UNIVERSITY OF PISA

DEPARTMENT OF COMPUTER SCIENCE

PRACTICAL IMAGE RETARGETING IN WEB PAGES

Author: Edoardo Alberto Dominici

In collaboration with: Prof. Marco Tarini

Supervisor: Prof. Antonio Cisternino

December 2016

Abstract

Many different retargeting solutions have been proposed in the past years for all different kind of media and especially for images. While they are somehow integrated in software editing packages, their use in more consumer oriented applications such as web browser is extremely limited if not non-existent. We propose a completely open-source and almost transparent frontend implementation for browsers that allows for images to be dynamically retargeted, thus giving the frontend developers the power to create truly fluid layouts without the current constraints of images' aspect ratio. It is based upon a robust, yet simple and performing state-of-the-art retargeting algorithm which is used in an initial step in order to enrich images with custom metadata that is later interpolated by the frontend Javascript library to effectively do content-aware retargeting. As algorithms have gotten incrementally more efficient, both memory and time wise, we think they are ready to be used in production and web pages are the perfect testing ground. The project is completely open source and hosted on Github: github.com/sparkon/retarget

This is dedicated to Aramis.

Contents

1 Retargeting of visual media **4**
1.1 Different types of retargeting 4
1.2 Images 5
1.3 Current direction and state of the art 7

2 Axis-aligned image retargeting **9**
2.1 Introduction 9
2.2 Inner workings 9
2.3 Energy systems 11
2.4 Why and how we used it 13

3 Implementation **14**
3.1 Approaches 14
3.2 Backend: retarget-lib 15
3.3 Backend: retarget-make 17
3.4 Frontend: retarget-js 17

4 Tools and Workflow **21**

5 Conclusions and further improvements **27**

1 Retargeting of visual media

1.1 Different types of retargeting

Due to today's extreme variety in reproduction technologies the need for retargeting different kinds of visual media has risen and such techniques are increasingly more integrated in modern software. Even though the most known field of retargeting is indeed imaging, adapting a media designed with a specific reproduction device in mind to a destination device with different characteristics, while preserving as much as possible its properties, is a problem that can be found in many different fields.

Examples include 3D modeling where a complex scene can be divided into components so that structural properties such as simmetry and regularity can be exploited to aid the retargeting transformation in preserving connectivity components[14]. Another interesting field of application that is more and more relevant today is inside toolchains for procedurally generated content. Where the computer cannot produce pleasant results the human's hand usually comes into play but, in order not to create content for all the possible, sometimes infinite, generated resources, some kind of retargeting has to take place, where the general purpose data is applied to the randomly generated one. One production ready example of such technologies is the one behind Spores [3], where the player is able to create custom animal morphologies and each one of them has different challenges when it comes to animations. Even though the content is not directly generated by a CPU, there are no real constraints on what can be achieved and the creation of all the possible key-framed animations for the different morphologies is unfeasible. All of the above has to be done while maintaining professional level quality. Spore's developers [3] answer to this problem in a two-step retargeting work flow, by having the animators generate semantic data in a first generalized form, that is then subsequently specialized (retargeted) at user's creation time.

The aforementioned techniques are extremely useful in real-world scenarios but they have the inherited advantage that, for the most part, they are not real-time and processing is allowed not to be instantaneous. Obviously the timing may vary and it still has to be reasonable for rapid development iterations and usability, but there is no hard upper limit on the processing time. Continuous media such as videos unfortunately have that. Although not spread in consumers' video players [KLHG09][7] proposed a GPU-accelerated system to retarget streaming videos that relies on multiple content analyzation post processing detection techniques for saliency, edges and motions to generate a non axis-aligned warp grid that can be applied to the input image in order to generate the output stream at the desired resolution. The motivation behind it is that resolution adaptation might not suffice in certain types of scenes, especially in small size displays [1]. Another example of retargeting that applies 2D motions to different shapes is presented in [RG08][4], that differs by shape-based cartoon motion captures techniques and presents a transform-based approach that is then retargeted preserving the visual style.

As it can be seen retargeting is quite a general term that is applied in different contexts with different scopes, and might not always be referred as such. The other important field where retargeting is critical is imaging. As it is the focus of this thesis we will now present a general overview of image retargeting and its current standard. Subsequently we will provide a practical example of how relatively modern techniques can be used on today's hardware to enrich end-user's experience.

1.2 Images

As suggested in 1.1 retargeting is the process of adapting a source media, for what concerns us a 2D image, designed with a certain display in mind, to a destination display with different specifications. Characteristics vary and include, but are not limited to dynamic range, gamut, resolution and refresh rate. [BAA et al.][10] presents an overview of different retargeting techniques that address the lack or presence of the aforementioned features. With the sheer number of different devices being sold today from ultra-high-definition televisions to low end phone displays in portrait mode, the dynamic resolution problem is prevailing.

The usual approach to retargeting an image starts with the source image in its original resolution and is followed by the generation of an importance map, that can be automatically created starting with simple edge detection or using non context-aware techniques [11] that rely on histogram-based contrast detection, up until algorithms who are specifically designed to identify the predominant objects and their surrounding context in the scene [12]. The importance map is a general term that encompasses different characteristic, another term that has a more precise definition is *saliency*. While the latter could be considered a subset of the former, there terms are often used interchangeably. Koch and Ullman in "Shifts in selective visual attention. Towards the underlying neural circuitry" defined saliency at a given location as the difference of that location from its surround in color, orientation, motion, depth, etc.. From now on we will mostly use saliency but we will be specific in case there is a different meaning to it. There is an abundant number of saliency detection techniques that may suit different kind of images. This system has the two inherited advantages: being modular and being offline. The former refers to the fact that the retargeting operation itself is decoupled from the saliency detection, meaning that different systems can be plugged in on a per-image basis. The latter is that the saliency map can be created once prior to the retargeting itself, thus more advanced and time consuming techniques can be used. Until now we have assumed no human interaction, but the automated generation can be only the initial step after which the artist finishes the job by tweaking the image in the zones where the algorithm fails. Additional parameters can be provided depending on the retargeting function in use.

For the retarget function many different approaches exists, but for the most part they can be grouped in:

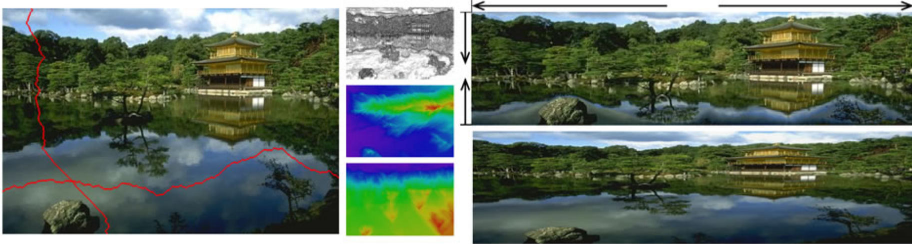

Figure 1: Seam-carving in action. From left to right, input image with initial seams, maps driving the algorithm and the final result compared to what would happen without retargeting [2].

- Pixel-based
- Mesh-based warping

Before moving on and introducing the different techniques a note has to be made clear. Retargeting is a process that is strictly correlated with aspect ratio, **not** resolution, meaning that up-scaling or down-scaling an image maintaining its original aspect ratio is a straightforward process, as no *stretching* occurs. Before moving on some terminology has to be defined:

- *Image*: 2D array of pixels containing color data in some color space. Retargeting is usually well suited for photographic images, not logos, nor vector art.
- *Content*: Set of *regions of interest* (ROI) for an image. Photographic images are often identified by one or more focus subjects that are not necessarily people, but any relevant object in the picture's context. All the subjects are from now on referred to as *objects* and are vital for retargeting as their importance in the scene is what guides the process itself. It is clear that retargeting might not work extremely well for certain images, such as landscapes, where no specific objects can be identified. An *object* is generally referred to as *region of interest* but, for the purpose of this overview, they have the same meaning.
- *Energy* is a loose term that varies depending on the context: in its general form is a mean of describing the amount of information. Thus "minimizing the energy change" means trying to avoid information loss as much as possible when operating on the image.

Finally, if not specified all the following algorithms assume an input importance map and optional human input. Suppose that the destination image height is less than the source image, the most naïve approach would be to discard those pixels that are not important in the context of the scene, this is what *cropping* revolves around. In its most basic form it simply drops entire rows and columns of pixels from the border while trying to keep the overall saliency

value of the resulting image as high as possible. Thus column 0 is dropped over column width-1, because it contains more *object's* data. Although it might not seem to produce pleasant results, allowing the retarget function to drop any continuous line of pixels in the image dramatically improves the output. Seam carving [2] does exactly this, using dynamic programming a vertical and an horizontal seam are chosen every iteration protecting the content of the image, thus avoiding *objects* and minimizing local energy dispersion. Subsequently pixels are either dropped or duplicated depending on the destination's resolution (Figure 1). Improvements on the paper, such as [5] proposed to choose pixels whose removal reintroduce the least amount of energy in the whole system.

While the previous *cropping* based image techniques have one single output image, there are interesting derivatives of *rapid serial visual presentation*(RSVP) who propose to simply extract the *objects* and display them sequentially. The trade-off here is between an obvious improvement in the image quality, as **no** stretching occurs, with the time and learning drawbacks. Once all the *objects* are separated they can also be merged together by introducing seams in a non photo realistic manner [1].

Unfortunately all of these techniques suffer from the complete loss of data after the operator is applied, mesh based approaches overcome this aspect by preserving almost all data, *objects* and their context. They work by either placing a fixed mesh over the image, or computing one from the image and non-linearly distorting the vertexes based on the input importance map, as the vertexes maintain their original texture coordinates sampling from them after they have been displaced produces the distorsion. Computing the distortion is essentially a non-linear optimization problem, but clever ways have been found to work around that problem. [6] allows all the regions in the image to absorb the distortion, and even if the resulting optimization problem is non-linear, it iterates over the linear minimization problem of determining the per-quad transformations with fixed constraints until a certain threshold has been reached. Another grid based approach that will be thoroughly discussed in the next section is the one proposed in [13] that places a low res grid over the source image and simplifies the problem by allowing only axis-aligned transformations. This renders the system a convex quadratic programming problem that can be efficiently solved. For a more detailed overview [10] provides a in-depth review of the evolution of image retargeting techniques up 2010.

While this section is not as in-depth as the aforementioned review, it wants to roughly introduce what image retargeting techniques are based on, and how constraining and simplifying the problem does not mean a reduction in quality but it does allow a potential use in more responsive user applications.

1.3 Current direction and state of the art

Browsers and thus web pages can be considered the major consumers of images and as it currently stands there is no real context-aware retargeting happening. Some tricks are used to try and preserve some image quality [15], especially when downscaling and pixels are lost, but they have no notion of what the

image contains. For this very reason modern web pages with fluid layouts try to preserve the aspect ratio of the source image by any means. This has the clear advantage that the image can be used as it is, but it creates an obvious discrepancy with the rest of the content. While `text`, `divs` and other HTML elements can be resized at will without any ratio constraint, images do have that and often force their containers to scale accordingly. The implementation based on the algorithm explained in the following section addresses this exact problem, by not forcing any aspect ratio on images and thus making them act homogeneously as all the other elements. This enables frontend developers to take advantage of it and design web pages that are truly fluid. Modern retargeting techniques combined with the vast support for hardware accelerated rendering (WebGL) in browsers make this the perfect time to show how real-time image retargeting can be part of everyone browser experience.

2 Axis-aligned image retargeting

2.1 Introduction

We will now present the retargeting algorithm that has been used as backend for the implementation. More reasons why it has been picked over the alternatives are discussed in 2.4. The technique will be explained carefully for the subjects that interest us, the paper [PWS12][13] also goes further in describing additional features that have not been considered for the current implementation. A mesh-based retargeting solution is proposed that only allows axis-aligned deformations to be performed in the retarget function. This reduces the possibility of certain rotations or displacements that are detrimental for the image to take place. Having 1D transformations, as they are separated between the horizontal as vertical cells, limits the possible retargeting scenario, but simplifies the system to a quadratic convex optimization problem, while at the same time providing good enough results (Figure 2).

> The space of axis-aligned deformations is the appropriate space for content-aware image retargeting.[13]

It seats in a sweet spot where on one hand it is perfectly usable as a real time solution and on the other it does not suffer from imperfections caused by fast or partial computation of other approaches.

2.2 Inner workings

As previously mentioned the problem of deforming an image accounting exclusively for axis-aligned transformation can be casted to a quadratic program that is linear in the size of the grid used for deforming. As with most mesh-based retargeting techniques the process is driven by a energy equation that evaluates the transformations to be applied based of an input saliency map. Some algorithms strive to minimize the local energy changes, while others account for it globally.

The main issue with earlier image warping techniques is that they would either be extremely efficient by using linear energy systems, but be very prone to artifacts in the final output, or overcome the artifact issue by over-complicating the energy system itself to account for border cases. Interesting approaches like [FCK et al.][9] get extremely close to a valid solution by formally preventing foldovers and describing a globally optimizable energy function based on quadratic programming formulation. It also strives for axis-alignment by trying to preserve straight lines by penalizing certain kind of deformations. The downside is that if too many constraints on the energy equation are posed it can turn out to be infeasible, in that case those constraints have to softened.

Another approach that strongly indicates having straight line as a benefit is [WTSL08][6] that approaches the problem by dividing the input in regions and

Figure 2: Axis-aligned retargeting in action given an input saliency map[13].

allowing important regions to scale respecting the original aspect ratio, while unimportant regions can stretch, that is why it is referred to as "Scale and Stretch". No constraints in the energy system guarantee lines to be straight, but they say how it does benefit the system.

Seeing how axis-aligned was a good property of the system [PWS12][13] decided to approach the problem directly from that side by limiting the space of transformation **exclusively** to axis aligned. The algorithm given an image with dimensions W and H, respectively width and height, computes a deformed grid where cells are W/N and H/M pixels in width and height for a destination image with dimensions W' and H'. The deformations is represented as a one-dimensional array of unknown width and height ($s = s^{rows}, s^{cols}$) where $\sum s^{rows} = W'$ and $\sum s^{cols} = H'$. The objective function is described as:

$$\boldsymbol{s}^T Q \boldsymbol{s} + \boldsymbol{s}^T \boldsymbol{b} \tag{1}$$

While the constraints of the system are:

$$s_i^{rows} \geq L^h, 1 \leq i \leq M \tag{2}$$

$$s_j^{cols} \geq L^w, 1 \leq j \leq N \tag{3}$$

$$s_1^{rows} + ... + s_M^{rows} = H' \tag{4}$$

$$s_1^{cols} + ... + s_N^{cols} = W' \tag{5}$$

2 and 3 by giving a lower boundary to the deformations are also guaranteeing that no fold-overs will happen as vertexes cannot have negative deformations, thus grids cannot overlap one another. L^h and L^w define the minimum possible size in pixels of a cell, an example value could be 0.15. 4 and 5 formally fix the resolution of the destination image, thus making sure that the deformations does not overflow the boundaries. The objective matrices are guided by the energy system, in detail: $Q \in \mathbb{R}^{(M+N)x(M+N)}$ and $\boldsymbol{b} \in \mathbb{R}^{M+N}$. When working with optimization problems there are different possible relationships between the constraint region and the objective function:

- Infeasible is when the constraint set **cannot** be satisfied, thus no solution exists.
- Feasible is when we have at least one set of values that satisfies all of the constraints.
- Unbounded is when the constraint region is unbounded, thus for each solution it always exists another set of values that outperforms the previous.

In order to mantain the QP feasible there is only the need for $L^h \leq H'/M$ and $L^w \leq W'/N$. With these relationships in mind we can be sure that the objective function is finite in the feasible region. As we can see the problem just formulated is indeed a quadratic programming one as all the constraints are linear and the objective function is made of a symmetric matrix Q and a linear cost function $\boldsymbol{b}$. While quadratic programming a special case of non-linear programming it is still very hard to solve especially in real-time scenarios, that is why we need to make sure that our objective function is convex. If that is guaranteed it can be solved as a convex optimization problem that greatly simplifies but more importantly makes more efficient the optimization. A function $f : S \mapsto \mathbb{R}$ is defined as such if:

$$\forall (x, y) \in S, \forall t \in [0, 1] f(tx + (1 - t)y) \leq tf(x) + (1 - t)f(y)$$

This means that a local minimum of the function is necessarily the global minimum, and the problem boils down to finding the local minimum. In the context of QP programming we can make sure that the objective function is convex if Q is semi-definite, this is guaranteed by the different energy systems. Once all of this is set up very efficient convex optimizers can be used to solve the problem in real-time.

2.3 Energy systems

The paper[13] proposed two energy systems that are not extremely different and whose results can also be linearly interpolated. Given $\Omega \in \mathbb{R}^{M \cdot N}$ as saliency map they are:

- *ASAP* (As-Similar-As-Possible) as seen in [8]. The version proposed is a simplification of the original one as the set of transformation is reduced.

The idea behind the original was the creation of *similarity constraints* that would reduce the deformations in important regions, which should be homogeneously scaled, by dispersing distortions in other less important parts of the image. The mesh contains control points who are grouped into *handles*, where each handle can be contained in different control points. Edges correspond to image features that have to be distorted as little as possible. The energy for the patch $P->P'$ is then calculated as[8]:

$$\epsilon(P, P') = min_{s \in S} \sum_{i=1}^{m} |s(p_i) - p'_i|^2$$

Where S is the set of similarity transformations [8]. Removing the need for rotation the similarity transformations are just scale and translation, the objective is thus to minimize non-uniform scaling [13], making the energy equation[13]:

$$grid_x = M/H$$

$$grid_y = N/W$$

$$\epsilon_{ASAP} = \sum_{i=1}^{M} \sum_{j=1}^{N} (\Omega_{i,j}(grid_x s_i^{rows} - grid_y s_j^{cols}))^2 \tag{6}$$

The $K \in \mathbb{R}^{(MN)\cdot(M+N)}$ matrix from which we will construct the QP system is thus[13]:

$$r(k) = \lceil k/N \rceil \tag{7}$$

$$c(k) = ((k-1) \bmod N) + 1 \tag{8}$$

$$K_{k,l} = \begin{cases} \Omega_{r(k),c(k)} \cdot grid_x, & \text{if } l = r(k) \\ -\Omega_{r(k),c(k)} \cdot grid_y, & \text{if } l = M + c(k) \\ 0, & \text{otherwise} \end{cases}$$

In order to plug them in in the QP system we have the positive semi-definite $Q = K^T K$ and $\boldsymbol{b} = 0$.

- *ARAP* (As-Rigid-As-Possible) While the previous energy system penalized non-uniform scaling ARAP penalizes all non-rigid transformations. A transformation is defined as non-rigid whenever the distance between two arbitrary points is changed by the operation itself. In a strictly axis-aligned context rigid transformations exclude scaling too, leaving us with just translations. Keeping in mind the definitions for $grid_x$ and $grid_y$ in the previous section we have[13]:

$$\epsilon_{ARAP} = \sum_{i=1}^{M} \sum_{j=1}^{N} \Omega_{i,j}^2((grid_x s_i^{rows} - 1)^2 + (grid_y s_j^{cols} - 1)^2) \tag{9}$$

Using the same definitions for $r(k)$ (7) and $c(k)$ (8) two matrices are defined $R^{top}, R^{bottom} \in \mathbb{R}^{(MN)\cdot(M+N)}$ [13]:

$$R^{top}_{k,l} = \begin{cases} \Omega_{r(k),c(k)} \cdot grid_x, & \text{if } l = r(k) \\ 0, & \text{otherwise} \end{cases}$$

$$R^{bottom}_{k,l} = \begin{cases} \Omega_{r(k),c(k)} \cdot grid_y, & \text{if } l = M + c(k) \\ 0, & \text{otherwise} \end{cases}$$

That are used to to build the QP system as[13]:

$$Q = \begin{bmatrix} R^{top} \\ R^{bottom} \end{bmatrix}^T, \boldsymbol{b} = -2 \begin{bmatrix} R^{top} \\ R^{bottom} \end{bmatrix} \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{v} \end{bmatrix}$$

where $\boldsymbol{v}$ is the vector of saliency defined as: $\boldsymbol{v} \in \mathbb{R}^{MN}, v_k = \Omega_{r(k),c(k)}$.

2.4 Why and how we used it

The limitations imposed by the axis-alignment of the transformations should not be seen as negative, they actually make the algorithm way more robust for real-time consumer applications. The worst case scenario is softened, as no damaging warps or fold-overs can happen. As the technique is very flexible different energy systems could be used for different images and destination resolutions. For our purposes and from observations ASAP seemed the best in a general purpose scenario. This does not prevent us from embedding more and different energy systems in the future. As said above, the only requirement for the energy system is to output a Q matrix that is **semi-definite**, thus rendering the QP problem convex. Even though [13] proves us that it is feasible to retarget images using their method in real-time we decided to use it offline solving the system at different destination aspect ratios. This provided us with a set of *spacings* for all the grid cells. Later the real-time retargeting will not need to solve the system again for a specific aspect ratio, as we probably have already solved it for an upper and lower bound, this leaves the retargeting application the only task of interpolating accordingly to the target aspect ratio.

3 Implementation

3.1 Approaches

As hinted in the previous section one of the advantages of retargeting using fixed grid axis-aligned deformations is that the solutions to the system (for different resolutions) can be computed offline in a fairly efficient manner and results can be subsequently interpolated. While the offline computation and the choices made will be explained in detail in subsection 3.2, this chapter will explain the two approaches that have been evaluated prior to starting the implementation. Assuming the user has enriched the images with the appropriated metadata and eventually flagged them as retargetable the actual retarget function could either take place in one of the following contexts:

- Web Server (Backend) : Extremely wasteful time-wise and bandwidth-wise to recompute everything and then send the new image on every resize operation. This has not really be taken into consideration at all.
- Browser's rendering engine (Frontend) : This would mean that the user except for providing the enriched images does not have any other control over the retargeting as it happens inside the rendering engine, be it Gecko (Mozilla Firefox) or any kind of WebKit flavour (Chromium, Safari and most of the others browsers). This is likely to be the most efficient solution as:
  - There are no language constraints nor overheads. Even though V8 and SpiderMonkey are extremely optimized (recent LLVM integrations [16]) it is not comparable to native code and we are anyway subjugated to language limitations.
  - Metadata loading, image retargeting and all other operations can happen **before** the initial image is displayed and thus the user would not even notice that additional computations took place.

  While the above looks indeed interesting it is not exempt from its share of issues, especially regarding the implementation in multiple rendering engines, the non-existent backward compatibility and the opaqueness of the whole process.
- JavaScript library (FrontEnd) : The other frontend approach is the implementation of the metadata extraction and retargeting inside a client-side script. Compared to the in-browser implementation it suffers from:
  - Inevitable performance reduction, that is partially hidden by the fact that everything happens asynchronously.
  - Delayed loading, the fact that everything happens while the user is browsing the page, thus without introducing any loading, still implies that images are retargeted incrementally with possibly noticeable popping.

– Same-origin policy restrictions. The above two issues are not real constraints, but simply degrade the user's experience. Using JavaScript you have **no** access to the raw image data starting from the headers, we then need an additional `XMLHttpRequest` that falls under the SOP scope. SOP states that a script loaded from a domain A can **not** request resources from a different domain or the very same domain with a different port or protocol [17]. Although this looks like a daunting problem it can be worked around using *HTTP Access Control* (CORS) with a small help from the resource's host.

On the other hand using a JavaScript library has benefits that in certain scenarios out weight the problems. WebGL is nowadays a de facto standard and this enables us to fully exploit hardware-accelerated rendering for image distortion on a *canvas* without the need for low-performance HTML hacks. More importantly the purpose behind the whole project was to give something practical to web developers today. Requiring them to download a custom browser build (until the eventual pull request would be merged) or restricting the possible audience only to the last build goes directly against what we want to achieve.

It is pretty obvious from the above description what decision was made for the frontend. Using a JavaScript library with the expense of some small degradation in user's experience allows us to reach a much wider audience that can get started immediately with this technology. The following subsections will describe in detail how each of the components function, while 4 will describe an example pipeline starting from a non-enriched image up to see its retargeting in your web page.

3.2 Backend: retarget-lib

Retarget-lib is the backend for all the offline processing utilities that might be created. This includes, but not limited, to Photoshop plugins, command line utilities and custom saliency painters. It exposes an extremely simple API

```
CalculateSpacingsNxN(int src_resolution[2], int dest_resolution[2]
                        unsigned char input_saliency[], double
                            output_spacings[]);

EncodeMetadata("src_image", metadata, "output_image");
```

- *CalculateSpacings* This routine is responsible for the first part of the processing. The user would open their favourite image editing program and paint a greyscale version of the original image defining the saliency map. As suggested in previous chapters this step could be automated, yet with very minimal effort from the designer good results can be achieved with non pixel perfect saliency maps. For higher resolution grids better results might be obtained using more fine grained tools. Once the saliency map has been created it can be exported and ready for use. After the enriching

process is done the original saliency texture can be discarded as there is no more use for it.

Once loaded in memory *CalculateSpacings* will generate a low resolution saturated version of the saliency map, where per cell values will be directly used for the creation of the energy system's matrices as seen in 2. Once the matrices for the energy systems have been created, they are packed and ready to be fed in the quadratic convex optimizer. The additional parameters used in the QP system such as b are computed based on the energy system in use. As it currently stands CalculateSpacingsNxN is limited to a 25x25 grid and $ASAP$ energy system. This can be easily, and will in future versions, extended to support different parameters.

$ASAP$ has been chosen because it seemed the system that yielded better results overall, this does not mean that other systems such as $ARAP$ or hybrid versions perform worse, but we felt like they can have different usage scenarios. One very good advantage of this offline process is that every image can be retargeted using different parameters that best suit the needs of the contents, as what we are interested in exporting are the results of the computation. *CalculateSpacings* solves a single system for a single destination resolution. Depending on the needs of the web page and the image a different number of aspect ratios should be exported.

- *EncodeMetadata* Once a sufficient number of spacings has been calculated it is time to embed them as XMP metadata. XMP is a standardized file labeling technology created by Adobe that allows the user to embed metadata in most of existent image formats[18]. It has been chosen over custom approaches or $EXIF$ because:

  - It is extensively used in all Adobe products and widely used outside the Adobe ecosystem to communicate non-pixel data between different applications.
  - Integrates very well with the Adobe suite of products as different SDKs are provided for different application. Thus writing a Photoshop plugin would boil down to writing glue code for *retarget-lib*.
  - Is not limited to a single format, but supports many different including, but not limited to PNG, JPEG and RIFF.
  - Metadata is encoded in plain text XML, thus decoding it from the Javascript script does not require any external library. Although not endorsed, it is considered valid to manually look for specific tags in a text view of the binary file. Once all the different pieces (as they are scattered throughout the file) have been found and merged a $DOMParser$ can be used to interpret its content.

  Plain-text XML does have some space overheads, but our metadata requirements are very low. The exported *Metadata* is divided into *MetadataEntries* each one contains $W + H$ floats where W and H are respectively the number of cells in a row and number of cells in a coloumn.

Floating point numbers are not required and can be easily normalized as we know the minimum and maximum value a cell can have for each *MetadataEntry*. From 4 or 8 bytes we can reduce down to 2 bytes plus the storage for the normalization factor for each entry. Note how a cell has a lower hard-coded boundary and an upper boundary that is the destination width or height minus the lower bound times all the remaining cells, in formulas:

$$cell \in \mathbb{R}^2$$
$$cell \in [(a_1, a_2), (b_1, b_2)]$$
$$a_1 = b_1 = 0.15$$
$$a_2 = dest_w - (cells_x - 1) \cdot a_1$$
$$b_2 = dest_h - (cells_y - 1) \cdot b_1$$

Additional compression could be done, but as the number of aspect ratios exported is usually not high, good results can be achieved with as few as 10 different aspect ratios.

3.3 Backend: retarget-make

retarget-make is a simple tool created in order to show how easy it is to create custom utilities to generate and encode metadata using *retarget-lib* and an input saliency map. The process is very straightfoward and follows the guidelines described in the previous subsection. Loads a saliency image, scans for it is first channel and generates a byte array containing the saliency that is then fed into *retarget-lib* at different resolutions to generate some metadata.Additional parameters will include the output resolutions to be exported and flags reflecting what the internal *retarget-lib* exposes.

3.4 Frontend: retarget-js

The javascript library is the core of the project as given a web page retargets all the images that have metadata in them. While some of the steps are indeed trivial other presents interesting quirks worth going over. Overall the process follows this flow:

- Finding images and metadata extraction The retargeting script registers itself for `DOMContentLoaded` event to start finding valid images as soon as possible. The advantage over the standard *load* event is that we does not wait for resources such as stylesheets, images or frames to load [19]. All the `img` tags in the loaded DOM are then scanned. By default all the images are scanned for retargeting metadata. Scanning can be explicitly stopped by setting `data-retarget="false"` as the `img` attribute. The attribute has that specific name because the HTML5 standard requires custom XML attributes to prepend `data-` to identify them as such.

Using simply *retarget* will probably work across most of the browsers without any problem as most of the errors and warnings are suppressed by browsers, but is not really future-proof. Now the image's original raw data, not just pixels, is scanned for XMP packets, as the packets are nothing else than standard XML tags, the different initial and end tags are found. Then the sub-strings are extracted from the image and fed into the `DOMParser` for easier processing as XMP makes **heavy** use of XML namespaces.

In 3.1 we have mentioned how SOP is a major limiting factor. This is where we can feel it in action. In order to extract metadata we can not just use the raw pixels of the image, we need the whole data, starting from the header. Unfortunately JavaScript and the underlying browser APIs give us no access at all to those images. This means that we manually have to download them again. This is the major bottleneck of the client-side JavaScript implementation, as in order to *GET* request them we need to make an `XMLHttpRequest` that is subject to the same-origin policy. The obvious additional drawback is the double download of the image. For today's standards is not really a problem, it might only cause issues on mobile networks with limited data plans. Unfortunately we have found no workaround for that, the best case scenario would be to propose a *window* API that would allow us to retrieve image's metadata.

The metadata extractor follows the format specified in the first part of the XMP standard. Metadata is contained inside a so called *packet*, that can be looked for based on specific start and end tag, respectively `<? xpacket begin=` and `<?xpacket end=`. Even though most of the metadata is always contained in this first region, in some scenarios extra data is contained in other regions that are **not** enclosed in *packets* (Section 2.1.3.1 of the standard). Their are fundamentally stripped versions enclosed in `<x :xmpmeta>` tags, those are scanned aswell for additional data. The XML is pretty standard and there is nothing special to it, the XMP documentation can be consulted for more information.

The only additional issue encountered was the incompatibility between the *DOMParser* and the utf-8 encoded XML, as the latter has some characters that need to be removed before being serialized or deserialized. This is avoided by doing a sanitation pass on the input XML. The *retarget-js* library is completely independent from the grid's size, the current 25x25 limitation is due to the encoding process.

- Initialization and replacement The download and metadata extraction does not stop the user from viewing the web page as the page's elements have not been touched yet. Now it is time to initialize a WebGL context. As every image will be replaced with a *canvas* there will be one context for each retargeted image. Initialization of resources, shader compilation, uploading of texture data takes some time and it is done in a *canvas* not appended to the document, thus not displayed. In theory some time could

be saved by sharing shaders and common states between the contexts, but the complexity overhead of multiple shared contexts does not justify the potentially negligible memory footprint. A WebGL 1.0 context is limited to a OpenGL ES 2.0 feature level, basically a programmable pipeline without geometry, compute or tesselation shaders, but it is perfectly fine for our needs.

Having full real-time retargeting in the browser using GPGPU without any offline postprocessing could be interesting, yet the only real benefit would be a slight improvement in image quality. After all the GPU resources have been created and uploaded the original image is hidden by changing its visibility property and the *canvas* is inserted after the source image's element. There are many small tricks that can be used to optimize this, but in the end, as we will see in 3.4, they are not generally worth it and possibly even degrading.

- Updating and Rendering For real-time applications rendering in *WebGL* the `setInterval(ms_interval, callback)` API is used to request the `render()` function to be called a certain amount of times every second. For our purposes this would be an extreme waste of resources as retargeting has to only be done **once** every time the image is resized. We do this by registering the retarget function to the `window.onResize()` event on which we appropriately resize the backbuffer, the viewport and proceed rendering the image. This means that scrolling the web page will not trigger the resize event, thus having no overhead. The rendering and image distortion is pretty straightforward. It works by generating a 25x25 grid with respective vertex and index buffer. The input layout for each vertex is comprised of a screen space position and a non-distorted texture coordinate calculated as follows:

$$uv_{xy} = tuple(pos_x/(gridcells_x - 1), pos_y/(gridcells_y - 1))$$

Texture coordinates are constant and always remain the same, what is modified is the position of each vertex that then is sampled using the original *uv* coordinate, thus creating the distortion. The last step is going from the offline calculated spacings for a different number of fixed resolutions to the NDC for the specific resolution. This is done in two steps. First we find the two wrapping aspect ratio for the destination one. If this falls out of range then lower and the upper bound point to the same aspect ratio that is the closest to the set currently in use. Once they have been found interpolation occurs. Since we are dealing with scales, it is generally recommended to use logarithmic interpolation. One simple example of the different behavior is explained trying to find the answer for:

$$interpolation(a = 0.5, b = 2, f = 0.5)$$

Linear interpolation would give us:

$$lerp(a, b, f) = a + (b - a) \cdot f$$

$$lerp(0.5, 2, 0.5) = 0.5 + (2 - 0.5) \cdot 0.5 = 1.25$$

And this makes sense if we were to plot a point in between a and b, but from a scaling perspective is this what we really want ?. Rephrasing the question to our needs it asks for the mean value between an image that is **half** the size of the original one and one that is **double** the size of the original one. Obviously the common answer to this question is "the original size", this is where logarithmic interpolation comes in our help:

$$loginterp(x, y, f) = x^f \cdot y^{1-f}$$

$$loginterp(0.5, 2, 0.5) = 0.5^{1/2} \cdot 2^{1/2} = \sqrt{0.5} \cdot \sqrt{2} = 1$$

The f factor is calculated from the starting and ending aspect ratio (ar stands for aspect ratio):

$$f = (ar_{start} - ar_{min})/(ar_{max} - ar_{min})$$

Once every single new spacing has been found we need to transform them to Normalized Device Coordinates (shifted by 1) by:

$$\forall spacing : spacing = spacings \cdot 2/\sum_{n=1}^{n=gridcells} spacings[n]$$

The 2 is because NDCs range from -1 to 1 and when setting the vertex coordinates we incrementally add them starting from -1. After all the vertex positions have been updated we upload them to GPU and call `gl.drawElements`, the index buffer obviously is subject to no modifications.

4 Tools and Workflow

In this section a sample workflows that goes from a standard wepage without any retargeting to a retargerted web page will be presented. The image has been taken from [20] and chosen explicitly to show how this retargeting technique works well even if there are multiple subjects in the scene. The following code listing creates a simple page with an image and text on the right (as the image floats left). The image will obviously stretch and will be unpleasant to view after a certain window's width.

```
<html>
<head>
<style>
<!-- Inline CSS -->
.test {
    width: 50%;
    height: 200px;
    float: left;
    border: 20px solid white;
}
body{
font-size: 20px;
}
h1{
font-size: 40px;
}
</style>
</head>
<body>
<h1>Example Page</h1>
<!-- Image -->
<img class="test" src="color.jpeg" />
<!-- Text -->
Lorem ipsum dolor sit amet, <!-- Insert sample quote -->... consequat.
</body>
</html>
```

The original resolution of the picture in 1000x668 (3) and the screen-shot has been taken while being viewed fullscreen on a 1920x1080 display.

The first step in enabling the image to be retargeted is to define a saliency map for it. If the user prefers he can use any kind of automatic saliency generator, here we will define a saliency map created manually. We will use GIMP for the job as it is completely free and open source, but the same can be achieved with any other image editing tools. First the image is loaded in the workspace and a layer for it will be created. Then a second layer is created on top of the first one (possibly transparent), this allows the user to paint the saliency map directly on top of the image, thus being able to see exactly what regions of the image we are painting. Now, using any kind of brush, does not have to be hard, will work with soft ones too, we can just paint over the *objects* we desire to preserve in the newly created layer. In our example we painted over the horses as they are the focus of the picture:

Once this is done, the upper layer should be exported. What kind of back-

Figure 3: Original picture with preserved aspect ratio.

Figure 4: Test web page without any retargeting applied.

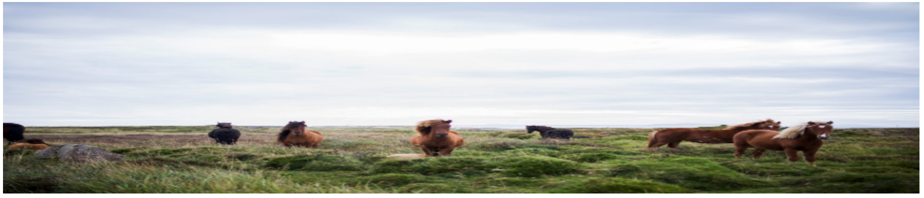

Figure 5: Close-up of the stretched picture.

Figure 6: Screenshot of GIMP while editing the saliency map.

ground or transparency has been chosen does not really matter. What is important in the end is the value in the R channel (as multiple channels are not needed), as a matter of fact having chosen a grey scale image *or* setting the brush color to RGB(255, 0, 0) would have produced the same result. The output format of the upper layer does not matter either, as long as it is one of the common ones including PNG, JPG, BMP, etc.. Once exported in its own file the saliency map and the original color map have to be merged together. In a future scenario this would be done directly from the image program using a custom plug-in. As it stands now, making sure the executable and the image are in the same directory, the following command can be run:

```
retarget-make -s"saliency.png" -i"color.png" -o"retarget_color.png"
```

This assumes that the saliency map is saved as *saliency.png* and the color map was called *color.png*, names can be changed if it is not the case. If no *-o* is specified the input color file will be overwritten.

The great advantage of having implemented everything as a frontend JavaScript library comes into play exactly here. In order to enable retargeting in your page all there is to do is include the script at the end of the body in the HTML page. Hot-linking the script will be added using public hosting services.

```
<script src="../retarget-js/retarget.js"></script>
```

The metadata **does not** corrupt the image in any way, it is still perfectly valid, usable in the same contexts and correctly readable by all the image viewers. Once by the web browser two things will happen:

- All images that **have** metadata in them will be retargeted by default. This behavior has been chosen in order to minimize the amount of work

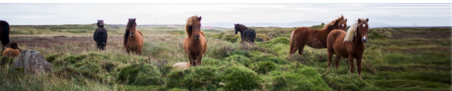

Figure 7: Web page with `retarget-js` linked.

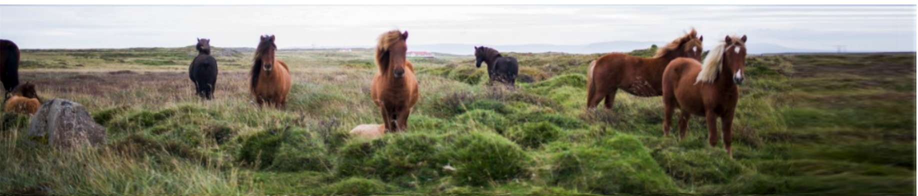

Figure 8: Close-up of the image.

the web designer has to do, adding a custom tag to all the images in a web page (even if auto-generated) is a too expensive time-wise.

- Behavior can be also manually controlled by the *data-retarget* attribute, whose values can be *true* or false. In the former case the image will be retargeted, as default behavior, in the latter the script will explicitly skip the image in question.

In order to get more information on the process useful information is logged by default in the browser's console. Setting `Retarget.logging_enable = false` will disable such feature. Once the script has been included the retargeted image should be correctly displayed as in Figure 7. In the following pages a collection of web pages with `retarget-js` embedded can be found.

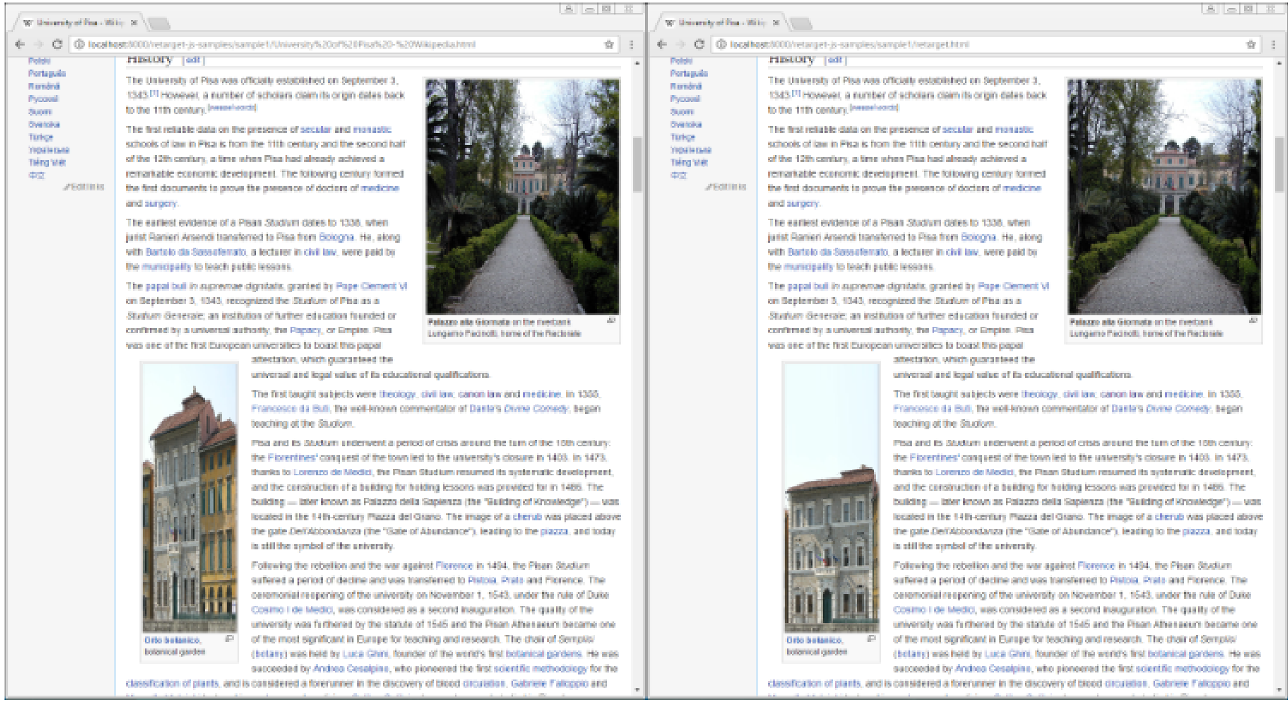

Figure 9: Wikipedia page with retargeting applied to it. Tweaked only to allow for stretching in order to show how retargeting works.

Figure 10: Comparison of the test web page at different resolutions. Left is default, Right is retargeted using our script.

5 Conclusions and further improvements

The purpose of this project was to show how retargeting techniques are ready to be used in the real world and give a basic toolkit for such purpose. In its current form there are many features that are missing, starting from [13] that also proposed:

- *Laplacian regularization* in order to smooth the deformation results across the image.
- *Cubic B-spline interpolation* to help the process of up-scaling the low resolution grid.
- *Cropping* could be enabled by simply setting L^w and L^h to 0, thus giving the user more access to the retarget parameters.
- *Grid resolution* could be changed depending on the image to either save memory and processing by reducing the resolution or improving quality at higher resolutions by increasing the grid's density.

Additionally, the saliency map is currently exclusively provided by the user, but giving an option to automatically generate one would be helpful if the image database was to be considerably big. The clear advantage of this system, as already mentioned, is that the offline processing is almost entirely decoupled from the rendering. Thus any kind of offline tool and algorithm could be used to generate the *spacings* without having to modify the frontend.

Tweaking the JavaScript library would be needed only if different retarget functions, non axis-aligned mesh based for instance, were to be used but, as simple as [13] is, it is robust and provides high quality results at the same time. The perfect scenario for retargeting is inside web browsers and we hope that our implementation will bring more attention to this already very powerful technology.

References

[1] V. Setlur, S. Takagi, R. Raskar, M. Gleicher, and B. Gooch, "Automatic image retargeting", in *Proceedings of the 4th International Conference on Mobile and Ubiquitous Multimedia*, ser. MUM '05, Christchurch, New Zealand: ACM, 2005, pp. 59–68, ISBN: 0-473-10658-2. DOI: 10.1145/1149488.1149499. [Online]. Available: http://doi.acm.org/10.1145/1149488.1149499.

[2] S. Avidan and A. Shamir, "Seam carving for content-aware image resizing", *ACM Trans. Graph.*, vol. 26, no. 3, Jul. 2007, ISSN: 0730-0301. DOI: 10.1145/1276377.1276390. [Online]. Available: http://doi.acm.org/10.1145/1276377.1276390.

[3] C. Hecker, B. Raabe, R. W. Enslow, J. DeWeese, J. Maynard, and K. van Prooijen, "Real-time motion retargeting to highly varied user-created morphologies", in *Proceedings of ACM SIGGRAPH '08*, http://chrishecker.com/Real-time_Motion_Retargeting_to_Highly_Varied_User-Created_Morphologies, 2008.

[4] M. Rastegari and N. Gheissari, "Multi-scale cartoon motion capture and retargeting without shape matching", in *Proceedings of the 2008 Digital Image Computing: Techniques and Applications*, ser. DICTA '08, Washington, DC, USA: IEEE Computer Society, 2008, pp. 320–326, ISBN: 978-0-7695-3456-5. DOI: 10.1109/DICTA.2008.51. [Online]. Available: http://dx.doi.org/10.1109/DICTA.2008.51.

[5] M. Rubinstein, A. Shamir, and S. Avidan, "Improved seam carving for video retargeting", *ACM Trans. Graph.*, vol. 27, no. 3, 16:1–16:9, Aug. 2008, ISSN: 0730-0301. DOI: 10.1145/1360612.1360615. [Online]. Available: http://doi.acm.org/10.1145/1360612.1360615.

[6] Y.-S. Wang, C.-L. Tai, O. Sorkine, and T.-Y. Lee, "Optimized scale-and-stretch for image resizing", *ACM Trans. Graph.*, vol. 27, no. 5, 118:1–118:8, Dec. 2008, ISSN: 0730-0301. DOI: 10.1145/1409060.1409071. [Online]. Available: http://doi.acm.org/10.1145/1409060.1409071.

[7] P. Krähenbühl, M. Lang, A. Hornung, and M. Gross, "A system for retargeting of streaming video", *ACM Trans. Graph.*, vol. 28, no. 5, 126:1–126:10, Dec. 2009, ISSN: 0730-0301. DOI: 10.1145/1618452.1618472. [Online]. Available: http://doi.acm.org/10.1145/1618452.1618472.

[8] G.-X. Zhang, M.-M. Cheng, S.-M. Hu, and R. R. Martin, "A shape-preserving approach to image resizing", *Computer Graphics Forum*, 2009, ISSN: 1467-8659. DOI: 10.1111/j.1467-8659.2009.01568.x.

[9] D. Freedman, R. Chen, Z. Karni, C. Gotsman, and L. Liu, "Content-aware image resizing by quadratic programming", *The 3rd Workshop on Non-Rigid Shape Analysis and Deformation Image Alighment*, 2010.

[10] F. Banterle, A. Artusi, T. Aydin, P. Didyk, E. Eisemann, D. Gutierrez, R. Mantiuk, and K. Myszkowski, "Multidimensional image retargeting", in *ACM Siggraph ASIA 2011 Courses*, ser. ACM Siggraph ASIA, ACM, Dec. 2011. [Online]. Available: http://vcg.isti.cnr.it/Publications/2011/BAADEGMM11.

[11] M.-M. Cheng, G.-X. Zhang, N. J. Mitra, X. Huang, and S.-M. Hu, "Global contrast based salient region detection", in *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR '11, Washington, DC, USA: IEEE Computer Society, 2011, pp. 409–416, ISBN: 978-1-4577-0394-2. DOI: 10.1109/CVPR.2011.5995344. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2011.5995344.

[12] S. Goferman, L. Zelnik-Manor, and A. Tal, "Context-aware saliency detection", *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 10, pp. 1915–1926, Oct. 2012, ISSN: 0162-8828. DOI: 10.1109/TPAMI.2011.272. [Online]. Available: http://dx.doi.org/10.1109/TPAMI.2011.272.

[13] D. Panozzo, O. Weber, and O. Sorkine, "Robust image retargeting via axis-aligned deformation", *Comput. Graph. Forum*, vol. 31, no. 2pt1, pp. 229–236, May 2012, ISSN: 0167-7055. DOI: 10.1111/j.1467-8659.2012.03001.x. [Online]. Available: http://dx.doi.org/10.1111/j.1467-8659.2012.03001.x.

[14] C.-K. Huang, Y.-L. Chen, I.-C. Shen, and B.-Y. Chen, "Retargeting 3d objects and scenes with a general framework", *Computer Graphics Forum*, 2016, ISSN: 1467-8659. DOI: 10.1111/cgf.13001.

[15] [Online]. Available: https://github.com/mozilla/gecko-dev/blob/master/image/Downscaler.cpp.

[16] [Online]. Available: https://trac.webkit.org/wiki/FTLJIT.

[17] [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy.

[18] [Online]. Available: http://www.adobe.com/products/xmp.html.

[19] [Online]. Available: https://developer.mozilla.org/en/docs/Web/Events/DOMContentLoaded.

[20] [Online]. Available: http://publicdomainarchive.com.