OpenVL: An Abstraction for Developer-Friendly Computer Vision

Gregor Miller and Sidney Fels
Human Communication Technologies Laboratory
University of British Columbia
Vancouver B.C., Canada
{gregor,ssfels}@ece.ubc.ca

ABSTRACT

Research into computer vision techniques has far outpaced the development of interfaces (such as APIs) to support the techniques' accessibility, especially to developers who are not experts in the field. We present a new description-based interface designed to be mainstream-developer-friendly while retaining sufficient power and flexibility to solve a wide variety of computer vision problems. The interface presents vision at the task level (hiding algorithmic detail) and uses a description derived from definitions of vision problems. We show that after interpretation, the description can be used to invoke an appropriate method to provide a result. Our implementation interprets the description and invokes various vision methods with automatically derived parameters, which we demonstrate on a range of tasks.

Keywords

Computer vision, shader language, abstraction layer

1. INTRODUCTION

Recent advances in the robustness of vision algorithms have led to a rise in production of real-world applications, from face detection on consumer cameras to advanced articulated modelling such as that on the Microsoft Kinect™. Little work has been published on how to provide access to these sophisticated techniques to developers; effective use of these methods requires extensive knowledge of how the algorithms work and how their parameters affect the results, expertise beyond the scope of mainstream developers.

The contribution is a task-based description applied as an abstraction to computer vision, to hide the details of specific methods and their configuration. The abstraction may be used by developers to describe the type of vision problem they are trying to solve, and our novel interpreter uses the description to select an appropriate algorithm (with parameters derived from the developer's description).

Developing a high-level abstraction for computer vision is important for various reasons: 1) Mainstream developers may focus on their application's main task, rather than the algorithms; 2) Advances in the state-of-the-art can be incorporated into existing systems without re-implementation; 3) Hardware acceleration of algorithms may be used transparently; 4) The limitations of a particular platform can be taken into account automatically e.g. mobile devices may require a set of low-power consuming algorithms; 5) Computer vision expertise can be more readily adopted by researchers in other disciplines and general developers. This idea has been applied successfully in many other fields, notably OpenGL[13], but not yet in computer vision.

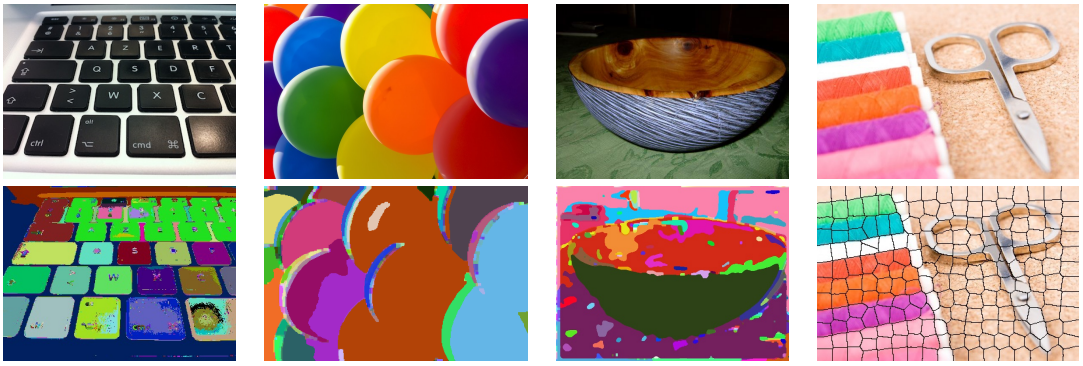
In this paper the goal is to describe as vision problems with the smallest description language, thereby laying the foundations for a more general abstraction. We present a variety of fundamental computer vision problems which can be described using our interface, and which provide access to the sophisticated methods hidden beneath the abstraction. The examples we provide are intended to demonstrate how task-based descriptions may be used as interfaces to sophisticated vision technologies. The research we present here is intended as a positive first step towards a general developer-friendly abstraction for computer vision.

2. RELATED WORK

There are many openly available computer vision libraries that provide common vision functionality, such as OpenCV [3], Mathworks Vision Toolbox[1] and Gandalf[2]. These libraries often provide utilities such as camera capture or image conversion as well as suites of algorithms, which has previously been shown to lessen the effectiveness on application [8]. All of these software frameworks and libraries provide vision components and algorithms without any context of how and when they should be applied, and so often require expert vision knowledge for effective use.

User-friendly and developer-friendly computer vision is an active research topic in the Human-Computer Interaction community, mostly investigating design guidelines for graphical user interfaces when providing access to some subset of vision methods. For example, [5] developed an interactive tool which incorporates a simple painting metaphor for users to train a machine learning system. The interface presents the image training set, the pixel-level classification and re-classification options, which allows a user to develop a detector for any subject, given enough representative data.The breadth of possible techniques is limited to classification problems, whereas we would provide a more general purpose vision framework. [6] introduced a toolkit targeted towards the creation of tangible input systems, using some basic vision methods to support the use of cameras. Objects are defined through selection in an image and then

[1]http://www.mathworks.com/products/computer-vision
[2]http://gandalf-library.sourceforge.net



(a) (b) (c) (d)

Figure 1: The segmentation description operates on the basis of properties such as colour (a), intensity (b) and texture (c), while also providing constraints on size, quantity and shape regularity (d).

represented within the API, from where they can be tied to various actions or names, essentially allowing supervised classification at the developer level. However, the developer is not interacting with computer vision, but with the result of a routine written by the authors, and therefore this targets a different audience from the interfaces we are investigating. [9] introduced the GUI Eyepatch with similarities to form designers in Visual Basic, to provide users with a mechanism to tie computer vision tasks to actions. The tasks were constrained to classification and segmentation, using binary classifiers to produce image regions as a result, which limits the range of application. An environment called *Gestalt* [11] supports the debugging of machine learning, on the basis that programming with machine learning is significantly different from traditional programming. Their approach uses a pipeline model for debugging software which utilises machine learning, with visualizations after each step. This approach is limited as it requires knowledge of how machine learning works and how to fix it when it fails.

3. TASK-BASED ABSTRACTION

Our principle goal with this research is to provide developers with an intuitive interface to sophisticated computer vision methods. Due to the nature of computer vision there are no well-defined rules to extract a model from an image - unlike the inverse problem of computer graphics, which uses well-defined rules to render models to an image. Therefore a vision abstraction requires an additional level of complexity in order to provide a similar level of usability as that of, for example, a graphics abstraction.

This additional level in our framework is an interpreter which encodes expert knowledge of computer vision and contains a suite of vision algorithms. The interpreter is given a description of the task which contains sufficient information such that an algorithm can be chosen and its parameters derived automatically from the description (in some cases more than one algorithm may be invoked).

3.1 Single Viewpoint

We begin with the task-based description of single viewpoint problems: this includes segmentation, matting, object detection and recognition. The first description we require from the developer is the image properties, such as the quantity of noise. We use these to help in the method selection process or to pre-process images (e.g. filtering noise).

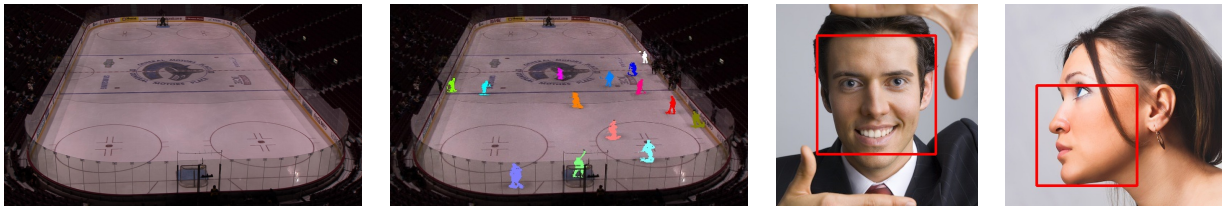
To allow developers to intuitively describe the contents of images we use a *segment* metaphor as the basis for our abstraction: this metaphor has no direct link to algorithms and does not require developers to think of specific methods. Developers are required to provide a description of the kind of segments they require for their task; this is accomplished through a description of *segmentation*.

3.1.1 Segmentation

Due to the complexity of the problem, we limit ourselves to a basic definition of *segmentation*: producing a set of distinct regions (*segments*) within the image. We apply the idea of *properties* to provide the developer with control over what makes a segment distinct. A property may be anything measurable over a region of the image, which leads to an extensive list of possibilities, such as colour, intensity, texture, shape, contour, etc. Conceptually, a segment is bounded by a smooth, continuous contour, and is not dependent on pixels or any other discrete representation. Points within a region share one or more properties, thus each region is distinct from its surroundings. Segments must have at least one property, however there is no limit on how many properties may be defined. Segment properties allow developers to decompose the image based on what they consider to be important to their problem.

Each property is associated with a *distinctiveness*, to allow the developer to define how distinct each segment should be relative to that property. For example, we could segment the image using the colour property and a low distinctiveness, which would result in a small number of segments, each consisting of relatively similar colours. The higher the distinctiveness, the more strict the comparisons and the higher the number of segments produced. Example segmentations based on a property with associated distinctiveness are shown in Figure 1(a-c). The model also allows the specification of multiple properties for a single segmentation. This will lead to segments which are distinct based on all specified properties, although each property may have a different level of distinctiveness.

The segmentation operation itself is defined at quite a low level, with no grouping or high-level labelling. There



(a) (b) (c) (d)

Figure 2: The detection description in our framework can be given an example image (a) and be told to find segments which are dissimilar to the example at equivalent scale (b). Alternatively a literal description can be given, in this case of a human face, and the developer can describe the type of face they would like to detect: (c) large, frontal, no occlusion or (d) medium, profile, partial occlusion.

are no provisions for developers to request that only certain segments be returned, e.g. "segment only red regions", or be labelled in any way other than by property. After segmentation is performed developers may select segments from the result based on their properties. The properties form what we define as the *requirements* of segmentation. The other component we need to complete our description is *constraints*. The three central constraints we provide are *size*, *quantity* and *regularity*. An example of a size and regularity constrained segmentation is shown in Figure 1(d). Size governs the final size of the segments (based on some hint, such as `exact` or `average`), quantity the number (again, using a hint-based system), and regularity the level of variation allowed in the gradient of the segment's contour.

Regularity constrains the shape of the segments: zero regularity does not constrain the shape at all (this is the default) and full regularity constrains the shape of every segment to be the same (discretising the image).

3.1.2 Object Detection

Detection is the process of identifying segments within an image which correspond to an instance of a class defined by a *template* (e.g. finding regions of an image which contain a human face). In our framework we use templates to allow developers to describe objects. Templates are fairly general, and can be based on example images or literal descriptions.

Example Images: In the majority of detection applications there will not be a pre-existing model, or a specific algorithm for the required class, so we provide a detection template which can be given an example image. The developer describes the template-matching process for the input images, with a similarity, scale and object size description. An example-based template detection is shown in Figure 2; the description used for the result shown was to find mostly dissimilar objects ($similarity = 0.2$) with equivalent image size ($scale = 1$) and small matching size ($size = [0, 0.2]$). The template example image is shown in (a) (a 'background image'), and the input image is analysed to find regions which satisfy the description, with the result shown in (b).

Literals: Detection literals are direct descriptions of known classes for which a small set of examples is not sufficient. We allow the developer to provide the parameters of a face and the interpreter chooses a method capable of finding the described face. Our description provided parameters for pose (2D and 3D), size, occlusion, gender, age, expression and illumination. Examples of this description applied to images are shown in Figure 2(c-d), for frontal and profile faces.

3.2 Multiple Viewpoints

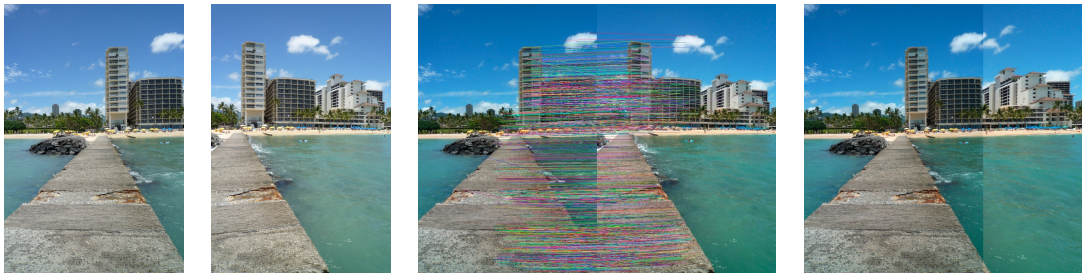
The next class of problem is that of *multiple viewpoints*: we define this as different images conceptually captured simultaneously. The developer would like to find a model which relates segments and image properties among images. We accomplish this by providing segment *variances*: a model of how each segment property varies between images. For example, in image registration a segment's position in the image's frame of reference will change, as well as properties such as intensity (HDR registration) or blur (multi-focal registration). Variances provide more flexibility and a more concise form than using an image-level description. For example, segment variances encode image properties such as overlap, structural change, and variability of blur or intensity, whereas individual descriptions for each of these would be required for image variances.

Correspondence: The first problem we describe for multiple viewpoints is one of the fundamental vision tasks, *correspondence*, which we define here as the problem of finding unique matches across images. The developer provides variances to describe how much a segment's position varies and if the properties will change (such as intensity or blur). Constraints define how strong the match should be. The density of matches is defined by the strength. An example correspondence is presented in Figure 3(c), with strong matches highlighted with colourful lines.

Image registration: This is the problem of finding transformations which will align multiple images together in as seamless a manner as possible. Our framework supports registration as an extension of correspondence; if the developer supplies segment descriptions and variances, an image-level solution can be requested based on the correspondences. This will initiate an optimisation over the correspondences to find a single transform for every image.

4. IMPLEMENTED ABSTRACTION

Our implementation has been created with C++, and uses many freely available methods and some algorithms implemented by ourselves. For segmentation (using Figure 1 for reference): colour-based feature-space clustering and labelling (a) [2]; colour-based SLIC super pixels (d) [1]; colour-based seeded image-space region growing [12]; intensity-based seeded image-space region growing (b) [12]; texture-based seeded image-space region growing (c) [12]. The other methods we use are: threshold-based background subtraction for example-based object detection (Figure 2b)



(a) (b) (c) (d)

Figure 3: In (c), the correspondence variances were defined as *horizontal position* $= 0.3$**,** *vertical position* $= 0.1$**,** *colour* $= 0.1$ **to accommodate a horizontal shift and any small changes in colour. The strength was set as high, which leads to many correspondences but not a match for every segment. A global transform is computed for each image based on the correspondences, and the result in (d) is produced.**

[10]; boosted cascade of simple features for face detection, using OpenCV (Figure 2c-d) [3]; scale-invariant feature transforms (SIFT) for strong feature matching (Figure 3c) [7]; SIFT with global feature optimization specialised for creating panoramas (Figure 3d) [4]. Our framework is very fast, since the interpretation does not require much data processing; however, the algorithms we use are in general fairly slow. In principle the framework could work easily in real-time given another set of algorithms, perhaps optimised using GPU-based methods.

5. CONCLUSIONS

We have presented our novel abstraction for developer-friendly computer vision, which hides the details of algorithms by providing the developer with an intuitive description model. This can be used to describe the vision problem conditions from which our novel interpreter will select an appropriate method to produce a solution. We have presented descriptions and results for image segmentation, object detection (using templates which are example- or literal-based), correspondence and image registration. As an abstraction, this could lead to multiple implementations: a reference software version, GPU, FPGA, mobile, embedded, etc. The framework currently requires the developer to specify which high-level problem to solve (such as detection) and then provide the description. We are working on a new higher-level description which would let developers customise the steps taken to solve the problem: essentially creating a computer vision 'shader' language.

6. REFERENCES

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Sabine. Slic superpixels. Technical Report 149300, EPFL, June 2010.

[2] S. T. Bow. *Pattern Recognition and Image Preprocessing*. CRC Press, 2nd edition, January 2002.

[3] G. Bradski and A. Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, Inc., 1st edition, October 2008.

[4] M. Brown and D. G. Lowe. Recognising panoramas. *Proceedings of the 9th International Conference on Computer Vision*, 2:1218–1225, October 2003.

[5] J. Fails and D. Olsen. A design tool for camera-based interaction. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '03, pages 449–456, New York, NY, USA, 2003. ACM.

[6] S. R. Klemmer, J. Li, J. Lin, and J. A. Landay. Papier-mache: toolkit support for tangible input. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 399–406. ACM, 2004.

[7] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91, Nov 2004.

[8] A. Makarenko, A. Brooks, and T. Kaupp. On the benefits of making robotic software frameworks thin. In *Proceedings of the Workshop on Measures and Procedures for the Evaluation of Robot Architectures and Middleware*, IROS'07, New York City, New York, U.S.A., October/November 2007. IEEE. Invited Presentation.

[9] D. Maynes-Aminzade, T. Winograd, and T. Igarashi. Eyepatch: prototyping camera-based interaction through examples. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*, UIST '07, pages 33–42, New York, NY, USA, 2007. ACM.

[10] D. H. Parks and S. Fels. Evaluation of background subtraction algorithms with post-processing. In *IEEE International Conference on Advanced Video and Signal-based Surveillance*, 2008.

[11] K. Patel, N. Bancroft, S. M. Drucker, J. Fogarty, A. J. Ko, and J. Landay. Gestalt: integrated support for implementation and analysis in machine learning. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, UIST '10, pages 37–46, New York, NY, USA, 2010. ACM.

[12] K. B. Shaw and M. C. Lohrenz. A survey of digital image segmentation algorithms. Final Technical Report ADA499374, Naval Oceanographic and Atmospheric Research Lab, January.

[13] D. Shreiner, M. Woo, J. Neider, and T. Davis. *OpenGL(R) Programming Guide : The Official Guide to Learning OpenGL(R), Version 2 (5th Edition)*. Addison-Wesley Professional, Aug. 2005.