OpenVL: A Task-Based Abstraction for Developer-Friendly Computer Vision

Gregor Miller and Sidney Fels
Human Communication Technologies Laboratory
University of British Columbia
{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 task-based description derived from definitions of vision problems. We show that after interpretation, the description can be used to invoke an appropriate method to provide the developer's requested result. Our implementation interprets the description and invokes various vision methods with automatically derived parameters, which we demonstrate on a range of tasks.

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 of this paper 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. If any abstraction is used to access vision methods, hardware and software developers of the underlying mechanisms are free to continually optimise and add new algorithms. This idea has been applied successfully in many other fields, notably OpenGL[23], but not yet in computer vision.

There has been a recent industry push to define standards for access to computer vision: Khronos have a working group developing a hardware abstraction layer (tentatively titled CV HAL) to accelerate vision methods.[1] Khronos are proposing a layer beneath libraries such as OpenCV [3] in order to accelerate existing library calls (much like projects such as OpenVIDIA[2]). However, the OpenCV API presents algorithms directly to the developer, and, without expertise in vision, developers are not able to fully take advantage of the methods within. Our proposed abstraction would act as an additional higher-level layer to hide the details of algorithms and hardware acceleration from developers and allow them to focus on developing applications.

In this paper the goal is to describe as many problems as possible 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. We do not attempt to provide a completely general abstraction, since this would be larger than the scope of a single paper. 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.

[1]http://www.khronos.org/vision
[2]http://openvidia.sourceforge.net

2. Related Work

Many attempts have been made to develop computer vision or image processing frameworks that support rapid development of vision applications. Image understanding systems attempted to make use of developments in artificial intelligence to automate much of the vision pipeline [14, 10, 5]. The Image Understanding Environment project (IUE) [16] in particular attempted to provide high-level access to image understanding algorithms through a standard object-oriented interface in order to make them accessible and easier to reuse. More recently the OpenTL framework [18] has been developed to unify efforts on tracking in real-world scenarios. All of these approaches essentially categorise algorithms and provide access to them directly, requiring developers to have expert knowledge of vision methods and to deal directly with images and algorithms.

Visual programming languages that allow the creation of vision applications by connecting components in a data flow structure were another important attempt to simplify vision development, including [11] and Apple's Quartz Composer[3]. These contained components such as colour conversion, feature extraction, spatial filtering, statistics and signal generation, among others. Declarative programming languages have also been used to provide vision functionality in small, usable units, e.g. ShapeLogic[4] or FVision [21], although they are limited in scope due to the difficulty of combining logic systems with computer vision. While these methods provide a simpler method to access and apply methods, there is no abstraction above the algorithmic level, and so users of these frameworks must have a sophisticated knowledge of computer vision to apply them effectively.

One previous attempt at overcoming the usability problems associated with image understanding is discussed in the RADIUS project [7], which employed user-manipulated geometric models of the scene to help guide the choice of image processing algorithms. This operates at a higher level than the previous frameworks, however it trades off power, breadth and flexibility to provide its abstraction.

There are many openly available computer vision libraries that provide common vision functionality, such as OpenCV [3], Mathworks Vision Toolbox[5] and Gandalf[6]. 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 [13]. 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.

[3]https://developer.apple.com/technologies/mac/graphics-and-animation.html

[4]http://www.shapelogic.org

[5]http://www.mathworks.com/products/computer-vision

[6]http://gandalf-library.sourceforge.net

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, Fails and Olsen [6] 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.

Klemmer *et al*. [9] 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 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.

Maynes-Aminzade *et al*. introduced the GUI Eyepatch [15] 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.

A development environment called *Gestalt* [20] supports the application of machine learning by non-experts, 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.

In general it is difficult for developers to choose the optimal method for their particular application. Based on informal interviews, we have found that many developers still use trial-and-error to select an approach from a computer vision library, and most often do not get the results they desire. One example of this is the OpenCV face detection implementation, which is unable to detect off-axis faces (between front and profile) but developers often expect it to do so. In our framework, we present developers with an interface to describe the conditions of the problem and our interpreter attempts to find an acceptable solution.

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

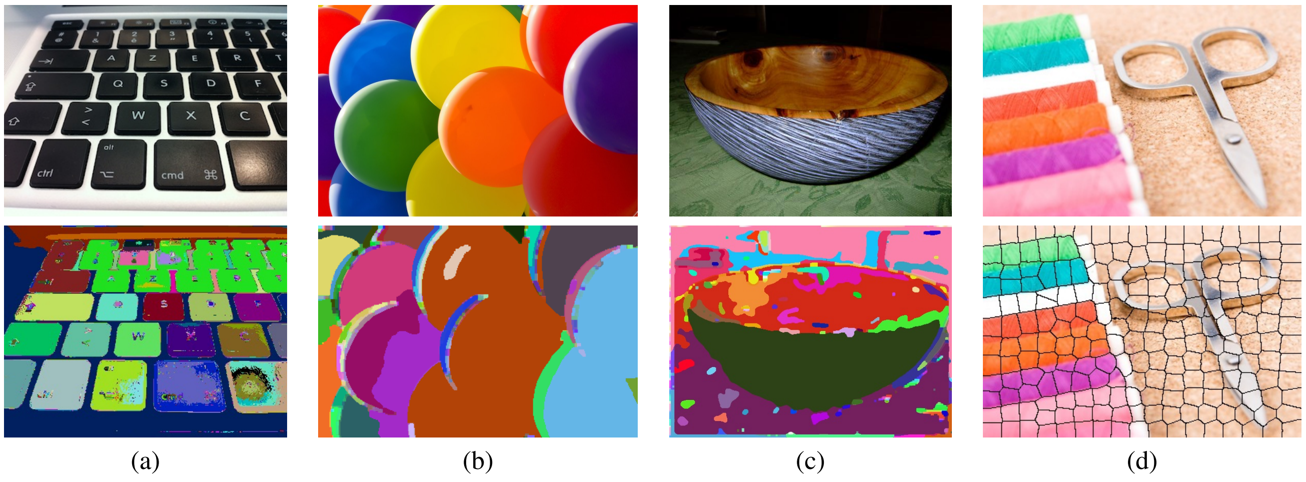


Figure 1. The original images (top) with the results of segmentation (bottom). The segmentation description operates on the basis of properties such as colour (a), intensity (b) and texture (c), while also providing constraints on size, quantity and shape regularity (d).

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). The steps involved in the abstraction process are:

1. *Developer*: Provide the interpreter with a description of the problem and the images to analyse.
2. *Interpreter*: Establish problem based on description.
3. *Interpreter*: Choose an algorithm and derive its parameters from the description.
4. *Interpreter*: Convert input images to algorithm format; execute algorithm.
5. *Algorithm*: Find a solution based on interpreter-defined parameters and input.
6. *Algorithm*: Convert solution into framework-defined format (for a consistent result).
7. *Interpreter*: Package solution for use by the developer.

The main task of the interpreter is to choose an algorithm and derive the parameters. Algorithms are linked to the interpreter via a *condition matrix*: the rows of the matrix are algorithms, the columns are aspects of the task description (conditions). Each cell of the matrix contains a tuple list which defines the range in which the algorithm works under that particular condition. Each row is treated as a single sample of algorithm effectiveness, so that an algorithm's working conditions can be dependent (e.g. an algorithm may work well in the presence of blur or noise, but not both). For this reason, an algorithm may be listed in multiple rows to enumerate all of the conditions in which it works effectively. An example condition matrix is shown in Table 1; values in the condition matrix are defined by those who implemented the vision algorithms.In addition to the condition matrix, the developer may provide *hints* to the interpreter for the type of processing required, such as constraints on determinism or speed/accuracy requirements.

One of the disadvantages to using an abstraction is the tendency to lose flexibility and power. Our goal is to provide an abstraction at the lowest possible level above algorithmic detail to retain as much power as possible. We believe that with a sufficiently rich description model and a correspondingly complete set of algorithms (in the sense that all possible ranges offered by the description are covered), very little flexibility is lost. Since the parameters of the algorithm will directly affect the result, the interpreter's derivation of the parameters from the description offers a highly customised solution to the developer.

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 meth-

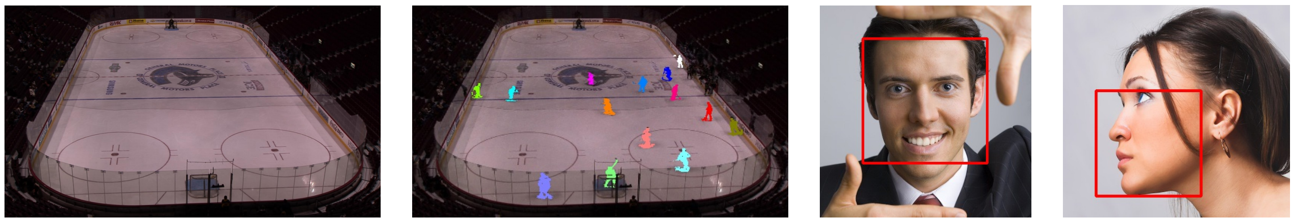


Figure 2. The detection description in our framework can be given an example image (a) and be told to find segments which are dissimilar to the example at equivalent scale (b). Alternatively a literal description can be given, in this case of a human face, and the developer can describe the type of face they would like to detect: (c) large, frontal, no occlusion or (d) medium, profile, partial occlusion. (Note that the visualization of face detection is not segments; we are working on extracting the segments which correspond to just the face.)

ods. 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 will be 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. One method of providing a segmentation with multiple properties would be to segment the image once per property, and then perform an intersection operation on all images to collect the final set of segments. However, there are also algorithms which can perform multiple property segmentations simultaneously and therefore perform more efficiently. The advantage to the description model is that the details are hidden from the developer, and so they do not need to take this into account when creating an application.

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

Segments are used for two purposes: the first as the representation if the developer requests an image segmentation; the second, and more important, is as the unit used to facilitate description of the contents of images, and as the basis for comparisons and providing results. Often a segmentation will be performed even if that is not the described problem, so that the framework can return segments as the result of another operation, such as detection.

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

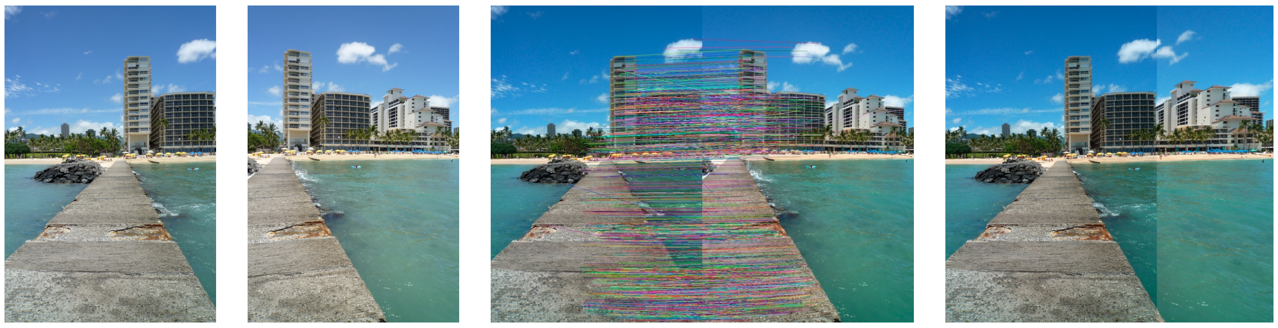

(a) (b) (c) (d)

Figure 3. The correspondence description uses segment variances to provide an initial estimate of where to look between images for correspondences. In (c), the 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.

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. First, a similarity must be defined to constrain possible matches: a value of 0 for completely dissimilar, and a value of 1 for identical. A scale is defined to relate the size of the example image to that of the input image.Finally, a matching size is provided to determine how much of the example template to match to the input image: a size of 1 means match the entire template, and values in the range $[0, 1)$ will match portions of the image.

Our detection process requires a segmentation description in addition to the template 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).

There are many other uses for this type of description: matting and chroma-keying, content-based retrieval, background subtraction etc. Our model at the moment is very basic (for example, we need to be able to define how similarity operates, such as on segment properties), but it demonstrates the potential power of the abstraction.

Literals: Detection literals are direct descriptions of known classes for which a small set of examples is not sufficient. We use face detection to illustrate our approach: unlike frameworks such as OpenCV [3] (which uses algorithm-level APIs - the function to detect faces is called `cvHaarDetect`) 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 is based on a prior examination of the face detection problem: Jang *et al.* [8] presented a taxonomy of face detection methods using an arrangement based on the problem conditions. Their 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. While methods for face detection are well-known, we believe our problem description is scaleable and relatively future-proof when it comes to algorithm development, as new algorithms can be added without any other implementation changes (this is true of the entire abstraction). The description also acts as a filter if we have prior knowledge: if the developer is only interested in large frontal faces, the internal algorithms are optimised for this and return results much more quickly than if no description was given.

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

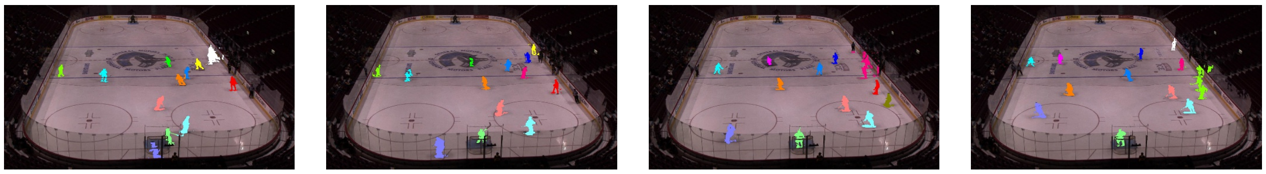


Figure 4. The object tracking description uses the example-based detection shown in Figure 2 with the addition of images with relative timecodes, and segments with a set of temporal variances set to allow a small amount of movement over a short period of time (consistent with the motion of a hockey player). These images show a progression in the sequence, with approximately $0.5s$ between each image.

each of these would be required for image variances.

3.2.1 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. Given a segmentation description, 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 number of segments requested in the segmentation description: if the quantity is high, size is small or property distinctiveness is high then many more segments will conceptually be produced, which leads to a search for many more matches. An example correspondence is presented in Figure 3(c), with strong matches highlighted with colourful lines.

3.2.2 Image Registration

Image registration 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 optimization over the correspondences to find a single transform for every image.

3.3. Spatio-temporal

The final range of problems we discuss are *spatio-temporal*, where we introduce a description of time to our single viewpoint scenario. Within computer vision, time is effectively dealt with as a derivative, i.e. what is the rate of change of our data and how is that reflected in the resulting model. Therefore we allow developers to attach relative timecodes to the input images (so that the developer can specify their own temporal coordinate system - this also removes the requirement for the segment metaphor to deal with frames, which is helpful since we have already removed pixels from the abstraction). For the period of time between the capture of two images, we can represent the rate of change for segments using the same variances structure as defined for correspondence (Section 3.2.1. The developer provides the $temporal$ variances with a length of time for which the variances correspond (e.g. if the position variance is set to 0.05 and the length of time is set to 0.33, then we know the segments' positions vary by $0.05 \times image\ width$ every 0.33 units of time).

3.4. Object Tracking

As an example of a spatio-temporal problem, we have a simple object tracking description available to developers, an example of which is shown in Figure 4. We use the description for example-based detection in Section 3.1.2, and provide a small temporal variance for position every $\frac{1}{30}$ seconds. Combined with timecodes for the input images and a timecode to indicate when the analysis should be performed, the interpreter has enough information to perform simple tracking. This is the first problem where the interpreter is required to keep an internal state between algorithm executions: if no state is stored, tracking must be re-initialised every time which requires having all previous images in memory. We have adopted a small state to let processed images be released from memory while maintaining information on previously detected and tracked objects.

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 [22]; intensity-based seeded image-space region growing (b) [22]; texture-based seeded image-space region growing (c) [22]; we also use combinations of these, such as texture segmentation with super pixels to provide texture-based regularly shaped segments.

Table 1 shows the condition matrix we use for segmentation. Internally, each algorithm registers itself with the interpreter along with the permutations of the description for which it can provide solutions. This may be a single set of ranges which define a single volume in description space

Table 1. An example condition matrix for image segmentation algorithms. Each row specifies an algorithm, followed by a property for which the algorithm can provide a solution, and the set of constraints the algorithm can satisfy. Note that for [2] and [1], multiple specific cases are presented since other permutations would not be permitted. The other methods satisfy any permutation of their given conditions

Algorithm	Properties	Distinctiveness	Size	Quantity	Regularity
[2]	Colour	High	All	All	0
	Colour	All	Large	All	0
[1]	Colour	All	Exact	All	$[0.0, 1.0]$
	Colour	All	All	Exact	$[0.0, 1.0]$
	Colour	All	All	All	$[0.26, 1.0]$
[22]	Colour	Low, Medium	Small, Medium	All	$[0, 0.25]$
[22]	Intensity	All	All	All	$[0, 0.25]$
[22]	Texture	All	All	All	$[0, 0.25]$

(such as Table 1 (c-e)) or multiple sets of ranges which define more than one volume (Table 1 (a,b)). The interpreter constructs the condition matrix by collating all ranges.

Our example GUI for segmentation, which is meant for illustrative purposes only (it uses the developer interface internally), was developed in Apple Xcode using objective-c as a front-end. Figure 5 provides screenshots illustrating high distinctiveness colour segmentation (a), and medium distinctiveness medium size texture segmentation (b).

The other methods we use are: threshold-based background subtraction for example-based object detection (Figure 2b) [19]; boosted cascade of simple features for face detection, using OpenCV (Figure 2c-d) [24, 3]; scale-invariant feature transforms (SIFT) for strong feature matching (Figure 3c) [12]; SIFT with global feature optimization specialised for creating panoramas (Figure 3d) [4]; threshold-based background subtraction with particle filter tracking with colour-histogram comparison (Figure 4a-d) [19, 17].

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, image registration and object tracking. The descriptions for these are simple compared to the task of finding an appropriate algorithm manually, and all that is required of the developer is a knowledge of the problem and which result they require.

As an abstraction, this could lead to multiple implementations: a reference software version, GPU, FPGA, mobile, embedded, etc. The single interface would work with any of these back-end implementations, leading to portable code and an expansion in the use of computer vision. It would also provide computer vision methods to general developers without requiring special training, and state-of-the-art methods would be available to researchers in other fields.

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. We are also continuously adding implementations of algorithms to support our descriptions, and also adding new descriptions to support existing problems with more depth and new problems. In the near future we hope to have optical flow, stereo reconstruction and camera calibration included in our framework.

6. Acknowledgements

We gratefully acknowledge support from NSERC (grant provided for "*Diving experiences: wayfinding and sharing experiences with large, semantically tagged video*"), Bell Canada, Avigilon Corporation and Vidigami Media Inc.

References

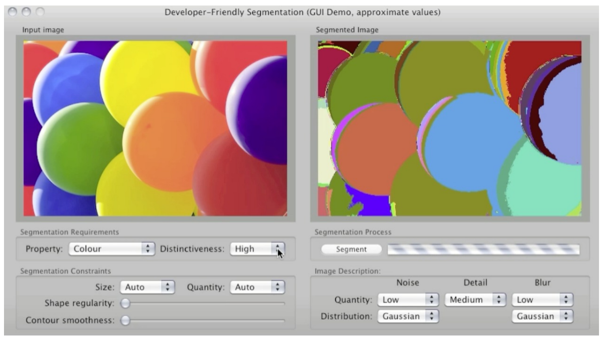
[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Sabine. Slic superpixels. Technical Report 149300, EPFL, June 2010. 6, 7

[2] S. T. Bow. *Pattern Recognition and Image Preprocessing*. CRC Press, 2nd edition, January 2002. 6, 7

[3] G. Bradski and A. Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, Inc., 1st edition, October 2008. 1, 2, 5, 7

[4] M. Brown and D. G. Lowe. Recognising panoramas. *Proceedings of the 9th International Conference on Computer Vision*, 2:1218–1225, October 2003. 7

[5] R. Clouard, A. Elmoataz, C. Porquet, and M. Revenu. Borg: A knowledge-based system for automatic generation of im-



Property: Colour Distinctiveness: High

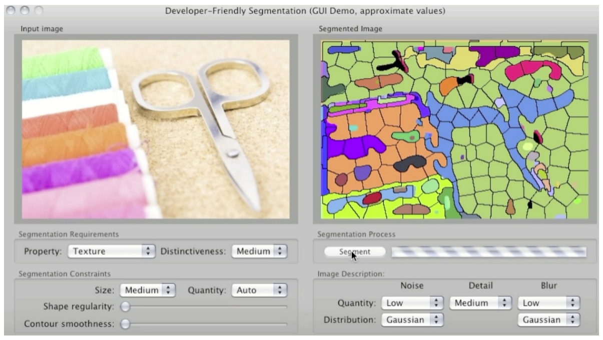
Size: Auto Quantity: Auto

Shape regularity:

Contour smoothness:

	Noise	Detail	Blur
Quantity:	Low	Medium	Low
Distribution:	Gaussian		Gaussian

(a)



Property: Texture Distinctiveness: Medium

Size: Medium Quantity: Auto

Shape regularity:

Contour smoothness:

	Noise	Detail	Blur
Quantity:	Low	Medium	Low
Distribution:	Gaussian		Gaussian

(b)

Figure 5. Screen captures of our segmentation UI to demonstrate the power of our abstraction. (a) demonstrates the options used to provide a high distinctiveness colour segmentation; (b) demonstrates a medium distinctiveness medium size texture segmentation.

age processing programs. *Transactions on Pattern Analysis and Machine Intelligence*, 21:128–144, February 1999. 2

[6] 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. 2

[7] O. Firschein and T. M. Strat. *RADIUS: Image Understanding For Imagery Intelligence*. Morgan Kaufmann, 1st edition, May 1997. 2

[8] D. Jang, G. Miller, S. Fels, and S. Oldridge. User oriented language model for face detection. In *Proceedings of the 1st Workshop on Person-Oriented Vision (POV)*, WVM'11, pages 21–26, New York City, New York, U.S.A., January 2011. IEEE. 5

[9] 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. 2

[10] C. Kohl and J. Mundy. The development of the image understanding environment. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, CVPR'94, pages 443–447, Los Alamitos, California, U.S.A., June 1994. IEEE Computer Society Press. 2

[11] K. Konstantinides and J. R. Rasure. The Khoros software development environment for image and signal processing. *IEEE Trans. on Image Processing*, 3:243–252, 1994. 2

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

[13] 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. 2

[14] T. Matsuyama and V. Hwang. SIGMA: a framework for image understanding integration of bottom-up and top-down analyses. In *Proceedings of the 9th international joint conference on Artificial intelligence*, volume 2, pages 908–915. Morgan Kaufmann Publishers Inc., 1985. 2

[15] 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. 2

[16] J. Mundy. The image understanding environment program. *IEEE Expert: Intelligent Systems and Their Applications*, 10(6):64–73, 1995. 2

[17] K. Nummiaro, E. Koller-Meier, and L. V. Gool. An adaptive color-based particle filter. *Image and Vision Computing*, 21(1):99 – 110, 2003. 7

[18] G. Panin. *Model-based Visual Tracking: the OpenTL Framework*. John Wiley and Sons, 1st edition, 2011. 2

[19] 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. 7

[20] 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. 2

[21] J. Peterson, P. Hudak, A. Reid, and G. D. Hager. Fvision: A declarative language for visual tracking. In *Proceedings of the Third International Symposium on Practical Aspects of Declarative Languages*, PADL '01, pages 304–321, London, UK, 2001. Springer-Verlag. 2

[22] 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 1995. 6, 7

[23] 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. 1

[24] P. Viola and M. J. Jones. Robust real-time face detection. *Int. J. Comput. Vision*, 57:137–154, May 2004. 7