Title: Batching APIs, as applied to web applications.

Not for publishing. I need to make some changes first.

Abstract:

Batching APIs, as applied to web applications, is what this paper is about. It explains what a batching API is, with it's advantages, and limitations. Then it shows how to reduce latency for large amounts of content --- by combining things together. As well as explaining the reasons for the latency, it also discusses what advantages presenting 10x more content at the same speed can offer. One technique this paper discusses is packing images. Packing all of the images on a page into one big image - with efficient algorithms, then using CSS to place the image onto the page correctly.

Introduction.

[TODO] copy introduction back in.

Some examples of batching apis.

- Vertex buffers in opengl.   You can render a bunch of triangles at once... rather than one at a time.
- Execute many in the python db api. Allows you to do many SQL queries at once, rather than one at a time.
- fread() instead of getchar() from stdio. Reading many bytes at a time, instead of one at a time.

What are batching APIs and what are they good for?

Batching apis are about doing operations on multiple things rather than one thing at a time.

By using a batching api, you gain a few things:

- reduce initialization code.
- less function call overhead.
- opportunity to make optimizations which can not be done(or not worth doing) on one element.
- Easier to use, and reduces code size, by not forcing the user to redo a loop each time you need to use the function.
- Data parallelism is easier.
- Reduce the amount of locking needed.

If there are lots of objects which you are acting on, a batching api can give you major benefits.

A non batching api is presented here. Generally a function is made to act on one element at a time.

```
def do_something(a):
    # some initialization code.
    y = 2
    # the processing on a element.
    return a + y
somethings = []
somethings_append = somethings.append
for x in range(100):
    # call the function here
    r = do_something(x)
    somethings_append(r)
```

Here the batching api function is made to act on multiple elements at a time.

```
def do_somethings(a_seq):
    # some initialization code.
    y = 2
    somethings = []
    somethings_append = somethings.append
    # loop over the sequence.
    for x in a_seq:
       # processing on a element.
       somethings_append(x + y)
    return somethings
somethings = do_somethings(range(100))
```

My system currently does not act on many objects

Are you sure this will be the case in the future? Most systems end up acting on many objects instead of one.

If your system can scale up to handle more objects efficiently then it is more robust.

Designing your api as a batching one gives you little overhead(mostly none), and allows you to plan for the future.

Of course over engineering things is not the best idea, and if something is more elegantly coded as a function which acts on a single object, then that's great. However by designing your API to use from the beginning means that later it's easier to optimize that function.

When I was looking at making parts of pygame more suitable for threading I chose to look for batching APIs. Some of the pygame APIs do work in batches, but lots do not. There is pygame.draw.lines for example - that lets you pass it a list of points you want to draw lines. This function is much faster than calling draw.line many times, for multiple reasons.

For each line drawn it avoids:

- a python function call.
- a surface lock, and unlock.
- variable initialisation code.

Now because pygame.draw.lines is a batching API I was able to optimize it for multi threading without users having to change how they use the code.

Another bonus for batching APIs is that you can avoid the cost of releasing the python Global Interpreter Lock (GIL) for each line.

How to combine multiple images.

Heuristics for combining

There are many things to consider when combining images together. Mostly it depends on the web page that you are combining the images for. However there are enough similarities between web page uses of images to make things work.

Sort by size, and also by which images need to be shown first. If grouping by height first, then not as much space needs to be used.

- waste little space.
- fit the images quickly into a big image.
- what if the one in 50 images change?
    - use existing sets of big images.

- loading different images first. The images you want attention paid to, those images in the initial view port.

- colour depth of images. Gif images encoded into jpeg.
- lossiness of images. Some images require lossless.
- losing more detail by re-compressing images. Generational loss.
    - using source images when possible.

* which images change often or not at all?
* will there be new images on the page all the time?
* the cost of generating the combined images needs to be taken into account.
  - memory usage of combining images.

Changing images, and how this affects things.

Some web pages are more static than others. Figuring out which images change on a page is important for figuring out which images to combine.

If there are 20 images on a page, and they never change - then there are no problems with cachability. If one images changes on that page then the 19 other images will not be cached. The reason for this is that your master combined image will have to be regenerated. The browser does not know that the 19 out of 20 parts of that image are the same.

It is best to keep regularly changing images in a different set of images.

When an image is used multiple times on a page.

When an image is used multiple times on a page it might end up repeating in the image sheet. Using up more space - and therefore slowing things down.

So when constructing your master image sheet, make sure to remove duplicates.

Empirical measurements of performance improvements.

[TODO] put timing of page load times in here. Local loading, as well as remote loading.

I have made some measurements with a document containing a lot of images (113). One with all separate images, and one that combines the images together.

The original document:
http://rene.f0o.com/~rene/stuff/europython2007/website_batching/reviews.html

The document which uses combined images:
http://rene.f0o.com/~rene/stuff/europython2007/website_batching/reviews.out.html

This document has quite a few(113) tiny images. Each image is between 2-6KB in size. So

Time until finished loading:

Athlon 850mhz, swiftfox 2.0 (a firefox clone), linux - internet 250ms server, 8mbit ADSL -- cleared cache:

original 6.6 seconds
combined images 4.46 seconds.

IE 6, winXP p3 1ghz - internet 250ms server, 8mbit ADSL -- cleared cache:

original 33.0 seconds
combined images 11.0 seconds.

Safari 2.0.4, macbook dual core 1.8-2ish ghz- internet 250ms server, 8mbit ADSL -- cleared cache:

original 10 - 13.3 seconds
combined images 2.53 seconds.

So as you can see there's quite a massive speed up for Internet explorer, and Safari with a good increase for Firefox.

Athlon 850mhz, swiftfox(a firefox clone), linux - internet 250ms server, 8mbit ADSL -- cached image, reget html:

original 1.9 seconds
combined images 4.46 seconds.

Rendering is actually slower in this case for the combined images page. Most of the time is from downloading the css, and html. However there is also the time to render the page. From this we can see that rendering the page is the slow part of using the combined images with a slow computer, and when using firefox.

So I tried it with a much faster dual core macbook running OSX and firefox.

Firefox, macbook dual core 1.8-2ish ghz- internet 250ms server, 8mbit ADSL -- cleared cache:

original 30 - 37 seconds
original cached 0.197 seconds
combined images 2.88 - 3.38 seconds.
combined images cached 0.25 seconds.

Wow, this supprised me. The original page was much slower to load than the combined images page. I'm not sure why, maybe something to do with a slower HD, and slower file system compared to my linux machine. It is still faster to render the original page if it's cached... however the faster CPU brings the difference down to a tenth of a second.

IE 6, winXP p3 1ghz - internet 250ms server, 8mbit ADSL -- full cache:

original 0.64 seconds
combined images 0.57 seconds.

Using internet explorer here is faster because it is caching the html and the images. The combined images load slightly faster, but both are about the same speed.

I did not test what happens when the browsers disk cache is not in the OS disk cache. In this situation, I think that loading 111 extra files from the browser disk cache will make it slower. With a slower laptop HD the extra files will make a big difference.

In conclusion we can say that it is always faster on initial load to use the combined images. From 1.5X - 13x faster. On a computer with a slow CPU, and firefox loading the cached page will be slower.

File size.

Original individual images: 536KiB
Combined images: 243K

Note, that I don't know what jpeg compression settings were used in each case... so that could be part of the cause of the size difference. However there is also overhead for each smaller image - and combining images allows the compression to work better.

Also note that there is reduced overhead from less HTTP traffic, and less TCP overhead.

Network use.

The load is reduced on the server because there are less syscalls for loading the page total. There are also less disk seeks, because there are 111 files less to download. There was less cpu used, and there were less connections made - so less memory and other resources were used for the download. There were less entries made in the webserver log file(111 less).

507KB was downloaded with the original document.
114 requests were made with the original document.

450KB was downloaded with the optimized document.
3 requests were made to the server.

Memory usage of popular browsers.

IE 6, winXP p3 1ghz - internet 250ms server, 8mbit ADSL -- cleared cache, task manager used to measure memory:

clean start	11200
original	18540
combined images	15204

Here we see that it takes up less memory to render the combined images with IE 6.

Safari 2.0.4, macbook dual core 1.8-2ish ghz- internet 250ms server, 8mbit ADSL -- cleared cache:

Measured with 'top'.

	RSHRD	RSIZE	VSIZE
clean start	21.2M	14.1M	354M
original	21.6M	18.6M	356M
combined images	21.6M	20.5M	358M

It uses up more memory to render the combined images with Safari.

[TODO] put results of memory tests in here. How to best show them?

How much extra content can be shown at the same time - compared to another.

Recommended image sets.

- Gif images. For transparency, and lossless at low colour depths.
- jpeg images. Since they are lossy.
- Never changing images. Since these images never change it's good to group them together.
- regularly changing images. Since these are less likely to be cached.

[TODO] Why?

Not easily able to check if an image is cached.

So we have to make some assumptions about which parts change. Using assumptions like 'decorative images on a page will only change with a redesign.

How packing images for use on the web differs from packing images for 2d games, and packing for 3d games.

Using the packed images in a browser.

```
<style>



</style>

<hr>

<p>
Here we show how normally two images are placed - using the img tag.

<img src="reviews_files/matthew-bournes-swan-lake-review-listing.jpg" border="0" height="68"
width="86">
```

```html
<br>

<img src="reviews_files/lisa-miller-morning-in-the-bowl-of-night-review-l.jpg" border="0" height="68"
width="86">

<p> To img tags side by side.
<img src="reviews_files/matthew-bournes-swan-lake-review-listing.jpg" border="0" height="68"
width="86"><img src="reviews_files/lisa-miller-morning-in-the-bowl-of-night-review-l.jpg" border="0"
height="68" width="86">

<hr>

<p>
We use a div tag, and a background option.

<p>
The problem with this is that it will not allow selection of the image, and it also does not place on the
page in the same way as an image.

<p>
I tried to fix the placement with a "float:left;" style, but that doesn't work.

<p>
But selecting images is useful, however maybe not being able to select something could be ok for your
uses.

<div style="background: transparent url(reviews_files/combined_1.jpg) 0px 0px no-repeat; width:
86px; height: 68px; float left;"></div>
<br>
<div style="background: transparent url(reviews_files/combined_1.jpg) 0px -68px no-repeat; width:
86px; height: 68px; float left;"></div>

<br>
<p>Should be two images side by side, using divs.  It doesn't work.

<div style="background: transparent url(reviews_files/combined_1.jpg) 0px 0px no-repeat; width:
86px; height: 68px; float left;"></div><div style="background: transparent
url(reviews_files/combined_1.jpg) 0px -68px no-repeat; width: 86px; height: 68px; float left;"></div>


<hr>

<p>
We try and apply the clip directly to an img tag.

<p>
However this has the problem that the img tags need to be absolute.

<p>Clip only works with

<p>
The rect arguments are: rect(top, right, bottom, left)  However we can't use commas as IE is buggy in
this regard (even IE 7) so use spaces instead.

<br>

<img src="reviews_files/combined_1.jpg" border="0" style="position:absolute;clip:rect(0px 86px 68px
0px)">
<img src="reviews_files/combined_1.jpg" border="0" style="position:absolute;clip:rect(68px 86px
136px 0px)">
```

<br> <br> <br> <br> <br> <br> <br> <br> <br> <br>

<hr>
<p>
Here we use two divs to wrap the img tag. This clipping is done with the divs. It's more complex than previous methods, but somehow seems more 'correct' - because you use an actual img tag.

<p>The benefit here is that selecting, and other properties of an img tag work ok.

<p>
Note that you don't really need the style in the img tag... but theoretically a browser could use this information to show less of an image.

<p>
rect(top, right, bottom, left)

<p>
We use the top, and left parts so that it hides the border of the image. They should be the negative.

```
<div style="position:relative;height:68px;width:86px;float:left;">
   <div style="position:absolute; clip:rect(0px 86px 68px 0px); top:-0px; left:-0px">
    <img src="reviews_files/combined_1.jpg" border="0"style="position:absolute; clip:rect(0px 86px 
68px 0px);"/>
   </div>
</div>

<div style="position:relative;height:68px; float:left;">
   <div style="position:absolute; clip:rect(68px 86px 136px 0px);top:-68px; left:-0px;">
    <img src="reviews_files/combined_1.jpg" border="0"style="position:absolute; clip:rect(68px 86px 
136px 0px);"/>
   </div>
</div>
```

How to arrange the small images onto the big one?

Store images on top of each other if possible. Instead of having one really wide mast image, it's best to have it not very wide, and longer.

This is because images are stored in memory width first. So if you have many images all next to each other, then rendering them requires jumping all around in memory. Instead of jumping around in memory you want each image to be fairly close together in memory - since jumping around memory can slow memory reads. With slow computers you can sometimes see images drawing from top to bottom - that's how they are laid out in memory.

[TODO] diagram of how an image is laid out in memory.

Why do you want to download ten 'pages' of a website at once?

Slide shows of information. Show lots of information like a presentation.

Pre cache information - so it can be loaded in the background.

Lots of separate parts of a page. - As seen on many 'AJAX' websites many sections of a page load separately.

The problem with these pages is that it is slower to initially load. Only loading part of a page allows you to do something quicker - update part of a page without having to update the whole thing.

Protocols to load multiple URLs at once.

A GET is only allowed to be 4096 bytes long, maximum. So combining some very long URLs together could be problematic.

Mostly you shouldn't have problems combining urls - since mostly urls are fairly short anyway. You could fit more than 40 urls which are longer than a normal browser address bar.

Using a new made up method (eg, MGET) is not really possible, since so much internet infrastructure only supports GET, HEAD, and POST.

Encode multiple urls into one url using GET.

If using long urls, where all of them combined would be longer than 4096 bytes use POST. Unfortunately POST doesn't have the same semantics as GET, so using GET is preferred.

Using shorter URLs is an option.

- only combine GET urls, since using POST urls could have side effects.

Create new urls for sets of urls. For example, the 'front-page' url would be the url for a group of maybe ten other urls for sections.

Shortening the url query string variables is another technique that can be used. eg. instead of /?search= use /?s=

Problems with combining.

Different latencies of different sections of the page mean that the page will be able to start loading once the slowest part is complete. eg. The chat window in gmail seems to take longer than the rest of the page. If it was combined into the main page, then the whole page will not load until that slow part had finished processing.

References.

Fast rollovers: http://wellstyled.com/css-nopreload-rollovers.html
CSS Sprites: http://alistapart.com/articles/sprites
CSS clip images: http://www.seifi.org/css/creating-thumbnails-using-the-css-clip-property.html
Optimizing page load time: http://www.die.net/musings/page_load_time/
Mozilla pipelining FAQ: http://www.mozilla.org/projects/netlib/http/pipelining-faq.html