Head Gestures for Computer Control

Rick Kjeldsen
IBM T.J. Watson Research Center
fcmk@us.ibm.com

Abstract

This paper explores the ways in which head gestures can be applied to the user interface. Four categories of gestural task are considered, Pointing, Continuous Control, Spatial Selection and Symbolic Selection. For each, the problem is examined in the abstract, focusing on human factors and an analysis of the task, then solutions are presented which take into consideration sensing constraints and computational efficiency. A hybrid pointer control algorithm is described that is better suited for facial pointing than either pure rate control or pure position control approaches. Variations of the algorithm are described for scrolling and selection tasks. The primary contribution is to address a full range of interactive head gestures using a consistent approach which focuses as much on user and task constraints as on sensing considerations.

1. Introduction

As cameras become cheap and ubiquitous we have the opportunity to use them as another way to control the computing devices in our lives - as a user interface device. Is this a desirable goal? Camera-based interface tools may offer increased flexibility and bandwidth with respect to current devices, but they may also simply increase the complexity of the interface without providing worthwhile benefit. We have been exploring their potential empirically by developing user interface components for tasks such as pointing and scrolling based only on camera input.

This paper considers one class of camera-based interaction, those using intentional Head Gestures such as aiming your nose or nodding your head. Intent is important because there are various ways to use knowledge of a users activity without there being intent, or even awareness, on the user's part, e.g. [15]. In this work, the feedback between the user's action and the system's response is an essential part of the interaction.

We contend that practical head gesture interactions fall into four categories:

- **Pointing:** accurately identifying an arbitrary location on the display with high resolution.
- **Continuous Control**: determining the value of some continuous parameter such as color or speed.
- **Spatial Selection**: identifying one of several alternatives distributed in either screen or image space, e.g. tilting your head left to select the left button.
- **Symbolic Selection**: identifying one of several alternatives by non-spatial means, e.g. nodding for yes.

This paper will consider each of these gesture types in turn with the goal of increasing our understanding of head-gesture-based interaction by considering the constraints imposed by the user, the task, and the sensing modality.

2. Related work

There has recently been considerable interest in automated understanding of human action and gesture [6], [12], but the vast majority of the work has focused on the recognition aspects of the problem, with little consideration of how the information gained about the user is to be used. The workshop on Perceptual User Interfaces [7] is one attempt to address this problem, by attracting contributions from various fields including HCI, psychology, human factors and the like.

Research specifically on Head Gestures is more unusual. There has been some work describing isolated applications such as [11] and [3], but nothing that we are aware of has examined the topic as a whole. As with gesture recognition in general, the bulk of the work focuses on the tracking or system integration aspects of the problem e.g. [5] and [9].

Some of the algorithms and human factors work done in the context of current pointing devices is applicable to head gesture, including [8] and [14].

0-7695-1074-4/01 $10.00 © 2001 IEEE

3. Pointing

Pointing is an important part of many types of human computer interaction, and will no doubt remain important as interfaces become more sophisticated [2]. This section will address the high accuracy pointing we see in today's GUI, identifying screen locations at pixel or character resolution. In addition to the familiar direct manipulation tasks, pointing can also be used as an intermediate step for higher level interpretation. We will give an example of this in the next section.

With camera-based gestural interactions there is no tactile feedback for the user. There is also a fair amount of error inherent in any method of sensing the position of the user's body, and often some amount of uncertainty on the part of the user as to exactly where they are pointing. For accurate localization it is therefore essential that the user have visual feedback of where the system believes they are pointing. In this sense, accurate facial pointing is equivalent to moving a pointer about on the screen in response to facial movement.

The next sections will briefly consider the following:

- The **Head Motions** well suited to this type of interaction, and the way they are best used.
- **Facial Tracking** algorithms that can be used to gather input signals from those motions.
- **Pointer Control** algorithms which translate the input signals into movement of the pointer.

For a more complete discussion see [4].

3.1. Head motions

Camera-based sensing must rely on changes to visible characteristics of objects like shape, orientation and location. There can be no analog to the isometric joystick. For any head gestures, shape (e.g. facial expression) is probably not a viable option for a range of reasons, both social and practical.

In the absence of body motion, changes to head location are primarily achieved by tilting. Excursions in tilt should be limited in amplitude, duration and frequency, however, because in tilt the head is not balanced over the spine, making the neck muscles work to hold it up. One of our early approaches to pointing made use of left/right tilt, but the repeated and extended tilting required quickly became uncomfortable. Tilt should be reserved for intermittent tasks.

For head-based pointing, this leaves rotation (left/right and up/down) as the best control mode. Fortunately, people are good at accurately positioning their head (aiming their face) under rotation.

Whatever head motion is to be used, a control signal for pointer movement must be extracted from it. There are two types of control signals which we will consider:

- **Absolute signals**, which record the state of the head with respect to some fixed reference.
- **Relative signals**, which record the state of the head with respect to its previous state.

When referring to head rotation, these correspond to the rotation angle and angular velocity respectively.

Relative signals are difficult to use for facial pointing because of the constraints imposed by the limited range of motion and the need for the user to look at the screen. Absolute signals have their own set of complications. A reference state must be determined, then either remain fixed, or be tracked during the interaction. Our work to this point has explored only absolute control signals, but it would be interesting to try to make use of relative signals as well.

Similarly, the location of the pointer on the screen can be controlled using either its motion relative to a previous location (rate control) or by setting its absolute location with respect to the screen (position control). Consider examples of these two types of interaction:

Absolute Signals controlling Position (AP): The location of the facial image within a box positions the pointer on the screen correspondingly.

Absolute Signals controlling Rate (AR): The rotation of the face from straight ahead determines the speed of the pointer in the corresponding direction.

True nose pointing, in which the pointer is positioned where a really long nose would touch the screen, is an instance of AP where the mapping function from orientation to screen location takes into account the 3D geometry of the environment. Any AP pointing scheme needs additional processing, typically filtering, as the sensing will almost certainly have much less resolution than the screen.

With any facial pointer, care must be taken so that the user can always look comfortably at the pointer. This problem is particularly acute with rate control approaches. It is easy to end up with a situation where, for example, the pointer is on the extreme right of the screen and to move it slightly left the user must aim their face to the left the of the screen. Looking at the pointer out of the corner of the eyes rapidly becomes irritating.

3.2. Facial tracking

The most obvious way to extract a pointer control signal from an image stream is to infer the 3D state of the head. Currently inferring 3D information is

computationally expensive, noisy and somewhat unreliable with respect to the demands of real time interaction [13]. While these problems will fade with time, we have chosen to follow in the spirit of Brooks [1], using the information in the image directly rather than maintaining a world model. We track image features which move or change as a function of the desired head motion, and use them directly as the control signal.

The advantage of decreased computational complexity should not be underestimated. Higher frame rates contribute directly to better usability, moreover camera-based interactions are likely to be used in concert with other resource intensive applications such as voice recognition as part of a multi-modal user interface. Even with the faster machines several years in the future, users will want their computing resources available for the target task, not squandered by the interface.

If the camera is looking at the user approximately head-on, small 3D head rotation appears as 2D translation of the facial image (with increasing amounts of image warping). The expected range of facial image motion can be computed to within small error using the size of the facial image and typical head dimensions. This allows the location of the facial image within its range of motion to be used as a proxy for head rotation.

The following subsections describe a face tracker based on this technique.

3.2.1. Training. While a completely automated system is clearly desirable, we have found that a quick and easy training step at the beginning of a session is usually acceptable. Later, when the system is running, performance can degrade due to lighting or geometric variation beyond the ability of the system to compensate. When this occurs the user can retrain by pressing a "hot key".

Training establishes both the appearance of the user's face in the current environment and expected range of motion. The user is asked to look at the center of the screen and press a key. Their face is located automatically [10] in a training image, and the user is asked to verify that the location is correct.

From the training image, the face is cropped between the top lip and the eyebrows, and between the outside corners of the eyes. This region stays relatively stable during facial deformation (speaking, smiling, etc.), has enough contrast in orthogonal directions for good spatial localization, and remains in view during the head rotations used for pointer control. This becomes our "face template". The location of the template in the training image becomes the reference point (*B* in Fig. 1).

3.2.2. Tracking. The face is tracked by matching the face template within a small search region in each frame. The average of the absolute value of the difference between the gray value of corresponding pixels seems to give better stability than using a squared pixel difference.

The search region is computed using the maximum reasonable head speed during pointing, determined empirically to be about 3 facial diameters per second, and the instantaneous frame rate. Expanding the search region on two sides, if needed, to ensure that it includes *B* gives the user a simple recovery mechanism when the tracker looses track of their face. Orienting their face back to the position used in training (looking at the center of the screen) almost always results in the face being found immediately.

This relatively simple template matching tracker has proven robust under a wide range of circumstances. In good lighting, we can track the face through about a 60 degree arc horizontally (30 degrees on either side of the camera), and through about 40 degree arc vertically. Tracking speed is excellent. At 160x120 pixel resolution we can track the user's face at nearly 30 frames per second, using only a small fraction of the CPU[1]. The impediment to using higher resolution is the connection to the camera in our experimental setup (USB), not CPU utilization. Fortunately, low resolution does not affect usability very much, because of the ability of the pointer control algorithm to compensate.

The technique is sensitive to strong lighting variation as the user turns their head (e.g. strong side lighting). Empirically this becomes an issue in only a few environments. Of course, when it does occur, it often makes the system unusable because the user can not point to large regions of the screen. Therefore, we are exploring techniques that use both the original face template, and the most recent face image in order to accommodate more environmental

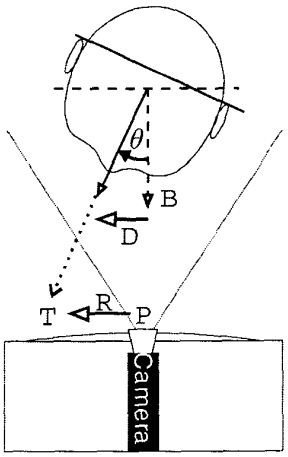

Figure 1: Geometry of Facial Pointing

[1] All performance numbers were obtained on a 700 MHz Intel processor running Windows 2000. Code was written entirely in C++ using only basic optimization techniques.

variation. Results are promising, but not yet ready to report.

3.3. Pointer control

For the purposes of the following discussion, consider the configuration of Figure 1. The user's head is facing the screen, imaged by a camera on top. For the control algorithm we will describe the control signal, C, can either be D, the displacement of the facial plane (or its image) from the base location B, or θ, the 3D change in head orientation.

Our early work mapped C to either an absolute pointer position T (AP interaction) or a pointer motion R (AR interaction) by way of a transfer function.

AP interactions required significant filtering of T to compensate for noise in and coarse resolution of C. While this approach worked, it did not give a pleasing and responsive pointer motion, which had a big impact on usability and user satisfaction.

With AR interactions we were able to tune the transfer function between C and R to improve pointer dynamics. While this approach worked well for other body parts, e.g. hand tracking, with face tracking we encountered the problem mentioned previously where the face is aimed away from the pointer under common circumstances, making viewing it uncomfortable. We compensated for this by adjusting B according to the current location of the pointer P, so that R=0 when the user's face aimed at P. Unfortunately, this made the algorithm cumbersome to work with, limiting further development.

This lead us to the following method, which is a hybrid rate control and position control scheme. Mathematically, it is very similar to the modified rate control method, but in practice it is much easier to work with. In the hybrid method C is translated to a target screen location T. The distance F between P and T determines the rate R at which P moves toward T.

Pointing performance is not significantly impacted even by relatively large inaccuracies in T, so the exact method of converting C to T is not critical. A good approach for 2D tracking is to use the size of the facial image and some assumptions about the environmental geometry to estimate the facial image displacement when the user is looking at the edge of the display. The displacement of the facial image within this range determines T.

Once T is determined, it is important to have a good transfer function to generate R from $F = |T - P|$. It is this transfer function, more than anything else, that affects usability. We have had good results using a sigmoidal relationship

$$R = \frac{R_{\max}}{1 + e^{-\left|\frac{F-k}{\mu}\right|}} \quad \text{(Figure 2)}$$

such that when T is near P large head movements are needed to move it, making fine positioning easier. When $|T-P|$ is large, $R \simeq T-P$ so that P tracks T very closely, making for rapid long distance moves. Importantly, a sigmoid is easy to tune by adjusting knee κ, slope μ and the maximum output value $R_{\max}$. The resulting transfer function is similar to successful solutions in other areas of pointer control, for example the function developed for the isometric joystick found on many current laptops [8].

One final note. R is a true rate, meaning that it has the units screen-pixels-per-second. Before it is applied to move the pointer it is multiplied by the current frame rate to determine the actual pointer displacement. This compensates for variations in frame rate due to lighting or system resource contention, and is important in obtaining a smooth pointer response.

The main advantage of this hybrid algorithm is that it allows us to separate issues specific to sensing the control signal from issues of pointer dynamics and usability. This has made it easier for us to both tune the system for good usability by adjusting pointer dynamics, and also to make progress on the remaining problems such as compensating dynamically for changes in user / screen geometry.

Pure AR interaction does have its place. It is better suited for input signals that do not map well to a screen location, making it suitable for use with a wider range of human motion. It is also easier to adapt for use with relative control signals. For facial pointing, however, the hybrid method is easier to work with.

Using 2D tracking and the hybrid pointer control algorithm we have implemented a face tracking pointer which allows a user to easily identify screen objects the

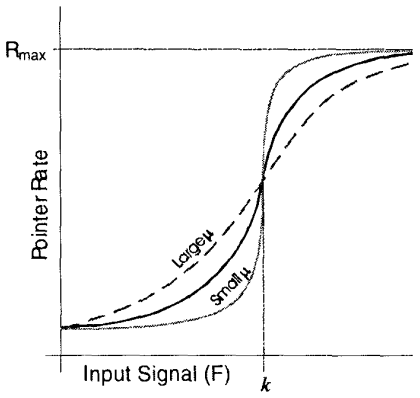

Figure 2: Sigmoid Transfer Function

size of small characters. We have not yet obtained empirical data from controlled user studies, but informal observations based on several users and many hours of use during development and demonstration suggest that screen objects the size of typical buttons or icons can be identified as readily as with current pointing devices, while character level accuracy takes a little more effort. Pointer motion is smooth and predictable, and it is easy to keep the pointer still at any location. We are in the process of designing user studies to evaluate performance more formally.

4. Continuous control

Many user interface tasks can be viewed as controlling a continuous variable. Whenever a user moves a slider in a current WIMP interface they are doing this type of interaction. While there are several tasks where continuous control is useful, we will examine the special case of scrolling around a file. The techniques described are applicable with only minor modification to other tasks.

Scrolling is the act of selecting which portion of a file to display when it is too large to fit within the displayable area (e.g. window) on the screen. Specifically, the variable being controlled is an offset within the file where the display begins. The problem with a typical scroll bar is that the process of finding and manipulating it require the user to change their focus of attention away from the file itself [14]. Using head gesture for scrolling has the potential to be far less distracting.

Controlling a continuous variable is very similar to pointing, or more specifically pointing is an important special case of controlling a variable, so the bulk of the task analysis from the previous section applies. There are some differences, however. For facial scrolling Relative control signals are difficult to use for the same reasons as for facial pointing (Section 3.1), but it is easy to envision other continuous control applications where they may be a good alternative. For facial scrolling, unlike facial pointing, pure rate control approaches generally give better results than pure position control due to different task constraints. In pointing the user would like to be left looking in the direction of the pointer when it is not moving. In scrolling the user wants to be left looking near the center of the display window when not scrolling (the difference between the center of the display window and the exact point which the user is attending to is easily compensated for by eye movement).

In a system where facial pointing has been implemented, one obvious way to implement scrolling is to use the position of the pointer relative to the display window. When the (possibly invisible) pointer moves above the display window, the file is scrolled upward, when it is moved below, the file scrolls down. Effectively the vector from the edge of the display window to the pointer becomes a control vector that will be translated into a scrolling rate. This has the advantage of automatically adjusting for the position of the display window.

If scrolling is being implemented independently of pointing, it is possible to map directly from head movement to scrolling, saving some amount of computational and conceptual complexity with respect to the pointer-based approach. Again either head rotation or facial offset can be used as the control signal. A sigmoidal transfer function can again be used to map the control signal directly to a rate of change of the variable.

With the direct approach, the scrolling dynamics are subtly different than when interpreting the result of facial pointing. When using pointing, scrolling begins when the user is pointing to the edge of the display window. With the direct approach, action is independent of window size. In practice the difference doesn't seem to matter.

Scrolling is special case of continuous control in that it is inherently spatial. For many variables, there is no spatial element, meaning no natural mapping from directions and amounts of facial movement to changes in variable value. We have not yet implemented control of variables other than scrolling. It will be interesting to see if this difference matters.

5. Spatial selection

For some tasks, it is desirable to select one of a small number of spatially distributed options quickly and efficiently. Two quick examples are selecting one of several buttons in a dialog box and selecting which window should receive keyboard input. These interactions typically do not have a direct analog in current computer interfaces, instead selection is nearly always performed with high resolution pointing. We do see an analog in human to human interactions, where a person may tip their head or look in a direction to indicate which alternative they are referring to (e.g. saying "Put it in that cabinet" and tipping their head toward the cabinet on the left). The gesture is not very precise, but it need not be given the context.

Compared to high resolution pointing, there seem to be some human factors advantages to using this type of selection. Selection can be made faster and less distracting, with a tip of the head substituting for finding

the pointer, then moving it to a relatively small target. There also may be less chance of error from selection of erroneous alternatives.

To provide the needed contextual information, the selection software must have some type of feedback from the application. We will not address this issue, except to say that in many cases this information is easily available.

As with continuous control, we have explored two approaches to spatial selection, one using facial pointing, the other using targets to interpret the input signal directly.

5.1. Pointing-based selection

Pointing-based approaches rely on context-aware interpretation of the pointer location. The options are distributed in screen-space and the user points approximately in their direction. Pointing like that described in Section 3 can be used as is, except that in this case the transfer function can be optimized for low latency, as stability and accuracy are not as big a concern. This is easily achieved with the transfer functions we have described by reducing k, the knee of the sigmoid, and adjusting μ for a flatter curve. In many cases the pointer itself need not be visible.

The pointer location is then divided into regions corresponding to the selection alternatives, possibly with neutral areas between them. This subdivision must be done in such a way as to be obvious to the user, which generally implies a division corresponding to the layout of the alternatives on the screen. For example, in the case of a binary decision the left half of the screen can be used for Continue and the right for Cancel, with a small neutral column between them. In simple cases, entry into a region is often sufficient for selection, but having a dwell time during which a pointer must reside in a region before it is selected can make the interface more forgiving. When there are more alternatives, some method of indicating when a selection is finished is needed - the equivalent of a click. Some methods we have investigated include trajectory extrema, motion orthogonal to the trajectory, motion orthogonal to the layout of regions, and of course other modes such as voice. Whatever method is used, care must be taken to avoid the "Midas Touch" problem, where the user unintentionally activates everything they look at.

The examples of this techniques so far have focused on the user providing responses to system queries. An on-screen keyboard we developed provides a different type of example. Here the letters of the keyboard are aligned across the top of the screen. Left / Right motion of the face highlights the letter the user is "pointing at" (the pointer is not displayed). A nod of the head types the character, down typing lower case, up typing upper case.

5.2. Target-based selection

Another approach to implementing spatial selection is to have the options distributed around the user in "image space" such that they can move to "touch" the target corresponding to their choice. We have implemented a solution in which image regions are defined by dragging and resizing rectangles. These targets are trained by demonstration. The targets detect a touch by matching color histograms of the training and current images. The distance threshold for the histogram match can be tuned to adjust the sensitivity of the target. This approach has been used in the TouchFree Switch an interface tool for physically disabled users developed jointly by IBM Research and Edmark.

With a target-based approach, the spatial distribution of the targets need not correspond to screen objects. For example we have implemented a simple demo application we refer to as Air Piano, where targets are arrayed around the user at a comfortable distance. Each target plays a note so that the user can play music simply by reaching out into space around them.

Depending on the application it may be desirable to augment a simple touch with a dwell time or hysteresis to avoid the Midas Touch problem.

An important application of these selection techniques is to identify temporal events - say the system was scanning through several options, or the user wanted to trigger a capture operation. Here the user simply needs to generate a binary signal at the correct time - effectively saying "now" by nodding their head.

6. Symbolic selection

As the name implies, spatial selection relies on a spatial interpretation of the user's actions - we assume they are either identifying regions of screen or image space. Symbolic Selection, on the other hand is the ability to identify one of several alternatives using a non-spatial interpretation. The symbols can be a simple binary signal, or could use a more complex alphabet. The most obvious example is to shake your head yes or no in response to a dialog box.

While spatial selection can often be done on a frame-by-frame basis, symbolic interpretation often involves examination of the head motion over longer sequences.

We have just begun to explore symbolic interpretation of head movements, and will have more to say in the future.

7. Summary and Conclusion

This paper has attempted to examine the roles head gesture can play in a user interface from a point of view that includes human factors, computer vision and the task the user is going to perform.

Head-gesture-based interactions have been classified into four types of tasks: Pointing, Continuous Control, Spatial Selection and Symbolic Selection. Pointing was examined in some detail as it is the most mature application, and has much in common with our approach to the other tasks. The discussion included the facial movements which can be used, the various types of control signals which can be extracted from visual observation of these movements, and how they can be best used to control a pointer.

One of the more important elements of any body tracking pointing system is a transfer function which translates the raw control signal into the pointer movement. We described a sigmoid-based transfer function which combines elements of rate and position control for good usability and easy tuning. Though designed for facial pointing, it is flexible enough to adapt to a range of body tracking applications.

Continuous Control was examined using the example of scrolling. We presented two approaches, one using interpretation of a facial pointer position, and the other using direct control of scrolling rate from head movement. The direct approach uses the same basic techniques as facial pointing, adapted based on an analysis of the scrolling task. Similarly for Spatial Selection tasks. Finally we briefly considered Symbolic Selection, where higher level symbolic interpretation of a user's actions pose a very interesting set of problems for future work.

Previous work on head gesture has focused primarily on pointing. The contribution of this work has been to address a full range of interactive head gestures using a consistent approach which focuses as much on user and task constraints as on sensing considerations. The interactions we have implemented so far show much promise. They are easy and comfortable for the user to perform accurately for long periods of time. While much remains to be done before head gesture becomes a practical part of a user interface,, hopefully a more comprehensive examination of the topic is one step toward that goal.

8. References

[1] R. Brooks, "Intelligence Without Representation" in Artificial Intelligence Journal, Vol. 47, p139, 1991.

[2] N. Jojic, B. Brumitt, B. Meyers, S. Harris, and T. Huang,"Detection and Estimation of Pointing Gestures in Dense Disparity Maps" in [6].

[3] S. Kawato and J. Ohya, "Real-time Detection of Nodding and Head-shaking by Directly Detecting and Tracking the 'Between-Eyes'", in [6].

[4] R. Kjeldsen, "Facial Pointing", presented at the 4th International Workshop on Gesture and Sign Language based Human-Computer Interaction (Gesture Workshop 2001), proceedings forthcoming.

[5] T. Maurer and C. von der Malsburg, "Tracking and Learning Graphs and Pose on Image Sequences of Faces" in Proceedings of the Second International Conference on Automatic Face and Gesture Recognition, October 1996, Killington Vt., IEEE Computer Society Press Order Number PR07713.

[6] Proceedings of the Fourth International Conference on Automatic Face and Gesture Recognition, 28-30 March, 2000, Grenoble, France. IEEE Computer Society Order Number PR00580

[7] Proceedings of the 1998 Workshop on Perceptual User Interfaces (PUI '98), San Francisco, CA, Nov. 1998, M.Turk editor

[8] J. Rutledge and T. Selker, Force-to-Motion Functions for Pointing, in the Proceedings of the IFIP TC 13 Third International Conference on Human-Computer Interaction, August 1990 (Interact '90)

[9] K. Schwerdt and J. Crowley, "Robust Face Tracking Using Color" in [6].

[10] A.W.Senior Face and Feature Finding for a Face Recognition System In proceedings of Audio- and Video-based Biometric Person Authentication '99 Washington D.C. USA, March 22-24, 1999

[11] K. Toyama, "Look, Ma - No Hands! Hands-Free Cursor Control with Real-Time 3D Face Tracking" in [7].

[12] Y. Wu and T. Huang, "Vision-based gesture recognition: A review" in Lecture Notes in Artificial Intelligence V1739 1999

[13] Y. Wu and K. Toyama, "Wide-Range, Person- and Illumination-Insensitive Head Orientation Estimation", in [6].

[14] S. Zhai and B.A. Smith, "Multi-Stream Input: An Experimental Study of Document Scrolling Methods", IBM Systems Journal, Vol 38, No.4, p 642-651.

[15] S. Zhai, C. Morimoto and S. Ihde, "Manual and Gaze Input Cascaded (MAGIC) Pointing", in Proc. CHI '99. Also see www.almaden.ibm.com/cs/blueeyes/magic.html