How Do Humans Sketch Objects?

Mathias Eitz[*]
TU Berlin

James Hays[†]
Brown University

Marc Alexa [‡]
TU Berlin

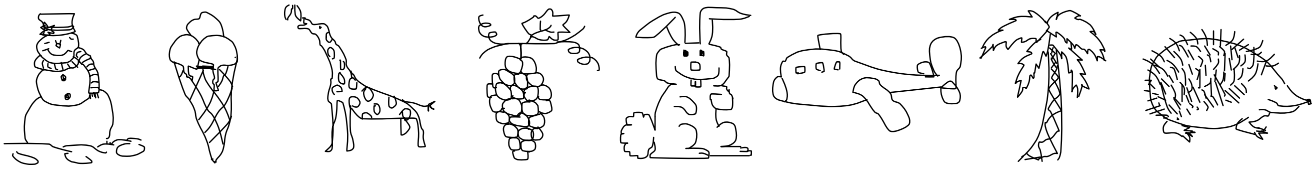

Figure 1: *In this paper we explore how humans sketch and recognize objects from 250 categories – such as the ones shown above.*

Abstract

Humans have used sketching to depict our visual world since prehistoric times. Even today, sketching is possibly the only rendering technique readily available to all humans. This paper is the first large scale exploration of human sketches. We analyze the distribution of non-expert sketches of everyday objects such as 'teapot' or 'car'. We ask humans to sketch objects of a given category and gather 20,000 unique sketches evenly distributed over 250 object categories. With this dataset we perform a perceptual study and find that humans can correctly identify the object category of a sketch 73% of the time. We compare human performance against computational recognition methods. We develop a bag-of-features sketch representation and use multi-class support vector machines, trained on our sketch dataset, to classify sketches. The resulting recognition method is able to identify unknown sketches with 56% accuracy (chance is 0.4%). Based on the computational model, we demonstrate an interactive sketch recognition system. We release the complete crowd-sourced dataset of sketches to the community.

CR Categories: I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques;

Keywords: sketch, recognition, learning, crowd-sourcing

Links: DL PDF

1 Introduction

Sketching is a universal form of communication. Since prehistoric times, people have rendered the visual world in sketch-like petroglyphs or cave paintings. Such pictographs predate the appearance of language by tens of thousands of years and today the ability to draw and recognize sketched objects is ubiquitous. In fact, recent neuroscience work suggests that simple, abstracted sketches activate our brain in similar ways to real stimuli [Walther et al. 2011]. Despite decades of graphics research, sketching is the *only* mechanism for most people to render visual content. However, there has never been a formal study of how people sketch objects and how well such sketches can be recognized by humans and computers. We examine these topics for the first time and demonstrate applications of computational sketch understanding. In this paper we use the term 'sketch' to mean a non-expert, abstract pictograph and not to imply any particular medium (e.g. pencil and paper).

There exists significant prior research on retrieving images or 3d models based on sketches. The assumption in all of these works is that, in some well-engineered feature space, sketched objects resemble their real-world counterparts. But this fundamental assumption is often violated – most humans are not faithful artists. Instead people use *shared, iconic* representations of objects (e.g. stick figures) or they make dramatic simplifications or exaggerations (e.g. pronounced ears on rabbits). Thus to understand and recognize sketches an algorithm must learn from a training dataset of real *sketches*, not photos or 3d models. Because people represent the same object using differing degrees of realism and distinct drawing styles (see Fig. 1), we need a large dataset of sketches which adequately samples these variations.

There also exists prior research in sketch recognition which tries to identify predefined glyphs in narrow domains (e.g. wire diagrams, musical scores). We instead identify objects such as snowmen, ice cream cones, giraffes, etc. This task is hard, because an average human is, unfortunately, not a faithful artist. Although both shape and proportions of a sketched object may be far from that of the corresponding real object, and at the same time sketches are an impoverished visual representation, humans are amazingly accurate at interpreting such sketches.

We first define a taxonomy of 250 object categories and acquire a large dataset of human sketches for the categories using crowdsourcing (Sec. 3). Based on the dataset we estimate how humans perform in recognizing the categories for each sketch (Sec. 4). We design a robust visual feature descriptor for sketches (Sec. 5). This feature permits not only unsupervised analysis of the dataset (Sec. 6), but also the computational recognition of sketches (Sec. 7). While we achieve a high computational recognition accuracy of 56% (chance is 0.4%), our study also reveals that humans still perform significantly better than computers at this task. We show several interesting applications of the computational model (Sec. 8): apart from the interactive recognition of sketches itself, we also demonstrate that recognizing the category of a sketch could improve image retrieval. We hope that the use of sketching as a visual input modality opens up computing technology to a significantly larger user base than text input alone. This paper is a first step

[*]e-mail: m.eitz@tu-berlin.de
[†]e-mail: hays@cs.brown.edu
[‡]e-mail: marc.alexa@tu-berlin.de

toward this goal and we release the dataset to encourage future research in this domain.

2 Related work

Our work has been inspired by and made possible by recent progress in several different areas, which we review below.

2.1 Sketch-based retrieval and synthesis

Instead of an example image as in content-based retrieval [Datta et al. 2008], user input for sketch-based retrieval is a simple binary sketch – exactly as in our setting. The huge difference compared to our approach is that these methods do not learn from example sketches and thus generally do *not achieve semantic understanding* of a sketch. Retrieval results are purely based on geometric similarity between the sketch and the image content [Chalechale et al. 2005; Eitz et al. 2011a; Shrivastava et al. 2011; Eitz et al. 2012]. This can help make retrieval efficient as it often can be cast as a nearest-neighbor problem [Samet 2006]. However, retrieving perceptually meaningful results can be difficult as users generally draw sketches in an abstract way that is geometrically far from the real photographs or models (though still recognizable for humans as we demonstrate later in this paper).

Several image synthesis systems build upon the recent progress in sketch-based retrieval and allow users to create novel, realistic imagery using sketch exemplars. Synthesis systems that are based on user sketches alone have to rely on huge amounts of data to offset the problem of geometric dissimilarity between sketches and image content [Eitz et al. 2011b] or require users to augment the sketches with text labels [Chen et al. 2009]. Using template matching to identify face parts, Dixon et al. [2010] propose a system that helps users get proportions right when sketching portraits. Lee et al. [2011] build upon this idea and generalize real-time feedback assisted sketching to a few dozen object categories. Their approach uses fast nearest neighbor matching to find geometrically similar objects and blends those object edges into rough shadow guidelines. As with other sketch-based retrieval systems, users must draw edges faithfully for the retrieval to work in the presence of many object categories – poor artists see no benefit from the system.

2.2 Sketch recognition

While there is no previous work on recognizing sketched objects, there is significant research on recognizing simpler, domain specific sketches. The very first works on sketch-recognition [Sutherland 1964; Herot 1976] introduce sketching as a means of human-computer interaction and provide – nowadays ubiquitous – tools such as drawing lines and curves using mouse or pen. More recent approaches try to understand human sketch input at a higher level. They are tuned to automatically identify a small variety of stroke types, such as lines, circles or arcs from noisy user input and achieve near perfect recognition rates in real-time for those tasks [Sezgin et al. 2001; Paulson and Hammond 2008]. Hammond and Davis [2005] exploit these lower-level recognizers to identify higher level symbols in hand-drawn diagrams. If the application domain contains a lot of structure – as in the case of chemical molecules or mathematic equations and diagrams – this can be exploited to achieve very high recognition rates [LaViola Jr. and Zeleznik 2007; Ouyang and Davis 2011].

2.3 Object and scene classification

Our overall approach is broadly similar to recent work in the computer vision community in which large, categorical databases of visual phenomena are used to train recognition systems. High-profile examples of this include the Caltech-256 database of object images [Griffin et al. 2007], the SUN database of scenes [Xiao et al. 2010], and the LabelMe [Russell et al. 2008] and Pascal VOC [Everingham et al. 2010] databases of spatially annotated objects in scenes. The considerable effort that goes into building these databases has allowed algorithms to learn increasingly effective classifiers and to compare recognition systems on common benchmarks. Our pipeline is similar to many modern computer vision algorithms although we are working in a *new domain* which requires a new, carefully tailored representation. We also need to generate our data from scratch because there are no preexisting, large repositories of sketches as there are for images (e.g. Flickr). For this reason we utilize crowd-sourcing to create a database of human object sketches and hope that it will be as useful to the community as existing databases in other visual domains.

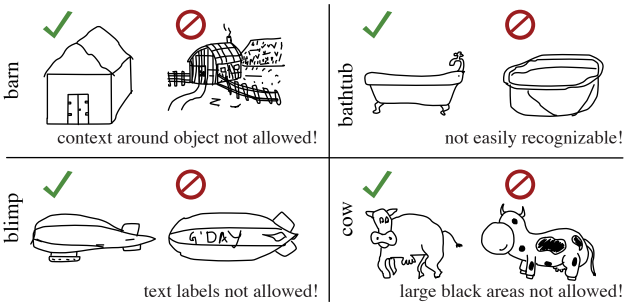

Figure 2: *Instructional examples shown to workers on Mechanical Turk. In each field: desired sketching style (left), undesired sketching style (right).*

3 A large dataset of human object sketches

In this section we describe the collection of a dataset of 20,000 human sketches. This categorical database is the basis for all learning, evaluation, and applications in this paper. We define the following set of criteria for the object categories in our database:

Exhaustive The categories exhaustively cover most objects that we commonly encounter in everyday life. We want a broad taxonomy of object categories in order to make the results interesting and useful in practice and to avoid superficially simplifying the recognition task.

Recognizable The categories are recognizable from their shape alone and do not require context for recognition.

Specific Finally, the categories are specific enough to have relatively few visual manifestations. 'Animal' or 'musical instrument' would not be good object categories as they have many subcategories.

3.1 Defining a taxonomy of 250 object categories

In order to identify common objects, we start by extracting the 1,000 most frequent labels from the LabelMe [Russell et al. 2008] dataset. We manually remove duplicates (e.g. car side vs. car front) as well as labels that do not follow our criteria. This gives us an initial set of categories. We augment this with categories from the Princeton Shape Benchmark [Shilane et al. 2004] and the Caltech 256 dataset [Griffin et al. 2007]. Finally, we add categories by asking members of our lab to suggest object categories that are not yet in the list. Our current set of 250 categories is quite exhaus-

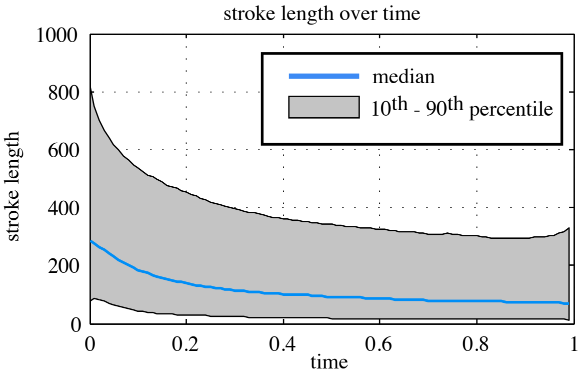

Figure 3: *Stroke length in sketches over drawing time: initial strokes are significantly longer than later in the sketching process. On the x-axis, time is normalized for each sketch.*

tive as we find it increasingly difficult to come up with additional categories that adhere to the desiderata outlined above.

3.2 Collecting 20,000 sketches

We ask participants to draw one sketch at a time given a random category name. For each sketch, participants start with an empty canvas and have up to 30 minutes to create the final version of the sketch. We keep our instructions as simple as possible and ask participants to "sketch an image [...] that is clearly recognizable to other humans as belonging to the following category: [...]". We also ask users to draw outlines only and not use context around the actual object. We provide visual examples that illustrate these requirements, see Fig. 2. We provide undo, redo, clear, and delete buttons for our stroke-based sketching canvas so that participants can easily familiarize themselves with the tool while drawing their first sketch. After finishing a sketch participants can move on and draw another sketch given a new category. In addition to the spatial parameters of the sketched strokes we store their temporal order.

Crowd-sourcing Collecting 20,000 sketches requires a huge number of participants so we rely on crowd-sourcing to generate our dataset. We use Amazon Mechanical Turk (AMT) which is a web-based market where requesters can offer paid "Human Intelligence Tasks" (HITs) to a pool of non-expert workers. We submit $90 \times 250 = 22{,}500$ HITs, requesting 90 sketches for each of the 250 categories. In order to ensure a diverse set of sketches within each category, we limit the number of sketches a worker could draw to one per category.

In total, we receive sketches from 1,350 unique participants who spent a total of 741 hours to draw all sketches. The median drawing time per sketch is 86 seconds with the 10th and 90th percentile at 31 and 280 seconds, respectively. The participants draw a total of 351,060 strokes with each sketch containing a median number of 13 strokes. We find that the first few strokes of a sketch are on average considerably longer than the remaining ones, see Fig. 3. This suggests that humans tend to follow a coarse-to-fine drawing strategy, first outlining the shape using longer strokes and then adding detail at the end of the sketching process.

Our sketching task appears to be quite popular on Mechanical Turk – we received a great deal of positive feedback, the time to complete all HITs was low, and very few sketches were unusable, adversarial, or automated responses. Still, as with any crowd-sourced data collection effort, steps must be taken to ensure that data collected from non-expert, untrained users is of sufficient quality.

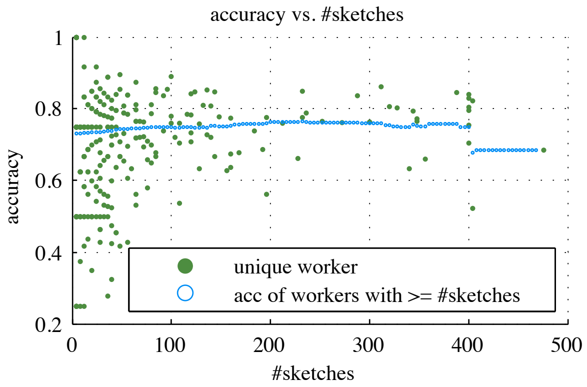

Figure 4: *Scatter plot of per-worker sketch recognition performance. Solid (green) dots represent single, unique workers and give their average classification accuracy (y-axis) at the number of sketches they classified (x-axis). Outlined (blue) dots represent overall average accuracy of all workers that have worked on more than the number of sketches indicated on the x-axis.*

Data verification We manually inspect and clean the complete dataset using a simple interactive tool we implemented for this purpose. The tool displays all sketches of a given category on a large screen which lets us identify incorrect ones at a glance. We remove sketches that are clearly in the wrong category (e.g. an airplane in the teapot category), contain offensive content or otherwise do not follow our requirements (typically excessive context). We do *not* remove sketches just because they are poorly drawn. As a result of this procedure, we remove about 6.3% of the sketches. We truncate the dataset to contain exactly 80 sketches per category yielding our final dataset of 20,000 sketches. We make the categories uniformly sized to simplify training and testing (e.g. we avoid the need to correct for bias toward the larger classes when learning a classifier).

4 Human sketch recognition

In this section we analyze human sketch recognition performance and use the dataset gathered in Sec. 3 for this task. Our basic questions are the following: given a sketch, what is the accuracy with which humans correctly identify its category? Are there categories that are easier/more difficult to discern for humans? To provide answers to those questions, we perform a second large-scale, crowd-sourced study (again using Amazon Mechanical Turk) in which we ask participants to identify the category of query sketches. This test provides us with an important human baseline which we later compare against our computational recognition method. We invite the reader to try this test on the sketches shown in Fig. 1: can you correctly identify all categories and solve the riddle hidden in this figure?

4.1 Methodology

Given a random sketch, we ask participants to select the best fitting category from the set of 250 object categories. We give workers unlimited time, although workers are naturally incentivized to work quickly for greater pay. To avoid the frustration of scrolling through a list of 250 categories for each query, we roughly follow Xiao et al. [2010] and organize the categories in an intuitive 3-level hierarchy, containing 6 top-level and 27 second-level categories such as 'animals', 'buildings' and 'musical instruments'.

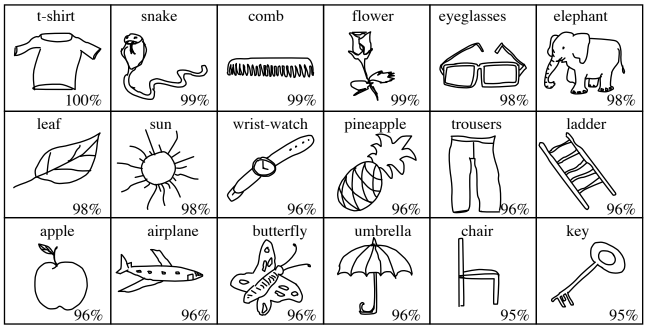

Figure 5: *Representative sketches from 18 categories with highest human category recognition rate (bottom right corner, in percent).*

We submit a total of 5,000 HITs to Mechanical Turk, each requiring workers to sequentially identify four sketches from random categories. This gives us one human classification result for each of the 20,000 sketches. We include several sketches per HIT to prevent workers from skipping tasks that contain 'difficult' sketches based on AMT preview functions as this would artificially inflate the accuracy of certain workers. In order to measure performance from many participants, we limit the maximum paid HITs to 100 per worker, i.e. 400 sketches (however, three workers did 101, 101 and 119 HITs, respectively, see Fig. 4).

4.2 Human classification results

Humans recognize on average 73.1% percent of all sketches correctly. We observe a large variance over the categories: while all participants correctly identified all instances of 't-shirt', the 'seagull' category was only recognized 2.5% of the time. We visualize the categories with highest human recognition performance in Fig. 5 and those with lowest performance in Fig. 6 (along with the most confusing categories). Human errors are usually confusions between semantically similar categories (e.g. 'panda' and 'bear'), although geometric similarity accounts for some errors (e.g. 'tire' and 'donut').

If we assume that it takes participants a while to learn our taxonomy and hierarchy of objects, we would expect that workers who have done more HITs are more accurate. However, this effect is not very pronounced in our experiments – if we remove all results from workers that have done less than 40 sketches, the accuracy of the remaining workers rises slightly to 73.9%. This suggests that there are no strong learning effects for this task. We visualize the accuracy of each single worker as well as the overall accuracy when gradually removing workers that have classified less than a certain number of sketches in Fig. 4.

5 Sketch representation

In our setting, we consider a sketch $\boldsymbol{S}$ as a bitmap image, with $\boldsymbol{S} \in \mathbb{R}^{m \times n}$. While the dataset from Sec. 3 is inherently vector-valued, a bitmap-based approach is more general: it lets us readily analyze any existing sketches even if they are only available in a bitmap representation.

An ideal sketch representation for our purposes is invariant to irrelevant features (e.g. scale, translation), discriminative between categories, and compact. More formally, we are looking for a feature space transform that maps a sketch bitmap to a lower-dimensional representation $\boldsymbol{x} \in \mathbb{R}^d$, i.e. a mapping $f : \mathbb{R}^{m \times n} \rightarrow \mathbb{R}^d$ with (typically) $d \ll m \times n$. In the remainder of this paper we call this representation either a feature vector or a descriptor. Ideally, the mapping f preserves the information necessary for $\boldsymbol{x}$ to be distinguished from all sketches in other categories.

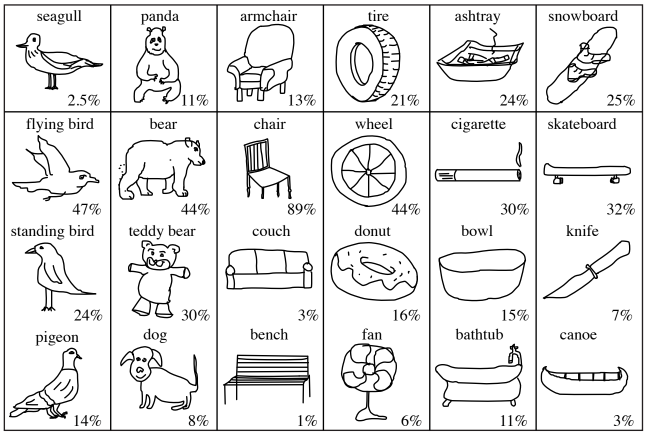

Figure 6: *Top row: six most difficult classes for human recognition. E.g., only 2.5% of all seagull sketches are correctly identified as such by humans. Instead, humans often mistake sketches belonging to the classes shown in the rows below as seagulls. Out of all sketches confused with seagull, 47% belong to flying bird, 24% to standing bird and 14% to pigeon. The remaining 15% (not shown in this figure) are distributed over various other classes.*

Probably the most direct way to define a feature space for sketches is to directly use its (possibly down-scaled) bitmap representation. Such representations work poorly in our experiments. Instead we adopt methods from computer vision and represent sketches using local feature vectors that encode *distributions* of image properties. Specifically, we encode the distribution of *line orientation* within a small local region of a sketch. The binning process during construction of the distribution histograms facilitates better invariance to slight offsets in orientation compared to directly encoding pixel values.

In the remainder of this paper, we use the following notational conventions: k denotes a scalar value, $\boldsymbol{h}$ a column vector, $\boldsymbol{S}$ a matrix and $\mathcal{V}$ a set.

5.1 Extracting local features

First, we achieve global scale and translation invariance by isotropically rescaling each sketch such that the longest side of its bounding box has a fixed length and each sketch is centered in a 256 × 256 image.

We build upon on a bag-of-features representation [Sivic and Zisserman 2003] as an intermediate step to define the mapping f. We represent a sketch as a large number of local features that encode local orientation estimates (but do *not* carry any information about their spatial location in the sketch). We write $\boldsymbol{g}_{uv} = \nabla \boldsymbol{S}$ for the gradient of $\boldsymbol{S}$ at coordinate (u, v) and $o_{uv} \in [0, \pi)$ for its orientation. We compute gradients using Gaussian derivatives to achieve reliable orientation estimates. We coarsely bin $||\boldsymbol{g}||$ into r orientation bins according to o, linearly interpolating into the neighboring bins to avoid sharp energy changes at bin borders. This gives us r orientational response images $\boldsymbol{O}$, each encoding the fraction of orientational energy at a given discrete orientation value. We find that using $r = 4$ orientation bins works well.

For each response image, we extract a local descriptor $\boldsymbol{l}_j$ by binning the underlying orientational response values into a small, lo-

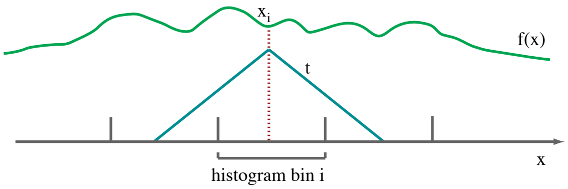

Figure 7: *1d example of fast histogram construction. The total energy in histogram bin i corresponds to $(f \star t)(x_i)$ where x_i is the center of bin i.*

cal histogram using 4×4 spatial bins, again linearly interpolating into neighboring bins. We build our final, local patch descriptor by stacking the orientational descriptors into a single column vector $\boldsymbol{d} = [\boldsymbol{l}_1, \ldots, \boldsymbol{l}_r]^T$. We normalize each local patch descriptor such that $\|\boldsymbol{d}\|_2 = 1$. This results in a representation that is closely related to the one used for SIFT [Lowe 2004] but stores orientations only [Eitz et al. 2011a].

While in computer vision applications the size of local patches used to analyze photographs is often quite small (e.g. 16×16 pixels [Lazebnik et al. 2006]), sketches contain little information at that scale and larger patch sizes are required for an effective representation. In our case, we use local patches covering an area of 12.5% of the size of $\boldsymbol{S}$. We use $28 \times 28 = 784$ regularly spaced sample points in $\boldsymbol{S}$ and extract a local feature for each point. The resulting representation is a so-called bag-of-features $\mathcal{D} = \{\boldsymbol{d}_i\}$.

Due to the relatively large patch size we use, the regions covered by the local features *significantly overlap* and each single pixel gets binned into about 100 distinct histograms. As this requires many image/histogram accesses, this operation can be quite slow. We speed up building the local descriptors by observing that the total energy accumulated in a single spatial histogram bin (using linear interpolation) is proportional to the convolution of the local image area with a 2d tent function having an extent of two times bin-width. We illustrate this property for the 1d case in Fig. 7. As a consequence, before creating spatial histograms, we first convolve the response images $\boldsymbol{O}_{1\ldots r}$ with the corresponding function (which we in turn speed up using the FFT). Filling a histogram bin is now reduced to a *single* lookup of the response at the center of each bin. This lets us efficiently extract a large number of local histograms which will be an important property later in this paper.

At this point, a sketch is represented as a so-called bag-of-features, containing a large number of local, 64-dimensional feature vectors (4×4 spatial bins and 4 orientational bins). In the following, we build a more compact representation by quantizing each feature against a visual vocabulary [Sivic and Zisserman 2003].

5.2 Building a visual vocabulary

Using a training set of n local descriptors $\boldsymbol{d}$ randomly sampled from our dataset of sketches, we construct a visual vocabulary using k-means clustering, which partitions the descriptors into k disjunct clusters C_i. More specifically, we define our visual vocabulary $\mathcal{V}$ to be the set of vectors $\{\boldsymbol{\mu}_i\}$ resulting from minimizing

$$\mathcal{V} = \underset{\{\boldsymbol{\mu}_i\}}{\arg\min} \sum_{i=1}^{k} \sum_{\boldsymbol{d}_j \in C_i} \|\boldsymbol{d}_j - \boldsymbol{\mu}_i\|^2 \quad (1)$$

with

$$\boldsymbol{\mu}_i = 1/|C_i| \sum_{\boldsymbol{d}_j \in C_i} \boldsymbol{d}_j.$$

5.3 Quantizing features

We represent a sketch as a frequency histogram of visual words $\boldsymbol{h}$ – this is the final representation we use throughout the paper. As a baseline we use a standard histogram of visual words using a 'hard' assignment of local features to visual words [Sivic and Zisserman 2003]. We compare this to using 'soft' kernel-codebook coding [Philbin et al. 2008] for constructing the histograms. The idea behind kernel codebook coding is that a feature vector may be equally close to multiple visual words but this information cannot be captured in the case of hard assignment. Instead we use a kernelized distance between descriptors that encodes weighted distances to all visual words – with a rapid falloff for distant visual words. More specifically, we define our histogram $\boldsymbol{h}$ as:

$$\boldsymbol{h}(\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{d}_i \in \mathcal{D}} q(\boldsymbol{d}_i) / \|q(\boldsymbol{d}_i)\|_1 \quad (2)$$

where $q(\boldsymbol{d})$ is a vector-valued quantization function that quantizes a local descriptor $\boldsymbol{d}$ against the visual vocabulary $\mathcal{V}$:

$$q(\boldsymbol{d}) = [K(\boldsymbol{d}, \boldsymbol{\mu}_1), \ldots, K(\boldsymbol{d}, \boldsymbol{\mu}_k)]^T$$

We use a Gaussian kernel to measure distances between samples, i.e.

$$K(\boldsymbol{d}, \boldsymbol{\mu}) = \exp(-\|\boldsymbol{d} - \boldsymbol{\mu}\|^2 / 2\sigma^2).$$

Note that in Eqn. (2) we normalize $\boldsymbol{h}$ by the number of samples to get to our final representation. Thus our representation is not sensitive to the total amount of local features in a sketch, but rather to local structure and orientation of lines. We use $\sigma = 0.1$ in our experiments.

6 Unsupervised dataset analysis

In this section we perform an automatic, unsupervised analysis of our sketch dataset making use of the feature space we developed in the previous section (each sketch is represented as a histogram of visual words $\boldsymbol{h}$). We would like to provide answers to the following questions:

- What is the distribution of sketches in the proposed feature space? Ideally, we would find clusters of sketches in this space that clearly represent our categories, i.e. we would hope to find that features *within* a category are close to each other while having large distances to all other features.
- Can we identify iconic sketches that are good representatives of a category?
- Can we visualize the distribution of sketches in our feature space? This would help build an intuition about the representative power of the feature transform developed in Sec. 5.

Our feature space is sparsely populated (only 20,000 points in a high-dimensional space). This makes clustering in this space a difficult problem. Efficient methods such as k-means clustering do not give meaningful clusters as they use rigid, simple distance metrics and require us to define the number of clusters beforehand. Instead, we use variable-bandwidth mean-shift clustering with locality sensitive hashing to speed up the underlying nearest-neighbor search problem [Georgescu et al. 2003]. Adaptive mean-shift estimates a density function in feature space for each histogram $\boldsymbol{h}$ as:

$$f(\boldsymbol{h}) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{b_i^d} K\left(\|\boldsymbol{h} - \boldsymbol{h}_i\|^2 / b_i^2\right),$$

where b_i is the bandwidth associated with each point and K is again a Gaussian kernel. Given this definition, we compute the modes, i.e.

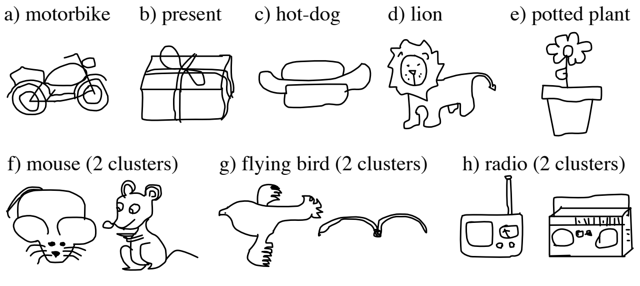

Figure 8: *A sample of the representative sketches automatically computed for each category. Bottom row: categories that produce multiple clusters and thus more than one representative.*

the maxima of f by using an iterative gradient ascent approach. As is standard, we assign all features $\boldsymbol{h}$ that are mean-shifted close to the same maximum to a common cluster.

6.1 Intra-category clustering analysis

We perform a local analysis of our dataset by independently running adaptive mean-shift clustering on the descriptors of each individual category. The resulting average number of clusters within our 250 categories is 1.39: the sketches *within* most categories in our dataset are reasonably self-similar. We can, however, identify categories with several distinct clusters in feature-space – for these categories people seem to have more than one iconic representation. We visualize examples of such categories in the bottom row of Fig. 8.

6.2 Iconic or representative sketches

We denote by $\mathcal{C}_i$ the set of all descriptors belonging to cluster i. To identify *iconic* sketches that might be good representatives of a cluster and thus the corresponding category, we propose the following strategy: a) compute the average feature vector $\boldsymbol{a}_i$ from all features in that cluster:

$$\boldsymbol{a}_i = 1/\left|\mathcal{C}_i\right| \sum_{\boldsymbol{h}_j \in \mathcal{C}_i} \boldsymbol{h}_j .$$

And b) given $\boldsymbol{a}_i$ find the closest actual descriptor – our final representative $\boldsymbol{r}_i$ – in that cluster:

$$\boldsymbol{r}_i = \operatorname*{arg\,min}_{\boldsymbol{h}_j \in \mathcal{C}_i} \left\|\boldsymbol{h}_j - \boldsymbol{a}_i\right\|^2 .$$

The sketches corresponding to the resulting $\boldsymbol{r}_i$'s are often clear representative sketches of our categories and we show several examples in Fig. 8.

6.3 Dimensionality reduction

To visualize the distribution of sketches in the feature space we apply dimensionality reduction to the feature vectors from each category. We seek to reduce their dimensionality to two dimensions such that we can visualize their distribution in 2d space, see Fig. 9. Using standard techniques such as PCA or multi-dimensional scaling for dimensionality reduction results in crowded plots: many datapoints fall close together in the mapped 2d space, resulting in unhelpful layouts. Van der Maaten and Hinton [2008] propose t-distributed stochastic neighbor embedding (t-SNE), a dimensionality reduction technique that specifically addresses this crowding problem: t-SNE computes a mapping of distances in high-dimensional space to distances in low-dimensional space such that smaller pairwise distances in high-dimensional space (which would produce the crowding problem) are mapped to larger distances in 2d while still preserving overall global distances.

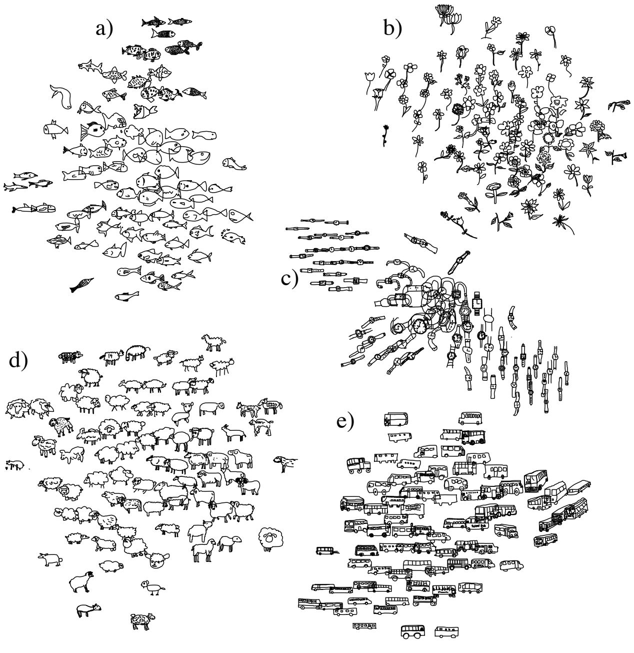

Figure 9: *Automatic layout of sketches generated by applying dimensionality reduction to the feature vectors within a category. a) fish; b) flower with stem; c) wrist watch; d) sheep and e) bus.*

We apply t-SNE to the feature vectors from all categories separately. The resulting layouts are an intuitive tool for quickly exploring the distinct types of shapes and drawing styles used by humans to represent a category. Some of the observed variations are continuous in nature (e.g. sketch complexity), but others are surprisingly discrete – there tend to be one or two canonical viewpoints or poses in a given category (see Fig. 9 for several examples).

7 Computational sketch recognition

In this section, we study computational sketch recognition using state of the art machine learning techniques. As a baseline, we employ standard nearest-neighbor classification which is fast, easy to implement, and does not require an explicit training step. We compare this to multi-class support vector machines (SVM), a popular and effective supervised learning technique [Schölkopf and Smola 2002]. All classification algorithms operate in the feature space described in Sec. 5 using the dataset from Sec. 3.

7.1 Models

Nearest-neighbor classification Given a histogram $\boldsymbol{h}$ we find its k nearest neighbors (knn) in feature space using a distance function d. We classify $\boldsymbol{h}$ as belonging to the category that the majority of k nearest neighbors belongs to.

SVM classification We train *binary* SVM classifiers to make the following decision: does a sketch belong to category i or does it rather belong to any of the remaining categories? We say $\operatorname{cat}(\boldsymbol{h}) = i$ to denote that the sketch corresponding to histogram $\boldsymbol{h}$ belongs to category i. More specifically, for each category i we learn a

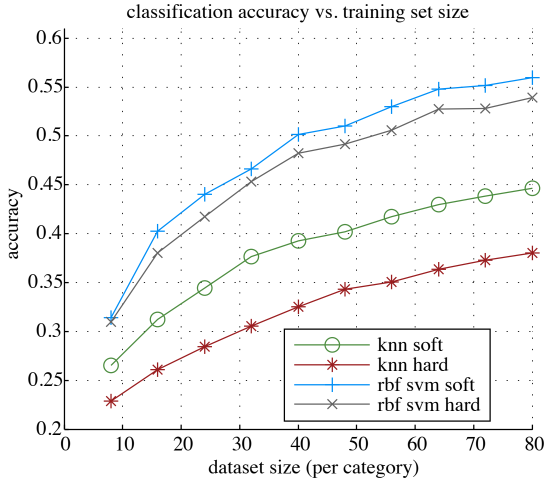

Figure 10: *Training set size vs. computational classification accuracy. All models use best parameters determined using grid-search. 'hard' and 'soft' refer to one nearest-neighbor and kernel-codebook quantization methods, respectively.*

classifier function

$$c^i(\boldsymbol{h}) = \sum_j \alpha_j^i K(\boldsymbol{s}_j^i, \boldsymbol{h}) + b \tag{3}$$

with support vectors $\boldsymbol{s}_j$, weights α_j and bias b determined during the SVM training phase. As before, $K(\cdot,\cdot)$ is a kernel (in our experiments, Gaussian) that measures similarity between a support vector and the histogram $\boldsymbol{h}$ that we wish to classify. Given a training dataset, we use the sketches from category i as the positive examples and all the remaining sketches as the negative examples.

SVMs are inherently *binary* classifiers but we wish to distinguish 250 categories from each other. We train 250 classifiers – one for each category, each able to discern its category from the union of the remaining categories (one-vs-all approach). To decide $\text{cat}(\boldsymbol{h})$ we classify a sketch as belonging to the category that yields the largest classification response, i.e.

$$\text{cat}(\boldsymbol{h}) = \underset{i=1,\ldots,250}{\arg\max}\; c^i(\boldsymbol{h}) \tag{4}$$

7.2 Recognition experiments

In this section we determine the best parameters for computational sketch recognition. In all cases, we train on a subset of the sketches while evaluating on a disjunct testing set. We always use 3-fold cross-validation: we partition the respective dataset into 3 parts, use 2 parts for training and the remaining part for testing. We use stratified sampling when creating the folds to ensure that each subset contains (approximately) the same number of category instances for a given category. We report performance of a model using its accuracy, averaged over all 3 folds, i.e. the ratio of correctly classified samples to the total number of positive samples.

Dataset & features We use the complete 250 category dataset described in Sec. 3 and rasterize each sketch into a grayscale bitmap of size 256 × 256. We extract 28 × 28 = 784 local features sampled on a regular grid from each sketch. To create a visual vocabulary, we need to learn from a large number of samples. For computational reasons, we generate the visual vocabulary $\mathcal{V}$ from a randomly sampled subset of all 20,000 × 784 local features (see Eqn. (1); we use n = 1,000,000 features). We set the number of visual words in the vocabulary to 500 as a coarse evaluation indicates good classification rates at a moderate cost for quantization. After quantization of the local feature vectors according to Eqn. (2), this results in a 500-dimensional histogram of visual words representing a sketch.

Table 1: *Best parameters for knn/SVM model using histograms generated with soft/hard quantization.*

	knn		SVM	
	k	d	γ	C
soft	4	l_1	17.8	3.2
hard	5	cosine	17.8	10

Model search Classification performance of both the knn as well as the SVM classifier can be quite sensitive to user-chosen model parameters. Therefore we first identify the best performing model parameters for each case. We perform a standard grid search over the 2d space of model parameters. For knn we use $\mathrm{k} \in \{1,\ldots,5\}$ (number of nearest neighbors used for classification) and distance measures $\mathrm{d} = \{\mathrm{l}_1, \mathrm{l}_2, \mathrm{cosine}, \mathrm{correlation}\}$. For SVM we use $\gamma \in \{10,\ldots,100\}$ (the Gaussian kernel parameter) and $C \in \{1,\ldots,100\}$ (the regularization constant), both with logarithmic spacing. We use a 1/4 subsample of the whole dataset to speed-up the search for SVM, for knn we use the full dataset. We list the resulting best model parameters in Tab. 1 and use those in the remainder of this paper.

Influence of training set size To identify how the amount of training data influences classification accuracy, we split the whole dataset into increasingly larger sub-datasets (8, 16, … ,80 sketches per category). For each of those sub-datasets, we measure average 3-fold cross-validation accuracy. It turns out that computational classification accuracy highly depends on the number of training instances available, see Fig. 10. This is to be expected as the sketches in many categories are highly diverse (see e.g. the sketches in the layouts shown in Fig. 9). The performance gain for larger training set sizes becomes smaller as we approach the full size of our dataset: this suggests that the dataset is large enough to capture most of the variance within each category.

Human vs. computational classification A manual inspection of the computational as well as human classification results reveals that the confusions humans make are often of hierarchical nature: bear and teddy bear are often confused with e.g. panda (see Fig. 6). We hypothesize that the participants in our experiment were satisfied once they found a category that matches reasonably well, and may not have noticed the presence of semantically similar categories in our taxonomy. And, indeed, computational classification can perform better in such cases: for 'armchair' and 'suv' the computational model achieves significantly higher accuracy than humans. To visualize such cases, we show the *difference* between human and computational confusion in Fig. 11 (for the set of categories with overall lowest human recognition performance).

Computational classification – in the case of a failure – sometimes makes predictions that are far off from what a human would possibly predict. It seems that despite the large dataset humans are still much better at generalizing from a smaller number of samples.

Computational classification summary Our experiments with computational classification clearly demonstrate that an SVM

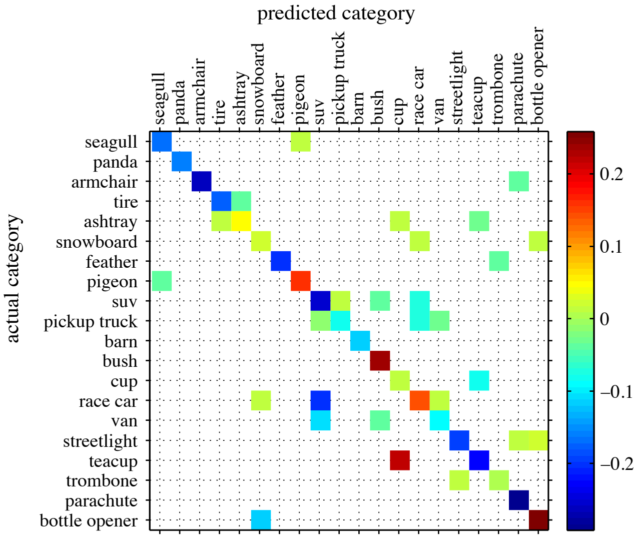

Figure 11: *Confusion matrix for selected categories showing* ***difference*** *between human and computational classification performance. Positive (red) entries mean humans are better, we hide the zero-level for clarity.*

model with kernel codebook coding for constructing the histograms is superior to other methods. A knn classification performs *significantly* worse for this task. In the next section we utilize our best performing computational model for several applications.

8 Applications

Computational sketch recognition lets us design several interesting applications. We build upon on the multi-class support vector sketch recognition engine as developed in Sec. 7.

8.1 Interactive sketch recognition

We propose an interactive human sketch recognition engine that recognizes sketches in real-time. After each stroke a user draws, we run the following recognition pipeline:

1. Extract local features from the sketch and quantize them against the visual vocabulary, this gives us a histogram of visual words $\boldsymbol{h}$.

2. Classify $\boldsymbol{h}$ using the SVM model according to Eqn. (4).

3. Display the top-20 categories with highest scores (rather than simply displaying the single best category).

The resulting application is highly interactive (about 100 ms for the complete recognition pipeline on a modern computer). Users report that the application is fun and intuitive to use. In case the classification is not as expected or the desired classification does not come up on rank one, we observe that users explore small modifications of their sketch until the classification is finally correct. Users are often amazed that even their very first sketch is correctly recognized – our system supports a large variety of sketching styles by learning from a large real-world set of sketches. As sketching is very natural to humans, there is basically no learning required to successfully use our system. The interactive application works well with either a touchscreen (as on a tablet) or with a mouse.

Recognition is also reasonably stable: as soon as a sketch is close to completion, oversketching, additional strokes, and even a small number of random background scribbles typically do not influence the recognition. On the other hand, if there are only slight differences between two categories, a single stroke can make all the difference: adding a handle to a bowl reliably turns it into a teacup, sketching vertical indentations on an apple turns it into a pumpkin. We show several examples in Fig. 12a but urge the reader to experience the interactivity of our system in the accompanying video.

8.2 Semantic sketch-based retrieval

Compared to existing sketch-based retrieval approaches [Chalechale et al. 2005; Eitz et al. 2011a; Shrivastava et al. 2011], we are now able to identify the *semantic* meaning of what a user sketches – even if the sketch is geometrically far from its corresponding real-world shape. This is a situation where traditional sketch-based retrieval engines naturally have problems. We propose the following extension to sketch-based image retrieval: a) perform classification on the user sketch and query a traditional keyword based search engine using the determined category; b) (optionally) re-order the resulting images according to their geometric similarity to the user sketch. For step b) we can make use any traditional sketch-based search engine. We demonstrate step a) of this approach on a dataset of 125,000 images downloaded from Google Images (for each category the top 500 images returned when searching for the keyword), see Fig. 12b for an example.

8.3 Recognizing artistic and historical sketches

A challenging evaluation for recognition systems is whether the method can generalize to styles not seen at training time. If sketch representations are really a universal, shared vocabulary for humans then our algorithm should be able to recognize existing pictographs from other domains. We present a limited example of this by running our sketch recognition pipeline on two famous sets of sketches – animal sketches by Pablo Picasso, and ancient cave paintings from Lascaux, France (we converted the cave paintings into sketches by manually tracing their feature lines). Our system predicts the dove, camel, cave horse correctly, and the antelope, which is not part of our categories, is classified as a sheep (see Fig. 12c).

9 Discussion and Limitations

The feature space for sketches we propose in this paper builds upon the successful bag-of-features model. However, this approach also comes with certain limitations, which we discuss in this section.

9.1 Spatial layout of strokes

The features do not encode spatial location, although the meaning of certain features might be dependent on context in the sketch. We experimented with the well known spatial pyramid representation [Lazebnik et al. 2006] (an overcomplete hierarchy of bags-of-words) but it did not significantly improve performance. We hypothesize that better representations of spatial information would significantly improve performance, but those representations might be distinct from the features developed for the image domain. We have not analyzed if it is advantageous to make features rotation invariant. It appears this would serve recognition at least for some categories, where sketches are clearly rotated/reflected versions of each other, see for example Fig. 9.

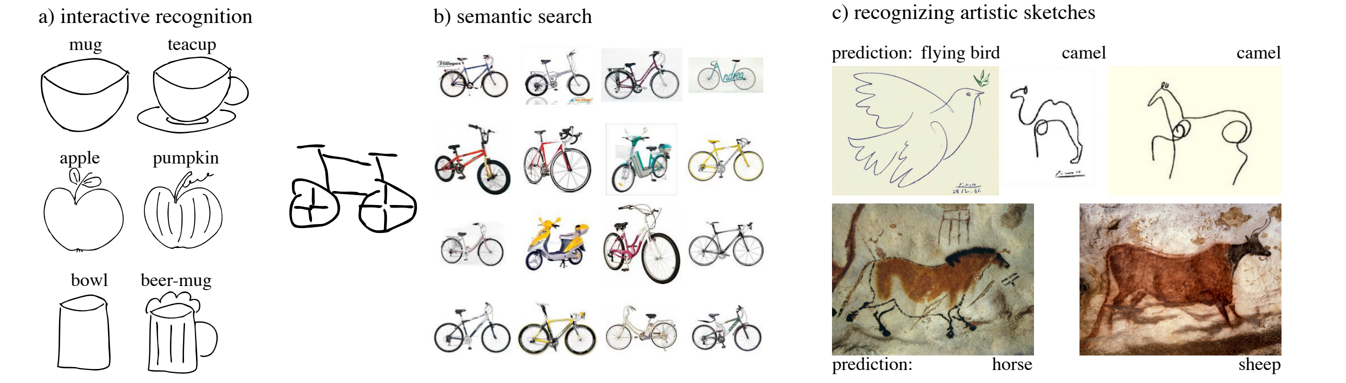

Figure 12: *Applications enabled by semantic sketch recognition: a) stable interactive recognition that reliably adapts its classification (left column) when adding additional detail (right column). b) semantic sketch based image search, the only input is the sketch on the left and c) recognition of artistic and ancient sketches.*

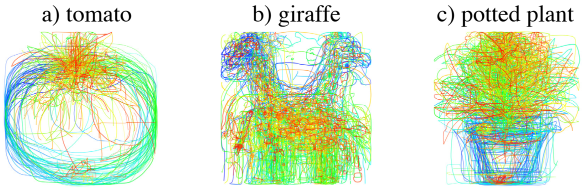

Figure 13: *Temporal order of strokes averaged for all sketches within a category. We color-code time, blue: beginning, red: end).*

9.2 Temporal information

The raw stroke-based sketch data contains information about the temporal order in which humans have drawn those sketches. The order of strokes seems to be quite consistent for certain types of sketches, see Fig. 13 for several examples. Recent research shows that a plausible order can even be automatically generated [Fu et al. 2011]. In our current representation we do not exploit temporal order of strokes. We have performed initial experiments with descriptors that additionally encode temporal order but have so far only achieved small improvements at the cost of a much higher-dimensional representation. Nevertheless, we believe that this is an interesting and fruitful area for further research. Note that our dataset can be directly exploited for such experiments as it comes with full temporal information.

9.3 Sketch representation

We have chosen to compute the bag-of-features from *rasterized* sketches. While this is general and accurate, there is no generative process to map from our features back to a sketch. A stroke-based model might be more natural and facilitate easier synthesis applications such as simplification, beautification, and even synthesis of novel sketches by mixing existing strokes.

9.4 Human sketch recognition

In the experiments we ran on AMT, we did not provide direct incentives for workers to achieve maximum recognition rate. Rather, the incentive was to complete the task as quickly as possible. We thus hypothesize that the human sketch recognition rate of 73% as determined in Sec. 4 is a lower bound to the actual maximally possible recognition rate.

10 Conclusions and Future Work

This paper is the first large scale exploration of human object sketches. We have collected the first dataset of sketches and used it to evaluate human recognition. We have demonstrated that – given such a large dataset – reasonable classification rates can be achieved for computational sketch recognition.

We feel that sketch synthesis is an interesting, unexplored problem. How could the computer generate distinctive sketches that are immediately recognizable by humans? If this question can be answered, many new applications could benefit from the research we have started here.

We also believe that the computational model could be useful for supervised simplification and beautification of sketches. Simplification has been well-studied in certain graphics domains such as 3d geometry [Garland and Heckbert 1997]. The general strategy for such techniques is to remove complexity (e.g. delete edges) while staying as close as possible to the original instance according to some geometric error metric. Such unsupervised heuristics have no semantic understanding of each instance and therefore will often make simplifications which are geometrically modest but perceptually jarring (e.g. smoothing away the face of a statue). We believe that an ideal simplification algorithm would consider the *semantic meaning* of each instance when deciding which simplifications to perform.

Another open question is how universal sketching and sketch recognition is among humans. Our sketches come from AMT workers all over the world, but it is certainly not a uniform sample of different cultures, ages, genders, artistic expertise, etc. How do these factors and many others affect sketching? Are the stylizations different cultures use for a certain object similar and even mutually recognizable?

Finally we hope that better computational understanding of sketches will lead to better computer accessibility. Virtually everybody is able to sketch a face or recognize a sketched face. Writing and reading, which today are still the standard way of communicating with computers, are much less widespread. By some definitions, functional illiteracy, even in first-world countries, is up to 20% of adults. If computers were to understand sketches as we do, sketching would give a much larger audience access to the data that has been gathered digitally over the last decades.

Acknowledgements

We are grateful to everyone on Amazon Mechanical Turk that helped create the large amount of sketches required for this work. This work has been supported in part by the ERC-2010-StG 259550 XSHAPE grant. This work has also been funded by NSF CAREER Award 1149853 to James Hays as well as gifts from Microsoft and Google.

References

Chalechale, A., Naghdy, G., and Mertins, A. 2005. Sketch-based image matching using angular partitioning. *IEEE Trans. Systems, Man and Cybernetics, Part A 35*, 1, 28–41.

Chen, T., Cheng, M., Tan, P., Shamir, A., and Hu, S. 2009. Sketch2Photo: internet image montage. *ACM Trans. Graph. (Proc. SIGGRAPH ASIA) 28*, 5, 124:1–124:10.

Datta, R., Joshi, D., Li, J., and Wang, J. 2008. Image retrieval: ideas, influences, and trends of the new age. *ACM Computing Surveys 40*, 2, 1–60.

Dixon, D., Prasad, M., and Hammond, T. 2010. iCanDraw?: using sketch recognition and corrective feedback to assist a user in drawing human faces. In *Proc. Int'l. Conf. on Human Factors in Computing Systems*, 897–906.

Eitz, M., Hildebrand, K., Boubekeur, T., and Alexa, M. 2011. Sketch-based image retrieval: benchmark and bag-of-features descriptors. *IEEE Trans. Visualization and Computer Graphics 17*, 11, 1624–1636.

Eitz, M., Richter, R., Hildebrand, K., Boubekeur, T., and Alexa, M. 2011. Photosketcher: interactive sketch-based image synthesis. *IEEE Computer Graphics and Applications 31*, 6, 56–66.

Eitz, M., Richter, R., Boubekeur, T., Hildebrand, K., and Alexa, M. 2012. Sketch-based shape retrieval. *ACM Trans. Graph. (Proc. SIGGRAPH) 31*, 4. to appear.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. 2010. The PASCAL visual object classes (VOC) challenge. *Int'l. Journal of Computer Vision 88*, 2, 303–338.

Fu, H., Zhou, S., Liu, L., and Mitra, N. 2011. Animated construction of line drawings. *ACM Trans. Graph. (Proc. SIGGRAPH ASIA) 30*, 6, 133:1–133:10.

Garland, M., and Heckbert, P. 1997. Surface simplification using quadric error metrics. In *Proc. SIGGRAPH*, 209–216.

Georgescu, B., Shimshoni, I., and Meer, P. 2003. Mean shift based clustering in high dimensions: a texture classification example. In *IEEE Int'l. Conf. Computer Vision*, 456–463.

Griffin, G., Holub, A., and Perona, P. 2007. Caltech-256 object category dataset. Tech. rep., California Institute of Technology.

Hammond, T., and Davis, R. 2005. LADDER, a sketching language for user interface developers. *Computers & Graphics 29*, 4, 518–532.

Herot, C. F. 1976. Graphical input through machine recognition of sketches. *Computer Graphics (Proc. SIGGRAPH) 10*, 2, 97–102.

LaViola Jr., J. J., and Zeleznik, R. 2007. MathPad: a system for the creation and exploration of mathematical sketches. *ACM Trans. Graph. (Proc. SIGGRAPH) 23*, 3, 432–440.

Lazebnik, S., Schmid, C., and Ponce, J. 2006. Beyond bags of features: spatial pyramid matching for recognizing natural scene categories. In *IEEE Conf. Computer Vision and Pattern Recognition*, 2169–2178.

Lee, Y., Zitnick, C., and Cohen, M. 2011. ShadowDraw: real-time user guidance for freehand drawing. *ACM Trans. Graph. (Proc. SIGGRAPH) 30*, 4, 27:1–27:10.

Lowe, D. G. 2004. Distinctive image features from scale-invariant keypoints. *Int'l. Journal of Computer Vision 60*, 2, 91–110.

Ouyang, T., and Davis, R. 2011. ChemInk: a natural real-time recognition system for chemical drawings. In *Proc. Int'l. Conf. Intelligent User Interfaces*, 267–276.

Paulson, B., and Hammond, T. 2008. PaleoSketch: accurate primitive sketch recognition and beautification. In *Proc. Int'l. Conf. Intelligent User Interfaces*, 1–10.

Philbin, J., Chum, O., Isard, M., Sivic, J., and Zisserman, A. 2008. Lost in quantization: improving particular object retrieval in large scale image databases. In *IEEE Conf. Computer Vision and Pattern Recognition*, 1–8.

Russell, B., Torralba, A., Murphy, K., and Freeman, W. 2008. LabelMe: a database and web-based tool for image annotation. *Int'l Journal of Computer Vision 77*, 1, 157–173.

Samet, H. 2006. *Foundations of multidimensional and metric data structures*. Morgan Kaufmann.

Schölkopf, B., and Smola, A. 2002. *Learning with kernels*. MIT Press.

Sezgin, T. M., Stahovich, T., and Davis, R. 2001. Sketch based interfaces: early processing for sketch understanding. In *Workshop on Perceptive User Interfaces*, 1–8.

Shilane, P., Min, P., Kazhdan, M., and Funkhouser, T. 2004. The Princeton Shape Benchmark. In *Shape Modeling International*, 167–178.

Shrivastava, A., Malisiewicz, T., Gupta, A., and Efros, A. A. 2011. Data-driven visual similarity for cross-domain image matching. *ACM Trans. Graph. (Proc. SIGGRAPH ASIA) 30*, 6, 154:1–154:10.

Sivic, J., and Zisserman, A. 2003. Video Google: a text retrieval approach to object matching in videos. In *IEEE Int'l. Conf. Computer Vision*, 1470–1477.

Sutherland, I. 1964. SketchPad: a man-machine graphical communication system. In *Proc. AFIPS*, 323–328.

van der Maaten, L., and Hinton, G. 2008. Visualizing data using t-SNE. *Journal of Machine Learning Research 9*, 2579–2605.

Walther, D., Chai, B., Caddigan, E., Beck, D., and Fei-Fei, L. 2011. Simple line drawings suffice for functional MRI decoding of natural scene categories. *Proc. National Academy of Sciences 108*, 23, 9661–9666.

Xiao, J., Hays, J., Ehinger, K. A., Oliva, A., and Torralba, A. 2010. SUN database: large-scale scene recognition from abbey to zoo. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 3485–3492.