Leseprobe aus

Flash MX professionell

von Carlo Blatz und den Powerflashern

Galileo Design
ca. 600 S., mit CD, 44,90 Euro
ISBN 3-89842-222-4, ab Ende Juli

Galileo Design

Inhalt

Einführung

Was war, ist und sein wird

Und wieder geht es los. Eine neue Flash-Version, neue Ideen, neue Möglichkeiten, neue Workarounds, aber auch neue Ansprüche. Auch an dieses Buch.

Vorwort

Über zwei Jahre ist es inzwischen schon wieder her, dass ich das erste Mal gebeten wurde, ein Buch über meine inzwischen fünf Jahre Flash-Erfahrung zu schreiben. Damals habe ich zusammen mit Gerald und den Flashworkern geschrieben. Mit dem ersten deutschen Flash-Buch für professionelle Anwender haben wir versucht, das Beste des deutschen Flash-Know-hows der letzten Jahre in einem Buch zu bündeln. Dank dem Verlag hatten wir keine Seitenbegrenzung und durften über unsere 600 Seiten hinaus einige Online-Tutorial-Autoren überzeugen, ihre Arbeit weiterzuentwickeln und schließlich mit uns zu veröffentlichen. Vielen Dank noch einmal an alle Beteiligten.

Diesmal machen wir es anders herum und haben uns die Genehmigung geholt, die dynamischsten Kapitel des Buchs als Tutorial zu veröffentlichen. So werden insbesondere die Flash-Detection und die PF ActionScript Conventions bei www.flashworker.de ständig aktuell gehalten.

Nun muss es also weitergehen. Was geschrieben wurde, ist schon geschrieben. Bis auf stark erweiterungsfähige oder besonders zentrale Elemente werden wir nichts aus älteren Büchern übernehmen. Eine Flash-Version später stehe ich also wieder vor der Frage: Wie geht es weiter? Flash-Bücher gibt es weiß Gott genug auf dem Markt. Was es aber noch nicht gibt, ist ein Buch von professionellen Flashern, die seit Jahren zusammen in einem Team ihr Geld ausschließlich mit Flash verdienen und die aus der Praxis schreiben können.

Mit Flash MX stehen wir vor einer neuen Generation des Internets, aber für alle ist es bisher Neuland. Umso schwieriger ist es, fundiertes und professionelles Praxiswissen zu dokumentieren. Man kann nur ahnen, was Sie und wir alles damit möglich machen werden. Was uns aber fast

unmöglich erschien, als wir dieses Buch zu schreiben begannen: Wir durften bereits einige Projekte mit Flash MX für Kunden realisieren.

Dabei wurde klar: Vieles hat sich nicht verändert und zählt seit Jahren zu unserem täglichen Geschäft – Workarounds, Fakes, Optimierungen und sicher auch einige altbekannte Bugs. Was liegt also näher, als in diesem Buch einige Tipps, Tricks und Geheimnisse des bisher der Öffentlichkeit vorenthaltenen Powerflasher-Know-hows preiszugeben.

Das machen wir nicht ganz uneigennützig. Insgeheim hoffen wir, die Standards, nach denen wir im Team programmieren und flashen, einigen Entwicklern unter Ihnen z. B. mit den ActionScript Conventions schmackhaft machen zu können. So wie das Teamwork unter uns bereits kompatibel ist, sollte die deutsche Flash-Szene auch funktionieren.

Na, und dann sind da ja noch ein paar weitere Spezialisten– z. B. Eric. Eric habe ich aus dem tiefen Underground der internationalen Flash-Szene an die Oberfläche gezerrt. Warum? Davon sollten Sie sich beim Lesen des XML-Kapitels überzeugen. Auch Philipp Cielen, einen der erfahrensten und engagiertesten Coldfusion-Guys des Landes, konnten wir für dieses Buch gewinnen. Nicht zu vergessen auch die Flashworker Danny, Ingo, Phillip und einige mehr aus der Szene, die wir persönlich und für Ihr Know-how in den letzten Jahren sehr zu schätzen gelernt haben. Es ist mir eine große Ehre, Ihnen dieses Buch gemeinsam mit ihnen zu präsentieren.

Noch etwas über die Entstehung dieses Buchs: Die lange Entwicklungszeit (sorry, dass wir nicht schon mit dem MX Release fertig waren) lag vor allem an einer dreistufigen QS. Nach der Autorenkorrektur wurde jedes Kapitel in die Hände eines unbeteiligten Flashers gegeben, der der Zielgruppe entsprach, um die Nachvollziehbarkeit zu testen. Alle Verständnisprobleme wurden daraufhin bereinigt und schließlich vom Verlagslektorat (vielen Dank für alles, Ruth) in Deutsch umgeschrieben ;-). Ich bin mir sicher, dass sich noch immer der eine oder andere Fehler eingeschlichen hat, aber das bleibt auf über 500 Seiten Know-how wohl kaum aus. Da wir sie nicht ganz verhindern können, haben sich alle Autoren bereit erklärt, im offiziellen Forum zum Buch Support zu leisten. Hier werden auch eventuelle Updates und Ergänzungen als PDFs bereitgestellt. Die URL ist www.galileodesign.de; auf dem Buchrücken finden Sie einen Code, mit dem Sie sich dort registrieren können.

In diesem Sinne viel Spaß mit unserem neuen, dicken, gar nicht bösen, großen Flash-Buch.

Was Sie erwartet

Der Titel

Flash MX professionell – Fortgeschrittene Workshops aus der Praxis.

Um dem langen Buchtitel gerecht zu werden, haben wir also sechs Ansprüche:

1. Wir schreiben über *Flash MX*. Dabei greifen wir höchstens auf einige nach wie vor aktuelle Tricks, Ideen *und* Optimierungen aus älteren Flash-Versionen zurück.
2. *ActionScript* wird eine zentrale Rolle dabei spielen. Wir zeigen, wie man sinnvoll mit dieser Programmiersprache umgeht und was man aus ihr herausholen kann.
3. Wir widmen uns *professionellen* Ansprüchen und Anwendungen.
4. Dabei richten wir uns an *fortgeschrittene* Anwender. Wir setzen also einen gewissen Grad an Vorwissen voraus und beginnen wenn möglich direkt mit der Vorgehensweise, ohne die Basics dafür zu erklären.
5. Wir zeigen zu jedem Thema einen oder mehrere *Workshops*, in denen wir die erklärten Techniken verdeutlichen und anwenden.
6. Und dabei beziehen wir die Auswahl der Themen *aus der Praxis*.

Die Struktur

Im ersten Teil des Buchs werden wir eine kurze Angleichung vornehmen. In dieser Angleichung versuchen wir nicht, Ihnen binnen weniger Seiten ActionScript beizubringen, sondern verdeutlichen Ihnen unsere Programmierweise, Syntax und Gedankenstrukturen. Das soll Ihnen helfen, die Logik der folgenden Seiten besser zu begreifen.

In den eigentlichen Workshops werden wir Ihnen zu dem jeweiligen Thema einen kurzen Rundumschlag geben, mit allen notwendigen Infos, vorausgesetztem Know-how und so manchem Tipp, der zum Thema passt.

Danach geht es dann jeweils in den Workshop, dessen Ziel es ist, eine sinnvolle Anwendung Schritt für Schritt zu realisieren. Dabei beginnen wir mit der Theorie, beschreiben die gewählte Lösung und ggf. die Gründe, warum andere Lösungen verworfen wurden, und erklären dann die einzelnen Elemente mit ihren Scripts.

Zum reinen Nachklicken eignen sich unsere Ausführungen nicht. Wir erklären nicht, wie man einen Button erstellt. Grundsätzlich sollten diese Workshops nur Inspiration sein, um weiterzumachen und eigene Ansätze zu verfolgen.

Was wir erwarten

Wir setzen voraus, dass Sie die Grundfunktionen von Flash beherrschen, und natürlich, dass Sie mit Flash 5 ActionScript bereits vertraut sind. Sicherlich kann man auch vieles lernen, ohne es vollkommen zu verstehen. Natürlich kann man sich auch, ohne die Dimension des eigentlichen Scripts zu überblicken, Teile der angewandten Technik aneignen. Definitiv möchte ich nach diesem Buch aber die folgende Kritik bei Amazon nicht mehr lesen: »Für mich als Anfänger ist dieses Buch leider nicht geeignet...«

Wenn Sie aber nichts mehr lernen wollten oder bräuchten, hätten Sie dieses Buch wohl nicht gekauft. Um klarzustellen, was Sie allerdings schon wissen sollten, grenze ich den Bereich des vorausgesetzten Vorwissens etwas ein:

- Grafiken, Buttons und Filmsequenzen brauche ich wohl nicht zu erwähnen.
- Scripts auf Zeitleisten oder Objekten (onClip-Events)
- Dotsyntax zur Steuerung von Zeitleisten und Eigenschaften von Objekten
- Variablen, Schleifen und Bedingungen
- Das ist das Grundwissen, das wir in unseren Schulungen in Flash Advanced und Flash ActionScript legen. Speziellere Sachen wie Arrays, Objekte etc. habe ich jetzt einmal bewusst weggelassen, da sie nicht in jeder Anwendung benötigt werden. Sollte tieferes Vorwissen erforderlich sein, werden wir es vorher in der Theorie des jeweiligen Kapitels erklären oder darauf hinweisen, sodass Sie es sich ggf. noch anlesen können.

Und obwohl dieses Buch schon wenige Wochen nach dem Flash MX Release erschienen ist, setzen wir auch den Umgang mit der neuen Oberfläche voraus und erklären nicht, mit welchen neuen Shortcuts jetzt alte ausgetauscht wurden o. Ä.

Sollte es an Vorwissen mangeln, kann man die einzelnen Funktionen sehr angenehm in der internen Referenz oder Online-Dokumentation nachlesen. Wer ein ganzheitlicheres Verständnis sucht, sollte sich das ActionScript-Buch von Galileo Press ansehen:

Sascha Wolter: ActionScript. Codedesign und Objektorientierung mit Flash MX. ISBN: 3-89842-221-6 (erscheint im September 2002).

Ich hoffe, es gibt keine Missverständnisse mehr, und wir können nun richtig loslegen.

Danksagung

Für das Verständnis und die Unterstützung meines seit Jahren zum Beruf gewordenen zeitaufwändigen Hobbys danke ich meiner ganzen Familie (besonders Lili) und allen zu kurz kommenden Freunden.

Für den Support, den Austausch und die Hilfe, Unmögliches möglich zu machen, danke ich allen Flashworkern (Petter, Phaeton, Danny, Nick, … – meinen Helden) und unserer kleinen macromedia.general.germany-WG.

Last but not least danke ich für die Nerven, Geduld, Zeit und Power, unseren Traum wahr zu machen, dem ganzen Power-Team: Andre, Bernd, Björn, Carsten, Daniel, David, Hassan, Karina, Lasse, Martin, Nico, Ramon und Thomas.

In eigener Sache 1

Macromedia hat mit Flash ein Tool auf den Markt gebracht, das User-Support und Communities wie kaum ein anderes Programm im Internet hat. Es gibt eine Vielzahl von weltweiten Usergroups, Ressourcen und Open-Source-Bewegungen. Diese Menschen glauben wie wir daran, dass es eine kurzfristige Strategie ist, Know-how zurückzuhalten, dass man gemeinsam stärker ist und dass der gegenseitige Austausch alle weiterbringt. Wenn Sie von Webseiten wie www.flashworker.de profitieren, dann geben Sie ihnen auch etwas zurück, und wenn es nur ein FLA, ein Tutorial, ein Newsletter-Beitrag, FAQ, Sound oder sonst etwas ist. Nur so ist die langfristige Existenz solcher Bewegungen gesichert.

In eigener Sache 2

Um das gleich zu Beginn hinter mich zu bringen: Powerflasher sucht ständig gute Leute zur Festanstellung. Bewerben Sie sich – bitte keine Freelancer. Trotz bzw. wegen schlechter Freelancer-Erfahrung pflegen wir aber sehr gern den Austausch und Kontakt mit anderen professionellen Teams.

Aachen, im Juni 2002
Carlo Blatz

Optimierung

Flasher ist nicht gleich Flasher

Wie in jedem Handwerk gibt es viele Möglichkeiten, sauberere, schnellere, kleinere und bessere Ergebnisse zu erzielen. Neben den ActionScript Conventions sind auch (oder vielleicht gerade) rechts und links vom Script viele Ansätze zur Optimierung offen. Unsere Tricks dazu zeigen wir Ihnen hier.

Dieses Kapitel gab es bereits in der Flash 5-Auflage dieses Buchs. Es wurde für die aktuelle Auflage neu überarbeitet und ergänzt.

Vorbereitungen

Grundsätzliches

Das Wichtigste ist, einen Plan zu haben. Ein roter Faden und ein Konzept in der Animation machen ein Projekt erst rund und ansprechend. Alles andere wirkt wie eine sinnlose Aneinanderreihung von Effekten. Außerdem spart man unter Umständen viel Zeit. Überlegt man sich vorher nicht, welche Probleme auftreten können, welche Anforderungen gestellt werden, produziert man am Ende nur Kompromisslösungen oder muss sogar neu anfangen.

Hat man also einen solchen Plan, z. B. in Form eines Storyboards, gilt es erst einmal, die Daten zu sichten. Sind alle Bilder da, sind alle Texte da, liegen die Logos als Vektoren vor, gibt es die richtigen Schriften? Sie sollten sich jeden Schritt vom Kunden absegnen lassen. Wurden die Texte geschrieben, soll der Kunde unterschreiben, dass sie genau so verwendet werden sollen. Wurden die Layouts erstellt, soll der Kunde bestätigen, dass sie genau so umgesetzt werden können. Und wurde die erste Seite animiert, soll der Kunde unterschreiben, dass die anderen Seiten genau so adaptiert werden können.

Kommt der Kunde dann im Nachhinein mit Änderungen, kann man diese auch berechnen – und meist erübrigen sich dann die meisten Änderungswünsche. Im Angebot bzw. Auftrag sollte natürlich ein Korrekturschritt vereinbart sein, wo der Kunde die Chance hat, eine Liste von Änderungen (außerhalb der bereits unterschriebenen Schritte) vorzulegen. Sind diese Korrekturen wunschgemäß ausgeführt, sind alle weiteren Korrekturwünsche wieder berechnungsfähig.

Ausführlichere Informationen über diesen Prozess finden Sie im Kapitel »Projektmanagement«.

Bildrate

Eine der wohl spektakulärsten Speed-Optimierungen seit dem letzten Buch, vor allem für Macintosh-Player, ist die korrekte Einstellung der Bildrate. Erst vor wenigen Monaten wurde entdeckt, nachgewiesen und mir von Eric Wittman (Productmanager of Flash) offiziell bestätigt, dass es einen Bug im Mac-Player gibt. Dieser ignoriert Unterschiede in den Bildraten zwischen 19 und 30 Bildern pro Sekunde. Alle Filme werden gleichermaßen mit 19 bps abgespielt. Ab einer Bildrate von 31 bps haben wir erst wieder eine echte Kontrolle.

Es war also keineswegs ein Performance-Problem, dass Flash-Filme auf dem Mac derartig ruckelig liefen. Zur Erinnerung: 10 Bilder pro Sekunde hat Southpark, 17 bis 20 bps sind die Schmerzgrenze für flüssige Animationen (so weit sollten Sie nur bei prozessor-intensiven Animationen reduzieren), 24 bps sind für das Auge flüssig, 25 leichter zu rechnen – und nun neu: 31 bps sogar flüssig auf dem Mac.

Bei reinen ActionScripts darf die Bildrate auch gern auf 80 bps und mehr geschraubt werden. Dabei sollte man aber auf die CPU-Auslastung achten. Flash ist aggressiv und lässt anderen Programmen keine Systemressourcen übrig, wenn viel von ihm verlangt wird. Daher sollte man den fertigen Film auf solche Dauerbelastungen testen. Unter Windows NT und 2000 kann man die Systemleistung bequem mit STRG-ALT-ENTF im Taskmanager anschauen. Für andere Systeme gibt es dafür separate Tools. Zum Testen sollte man natürlich ein der Zielgruppe entsprechend konfiguriertes System einsetzen.

> Im I.D.E. (Testmodus in Flash) von Flash 5 geht die Performance sogar bei einem leeren Film auf 100%. Flash scheint schon einmal sicherheitshalber alles zu bunkern. Ein realistisches Ergebnis über die Systemressourcen sieht man also nur im Browser.

Die Tatsache, dass das Flash-Plugin so ressourcen-aggressiv ist, gibt dem sauberen Programmierer weitere Optimierungsansätze. Längere Schleifen sollten z. B. kurze Pausen haben, in denen andere Programme »Luft holen« können (z. B. Word autosave). Insbesondere `enterFrames` nagen konstant an den Ressourcen, auch wenn darin (z. B. wegen einer nicht wahren Bedingung) kein Script ausgeführt wird. Ein `enterFrame` sollte also nie unnötig laufen.

Bühnengröße

Aber wir waren bei den Vorbereitungen stehen geblieben. Ist es nun endlich so weit, dass es »ins Flash« geht, muss die schwierigste aller Fragen geklärt werden – die Bühnengröße.

Vorab entscheidet man, in welchem Umfeld die Anwendung präsentiert werden soll. In einem Projektor gibt es die Möglichkeiten **Fullscreen** oder **feste Größe.** Eine feste Größe ist natürlich einfach, bei Fullscreen muss man schon eher die leicht verschiedenen Proportionen z. B. zwischen 800 x 600 und 1024 x 768 Pixel berücksichtigen.

Spätestens wenn man aber Grafikkarten-intensive Animationen verwendet, sollte man den Film unskalierbar zentriert exportieren und rechts und links von der eigentlichen Bühne ein weitergeführtes Layout oder einen Rahmen bis zum Rand der größten Auflösung ziehen. So wird der eigentliche Content nicht mitskaliert, was z. B. für Pixeldesigns, großflächige Animationen etc. sehr wichtig ist. Gleiches gilt natürlich auch für einen Browser, der fullscreen geöffnet wird. Fullscreen ist im Übrigen oftmals nicht gern gesehen, da es als dominant und aggressiv angesehen wird und zudem jeden Multitasker seiner Taskleiste beraubt.

Bei den folgenden Screenshoots (www.esser-rechtsanwaelte.de) sieht man zwei Popups, die die gleiche SWF-Datei darstellen. Bei 1024er Auflösung oder höher öffnet sich automatisch das »normale« Popup; bei 800 x 600er Auflösung sieht man die aufrufende Seite mit einem Hinweis, dass die Seite für 1024+-Auflösungen optimiert ist, und einem Button, mit dem man dieses kleinere Popup öffnen kann. Da die Bildqualität wichtig ist und eine Pixelschrift verwendet wurde, darf das SWF nicht skalierbar sein. So wird im kleineren Popup durch geschicktes Positionieren des SWFs rechts und links etwas vom eigentlichen Layout abgeschnitten, ohne wichtige Contents anzuschneiden. Solche und ähnliche Tricks sind immer wieder erforderlich, um eine bestmögliche Darstellung zu erreichen. Daher kann man auch kein allgemeines »Exportrezept« aussprechen.

▲ **Abbildung 1**
Unskalierbares Popup bei 1024er Auflösung

▲ **Abbildung 2**
Gleiches Popup bei 800er Auflösung: Gleiches SWF, kleineres Popup durch JavaScript

Ansonsten sind bei Popup und Standard-Browser häufig Entscheidungsfindungen gefragt. Grundsätzlich ist nicht bei jedem User ein Popup gern gesehen. Schließlich kommt man nicht an seine Browser-Features. Das kann natürlich vom Programmierer z. B. wegen des fehlenden ZURÜCK-

Buttons durchaus erwünscht sein. Die Entscheidung gegen ein Popup wegen solcher Argumente ist, denke ich, nicht tragfähig, wenn ein Popup sinnvoll ist. Es ist inzwischen verbreitet und akzeptiert.

Ein Popup verwendet man vorrangig, wenn man ein festes unskalierbares Design angelegt hat, um die Proportionen zu schützen oder einfach um eine bestmögliche Bildqualität zu gewährleisten. Schließlich franst jedes Pixelbild bei nur 0,1 Prozent Skalierung unschön aus.

Bei Bannern und Popups sowie bei festen Frames oder Tabellen ist die Antwort also einfach. Wenn man aber für variable Frames, Standard-Browser oder eben gar Fullscreen-Browser entwickelt, muss man auf die Frage der Bühnengröße mehr Zeit verwenden.

Als Erstes könnte man z. B. den Screenshoot von einem gängigen Browser in Fullscreen bei einer gängigen Auflösung erstellen und die nutzbare Fläche des Browsers ausmessen. Die derzeit immer noch gängigste Konfiguration ist PC, Windows, Internet Explorer 5 bei einer Auflösung von 1024 x 768. Die nutzbare Fläche des Browsers entspricht hier in der Standardeinstellung der Symbolleisten ca. 1020 x 610 Pixel. Legt man die Bühne mit diesen Werten an, hat man schon für eine maximale vereinheitlichbare Menge von Benutzern einen guten Wert. Auch hier darf man nicht auf den Pixel genau arbeiten, sondern muss das Design über den Anschnitt hinaus anlegen. Eine weitere Browser-Zeile wie z. B. für Favoriten, Google o. Ä. verzieht die Proportionen bereits. Außerdem gibt es natürlich auch noch andere Browser, Auflösungen, Symbolleisteneinstellungen und den Macintosh, in dem die Anwendermasse Ihren Browser seltenst in einer verlässlich einheitlichen Größe öffnet.

Man muss sich also überlegen – eigentlich muss das der Grafiker schon –, was passieren soll, wenn der User eine andere Auflösung verwendet oder seinen Browser anders skaliert. Die Auflösung an sich ist ja nicht die eigentliche Schwierigkeit; dank Vektoren lässt sich Flash ja ohne Qualitätsverlust skalieren. Vielmehr ist es wichtig, die Proportionen zu wissen. Die sichtbare Fläche eines Browsers ist meist etwa im 4:3-Format. Doch bei präzisen Layouts kann 4:3,2 oder 4,1:3 schon viel ausmachen.

Der Schlüssel liegt in den Exporteinstellungen. Hier kann man diverse Optionen beeinflussen, wie Flash mit dem Anschnitt umgehen soll. Publiziert man auf einer festen Pixelgröße, ist es nicht so wichtig; bei 100% x 100% können aber doch sehr unschöne Effekte wie Verzerrungen oder falscher Anschnitt auftreten.

Im Folgenden sehen Sie Screenshoots von www.sebo.de (1999). Hier haben wir ganz bewusst mit dieser freien Skalierbarkeit gespielt und das

Layout vertikal im Anschnitt über die Bühne hinaus angelegt und horizontal bereits in der Bühne begrenzt. Die Exporteinstellungen besagen, dass der Film 100%x100% frei skalierbar ist, horizontal links und vertikal zentriert positioniert wird, dass eine überproportionale Fläche über den Bühnenrand hinaus gezeigt werden darf und dass die Bühne nicht angeschnitten werden darf. Das Layout sieht so also trotz freier Skalierbarkeit bei fast jeder Browser-Größe »gewollt« aus.

▲ **Abbildung 3**
Browser maximiert

▲ **Abbildung 4**
Browser horizontal gestaucht

▲ **Abbildung 5**
Browser vertikal gestaucht

Grundsätzlich gibt es hier keine Idealeinstellung, die wir als Optimum präsentieren können. Man muss von Layout zu Layout neu entscheiden. Macromedia Dreamweaver MX z. B. hilft besonders, da man hier nach dem WYSIWYG-Prinzip (What You See Is What You Get) mit den Einstellungen experimentieren kann, bis man ein gewünschtes Ergebnis hat.

Auch für den Import der Pixelbilder (mehr dazu erfahren Sie im Abschnitt »Pixelbilder« weiter unten) ist es wichtig, die genaue Bühnengröße zu wissen. Ohne Pixelbilder könnte man den Film genauso in 500 x 300 Pixeln anlegen, solange die Proportionen stimmen. Mit Pixelbildern würde aber eine nur einprozentige Skalierung eine schlechtere, ausgefranste, unscharfe Bildqualität bedeuten – ganz zu schweigen von Pixelfonts, die ja auf keinen Fall in irgendeiner Weise transformiert werden dürfen.

Wird der Film hingegen auf 2000 x 1200 Pixel angelegt, ist die Bildqualität zwar auch in größeren Auflösungen gut, die Dateigröße aber unsinnig groß. Auch hier fransen die Bilder in jeder kleineren Auflösung aus, da die angelegten Pixel von der Grafikkarte auf die darzustellenden Pixel auf dem Monitor interpoliert werden müssen. Es ist also sinnvoll, nur mit Vektoren auszukommen, bzw. eine feste Größe, am besten mit variablen Anschnitten, zu verwenden. Ist beides nicht möglich, muss man im Kompromiss zwischen Dateigröße und Qualität für eine durchschnittliche Auflösung optimieren.

Struktur

Die Struktur ist ebenfalls eine sehr wichtige Komponente sowohl bei der Arbeit, wie bereits angesprochen, als auch für das spätere Ergebnis. Das Internet ist ein sehr schnelles Medium. Durchschnittliche Verweilzeiten liegen im Sekundenbereich. Da ist es dringend notwendig, den User zu entlasten und ihm eine Struktur zu bieten, die er schnell durchschauen kann. Gut – nun bietet sich gerade Flash an, um ausgefallene Sachen zu machen, die eben in kein Baumdiagramm einzuordnen sind. Aber auch hier sollte man gewisse Regeln einhalten.

So sollte eine Seite zum Beispiel nicht zu tief verschachtelt sein. Maximal zwei Ebenen tief sollte man in eine Struktur eintauchen können. Sind weitere Unterpunkte notwendig, kann man auch animierte Flash-simulierte Popups, echte Popups etc. verwenden, die verhindern, dass sich der Anwender verläuft.

Man sollte die Navigation auch nicht innerhalb der Seite ändern. Eine Seite muss nicht aussehen, wie man es gewohnt ist, aber sie sollte auch nicht in sich widersprüchlich sein. Der Mensch ist ein Gewohnheitstier, und zum Glück gewöhnt er sich recht schnell an neue Layouts. Wenn er auf einer Seite aber gar kein System findet, strengt das unnötig an, und dafür bedankt sich niemand.

Das von uns Powerflashern zu hören mag etwas ironisch klingen, schließlich spielen wir mit diversen Navigationen innerhalb der Seite. Wir trennen einfach zwischen der Usability für unsere Kunden (bzw. deren Kunden) und der Experimentierfreudigkeit unserer Besucher. Mit einer durchschnittlichen Verweilzeit von fast sieben Minuten auf der Seite ist auch dieser Neugierde-Effekt nicht zu unterschätzen. Klar ist aber auch, dass hier niemand binnen Sekunden an Informationen kommen muss/soll.

Zur Struktur gehört auch, dass man das Rad nicht auf jeder Seite neu erfindet. Man sollte sich auf einige Animationen oder Arten der Überblendung etc. festlegen und diese konsequent verfolgen. Nur so kann ein geschlossenes Konzept ohne Bruch entstehen.

Tricks für kleine Dateigrößen

Greifen wir noch einmal die Bildrate auf. Bei sehr Tweening-lastigen Animationen sollte man die Bildrate nicht zu hoch stellen. 31 Bilder pro Sekunde (wenn so viel möglich ist) sind, wie eben gelernt, auch für Macintosh-User fair. Da jedes Bild in einem Tweening einige Bytes »frisst«, sollte die Bildrate aber nicht unnötig höher gestellt werden. Für eine Sekunde Animation braucht man sonst entsprechend mehr Bilder, die etwas mehr Dateigröße beanspruchen. Wir sprechen hier von 32 Byte pro Bild bei einem nicht getweenten Bild, aber wie so oft macht auch Kleinvieh Mist.

Das A und O für kleine Dateien ist aber natürlich der sinnvolle Einsatz von Symbolen. Jedes Objekt, das mehr als einmal gezeigt wird – und das ist bei jedem Tweening der Fall –, sollte ein Symbol sein. Andersherum sollten Objekte, die lediglich einmal verwendet werden, keine Symbole sein (sofern man keinen Instanzeffekt anwenden muss).

Das Grafik-Symbol

Das Symbol-Verhalten `Grafik` sollte man im Übrigen ganz vermeiden. In der letzten Auflage habe ich noch geschrieben, dass es sich hier seit Version 2 um einen Bug in Flash handelt. Inzwischen revidiere ich diese Aussage und behaupte, dass es in der Natur einer Grafik liegt, unter Umständen mehrmals übertragen zu werden.

Eine Grafik ist das optische Äquivalent zu einem Streaming-Sound. Es streamt trotz Symbolverschachtelung in jedem Frame zur Hauptzeitleiste und wird dementsprechend auch bei jeder Wiederholung neu übertragen. Eine Grafik aus nur einem Schlüsselbild hat diesen Effekt entsprechend nicht.

Das muss man wissen und bei seiner Arbeit berücksichtigen, denn jedes Mal, wenn man die Instanz einer Grafik verwendet, wird diese neu übertragen. Damit nicht genug: Die Dateigröße wächst eben auch mit der Frame-Zahl, die eine solche Grafik auf der Zeitleiste bekommt, wenn sie auf WIEDERHOLUNG steht – sie streamt.

Festgestellt haben wir diesen Bug, als wir an einem Intro für Advocard arbeiteten und der Kunde im Nachhinein sein Timing geändert haben

wollte. Wir haben das Intro auf Kundenwunsch lediglich verlangsamt, also mehr Frames hinzugefügt, und die Dateigröße stieg rapide – ohne dass eine neue Animation oder ein neues Symbol hinzugefügt wurde. Wenn Sie also dieses Streaming-Verhalten des Symbols nicht explizit benötigen, sollten Sie einfach MovieClips statt Grafiken verwenden.

Bei Trickfilmen o. Ä. in Flash sind Grafiken natürlich sinnvoll, weil man bereits im Bearbeitungsmodus die Animation im Symbol sehen und synchronisieren kann. Man sollte aber auch dann in der finalen Fassung einmal testen, was passiert, wenn man alle Grafiken in VERHALTEN auf FILMSEQUENZ stellt. Falls die Animation im Symbol auch stoppen soll, wenn die Zeitleiste gestoppt wird, kommt man allerdings ohne ActionScript kaum an Grafiken vorbei. Versierte Scripter könnten alle MovieClip-Instanzen in einem Array speichern und dann eine neue Stop-Funktion schreiben, die entsprechend das gesamte Array mittels einer `for`-Schleife »durchstoppt«. Eine entsprechende Play-Funktion wäre dann natürlich auch wieder sinnvoll.

Bewegungstween erstellen

Ein häufiger Fehler ist die Verwendung der Option BEWEGUNGSTWEEN ERSTELLEN. Wer hier nicht vorher dafür gesorgt hat, dass im Anfangs- und im Endbild ein Symbol steht, dem setzt Flash ein Symbol in das Anfangsbild. Nicht nur, dass dieses Symbol »Tween x« (x = laufende Nummer) heißt, es ist zudem auch eine Grafik, was schlecht ist, wie wir eben gesehen haben. Und noch schlimmer: Im Endbild hat Flash es beim ursprünglichen Objekt belassen. Das Objekt wird also so schon doppelt übertragen.

Schrifteinbindung

Dann gibt es natürlich noch diverse Feinheiten. So sollte man nicht automatisch eine Outline um jeden Vektor machen, wenn man sie eventuell sowieso nicht sehen würde. Das beansprucht bei einem Viereck nur wenige Bytes, aber ich finde, man sollte da von vornherein gründlich sein. Warum muss im Aktiv-Zustand eines Buttons z. B. eine Outline vorhanden sein? Man sieht sie doch nicht. Ich versuche Ihnen mit diesem Beispiel ein Gefühl dafür zu geben, wie Flash im Innersten funktioniert. Wer das verinnerlicht und auch für solche Kleinigkeiten etwas Feingefühl entwickelt, kann so manches KB sparen.

Erstaunlich viele Kilobytes kann man durch den richtigen Umgang mit dynamischen Textfeldern sparen. So wird standardmäßig die gesamte

Schriftart eingebunden. Das ist, wenn man keine mehrsprachige Webseite inklusive französischer, polnischer etc. Übersetzung erstellt, nur selten notwendig. Eine durchschnittliche Schrift hat etwa 20 KB – das kann durchaus bis 150 KB hoch gehen.

Ich erstelle ein dynamisches Textfeld und binde den gesamten Arial-Zeichensatz ein. Mit der Einstellung GRÖSSENBERICHT ERSTELLEN erhalte ich beim Export eine Textdatei, in der mir jedes eingebundene Zeichen explizit aufgelistet wird. Bei einer recht kleinen Schrift (weil serifenlos) wie Arial bindet Flash folgende Zeichen ein:

Arial 20551 Byte

!»#$%&'()*+,./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstvwxyz{|}~□□□□□□□ ¡¢£¤¥¦§¨©ª«¬-®¯°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþÿŒœŠšŸzZƒ˜––''‚«»«†‡•…‰‹›€™

Lediglich die hier fett gestellten Zeichen werden in der deutschen Sprache verwendet. Diese Zeichen lässt man sich in der Textdatei oder im Ausgabeinspektor des Testmodus darstellen und fügt sie mit Copy-and-Paste von Hand als verwendete Zeichen in die Textfeldeigenschaften ein. Das Eurozeichen sollte man natürlich noch hinzufügen. So sind hier bereits 14 KB gespart – die Restschrift ist nur noch 6 KB groß.

Das kann man weiter treiben – wenn eine bestimmte Schrift nur z. B. für einen Preloader als dynamisches Textfeld verwendet wird, genügen doch alle Zahlen und das % Zeichen – satte 18 KB sind allein bei einer Arial wieder gespart. Diese 18 KB müssten in diesem Fall ja sogar geladen werden, bevor man nur den Preloader sieht.

Mit durchschnittlich drei Schriftarten in einem Projekt kann man hier schon 50 KB und mehr sparen.

Film komprimieren

Die wohl naheliegendste Dateigrößenoptimierung ist die Flash MX-Exporteinstellung FILM KOMPRIMIEREN. Flash verfügt seit dem Flash 6-Plugin über eine Z-LIB-Kompression, wie man sie z. B. von ZIP oder RAR kennt. Hier werden insbesondere Texte durch mathematische Komprimierungsalgorithmen verkleinert. Das hat zur Folge, dass der Film natürlich nicht mehr abwärtskompatibel ist und auch etwas langsamer läuft. Die SWF-Datei muss natürlich vom Plugin wieder dekomprimiert werden.

Einen spürbaren Effekt hat diese Option nur bei sehr textlastigen Filmen und bei sehr ActionScript-lastigen Filmen. Letzteres hat nebenbei noch den Effekt, dass ActionScript-Editoren wie der »Burak ActionScript

Viewer« bis Version 2.5 nicht mehr den Sourcecode ausspionieren können. Mit den neueren Versionen geht es aber schon wieder. Auch Vektoren werden hier noch stark komprimiert, vor allem, wenn sie aus Programmen wie Swish 3D kommen. Hier wird eine optische Linie aufgrund von Rechenfehlern aus zwei übereinander liegenden Linien beschrieben. Solche und andere optisch unsichtbare Optimierungen werden also ebenfalls vorgenommen. Pixelbilder und Sounds werden von der Z-LIB-Kompression nicht weiter verkleinert, demzufolge leidet auch keine optische oder akustische Qualität.

Pixelbilder

Flash kann diverse Formate importieren, aber nur wenige davon sind sinnvoll. Wir haben GIF, PNG, JPG und bei Windows BMP als mögliche Pixelbild-Importformate bzw. mit installiertem QuickTime noch weitere Formate wie TIF (eq. BMP).

GIF verwendet man meist, um Bilder zu importieren, die man auf wenige Farben reduzieren kann.

PNG eignet sich dafür aber noch besser. Steht ein PNG in der Bibliothek auf VERLUSTLOS, hat das Bild eine gestochen scharfe Qualität. PNG hat noch einen weiteren Vorteil. Man kann mit PNGs auch komplexe Alphaverläufe importieren, während man bei GIF nur eine Farbe auf transparent stellen kann. Allerdings steigt die Dateigröße bei transparenten Verläufen sehr. Seit Flash 5 kann man auch aus Fireworks gespeicherte Vektoren im PNG-Format übertragen. Wenn eine Fireworks-PNG-Datei über die Zwischenablage mit AUSSCHNEIDEN und EINFÜGEN importiert wird, wird die Datei in eine Bitmap konvertiert.

Kann man ein Bild nicht auf zwei bis maximal vierundsechzig Farben reduzieren, eignet sich PNG nicht. Die meisten verwenden dann **JPG**. Allerdings lohnt es sich nicht, ein JPG vorher zu komprimieren und es dann in Flash erneut komprimieren zu lassen. Die Datei wird dadurch nicht kleiner, sondern nur die Qualität schlechter. Ist das JPG z. B. von Fireworks oder Photoshop 5.5+ komprimiert worden, bietet Flash in den Bitmap-Eigenschaften jedes Bildes IMPORTIERTE JPG-DATEN VERWENDEN statt STANDARDQUALITÄT DES DOKUMENTS VERWENDEN an. Ist man gewöhnt, seine Bilder in diesen Tools zu komprimieren oder möchte man größere Mengen batchen, kann das durchaus eine Alternative sein.

▲ **Abbildung 6**
Importierte JPG-Daten verwenden

Die besten Erfahrungen haben wir bei nicht-farbreduzierbaren Bildern aber mit **BMP** gemacht. Entgegen allen Gerüchten ist die Flash-interne JPG-Komprimierung gar nicht so schlecht. Für beste Ergebnisse benötigt Flash einfach ein unkomprimiertes Format wie BMP. In vielen Tests hat sich herausgestellt, dass importierte (auch vorkomprimierte) JPGs schlechter und größer sind als importierte BMPs, die Flash komprimiert. Flash behält die Transparenzeinstellungen importierter Bitmaps übrigens bei. Das funktioniert aber nur über das normale Importieren, in der Zwischenablage gehen die Transparenzeinstellungen verloren.

Die allerwichtigste – fast selbstverständliche – Regel ist allerdings, dass die Bilder mit 72 dpi (Bildschirmauflösung) importiert werden und vor allem in 100% Größe der späteren Endgröße auf der Bühne. Es nützt nichts, ein Bild mit 600 x 600 Pixeln zu importieren und es auf der Bühne auf nur 300 x 300 Pixel zu skalieren. Die Datei bleibt groß. Für die Qualität ist auch eine einprozentige Skalierung von Nachteil. Das Bild franst aus, da das eigentlich Pixel für Pixel angelegte Bild sich nun in dem entsprechenden Skalierungsfaktor auf eine andere Anzahl Pixel interpoliert verteilen muss.

In Abbildung 7 sehen Sie eine Seite von www.montblanc.com. Hier haben wir wegen der enormen Pixelbildlastigkeit ein festes, unskalierbares Popup genommen. Die Bilder wurden unkomprimiert und unskaliert als BMP oder PNG importiert. BMPs wurden entsprechend in Flash als JPGs exportiert.

▲ **Abbildung 7**
Beste Bildqualität durch festes Popup, unskalierte Bilder etc.

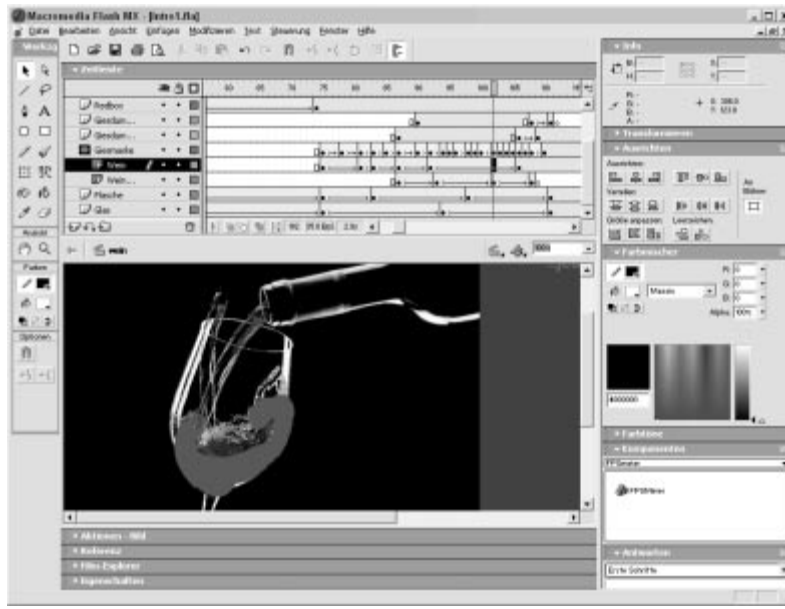
Wenn man eine Reihe von Pixelbildern hat – z. B. ein Video aus Einzelbildern, braucht man diese nicht von Hand einzeln zu importieren. Solange die Bilder als letztes Zeichen eine fortlaufende Nummer haben, erkennt Flash beim Import eines Bildes, dass es sich um eine Bildsequenz handeln könnte, und fragt, ob man sie komplett importieren möchte oder nicht.

Formtweenings

Sehr oft braucht man Formtweenings nicht, doch so mancher Effekt lässt sich nur damit zaubern – z. B. ein Maskeneffekt, in dem aus einem Foto (im Auftrag von Koch Marketing Schweiz für Rutishauser Wein) das flüssige Eingießen eines Weinglases gezaubert werden sollte (siehe Abbildung 8 und 9).

Frei nach dem Motto »Flash ist auch nur ein Mensch« hat Flash auch seine Grenzen. Die Grenzen bei Shapetweens sind zudem noch recht schnell erreicht. Dennoch gibt es einige Tricks, seine Shapes so zu optimieren, dass diese wie gewünscht aussehen.

Die wichtigste Regel ist die Reduktion von Vektoren auf ein Minimum. Vielmehr sollte man sogar – wenn es geht – eine zu verformende Grafik in mehrere Teile aufteilen und diese auf verschiedene Ebenen verteilen. So könnte ein zu tweenendes Gesicht in Mund, Nase, Auge, Auge und Hintergrund aufgeteilt werden.



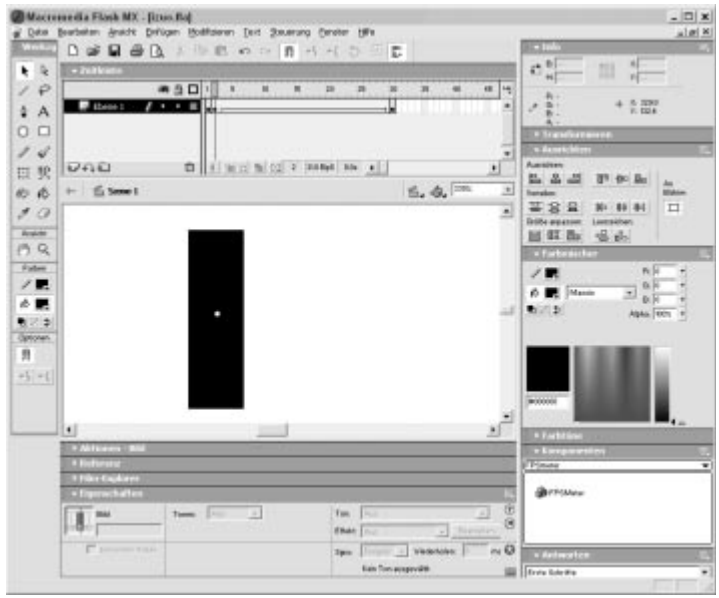
▲ Abbildung 8
Ein Pixelbild wurde mit Masken in Einzelteile zerlegt und getweent.

▲ Abbildung 9
Fertig: siehe www.rutishauser-wein.ch

Die Chance, dass Flash die Zusammenhänge von vier Vektoren logisch nachvollziehen kann, ist größer, als die Zusammenhänge von 25 Vektoren zu organisieren. Man macht es Flash so etwas einfacher, worüber sich auch der Prozessor und die Grafikkarte beim Abspielen freuen.

Formmarken können die Animation natürlich am besten beeinflussen. Man kann maximal 25 davon setzen und sollte sie im Uhrzeigersinn platzieren. Am effektvollsten sind Formmarken an Schnittpunkten und Ecken. Das Prinzip kennt man vom »Morphing«.

Flash rechnet auch nur mit Mathematik, und daher gibt es ein paar Prinzipien, die man einhalten sollte, um schöne Animationen zu erstellen. Als beliebtes Beispiel verwende ich in Schulungen ein Formtweening von I nach O. In jedem Formtweening sollten die Vektoren im Anfangs- und im Endbild vergleichbar sein. I und O sind das nicht. Das O hat ein »Loch« im Vektor. Um die Animation überhaupt rechnen zu können, macht Flash im ersten Bild des Tweenings ein riesiges Loch, um dieses dann zum O zu tweenen. Mit einer kleinen Hilfe macht Flash die Animation aber flüssig. Dafür erstellt man das erste Bild der Animation von Hand und radiert ein kleines Loch in das I. Nun sind Anfangs- und Endvektor vergleichbar. Den Trick sieht der User nicht.



▲ **Abbildung 10**
Durch Radieren eines kleinen Lochs kann man ein I zu einem O formtweenen.

Eine weitere Möglichkeit, die Animation zu beeinflussen, sind weitere Schlüsselbilder, in denen man die Animation »korrigiert«. Wenn man Formmarken verwendet, muss man beachten, dass man auch immer ein neues Anfangsbild der Animation benötigt. Ein Korrekturschritt erfordert dann also zwei Schlüsselbilder.

Bewegungstweenings

Optimieren heißt beim Motiontween vor allem beschleunigen. Ein optimales Motiontween ruckelt nicht. Bei manchen Animationen lässt es sich nicht verhindern, aber es gibt ein paar klare Tipps, die man befolgen kann.

Alpha-Effekte sind für den Prozessor und vor allem für die Grafikkarte sehr arbeitsintensiv. Man sollte sie nur im Notfall verwenden. Wir sprechen nicht von kleinen Bannern, sondern von großen Animationen. Fließtexte sind ein schönes Beispiel: viele Vektoren auf großer Fläche. Wenn ein einfarbiger Hintergrund vorliegt, ist die Lösung nahe liegend. Man färbt einfach das Symbol in der entsprechenden Hintergrundfarbe, und somit ist es unsichtbar. Bei Schwarz und Weiß kann man noch einfacher die Helligkeit verwenden. Der Effekt ist phänomenal. Auch bei Pixelbildern macht dieser Trick enorm viel aus. Die Frameraten verdoppeln sich bei diesen kritischen Animationen zum Teil bzw. erhöhen sich noch mehr.

Aber auch bei verschiedenfarbigen Hintergründen lässt sich oft ein Alpha-Effekt vermeiden. Man nimmt einfach einen Wert, der dem Durchschnitt des Hintergrundes nahe kommt. Solange die Animation nur wenige Frames lang ist, bemerkt der User den Trick nicht.

Beim einem Projekt www.powerflasher.de/elsa/xmas (der Fiskus hab sie selig) haben wir sogar festgestellt, dass unanimierte Bilder mit einem Farbeffekt von Flash deutlich langsamer zu animieren sind als ohne und dass Vektoren mehr Performance schlucken als Pixelbilder. Natürlich – die Engine von Flash muss ja die Vektoren erst einmal errechnen, darstellen und das ggf. 30-mal in der Sekunde. So bauen wir alle großflächigen Animationen möglichst aus Pixelbildern auf. Der Hintergrund dieses Spiels – die Häuserzeile – ist ein Pixelbild. Wenn man nun den MovieClip, in dem das Pixelbild liegt, einfärbt – nur 10% Blau hinzugibt oder es 20% aufhellt –, sinkt die Performance spürbar. Man sollte solche Änderungen also vorher im Grafikprogramm vornehmen.

▲ **Abbildung 11**
Bild 1 von 5 bei einer Animation mit Vermeidung eines Alpha-Effekts

Nicht nur ein Farbeffekt wirkt sich negativ auf die Performance aus. Auch eine Rotation, Skalierung oder sonstige Transformation verlangsamt die Performance. Das Objekt sollte also im Symbol verkleinert oder gedreht und nicht mit konstanter Skalierung oder Rotation getweent werden.

ActionScript

Variablen

Besonders in ActionScript gibt es viele Ansatzpunkte zur Optimierung. Leider geht das oftmals auf Kosten der Übersichtlichkeit und widerspricht so den ActionScript Conventions. Dennoch gibt es immer wieder Anwendungen, die auch den letzten Trick brauchen, um eine annehmbare Bildrate zu erreichen.

Allein um ins Innere unserer Programmiersprache zu sehen, sind die folgenden Tests schon interessant. Das folgende Script haben wir zur Messung verwendet:

```
function TestClass () {this.func1=function () {
      j++;
   };
}
_root.onLoad=function () {
   testObj=new TestClass();
   j=1;
}
_root.onEnterFrame=function () {
   fps++;
   for (i=0;i<500;i++) {
//    j++;
//    j=j+1;
      testObj.func1();
   }
}
function timer () {
   trace (fps);
   fps=0;
}
setInterval (timer,1000);
```

Die Ergebnisse waren erstaunlich. Während die leere Schleife eine Framerate von 113 Bildern pro Sekunde erreichte, waren es mit `j++`; 77 bps und mit `j=j+1`; 72 bps. Das letzte Ergebnis war übrigens bei allen Operatoren gleich, woraus wir folgern, dass z. B. eine Division nicht langsamer als eine Addition ist.

Sage und schreibe nur magere 32 Bilder pro Sekunde wurden beim Aufruf aus einer Funktion erreicht.

Kurze Zuweisungen

Wo wir schon bei Berechnungen waren: Da gibt es ja noch mehr Unterschiede. So kann man z. B. eine Operation über mehrere Zeilen hinweg ausführen oder versuchen, sie in möglichst einer Zeile abzuarbeiten. Letzteres ist fast doppelt so schnell:

```
_root.onEnterFrame=function () {
   fps++;
```

```
   a=2;
   b=8;
   c=16;
   e=0; //ergebnis
   for (j=0;j<500;j++) {
      e=(a*j+b*j+c*j)*j;//35 fps
//    aj=a*j;//20 fps
//    bj=b*j;
//    cj=c*j;
//    e=aj+bj+cj;
//    e*=j;
   }
}
function timer () {
   trace (fps);
   fps=0;
}
setInterval (timer,1000);
```

Lokale Variablen

Nun möchten wir die Verwendung von Klassenvariablen und lokalen Variablen testen sowie deren Abhängigkeit bei langen Variablennamen.

- `func1` – Die Klasse erreicht nur 19 bps.
- `func2` – Die lokalen Variablen erreichen immerhin 31 bps.
- `func3` – Mit längeren Namen kommen wir auf 5 bps weniger: 26 bps.

Hier sehen Sie das verwendete Testscript. Von den Variablen wurden jeweils neun Stück erstellt, was wir hier durch »...« abgekürzt haben:

```
function TestClass () {
   this.x1=1;
   ...
   this.x9=1;
   this.func1=function () {
      for (var i=0;i<500;i++) {
         this.x1++;
         ...
         this.x9++;
```

```
      }
   };

   this.func2=function () {
      //lokale Variablen anlegen
      var x1=this.x1;
      ...
      var x9=this.x9;
      for (var i=0;i<500;i++) {
         x1++;
         ...
         x9++;
      }
      //Variablen zurückschreiben
      this.x1=x1;
            ...
      this.x9=x9;
   };
   this.func3=function () {
      //lokale Variablen anlegen
var sososox1=this.x1;
...
var sososox9=this.x9;
      for (var i=0;i<500;i++) {
sososox1++;
...
sososox9++;
      }
      //Variablen zurückschreiben
      this.x1=sososox1;
      ...
      this.x9=sososox9;
   };
}
_root.onLoad=function () {
   testObj=new TestClass();
}
_root.onEnterFrame=function () {
```

```
   fps++;
// testObj.func1();
// testObj.func2();
   testObj.func3();
}
function timer () {
   trace (fps);
   fps=0;
}
setInterval (timer,1000);
```

Methoden

Nicht ganz so schwankend sah es bei den Methoden aus, wobei die generelle Anwendung von Methoden bereits sehr am Prozessor zehrt. So ließ sich messen, dass Prototype-Methoden genauso schnell sind wie Klassen-Methoden. Und auch die Länge des Methodennamens spielt dabei zum Glück keine messbare Rolle.

- ohne Methode: 113 fps
- mit `func1`:38 fps
- mit `func2`:38 fps
- mit `dasisteinlangermethodenname`: 38 fps

Das Testscript sah dabei folgendermaßen aus:

```
function TestClass () {
this.func1=function () {
   };
}
TestClass.prototype.func2=function () {
};
TestClass.prototype.dasisteinlangermethodenname=function () {
};
_root.onLoad=function () {
   testObj=new TestClass();
}
_root.onEnterFrame=function () {
   fps++;
   for (i=0;i<500;i++) {
```

```
//    testObj.func1();
//    testObj.func2();
//    testObj.dasisteinlangermethodenname();
   }
}

function timer () {
   trace (fps);
   fps=0;
}
setInterval (timer,1000);
```

Arrays

Bei der Array-Verarbeitung spielen zwei Geschwindigkeitsfaktoren eine Rolle. Zum einen müssen Sie auf die umschließende Kontrollstruktur achten, und zum anderen kommt es auf die Zuweisungsmethode an.

Beim Erstellen von Arrays muss man zunächst unterscheiden, ob es sich um ein zählbares Array handelt oder ob die Werte an das Ende des Arrays angehängt werden sollen. Ist ein eindeutiger Index vorhanden, dann ist die schnellste Methode das direkte Indizieren.

Bei 500 Zuweisungen erreicht die Methode 90 Bilder pro Sekunde:

```
a[i]=j;
```

Soll ein Wert an das Array angehängt werden, kann man die Länge des Arrays bestimmen und auf diese Position schreiben oder ein Element mit `PUSH` anhängen. Das Auslesen der Array-Länge braucht jedoch Rechenzeit. Dadurch kommt es wieder bei 500 Zuweisungen zu dem folgenden Ergebnis:

```
a[a.length]=j;//76 fps
a.push(j);  //90 fps
```

Beim Auslesen eines Arrays hängt die Verarbeitungsgeschwindigkeit lediglich von der Kontrollstruktur ab. Soll das ganze Array ausgelesen werden, kann man eine `for-in`-Schleife verwenden. Da hierbei kein Index mitgezählt werden muss, ist die Verarbeitung deutlich schneller als bei einer Zählschleife. Folgender Code veranschaulicht dies:

```
_root.onLoad=function () {
   a=[];
   for (var i=0;i<500;i++) {
      a[i]=i;
   }
}

_root.onEnterFrame=function () {
   fps++

// for (var i=0;i<500;i++) { //90 fps
//    j=a[i];
// }

   for (var i in a) { //122 fps
      j=a[i];
   }

}

function timer () {
   trace (fps);
   fps=0;
}
setInterval (timer,1000);
```

Objekte

Bei der Objektverarbeitung ergeben sich die deutlichsten Geschwindigkeitsunterschiede bei der Objekterstellung. Das Arbeiten mit Klassen ist sicherlich sehr komfortabel. Leider ist das Ableiten von Instanzen aber sehr langsam. Im folgenden Beispiel wird ein Objekt einmal von einer Klasse abgeleitet und einmal direkt erstellt. Der Geschwindigkeitsunterschied ist enorm:

```
function Class (a,b,c) {
   this.a=a;
   this.b=b;
   this.c=c;
```

```
}
_root.onEnterFrame=function () {
```

So schafft die folgende Schleife nur 27 Bilder pro Sekunde:

```
for (var i=0;i<500;i++) {
      c=new Class (0,0,0);
}
```

Während diese Schleife stolze 90 Bilder pro Sekunde auf unserem Testsystem erreicht:

```
for (var i=0;i<500;i++) { c={a:0, b:0, c:0 };
}
}
```

Loading

Preloader sind je nach Projekt notwendig, um einen ruckligen Seitenaufbau zu vermeiden. Bei streamenden Inhalten muss der Preloader nie 100% vorladen, sondern abhängig von der Übertragungsgeschwindigkeit nur den notwendigen Buffer.

Dateigrößenrelevante Inhalte, die nicht zwangsläufig, sondern nur je nach Interesse und Surfverhalten des Users aufgerufen werden, gehören in externe SWFs und werden erst bei Bedarf geladen. Das verringert die Ladezeit und den teuren Traffic.

Bei www.powerflasher.de z. B. wird erst nur der Weg zum Hauptmenü geladen, dann streamt die Seite, bis das Hauptmenü geladen ist, und hier werden erst wieder die Unterseiten geladen, wenn ich diese anwähle. In der Team-Sektion beispielsweise geht diese Verschachtelung mit den einzelnen Loadings der Vektorvideos noch weiter. Diese zwei bis fünf Sekunden langen Loadings werden mit kleinen Animationen verziert und schmerzen nicht so sehr wie eine Minute mehr Loading, um alle bereits zu cachen – nebenbei bemerkt: Bei HTML-Seiten macht man dies ja auch nicht. Es gilt so zwar viele externe SWF-Dateien zu verwalten, aber die Ladezeitoptimierung lohnt sich. Warum soll sich ein User 500 KB für die gesamte Internet-Seite herunterladen, wenn er eigentlich nur die Telefonnummer des Unternehmens finden möchte?

Sound

hasAudio?

Auch beim Sound ist es möglich, einiges zu optimieren, z. B. das Ladeverhalten. Hierfür lagert man einfach alle Sounds in externen SWF-Dateien. Ob diese Dateien dann geladen werden sollen oder nicht, wird zunächst davon abhängig gemacht, ob der User überhaupt ein Audiodevice hat. Einige PDAs z. B. besitzen so etwas nicht. Da Sounds nicht gerade klein sind, lohnt sich die Mühe hier – man bedingt das Loading einfach mit:

```
System.capabilities.hasAudio
```

GlobalSounds

Ein kleiner Tipp: Wir haben uns angewöhnt, Sound-Effekte (vor allem für Buttons) immer global anzulegen. So sind die Aufrufe sehr einfach, und auch Änderungen kann man »mal eben« durchführen, ohne gleich in den Flash-internen Waveeditor zu gehen. Ein Aufruf ist z. B.:

```
on (rollOver) { fx ('fx1', 25); }
on (release) { fx ('fx2', 50); }
```

LowQuality / HighQuality

Und noch eine Inspiration:

```
Movieclip.prototype.fx = function (which, volume) {

if (_root.highQuality && _root.playFX) {
   _level3.fx (which, volume);
} else if (_root.lowQuality && _root.playFX) {
   _level2.fx (which, volume);
}
```

Sobald `lowQuality` geladen ist, wird `highQuality` nachgeladen, und danach werden die Sounds aus der `highQuality`-Bibliothek verwendet. Es werden also zwei SWFs geladen: der stark komprimierte schlechte, aber kleine Sound wird in Level 2 geladen, der bessere in Level 3. Wenn Level 3 meldet, dass er geladen ist, ändert sich das Bedingungsergebnis, und die gute Soundqualität wird abgespielt.

Usability

Sind 99% aller Flash-Seiten schlecht?

Es liegt in der Geschichte des Computers, dass sich bei jeder neuen Entwicklung extreme Parteien bilden. So ist es mit PC und Mac, so ist es mit Windows und Linux, und es gibt zahlreiche weitere Beispiele, in denen bestimmte Typen nicht zusammenfinden. So erging und ergeht es auch der Internet-Technologie Flash: Während die einen darin schon lange eine »HTML-Ablösung« sehen, tun andere es als quirligen Unsinn ab.

Das historische Internet

Das Internet, 1962 ursprünglich als militärische Einrichtung gegründet, wurde über viele Jahre ausschließlich als Datenaustauschmedium genutzt. Erst 20 Jahre später sprach man erstmalig vom »Internet« – es wurde vorrangig von Universitäten als Informationsmedium genutzt. Weitere zehn Jahre später, erst 1992, wurde das World Wide Web entwickelt, das wir in seiner Basis bis heute nutzen. Mit der Einführung von Nameservern und HTML-1 wurde das Netz langsam benutzerfreundlich. Es war erstmals möglich, überhaupt kleine Formatierungen, Bilder und Hyperlinks in den vorher rein textbasierten Seiten zu nutzen.

Das Netz lebt

Seit diesem Tag fasziniert die Benutzer, was das Internet noch alles können wird. Mit dem Einfall des kommerziellen Nutzens und wenig später auch der privaten Nutzer öffnete sich das Netz für neue Ideen über reine Informationen hinaus. Diesen Tag verfluchen einige alteingeschworene Pioniere bis heute, aber damals erfolgte der Startschuss für eine rasante Entwicklung. Neben der Informationssuche wird seitdem verkauft, kommuniziert, gespielt, präsentiert etc. Neue Technologien schießen seitdem aus dem Boden: Video- und Voicestream, Chat und so auch 1996 *Futuresplash* – das von Macromedia gekauft wurde und seit der zweiten Version »Flash« heißt.

Flash

Flash machte es erstmals möglich, neben Text und Pixelbildern auch Vektoren im Internet darzustellen. Vektoren sind mathematisch gesehen Punkte und Kurven, mit denen Flächen und Linien beschrieben werden. Das bedeutet sehr kleine Dateien, die sogar streamen und das bei maxi-

mal darstellbarer Monitorqualität von frei skalierbaren Dateien. Darüber hinaus verfügt Flash über eine Zeitleiste, mit deren Hilfe man Animationen erstellen kann. Das war neben kleinen unruhigen GIF-Animationen innovativ.

Allein die Möglichkeit, auf einer freien Bühne zeichnen zu können und ohne Frames oder Tabellen irgendwelchen Einschränkungen zu unterliegen, überzeugte schnell.

Auch Sound brachte Flash seit der Version 3 mit. Ab Flash 4 begann man weiterhin mit kleinen Aktionen zu programmieren, seit Version 5 steht die interne Programmiersprache »ActionScript« dank ECMA 262-Standard sogar Javascript kaum mehr nach. Und heute in Version 6 überzeugt das Streamingformat mit internem Videocodec, Vektorscripting, Webcam- und Mikrofonsteuerung u.v.m. So kamen nach Intros, Produktpräsentationen, Fullflash-Seiten auch Spiele, Programme, WBTs, dynamischer Content und vieles mehr. Flash zieht in viele Bereiche ein und kreiert darüber hinaus sogar neue.

Jacob Nielsen

Nun hat der Usability-»Guru« Jacob Nielsen vor einiger Zeit einen sehr umstrittenen Artikel über Flash geschrieben. Er hat behauptet, dass 99 Prozent der Flash-Sites schlichtweg schlecht seien. So eine Aussage sei dahin gestellt, und ich möchte seine Ausführungen im Detail auch gar nicht weiter kommentieren. Da ich aber meist interviewt werde, wenn ein gegenteiliges Ergebnis gewünscht wird, möchte ich diese Thematik zu erörtern versuchen.

Da leider nicht jeder Leser die Artikel eines »Gurus« kritisch liest, möchte ich die Diskussion relativieren. So viel vorweg: Auch ich bin ein Freund von Usability und versuche, sie bestmöglich durchzusetzen. Das geht von Testemonials bis zu Augenbewegungsmessungen, die wir mit der RWTH Aachen an unseren Arbeiten durchführen.

Ich gebe Nielsen in vielen Punkten Recht. Hier werden aber alle Flash-Seiten in einen Topf geworfen, und die Zahl 99% wirkt dabei nahezu vernichtend. Daher werde ich mich im Folgenden bewusst »auf die andere Seite stellen«.

Der Begriff Usability

Vorab: Wer sich näher über das Thema informieren möchte, sollte http://webdev.khm.de/ ansurfen. Hier gibt es einen Artikel mit Schlachtrufen

wie »Ästhetik ist im Usability Engineering ein Nicht-Wort!« oder »Der ingenieurwissenschaftliche Hintergrund des 'Usability Engineering' macht auch vieles zunichte, was das Design mühsam etabliert hat«. Mehr unter: http://webdev.khm.de/themen/usability_brauchbarkeit.

Infotainment vs. Entertainment

Es gibt (gezwungenermaßen aus der Entwicklung des Internets) Menschen, die das Netz »nur« nutzen, um möglichst schnell und übersichtlich an Informationen zu kommen, wie es vor 20 Jahren ausschließlich möglich war. Jacob Nielsen ist ein sehr intelligenter Mann, der viel Richtiges in seinem Leben gesagt und damit das Internet positiv beeinflusst hat. Was seine letzte Abhandlung über Flash betrifft, sind wir aber geteilter Meinung.

Standard-Flash-Seiten unter Usability-Aspekten zu betrachten ist für mich gleichzusetzen mit einem Versuch, das Printmedium mit dem Fernsehen zu vergleichen. Es geht nicht!

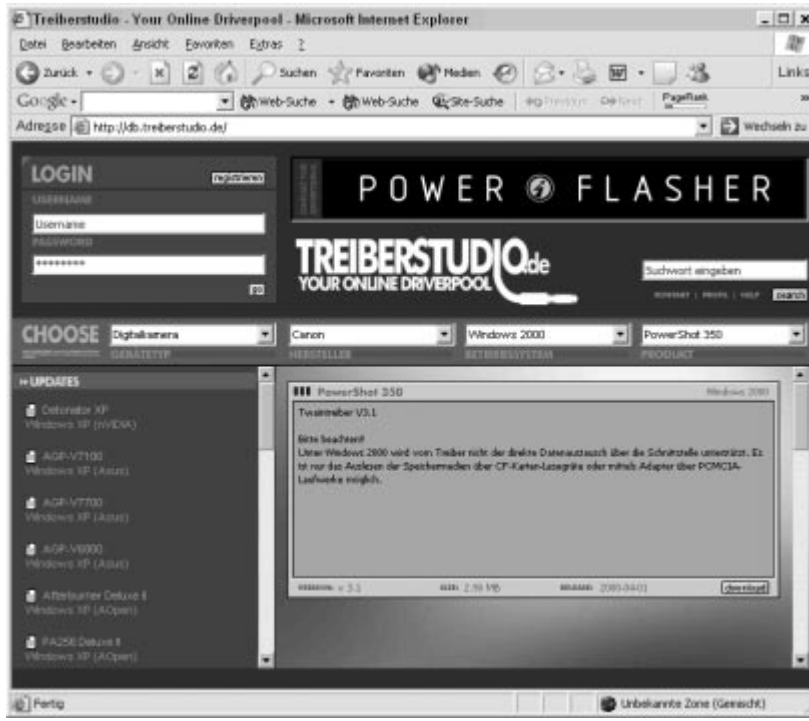
Ist Flash der HTML-Killer?

Natürlich ist es auch möglich, mit Flash gute Usability-Seiten zu erstellen – aber es ist nicht der ausschließliche Anspruch an Flash-Seiten. Damit will ich nicht sagen, dass Seiten, die mit Flash erstellt werden, nie den Usability-Richtlinien folgen müssen.

Macromedia spricht in Beispielen wie der Flash-Lösung in www.broadmoor.com von »OneScreenApplications« – ein komplettes Hotelreservierungssystem auf einer Bildschirmseite. Hier wird Flash ganz bewusst eingesetzt, um neue Usability intuitiv zu definieren. Aber auch einfache Informationsseiten wie www.sebo.de können mit Flash eine neue intuitivere Steuerung erhalten. Diese Seite beinhaltet mehr als 60 Unterseiten, und dennoch kann man sich in der Hierarchie nicht verlaufen.

Ein Beispiel für OneScreenApplications und gute Usability ist weiterhin www.treiberstudio.de. Mit einer intuitiven Navigation kann man eine komplexe Datenbank nach Treibern durchsuchen und das rasant schnell, sogar mit kleinen Animationen – FullFlash. Die Seite haben wir 2001 erstellt – sie wird sogar offline als Kauf-CD von Chip vertrieben.

Wenn es möglich und vor allem notwendig ist, dann gibt es sicher Möglichkeiten, gute Usability-Seiten zu erstellen – in dem Punkt gebe ich Nielsen recht.



▲ **Abbildung 12**
OneScreenApplication Treiberstudio – online wie offline

Ich gehe sogar einen Schritt weiter und behaupte, dass man dazu mit Flash durch intuitive Benutzerführung mehr Möglichkeiten hat. Man muss für sich nur klären, ob man eine multimediale Präsentation anstrebt.

Wenn man ausschließlich schnelle und einfache Informationen transportieren möchte – ohne weitere Ansprüche an Multimedia –, ist der Einsatz von Flash selbstverständlich zweifelhaft. Warum sollte Google Flash fahren? Man muss für so einen Vergleich also erst einmal den Informationsanspruch vom Entertainment-Anspruch trennen. Somit wird Flash HTML niemals ablösen. Es ist die Antwort auf eine neue Entwicklung des Internets.

Prestige

Flash ist für mich eine Reaktion auf den Zwang, sich im Internet darzustellen. Warum sollte z. B. eine computerfremde Firma wie McDonalds im Internet vertreten sein? Es wird kein E-Commerce genutzt, es gibt keinen

Bedarf an sonderlich aktuellen Informationen (»Wir haben kein BSE«). Natürlich kann man sich über die Firma informieren; der Hauptcontent ist doch aber Entertainment. Im Internet zu sein ist inzwischen eine Form von Prestige, so wie das Firmengebäude. Ein guter Internet-Auftritt kann dabei eine kleine Firma wie ein Weltunternehmen positionieren, anders herum aber auch ein Weltunternehmen wie einen Ein-Mann-Betrieb aussehen lassen.

Die echte Präsentation

Jede Firma muss also einen Weg finden, den Anspruch an Ihre Internet-Seite zu formulieren und dann bestmöglich zu realisieren. Wenn der Anspruch Entertainment ist, liegt die Lösung nahe, Flash einzusetzen. Aber auch wenn Entertainment nicht das Ziel ist, sondern »nur« die möglichst »echte« Darstellung und Präsentation der Firma im Netz, wird man häufig zu Flash greifen.

Flash gibt dem Designer freie Hand, er hat freie Schriftwahl, es gibt keine Darstellungsdifferenzen zwischen System und Browsern, außerdem kann man mit geschickten Animationen und dem sinnvollen Einsatz von Sound Emotionen erzeugen und somit das Image des Kunden beeinflussen.

Flash vs. HTML

Zudem: Eine in Flash nachgebaute Seite, die wie in HTML aussieht, wird in Flash nicht größer sein und wahrscheinlich noch geschickter streamen. Eventuell kann sogar durch Vektoren das eine oder andere KB gespart werden. Auch wenn diese in Flash nachgebaute HTML-Seite schlicht animiert wird, wird sie nicht viel größer sein. Wenn Flash nun aber in seiner vollen Dimension genutzt wird, wenn Sound, 3D-Animationen und solche Multimedia-Highlights hinzukommen, dann sieht die Sache anders aus. Weil Flash mehr kann (aus Multimedia-Sicht) als HTML, nutzt man i.d.R. auch mehr davon – und damit steigt auch die Dateigröße. Doch selbst hier sei gesagt: Gute Flash-Seiten müssen keinen Preloader von 1%, 2%, 3% etc. haben. Sie würden sich wie HTML-Seiten sukzessive elementweise aufbauen. Im Gegensatz zu HTML darf so etwas bei Flash-Seiten aber nicht sein. Gut angelegte Flash-Seiten laufen aber dank Streaming auch ohne Ladezeiten bzw. überbrücken die Ladezeit mit einem angenehmen Intro oder zeigen schon die bereits geladenen Seiten.

Jacob Nielsens Aussage müsste also – wenn überhaupt – lauten: »99 Prozent aller Flash-Seiten dienen nicht ausschließlich dem Informationstransport«. Und das liegt mit allen medialen Möglichkeiten in der Natur von Flash. Aber auch Emotionen, die mit guten Timing, Ton und Animation erzeugt werden, sind Informationen.

Eingefleischte Usability-Freaks wird man nie zu Flash bekehren. Das ist aber auch nicht nötig. Ich wollte nur zeigen, dass man solche Aussagen relativieren muss und dass es verschiedene Ansprüche an Internet-Seiten gibt. Sogar wir als spezialisierte Flash-Agentur verweisen Kunden an HTML-Agenturen. Flash ist durchaus nicht immer sinnvoll, auch HTML hat Stärken – nur andere.

Index

B

C

D

E

F

K

L

M

N

T

U

V

W

X

Z