Introduction.

Consider an idealized computing device, specifically designed for extra-curricular educational purposes. We postulate that it has several characteristics:

1. It will be small, easily transportable, with high resolution display and long battery life.
2. Mathematics can be entered interactively, as well as displayed in high quality.
3. Graphics (both 2D and 3D) can be rendered interactively.
4. It will have a functional keyboard.
5. Video can be displayed at high frame rates and supports internet connectivity.
6. It will have a browser which can access World Wide Web, and supports advanced text and graphics effects (similar to Flash).
7. It will have a built in programming language.
8. It can download web applications and work offline.
9. It will be inexpensive, and will be purchased by 10’s of millions of students (or their parents).

Such a device in fact exists, and is known to us today as a *smart phone* or a *netbook*. Related devices (iPhone, iPod touch, iPad) also fit the above characterization.

The educational potential of such devices is only now being developed.

The characteristics described in 1, 4 and 9 are physical and economic properties of present day smart phones, netbooks (and their cousins) and can easily be verified.

In this workshop, we want to demonstrate how one can take advantage of the functional characteristics described in 2, 3, 5, 6, 7, and 8. In particular, we want to demonstrate them in an educational context.

We all know how much time students (ages 5-18) spend on cell phones. Just think if they devoted a fraction of this time to well designed and well thought out educational applications!

Let’s take a closer look at how one can develop such applications, today…

**Chapter 1. Rendering Mathematics**

One of the first ways that mathematics was rendered on web pages was through embedded graphics (usually GIFs). For example, a source document (usually in LaTeX) was processed offline through a package (perhaps LaTeX2HTML) and all the equations were rendered as GIFs inside image tags on the HTML page.

The results were usually reasonable, but there were several disadvantages. Any time the source document was updated, it had to be processed offline. Any time an equation image changed, the page had to be reloaded, which discouraged interactivity.
Another early alternative was the development of plugins such as Adobe Reader which could render PDF documents inside the browser. They were typically of very high quality, but limited interactivity.

A newer way to render mathematics is through MathML (Math Markup Language) which is an XML specification supported by many browsers. It depends very heavily on the browser supporting MathML fonts natively (or through plugins).

Interactivity in modern HTML is greatly facilitated by the DOM (Document Object Model) way of viewing web pages. Pages can be modified and constructed on the fly quite easily.

There are at least 3 ways in which mathematics can be rendered, interactively.

1. MathML fonts. This is the approach taken by the package AsciiMathML (http://asciimathml.sourceforge.net/). If you right-click on an equation, one of the options is “view MathML source”. This is very illustrative, and shows you quickly why very few people compose documents from scratch in XML or MathML!

2. Images. This is the approach that jsMath (http://www.math.union.edu/~dpvc/jsMath/) and MathQuill (http://laughinghan.github.com/mathquill/) take. They use rather large JavaScript libraries to render mathematics in fonts (if available) or through images (if fonts are not available).

3. Style Sheets: This is the approach that MathJax (http://www.mathjax.org/) takes.

Several interactive examples (with source code) are shown on this website for jsMath and MathQuill. On the home pages of AsciiMathML and MathJax there are links to interactive demo pages.

The common denominator of all of these packages is the ability to enter raw LaTeX into the HTML source (with or without $ delimiters), and have the document re-rendered (using JavaScript libraries) when it is loaded into the page.

This also makes “algorithmic” LaTeX expressions possible. LaTeX expressions can be loaded in with parametric constants embedded, which can then be instantiated using random numbers. This makes designing algorithmic quizzes possible using high-quality mathematics notation. Several examples (with source code) are included on this website.

Chapter 2. Rendering Graphics

One of the things that HTML5 supports is the canvas element. Once a canvas object is created using the <canvas> tag, it supports a large number of features (http://www.selfhtml5.org/wp-content/uploads/2010/07/HTML5_Canvas_Cheat_Sheet.png).

Using basic constructs like lineTo( ), moveTo( ) and stroke( ), simple planar images can be constructed (after defining a suitable coordinate system of course). An example of a simple 2D function grapher is given.

To graph 3D images using the <canvas> object, one must have vector representations of coordinates and subroutines to perform all the relevant matrix rotations, scalings, translations,
projections, etc. Fortunately, these are well understood and fairly easy to code. An example of a simple 3D wireframe renderer is given. These examples are fully interactive, with pull down menus loading sample functions into a text area, which can then be edited.

Chapter 3. Displaying Video

There are many video standards, but there does not seem to be one that can be displayed on all web browsers on all platforms (Android, Mac OS X, Windows, and Linux). HTML5 supports two standards (OGG and MPEG-4). Since QuickTime and Flash both support MPEG4 video, with a little wizardry one can coax QuickTime and Flash to display MP4 videos.

The web site html5media.googlecode.com has a core JavaScript library that performs the following sequence of actions. When the page loads, it discovers which browser and OS it is being displayed in. On Apple systems (iPhone, iPad, iTouch, Mac OS X) it creates a QuickTime wrapper around the video, and uses the QuickTime plugin or viewer. On Windows system, it uses a Flash wrapper (or fallback) or native MP4. On Linux systems it uses a Flash wrapper.

This allows you to display the video with only a few lines of code, and encode it in just one format, MPEG-4. One example (with source code) is provided on the website.

Other niceties such as displaying controls, checking the size, etc. are handled by the JavaScript library.

Another alternative is taken by the vimeo web site. After uploading a video, huge banks of servers translate the video into various formats (Flash, QuickTime, Mpeg-4, and Ogg) and display the appropriate format to the client. This approach has been taken by YouTube to circumvent the lack of flash on Apple devices.

Chapter 4. Browser with Internet Connectivity, Advanced Text and Graphics

In the desktop world, there are several dominant flavors of browsers – Internet Explorer (for Windows), Safari (for Mac OS), and Mozilla/Firefox/Chrome (for Android and Linux).

In the world of smart phones, Safari and Firefox/Chrome seem to have the dominant hand. Since Apple will not support Flash (due to its poor performance under Mac OS), Apple is developing its Web browser with advanced HTML5 characteristics (video, offline caching, JavaScript, CSS3, etc). Google Chrome is also developing similar features, as well as enhanced JavaScript processing speed.

It seems to be the belief that HTML5 will be able to fill the role (on smart phones) that Flash has played on the desktop. This remains to be seen, though HTML5 and Flash seem to be within the same order of magnitude in terms of performance.

Style sheets (CSS) and style sheets with 3D extensions (CSS3) allow an enormous variety of effects with just a few lines of code. The combination of JavaScript and CSS gives rise to what is known as dynamic HTML (or DHTML).
Chapter 5. Built in Programming Language

There are many different programming languages that have been integrated with HTML.

On the server side (where the language is parsed, and HTML is written and passed to the client) there were cgi-scripts (using Perl), PHP scripts, Python scripts, as well as others. Since server side scripting is not seen by the client, it is essentially independent of the client. This is a tremendous advantage, unless one wants to go completely offline, in which case the communications link breaks and the code is non-functional.

On the client side, JavaScript has been the scripting language of choice (although Java is another widely used language, requiring a virtual Java machine on the client). Java originally was developed with the goal of “write once, run everywhere”. In practice it ended up as “write once, debug everywhere”...

JavaScript is an interpreted language, like Basic, but it has grown immensely in what it can do. Some packages like JQuery (http://jquery.com/) give complete access to the Document Object Model (DOM) and are used as the basis for many sophisticated packages.

The packages jsMath and AsciiMathML use sophisticated JavaScript packages in order to make their actions more or less “transparent”, and allow a measure of high level user programming.

Chapter 6. Work Offline

One of the most useful features of HTML5 is the ability to download web pages and work offline. As long as all the necessary resources (i.e. JavaScript, CSS/CSS3, images, videos, etc) are downloaded then the web page will be able to be accessed offline.

If you require access to anything not in the cache (e.g. a server side script or a font) then the application will break.

This means that some of the very large math-rendering projects (jsMath, AsciiMathML, MathJax) cannot be completely downloaded to the client (some of the packages are 80 Mb or larger!) and hence cannot be completely standalone. MathQuill is small enough (less than 1Mb) that it can be downloaded in its entirety.

One caveat is that cache memory on smart phones is limited. On the iPhone/iPod/iPad devices, the offline cache memory is limited to 5Mb. This may change in the future, and may be increased in future OS releases.
Summary.

One of the most exciting aspects of these new standards (contained in HTML5) is that students (in the future) will be able to visit sites, be immersed in interactive text, graphics and video, and then continue interacting even when not connected to the network.

Algorithmic Instruction, Algorithmic Quizzes, even Algorithmic Homework working offline, will be a possibility.

One feature that we have not discussed is offline databases. WebSQL, SQLite, and other similar products have been implemented. Flat file database systems have even been developed. Cookies and session variables have been implemented, in order to store persistent information between sessions. Privacy issues clearly have been identified in this area.

Bonus Feature

As an added feature, I have developed a simple, easy to use “Clicker” application that will record the values of a slider and/or a text message from virtually any wired/wireless device with a modern browser. The numerical results are polled and displayed every 3 seconds in a popup window, and the verbal responses are collected in a table which is similarly updated.

Try it, and see how it works!