RESEARCH FEATURE

A Practical Paradigm and Platform for Video-Based Human-Computer Interaction

Jason J. Corso, Guangqi Ye, Darius Burschka, and Gregory D. Hager
Johns Hopkins University

New technologies that use multimodal input, human experience, and modern hardware's full computational power could mitigate current limitations in human-computer interaction. The 4D Touchpad, a video-based interaction platform, makes robust, natural interaction between humans and computers possible.

Computing is an integral part of our daily routines. However, despite numerous technological advances, how we interact with computers has changed little in the past three decades. Current systems use conventional input devices, such as a mouse, that limit functionality and often require applications to define a complex, nonintuitive interaction language.

In this new age of human-centered computing, human-computer interaction (HCI) must look to new technologies for building modern, natural, and intuitive interfaces. Video is one such technology. Computer vision techniques could provide far richer interactivity than conventional devices. With the video input signal, systems could use large-scale, unencumbered motion from multiple concurrent users leading to more direct, robust, and effective computing.

Using video in HCI is difficult, as evidenced by the absence of video-based interaction systems in production. Some research systems exist, however, as the "Examples of Video-Based HCI" sidebar describes.[1,2]

The visual interaction cues (VICs) paradigm uses a shared perceptual space between the user and the computer. In the shared space, the computer monitors the environment for sequences of expected user activity at locations corresponding to interface components. Approaching the problem this way removes the need to globally track and model the user. Instead, the system models the sequence of localized visual cues that correspond to the user interacting with various interface elements. The 4D Touchpad is a video-based computing platform based on the VICs paradigm. Together, the VICs paradigm and the 4DT provide a rich new set of techniques that bring the human to the center of HCI design and surpass some restrictions of conventional interface technology.

VICS INTERACTION MODEL

An interaction model is a set of principles, rules, and properties guiding an interface's design.[3] The model describes how to combine interaction techniques both meaningfully and consistently and defines the interaction's look and feel from the user's perspective.

The current windows, icons, menus, and pointers (WIMP)[4] interface technology is a realization of the direct manipulation interaction model.[5] The four principles of direct manipulation are

- continuous representation of the objects of interest;
- physical actions (movement and selection by mouse, joystick, touch screen, and so on) or labeled button presses instead of complex syntax;

Published by the IEEE Computer Society

- rapid, incremental, reversible operations for which the impact on the object of interest is immediately visible; and
- a layered or spiral approach to learning that permits use with minimal knowledge.

The WIMP implementation of the direct interaction model brought proficiency to a broad spectrum of users. However, WIMP's use of a pointing device to map the user to the interface has notable drawbacks:

- It limits the number of active users to one at any given time.
- It restricts the actions a user can perform on an interface component to click and drag.
- Because of the limited set of actions, the user must learn and perform a complex sequence of actions to issue some interface commands.

The restrictive mapping often results in the user manipulating the interface instead of the application object,[3] and greatly restricts the interface's naturalness and intuitiveness. Using video as input removes the need for such mediation. It lets users bring real-world experience to help make intuitive HCI possible. Our approach extends the direct interaction model to better use future interfaces' multimodal nature.

Our model follows four principles.

First, it recognizes two classes of *interface components*: direct and indirect objects.

Direct objects should be continuously viewable to the user and functionally rendered. They should have a real-world counterpart, and their use in the interface should mimic their real-world use. A simple push button is a good example of a direct object: In the real world and the interface, buttons are visually similar and are activated (pushed) in the same way.

Indirect objects, or interface tools and components, might or might not have a real-world counterpart. Indirect objects should be obvious to the user, and a standard interaction language should govern their use. An example of such an interface tool is a corner tab that a user can grab and use to resize the window.

The second principle is *sited interaction*. All physical interaction with the system should be localized to specific areas (or volumes) in the interface to reduce the ambiguity of the user's intention. Generally, sited interaction implies that all interaction involves the interface elements, and the system can monitor the local volume around each element for user gestures. For the button example, the system need not visually track the user's hand around the entire environment. Instead, it can monitor only the local volume around the button, waiting for the user's hand to enter and press.

The third principle is *feedback-reinforced interaction*. Because the interaction is essentially a dialog between the user and the computer system (with little or no mediation), the system must supply continuous feedback to the user throughout the interaction and immediately there-

Examples of Video-Based HCI

Most video-based approaches rely on visual-tracking and template-recognition algorithms as the core technology. PFinder[1] uses a statistics-based segmentation technique to detect and track a human user as a set of connected "blobs." Dmitry Gorodnichy and Gerhard Roth[2] developed an algorithm to track the face (the nose) and map its motion to the cursor. Christian von Hardenberg and Francois Berard[3] developed a simple, real-time finger-finding, tracking, and hand posture recognition algorithm and incorporated it into perceptual user interface settings. ZombiBoard[4] and BrightBoard[5] extend classic 2D point-and-click-style user interfaces to desktop- and blackboard-style interactions. Visual Panel[6] tracks an arbitrary planar surface and a fingertip for use in 2D and 3D environments. The Everywhere Display project[7] uses a custom projector system to render interface components at arbitrary locations in the environment and site-localized image cues for gesture recognition.

References

1. C.R. Wren et al., "Pfinder: Real-Time Tracking of the Human Body," *IEEE Trans. Pattern Analysis and Machine Intelligence* (TPAMI), vol. 19, no. 7, 1997, pp. 780-785.
2. D.O. Gorodnichy and G. Roth, "Nouse 'Use Your Nose as a Mouse'—Perceptual Vision Technology for Hands-Free Games and Interfaces," *Image and Vision Computing*, vol. 22, no. 12, 2004, pp. 931-942.
3. C. von Hardenberg and F. Berard, "Bare-Hand Human-Computer Interaction," *Proc. Workshop Perceptual User Interfaces*, ACM Press, 2001, pp. 113-120.
4. T.P Moran et al., "Design and Technology for Collaborage: Collaborative Collages of Information on Physical Walls," *Proc. ACM Symp. User Interface Software and Technology* (UIST), ACM Press, 1999, pp. 197-206.
5. Q. Stafford-Fraser and P. Robinson, "BrightBoard: A Video-Augmented Environment," *Proc. Conf. Human Factors in Computing Systems* (CHI), ACM Press, 1996, pp. 134-141.
6. Z. Zhang et al., "Visual Panel: Virtual Mouse, Keyboard, and 3D Controller with an Ordinary Piece of Paper," *Proc. Workshop Perceptual User Interfaces*, ACM Press, 2001, pp. 1-8.
7. R. Kjeldsen, A. Levas, and C. Pinhanez, "Dynamically Reconfigurable Vision-Based User Interfaces," *Proc. Int'l Conf. Computer Vision Systems*, LNCS 2626, Springer, 2003, pp. 323-332.

after. For the button example, the system could highlight the button whenever it notices activity on it. This gives the user immediate feedback about the computer's state, and indicates that the user might need to adjust a gesture for successful recognition.

Finally, we separate the learning involved in using the system into two distinct stages.

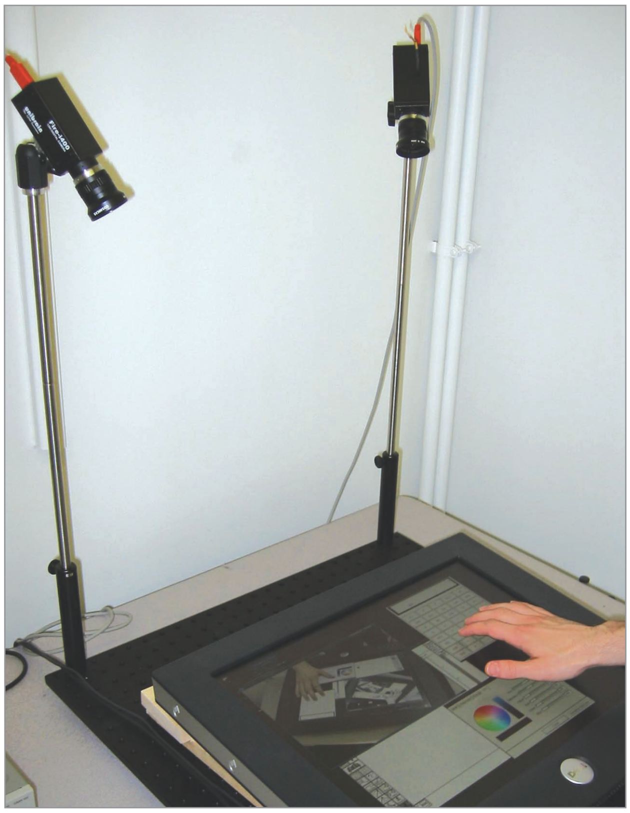
Figure 1. The 4D Touchpad system with a standard flat-panel display.

In the first stage, the user must learn the intuitive set of initial techniques and procedures for interacting with the system. Essentially, new users should be able to apply their real-world experience to immediately begin using the direct interaction objects. For example, a user's knowledge of button pushing in the real world directly transfers to the interface.

In the second stage, duplex learning ensues. In this stage, the system adapts to the user, and the user can learn more complex interaction techniques. We expect that such duplex learning will be an integral part of future interfaces centered on human experience. This learning is necessary, for example, in an interface component, such as a scroll bar, that has no (or weak) translation to the real world. A user might have to physically grasp the scroll bar handle with a thumb and forefinger, then translate it up and down. This is a more complex situation, and the user will have to learn how to properly grasp the handle. However, because human-centered computing is the goal, the computer system should also adapt to the user's grasp-style for a more natural and robust interaction experience.

The VICs interaction model adds constraints to enforce the naturalness of the interaction between the user and a video-equipped computer system. We distinguish between direct and indirect interface objects to avoid the problem of the user manipulating the interface rather than the application objects and to simplify the learning required to use the interface. For example, a common real-world interface object is a circular dial often used to adjust the volume in a stereo system. To use such a dial in the real world, a user grasps it and rotates it. Naturally, the user would expect an interface dial to operate the same way.

4DT: A VICS PLATFORM

Our video-based human-centered computing system, shown in Figure 1, directs a pair of wide baseline cameras at the interaction surface, which is a standard flat-panel display laid flat.

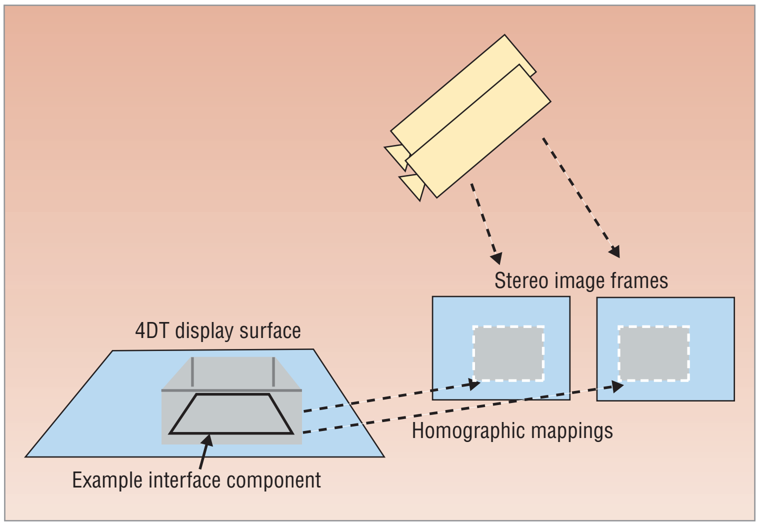

Figure 2. The principle of site interaction for the visual interaction cues (VICs) paradigm.

Geometric calibration

The sited-interaction principle is the motivating factor behind 4DT's geometric calibration procedure. As Figure 2 shows, because all gesturing occurs in an interface component's local neighborhood, or *region of interest*, we must compute a mapping from each region of interest to its projection in the video frames. We call this the *interface component mapping*. The vision system monitors each localized video region of interest and performs gesture recognition using spatio-temporal pattern recognition.

To solve the calibration problem, we use planar homographies. Similar to Rahul Sukthankar and his colleagues,[6] we assume that the cameras' intrinsic and extrinsic parameters are unknown and that we can model the camera and projector optics using perspective projection. We can model the projection of a set of coplanar points by a linear transformation, H, with 8 degrees of freedom (DOF). The coplanar constraint reduces the standard perspective projection model from 11 to 8 DOF. This constrained projection—a planar homography—maps a point on a projective plane $q \in \mathbb{P}^2$ to a point in the image (or model plane) $p \in \mathbb{P}^2$: $p = Hq$.

We compute $H_{i \in \{1,2\}} : \mathbb{P}^2 \rightarrow \mathbb{P}^2$, which rectifies a camera image into model space. Let $\hat{p}_{j \in \{1 \ldots n\}} \in \mathbb{P}^2$ be an imaged point and let $b_j \in \mathbb{P}^2$ be the corresponding model point in rectified space. In our context, we assume that the model is constructed by a set of known points that have been rendered on the flat-panel display surface (a calibration pattern). For each camera, we can write $b_j H_i \hat{p}_j, \ i \in \{1,2\} \wedge j \in \{1 \ldots n\}$.

We recover the two homographies using a least-squares method driven by these corresponding points.[6] After solving for H_1 and H_2, the system is geometrically calibrated. Each homography provides enough information for the interface component mapping, so we can localize each interface component (in model space) to a subregion of the incoming images. In practice, we rectify these images into the model space to simplify the subregion geometry.

Applying the homographies warps both camera images such that all points on the display surface appear at the same position. For this homographic calibration, we can use a simple, region-based stereo calculation to detect contact with the surface, as Figure 3 illustrates. In practice, we perform a rapid, approximate stereo calculation in the volume above the surface for a richer gesture descriptor.

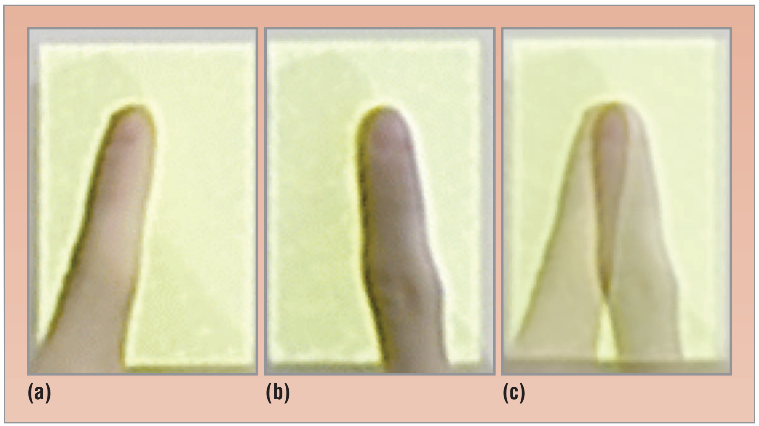

Figure 3. Geometric calibration in 4DT. (a) Rectified subimage 1, (b) rectified subimage 2, and (c) overlaid images of the finger.

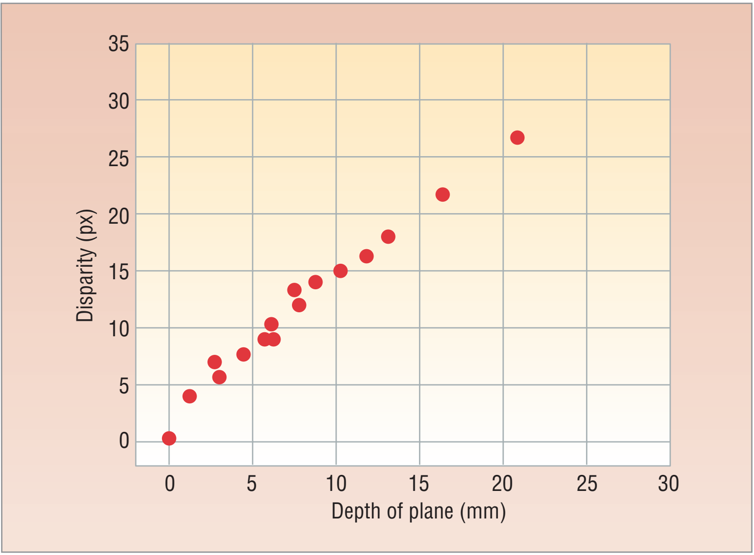

Figure 4. A graph showing the 4DT system's depth resolution.

Figure 4 shows a graph of our system's depth resolution. The high depth discrimination is due to the stereo system's wide baseline, and it greatly helps in recognition of the articulated hand gestures in use.

Color calibration and foreground segmentation

Because the interaction occurs in the volume directly above the interface, and the VICs system is aware of the rendered interface, we can perform background subtraction to segment objects of interest (for example, gesturing hands) in the interaction volume. We directly model the background's appearance by color-calibrating the rendered scene and the images of the scene the cameras capture.

We model the background color using an affine model. We represent the color images from the cameras and the rendered scene in YUV format in which Y represents pixel luminance and U and V carry color information. YUV is efficient: It is the cameras' native format, and converting to a different color space (such as RGB) is computationally wasteful. An affine model represents the transform from the color of a pixel in the rendered scene ($s = [Y_s, U_s, V_s]^T$) to the color of a pixel in the camera image ($c = [Y_c, U_c, V_c]^T$). Using a 3×3 matrix $\mathbf{A}$ and a vector $\mathbf{t}$, we represent this model using the equation $c = \mathbf{A}s + \mathbf{t}$.

We learn the model parameters, $\mathbf{A}$ and $\mathbf{t}$, using a color calibration procedure. We generate a set of N scene patterns of uniform color, $\mathcal{P} \doteq \{P_1 \ldots P_N\}$. To ensure the modeling's accuracy over the entire color space, the colors of the N calibration patterns occupy as much of the color space as possible.

We display each pattern P_i and capture an image sequence S_i of the scene. We compute the corresponding

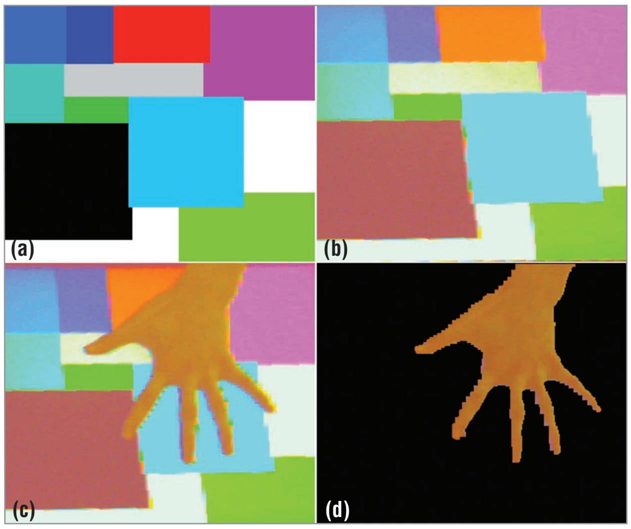

Figure 5. An example of image segmentation based on color calibration. (a) The original pattern rendered on the screen, (b) the geometrically and chromatically transformed image of the rendered pattern, (c) the image actually captured by the camera, and (d) the segmented foreground image.

image C_i as the average of all the images in the sequence S_i. The smoothing process is intended to reduce the imaging noise. For each pair of P_i and C_i, we randomly select M pairs of points from the scene and the images. We construct a linear equation based on these $N \times M$ correspondences and obtain a least-squares solution for the 12 model parameters.

We use image differencing to segment the foreground. Given background image I_B and an input image I_F, a simple way to segment the foreground is to subtract I_B from I_F. We compute the sum of absolute differences for each pixel's color channels. If the SAD is above a certain threshold, we set the pixel to foreground.

Figure 5 shows an example of the segmentation. Because we know that the foreground of interest will be skin color, we include an additional skin color model in the YUV signal's UV color components. We choose a simple linear model in UV space. Basically, we collect skin pixels from segmented hand images and fit the model as a rectangle in the UV plane. The additional skin model improves the foreground segmentation's robustness, as some brief experiments demonstrate.

To examine the segmentation's robustness and stability on our 4DT system, we train the affine color model using 343 unicolor image patterns that are evenly distributed in the color space. To test the learned affine model's accuracy, we display more than 100 randomly generated color images and examine the resulting segmentation. In this case, the ground truth is an image marked completely as background pixels. For both cameras, the system achieves more than 98 percent segmentation accuracy.

To test the linear skin model, we analyzed image sequences containing the hands of more than 10 people. During testing, we asked users to place a hand on the flat panel and keep it still. Next, we rendered a background that is known to perform well for skin segmentation and treated the resulting skin foreground segmentation as the ground truth. We asked users to keep their hand steady while we rendered a sequence of 200 randomly generated patterns. For each image, we counted the number of incorrectly segmented pixels against the true segmentation. The overall skin segmentation accuracy is more than 93 percent.

Gesture recognition

The action of a user gesturing over an interface component presents a sequence of visual cues—a gesture's *spatiotemporal signature*.[1,2,7]

Consider a standard push button with a known interface component mapping for a single color camera. We can decompose the video of the user pushing the button into a set of discrete stages, as Figure 6 shows.

1. The user enters the local region. Visually, there is a notable disturbance in the local region's appearance. A simple thresholded, image-differencing algorithm could detect the disturbance.
2. The finger moves onto the button, presenting itself as a large color blob in the local region.
3. The finger pushes the button. From one camera, it's difficult to reliably detect the pushing (nothing physically changes as a result of the user action), but we can assume the pushing action has a certain, fixed duration.
4. The button is pressed and processing completes.

In practice, manually designing such parsers is tedious and it's generally difficult to guarantee robust recognition. Therefore, we use pattern-recognition techniques to automatically learn a model for each low-level gesture's spatiotemporal signature.

To define the feature space for use in recognition, we use an approximate block-matching stereo calculation procedure in the region of interest, as Figure 7 illustrates. Formally, let I_l and I_r be a pair of rectified images of the scene. We split the images into tiles of equal size $w \times h$. Here, w and h refer to the tile's width and height, respectively. Suppose we consider only a local area of size $m \times n$ patches, starting at patch (x_0, y_0). Given a discrete parallax search range of $[0, (p-1) \times w]$, we can characterize the scene using an $m \times n \times p$ volume V as:

$$V_{x,y,z} = D\left(I_{l(x_0+x,y_0+y)} I_{r(x_0+x+z,y_0+y)}\right)$$
$$x \in [0, m-1], y \in [0, n-1], z \in [0, p-1]$$

where D is some matching function like SAD or normalized cross-correlation. Note that in this equation, the image index indicates a patch of the image, not a particular pixel. Under this scheme, the feature space is relatively high-dimensional (on the order of hundreds). So, we use an unsupervised K-Means clustering algorithm to learn the feature space's underlying structure. We choose K empirically (for example, 20), but minimum description length methods could guide the choice.

Finally, to model the dynamics, we use standard forward hidden Markov models. The input to the model is a cluster index for each frame. The idea is to learn a battery of HMMs, one for each gesture, that we train using the Baum-Welch algorithm. During recognition, we compute the sequence probability against each HMM in the battery. The model showing the highest probability above a threshold wins, and the system triggers the corresponding gesture. The threshold, which is learned from the training data, prevents invalid sequences from always returning some gesture.

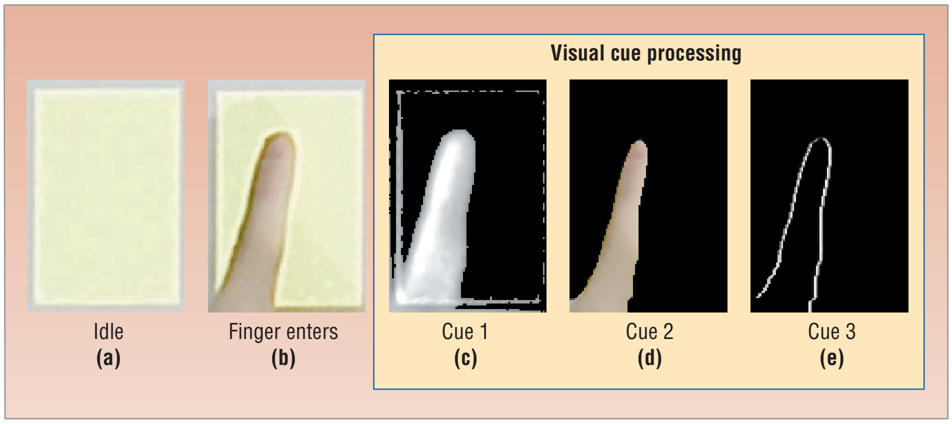

Figure 6. Cue parsing example. (a-b) A finger enters the local region to press a button. A button press is represented as the sequence of three visual cues in the region of interest for an interface component. Independent image-processing modules detect the three cues, which are (c) background disturbance, (d) color, and (e) shape.

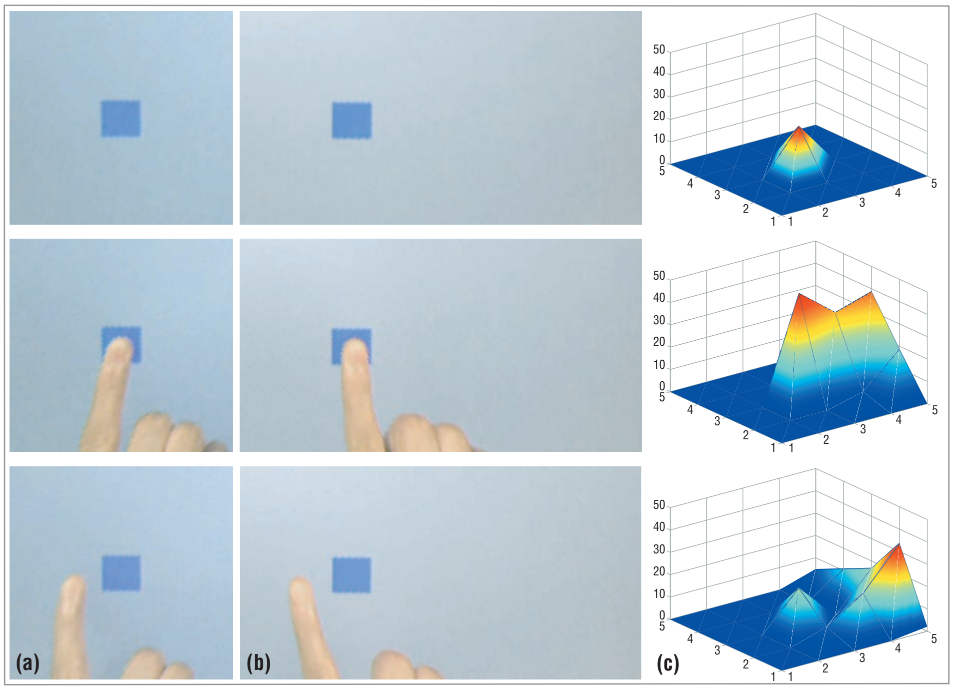

Figure 7. Examples of the image pair and extracted appearance feature. (a) Left and (b) right images of the scene, respectively. (c) The bottom layer of the feature volume (that is, $V_{x,y,z}$ with $z = 0$).

A natural-gesture language

A natural gesture language implements the VICs interaction model on the 4DT platform. The gesture language comprises a vocabulary of individual gestures and a gesture grammar that promotes a natural interaction by grouping individual gesture words into gesture sentences.

Table 1 lists the vocabulary's 14 individual gesture words. Grouping these individual words together into a comprehensive gesture language gives the nine composite gesture sentences in Table 2. We use a probabilistic bigram model to represent the gesture language, and a greedy inference algorithm[1] to perform recognition at interactive rates. The bigram model improves accuracy by constraining the prior distribution on the "next expected word" given the current gesture. The proposed gesture language defines a complete set of interactions that have direct counterparts in the real world.

To map the gestures into conventional windowing interfaces, we simply need to send events to the windowing system for button presses and mouse drags. We map the pushing gesture to the button press event; the system invokes a "button press" when it recog-

Table 1. Individual gesture vocabulary.

Gesture	Description
Push	Finger on center
Press-left	Finger on left edge
Press-right	Finger on right edge
Pick	Index finger and thumb half-closed
Drop	Index finger and thumb open
Grab	Index finger and thumb close together
Stop	An open hand
Silence	No recognizable gesture
Twist	Clockwise twisting
Twist-anti	Counterclockwise twisting
Flip	Mimics flipping a coin over
Move	Tracks two translational DOF
Rotate	Tracks one rotational DOF
Resize	Tracks two DOF of scale

Table 2. An intuitive set of gesture sentences.

Gesture	Sentence
Pushing	Push → Silence
Twisting	Press-right → Twist → Silence
Twisting-anti	Press-left → Twist-anti → Silence
Dropping	Pick → Drop → Silence
Flipping	Pick → Flip → Silence
Moving	Pick → Move → Drop → Silence
Rotating	Pick → Rotate → Drop → Silence
Stopping	Stop → Silence
Resizing	Grab → Resize → Stop → Silence

Table 3. Recognition results of composite gestures.

Gesture	Sentence	Percentage correct
Pushing	35	97.14
Twisting	34	100.00
Twisting-anti	28	96.42
Dropping	29	96.55
Flipping	32	96.89
Moving	35	94.29
Rotating	27	92.59
Stopping	33	100.00
Resizing	30	96.67
Total	**283**	**96.47**

nizes the push gesture and a "button release" upon silence. Similarly, we use the moving gesture for dragging. When the system recognizes the pick gesture, it invokes a "button press." It issues "motion notify" events during the hand's translational tracking and a "button release" upon silence. The other seven gestures in our language represent novel interactions that can potentially enhance the functionality of everyday computing.

To examine the accuracy and utility of our platform and gesture language, we performed a study with 16 users with the entire language. To the best of our knowledge, this is the largest gesture vocabulary attempted for such a large user base. We implemented efficient approaches for modeling both low-level and composite gestures to achieve interactive processing so that we could integrate our gesture interface into average computation environments.

We used both text and video cuing to train the users before recording their gesture sequences. We used more than 150 labeled sequences to train the high-level gesture recognition and 283 labeled sequences in testing. Table 3 shows the recognition results on our testing set. We say a sequence is misrecognized when any one of its constituent individual gestures is incorrectly classified (a conservative measure). These results show that our high-level gesture model accurately recognizes continuous gestures over a large group of users.

During the experiment, we also asked participants to complete a feedback form about our gesture system. They all agreed that our gesture vocabulary is easy to remember and learn. When asked about the convenience after prolonged use—a factor that current research often ignores—half of the subjects thought that the gesture system is comparable to a mouse. Six subjects responded that they felt more tired than when using a mouse, and two subjects that they felt less tired. Seven subjects thought that our gesture-based interface is more convenient and comfortable than GUIs with a mouse, seven regarded them as comparable, and two subjects thought our system is more awkward because they had to first learn how to use it. Overall, 14 subjects (88 percent) considered using our platform and natural-gesture language comparable to or more convenient than traditional mouse-based GUIs.

Our research represents a concrete step toward natural human-centered computing. To achieve this ultimate goal, we continue to explore the information-rich video-based interaction. In our current work, we're exploring techniques to build smart environments in which any surface has the potential to dynamically become part of the interface. We're also investigating dynamic user modeling that will let the computer system recognize and adapt to individual users based on habit and style. We expect this adaptability to be a core characteristic of future human-centered computing systems.

References

1. J.J. Corso, G. Ye, and G.D. Hager, "Analysis of Composite Gestures with a Coherent Probabilistic Graphical Model," *Virtual Reality*, vol. 8, no. 4, 2005, pp. 242-252.
2. G. Ye et al., "VICs: A Modular HCI Framework Using Spatio-Temporal Dynamics," *Machine Vision and Applications*, vol. 16, no. 1, 2004, pp. 13-20.
3. M. Beaudouin-Lafon, "Instrumental Interaction: An Interaction Model for Designing Post-WIMP User Interfaces," *Proc. SIGCHI Conf. Human Factors in Computing Systems*, ACM Press, 2000, pp. 446-453.
4. A. van Dam, "Post-WIMP User Interfaces," *Comm. ACM*, vol. 40, no. 2, 1997, pp. 63-67.
5. B. Shneiderman, "Direct Manipulation: A Step beyond Programming Languages," *Computer*, Aug. 1983, pp. 57-69.
6. R. Sukthankar, R.G. Stockton, and M.D. Mullin, "Smarter Presentations: Exploiting Homography in Camera-Projector Systems," *Proc. Int'l Conf. Computer Vision*, vol. 1, IEEE CS Press, 2001, pp. 247-253.
7. G. Ye, J.J. Corso, and G.D. Hager, "Visual Modeling of Dynamic Gestures Using 3D Appearance and Motion Features," *Real-Time Vision for Human-Computer Interaction*, B. Kisacanin, V. Pavlovic, and T.S. Huang, eds., Springer, 2005, pp. 103-120.

***Jason J. Corso** is an assistant professor of computer science and engineering at the University at Buffalo, the State University of New York. His research interests include computer and medical vision, computational biomedicine, machine intelligence, statistical learning, perceptual interfaces, and smart environments. Corso received a PhD in computer science from the Johns Hopkins University. He is a member of the IEEE and the ACM. Contact him at jcorso@cse.buffalo.edu.*

***Guangqi Ye** is a staff software engineer at PayPal, an eBay company. His research interests include human-computer interaction, machine learning, and computer vision. He received a PhD in computer science from the Johns Hopkins University. He is a member of the IEEE. Contact him at gye@jhu.edu.*

***Darius Burschka** is an associate professor of computer science at the Technische Universität München, where he heads the computer vision and perception group. His research interests include sensor systems for mobile robots and human-computer interfaces, vision-based navigation, and 3D reconstruction from sensor data. Burschka received a PhD in electrical and computer engineering from the Technische Universität München. He is a member of the IEEE. Contact him at burschka@cs.tum.edu.*

***Gregory D. Hager** is a professor of computer science at the Johns Hopkins University and the deputy director of the Center for Computer-Integrated Surgical Systems and Technology (www.cisst.org). His research interests include computer vision and robotics. Hager received a PhD in computer science from the University of Pennsylvania. He is a Fellow of the IEEE. Contact him at hager@cs.jhu.edu.*