Volume xx (200y), Number z, pp. 1–10

PileBars: Scalable Dynamic Thumbnail Bars

Paolo Brivio[1], Federico Ponchio[2], Marco Tarini[1], and Paolo Cignoni[2]

[1] University of Insubria, Varese - Italy, E-mail: {first.last}@uninsubria.it.
[2] ISTI-CNR, Pisa - Italy, E-mail: E-mail: {first.last}@isti.cnr.it.

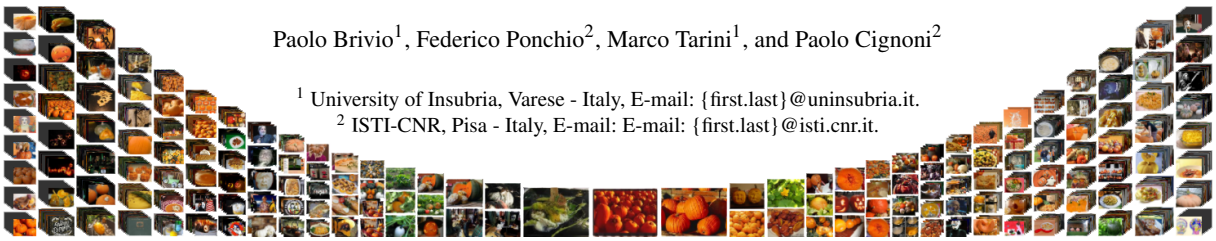

Figure 1: *A dataset of a thousand images visualized globally. Near the center, each individual image is visible as a larger thumbnail. At the side, images far from the area of interest are piled together into increasingly taller stacks. As the area of interest changes, the thumbnail bar dynamically rearranges accordingly with a smooth animation.*

Abstract

We introduce PileBars, a new class of animated thumbnail-bars aimed at easing the browsing of very large image datasets (thousands of images). Since the bar is meant to be just one element of a GUI, it covers only a small portion of the screen; yet it provides a global view of the entire dataset, without any scrolling panels. Instead, thumbnails are dynamically rearranged, resized and reclustered into adaptive layouts during the entire browsing process. The objective is to make the user able, at the same time, to accurately pinpoint a specific image (even among semantically close ones), and to jump anywhere to "distant" parts of the dataset. The used thumbnail layouts also maximize temporal coherence, thus allowing for smooth animations from one layout to the next.
The system is very general: it can be driven by any application-specific image-to-image semantic distance function, and respects any user-defined total ordering of the images; the ordering can be either inferred from the semantic or be chosen independently from it, depending on the application.
The applicability of the resulting system is tested in a number of practical applications: browsing image collections from the web or personal achieves; navigation of 3D virtual scenes based on calibrated pictures; selection of a frame in a movie. In each test scenario, proper distance and ordering functions are fed to the system.

Categories and Subject Descriptors (according to ACM CCS): H.5.2 [Information Systems]: User Interfaces—Graphical User Interfaces I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques

1. Introduction

In the aftermath of the explosion of digital imagery, large image datasets are becoming ubiquitous. Datasets composed of thousands or even several tens of thousands of digital images are made readily available in scenarios like picture collections harvested from the web, photographic acquisition campaigns in architectural, cultural heritage, archeological field (sometimes paired with parallel 3D acquisition of scenes), personal picture collections, image-based web-search, and others.

Dealing with image dataset of that size is not trivial. Even plain browsing becomes a difficult task, e.g. how to let an user locate one image, or how to visualize the entire dataset as a whole, or how to navigate thought the dataset.

In this context, thumbnail-bars are a very common browsing GUI mechanism.

A thumbnail-bar is but one element of an interface: only a small part of the screen space is devoted to it. They are used in the many end applications where the the main area of the screen is used for something else (e.g. to display the current image at full res).

1.1. Thumbnail-bars and PileBar

In a thumbnail-bar, each image is represented in the GUI by a smaller version of itself, i.e. a *'thumbnail'* (a downsized and/or cropped version of the original image).

Conventional thumbnail-bars display thumbnails over a panel, laid in a line or in a regular grid. When the panel size exceed the screen size devoted to the thumbnail-bar, the panel is made scrollable and scrolling mechanisms are provided to the user to navigate the dataset.

In presence of thousands of images this type of interface quickly becomes inadequate: either thumbnails are excessively downsized to be meaningful, or the amount of scrolling required becomes unfeasible. The problem is only partially mitigated by statically grouping images e.g. in folders, which anyway are not always available.

We propose a focus-plus-context oriented, general type of thumbnail-bar which we term *'PileBar'*, which targets scalability, i.e. which can be used on very large image dataset.

The challenge lies primarily in the conception of effective, new info-visualization and interaction mechanisms, but it also embeds a system design one: images in the dataset can be just too many for their thumbnail to be hosted simultaneously in RAM (or in texture RAM); also in cases where the dataset is hosted in a remote server, e.g. in web applications, it would be impractical to download the entire dataset before browsing it.

Often the images come enriched with meta-data, ranging from tagging (e.g. from "semantic web"), to time of shoot/creation, to per-image extrinsic camera calibration (position/orientation of the camera), and others yet can be extracted with various kind of content analysis. PileBars can take advantage of that by using them either as ordering functions or as semantic distance functions defined over the dataset, to aid visualization and navigation.

2. Related work

Image browsers are very common. Every modern Operating System embeds at least one, and with the ubiquitous diffusion of small digital cameras we can find browsers even in mobile phones. A conventional solution that image browsers often adopt is to present images on a linear thumbnail-bar [**?**] or arranged in a simple grid layout. As discussed these simple approaches do not scale up well with the number of images.

In order to improve the efficiency while presenting a limited number of images, a trend consists in following hierarchical approaches by clustering images exploiting some kind of image semantic, like combining time [**?**], time and space [**?**], color [**?**], spatial image-distances [**?**, **?**], or a mixture of them [**?**] to automatically, or interactively [**?**] compute image-clusters. Most of these works focus on the issue of finding a good clustering of images, while often consider less important the actual dynamic presentation and exploration of a clustered dataset. Our approach is orthogonal to these: we can build over many of these clustering techniques, while focusing on the issue of compact dynamic presentation of the hierarchy itself.

Other approaches try to exploit more sophisticated image arranging schemes like [**?**, **?**, **?**] that exploits treemaps-like techniques, or [**?**] that uses spiraling arrangements, or [**?**, **?**] that arrange the images in a 3D space relying on their previous calibration. With respect to these techniques, our approach offers a very compact presentation of many thumbnails, while retaining a hierarchical organization. It can also preserve the linear ordering of the images and leaves a significant space for the focused image or the presentation of a 3D environment.

In a very recent paper, Brivio et al. [**?**] presented a Voronoi based technique to arrange thumbnails that shares some similarities with PileBars. In their work, the authors exploit a total ordering defined over the image set and a Voronoi relaxation scheme is used to place thumbnails in a liquid free, well distributed way. The authors customize the Voronoi metric such that size and density of the thumbnails follows a focus+context schema. While the proposed approach shares some similar basic focusing idea, the PileBars technique overcomes [?] in a number of ways:

- Efficiency and simplicity - the presented technique does not have strong computational requirements and is quite easy to implement.
- Regularity and stability - piles are designed to be stable, while in [?] even small sideways thumbnails can move as a consequence of a relaxation process.
- Scalability - PileBars scales better and always present more informative clusters (i.e. the number of represented images is represented in the interface).
- Flexible clustering schema - our work supports different and adaptable clusterings, while in [?] a fixed a-priori clustering is provided, untied from the dataset image-features.

3. PileBars overview

In thumbnail-bars (including PileBars), at any given time, one image of the dataset is the current *'focus'*, selected by the user. The focus image can be regarded as the current "position" where the user sits, within the dataset. The exact meaning of selecting a focus depends on the application. In an image viewer, the focus can be the image being shown at full resolution in the main area of the screen; in a 3D navigator application, where images (here, pictures) are linked to viewpoints (the physical camera positions of the shot), the focus image can determine the position of the virtual camera used to render the 3D scene inside the main screen area (see Fig. **??**) [?]; and so on.

In a PileBar, thumbnails dynamically change size and po-

Figure 2: *Example of thumbnails piled inside a slot (shown dashed). From left to right: a pile of 1, 2, 4, 12, 100 thumbnails.*

sition during the browsing, and can be laid on top of each other forming *'piles'*, of varying heights. During the browsing, piles dynamically splits into smaller piles or are merged into larger piles. In a pile, only the top thumbnail is fully visible to the user. Therefore, it is important that the image it shows is a good representative of the images shown by the underlying thumbnails (which are hidden in part, or, depending on the stack size, even totally). Piling is a natural visual way of clustering similar images together, to make it possible to view the entire dataset.

The current focus determines the layout of the thumbnail-bar, i.e. the position, size and piling of every thumbnail in the dataset (Sec. **??**) (see Fig. **??**). A browsing session consists in a sequence of selections of a new focuses, made by the user. The change of current focus determine the animation from a layout of thumbnails to the next.

The main idea follows the spirit of "focus-plus-context" principle: images semantically closer to the current focus are displayed as larger thumbnails: this constitutes the "focus" part of the interface, where the user must be able to see, and tell apart, images close to its current "position" on the dataset. Conversely, images farther from the focus will be represented by thumbnails which are smaller and stacked in taller piles: this is the "context" part of the interface, where the user is given a global vision of the entire dataset. In the latter, the images fully visible on top of each pile are but a sub-sampling of the entire dataset.

A strength of PileBars is that there is a spatial and temporal continuum going from this "focus" to the "context" part. Spatially, the change of thumbnail sizes and piling is progressive, from the central focus thumbnail (large, isolated thumbnails) to the peripheral parts of the bar (small, piled thumbnails). Temporally, the transitions between a thumbnail disposition and another (triggered by a change of the current "focus") are smooth, meaning that the amount of change in size, position and piling is kept small by design, and it is also masked by means of smooth animations (Sec. **??**).

4. PileBars

As discussed, thumbnails in a PileBar dynamically change screen-size and position inside the thumbnail bar during the browsing. The current focus image determines a configuration (Sec. **??**), and as the current focus changes, the system animates to pass from a configuration to the next (Sec. **??**). The configurations are designed to comply with the focus-plus-context principle, as described above, but also to meet the objective of producing the least amount of change from the previous configuration, to help temporal coherence.

4.1. Assumptions and objectives

Total ordering: PileBars implicitly assume that a total ordering is defined over the images: i.e. images in the datasets are assumed to be (implicitly) numbered from the first I_0 to the last I_{N-1}. This is a central choice of our approach, as navigating a linearized dataset is intrinsically easier and more intuitive than navigating one disposed over multiple dimensions; this is especially with thumbnail-bars, whose shape have a single largely dominant dimension (in this paper, we assume it to be the horizontal one, but vertical bars are also possible).

A discussion about different possible orderings is in Sec. **??**.

An reason to adopt a total ordering is also that it provides an intuitive visual aid to the user during the navigation process: the total ordering will be reflected in the left-to-right spatial dispositions of the thumbnails: any image preceding in the ordering a given image will appear on its left. This is also valid for the focus image, therefore the left (right) half of the thumbnail will be covered by thumbnails of images preceding (following) the current focus in the total ordering.

Semantic distance function: PileBars *can* take advantage, in a natural way, of any meaningful image-to-image distance function $\{(I_a, I_b)$ available in the context of the dataset, and use this information as a way to drive thumbnail piling. Piling will put together images which are closer (according to this function) more often than images that differ much from each other. Images that are detected as different can still be piled together, but they must be farther away from the current focus. This way, the image visible on top of the pile will be a good representative of the images partially hidden behind it.

The same objective can also be restated in term of the set of 'top-thumbnails', i.e. the subset of thumbnails which appear as on top of a pile: while this subset must be sampled more tightly in proximity of the focus, and more sparsely away from it (for the focus-and-context principle), we also want the sampling to be approximatively regular in semantic space, for a given distance from the focus. For example,

a dozens outlier images must be lodged in a dedicated pile even in regions where other piles of more common images are composed of hundreds of images, so that they are still visible from a distance.

The definition of a image-to-image distance function offers another way to taylor the behavior of a PileBar around the needs to the application scenario. Several options, both very general ones and ones useful in specific applications, will be summarized in Sec. **??**. It should be noted that this is not a requirement for the PileBars: in absence of any helpful distance metric, images can be simply assumed to be all semantically equidistant. In this case, top-thumbnails will represent just a sampling which becomes increasingly sparser farther away from the focus.

4.2. Defining a PileBar shape

A standard thumbnail-bar has a few spatial parameters like size, thumbnail size, spacing between thumbnails, etc. A PileBar has many more degree of freedom and it is defined mainly by a configuration of *'slots'*, which partition the 2D region of the screen it covers (see Fig. **??**). A slot is a fixed rectangular region that will be occupied by a single pile of thumbnails.

Figure 4: *Another example of layout of slots (as in Fig* **??***), this time for a PileBar with a rectangular shape.*

Each slot S_i has a set of predefined fixed attributes: a 2D position p_i, a size d_i, and a *'stacking value'* k_i affecting the height of the piles that will occupy it: specifically, slot S_i is expected to host piles of around 2^{k_i} thumbnails. The special value $k_i = -1$ is used to impose that slot S_i will host piles consisting of *strictly* one thumbnail. These values don't usually change when the PileBar is used (barring resizing events, see Sec. **??**).

The slots and their attributes defines the shape, the size and the behavior of the PileBar, and can be designed freely to fit the needs of the hosting application.

A the slot S_0, termed *'central'* slot, is preposed to contain the focus thumbnail. That slot is maximal in size, central in position, and will carry a pile of only one thumbnail (e.g. it has a stacking value of $k_0 = -1$). Other $2n$ slots, S_i, $i \in \{-n+1..n-1\}/0$, are arranged on both sides (negatively indexed on the left and positively on the right), so that they partition the space of the PileBar. As mentioned, these slots becomes progressively smaller when farther from S_0 on the x axis, and have bigger values of k_i, as i increases. Moreover, consecutive slots on both side are grouped into a number of side-to-side *'columns'*; each column is composed of a number (usually 1 to 5) of vertically aligned slots of the same size and screen x positions.

Arrangements of slots following the above rules, like the ones depicted in Figs. **??** and **??** can be easily constructed procedurally, by selecting a few parameters determining the shape (height as a function of position), number of columns, number of slot per column and k values as a function of column index, etc.

4.3. Arranging and piling thumbnails into slots

In a preprocessing stage, each image I_i in the dataset, $i \in \{0..N-1\}$, is assigned to a integer number σ_i, which represents its 1D position into a 'linear semantic space'; the values are such that, for two consecutive images in the dataset (in the total ordering), the difference of their σ is proportional to their image-to-image distance (up to the rounding), like in the following pseudo-code:

```
σ_0 ← 0
foreach i in {0..N−2}:
  σ_{i+1} ← σ_i + ⌈K · {(I_i, I_{i+1})⌉
```

where $\mathcal{F}$ is the image distance function, and K is an arbitrarily chosen variable, used to make $\mathcal{F}$ map to different integer values. The ceiling function returns at least 1 (distances are always positive for different images), ensuring that no two images will have the same value of σ.

At runtime, each thumbnail T_i of image I_i is assigned to a slot S_j. The linear semantic space is subdivided into chunks of increasing sizes, and thumbnails of consecutive images with σ values falling into the same chunk are piled into the same slot. Images are processed from the focus image I_f, whose thumbnail T_f is assigned to the central slot S_0, to the last image $N-1$. Every time an image is found to belong to a different chunk of semantic space than the previous image, the next slot gets used. The size of the chunk depends on the clustering factor k_j of the currently slot S_j.

```
procedure Arrange:
j ← 0
c_last ← Cluster(σ_f, k_j)
foreach i in {f..N−1}:
    c_now ← Cluster(σ_i, k_j)
    if c_now ≠ c_last then j ← j+1
    Assign T_i to S_j
    c_last ← c_now
```

where Cluster is a function returning an unique index of a chunk of semantic space, given a position in linear semantic space σ and the clustering factor k, as described below. The process is repeated backward to assign all thumbnails for T_f to T_0 to a left-hand slot.

```
function Cluster(σ,k):
    if k=−1 return σ
```

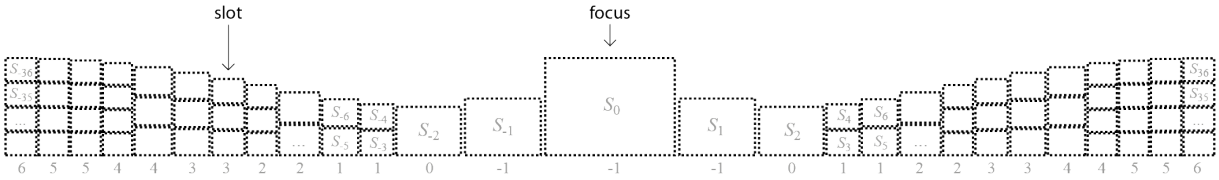

Figure 3: *An example of the arrangement of slots for a PileBar. Below each column of slots we report the value of k for all the slots in that column. Each slot will host a pile of, on average, 2^k thumbnails (or exactly one when $k = -1$).*

```
else return σ >> (norm+k)
```

where $>>$ is the right shift operation in base 2. In the $k = -1$ case, a cluster will span exactly one value of the linear semantic space, meaning that, as expected, the slot contains exactly one image. The integer value *norm* is a normalization factor used so that, when $k = 0$, there is approximately one cluster of semantic space per image in the dataset: $norm = \log_2(\sigma_{N-1}/N)$, rounded to the closest integer. For any $k \neq 0$, the semantic space will be clustered into approximatively $N/2^k$ chunks, resulting on an average of around 2^k images per slots. Note that a slot with $k = 0$ will usually contain one image, but could also contain many semantically close ones, whereas with $k = -1$ a slot will contain exactly one image.

Procedure Arrange (and function Cluster) are designed so that top-images are always be selected consistently, in the following sense: consider the set of top-thumbnails, i.e. the set of images that, being on top of their own pile, are fully visible (that set is composed of the images which are the first to be assigned to a new slot in procedure Arrange); if one image belongs to this set when it is at a certain distance from the focus, it will always belong to this set if the focus is moved closer. This is important in order to achieve better transition between thumbnail layouts.

Sort among *y*: while, as mentioned, the total ordering of images will be reflected by the ordering of thumbnails on the *x* screen axis, the distribution of thumbnails on the *y* screen axis, which is not attached to any semantic, is used to maintain temporal coherence as much as possible, i.e. to maximize the similarity with the last computed thumbnail distribution. To achieve this, we kee[track of the *xy* position that each thumbnail had in the last frame (Sec. **??**). Within each column of slots, slots are re-sorted vertically according to ordering on the last *y* values of the thumbnail on top of that slot. This simple heuristic ensures that, when, as a consequence of a new focus selection, piles divide/merge, they rearrange in a way that minimize the total magnitude of movements (see attached videos).

Determining position/size of thumbnails: once every thumbnail has been assigned to a slot, the *'Status'* of each must be determined: the Status of a thumbnail encompasses its *xy* position in screen space, its screen size, and its z-order. In a slot which ended up with just one thumbnail, that thumbnail simply inherits status values from the slot attributes (size, position).

When multiple thumbnail end up in a slot, they are stacked into a pile, with the 1st one on top and the other ones behind it, following the total order. The entire pile fits into the slot (see Fig. **??**): thumbnails are slightly reduced in size by a factor *r*, determined by the number n_p of thumbnail in the pile ($r = 1 - r_0 \ln(n_p)$, with $r_0 = 0.3$). The upper left border of the slot (or upper right, on the left-sided slots) is used to space equally the n_p thumbnails.

4.4. Transitions

Change in thumbnails disposition induced by choice of new focus is performed through an animation which interpolates smoothly between the old and the new layout. This is trivial to do, because the layout consist is a Status per each thumbnail, and Statuses can be linearly interpolated by linearly interpolating all their attributes.

For each thumbnail, the current Status s_i^c and the new Status s_i^n are kept for each thumbnail. A new disposition induced by a focus change reassigns all s_i^n. At every frame, we simply perform $s_i^c = h * s_i^c + (1-h) * s_i^n$, with a speed parameter h (we use $h = 0.9$). This results in a natural exponential movements that start fast and end slow, and the speed is nicely proportional to the traveled distance (when the distance between s_i^c and s_i^n is below a small threshold, s_i^c is snapped to s_i^n).

Animations of this kind also overlap in a simple and clean way: if, before the previous animation finishes, a new user interaction triggers a new layout, a new s_i^n is recomputed and the animation start from the current s_i^c, as normal.

4.5. Liquid PileBar shapes

Even if the layouts and attributes of the slots are kept fixed during normal data-set navigation, they can be easily readapted dynamically to accomodate, for example, a resizing of the PileBar shape.

Resizes which preserve aspect ratios of the PileBar do not affect anything else. The effect of changing the aspect ratio of the PileBar, instead, has a potentially useful and intuitive

effect: if the pile bar is reduced only vertically, and slot layout and aspect ratio is kept fixed, then the slot sizes will reduce so that they can fit vertically in the new shape. This will make more slots columns appear at the sides.

The values of slot attribute k can be made adaptive: if empty slot columns are detected on either size, the values of k of a few slots are automatically decreased so that the piles will partially unfold and spread into more slots, covering the empty space. Conversely, if slots inside screen space are not enough to hold the piles, the values of k are increased to produce taller and fewer slots, and so on until necessary.

The net effect is that, making the PileBar thinner will be reduced the thumbnail size, but result in less piling, thus revealing more images. Conversely, widening the bar vertically causes less slots to be visible at sides, thus resulting if larger but more clustered PileBar (see attached Video 5).

5. Interacting with the pileBar

The system provides several natural interaction mechanism with the pileBar.

A few of them require the user to pick a thumbnail with the pointer (e.g. mouse). Note that, thanks to the way thumbnails are arranged into piles (see Fig. **??**), not only the top thumbnail of each pile can be picked, but also to the ones behind it, by selecting their smaller area which is left visible at their sides.

The main interaction happens by changing the current focus.

Focus changes Ways to change the current focus include:

- pointing and *clicking* on an thumbnail anywhere in the pileBar. Selected thumbnails becomes the new focus;
- *dragging* a thumbnail with the pointer over the x axis into a new position causes the focus to change so that the user selected x position is matched by the layout induced by the new focus. This is the user action corresponding to a scroll action in a conventional thumbnail bar.
- *next* and *prev* keys (or the mouse wheel) cause the focus to change by one image in either direction.

The dragging action is very general: bringing a thumbnail in the place of the current focus is will make it the new focus; dragging a thumbnail away in the peripheral region of the pileBar causes the focus to change drastically, because of the exponential piling factor. The next and prev keys allow a fine tuning. However, enabling key auto-repeat, the focus can change by as many as 60 unit per seconds (e.g. a dataset of 1.5 thousands images in under 30 seconds): thanks to the animation technique (Sec. **??**), the net effect is a very fast douse of thumbnails seamlessly passing though the focus images (see attached video 4).

Additional application specific mechanisms (e.g. returning to a previously bookmarked thumbnail, image searches, etc) can be added to change the current focus.

Whatever is the reason that triggered the change of focus, the effect is that, due exponential effect of the piling nature, the small piles close to the focus will move very fast, while the taller piles at the borders will move at a much lower speed; all piles will progressively split into smaller piles (as the get closer to the focus), or merge into taller piles (as they get farther from the focus). Importantly, small changes of focus will cause correspondingly small layout changes.

5.1. Other interactions

Figure 5: *A screenshot of the effect of "peeking" at a thumbnail inside a pile.*

Peeking: The user can *peek* at thumbnails, even if they are far away from the focus, by positioning the pointer over them, while (for example) pressing the right mouse button. A peeked thumbnail image will be shown above the pileBar fully unoccluded and at maximal thumbnail size, regardless of its position and size (see Fig. **??** and Video 3). This can be useful to enlarge small thumbnails at PileBar periphery, and to reveal thumbnails partailly-hidden behind other in piles.

Vertical drags: the effect of dragging a thumbnail solely along the y screen direction is simply to change the position of the moved pile from a slot to another in the same column. As mentioned, the y positioning of piles inside one column is arbitrary

Reordering: When more than one total ordering among images is available and meaningful (Sec. **??**), the user can be allowed to switch dynamically from an ordering to another, e.g. by the click of a button, similarly to what happens a traditional image or file browsers (where all data is displayed on a grid, with one raw per item, and one column per attribute, and clicking on the causes items to be sorted according to the corresponding attribute). The new ordering will automatically induce a new disposition of thumbnails inside the bar (position, sizing, piling of thumbnails): temporal coherence can be maintained, as the current disposition animates as normal (see Sec. **??**) to reach the newly induced one. The net effect is that the focus thumbnail remains fixed, and the other thumbnails redispose around it. The change in layout here is much more drastic than with respect to the ones triggered by focus changes.

Resizing: as mentioned in Sec. **??**, the pileBar is liquid in the sense that it is capable of gracefully adapting to different sizes and aspect ratios. This happens when the entire application embedding the pileBar is resized, but the user can be also be given the opportunity to directly reshape the PileBar, redistributing screen space within the application GUI.

6. Image Ordering and Image Distance Functions

As mentioned, an image-to-image distance function is used in the system in order to visually cluster thumbnails relative to "close" images in piles. That function is used to capture some aspects of the semantics in the images, to drive the image piling in a meaningful way.

A strength of our approach is that any distance function defined between images can be profitably employed for this purpose, regardless of the range, the scale, and the distributions of the returned values. Computational time is not critique here either because the distance will be pre-computed only for pairs of images that are consecutive in the total ordering, which is feasible even for the largest image datasets (Sec. **??**).

There are plenty image distance metrics readily available, and it is easy to conceive new ones tailored around specific needs. For example, we found the following distance functions to be useful:

1. time (the distance in seconds between the shot times);
2. shot positions (the distance in cm between the shot positions);
3. shot directions (the angle between the two shot directions — note that, conversely, the rotation of the camera around the shot direction is usually not meaningful);
4. view frustum similarity (the number of feature points present in only one of either images);
5. average color (distance between average colors of images);
6. color distribution (distance between normalized color histograms);
7. color spatial distribution (averaged per-pixel distance between severely downsized versions of the two images);
8. tagging distance (distance in tag space).

Other image content-based distances can also be used. Clearly, not every measure is available in every dataset. For example, shot-positions and shot-directions are available when per image camera calibration data is available, as in datasets resulting from photographic campaigns processed with 3D photo recostruction techniques [**?**, ?, **?**]. Such images are commonly reciprocally calibrated by identifications of common feature-points. In these cases, the number of feature-points shared by two images can also provide an additional measure of image similarity (the 4th item in the above list). Spatial measures result in intuitive pilings, as images shot from the same locations are piled together, and further from the focus images shots from larger regions are piled together.

Most photographic collections (including most personal image collections) come with per-image time-of-shot data, which can be retrieved from the file system or from *exif* metadata. Time distances can range from a few milliseconds to several months, within the same dataset. This is not a problem, thanks to the normalization described in Sec. **??**. Time distances are often a very natural choice, as it results in images being piled (and dynamically re-piled) per hour, day, week, month, etc. (according to the distance from the focus).

The color based metrics (distances 5, 6, 7) are the simplest content-based distance functions, but clearly any higher level metric could be plugged in. The ones listed are always available, on *any* image dataset, and they are usually good indicators of image semantic similarity. Note, however, that while the system can accommodate for any image-to-image distance function, it does not need one: if no meaningful distance is available, a constant image-to-image semantic distance semantic can be assumed. In that case, images will be piled solely according to distance from the focus, and images appearing on top of piles will constitute a regular sampling of the images (but increasingly sparser the further the images are from the current focus).

Color metric distances are useful, for example, to pile together images belonging to the same shot in datasets featuring one image per frame of a movie.

Regardless of its origin, each distance function results in a scalar positive quantity returned for an image pair. Therefore, different distance functions can be blended together to form new *mixed* distance functions, weighting each component arbitrarily. For example, shot position and shot direction can be blended together into a measure taking in account both factors. Color metrics can be blended with spatial metrics so that, for example, images taken at night time will not be piled with images taken at daytime even if the shot position matches. In the final interface, the GUI designer can preset blended image distance-functions that are ready to use.

6.1. Examples of image ordering functions

As mentioned, a total ordering is assumed to be defined over the images.

Many applicative contexts come with one (or more) "natural" image orderings: e.g. a photographic campaign can use the ordering induced by the time of shot; web search results can be ordered by relevance; tagged images can be ordered alphabetically by tags, etc.

Useful orderings can also be inferred from any of the semantic function described above (including the mixed ones): the image dataset can be considered a weighted fully connected graph, and the (approximatively) minmal Hamiltonian path will define a linear ordering of the images. The fastest heuristic that can be employed to find these paths (the task is well known to be NP-complete) are require a quadratic number of image-to-image distance computations, and are too computationally demanding to be performed on the fly on very large datasets, so we have chosen to precompute the resulting ordering(s) and stored them for later use.

7. Implementation issues

Since the focus of PileBars is scalability with data-set size, implementation efficiency is crucial. Not only real time performances must be reached, but system resources must be left untapped for the rest of the application where the PileBar is embedded in.

Arranging thumbnails according to a new focus Sec. **??** is very efficient, and must be performed once per focus change. Animating the thumbnail layout Sec. **??** must be done once per frame, but is an even less demanding operation, boiling down to interpolation of a few scalar parameter per image. Tests show that both tasks present no performance hit even for the datasets with 10^6 images.

More care must be used with rendering performance, and memory/bandwith requirements.

7.1. Rendering system

PileBars can take advantage of basic rendering mechanisms hardwired (and heavily optimized) in any GPU card. A thumbnail is simply a textured quad, with a low resolution texture (not higher than the pixel size largest slot in the bar). Isotropic MIP-mapping works very well to achieve acceptable quality, continuous image reduction. Depth test can be easily employed to implement occlusions between thumbnails in a pile (z-sorting could be easily adopted as well).

However, plain rendering of several thousands of textured quads can still downgrade system performance. Technically, the problem is tied to the extreme depth complexity reached in piles holding hundreds of images, which tend to make the application fill limited (even if the per-fragment workload is very small). For example, a rendering of 10^5 thumbnails as small as 64×64 pixels results in more than 40 MegaPixels, which is as many pixels as in 40 full screens. Another problem is that displaying each thumbnail require a context switch to enable to correct texture.

Both issue are solved by keeping track, for each thumbnail T_i, of the index of the thumbnail T_j directly above it in the current layout (if any). At rendering time, we discriminate three cases:

1. T_j exists and covers T_i fully (in this frame);
2. T_j exists and leaves only at most one pixel of T_i visible (in this frame);
3. none of the above.

In the first case (which for example happens if T_j is fully aligned with T_i and same size) rendering of T_i can be skipped. In the second case, only a row and a column of pixels of T_j is rendered (with line primitives). In the third case, the thumbnail is fully rendered. In a very tall pile, static or moving, most rendering can be skipped; a minority will be rendered producing few fragments; only one will be fully rendered. Only form piles that are folds or unfolds during animation, and for the few very short piles, thumbnail will fall in the third case.

7.2. Cache system

A technical issue accompanying large dataset visualization is tied to the need of keeping in memory a very large number of thumbnails. Even considering that they can be much smaller than the real images they represent, the memory requirement quickly surpasses RAM and texture RAM capacities.

For example, a dataset composed of 10^5 images, with thumbnail sized 256^2 RGB pixel, will require 20 GBytes of raw pixel data in texture memory. Adding the requirement of prefiltering (MIP-mapping), the requirement rises to 27 GBytes. Adopting a texture compression scheme (DXT1, [**?**]) takes that number down to 4.8 GBytes, which is still too much, especially considering that the thumbnail bar is, likely, but one component of the application that embeds it. Additionally, in a distributed application, as a web application, it would be unpractical to exchange large amount of data between the server originally hosting the thumbnails and the client before the dataset can be browsed.

To address both issues, we adopt a strictly inclusive, multilevel, multithreaded cache system, similarly to [**?**], consisting of four layers: (1) remote server (in remote applications); (2) local disk; (3) RAM; (4) texture RAM, on board of graphic card. Each stage of the cache is executed in its own thread to avoid stalling the system on load/download.

In layers (1) and (2) thumbnails are stored in the compact JPG format. In layers (3) and (4) they are stored in the less compact but GPU friendly DXT1 compression. When passing objects from (2) to (3), fast JPG compression (libjpeg_simd) and fast DXT recompression [**?**] libraries are used; they are both capable of performing at about 100 MPixels/sec (meaning that computational resources are left untapped).

The total ordering of the thumbnails is used to assign a priority to each thumbnail equal to the image pixels actually shown on screen, computed as in Sec. **??**. The cache dynamically manages the available memory so to keep the highest priority thumbnails as close as possible to the last stage.

Thumbnails not fitting in memory, or not being downloaded/loaded yet, are shown as rectangles colored as the average color of the given thumbnail (which is pre-computed and fits into main memory). Given the priority function used, and assuming sufficient band and low network lag, this happen only where it is not visually distracting.

8. Application scenario examples

We tested PileBars in a number of application scenario.

1. Cavalieri square, Archeo01, Arche02: three sets of calibrated images resulting from a photographic campaign;

2. Pumpkin: set of images resulting from a Google search;
3. Toy: set of frames of a movie.

In table 1, we show data relative to each dataset, including ordering functions used (Sec. **??**), metric distance function used (Sec. **??**), and links to images in the paper and attached videos.

In Fig. **??** an prototypal application is shown using PileBars [APP NAME REMOVED FOR BLIND REVIEW].

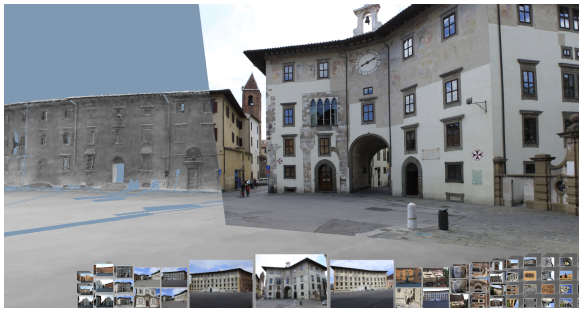

Figure 6: *An application using the PileBar (Cavalieri Square dataset). The main area of the screen depicts a 3D scene, navigable in 3D. Selecting a focus image on the bar causes the viewpoint to jump of the corresponding virtual camera.*

Results: Fig.**??** shows an example of a thumbnail arrangement produced by PileBars. However, a still image is inadequate to show its effect and the reader invited to see the attached video. Performance-wise, the system always goes at (60fps) performance on off the shelf computers, leaving RAM, CPU and texture memory untapped.

9. Conclusions

We have introduced PileBars, a scalable technique to effectively navigate, browse, and compactly visualize very large image datasets. The technique offers various advantages that have never been simultaneously present in previous techniques:

- in its basic form it is very easy to be implemented and can scale up to several thousands of images;
- it exploits clustered hierarchies to summarize set of images, but the approach is inherently independent of them and simply relies on a plain image-to-image distance definition;
- it is able to preserve the natural ordering of the images, so that during the browsing the user always knows where in the dataset he is focusing;
- it is designed to be visually compact, thanks to a clean regular image arrangement;
- it is dynamic yet as much as stable as possible: when focus shifts the changes to the arrangements are minimized in order to do not introduce distracting elements.

Moreover, thanks to all the above features, the proposed techinque implements in a GUI element that is well suited to be integrated as a non intrusive image browsing component within complex software applications.

9.1. Limitations and Future Work

Clearly, there are also a few limitations that, for sake of clarity, we want to mention. First of all the technique is tailored for images with a constant aspect ratio, so, when a significant portion of the images has an extreme aspect ratio, the proposed approach can waste some interface space. Then we realize that a thorough user evaluation is strongly needed in order to assess the actual impact of these methods on common image-related workflows, but we feel that such a study is beyond the scope of this paper and it will be the subject of our future work.

References

[Bed01] BEDERSON B. B.: Photomesa: a zoomable image browser using quantum treemaps and bubblemaps. In *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology* (New York, NY, USA, 2001), ACM, pp. 71–80.

[BTC10] BRIVIO P., TARINI M., CIGNONI P.: Browsing large image datasets through voronoi diagrams. *IEEE Transactions on Visualization and Computer Graphics (Proceedings Visualization 2010) 16*, 6 (November - December 2010), 1261–1270.

[EOWZ07] EPSHTEIN B., OFEK E., WEXLER Y., ZHANG P.: Hierarchical photo organization using geo-relevance. In *GIS '07: Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems* (New York, NY, USA, 2007), ACM, pp. 1–7.

[GGMPW02] GRAHAM A., GARCIA-MOLINA H., PAEPCKE A., WINOGRAD T.: Time as essence for photo browsing through personal digital libraries. In *Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries* (New York, NY, USA, 2002), JCDL '02, ACM, pp. 326–335.

[GMIL08] GOMI A., MIYAZAKI R., ITOH T., LI J.: Cat: A hierarchical image browser using a rectangle packing technique. In *Information Visualisation, 2008. IV '08. 12th International Conference* (2008), pp. 82 –87.

[Goo04] GOOGLE: Picasa. `http://picasa.google.com/`, 2004.

[GSW*09] GIRGENSOHN A., SHIPMAN F., WILCOX L., TURNER T., COOPER M.: Mediaglow: organizing photos in a graph-based workspace. In *Proceedings of the 13th international conference on Intelligent user interfaces* (New York, NY, USA, 2009), IUI '09, ACM, pp. 419–424.

[HBB07] HILLIGES O., BAUR D., BUTZ A.: Photohelix: Browsing, sorting and sharing digital photo collections. *TABLETOP 2007: Second Annual IEEE International Workshop on Horizontal Interactive Human-Computer Systems 0* (2007), 87–94.

[JYC09] JANG C., YOON T., CHO H.-G.: A smart clustering algorithm for photo set obtained from multiple digital cameras. In *Proceedings of the 2009 ACM symposium on Applied Computing* (New York, NY, USA, 2009), SAC '09, ACM, pp. 1784–1791.

[KB04] KHELLA A., BEDERSON B. B.: Pocket photomesa: a zoomable image browser for pdas. In *MUM '04: Proceedings of*

DATASET NAME	IMAGE NO.	ORDERING FUNCTION(S)	DISTANCE FUNCTION(S)	FIG.	VID.
Cavalieri Square	461	Shot position + orientation	time of shot	**??**	
Archeo01	430	Time of shot, custom metric	time of shot + constant, shot position		1,2
Archeo02	333	Time of shot, shot position	constant, images spatial colour layout	4	3,5
Pumpkin	992	Relevance, images spatial colour layout	images spatial colour layout, constant	**??**	4
Toy	10049	Sequence of frames, images color, images spatial colour layout	images color, images spatial colour layout		6

Table 1: *Images used and relative data*

Figure 7: *A screenshot of PileBars in action. Also refer to attached videos.*

the 3rd international conference on Mobile and ubiquitous multimedia (New York, NY, USA, 2004), ACM, pp. 19–24.

[MFGJ08] MOTA J. A., FONSECA M. J., GONÇALVES D., JORGE J. A.: Agrafo: a visual interface for grouping and browsing digital photos. In *Proceedings of the working conference on Advanced visual interfaces* (New York, NY, USA, 2008), AVI '08, ACM, pp. 494–495.

[Mic07] MICROSOFT: Photosynth. `http://photosynth.net`, 2007.

[PCF02] PLATT J. C., CZERWINSKI M., FIELD B. A.: *PhotoTOC: Automatic Clustering for Browsing Personal Photographs*. Technical Report MSR-TR-2002-17, Microsoft Research, 2002.

[RCC10] RYU D.-S., CHUNG W.-K., CHO H.-G.: Photoland: a new image layout system using spatio-temporal information in digital photos. In *Proceedings of the 2010 ACM Symposium on Applied Computing* (New York, NY, USA, 2010), SAC '10, ACM, pp. 1884–1891.

[RJL07] RENAMBOT L., JEONG B., LEIGH J.: Real-time compression for high-resolution content.

[SSS06] SNAVELY N., SEITZ S. M., SZELISKI R.: Photo tourism: exploring photo collections in 3d. *ACM Trans. Graph. 25* (July 2006), 835–846.

[Tsa86] TSAI R. Y.: An efficient and accurate camera calibration technique for 3d machine vision. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition* (Miami Beach, Florida, USA, 1986), pp. 364–374.

[VVG06] VERGAUWEN M., VAN GOOL L.: Web-based 3d reconstruction service. *Mach. Vision Appl. 17*, 6 (2006), 411–426.

[vW06] VAN WAVEREN J. P.: *Real-Time DXT Compression*. Tech. rep., Intel Software Network, 2006.

[vW07] VAN WAVEREN J. P.: *Real-Time Texture Streaming and Decompression*. Tech. rep., Intel Software Network, 2007.