Raydiance: A Tangible Interface for Teaching Computer Vision

Paul Reimer, Alexandra Branzan Albu, and George Tzanetakis

University of Victoria
Victoria, BC, Canada
paulreimer@gmail.com, aalbu@uvic.ca,
gtzan@cs.uvic.ca

Abstract. This paper presents a novel paradigm for prototyping Computer Vision algorithms; this paradigm is suitable for students with very limited programming experience. Raydiance includes a tangible user interface controlled by a spatial arrangement of physical tokens which are detected using computer vision techniques. Constructing an algorithm is accomplished by creating a directed graph of token connections. Data is processed, then propagated from one token to another by using a novel Light Ray metaphor. Our case study shows how Raydiance can be used to construct a computer vision algorithm for a particular task.

Imagine you are an undergraduate student registered in a Computer Vision class. You need to prototype a multi-step computer vision process for your class project. You have limited experience with programming environments such as Matlab and C++. For each processing step, many algorithms are available through the Matlab Image Processing Toolbox and OpenCV[2]. You need to test all these algorithms in order to make an informed choice. You also need to write the software that integrates all selected algorithms into a computer vision system. Each algorithm typically works with several parameters, thus when the complexity of the computer vision task increases, the combinatorial difficulty of selecting the best algorithms and optimizing their parameters may easily grow out of control.

The scenario described above represents a typical bottleneck in project-based undergraduate and even Masters-level Computer Vision classes. This raises the following questions: Can we teach Computer Vision with less emphasis on the low-level programming tasks? Can we teach Computer Vision to students with limited experience in programming?

During the last two decades, significant progress has been made in major areas of computer vision, with numerous robust algorithms being developed for image enhancement, segmentation, motion tracking and object recognition. Implementations of such algorithms are available through the MATLAB Image Processing Toolbox and the OpenCV library[2]. However, the task of integrating existing algorithms into a functional system is not trivial, since one needs to program the glue code to link these algorithms.

G. Bebis et al. (Eds.): ISVC 2011, Part II, LNCS 6939, pp. 259–269, 2011.

This paper proposes a new paradigm called Raydiance to assist novice programmers in the design, testing, and visualization of Computer Vision algorithms. Raydiance includes a tangible user interface controlled by a spatial arrangement of physical tokens which are detected using computer vision techniques. Constructing an algorithm is accomplished by creating a directed graph of token connections. Data is processed, then propagated from one token to another by using a novel Light Ray metaphor. We show how Raydiance can be used to construct a computer vision algorithm for a particular task. Raydiance makes use of image processing techniques in OpenCV[2], and libCVD[1].

The remainder of our paper is structured as follows. Section 1 discusses similar approaches and implementations of visual programming interfaces used for rapid software prototyping, and the foundations of tangible computing interfaces using fiducial markers. Section 2 describes the proposed approach for the design of Raydiance. Section 3 presents a case study which consists of a detection task implemented in Raydiance. Section 4 draws conclusions and outlines future work directions.

1 Related Work

Raydiance is based on a dataflow programming paradigm. Unlike other visual programming environments, Raydiance uses fiducial markers to create a tangible interface which avoids the use of the keyboard and mouse. Concepts of dataflow programming are reviewed in section 1.1. Visual programming environments are discussed in section 1.2. Section 1.3 explains how fiducial markers can be used to implement a tangible computing interface.

1.1 Dataflow Programming

The structuring of computer programs as a sequence of interconnected modules is known as dataflow programming. This approach was proposed by Morrison[12] in the early 1970s. This concept was first used to design, implement and visualize processes involved in processing banking transactions. In addition to the ability to visualize algorithms that have a complex dependency graph, dataflow programming also presents an efficient model for processing data. Kernels operate on blocks of data, and are combined to form a directed graph of data dependencies, often using a visual programming environment. The resulting network can be scheduled to process the data in parallel where there are no data dependences, or to dynamically allocate processing resources to prioritized tasks. The flow-based programming paradigm has seen several variants and many different implementations. Johnston, Hannah and Millar[9] give a history of the transition from fine-grained hardware-focused dataflow programming to more coarse-grained, modular designs. One of the most significant advances in dataflow programming is the emergence of visual programming environments tailored towards building dataflow networks.

1.2 Visual Programming

Visual programming environments present a number of benefits to users: intuitive visualization of control flow, no requirement for mastering a computer language grammar/syntax, and the potential for interactive control of parameters and variations of control flow without the need for making changes in source code. For rapid software prototyping, Zhang, Song and Kong describe the benefits of visual programming environments in [14], while Lomker et al. [11] present a visual programming environment (with elements of dataflow programming) for designing a computer vision algorithm.

1.3 Tangible, Fiducial-Based Interfaces

A tangible interface for controlling a computer describes a setup where affordances are provided by physical components of the interface. This is in contrast to the use of keyboard/mouse driven interfaces which employ the same hardware to control a variety of software. A tangible interface embodies a direct manipulation paradigm. This allows users to physically manipulate a hardware setup, which in turn affects the behaviour of a software application. Tangible interfaces are an emerging trend in computing, and are especially common in interactive, multimedia installations.

Recently, tangible computing interfaces using tokens detected by computer vision techniques–such as the reacTable proposed by Kaltenbrunner, Jorda, and Geiger [10]–have been tailored specifically for controlling multimedia processing algorithms. The shape, translation, and rotation of tokens placed on a planar desktop surface control some aspect of a multimedia processing pipeline. Early versions of these interfaces had an audio focus, to complement the visual process of designing an audio processing interface (e.g. a musical instrument). Tokens designed specifically for detection, classification, and spatial location/orientation are known as fiducial markers.

Fiducal marker detectors and trackers operate by identifying known objects with distinct visual properties. A common choice is a hierarchy of shapes contained within the fiducial design, represented as a region adjacency graph (RAG), described by Costanza et al in [6] [7]. Bencina et al [5] improve on the topological fiducial detector.

We translate the concept of a tangible, fiducial marker-based interface typically used in artistic, multimedia applications to an educational environment used for prototyping computer vision algorithms using dataflow programming. We use a light ray metaphor to automatically establish connections between stages in a computer vision algorithm. The next section details our proposed approach.

2 Proposed Approach

Raydiance represents kernels of computer vision code via tokens. One might think of these tokens as symbolic visual representations of their associated code.

Fig. 1. Apparatus side-view; inset: top-view

Each token represents a distinct processing task, such as thresholding, background subtraction, etc. The tokens are embodied by fiducial markers which are placed on a planar surface within the field of view of a camera. Physical controls for parametric inputs, and visualizations of the output produced by each kernel, are rendered to the display surface located just underneath the token. The connection between kernels of code is performed geometrically, using a novel light ray metaphor(see 2.1). One should note an interesting duality: computer vision controls the functioning of Raydiance, which in turn is used for prototyping computer vision systems.

The current version of the Raydiance uses a planar arrangement of tokens, which are placed on a horizontal surface and filmed with a top-mounted camera as seen in Figure 1. The image plane of the camera is parallel to the planar surface used as the desktop. In the setup shown, the desktop surface and the visualization surface are the same: the desktop surface extends to the corners of a computer screen placed horizontally on a physical desktop, and the camera is aligned to capture all corners of the screen. Figure 1 shows a laptop with the screen fully opened, and fiducial tokens placed directly on the laptop screen. The user interface is designed so that controls for a particular token are drawn directly below the token, and move consistently with the token if the token is displaced.

The horizontal configuration of the display enables the user to view the desktop from any angle and opens the possibility of collaborative interaction among multiple users.

The remainder of this section is structured as follows. Subsection 2.1 discusses the proposed light ray metaphor for token interconnections. Dataflow programming and visualization are discussed in subsection 2.2. Details on fiducial detection and tracking are given in subsection 2.3.

2.1 Light Ray Metaphor

A token-based software prototyping scheme has been proposed before in [14]; this scheme links tokens based on proximity criteria. This approach does not scale well for complex algorithms, since proximity-based connections are limited to 1DOF. Systems such as the reacTable[10] enable a slow, gradual building of audio processing systems, since the placement of each token has a global effect on the entire canvas; reconfiguring certain processing steps requires the repositioning of multiple tokens. For prototyping computer vision systems, more flexibility is desired. That is, one shold be able to add/remove processing steps by displacing as few tokens as possible. This paper proposes therefore a new approach for linking tokens together and reconfiguring them with ease.

We use a light ray metaphor for constructing directed graphs assembled from tokens located on a surface which represents a desktop. Tokens are either connected to, or disconnected from, a graph; a token may be a node in one or zero graphs. Each token that is connected to a graph searches for connections to tokens which will accept as input a similar data structure to that which the token produces. A connection is determined according to an intersection criterion, which for our application is represented by a light ray model. Each output port of the token emits a ray in the plane described by the desktop surface. The 2D spatial location of each token located on the desktop surface is used as the origin point for the ray, and the rotation of the token about the axis normal to the desktop surface, with respect to the coordinate system of the desktop surface is used to determine the direction of the ray.

Many input and output rays may be associated with the same token. Figure 2 shows a simple example usage of "prism" tokens for the decomposition of a colour image into three channnels, followed by the recomposition of two of these channels. Therefore, Raydiance can be customized by choosing offsets for both translation and rotation of each output ray. The translation and rotation offsets are used to separate the outputs from each token; this permits a token to direct each output to multiple distinct tokens, by either varying the translation offset to form parallel rays, or varying the rotation offset to create a fan effect, or any arbitrary combination suitable to the application. A constant translation offset can add contextual meaning to the rays displayed on the visualization screen. For example, this can make it appear as if the rays emanate from image data below the fiducial tokens, rather than in the zero-offset case where rays are

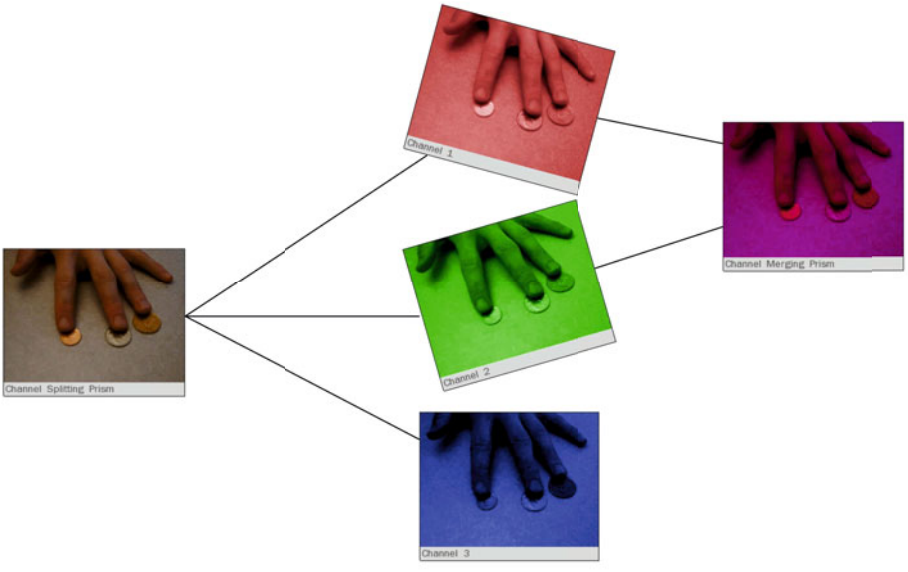

Fig. 2. Multiple output, multiple input "prism" tokens

directly emanating from the token. Figure 2 shows a constant translation offset to the middle of the right-hand side of each token, and a 40-degree rotation offset applied incrementally to each output ray.

Incident rays intersecting the bounds of another token denote a connection between the token that emitted the ray and the incident token. The connection is triggered by a positive result of a line-segment intersection test.

The intersection test is illustrated in Figure 4. Let $R = (R.a, R.b)$ be a ray emanating from a 'radiating' token and m the number of sides (typically $m = 4$) of the token we are considering for intersection. For every side i, $i = 1..m$ we compute the intersection point between the side and the ray R. The green circle indicates the intersection point with minimum distance, the orange circle denotes an alternate valid intersection at a greater distance, and the red circles represent invalid intersection points. The side that provides a valid intersection point located at the shortest distance from the 'radiating' token is selected to establish a connection between the tokens. If no valid intersection points are found, then the two tokens are not connected.

2.2 Dataflow Programming

Interconnecting tokens results into a graph of computer vision kernels. The graph is used to represent an image/video processing algorithm, where each node of the graph represents a series of data transformations. Tokens represent instantiations of a particular type of tranformation. Each token performs an image/video processing task, which can be completed in real-time for 640x480 pixel images at

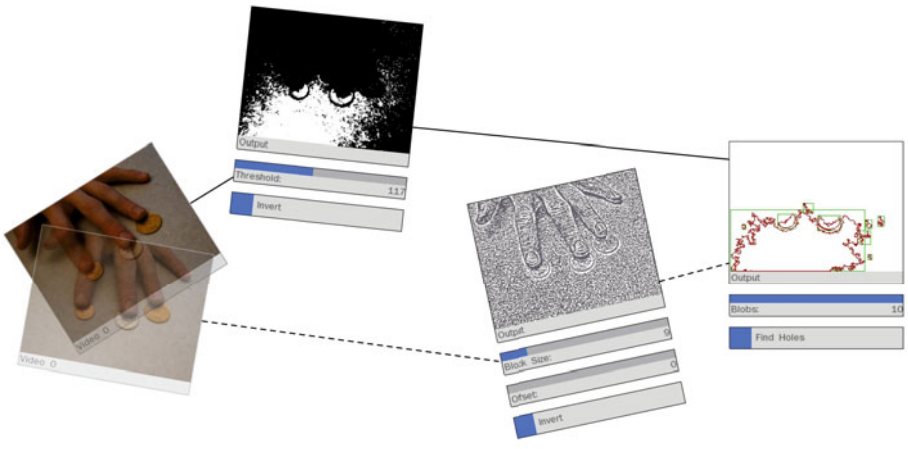

Fig. 3. Using rotation to select from two similar tokens. Dashed lines and translucent images denote an (inactive) alternate processing path. The output of the alternate path is not shown.

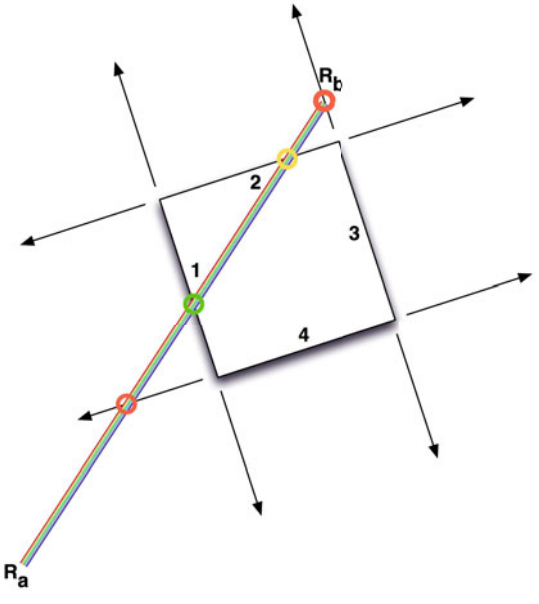

Fig. 4. Using a line-segment intersection test to determine token interconnection

30 frames per second (fps). The input data is a video stream collected from one of multiple attached cameras, clocked at the specified framerate for that camera. Output data is collected from the final node(s) of the graph.

To enable efficient prototyping of computer vision systems, several alternative implementations of common computer vision tasks (e.g. background subtraction, feature extraction) are included in Raydiance. This enables the direct comparison of two (or more) algorithms designed for the same task by comparing their visual output obtained for the same input data. An example of comparison of two thresholding algorithms is shown in Figure 3. Rotating the first token selects between two alternative processing paths.

Data can be visualized at each stage of processing, in the spatial proximity of the token for that processing stage. Parameter values for the specific kernel represented by the token are also shown (see Figure 5).

2.3 Fiducial Detection and Tracking

A fiducial detector based on binary shape detection was chosen to avoid potential issues of colour imbalance due to ambient lighting variations. The graph building framework supports the use of several open-source detectors, and makes it simple to replace these detectors with alternate methods or an improved version of the same detector. The current version of Raydiance uses Libfidtrack[4], the same detector used in reacTable[10]. New fiducials can be generated automatically using the genetic algorithm proposed in [3], implemented in the open-source software Fid.Gen[13].

A tracker maintains a list of detected fiducials, switching among the 'found', 'lost', and 'updated' states for each fiducial depending on the number of consecutive frames in which a fiducial has been detected. A similar tracker is also

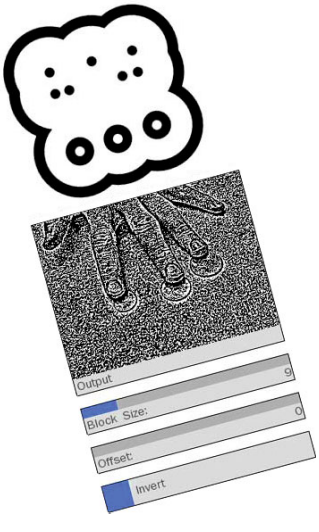

Fig. 5. Rotated fiducial marker and associated visualization

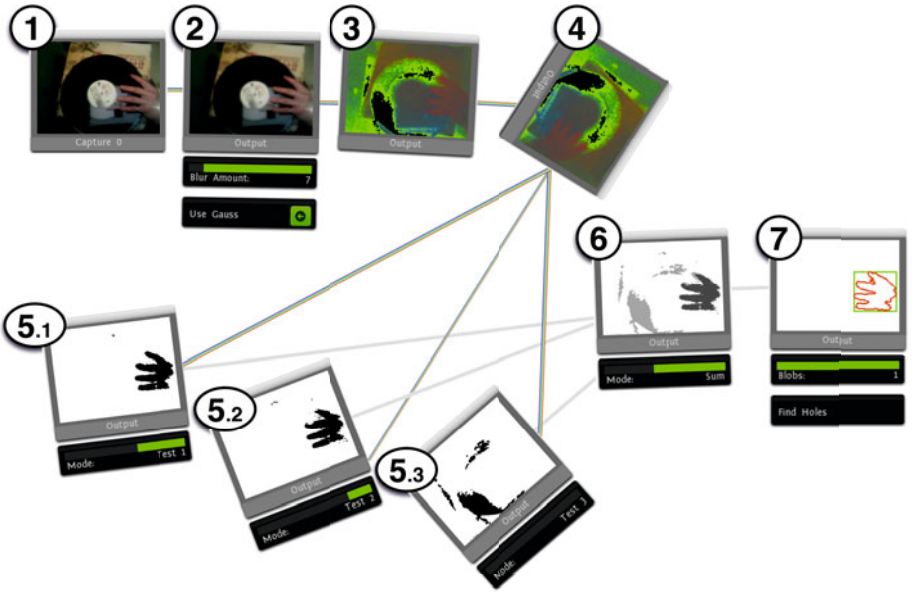

Fig. 6. A Raydiance implementation of an algorithm for hand localization

maintained for the connections between fiducials. Only fiducials present in the fiducial tracker list serve as candidates for ray intersection tests, and these intersections are recomputed with each new video frame.

3 Case Study

This case study considers the task of detecting a human hand within each frame of a video stream from a webcam. This task is sufficiently simple to be suitable for a beginner-level computer vision course project, and it is interesting because of its applicability to real-world applications. Hand detection can be used, for example, to control a computer program using simple gestures made by a waving a hand in front of a camera. To constrain the problem, it is assumed that only one hand is present in a camera frame, and that the bare skin is sufficiently lit to be visible in the video stream recorded by the camera.

A multi-step hand detection algorithm is implemented in Raydiance as follows. Step A (module 2 in Figure 6) Gaussian blur is applied to the RGB image to remove small lighting artifacts and noise. Step B (modules 3-4 in Figure 6) represents a colour space transformation, which is a preprocessing step for colour-based skin detection. This colour space transformation is presented in [8]. Step C (modules 5.1, 5.2, and 5.3 in Figure 6) implements three tests in [8] in order to classify each pixel in the current frame as skin-colored or not. Each test produces a binary mask of pixels. In step D (module 6 in Figure 6) the results

of the three tests are compared and integrated, and the centroid of the hand is computed. The last step (module 7 in Figure 6) computes the bounding box and the contour of the hand.

The hand detection algorithm is prototyped in Raydiance by selecting the appropriate modules and interconnecting them. No additional 'glue code' is necassary. A student with little programming experience benefits from being able to understand how algorithms work by studying their behaviour to different inputs, and by comparing algorithms designed for the same task (i.e the tests for skin detection).

4 Conclusion

This paper presents a novel paradigm for prototyping Computer Vision algorithms which is suitable for students with very limited programming experience. From an educational point of view, this enables decoupling the relatively steep learning curve in learning programming from learning how computer vision algorithms work and behave to different inputs. Therefore, we argue that this paradigm is well-suited for teaching computer vision to freshmen students in engineering and computer science as part of design courses. Moreover, the same paradigm can be used for teaching computer vision for non-technical audiences, such as students in visual arts etc. The technical contribution of the paper consists in a new strategy for interconnecting tokens in a tangible interface via a light ray metaphor. Future work will explore the scalability of this novel approach to more complex computer vision systems and large tabletop displays.

References

1. Cvd projects (2010), `http://mi.eng.cam.ac.uk/~er258/cvd/index.html`
2. Opencv wiki (2010), `http://opencv.willowgarage.com/wiki`
3. Bencina, R., Kaltenbrunner, M.: The design and evolution of fiducials for the reactivision system. In: Proceedings of the 3rd International Conference on Generative Systems in the Electronic Arts (3rd Iteration 2005), Melbourne, Australia (2005)
4. Bencina, R., Kaltenbrunner, M.: libfidtrack fiducial tracking library (2009), `http://reactivision.sourceforge.net/files`
5. Bencina, R., Kaltenbrunner, M., Jorda, S.: Improved topological fiducial tracking in the reactivision system. In: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR) - Workshops. IEEE Computer Society, Washington, DC (2005)
6. Costanza, E., Robinson, J.: A region adjacency tree approach to the detection and design of fiducials. In: Video, Vision and Graphics, pp. 63–69 (2003)
7. Costanza, E., Shelley, S.B., Robinson, J.: Introducing audio d-touch: A tangible user interface for music composition. In: 6th Intl. Conference on Digital Audio Effects, (DAFX-03) (2003)
8. Gomez, G., Morales, E.F.: Automatic feature construction and a simple rule induction algorithm for skin detection. In: Proc. of the ICML Workshop on Machine Learning in Computer Vision, pp. 31–38 (2002)

9. Johnston, W.M., Hanna, J.R.P., Millar, R.J.: Advances in dataflow programming languages. ACM Computer Survey 36(1), 1–34 (2004)
10. Jordà, S., Geiger, G., Alonso, M., Kaltenbrunner, M.: The reactable: Exploring the synergy between live music performance and tabletop tangible interfaces. In: Proceedings Intl. Conf. Tangible and Embedded Interaction, TEI (2007)
11. Lomker, F., Wrede, S., Hanheide, M., Fritsch, J.: Building modular vision systems with a graphical plugin environment. In: International Conference on Computer Vision Systems, p. 2 (2006)
12. Morrison, J.P.: Data responsive modular, interleaved task programming system vol. 13(8) (January 1971)
13. toxmeister. Fid.gen reactivision fiducial generator (2009), `http://code.google.com/p/fidgen`
14. Zhang, K., Song, G.-L., Kong, J.: Rapid software prototyping using visual language techniques. In: IEEE International Workshop on Rapid System Prototyping, pp. 119–126 (2004)