Developer-Friendly Segmentation using OpenVL, a High-Level Task-Based Abstraction

Gregor Miller[1], Daesik Jang[2] and Sidney Fels[1]
[1]Human Communication Technologies Laboratory
University of British Columbia
{gregor,ssfels}@ece.ubc.ca
[2]Department of Electrical Engineering
Kunsan National University
dsjang@kunsan.ac.kr

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 interface, specifically for segmentation methods, designed to be application-developer-friendly while retaining sufficient power and flexibility to solve a wide variety of problems. The interface presents segmentation at a higher level (above algorithms) and uses a task-based description derived from definitions of low-level segmentation. We show that through interpretation, the description can be used to invoke an appropriate method to provide the developer's requested result. Our proof-of-concept implementation interprets the model description and invokes one of six segmentation methods with automatically derived parameters, which we demonstrate on a range of segmentation tasks. We also discuss how the concepts presented for segmentation may be extended to other computer vision problems.

1. Introduction

Advances in the robustness of vision methods have led to a surge in real-world applications, from face detection on consumer cameras to articulated modelling such as that on the Microsoft Kinect™. Little work has been published on how to provide access to these sophisticated vision techniques directly to mainstream developers; effective application of these methods requires extensive knowledge of how they work and how their parameters affect the results. This level of knowledge is beyond the scope of most general software developers, especially taking into consideration the wide variety of computer vision topics.

Our novel contribution is a developer-centred interface which uses a task description, providing an abstraction over the details of vision methods. The task-based description language is based on definitions of low-level segmentation, which is formed into an interface to be used by developers to describe the type of segmentation problem they are trying to solve; our novel interpreter uses the description to select an appropriate algorithm and derive its parameters.

Developing an abstraction for computer vision is important for many reasons: 1) Developers may focus on their applications 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 in graphics [21], but none have yet been successful within vision.

There has been a recent industry push to define standards for access to computer vision: the standards group Khronos are developing a hardware abstraction layer to accelerate core 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]). We believe this abstraction layer has been targeted at too low a level to be useful for mainstream developers. We propose an additional higher-level layer using a task-based abstraction to hide the details of algorithms,

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

platforms and hardware acceleration.

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 [11, 7, 4]. The Image Understanding Environment project (IUE) [12] 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 [14] 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.

The RADIUS project attempted to overcome the usability problems associated with image understanding [6] by employing user-manipulated geometric models of the scene to help guide the choice of image processing algorithms. This operates at a higher-level than the previously mentioned frameworks, however it trades off power, breadth and flexibility to provide its abstraction.

There are many open vision libraries that provide common vision functionality, such as OpenCV [3], Mathworks Vision Toolbox[3] and Gandalf[4]. 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 [10]. 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.

The previous references covered work on computer vision frameworks for developers; the following discusses the related work on segmentation. Various surveys provide excellent overviews of the versatile approaches used for image segmentation. Shaw *et al*. surveyed important methods for segmentation based on intensity, colour and texture properties [19]. Skarbek *et al*. categorised various approaches more in depth mainly focused on colour segmentation [22] . Raut *et al*. added some modern approaches as well [16]. From these analyses we can summarise the important approaches of segmentation as follows:

Thresholding: These are generally used for greyscale images and are simple to implement [13]. Some methods use multi-dimensional histograms to extend this approach to include colour and texture properties for the segmentation [16].

Clustering: Many techniques have been proposed in the literature of cluster analysis [2]. A classical technique for colour segmentation is k-means [9]. ISODATA (Iterative Self-Organizing Data Analysis Techniques) is another algorithm often used for colour space clustering [2]. Connected component labelling methods are used for compute the final segmentation based on the clusters [18]. Clustering-based approaches are useful when the clusters of features are normal and easily distinguishable. If the features are cluttered among objects, this approach can not be guaranteed to give a good segmentation.

Region: Region growing and region splitting-merging are the main procedures in this approach [8, 2, 15]. The region growing method groups pixels or sub-regions into large regions based on pre-defined criteria. An initial set of seed points are created and from these points regions grow if neighbouring pixels have similar properties to that of the seed. Selection of seed points is critical for colour images if a priori information is not available . Hence a set of descriptors based on intensity levels and spatial properties are required. Because the initial seeds are crucial in this approach, the result is highly dependent on them.

Boundary: Edge detection is by far the most common approach for detecting meaningful discontinuities in grey level images [2]. In practice, edge-based techniques using sets of pixels seldom characterise an edge completely due to noise and non-uniform illumination which creates spurious intensity discontinuities. Hence edge detection algorithms need additional post processing by using linking procedures to assemble edge pixels into meaningful edges.

Graphing: The image is modelled as a weighted undirected graph [5]. Each pixel is a node in the graph, and an edge is formed between every pair of pixels. The weight of an edge is a measure of the similarity between the pixels. The image is partitioned into disjoint sets by removing the edges connecting the segments. The optimal partitioning of the graph is the one that minimises the weights of the edges that were removed. Shi's algorithm seeks to minimise the "normalised cut", which is the ratio of the 'cut' to all of the edges in the set [20].

Morphology: One of the more stable techniques, the Watershed transformation considers the gradient magnitude of an image as a topographic surface [17]. Pixels having the highest gradient magnitude intensities (GMIs) correspond to watershed lines (which represent the region boundaries) - water placed on any pixel enclosed by a common watershed line flows downhill to a common local intensity minima (LMI). The method is initialised with markers to avoid over-segmentation due to noise and local gradient irregularities.

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

3. Developer-Centred Segmentation

The goal of this work is to provide segmentation methods to non-experts in an intuitive manner. Our contribution is a developer interface to segmentation based on a description model to allow the developer to specify what the problem is, instead of how to solve it. The description is interpreted to provide an appropriate solution to the problem.

3.1. A Task-Based Description of Segmentation

Due to the complexity of the problem as a whole, we use a relatively simple low-level 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 the type of segmentation. A property may be anything measurable over a region of the image, 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. Developers must specify at least one property to define the segmentation of the image: properties allow decomposition of the image based on what is considered important to their problem, and provides us with the information required to produce a segmentation.

Each property is associated with a *distinctiveness* to allow the developer to define how distinct the segments should be relating to that property. Due to the range of possible methods of segmentation, the term *distinct* was chosen as the abstraction of the conceptual meaning. This was in preference to terms such as *threshold* or *distance* (from region-growing or clustering) which would not be applicable in all cases. The description also allows the specification of multiple properties for a single segmentation. Conceptually this will lead to segments which are distinct based on all specified properties.The advantage of the task-based description is the details of how this is performed are hidden from the developer, and so they do not need to take this into account when developing an application.

When defining the available set of properties we attempt to make sure each is orthogonal to the others, to avoid repetition and encourage completeness. Our eventual goal is to create a unified space for vision descriptions, to apply to all problems, which can be interpreted into algorithms and parameters to provide the developer with a solution. The description space should be kept as small as possible while still maintaining a wide coverage to help minimise the complexity as the description language is extended.

The last property-related aspect of the description is the use of *weights*. These are supplied to let the developer specify which points in property space (e.g. a colour in colour-space or intensity in intensity-space) should approximately form central points of segments. With the addition of weights in the description, the developer can define points in the property space towards which the segmentation should be biased, which is useful in applications such as chroma-keying or skin-colour detection. This is an abstracted form of choosing seed points for segmentation, or feature vectors for clustering.

The properties and weights together form what we define as the *requirements*. The last component we need to complete our description is *constraints*. Constraints introduce some additional complexity to the operation, because they are capable of overriding the distinctiveness requirement. The three constraints we provide are *size*, *quantity* and *regularity*. Size governs the final area of the segments, quantity the number, and regularity the level of variation allowed in the gradient of the segment's contour. Size and quantity are related and must trade off against one another; Regularity constrains the overall shape of the segments: a regularity of 0 does not constrain the shape at all and a value of 1 constrains the shape of every segment to be the same.

3.2. Automatically Interpreting the Description

The interpreter is the first component encountered after the description is passed in through the interface (e.g. API). It is responsible for choosing an appropriate segmentation algorithm based on the image properties, required segment properties (and weights) and the constraints, as well as deriving the parameters for each algorithm automatically.

To ensure a simple 'plug-in' system for algorithms, an internal interface for segmentation is defined which each algorithm must implement; this interface is used by the interpreter to provide the algorithm with the input images and the full user-defined description. The algorithm produces segments in the interface-defined representation, so that all algorithms return the same type to the user.

To add a new algorithm to the set of possible solutions, a number of processes must be defined:

- The conditions under which the algorithm is designed to return reliable results, defined using ranges of values within the segmentation description (which implies a large dimensional volume defined by the description which has sub-volumes occupied by the algorithms).
- The input must be converted from the global framework's format to the format used by the algorithm.
- After execution the segmentation must be checked to ensure it satisfies the constraints; if it does not, post-processing or re-segmentation must be performed.
- The segmentation must be converted into the global framework's format for presentation to the developer.

Defining the conditions for the algorithm based on the description is a non-trivial task: there may be many ranges under which it works well; it may perform best under certain circumstances, but well enough under others; it may not work as well as other algorithms under overlapping conditions and so a ranking system may be required. There is

also the problem of who should define the conditions: the creator of the algorithm could define them (or an expert in the area) or a standardised dataset used within an evaluation framework - we ask the algorithm implementor to define them, in consultation with the literature.

Under our current interpreter, if a developer-defined description has more than one corresponding algorithm available, we choose the 'best'. We use a very simple ranking system based on accuracy and efficiency: if one algorithm has the same efficiency but is more accurate, or more efficient and equally accurate, it is chosen over the others. (This ranking also allows us to define an efficiency versus accuracy tradeoff for the developer to use.) We could also choose to execute more than one algorithm and accept the results of the one which performed best - however measuring this may be challenging.

4. Evaluation of the Task Description

The framework for segmentation descriptions is implemented in C++, with three separate layers. The first is the description layer and provides the developer with the necessary tools to describe the segmentation problem. The second is the interpretation layer: a thin layer which provides the mechanisms to tie algorithms to descriptions and any required pre- or post-processing. The final layer is for algorithms: six different segmentation algorithms are used to cover as wide a spread of the description as possible. Each algorithm provides the interpretation layer with the conditions under which it may operate (based on the description model), derives its own parameters from the supplied description and converts its output into the framework's segment representation. We measure distinctiveness as a real-valued number in the range $[0, 1]$. We also provide various constants to indicate to the interpreter approximate requirements: `Low`, `Medium` and `High`.

4.1. Parameters of Segmentation

Our interpreter supports three segment properties: Colour, Intensity and Texture. We use real-valued RGB to represent colour, a single channel real-value for intensity and a real-valued wavelength for texture. Texture segmentation works only at a particular wavelength each time (defined by the developer), however we are working to expand this to allow a range of wavelengths.

The constraints of Size, Quantity and Regularity are supported: Size and Quantity are approximately satisfied by adjusting the internal distinctiveness. All three constraints are used in the algorithm selection process, with Regularity having the most impact (since regularly shaped segments require an optimisation instead of just region-growing or property-selection). Size is measured in the range $[0, 1]$ relative to the width of the image, or with the hints Small, Medium and Large; Quantity is defined numerically or with the hints Low, Medium and High; Regularity is measured in the range $[0, 1]$ with no regularity at 0, and completely regular segments at 1 (in the ideal case, however this will not always be possible due to the distinctiveness of properties).

(a) Colour, Feature-Space Clustering [2]
(b) Colour, SLIC Super Pixels [1]
(c) Colour, Seeded, Image-Space Region-Growing [19]
(d) Intensity, Seeded, Image-Space Region-Growing [19]
(e) Texture, Seeded, Image-Space Region-Growing [19]

Figure 1. The algorithms implemented within our interpreter and mapped to segmentation descriptions using Table 1.

4.2. Post-Processing

As described in the previous section, post-processing of the output is important to ensure the result satisfies the developer's description and that the result is returned in the framework's (not the algorithm's) format. Our chosen format for current testing purposes is an image with unique colours used as segment identifiers. If multiple properties were requested by the developer, a different algorithm is chosen for each one and the set of results are merged in the post-processing step.

4.3. Segmentation Algorithms

The algorithms implemented within our interpreter are shown in Figure 1. We have only included algorithms which do not require additional input from the developer, such as a marker for watershed-based methods. We do intend to support such methods as an option in the future (as part of a matting abstraction), however they will require additional input types for the description and are not suitable for automatic applications. The region-growing algorithms are extremely slow; if the developer prioritises efficiency with a Colour property, algorithm (a) is given more weight.

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 (such as Table 1 (c-e)) or multiple sets of ranges which define more than one volume (Table 1 (a,b)). The set of mappings from description to algorithm shown in Table 1 are more expansive than would be the case where we had many more algorithms from which to choose. The algorithms are more inclusive to allow for more cases where a solution can be returned. In the cases where a developer supplies a description which is not represented by our interpreter's algorithms (such as Intensity or Texture with a large regularity), we can perform a 'best effort' and provide the closest match, or provide an informative error. The conditions chosen for each algorithm are examples and not meant to be definitive. Choosing the conditions is a difficult problem, and the only real solution is an evaluation of each algorithm under all known condi-

Table 1. The mapping from description to algorithm within our interpreter, using the itemised list of algorithms in Figure 1 for reference. For (a,b), multiple specific cases are presented since other permutations would not be permitted. The other methods satisfy any permutation of their conditions. Note that the enumerations 'High', 'Exact' etc. are used for illustration - the API supports exact configuration.

Algorithm	Properties	Distinctiveness	Size	Quantity	Regularity
(a)	Colour	High	All	All	0
	Colour	All	Large	All	0
(b)	Colour	All	Exact	All	$[0.0, 1.0]$
	Colour	All	All	Exact	$[0.0, 1.0]$
	Colour	All	All	All	$[0.26, 1.0]$
(c)	Colour	Low, Medium	Small, Medium	All	$[0, 0.25]$
(d)	Intensity	All	All	All	$[0, 0.25]$
(e)	Texture	All	All	All	$[0, 0.25]$

tions on a ground-truth dataset. We can then either assign the conditions from this evaluation, or use a learning-based approach to bypass the literal conditions completely - although this would require a differently designed interpreter.

4.4. Results

We provide results from four different descriptions where each activates a different algorithm and provides a demonstrative result. This is intended to prove coverage of a large area of segmentation problems important to developers, and a link to the sophisticated algorithms created by researchers. The results on four different input images for each description are shown in Figure 2. The descriptions in the left column were given to our segmentation interpreter with the four images in the top row. The results are tabulated, demonstrating that the result provided by our framework matches the description given, and that the algorithm chosen based on the description is suitable (illustrating the utility of the algorithm conditions).

5. Conclusions

We have presented a novel description model as an abstraction over segmentation algorithms, designed for use by mainstream developers without expert knowledge in segmentation. The description uses a measure of distinctiveness on properties (such as colour or texture) to define segments, utilising extra information on image properties (e.g. noise, detail and blur) and operational constraints (size, quantity and regularity). Developers use this model to describe their particular segmentation problem and supply it to our interpreter, which will select an appropriate algorithm to provide a result. Our proof-of-concept implementation demonstrates the utility of the description, and the results demonstrate a clear link between description and result.

One disadvantage in the current approach is a discontinuous jump in the results for a continuous change in the description along one dimensions. For example, changing the regularity from 0.25 to 0.26 may initiate a complete change of algorithm. Our future work is to address this issue, provide additional algorithms so that more of the description space is covered, and automatically define the algorithms' working conditions based on a quantitative evaluation.

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. 4

[2] S. T. Bow. *Pattern Recognition and Image Preprocessing*. CRC Press, 2nd edition, January 2002. 2, 4

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

[4] R. Clouard, A. Elmoataz, C. Porquet, and M. Revenu. Borg: A knowledge-based system for automatic generation of image processing programs. *Transactions on Pattern Analysis and Machine Intelligence*, 21:128–144, February 1999. 2

[5] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59:167–181, September 2004. 2

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

[7] 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

[8] L. Lucchese and S. K. Mitra. Advances in color image segmentation. In *Proceedings of Global Telecommunications Conference*, pages 2038–2044, Berkeley, Calif, USA, December 1999. IEEE Computer Society. 2

Figure 2. The results of segmentation on four images (first row) based on the descriptions (first column). The interpreter analyses the description and chooses an appropriate algorithm (final column) based on the mappings in Table 1; parameters are derived for the algorithm based on the description. The descriptions are the basis of our abstraction over segmentation: it can be seen in the result images that the segmentations match the descriptions provided, thus validating our claim that the description-based abstraction can be of use to developers.

[9] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, Berkeley, Calif, USA, February 1967. University of California Press. 2

[10] 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

[11] 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

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

[13] N. Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man and Cybernetics*, 9:62–66, January 1979. 2

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

[15] T. Pavlidis and Y. T. Liow. Integrating region growing and edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12:225–233, May 1990. 2

[16] S. Raut, M. Raghuvanshi, R. Dharaskar, and A. Raut. Image segmentation : A state-of-art survey for prediction. In *Proceedings of International Conference on Advanced Computer Control*, pages 420–424, New York, New York, U.S.A., January 2009. IEEE Computer Society. 2

[17] J. B. T. M. Roerdink and A. Meijster. The watershed transform: definitions, algorithms and parallelization strategies. *Fundamenta Informaticae-Special issue on mathematical morphology*, 41:187–228, January 2000. 2

[18] H. Samet and M. Tamminen. Efficient component labeling of images of arbitrary dimension represented by linear bintrees. *Transactions on Pattern Analysis and Machine Intelligence*, 10:579–586, July 1988. 2

[19] 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. 2, 4

[20] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:888–905, August 2000. 2

[21] 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

[22] W. Skarbek and A. Koschan. Colour image segmentation - a survey. Technical report, Institute for Technical Informatics, Technical University of Berlin, October 1994. 2