OpenVL: Abstracting Vision Tasks Using a Segment-Based Language Model

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

***Abstract*—Computer vision is a complex field which can be challenging for those outside the community to apply in the real world. In this paper we show one method to provide access to sophisticated computer vision methods to general developers, hobbyists or researchers outside the field. Our contribution is an abstraction utilising fundamental vision operations based on a single unit, the *segment*, can be used to describe images, local image conditions and between-image conditions. We illustrate how a descriptive language model can be built on the segment to provide an intuitive mental model of computer vision to mainstream developers. We demonstrate how we can map a description of the task composed of the segment-based language into the space of algorithms, to choose an appropriate method to solve the problem. We use the problems of segmentation, correspondence and image registration to show how end-to-end problems may be constructed using our novel metaphor.**

***Keywords*-Vision abstraction; applications; software tools**

I. 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 developers with access to these sophisticated techniques; effective use of these methods requires extensive knowledge of how algorithms work and how their parameters affect the results. We have created *OpenVL* (Open Vision Language) [1] to provide an interface to access vision solutions at a higher level than is currently available within existing frameworks.

The contribution of this paper is a step towards a general abstraction layer over computer vision, through the introduction of a mental model based on the concept of *segments* tied into a descriptive language model (OpenVL). The language consists of operations we consider fundamental to high-level vision problems, such as `Segment` and `Match`; this is based on prior work in which we argued that the decomposition of vision into axioms can allow the description of higher-level problems [2]. We subsequently demonstrated how to define vision operations for matching, detection and optimisation such that a developer could describe basic vision problems [3]. We use those operations as a basis for this work, however they have been extended so that they all use a unified basis for description: *segments*. We also add an operation for segmentation to the existing set, providing the base from which the other operations function. The combination of both operations and segments allows users to characterise the contents of images and describe how to process them, and subsequently an interpreter can analyse the image description and automatically select an appropriate method to provide a solution.

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[4], 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 called *OpenVX* to accelerate vision methods[1]. Khronos are proposing this act as a layer beneath libraries such as OpenCV [5] 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.

II. Previous Work

Many attempts have been made to develop computer vision or image processing frameworks that support rapid development of vision applications. Image understanding

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

systems attempted to make use of developments in artificial intelligence to automate much of the vision pipeline [6], [7], [8]. The Image Understanding Environment project (IUE) [9] 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 [10] 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 [12], 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.

There are many openly available computer vision libraries that provide common vision functionality, such as OpenCV [5], 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. Miller and Fels [14] discuss a set of guidelines for API designers to follow when creating a computer vision API targeted towards mainstream developers or others without significant expertise in vision; we follow these guidelines for the approach presented in this work.

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 [15] 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.* [16] 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 [17] 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* [18] 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 visualisations 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.

III. Abstraction Through Segments

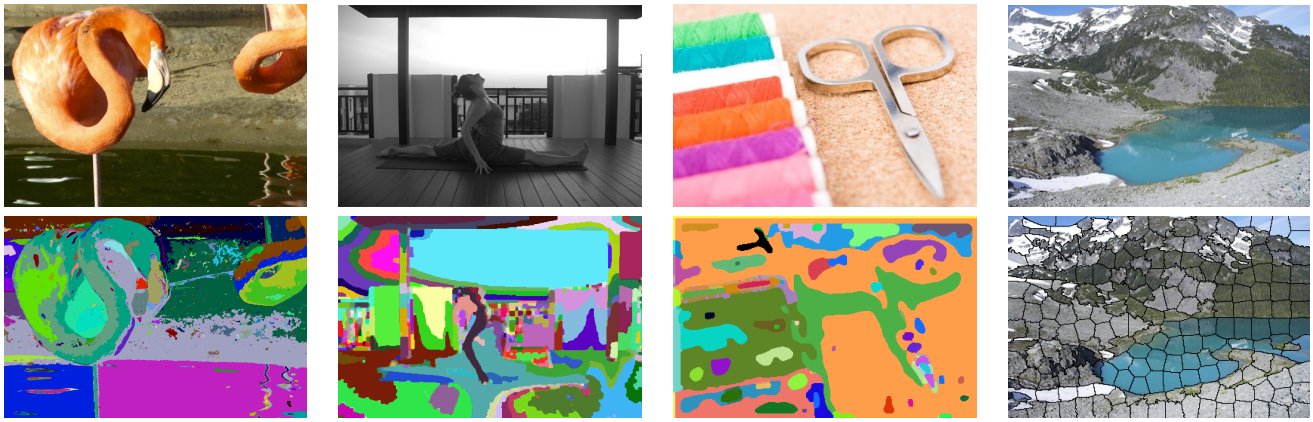
We have designed an interface through which a user describes the conditions of the task and our novel method interprets the description and provides a solution to the problem. We have previously found that asking developers to list the conditions of their problem was not intuitive and led to confusion when applying the abstraction (although this was much less confusing than asking them to use algorithms directly). Instead, we have employed a basis for image-level description combined with fundamental operations to guide the developer through the description.

[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



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

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).

We introduce the *segment* as a metaphor to enable users to describe their problem and access vision methods. Our definition is simple: a segment is a distinct 2D region in the image. We apply *properties* to provide 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, etc. Figure 1 demonstrates example segmentations based on colour, intensity and texture properties. All segments have a *region* property, which represents the area of the image covered by the segment. The region is bounded by a smooth, continuous contour, and is not dependent on pixels or any other discrete representation. Segments must have at least one other property to define their distinctiveness and generate the region, however there is no limit on how many may be defined. Segment properties allow developers to conceptually decompose the image based on what they consider to be important to their problem. These form local image properties and provide our interpreter with a description of the variances within the image, which is important for selection of an algorithm and tuning its parameters.

The properties form what we define as the *requirements* of segmentation. The other component we need to complete our description is *constraints*. Constraints introduce some complexity to the operation, because they are capable of overriding distinctiveness. 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. Size and quantity are clearly related and must trade off against one another; in the implementation we have a requirement that only one may be set, although other implementations may choose another solution to this problem. 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 also provide us with a mechanism to describe variances in the input images. After developers conceptually create segments with a set of properties, all segments have some measure of this property. Since these properties vary from segment to segment (otherwise they wouldn't be distinct) we provide an interface to describe how the segments vary. The variances can be described within images, across images, or both. The ability to vary properties is important for problems such as correspondence, where the segments may vary in some property across images e.g. intensity in a high-dynamic range registration.

The final set of descriptions is of the image as a whole, which can help internally with image filtering or preparation, and also with algorithm selection. Many of the properties may come from associated metadata (e.g. EXIF), but also directly from the user by describing elements such as the scene illumination, noise level or amount of detail.

For our segment-based description, we attempt to make sure each element is orthogonal to the others, to avoid repetition in the description space and encourage completeness. (Higher-level abstractions can be created using this base description, which contains repetition and provides developers with multiple paths to describing the same idea.) Our 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 user with a solution.

Segments are used for two purposes: the first as the representation if the developer requests an image segmentation (or if the result of additional operations is directly related to image regions); the second, and more important, is as the

unit used to facilitate description of the contents of images, and as the basis for comparisons. 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 matching.

IV. Abstracting Tasks with Segment-Based Operations

We have discussed how segments may be used to describe the contents of images and the differences between them; we will continue by covering our approaches for solving the problems of segmentation, correspondence searches and 2D image registration. Our abstraction consists of two components. First, an orthogonal description of the problem allows researchers to identify uniquely within a particular domain the specific conditions of their problem. This orthogonal description makes the conditions of the problem explicit, forcing researchers to think directly about the context within which the problem must be solved; an aspect which dramatically affects algorithm performance and parameter settings. This leads to our second component, a mapping of how different description parameters affect the solution. This can be provided as an expert system which uses rules, provided by algorithm developers or vision experts, to select algorithms and parameters given the problem description. The interpreted mapping could also be accomplished through a direct evaluation of the performance of algorithms under different conditions, or through analytic methods on the algorithms.

A. Segmentation

Segmentation is simply performed using the operation `Segment`; it is the first operation performed for all problems (since every other operation is based on segments). `Segment` takes the image description as previously discussed and the interpreter uses it to select an appropriate segmentation algorithm. This is the only time when the description provided by the developer will be directly applied to compute the final result - although the other operations are based on conceptual segments as input, an actual segmentation is not necessarily performed, since the segments are used as the metaphor for developers to describe their problem. For example, segmentation would not necessarily help algorithmically to solve an image registration problem.

To perform the segmentation, we define the properties used to measure distinctiveness (such as colour), the requirements (such as size) and the interpreter selects an appropriate algorithm to segment the images. Example results are shown in Figure 1: the algorithms applied were (a) intensity-based seeded image-space region growing[19], (b) colour-based feature-space clustering and labelling [20]; (c) texture-based seeded image-space region growing [19]; and (d) colour-based SLIC super pixels [21].

The use of segments as a metaphor for the interface leads to natural mental visualisation as well as a literal visualisation (as shown in the examples). This may lead to intuitive methods for debugging vision solutions which has previously not been possible with computer vision frameworks.

B. Correspondence

One of the fundamental vision tasks, *correspondence* is provided to the developer via the operation sequence `Segment` $\rightarrow$ `Match`. The metaphor for the developer is that correspondence is the search for matching segments within or across images.

A segmentation description is provided, as before, and provided to the interpreter via the `Segment` operation. 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 internally leads to a search for more matches (by adjusting matching algorithm parameters).

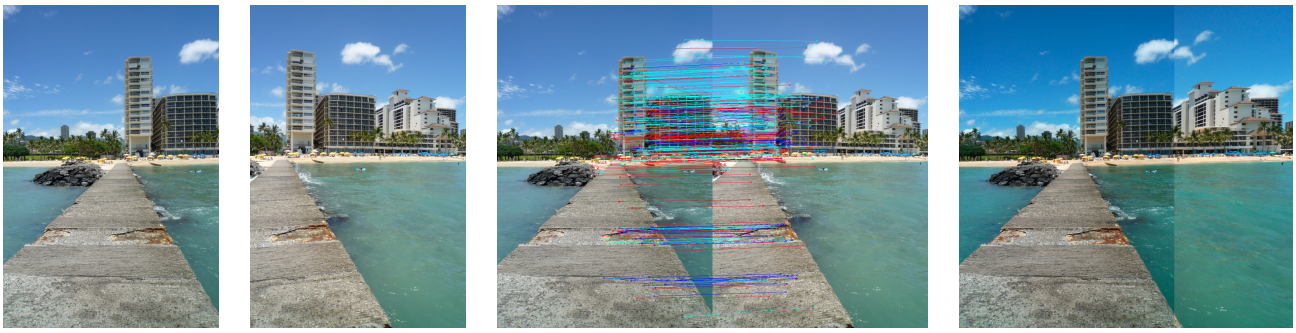
The developer provides variances to describe how much matching segments' position and orientation varies and if the properties will change (such as intensity or blur). Constraints define how strong the match should be. An example correspondence is presented in Figure 2(c), with strong matches highlighted with colourful lines.

Importantly, if the developer only requests a correspondence then a segmentation is never performed, but the description is used to help choose the best algorithm for the correspondence problem. As far as the developer is concerned, a segmentation was performed but the results were not returned. Had the developer requested access to the segments as well as the matches, then a segmentation and correspondence would be performed, the results linked together and returned to the developer.

We apply various feature matching algorithms and adjust their parameters based on the description or eliminate features if they do not apply given the developer's description of the conditions. For example, one of the variances we have for segmentation is *tilt distortion*: if this is specified as high, then the interpreter will give more weight to results from matches found by SIFT, compared to SURF. More details on the abstraction for correspondence are covered in the next section discussing image registration.

C. Image Registration

Image registration is the problem of finding transformations which will align multiple images together in as seamless a manner as possible. Figure 3 introduces three common image registration problems (stitching, focal stacks, and high-dynamic-range stacks) and presents the solution provided by the algorithm selected to align the images by our interpreter. Each of these problems have different conditions



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

Figure 2. 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.

which must be expressible with our segment-based description. Since image registration relies on matching between images, finding corresponding segments is the conceptual issue for the description and so the segment variances are important.

Our framework supports registration as an extension of correspondence: the operation sequence is `Segment` $\rightarrow$ `Match` $\rightarrow$ `Solve(Image)`. The developer supplies segment descriptions and variances just as in the correspondence problem, and additionally provides a description of the optimisation to perform via `Solve`. This includes a description of the required result (e.g. an affine transform); for registration, an image-level affine transform is usually requested based on the correspondences. This will *conceptually* initiate an optimisation over the correspondences between segments to find a single transform for every image. Again, if the segments or correspondences are not requested, these operations will not necessarily be performed since we have an efficient image registration algorithm available to the interpreter. This can lead to a much more highly optimised framework than if all tasks were done literally as described.

Table I introduces our orthogonal descriptor of image registration conditions. Using the descriptor to specify properties of the images provides clues to the type of registration problem the user is attempting to solve; the description can be interpreted to select an appropriate method and a solution found. For example by specifying intensity as a variance we can identify the input as a set of high-dynamic-range images. Similarly by specifying the amount of overlap we can distinguish between a typical stitching problem and a stacking problem. Further, we can begin to express problems which contain multiple forms of variation not seen within the common classes of problem. The descriptor forms an N-dimensional problem space, where individual problems are points, and problem classes are volumes. Table II presents three problem points which are intersected with the volumes defined by each method's capabilities, and the appropriate method is chosen; the method is also shown in Table II and the result of the registration shown in the third column of Figure 3.

The values in the tables are relative for position (in normalised device coordinates), absolute degrees for rotation and relative size of kernel for blur. One important concept in registration is the amount of suspected overlap between images. This can be represented by the variance of the segment position: a variance of 0.5 in width means the segment may be up to half an image width away from its original position, therefore the overlap is up to half an image width. Also, we express relative values of intensity in exposure value (EV) units, a logarithmic scale where an increase of one EV converts to double the measured illumination. Conversion to segment-based description is shown in Table I, and used for the problem description in Table II. We discuss the registration methods using overlap and exposure values to simplify the discussion.

Method selection based on this descriptor relies on knowledge of the limitations and performance of each algorithm. Significant analysis is required in order to understand where an algorithm will perform well, and where it will fail. It is the expert knowledge of which algorithm to use under different conditions which we encapsulate within our interpreter. Using current vision frameworks anyone wishing to use image registration must perform this evaluation themselves or check the literature (which can be challenging for developers outside of academia). This is additionally challenging to do purely based upon the reported findings within the literature because there has been no full evaluation of registration methods on the same data; therefore, as a secondary contribution, we have performed an investigation for four algorithms which cover a range of common image registration problems. By mapping each algorithm's performance into our description we can provide the interpreter with a guideline of when it is appropriate to select these algorithms. The description of the problem can also be

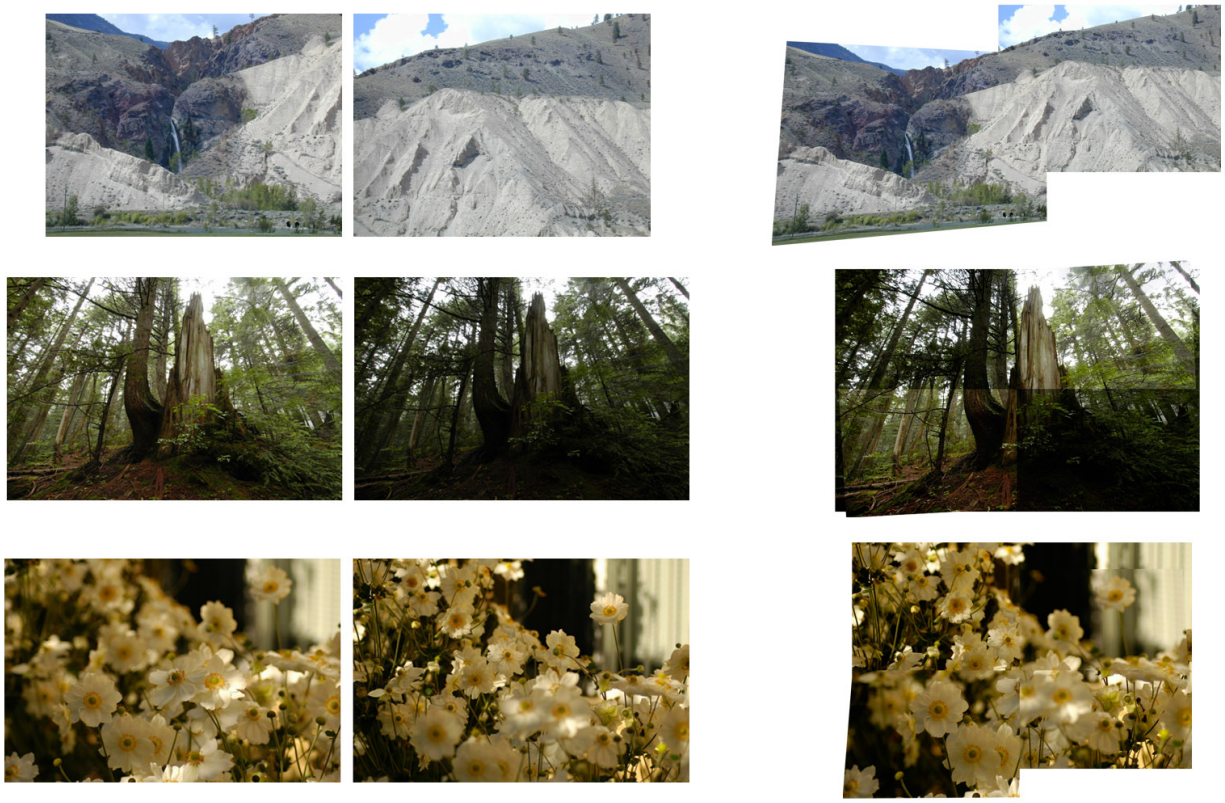


Figure 3. Example image pairs aligned using our task-centred methodology. Table II presents the description parameters specified and the method subsequently selected based on the description to perform the alignment.

Table I
PARAMETERS OF THE DESCRIPTOR FOR THE CONDITIONS OF IMAGE REGISTRATION. THE ORTHOGONAL REPRESENTATION IS PRESENTED IN THE TOP SECTION, WITH IMAGE AND SEGMENT PROPERTIES AND ASSOCIATED SEGMENT VARIANCES. THE BOTTOM SECTION PRESENTS CONCEPTS OF THE REGISTRATION PROBLEM EXPRESSED IN SEGMENT TERMS.

Type	Properties
Image	Noise, Detail
Segment	Colour, Texture, Intensity, Blur
Segment Variance	Position, Rotation, ¡all properties of Segment¿
Overlap	Equivalent to segment position/rotation variance
Difference in Exposure Value	Equivalent to segment intensity variance
Focal Overlap	Inversely proportional to segment blur variance

used to set appropriate parameters within a given algorithm following a similar approach. These four algorithms were specifically chosen because they should be impacted by different descriptor parameters and have been described in the literature as each solving a subset of problems that do not overlap.

For our first algorithm, we examined a gradient descent intensity-based forwards additive method developed by Lucas and Kanade [23], which minimises the square error between aligned image intensities. This method is described as working well for similar intensity images, and unlike most methods should be invariant to changes in focus. It does not perform well when there is little overlap between images, limiting its use when overlap drops below 60%. A direct comparison on a range of image pairs reveals that it does not align images as accurately as the feature-based method, but that for image pairs which contain a limited focal overlap, corresponding within our description to an overlap of identical focus of 60% or less this is the appropriate method. In instances where there is low focal

Table II
DESCRIPTION OF THE CONDITIONS FOR OUR THREE EXAMPLE PROBLEMS SHOWN IN FIGURE 3; THE LAST COLUMN PRESENTS THE METHOD CHOSEN BY OUR TO INTERPRETER TO ALIGN THE IMAGE PAIR. THE RESULT OF THIS ALIGNMENT IS SHOWN IN THE THIRD COLUMN IN FIGURE 3. (NR = NOT REQUIRED.)

Task	Image		Segment	Segment Variances	Method
	Noise	Detail			
Fig 3, Row 1	Low	High	Intensity, Texture	Position(0.25)	[22]
Fig 3, Row 2	High	High	Colour, Texture	Intensity(1EV), Position(0.1), Rotation(15°)	[22]
Fig 3, Row 3	Medium	Low	Colour, Blur	Blur(0.3), Position(0.1), Rotation(15°)	[23]

overlap and also low position overlap the algorithm settings could be modified so that a multi-scale search is performed. The third image pair in Figure 3 was aligned using this method.

Second, we examined a median-based method developed by Ward *et al.* [24] described as suitable for intensity-varying images which uses a translation only transform to align images. Again this algorithm is designed to work for image stacks, limiting its use to image pairs with at least 80% overlap. It works well in instances of almost complete overlap, where the intensity difference is limited to within 2EV. If a lower degree of overlap were encountered the parameters could be modified to perform an affine search, however in our experience this algorithm is outperformed by the feature based algorithm in almost all cases.

The third algorithm is a feature-based method which utilises Harris corner detection [25], scale invariant feature transforms (SIFT) [26] and random sample and consensus (RANSAC) [27] to solve for alignment. Extensive testing of this algorithm has shown that its performance works across the range of overlap, aligning images with as little as 5% overlap. Surprisingly it is also invariant to a significant amount of exposure variation. For exposure increases of ± 1 the algorithm's success decreases slightly from 85% to 80%. Variance of $\pm$2EV measured a decline to 63%, and drops to 25% at $\pm$3EV. The setting of parameters can further affect performance. If intensity differences are present in the conditions the algorithm could be adjusted to only select matches during the RANSAC stage which occur in all images in the set [28]. We intend to report more detailed findings on the evaluation of registration methods in a future paper.

Finally, Vandewalle *et al.* [29] align images at the sub-pixel with a translation and rotation estimation in the frequency domain. For images with almost complete overlap, and no other forms of variation this algorithm should perform quite well, providing an alignment which outperforms our third algorithm, although this has not been verified. It is worth noting that the volume covered by this algorithm in the problem description space fits entirely within the volume of the previous algorithm. This is an example of a niche algorithm being used to solve a particular set of problem conditions.

V. CONCLUSION

We have presented our formulation for a computer vision task-based abstraction using an interface metaphor utilising fundamental vision operations centred on the segment. Our abstraction is designed to provide non-expert users with an interface to methods which does not expose algorithmic detail. We have demonstrated how the abstraction may be used to describe segmentation, correspondence and image registration. We have outlined various methods used within these fields and demonstrated how they map into our problem description, showing how it is possible for a sufficiently rich description to represent them and in turn present a useable interface to end-users.

Following the methodology in Section IV and the example problems, it is our hope that other computer vision problems may be represented in this way. Representing vision problems through a description, and mapping the different problem conditions which impact algorithm performance, provides researchers and developers who are not experts at specific vision problems with access to sophisticated algorithms, without requiring direct knowledge of the field. We intend to continue developing this model for new problems, expanding the abstraction to cover new details (such as the scene and not just images) and to provide a full proof-of-concept framework for researchers, developers and everyone else to try.

REFERENCES

[1] G. Miller and S. Fels, "Openvl: A task-based abstraction for developer-friendly computer vision," in *Proceedings of the 13th IEEE Workshop on the Applications of Computer Vision (WACV)*, ser. WVM'13. New York City, New York, U.S.A.: IEEE, January 2013, pp. 288–295.

[2] G. Miller, S. Fels, and S. Oldridge, "A conceptual structure for computer vision," in *Proceedings of the 8th Canadian Conference on Computer and Robot Vision*, ser. CRV'11, CIPPRS. New York City, New York, U.S.A.: IEEE, May 2011, pp. 168–174.

[3] G. Miller, S. Oldridge, and S. Fels, "Towards a general abstraction through sequences of conceptual operations," in *Proceedings of the 8th International Conference on Computer Vision Systems (ICVS)*, ser. LNCS. Berlin / Heidelberg, Germany: Springer, September 2011, vol. 6962, pp. 183–192.

[4] 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.

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

[6] 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*, vol. 2, 1985, pp. 908–915.

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

[8] 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*, vol. 21, pp. 128–144, February 1999.

[9] J. Mundy, "The image understanding environment program," *IEEE Expert: Intelligent Systems and Their Applications*, vol. 10, no. 6, pp. 64–73, 1995.

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

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

[12] 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*, ser. PADL '01. London, UK: Springer-Verlag, 2001, pp. 304–321.

[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*, ser. IROS'07. New York City, New York, U.S.A.: IEEE, October/November 2007, invited Presentation.

[14] G. Miller and S. Fels, "Developer-centred interface design for computer vision," in *Proceedings of the 6th International Workshop on Human-Computer Interaction*, ser. ICCV'11. New York City, New York, U.S.A.: IEEE, November 2011, pp. 437–444.

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

[16] 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*. ACM, 2004, pp. 399–406.

[17] 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*, ser. UIST '07. New York, NY, USA: ACM, 2007, pp. 33–42.

[18] 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*, ser. UIST '10. New York, NY, USA: ACM, 2010, pp. 37–46.

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

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

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

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

[23] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision (darpa)," in *Proceedings of the 1981 DARPA Image Understanding Workshop*, April 1981, pp. 121–130.

[24] G. Ward, "Robust image registration for compositing high dynamic range photographs from handheld exposures," *Journal of Graphics Tools*, vol. 8, pp. 17–30, 2003.

[25] C. Harris and M. Stephens, "A combined corner and edge detector," *Alvey Vision Conference*, pp. 147–151, 1988.

[26] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, p. 91, Nov 2004.

[27] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, June 1981.

[28] A. Tomaszewska and R. Mantiuk, "Image registration for multi-exposure high dynamic range image acquisition," in *Proc. Int'l Conf. Central Europe on Computer Graphics, Visualization, and Computer Vision (WSCG)*, 2007.

[29] P. Vandewalle, S. Süsstrunk, and M. Vetterli, "A frequency domain approach to registration of aliased images with application to super-resolution," *EURASIP J. Appl. Signal Process.*, vol. 2006, pp. 233–233, January 2006.