Photosketcher: interactive sketch-based image synthesis

Mathias Eitz[1], Ronald Richter[1], Kristian Hildebrand[1], Tamy Boubekeur[2] and Marc Alexa[1]
[1]TU Berlin [2] Telecom ParisTech/CNRS

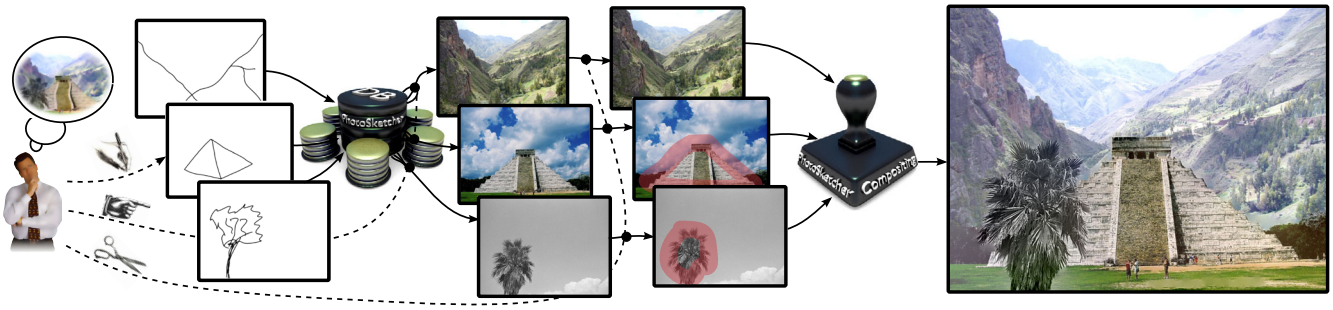

Figure 1: *Photosketcher: realistic images from user sketches using large image collection queries and interactive image compositing.*

Abstract

We introduce Photosketcher, an interactive system for progressively synthesizing novel images using only sparse user sketches as the input. Compared to existing approaches for synthesising images from parts of other images, Photosketcher works on the image content exclusively, no keywords or other metadata associated with the images is required. Users sketch the rough shape of a desired image part and we automatically search a large collection of images for images containing that part. The search is based on a bag-of-features approach using local descriptors for translation invariant part retrieval. The compositing step again is based on user scribbles: from the scribbles we predict the desired part using Gaussian Mixture Models and compute an optimal seam using Graphcut. Optionally, Photosketcher lets users blend the composite image in the gradient domain to further reduce visible seams. We demonstrate that the resulting system allows interactive generation of complex images.

1 Introduction

Suppose you are thinking of an image depicting a real scene. How would you create a digital representation of this image? If the scene exists in reality it could be photographed, however, this is expensive and time-consuming. With the advent of large open image collections it is possible that a similar photograph has already been taken and made available. Yet, it is to be expected that any existing photograph is only an approximation of your imagination. It seems that for a more complete rendition of the mental image it is necessary to create it with the help of a computer, either by directly drawing it, or by creating a 3D scene that can be rendered into an image. Both solutions require talent, skill, and experience. We propose a system that combines aspects of all of the solutions above: it is based on photographs but lets the user interactively search and compose them, allowing anyone to create complex pictures without any particular artistic skill.

Our approach, named Photosketcher, combines two main ideas:

1. Instead of searching for a *single* image that perfectly fits the requirements, the image is interactively composed out of *parts* of images. The idea is that it is more likely that *each part* of the desired scene is *part of a picture* in the collection than finding a single, perfectly matching image. This has been shown to work well for filling holes in given images [Hays and Efros 2007] or stitching together a single image from a series of photographs with similar content [Agarwala et al. 2004].

2. All user interaction with Photosketcher is based on simple sketches and scribbles: the image parts are queried based on a sketching interface, requiring the user to only enter few rough feature lines of the desired structures. The compositing interface relies on sparse user scribbles that provide initial hints about the part to be selected.

Overall, synthesizing novel imagery with our system only requires two simple interaction steps: 1) sketching outlines that depict the desired scene element; as a result Photosketcher presents a small set of images that match the input and 2) selecting the best matching result; Photosketcher adds this image to the current intermediate result using foreground extraction and matting techniques. Users repeat those two steps until the result eventually fits their intentions and contains all desired scene elements (see Fig. 1). We give a more detailed explanation of this pipeline in Sec. 3.

Note that the system described in this paper is an extended version of the synthesis system proposed by Eitz *et al.* [2009]. While we follow the overall original idea of synthesing images from sparse sketches only, this papers differs in the following aspects: compared to the original single page version, we now have sufficient space to describe all pipeline step in full detail. We use Gaussian Mixture Models (GMM) for learning an image model, use bag-of-features sketch-based search rather than global descriptors and present and additional user study evaluating the retrieval quality.

Challenge: sketch-based search The first major technical challenge in the Photosketcher pipeline is the search for images resembling the user sketch. While searching image collections – given example photographs – is a common problem, we consider queries using sketched feature lines as input as an even more difficult problem. Since the information contained in a sketch is sparser than in a photograph, a larger variety of images can be matched and the search will be less precise.

We now formulate several desired properties of a search engine using sketched feature lines as input: a) the search should – despite the sparse input – be reasonably precise and the matches perceptually similar to the sketch. b) It should be invariant to the absolute position of the sketch on the canvas; this is important as we expect

users to search for *parts* of images and the absolute position of the desired part in a collection image is irrelevant. c) It should be tolerant against local as well as global deformations since we expect users' sketches not to exactly depict the outlines and proportions of photographed objects. d) The search should be fast yielding interactive retrieval rates even on collections possibly containing millions of images. To achieve these goals, we propose to base the search on a bag-of-features approach using local features encoded as histograms of gradient orientation [Eitz et al. 2011]. This approach has been shown to outperform other current approaches and due to its use of local features it is translation invariant as well as tolerant against local and global deformations. Finally, using inverted indices for the search yields the desired fast retrieval rates even on large collections. We describe this approach in more detail in Sec. 4.

Challenge: composing images The second major challenge in Photosketcher is to form a realistic image given the parts retrieved by sketching. This is extremely difficult as humans are very sensitive to several artifacts that arise from a naive composite: if the parts differ in texture, color or illumination a visible seam around the composition destroys the illusion of a realistic image. We address this by trying to hide the seam in regions where it is difficult to notice using Graphcut optimization. Retrieved parts might also be perspectively distorted with respect to the currently assembled image, such that the impression of a realistic image would be ruined. We do not try to fix this algorithmically but rather rely on the sheer amount of data available in today's image collections: as the number of images in the collection is further increased, the probability that it contains the desired part under the correct perspective also increases. Rather than forcing users to accept an automatic composite (which may or may not fit their intention), our key idea is to *start* from an automatically generated composite and provide scribble-based interfaces that allow interactively improving the automatically generated seam, if so desired. We describe the user interface in more detail in Sec. 3.

The resulting system can be used by novice users to quickly generate compound images (see Sec. 6). The power of the system stems from exploiting the vast amount of existing images, which offsets obvious deficits in search and composition. We present more detailed conclusions in Sec. 7.

2 Related work

The idea that *all* user input is based on sketching only is what makes Photosketcher novel and different from similar approaches, such as Sketch2Photo [Chen et al. 2009] which uses keywords *and* sketches. To put our work into context, we discuss related work in chronological order and highlight major similarities as well as differences in each case.

Graphcut Textures [Kwatra et al. 2003] introduces the first approach for synthesizing novel texture images by copying irregularly shaped parts – optimized using Graphcut – from multiple source textures. All of the following systems (including Photosketcher) make use of a variant of this technique in order to avoid visible seams in the final composite image/texture. Agarwala *et al.* [2004] introduce a framework for "interactive digital photomontage" (IDP). Given a stack of photographs and image objectives that encode the properties that are desired in the final composite, IDP makes use of Graphcut optimization to find a composite that best fullfills all objectives. While the compositing stage in Photosketcher has been inspired by Graphcut Textures as well as IDP, the main difference lies in the type of images used to form the composite: in IDP as well as Graphcut Textures the source images are closely related and depict the same scene or object but with slightly different properties such as depth-of-field, face expression or illumination. In Photosketcher this is very different: parts typically are selected from a large number of unrelated images and this makes compositing them into a seamless result image harder.

Rother *et al.* [2006] define an automatic algorithm for assembling representative elements from a set of unrelated images into a convincing collage (AutoCollage). Generating the collage is formulated as a set of optimization problems, which include: automatically selecting the regions of interest from the input images, tightly packing the extracted regions into a collage such that important areas such as faces remain visible and finally making the seams between the collage elements visually undisturbing. AutoCollage is similar to Photosketcher with regard to the composition step in that unrelated image content is to be assembled into a final result image. However, it differs by its final goal: while in AutoCollage the result is a tightly packed *collage* of image elements our aim is to generate a realistic image that in the optimal case would be undistinguishable from a real photograph. Johnson *et al.* [2006] present Semantic Photo Synthesis, a system specifically designed for synthesizing photographs – rather than collages – from parts of existing images. Their main idea is to create an image that is semantically similar to the desired one by placing text labels on a canvas. Those labels indicate rough position and category of the desired part. Similarly to all other systems, the final composite image is generated using Graphcut optimization. Photo Clip Art [Lalonde et al. 2007] lets users insert new objects – such as cars or humans – into *existing* photographs. Differently from all other systems, the central idea of Photo Clip Art is to *avoid* modifying the inserted object. Instead, the idea is to let users only choose from objects that already have the required properties (such as lighting, perspective and resolution). In case such objects exists, the resulting composite images are extremly convincing. The main difference between Photo Clip Art and Photosketcher therefore lies in the type of image data required: while Photo Clip Art relies on a medium-sized collection of images that contains presegmented and annotated object instances, Photosketcher relies on the variety that comes with a much larger image collection (without any metadata).

Chen *et al.* [2009] propose Sketch2Photo which uses a *text-labelled* sketch indicating the layout of a desired picture as the input. This *annotated* sketch is used to initially perform a keyword search on large online image collections (such as Google Images). For each element, the resulting set of pictures is filtered to retain only those images that are easy to automatically composite and paste them at the location which is indicated by the text-labelled sketch. The main difference to Photosketcher is that Sketch2Photo does not perform pure visual search: each object drawn by the user must be annotated with a keyword and a classical text-based search is then performed. Second, the system is not interactive and may require hours to synthesize novel images, while Photosketcher provides interactive output, such that users can edit and modify the pictures in real time. In case a computer generated rendition of the desired scene is already given, Johnson *et al.* [2011] propose the CG2Real system which replaces a CG image with parts of images from a large image collection. The resulting composite image is structurally similar to the original CG image but appears more realistic to the human observer.

Overall, Semantic Photo Synthesis [Johnson et al. 2006], Photo Clip Art [Lalonde et al. 2007] as well as Sketch2Photo [Chen et al. 2009] are the systems that are most closely related to Photosketcher. While all systems share the same overall goal of synthesizing realistic novel photographs, to the best of our knowledge Photosketcher is the first one that *interactively* works on large, *untagged* image collections.

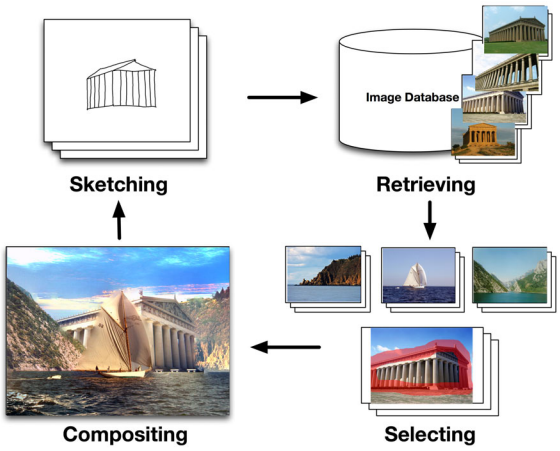

Figure 2: *Photosketcher workflow: the user starts by sketching a part of the desired content, the system retrieves best matches from a large collection of images and presents those to the user. The user selects the best match and uses interactive compositing to create an updated version of the desired image. This process is iterated until the user is satisfied with the composite image.*

3 Sketching

The aim of Photosketcher is to provide the user with an easy-to-use interface for interactively composing an image from parts of existing images. The fundamental principle is a loop, in which a user queries a large collection of images for matches to an outline sketch, chooses a query result, extracts a region from the chosen image, and pastes it into the current intermediate result (see Fig. 2).

An important interface question is how to query the database. While using text queries alone would be too imprecise, example-based querying is also not an option as the required examples are typically not at hand. Instead, we argue that sketching feature lines is both a natural and simple way to quickly communicate the mental concept of a shape. Additionally, using only information contained in the images themselves rather than using additional metadata (e.g. keywords) makes Photosketcher work on *arbitrary* image collections. Note that we only allow binary sketches, i.e. feature lines do not carry color information. We have initially experimented with colored strokes but users found this additional input dimension difficult to exploit.

In particular, users perform the following steps in one loop of the Photosketcher image generation process:

sketch and retrieve: users sketch binary outlines to define the desired *shape* of the content. We use this sketch to query a large image collection. The result of the query for shapes is a small set of pictures with similar structure from which users select the desired one.

transform and select: once a query result is chosen, users roughly draw a closed boundary around the desired part using a thick pen. This defines an initial selection which the user then translates and scales until the part is in the right position.

extract and composite: from the rough selection in the previous step we learn a model of desired and undesired image parts and automatically compute an initial composition that takes the learned model into account. Using a stroke based interface, users can iteratively refine the composition by drawing scribbles that indicate that some parts should be additionally included/removed from the final composite image.

Once the part is merged into one image, the user continues with sketching the next desired object or part. This loop repeats until the user is satisfied with the result. We illustrate this workflow in Fig. 2. We let users (optionally) select an initial background image using an initial sketch and retrieve step before the actual compositing loop begins.

We use a Cintix UX21 touch sensitive monitor as the input device such that users are able to sketch their queries naturally as if using pen and paper.

4 Searching

We propose using a bag-of-features (BoF) approach [Sivic and Zisserman 2003] for the sketch-based retrieval pipeline of Photosketcher. BoF search is a popular technique for example-based image retrieval and is – due to its use of *small local* features – inherently translation invariant. By quantizing the local features to a small finite set of visual words, the search can be accelerated using an inverted index which is a well-proven technique from text retrieval that is easily able to handle huge collections. In particular, we use the approach of Eitz *et al.* [2011] that generalizes BoF search for sketch-based image retrieval and therefore fulfills our requirements defined in Sec. 1. We now give a quick overview of the components of a typical BoF image retrieval system:

- Sampling in image (scale) space. This defines local pixel coordinates from which features are extracted. Often, an important algorithmic aspect is to make detection of those coordinates repeatable: in a transformed version of the original image (rigid or perspective transformation, different lighting etc.) the sampler should be able to primarily detect coresponding coordinates.
- Feature extraction and representation. Given a local coordinate, a feature is extracted that represents the image content within a small local window around that coordinate. The feature should be distinctive, i.e. perceptually different local regions should result in clearly discernable features.
- Quantization of features. In order to speed up retrieval as well as to achieve invariance to small local deformations, local features are quantized, yielding – depending on the application – a ”visual vocabulary“ containing between 1,000 und 250,000 visual words. Each image is then represented by a (sparse) histogram of visual word occurances.
- Search using inverted indices. An inverted index is typically constructed in an offline process and maps from a visual word to the list of images in which that word appears. This can significantly speed up the search process as only those images need to be considered that have at least one visual word in common with the query image.

In the following paragraphs, we quickly review the stages of the retrieval pipeline applied in Photosketcher (which is similar to the one in Eitz *et al.* [2011]) and highlight where the sketch-based search differs from a typical example-based retrieval system.

Edge extraction One important point is the pre-processing step required to extract the edge and contour information from the photographs in the image collection. This is an offline process and done only once for each image in the collection. Ideally, the resulting edge images would be perceptually very close to user sketches and retain only those edges that users are likely to sketch. This is a difficult problem and we resort to the standard approach of extracting

Canny edges from the images.

Sampling We perform random sampling in image space, using 500 random samples per sketch/edge image. Note that we do not apply interest point detection as the sketches typically are very sparse and only very few interest points could be detected. We also do not apply any scale-space detection in sketch-space, although this could be an interesting direction for future work. We discard all sample points that would result in an empty feature beeing extracted.

Feature representation We use the best performing feature representation from Eitz *et al.* [2011]: the SHoG descriptor essentially is a histogram of oriented gradients representation of the local image region. We use 4×4 spatial and 8 orientational bins. This makes the descriptor very similar to the SIFT descriptor except that no information about edge direction is stored (not available in binary sketches, only orientations from 0 to π can be discerned). Also, gradient magnitude is regarded as constant for edges/sketch lines.

Learning a visual vocabulary Given the local features from all collection images, we randomly sample 1 million local features and use them for learning a visual vocabulary by performing standard k-means clustering. This approach ensures that the visual vocabulary is learned from a wide variery of images and thus general enough to handle arbitrary user sketches. We use $k = 1000$ clusters.

Inverted index Given the visual vocabulary, we quantize each local feature to its nearest visual word (as measured using the standard Euclidean distance metric). As is standard for BoF approaches we build an inverted index that maps from visual word to the list of images containing that word.

Query given user sketch While running the Photosketcher system, the inverted index as well as the visual vocabulary are accessible to the system. Given a user sketch, we perform the following steps to retrieve similar images: a) sampling and extracting local features; b) quantization of all features against the visual vocabulary and c) lookup in the inverted index and ranking using tf-idf weights [Sivic and Zisserman 2003; Eitz et al. 2011]. The most expensive step is the quantization stage and we speed this up by performing the quantization for all features in parallel.

5 Compositing

Assuming a reasonably matching image $\boldsymbol{S}$ for a user sketch has been found, the last key step in Photosketcher is to progressively assemble the object(s) underlying the sketch with the existing background image $\boldsymbol{B}$. The difficulty in the compositing step lies in finding a tradeoff between two conflicting requirements:

1. Segmenting the semantically relevant content from the queried image. It would be visually disturbing to cut away parts of the object a user wants to paste into the final image.
2. Creating a composition that is perceptually plausible, i.e. that introduces no visible artifacts along the seams.

In Photosketcher, semantically important regions are specified using scribbles: initially, users roughly trace the border of the desired object with a thick pen. This partitions $\boldsymbol{S}$ into 3 regions: a "**d**esired" region Ω_D that must definitely come from $\boldsymbol{S}$, an "**u**ndesired" region Ω_U that must not come from $\boldsymbol{S}$ as well as a "**b**oundary" region Ω_B for which we are not sure yet whether the pixels in it rather belong to Ω_D or Ω_U (see Fig. 3). Our job now is to search for a seam in Ω_B that results in an optimal composition result. Once an initial composition is found (which is typically not yet perfect) users can sketch additional small scribbles to progressively correct errors in the initial result. We make two types of scribbles available for that task: one constrains the pixels lying under it to be included in Ω_D, the other constrains its underlying pixels to come from Ω_U. This procedure is similar to that used in several existing image segmention approaches but differs in that we also need to take the second requirement into account when computing the segmentation of $\boldsymbol{S}$.

Overall, our compositing algorithm consists of three steps: First, given the user scribbles, we learn a Gaussian Mixture Model (GMM) that we employ to predict if pixels in Ω_B should rather belong to Ω_D or Ω_U. Second, given the predicition from the GMM we use Graphcut to find a binary segmentation of $\boldsymbol{S}$ that results in an optimal composite image. And third, we optionally perform a blending step that smoothes out potentially remaining disturbing luminance differences along the seam between $\boldsymbol{S}$ and $\boldsymbol{B}$. We now describe those three components in detail.

Gaussian Mixture Model A GMM is a parametric representation of an arbitrary dimensional probability density function (pdf) as a linear combination of Gaussians. In Photosketcher, we learn two GMMs that represent a model of the desired and undesired parts of $\boldsymbol{S}$. Intuitively, the idea is that we have two models of $\boldsymbol{S}$: for each pixel $\boldsymbol{n} \in \boldsymbol{S}$ they give us the probability (actually the probability density) that $\boldsymbol{n}$ should be in the composite image. Remember that in Photosketcher the user scribbles define a trimap and partition $\boldsymbol{S}$ into three disjunct regions: one that should definitely be in the composite image, one that should definitely not be; and an unknown region (see Fig. 3). The main idea is to learn models of the color distribution in those regions given the trimap: we learn a model of the desired region by sampling pixels from Ω_D; similarly for the undesired regions by sampling from Ω_U. Finally, this allows us to make a prediction about the pixels in the unknow region Ω_U.

A probability density function modeled as a linear combination of k Gaussians is defined as:

$$P(\boldsymbol{n}|\boldsymbol{\Theta}) = \sum_{i=1}^{k} p_i N(\boldsymbol{n}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i), \tag{1}$$

with $\sum p_i = 1$ and $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ denoting the mean vector and covariance matrix of a multivariate Gaussian defined as:

$$N(\boldsymbol{n}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{\frac{M}{2}} |\boldsymbol{\Sigma}|^{\frac{1}{2}}} \exp\left[-\frac{1}{2}(\boldsymbol{n}-\boldsymbol{\mu})^{\mathsf{T}} \boldsymbol{\Sigma}^{-1} (\boldsymbol{n}-\boldsymbol{\mu})\right].$$

The unknown parameters of a GMM (which we need to estimate) are $\boldsymbol{\Theta} = [(p_1, \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1), \ldots, (p_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)]$. In Photosketcher we work on 3-dimensional pixel values in the L*a*b color space (i.e. $M = 3$) such that Euclidean distance between L*a*b triplets corresponds closely to their perceptual distance. We randomly sample $m = 4000$ samples from both Ω_D and Ω_U and use those samples to learn the parameters for two GMMs. We use $k = 5$ Gaussian components for each model and perform standard expectation-maximization [Bishop 2006] to estimate $\boldsymbol{\Theta}_{D,U}$. We denote by $P_D(\boldsymbol{n}|\boldsymbol{\Theta}_D)$ the learned model of the desired image part and by $P_U(\boldsymbol{n}|\boldsymbol{\Theta}_U)$ the model of the undesired part.

Finally, we are now able to predict for any unknow pixel $\boldsymbol{x} \in \boldsymbol{S}_{\Omega_B}$ whether it rather belongs to Ω_D or Ω_U. This is done by simply plugging $\boldsymbol{x}$ into Eqn. (1) given $\boldsymbol{\Theta}_D$ and $\boldsymbol{\Theta}_U$, respectively. We visualize the result of such a prediction in Fig. 3.

In the remaining part of this paper we write $P_D(\boldsymbol{n})$ as a shorthand for $P_D(\boldsymbol{n}|\boldsymbol{\Theta}_D)$.

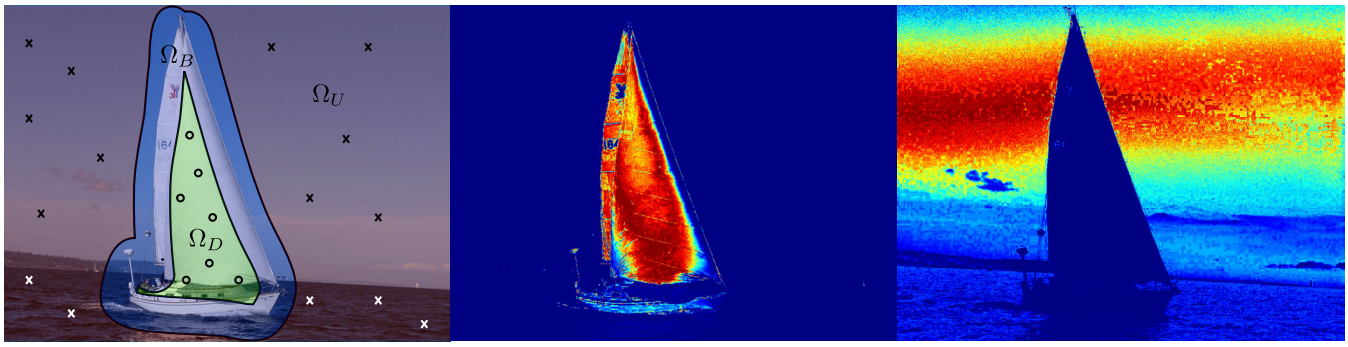

Figure 3: *The initial user-drawn boundary curve (blue) partitions S into three disjoint regions. We train two Gaussian Mixture Models using samples from Ω_U (crosses) and Ω_D (circles). This allows to predict for any pixel whether it rather belongs to Ω_D (middle) or Ω_U (right). Redder values encode higher probabilities.*

GraphCut We treat finding those regions in $\boldsymbol{S}$ that yield a semantically as well as visually plausible composition as a binary pixel labeling problem. Each pixel $\boldsymbol{n} \in \boldsymbol{S}$ is assigned a label $l_p \in \{0, 1\}$ that describes whether $\boldsymbol{n}$ should be taken from the source image $\boldsymbol{S}$ ($l_p = 0$) or the background image $\boldsymbol{B}$ ($l_p = 1$) in order to form an optimal composite image. Our task is to find a labeling that is a good compromise between both requirements. We find such a labeling by minimizing the cost function

$$E(\boldsymbol{S}, \boldsymbol{B}) = E_d(\boldsymbol{S}) + \lambda E_s(\boldsymbol{S}, \boldsymbol{B}). \quad (2)$$

$E_d(\boldsymbol{S})$ is a function of all pixels in image $\boldsymbol{S}$ and encodes our first requirement of including all semantically relevant object parts. Intuitively, $E_d(\boldsymbol{S})$ should return a small value if all important parts are included in the compositon and a high value if semantically important parts are missing. $E_s(\boldsymbol{S}, \boldsymbol{B})$ is a function of the pixels of both $\boldsymbol{S}$ and $\boldsymbol{B}$ and encodes the second requirement, i.e. that seams should run through regions where they produce the least amount of artifacts in the final composite image. Again, this should return high values for visible seams and low values otherwise.

More specifically, we define both terms as:

$$E_d(\boldsymbol{S}) = \sum_{\boldsymbol{n} \in \boldsymbol{S}} [(1 - l_p) \ln P_D(\boldsymbol{n}) + l_p \ln P_U(\boldsymbol{n})] \quad (3)$$

where $P_{D,U}(\boldsymbol{n})$ is the probability (density) of pixel $\boldsymbol{n}$ belonging to Ω_D and Ω_U, respectively (see Eqn. (1)). Since we later perform an additional blending step in the gradient domain, we use the cost function proposed for this problem by Hays and Efros [2007]. To simplify our notation we first define a single-channel intermediate image $\boldsymbol{I}$ that encodes the sum of squared differences between $\boldsymbol{S}$ and $\boldsymbol{B}$, i.e. the pixel-wise squared Euclidean distance between $\boldsymbol{S}$ and $\boldsymbol{B}$. Let $\{p, q\} \in N_p(\boldsymbol{I})$ denote the four pairs of pixels that form the 4-neighborhood of pixel p in $\boldsymbol{I}$. Then we define our energy measure as:

$$E_s(\boldsymbol{S}, \boldsymbol{B}) = \tilde{E}(\boldsymbol{I}) = \sum_{\{p,q\} \in N_p(\boldsymbol{I})} |l_p - l_q| \parallel p - q \parallel_2^2 . \quad (4)$$

Note that in Eqn. (3) and Eqn. (4) the terms $(1 - l_p)$, l_p and $|l_p - l_q|$ are used as indicator functions: they can take values of either 0 or 1. Their job is to select one of both terms in Eqn. (3) depending on whether pixel p gets labeled as 0 or 1. Similarly, the inner term of Eqn. (4) returns a value $\neq 0$ only if two neighboring pixels $\{p, q\}$ are separated by the seam, i.e get assigned different labels. To summarize: Eqn. (3) makes sure that the seam respects the learned model for the desired and undesired region while Eqn. (4) tries to force the seam through regions where the cut is visually least disturbing.

We efficiently find the labeling that minimizes Eqn. (2) using Graphcut [Boykov and Kolmogorov 2004]. More specifically, we define a graph $G = (V, E)$ where V is the set of $m \times n + 2$ vertices corresponding to the pixels in $\boldsymbol{S}$ plus two additional terminal nodes, the *source* S and the *sink* T. We define the edges in G to correspond to the 4-connectivity of the image, plus, for each node, two edges to the terminal nodes S and T. Associated with each edge is a weight computed from the term $\parallel p - q \parallel_2^2$ from Eqn. (4). Associated with each edge to S and T is the weight $\ln P_D(\boldsymbol{n})$ and $\ln P_U(\boldsymbol{n})$, respectively (see Eqn. (3)). To incorporate the user constraints coming from the scribbles, we replace the weights for edges to S by ∞ if n is contained in Ω_D and similarly for the edges to T if n is contained in Ω_U. To compute the minimum cut in G – which in turn gives us a labeling that minimizes Eqn. (2), we use the Graphcut implementation of Boykov et al. [2004]. To further reduce visible seams that might result from pasting the resulting region onto the canvas we perform an additional blending step.

Blending In Photosketcher, we provide the user with three options for blending the selected object into the existing image:

- Direct copy of the pixels labeled as foreground to the backgroud. This most simple option is extremely fast and performs astonishingly well in many cases, especially when the seam can be hidden in highly textured regions of the background image.
- Feathering, i.e. alpha blending of the pixels within a certain small band of k pixels along the seam.
- Poisson blending [Pérez et al. 2003], i.e. compositing in the gradient domain instead of directly in the spatial domain. Working in the gradient domain retains the relative luminances of the pasted pixels but adapts the absolute luminance to that of the background luminances along the seam. This option can be helpful if the foreground and background are similarly textured along the seam but are differenly illuminated.

Following our previous notation and the notation in Pérez *et al.* [2003] we denote by Ω the region in $\boldsymbol{S}$ that has been labeled with 0s in the Graphcut step, i.e. $\Omega = \Omega_F$. We denote by $\partial\Omega$ the boundary of Ω. $\boldsymbol{S}_\Omega$ then defines the set of actual pixel values of $\boldsymbol{S}$ lying in the region Ω. We now wish to find a set of unknown, optimized pixel values $\tilde{\boldsymbol{S}}_\Omega$ that have similar gradients (in a least-squares sense) as the pixels in $\boldsymbol{S}_\Omega$ but at the same time minimize color offsets along $\partial\Omega$, which would otherwise be visually disturbing. This problem is commonly formulated as a quadratic optimiza-

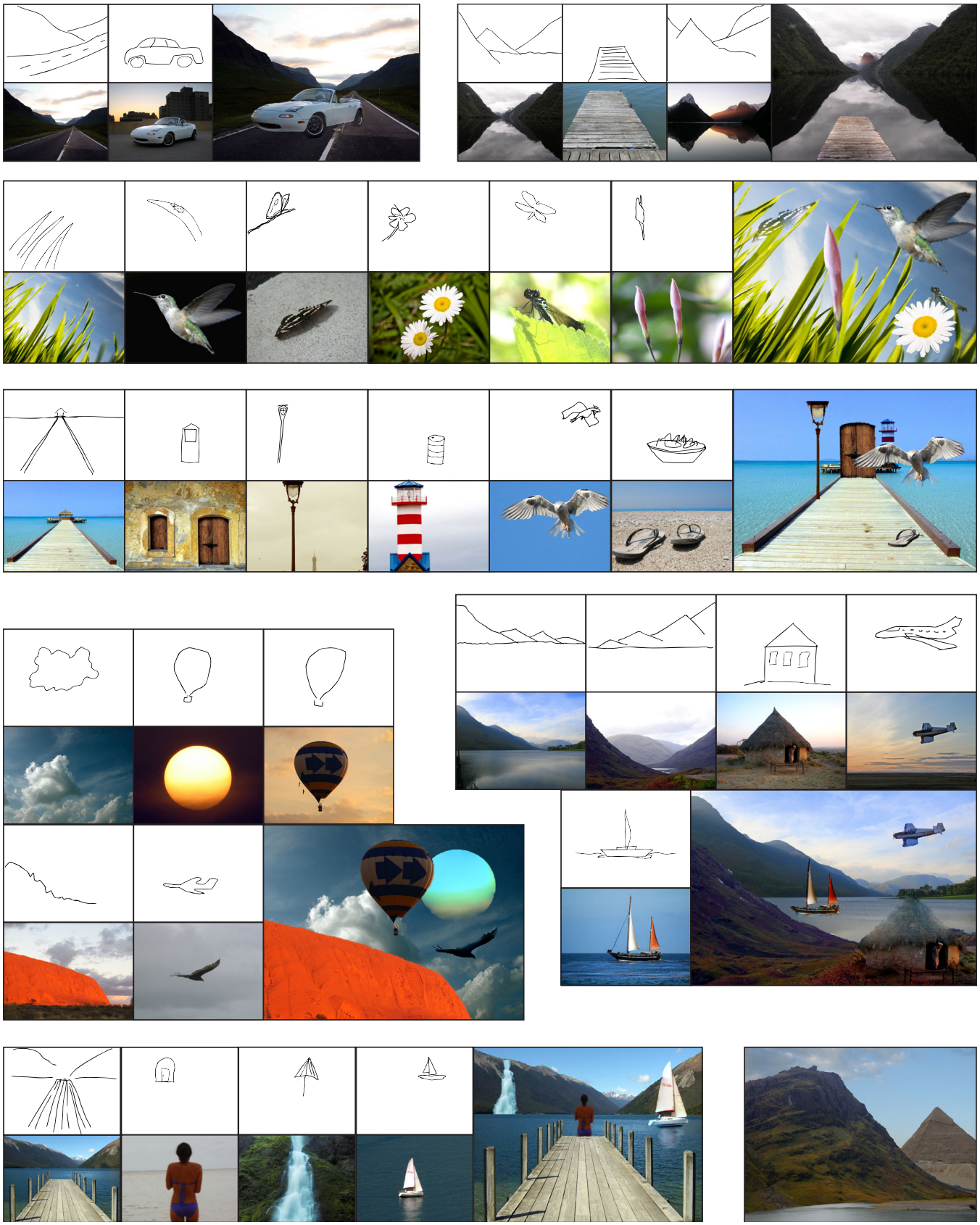

Figure 4: *Final compositing results created with the Photosketcher system. The user's input sketches are shown in the top row, chosen search results below and the final composite on the right. Bottom right: can you determine the number of components used? (solution: 4 – mountain, water, pyramid and clouds). Note that users not always choose images that match the input sketches (e.g. bottom row: door and umbrella).*

Figure 5: *Comparison of blending options in Photosketcher: from left to right: copy, feathering, gradient domain. Note that gradient domain blending is able to hide the artifact between the two birds but also drastically changes their overall color.*

tion problem:

$$\tilde{\boldsymbol{S}}_\Omega = \arg\min_{\tilde{\boldsymbol{S}}_\Omega} \|\nabla\tilde{\boldsymbol{S}}_\Omega - \nabla\boldsymbol{S}_\Omega\|_2^2 \text{ with } \tilde{\boldsymbol{S}}_{\partial\Omega} = \boldsymbol{B}_{\partial\Omega}, \quad (5)$$

and its minimizer $\tilde{\boldsymbol{S}}_\Omega$ is known to be the solution to the Poisson equation with Dirichlet boundary conditions [Pérez et al. 2003]. The resulting system matrix is symmetric and positive definite which means we can solve for $\tilde{\boldsymbol{S}}_\Omega$ very efficiently by computing a Cholesky decomposition of the system matrix and performing backsubstitution. We reuse the decomposition for all color channels and only recompute the decomposition if Ω changes.

6 Results

We have implemented a prototype system and show results in Fig. 1 and Fig. 4. The user interface is exclusively operated by sketches and scribbles for searching and compositing, respectively. The use of simple outlines for querying the database as well as the use of scribbles in the compositing step has been intuitive in our experiments with the Photosketcher system. We now discuss results of all parts of the Photosketcher pipeline.

Image collection and pre-processing We use a collection of 1.5 million images downloaded from Flickr. Each image has a resolution of 1024×768 pixels and is stored in JPEG format. The complete collection takes approximately 400Gb of disk space. Computing descriptors, learning the visual vocabulary and computing the inverting index takes approximately 24 hours on a single modern 8-core machine (all steps have been parallelized such that all cores are fully utilized). We extract 500 random samples from each image and use a visual vocabulary of 1000 visual words.

Evaluation of sketch-based retrieval We have performed an objective evaluation of the underlying sketch-based retrieval system described in Sec. 4. Using a set of 20 typical user sketches we have automatically generated the top 36 matches for each sketch (we also show 36 matches in our GUI). Showing a sketch along with a corresponding retrieval result, we have asked seven users (including two of the authors) to give binary relevance ratings, determining whether a result is a good match or not. Our evaluation shows that on average 29.1% of the images are considered good matches. In Fig. 6 we show more detailed evaluation plots: there seems to be a clear difference in what images users consider relevant. While participant #1 has been very strict and considered only 9.2% relevant, participant #7 considered 52.9% of the images to be relevant matches (the authors are #4 and #5). Also retrieval quality clearly depends on the input sketches: while sketch #9 generated 57.9% relevant matches, sketch #10 only generated 4.4%. While those numbers show that there is much room for improvement in precision (as expected), they also tell us that among the results we display in the GUI, several relevant results appear and users are typically able to spot and select those very quickly. Additionally, since our system is interactive, users can simply modify their sketches to retrieve new results in case none of the matches is satisfying – an important option that is not reflected in our evaluation numbers. Additionally, we show several sample retrieval results in Fig. 1 and Fig. 4.

Interactivity Overall, our experiments with Photosketcher show that one key aspect in its design is the interactivity of all pipeline steps: a) resketching to find better matches; b) optimizing a cut using scribbles and c) choosing among several alternative blending methods to further hide visible seams. All those steps provide interactive response times: our sketch-based search has response times of typically below 500ms; the iterative compositing between about 250ms and 1s (see Tab. 1 for more detailed response times). Note that both the GMM training stage (learning $\boldsymbol{\Theta}_{D,U}$ from $\Omega_{D,U}$) as well as the Graphcut step can be accelerated by reusing previous solutions once an initial solution has been found. The idea is that an additional user scribble typically encodes only *local* constraints and therefore should evoke mostly local changes in the new solution. In this case, we do not expect the model to change a lot and start the GMM training stage from the previously learned $\boldsymbol{\Theta}_{D,U}$. Typically, only a few iterations are needed until the GMM converges to the new solution.

Compositing All results in this paper have been generated using the compositing interface proposed in Sec. 5. We train our GMMs from L*a*b color triplets – we found this to work slightly better than using rgb triplets, probably due to the perceptually more uniform colorspace. We also found that the GMMs (using only pixel data for training) are often able to successfully make predictions about large, smooth regions. However, the prediction often fails at the borders of those regions. On close inspection, borders often have a very different color distribution than their adjacent regions (typically darker) and therefore a GMM is unable to predict those borders correctly in case corresponding training data has not been available. The compositing step suffers from imperfections in the images returned from the search: when searching for an object with a given shape, we do not consider the background around that object. A matching background texture and color is crucial, though, for achieving good results when employing Poisson blending. In a similar vein, it would be useful to align and snap the selected image objects to the current background.

7 Discussion and conclusion

Overall, the compositing step is an extremely difficult task due to mostly very dissimilar images that users try to merge. Our compositing step relies heavily on the Gaussian Mixture Models: if their prediction is "incorrect", this incorrect data is used as a prior for the Graphcut stage and the resulting seam will not be optimal. We usually use 5 components for the GMM, however an automatic selection could be desirable: if the image contains large, smooth areas, less components would be appropriate while other image areas containing detail and a wide variety of colors might require more than 5 components in order to be accurately represented. Compared to GrabCut [Rother et al. 2004] we find that learning full covariance matrices is possible in an interactive setting using a reasonable number of training samples (4000 in our case).

Note that our system (on purpose) does not automatically filter out search results that are hard to composite (as e.g. Sketch2Photo does [Chen et al. 2009]). Again we argue that interactivity is key: since all pipeline steps are interactive our strategy is to give complete control to our users and let *them* decide whether an automatic segmentation is acceptable, further manual work is required

Table 1: *Photosketcher pipeline stages timings (in ms).*

stage:	search	mask	gmm-train	gmm-predict	graphcut	blend
time (ms):	300	< 1	80 (initial) 10 (reused)	100	400 (inital) 100 (reused)	< 1 (copy) 10 (feathering) 500 (poisson)

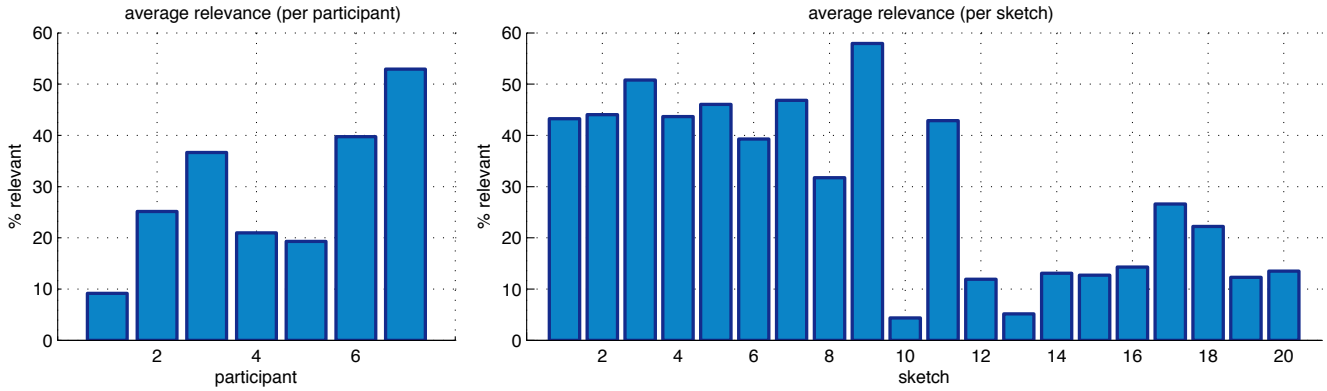

Figure 6: *Evaluation of sketch-based retrieval stage: we show the percentage of relevant results in the top 36 matches. Left: average percentage by participant (on average, participant 1 considered less results relevant than e.g. participant 3). Right: average percentage per sketch (our system returns fewer relevant results for sketch 10 than for e.g. sketch 11).*

or rather another image needs be selected.

While we do not require any metadata coming along with the images, constructing a search index (see Sec. 4) in an offline-precomputation step along with nearest-neighbor search during retrieval can be considered a sort of automatic tagging. Ideally, the descriptors describing our images in high-dimensional feature space would form semantically closely related clusters – in its current form, the retrieval stage frequently does not respect semantic closeness (which can be observed in Fig. 4).

Failure cases We observed failure cases mostly in 3 situations:

- The image collection does not contain any object semantically close to the searched one. We admit that it is difficult to determine whether the desired image is not in the collection or whether the sketch-based image retrieval step was the limiting factor. Note however that geometric similarity is almost always matched.
- Abstract sketches with a very specific semantic meaning attached to them – such as stick figures for instance – do not result in meaningful query results. This is since Photosketcher only measures the *geometric* similarity between a sketch and a picture for providing it as a query result.
- Photosketcher relies on the compositing step to generate convincing results. Often this is an extremely difficult task, e.g. when background textures do not match or the lighting of the part to be inserted is too different from that of the existing image.

Future work A possible direction for future work in sketch-based retrieval could be to consider *perceptual* gradient orientations, as long structures in a given direction are perceived as more important, even if unstructured gradients define another direction in the same area (e.g. tree roots contours vs. grass direction). The Canny edge detector has problems in such cases and we believe that generating high quality line drawings from the image collection is essential for successful sketch-based retrieval.

Extending sketch-based retrieval systems to handle abstract or symbolic sketches and more semantic information, is definitely an interesting and important direction for future work.

An interesting observation is the dependence of the system on the collection content. Our collection contains very few objects in a simple frontal view (i.e. the front side of the house, the side view of a car). However, most users tend to sketch objects from these points of view and will find that only few images match their sketch – simply because there are no objects in the collection with silhouettes as sketched by the user. We believe that making a system return perspectively projected views from a simple frontal view would help boosting sketch-based retrieval performance tremendously.

A final aspect for future work would be to consider the existing image when searching for parts to be added – this could be realized by additionally ranking the objects according to the estimated composite quality. This is an approach also taken in Photo Clip Art [Lalonde et al. 2007]. Note that this would be much harder in Photosketcher since the image collection used is much larger and there is no presegmentation of all images into objects from which the desired properties could be precomputed.

Conclusion Photosketcher makes a first step towards the creation of new image content from simple sketches of feature lines, exploiting large online image collections *without* the need of additional metadata besides the raw images. In case high-quality metadata is available, this can be exploited and existing systems such as Photo Clip Art and Sketch2Photo [Lalonde et al. 2007; Chen et al. 2009] have shown extremely convincing results using this approach. The results of the final composite images generated by Photosketcher are very promising, despite the fact that two inherently difficult tasks – finding images according to sketches and compositing unrelated images – are combined into a single system. We found that interactivity is a critical component for usability of the system: users explore the space of possible pictures progressively and typically go through an intensive try-and-test session where all these steps

might be repeated several times. In such a context, we believe that Photosketcher offers an advantage over offline systems such as e.g. Sketch2Photo [Chen et al. 2009].

References

AGARWALA, A., DONTCHEVA, M., AGRAWALA, M., DRUCKER, S., COLBURN, A., CURLESS, B., SALESIN, D., AND COHEN, M. 2004. Interactive Digital Photomontage. *ACM Transactions on Graphics 23*, 3, 294–302.

BISHOP, C. 2006. *Pattern Recognition and Machine Learning*. Springer.

BOYKOV, Y., AND KOLMOGOROV, V. 2004. An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26*, 9, 1124–1137.

CHEN, T., CHENG, M.-M., TAN, P., SHAMIR, A., AND HU, S.-M. 2009. Sketch2Photo: Internet Image Montage. *ACM Transactions on Graphics 28*, 5, 124:1–124:10.

EITZ, M., HILDEBRAND, K., BOUBEKEUR, T., AND ALEXA, M. 2009. PhotoSketch: a sketch based image query and compositing system. In *ACM SIGGRAPH 2009 Talk Program*.

EITZ, M., HILDEBRAND, K., BOUBEKEUR, T., AND ALEXA, M. 2011. Sketch-Based Image Retrieval: Benchmark and Bag-of-Features Descriptors. *IEEE Transactions on Visualization and Graphics*, PrePrint.

HAYS, J., AND EFROS, A. A. 2007. Scene completion using millions of photographs. *ACM Transactions on Graphics 26*, 3, 4:1–4:7.

JOHNSON, M., BROSTOW, G., SHOTTON, J., ARANDJELOVIC, O., KWATRA, V., AND CIPOLLA, R. 2006. Semantic Photo Synthesis. *Computer Graphics Forum 25*, 3, 407–413.

JOHNSON, M., DALE, K., AVIDAN, S., PFISTER, H., FREEMAN, W. T., AND MATUSIK, W. 2011. CG2Real: Improving the Realism of Computer Generated Images using a Large Collection of Photographs. *IEEE Transactions on Visualization and Computer Graphics 17*, 9, 1273–1285.

KWATRA, V., SCHÖDL, A., ESSA, I., TURK, G., AND BOBICK, A. 2003. Graphcut Textures: Image and Video Synthesis Using Graph Cuts. *ACM Transactions on Graphics 22*, 3, 277–286.

LALONDE, J.-F., HOIEM, D., EFROS, A. A., ROTHER, C., WINN, J., AND CRIMINISI, A. 2007. Photo Clip Art. *ACM Transactions on Graphics 26*, 3, 3:1–3:10.

PÉREZ, P., GANGNET, M., AND BLAKE, A. 2003. Poisson Image Editing. *ACM Transactions on Graphics 22*, 3, 313–318.

ROTHER, C., KOLMOGOROV, V., AND BLAKE, A. 2004. "GrabCut" – Interactive Foreground Extraction using Iterated Graph Cuts. *ACM Transactions on Graphics 23*, 3, 309–314.

ROTHER, C., BORDEAUX, L., HAMADI, Y., AND BLAKE, A. 2006. Autocollage. *ACM Transactions on Graphics 25*, 3, 847–852.

SIVIC, J., AND ZISSERMAN, A. 2003. Video Google: A Text Retrieval Approach to Object Matching in Videos. In *IEEE International Conference on Computer Vision*, 1470–1477.