Towards a General Abstraction Through Sequences of Conceptual Operations

Gregor Miller, Steve Oldridge and Sidney Fels

Human Communication Technologies Laboratory
University of British Columbia
Vancouver, Canada V6T1Z4
{gregor,steveo,ssfels}@ece.ubc.ca
http://hct.ece.ubc.ca

Abstract. Computer vision is a complex field which can be challenging for those outside the research community to apply in the real world. To address this we present a novel formulation for the abstraction of computer vision problems above algorithms, as part of our OpenVL framework. We have created a set of fundamental operations which form a basis from which we can build up descriptions of computer vision methods. We use these operations to conceptually define the problem, which we can then map into algorithm space to choose an appropriate method to solve the problem. We provide details on three of our operations, `Match`, `Detect` and `Solve`, and subsequently demonstrate the flexibility of description these three offer us. We describe various vision problems such as image registration and tracking through the sequencing of our operations and discuss how these may be extended to cover a larger range of tasks, which in turn may be used analogously to a graphics shader language.

Keywords: OpenVL, Computer Vision, Abstraction, Language, Vision Shader

1 Introduction

One of the main problems preventing widespread adoption of computer vision is the lack of a formulation that separates the need for knowledge of a concept from knowledge of algorithms. In this paper we present a first step towards an abstraction layer over computer vision problems. We introduce a new abstraction which allows us to describe common vision sub-tasks which we build upon to represent more sophisticated problems.

We work on the basis that algorithms and their parameters are low-level details with which a general user should not be concerned. The development of our abstraction layers is motivated by four reasons: 1) access is subsequently possible by those who are not experts in the field; 2) advances in the state-of-the-art can be incorporated into existing systems without re-implementation; 3) multiple back-end implementations become possible, allowing development of

hardware acceleration or distributed computing; and finally, 4) the abstractions provide a mechanism for general comparison of algorithms, thereby contributing to researchers in the field as well as general users.

This idea has been applied successfully in many other fields, notably the OSI reference model in networking [7] and OpenGL in graphics [20]. We have previously developed abstractions within sub-categories of computer vision, including access to and organisation of cameras [16], transport and distribution of vision tasks [2], and investigating the benefits of separating management and analysis of image data [1]. We also developed a conceptual framework for the effective development of computer vision analysis interfaces [17] and began the development of a vision shader language [18]; this paper presents advances on both of these contributions. Our main contribution is a novel abstraction layer for simpler access to sophisticated computer vision methods, presented in a general and extensible framework developed specifically for computer vision.

OpenCV [3], Matlab and similar frameworks are extremely useful but do not provide a user experience at the level we are proposing. These are libraries consisting of algorithms with complicated parameters. For example, the excellent OpenCV face detector [27] requires a large XML file (the result of extensive training on images of faces) as well as other image-based parameters, and implements a particular solution for a given set of training data (frontal faces only, etc.). Our abstraction is not intended as a replacement for existing libraries, but to complement them by providing a larger audience with access to the sophisticated methods.

Our abstraction is formulated through recognition of the common tasks within computer vision, abstracting these individually as *operations* (Section 3) and then providing a mechanism to define sequences which represent a more sophisticated task (Section 4). We provide detail on three of our operations (`Detect`, `Match`, `Solve`) and demonstrate the flexibility that can be achieved using such a small set.

2 Previous 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 [15, 13, 6]. The Image Understanding Environment project (IUE) [19] 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 [22] 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, which is at a lower-level than we are proposing in this paper.

Visual programming languages that allow the creation of vision applications by connecting components in a data flow structure were another important at-

tempt to simplify vision development [14, 25]. 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 [26, 23], 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 [3, 5, 10, 24, 28]. These have been helpful in providing a base of knowledge from which many vision applications have been developed. These libraries often provide utilities such as camera capture or image conversion as well as suites of algorithms. All of these methods provide vision components and algorithms without a context of how and when they should be applied, and so often require expert vision knowledge.

One previous attempt at overcoming the usability problems associated with image understanding is discussed in the RADIUS project [8], 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 our proposed method, however it trades off power, breadth and flexibility to provide its abstraction. The abstraction we present in this paper is aimed to be extensible enough to provide accessible vision methods across the entire field.

3 Conceptual Operations

Many computer vision problems can be divided into smaller sub-problems and solved by providing solutions to each sub-problem. This applies conceptually as well as algorithmically and so we base our idea of *operations* on this principle. We allow the user to conceptually describe vision tasks by dividing the problem into conceptual sub-tasks, then the description is analysed and a suitable method selected. For example, image registration is typically solved by matching identical regions across images, and globally optimising for the alignment. There are many different methods for each stage of this problem, and some which combine them into a single step [21]. Under our approach, a user would describe the problem as a correspondence search, followed by a global optimisation. Our abstraction framework would then interpret this sequence and select the most appropriate method. This is the main contribution of our work, since the abstraction may select individual algorithms for match and solve, or one which does both, and hides the details from the user. Not only does this lead to simpler access to vision, but also opens the possibility of multiple implementations, by different universities and companies, in both software and hardware.

Our operations use various inputs and outputs, task parameters and constraints, all of which contribute to the problem description. For example, in a correspondence search (abstracted by our `Match` operation) we use constraints to

Example	Search Space	N_D	N_I	Problem Description
(a)	$Image$	1	0	Single match in source image
(b)	$Image$	1	1	Single match in single other image
(c)	$Image$	1	N	Single match in each of N images (excluding source)
(d)	$Image$	K	1	K matches in single other image
(e)	$Image$	K	N	K matches in each of N images
(f)	Set	1	$\|\mathcal{I}\|$	Single match from the set of images
(g)	Set	K	$\|\mathcal{I}\|$	K matches from the set of images
(h)	Set	K	N	K matches from subset of N images
(i)	Set	K	0	K matches in source image

Table 1. Example variable values and the problem conditions described using `Match`. N can easily be substituted for $|\mathcal{I}|$ to apply to all images rather than a subset. However, this will not enable search within the source image: this is only accessible via the explicitly defined cases in (a) and (i).

define the search space, problem parameters to indicate the number of matches (in a given number of images) and variances to indicate the differences across images. The operations are explained in the following section.

3.1 Operations

We have a small suite of operations we currently use to provide solutions for detection, tracking, correspondence, image registration, optical flow, matting and background subtraction. We do not attempt in this paper to provide a complete and finished formulation - this is a piece of on-going work, and our current set is intended to be a proof-of-concept which we will continue to expand upon. There is also a substantial quantity of subtle tweaks and defaults which could be made within an implementation; for this paper we are focussing on the abstraction, and will extend the work in future to define details of a framework implementing the abstraction.

Match: Our `Match` operator is used to extract a set of features $\mathcal{F}$ from a set of images $\mathcal{I}$ (containing $|\mathcal{I}|$ images) and find correspondences among $\mathcal{F}$. For a given feature $f \in \mathcal{F}$ multiple correspondences may be found within $\mathcal{I}$, or even within a single image $I \in \mathcal{I}$. The current problem defines which matches are important so we have developed a set of constraints and parameters to describe which features should be selected.

Problems which include correspondence can be described using three parameters: the number N_M of matches required; the number N_I of images to match across (where $N_I \leq |\mathcal{I}|$); and whether to return N_M matches per image (in which case $N_M N_I$ matches are returned) or for $\mathcal{F}$ (where N_M matches are returned). N_M can be specified as an exact, minimum or maximum requirement. The distinction between per-image and entire-set correspondence allows us to define problems which treat a set of images as a single input (such as image

Example	Search Space	N_D	N_I	Problem Description
(a)	$Image$	1	1	Single detection in single image
(b)	$Image$	1	N	Single detection in each of N images
(c)	$Image$	K	1	K detections in single image
(d)	$Image$	K	N	K detections in each of N images
(e)	Set	1	$\|\mathcal{I}\|$	Single detection from the set of images
(f)	Set	K	$\|\mathcal{I}\|$	K detections from the set of images
(g)	Set	K	N	K detections from subset of N images

Table 2. Example variable values and the problem conditions described using `Detect`. N can easily be substituted for $|\mathcal{I}|$ to apply to all images rather than a subset.

registration), require matches from some but not all images, and require unique matches across the set or the images. We also include an option to allow search within the feature's source image (by default this is not the case) and an option for the trade-off between feature strength against density of search.

An important aspect of the correspondence problem is applying the correct method to account for variances across images. We can allow for spatial variance and constrain the search for a match in other images using some distribution over the surrounding area centred at the current feature. Other appearance-based properties can be defined, such as variance in blur, intensity, scale, colour, etc. which will aid in the selection of an appropriate method to determine correspondence.

`Match` provides an abstraction over correspondence, which can be used as an input to another operation to define a different task: typically it is used in conjunction with `Solve`. In Section 4 we explore this relationship, examining the problems which can be expressed using the two operations together with each set of conditions.

We use the following notation for `Match`:

$$\texttt{Match}\ (Image|Set,\ Exact|Min|Max)\ [N_M, N_I]\ variances,\ images \quad (1)$$

The user can choose between $Image$ and Set for correspondence search, and then $Exact$, Min or Max for the interpretation of N_M. The *variances* are specified as a distribution over a range (e.g. uniform, Gaussian) and the input is the set of *images*. If N_I is zero, the operation will only return matches in the image from which the feature was generated (regardless of search space used). If N_I is one, the operation will return matches from one *other* image from the image where the feature was generated. Table 1 outlines some possible descriptions and corresponding results for our `Match` variables.

Detect: This operation is similar to `Match` except instead of conceptually matching all features to all others, it finds image regions in the set of images which match a user-supplied template. The template may be an example image or a high-level description of a detection problem. It has similar constraints to

`Match` and provides a set of detected image regions which match the provided template. As with matching, a distinction must be made whether the number of detections is in the context of every image or across the set of images.

We use the following notation for `Detect`:

$$\texttt{Detect}\ (Image|Set,\ Exact|Min|Max)\ [N_D, N_I]\ template,\ images \quad (2)$$

Table 2 specifies a few of the different forms of detection which may be expressed using this abstraction. In (b), we specify a per-image search and ask for a single result from a single image: this describes a search for a particular region throughout a set of images, and returning the most likely detection. Example (c) goes one step further and requests a single detection in each image. If this were to be qualified with Min then the result would be at least one detection, however likely or unlikely, from each image. (d) presents an interesting case, where multiple detections are requested in a single image. The user does not choose which image this is: rather the framework decides which image had the best detections and chooses these. From the table it can also be seen that the descriptions in (b) and (f) are equivalent, since we are asking for a single detection from any image (but only one) from the set in (b), and we are asking for a single detection across the set of images in (f).

Solve: The `Solve` operation covers a wide range of functionality representative of optimisation algorithms. Within the context of the computer vision problems which we have so far explored the two solutions which may be solved for are spatial transforms and correspondences. The role of this operation will continue to expand as we abstract more problems and methods (e.g. we are working on the problem of matting, where `Solve` is used to optimise the boundary between two image regions). In both cases the input is a set of correspondences from `Match` or a set of detections from `Detect`. The operation's conceptual task description is slightly different from those previous, because the *type* of output requested is used as part of the description: currently we use the types of *transforms* and *matches*. We also provide a variable N_S to define how many solutions are requested (although sometimes this is not required).

There are two distinct models for solutions returned by `Solve`: *Local* and *Global*, and the meaning is dependent on the current context. If solving for a transform with correspondences, local will return a transform per match (e.g. optical flow) and global will return a transform per image (e.g. image registration). If more than one match per feature is available, N_S is used to determine how many solutions should be returned. This allows the solution operation to take existing matches into account and optimise over these as additional information and provide the best solution. There is no problem type defined for finding a solution using detections as input.

`Solve` may also be used to optimise the number of correspondences by constraining them to produce a subset of correspondences which are more accurate with respect to the task, or to provide the most likely path through a set of

Sequence	Table 1	Output Type	Constraint	Problem
(i)	(c)	Transform	Global	Registration [4]
(ii)	(e)	Transform	Global	Stochastic Registration [9]
(iii)	(b)	Transform	Local	Image differencing
(iv)	(c)	Transform	Local	Optical Flow [12]
(v)	(e)	Transform	Local	Stochastic Optical Flow [11]
(vi)	(e)	Matches	Local	Feature Tracking (local matches as prior)
(vii)	(e)	Matches	Global	Feature Tracking (all matches as prior)

Table 3. Problem types when sequencing a `Solve` with a `Match` operation. Registration and optical flow become the most apparent choices for these scenarios, however with additional abstractions this may lead to structure-from-motion, self-calibration and 3D fusion.

images for a given match/detection (which is a form of tracking for a constraint down to a unique match per image, although this is not very sophisticated).

We use the following notation for `Solve`:

$$\texttt{Solve}\ (Local|Global)\ [N_S]\ (matches|detections)\ (transforms|matches) \quad (3)$$

The solve operation is used in conjunction with other operators. In Section 4 we explore the relationship of the solve operator in conjunction with other operations.

4 Sequencing Operations

Interpreting the sequence of our operations (and their associated inputs, outputs and parameters) to infer the problem and select an appropriate method to solve that problem is one of our contributions. Combining the operators `Match` and `Solve` allows for the description of an even greater range of vision problems. As with our investigation of detection, we have explored the intricacies of each set of options on the vision problem. The flexible nature of our operations also leads to combinations of options which are not associated with specific or well known vision problems. We hope this will lead us to novel solutions to problems which may be solvable with combinations of existing methods, or to provide descriptions of problems which have yet to be investigated.

Table 3 demonstrates the different problem types when sequencing `Match` then `Solve` operations for various parameters, based on the `Match` examples defined in Table 1. The example in Table 3(i) states that given a set of images, find a single correspondence in each of N images for each $f \in I_0$, then globally solve for a single transform per-image: this is a basic image registration. The same formulation with a local solve would produce a set of transforms which provide a measure of optical flow, defined in Table 3(iv). Variations of the parameters allow us to describe image differencing, shown in Table 3(iii); we can also ask for more than one match so that we can optimise for the best match later when more data is available (Table 3(ii) and 3(v)).

When solving for a set of correspondences the constraints placed on the optimisation guide the reduction of or path through correspondences. We may use the set of matches found for a given set of images as the prior for optimising the path over the matches, for tracking, or for pruning the number of matches using the appearance models of the matches as a prior to solve for the best match. For example a set of detected objects with multiple detections per image may be constrained by the last known position and motion model of a previous detection in order to improve detection or to track an object. Similarly a set of features may be constrained to reduce the set of features while maintaining features across the image as we see in adaptive non-maximal suppression for image registration [4]. The examples in Table 3(vi) and 3(vii) are for the case where `Solve` is asked to produce *matches*, in the case where `Match` returns multiple *matches* per feature. The result is an optimisation of the path through the images for each feature; for local, each path is evaluated individually, and for global each path is evaluated with knowledge of the others.

5 Conclusion

We have presented our novel abstraction for various computer vision tasks through our small and flexible set of operations which may be sequenced to infer a larger problem. Our research is in the preliminary stages, investigating the effectiveness of our abstraction for describing various low-level tasks within vision with a view to expanding in the future to encompass successively more sophisticated problems. With the detailed representations of `Match`, `Detect` and `Solve` we have been able to describe correspondence, image registration, optical flow, detection and primitive tracking. After the descriptions have been analysed and the problem inferred, the abstraction may select an appropriate method to solve the user's problem.

This is a small part of a very large problem within computer vision, and we are working to expand the language model, notation and the abstraction to cover more issues, and expand the utility of our OpenVL framework. We are simultaneously creating an implementation of the OpenVL framework which provides the language model coupled with implementations of the vision tasks it abstracts. With this framework we hope to provide computer vision to a much larger audience in an intuitive and accessible manner.

References

1. A. Afrah, G. Miller, and S. Fels. Vision system development through separation of management and processing. In *Workshop on Multimedia Information Processing and Retrieval*. IEEE, December 2009.
2. A. Afrah, G. Miller, D. Parks, M. Finke, and S. Fels. Hive a distributed system for vision processing. In *Proc. 2nd International Conference on Distributed Smart Cameras*, September 2008.
3. G. Bradski and A. Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, Inc., 1st edition, October 2008.

4. M. Brown and D. G. Lowe. Recognising panoramas. *Proceedings of the Ninth IEEE International Conference on Computer Vision*, 2:1218–1225, 16-16 Oct. 2003.
5. Camellia. http://camellia.sourceforge.net/.
6. R. Clouard, A. Elmoataz, C. Porquet, and M. Revenu. Borg: A knowledge-based system for automatic generation of image processing programs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 21:128–144, February 1999.
7. J. D. Day and H. Zimmermann. The OSI reference model. In *Proceedings of the IEEE*, volume 71, pages 1334–1340, 1983.
8. O. Firschein and T. M. Strat. *Radius: Image Understanding For Imagery Intelligence*. Morgan Kaufmann, 1997.
9. A. W. Fitzgibbon. Stochastic rigidity: Image registration for nowhere-static scenes. *Computer Vision, IEEE International Conference on*, 1:662, 2001.
10. Gandalf. http://gandalf-library.sourceforge.net/.
11. S. Gupta, E. N. Gupta, and J. L. Prince. Stochastic formulations of optical flow algorithms under variable brightness conditions. In *In Proceedings of IEEE International Conference on Image Processing, volume III*, pages 484–487, 1995.
12. B. K. Horn and B. G. Schunck. Determining optical flow. *Artificial Intelligence*, 17(1-3):185 – 203, 1981.
13. C. Kohl and J. Mundy. The development of the image understanding environment. In *in Proceedings 1994 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 443–447. IEEE Computer Society Press, 1994.
14. K. Konstantinides and J. R. Rasure. The khoros software development environment for image and signal processing. *IEEE Transactions on Image Processing*, 3:243–252, 1994.
15. 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, San Francisco, CA, USA, 1985. Morgan Kaufmann Publishers Inc.
16. G. Miller and S. Fels. Uniform access to the cameraverse. In *International Conference on Distributed Smart Cameras*. IEEE, September 2010.
17. G. Miller, S. Fels, and S. Oldridge. A conceptual structure for computer vision. In *Conference on Computer and Robot Vision*, May 2011.
18. G. Miller, S. Oldridge, and S. Fels. Towards a computer vision shader language. In *Proceedings of International Conference on Computer Graphics and Interactive Techniques, Poster Session, SIGGRAPH 2011*. ACM, August 2011.
19. J. Mundy. The image understanding environment program. *IEEE Expert: Intelligent Systems and Their Applications*, 10(6):64–73, 1995.
20. J. Neider and T. Davis. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Release 1*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1993.
21. S. Oldridge, G. Miller, and S. Fels. Mapping the problem space of image registration. In *Conference on Computer and Robot Vision*, May 2011.
22. G. Panin. *Model-based Visual Tracking: the OpenTL Framework*. John Wiley and Sons, 1st edition, 2011.
23. J. Peterson, P. Hudak, A. Reid, and G. Hager. Fvision: A declarative language for visual tracking, 2001.
24. A. R. Pope and D. G. Lowe. Vista: A software environment for computer vision research, 1994.
25. Quartz Composer by Apple. http://developer.apple .com/graphicsimaging/quartz/quartzcomposer.html.

26. ShapeLogic. http://www.shapelogic.org.
27. P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, page 511, 2001.
28. VXL. http://vxl.sourceforge.net/.