Developer-Centred Interface Design for Computer Vision

Gregor Miller and Sidney Fels
Human Communication Technologies Laboratory
University of British Columbia
2332 Main Mall, Vancouver, B.C. V6T 1Z4
{gregor,ssfels}@ece.ubc.ca

Abstract

The rise in popularity of products and interfaces which use computer vision has not been matched by a rise in usability of frameworks which present computer vision methods to end-users, hobbyists, general developers or researchers outside the field. This position paper presents a work-in-progress set of design guidelines geared towards developer-centred interfaces in order to help provide computer vision in an intuitive and accessible manner. The guidelines were developed through examination of previous work in computer vision and human-computer interaction, analysis of vision problems and inspiration from successful abstractions in other fields, and are intended as a positive reflection on the current state of computer vision interfaces. Our key guideline states that developer interfaces to computer vision must hide details regarding specific algorithms, and we discuss the implications of frameworks which support this guideline.

1. Introduction

Computer vision has a new and important role on the world technological stage with the advent of cheap cameras and high-performance low-power processors. Examples of its application are available throughout industry from simple face detection on compact cameras to advanced articulated modeling such as that on the Microsoft Kinect™. However, implementation of a simple face detection system requires advanced knowledge of existing algorithms and their parameters, which is beyond the scope of general developers. This has inspired our research theme, *access to computer vision*: we would like computer vision to be open and accessible beyond the confines of academia and computer vision experts. While many algorithms are freely and openly available, we do not consider these to be accessible, due to the knowledge and expertise required to effectively apply these to real-world problems. The direction of research we are pursuing is to create a basis from which access to computer vision techniques can be provided without requiring specialist knowledge.

This paper's contribution is a set of guidelines to help framework architects design accessible and intuitive interfaces to computer vision methods for hobbyists, general developers and researchers who do not specialise in computer vision. The key concept presented in our guidelines is that to provide an accessible and intuitive interface, the details of representation and method should be encapsulated within a task-centred mental model. We also believe that targeting an interface towards developers will lead to many simpler and more intuitive interfaces for end-users, rather than researchers providing a specific interface for end-users.

The development of our guidelines has been motivated by four reasons: 1) a framework satisfying the guidelines should provide access to 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 [8] and OpenGL in graphics [40], but none has yet been successful within computer vision.

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 [21, 17, 5]. The Image Understanding Environment project

(IUE) [28] 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 [31] 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 [18, 35]. 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 [39, 33], 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 [1, 12, 42, 3, 34]. 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, which has previously been shown to lessen the effectiveness in application of the frameworks [20]. All of these methods provide vision components and algorithms without any 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 [11], which employed user-manipulated geometric models of the scene to help guide the choice of image processing algorithms. This operates at a higher-level than the previously mentioned frameworks, however it trades off power, breadth and flexibility to provide its abstraction. The guidelines we present in this paper are aimed to be extensible enough to provide accessible vision methods across the entire field to a large audience.

There has been some design research in the related field of machine learning. Fails and Olsen [10] 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 system is presented at a high enough level for users without experience in computer vision or machine learning to customise a general-purpose classifier; the level of abstraction for this interface is significantly higher than our target, which is to provide developers with an interface to invoke computer vision methods. Additionally, the breadth of possible techniques is limited to classification and similar problems, whereas we would like to have a general purpose vision framework.

A development environment called *Gestalt* [32] was created to support the process of applying machine learning by non-experts. Gestalt was developed on the basis that programming with machine learning is significantly different from traditional programming, and the authors describe one of their key points that general support cannot be achieved by hiding steps in the pipeline. While this may be true for machine learning (and we do not suggest otherwise), we contend that for computer vision the opposite is true: general support cannot be achieved *unless* we hide the detailed steps involved in computer vision methods. However, this may come down to a level of detail issue: the conceptual steps of a vision problem are important, and so the ideas developed in Gestalt may be useful in this case. In general however, we argue that an analysis of the complexities of the problem can yield a description rich enough for developers to use as an interface to manipulate algorithms, without ever explicitly dealing with them.

Klemmer et al. [16] introduced a toolkit targeted towards the creation of tangible input systems, and used some basic computer vision methods to support the use of cameras. The abstraction used is based on finding objects in the view against a known background based on a segmentation and then connected-components analysis. These objects are represented within the API and can be tied to various actions or names, essentially allowing user-based classification at the developer level. The developer is not interacting with computer vision, but with the result of a computer vision routine written by the authors, and is therefore targeting an altogether different audience from the interfaces we are investigating.

Maynes-Aminzade et al. introduced Eyepatch [22], a graphical user interface, with strong 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. The classifiers employed spanned a wide variety of tasks, ranging from feature transforms to motion models to gesture recognition, and allowed users to tie the results of these to application actions. While this work is an important step towards accessible computer vision, the target user is a high-level developer and the range of applications is limited. We are attempting to provide a set of guidelines for a framework as wide-ranging and flexible as OpenCV but with the accessibility and usability of OpenGL.

There have been no attempts to provide guidelines for developer interfaces to computer vision, to our knowledge. While computer vision is extensively used this is usually through the use of a problem-specific library and often expert help. We would like to change this, and introduce new vision frameworks which are accessible by a much larger audience through abstractions based on our guidelines. We have previously discussed a conceptual structure for computer vision which may provide a more accessible framework for users [24]: in conjunction with the guidelines we present here this will allow researchers in HCI (and other fields) and general developers to access sophisticated computer vision methods without requiring expert vision knowledge.

3. Computer Vision Interface Design

We are developing techniques to make computer vision methods more accessible to a larger audience, expanding from the current narrow set of academics and specialists. Through our research in accessible computer vision we realised there are a number of separate issues which must be solved to provide an interface to vision methods for developers. To this end we have formulated a set of guidelines to positively influence the design of developer-centred interfaces for computer vision. The guidelines are not validated through studies; instead they have been collated through examination of previous work in computer vision, analysis of vision problems, inspiration from successful abstractions in other fields and some common sense. Note that whenever we use the term *user* we are targeting a developer.

We shall begin by considering how the input and output should be presented to the developer, followed by a discussion of how measurement schemes should operate and finally how to provide the developer with access to algorithms for particular types of problem. Throughout we endeavour to apply aspects of software engineering (such as encapsulation), human-computer interaction and of course computer vision. The inspiration for the guidelines comes from our experience working within both vision and HCI and a frustration from attempting to use existing computer vision frameworks on a day-to-day basis.

As with any framework the input is an excellent place to start, and one that is often overlooked in terms of design. Many vision algorithms require the input images to be presented in a particular way. For example, some require HSV colour space images for skin-colour [44], greyscale images for intensity-based processing [43], rectified images for depth or disparity calculations [13], etc. This problem is not limited by any means to computer vision: until OpenGL 2.0 [40] images supplied for use as textures were required to have power-of-two dimensions. While there are usually reasons to do with efficiency or flexibility which override design concerns, as an interface to sophisticated elements we believe it is important to remove issues such as algorithm-required image format conversion or resampling from the developer-space and concentrate developer effort on the actual problem they are trying to solve. This leads us to our first developer-centric interface design element:

Guideline 1 *The interface should not require special ordering, manipulation, conversion, filtering or any other preparation of the input from the developer.*

A simple solution to this guideline would be to document a single image format for the interface and only accept images of this type as input. If an algorithm requires a different format, it (or the framework) must perform the conversion. In a similar vein, parameters of algorithms are often presented in units based on the image resolution; e.g. if running a detection in the image, one of the parameters is usually the target's size, and it is requested in pixels. While the idea of size is sensible (to reduce the search space), requiring a size in pixels from the user is counter-intuitive: if we were to have two cameras, identical except for pixel density on the sensor, then the size of the target would be the same relative to the sensor size, but the algorithm would require a different size measured in pixels for each image.

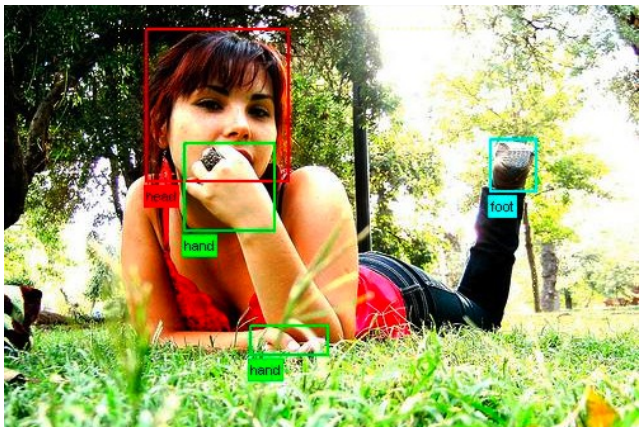
Guideline 2 *Measurements should be independent of image resolution.*

Some interfaces follow this guideline, such as OpenGL texture indexing which operates on the interval $[0, 1]$ (termed *normalised device coordinates*), but even popularly used frameworks such as OpenCV [1] require pixel-based measurements for detection (as shown in Figure 2(c) in the `cvSize` method), and much of the stereo vision literature operates using pixel-based units for comparison windows [37] (which is quite odd given that they are essentially constructing these windows in 3D space, and the size of the 3D window when defined in pixels would vary between cameras [27]). This could be solved using normalised device coordinates although care should be taken to preserve aspect ratio if this is important to the problem.

The idea that measurements not operate on arbitrary scales such as image resolution, because this is not representative of the concept, can be extended to apply to the input and output of a computer vision interface. If the current process requires or produces a particular concept encoded as an image (such as a face, a colour, etc.) then the representation should not include any part of the image which does not correspond to the original concept.

Guideline 3 *Region-based input or output should be concisely representative of the conceptual task.*

An example of this guideline being broken can be found on almost any consumer digital camera: when employing the face detection system for auto-focus etc. the detections



(a) Bounding boxes vaguely represent concepts

(b) Ambiguous definition of concept

(c) Original Image

(d) Subjective segmentation of (c), which directly represents the concept

Figure 1. An illustration of representations used within computer vision: (a) and (b) demonstrate the use of bounding boxes to represent concepts such as *head*, *hand* and *foot*, where each box includes content not representative of the concept. The box marked *head* on the woman in (b) does not include her hair, illustrating an ambiguity in the definition of *head* (since hair is included in (a)). The segmentation in (d) of the image in (c) is an example of a good representation of the concept they encode. However, this form of segmentation is subjective ("what is a bike?") and should be used with care. These images are examples from the test data set of the Visual Object Challenge [9].

are visualised as a box surrounding part of the face. If too large, the boxes include regions of the image which are not a part of the face; if too small, regions of the face are not included (illustrated in Figure 1 (a) and (b)). While this is likely a direct effect of the detection algorithm in use (since most use a "contains-a" rather than an "is-a" classification method [41]) it is technically feasible to further process the output of the detector and produce a region with a closer fit to the target concept (e.g. a skin-colour-based algorithm [15]). There is also a more subtle issue with this guideline: if the task is to represent a person, it is often ambiguous what should be included, as can be seen in Figure 1(b) where not all of the subject's hair is inside the bounding box; for this example we would need to define *head*. Guidelines to specifically deal with this issue are often provided for vision datasets [9] but not algorithms or interfaces (this is addressed further in Guideline 8). We progress the idea of concise representation further to break the commonly-used link between the scene, a smooth and continuous 3D space (ignoring time for now), and the image, an ordered set of bounded 2D regularly discretised samples, since a concise representation is not possible with a discrete basis:

Guideline 4 *Representations of regions within images should be continuous.*

While it may appear counter-intuitive to use a continuous representation when the images we are analysing are discrete, we argue that from the user's perspective the discrete representation is irrelevant: the reason the images are under analysis is to establish some model of the scene, and since

```
(a) Usage: facedetect --cascade=''<cascade_path>'' [filename|camera_index]

(b) cascade = (CvHaarClassifierCascade*)cvLoad( cascade_name, 0, 0, 0 );

(c) CvSeq* faces = cvHaarDetectObjects( img, cascade, storage,
                                        1.1, 2, CV_HAAR_DO_CANNY_PRUNING,
                                        cvSize(40, 40) );
```

Figure 2. Lines of code taken from the OpenCV face detection example. (a) demonstrates the method of execution, requiring a cascade as input; (b) loads the cascade, an XML file over 23000 lines long containing the parameters of the detection algorithm (which is the result of an extensive training process); (c) uses the cascade to detect the objects it has been trained on, in this case vertically-posed unoccluded frontal faces. This example directly breaks Guidelines 2 (uses pixel-based measurement), 3 (result is a rectangle, not representative of the concept), 5 (uses pixel-based representation) and 9 (requires developer use algorithms directly).
Source: http://opencv.willowgarage.com/wiki/FaceDetection/

the scene is continuous it seems logical that the representation should be too. Especially since in the case of computer vision, the result is always a model (never an image) and the model does not need to be discretely represented. The process of converting a model into an image is the domain of computer graphics, and so problems such as matting [4] and image blending [2] are products of both fields.

Establishing Guidelines 2, 3 and 4 has led to one of our most important results in the design of developer-centric interfaces for computer vision:

Guideline 5 *A continuous representation of the input should be used.*

This guideline may be controversial since pixels are well understood by many, due to their presence as essentially an industry standard in image manipulation or photography packages, as well as universally used (and, we argue, abused) in computer vision, but we believe they are not illustrative for the application of computer vision methods. This is partly motivated by the previous three guidelines, but since pixels are area samples of projected light through a lens, there are additional reasons to avoid them: the measured intensity of a single pixel can come from more than one surface through a step-discontinuity, i.e. at the projection of the edge of a surface[45]; the properties of a pixel (colour and position) are usually easier to manipulate when treated as a point source rather than area sample, and in this case the point's colour value is often interpolated from adjacent samples[6]; additional properties such as texture, detail, shape, etc. are representable by regions but not directly by pixels [29]; significant effort has been put into computer vision algorithms for robust interest points [38] and features [19, 7] which are closer to geometric entities and not accurately representable as pixels.

As a short aside, we propose a similar idea for temporal aspects of the interface for problems such as tracking:

Guideline 6 *Representation of time should be continuous.*

Algorithms operating in the spatio-temporal space typically operate using frames as a basis for representation [31]. As an implementation method this utilises all available information but for a user it does not necessarily offer the most intuitive access. Motion is continuous in time as well as space and attempting to describe this discretely suffers from largely the same issues as the spatial case. Particular events observed over an interval (however large or small) rarely align exactly with the instant or interval a particular image was captured, and the result of the analysis may not be frame-aligned either. In the multi-view case, if the shutter release, exposure length and frame rate are not synchronised (or genlocked) then observations will not be equivalent across views which offers further motivation to use a continuous representation.

Returning to the discussion of Guideline 5, we argue that one of the reasons pixels should not be used as a developer-centric representation is their lack of properties which relate to vision problems, such as shape or texture. Typically an algorithm will use some form of representation which favours the conditions under which it operates, and which will have its own set of properties. From the developer's perspective a single representation that either has these properties or can be transformed to produce them is preferable.

Guideline 7 *Image representation should be based on developer-centric semantics.*

We believe that the properties in the developer-space should encode some semantic concept (such as colour or shape) instead of anything algorithm-specific in order to maintain generality and a coherent interface. The representation of the image would then be based on these semantics. This is a familiar concept in computer graphics: the scene model

has properties which are described in a space familiar to everyone, and not in the representation actually employed by the algorithms used to render e.g. the scene has lights which are defined by colour, intensity, position and type (spotlight, directional, etc.) and the mechanics involved to render this property of the scene are hidden from the developer [40]. Additionally, the behaviour of the properties in computer graphics are precisely defined and well-documented, and generally similar across the field. There are many terms in computer vision which vary in meaning across publications and even sub-topics of the field: segmentation can mean a decomposition of the image into similar regions [36], or a subjective clustering of regions into categories of object ('person', 'cat', etc.) [9]; 'object', 'feature' and 'salient points' are a few examples where the intended meaning can vary significantly, since the general meaning is vague.

Guideline 8 *Image representations and computer vision problems should be precisely defined and consistent.*

A computer vision interface should have a single definition of each term used and precise documentation of the intended meaning, process and solution, mainly to avoid confusion but also to document the exact solutions available. Frameworks that provide algorithms often neglect to provide consistent and concise definitions, usually because they are collections of various algorithms contributed by different vision researchers. These frameworks are excellent and useful resources, but they are inaccessible to the general developer, not just because of the previously discussed issues but also due to the complexity of the algorithms themselves. Computer vision algorithms are often difficult to understand and to apply robustly for those not specialised in the field, since expert knowledge is often required to establish the conditions under which each algorithm performs, to configure the specific parameters to correctly process the input and to debug the algorithm when the result is not as expected [32]. There are many useful repositories of computer vision algorithms [1, 31, 42, 3, 12] but they require extensive training and expertise to understand and use effectively. As an example, the excellent face detection algorithm [41] in the OpenCV library [1] is chosen by calling the method `cvHaarDetectObjects` and passing in a parameter variable `cascade` which contains the contents of an XML document over 23000 lines long (the load method is shown in Figure 2(b) and the face detection function is shown in Figure 2(c)). The example program included with OpenCV is executed on the command line, requiring the name of the XML file as shown in Figure 2(a). The XML document contains the result of training a detector with positive and negative examples of faces, and so calling the method with this training data creates a particular solution of the general algorithm to detect faces. There are four other parameters passed in (excluding the image, `img` and some temporary memory, `storage`) to control the detection process. The complexity of using this method to solve the apparently simple problem of detecting a face in an image leads us to our next guideline:

Guideline 9 *Developers should not be required to select specific algorithms, or tune algorithm-specific parameters.*

This is the most important guideline in this paper, and is the central idea which should be used when defining interfaces for access to computer vision. Understanding the details of computer vision algorithms and knowing under which conditions they operate most effectively requires expert knowledge, extensive training and experience. Even within the field, a computer vision expert in 3D reconstruction will not necessarily know the best algorithm to use for articulated tracking, or how to apply it effectively. We argue that developers should not be burdened with the task of choosing which algorithm to solve their problem and learning how to tune the algorithm-specific parameters. Instead, a suitable abstraction layer can be defined to act as the interface to the algorithms. There are various ways to provide an abstraction, for example: through a pipelined set of modular processing blocks [23] which could be presented through an interface much like the one in Lego Mindstorms NXT; through a very high-level abstraction where the developer asks for a particular high-level problem to be solved (e.g. registration, face detection, stereo reconstruction) and the interface automatically selects the algorithm and computes the parameters based on the input images [30, 14]; allow the user to describe the problem conditions (what the images represent, which parts of them are important, how the images differ, some general properties) and use this to infer a suitable algorithm to apply [25, 26]. These abstractions vary in flexibility and power and different versions would be required for different user targets and for different problems. The most effective abstraction may be a combination of these different approaches, with varying levels of detail supporting the level of access required by the user. We believe this is the key to providing computer vision to a much larger audience.

4. Conclusion

We have presented a new set of guidelines which we argue should be satisfied by all developer-based computer vision interfaces. Frameworks designed with our guidelines should provide intuitive and simple access to computer vision methods for hobbyists, general developers and researchers who do not specialise in computer vision. We conclude that the most important guidelines to follow are to avoid pixels as a means for representation (Guideline 5), and to hide algorithmic detail from the developer (Guideline 9). Algorithms are the domain of specialists and we argue that it is possible to apply them effectively through an

abstraction. We believe an abstraction based on a description of the problem space can allow automatic selection of appropriate methods. While we have not determined our guidelines' validity through formal studies we have motivated them through reason, literature review, analysis of existing frameworks and examining successful abstractions in other fields. We intend to evaluate these with a set of abstractions in the near future.

5. Acknowledgements

We gratefully acknowledge the support of NSERC, Bell Canada, GRAND NCE, Vidigami Media Inc. and Avigilon Corporation.

References

[1] G. Bradski and A. Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, Inc., 1st edition, October 2008. 2, 3, 6

[2] 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

[3] Camellia. http://camellia.sourceforge.net/. 2, 6

[4] Y.-Y. Chuang, B. Curless, D. H. Salesin, and R. Szeliski. A bayesian approach to digital matting. In *Proceedings of IEEE CVPR 2001*, volume 2, pages 264–271. IEEE Computer Society, December 2001. 5

[5] 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. 1

[6] W. B. Culbertson, T. Malzbender, and G. Slabaugh. Generalized voxel coloring. In *International Workshop on Vision Algorithms*, Corfu, Greece, 1999. 5

[7] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *In CVPR*, pages 886–893, 2005. 5

[8] J. D. Day and H. Zimmermann. The OSI reference model. In *Proc. of the IEEE*, volume 71, pages 1334–1340, 1983. 1

[9] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2010. http://www.pascal-network .org/challenges/VOC/voc2010/workshop/index.html. 4, 6

[10] J. Fails and D. Olsen. A design tool for camera-based interaction. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '03, pages 449–456, New York, NY, USA, 2003. ACM. 2

[11] O. Firschein and T. M. Strat. *Radius: Image Understanding For Imagery Intelligence*. Morgan Kaufmann, 1997. 2

[12] Gandalf. http://gandalf-library.sourceforge.net/. 2, 6

[13] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004. 3

[14] D. Jang, G. Miller, S. Fels, and S. Oldridge. User oriented language model for face detection. In *Proceedings of International Workshop on Person-Oriented Vision*. IEEE, January 2011. 6

[15] P. Kakumanu, S. Makrogiannis, and N. Bourbakis. A survey of skin-color modeling and detection methods. *Pattern Recognition*, 40(3):1106 – 1122, 2007. 4

[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*, pages 399–406. ACM, 2004. 2

[17] C. Kohl and J. Mundy. The development of the image understanding environment. In *Proc. 1994 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 443–447. IEEE Computer Society Press, 1994. 1

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

[19] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91, Nov 2004. 5

[20] A. Makarenko, A. Brooks, , and T. Kaupp. On the benefits of making robotic software frameworks thin. In *International Conference on Intelligent Robots and Systems*, 2007. 2

[21] 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. 1

[22] D. Maynes-Aminzade, T. Winograd, and T. Igarashi. Eyepatch: prototyping camera-based interaction through examples. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*, UIST '07, pages 33–42, New York, NY, USA, 2007. ACM. 2

[23] G. Miller, A. Afrah, and S. Fels. Rapid vision application development using hive. In *Proc. International Conference on Computer Vision Theory and Applications*, February 2009. 6

[24] G. Miller, S. Fels, and S. Oldridge. A conceptual structure for computer vision. In *Proceedings of Canadian Conference on Computer and Robot Vision*, May 2011. 3

[25] 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. 6

[26] G. Miller, S. Oldridge, and S. Fels. Towards a general abstraction through sequences of conceptual operations. In *Proceedings of International Conference on Vision Systems*. Springer, September 2011. 6

[27] G. Miller, J. Starck, and A. Hilton. Projective surface refinement for free-viewpoint video. In *Proc. Conference on Visual Media Production*. IET, November 2006. 3

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

[29] T. Ojala and M. Pietikinen. Unsupervised texture segmentation using feature distributions. *Pattern Recognition*, 32(3):477 – 486, 1999. 5

[30] S. Oldridge, S. Fels, and G. Miller. Classification of image registration problems using support vector machines. In *Proceedings of Workshop on the Applications of Computer Vision*. IEEE, January 2011. 6

[31] G. Panin. *Model-based Visual Tracking: the OpenTL Framework*. John Wiley and Sons, 1st edition, 2011. 2, 5, 6

[32] K. Patel, N. Bancroft, S. M. Drucker, J. Fogarty, A. J. Ko, and J. Landay. Gestalt: integrated support for implementation and analysis in machine learning. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, UIST '10, pages 37–46, New York, NY, USA, 2010. ACM. 2, 6

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

[34] A. R. Pope and D. G. Lowe. Vista: A software environment for computer vision research, 1994. 2

[35] Quartz Composer by Apple. http://developer.apple .com/graphicsimaging/quartz/quartzcomposer.html. 2

[36] X. Ren and J. Malik. Learning a classification model for segmentation. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 10 –17 vol.1, oct. 2003. 6

[37] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47(1-3), April-June 2002. 3

[38] C. Schmid, R. Mohr, and C. Bauckhage. Evaluation of interest point detectors. *Int. J. Comput. Vision*, 37:151–172, June 2000. 5

[39] ShapeLogic. http://www.shapelogic.org. 2

[40] 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, 3, 6

[41] P. Viola and M. J. Jones. Robust real-time face detection. *Int. J. Comput. Vision*, 57:137–154, May 2004. 4, 6

[42] VXL. http://vxl.sourceforge.net/. 2, 6

[43] I. Wells, W.M., W. Grimson, R. Kikinis, and F. Jolesz. Adaptive segmentation of MRI data. *Medical Imaging, IEEE Transactions on*, 15(4):429 –442, aug 1996. 3

[44] B. Zarit, B. Super, and F. Quek. Comparison of five color models in skin pixel classification. In *Proceedings of International Workshop on Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems*, pages 58 –63, 1999. 3

[45] C. L. Zitnick, S. B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski. High-quality video view interpolation using a layered representation. *ACM Trans. Graph.*, 23:600–608, August 2004. 5