3D Object Retrieval Assignment

Summer term 2019
March 20, 2019

Requirements

The assignment involves the implementation and evaluation of an interactive and intelligent user interface, providing guidance for sketching based on similarity properties. The given requirements represent the minimal requirements. You are free to implement additional features and functionalities which will also be awarded with additional points. To settle any doubts it is advised to consolidate with a tutor concerning the feasibility and reward concerning a specific feature.

Implementation

One of the most frequently used languages with respect to computer graphics is C++, which also offers an extensive amount of useful libraries. Therefore you are strongly advised to use C++ for your implementation. If you are not familiar with this programming language there are many online tutorials (e.g. [1]) which allow for a quick start. Furthermore, the Qt library [2] is recommended to use for Graphical User Interface (GUI) generation and the OpenCV library [3] for image processing tasks. Basically you are also allowed to use another programming language but please beware that we can offer limited or no support and that the grading will be purely based on what can be presented at the final presentation.

Grading details

A total of 45 points can be achieved through the compulsory tasks and a variable number of bonus points by implementing the bonus tasks or self-conceptualized fitting ideas. The outcome of the practical should be presented at the final presentation on **June 19, 2019**, which will be the basis for the grading. The used source code shall be submitted by email not later than **June 19, 2019** at **23:59**.

No.	**Task**	**Points**	**Bonus Points**
1	Framework setup	15	
2	Shadowdraw sketch-assistance	30	15
	Sum	**45**	**15**

The grade for the assignment will be derived from the following point system:

Points	**Grade**
below 22	5.0
more than 22	4.0
more than 28	3.0
more than 34	2.0
more than 40	1.0

Part 1: Framework setup

The first task involves setting up you own framework. This is an essential prerequisite for the subsequent tasks and cannot be skipped. The here presented points represent the mandatory requirements which can be extended by other functionality.

1. Graphical User Interface

   Create a simple GUI which is capable of displaying an image. The user should be able perform zooming and dragging of the image[1].

   3 Points

2. File I/O

   Your implementation should be able load and export images through typical file dialogs, which should be triggered by respective buttons in the GUI, e.g. "load" and "export".

   2 Points

3. Drawing tools

   The second part of the assignment requires the user to provide input by sketch lines. Implement a basic brush/pencil tool which allows to add lines on top of a loaded image like in an image editor[2]. Additional options could include changing of brush size, shape and color. Note that the brush should adopt to zooming so that the strokes present painted on top of the image are of constant size.

   4 Points

4. Eraser tool

   An eraser tool should be able to (similar to the drawing tools) remove parts of the user-drawn sketch lines and the imported image.

   2 Points

5. Undo, redo

   Save the stroke lines drawn by the user in a suitable data structure and enable undo and redo the latest actions through dedicated buttons or keyboard input. Think of a reasonable way this could work in combination with the eraser tool.

   4 Points

[1]Example for Qt: `https://doc.qt.io/qt-5/qtwidgets-widgets-imageviewer-example.html`
[2]Example for Qt: `https://doc.qt.io/qt-5/qtwidgets-painting-basicdrawing-example.html`

Part 2: Shadowdraw sketch-assistance

The second part involves adding user guidance for sketching implemented in Part 1. The implementation should be loosely based on the "shadowdraw" approach, described by [4], which provides a suggestion for placement of additional sketch lines based on already present sketch lines and their similarity to images in an associated database. Your task involves to implement a functionality similar to this, but with a user input not consisting solely of sketch lines, but a hybrid of image and sketch as depicted in Figure 1. For this task you will be supplied with a a reference database[3] and a series of exemplary queries[4] which shall be completed by sketch. All images of the database and the queries are already aligned and scaled to an uniform size of 300 x 300 pixels rendering the alignment step mentioned in [4] obsolete. In the following, necessary subtasks for this task are presented with some details given. Note that the proposed algorithms are mere suggestions and you are encouraged to research and test others.

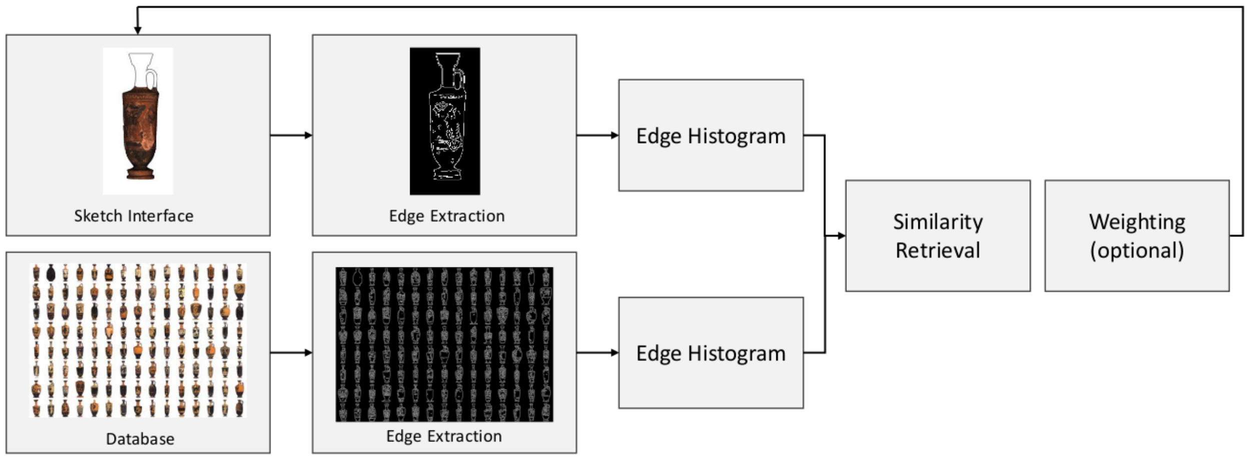

Figure 1: Application pipeline for a shadowdraw implementation.

- Edge extraction

  The first stage involves extracting the edges of all the images of the database as well as the image currently present your the sketch interface. Suitable methods for this are the Canny edge detector [5] (OpenCV features an implementation[5]) or the long edge detector [6], which is more adopted to edges generated by user sketch. The result of this step should be an edge image as suggested in Figure 1.

5 Points

[3]The database is provided as a zip file (`database.zip`) at containing jpg images at `https://puck.cgv.tugraz.at/s/2z5bz36TMGt25Re`. **Hint:** To speed up computation it is convenient to use just a fraction of the database for testing.
[4]Provided in `queries.zip` at `https://puck.cgv.tugraz.at/s/2z5bz36TMGt25Re`
[5]`https://docs.opencv.org/3.1.0/da/d22/tutorial_py_canny.html`

- Patch descriptors

  From the edge images a set of edge descriptors has to be computed. A standard approach to describe a patch is the Histogram of Gradients (HoG) by [7] which creates a histogram of local pixel gradients[6]. The Binary Coherent Edge Descriptor by [8] might be better suited for this task for it is based on length of edges rather than their relative gradient magnitude. Since we deal with incomplete images it is appropriate to compute the descriptors locally for patches of the overall image ([4] suggests a patch size of 60x60 pixels).

  5 Points

- Image matching

  Compare the edge descriptor computed from the query image to all of the edge descriptors from the search space. The similarities can be computed by a simple distance metric like the Euclidean distance [7]. The outcome of this should be a list of distances (one for each of the images in the database).

  7 Points

- Image blending

  After sorting the results, the n closest matches should be used for a sketching-suggestion. Therefore, the edge lines of those n images should be superimposed onto the user interface presented to the user. Evaluate which values of n provide a good balance between assistance and information overload. It is convenient to draw this edges lines on a separate image and blending it with the image currently present in the sketch-interface.

  8 Points

- Varying patch size **(Bonus task)**

  The patch size for the descriptor in the previous subtask is just a arbitrary choice. Implement a means to change the patch size based on user input and evaluate the effects of different sizes.

  3 Points

- Image weighting **(Bonus task)**

  The suggestions can be differently salient depending on how close the respective match is. The suggestion based on the best match should be more prominent that the suggestion based on the n-th match. A gradient can be easily achieved by adjusting brightness or transparency. Conceptualize and implement a mapping between match similarity and visibility of the edge suggestion. How does it improve the experience for different values of n?

  5 Points

[6]Implementation available in OpenCV: https://www.learnopencv.com/histogram-of-oriented-gradients/
[7]https://en.wikipedia.org/wiki/Euclidean_distance

- Mouse based weighting **(Bonus task)**

  A weighting can be also made based on the mouse position in the sketch-interface. Edge suggestions in close proximity to the mouse pointer should thereby be more prominent than such further away as depicted in Figure 2. Think of a way to decrease the visibility of edge suggestions based on the distance from the mouse pointer.

7 Points

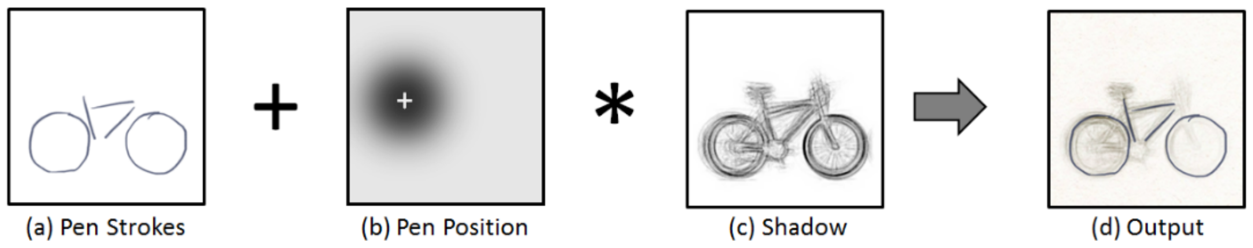

Figure 2: Edge suggestions to the sketch-image (a) can be enhanced in regions in close proximity to the pen (b).

References

[1] C++ Language Tutorial http://www.cplusplus.com/doc/tutorial/

[2] QT Developer Homepage http://trolltech.com/developer

[3] OpenCV (Open Source Computer Vision Library) https://opencv.org

[4] Lee, Yong Jae, C. Lawrence Zitnick, and Michael F. Cohen. "Shadowdraw: real-time user guidance for freehand drawing." *ACM Transactions on Graphics (TOG)*. Vol. 30. No. 4. ACM, 2011.

[5] Canny, John. "A computational approach to edge detection." *Readings in computer vision. Morgan Kaufmann*, 1987. 184-203.

[6] Bhat, Pravin, et al. "GradientShop: A gradient-domain optimization framework for image and video filtering." *ACM Trans. Graph.* 29.2 (2010): 10-1.

[7] Dalal, Navneet, and Bill Triggs. "Histograms of oriented gradients for human detection." *international Conference on computer vision & Pattern Recognition (CVPR'05)*. Vol. 1. IEEE Computer Society, 2005.

[8] Zitnick, C. Lawrence. "Binary coherent edge descriptors." *European Conference on Computer Vision*. Springer, Berlin, Heidelberg, 2010.