Application of Head Tracking for Interactive Data Visualization

Phillip C. S. R. Kilgore
Dept. of Computer Science
LABi @ LSU Shreveport
Shreveport, United States
Email: kilgorep54@lsus.edu

Charles D. McCarthy
Dept. of Computer Science
LABi @ LSU Shreveport
Shreveport, United States
Email: mccarthy20@lsus.edu

Urška Cvek
Dept. of Computer Science
LABi @ LSU Shreveport
Shreveport, United States
Email: ucvek@lsus.edu

Marjan Trutschl
Dept. of Computer Science
LABi @ LSU Shreveport
Shreveport, United States
Email: mtrutsch@lsus.edu

Abstract—**The utilization of head tracking in gaming and animation applications has increased due to greater availability of relevant libraries and hardware. However, its application to interactive data visualization has not followed the same trajectory. In this paper, we describe an extensible platform permitting the integration of head-tracking interactivity into data visualization software. This platform utilizes Haar classification to provide the recognition of facial features, and uses Kalman filtering to smooth transient input. We also discuss the application of head tracking to data visualization, and address its challenges.**

Keywords- human computer interaction; graphical user interfaces; interaction styles; computer vision; scene analysis; tracking;

I. Introduction

Data visualization focuses on visual presentation of data that is usually highly abstract, high-dimensional and structured, without a "natural" representation on a two-dimensional (2D) plane or three-dimensional (3D) space. One of the simplest data visualization examples is a scatter plot. More complex examples include Radviz, parallel coordinates, multidimensional scaling, or other projections of the data that give insights and uncover previously unknown relationships. Interactivity in high-dimensional data visualizations has been shown to be highly beneficial for the data exploration approach [1]. Interactive data visualization is used for exploration, analysis and presentation of the data [2]. Together with animation, 3D increases the density of information that can be presented on the same screen and thus increases the intrinsic dimensionality of visualizations [3].

3D visualization enables the user to make use of spatial memory. User interface animation in 3D spaces can reveal process and structure (by moving the viewpoint) as previously discussed by Baecker and Small [4]. Investigations of Ware and Franck into motion cues in 3D visualization [5] noted that simple rotation about an axis is effective in interpreting 3D information structures. Three dimensions can bring about problems, including depth perception and occlusion. Occlusion has already been addressed by Elmqvist and Tsigas [6] and others. In this paper, we address the problem of depth perception by using head tracking as a more intuitive interactive approach. Augmented reality, or enriched real environment, has been explored as a tool for interacting with multidimensional information visualizations based on the 3D scatter plots [7] and has been shown to enhance a user's data exploration experience [8].

Head tracking facilitates in determining the location of a user's head relative to a particular focal point. In the context of human computer interaction, this information can be used to change the presentation of application content. Head tracking is common in CAVE and CAVE-like environments that are geared towards groups. [9] Our platform is designed for a single person using a standard desktop computer with an off-the-shelf webcam. It is extensible for the easy integration of head tracking-based interactivity into additional data visualizations.

In this paper, we first describe the detection of features such as face and eyes using the cross-platform Open Computer Vision Library (OpenCV) [10]. Later, we integrate the approach into sample well-known visualizations and address the calibration and optimization. To provide for wide adoption of the system, the system utilizes platform independent tools such as OpenGL [11] for 3D graphics and Qt [12] for the graphical user interface. We conclude the paper with a list of challenges, followed by plans for future work.

II. Approach

In order to adjust the view of the 3D scene, we first have to identify the user's position relative to the monitor. We capture frames in a free-head fashion, with an off-the-shelf webcam (Logitech® Webcam C905) centered at the top of the monitor. We then calculate the user's head position using an object detection algorithm. Many different algorithms exist to perform face detection, although most are computationally expensive [13]. We use the Haar Classifier to detect facial features in frames, and later use Kalman filtering to reduce jitter between frames.

In the reference capture stage, our system extracts a frame from the capture device and analyzes features from the Haar classifier, yielding reference points used to calculate the position of the head. The output of this stage is supplied to a head position calculation, which performs calibration if necessary, and supplies the resulting head position to the visualization for use in the projection matrix calculation. The projection matrix is set using matrix transformations calculated from the head position, and is followed by offsetting the model view matrix by the position of the head. Finally, the visualization is rendered as it would have been otherwise. Each of these stages may be executed concurrently with one another, allowing the process to be run asynchronously on multiple threads.

A. Head tracking through eye detection

We chose to track the user's eyes as head reference points due to the relative ease of tracking a user's eyes and their constant presence while exploring a visualization. We assume that the user would be looking directly at the monitor, and thus into the webcam at the top of the monitor. This gives us a frontal face with better view of the eyes.

We considered a number of different eye detection classifiers, from general single eye, to eye pair, to separate classifiers for the left and right eyes [14]. Based on the evaluations, a Haar cascade is run on each region using individualized classifiers for each eye [15].

B. Open Computer Vision library

We accomplish eye detection and head tracking through the use of the open source Open Computer Vision Library, (OpenCV) which was developed initially by Intel [10]. We make use of two OpenCV implementations of algorithms, the Haar Classifier [16], and the Kalman filter [17]. We also take advantage of the camera capture functions provided by the library.

C. Haar Classifier

Rather than looking at individual pixels, Viola and Jones devised an algorithm called the Haar Classifier to rapidly detect objects, including human faces, using AdaBoost classifier cascades that are based on Haar-like features [16]. Haar-like features are rectangular patterns of black and white areas, which define the change in contrast values between adjacent groups of pixels. The simple rectangular features of an image are calculated using an intermediate representation of an image, called the integral image [16]. The pixels of the entire rectangular subsection of the source image are summed and subtracted from a scaled sum of the pixels masked by the black region in the Haar-like feature.

Many of the Haar-like features will contain common regions of pixels [16]. Calculating the sums iteratively for each feature would result in massive amounts of redundant calculations. The problem can be reduced to a simple four-element summation for each region of the selected feature pattern area by utilizing an integral image [16]. The integral image is calculated by summing all the intensity values of the pixels to the top and left of the (i, j) pixel, and placing that value in the (i, j) pixel of the integral image. It only takes two passes to compute both integral image arrays, one for each array. Calculating a feature is extremely fast and efficient, as it only takes the difference between six to eight array elements forming two or three connected rectangles to compute a feature of any scale. We chose to use one of the frontal face cascades provided by OpenCV due to its high detection rate and low false positive rate [15].

D. Kalman filter

Haar classifiers are not perfectly accurate, and occasionally produce false positives. To mitigate these errors we use a Kalman filter [17] implemented by OpenCV. The Kalman filter was developed to predict the state of a system in the presence of noisy measurements.

Creating a Kalman filter for each eye introduces new problems, such as inconsistencies in the eye distance. We chose to filter the center of the eye pairs in order to maintain a more consistent eye separation. This approach still results in noisy eye separation measurements, so we also create a Kalman filter for the eye separation value. The filtered eye position and eye separation are then recombined to form a filtered eye pair.

III. Implementation

We utilize the camera functions provided by OpenCV to configure the webcam and to query frames. The reference capture stage begins by capturing a frame from the webcam, and passing it to the face detection function.

A. Face detection

The Haar classifier (Sec. II-C) is passed the region of the camera capture frame that is likely to contain a face and, if a face is found, returns a rectangle describing the face location. Due to minor shifts in the position and size of the face rectangle, we cannot utilize this object as the location of the user's head. Therefore, we proceed to eye detection as a method for more precisely locating the center of the users face as the origin of the head.

B. Eye detection

The face region is then subdivided into regions that likely contain the users eyes (Sec. V-B), and each region is scanned with an individualized Haar classifier cascade [15]. If each eye region finds an eye, the eye group is analyzed to determine user distance and position. We reduce the search region, since running eye detection on an entire capture frame would increase the number of false positives and decrease performance.

C. Update Kalman filters

The output from eye detection step is more stable than the face detection step, but there is still minor variation in the coordinates for each eye. We attempt to smooth the eye coordinates in both position and relative distance by computing the center of the two points, and then passing

Table I
SYMBOLS USED IN CALCULATIONS

Source	*Symbol*	*Description*	*Unit*
Head Position Calculation			
Input	x_c, y_c	center position	pixel
Input	w, h	capture dimensions	pixel
Config	s	screen height	mm
Config	d_m	actual dot separation	mm
Config	f	field of view	rad
Calc	d_p	measured dot separation	pixel
Calc	r	camera radians per pixel	rad/pixel
Calc	θ	point angle of separation	rad
Calc	β, λ	relative horizontal/vertical angle	rad
Calc	ρ	camera vertical angle	rad
Calc	z'_h	head distance to camera	s
Output	x_h, y_h	head position	s
Output	z_h	head distance	s
Projection Matrix Calculation			
Const	p_n	near plane	s
Const	t_w	virtual room width	s
Const	t_h	virtual room height	s
Output	p_l, p_r	left/right planes	s
Output	p_t, p_b	top/bottom plane	s
Eye Search Region Calculation			
Input	x_f, y_f	face location	pixel
Input	w_f, h_f	face dimensions	pixel
Input	i	eye index: 0 = right, 1 = left	pixel
Output	x_e, y_e	eye location	pixel
Output	w_e, h_e	eye width	pixel

the coordinates from the capture frame to individual Kalman filters. The distance between the user's eyes must also remain constant in order to prevent erratic movement on the z axis. Once filtered, the values are recombined into an eye group with each eye at the same vertical position. A vector could be computed from the raw eyes before filtering in order to restore the eye tilt, but this is unnecessary for our purposes as we only require three degrees of freedom.

D. Head position calculation

The head position calculation stage takes the filtered 2D eye group output from the reference capture stage as parameters to calculate the 3D head location. First, the distance between the eyes (in pixels) and the point at the center of the eye group is calculated. In the beginning of the head position calculation stage, the field of view of the capture device (provided by the configuration file discused in Sec. IV-B1) and the current width of the capture frame are used to compute the radians per pixel of the capture frame (Eq. 1). This is done with each pass to account for changes in capture resolution made by the user.

$$r = f/w \tag{1}$$

Then we calculate half of the angle separation between the reference points. (Eq. 2)

$$\theta = rd_p/2 \tag{2}$$

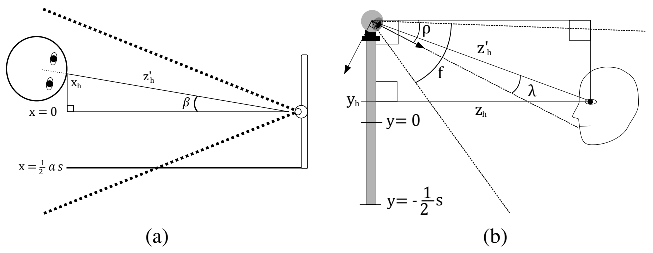

Figure 1. a) Horizontal position of head; b) vertical distance to head

Next, the head distance (screen units) is found using half the actual distance between the user's eyes (mm), the screen height (mm) and the cotangent of the eye angle of separation (Eq. 3).

$$z'_h = \frac{d_m \cot\theta}{2s} \tag{3}$$

The x position of the user's head is then found by taking the sine of the angle between the user's horizontal position (from the reference capture device) and the center of the display surface (Eq. 4, 5). This value is then scaled by the user's distance from the screen. (Fig. 1.a).

$$\beta = r(x_c - \frac{w}{2}) \tag{4}$$

$$x_h = \sin(\beta) z'_h \tag{5}$$

The angle of the camera relative to the user's head is found by calculating the number of pixels that the center point resides above the middle of the frame and multiplying this value by the number of radians in one pixel shift for the current camera (Eq. 6). This value is loaded at startup from the configuration file described in Sec. IV-B1.

$$\lambda = (y_c - \frac{h}{2})r \tag{6}$$

The vertical position of the user can then be calculated for the user's head (Fig. 1.b). In order to account for any vertical tilt that may exist in the cameras view, a calibration routine can be performed at this time if it's required (Sec. IV-B2).

$$y_h = \frac{1}{2} + \sin(\lambda + \rho) z'_h \tag{7}$$

The last step in the head position stage is to calculate the actual distance between the user and the surface of the display (Eq. 8).

$$z_h = \cos(-\lambda - \rho) z'_h \tag{8}$$

E. Projection matrix calculation

$$p_l = \frac{p_n(-\frac{1}{2}t_w - x_h)}{z_h} \tag{9}$$

$$p_r = \frac{p_n(\frac{1}{2}t_w - x_h)}{z_h} \tag{10}$$

$$p_t = \frac{p_n(-\frac{1}{2}t_w - y_h)}{z_h} \tag{11}$$

$$p_b = \frac{p_n(\frac{1}{2}t_w - y_h)}{z_h} \tag{12}$$

The camera is positioned at the origin facing in the direction of the negative z axis with an up vector aligned with the y axis. The OpenGL projection matrix is altered to simulate perspective and field of view changes. The camera is positioned at the origin facing in the direction of the negative z axis (the positive z axis points out of the screen [11]), with an up vector aligned with the y axis. This stage takes the 3D head position as input to calculate the values of the left, right, top, and bottom walls of the near plane (Table I, Eq. 9 – 12 [11]). These walls along with values for the near and far plane are set as parameters to the OpenGL view frustum call.

The actual location of the camera in the OpenGL scene is not altered. Instead, we skew the viewport to achieve field of view changes and move the entire scene opposite the head movement to give the appearance of a receding background as the user moves away. In any case, the projection matrix calculation is isolated from the visualization stage to make extensions easier to write; this matrix is calculated ahead of time and can be utilized when updating the visualization.

IV. Calibration

Due to the variance in consumer webcams, reference point separation, and display device aspect ratio, we chose to create a file to store persistent configuration information. This allows us to retrieve previously stored configurations without having to setup the environment every time the system is restarted. In order to calibrate the system, we have to obtain the capture configuration and subsequently calibrate the software.

A. Reference capture device configuration

We generalize reference capture in order to keep our software independent from the type of capture device. Our software can easily adapt to new types of reference capture devices without altering the overall structure. This gives us a framework that is easily adaptable to any capture device, from the webcam to the Wii Remote or other device. We are using the reference point distance, in addition to the height in order to calibrate the system.

Screen height is used as a base unit for all measurements. The head position is given in units of "screen height." The aspect ratio of the capture device is calculated in order to scale the x axis translation of the user and to fit the virtual room to the corners of the monitor. This sets the resolution of the system.

B. Software calibration

The vertical angle of the camera in relation to the display is calculated at run time during a calibration. Initiating the calibration routine causes our software to use the next head vertical position as the user's center view of the screen. Information about the camera specifications and the user's eye distance is loaded from configuration file at runtime.

1) Configuration file: An external configuration file that contains hardware and user data is loaded at startup. This file stores information about the position of the camera, the aspect ratio of the computer monitor, the monitor's physical height, and the distance between the user's eyes from the center of one pupil to the center of the other.

2) Calibration routine: Because of factors such as the height of the user's monitor, the vertical angle of the camera must be determined. Without knowing the vertical angle of the camera, the position of the user's head cannot be reliably calculated. If one could accurately estimate the size of a screen unit, then it would be possible to perform calibration by positioning oneself in any orientation. Instead of asking the user to properly position themselves one screen unit away from the display, we ask the user to position themselves close to the center of the screen, so that the user's y value is known to be half the screen height. Because we have made this assumption and know both the height of the screen and radians per pixel, we can calculate the vertical angle of the user relative to the camera (Eq. 13). We also have enough information to calculate the head distance of the user, and can use this information to derive the vertical angle of the camera.

$$\rho = -\frac{\pi}{2} + \cos^{-1}(\frac{1}{2z'_h}) - \lambda \tag{13}$$

V. Optimization

Performing face detection using a single classification is time-consuming and inefficient due to the redundant nature of the algorithm [18]. Due to the real time nature of head tracking, we performed several optimizations to first set a smaller rough location of the face, which is then used by the classifier. After the classifier is run and the face is detected, it is more efficient to run the eye feature classifier. We use the distances and set formulas that help us identify the eye regions.

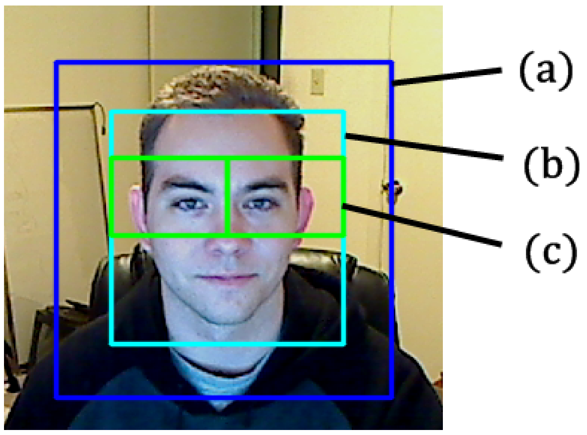

Figure 2. Webcam capture frame: (a) Face search region (b) Detected face (c) Eye search region

A. Face search region reduction

Reducing the face search region can result in improved performance, since the Haar cascade is computationally expensive. We use a simple algorithm developed by [19] to increase face detection performance on the iPhone. The algorithm works by storing the last region that a face was found in and assumes that the next frame will contain a face in the same general region (Fig. 2.a). There is a small amount of padding added to the face region to prevent the face region from becoming smaller with each frame. This happens because as the search region becomes smaller the Haar classifier is more likely to find a face smaller than the previous face, leading to cyclical shrinking. Eventually, the search region becomes too small to contain a face, and the region is reset to the size of the entire frame, therefore degrading performance.

The padding must be as small as possible to maximize the benefits of the algorithm. However, making the padding too small also degrades performance due to the increased probability that a smaller face will be detected with each frame. Reducing the margin also increases the number of times the user's face is lost due to movement outside the bounds of the search region.

B. Eye search region reduction

Once a face is identified, the region is divided into subregions that should contain the user's eyes (Fig. 2.c). The eyes are assumed to lie on approximately the same horizontal line, and the head is assumed to be in an upright orientation. Therefore, the eye regions are defined by the following formulas:

$$x_e = x_f + i(\frac{w_f}{2}) \tag{14}$$

$$y_e = y_f + \frac{h_f}{5} \tag{15}$$

$$w_e = \frac{w_f}{2} \tag{16}$$

$$h_e = \frac{h_f}{3} \tag{17}$$

Reducing the eye search region in this way helps to prevent false positives and increases performance by eliminating regions that are unlikely to contain eyes.

VI. Visualization

Head tracking enabled visualizations allow the user to control the perspective of the view in a natural way. The data is presented in a virtual room to heighten the perception of depth.

A. Objectives and application

Visualization in 3D space can be useful with multidimensional data sets as compared to two-dimensional visualizations by affording an extra spatial dimension for points to lie on. In some cases, this extra spatial dimension can assist in the visual detection of outliers. The Abalone data set [20], a nine-dimensional data set containing 4,177 records, is such an example. Outliers in the Abalone dataset are clearly exposed when ring count is plotted against height, but are obscured when it is plotted against visceral weight.

However, traditional mechanisms of manipulating this space introduce new problems regarding interaction with the visualization. The first problem arises because these visualizations are operated using input devices that are intended for manipulation of objects on a plane, such as a mouse or a trackball. An attempt has been made to address these concerns in specialized 3D input devices, such as 3DConnexion's SpacePilot PRO [21]. However, these devices are not widespread, and may require considerable support.

We have proposed head tracking, via a web cam or similar device, as an alternative to traditional 3D input devices. Devices such as these are inexpensive, and are supported in a variety of modern environments [22]. In a 2007 study published by Dynamics of Institutions and Markets Europe, 31.6% of 2094 participants owned web cams [23]. Because of their widespread usage and availability, optical input devices lend themselves to the navigation of 3D visualizations. Computer vision has previously been used as mechanisms of cursor control for those who have limited motor capability [22], and is currently being implemented by manufacturers of video game consoles [24]. With this research in mind, we considered head tracking as a viable input device to 3D visualizations.

We created an implementation of the traditional scatter plot in three dimensions that allowed for manipulation of the camera via eye tracking. We designed the operation of the visualization viewport to be analogous to looking through a

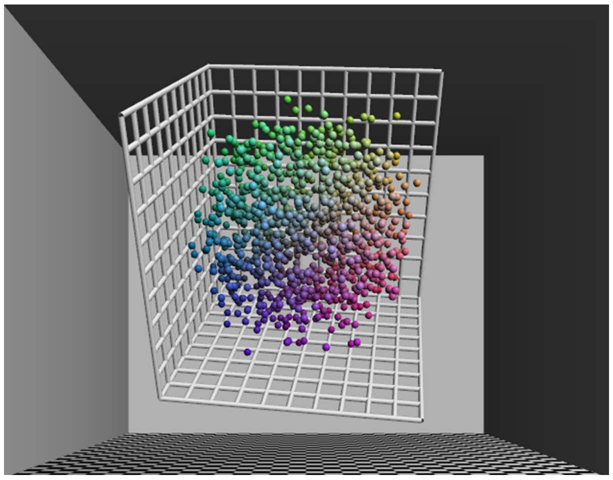

Figure 3. 3D scatter plot interaction utilizing our head tracking framework

window. Eye tracking was not the sole method of input; mouse input was used to manipulate the position of the visualization itself, affording zoom, panning, and rotation. 3D scatter plot (Fig. 3) and PCA projections are trivial examples of 3D visualization, other visualizations may benefit from the addition of head tracking as a navigation method. For instance, volume visualizations could benefit from this input method, allowing users simple access to data obscured by the corners, overlap or pattern in the model. Similarly, projections of classical 2D visualizations into 3D space can benefit from being able to temporarily hone in on data in much the same manner.

B. Challenges

In order to accommodate eye tracking, the visualization viewport is treated as an aperture. As the user moves closer to the aperture, more of the scene is revealed, and the field of view with respect to the scene becomes wider. This variable field of view yields a "dolly-and-zoom" effect, where objects in the background of a scene appear to move into its horizon [25]. Although this kind of perspective distortion is consistent with what is currently known about optics, it appearance seems unnatural, as the eye would never see it the same way, leading to its use in cinematography for dramatic effect. However, this behavior can result in confusion in a visualization system in the absence of a static point of reference, and it was observed in a pre-test of the software on a small group of users. 3D perspective distortion can be described with a 2D distorting visual transfer function, and has been previously detected in information visualizations [26].

Initial testing of the environment revealed that users were mislead into believing that some objects in the visualization were being translated towards the horizon. As a result, we added a room to serve as a static reference point. Like other objects in the background, the back wall of a room will recede into the background as the user moves towards the viewport. The wall is known to be stationary with respect to the user's position, however, and prevents the erroneous conclusion that these objects are in motion.

The notion of a virtual room as static reference point could become ineffective if objects in the visualization were ever to penetrate one of the walls. If one chose to implement zoom by scaling the visualization in question, this problem might be unavoidable, and continuity will be broken. Instead of growing and shrinking the objects in the visualization, we found that an alternative metaphor was bringing the object closer to or further away from the camera. By doing this, one can ensure that the content of the visualization is never large enough to cross the boundaries provided by the virtual room.

Reduction of the search region results in increased performance due to its smaller area that must be analyzed by the Haar classifier, but results in false negatives as the search region approaches face size. We suggest a "user movement vector" to dynamically resize the region based on the user's predicted movement. The user movement vector is a two-dimensional vector that is representative of velocity of the user's head. By utilizing a user movement vector, one can resize the search region to account for quick movements of the head.

VII. Conclusions and Future Work

In this paper, we presented an alternative interactive mechanism that may be used to augment traditional three-dimensional visualizations. This method uses resources that are readily available and may be obtained at relatively little expense. We discussed the implementation of such a system, and steps that are taken in our system to optimize its performance. With these optimizations in place, we were able to implement an interactive 3D scatter plot and PCA projection.

We discussed several challenges that restrict the usability of head tracking in a visualization setting, and how we addressed these issues. One notable challenge we faced was transient input from the input capture system, which we addressed by removing noise generated by Haar classification using a Kalman filter. We also found that problems with maintaining perspective arose, such as perspective distortion introduced by dolly zoom.

We hypothesize that head tracking can be applied, in a general sense, to any 3D visualization. In future work, we would like to apply this research to other types of 3D visualizations, in addition to scatter plots. We would also like to compare the current method of filtering transient input (Kalman filtering) to double exponential smoothing, which claims performance that is two orders of magnitude greater than Kalman filtering.

Implementation of a user movement vector (as described in Sec. VI-B) may increase performance by resizing search regions based on context. We would finally like to perform extensive usability testing to gather evidence to sustain or deny our hypotheses. Though we have performed preliminary testing of our head tracking system, we plan to perform a full study involving users in the future.

VIII. Acknowledgements

This project was supported by grants from the National Center for Research Resources (5P20RR016456-11 and 5P20RR018724-10) and the National Institute of General Medical Sciences (8 P20GM103424-11 and 8 P20 GM103433-10) from the National Institutes of Health. Its contents are solely the responsibility of the authors and do not necessarily represent the official views of NIH.

References

[1] A. Buja, D. Cook, and D. F. Swayne, “Interactive high-dimensional data visualization,” *Journal of Computational and Graphical Statistics*, vol. 5, pp. 78–99, 1996.

[2] R. Kosara and H. Hauser, “An interaction view on information visualization,” in *State-of-the-Art Proceedings of EUROGRAPHICS (EG 2003)*, Granada, Spain, 2003, pp. 123–137.

[3] G. G. Robertson and S. K. Card, “Information visualization using 3d interactive animation,” *Communications of the ACM*, vol. 36, no. 4, pp. 57–71, Apr. 1993.

[4] R. Baecker and I. Small, *The Art of Human-Computer Interface Design*. Addison-Wesley, 1990.

[5] C. Ware and G. Franck, “Evaluation stereo and motion cues for visualizing information nets in three dimensions.” *ACM Transactions on Graphics*, vol. 15, Apr. 2006.

[6] N. Elmqvist and P. Tsigas, “View projection animation for occlusion reduction,” in *Proceedings of the Working Conference on Advanced Visual Interfaces*, ser. AVI ’06, May 2006, pp. 471–475.

[7] B. S. Meiguins, R. M. Casseb do Carmo, A. S. Gonclaves, P. I. Alves Godinho, and M. de Brito Garcia, “Using augmented reality for multidimensional data visualization,” in *Proceedings of the the Conference on Information Visualization*, Jul. 2006.

[8] R. M. Casseb do Carmo *et al.*, “Coordinated and multive views in augmented reality environment.” in *Proceedings of the 11th International Conference on Information Visualization*, Jul. 2007, pp. 156–162.

[9] J. Jacobson *et al.*, “The CaveUT system: immersive entertainment based on a game engine,” in *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, ser. ACE ’05. New York, NY, USA: ACM, 2005, pp. 184–187. [Online]. Available: http://doi.acm.org/10.1145/1178477.1178503

[10] Intel Coproration, *Open Computer Vision Library Reference Manual*, 2001.

[11] Khronos Group, “OpenGL - The Industry Standard for High Performance Graphics,” www.opengl.org, Feb. 2012.

[12] Nokia Corporation, “Qt – A cross platform application and UI framework,” http://qt.nokia.com, Feb. 2012.

[13] G. Bradski, “Computer vision face tracking for use in a perceptual user interface,” *Intel Technology Journal*, 1998, 2nd Quarter.

[14] M. Castrillón Santana *et al.*, “Face amd feature detection evaluation,” in *Third International Conference on Computer Vision Theory and Applications*, ser. VISAPP, 2008.

[15] ——, “ENCARA2: Real-time detection of multiple faces at different resolutions in video streams,” *Journal of Visual Communication and Image Representation*, pp. 130–140, 2007.

[16] P. Viola and M. Jones, “Rapid object tection using boosted cascade of simple features,” in *IEEE Conference on Computer Vision and Pattern Recognition*, 2001.

[17] R. E. Kalman, “A new approach to linear filtering and prediction problems,” *Journal of Basic Engineering*, vol. 82D, no. 1, pp. 34–45, 1960.

[18] L. Liuxia, “Research on face detection classifier using an improved adaboost algorithm,” in *International Symposium on Computer Science and Computational Technology*, 2008, pp. 78–81.

[19] More Than Technical, “Near realtime face detection on the iPhone w/ OpenCV port,” http://www.morethantechnical.com, Feb. 2012.

[20] W. J. Nash *et al.*, “The population biology of abalone (haliotis species) in tasmania, i. blacklip abalone (h. rubra) from the north coast and islands of bass strait.” Sea Fisheries Division, Tech. Rep. 48, Dec. 1995, UCI Machine Learning Repository. http://archive.ics.uci.edu/ml/datasets/Abalone.

[21] 3DConnexion, http://www.3dconnexion.com, Feb. 2012.

[22] P. Mauri, T. Ganollers, and M. Garcia, “Computer vision interaction for people with severe movement restrictions,” *Human Technology Journal*, vol. 2, no. 1, pp. 38–53, 2006.

[23] F. van Rijnsoever and C. Castaldi, “Perceived technology clusters and ownership of related technologies: the role of path-dependence,” in *DIME Workshop on Demand, Product Characteristics and Innovation*, Jena, 2007.

[24] Microsoft Corporation, “Xbox Kinect,” http://www.xbox.com/kinect, Feb. 2012.

[25] K. E. Zheng *et al.*, “Parallax compution: Creating 3d cinematic effects from stills,” in *Graphics Interface Conference*, Kelowna, British Columbia, Canada, 2009.

[26] J. D. Mackinlay, G. G. Robertson, and S. K. Card, “The perspective wall: Detail and context smoothly integrated,” in *SIGCHI ’91*, 1991, pp. 173–179.