Developing an on-line Cree read-along with syllabics[1]

Radu Luchian and Marie-Odile Junker
Carleton University
e-mail: radu@monicsoft.net, mojunker@ccs.carleton.ca

Introduction

East Cree is a Native American language spoken on the Eastern coast of James Bay, Quebec, Canada. Like many other Aboriginal languages, it is struggling to survive. Using participatory action research (Morris&Muzychka, 2002; Junker, 2002), the *eastcree.org* project (www.eastcree.org) is exploring how Information Technology can assist language documentation, preservation and transmission. We report here on the development of on-line read-along material, whose goal is to strengthen literacy in Cree syllabics. Since Cree became the language of instruction in all Cree schools in 1995, the department of Cree Programs (the curriculum development unit of the Cree School Board for Cree language and culture) published hundreds of books in Cree syllabics (Burnaby et al. 1999a, 1999b). We were asked to explore possibilities of adapting such books to the web in order to have the story read back to the user and also to teach correct spelling by highlighting the portions of text on each page, as it is being read.

Why the web?

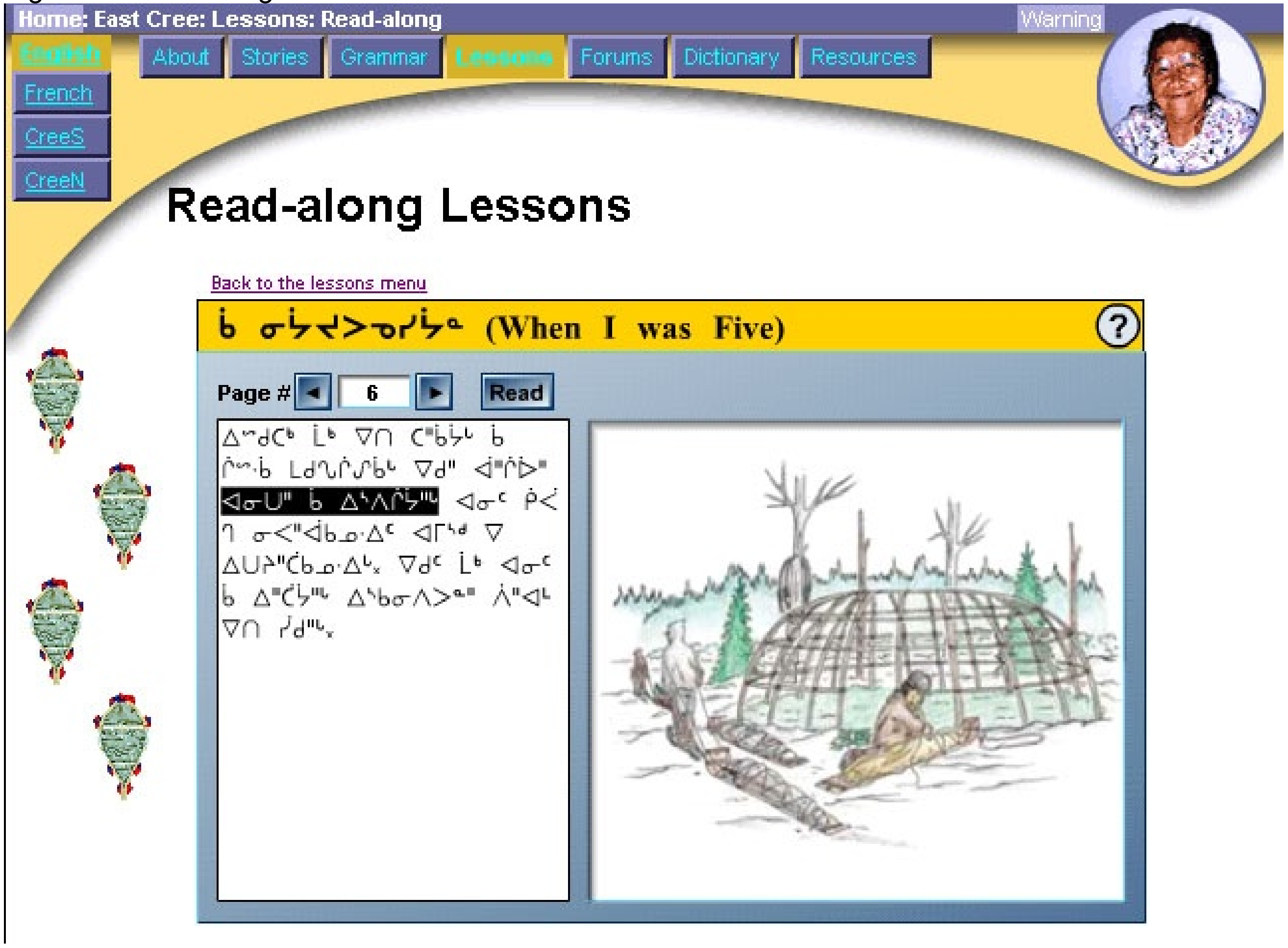
Most of the existing tools we are aware of require the care of a rather large group of specialists in various arts and technologies: text editing and translation, sound editing, digital painters, multimedia editors and programmers, web designers... In our read-along project, as with most of the other eastcree.org projects (Junker and Luchian, 2003), we tried to streamline the process, compile or create a list of tools and tasks as short as possible so that Aboriginal students or staff members of various aboriginal organizations could record and distribute on-line their stories, with as little training as possible.
We had to look at all the technical problems of multimedia distribution over the Web from the point of view of the non-technical person. Our task is to create a piece of software that would work on as many platforms as possible from the ones in which current Cree users have already invested resources (money for buying equipment, time for learning the software and specific interfaces, etc.). We have to allow them access to their publications either on-line, using the most widespread Internet channel, the World Wide Web, or locally, on centrally-distributed CDs. Like books, CDs can be only snapshots at a given point in time of a published product. There are positive sides to that (like a common, stable reference system), and negative sides (any errors can't be corrected efficiently and additions are practically impossible). The ease of development we mentioned before as prime requirement, suggested a platform that would allow dynamic (user-centric) use and development rather than the classic static (computer-and-developer-centric) model that states that the users do not know what they need and it's up to the developer to figure it out and compile it into software. We chose to concentrate on the on-line development, with an option to make any part of our project 'photographed' at any point in time in a CD version.
The Web alternatives we have investigated so far are based on the three most available cross-platform technologies we are aware of: in-browser JavaScript, Macromedia Flash-based ActionScript, and applet-based Java.
Early in the design phase we have excluded Java applets because of the extensive programming knowledge involved. In this report we will concentrate on the two options we have explored rather thoroughly: DHTML and Flash. But first let us take a look at our prototype.

[1] Carleton University Cognitive Science Technical Report 2006-01. http://www.carleton.ca/ics/TechReports

The prototype

The Cree Programs books are lovingly accompanied by drawings, painting or pictures by native artists, we thus decided to preserve the page-by-page format of the original books. The prototype illustrated in Figure 1 is based on a book written by Daisy Moar, and illustrated by her son Willard Moar. Ms Moar read her story in a professional recording studio and checked the accuracy of the sound editing and final product. Such an ideal author's involvement may not be possible to replicate in all cases, but is highly recommended since it ensures cultural accuracy and appropriate development of the medium.

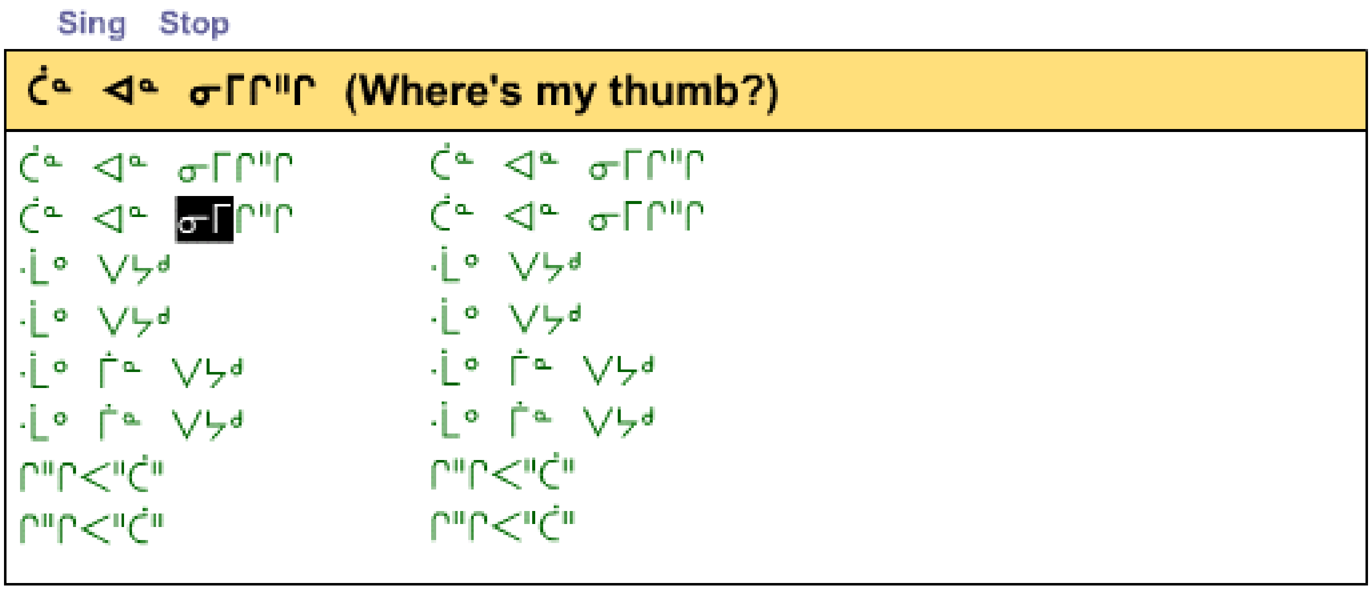
Figure 1: Read-along



ᐃᐦᑯᑕᒃ ᒦᒃ ᐁᑎ ᑕᐦᑳᔮᒡ ᑳ
ᒌᔥᐧᑳ ᒪᑯᓭᒌᔑᑳᒡ ᐁᑯᐦ ᐋᐦᒌᐆᐦ
ᐊᓂᑌᐦ ᑳ ᐃᔅᐱᒌᔮᐦᒡ ᐊᓂᑦ ᒌᐹ
ᒉ ᓂᐸᐦᐋᑳᓈᐧᐃᑦ ᐊᒥᔅᑯ ᐁ
ᐃᑌᔾᐦᑖᑲᓈᐧᐃᒡ᙮ ᐁᑯᑦ ᒦᒃ ᐊᓂᑦ
ᑳ ᐃᐦᑖᔮᐦᒡ ᐃᔅᑲᓂᐱᐳᓐᐦ ᐲᐦᐋᒻ
ᐁᑎ ᓰᑯᐦᒡ᙮

The prototype shown in Figure 1 is the latest we made. The first Flash prototype we built in JavaScript and then tested in Flash was a sing-along one - see Figure 2. As you can see, it is much less complicated: one page per song, no image to download and faster-moving, more detailed text selections. We hope that the smaller selection, coupled with the rhythm of the song can help novices learn the prosody of the language faster, and to better associate the syllable glyphs with their respective sounds. However the way we embedded the sound file in the Flash applet results (again) in speed inconsistencies (selection vs. sound), when playing on computers of really different speeds, but these inconsistencies are much smaller than the best we could achieve with JavaScript.

Figure 2: Sing-along



ᒑᓐ ᐊᓐ ᓂᒥᒋᐦᒋ (Where's my thumb?)

ᒑᓐ ᐊᓐ ᓂᒥᒋᐦᒋ ᒑᓐ ᐊᓐ ᓂᒥᒋᐦᒋ
ᒑᓐ ᐊᓐ ᓂᒥᒋᐦᒋ ᒑᓐ ᐊᓐ ᓂᒥᒋᐦᒋ
ᒹᐤ ᐯᔭᒃ ᒹᐤ ᐯᔭᒃ
ᒹᐤ ᐯᔭᒃ ᒹᐤ ᐯᔭᒃ
ᒹᐤ ᒌᓐ ᐯᔭᒃ ᒹᐤ ᒌᓐ ᐯᔭᒃ
ᒹᐤ ᒌᓐ ᐯᔭᒃ ᒹᐤ ᒌᓐ ᐯᔭᒃ
ᒋᐦᒋᐸᐦᒑᐦ ᒋᐦᒋᐸᐦᒑᐦ
ᒋᐦᒋᐸᐦᒑᐦ ᒋᐦᒋᐸᐦᒑᐦ

The first problem we had was representing the syllabic script (the glyphs seen in Figure 1 and 2).

Unicode or 8bit fonts?

Unicode is a data-encoding system that uses a minimum of two bytes to represent each character in a text file. This allows editors to place any number of languages, with a very large number of glyphs on one page or text field, without using a lot of formatting artifice. Older systems (up to Windows 2000 and MacOS X), allowed a theoretical maximum of 256 different symbols to be displayed at once ($256=2^8$, thus that form of encoding is called 8bit). When using 8bit encoding, in order to switch from one language to another, some code has to be added (depending on the display method and software, all space-consuming solutions, requiring increased maintenance). As a relief from that problem, Unicode encodes each glyph to be displayed (e.g. character, ideogram, symbol), in two bytes (thus expanding the theoretical maximum to 2^{16}=65536 different glyphs), and tells the user's computer which letters or symbols to place on the page (Jancewicz and Junker 2002, 2003).
The theoretical difference between Unicode and 8-bit fonts resides in file sizes (twice as big when using Unicode). But other encoding standards, as UTF-8 allow embedding of Unicode characters in files which contain mainly one page of 8-bit fonts (by adding a third byte, prepended to each Unicode character).
Even though we started both in DHTML and in Flash by using Unicode fonts for their portability and relative layout economy, in versions 5 and 6 of the Flash development environment and in the player, Unicode fonts proved to be almost impossible to use. In 2002, however, with Flash MX, Macromedia introduced support for embedding UTF-8 files, but there was no way to do any editing within their development environment, and we could not embed the Unicode fonts. This was fixed recently, in MX2004 - and a few patches later, we can actually work with Unicode embedded fonts.
This means that the stories or any other content passed through our Flash applet will be seen the same way on any platform for which Flash has a player.
Let us now discuss the web programming options we explored thoroughly.

The programming options: DHTML and Flash

DHTML (Dynamic Hypertext Markup Language) is a text-oriented way to publish data in a way that reduces communication between the publisher's server computer and the user's computer, while keeping the project consistent and easily updateable. The version we chose uses HTML (HyperText Markup Language: a way of adding pseudo-semantic data to documents which by now may be familiar to most people), JavaScript (client-side, browser-based, scripted programming language), and CSS (Cascading Style Sheets: text-based description of the visual characteristics of the layout

elements used in publishing). We chose HTML rather than XML and JavaScript from among alternatives like VisualBasicScript and C# because of the wider, more consistent support that various forms of HTML and JavaScript have in existing browsers (e.g. Internet Explorer, Netscape-Mozilla, Opera). With DHTML (as opposed to some HTML version that absolutely all browsers use), we can increase the consistency of presentation layout and reduce maintenance time. The World Wide Web Consortium (W3C) recommends the use of a DOM (Document Object Model), over browser-specific DHTMLs (W3C, 2003). When the available browsers will implement that recommendation we will have a much easier time trying to cover the quirks of programming required by each platform and browser. Until then, we depend on a set of 3-4 JavaScript objects which cover rather consistently the browsers we are targeting, and a well-supported version of HTML (3.2). And the W3C has published and is continuously updating a reliable checklist (W3C, 2000) for making sure dynamic websites are as accessible as possible.
Flash, on the other side, is a graphics-oriented multimedia tool, currently developed by Macromedia, Inc., and optimized for use over the Web. It shares with well-designed-DHTML methods like streaming (to get data displayed as soon as it reaches the browser rather than waiting for an entire layout to load), and programming (with ActionScript, a subset of the language on which various forms of JavaScript are based), but it adds compression and use of vector graphics (to make files smaller thus faster displayed) and an almost playful capacity to create animations and/or interactive presentations.
DHTML is an open set of technologies, designed by the W3C while Flash is mostly confined to a comparatively closed development environment designed by one company. This is why we started developing in DHTML and defaulted to Flash only when it turned out that cross-platform bugs and browser wars were making almost impossible a consistent implementation based on W3C protocols.

First Try: DHTML and Unicode font

Most of what we have to communicate is text in several languages (Cree syllabics, French, English), and the text editors are very cheap (the ones which are bundled with Windows or MacOS - Notepad and TextEdit respectively, are good enough). Unicode and the test-based DHTML seemed the best approach, since the first allows us to place all three languages on one page without using a lot of artifice, and the second allows us to handle data on the user's computer, thus reducing the time wasted waiting for the server to render and transmit each layout.
With DHTML, the main problem is heterogenic support. Each computer has a different set of software installed, and diverging marketing campaigns known as the 'browser wars' resulted in a relatively low level of compatibility in the dynamic features of different Web browsers. Some browsers don't support DHTML at all, or have features that allow their user to turn off JavaScript. Of the browsers, which do handle DHTML, each browser type and even each version is using a more or less different version of JavaScript and CSS. The final result is that on some computers a DHTML application would look and work just fine and on another computer it would look differently or not work at all. Indeed, there are ways to make the application notice what type of computer and browser it happens to be on, and use the features available on that platform. This is the approach we have tried, but the design process is much slower and requires a prior knowledge of what features work on each platform and how, or it requires knowing what combination of browsers and operating systems the intended audience has. This solution is partly self-defeating: with each platform checked, code has to be added to handle the differences and that increases the size of the data sent from the server. Also the code becomes less and less legible and more and more difficult to maintain. There are also tools being developed by different software companies, which make it easier to develop dynamic code that works cross-platform; however, these tools tend to be expensive and have a rather steep learning curve.
To compound that problem, not all platforms support Unicode. MacOS up to version X doesn't support it at all. Windows up to WinNT SP4 does not allow data entry using Unicode.
Lastly, since we are using quite a lot of sound in these interactive books, an extra plug-in is needed on the user's computer to play back those sounds. Using DHTML, developing a dynamically

rendered player was relatively easy, but it is still dependent on the software installed on the user's computer (like QuickTime or WindowsMediaPlayer).
A minor problem in the DHTML version is that it requires the presence of a Cree font on all computers which have to interact with our web site, or the special characters which make up the Cree syllabary would be displayed as squares or question marks. We considered using some of the tools available to serve the font together with the web page, but found them to be more expensive and work on less platforms than the Flash alternative. Another problem was that, like Flash, they also require the use of an applet or plug-in.
The final DHTML prototype was disappointing: even with the same software. On machines of different speeds there was a lack of synchronicity between the speeds with which the sound was playing and the speed of the highlighted text. On the development computer, equipped with an Intel Celeron processor running at 900Mhz, the story and text highlighting would be in synch, while on an AMD K6 at 500MHz, the sound would tend to lag behind the selection and on an Intel Pentium4 at 2.7GHz the selection would lag behind the sound.

Second try: Flash and Unicode font

So, once we noticed that in order to target the Web population - very diverse and sometimes containing obsolete platforms and applications - we would have to increase development and maintenance, investing in at least one tool for development becomes necessary. To display custom fonts automatically, a plug-in had to be installed; to play sound we needed another plug-in; to display any animation and other eye candy, if we wanted to, we needed yet another plug-in; to make sure out code worked cross-browser we had to resort to intricate conditional layout rendering... It all pointed in the direction of the one plug-in that could do all these and still be free for the end-user. Our choice is Flash because of the rather intuitive development interface and its integration with other Macromedia media editing tools one would anyway have to use (Gonsalves, 2004). Fireworks is a good hybrid editor, as it handles both bitmap and vector graphics. Dreamweaver is one of the best web design tools we found, available in a development package with an education price of only US$199 for two development seats. The alternative would be the Adobe graphical line (Photoshop for professional bitmap images, Illustrator for vector graphics, Audition for sound editing and Acrobat to create passable multimedia documents).
Flash did not force us to reduce modularity (as a PDF solution would). In the read-along, we stream both sound and images, as the user requires them. The read-along player is small (20k, the size of an average screen image), and it loads the book description (an XML file), then on each page two files: (1) the image for each page (5-40k each), and (2) the associated sound file (26-111k). This results in a more responsive interface and a better
An extra bonus is better organization of the project files. Unlike in a text-based solution (like our DHTML one, in which any sound and images and other bits of pages have to be present in preset places in a folder structure), Flash allows all the unchangeable elements of the interface to be packaged in one file, which makes it much easier to maintain, smaller and less confusing.
The only problems remain the rather clumsy and buggy implementation of Unicode in Flash (only UTF-8 can be used, with imported XML files), and the fact that Flash being a commercial tool can decide at any time to increase its prices. One place where its prices are already way up is in development support and training: access to reliable answers is tagged at a minimum of US$500 per year and training courses are correspondingly high.

Final task list

Let us conclude with a final task list, as they currently stand for our Cree read-along learning tool:
1. Designers and programmers, after consulting aboriginal artists, create one or more templates for specific subsets of books (e.g. legends, histories, memories), which provide the cultural context and coding needed to run the interactive books.
2. (Native) story editors' use:
2.1. A simple text editor (e.g. Notepad under Windows), to edit a text file provided by the designers, by simply changing bits of text to fit the book they are editing.

2.2. Photoshop or Fireworks or any other image editor can be used to scan and save the images, which accompany the book.
2.3. Record the story with Adobe Audition or SoundForge, save it, and make one file per book page.
3. Type or paste the text for each page into the book template;
4. Scan images and place them in a common subdirectory; add relative path to the subdirectory to the book template;
5. In the sound editor, mark selection times (the time each piece of text we want highlighted is read in the sound file), and add these times to the book template;
6. Count letters or use the Flash editing environment to determine the corresponding highlight spans and add them to the book template;
7. Use Flash to publish the completed book template as an interactive book (with one keystroke or two mouse clicks);
8. Test and if necessary redo parts of points 3-7;
9. Upload the book, sound files and image files in the corresponding structure on the web site or on a CD.

References

Burnaby, Barbara, Marguerite MacKenzie and Luci Bobbish-Salt. 1999a. “Factors in aboriginal mother tongue education: the Cree School Board Case. In David Pentland (ed.) *Papers of the Twenty-ninth Algonquian Conference*, 62-73. Winnipeg: University of Manitoba.

Burnaby, Barbara, Marguerite MacKenzie and Luci Bobbish-Salt. 1999b. “Native language for every subject: the Cree Language of Instruction Project.” In Jon Reyhner, Gina Cantoni, Robert N. St. Clair, and Evangeline Parsons Yazzie (eds.). *Revitalizing Indigenous Languages* web version. Flagstaff, AZ: Northern Arizona University. http://jan.ucc.nau.edu/~jar/Burnaby.html

Gonsalves, Antone. 2004 .Net? Java? No Thanks, We'll Take Macromedia Instead. *InternetWeek* (http://www.internetweek.com/breakingNews/showArticle.jhtml?articleID=14800074)

Jancewicz, Bill and Marie-Odile Junker. 2002. Cree on the Internet: How to Integrate Syllabics with Information Technology and the Web. Presented at the *34th Algonquian Conference,* Kingston: Queen's University.

Jancewicz, Bill and Marie-Odile Junker. 2003. Frequently asked questions about Cree syllabics, Computer technology and the Web. In. www.resources.eastcree.org , web pages and PDF download.

Junker, Marie-Odile (ed.) 2000-2004. The East Cree Language Web. www.eastcree.org

Junker, Marie-Odile. 2002. Participatory Action Research in Linguistics: What Does it Mean? / La recherche participaction en linguistique: Enjeux et significations. Presented at the Session on Ethics of Archiving Languages and Fieldwork, organized by the Aboriginal Language Committee, *Canadian Linguistics Association Annual Congress*, University of Toronto, May 2002.

Junker, Marie-Odile and Radu Luchian. 2003. Building a Cree Oral Stories Web Database Through Participatory Action Research. Presented at the *35th Algonquian Conference,* University of Western Ontario.

Morris &Muzychka. 2002. *Participatory research and action.* Ottawa: Canadian Research Institute on the Advancement of Women.

World Wide Web Consortium (W3C). 2003. Document Object Model Frequently Asked Questions, available online at http://www.w3.org/DOM/faq.html - DHTML-DOM

Software cited:

Adobe. 2003. Photoshop Elements, available online at http://www.adobe.com/products/photoshopel/main.html

Adobe. 2004. Audition, available online at http://www.adobe.com/products/audition/

Adobe. 2004. Acrobat Professional, available online at http://www.adobe.com/products/acrobatpro/main.html

Macromedia. 2004. StudioMX2004, available online at http://www.macromedia.com/software/studio/

Macromedia. 2004. Flash, available online at http://www.macromedia.com/software/flash/