelle2 = function() { }
		this.Schnittstelle3 = function() { }
	}
}
```

Es existiert somit 2 Klassen mit denselben öffentlichen Methoden:

```
Graphics.svg
Graphics.canvas
```

Beim Erzeugen eines neuen Diagramm-Objekts wird entweder eine Instanz der Klasse `Graphics.svg` oder eine Instanz der Klasse `Graphics.canvas` für die Grafikausgabe übergeben. In beiden Fällen kann das Diagramm-Objekt beispielsweise die `Schnittstelle3()` aufrufen. Sie bewirkt dieselbe Grafikoperation, doch im Hintergrund werden zwei verschiedene Techniken eingesetzt.

Die Verknüpfung von HTML-Element, Grafikschnittstelle und Diagramm-Objekt geschieht in der Funktion `registerNewNode` in 3 Schritten:

1. Es wird geprüft, ob ein Zusammenhang zwischen Tag-Name und Attribut-Name besteht. Dazu werden alle unterstützten Tags der Grafikbibliothek durchlaufen und mit den zu registrierenden HTML-Knoten verglichen. Eine genaue Liste der gültigen HTML-Attribute liegt in der Hilfsfunktion `bindNode`.
2. Eine neue Instanz der Grafikschnittstelle wird erstellt. Dem Konstruktor wird das entsprechende HTML-Element übergeben auf den gezeichnet werden soll.

3. Ein neues Diagramm-Objekt wird erzeugt. Der Diagrammtyp ist abhängig vom gesetzten HTML-Attribut. Beim Instanziieren wird dem Konstruktor eine Instanz der Grafikschnittstelle übergeben. Diese wird zum Zeichnen des Diagramms genutzt.

Der entsprechende Quelltext lautet wie folgt:

```
// key ist im ersten Durchlauf "svg", im zweiten "canvas"
for(var key in Graphics)
{
      if(node['nodeName'].toUpperCase() == key.toUpperCase())
      {
            bindNode(node, new Graphics[key](node));
      }
}
```

Die Grafikschnittstelle beinhalten Operationen wie das Zeichnen einer geglätteten Linie oder das Erstellen von einem Farbverlauf.

Diese Liste ist eine Auswahl der implementierten Schnittstellen:

```
Clear()                          // Bildschirmausgabe löschen
Lines(lines, width, color)       // mehrere Linien zeichnen
Path(points, width, color, fill) // eckige oder runde Linie zeichnen
Polygon(points, color)           // Vieleck zeichnen
Rectangle(coordinate, width, height, color) // Rechteck zeichnen
FramedRectangle(coordinate, width, height, options) // Rechteck mit Rahmen
                                                    // und Fülleffekt
Circle(coordinate, radius, color) // Punkte zeichnen
SetPixel(bits, color)             // mehrere Pixel manipulieren,
                                  // bei SVG Rechtecke
```

Der Schnittstelle `Path` wird ein Array (also eine Aufzählung) von Koordinatenpunkten übergeben. Diese werden Schritt für Schritt zu einer geschlossenen Linie verbunden. Diese Linie ist nicht geglättet:

```
[[0, 2], [1, 1],[3, 1]]
```

(Pseudocode)

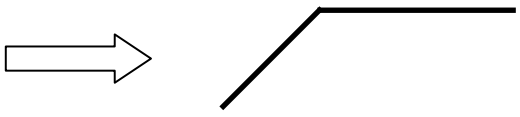


Abb. 18: nicht geglättete geschlossene Linie

Ist bei einem Punkt nachträglich die Eigenschaft `Circle` gesetzt, wird mit der Hilfe diesem Punkt und den vorangegangenen Punkt eine quadratische Bézierkurve erzeugt:

```
[[0, 2],
 [1, 1].Circle(1, 2),
 [3, 1].Circle(2, 0)]
```

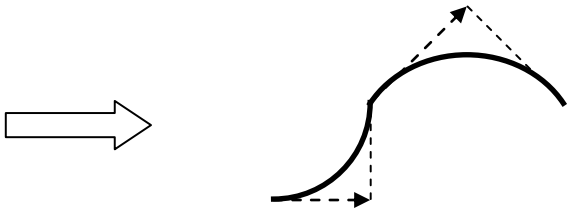


Abb. 19: quadratische Kurve mittels Stützpunkte

Über diese Hilfskoordinaten kann eine Linie geglättet werden. Das Kreisdiagramm nutzt diese Stützpunkte um einen gestauchten Kreisbogen zu erzeugen. Wird bei einem SVG-Liniendiagramm die Ansicht stark vergrößert, so besitzt die Linie mit dieser Technik im Idealfall keine scharfen Ecken.

Jede Funktion liegt wie bereits erwähnt in zwei Varianten vor. Beim HTML5 <canvas> Objekt ist eine richtige Leinwand vorhanden auf der gezeichnet werden kann. Diese ist als Feld mit dem Bezeichner „context“ in der Klasse hinterlegt:

```
this.Circle = function(coordinate, radius, color)
{
      if(notSupport) { return; }
      var color = color || "black";
      context.fillStyle = color;
      context.beginPath();
      context.arc(coordinate.X, coordinate.Y, radius, 0, 2 * Math.PI);
      context.fill();
}
```

Bei SVG muss hingegen mit dem DOM-Baum gearbeitet werden. Jedoch nicht mit dem klassischen `document.createElement()`-Befehl, sondern ein DOM Level höher:

```
this.Circle = function(coordinate, radius, color)
{
      var color = color || "black";
      var shape = createSvgShape();
      document.createElementNS("http://www.w3.org/2000/svg", "circle");
      shape.setAttributeNS(null, "cx", coordinate.X);
      shape.setAttributeNS(null, "cy", coordinate.Y);
      shape.setAttributeNS(null, "r",  radius);
      shape.setAttributeNS(null, "fill", color);
      element.appendChild(shape);
}
```

Bei SVG müssen für einen Farbverlauf mehrere Elemente erzeugt werden. Zunächst wird ein Gradienten-Element erzeugt und in den DOM-Baum hinzugefügt. Diese nimmt die Parameter mit Hilfe weiteren Unterknoten auf. Den Gradienten vergibt man eine HTML-ID. Dieser wird bei dem Füll-Attribut im entsprechenden SVG-Shape angegeben.
Bei dieser Arbeit wurde eine fortlaufende globale ID verwendet die mit „svg-gradient-“ beginnt. Bei jedem neuem Farbverlauf wird die ID um eins erhöht. Wird die Zeichenfläche oder der letzte Gradient gelöscht wird die ID nicht herunter gesetzt. Der Grund hierfür ist die Vielzahl an Grafik-Instanzen die gleichzeitig existieren. Diese erzeugen und löschen Gradienten in einer unterschiedlich zeitlichen Abfolge.

```
<linearGradient id="svg-gradient-18" x1="0%" y1="0%" x2="100%" y2="0%">
    <stop style="stop-color:#999999" offset="0"></stop>
    <stop style="stop-color:rgba(213, 213, 213, 1)" offset="1"></stop>
</linearGradient>
<polygon points="18,54 473,54 473,146 18,146" fill="url(#svg-gradient-18)" stroke="black"
stroke-width="1.5" stroke-linejoin="miter" stroke-linecap="butt"></polygon>
```

Abb. 20: Linearer Gradient

3.3.4 Koordinatensystem

Im vorangegangenen Kapitel wurde geklärt wie die Grafikausgabe im Groben funktioniert. In diesem Kapitel geht es um das Erzeugen und Zeichnen des Koordinatensystems. Als ersten Schritt wird auf der logischen Ebene eine Klasse abstrahiert die zwischen den Punkten auf der Zeichenfläche und denen im Koordinatensystem eine Verbindung herstellt. Beide Unterscheiden sich. Die vertikale Ausrichtung verlaufen unterschiedlich und das dargestellte Koordinatensystem besitzt eine andere Skalierung.
Zwei Methoden sollen die Koordinaten von der einen Skalierung in die andere umrechnen:

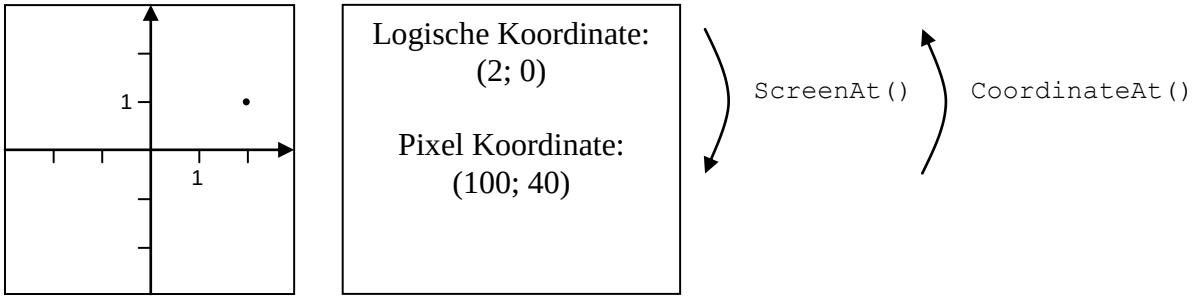


Abb. 21: Umrechnung zwischen logischer und dargestellter Koordinate

Dabei muss die Breite und die Höhe der Zeichenfläche berücksichtigt werden. Es werden nur die Gitternetzlinien gezeichnet, die tatsächlich im Darstellungsbereich liegen. Hinzu kommt der Vergrößerungsfaktor und die aktuelle Verschiebung. Diese Parameter sind in verschiedenen Feldern abgespeichert. Relevante Daten über das Koordinatensystem sind öffentlich zugänglich und gegebenenfalls über Methoden änderbar.

Die fertige Gleichung für die Hin- und Zurückrechnung wurde in den zwei öffentlichen Methoden implementiert.

```
this.ScreenAt = function(x, y)
{
  return new Coordinate
  (
     (x + this.Position.X) * zoomFactor * zoomX + (Size.X / 2) + this.Offset.X,
    -(y - this.Position.Y) * zoomFactor * zoomY + (Size.Y / 2) + this.Offset.Y
  );
}

this.CoordinateAt = function(x, y)
{
  return new Coordinate
  (
     (x - (Size.X / 2)) / zoomFactor / zoomX - this.Position.X,
    -(y - (Size.Y / 2)) / zoomFactor / zoomY + this.Position.Y
  );
}
```

Der Offset dient dazu die Darstellung um einen halben Pixel zu verschieben damit eine Linie die eine Breite von genau einem Pixel hat nicht auf zwei benachbarte Pixel zur Hälfte gezeichnet wird. Für das Histogramm soll es die Möglichkeit geben die Darstellung zu strecken und zu stauchen, sowohl horizontal als auch vertikal. Zum Beispiel ist es ratsam die y-Achse nur zwischen 0 und 1 darzustellen. Dafür sind die Felder `zoomX` und `zoomY` zuständig.

Der Vergrößerungsfaktor wird nicht linear verändert. Kleine Intervalle könnten so nicht dargestellt werden und größere nie erreicht. Stattdessen wird ein separates Feld „`zoom`“ genutzt. Diese wird linear geändert und bildet ein Maß für die aktuelle Vergrößerung. Um den tatsächlichen Vergrößerungsfaktor zu erhalten wird Potenziert, `zoom` steht dabei im Exponenten. Anschließend wird mit dem Gitterlinien-Abstand multipliziert. Bei unveränderter Darstellung ist der Abstand zwischen zwei Einheiten 10 Pixel breit.

`zoom`	`zoomFactor = Math.pow(2.0, zoom / 2.0) * gridGab`
-2	5
-1	7.07
0	10
1	14.14
2	20
3	28.28

Das `zoom` Feld wird somit für den Interpreter immer als Ganzzahl interpretiert, es kommt zu keinen Rundungsfehlern wenn stark vergrößert und danach wieder stark verkleinert wird. Weil JavaScript schwach typisiert kann ohne weiteres Zutun die Schrittweite verkleinert werden. Dann wird der Wert als Gleitkommazahl in der für den Prozessor üblichen IEEE 754 Norm gerechnet.

Das Verschieben und Vergrößern des Koordinatensystems geschieht über die Maus. Dabei sind verschiedene Dinge zu beachtet. Wird innerhalb der Zeichenfläche die linke Maustaste gedrückt, beginnt das Verschieben des Diagramms. Das Loslassen kann auf zwei Bereichen geschehen.

Entweder wird die Maustaste innerhalb der Grafik losgelassen:

```
element.addEventListener("mouseup", function(e) { }, false);
```

Oder außerhalb:

```
window.addEventListener("mouseup", function(e) { }, false);
```

Soll mit Linksklick ein neuer Punkt im Streudiagramm gesetzt werden, so braucht das Diagramm erst die Koordinaten. Für die <canvas>-Oberfläche wird im Maus-Ereignis die Werte pageX bzw. pageY in Kombination mit den Koordinaten der Zeichenfläche genutzt.

SVG Grafik besitzen jedoch selbst eine Vielzahl von HTML Unterknoten. Ein Maus-Over oder ein Maus-Klick geben die Koordinaten eines untergeordneten Shapes zurück, zum Beispiel innerhalb eines bereits vorhandenen Rechtecks. Hier ist man auf spezielle SVG Funktionen wie `getScreenCTM` angewiesen.

Um das Maus-Rad richtig abzufragen sind verschiedene Browser zu berücksichtig. Je nach Kompatibilität wird das Ereignis `"mousewheel"` oder `"DOMMouseScroll"` ausgelöst. Beim Internet Explorer werden die Informationen über das Ereignis nicht als Parameter mit der aufgerufenen Funktion übergeben. Stattdessen befinden sich diese Informationen im `window` Objekt:

```
var event = window.event || event;
```

Alle diese Workarounds sollen nicht jedes Mal neu geschrieben werden. Sie werden in die Klasse „`MouseHelper`“ gekapselt. Nach außen hin stellt sie eine einfach funktionierende Schnittstelle bereit. Dem Konstruktor wird das HTML-Element übergeben an den die Klasse gebunden wird. Die Instanz löst dann automatisch Ereignisse aus, die für das Koordinatensystem relevant sind:

```
var mouse = new MouseHelper(element);
mouse.MoveHandler = function(move)
{
    // Das Diagramm wurde von der Maus verschoben.
    // Die Richtung und Weite ist in move gespeichert.
}

mouse.WheelHandler = function(delta)
{
    // Mausrad wurde gedreht.
    // Ist delta == true, dann heranzoomen
    // sonst heraus.
};
```

Die Klasse für das Koordinatensystem zusammen mit der Maus-Hilfe wird in der Klasse genutzt, die ein interaktives Koordinatensystem repräsentiert:

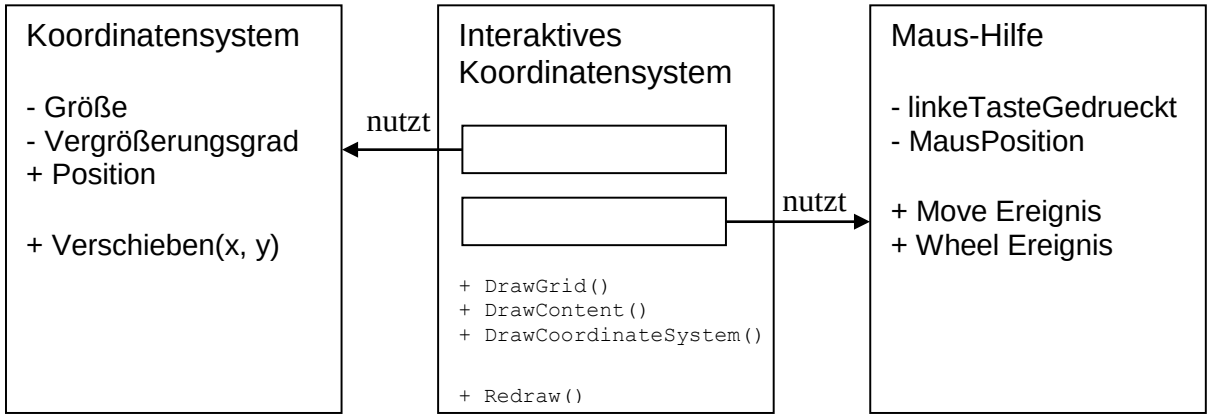


Abb. 22: Aufbau der Oberklasse für ein interaktives Koordinatensystem

Diese Klasse wird für alle Unterklassen mit einen Koordinatensystem eingesetzt. Sie besitzt drei Grafik-Operationen die öffentliche sind.

→ `DrawGrid()`
Das Koordinatensystem kann ein optionales Gitternetz besitzen. Das hängt davon ab ob das HTML-Attribut „`grid`“ gesetzt ist.

→ `DrawContent()`
Hier wird abhängig vom dargestellten Bereich der Inhalt gezeichnet. Beim Liniendiagramm ist es ein Graph, beim Streudiagramm sind es die Punkte mit der Korrelation, beim Balkendiagramm sind es das stetige Zielmerkmal, und so weiter.

→ `DrawCoordinateSystem()`
Hier wird über das Zeichenfeld ein Koordinatensystem mit korrekt skalierte Achsen und Beschriftung gelegt.

Diese 3 Operationen werden nach einander in der Methode `Redraw()` aufgerufen. Sie löscht vorher das Zeichenfeld. Sobald sich der darzustellende Bereich oder die Daten vom Diagramm ändern, wird diese erneut aufgerufen.

Soll ein Gitternetz gezeichnet werden besteht ein ähnliches Problem wie beim Vergrößerungsfaktor. Der Abstand der Gitternetz Linien darf nicht proportional zum Vergrößerungsfaktor sein. Wird für ein größeres Intervall die Darstellung verkleinert, so sind zu viele Gitternetzlinien auf einmal sichtbar. Zum Beispiel 1000-mal pro Einheit bei einem Maßstab von 1 zu 1000. Die Ausgabe wäre dann Schwarz oder reagiert nicht mehr.

Ohne auf die mathematische Herleitung einzugehen, wird der logische Abstand des Gitternetzes in Abhängigkeit vom aktuellen Vergrößerungsfaktor über diese Zeile bestimmt:

```
var grid = Math.pow(10, 1 - Math.floor(Math.log(zoomFactor / 2) / Math.log(10)));
// Math.floor runden auf die naechste Ganzzahl ab
// Math.log bildet den natürlichen Logarithmus
```

zoom	zoomFactor	grid
0	10	10
2	20	1
4	40	1
6	80	1
8	160	1
10	320	0.1
12	640	0.1
14	1280	0.1
16	2560	0.01

Damit steht die Klasse für ein interaktives Koordinatensystem. Um sie sinnvoll zu nutzen reicht es aus die Methode `DrawContent()` in der jeweiligen Unterklasse zu überschreiben. Liegen dem Diagramm neuen Daten vor, so wird `Redraw()` aufgerufen. Der Rest wird von den gekapselten Methoden erledigt:

```
LineChart.prototype = new AttributeSetter();
function LineChart(graphics)
{
    var interaction = new InteractiveCoordinateSystem(graphics);
    interaction.DrawContent = function()
    {
        /* Graph zeichnen */
    }

    this.AttributesChange.LINECHART = function() { interaction.ReDraw(); }
}
```

Alle hier vorgestellten 2D-Diagrammen folgen demselben Prinzip. Sie überschreiben die `DrawContent()`-Methode und lesen ihre HTML-Attribute aus. Die Methode muss wissen was sie Zeichnen soll. Beim Liniendiagramm wird dazu die Zeichenkette im HTML-Attribut „`linechart`“ ausgewertet, beim Balkendiagram „`barchart`“, bei Kreisdiagramm „`piechart`“, und so weiter. Jedes Diagramm besitzt ein anderen Inhalt, somit unterscheidet sich der Aufbau der Zeichenkette im HTML-Attribut.

Zur Erinnerung: Soll beispielsweise ein Liniendiagramm zwei Graphen besitzen, soll es ausreichen folgendes in die HTML-Seite zu schreiben:

```
<svg width="500" height="500" linechart="x^2, red; sin(x), blue"
    grid
</svg>
```

Die Graphen sind durch Semikolon getrennt und per Komma wird optional eine Farbe angegeben. Somit werden zwei Graphen mit folgenden Eigenschaften definiert:

1. Graph Farbe: Rot Gleichung: $y = f(x) = x \bullet x$
2. Graph Farbe: Blau Gleichung: $y = f(x) = \sin(x)$

Um aus einer gegebenen Zeichenkette die einzelnen Parameter zu extrahieren, wurde zunächst die Standard-Funktion `split()` genutzt:

```
var s = "term1;term2;term3".split(';')
// s ist ein Array mit 3 Einträgen ["term1", "term2", "term3"].
```

Jedoch können Farbangaben per CSS3 Kommas beinhalten, die sich in Klammern befinden.

```
rgb(255, 255, 255)
```

Auch der hier umgesetzte Colorpicker liefert bei einen transparenten Alpha Kanal so eine Zeichenkette. Der Aufruf von `split(',')` teilt hier die Farbangabe. Das ist nicht wünschenswert, weil so die Farbe nicht mehr erkannt wird.

Um dieses Problem zu lösen wurde die Funktion `splitBrackets` implementiert. Sie verhält sich genauso wie `split`, jedoch wird innerhalb einer Klammer nicht geteilt.

3.3.5 Parser

Nachdem geklärt ist wie das Koordinatensystem in den einzelnen Diagramm-Objekten entsteht und wie die Daten gespeichert werden, geht es in diesen Kapitel um die Verarbeitung der Eingaben vom Nutzer. Diese landet letztendlich als Zeichenkette in das HTML-Attribut der Zeichenfläche. Beim Liniendiagramm ist dort der Funktionsgraph hinterlegt.

Um den Graphen zu zeichnen werden in die Funktion verschiedene Werte eingesetzt. Gestartet wird auf der linke Seite der Zeichenfläche. Dort wird im Koordinatensystem der logische x-Wert bestimmt, in die Funktion eingesetzt um daraus den y-Wert zu berechnet. Dies wird mit dem rechts stehenden Pixel solange wiederholt bis die rechte Seite der Zeichenfläche erreicht ist. Mit Hilfe Grafikschnittstelle „`Path`“ und den ermittelten Punkten wird so eine geschlossene Linie erzeugt.

Aber wie wird die Gleichung aufgelöst wenn es sich hierbei um eine Zeichenkette handelt? Eine rabiate Lösung ist der Einsatz der `eval()`-Funktion. Sie gehört zum JavaScript Sprachumfang und führt das übergebene Argument aus, wenn diese interpretierbar ist. Handelt es sich um Anweisungen werden diese ausgeführt, wird ein Objekt als Zeichenkette übergeben gibt sie eine Instanz zurück oder, und das ist der wichtige Teil, wird ein arithmetischer Ausdruck übergeben wird dieser aufgelöst. Dieser kann auch komplizierte Klammerausdrücke beinhaltet.

Dieses Vorgehen hat einen Haken. Die ungefilterte Weitergabe von Nutzereingaben an `eval()` ist eine Sicherheitslücke. Es wird ein unvertrautes Skript in einer vertrauten Umgebung ausgeführt. Das heißt, wenn der Nutzer statt einen arithmetischen Ausdruck den Befehl „`alert("XSS!");`“ eingibt, erscheint im Browser ein Popup. Nicht umsonst sprechen manche Programmieren von „eval is Evil“.

Aus diesem Grund wird in dieser Arbeit einen eigenen JavaScript-Parser eingesetzt, um einen arithmetischen Ausdruck mit einen oder mehreren Unbekannten aufzulösen. Für die Nutzung wird eine neue Instanz vom Parser erstellt. Dem Konstruktor wird der gegebene arithmetische Ausdruck und eine Liste der darin vorkommenden unbekannten Variablen übergeben. Beide Parameter sind Zeichenketten. Die Variablen dürfen jeweils nur ein Zeichen lang sein.

```
var parser = new Parser("x * (y + 1)", "x y");
```

Die Parameter können später über die Methode `New()` geändert werden, ohne eine neue Instanz zu erzeugen. Die Methode `OK()` überprüft, ob es sich um einen gültigen Ausdruck handelt:

```
if(!parser.OK()) { return; }
```

Nach der Übergabe des Ausdrucks geben die Ereignisse `Confirmed` und `Error` eine Beschreibung über den Zustand das Parsers zurück. Zum Beispiel der Hinweis ob eine Klammer nicht geschlossen wurde.

Die Methode `Solve()` löst den Ausdruck auf. Über ein Objektliteral werden die einzelnen Werte für die unbekannten Variablen übergeben:

```
var z = Parser.Solve({x: 4, y : 1});
```

Die Methode liefert als Rückgabewert das Ergebnis der Auflösung.

Der Parser arbeitet vereinfacht beschrieben wie folgt. Die Zeichenkette wird in seine Bestandteile zerlegt. Dabei findet die erste Überprüfung der Syntax statt (doppelter Operator, Klammer nicht geschlossen, …).

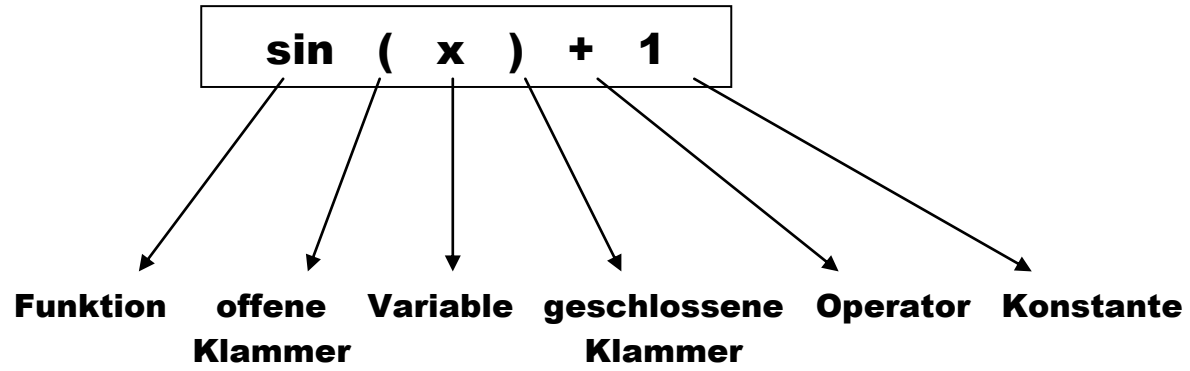
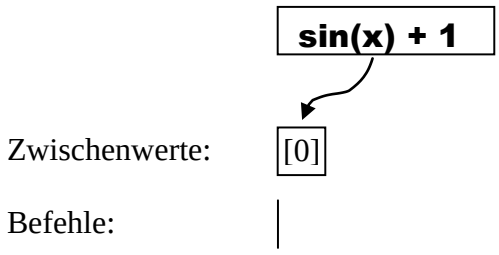


Abb. 23: Term-Zerlegung

Nun werden die Prioritäten, die durch die Klammer-Ausdrücke entstehen vergeben:

0 1 1 1 0 0

Es werden zwei Arrays angelegt. Eins für die Zwischenwerte die beim Auflösen eines Ausdrucks auftreten können und eins für die Befehle die auf die Zwischenwerte angewendet werden. Die ersten Zwischenwerte sind ein Verweis zu den bereits vorhandenen unbekannten Variablen. Jede Variable besitzt immer ein Verweis zu einer bestimmten Speicherzelle. Im Laufe der Auflösung ändert sich diese. Die einzelnen Befehle werden in Form von Funktionen hinterlegt.



Nun beginnt die Schrittweise Reduktion. Dabei wird der Vorrang der Operatoren berücksichtigt. Als erstes werden jedoch die Elemente mit der höchsten Priorität aufgelöst.

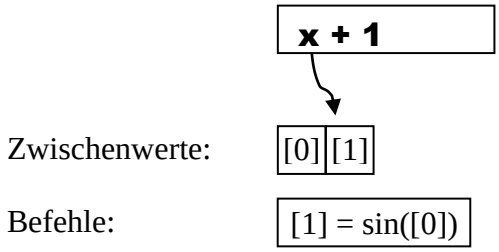


Abb. 24: Erstellung einer Befehlsfolge bei der Term-Auflösung

Kann nicht mehr reduziert werden, gibt es drei Fälle:

1. Es ist eine Konstante übrig geblieben. Dann liefert `Solve()` immer denselben Wert.
2. Es ist eine Variable übrig geblieben. Sie zeigt auf die Speicherzelle in dem nach der Auflösung das Ergebnis drinnen steht.
3. Es sind zwei oder mehrere Elemente übrig geblieben. Dann ist der Term ungültig.

Beim Aufruf der `Solve()`-Methode werden die einzusetzenden Variablen in die ersten Speicherzellen des Zwischenwert-Arrays geschrieben. Danach werden die gespeicherten Befehle in der richtigen Reihenfolge ausgeführt. Diese erzeugen neue Zwischenwerte. Ein Feld zeigt nach Abarbeitung auf die Speicherstelle mit dem Ergebnis.

Das Umwandeln der Zeichenkette in ein Array aus Befehlen geschieht nur *einmal*. Sodass dieser Overhead nur kurz auftritt, sobald der Nutzer den Term abändert. Beim wiederholten Aufruf der `Solve()`-Methode mit unterschiedlichen Variablen wird einmalig das Array mit den Befehlen durchlaufen. Es findet jedoch keine neue Umwandlung der Zeichenkette statt.

Um die Laufzeit zu verkürzen wird für den Puffer mit dem Zwischenwerten ein typisiertes Array eingesetzt:

```
_puffer = new Float32Array(_puffer);
```

Diese Art von Arrays sind neu. Sie vergeben jeden Eintrag einen bestimmten Typ. `Float32Array` bedeutet, dass statt den üblichen 64 Bits für die Kodierung von Gleitkommazahlen nur 32 Bit Genauigkeit genutzt wird. Je nach Prozessor also eine Halbierung der Laufzeit. 32 Bit reichen für diesen Anwendungsfall völlig aus.

Nachdem nun innerhalb der Bibliothek ein funktionierenden Parser bereitsteht, soll nun der Nutzer bei der Eingabe in einem <input>-Feld eine Hilfestellung bekommen, sobald es zur Fehleingabe kommt. Das Textfeld soll blinken und unterhalb soll die Begründung dafür stehen. Dieser taucht durch ein Fade-Out auf:

Abb. 25: Fehleingabe eines Terms

Für die Umsetzung wurden die neuen CSS Eigenschaften genutzt, welche die JavaScript-Bibliothek in den <head>-Bereich der Seite und in das `style`-Attribut schreibt.

Das Blinken des Eingabefelds geschieht über die Definition eines Keyframes. Dieser beschreibt den Übergang zwischen einen transparenten und farbigen Rahmen.

```
@keyframes termerroranimation
{
	0% { box-shadow: rgba(255, 0, 0, 0) 0px 0px 2px 1px; }
	100% { box-shadow: red 0px 0px 2px 1px;  }
}
```

Das Eingabefeld wird den Keyframe zugeordnet, das geschieht mit Hilfe der CSS-Eigenschaft „`animation`“. Die Animation soll 4 Sekunden dauern und endlos laufen:

```
element.style
{
      animation: termerroranimation 4s infinite;
}
```

Zurzeit ist bei der `animation`-Eigenschaft zwingend der Einsatz von Vendor-Präfixe erforderlich. Damit das Blinken nicht nur in einem Browser funktioniert, werden also zusätzlich die Eigenschaften „`-o-animation`“, „`-moz-animation`“ und „`-webkit-animation`“ gesetzt.

Für den Fade-Out bzw. Fade-In recht es aus die Änderung der Position für die Fehlermeldung zu verlangsamen. Dazu dient die CSS-Eigenschaft „`transition`“. Die Box für die Fehlermeldung wird *hinter* dem Eingabeformular platziert. Die Änderung der Position erfolgt über die CSS-Eigenschaft `padding-top`.

Um die Sache abzurunden verwischt die Schriftzug sobald die Eingabe korrigiert wurde. Das geschieht über ein Schatten der allmählich größer wird.

Neben der Eingabe einer gültigen Funktionsgleichung ist eine komfortable Auswahl einer Farbe angebracht, um so beispielsweise die verschiedenen Merkmalsausprägungen visuell hervorzuheben. Dieses Problem wird im nächsten Kapitel gelöst.

3.3.6 Colorpicker

Bei mehreren Linien ist es ratsam verschiedene Farben einzusetzen. Für eine komfortable Auswahl wird in der Informatik eine „Farbauswahlbox“ bzw. ein „Colorpicker“ eingesetzt. In dieser Arbeit wurde ein eigener Colorpicker entwickelt, der bei den verschiedenen Diagrammtypen Anwendung findet.

Er besteht aus 3 Reglern für die Farben Rot, Grün und Blau und einem Kanal für die Transparenz. Das große Rechteck dient für die Festlegung der Farbsättigung. Alle enthaltenen HTML-Elemente werden dynamisch per JavaScript erzeugt.

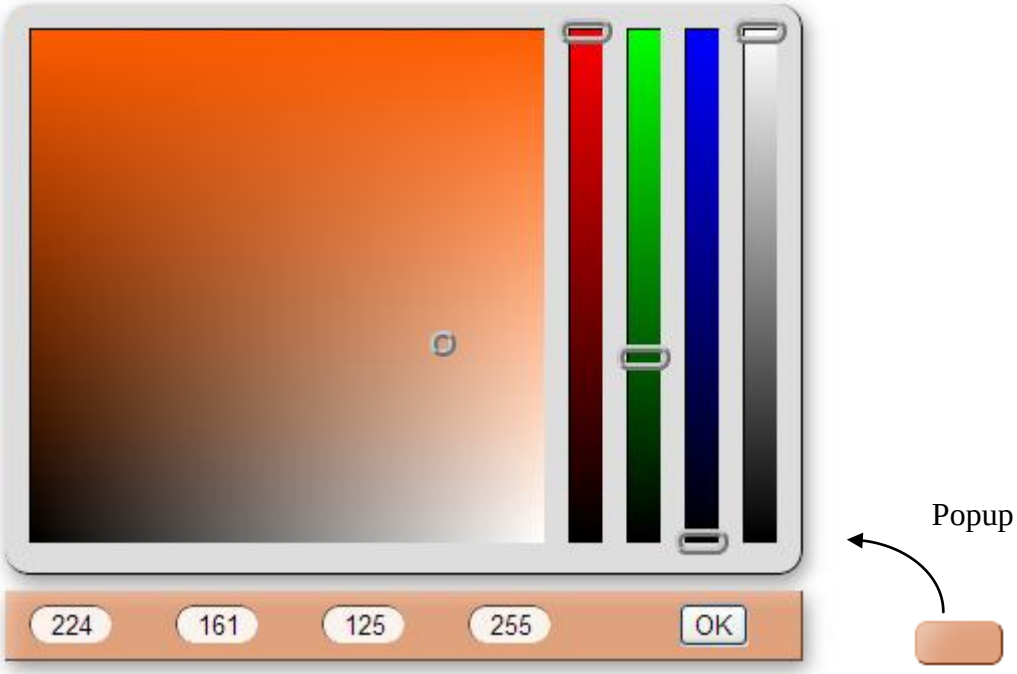


Abb. 26: Colorpicker zur Laufzeit

Ziel ist es den technischen Aspekt, also die Angabe von Hexadezimalwerte vor dem Nutzer zu verbergen.

Um den Colorpicker in eine Seite einzubinden, wird ein <input>-Feld erstellt bei dem das „colorpicker“ Attribut gesetzt ist:

```
<input colorpicker>
```

Wird beim Laden der Seite oder beim Hinzufügen eines neuen HTML-Knoten dieses Attribut erkannt, wird zunächst das `value` Attribut ausgewertet. Dieses nimmt die Eingaben des Nutzers entgegen. Daraus wird die Anfangsfarbe bestimmt. Ist kein Eingabewert vorhanden wird die Farbe Grau gewählt.

Beim ersten Start wird aus der Anfangsfarbe die Farbsättigung errechnet und im Auswahlfenster eingetragen. Danach wird an Hand der Farbsättigung zusammen mit der Anfangsfarbe der der Rot, Grün und Blau Anteil bestimmt. Damit werden die Regler eingestellt. Der Transparenz Kanal wird unverändert übernommen, falls einer vorhanden ist. Beim nachträglichen Schließen und Öffnen des Colorpickers entfällt dieser Schritt, der innere Zustand wird gespeichert.

Das <input>-Feld wird in einen Button umgewandelt und grafisch angepasst. Er bekommt immer die Farbe die der Nutzer gerade ausgewählt hat. Die Ecken werden abgerundet, ein Rahmen wird gesetzt und es wird eine leichte Reflektion mit Hilfe eines inneren Schatten erzeugt:

```
element.style.boxShadow = "rgba(255, 255, 255, 0.5) 25px 18px 23px -25px inset";
```

Das besondere an diesen Colorpicker sind die zusätzlichen HTML-Attribute die für die einzelnen Kanäle gesetzt werden:

```
<input colorpicker value="#40C4FF" red="64" green="196" blue="255" alpha="255">
```

Das Auslesen und Verändern der einzelnen Farbwerte ist damit über JavaScript leichter möglich. Beim Balkendiagramm, Kreisdiagramm oder dem Histogramm wurde diese Attribute genutzt um ein Farbverlauf oder das Abdunkeln der ausgewählten Farbe zu erzeugen.

Die Eingabe von CSS Farbwerten fällt mit diesen Colorpicker weg. Intern muss jedoch weiter damit gearbeitet werden. Wird ein neuer CSS Farbwert über das `value`-Attribut gesetzt, muss dieser korrekt erkannt werden. Dafür existieren verschiedene Schreibweisen:

Hexadezimal kurz:	#f00
Hexadezimal lang:	#ff0000
Rot-Gelb-Grün-Wert:	rgb(255, 0, 0)
RGB-Wert mit Transparenz	rgba(255, 0, 0, 1)
Farbton, Sättigung, Helligkeit:	hsla(0, 100%, 50%)
HSL-Wert mit Transparenz:	hsla(0, 100%, 50%, 1)
Farbname:	red

Die letzten drei Schreibweisen wurden nicht umgesetzt.
Prinzipiell können in JavaScript alle Schreibweisen bis auf die letzte mit Hilfe von regulären Ausdrücken erkannt und ausgewertet werden. Reguläre Ausdrücke dienen dazu eine Zeichenkette nach bestimmten syntaktischen Regeln zu durchsuchen.
Nehmen wir den RGB-Wert. Eingeleitet wird dieser über die Zeichenfolge „rgb“. Daraufhin folgt eine offene und geschlossene Klammer in dem drei Ganzahlen mit Komma getrennt stehen. Dazwischen können sich beliebig viele Leerzeichen befinden. Beliebig viele Leerzeichen wird über die Folge `\s*` notiert:

```
var values = argument.match(/^rgb\s*\(\s*(\d+)\s*,\s*(\d+)\s*,\s*(\d+)\s*\)$/i);
```

Ganzahlen werden über die Folge `\d+` angegeben. Das Einklammern bewirkt, dass diese Werte aus der Zeichenkette heraus gespeichert werden. Nach diesen Werten wird gesucht.

Bei der RGBA-Angabe wird jedoch als vierten Parameter eine Gleitkommazahl angegeben. Bei einem regulären Ausdruck bleibt einen nichts anderes übrig den Aufbau einer Gleitkommazahl nachzubauen:

[+-]?[0-9]*\.?[0-9]*

Bei den hexadezimal Werten wurde hingegen auf die einzelnen Zeichen der Zeichenkette zugegriffen, um sie mit Hilfe der JavaScript Funktion `parseInt()` in eine Zahl umzuwandeln. Über den zweiten Parameter wird optional das Zahlensystem angegeben, in diesen Fall `16`.

3.3.7 Einbindung und Nutzung der Komponenten

Mit den Klassen für die Diagrammen, den Parser und den Colorpicker haben wir die wichtigsten Komponenten zusammengestellt. Mit diesen kann eine Demo-Seite mit dem Liniendiagramm erstellt werden.

Alle Klassen, die Hilfsfunktionen sowie die globale Variablen sind gekapselt und nach außen nicht sichtbar. Dies geschieht über eine anonyme Funktion die sich selber aufruft. Eine anonyme Funktion ist eine Funktionsdeklaration ohne Funktionsnamen.

```
(function(window)
{
        // Deklaration BarChart, LineChart, PieChart, ...

        window.addEventListener("load", function()
        {
                /* Initialisierung, nachdem die Seite fertig geladen ist */
        } , false);
})(window);
```

Diese Technik heißt „*Block Scope*“. Sie sorgt für eine modulare Nutzung der Bibliothek parallel zu anderen, ohne dass es zu Namenskonflikten kommt. Globale Variable sind nach außen versteckt. Die einzige Schnittstelle nach draußen ist das setzen von zusätzlichen HTML-Attributen.

Um die Bibliothek zu nutzen wird diese über das <script>-Tag eingebunden:

```
<script type="text/javascript" src="grapher.js"></script>
```

Danach wird im <body>-Bereich eine Zeichenfläche deklariert. Diese enthält die zusätzlichen HTML-Attribute:

```
<svg width="500" height="500" linechart="x^2 / 20"></svg>
```

Damit wird ein fertiges Diagramm erzeugt, bei dem der Nutzer bereits die Maus nutzen kann, um die Darstellung zu ändern. Der dargestellte Graph ist statisch vorgegeben. Was fehlt ist die Dynamik, damit der Betrachter der Seite den Graphen abändern oder einen weiteren hinzufügen kann.
Dazu ist zusätzliches JavaScript nötig. Es ändert die Attribute im <svg> oder <canvas>-Tag mit Hilfe der Standard-Funktionen `setAttribute()` oder `removeAttribute()` ab.

Auf diese Weise werden die dargestellten Daten aktualisiert, die Farben der verschiedenen Graphen geändert, die Punkte im Histogramm ausgeblendet oder einen Innenkreis im Kreisdiagramm erzeugt.

Die Demo-Seite für das Liniendiagramm soll zwei Graphen gleichzeitig darstellen und die eingeschlossene Fläche bestimmen. Das Intervall für die Fläche lässt sich eingrenzen:

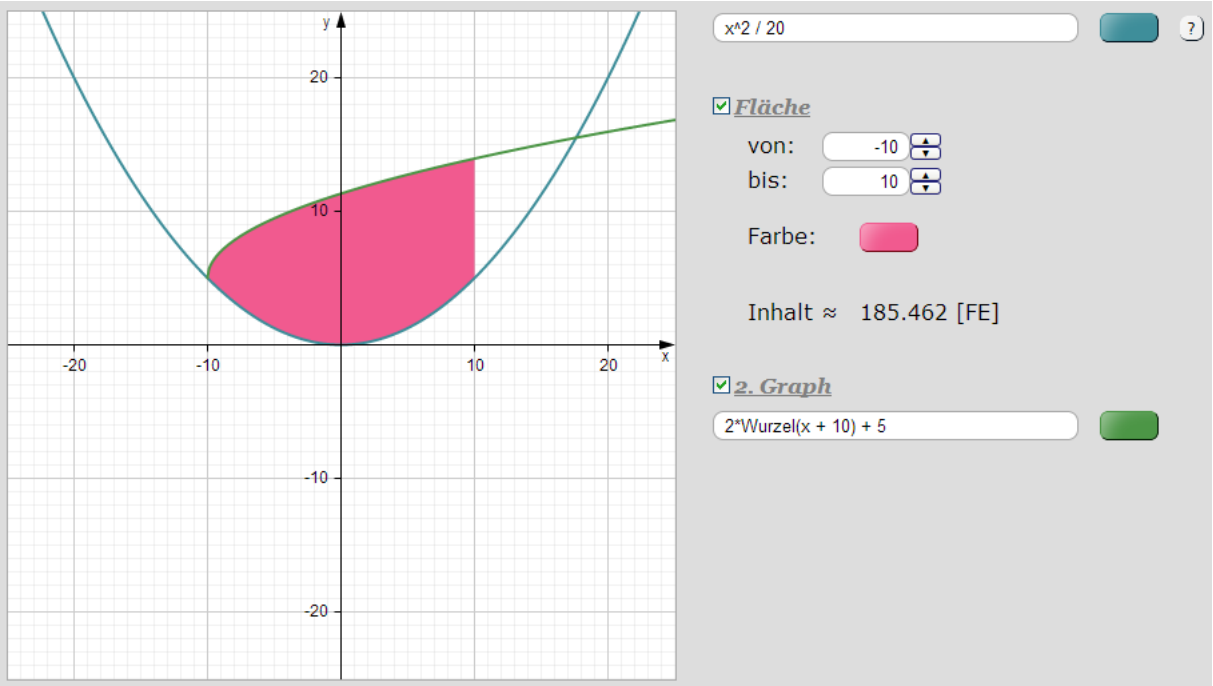


Abb. 27: Liniendiagramm zur Laufzeit

Zu jeden Diagrammtyp ist eine Beispiel-Seite auf der beiliegenden CD beziehungsweise auf www.hs-merseburg.de/~0mwantke zu finden.

3.3.8 Implizites Liniendiagramm

Als Programmierer findet man immer wieder Lösungen oder neue Techniken die nützlich sind. Das Parallelisieren über die WebWorker-Klasse ist so ein Beispiel. Im letzten Teil will ich zwei davon vorstellen. Als erstes möchte ich zeigen wie ich die implizite Darstellung F(x, y) = 0 umgesetzt habe. Der zweite Punkt ist das Erzeugen von 3D Polygonen mit Hilfe von WebGL am Beispiel des Oberflächendiagramms.

Die implizite Darstellung lässt sich ohne spezielle Algorithmen auf einfachere Weise umsetzen. Dazu wird für jeden Bildpunkt die Gleichung aufgelöst, hier am Beispiel von **$0 = F(x,y) = (x^2 + y^2)^{0.5} - 1$**.

	x=	-1,3	-1,1	-0,9	-0,7	-0,5	-0,3	-0,1	0,1	0,3	0,5	0,7	0,9	1,1	1,3
y=															
-1,3		0,84	0,7	0,58	0,48	0,39	0,33	0,3	0,3	0,33	0,39	0,48	0,58	0,7	0,84
-1,1		0,7	0,56	0,42	0,3	0,21	0,14	0,1	0,1	0,14	0,21	0,3	0,42	0,56	0,7
-0,9		0,58	0,42	0,27	0,14	0,03	-0,1	-0,1	-0,1	-0,1	0,03	0,14	0,27	0,42	0,58
-0,7		0,48	0,3	0,14	-0	-0,1	-0,2	-0,3	-0,3	-0,2	-0,1	-0	0,14	0,3	0,48
-0,5		0,39	0,21	0,03	-0,1	-0,3	-0,4	-0,5	-0,5	-0,4	-0,3	-0,1	0,03	0,21	0,39
-0,3		0,33	0,14	-0,1	-0,2	-0,4	-0,6	-0,7	-0,7	-0,6	-0,4	-0,2	-0,1	0,14	0,33
-0,1		0,3	0,1	-0,1	-0,3	-0,5	-0,7	-0,9	-0,9	-0,7	-0,5	-0,3	-0,1	0,1	0,3
0,1		0,3	0,1	-0,1	-0,3	-0,5	-0,7	-0,9	-0,9	-0,7	-0,5	-0,3	-0,1	0,1	0,3
0,3		0,33	0,14	-0,1	-0,2	-0,4	-0,6	-0,7	-0,7	-0,6	-0,4	-0,2	-0,1	0,14	0,33
0,5		0,39	0,21	0,03	-0,1	-0,3	-0,4	-0,5	-0,5	-0,4	-0,3	-0,1	0,03	0,21	0,39
0,7		0,48	0,3	0,14	-0	-0,1	-0,2	-0,3	-0,3	-0,2	-0,1	-0	0,14	0,3	0,48
0,9		0,58	0,42	0,27	0,14	0,03	-0,1	-0,1	-0,1	-0,1	0,03	0,14	0,27	0,42	0,58
1,1		0,7	0,56	0,42	0,3	0,21	0,14	0,1	0,1	0,14	0,21	0,3	0,42	0,56	0,7
1,3		0,84	0,7	0,58	0,48	0,39	0,33	0,3	0,3	0,33	0,39	0,48	0,58	0,7	0,84

Danach wird die so erzeugte Matrix Zeilenweise abgearbeitet. Es werden immer zwei benachbarte Elemente in einer Zeile verglichen. Sobald sich diese im Vorzeichen unterscheiden, werden beide selektiert:

0,84	0,7	0,58	0,48	0,39	0,33	0,3	0,3	0,33	0,39	0,48	0,58	0,7	0,84
0,7	0,56	0,42	0,3	0,21	0,14	0,1	0,1	0,14	0,21	0,3	0,42	0,56	0,7
0,58	0,42	0,27	0,14	0,03	-0,1	-0,1	-0,1	-0,1	0,03	0,14	0,27	0,42	0,58
0,48	0,3	0,14	-0	-0,1	-0,2	-0,3	-0,3	-0,2	-0,1	-0	0,14	0,3	0,48
0,39	0,21	0,03	-0,1	-0,3	-0,4	-0,5	-0,5	-0,4	-0,3	-0,1	0,03	0,21	0,39
0,33	0,14	-0,1	-0,2	-0,4	-0,6	-0,7	-0,7	-0,6	-0,4	-0,2	-0,1	0,14	0,33
0,3	0,1	-0,1	-0,3	-0,5	-0,7	-0,9	-0,9	-0,7	-0,5	-0,3	-0,1	0,1	0,3
0,3	0,1	-0,1	-0,3	-0,5	-0,7	-0,9	-0,9	-0,7	-0,5	-0,3	-0,1	0,1	0,3
0,33	0,14	-0,1	-0,2	-0,4	-0,6	-0,7	-0,7	-0,6	-0,4	-0,2	-0,1	0,14	0,33
0,39	0,21	0,03	-0,1	-0,3	-0,4	-0,5	-0,5	-0,4	-0,3	-0,1	0,03	0,21	0,39
0,48	0,3	0,14	-0	-0,1	-0,2	-0,3	-0,3	-0,2	-0,1	-0	0,14	0,3	0,48
0,58	0,42	0,27	0,14	0,03	-0,1	-0,1	-0,1	-0,1	0,03	0,14	0,27	0,42	0,58
0,7	0,56	0,42	0,3	0,21	0,14	0,1	0,1	0,14	0,21	0,3	0,42	0,56	0,7

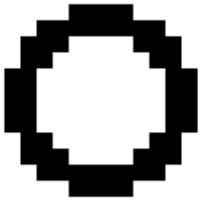
Dasselbe Verfahren wird Spaltenweise angewendet:

0,84	0,7	0,58	0,48	0,39	0,33	0,3	0,3	0,33	0,39	0,48	0,58	0,7	0,84
0,7	0,56	0,42	0,3	0,21	0,14	0,1	0,1	0,14	0,21	0,3	0,42	0,56	0,7
0,58	0,42	0,27	0,14	0,03	-0,1	-0,1	-0,1	-0,1	0,03	0,14	0,27	0,42	0,58
0,48	0,3	0,14	-0	-0,1	-0,2	-0,3	-0,3	-0,2	-0,1	-0	0,14	0,3	0,48
0,39	0,21	0,03	-0,1	-0,3	-0,4	-0,5	-0,5	-0,4	-0,3	-0,1	0,03	0,21	0,39
0,33	0,14	-0,1	-0,2	-0,4	-0,6	-0,7	-0,7	-0,6	-0,4	-0,2	-0,1	0,14	0,33
0,3	0,1	-0,1	-0,3	-0,5	-0,7	-0,9	-0,9	-0,7	-0,5	-0,3	-0,1	0,1	0,3
0,3	0,1	-0,1	-0,3	-0,5	-0,7	-0,9	-0,9	-0,7	-0,5	-0,3	-0,1	0,1	0,3
0,33	0,14	-0,1	-0,2	-0,4	-0,6	-0,7	-0,7	-0,6	-0,4	-0,2	-0,1	0,14	0,33
0,39	0,21	0,03	-0,1	-0,3	-0,4	-0,5	-0,5	-0,4	-0,3	-0,1	0,03	0,21	0,39
0,48	0,3	0,14	-0	-0,1	-0,2	-0,3	-0,3	-0,2	-0,1	-0	0,14	0,3	0,48
0,58	0,42	0,27	0,14	0,03	-0,1	-0,1	-0,1	-0,1	0,03	0,14	0,27	0,42	0,58
0,7	0,56	0,42	0,3	0,21	0,14	0,1	0,1	0,14	0,21	0,3	0,42	0,56	0,7
0,84	0,7	0,58	0,48	0,39	0,33	0,3	0,3	0,33	0,39	0,48	0,58	0,7	0,84

Beide Mengen der selektierten Elemente werden vereinigt:

0,84	0,7	0,58	0,48	0,39	0,33	0,3	0,3	0,33	0,39	0,48	0,58	0,7	0,84
0,7	0,56	0,42	0,3	0,21	0,14	0,1	0,1	0,14	0,21	0,3	0,42	0,56	0,7
0,58	0,42	0,27	0,14	0,03	-0,1	-0,1	-0,1	-0,1	0,03	0,14	0,27	0,42	0,58
0,48	0,3	0,14	-0	-0,1	-0,2	-0,3	-0,3	-0,2	-0,1	-0	0,14	0,3	0,48
0,39	0,21	0,03	-0,1	-0,3	-0,4	-0,5	-0,5	-0,4	-0,3	-0,1	0,03	0,21	0,39
0,33	0,14	-0,1	-0,2	-0,4	-0,6	-0,7	-0,7	-0,6	-0,4	-0,2	-0,1	0,14	0,33
0,3	0,1	-0,1	-0,3	-0,5	-0,7	-0,9	-0,9	-0,7	-0,5	-0,3	-0,1	0,1	0,3
0,3	0,1	-0,1	-0,3	-0,5	-0,7	-0,9	-0,9	-0,7	-0,5	-0,3	-0,1	0,1	0,3
0,33	0,14	-0,1	-0,2	-0,4	-0,6	-0,7	-0,7	-0,6	-0,4	-0,2	-0,1	0,14	0,33
0,39	0,21	0,03	-0,1	-0,3	-0,4	-0,5	-0,5	-0,4	-0,3	-0,1	0,03	0,21	0,39
0,48	0,3	0,14	-0	-0,1	-0,2	-0,3	-0,3	-0,2	-0,1	-0	0,14	0,3	0,48
0,58	0,42	0,27	0,14	0,03	-0,1	-0,1	-0,1	-0,1	0,03	0,14	0,27	0,42	0,58
0,7	0,56	0,42	0,3	0,21	0,14	0,1	0,1	0,14	0,21	0,3	0,42	0,56	0,7
0,84	0,7	0,58	0,48	0,39	0,33	0,3	0,3	0,33	0,39	0,48	0,58	0,7	0,84

Daraus ergeben sich die Pixel der gesuchten Grafik:



3.3.9 Umsetzung Oberflächendiagramm

Das zweidimensionale HTML5 <canvas>-Element stößt bei richtiger 3D-Grafik an seine Grenzen. Die Laufzeit ist zu lang und es treten unter Umständen Grafikfehler auf, weil die Kantenglättung bei jeder Grafikoperation durchgeführt wird. Während bei normalen Anwendungen mehrere Schnittstellen für den GPU-Zugriff zur Verfügung stehen, gibt es im Web für JavaScript momentan nur eine Alternative: WebGL. In Kombination mit einer Bibliothek lassen sich so bequem die Funktionen der Grafikpipeline nutzen. Für die Umsetzung des Oberflächendiagramms habe mich für die WebGL-Grafikbibliothek **„three.js“** entschieden.

Im Grunde kann auch direkt mit WebGL programmiert werden. Diese Grafikschnittstelle arbeitet mit so genannten *Shadern*. Diese Anzulegen ist ein Grund warum das direkte Programmieren aufwendig und fehleranfällig ist. Nicht jede Shader-Typ wird von einer GPU unterstützt. WebGL-Grafikbibliotheken beseitigen diese Konflikte und bieten eine einfache Möglichkeit Effekte wie Licht, Schatten oder Kantenglättung umzusetzen.

In three.js gibt es zwei wichtige Objekte. Die *Szene*, welche die Umgebung mit allen seinen Elementen beschreibt, und die *Kamera*. Davon kann es mehrere geben, aber mindestens eine. Dies ist auf die Szene gerichtet und kann nachträglich ihre Position ändern.

Die Szene wird durch einzelne Objekten beschrieben, die wiederum aus Geometriedaten bestehen. Sowohl die Objekte als auch die Geometriedaten selber können zu einer Gruppe zusammengefasst werden. Gruppen und Szenen haben dieselben Eigenschaften und Methoden wie das Drehen um die eigene Achse.

Innerhalb der Grafikpipeline wird für die 3D-Darstellung mit *Polygonen* gearbeitet. Das sind Dreiecke im Raum. Diese können eine Farbe oder eine Textur besitzen. Für diese Aufgabe reicht ein einfaches Einfärben aus.

Aus der x-y-Ebene wird ein Rechteck gepickt. Zu jeder Ecke wird aus den entsprechenden Argumenten und der gegebenen Funktion die fehlenden z-Werte bestimmt. Die erzeugte Oberfläche aus den x-y-z-Punkten kann mit Hilfe von 2 Polygone im Raum dargestellt werden. Ein Polygon hat jedoch immer nur eine Seite, sodass für die Rückseite erneut 2 Polygone nötig sind.

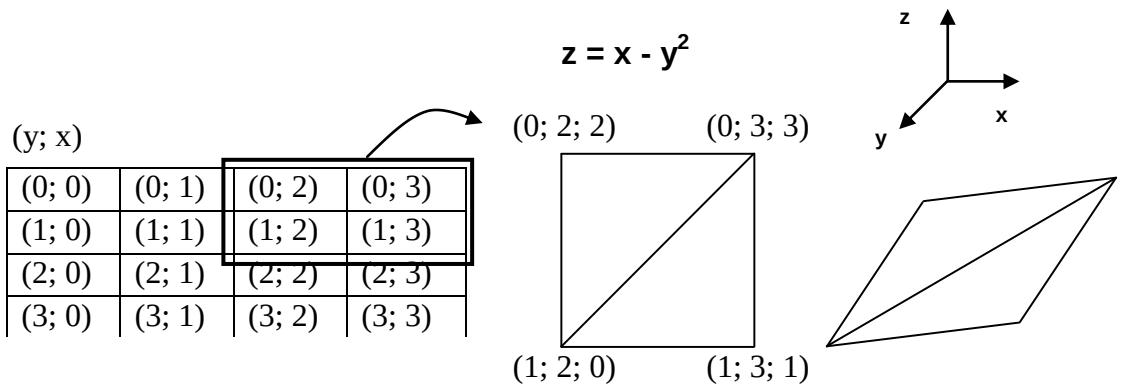


Abb. 28: Berechnung der Oberflächen-Polygone

Pro Rechteck werden so 4 Polygone benötigt. Für die gesamte Oberfläche wird dieser Vorgang mit gleichgroßen Rechtecken an den Nachbarstellen wiederholt. Alle diese Polygone werden als Geometriedaten zu einer Gruppe zusammengefasst und in die Szene hinzugefügt. Das Raster muss nur genügend verkleinert werden um den Eindruck einer gewölbten Oberfläche zu erzeugen. Auf der Demo-Seite wird dies über den Parameter „Genauigkeit“ bestimmt:

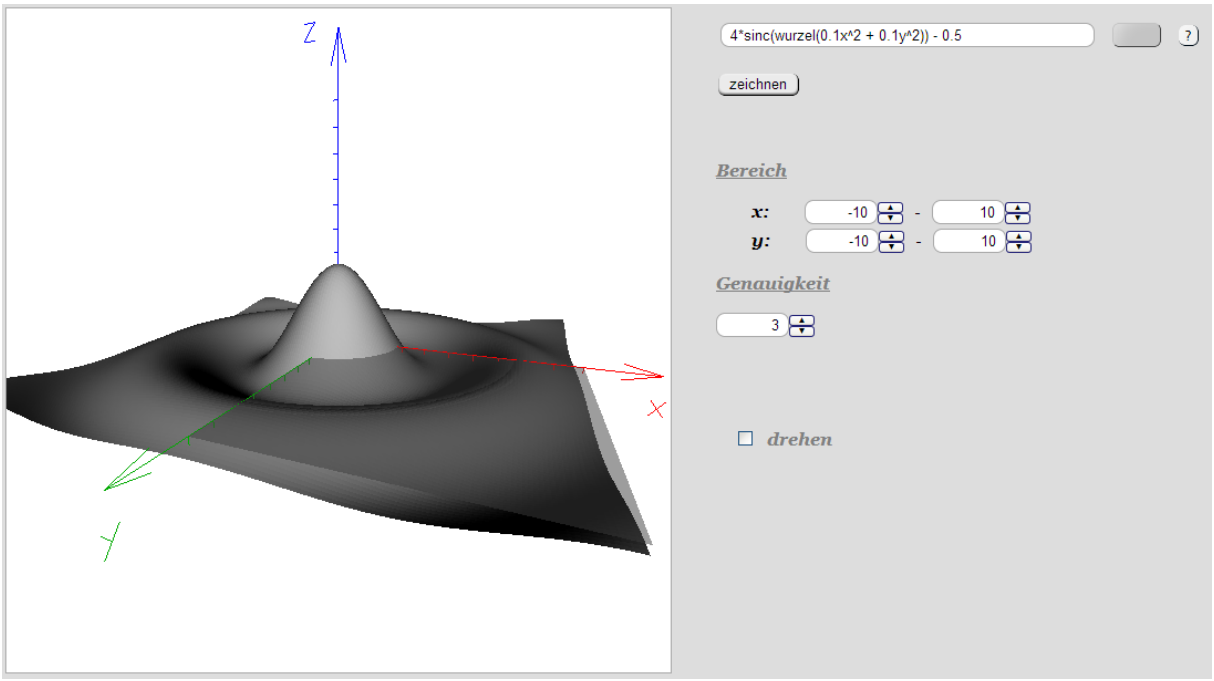


Abb. 29: Oberflächendiagramm zur Laufzeit

Hier zeigt sich der Nachteil von JavaScript. Das Erzeugen der Polygone dauert recht lange. Dieser Vorgang läuft auf der CPU ab, jedoch nicht direkt in Maschinensprache sondern über den Interpreter. Nachträgliches Transformieren der Szene (Drehen, Verschieben) findet ohne Verzögerungen statt, weil dieser Vorgang direkt auf der Grafikkarte stattfindet.

Der Geschwindigkeitsunterschied zwischen Web- und Desktopanwendung kann je nach Hardware, Betriebssystem und Browser zwischen dem drei und vierfachen liegen:

```
// JavaScript
var startTime = (new Date()).getTime();
window.setTimeout(function()
{
    for(var counter1 = 0; counter1 != 100000; counter1++)
    {
        for(var counter2 = 0; counter2 != 100000; counter2++) { }
    }
    var stopTime = (new Date()).getTime();

    console.log(stopTime - startTime); // 16133 ms
}, 0);
```

```
// C#
Stopwatch watch = Stopwatch.StartNew();
for (int counter1 = 0; counter1 != 100000; counter1++)
{
    for (int counter2 = 0; counter2 != 100000; counter2++) { }
}
Console.WriteLine(watch.ElapsedMilliseconds); // 4845 ms
```

4. Resümee

In dieser Arbeit ging es ausschließlich um die Möglichkeiten der Web-Programmierung bei derzeitigem Stand der Technik. Im Vergleich zur klassischen Anwendungsprogrammierung existieren noch immer gravierende Unterschiede.

Normalen Anwendungen ändert regelmäßig ihre Umgebung in denen sie laufen. Allein das Betriebssystem wird regelmäßig erneuert. Das ist nicht der einzige Grund für so viele unterschiedliche Programmiersprachen, Bibliotheken, Schnittstellen, Versionsnummern und Standards. Immer kleinere Strukturen machen alte Prozessoren zum Stromfresser, größere Monitore erfordern neu Kompressionen und Protokolle, Hard- und Software sind nicht mehr mit der alten Generation kompatibel. Es vergeht kein Jahr in dem ein Smartphone veraltet ist, weil der Hersteller keine Updates mehr anbietet. Und das obwohl die Hardware noch völlig ist.

Während in vielen Bereichen der Informatik so ein Kommen und Gehen an der Tagesordnung ist, hat das Web seit seiner Entstehung seine ursprüngliche Form beibehalten. Alles baut auf denselben Protokollen und Sprachen auf wie bei den ersten Browser im Jahr 1990.

Zunächst ging es nur darum statische Inhalte weltweit zu verteilen. Diese Grundlage wurde Schritt für Schritt um neue Funktionen erweitert. Heutige Webanwendungen bieten dynamisch generiertem Inhalt, Ende-Zu-Ende Verschlüsselung bis hin zu Peer-To-Peer Lösungen. Bei jeder dieser Anwendung wird zunächst derselbe Vorgang angestoßen wie vor 20 Jahren, was sie von klassischen Programmen unterscheiden.

Das bringt auch Nachteile mit sich. Durch die weltweite Verbreitung bei gleichzeitiger Gewährleistung der Kompatibilität zu jedem Gerät, entwickeln sich Web-Anwendungen nur schleppend, laufen langsamer und besitzen starke Einschränkungen sowie Mängel an der Sicherheit. Der Entwicklungsaufwand ist höher im Vergleich zur Plattformgebundenen Anwendung. Bei dem Weg von einem HTML Element, über JavaScript bis hin zu einem PHP oder Java Webdienst und wieder zurück hängen viele unterschiedliche sowie zeitliche Bedingungen ab damit die Ausführung konsistent bleibt. Werden neue Techniken eingeführt ist das keine Garantie dafür, dass sie im jeden Browser einwandfrei funktionieren. Dennoch vertrete ich die Meinung, dass in Zukunft eine URL ausreicht, um jede Form von Anwendung zu nutzen. Closed-Source und Software-Abhängigkeiten halte ich für überholt. Der Hauptvorteil von Webanwendungen ist ihre Plattformunabhängigkeit und die weltweite Verfügbarkeit.

Mit der Einführung von HTML5 wird deutlich, dass bereits jetzt auf Plattformgebundene Audio, Video und Netzwerkprogrammierung verzichtet werden kann. Portierungen von grafiklastigen Computerspielen existieren bereits, auch wenn diese sich im Blick auf die Spieleindustrie noch nicht etabliert haben.

Die hier umgesetzte Bibliothek hat gezeigt, wie mit neuer Technik dynamisch generierte Grafiken erzeugt werden kann, auf denen der Nutzer zur Laufzeit Einfluss nimmt.
Andere Aspekte wie Netzwerkkommunikation, Audio- oder Videobearbeitung wurden nicht umgesetzt. Welche Anwendungsmöglichkeiten HTML5 noch bietet bleibt offen.

Ob jegliche Art von Computerprogrammen von Webanwendungen verdrängen werden, es weiterhin zwei Welten gibt oder der Marktanteil von Plattformgebundenen Anwendungen wächst, wird die Zukunft zeigen.

5. Dokumentation

Die erstellte Bibliothek richtet sich an Webseitenentwicklern. Sie dient dazu möglichst komfortabel ein Diagramm einzubetten. Es folgt eine Übersicht über die Attribut-Parameter der einzelnen Diagrammtypen. Auf die Angabe `heigth` und `width` wurde verzichtet.

Liniendiagramm*
Wird über das Attribut `linechart` eingeleitet.

```
<svg linechart="x, green; x*x blue; surface -10 10 red" grid area="#area">
```

Einzelne Graphen werden über das Semikolon getrennt. Soll die Farbe des Graphs nicht schwarz sein, kann nach der Graphen-Gleichung nach setzten eines Kommas ein Farbwert bestimmt werden. Wird statt der Graphen-Gleichung die Zeichenkette `surface` angegeben, so wird eine Fläche gezeichnet, die durch die ersten zwei Graphen eingeschlossen ist. Ist nur ein Graph vorhanden wird y=0 als zweiten Graph verwendet. Nach `surface` sind mit Leerzeichen die Parameter anzugeben. Die ersten zwei Zahlen geben das Intervall für die Fläche an. Der letzte Wert bestimmt die Farbe, diese Angabe ist optional. Über das Attribut `area` wird der geschätzte sichtbare Flächeninhalt ausgegeben. Angegeben werden ein oder mehrere CSS-Selektoren. Handelt es sich um ein <input>-Feld wird das Ergebnis in `value` geschrieben, ansonsten wird `.innerHTML` genutzt.

Die implizite Darstellung wird durch das Setzen des Attributs `implicit` erwirkt.

Balkendiagramm
Wird über das Attribut `barchart` eingeleitet.

```
<svg barchart="'klasse1' 1 red, 'klasse2' 3 blue">
```

Die einzelnen Merkmale werden durch ein Komma getrennt. Die minimale Angabe ist ein Zahlenwert, der das stetige Zielmerkmal angibt. Die Merkmalsausprägung wird durch eine Zeichenkette in Hochkommas angegeben. Hochkommas innerhalb dieser Zeichenkette sind doppelt anzugeben. Für des Zielmerkmal kann eine Farbe angegeben werden. Für ein Farbüberlauf eine zweite. Die Reihenfolge von Zielmerkmal, Merkmalsausprägung und Farbe(n) spielt keine Rolle.

In der Standardeinstellung wird die des Zielmerkmal bei der Darstellung normiert. Alle zusammen nehmen den vollen Platz ein. Das Attribut `noscaling` schaltet dieses Verhalten aus. Es wird ein Zahlenwert übergeben. Über das Mausrad kann dieser Wert verändert werden.

Das Attribut `movable` ermöglicht es den Nutzer das Diagramm zu verschieben.

Kreisdiagramm
Wird über das Attribut `piechart` eingeleitet.

```
<svg piechart="'klasse1' 2 red, 'klasse2' 1 yellow, 'klasse3' 4 gray">
```

Die Formatierung der Merkmale entspricht dem des Balkendiagramms. Wird bei einem Merkmal eine zweite Farbe angeben, wird sie in der 3D Darstellung für den Randbereich genutzt.

Weitere Attribute sind:

`incircle`	Erzeugt einen Innenkreis. Gültig ist ein Wert zwischen 0 und 1.
`rotate`	Darf der Nutzer das Diagramm per Maus drehen?
`limit`	Soll der volle Bereich genutzt werden? 0.5 entspricht ein Halbkreis, 1 ein Vollkreis.
`isometry`	Ist dieses Attribut gesetzt, wird in die 3D-Darstellung gewechselt. Erfordert die Angabe der Tiefe des Kreisdiagramms in Pixel.

Streudiagramm*
Wird über das Attribut `pointcloud` eingeleitet.

```
<svg pointcloud="3 1, 1 2, -1 0" line grid slope=".slope">
```

Einzelne Punkte werden durch Komma getrennt. Bei jedem Punkt wird zuerst die x-Koordinaten und dann die y-Koordinate angeben. Optional kann noch die Farbe für den Punkt vergeben werden.

Das Attribut `line` erzeugt eine Linie für die Korrelation. Die Ausgabe des Anstiegs dieser Linie kann über den CSS-Selektor erfolgen, der bei dem Attribut `slope` angegeben ist.

Histogramm*
Es handelt sich um ein Streudiagramm das zusätzlich das Attribut `histogram` besitzt. Dort werden die Klassenbreiten definiert. Es beinhaltet eine Auflistung von x-Werten. Diese werden sortiert um daraus die Intervalle zu bilden.

```
<svg pointcloud histogram="80 95 100 105 120, #FFDBB7 rgba(235,199,163,1)">
```

Nach dieser Auflistung kann durch ein Komma die Farbe der Rechtecke angegeben werden. Eine zweite Farbangabe sorgt für einen Farbverlauf. Über das Attribut `hidepoints` werde die einzelnen Punkte vom Streudiagramm ausgeblendet.

*Das Liniendiagramm, Streudiagramm und Histogramm besitzen gemeinsame Parameter:
Das Attribut `grid` blendet ein Gitternetz ein. Über `zoom` wird die Vergrößerung angegeben, ein Zahlenwert der sich in der Nähe von 0 befindet. `viewpoint` setzt den Betrachtungspunkt, `viewpoint="0 0"` wäre der Koordinatenursprung. Mit Hilfe von `scalex` oder `scaley` kann die Darstellung gestreckt oder gestaucht werden.
Das Attribut `click` gibt eine JavaScript Funktion an, die beim Klicken auf die Zeichenfläche aufgerufen wird, übergeben werden zwei Parameter für x und y. Das Attribut `hover` verhält sich so ähnlich. Die angegebene Funktion wird immer aufgerufen sobald die Maus über die Zeichenfläche bewegt wird. `out-x` und `out-y` geben die Koordinaten der Mausbewegung aus. Hier werden ebenfalls CSS-Selektoren angegeben. Die Genauigkeit (Anzahl der Stellen) von `hover`, `out-x` und `out-y` wird über das Attribut `precision` bestimmt.

Colorpicker
Es handelt sich um ein <input>-Feld bei dem das Attribut `colorpicker` gesetzt ist. Der Colorpicker erscheint sobald der Nutzer auf den Button klickt.

```
<input colorpicker value="#DDD">
```

Es erzeugt nach der Einbindung die Attribute `red`, `green`, `blue` und `alpha`.

Numeric-Up-Down

Im neuen Standard kann ein <input>-Feld den Typ „`number`“ erhalten. Man erhält ein Eingabefeld das Zahlen entgegen nimmt, statt einem Text. Derzeit wird dies nur vom Browser Chrome unterstützt. In anderen Browsern kann dieses Feld nachgebaut werden.

Es reicht also aus das Attribut `type` auf `"number"` zu setzen:

```
<input type="number" min="-6" max="10" step="2">
```

Das Attribut `min` und `max` gibt die Unter- und Obergrenze des Zahlenbereichs an. Die Schrittweite wird mit `step` festgelegt. Diese Attribute sind im Standard vorgeschrieben.

Wird das Attribut `permanent` gesetzt, so unterscheidet sich die Darstellung in allen Browsern nicht.

Eingabefeld für einen arithmetischen Ausdruck
Wird im <input>-Feld über das Attribut `term` eingeleitet.

```
<input term="x y">
```

Das Attribut enthält eine Auflistung von Variablen, die im Ausdruck vorkommen dürfen. Treten bei der Eingabe Fehler auf, erscheint ein Hinweistext. Die Eingabe bleibt unverändert.

Eingabe Hilfe für einen arithmetische Ausdrücke
Wird im <input>-Feld über das Attribut `termhelper` eingeleitet.

```
<input termhelper>
```

Klickt der Nutzer auf diesen Button erscheint ein Fenster mit allen gültigen Funktionsamen und Konstanten die der integrierte Parser versteht.

Der Button wird die CSS-Klasse „helpbutton“ zugewiesen und das Popup-Fenster „help“.

6. Quellen

(1) Alexander Wiechert, "Client-Server Prinzip", MediaWiki, abgerufen am 10.9.2013
http://www.fachadmin.de/index.php/Client-Server_Prinzip

(2) "HTML-Renderer", Wikimedia Foundation Inc., abgerufen am 10.9.2013
http://de.wikipedia.org/wiki/HTML-Rendering

(3) "Cloud-Computing", Wikimedia Foundation Inc., abgerufen am 10.9.2013
http://de.wikipedia.org/wiki/Cloud-Computing

(4) "Virtuelle Server", Wikimedia Foundation Inc., abgerufen am 10.9.2013
http://de.wikipedia.org/wiki/Server#Virtuelle_Server

(5) Prof. Dr. Tobias Kollmann, "Browser", Gabler Wirtschaftslexikon,
abgerufen am 10.9.2013
http://wirtschaftslexikon.gabler.de/Definition/browser.html

(6) "Peer-to-Peer-Netz", DATACOM Buchverlag GmbH, abgerufen am 10.9.2013
http://www.itwissen.info/definition/lexikon/Peer-to-Peer-Netz-P2P-peer-to-peer-network.html

(7) "BitTorrent", Wikimedia Foundation Inc., abgerufen am 10.9.2013
http://de.wikipedia.org/wiki/BitTorrent

(8) "Bitcoin: Peer-to-Peer-Netzwerk", Wikimedia Foundation Inc.,
abgerufen am 10.9.2013
http://de.wikipedia.org/wiki/Bitcoin#Peer-to-Peer-Netzwerk

(9) "Was ist Bitcoin - So funktioniert die digitale Währung", Softonic International S.L.,
abgerufen am 10.9.2013
http://artikel.softonic.de/was-ist-bitcoin-so-funktioniert-die-digitale-wahrung

(10) "Bitcoins, digitale Goldgrube oder Fluch?", Euronews, abgerufen am 10.9.2013
http://de.euronews.com/2013/05/28/bitcoins-digitale-goldgrube-oder-fluch/

(11) Marco Holmer, "Flucht in den Bitcoin", go-bitcoin.com, abgerufen am 10.9.2013
http://go-bitcoin.com/2013-03-26-flucht-den-bitcoin

(12) "WikiLeaks nutzt Bitcoin-Projekt für Spenden", W. Girardet GmbH & Co. KG,
abgerufen am 10.9.2013
http://www.wz-newsline.de/home/multimedia/wikileaks-nutzt-bitcoin-projekt-fuer-spenden-1.686185

(13) "Button-Lösung", Wikimedia Foundation Inc., abgerufen am 10.9.2013
http://de.wikipedia.org/wiki/Button-L%C3%B6sung

(14) "So arbeitet Onion Routing", CHIP Communications, abgerufen am 10.9.2013
http://www.chip.de/artikel/Anonym-Surfen-Die-besten-Anonymisierer-im-Test-3_38042019.html

(15) "AES-Verschlüsselung", DATACOM Buchverlag GmbH, abgerufen am 10.9.2013
http://www.itwissen.info/definition/lexikon/advanced-encryption-standard-AES-AES-Verschluesselung.html

(16) "Wie wird bei RShare Anonymität hergestellt?", MediaWiki, abgerufen am 10.9.2013
http://www.planetpeer.de/wiki/index.php/RShare_Dokumentation#Wie_wird_bei_RShare_Anonymit.C3.A4t_hergestellt.3F

(17) "I2P: Unterschied zu Tor", Wikimedia Foundation Inc., abgerufen am 10.9.2013
http://de.wikipedia.org/wiki/I2P#Unterschied_zu_Tor

(18) Sam Dutton, "Getting Started with WebRTC", HTML5 Rocks - Google, abgerufen am 10.9.2013
http://www.html5rocks.com/en/tutorials/webrtc/basics/

(19) "Explainer: what is geoblocking?", The Conversation Trust (UK), abgerufen am 10.9.2013
https://theconversation.com/explainer-what-is-geoblocking-13057

(20) Rainer Ihde und Dr. Marcus Dittmann, "Cookies im Internet – ein problematisches Marketing-Instrument", IHDE & PARTNER, abgerufen am 10.9.2013
http://www.onlinelaw.de/de/publikationen/artikel/artikel.php?we_objectID=50&level1=2&level2=

(21) Wolfgang Schwarz, "Cookies", javascript-workshop, abgerufen am 10.9.2013
http://javascript-workshop.de/buch/08.html

(21) "Versteckte Elemente", SELFHTML, abgerufen am 10.9.2013
http://de.selfhtml.org/html/formulare/versteckte.htm

(22) "Hypertext Markup Language", Wikimedia Foundation Inc., abgerufen am 10.9.2013
http://de.wikipedia.org/wiki/Hypertext_Markup_Language

(23) Patrick Schnabel, "DNS - Domain Name System", Elektronik-Kompendium, abgerufen am 10.9.2013
http://www.elektronik-kompendium.de/sites/net/0901141.htm

(24) "Transmission Control Protocol", Wikimedia Foundation Inc. , abgerufen am 10.9.2013
http://de.wikipedia.org/wiki/Transmission_Control_Protocol

(25) "MIME-Typen", SELFHTML, abgerufen am 10.9.2013
http://de.selfhtml.org/diverses/mimetypen.htm

(26) "Das HTTP Protokoll", Kioskea.net , abgerufen am 10.9.2013
http://de.kioskea.net/contents/44-das-http-protokoll

(27) Sam Dutton, "Entstehung des World Wide Web", SELFHTML, abgerufen am 10.9.2013
http://de.selfhtml.org/intro/internet/www.htm

(28) "IPv6", Wikimedia Foundation Inc., abgerufen am 10.9.2013
http://de.wikipedia.org/wiki/IPv6

(29) "Textauszeichnung", SELFHTML, abgerufen am 10.9.2013
http://de.selfhtml.org/html/allgemein/textauszeichnung.htm

(30) "Zeilenumbruch", SELFHTML, abgerufen am 10.9.2013
http://de.selfhtml.org/html/text/zeilenumbruch.htm

(31) "Allgemeine Elemente für Textbereiche", SELFHTML, abgerufen am 10.9.2013
http://de.selfhtml.org/html/text/bereiche.htm

(32) Sam Dutton, "HTML5 Canvas", W3Schools, abgerufen am 10.9.2013
http://www.w3schools.com/html/html5_canvas.asp

(33) "Über das World Wide Web Consortium", World Wide Web Consortium, abgerufen am 10.9.2013
http://www.w3c.at/about/overview.html

(34) "World Wide Web Consortium", Wikimedia Foundation Inc., abgerufen am 10.9.2013
http://de.wikipedia.org/wiki/World_Wide_Web_Consortium

(35) "World Wide Web Consortium: Specification Maturation", Wikimedia Foundation Inc., abgerufen am 10.9.2013
http://en.wikipedia.org/wiki/World_Wide_Web_Consortium#Specification_Maturation

(36) "Fertigstellung von HTML5", Wikimedia Foundation Inc., abgerufen am 10.9.2013
http://de.wikipedia.org/wiki/HTML5#Fertigstellung_von_HTML5

(37) "Browserkrieg", Wikimedia Foundation Inc., abgerufen am 10.9.2013
http://de.wikipedia.org/wiki/Browserkrieg

(38) "Mouse wheel programming in JavaScript", adomas.org, abgerufen am 10.9.2013
http://www.adomas.org/javascript-mouse-wheel/

(39) "HTML5 Input Types", W3Schools, abgerufen am 10.9.2013
http://www.w3schools.com/html/html5_form_input_types.asp

(40) Axel Pratzner, "CSS3 box-shadow: Schatten einfach gemacht", html-seminar.de, abgerufen am 10.9.2013
http://www.html-seminar.de/css3-box-shadow.htm

(41) "Browsertests: Probleme der unterschiedlichen Darstellung, SUMO GmbH, abgerufen am 10.9.2013
http://www.webmasterarchiv.com/browsertest-problem-unterschiedliche-darstellung/

(42) Jennifer Kyrnin, "CSS Vendor Prefixes", About.com Guide, abgerufen am 10.9.2013
http://webdesign.about.com/od/css/a/css-vendor-prefixes.htm

(43) "Cascading Style Sheets", Wikimedia Foundation Inc., abgerufen am 10.9.2013
http://de.wikipedia.org/wiki/Cascading_Style_Sheets

(44) "Meta-Angaben zum Inhalt", SELFHTML, abgerufen am 10.9.2013
http://de.selfhtml.org/html/kopfdaten/meta.htm

(45) "Tag (Informatik)", Wikimedia Foundation Inc., abgerufen am 10.9.2013
http://de.wikipedia.org/wiki/Tag_(Informatik)

(46) "Skype-Alternative Tox: einfach, verschlüsselt, dezentral", Heise Zeitschriften Verlag, abgerufen am 10.9.2013
http://www.heise.de/newsticker/meldung/Skype-Alternative-Tox-einfach-verschluesselt-dezentral-1926996.html

(47) "IETF 87: Wann ist ein Standard ein Standard?", Heise Zeitschriften Verlag, abgerufen am 10.9.2013
http://www.heise.de/newsticker/meldung/IETF-87-Wann-ist-ein-Standard-ein-Standard-1926705.html

(48) Katharina Eckhardt, "JavaScript - Entwicklung", scriptNews.de, abgerufen am 10.9.2013
http://www.scriptnews.de/js-entwicklung.htm

(49) "XMLHttpRequest", Mozilla Foundation, abgerufen am 10.9.2013
https://developer.mozilla.org/de/docs/DOM/XMLHttpRequest

(50) "Webanwendung", Wikimedia Foundation Inc., abgerufen am 6.8.2012
https://de.wikipedia.org/wiki/Webanwendung

(51) "Web 2.0: Marketing-Gag oder Internet-Paradigma",
Work With Us Medienagentur UG, abgerufen am 6.8.2012
http://work-with-us.de/index.php/blog/web-2-0-lexikon/33-web-20-marketing-gag-oder-internet-paradigma

(52) Prof. Dr. Richard Lackes, "Web 2.0", Gabler Wirtschaftslexikon,
abgerufen am 6.8.2012
http://wirtschaftslexikon.gabler.de/Definition/web-2-0.html

(53) "Die wichtigsten Programmiersprachen unserer Zeit", commag.org,
abgerufen am 6.8.2012
http://www.commag.org/die-wichtigsten-programmiersprachen-unserer-zeit/

(54) Tobias Schlitt, "Typsichere Programmierung in PHP",
Software & Support Media GmbH, abgerufen am 6.8.2012
http://phpmagazin.de/TypsichereProgrammierunginPHP-162936

(55) "V8 (JavaScript-Implementierung)", Wikimedia Foundation Inc.,
abgerufen am 6.8.2012
http://de.wikipedia.org/wiki/V8_(JavaScript-Implementierung)

(56) "Baseline wird zusätzlicher JavaScript-Compiler in Firefox",
Heise Zeitschriften Verlag, abgerufen am 6.8.2012
http://www.heise.de/developer/meldung/Baseline-wird-zusaetzlicher-JavaScript-Compiler-in-Firefox-1836857.html

(60) Jens Ihlenfeld, "Google stellt moderne Javascript-Alternative vor",
Computec Media AG, abgerufen am 6.8.2012
http://www.golem.de/1110/86926.html

(66) "Visual Basic Script", Heise Zeitschriften Verlag, abgerufen am 6.8.2012
http://www.heise.de/security/dienste/Visual-Basic-Script-403232.html

(67) "Mehr als ein Hype - Web 2.0 im Praxiseinsatz",
Heise Zeitschriften Verlag, abgerufen am 6.8.2012
http://www.heise.de/ct/artikel/Mehr-als-ein-Hype-290544.html

(68) "Browser-Plug-ins", Wikimedia Foundation Inc., abgerufen am 6.8.2012
http://de.wikipedia.org/wiki/Plug-in#Browser-Plug-ins

(69) "Objekte einbinden", SELFHTML, abgerufen am 6.8.2012
http://de.selfhtml.org/html/multimedia/objekte.htm

(70) U.Häßler, "CSS, HTML und Javascript mit {stil} - iframe versus object", mediaevent,
abgerufen am 6.8.2012
http://www.mediaevent.de/2008/12/html-object-tag/

(71) "Java-Applets einbinden", SELFHTML, abgerufen am 6.8.2012
http://de.selfhtml.org/html/multimedia/java_applets.htm

(72) "Warnung vor kritischer Java-Lücke", Heise Zeitschriften Verlag,
abgerufen am 6.8.2012
http://www.heise.de/security/meldung/Warnung-vor-kritischer-Java-Luecke-1675454.html

(73) "Kritische Schwachstelle in aktueller Java-Laufzeitumgebung", Bundesamt für Sicherheit in der Informationstechnik, abgerufen am 6.8.2012 https://www.bsi.bund.de/DE/Presse/Pressemitteilungen/Presse2013/Krit_Schwachstelle_Java-7-10_11012013.html

(74) "Wieder kritische Lücke in Java entdeckt", derStandard.at, abgerufen am 6.8.2012 http://derstandard.at/1373513176250/Wieder-kritische-Luecke-in-Java-entdeckt

(75) "Was ist der BKA-Trojaner?", ROCKIT-INTERNET GmbH, abgerufen am 6.8.2012 http://www.dworld.de/malware/ein-gratis-tool-gegen-den-bka-trojaner

(76) Thorsten Eggeling, "HTML5 vs. Flash", IDG Tech Media GmbH, abgerufen am 6.8.2012 http://www.pcwelt.de/ratgeber/Entwicklungsumgebungen-fuer-HTML5-Flash-4209174.html

(77) http://www.medien.ifi.lmu.de/lehre/ws1011/dm/dm10.pdf Prof. Hußmann, "Interaktive Web-Inhalte", Ludwig-Maximilians-Universität München, abgerufen am 6.8.2012

(78) Matthias Reuter, "Mit jQuery alten Browsern HTML5 beibringen", magjs.de, abgerufen am 6.8.2012 http://www.magjs.de/2012-01/reuter/reuter.html

(79) "Query", Computec Media AG, abgerufen am 6.8.2012 http://www.golem.de/specials/jquery/

(80) Herbert Braun, "Generationswechsel.js - Neue JavaScript-Frameworks für ambitionierte Webanwendungen", c't Magazin für Computer Technik, Nummer: 15, Seiten: 174, Heise Zeitschriften Verlag GmbH & Co. KG, Jahr: 2013, ISSN: 0735-8679 https://www.heise.de/artikel-archiv/ct/2013/15/174_Generationswechsel-js

(81) Jörg Dennis Krüger, "Trennung von Content, Layout und Funktion", B-Eye-Media GmbH, abgerufen am 6.8.2012 http://www.jdk.de/de/cms/wcm-cms-web-content-management/was-ist-wcm-cms/trennung-content-layout-funkt.html

(82) Janina Brandes, "8 Fragen zu flüchtigen Vervielfältigungen im Netz", Telemedicus e.V., abgerufen am 6.8.2012 http://www.telemedicus.info/article/2035-8-Fragen-zu-fluechtigen-Vervielfaeltigungen-im-Netz.html

(83) Marc Stenzel, "Kritische Sicherheitslücken im TYPO3 Core und FLOW3", CMS & Content-Migration, abgerufen am 6.8.2012 http://www.cms-content-migration.de/2012/03/kritische-sicherheitsluecken-im-typo3-core-und-flow3/

(84) "Sicherheitslücken in WordPress-Plug-ins betreffen Millionen Websites", ZDNet, abgerufen am 6.8.2012 http://www.zdnet.de/88158987/neue-sicherheitslucken-in-wordpress-plug-ins-betreffen-millionen-websites/

(85) "JDownloader: Gericht verbietet Stream-Recorder", Heise Zeitschriften Verlag, abgerufen am 6.8.2012 http://www.heise.de/newsticker/meldung/JDownloader-Gericht-verbietet-Stream-Recorder-1892674.html

(86) "DRM - Digital Rights Management", Computec Media AG, abgerufen am 6.8.2012 http://www.golem.de/specials/drm/

(87) "Streit um digitales Rechtekontrollmanagement in HTML5", ComputerBase GmbH, abgerufen am 6.8.2012 http://www.computerbase.de/news/2013-03/streit-um-digitales-rechtekontrollmanagement-in-html5/

(88) "Performing DDoS attacks with HTML5 Cross Origin Requests & WebWorkers", Attack and Defense Labs, abgerufen am 6.8.2012 http://blog.andlabs.org/2010/12/performing-ddos-attacks-with-html5.html

(89) Alexander Kirk, "DDoS-Attacke", computerlexikon.com, abgerufen am 6.8.2012 http://www.computerlexikon.com/begriff-ddos-attacke

(90) Arpit Bajpai , "Securing Apache, Part 8: DoS & DDoS Attacks", Linux For You, abgerufen am 6.8.2012 http://www.linuxforu.com/2011/04/securing-apache-part-8-dos-ddos-attacks/

(91) "CSS Display and Visibility", W3Schools, abgerufen am 6.8.2012 http://www.w3schools.com/css/css_display_visibility.asp

(92) "URL-Spoofing", Wikimedia Foundation Inc., abgerufen am 6.8.2012 http://de.wikipedia.org/wiki/URL-Spoofing

(93) "SYN Flood Attack Protection Technology White Paper", H3C Technologies Co., abgerufen am 6.8.2012 http://www.h3c.com/portal/Products___Solutions/Technology/Security_and_VPN/Technology_White_Paper/200812/624110_57_0.htm

(94) "CROSS-SITE-SCRIPTING", GlobalEvolutionSecurity, abgerufen am 6.8.2012 http://www.intelligentexploit.com/articles/[german]-Cross-Site-Scripting.pdf

(95) "Cross-Site Scripting (XSS) unterbinden", PHP-Kurs.com, abgerufen am 6.8.2012 http://www.php-kurs.com/cross-site-scripting-xss-unterbinden.htm

(96) David Müller , "Angriffe auf Webanwendungen", d-mueller.de, abgerufen am 6.8.2012 http://www.d-mueller.de/blog/angriffe-auf-webanwendungen-teil-1-xss-beispielangriff/

(97) Christian Wenz , "JavaScript und Sicherheit", Galileo Computing, ISBN: 3-89842-859-1, abgerufen am 6.8.2012 http://openbook.galileocomputing.de/javascript_ajax/30_sicherheit_001.htm

(98) "BaBaBanküberfall: Ungenügende Sicherheit bei Banken-Websites", Heise Zeitschriften Verlag, abgerufen am 6.8.2012 http://www.heise.de/ct/artikel/BaBaBanku-berfall-1102741.html

(99) "Passwortklau für Dummies", Heise Zeitschriften Verlag, abgerufen am 6.8.2012 http://www.heise.de/security/artikel/Passwortklau-fuer-Dummies-270910.html

(100) "Content Management System (CMS)", Bundesamt für Sicherheit in der Informationstechnik, abgerufen am 6.8.2012 https://www.bsi.bund.de/DE/Publikationen/Studien/CMS/Studie_CMS.html

(101) "Blogger demonstrieren gewieften Passwortklau", Heise Zeitschriften Verlag, abgerufen am 6.8.2012 http://www.heise.de/security/meldung/Blogger-demonstrieren-gewieften-Passwortklau-1761237.html

(102) "Online Banking fatal", Heise Zeitschriften Verlag, abgerufen am 6.8.2012 http://www.heise.de/security/artikel/Vom-ausgezeichneten-Online-Banking-zum-Security-Desaster-270914.html

(103) "JavaScript: Sicherheit", molily.de, abgerufen am 6.8.2012 http://molily.de/js/sicherheit.html

(104) "HTML5", World Wide Web Consortium, abgerufen am 18.8.2013
http://www.w3.org/TR/html5/

(105) "Web Hypertext Application Technology Working Group",
Wikimedia Foundation Inc., abgerufen am 18.8.2013
http://de.wikipedia.org/wiki/Web_Hypertext_Application_Technology_Working_Group

(106) "HTML5: WHATWG und W3C gehen unterschiedliche Wege",
Heise Zeitschriften Verlag, abgerufen am 18.8.2013
http://www.heise.de/developer/meldung/HTML5-WHATWG-und-W3C-gehen-unterschiedliche-Wege-1650028.html

(107) "W3C Confirms May 2011 for HTML5 Last Call, Targets 2014 for HTML5
Standard", World Wide Web Consortium, abgerufen am 18.8.2013
http://www.w3.org/2011/02/htmlwg-pr.html

(108) "HTML5: So nutzt man die verschiedenen Sektionselemente richtig", ZDNet,
abgerufen am 18.8.2013
http://www.zdnet.de/41558246/html5-so-nutzt-man-die-verschiedenen-sektionselemente-richtig/

(109) "HTML: Living Standard — Last Updated 16 September 2013",
Web Hypertext Application Technology Working Group, abgerufen am 18.8.2013
http://www.whatwg.org/specs/web-apps/current-work/multipage/

(110) "HTML5 Browser Support", Hurm IT-Services, abgerufen am 18.8.2013
http://hurm-it.de/blog/html5/html5-browser-support/#.UhDPrdJhhuA

(111) Christian Hinzmann, "Semantische Tags", BlauWeb.DE Internet-Solutions,
abgerufen am 18.8.2013
http://www.blauweb.de/BLOG/Webdesign/Semantische-Tags.html

(112) "HTML5 - Web Development to the next level", HTML5 Rocks - Google,
abgerufen am 18.8.2013
http://slides.html5rocks.com/#title-slide

(114) C. Elzer, S. Griesbach und M. Rinne,
"Flash stirbt: HTML 5 verdrängt Adobe's Plug-ins", CHIP Xonio Online GmbH 2013,
abgerufen am 18.8.2013
http://www.chip.de/artikel/HTML5-Das-Web-von-morgen_41539437.html

(115) "HTML <input> required Attribute", HTML5 Rocks - Google,
abgerufen am 18.8.2013
http://www.w3schools.com/tags/att_input_required.asp

(116) "HTML5: Neue Formulartypen in der Praxis", yeebase media GmbH,
abgerufen am 18.8.2013
http://t3n.de/news/html5-neue-formulartypen-praxis-346757/

(117) "HTML5 Video", HTML5 Rocks - Google, abgerufen am 18.8.2013
http://www.w3schools.com/html/html5_video.asp

(118) "Beispiele für die Verwendung von SVG und Canvas", Microsoft Corporation,
abgerufen am 18.8.2013
http://msdn.microsoft.com/de-de/library/ie/gg589488(v=vs.85).aspx

(119) Robert Gravelle, "HTML5 Canvas vs. SVG: Choose the Best Tool for the Job",
QuinStreet Inc., abgerufen am 18.8.2013
http://www.htmlgoodies.com/html5/other/html5-canvas-vs.-svg-choose-the-best-tool-for-the-job.html#fbid=R6-nHyMGaTz

(120) "HTML5 - MathML Tutorial", tutorialspoint, abgerufen am 18.8.2013
http://www.tutorialspoint.com/html5/html5_mathml.htm

(121) "hyphens", Mozilla Foundation, abgerufen am 24.9.2013
https://developer.mozilla.org/de/docs/CSS/hyphens

(122) Denis Potschien,
"Performantere JavaScript-Animationen mit requestAnimationFrame",
drweb.de GmbH, abgerufen am 18.8.2013
http://www.drweb.de/magazin/requestanimationframe-performante-javascript-animationen-ohne-settimeout-und-setinterval-37205/

(123) "Document.querySelectorAll", Mozilla Foundation, abgerufen am 18.8.2013
https://developer.mozilla.org/en-US/docs/Web/API/Document.querySelectorAll

(124) "Web Worker-Grundlagen", HTML5 Rocks - Google, abgerufen am 18.8.2013
http://www.html5rocks.com/de/tutorials/workers/basics/

(125) "Thread (Informatik)", Wikimedia Foundation Inc., abgerufen am 18.8.2013
http://de.wikipedia.org/wiki/Thread_(Informatik)

(126) Mike Houston, "General Purpose Computation on Graphics Processors (GPGPU)",
Stanford University, abgerufen am 18.8.2013
http://graphics.stanford.edu/~mhouston/public_talks/R520-mhouston.pdf

(127) "Getting started with WebGL", Mozilla Foundation, abgerufen am 18.8.2013
https://developer.mozilla.org/en-US/docs/Web/WebGL/Getting_started_with_WebGL

(128) "OpenGL ES 2.0 for the Web", Khronos Group, abgerufen am 18.8.2013
http://www.khronos.org/webgl/

(129) "Open Graphics Library for Embedded Systems", Wikimedia Foundation Inc.,
abgerufen am 18.8.2013
http://de.wikipedia.org/wiki/Open_Graphics_Library_for_Embedded_Systems

(130) Fedy Abi-Chahla, "DirectX 11 und OpenGL 3: Der 3D-API-Krieg ist beendet",
TechMedia Network, abgerufen am 18.8.2013
http://www.tomshardware.de/DirectX-OpenGL,testberichte-240159-2.html

(131) Markus Weißmann, "OpenGL vs. Direct3D: Ein 3D-API Vergleich", mweissmann.de,
abgerufen am 18.8.2013
http://www.mweissmann.de/downloads/Direct3D_vs._OpenGL.pdf

(132) "User Contributions", Khronos Group, abgerufen am 18.8.2013
http://www.khronos.org/webgl/wiki/User_Contributions

(133) "HTML5 und Local Storage: Daten für Offlinebetrieb speichern",
yeebase media GmbH, abgerufen am 18.8.2013
http://t3n.de/news/html5-local-storage-daten-offlinebetrieb-speichern-350314/

(134) Eric Bidelman, "Kennenlernen der FileSystem APIs", HTML5 Rocks - Google,
abgerufen am 18.8.2013
http://www.html5rocks.com/de/tutorials/file/filesystem/

(135) "Lesen von Dateien in JavaScript mit den File APIs", yeebase media GmbH,
abgerufen am 18.8.2013
http://www.html5rocks.com/de/tutorials/file/dndfiles/

(136) "Blob", Mozilla Foundation, abgerufen am 18.8.2013
https://developer.mozilla.org/en-US/docs/Web/API/Blob?redirectlocale=en-US&redirectslug=DOM%2FBlob

(137) Eric Bidelman, "Neue Tricks für XMLHttpRequest2", HTML5 Rocks - Google, abgerufen am 18.8.2013
http://www.html5rocks.com/de/tutorials/file/xhr2/

(138) "Ajax (Programmierung)", Wikimedia Foundation Inc., abgerufen am 18.8.2013
http://de.wikipedia.org/wiki/Ajax_(Programmierung)

(139) John McCutchan, "Pointer Lock and First Person Shooter Controls", HTML5 Rocks - Google, abgerufen am 18.8.2013
http://www.html5rocks.com/en/tutorials/pointerlock/intro/

(140) Philipp Wiegel, "Geo Koordinaten mit HTML5 ermitteln", Webentwickler Oase, abgerufen am 18.8.2013
http://www.webentwickler-oase.de/html5/geo-koordinaten-mit-html5-ermitteln/

(141) "Navigator.getUserMedia", Mozilla Foundation, abgerufen am 18.8.2013
https://developer.mozilla.org/en-US/docs/Web/API/Navigator.getUserMedia

(142) Eric Bidelman, "Capturing Audio & Video in HTML5", HTML5 Rocks - Google, abgerufen am 18.8.2013
http://www.html5rocks.com/en/tutorials/getusermedia/intro/

(143) Pete LePage, "This End Up: Using Device Orientation", HTML5 Rocks - Google, abgerufen am 18.8.2013
http://www.html5rocks.com/en/tutorials/device/orientation/

(144) Malte Ubl und Eiji Kitamura, "Einführung zu WebSockets: Sockets im Web", HTML5 Rocks - Google, abgerufen am 18.8.2013
http://www.html5rocks.com/de/tutorials/websockets/basics/

(145) "Socket", DATACOM Buchverlag GmbH, abgerufen am 18.8.2013
http://www.itwissen.info/definition/lexikon/Socket-socket.html

(146) "The ChannelSplitterNode Interface", World Wide Web Consortium, abgerufen am 18.8.2013
https://dvcs.w3.org/hg/audio/raw-file/tip/webaudio/specification.html#example-1

(147) Boris Smus, "Getting Started with Web Audio API", HTML5 Rocks - Google, abgerufen am 18.8.2013
http://www.html5rocks.com/en/tutorials/webaudio/intro/

(148) "Odinmonkey macht Javascript fast so schnell wie nativen Code", Computec Media AG, abgerufen am 18.8.2013
http://www.golem.de/news/asm-js-odinmonkey-macht-javascript-fast-so-schnell-wie-nativen-code-1303-98337.html

(149) "asm.js Working Draft", asmjs.org, abgerufen am 18.8.2013
http://asmjs.org/spec/latest/

(150) "Epic Citadel in HTML5", Computec Media AG, abgerufen am 18.8.2013
http://www.golem.de/news/unreal-engine-3-epic-citadel-in-html5-1305-99073.html

(151) "Diagramm", Wikimedia Foundation Inc., abgerufen am 17.9.2013
http://de.wikipedia.org/wiki/Diagramm

(152) "Explizite und Implizite Darstellung einer Funktion", PHTL Lienz, abgerufen am 17.9.2013
http://www.htl-lienz.tsn.at/htl/zusatz/unterricht_interaktiv/ganf/mathe/explizite_implizite_Funktion.pdf

(153) Veronika Schuster, "Streudiagramme – Korrelation, Regression, Glättung", rosuda.org, abgerufen am 17.9.2013
http://rosuda.org/lehre/WS04/SeminarPFDs/Streu_Glaett.pdf

(154) "Lineare Regression", Wikimedia Foundation Inc., abgerufen am 17.9.2013
http://de.wikipedia.org/wiki/Lineare_Regression

(155) "Statistische Variable", Wikimedia Foundation Inc., abgerufen am 17.9.2013
http://de.wikipedia.org/wiki/Statistische_Variable

(156) "Histogramm", Wikimedia Foundation Inc., abgerufen am 17.9.2013
http://de.wikipedia.org/wiki/Histogramm

(157) "Flächendiagramm", Wikimedia Foundation Inc., abgerufen am 17.9.2013
http://de.wikipedia.org/wiki/Fl%C3%A4chendiagramm

(158) Christian Wenz, "JavaScript", Galileo Computing, abgerufen am 18.9.2013
http://openbook.galileocomputing.de/javascript/

(159) "Object-Literale", SELFHTML, abgerufen am 18.9.2013
http://aktuell.de.selfhtml.org/artikel/javascript/organisation/#object-literale

(160) "document", SELFHTML, abgerufen am 18.9.2013
http://de.selfhtml.org/javascript/objekte/document.htm

(161) "Instanziierung", Wikimedia Foundation Inc., abgerufen am18.9.2013
http://de.wikipedia.org/wiki/Instanziierung

(162) "Objektorientierte Programmierung", Wikimedia Foundation Inc., abgerufen am18.9.2013
http://de.wikipedia.org/wiki/Objektorientierte_Programmierung

(163) "Appendix B: SVG Document Object Model (DOM)", World Wide Web Consortium, abgerufen am 18.9.2013
http://www.w3.org/TR/SVG/svgdom.html

(164) "SVG Gradients - Linear", World Wide Web Consortium, abgerufen am 18.9.2013
http://www.w3schools.com/svg/svg_grad_linear.asp

(165) "IEEE 754", Wikimedia Foundation Inc., abgerufen am 20.9.2013
http://de.wikipedia.org/wiki/IEEE_754

(166) "Objektunabhängige Funktionen - eval()", SELFHTML, abgerufen am 20.9.2013
http://de.selfhtml.org/javascript/objekte/unabhaengig.htm#eval

(167) "JavaScript typed arrays", Mozilla Foundation, abgerufen am 20.9.2013
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Typed_arrays

(168) "Math", SELFHTML, abgerufen am 20.9.2013
http://de.selfhtml.org/javascript/objekte/math.htm

(169) Peter Kröner, "Schönes neues CSS: RGBA und HSL(A)", peterkroener.de, abgerufen am 20.9.2013
http://www.peterkroener.de/schoenes-neues-css-rgba-und-hsla/

(170) Peter Kropff, "OOP mit JavaScript", peterkropff.de, abgerufen am 20.9.2013
http://www.peterkropff.de/site/javascript/oop.htm

(171) "String - match()", SELFHTML, abgerufen am 20.9.2013
http://de.selfhtml.org/javascript/objekte/string.htm#match

(172) "three.js documentation", threejs.org, abgerufen am 20.9.2013
http://threejs.org/docs/

(173) Frank Thürigen, "Das Module Pattern", Team23 GmbH & Co. KG, abgerufen am 28.9.2013
http://www.webmasterpro.de/portal/impressum.html

(174) Helge Toutenburg, Andreas Fieger und Christian Kastner, "Deskriptive Statistik für Betriebs- und Volkswirte", Prentice-Hall-Verlag, ISBN 3-93043-621-3

7. Abbildungsverzeichnis

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich diese Bachelorarbeit selbständig und mit eigenen Worten verfasst habe. Ich habe hierfür nur die in der Arbeit genannten Hilfsmittel und Quellen genutzt.

Martin Wantke