CSEP 576 PROJECT 3
CONTENT BASED IMAGE RETRIEVAL

Jeremy Calvert

1. Introduction

I opted, again, to write my project in Java rather than C/C++. I was able to reuse some utilities, and more significantly, some color clustering code from Project 1. I made use of two third party libraries.

The first library is Oracle BDB Java Edition (JE), a Berkeley B-tree based embedded database to persist application data to disk, so that I needn't reanalyze images each time the application is run. The database is log file based, and these files are included in my submission.

I also used Tiny Java Web Server (TJWS) to run a small HTTP server with which to host the user interface. It seemed to only make sense to choose a web-based UI for a search application, though as I'll argue below, the algorithms and data structures are not at all suitable for a web-scale search engine.

2. Region finding

My color clustering algorithm was taken from my best clustering submission for Project 1, with one exception. To correct a flaw in that implementation, rather than taking random seeds as the colors found at random points within the image, I took seeds chosen randomly in RGB space.

I automatically merge clusters that grow to be overly close to one another, while incrementing the threshold for "close" based on the movement of pixels among clusters. If pixels increase in volatility, the threshold for close increases so that clusters that are likely to be causing the volatility became more likely to be merged.

One bit of skeleton code that I had to recreate determined the connected components of an image, based on color clusters. By writing my own, I was not subject to the restriction of identifying less than 256 connected components. So I found as many as there could possibly be, and then eliminated small ones, defined as those smaller than a predetermined size. I chose 500 pixels or less.

The connected components finding algorithm I implemented was described early on in the class, and amounts to doing a breadth first search of multiple trees defined where nodes are pixels, and there is an edge between nodes if the pixels are in the same color cluster. If two trees are found to share a pixel, they are identified as the same connected component. In this way, identifying all connected components requires only a single pass over an image's pixels.

3. Region attributes

As suggested, I chose the following attributes for each region:

- centroid
- bounding box
- area
- cluster color
- texture

All of these are straightforward save texture. I used Local Binary Pattern (LBP) for describing texture within a region, with 8 neighboring points. So, for each pixel within the proper interior of a region (interior meaning all neighboring pixels are also in the region), we compute the pixel intensity at the eight symmetric points on the unit circle centered at that pixel. For neighboring pixels along diagonals, we use bilinear interpolation. Then, we compare these intensities with the intensities at the center point. If it's greater, assign a 1, less otherwise assign a 0, and assemble the bits to form a single (unsigned) byte, namely an integer between 0 and 255. We take the histogram of these integers over all points in the region, and normalize. The distance between these vectors among corresponding regions is used to provide a measure of similarity of textures between regions. This was done and stored as a Regions object for each image.

4. Region correspondence

I struggled for a bit to find a way to correlate regions between images, first trying to find some notion of similarity between shapes, then to come up with a reasonable greedy algorithm. I failed in both respects. In the end, I simply used the raw pixel overlap of the two regions as a measure of their correspondence, lining up the origin of each image. Granted, this ignores areas that are in one image but not the other, but for us, most of our images were of nearly the same size. This was done and stored as a Cooccurence object for each pair of images. This is a relatively sparse matrix of size NxM, where N is the number of regions in the first image, and M the number in the second. The overall size of this data is roughly the square of the number of images times the square of the average number of regions. Some optimizations could be made by taking advantage of the sparsity of the Cooccurence matrices, but the storage still grows with the square of the size of the catalog.

5. Distance measure

To avoid deciding, I allow for 3 distance measures, Color, Texture, and a combination of the two. I chose these for their relative simplicity. Since I had a relative measure of regional correspondences, which I stored in the embedded database, computing these measures amounted to simply taking an average of the pairwise distances of the color and/or texture vectors, weighted by the correspondence of that pair of regions. Unlike region and cooccurence identification, this is done on-line when a query is submitted.

6. Experiments and results

Each of the metrics seems to work fairly well. I think the combination of color and texture works best, but it's a pretty qualitative assessment and hard to say for sure (see conclusions). I am attaching a small screen shot of each of the requested result sets, but I encourage you to see these for yourself by:

1. Running
   ```
   > java -classpath lib/*:build edu.washington.csep576.ImageSearch
   ```
   ... from the same directory in which you find this document.
2. Pointing a browser to:
   http://localhost:8765/ui

This requires at least Java5, and has only been attempted on linux (ubuntu). In theory, it should work on windows as well.

7. Future work

One interesting direction to continue this work would be to incorporate feature detection and description, for obvious benefits, either used directly as a measure of similarity, or used in determining correspondence between regions.

I'd also be interested in seeing how use of a different color space would qualitatively affect the results. I stuck with RGB, but if it's true that other spaces have better correspondence between distance and human perception of color difference, this should markedly improve results making use of color distance.

A final interesting extension would be to try and work in some scale and shift invariance. For my region correspondence, I require similar sized images, so perhaps one might scale the images to some normal size, balancing respect for aspect ratio with need to crop, and then perform the region correspondence calculation. Shift invariance would be somewhat more difficult.

6. Summary and conclusion

I thought this project was a nice summary of several topics we covered in our course, and I particularly liked the liberty we were given in choosing how to implement it.

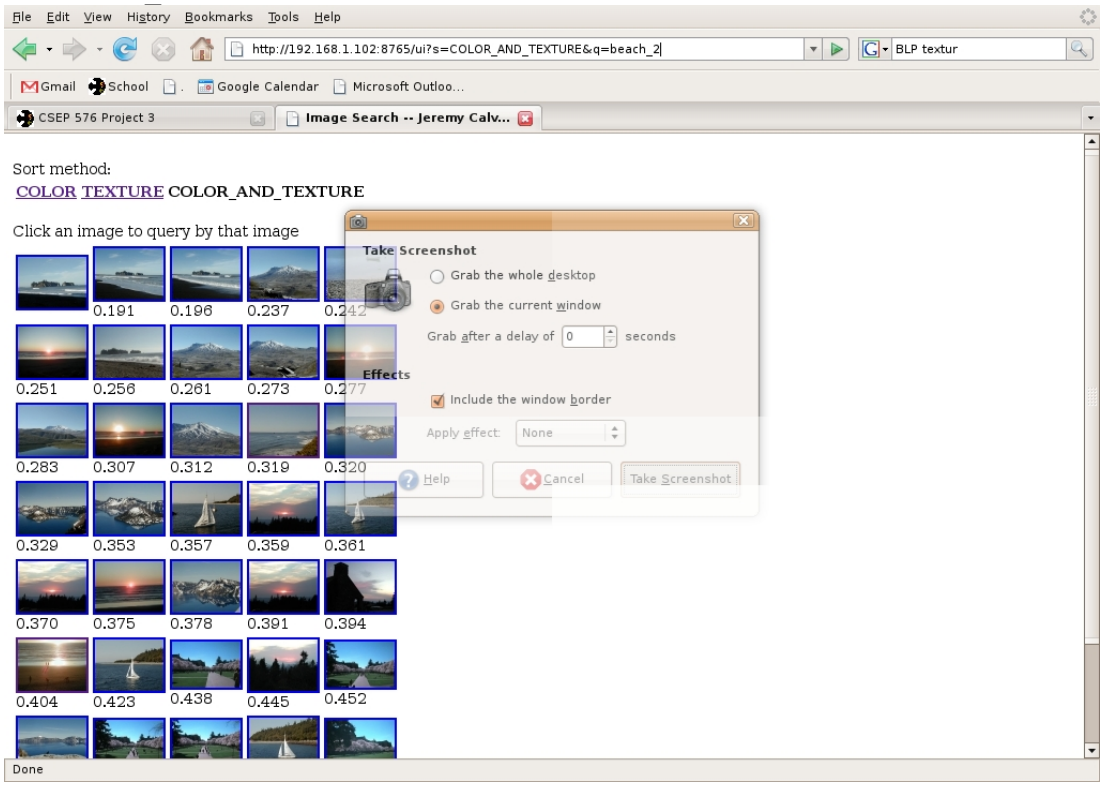
As for the utility of image-based search engines, I'm pretty skeptical. Here, by image-based, I mean engines such as this project, that use metrics on the images themselves as opposed to, say, google image search, which is essentially text search with images (in context) as results. There are two reasons for my skepticism.

First, images are inherently difficult to qualify. What makes an image what it is? It's invariably a combination of features, interests, subjects, and objects. And if we could quantify such beasts, on what basis would we do so? Which brings us to...

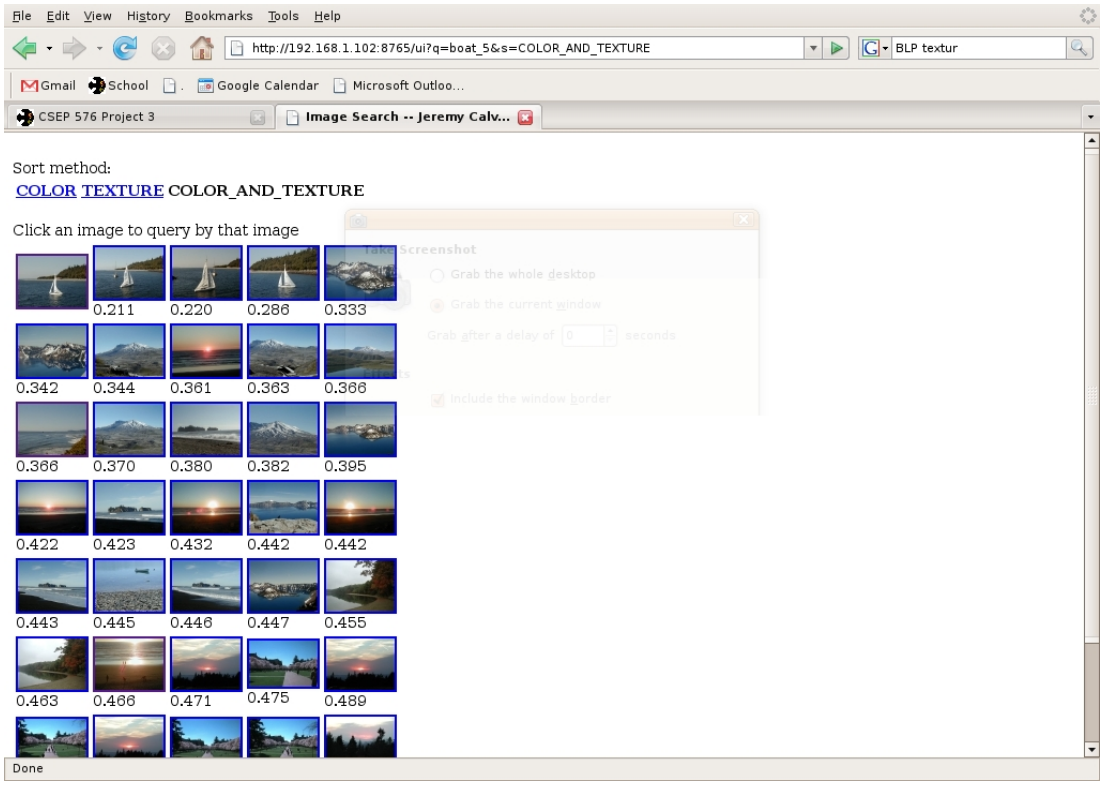
Second, what basis is even tractable? By the most liberal (including proper noun) estimates, the English lexicon is 40,000 terms. Despite the high dimensionality of its underlying space, the subspace of occurring vectors is ridiculously sparse. So we can make inverted, order sensitive, relatively small representations of such languages/subspaces. With raw image data, even conceding superb clustering, noise removal, and feature description, the simplest image has the expressive power of thousands of rare terms.

Appendix: Results of requested query images.

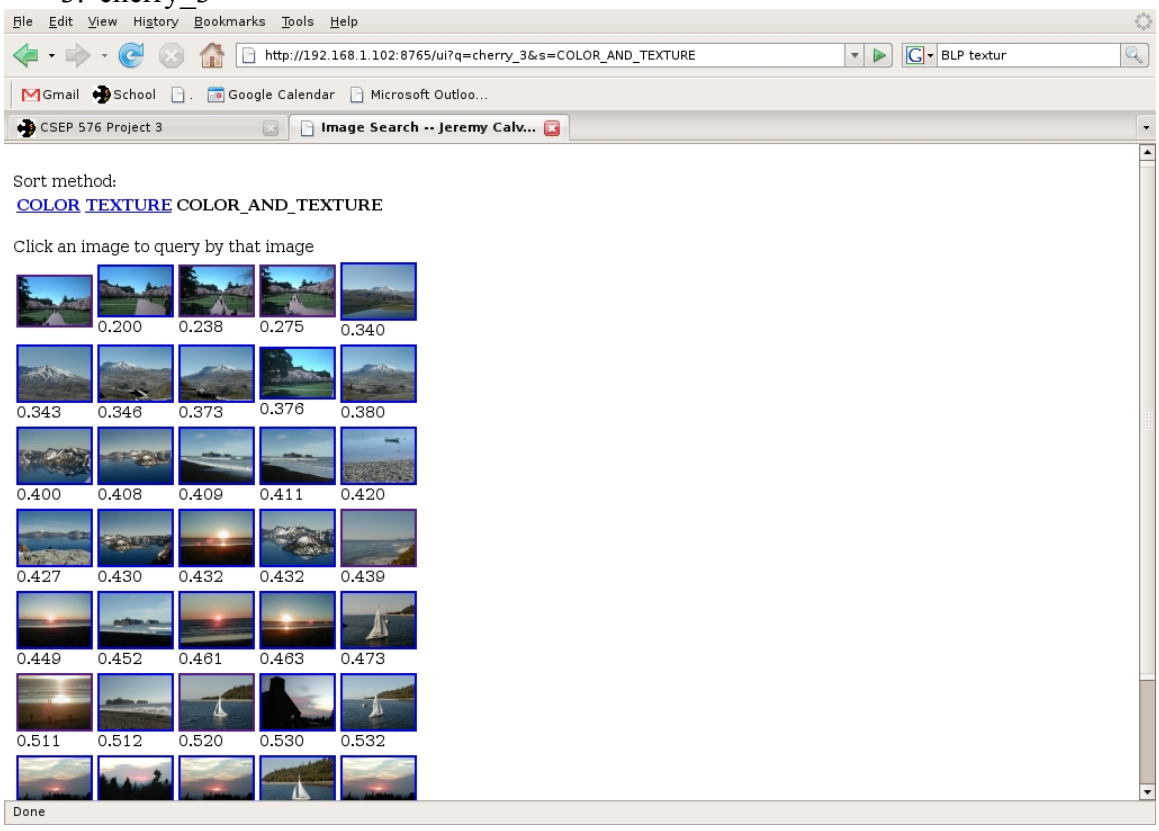
1. beach_2



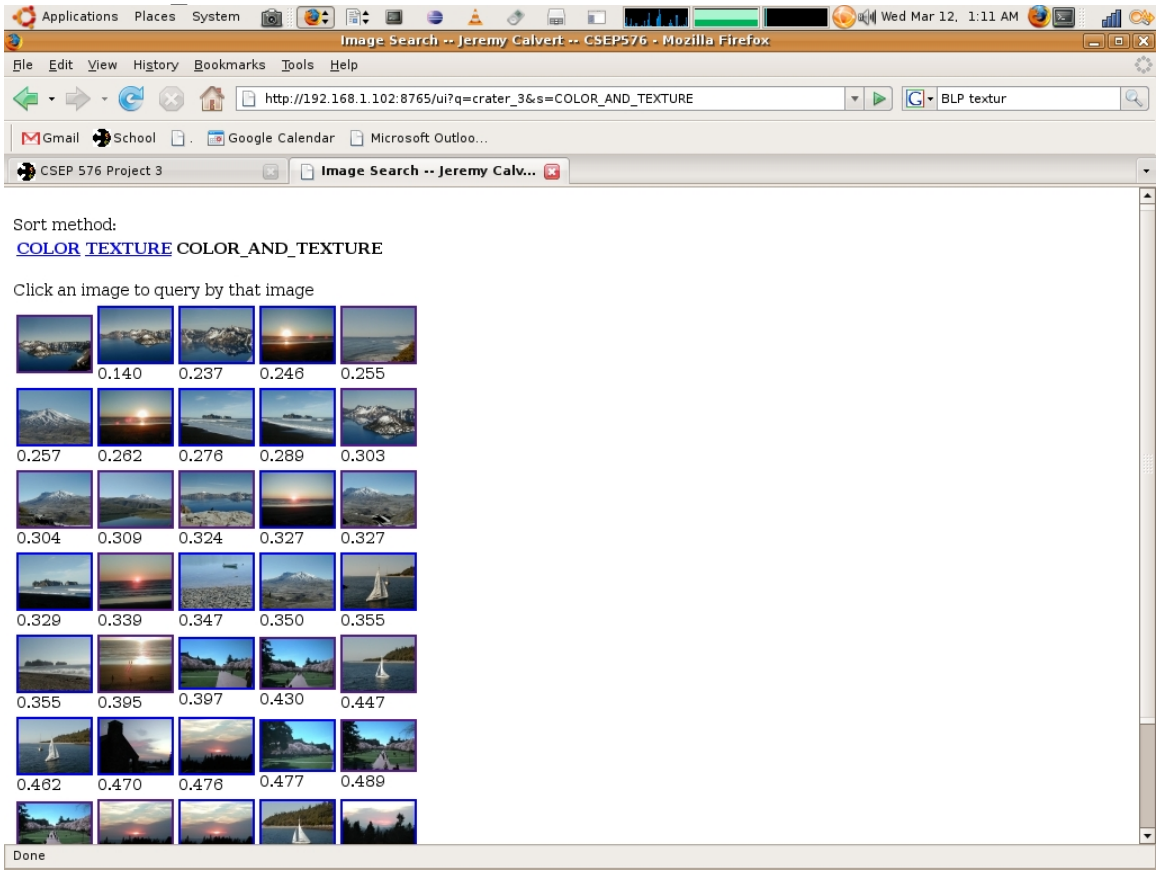
2. boat_5


File Edit View History Bookmarks Tools Help
http://192.168.1.102:8765/ui?q=boat_5&s=COLOR_AND_TEXTURE
BLP textur
Gmail School Google Calendar Microsoft Outloo...
CSEP 576 Project 3
Image Search -- Jeremy Calv...
Sort method:
COLOR TEXTURE COLOR_AND_TEXTURE
Click an image to query by that image
0.211 0.220 0.286 0.333
0.342 0.344 0.361 0.363 0.366
0.366 0.370 0.380 0.382 0.395
0.422 0.423 0.432 0.442 0.442
0.443 0.445 0.446 0.447 0.455
0.463 0.466 0.471 0.475 0.489
Done

3. cherry_3


File Edit View History Bookmarks Tools Help
http://192.168.1.102:8765/ui?q=cherry_3&s=COLOR_AND_TEXTURE
BLP textur
Gmail School Google Calendar Microsoft Outloo...
CSEP 576 Project 3
Image Search -- Jeremy Calv...
Sort method:
COLOR TEXTURE COLOR_AND_TEXTURE
Click an image to query by that image
0.200 0.238 0.275 0.340
0.343 0.346 0.373 0.376 0.380
0.400 0.408 0.409 0.411 0.420
0.427 0.430 0.432 0.432 0.439
0.449 0.452 0.461 0.463 0.473
0.511 0.512 0.520 0.530 0.532
Done

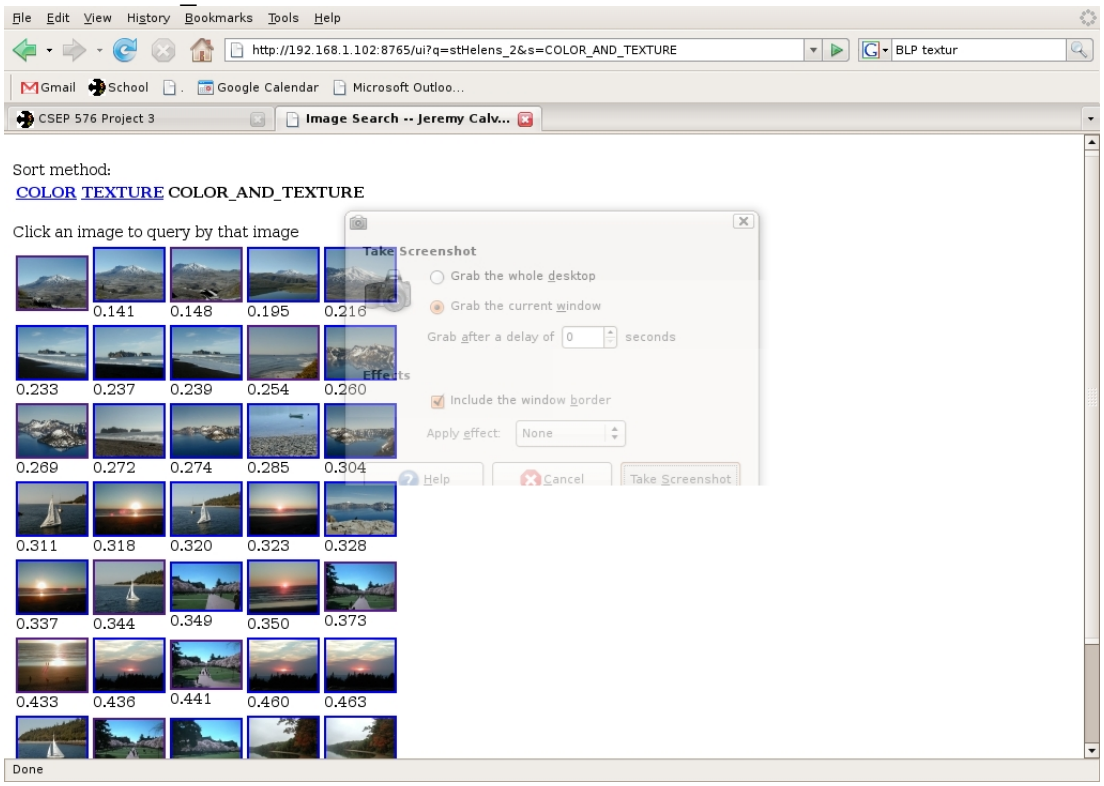
4. crater_3



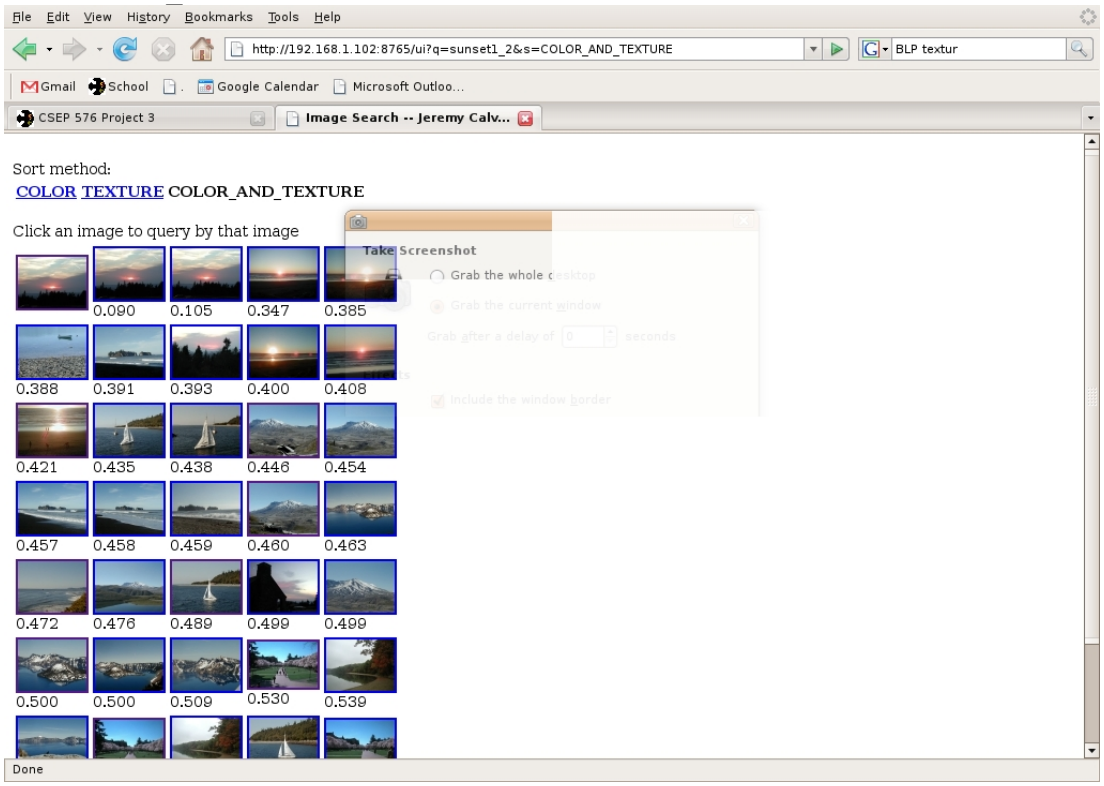
5. pond_2

No pond set :).

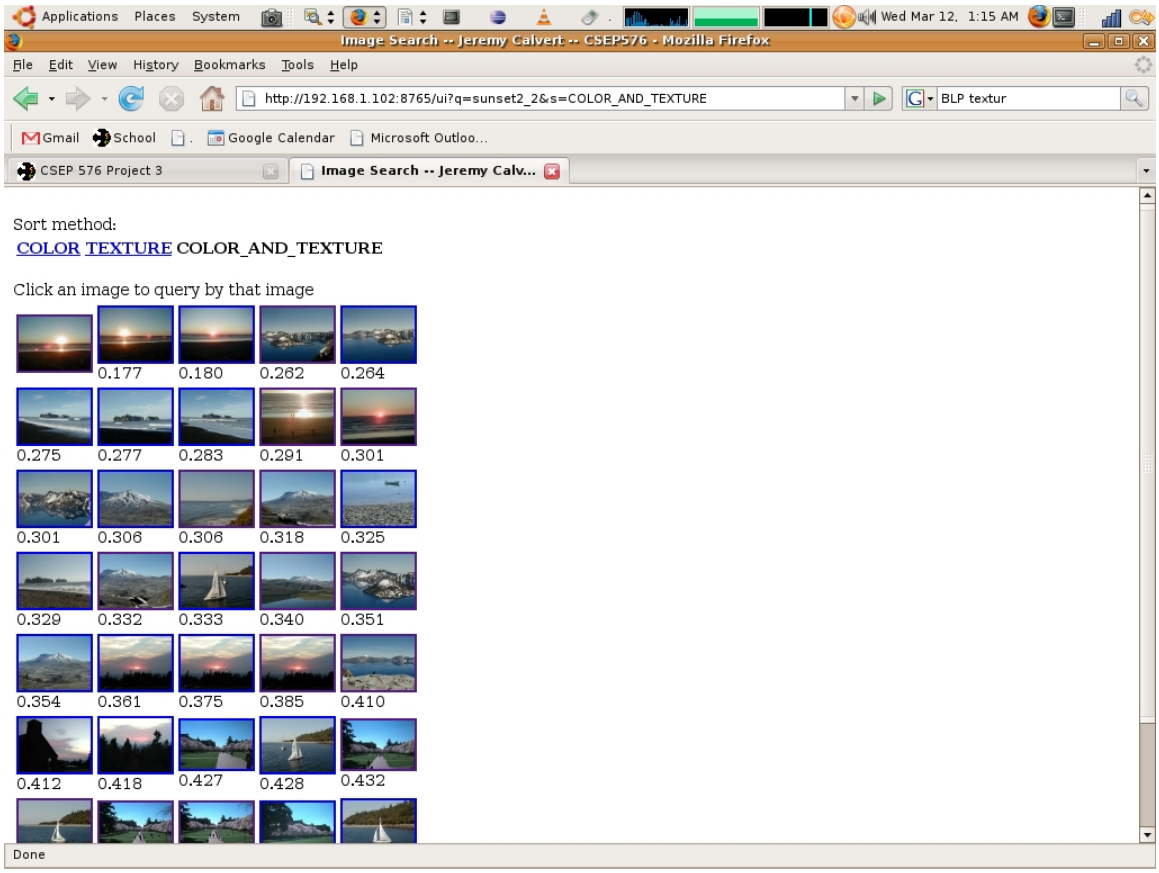
6. stHelens_2



7. sunset1_2



8. sunset2_2


Applications Places System
Wed Mar 12, 1:15 AM
Image Search -- Jeremy Calvert -- CSEP576 - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://192.168.1.102:8765/ui?q=sunset2_2&s=COLOR_AND_TEXTURE
BLP textur
Gmail School Google Calendar Microsoft Outloo...
CSEP 576 Project 3
Image Search -- Jeremy Calv...
Sort method:
COLOR TEXTURE COLOR_AND_TEXTURE
Click an image to query by that image
0.177 0.180 0.262 0.264
0.275 0.277 0.283 0.291 0.301
0.301 0.306 0.306 0.318 0.325
0.329 0.332 0.333 0.340 0.351
0.354 0.361 0.375 0.385 0.410
0.412 0.418 0.427 0.428 0.432
Done