

Visual Dynamic Gallery Engine

ideated and created by Alessandro Polo

July 2008

Abstract

It's a -proof of concept- album/image presentator, which groups and sorts images (of a folder).

Goals:

- Sort and display pictures in a clever way when there are not informations about them (but file system properties).
- Demonstrate that the algorithm works quite well (even without great math/optimization)
- Integrate some RAD Web2.0 technology such as YUI.

Often we have a set of images taken from a mobile phone or a digital camera, image is named adding a progressive index to the device's default prefix. Since the name is not useful, the most relevant parameter is timestamp. This is the target use-case of the system.

What's the innovation about it?

Mostly it's the concept of grouping and rendering groups.

Specifications:

- PHP 4.x, real-time grouping/layout
- Web2.0 support (YUI)

A Live Demo is available at: www.alessandropolo.name/pictures

Table of Contents

Visual Dynamic Gallery Engine.....	1
Abstract.....	1
Work Flow:.....	2
Grouping.....	2
Web 2.0.....	6
Drag&Drop (move).....	6
In-page Image preview.....	6
Conclusions.....	7
Note.....	7

Work Flow:

- configuration selector {folder, association, rendering, filters}
- file list loader (load media and related info)
- grouping { create basic media association and groups }
- layout { define position of groups and images }
- rendering { plot them }
- web2.0 { add some client-side features using YUI library }

Grouping

It's the first and one of most important step of the process. It has been designed to be independent of the layout/rendering, it's scope is to create some relational groups of given images.

Once groups are created, some statistics are evaluated for each group (group's parameters like average, variance, mse). Moreover, for each item of the group, the parameter(s) used during grouping are tuned and normalized.

How images are grouped?

Of course this is the main argument, since we assumed we have no particular informations about items but only file-systems properties (basically: name, time), grouping is actually based on time.

This is a very usual for sorting images, but the grouping concept makes a step further:

Given a set of items (a folder of images), these are the hypothesis we assume:

1. time is a key variable, easy to understand, always present as property of media;
2. time-stamp of a picture is correlated to its topic/context;
3. when there are many pictures taken in a short range of time, topic is more interesting and more correlated;

So, for each image, the algorithm evaluates the distance with the others (max $O(N^2)$), when the time-span is lower and a threshold, then the two image are correlated and registered into the same group.

```
Foreach IMAGE as IMG_REF {
  foreach IMAGE as IMG_CHK {
    timespan = IMG_REF.time - IMG_CHK.time
    if ( timespan < threshold )
      link( IMG_REF, IMG_CHK)
  }
}
```

where link() creates the group when the two images aren't already registered to any group, else it associates the image to the group of other image.

The algorithm may have a feature that is not explicit to everyone: a group is not a set of images taken within threshold time-span (ex. 60 seconds); but each image was taken within threshold time of another image.

Let's say we have a set of images taken periodically with $T = \text{threshold}-1$, then only one group will be created; if $T = \text{threshold}+1$, then no groups will be created. Of course threshold choice is very important and is configurable/adaptable. An improvement of the system may include a n-passes grouping with auto- adaptable threshold.

Some images may be not subscribed to any 'real' group, so these images are registered in a particular group (ID=0). Layout and Rendering of these images is processed while drawing groups, because these images will be placed outside normal groups.

After creating groups, some statistics are evaluated for each group and will be used by layout (rendering) engine.

Let's see a sample, in a set of 44 images in time range of 2 years, using 60 seconds as threshold:

```
ID: 9: ITEMS:18,36,37 | AV:36 | DM:20 |LMSE:242.666666667
ID: 11: ITEMS:20,24,29,25,27 | AV:29 | DM:53.2000000477 |LMSE:1029.76
ID: 12: ITEMS:28,32,41 | AV:41 | DM:28 |LMSE:458.666666667
ID: 0: ITEMS:11,17,42 | AV: | DM: |LMSE:
ID: 8: ITEMS:13,21,43 | AV:13 | DM:24 |LMSE:354.666666667
ID: 7: ITEMS:12,16,31 | AV:12 | DM:36.6666667461 |LMSE:726.222222222
ID: 3: ITEMS:2,5,33 | AV:33 | DM:17.3333332539 |LMSE:192.888888889
ID: 2: ITEMS:1,7,34,6 | AV:34 | DM:36.5 |LMSE:912.75
ID: 4: ITEMS:3,8 | AV:3 | DM:12 |LMSE:144
ID: 5: ITEMS:9,15,40,39,38 | AV:9 | DM:83.5999999046 |LMSE:3360.64
ID: 6: ITEMS:10,14,23,26 | AV:14 | DM:31.5 |LMSE:512.75
ID: 1: ITEMS:0,4,30 | AV:4 | DM:33.3333332539 |LMSE:651.555555556
```

UNLINKED: 11; 17; 42;

Let's watch details of group 11:

```
ID: 11: AV:29 | DM:53.2000000477
=====
ID:20: | TIME: 1169757942 |Delta: -1 |TS: 07-01-25 21:45:42 | Name: Foto (049).jpg
ID:24: | TIME: 1169757900 |Delta: -0.210526316497 |TS: 07-01-25 21:45:00 | Name: Foto (048).jpg
ID:29: | TIME: 1169757886 |Delta: 0.0526315780039 |TS: 07-01-25 21:44:46 | Name: Foto (047).jpg
ID:25: | TIME: 1169757846 |Delta: 0.804511276578 |TS: 07-01-25 21:44:06 | Name: Foto (045).jpg
ID:27: | TIME: 1169757870 |Delta: 0.353383457434 |TS: 07-01-25 21:44:30 | Name: Foto (046).jpg
```

Time is *UNIX epoch*, Delta is the normalized delta of previous algorithm (in fact we always have a n item with delta = 1 (-1) within a group, and it is the newest(oldest) image of the group.)

So, now groups and their statistics have been evaluated, we are ready to render them in some way.

Layout and Rendering

Some important aspects of layout and rendering are:

- we want to display thumbnails in a clever but also nice way (optically)
- we are working with HTML + CSS
- Canvas is 2D (X,Y), Y is reversed
- there are only squares (group is a square)



Let's forget a while about inter-group and HTML rendering and concentrate on intra-group layout.

- group center is the key-parameter (time) average.
- group center is origin (0,0), canvas data (top , left) is referred to this point.
- images' layout



An algorithm calculates position of each image $O(N)$, it is based on Polar coordinated and places images on a eclipse around group's center: angle and radius are variable.

```

Foreach IMAGE in GROUP {
  if ( abs(IMAGE.delta) < 0.9 ) swap = -swap
  angle = IMAGE.delta * 90 + 90
  radius = abs(IMAGE.delta) * $radius_max;
  IMAGE.top = radius * sin( deg2rad(angle) ) * swap
  IMAGE.left = radius * cos( deg2rad(angle) )
}

```

Let's analyze some use-case of the algorithm:

- $\text{delta} \sim 0$: image will be placed near center. (it's central image of group time-span)
- $\text{delta} \sim 1$: image will be placed on right center.. (it's most recent image)
- $\text{delta} \sim -1$: image will be placed on left center.. (it's oldest image)
- other will be placed from right (newest) to left (oldest) on the upper and lower sides (depends on 'swap' variable, optimizes layout switching the side on each image)



It may seem incredible that such an algorithm works so well, in fact it doesn't, these screen shots are taken after an optimization process that try to fix position of images reducing overlap and resizing group. This is how the previous group looks like without optimization.



So, how does that process work?

It's as simple as good, it's based on images' center distance and group's area.

Let's first see a particular but very common result of grouping mechanism:



The group (4) lists only two pictures; since the group's average is the (time) average of the two images, then normalization of delta will be the same (with opposed sign): 1 and -1. Because of that the images are placed on the opposite sides of the group and nothing is on the center (while with three images there is always one near it).

ID:	ITEMS:	AV:	DM:	LMSE:
4:	3,8	3	12	144
3:		TIME: 1173467870	Delta: -1	TS: 07-03-09 20:17:50 Name: Foto(067).jpg
8:		TIME: 1173467846	Delta: 1	TS: 07-03-09 20:17:26 Name: Foto(066).jpg

The optimization process will move the images to with following results:



As you seen in these two examples, optimization works in both ways: it can explode or implode images. Groups of two images are optimized in a bit different way:

```
in GROUP of IMG1, IMG2 {
    if ( evalOverlap(IMG1, IMG2) < 0 )
        explodeGroup()
    else
        implodeGroup()
}
```

where implosion/explosion of a group is implemented moving its images near/away center (multiplier is smaller/bigger than 1):

```
Foreach IMAGE in GROUP {
    IMAGE.left *= MULTIPLIER;
    IMAGE.top *= MULTIPLIER;
}
```

So group (its images) will be moved near center until there is no overlap, else it will be expanded. In a common case (group with more than two items):

```
Foreach IMAGE as IMG_REF {
    Foreach IMAGE as IMG_CMP {
        if ( getPicDistance(IMG_REF, IIMG_CMP) < THUMB_WIDTH )
            optimizePic(IMG_REF, IIMG_CMP)
    }
}
```

where optimizepic() moves one of the pics away:

```
given IMG_REF, IMG_CMP {
    IMG_REF.radius = sqrt(IMG_REF.top^2+IMG_REF.left^2)
    IMG_CMP .radius = sqrt(IMG_CMP .top^2+IMG_CMP .left^2)

    if ( IMG_REF.radius > IMG_CMP .radius )
    {
        IMG_REF.left *= 1.1;
        IMG_REF.top *= 1.1;
    } else {
        IMG_CMP.left *= 1.1;
        IMG_CMP.top *= 1.1;
    }
}
```

Optimization is obviously iterated, my current implementation is based on a recursive function.

Web 2.0

There are some nice extra features added to the plain PHP (server-side) engine, when *web2* option is enabled user is able to move (drag) canvas and groups, images are previewed in-page asynchronously (using DOM and events).

Drag&Drop (move)

Ability to move items is comes with YUI Drag&Drop library, code:

```
(function() {
    var canvas, group1;

    YAHOO.util.Event.onDOMReady(function() {
        canvas = new YAHOO.util.DD("canvas");
        group1 = new YAHOO.util.DD("group_1");
    })
})
```

and its required Javascripts from the online Yahoo repository:

```
<link rel="stylesheet" type="text/css" href="http://yui.yahooapis.com/2.5.2/build/fonts/fonts-min.css" />
<script type="text/javascript" src="http://yui.yahooapis.com/2.5.2/build/yahoo-dom-event/yahoo-dom-event.js"></script>
<script type="text/javascript" src="http://yui.yahooapis.com/2.5.2/build/dragdrop/dragdrop-min.js"></script>
```

In-page Image preview

This has been implemented ad hoc using DOM and events, let's see most significant code:

```
var counter = 0;
dom_previews = new Array();
dd_previews = new Array();

function showPreview(image) {
    counter++;
    dom_previews[counter] = document.getElementById('image_preview').cloneNode(true);
    dom_previews[counter].id = 'image_preview_' + counter.toString();
    dom_previews[counter].style.display = 'block';
    dom_previews[counter].childNodes[1].onclick = function ()
        { this.parentNode.parentNode.removeChild(this.parentNode); }

    var imgNew = document.createElement('img');
    imgNew.src = "/" + document.getElementById(image).alt;

    dom_previews[counter].appendChild(imgNew);
    var insertHere = document.getElementById('canvas');
    insertHere.parentNode.insertBefore(dom_previews[counter],insertHere);
    dd_previews[counter] = new YAHOO.util.DD('image_preview_' + counter.toString());
}
```

Global variables are used to create unique IDs and keep track of opened previews, of course user can open many and each preview is independent.

Where the original image preview canvas is defined as:

```
<div id="image_preview" class="images_preview">
    <div class="preview_btn_close">close</div>
</div>
```

While the close button is cloned within the parent node, the image object is created in run-time and its src comes from the thumbnail which was clicked:

```
<a href="#" onclick="showPreview('pic_16_1')" title="1">
  
</a>
```

Conclusions

Of course there is much more logic that wasn't explained, but this article is just a short presentation of the Visual Dynamic Gallery Engine proof of concept.

File Summary:

index.php	
class.apPic.php	apPic and apPicGroup classes
alpha.main.php	Short main script: <i>work()</i> and <i>render()</i> methods
alpha.config.php	Configuration (static) and user settings
alpha.loader.php	Load file lists (map times), create apPic instances
alpha.grouping.php	Create apPicGroup instances and associates images
alpha.layout.php	Plot the groups on a canvas
alpha.rendering.php	Write the HTML
alpha.web2.php	Add web 2.0 engagements
alpha.help.php	Show help page
alpha.admin.php	

Note

After the first alpha release, I verified a kind of issue with remote web servers: when you upload files their timestamp (mtime) is updated to current server's time. Current (quick) fix adds a timetable file to each folder, file is plain text and list (one image per line) its name and its original timestamp. Loading process intersects the file-system lists with this timetable.