ShadowDraw: Real-Time User Guidance for Freehand Drawing

Yong Jae Lee
University of Texas at Austin

C. Lawrence Zitnick
Microsoft Research

Michael F. Cohen
Microsoft Research

Figure 1: *Results of the user study: (top) Freehand drawings of objects without using ShadowDraw, (bottom) freehand drawings of objects using ShadowDraw. Notice the improved spacing and proportions while maintaining the subjects' own unique styles.*

Abstract

We present ShadowDraw, a system for guiding the freeform drawing of objects. As the user draws, ShadowDraw dynamically updates a *shadow image* underlying the user's strokes. The shadows are suggestive of object contours that guide the user as they continue drawing. This paradigm is similar to tracing, with two major differences. First, we do not provide a single image from which the user can trace; rather ShadowDraw automatically blends relevant images from a large database to construct the shadows. Second, the system dynamically adapts to the user's drawings in real-time and produces suggestions accordingly. ShadowDraw works by efficiently matching local edge patches between the query, constructed from the current drawing, and a database of images. A hashing technique enforces both local and global similarity and provides sufficient speed for interactive feedback. Shadows are created by aggregating the edge maps from the best database matches, spatially weighted by their match scores. We test our approach with human subjects and show comparisons between the drawings that were produced with and without the system. The results show that our system produces more realistically proportioned line drawings.

CR Categories: I.3.8 [Computing Methodologies]: Computer Graphics—Applications;

Keywords: large scale image retrieval, shape matching, interactive drawing

Links:

ACM Reference Format
Lee, Y., Zitnick, C., Cohen, M. 2011. Shadow Draw: Real-Time User Guidance for Freehand Drawing. *ACM Trans. Graph. 30,* 4, Article 27 (July 2011), 9 pages. DOI = 10.1145/1964921.1964922 http://doi.acm.org/10.1145/1964921.1964922.

1 Introduction

If asked to draw a face, the result for most of us (those with little practice in drawing) might look like one of those in the upper row of Figure 1, created by subjects in our user study using a standard drawing interface. Similarly, if asked to draw a bicycle, most of us would have a difficult time depicting how the frame and wheels relate to each other. One solution is to search for an image of the thing we want to draw, and to either trace it or to use it in some other way as a reference. However, aside from the difficulty of finding a photo of what we want to draw, simply tracing object edges eliminates much of the essence of drawing, i.e., there is very little freedom in tracing strokes. Conversely, drawing on a blank paper with only the image in the mind's eye gives the drawer a lot of freedom, but freehand drawing can be frustrating without significant training. To address this, we present ShadowDraw, a drawing interface that automatically infers what you are drawing and then dynamically depicts relevant *shadows* (Figures 5 and 6) underneath the drawing. These shadows may be either used or ignored by the drawer.

ShadowDraw preserves the essence of drawing, i.e., freedom and expressiveness, and at the same time uses visual references, *shadows*, to guide the drawer. Furthermore, shadows from real images can enlighten the artist with the gist of many images simultaneously. The creation becomes a mix of both human intuition and computer intelligence. The computer, in essence, is a partner in the drawing process, providing guidance like a teacher, instead of actually producing the final artwork. The drawings in the bottom row of Figure 1 were drawn by the same subjects, this time using ShadowDraw. Notice how the users' own creative styles remain consistent between the drawings, while the overall shapes and spacing are more realistic.

ShadowDraw consists of two main computational steps plus the user interface. The first offline step consists of building a database from a collection of 30,000 images collected from the Web. Each image is converted to an edge drawing using the long edge detector technique developed by [Bhat et al. 2009] and stored. Overlapping windows in each edge image are analyzed, coded, and stored. Each window is converted to edge descriptors, and further coded as *sketches* with distinct hash keys using *min-hash* [Chum et al. 2008]. In the second online step, as the user draws, ShadowDraw

analyzes the strokes using a similar encoding to determine hash keys for overlapping windows for fast matching with the database of images. The top 100 matching database edge images are further aligned to the drawing. A set of spatially varying weights blend the edge images into a *shadow image*. In the user interface, the strokes are overlaid on top of an evolving shadow image that provides guidance for future strokes.

Our main contribution is an interactive drawing system that dynamically adapts to the user's drawing and provides real-time feedback. A number of technical contributions make ShadowDraw unique. Although portions of ShadowDraw follow the basic framework of content based image retrieval, the technique of partial spatial matching that ShadowDraw employs is novel, in that it allows for multiple matching images based on different sub-regions of the image. In addition, the verification stage and methods for determining the blending weights are unique to this work. While there have been previous works that helps users draw basic shapes [Igarashi et al. 1999; Arvo and Novins 2000; Igarashi and Hughes 2001], to our knowledge, we are the first to develop an interactive user interface to assist freeform drawing. We test our approach with human subjects and show comparisons between the drawings that were produced with and without the system. The results show that our system produces more realistically proportioned line drawings, particularly for those who possess some skill but lack expertise. We purposely avoid in this paper, making any claims that the system helps produce more skilled drawers or more artistic drawings.

2 Related Work

The huge volume of image and video data available on the Web, scientific databases, and newspaper archives, along with recent advances in efficient (approximate) nearest neighbor matching schemes have opened the door for a number of large scale matching applications. The general field of content based image retrieval (CBIR) uses many different inputs modalities to search for similar images in a database (see [Datta et al. 2008] for a general survey of this field). Here, we briefly review related work in large scale image retrieval techniques, especially those that use line drawings for the query and/or are aimed at constructing new images and drawings.

There are a number of systems that produce photographic-like results by compositing portions of retrieved images. The Sketch-to-Collage system [Gavilan et al. 2007] produces a single composite collage by matching user provided color strokes to a database of images, segmenting out regions and interactively blending the retrieved segments. The Sketch2Photo system [Chen et al. 2009] produces a composite image from the user's sketch of a scene with text label annotated objects. Candidate image regions for each object are found on the Web and those that produce the best agreement are put together to form the final composition. PhotoSketch [Eitz et al. 2009] progressively creates images through a sketching and compositing interface. The user interacts with the system to segment and blend the images retrieved from a database of 1.5 million images. Rather than starting from a blank page, the Scene Completion algorithm [Hays and Efros 2007] performs a global scene match using a query image, which has "holes". The system finds the best images for completing the scene with objects that could have been in the missing regions. To improve retrieval accuracy of these sketch-based systems, researchers have also designed descriptors that provide better matches between human drawn sketches and natural images [Chalechale et al. 2005; Hu et al. 2010].

More general CBIR efforts include the early SIGGRAPH work, Fast Multiresolution Image Querying [Jacobs et al. 1995], that matches user paint strokes to underlying wavelet signatures of images in the database. The "Blobworld" approach [Carson et al. 2002] queries image regions rather than the entire image, to allow the user to specify which objects in the image are more relevant to the query. In [Nister and Stewenius 2006], a vocabulary tree is created to efficiently retrieve images from a large database. The tree defines a hierarchical quantization of the local image features and provides a multi-level scheme to score matching images. In [Chum et al. 2007], the idea of "query expansion" in text retrieval is applied to the image domain, where the highest ranked images from the original query are re-queried to generate additional relevant images.

A 3D photorealistic virtual space is created in [Sivic et al. 2008] to allow users to tour themes, such as city streets or skylines. Similar images are matched and stitched from a large (few hundred thousand) image collection downloaded from Flickr. Most recently, MindFinder [Cao et al. 2010] aims to improve image retrieval by allowing the user to input a text tag, a sketch, and a color as a query. The system is able to retrieve images that better match the image in the user's mind. Finally, in [Chaudhuri and Koltun 2010], data-driven suggestions are made for 3D modeling. The system presents suggestions by matching and retrieving relevant shapes in the database to the initial basic model.

ShadowDraw also leverages the idea of matching to a large database of images. Unlike previous methods, our end goal is to help the user draw rather than to perform an image composition, completion, retrieval, or 3D modeling. Furthermore, ShadowDraw uses only a partial and evolving drawing for the query rather than other images and/or textual descriptions. Our system requires the retrieval to run in real time. We have seen no other system that leverages image retrieval for the same kind of application or with partial drawings as the input.

There are interactive drawing interfaces such as Teddy [Igarashi et al. 1999], Fluid Sketches [Arvo and Novins 2000], and the 3D drawing system of [Igarashi and Hughes 2001] that strive to produce better drawings for the user. These methods provide low-level information feedback in the form of basic polygonal shapes, lines, and curves. More recently, the iCanDraw interface [Dixon et al. 2010] provides step-by-step instructions and corrective feedback to guide a user to draw a human face from a reference image. While similar in motivation, our approach provides guidance for drawing arbitrary high-level objects using only example images.

Recently, in [Cole et al. 2008], the authors studied artists' line drawings of 3D shapes to analyze which line segments were being emphasized, and to compute correlations between those segments and the contours produced using existing computer generated line drawing techniques and contour feature extractors. While our work aims to help users in freeform drawing, the outputs of our system can be a useful resource for this line of work, i.e., the user drawings produced with ShadowDraw can be used for analyzing the contours of more general objects occurring in natural images.

3 Approach

ShadowDraw includes three main components: (1) the construction of an inverted file structure [Witten et al. 1999] that indexes a database of images and their edge maps; (2) a query method that, given user strokes, dynamically retrieves matching images, aligns them to the evolving drawing and weights them based on a matching score; and (3) the user interface, which displays a shadow of weighted edge maps beneath the user's drawing to help guide the drawing process.

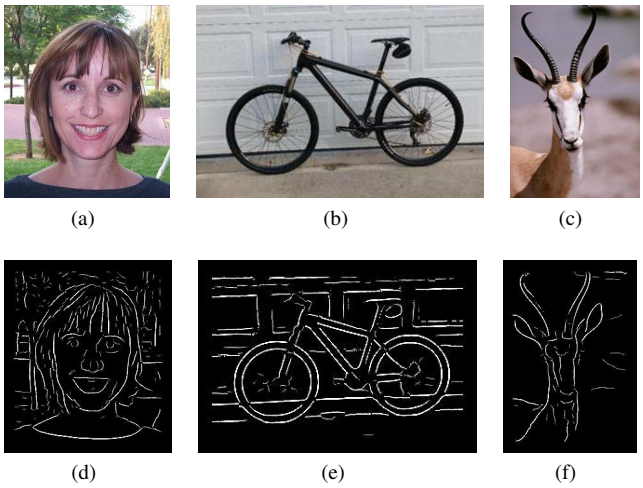

Figure 2: *Example database images and their corresponding edge images.*

3.1 Database Creation

The images in the database should be selected so that the objects depicted, as well as their appearance and pose, are likely to be similar to those drawn by the user. Ideally, a large database of hand drawn images would be used, but such a database is exceedingly hard to construct. Instead, we use a set of approximately 30,000 natural images collected from the internet via approximately 40 categorical queries such as "t-shirt", "bicycle", "car", etc. Although such images have many extraneous backgrounds, objects, framing lines, etc., the expectation is that, on average, they will contain edges a user may want to draw. We scale the images to fit a 300x300 pixel resolution, i.e., the long side is scaled to 300 pixels. We then process each image in three stages and add them to an inverted file structure. First, edges are extracted from the image. Next, local edge descriptors are computed. Finally, sets of concatenated hashes called *sketches* are computed from the edge descriptors and added to the database. We store the database as an inverted file, in other words, indexed by sketch value, which in turn points to the original image and its edges.

Edge extraction Given a natural image, we want to find the edges most likely to be drawn by a user while ignoring others. Perceptual studies show that long coherent edges are salient to humans even if faint [Beaudot and Mullen 2003; Elder and Goldberg 2001]. Inspired by these works, we use the long edge detector described in [Bhat et al. 2009]. The method locally normalizes the magnitudes of the edges, and then sums the normalized magnitudes, weighted by local curvature, along the length of the edge. The result is an edge response that is related to the length of the edge and its degree of curvature, rather than the magnitude of the intensity gradients. We store the edge images, E, using run length encoding. On average, each compressed edge image requires 5.3KB. Examples are shown in Figure 2 (d-f).

Patch descriptors For each edge image, we determine the edge positions by finding maxima in the responses perpendicular to the edge direction, similar to Canny edge detection [Canny 1986]. Given an image I in the database with corresponding edges E and orientations θ, we compute a set of edge descriptors $d_i \in D$. Since the goal is to match an edge image E to incomplete and evolving drawings, we compute the descriptors locally over 60x60 patches. The patches used to compute neighboring descriptors overlap by 50%, resulting in 81 descriptors over the 9×9 fixed grid of patches.

We encode each patch using the BiCE descriptor [Zitnick 2010], which is designed to encode the histogram of edge positions and orientations. Since the edges drawn by a user are typically less dense than in natural images, we use a low dimensional version of BiCE. We define a three dimensional histogram, with 4 discrete edge orientations, 18 positions perpendicular to the edge, and 6 positions tangent to the edge. Using the notation of [Zitnick 2010], we set $n_{x'} = 18$, $n_{y'} = 6$, $n_\theta = 4$, and $n_l = 1$. The buckets of the histogram are binarized by setting the top 20% to one and the rest to zero. The final descriptor, d_i, has 432 binary bits.

Other image patch descriptors, such as SIFT [Lowe 2004] and Daisy [Winder et al. 2009], rely on relative strength of the edge magnitudes to provide discriminability, and are shown to have reduced matching performance compared to BiCE in [Zitnick 2010]. They are also not applicable to our scenario, since the relative edge magnitudes on which these descriptors rely are not known for the user's drawing.

Min-hash A key feature of the BiCE descriptor is its binary encoding: it can be viewed as a set representation where the ones indicate edge presence. This makes it amenable to min-hash, which is an effective hashing technique for retrieval and clustering [Chum et al. 2008; Lee et al. 2010; Zitnick 2010]. Min-hash has the property that the probability of two sets having the same hash value (i.e., "colliding") is equal to their Jaccard similarity. The Jaccard similarity $sim(d_i, d_j)$ between two sets, d_i and d_j, is the cardinality of their intersection divided by the cardinality of their union:

$$sim(d_i, d_j) = \frac{\#(d_i \cap d_j)}{\#(d_i \cup d_j)}. \quad (1)$$

A min-hash function randomly permutes the set indices (i.e., the ones and zeros). All sets (BiCE descriptors) are permuted using the same min-hash function. The min-hash value for a given permuted set is its smallest index containing a value of one after the permutation. A single min-hash can be non-discriminative and introduce many false positive retrievals, especially if the descriptor is non-sparse. To increase precision, we compute k independent random min-hash functions, and apply them to all BiCE descriptors. We concatenate the resulting k min-hash values for each descriptor into k-tuples called *sketches*. The probability of two sketches colliding is thus reduced exponentially to $sim(d_i, d_j)^k$. To increase recall, this process is repeated n times using n different sets of k min-hash functions, resulting in n sketches per descriptor. To maintain high recall while reducing false positives, tradeoffs must be made between the number of sketches stored for each descriptor and the size of the sketch, k. In our experiments, we store $n = 20$ sketches of size $k = 3$ for each descriptor.

We store an inverted file structure for each of the n min-hash sketches. We allocate each unique sketch as a new entry in the structure. We record the image index and patch location of the descriptor instance that produced the sketch.

3.2 Image Matching

Our hashing scheme allows for efficient image queries, since only images with matching sketches need to be considered. In this section, we describe the real-time matching pipeline between the edge images in the database and the user's drawing, as shown in Figure 3. Initially, we use the inverted file structure to obtain a set of candidate matches. Next, each candidate match is aligned with

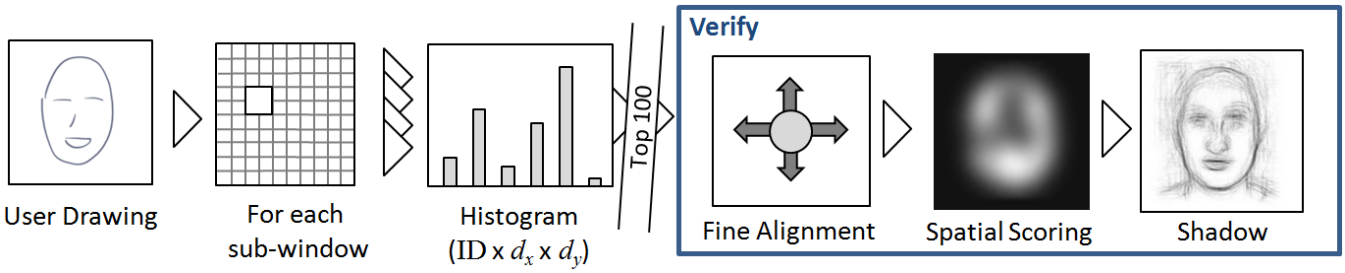

Figure 3: *An outline of the online processing pipeline. Given a user's strokes, sketches are computed for each sub-window, and matching votes are accumulated in a histogram. The top 100 matching images are aligned, scored, and weighted to generate the final shadow image.*

the user's drawing and scored. This two step matching procedure is necessary for computational efficiency, since only a small subset of the database images need to be finely aligned and weighted. We use the scores from the alignment step to compute a set of spatially varying weights for each edge image. The output is a shadow image resulting from the weighted average of the edge images. Finally, we display the shadow image to the user as described in Section 4.

Candidate matches We represent the user's drawing as a set of vectorized multi-segment strokes. We create an edge image $\tilde{E}$ from these strokes by drawing lines with a width of one pixel between the stroke points. The rendered lines have the same style as the edges extracted from the natural images in the database, i.e., the edge image $\tilde{E}$ used for matching does not use the stylized strokes that are seen by the user described in Section 4. Next, we compute BiCE descriptors and their corresponding sketches in the same manner as described in Section 3.1, this time using a higher resolution grid of $18 \times 18 = 324$ patches with 75% overlap between neighboring patches. We use a higher resolution grid to increase the accuracy of the predicted position, and to increase invariance to translations in the drawing. In our implementation, the user's drawing occupies an area of 480×480 pixels, resulting in 96×96 pixel patches with 24 pixel offsets between neighboring patches. We compute descriptors and sketches for each of the 324 patches.

Using the inverse lookup table, we match each sketch from the user's drawing to the sketches stored in the database. A matching sketch casts one vote for the corresponding database image and patch offset pair. We aggregate the matches in a histogram H storing the number of matching sketches for each image at each grid offset. To reduce the size of H, votes are only stored if the database patch offset is within four patch grid points of the patch being considered in the user's drawing. This corresponds to relative shifts of less than 96 pixels between the user's drawing and the database images. The resulting histogram has size $m \times 9 \times 9$, where m is the number of images in the database. After adding all the matches for each sketch to the histogram, we find the best matching offset for each image, and add the top 100 images to the candidate set C. As discussed in Section 3.1, we compute $n = 20$ sketches for each descriptor, resulting in a maximum possible 20 votes per sketch in the histogram. To reduce the bias from any single descriptor, we limit the contribution of each descriptor to four votes in the histogram.

Given a large database, computing the candidate set as described above can be computationally expensive. We take advantage of the fact that the user's strokes change gradually over time to increase performance. At each time step, only votes resulting from sketches derived from patches that have changed are updated. We accomplish this by subtracting the votes added from the previous sketches from H, followed by adding in the votes from the new sketches. At each time frame, we also include in the candidate set any database image that contributed to the shadow image in the previous time frame.

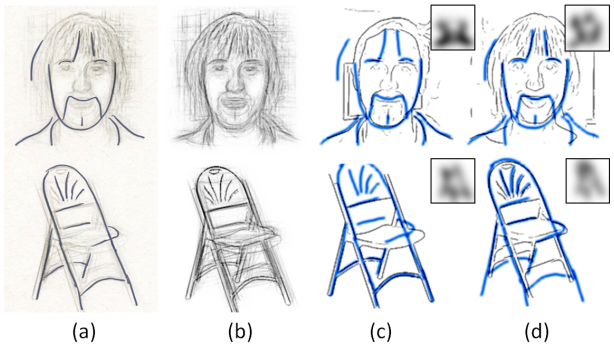

Figure 4: *Illustration of spatial weights: (a) user's view, (b) shadow image, (c, d) top two matches and corresponding spatial weights (top right).*

Image alignment The candidate image set C contains a set of images with approximate offsets d_x and d_y defined by the best matching offset as described above. The approximation arises from the discretization of the offsets in the grid of patches. We refine these offsets using a 1D variation of the Generalized Hough transform [Ballard 1981]. Using the standard Generalized Hough transform for 2D translations, we create a 2D histogram T over possible offsets x and y using:

$$T(x,y) = \sum_p \tilde{E}(p_x, p_y) E(p_x + d_x + x, p_y + d_y + y), \quad (2)$$

where $\tilde{E}(p_x, p_y)$ is the value of $\tilde{E}$ at pixel p in location (p_x, p_y), and similarly for the edge image E. We determine the best offset by finding the maximum value of $T(x,y)$. This approach is computationally expensive since we need to sum over the image for every possible combination of x and y offsets. Instead we compute the x and y dimensions separately using two histograms:

$$T_x(x) = \sum_p \sin(\tilde{\theta}(p_x, p_y)) \tilde{E}(p_x, p_y) \quad (3)$$
$$\sin(\theta(p_x + d_x + x, p_y + d_y)) E(p_x + d_x + x, p_y + d_y),$$

and similarly for T_y using the cosine of the angles. The sine of the edge angles provides higher weights to the more informative vertical edges when determining the horizontal offsets, and similarly for T_y and cosine with horizontal edges. We empirically found this approach to produce good results. Once the histograms T_x and T_y are created, they are slightly blurred with $\sigma_h = 2$. We determine

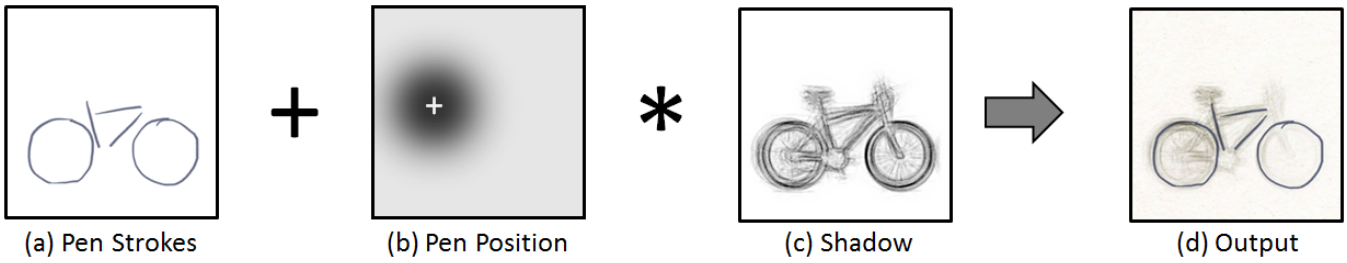

Figure 5: *Rendering pipeline for user interface: (a) user's rendered strokes, (b) weighting based on pen position, (c) shadow image, and (d) final rendering.*

the final sub-pixel offsets d'_x and d'_y by adding a quadratic interpolation of the resulting peak response in T_x and T_y to d_x and d_y. For additional accuracy, two iterations are run. To reduce computation, we limit the search range of x and y to twice the distance between the grid points. In addition, we compute Equation (3) on reduced resolution images of size 160×160. We notate the aligned edge image as E'_i. To refine the scale, an additional 1D histogram T_s may be similarly computed across scales and the peak found.

Image weighting We now have a set of candidate images, C, and their aligned edge images E'_i, aligned using offsets d'_i. Our goal is to blend these aligned edge images into a *shadow* image, S, that will help guide the user as they draw:

$$S = \sum_i W_i E'_i, \tag{4}$$

where W_i is the blending weight image, which we define below. The blending weight should be high for pixels where there is a good match between the drawing and the candidate's aligned edges and low for pixels where there is not. We construct the weight image from two terms, a global match term v_i and a spatially varying match term, V_i, which are normalized over all images in the candidate set:

$$W_i = \alpha \frac{v_i V_i}{\sum_j v_j V_j}. \tag{5}$$

The weight α is used to reduce the visibility of noisy shadows produced when the drawing has just started and all the match scores are low. An illustration of the spatial weighting is shown in Figure 4. The use of a spatial weighting results in shadows that are a composite of multiple distinct edge images, creating the appearance of an object that does not exist in a single database image.

We begin by defining the spatially varying match term V_i followed by the global match term v_i and α. Our goal is for a candidate image's weights to increase when its edges agree in position and orientation with the user's strokes. To compute V_i, we first decompose each candidate edge image into eight oriented images, ϑ_t, and similarly, the drawing image into eight oriented images, $\tilde{\vartheta}_t$, for $t = 1, \ldots, 8$. Each image captures only strokes parallel to one of eight evenly spaced orientations; i.e., ϑ_1 depicts horizontal edges, ϑ_5 depicts vertical edges, and the remaining six each capture one other orientation at 22.5° intervals. If an edge's orientation falls in between two orientations, its contribution is linearly divided between the two oriented edge images.

To provide some invariance to position, we blur the oriented edge images with a Gaussian kernel $\mathcal{G}(\sigma_s)$ with a standard deviation $\sigma_s = 1.25\varphi$, where φ is the relative distance between the grid points. It is also desirable that images that contain multiple edges near a stroke receive the same score contribution as those with a single edge. This is accomplished by limiting the magnitude of the response from the blurred edges to the maximum response that may result from a single edge in isolation.

We then compute positive and negative edge correlation images, ϑ^+ and ϑ^-, to determine where the edge image agrees and disagrees with the user's strokes. We define positive correlation using the product of the edge images with the same orientations, and define negative correlation using the product of orthogonally oriented edge images:

$$\vartheta^+ = \sum_{t=1,8} \vartheta_t * \tilde{\vartheta}_t \tag{6}$$

$$\vartheta^- = \sum_{t=1,8} \vartheta_t * \tilde{\vartheta}_{(t+4)\%8} \tag{7}$$

Our spatially varying match term V_i is simply a Gaussian blurred version of ϑ^+,

$$V_i = \mathcal{G}(\vartheta_i^+, 4\varphi) \tag{8}$$

We add a small offset to V_i to ensure non-zero values. To reduce computation, we compute the image scores and weights on reduced resolution images of 40×40 pixels.

Next, we define the global match term v_i in Equation (5) that is used to compute the edge image's blending weights W_i. To aid in the computation of v_i, we compute a global match score h_i for each image using the difference between ϑ^+ and ϑ^-,

$$h_i = \sum_p \vartheta_i^+(p) - \vartheta_i^-(p) \tag{9}$$

Equation (9) has a high positive value when the user's strokes and the image's edges agree in position and orientation. However, if a majority of the user's strokes are perpendicular to the image's edges, h_i may be negative.

Finally, we compute v_i using a nonlinear function of h_i and the average h^* of the five highest scores from the candidate set,

$$v_i = \max\left(0, \left(\frac{h_i - \gamma h^*}{h^* - \gamma h^*}\right)^\kappa\right). \tag{10}$$

We assign a value of 0.5 to γ, which means the value of v_i is greater than zero only if the score is greater than half the average of the five highest scores. A value of $\kappa = 2$ favors images with higher scores and sets the rate of weight decay as quadratic. In Equation (5), we set α to

$$\alpha = \frac{\sum_i v_i}{\epsilon + \sum_i v_i} \tag{11}$$

where ϵ to corresponds to the score that would result from drawing a single stroke of approximately 250 pixels. Using Equation (11), α increases as the user draws more stokes, resulting in greater visibility of the shadows.

Figure 6: *Video frames from four example drawing sessions: (top) final rendering visible to user, (bottom) shadow image. The last row demonstrates the shadow's robustness to clutter.*

To summarize, we compute a shadow image in real time by comparing the user drawn strokes to the candidate images extracted from the database. We compute global and spatially varying weights to blend the corresponding edge images to create the shadow image used in the drawing interface. We determine these weights by comparing local orientations of edges between the developing drawing and the database edge images.

4 User Interface

The ShadowDraw user interface appears at first like a standard drawing system on a blank paper-like surface. We used a WACOM Cintiq 21ux screen/tablet. The user can draw or erase strokes using a stylus. The user sees their own drawing formed with pen strokes superimposed on a continuously updated shadow S, see Figure 5. The drawing area is 480×480 pixels, and the line strokes are rendered using a dark blue marker style, as shown in Figure 5(a). To provide some color contrast, we render the shadow in a sepia tone.

To further make the shadow visible to the user, while not distracting from the user's actual drawing, we filter the shadow image to remove noisy and faint edges,

$$\tilde{S} = S * (\mathcal{G}(S, \sigma_r) - \varepsilon) \tag{12}$$

where $\sigma_r = 16$. Multiplying by a blurred shadow image strengthens edges that agree in position and weakens others. $\varepsilon = 1$ is used to additionally suppress faint edges. In addition, we apply a small amount of blur with a standard deviation of 1.25 to soften the shadows, see Figure 5(c). Finally, we weight the shadow S' to have higher contrast near the user's cursor position p_c:

$$S'(p) = \lambda \tilde{S}(p) + (1 - \lambda)\omega \tilde{S}(p) \tag{13}$$

where ω is a Gaussian weighted distance between p and p_c with a standard deviation of 120 pixels. λ may be set by the user and controls the contrast of the shadows. We render the final image on a paper textured background as shown in Figure 5(d).

5 Results

All our experiments are run using a database of approximately 30,000 images. We collected 28,000 images by crawling image web searches such as "bear", "bike white background", and "motorcycle honda". We removed duplicates and adjusted the resolution of the images such that intra-category objects have similar scales. We collected 435 face images from the Caltech 101 dataset [Fei-Fei et al. 2004]. Since the face images are of uniform scale, and users tend to draw faces at different scales, (full face vs. head and shoulders) we added randomly scaled images to the dataset for a total of 2,000 face images.

We implemented ShadowDraw on a modest system with a quad-core Intel i5 CPU with 4GB RAM. The compressed edge images from the database are stored in memory along with the inverse look-up table requiring 850MB of RAM. We implemented ShadowDraw using two threads to provide a smooth user interface. The foreground thread handles user input and renders the user's strokes. The background thread accepts as input the user's strokes and computes a shadow image. On average, a new shadow image is computed every 0.4 to 0.9 seconds depending on the number of new strokes. A fast response is critical in creating a positive feedback loop in which the user obtains suggestions while still in the process of drawing a stroke.

We chose the parameters for the BiCE descriptor, sketch sizes, number of sketches, and grid resolutions through quantitative tests on several categories, such as faces, motorcycles, and butterflies. We chose the parameters associated with fine alignment and spatial weighing by qualitatively examining several results.

Figure 6 shows several example drawing sessions using ShadowDraw. Please refer to the accompanying video to see the sessions in action and for more results. Note the drawing rate is accelerated to fit in the video. As the user draws new strokes and erases others, the shadows dynamically update to best match the user's drawing.

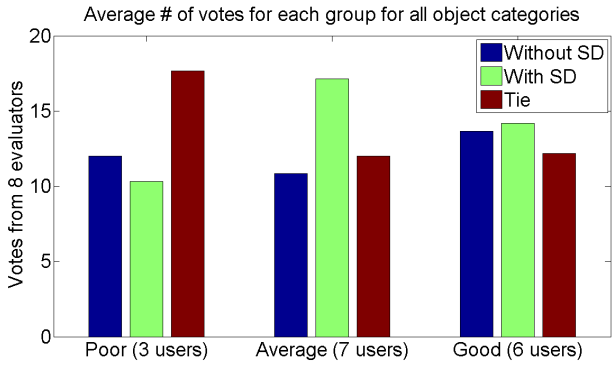

Figure 7: *Evaluation scores of drawings by users in the "poor", "average", and "good" groups. ShadowDraw achieves significantly better drawing results for the "average" group.*

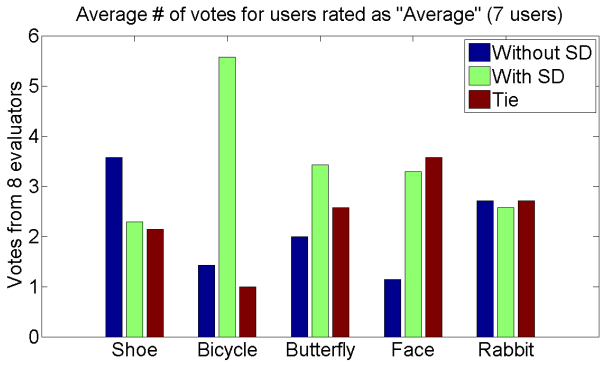

Figure 8: *Per category evaluation scores of drawings by users in the "average" group. ShadowDraw improves drawing results for most categories.*

Using our large database, the user can receive guidance for a variety of object categories, including specific types of objects such as office chairs, folding chairs, or rocking chairs. The last row of Figure 6 demonstrates the scoring function's robustness to clutter. Even when many spurious strokes are drawn, the correct images are given high weight in the shadow image.

5.1 User Studies

We conducted a user study to assess the effectiveness of ShadowDraw on untrained drawers. In the first stage, subjects produced quick 1-3 minute drawings with and without ShadowDraw. In a second stage, a separate set of subjects evaluated the drawings.

Drawing We conducted the experiment with 16 subjects, eight women and eight men. Each subject was asked to draw five objects, *Shoe, Bicycle, Butterfly, Face,* and *Rabbit*, with and without ShadowDraw, for a total of 10 drawings. We randomly permuted the sequence of objects presented to each user. The shadows appeared on every other drawing; thus half the objects were drawn first without shadows and the other half drawn first with shadows. We used *Rabbit* as a control variable to ensure fairness of the experiments: there were no rabbit images in our database. Please see the supplementary video for example drawings by the subjects.

Before beginning the study, we explained the UI: the functionality of the pen (drawing and erasing), the white canvas, and the "start" and "next" button to proceed to the next object once the current drawing was completed. We explained to the user that shadows would appear on every other drawing, and that he/she was free to use them for guidance or to completely ignore them. We also explained that the shadows could be removed by tapping on a region outside of the canvas (and would reappear as the user started drawing again). Warm-up exercises to acquaint them with the interface included drawing a circle and a t-shirt.

Each user was given 30 minutes to complete all 10 drawings. We recorded the sequence of keystrokes (both drawing and erasing), and time spent on each drawing. On average, the users completed the task in about 20 minutes, spending more time on objects which require more detail such as *Faces* and less on those such as *Shoes*. We also asked subjects to fill out a short questionnaire at the end of the study.

Evaluation Eight additional subjects (who did not participate in the drawing experiment) evaluated the drawings. We displayed each drawing pair (produced with and without ShadowDraw) side-by-side. We asked each evaluator to choose what she perceived to be the "better drawing". If she could not decide, she was given the option of choosing "tie". This was repeated for all drawing pairs. We randomized the order and placement of the drawings, so that the drawings produced with ShadowDraw did not always appear left or right.

To assess the user's drawing abilities, we next asked each evaluator to rate each user's collection of drawings produced without ShadowDraw on a scale of 1 to 5, where 1 is poor, 3 is average, and 5 is excellent. We grouped the users into three groups (poor, average, and good) based on their drawing ability score, by averaging the eight evaluator ratings. It should be noted that none of the subjects was rated as close to excellent, as all ratings averaged below 4.

Analysis and Remarks We first present our findings on the drawing scores per group for all object categories. Figure 7 shows the results. We compute an average score per group (i.e., by averaging the individual user votes in the group). The "poor" group has three users, with scores in $[1, 2)$; the "average" group has seven users, with scores in $[2, 3)$; and the "good" group has six users, with scores above 3.

Overall, ShadowDraw achieves significantly higher scored drawing results for the "average" group, and inconclusive results in the other two groups (see Figure 7). Figure 8 shows the per category breakdown of scores for the "average" group. ShadowDraw was particularly helpful for drawing structurally complex objects like bicycles. We see noticeable improvement for faces and butterflies as well. The rabbit category was used as a control variable, so the "tie" result was expected. Shoes showed some decline with ShadowDraw perhaps because the variability in shoe appearance produced a higher amount of noise in the shadow image.

The lack of higher scores in the "good" group confirms our intuition that people who can draw quite well will produce equally good drawings with or without ShadowDraw. The interesting phenomenon is the insignificant difference in the scores for the "poor" group. Upon closer inspection of their drawings, we found that these users were extremely poor drawers (i.e., the aspect ratios and basic shapes of their drawings were far off from those of the objects they were intending to draw) and thus the system had no chance to properly match and retrieve relevant database images. This also explains why we achieve significant improvement over the baseline for the "average" group. The users in this group are able to draw the basic shapes and rough proportions of the objects correctly, but

Figure 9: *Example face drawings produced without ShadowDraw and after training with ShadowDraw. Each pair was drawn by the same subject.*

have difficulty applying exact proportions and details essential for producing compelling drawings, which is precisely where ShadowDraw can help.

There are several remedies to allow even the poorest drawer to benefit from our system. The most obvious is to give them more practice to learn the capabilities of the system. To test this hypothesis, we asked each user to draw another face after they completed the previous task. We allowed the user to explore and suggested that the user draw a more "oval" vs. "round" face outline to get more relevant shadows. Figure 9 shows some examples of the users' drawings of faces before and after such practice. There is a noticeable change towards realistic proportions in the drawings for those with poor skill (left) and good skill (right). Notice how the subject's personal style is maintained between drawings, and that the more proficient drawers are not simply tracing the shadows. Although most would agree the poorer results have been improved, it becomes a matter of taste for the more skilled drawings whether the new drawings are *better*. Truly assessing the overall *aesthetic improvement* in the results is beyond the scope of this paper.

Figure 1 shows some more examples of the users' drawings with and without ShadowDraw. Each column shows the drawings produced by the same user. One can clearly notice a significant change towards more realism in the drawings, especially in terms of the proportions of the different parts of an object and including important features such as the structure and layout of the object (see the bicycles), and overall shape (see the butterflies).

User Satisfaction When asked in the questionnaire, *"How would you compare your drawing results with ShadowDraw vs. those without ShadowDraw?,"* on average, the users gave a score of 4.0 in a range where 1 is "much worse", 3 is "no difference", and 5 is "much better". When asked, *"How would you rate your satisfaction of drawing with ShadowDraw vs. without ShadowDraw?,"* on average, the users gave a score of 3.9. Some positive comments from an open ended question included:

- "Fun and helpful, I became dependent on it very quickly."
- "Helps in drawing faster than without ShadowDraw"
- "This is a great product and I already love it - got to have one. Really helps me relax!!"
- "Having no background in art, ShadowDraw made drawing a lot of fun when a shadow was provided."

There were a couple of comments indicating that ShadowDraw was sometimes distracting, such as "...I occasionally got confused about my lines vs. shadow lines." For this, we can create a button on the UI labeled as "Shadow on/off", so that the user can choose to view or hide the shadows.

Overall, the users appeared to enjoy using ShadowDraw and produced better drawings (as self-rated) than they could achieve without ShadowDraw. This is the essence of what defines ShadowDraw. It does not produce the final artwork; rather, it guides the user when the user wants the help. This makes the drawing experience fun, and with that, the final drawing becomes visually pleasing to one's self.

6 Conclusion and Discussion

We have presented a system that guides a user to draw by dynamically producing shadows derived from thousands of images. We have demonstrated that our method can retrieve relevant images in real time based on incomplete and evolving sketches by the user.

We reported on a user study that showed improved realism in many users' drawings, but more importantly, we learned a number of unexpected things from the study. Perhaps the most interesting finding was that ShadowDraw was most effective for those that had a modicum of drawing skill. We can surmise that those with little skill were unable to produce initial drawings sufficient for the retrieval of relevant images for the production of shadows. Those already possessing drawing skill may have been distracted by the shadows. There are a number of ways we hope to alleviate both issues to address the needs of a wider range of users. For more novice drawers, being able to specify the category class would trim the database search and would likely lead to more relevant shadows even when the initial drawing has little resemblance to the class. For more expert users, an interface providing more control over shadow strength can be easily added. Also, an ability to draw *negative* strokes to discourage shadows containing those strokes would again provide finer control over the shadow guidance.

We also want to explore the trade-offs between image database size and the shadows' efficacy. We have shown the ability to handle many thousands of images over a wide variety of categories. While more flexibility would be attained by a larger database for experts, the problems encountered by the novice drawers might be accentuated. More work is required to understand these trade-offs.

There are also many more subtle issues such as the tension between guidance and freedom in how a drawing expresses an object's essence versus getting all the proportions right. This will perhaps require exploring new scoring methods that allow for non-rigid transformations in the matching and shadow generation steps. We were encouraged to see that the users' personal drawing styles were maintained when using ShadowDraw. That said, more work is required to make the system truly aid in improving the overall aesthetic result. This will require a more nuanced understanding of the relationship between aesthetics and realism in drawings. If one could assemble a large database of artistic drawings, we may be able to use some of the technology we report to begin to learn such a relationship, but this is left for future work.

In conclusion, we have found this area of investigation very exciting and look forward to extending the work in many directions. We hope this paper has expressed both the scope of the specific work reported on, as well as the many possible related avenues for future exploration.

Acknowledgements

The authors would like to thank Ce Liu for many insightful discussions and help in shaping the work in this paper. We would also like to thank the anonymous reviewers for their many helpful comments. Finally, thanks to all the participants who were willing to draw as part of the user studies. Hopefully ShadowDraw will make them feel more comfortable picking up a pen to draw in the future.

References

ARVO, J., AND NOVINS, K. 2000. Fluid sketches: Continuous recognition and morphing of simple hand-drawn shapes. *ACM UIST*.

BALLARD, D. 1981. Generalizing the hough transform to detect arbitray shapes. In *Pattern Recognition*, vol. 13, 111–122.

BEAUDOT, W., AND MULLEN, K. 2003. How long range is contour integration in human color vision. In *Visual Neuroscience*, vol. 15, 51–64.

BHAT, P., ZITNICK, C. L., COHEN, M., AND CURLESS, B. 2009. Gradientshop: A gradient-domain optimization framework for image and video filtering. *TOG*.

CANNY, J. 1986. A computational approach to edge detection. In *TPAMI*, vol. 8, 679–698.

CAO, Y., WANG, H., WANG, C., LI, Z., ZHANG, L., AND ZHANG, L. 2010. Mindfinder: Finding images by sketching. In *ACM Multimedia International Conference*.

CARSON, C., BELONGIE, S., GREENSPAN, H., AND MALIK, J. 2002. Blobworld: Image segmentation using expectation-maximization and its application to image querying. In *TPAMI*, vol. 24, 1026–1038.

CHALECHALE, A., NAGHDY, G., AND MERTINS, A. 2005. Sketch-based image matching using angular partitioning. *IEEE Trans. Systems, Man, and Cybernetics*.

CHAUDHURI, S., AND KOLTUN, V. 2010. Data-driven suggestions for creativity support in 3d modeling. *ACM SIGGRAPH ASIA*.

CHEN, T., CHENG, M.-M., TAN, P., SHAMIR, A., AND HU, S.-M. 2009. Sketch2photo: Internet image montage. *ACM SIGGRAPH ASIA*.

CHUM, O., PHILBIN, J., SIVIC, J., ISARD, M., AND ZISSERMAN, A. 2007. Total recall: Automatic query expansion with a generative feature model for object retrieval. In *CVPR*.

CHUM, O., PHILBIN, J., AND ZISSERMAN, A. 2008. Near duplicate image detection: min-hash and tf-idf weighting. In *BMVC*.

COLE, F., GOLOVINSKIY, A., LIMPAECHER, A., BARROS, H. S., FINKELSTEIN, A., FUNKHOUSER, T., AND RUSINKIEWICZ, S. 2008. Where do people draw lines? *SIGGRAPH*.

DATTA, R., JOSHI, D., LI, J., AND WANG, J. Z. 2008. Image retrieval: Ideas, influences, and trends of the new age. In *ACM Computing Surveys*, vol. 40, 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. *ACM CHI*.

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

ELDER, J., AND GOLDBERG, R. 2001. Image editing in the contour domain. In *TPAMI*, vol. 23, 291–296.

FEI-FEI, L., FERGUS, R., AND PERONA, P. 2004. Learning generative visual models from few training examples: an incremental bayesian approach tested on 101 object categories. In *Workshop on Generative-Model Based Vision, CVPR*.

GAVILAN, D., SAITO, S., AND NAKAJIMA, M. 2007. Sketch-to-collage. *ACM SIGGRAPH - Posters*.

HAYS, J., AND EFROS, A. A. 2007. Scene completion using millions of photographs. *ACM SIGGRAPH*.

HU, R., BARNARD, M., AND COLLOMOSSE, J. 2010. Gradient field descriptor for sketch based retrieval and localization. *ICIP*.

IGARASHI, T., AND HUGHES, J. F. 2001. A suggestive interface for 3d drawing. *ACM UIST*.

IGARASHI, T., MATSUOKA, S., AND TANAKA, H. 1999. Teddy: A sketching interface for 3d freeform design. *ACM SIGGRAPH*.

JACOBS, C. E., FINKELSTEIN, A., AND SALESIN, D. H. 1995. Fast multiresolution image querying. In *SIGGRAPH*.

LEE, D. C., KE, Q., AND ISARD, M. 2010. Partition min-hash for partial duplicate image discovery. In *ECCV*.

LOWE, D. G. 2004. Distinctive image features from scale-invariant keypoints. *IJCV*.

NISTER, D., AND STEWENIUS, H. 2006. Scalable recognition with a vocabulary tree. In *CVPR*.

SIVIC, J., KANEVA, B., TORRALBA, A., AVIDAN, S., AND FREEMAN, W. 2008. Creating and exploring a large photorealistic virtual space. In *Workshop on Internet Vision, CVPR*.

WINDER, S., HUA, G., AND BROWN, M. 2009. Picking the best daisy. In *CVPR*.

WITTEN, I. H., MOFFAT, A., AND BELL, T. 1999. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann.

ZITNICK, C. L. 2010. Binary coherent edge descriptors. In *ECCV*.