

*Projektarbeit IP5*

# PGF-Plugin

---

---

Autoren:	Christian Züger, Nicolas Hofstetter
Bereich:	Technik
Studiengang:	Informatik
Ort und Datum:	Windisch, 16. Januar 2009
Betreuender Dozent:	Prof. C. Stamm
Auftraggeber:	xeraina GmbH, Zürich



## Zur Einleitung

### Zusammenfassung

Dieser Bericht behandelt die Projektarbeit zur Erstellung von Plugins für die Integration des PGF-Grafikformats in verschiedene Applikationen.

Der Bericht beginnt mit der Vorstellung der Aufgabenstellung und dem Pflichtenheft. Es folgen Berichte über die Realisierung der verschiedenen Plugins, wobei hier genauer auf die konkreten Ziele und spezifische Voraussetzungen eingegangen wird, sowie eine Übersicht über den organisatorischen Ablauf. Der Bericht schliesst mit einem Rückblick in Form einer persönlichen Reflexion über die aus der Arbeit gezogenen Schlüsse.

Der Bericht richtet sich an mit der Materie vertraute Leser. Insbesondere werden wir nicht näher auf die Hintergründe des PGF-Algorithmus eingehen und verweisen stattdessen auf [Sta02]<sup>1</sup>.

### Vorwort

Im Rahmen des Bachelor-Informatikstudiums an der Fachhochschule Nordwestschweiz FHNW sollen die Studierenden parallel zum Unterricht in jedem Semester eine Projektarbeit durchführen. In diesem fünften Semester haben wir diese erstmals in Einer- und Zweiergruppen durchgeführt.

Das Projekt wurde in unserem Fall von einer Firma der Fachhochschule angetragen. Kontaktperson für die Studierenden war dabei Herr Stamm als Projektbetreuer.

---

<sup>1</sup> C. Stamm. PGF - A new progressive file format for lossy and lossless image compression. Journal of WSCG, Vol. 10(2), 2002.

## Inhalt

Zur Einleitung .....	2
Zusammenfassung .....	2
Vorwort .....	2
Einleitung.....	5
Aufgabenstellung.....	5
Einarbeitung .....	5
Entwicklung .....	5
Dokumentation .....	5
Pflichtenheft .....	6
Situation .....	6
Auftrag.....	6
Ziele .....	6
Abgrenzung .....	6
Kosten.....	6
Zeitlicher Umfang .....	6
Werkzeuge.....	6
Kontakte .....	7
Gimp-Plugin .....	8
Vorbetrachtungen .....	8
Ziel .....	8
Architektur.....	8
Werkzeuge.....	9
Informationsquellen.....	9
Gimp .....	9
PGF.....	9
Implementierung.....	10
Laden eines PGF-Bildes.....	10
Speichern eines PGF-Bildes .....	11
Verschiedene Bildtypen.....	11
Optionale Features .....	12
Tests .....	12
Erreichte Ziele.....	13
Ausblick .....	13
Firefox-Plugin.....	14

Vorbetrachtungen .....	14
Ziel .....	14
Architektur.....	14
Unterschied zwischen Add-on und Plugin.....	14
Installation und Verwaltung .....	14
Werkzeuge.....	15
Informationsquellen .....	15
Firefox.....	15
PGF.....	16
Implementierung.....	16
Plugin-Instanz .....	16
Laden eines PGF-Bildes.....	16
Anzeigen eines PGF-Bildes .....	17
Verschiedene Bildtypen.....	17
Tests .....	17
Erreichte Ziele.....	17
Ausblick .....	17
Internet Explorer-Plugin .....	18
Situation zu Beginn des Projekts.....	18
Ziel .....	18
Vorgehen .....	18
Informationsquellen.....	18
Werkzeuge.....	18
Erkenntnisse und Umsetzung.....	18
Tests .....	21
Erreichte Ziele.....	21
Erlangtes Wissen .....	22
Ausblick .....	22
Reflexion.....	23
Zeitplanung.....	23
Christian Züger .....	23
Nicolas Hofstetter.....	24
Literaturverzeichnis.....	24
Anhang A: Zeitplan .....	25

## Einleitung

Die Projektarbeit beschäftigt sich mit der Entwicklung von Plugins zur Integration des PGF-Grafikformates in verschiedene Applikationen sowie der Signierung eines Plugins für den Microsoft Internet Explorer.

## Aufgabenstellung

Die Aufgabenstellung wurde in drei einzelne Bereiche aufgeteilt:

- Einarbeitung
- Entwicklung
- Dokumentation

## Einarbeitung

In der ersten Phase des Projekts wurden kommende Besprechungstermine mit dem Betreuer ausgehandelt. Weiter musste ein Zeitplan erstellt werden, welcher die wichtigsten Meilensteine definiert.

Ebenfalls zu Beginn mussten sich die Studierenden für zwei Plugintypen entscheiden, welche sie in den kommenden Wochen entwickeln sollten.

Pro Person wurde je eine Arbeitsstation zur Software-Entwicklung eingerichtet.

Diverse Literatur in verschiedener Form sollte zusammengetragen und studiert werden. Einerseits mit Informationen über die Host-Applikationen der gewählten Plugins, andererseits mit Angaben zur Signierung und Verbreitung des bereits existierenden ActiveX-Plugin.

PGF-Dateien mit unterschiedlichen Bildtypen wurden zur Verfügung gestellt und konnten zu Testzwecken benutzt werden.

## Entwicklung

Für die Entwicklung war vorgesehen, dass für zwei unterschiedliche Hostapplikationen jeweils ein PGF-Plugin erstellt wird. Die Plugins sollten funktionstüchtig und möglichst effizient sein. Spätere Nutzer sollten ein Plugin gratis über die Webseite von Xeraina<sup>2</sup> beziehen und es einfach installieren können.

Ein zweiter Teil in der Entwicklung sah vor, ein bereits entwickeltes PGF-Plugin für den Internet Explorer beziehungsweise dessen ActiveX-Control so zu erweitern, dass eine automatische Installation mit den Standardeinstellungen im IE möglich gemacht wird. In diesem Teil ging es generell um Signierung und Zertifikate.

## Dokumentation

Zum Abschluss des Projekts sind eine ausführliche schriftliche Dokumentation und eine Präsentation vorgesehen.

Die Dokumentation soll die Vorgehensweise, Konzepte, wichtigste Punkte der Implementierung sowie Testresultate enthalten. Ebenfalls sollen Zeitpläne und persönliche Erfahrungen festgehalten

---

<sup>2</sup> <http://www.xeraina.ch/pgf/pgfdownload.html>

werden. Das Zielpublikum sind Betreuer und zukünftige Studierende, welche in diesem Bereich weiterarbeiten wollen.

Die Schlusspräsentation verschafft den Anwesenden einen groben Überblick über die abgeschlossene Arbeit und hebt einige wichtige Details hervor. Eine anschließende Demonstration und Diskussion bietet die Möglichkeit, Fragen an das Projektteam zu tragen.

## **Pflichtenheft**

Das Pflichtenheft beschreibt die Ziele und Anforderungen an die Projektarbeit und dient damit als verbindliche Arbeitsgrundlage für die Studierenden. Es wurde zu Beginn des Semesters mit dem betreuenden Dozenten besprochen und unterzeichnet.

## **Situation**

Das Bildformat PGF existiert zwar schon länger, wird aber erst von sehr wenigen Software-Produkten unterstützt. Die Anfrage nach Unterstützung des Bildformats in nicht-kommerziellen und kommerziellen Anwendungen ist aber vorhanden. Durch die Projektarbeit "PGF-Plugins" sollen diese Anfragen umgesetzt werden.

## **Auftrag**

Der Auftrag ist im Abschnitt Aufgabenstellung des vorherigen Kapitels definiert.

## **Ziele**

Ziel der Projektarbeit ist die Lösung der in der Aufgabenstellung beschriebenen Aufgaben. Darüber hinaus soll eine Dokumentation erstellt werden, welche die Herangehensweise festhält und diese analysiert.

## **Abgrenzung**

Unter Zustimmung des Betreuers wurde vereinbart, dass die zu entwickelnden Plugins vorerst nur für das Betriebssystem Microsoft Windows geschrieben werden. Weitere Betriebssysteme (z.B. Linux, MacOS-X) sind optional.

## **Kosten**

Zur Durchführung der Diplomarbeit ist mit keinen zusätzlichen Kosten zu rechnen. Die notwendigen Werkzeuge werden von der Schule zur Verfügung gestellt oder können kostenlos bezogen werden.

## **Zeitlicher Umfang**

Das Projekt wird neben dem regulären Schulunterricht von den Studierenden bearbeitet und dauert das gesamte fünfte Semester über.

Es begann mit dem ersten Treffen zwischen dem betreuenden Dozenten und den Studierenden am 17. September 2008 und endet am 16. Januar 2009 mit der Präsentation des Projekts zum Ende des Semesters.

## **Werkzeuge**

Als Entwicklungswerkzeug kommt Microsoft Visual Studio als Referenzprodukt zur Arbeit mit C++ zum Einsatz. Den Studierenden stand weitere unterstützende Software zur Verfügung, darunter eine

vom betreuenden Dozenten zur Verfügung gestellte Software zum Anzeigen und der einfachen Manipulation von PGF-Dateien.

## Kontakte

Nicolas Hofstetter	nicolas.hofstetter@students.fhnw.ch	Studierender
Christian Züger	christian.zueger1@students.fhnw.ch	Studierender
Christoph Stamm	christoph.stamm@fhnw.ch	Betreuer

<http://web.fhnw.ch/technik/projekte/i/herbst09/HofstetterZueger/index.html>

## Gimp-Plugin

### Vorbetrachtungen

Gimp ist ein frei erhältlicher Grafikeditor, der eine Vielzahl von Grafikformaten unterstützt und umfangreiche Bildbearbeitungswerkzeuge zur Verfügung stellt. Die Software ist für Windows sowie Linux und Mac OS erhältlich und allgemein als Alternative zu kostenpflichtigen Grafikprogrammen akzeptiert.

Damit wurden bereits die wichtigsten Gründe, weshalb wir uns für ein Plugin unter Gimp entschieden haben, zusammengefasst: Es ist ein mächtiges, weit verbreitetes Grafikprogramm, das ausserdem kostenfrei erhältlich ist; dieses ist eine wichtige Voraussetzung für eine weitere Verbreitung des Plugins selbst. Daneben sind für Gimp bereits eine Vielzahl unterschiedlicher Plugins erhältlich, was darauf schliessen lässt, dass das Programm für eine Erweiterung durch Plugins durchaus geeignet ist.

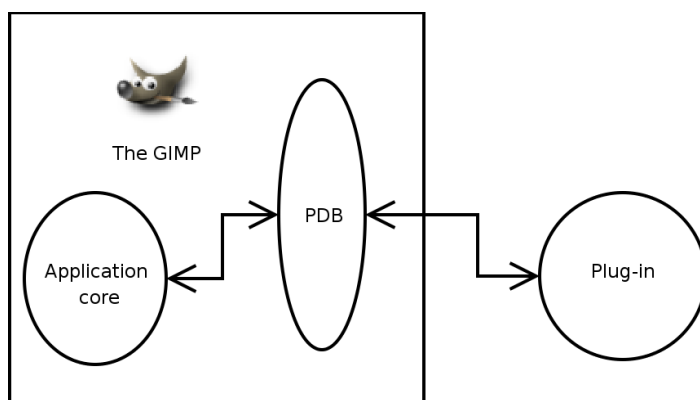
### Ziel

Ziel ist die Entwicklung eines Plugins zum Umgang mit dem PGF-Grafikformat. Dieses soll insbesondere die folgenden Funktionen erfüllen:

- Einlesen einer Bilddatei und deren Darstellung
- Speichern einer Bilddatei.

### Architektur

Gimp ist darauf ausgerichtet, seine Funktionalität durch in C oder C++ geschriebene Plugins zu erweitern. Dabei werden die Plugins intern wie Scripts behandelt, die über die so genannte Procedural Database (PDB) von der Hauptapplikation aufgerufen werden.



Die Gimp-Plugin-Architektur, von <http://developer.gimp.org>.

Ein Plugin ist ein in C / C++-Code geschriebenes Programm, welches als executable kompiliert und in einen entsprechenden Plugins-Unterverzeichnis der Gimp-Installation kopiert wird.

Das Plugin basiert dabei auf einer vorgeschriebenen Vorlage, welche die wichtigsten Funktionen definiert, die als Einstiegspunkt zum Starten des Plugins aus der Applikation heraus dienen. Die gesamte Funktionalität basiert dabei auf der *libgimp* genannten Basis-Bibliothek, welche sämtliche Gimp-spezifischen Funktionen enthält. Daneben ist es möglich, beliebigen C / C++-Code einzusetzen.

## Werkzeuge

Gimp selbst stellt für die Entwicklung entsprechende Übersetzungs- und Installationswerkzeuge für die Erstellung eigener Plugins zur Verfügung; nach umfangreichen Recherchen stellte sich jedoch heraus, dass diese einzig für Unix-Betriebssysteme zur Verfügung stehen. Da wir uns für eine Entwicklung unter Microsoft Windows entschieden haben, mussten wir demnach Alternativen suchen. Schliesslich haben wir zur Entwicklung unserer Plugins Microsoft Visual Studio eingesetzt, dabei jedoch die nötigen Bibliotheken von Gimp übernommen und die Entwicklungsumgebung entsprechend konfiguriert, sodass diese zur Erzeugung der von Gimp erwarteten executables befähigt wurde.

## Informationsquellen

Als recht schwierig erwies sich die Beschaffung von Quellen an Hintergrundwissen, und dieses auf zweierlei Seiten: Zum einen fehlte es an einer tiefer gehenden, praktisch orientierten Dokumentation der Gimp-Basisbibliothek *libgimp* sowie der Plugin-Architektur im Allgemeinen; andererseits warf auch die Verwendung der PGF-Bibliothek immer wieder Fragen auf.

## Gimp

Die Dokumentation zu Gimp beschränkte sich im Wesentlichen auf eine sehr oberflächliche API von *libgimp* sowie ein kurzes einführendes Tutorial zur Erstellung eines Plugins. Insbesondere fehlte ein umfangreicheres Tutorial zum konkreten Umgang mit Bild-Formaten sowie zur Anbindung zusätzlicher Funktionalität an die Standarddialoge von Gimp (Laden / Speichern).

Da das einzige zur Verfügung stehende Tutorial den Anforderungen des Projekts nicht genügte und das Zusammensuchen von ganzen Funktionsblöcken allein aus der API heraus nicht praktikabel war, musste nach einer konkreten, möglichst simplen Vorlage in Form eines Beispielplugins, welches das Laden und Speichern eines neuen Dateiformats beinhaltete, gesucht werden. Es war überraschend, nur ein einziges derartiges Programm inklusive Quellcode zu finden. Auch erwies sich dessen Anpassung und Integration in die Entwicklungsumgebung als äusserst zeitaufwendig. Es war offensichtlich, dass dieses C++-Programm zur Kompilierung unter Linux geschrieben worden ist und entsprechend eng mit Betriebssystem-spezifischen Bibliotheken verzahnt war.

Mangels Alternativen wurde dieses Programm dennoch als Basis für die Implementierung verwendet. Nachdem das Programm soweit angepasst war, dass es sich auch auf dem Entwicklungssystem kompilieren liess, erwies es sich allerdings als sehr geeignet, da es im Wesentlichen alle Funktionalität abdeckte, die anschliessend zur Verwendung mit der PGF-Bibliothek portiert werden konnte.

## PGF

Bezüglich der PGF-spezifischen Dokumentation wurden uns die Quelldateien der Basis-Bibliothek inklusive Quellcode-Dokumentation sowie die Quelldateien eines Kommandozeilenprogrammes zum Öffnen und Speichern von PGF-Dateien zur Verfügung gestellt. Besonderes Letzteres half ungemein, sich einen ersten Überblick über den Ablauf eines Lade- und Speichervorganges zu verschaffen, sowie über den grundlegenden Aufbau der Bibliothek. Der betreuende Dozent selbst stand dabei für weiter gehende Fragen zur Verfügung.

Im Verlauf der Entwicklung kamen jedoch viele Detailfragen auf; obwohl die uns übergebenen Quelldateien einen guten Überblick gaben, behandelte die Dokumentation doch nicht alles sehr detailliert, als Beispiel sei die Implementierung der Farbpalette genannt. Dies verursachte bald eine

recht unangenehme Situation, indem jeweils die Wahl bestand, mangels alternativer Informationsquellen entweder direkt mit dem Dozenten Kontakt aufzunehmen, oder aber es nach einem Try-and-Error-Prinzip selbst zu versuchen. In vielen Fällen erwies sich jedoch Letzteres als zeitlich nicht machbar, was schliesslich dazu führte, dass die Hilfe des Betreuers öfter als geplant in Anspruch genommen werden musste oder die Arbeit über eine gewisse Zeit still stand, weil wichtige Informationen fehlten.

## Implementierung

Im Folgenden wird auf den Ablauf der Implementierung eingegangen. Dabei begann die Implementierung mit der Hauptfunktionalität des Ladens eines Bildes, gefolgt vom Speichern und dem begleitenden Einbringen weiterer optionaler Funktionalität.

Beim Lade- wie auch beim Speichervorgang diente der Ablauf, wie er bei dem Kommandozeilen-Beispielprogramm implementiert worden ist, als Orientierungshilfe. Der springende Punkt war es nun, an den entsprechenden Codestellen die Funktionen der PGF-Bibliothek mit jenen der Gimp-Bibliothek zusammen arbeiten zu lassen. Auf Seiten der Gimp-Bibliotheksaufrufe konnte der Ablauf grob an den durch das Vorlagen-Plugin gegebenen Ablauf angelehnt werden. Hier wurden die entsprechenden Aufrufe der wichtigsten Funktionen bereits mit den vom Gimp-Framework verwendeten Callback-Funktionen verbunden, sodass der Programmablauf als solcher schon vorgegeben war.

Die grösste Schwierigkeit bestand in der "Feinabstimmung", indem die genaue Art, Anzahl und Reihenfolge der von den verschiedenen Funktionen erwarteten Argumente herausgefunden werden musste. Diese Information war oft aus der vorhandenen Dokumentation nicht ersichtlich und wurde zu einer eher mühsamen Probierarbeit.

### Laden eines PGF-Bildes

Das Laden eines Bildes wurde im Vorlagen-Plugin in einer einzigen Funktion behandelt, welche von der Start-Methode des Plugins aus ausgeführt wird. Entsprechend musste der Code dieser Funktion durch die eigene Implementierung ersetzt werden.

Für das Einlesen einer im PGF-Format kodierte Datei sowie deren Dekomprimierung stellte die PGF-Bibliothek entsprechende Funktionen bereit, welche zu Beginn des Programms aufgerufen werden.

Die dekomprimierten Bilddaten können in ein Buffer-Array abgelegt werden. Überraschenderweise konnte an dieser Stelle über einen entsprechenden Parameter bereits die Codierung entsprechend der Farbkanäle angegeben werden. Es ist in der Tat so, dass Gimp die Farbkanäle anders "sortiert", als dies von der PGF-Bibliothek mit Standard-Argumenten durchgeführt wird. In den ersten Versuchen führte dies zu mitunter recht kunterbunten Bildern. Es lief schliesslich darauf hinaus, durch reines Ausprobieren die entsprechende Sortierung der Farbkanäle herauszufinden, da die Gimp-Dokumentation sich über diesen Punkt vollständig ausschwig. Je nach Bildtyp (beispielsweise RGB oder mit zusätzlichem Alpha-Kanal) war diese Sortierung verschieden.

Beim Speichern des Bildes in einem Buffer-Array war es ausserdem entscheidend, die korrekte Anzahl Pixel pro Bildzeile herauszufinden. In ersten gescheiterten Versuchen führte eine zu geringe Anzahl Pixel zu der kuriosen Ausgabe, die ein Bild zeigte, welches sich deutlich verkleinert und mit separierten Farbkanälen präsentierte. Hier lag die grosse Schwierigkeit darin, festzustellen, dass

tatsächlich eine fehlerhafte Anzahl Pixel pro Zeile die Fehlerursache war; über längere Zeit wurde ein weiteres Problem in der Zusammenstellung der Farbkanäle vermutet.

Nachdem das Bild eingeladen und in einem für Gimp nutzbaren Format in einem Buffer lag, folgte nun eine Reihe Aufrufe der Gimp-Bibliothek, welche eine neue Arbeitsfläche erzeugten und das Bild auf dieser einbetteten. Einige Befehle wurden bereits von dem originalen Plugin eingesetzt, welches als Vorlage verwendet wurde, andere mussten aus der Gimp-API herausgesucht werden. Zum Beispiel erkannte Gimp zwar automatisch die Grösse des Bildes, eine Angabe, welche in der Titelleiste ausgegeben wird, nicht jedoch den Bildtyp und den Dateinamen, zwei weitere in der Titelleiste angegebene Informationen. Diese mussten Gimp daher zusätzlich übergeben werden.

### Speichern eines PGF-Bildes

Wie auch beim Laden eines Bildes bestand hier aus dem Vorlagen-Plugin die entsprechende Funktion bereits, deren Aufruf ebenso bereits implementiert war. Die Implementierung der Funktion gestaltete sich dennoch schwieriger als eine einfache "Umkehrung" des Lade-Prozesses, da hierbei teilweise andere Bibliotheks-Aufrufe zur Anwendung kamen.

Verschiedene Gimp-Funktionen ermöglichten zunächst das Auslesen diverser Bild-Informationen wie die Grösse und den Bildtyp.

Auf Seiten der PGF-Bibliothek wurden wiederum Funktionen verwendet, um die von Gimp ausgelesenen rohen Bilddaten zu komprimieren und einem Dateistream zu übergeben.

### Verschiedene Bildtypen

Der Lade- und Speichervorgang konnte für die Bildtypen RGB, RBGA sowie Graustufen implementiert werden. Für Bilder mit Transparenz (RGBA) mussten wie oben erwähnt die Farbkanäle angepasst werden; Graustufenbilder konnten indes wie RGB-Bilder behandelt werden.

Schwierigkeiten bereiteten vor allem die zum Schluss der Programmierphase behandelten Bitmap- und farbindizierten Bilder. Bei Bitmap-Bildern handelt es sich um stark komprimierte reine Schwarzweiss-Bilder, während farbindizierten Bildern eine benutzerdefinierte Farbpalette beigelegt wird, welche die standardmässige RGB-Farbpalette ersetzt. In der Tat stellten sich die Probleme, welche diese Bildtypen bereiteten, als letztlich unlösbar heraus.

Grössere Fortschritte wurden noch bei den farbindizierten Bildern erzielt. Hier ist es gelungen, die dem Bild beigelegte Farbtabelle tatsächlich zu laden beziehungsweise im Verlauf des Speichervorganges eine eigene zu erzeugen. Dabei war zunächst das Problem zu lösen, dass das originale Gimp-Plugin, auf das die Implementierung sich stützte, ebenso die Behandlung farbindizierter Bilder vorsah, die Farbpalette jedoch als externe Datei mitgab; der vom Dozenten zur Verfügung gestellte PGF-Viewer las die Farbpalette jedoch aus der Bilddatei selbst aus. Um die Kompatibilität zu diesem Programm zu erhalten und die PGF-Spezifikation zu erfüllen, musste daher dieses Vorgehen nachgebildet werden. Die Implementierung scheiterte schliesslich an der erneut problematischen Inkompatibilität der Gimp-Farbtabelle mit jenen, welche die PGF-Bibliothek erwartete. Dieses resultierte darin, dass das Plugin nun zwar farbindizierte Bilder speichert und lädt, die Farben jedoch falsch dargestellt werden, wobei die Darstellung einerseits in Gimp selbst von einem gespeicherten zu einem neu geladenen Bild abwich, andererseits auch bei einem Vergleich eines in Gimp und im PGF-Viewer geladenen Bildes.

Erneut fand sich keine Informationsquelle, welcher Details zur Farbverwaltung in Gimp zu entnehmen waren. Da diese Arbeit in der unterrichtsfreien Zeit über die Weihnachtsfeiertage stattfand, war es nicht möglich, den betreuenden Dozenten zu kontaktieren. Aufgrund der durch andere schulische Pflichten begrenzten Zeit wurde schliesslich davon abgesehen, durch Try-and-Error-Verfahren ziellos an der Farbpalette herumzubasteln und stattdessen eine Meldung programmiert, welche beim Laden und Speichern eines farbindizierten Bildes darauf hinweist, dass die Farben möglicherweise verfälscht sind.

Noch weniger erfolgreich verliefen die Bemühungen um eine Integration des Bitmap-Formates. Diesbezügliche Versuche führten aus weitestgehend unerklärlichen Gründen jeweils zu einem Absturz des Plugins. Auch dieses Vorgehen scheiterte mangels detaillierter Informationen zum Umgang mit Bilddaten am Finden der Ursache des Problems. Es war dann auch über die Weihnachtsfeiertage, als die Entscheidung getroffen werden musste, diese Funktionalität auszuklammern und durch die Ausgabe einer Fehlermeldung den Absturz des Plugins zu unterbinden.

### Optionale Features

Es gab eine Reihe weiterer Features, welche im Pflichtenheft nicht explizit erwähnt wurden, deren Realisierung mir jedoch wichtig und interessant schien. In erster Linie war das die Anzeige eines "Vorschaubildes" im Lade-Dialog sowie allgemein die weitere Anpassung der Dialoge, beispielsweise zur Angabe der gewünschten Kompressionsqualität beim Speichern.

Tatsächlich war es so, dass durch die sehr schwierige und zeitaufwendige Implementierung der Grundfunktionalität kaum noch Zeit für die Programmierung zusätzlicher Features blieb. Über meine Vorstellungen zu einer weiter gehenden Implementierung berichtet auch der Abschnitt Ausblick zum Schluss des Berichts.

Die Anzeige eines Vorschaubildes wurde rudimentär bereits von dem ursprünglichen Plugin zur Verfügung gestellt. Diese wurde einmal so übernommen, später fand sich jedoch keine Zeit mehr, um diese gemäss eigener Vorstellungen anzupassen.

Die allgemeine Anpassung des Lade- und Speicherdialoges stellte sich als äusserst kompliziert heraus. Zwar verfügte bereits das Vorlagen-Plugin über entsprechenden Code, dieser führte jedoch jedes Mal zu einem Absturz, sodass der Code schliesslich entfernt werden musste, um das Plugin überhaupt zum Laufen zu bringen.

Aufgrund seiner Komplexität war es allerdings nicht möglich, den Code später Schritt für Schritt wieder einzubauen. Er bestand aus einer ganzen Reihe von Funktionen, die sich gegenseitig voraussetzten und teilweise externe Funktionen aus der Gimp-Bibliothek verwendeten. Auch die Gimp-API als Sammlung von Bibliotheksaufrufen konnte hier nicht weiterhelfen; die einzige Hoffnung wäre ein weiteres Beispielplugin oder im günstigsten Fall ein Tutorial zum Umgang mit GUI-Elementen in Gimp gewesen, solches war jedoch nicht auffindbar. Ich sah mich gezwungen, auch diesen Teil der Implementierung vollständig zu streichen.

### Tests

Im Verlauf der Implementierung wurde die Funktionalität laufend getestet. Dabei wurde ein Testset aus PGF-Dateien genutzt, welches vom Dozenten zur Verfügung gestellt wurde. Später konnten mit Hilfe des Gimp-Plugins eigene PGF-Dateien aus in anderen Formaten vorliegenden Bildern erstellt und zum Testen im Original-PGF-Viewer geöffnet werden.

## Erreichte Ziele

An dieser Stelle werde ich die im Abschnitt zur Implementierung vorgestellten Ergebnisse zusammentragen. Das primäre Ziel des Projekts war ein Plugin für Gimp, das in der Lage ist, PGF-Dateien zu laden und anzuzeigen, sowie zu speichern. Dieses Ziel konnte im Kern erreicht werden.

Das Plugin liegt als .exe-Datei vor, welche in ein entsprechendes Plugin-Verzeichnis innerhalb der Gimp-Installation kopiert werden kann, um seine Funktionalität zu nutzen. Der Lade- und Speichervorgang für das PGF-Format wird in die Gimp-Standarddialoge integriert.

Im Detail wird die folgende Funktionalität von der finalen Plugin-Version unterstützt:

- Einlesen einer Bilddatei und deren Darstellung in den Typen RGB, RGBA, Graustufen.
- Speichern einer Bilddatei in den Typen RGB, RGBA, Graustufen.
- Anzeige eines Vorschaubildes.

Nicht oder nur eingeschränkt erreicht werden konnte wünschenswerte Funktionalität in den folgenden Bereichen:

- Der Umgang mit farbindizierten Bildern funktioniert nur eingeschränkt: Die Farbtöne werden nicht korrekt wiedergegeben.
- Das Bildformat Bitmap wird nicht unterstützt.
- Der Lade- und Speicherdialog sieht keine speziellen Optionen zum Umgang mit PGF-Bildern vor (z.B. Einstellung der Anzahl zu ladender / speichernder Layer).

## Ausblick

Die Zeit für die Implementierung der Plugin-Funktionalität im Rahmen des Projekts ist nun abgelaufen. Jedoch konnte die grundlegende Funktionalität des Plugins implementiert werden.

Genauso wichtig erscheint mir allerdings, dass dieses durchaus als Basis für künftige Weiterentwicklungen genutzt werden kann. Gerade ohne den kritischen Faktor Zeit, auf dessen Einfluss ich im Kapitel zur Reflexion noch eingehen werde, wird es für eine fachlich versierte Person kein grösseres Problem darstellen, die noch nicht implementierten Funktionen hinzuzufügen.

Gerade die Unterstützung sämtlicher Bildtypen dürfte unter der Voraussetzung eines Grundwissens über die internen Abläufe der PGF-Bibliothek nicht weiter problematisch sein; allein an dem fehlenden Wissen hierüber bin ich gescheitert. Was die Anpassung des GUIs zur Verwendung mit dem PGF-Format angeht, so wird es unumgänglich sein, dass sich ein Programmierer zunächst intensiv mit der Gimp-Bibliothek auseinandersetzt oder sich gleich an einen erfahrenen Gimp-Programmierer wendet, der dieses möglicherweise innert kurzer Zeit realisieren kann.

## Firefox-Plugin

### Vorbetrachtungen

Firefox ist ein Webbrowser und liegt aktuell in der Version 3.0.5 vor. Der Browser, der direkt nach dem Microsoft Internet Explorer auf Platz zwei der meistbenutzten Webbrowser gelistet wird, stammt aus dem Hause Mozilla<sup>3</sup>. Er ist Open-Source und unterstützt unter anderem die Betriebssysteme Microsoft Windows, Mac OS X und Linux.

Der Entscheid, ein Plugin für Firefox zu schreiben, war schnell gefällt: diesem Browser gehört die Zukunft und deshalb soll er auch das neue Bildformat PGF unterstützen können. Weiter konnte man von diesem Hostprogramm als Open-Source-Projekt viele Beispiel-Implementationen und Tutorials erwarten.

### Ziel

Als Ziel steht die Entwicklung eines Firefox Plugins, welches PGF-Bilder im Browser anzeigen kann. Die Vorteile und speziellen Eigenschaften des PGF (Bildtypen, einzelne Level laden) sollen in der Implementation beachtet und miteinbezogen werden.

### Architektur

#### Unterschied zwischen Add-on und Plugin

Um Verwechslungen vorzubeugen sei hier der Unterschied zwischen Firefox Add-ons<sup>4</sup> und Plugins<sup>5</sup> erklärt. Obwohl diese beiden Arten von Erweiterungen verschieden sind, werden sie oft im selben Atemzug erwähnt und sorgen so für Verwirrung.

Ein Plugin hilft dem Browser dabei, bestimmte Funktionen auszuführen. Dies kann z.B. die Anzeige eines Java-Applets, einer Multimedia-Datei oder – wie in diesem Fall – eines bestimmten Bildformats sein. Weit verbreitet sind unter anderem Plugins für den Acrobat Reader, den Flash Player von Adobe, Quicktime, Realplayer oder Windows Media Player.

Add-ons hingegen greifen auf die bestehenden Firefox-Funktionen zu, erweitern diese und sorgen so für einen personalisierten Webbrowser. So können z.B. Anzeigen und Farben verändert, Suchwerkzeuge hinzugefügt oder Mausgesten programmiert werden.

#### Installation und Verwaltung

Voraussetzung ist die bereits vorhandene Installation des Mozilla Firefox in der minimalen Version 3.0. Für ältere Versionen wurden keine Tests durchgeführt.

Der Installationsprozess des Plugins selbst ist sehr einfach. Es muss lediglich die heruntergeladene Plugin-dll in das Mozilla-Plugin-Verzeichnis (z.B. C:\Program Files\Mozilla Firefox\plugins) kopiert werden. Das Plugin erkennt ab diesem Zeitpunkt die in ein HTML-File eingebundenen PGF-Bilder und stellt diese dar.

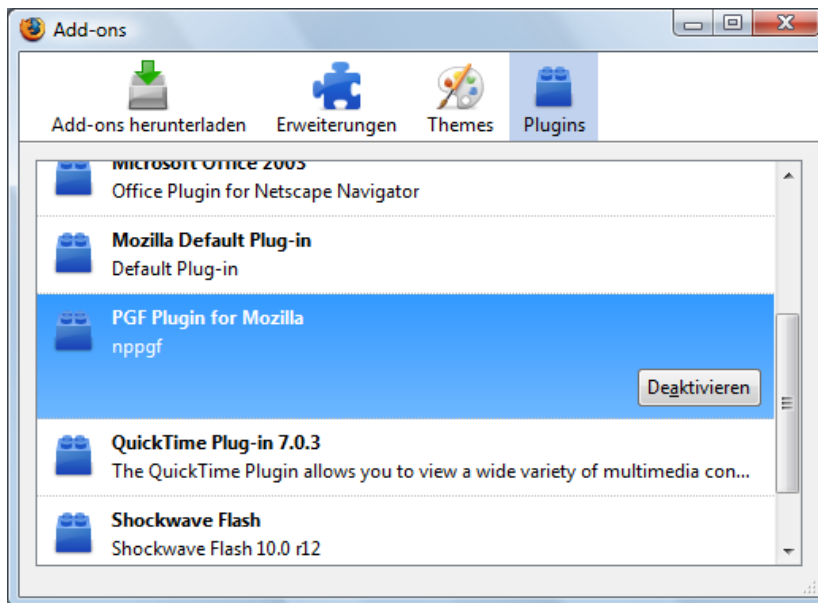
Eine erfolgreiche Installation kann im Firefox über Extras -> Add-ons kontrolliert werden. Ein kleines Fenster wird geöffnet und unter der Rubrik Plugins kann in der Liste das neue Plugin gefunden werden.

---

<sup>3</sup> <http://www.mozilla.org>

<sup>4</sup> <https://addons.mozilla.org/de/firefox/>

<sup>5</sup> <https://addons.mozilla.org/de/firefox/browse/type:7>



In diesem Fenster kann das Plugin auch deaktiviert werden. Dazu reichen ein Klick auf den Deaktivieren-Button und ein Neustart des Browsers. Das erneute Aktivieren erfolgt über dasselbe Menü.

Die Deinstallation des Plugins erfolgt durch das Löschen der dll-Datei im Mozilla-Plugin-Verzeichnis.

Informationen über installierte Plugins erfährt der Benutzer auch durch den Aufruf "about:plugins" in der Adresszeile.

## Werkzeuge

Das Plugin wurde unter Microsoft Windows Vista entwickelt.

Wie im Kapitel Pflichtenheft erwähnt, wurde Microsoft Visual Studio 2008 für die Entwicklung des Plugins benutzt. In dieser Umgebung war es ebenfalls möglich, intensives Debugging zu betreiben, was die Arbeit im Allgemeinen und vor allem das Finden von Fehlern enorm erleichtert hat.

Um mit PGF arbeiten zu können musste im Projekt die PGF-Library referenziert werden.

Damit die kompilierte Plugin-DLL direkt in den Mozilla-Programmordner kopiert werden konnte, musste das Visual Studio mit Administrator-Rechten gestartet werden.

## Informationsquellen

### Firefox

Informationen über Mozilla-Plugins sind im Gegensatz zu den oben beschriebenen Add-ons nur schwer zu finden. Nach längerer Suche konnte aber eine gute Anleitung<sup>6</sup> gefunden werden, auf der das neu entwickelte Plugin nun aufbaut. So kamen erste erfolgreiche Kompilerversuche zu Stande und mittels Manipulationen am Beispielplugin wurden erste Erfahrungen gesammelt.

<sup>6</sup> <http://blog.apijunkie.com/APIJunkie/blog/post/2008/09/Building-Firefox-Plugins-using-Visual-Studio.aspx>

Neben der genannten Webseite kam noch die Gecko Plugin API Reference<sup>7</sup> zum Einsatz, welche die gesamten Funktionalitäten eines Plugins beschreibt.

## PGF

Die Informationen über PGF konnten auf [www.xeraina.ch](http://www.xeraina.ch) gefunden, sowie vom Betreuer selbst bezogen werden.

Weitere Informationen zu PGF können im vorausgehenden GIMP-Abschnitt nachgelesen werden.

## Implementierung

Das bereits vorhandene Framework stellt alle allgemeinen Browser- und Plugin-Funktionen zur Verfügung. Das Basic-Beispielplugin, welches als Grundlage übernommen wurde, setzt sich aus den folgenden Dateien zusammen:

- np\_entry.cpp (Main Plugin Entry Point)
- npn\_gate.cpp (Browser Entry Points)
- npp\_gate.cpp (Plugin Entry Points)
- plugin.cpp (Implementation Plugin)

Die plugin.cpp war die einzige Datei, welche davon bearbeitet werden musste. Alle andern konnten übernommen werden.

## Plugin-Instanz

Als erstes muss vom Browser her eine Plugin-Instanz erzeugt werden, mit welcher später gearbeitet werden kann. Dass es sich beim anzuzeigenden Objekt um ein PDF-Bild handelt erkennt der Browser dank der "type"-Angabe im Object-Tag der HTML-Datei. Der Wert dieses Tags ist auf "image/pgf" gesetzt und verweist so auf das PGF-Plugin.

## Laden eines PGF-Bildes

Der Browser schickt die erkannte PGF-Datei nun als Stream zur Behandlung an das PGF-Plugin. Dies geschieht in der überschriebenen Write-Methode in plugin.cpp. In dieser Methode wird im mitgegebenen Stream nach dem Präfix "PGF" gesucht, welcher eine PGF-Datei identifiziert. Ist dies der Fall, wird der Stream zwischengespeichert und mit den folgenden Streams ergänzt.

Währenddem neue Streams eintreffen und das PGF-File vollständig eingelesen wird, kann mit den bereits erhaltenen Daten schon ein PGF-Objekt erzeugt werden. Dieses bietet – wenn es auch noch unvollständig ist – bereits jetzt wichtige Informationen an, welche aus dem Header gelesen werden können. Es sind dies z.B. Gesamtgrösse der Datei, Anzahl Level, Höhe und Breite des Bildes. So kann bereits zu Beginn der Übertragung genügend Speicher für das entstehende Bild alloziert werden.

Sobald der vollständige Stream eingelesen wurde, ruft der Browser die URLNotify-Methode auf. Diese wurde ebenfalls überschrieben und führt nun neu definierte Codezeilen aus. Erneut wird mittels übergebenem Stream ein PGF-Objekt erzeugt.

Als nächsten Schritt müssen die verschiedenen Bildtypen unterschieden werden, denn jeder Bildtyp braucht nun eine etwas andere Behandlung, bevor das fertig gestellte Bild an den Browser zurückgegeben werden kann. Dies ist insbesondere das Setzen einer ChannelMap (ordnet die Kanäle richtig) oder ein Parameter, der auf die Transparenz im Bild hinweist.

---

<sup>7</sup> [https://developer.mozilla.org/En/Gecko\\_Plugin\\_API\\_Reference](https://developer.mozilla.org/En/Gecko_Plugin_API_Reference)

Das Bild ist nun vollständig und kann im Browser angezeigt werden.

Für den Fall, dass mehrere PGF-Bilder auf einer HTML-Seite angezeigt werden müssen, werden die Streams voneinander unterschieden und die beschriebene Prozedur für jedes Bild einzeln durchgeführt.

### Anzeigen eines PGF-Bildes

Das Anzeigen und Aktualisieren des Bildes im Browser erfolgt nun wieder grösstenteils mit bereits vorhandenem Code. Der Aufruf geschieht mittels Callback, welcher das Bild in einem vordefinierten Platz im Browser zeichnet. Dies geschieht mit der Angabe der Plugin-Instanz und dem durch die Instanz referenzierten Bild.

### Verschiedene Bildtypen

Bis heute sind zwei verschiedene Bildtypen implementiert worden: RGB und RGBA. Bei beiden Typen musste eine ChannelMap für die richtige Reihenfolge der Kanäle eingesetzt werden. Zusätzlich musste beim RGBA ein Flag gesetzt werden, welches auf die enthaltene Transparenz hinweist.

### Tests

Die PGF-Testdateien wurden, wie im Abschnitt unter GIMP erklärt, vom Dozenten zur Verfügung gestellt. Die Tests alle wurden im Mozilla Firefox 3.0 durchgeführt, ältere Versionen wurden nicht berücksichtigt.

### Erreichte Ziele

Es ist möglich, ein PGF-Plugin sehr einfach zu installieren, aktivieren, deaktivieren oder zu löschen. Ein PGF-Bild wird – sofern der richtige Bildtyp vorliegt – im Browser angezeigt. Das Einbetten eines PGF in den HTML-Code ist ebenfalls sehr einfach. Die Bildgrösse kann selbst definiert werden.

Weitere Ziele, welche in einem nächsten Schritt verfolgt werden können, sind:

- falls im HTML keine Grössenangaben angegeben werden, soll die Defaultgrösse des PGF verwendet werden
- weitere Bildtypen sollen unterstützt werden
- optimierter Ladevorgang, indem nur die benötigten Level geladen werden
- z.B. ein "speichern unter..."-Menü, das mittels Rechtsklick auf das Bild realisiert wird

### Ausblick

Das entstandene Plugin kann weiter entwickelt werden. So wären Implementationen für alternative Betriebssysteme wie Linux und MacOS-X wünschenswert. Ebenfalls könnten noch nicht implementierte Bildformate integriert, sowie weitere Funktionalitäten hinzugefügt werden.

Das Visual Studio-Projekt kann auch für das Entwickeln weiterer Plugins als Hilfe oder Einstieg benutzt werden.

## Internet Explorer-Plugin

### Situation zu Beginn des Projekts

Zu Projektbeginn stand für den Microsoft Internet Explorer bereits ein Plugin zur Anzeige von PGF-Bildern zur Verfügung; dieses kann auf der Homepage der xeraina GmbH heruntergeladen werden.

Das Plugin besteht aus einer ActiveX-Komponente, welche im Browser installiert wird, sodass auf die Funktionalität zugegriffen werden kann.

### Ziel

Das bereits erhältliche Plugin ist nicht signiert; dies hat zur Folge, dass der Installationsprozess weniger benutzerfreundlich verläuft: Da der Browser ohne die Signatur die Herkunft der ActiveX-Komponente nicht erkennt, wird der Zugriff auf diese standardmässig gesperrt. Der Nutzer muss die Komponente manuell installieren oder aber die globale Sicherheitsstufe seines Browsers heruntersetzen. Keine der beiden Varianten ist für einen Nutzer besonders attraktiv.

Im Rahmen dieses Projekts soll darum eine Studie durchgeführt werden mit dem Ziel, die Hintergründe der Signierung von ActiveX-Komponenten kennen zu lernen und diese am Beispiel des bestehenden PGF-Plugin auszuprobieren, mit dem Ziel, dieses mit einer automatisch erkannten Signatur zu versehen.

Ein signiertes Plugin würde dem Browser erlauben, den Nutzer mit einer Nachfrage darauf hinzuweisen, dass eine signierte Komponente herunter geladen wird, sodass der Nutzer aufgrund des Signaturbesitzers entscheiden kann, ob er dem Herunterladen zustimmt oder nicht.

### Vorgehen

Dieses Teilprojekt bestand in erster Linie aus dem Zusammentragen von Informationen, deren Sichtung und anschliessender praktischer Anwendung.

### Informationsquellen

Als wichtigste Informationsquelle diente uns die Seite des Microsoft Developer Network (MSDN)<sup>8</sup>. Dabei handelt es sich um die zentrale Seite mit Hilfestellungen für Entwickler von Microsoft-Produkten und -Technologien. Die Seite ist sehr umfangreich und enthielt auch zu der Thematik des Projekts eine grosse Anzahl Erklärungen und Tutorials.

### Werkzeuge

Die verschiedenen Werkzeuge, welche zur Erstellung und Signierung von ActiveX-Komponenten verwendet werden, werden im Rahmen des Microsoft Visual Studio ausgeliefert oder können von einer Microsoft-Webseite heruntergeladen werden. Neben dem Visual Studio selbst, welches für die eigentliche Implementierung von ActiveX-Komponenten verwendet wird, kommen dabei zahlreiche weitere Tools, darunter verschiedene Kommandozeilen-Programme, zum Einsatz.

### Erkenntnisse und Umsetzung

Es hat sich gezeigt, dass sich die Signierung einer ActiveX-Komponente in der Tat aus einer Vielzahl teils sehr simplen Schritte zusammensetzt. Diese werden im Folgenden aufgeführt, während zugleich auf die Architektur einer signierten ActiveX-Komponente eingegangen wird.

---

<sup>8</sup> <http://msdn.microsoft.com/>

Eine ActiveX-Komponente besteht im einfachsten Fall aus einer ".OCX"-Datei. Als solche wird sie im Visual Studio erzeugt. Die Komponente kann als einfache Datei auf einen Webserver geladen werden und durch eine entsprechende OBJECT-Referenz in eine HTML-Seite eingebunden werden.

Ebenso ist es möglich, die ".OCX"-Datei in eine ".CAB"-Datei zu komprimieren. Dieses Archivformat kann dabei noch weitere Dateien mit einschliessen; dies wird auch im Rahmen der Signierung genutzt: Ausser der ActiveX-Komponente enthält die CAB-Datei eine zusätzliche Datei mit Installationsinformation für den Browser, die ".INF"-Datei. Ausserdem kann die CAB-Datei selbst mit einer entsprechenden Signatur versehen werden.

Der Signierungsprozess beginnt mit der Erstellung der INF-Datei, welche die nötigen Informationen über die in der CAB-Datei vorhandene ActiveX-Komponente enthält.

Ausgehend von einem Beispiel war somit eine Datei mit folgendem Inhalt zu erstellen:

```
[version]
signature="$CHICAGO$"
AdvancedINF=2.0
[Add.Code]
pgf.ocx=pgf.ocx
msvcrt.dll=msvcrt.dll
mfc42.dll=mfc42.dll
olepro32.dll=olepro32.dll
[pgf.ocx]
file-win32-x86=thiscab
clsid={DCF0768D-BA7A-101A-B57A-0000C0C3ED5F}
FileVersion=1,0,0,0
RegisterServer=yes
[msvcrt.dll]
FileVersion=4,20,0,6164
hook=mfc42installer
[mfc42.dll]
FileVersion=4,2,0,6256
hook=mfc42installer
[olepro32.dll]
FileVersion=4,2,0,6068
hook=mfc42installer
[mfc42installer]
file-win32-x86=http://activex.microsoft.com/controls/vc/mfc42.cab
run=%EXTRACT_DIR%\mfc42.exe
```

*Inhalt von pgf.inf.*

pgf.ocx entspricht dabei der ActiveX-Komponente. Ebenfalls sehr wichtig ist die Angabe zur CLASS-ID, im Abschnitt "pgf.ocx" mit "clsid" bezeichnet. Anhand dieser kann der Browser die ActiveX-Komponente identifizieren.

Diese Datei wird im nächsten Schritt zusammen mit der ActiveX-Komponente zu einer CAB-Datei verpackt. Dies geschieht über ein Kommandozeilen-Tool namens **cabarc**.

```
cabarc.exe -s 6144 N pgf.cab pgf.ocx pgf.inf
```

Dabei werden neben den zu verpackenden Dateien auch eine Anzahl Bytes angegeben, die der CAB-Datei als zusätzlicher Speicher zugewiesen wird. In diesen wird später die Signatur abgelegt.

Das Zertifikat haben wir zu Testzwecken mit Tools von Microsoft erstellt, welche gerade dem Zweck dienen, Zertifikate zu erstellen, die einzig zu Testzwecken dienen. Tatsächlich müssten in einer produktiven Umgebung Zertifikate eingesetzt werden, welche von einer anerkannten Certificate Authority ausgegeben werden, sodass der Nutzer eine gewisse Sicherheit hat, dass er dem Zertifikat vertrauen kann. Die nachfolgend eingesetzten Tools bieten dazu eine einfache und kostenlose Alternative.

Zunächst wird mit dem Tool **setreg** das System so eingerichtet, dass es im Test-Root erstellte Zertifikate korrekt behandelt.

```
setreg 1 true
```

Im Anschluss kann mit **makecert** ein X.509-Zertifikat erstellt werden.

```
makecert -sk pgfkey -n "CN=pgfproject" pgf.cer
```

Dieses wird auf "pgfproject" ausgestellt und enthält den private Key `pgfkey`.



Zum Einsatz als Signatur für eine ActiveX-Softwarekomponente wird aus diesem ein Software Publishing Certificate (SPC) erstellt, ein X.509-konformes Zertifikat. Hierzu dient das Tool **cert2spc**:

```
cert2spc pgf.cer pgf.spc
```



Die Signierung wird mit **signtool** durchgeführt, welches über einen Assistenten mit grafischer Benutzeroberfläche verfügt:

```
signtool signwizard
```

Im Rahmen des Assistenten werden die zu signierende CAB-Datei, die Zertifikats-Datei, der private Key, der Hash-Algorithmus und weitere Informationen angegeben.

Die Signatur wird abschliessend der CAB-Datei hinzugefügt. Sie kann beispielsweise über die Dateieigenschaften im Explorer überprüft werden.

Schliesslich wird die signierte ActiveX-Komponente in die Webseite eingebunden. Der folgende Code-Ausschnitt zeigt das entsprechende HTML-Tag.

```
<object classid="clsid:DCF0768D-BA7A-101A-B57A-0000C0C3ED5F" id="PGF"  
codebase="PGF.cab" width="156" height="118" border="0">  
  
<param name="Url" value="lena_lossless.PGF" ref>  
<param name="Thumbnail" value="0">  
  
<param name="Interactive" value="-1">  
<param name="_Version" value="65538">  
<param name="_ExtentX" value="3302">  
<param name="_ExtentY" value="2498">  
<param name="_StockProps" value="1">  
<param name="BackColor" value="16777215">  
<param name="Progressive" value="-1">  
<param name="Morphing" value="0">  
<param name="Level" value="2">  
  
<param name="Valign" value="top">  
<param name="Halign" value="left">  
If you do not see a second image, then please install the pgf plug-in for your browser.</object>
```

*Ausschnitt aus dem HTML-Code.*

Zentral sind dabei die Angabe der CLASSID sowie der Speicherort der CAB-Datei auf dem Server. Die weiteren Parameter beziehen sich auf das PGF-Plugin.

## Tests

Zu Testzwecken wurden die benötigten Dateien wie im vorhergehenden Abschnitt gezeigt vorbereitet und auf der lokalen Festplatte abgespeichert.

Beim Laden der Webseite im Internet Explorer wies dieser mit einer entsprechenden Warn-Meldung auf die Einbettung einer ActiveX-Komponente in der Webseite hin. Nachdem diesem zugestimmt wurde, erscheint unter dem besonders abgesicherten Windows Vista eine Meldung, dass die Komponente erfolgreich installiert wurde. Unter Windows XP erschien keine entsprechende Meldung.

Unter keinem der beiden Systeme wurde jedoch anschliessend das PGF-Plugin korrekt geladen. Es erschien lediglich der im HTML-Code definierte Alternativtext.

Das Plugin wurde so unter Windows XP mit Internet Explorer 6 und unter Windows Vista mit Internet Explorer 7 getestet und führte in beiden Fällen zum selben Resultat.

## Erreichte Ziele

Die Signierung des PGF-Plugins verlief nicht erfolgreich. Zwar konnte dieses vom Browser als ActiveX-Komponente erkannt werden, nicht erkannt wurde jedoch die beigefügte Signatur.

Es ist uns nicht klar, zu welchem Zeitpunkt des Signierungsprozess ein Fehler aufgetreten ist. Da das der CAB-Datei angefügte Zertifikat in den Dateieigenschaften betrachtet werden kann, kann der Prozess nicht gänzlich fehlgeschlagen sein. Vielmehr scheint es so, dass dieses vom Internet Explorer nicht korrekt erkannt wird.

Die Webseiten des MSDN behandeln in erster Linie die Erstellung von Zertifikaten; deren Nutzung, insbesondere im Rahmen von ActiveX-Komponenten wird jedoch nicht im Detail behandelt. Auch haben wir keine Informationsquellen gefunden, die uns bei der Suche nach Fehlern im Signierungsprozess weiter geholfen hätten.

### **Erlangtes Wissen**

Neben der Erstellung einer Signatur für das bestehende PGF-Plugin war es ein weiteres Ziel, die für uns neue Thematik der ActiveX-Komponenten und deren Signierung kennen zu lernen. Wir haben jetzt verschiedene Prozesse und die dahinter steckenden Technologien kennen gelernt; trotzdem hat es zeitlich nicht gereicht, sich eingehender mit der Thematik zu beschäftigen.

Da wir den Grossteil der Projektzeit mit der Fertigstellung der PGF-Plugins zugebracht haben, reichte die Zeit nicht für eine vertiefende Auseinandersetzung mit der Materie.

### **Ausblick**

Die von uns gesammelten Informationen zum Signierungsprozess dürften prinzipiell in die richtige Richtung weisen. Wir schliessen jedoch nicht aus, dass es noch andere Komponenten gibt, die wir im Rahmen unserer einführenden Betrachtung der Thematik noch nicht durchgesehen haben.

Insbesondere die Speicherung von heruntergeladenen Komponenten durch den Internet Explorer ist uns nicht gänzlich klar geworden. Ebenfalls ausser Acht gelassen haben wir den Quelltext der ActiveX-Komponente selbst; eine weitere Möglichkeit wäre daher, dass dieser speziell für die Signierung angepasst werden müsste.

## Reflexion

An dieser Stelle möchten wir kurz einen persönlichen Blick auf das Projekt werfen und ein Fazit ziehen im Sinne von "Lessons learned".

## Zeitplanung

Der Zeitplan wurde von uns zu Projektbeginn ausgearbeitet und später wieder überarbeitet. In Anhang A liegt die finale Fassung, wie sie am 24. September 2008 ausgearbeitet wurde.

Tatsächlich stellte sich jedoch bald heraus, dass es nicht sinnvoll wäre, stur einem vorgegebenen Zeitplan zu folgen. Unsere Arbeit geriet zunehmend in Verzug, wobei die Gründe an verschiedenen Stellen in diesem Dokument erläutert werden. Schliesslich erwies sich die Implementierung der Plugins als Hauptarbeitsgebiet und wir haben uns seit dem Zeitpunkt des planmässigen Beginns dieser Arbeit fast vollständig darauf konzentriert. Schliesslich zog sich diese Arbeit bis in die unterrichtsfreie Zeit über Weihnachten hinein; nebenbei wurden noch die notwendigen administrativen Arbeiten erledigt.

## Christian Züger

Die Gefühle zum Zeitpunkt der Projektabgabe sind gemischt. Zu gross waren die persönlichen Ziele und Vorstellungen, zu gering das nun vorliegende Endprodukt.

Sehr gerne hätte ich das Firefox-Plugin weiter ausgebaut, mehr Funktionen hinzugefügt. Doch wie so oft spielte der Faktor Zeit nicht mit und lässt die abgeschlossene Arbeit nun etwas unvorteilhaft dastehen. Zeit wurde sehr viel investiert. Wir haben uns hinter die Aufgabe geklemmt und die hochgesteckten Ziele stets im Auge behalten, gereicht hat es zum Schluss "nur" zu einem Grundprogramm.

Wie mein Kollege Nicolas Hofstetter in seiner persönlichen Reflexion ebenfalls erwähnen wird, war das Arbeiten am Projekt während der Unterrichtszeit nur bedingt möglich. Sobald eine Prüfung bevorstand oder Modulreporte/Labs – und das waren nicht wenige – abgeschlossen werden mussten, rückte das Projekt in den Hintergrund. Klar hätte man trotzdem parallel am Projekt arbeiten können, aber dass dies wiederum negative Auswirkungen auf die Noten der besagten Prüfungen/Labs haben würde, brauche ich wohl nicht weiter zu erklären. Und so kam es, dass die meisten Projektstunden während der Projektwoche oder der unterrichtsfreien Zeit über Weihnachten verzeichnet wurden. Die Projektwoche erachte ich übrigens als sehr sinnvoll. Diese Zeit kann effektiv und intensiv genutzt werden.

Im Speziellen möchte ich noch zwei aufgetretene Probleme/Unklarheiten aufzeigen:

Zum einen war dies der mir zuerst nicht bekannte Unterschied zwischen einem Firefox-Plugin und einem Add-on (die Unterschiede wurden in diesem Dokument aufgezeigt). Zu Beginn des Projektes bin ich davon ausgegangen, dass beides dasselbe sei. So fand ich zwar sehr schnell gute Tutorials, musste jedoch bald feststellen, dass diese für mich vollkommen nutzlos sind. Bis eine vernünftige Anleitung zum Bau eines Plugins gefunden wurde, verstrich leider viel Zeit.

Zum andern stiess ich auf ein Problem beim Erzeugen der DLL aus dem Visual Studio und dem automatischen Kopieren in den Mozilla-Ordner. Aufgrund des genutzten Betriebssystems Microsoft Vista und dessen etwas speziellen Rechteverwaltung wurde der Zugriff auf das gewünschte

Verzeichnis verhindert. Die Lösung war, das Visual Studio als Administrator auszuführen, und so die vollen Rechte zu erhalten.

Trotz allem war das Projekt rückblickend ein Erfolg. Die vielen Rückfälle, Probleme, der Zeitdruck, aber auch die Arbeit in einem gut harmonierenden Team und einem hilfsbereiten Betreuer waren gute Erfahrungen und haben seine Spuren bei mir hinterlassen.

## Nicolas Hofstetter

Ich muss zugeben, dass dieses Projekt von Anfang an ein Wagnis war. Mein Projektpartner und ich hatten sich bis zu diesem Zeitpunkt noch nicht mit der Thematik der Bildverarbeitung auseinandergesetzt, insbesondere auch, da der Besuch entsprechender Module erst in diesem Semester Sinn machte. Trotzdem haben wir uns für dieses Projekt entschieden, ich für meinen Teil, weil ich die interessantesten, aber hochkomplexen Projekte meiner Vertiefungsrichtung Distributed Systems erst als Bachelor Thesis mit dem Wissen aus weiteren besuchten Modulen "im Gepäck" bearbeiten wollte.

Nun ist das Projekt bereits vorbei, und ich ziehe eine gemischte Bilanz. Auf der einen Seite möchte ich positiv herausheben, dass ich tatsächlich noch in keinem Projekt zuvor an dieser Schule so viel fachliches Wissen sammeln konnte, und dies in einem Bereich, der sehr interessant ist. Auch konnte ich dieses auch sogleich praktisch umsetzen. Ich darf ohne Übertreibung sagen, dass ich in diesem Projekt mehr gelernt und verstanden habe als in manch anderem Modul dieses Semesters.

Andererseits bin ich jedoch auch enttäuscht, dass praktisch gesehen am Ende ein bisschen wenig dabei herausgeschaut hat. Ich konnte einiges nicht umsetzen, was ich mir vorgenommen hatte, ja ich kam noch nicht einmal dazu, alles Wissenswerte tatsächlich zu verstehen, was dazu nötig gewesen wäre.

Warum gab es diese Schwierigkeiten? Ich kann es anhand eines "Ressourcen-Kuchens" zeigen, welcher eben aus den für ein Projekt wichtigen Ressourcen besteht – Geld, Personal, Wissen und Zeit. Es ist offensichtlich, dass wir in allen Bereichen sehr eingeschränkt waren, und verbessern konnten wir uns stets nur zulasten der Zeit. Und diese war neben allen anderen Schulaufgaben äusserst knapp. Tatsächlich stellte sich die Projektarbeit in einer Schulwoche mit Prüfungen als fast unmöglich heraus. Man sitzt nicht täglich eine Viertelstunde hin für das Projekt und legt es dann wieder zur Seite – es benötigt die volle Konzentration und ruhiges Arbeiten über mehrere Stunden, in einer normalen Schulwoche eine Unmöglichkeit. So mussten wir schliesslich einen Grossteil der Arbeit in der Projektwoche und über Weihnachten erledigen.

Ich hoffe, dass sich dieses grosse Problem zumindest während der Bachelor-Arbeit nicht stellen wird.

## Literaturverzeichnis

Häufig haben wir das Internet als Informationsquelle herangezogen. Die entsprechenden Quellen sind in den betreffenden Seiten des Berichts in der Fusszeile aufgeführt.

Auch andere Informationsquellen sind jeweils als Fussnote realisiert.

## Anhang A: Zeitplan

	17.09.2008	24.09.2008	01.10.2008	08.10.2008	15.10.2008	22.10.2008	29.10.2008	05.11.2008	12.11.2008
	23.09.2008	30.09.2008	07.10.2008	14.10.2008	21.10.2008	28.10.2008	04.11.2008	11.11.2008	18.11.2008
Woche	1	2	3	4	5	6	7	8	9
Mittwoch									
Dienstag									
<b>Sitzungen mit betreuendem Dozent</b>		24.09.2008		08.10.2008		22.10.2008		05.11.2008	
Vorbereitung Infrastruktur									
Pflichtenheft									
Wissenssammlung gemeinsam (PGF)									
Wissenssammlung individuell (Plugins)									
Vorbereitung Testing, Bugfixing Allg.									
<b>Plugin 1: Firefox (CZ)</b>					X				X
<b>Plugin 2: Gimp (NH)</b>					X				X
Durchführung Testing, Bugfixing Plugins									
Wissenssammlung IE-Plugin									
<b>IE-Plugin</b>									
Durchführung Testing, Bugfixing IE									
Dokumentation									
Abschluss-Dokumentation									
Reserve (Bugfixing, Dokumentation)									
Vorbereitung Präsentation									
ITERATIONEN	1			2					3
Mittwoch	19.11.2008	26.11.2008	03.12.2008	10.12.2008	17.12.2008			07.01.2009	14.01.2009
Dienstag	25.11.2008	02.12.2008	09.12.2008	16.12.2008	20.12.2008			13.01.2009	20.01.2009
<b>Woche</b>	10	11	12	13	14	15	16		
<b>Sitzungen mit betreuendem Dozent</b>	19.11.2008		03.12.2008		17.12.2008			07.01.2009	
Vorbereitung Infrastruktur									
Pflichtenheft									
Wissenssamlung gemeinsam (PGF)									
Wissenssamlung individuell (Plugins)									
Vorbereitung Testing, Bugfixing Allg.									
<b>Plugin 1: Firefox (CZ)</b>									
<b>Plugin 2: Gimp (NH)</b>									
Durchführung Testing, Bugfixing Plugins									
Wissenssamlung IE-Plugin			X						
<b>IE-Plugin</b>									
Durchführung Testing, Bugfixing IE									
Dokumentation									
Abschluss-Dokumentation					X				
Reserve (Bugfixing, Dokumentation)									
Vorbereitung Präsentation									
ITERATIONEN			4						

Zeichenerklärung:

Gemeinsame Arbeit

Individuelle Arbeit

X Meilenstein