through each once **434**. Since hand contacts tend to lie in a ring this shortest graph cycle will tend to connect adjacent contacts, thus establishing a sensible ordering for them.

The next step **438** is to pick a contact at an extreme position in the ring such as the innermost or outermost and test whether it is a thumb (decision diamond **440**) or palm (decision diamond **442**). This can be done using contact features and fuzzy logic expressions analogous to those utilized in the thumb verification process and the, attractor weightings. If the innermost path is a thumb, step **444** concludes that contacts above are most likely fingertips, and contacts in the ring below the thumb are most likely palms. If (**442**) the innermost path is a palm heel, step **446** concludes the paths significantly above the innermost must be fingers while paths at the same vertical level should be palms. The thumb and palm tests are then repeated for the contacts adjacent in the ring to the innermost until any other thumb or palm contacts are found. Once any thumb and palm contacts are identified, step **448** identifies remaining fingertip contacts by their respective ordering in the ring and their relatively high vertical position.

Since this alternative algorithm does not include an attractor template to impose constraints on relative positions, the fuzzy verification functions for each contact may need to include measurements of the vertical position of the contact relative to other contacts in the ring and relative to the estimated hand offset. The attractor template embodiment is preferred over this alternative embodiment because the attractor embodiment more elegantly incorporates expected angles between contacts and the estimated hand offset into the finger identification process.

Hand identification is needed for multi-touch surfaces which are large enough to accomodate both hands simultaneously and which have the left and right halves of the surface joined such that a hand can roam freely across the middle to either half of the surface. The simplest method of hand identification would be to assign hand identity to each contact according to whether the contact initially touched down in the left or right half of the surface. However, if a hand touched down in the middle, straddling the left and right halves, some of the hand's contacts would end up assigned to the left hand and others to the right hand. Therefore more sophisticated methods which take into account the clustering properties of hand contacts must be applied to ensure all contacts from the same hand get the same identity. Once all surface contacts are initially identified, the path tracking module can reliably retain existing identifications as a hand slides from one side of the surface to the other.

The thumb and inner palm contact orientations and the relative thumb placement are the only contact features independent of cluster position which distinguish a lone cluster of right hand contacts from a cluster of left hand contacts. If the thumb is lifted off the surface, a right hand contact cluster appears nearly indistinguishable from a left hand cluster. In this case cluster identification must still depend heavily on which side of the board the cluster starts on, but the identity of contacts which recently lifted off nearby also proves helpful. For example, if the right hand moves from the right side to the middle of the surface and lifts off, the next contacts which appear in the middle will most likely be from the right hand touching back down, not from the left hand moving to the middle and displacing the right hand. The division between left and right halves of the surface should therefore be dynamic, shifting toward the right or left according to which hand was most recently near the middle. Since the hand offset estimates temporarily retain the last known hand positions after liftoff, such a dynamic division is implemented by tying the positions of left hand and right hand attractor templates to the estimated hard positions.

Though cases remain in which the user can fool the hand identification system with sudden placements of a hand in unexpected locations, the user may actually wish to fool the system in these cases. For example, users with only one hand free to use the surface may intentionally place the hand far onto the opposite half of the surface to access the chord input operations of the opposite hand. Therefore, when a hand cluster suddenly touches down well into the opposite half of the surface, it can safely be given the opposite halfs identity, regardless of its true identity. Arching the surface across the middle can also discourage users from sliding a hand to the opposite side by causing awkward forearm pronation should users do so.

FIG. **29** shows process details within the hand identification module **247**. Decision diamond **450** first determines whether the hand identification algorithm actually needs to be executed by checking whether all path proximities have stabilized. To maximize stability of the identifications, hand and finger identities need only be reevaluated when a new hand part touches down or disambiguating features of existing contacts become stronger. The contact size and orientation features are unreliable until the flesh fully compresses against the surface a few dozen milliseconds after initial surface contact. Therefore decision diamond **450** executes the hand identification algorithm for each proximity image in which a new contact appears and for subsequent proximity images in which the total proximity of any new contacts continues to increase. For images in which proximities of existing contacts have stabilized and no new contacts appear, path continuation as performed by the path tracking process **245** is sufficient to retain and extend (step **452**) the contact identifications computed from previous images.

Should the hand identification algorithm be invoked for the current image, the first step **453** is to define and position left and right hand attractor templates. These should be basically the same as the attractor templates (FIG. **24**, step **352**) used in within-hand identification, except that both left and right rings must now be utilized at once. The default placement of the rings relative to one another should correspond to the default left and right hand contact positions shown in FIG. **20**A. Each ring translates to follow the estimated position of its hand, just like the sloppy segmentation regions follow the hands in FIG. **20**B. Individual attractor points can safely be translated by their corresponding estimated finger offsets. Therefore the final attractor positions ($Aj_x[n]$,$Aj_y[n]$) for the left hand L and right hand H attractor rings are:

$$Laj_x[n]=Lh_{ecx}[n]+LFj_{ecx}[n]+Lfj_{defx} \quad (62)$$

$$Laj_y[n]=Lh_{ecy}[n]+LFj_{ecy}[n]+Lfj_{defy} \quad (63)$$

$$Raj_x[n]=Rh_{ecx}[n]+RFj_{ecx}[n]+Rfj_{defx} \quad (64)$$

$$Raj_y[n]=Rh_{ecy}[n]+RFj_{ecy}[n]+Rfj_{defy} \quad (65)$$

Basically the hand identification algorithm will compare the cost of assigning contacts to attractors in one ring versus the other, the cost depending on the sum of weighted distances between each contact and its assigned attractor. Adjusting the attractor ring with the estimated hand and finger offsets lowers the relative costs for assignment hypotheses which resemble recent hand assignments, helping to stabilize identifications across successive proximity images even when hands temporarily lift off.

Next a set of assignment hypotheses must be generated and compared. The most efficient way to generate sensible

hypotheses is to define a set of roughly vertical contour lines, one between each horizontally adjacent contact. Step **454** does this by ordering all surface contacts by their horizontal coordinates and establishing a vertical contour halfway between each pair of adjacent horizontal coordinates. FIGS. **30**A-C show examples of three different contours **475** and their associated assignment hypotheses for a fixed set of contacts. Each contour corresponds to a separate hypothesis, known also as a partition, in which all contacts to the left **476** of the contour are from the left hand, and all contacts to the right **477** of the contour are from the right hand. Contours are also necessary at the left and right ends of the surface to handle the hypotheses that all contacts on the surface are from the same hand. Contours which hypothesize more contacts on a given hand than can be caused by a single hand are immediately eliminated.

Generating partitions via vertical contours avoids all hypotheses in which contacts of one hand horizontally overlap or cross over contacts of the opposite hand. Considering that each hand can cause seven or more distinct contacts, this reduces the number of hand identity permutations to examine from thousands to at most a dozen. With fewer hypotheses to examine, the evaluation of each partition can be much more sophisticated, and if necessary, computationally costly.

The optimization search loop follows. Its goal is to determine which of the contours divides the contacts into a partition of two contact clusters such that the cluster positions and arrangement of contacts within each cluster best satisfy known anatomical and biomechanical constraints. The optimization begins by picking (step **456**) a first contour divider such as the leftmost and tentatively assigning (step **458**) any contacts to the left of the contour to the left hand and the rest to the right hand. Step **460** invokes the finger identification algorithm of FIG. **23**, which attempts to assign finger and palm identities to contacts within each hand. Decision diamond **360** avoids the computational expense of thumb verification **368** and statistics gathering **364** for this tentative assignment hypothesis.

Returning to FIG. **29**, step **462** computes a cost for the partition. This cost is meant to evaluate how well the tentatively identified contacts fit their assigned attractor ring and how well the partition meets between-hand separation constraints. This is done by computing for each hand the sum of weighted distances from each tentatively identified contact to its assigned attractor point as in Equation 54 of finger identification, including size and orientation feature factors for thumb and palm attractors. This sum represents the basic template fitting cost for a hand. Each hand cost is then weighted as a whole with the reciprocals of its clutching velocity, handedness, and palm cohesion factors. These factors, to be described below, represent additional constraints which are underemphasized by the weighted attractor distances. Finally, the weighted left and right hand costs are added together and scaled by the reciprocal of a hand separation factor to obtain a total cost for the partition.

If decision diamond **464** determines this total cost is lower than the total costs of the partitions evaluated so far **464**, step **466** records the partition cost as the lowest and records the dividing contour. Decision diamond **472** repeats this process for each contour **470** until the costs of all partitions have been evaluated. Step **473** chooses the partition which has the lowest cost overall as the actual hand partitioning **473**, and the hand identities of all contact paths are updated accordingly. Then step **474** reinvokes the within-hand finger contact identification process so that the thumb verification and statistics gathering processes are performed using the actual hand assignments.

Users often perform clutching motions in which the right hand, for example, lifts off from a slide at the right side of the surface, touches backdown in the middle of the surface, and resumes sliding toward the right. Therefore when a hand is detected touching down in the middle of the surface and sliding toward one side, it probably came from the at side. A hand velocity factor, plotted approximately in FIG. **31**A, captures this phenomenon by slightly increasing in value when a hand cluster's contacts are moving toward the cluster's assigned side of the board, thus decreasing the basic cost of the hand. The factor is a function of the average of the contacts' horizontal velocities the side of the surface the given cluster is assigned. Since high speeds do not necessarily give a stronger indication of user intent the factor saturates at moderate speeds.

Though the thumb orientation factors help identify which hand a thumb is from when the thumb lies in the ambiguous middle region of the surface, the vertical position of the thumb relative to other fingers in the same hand also gives a strong indication of handedness. The thumb tends to be positioned much lower than the fingertips, but the pinky tends to be only slightly lower than the other fingertips. The handedness factor plotted approximately in FIG. **31**B, takes advantage of this constraint by boosting the hand cost when the contact identified as the outermost fingertip is more than a couple centimeters lower than the next outermost fingertip contact. In such cases the tentative hand assignment for all contacts in the cluster is probably wrong. Since this causes the within-hand identification algorithm to fit the contacts to the wrong attractor ring, finger identities become reversed such that the supposedly lowered pinky is truly a lowered thumb of the opposite hand. Unfortunately, limited confidence can be placed in the handedness factor. Though the pinky should not appear lowered as much as the thumb the outer palm heel can, creating an ambiguity in which the thumb and fingertips of one hand have the same contact arrangement as the fingertips and outer palm heel of the opposite hand. This ambiguity can cause the handedness factor to be erroneously low for an accurately identified hand cluster, so the handedness factor is only used on clusters in the middle of the surface where hand position is ambiguous.

Distinguishing contact clusters is challenging because a cluster can become quite sparse and large when the fingers outstretched, with the pinky and thumb of the same hand spanning up to 20 cm. However, the palm can stretch very little in comparison, placing useful constraints on how far apart palm heel contacts and forepalms from the same hand can be. The entire palm region of an outstretched adult hand is about 10 cm square, so palm contact centroids should not be scattered over a region larger than about 8 cm. When a partition wrongly includes fingers from the opposite hand in a cluster, the within-cluster identification algorithm tends to assign the extra fingers from the opposite hand to palm heel and forepalm attractors. This usually causes the contacts assigned to the cluster's palm attractors to be scattered across the surface wider than is plausible for true palm contacts from a single hand. To punish such partitions, the palm cohesion factor quickly drops below one for a tentative hand cluster in which the supposed palm contacts are scattered over a region larger than 8 cm. Therefore its reciprocal will greatly increase the hand's basic cost. FIG. **31**C shows the value of the palm cohesion factor versus horizontal separation between palm contacts. The horizontal spread can be efficiently measured by finding the maximum and minimum horizontal coordinates of all contacts identified as palm heels or forepalms and taking the difference between the maximum and minimum. The measurement and factor value lookup are repeated for the

vertical separation, and the horizontal and vertical factors are multiplicatively combined to obtain the final palm cohesion factor.

FIG. **33** is an approximate plot of the inter-hand separation factor. This factor increases the total costs of partitions in which the estimated or actual horizontal positions of the thumbs from each hand approach or overlap. It is measured by finding the minimum of the horizontal offsets of right hand contacts with respect to their corresponding default finger positions. Similarly the maximum of the horizontal offsets of the left hand contacts with respect to their corresponding default finger positions is found. If the difference between these hand offset extremes is small enough to suggest the thumbs are overlapping the same columnar region of the surface while either touching the surface or floating above it, the separation factor becomes very small. Such overlap corresponds to a negative thumb separation in the plot. To encourage assignment of contacts which are within a couple centimeters of one another to the same cluster, the separation factor gradually begins to drop starting with positive separations of a few centimeters or less. The inter-hand separation factor is not applicable to partitions in which all surface contacts are assigned to the same hand, and takes on the default value of one in this case.

Alternative embodiments of this hand identification process can include additional constraint factors and remain well within the scope of this invention. For example, a velocity coherence factor could be computed to favor partitions in which all fingers within a cluster slide at approximately the same speed and direction, though each cluster as a whole has a different average speed and direction.

Sometimes irreversible decisions made by the chord motion recognizer or typing recognized on the basis of existing hand identifications prevent late changes in the identifications of hand contacts even when new proximity image information suggests existing identifications are wrong. This might be the case for a chord slide which generates input events that can not be undone, yet well into the slide new image information indicates some fingers in the chord should have been attributed to the opposite hand. In this case the user can be warned to stop the slide and check for possible input errors but in the meantime it is best to retain the existing identifications even if wrong, rather than switch to correct assignments which could have further unpredictable effects when added to the erroneous input events. Therefore once a chord slide has generated input events, the identifications of their existing paths may be locked so the hand identification algorithm can only swap identifications of subsequent new contacts.

This hand identification process can be modified for differently configured multi-touch surfaces and remain well within the scope of this invention. For surfaces which are so narrow that thumbs invade one another's space or so tall that one hand can lie above another, the contours need not be straight vertical lines. Additional contours could weave around candidate overlapping thumbs, or they could be perpendicular to the vector between the estimated hand positions. If the surface was large enough for more than one user, additional attractor rings would have to be provided for each additional hand, and multiple partitioning contours would be necessary per hypothesis to partition the surface into more than two portions. On a surface large enough for only one hand it might still be necessary to determine which hand was touching the surface. Then instead of hypothesizing different contours, the hand identification module would evaluate the hypotheses that either the left hand attractor ring or the right hand attractor ring was centered on the surface. If the surface was mounted on a pedestal to allow access from all sides, the hand identification module would also hypothesize various rotations of each attractor ring.

The attractor-based finger identification system **248** will successfully identify the individual hand contacts which comprise the pen grip hand configuration (FIG. **15**). However, additional steps are needed to distinguish the unique finger arrangement within the pen grip from the normal arrangement within the closed hand configuration (FIG. **14**). In this pen grip arrangement the outer fingers curl under toward the palms so their knuckles touch the surface and the index, finger juts out ahead of them. The pen grip detection module **17** employs a fuzzy pattern recognition process similar to the thumb verification process to detect this unique arrangement

An additional problem with handwriting recognition via the pen grip hand configuration is that the inner gripping fingers and sometimes the whole hand will be picked up between strokes, causing the distinguishing finger arrangement to temporarily disappear. Therefore the pen grip recognition process must have hysteresis to stay in handwriting mode between gripping finger lifts. In the preferred embodiment, hysteresis is obtained by temporal filtering of the combined fuzzy decision factors and by using the estimated finger positions in measurements of finger arrangement while the actual fingers are lifted off the surface. The estimated finger positions provide effective hysteresis because they temporarily retain the unique jutting arrangement before decaying back toward the normal arched fingertip positions a few seconds after liftoff.

FIG. **28** shows the steps within the pen grip detection module **17**. Decision diamond **485** determines whether all pen grip hand parts are touching the surface. If not decision diamond **486** causes the estimated finger and palm positions to be retrieved for any lifted parts in step **487** only if pen grip or handwriting mode is already active. Otherwise the process exits for lack of enough surface contacts. Thus the estimated finger positions cannot be used to start handwriting mode, but they can continue it. Step **488** retrieves the measured positions and sizes of fingers and palm heels which are touching the surface.

Step **489** computes a knuckle factor from the outer finger sizes and their vertical distance from the palm heels which peaks as the outer finger contacts become larger than normal fingertips and close to the palm heels. Step **490** computes a jutting factor from the difference between the vertical coordinates of the inner and outer fingers which peaks as the index fingertip juts further out in front of the knuckles. Step **491** combines the knuckle and jutting factors in a fuzzy logic expression and averages the result with previous results via an autoregressive or moving average filter. Decision diamond **492** continues or starts pen grip mode if the filtered expression result is above a threshold which may itself be variable to provide additional hysteresis. While in pen grip mode, typing **12** and chord motion recognition **18** are disabled for the pen gripping hand.

In pen grip mode, decision diamond **493** determines whether the inner gripping fingers are actually touching the surface. If so, step **495** generates inking events from the path parameters of the inner fingers and appends them to the outgoing event queue of the host communication interface. These inking events can either cause "digital ink" to be laved on the display **24** for drawing or signature capture purposes, or they can be intercepted by a handwriting recognition system and interpreted as gestures or language symbols. Handwriting recognition systems are well known in the art.

If the inner fingers are lifted, step **494** sends stylus raised events to the host communication interface to instruct the handwriting recognition system of a break between symbols. In some applications the user may need to indicate where the "digital ink" or interpreted symbols are to be inserted on the display by positioning a cursor. Though on a multi-touch surface a user could move the cursor by leaving the pen grip configuration and sliding a finger chord, it is preferable to allow cursor positioning without leaving the pen grip configuration. This can be supported by generating cursor positioning events from slides of the palm heels and outer knuckles. Since normal writing motions will also include slides of the palm heels and outer knuckles, palm motions should be ignored until the inner fingers have been lifted for a few hundred milliseconds.

Should the user actually pick up a conductive stylus and attempt to write with it, the hand configuration will change slightly because the inner gripping fingers will be directing the stylus from above the surface rather than touching the surface during strokes. Since the forearm tends to supinate more when actually holding a stylus, the inner palm heel may also stay off the surface while the hand rests on the sides of the pinky, ring finger and the outer palm heel. Though the outer palm heel may lie further outward than normal with respect to the pinky, the ring and pinky fingers will still appear as large knuckle contacts curled close to the outer palm. The tip of the stylus essentially takes the place of the index fingertip for identification purposes, remaining at or above the vertical level of the knuckles. Thus the pen grip detector can function in essentially the same way when the user writes with a stylus, except that the index fingertip path sent to the host communication interface will in actuality be caused by the stylus.

Technically, each hand has 24 degrees of freedom of movement in all finger joints combined, but as a practical matter, tendon linkage limitations make it difficult to move all of the joints independently. Measurements of finger contacts on a surface yield ten degrees of freedom in motion lateral to the surface, five degrees of freedom in individual fingertip pressure or proximity to the surface, and one degree of freedom of thumb orientation. However, many of these degrees of freedom have limited ranges and would require unreasonable twisting and dexterity from the average user to access independently.

The purpose of the motion component extraction module **16** is to extract from the 16 observable degrees of freedom enough degrees of freedom for common graphical manipulation tasks in two and three dimensions. In two dimensions the most common tasks are horizontal and vertical panning, rotating, and zooming or resizing. In three dimensions, two additional rotational degrees of freedom are available around the horizontal and vertical axes. The motion component extractor attempts to extract these 4-6 degrees of freedom from those basic hand motions which can be performed easily and at the same time without interfering with one another. When multiple degrees of freedom can be accessed at the same time they are said to be integral rather than separable, and integral input devices are usually faster because they allow diagonal motions rather than restricting motions to be along a single axis or degree of freedom at one time.

When only four degrees of freedom are needed, the basic motions can be whole hand translation, hand scaling by uniformly flexing or extending the fingers, and hand rotation either about the wrist as when unscrewing ajar lid or between the fingers as when unscrewing a nut. Not only are these hand motions easy to perform because they utilize motions which intuitively include the opposable thumb, they correspond cognitively to the graphical manipulation tasks of object rotation and sizing. Their only drawback is that the translational motions of all the fingers during these hand rotations and scalings do not cancel perfectly and can instead add up to a net translation in some direction in addition to the desired rotation or scaling. To allow all motions to be performed simultaneously so that the degrees of freedom are integral yet to prevent unintended translations from imperfectly performed scalings and rotations, the motion extractor preferentially weights the fingers whose translations cancel best and nonlinearly scales velocity components depending on their speeds relative to one another.

The processes within the motion component extractor **16** are shown in FIG. **34**. Step **500** first fetches the identified contact paths **250** for the given hand. These paths contain the lateral velocities and proximities to be used in the motion calculations, and the identifications are needed so that motion of certain fingers or palm heels which would degrade particular motion component calculations can be deemphasized.

The next step **502** applies additional filtering to the lateral contact velocities when finger proximity is changing rapidly. This is necessary because during finger liftoff and touch down on the surface, the front part of the fingertip often touches down before and lifts off after the back of the fingertip, causing a net downward or upward lateral translation in the finger centroid. Such proximity-dependent translations can be put to good use when slowly rolling the fingertip for fine positioning control, but they can also annoy the user if they cause the cursor to jump away from a selected position during finger liftoff. This is prevented by temporarily downscaling a finger's lateral velocity in proportion to large changes in the finger's proximity. Since other fingers within a hand tend to shift slightly as one finger lifts off, additional downscaling of each finger velocity is done in response to the maximum percent change in proximity among contacting fingers. Alternatively, more precise suppression can be obtained by subtracting from the lateral finger speed an amount proportional to the instantaneous change in finger contact height. This assumes that the perturbation in lateral finger velocity caused by finger liftoff is proportional to the change in contact height due to the back of the fingertip lifting off first or touching down last.

Process **504**, whose detailed steps are shown in FIG. **36**, measures the polar velocity components from radial (scaling) and rotational motion. Unless rotation is extracted from thumb orientation changes, at least two contacting fingers are necessary to compute a radial or angular velocity of the hand. Since thumb motion is much more independent of the other fingers than they are of one another, scalings and rotations are easier for the user to perform if one of these fingers is the opposable thumb, but the measurement method will work without the thumb. If decision diamond **522** determines that less than two fingers are touching the surface, step **524** sets the radial and rotational velocities of the hand to zero. FIG. **35** shows trajectories of each finger during a contractive hand scaling. The thumb **201** and pinky **205** travel in nearly opposite directions at roughly the same speed, so that the sum of their motions cancels for zero net translation, but the difference in their motions is maximized for a large net scaling. The central fingers **202-204** also move toward a central point but the palm heels remain stationary, failing to complement the flexing of the central fingers. Therefore the difference between motion of a central finger and any other finger is usually less than the difference between the pinky and thumb motions, and the sum of central finger velocities during a hand scaling adds up to a net vertical translation. Similar phenomena occur during hand rotations, except that if the rotation is centered at the wrist with forearm fixed rather than

centered at the forepalms, a net horizontal translation will appear in the sum of motions from any combination of fingers.

Since the differences in finger motion are usually greatest between thumb and pinky, step **526** only retrieves the current and previous positions of the innermost and outermost touching fingers for the hand scaling and rotation measurements.

Step **528** then computes the hand scaling velocity H_{vs} from the change in distance between the innermost finger FI and outermost finger FO with approximately the following equation:

$$Hvs[n] = \frac{d(FI[n], FO[n]) - d(fi[n-1], FO[n-1])}{\Delta t} \quad (66)$$

where d(FI[n],FO[n]) is the squared Euclidean distance between the fingers:

$$d(FI[n],FO[n]=\sqrt{(FI_x[n]-FO_x[n])^2+(FI_y[n]-FO_y[n])^2} \quad (67)$$

If one of the innermost or outermost fingers was not touching during the previous proximity image, the change in separation is assumed to be zero. Similarly, step **530** computes the hand rotational velocity H_{vr} from the change in angle between the innermost and outermost finger with approximately the following equation:

$$H_{vr}[n]\left(\frac{\angle(FI[n], FO[n]) - \angle(FI[n-1], FO[n-1]}{\Delta t}\right)\times\left(\frac{d(FI[n], FO[n]}{\pi}\right) \quad (68)$$

The change in angle is multiplied by the current separation to convert it to the same units as the translation and scaling components. These equations capture any rotation and scaling components of hand motion even if the hand is also translating as a whole, thus making the rotation and scaling degrees of freedom integral with translation.

Another reason the computations above are restricted to the thumb and pinky or innermost and outermost fingers is that users may want to make fine translating manipulations with the central fingers, i.e., index, middle, and ring, while the thumb and pinky remain stationary. If changes in distances or angles between the central fingers and the thumb were averaged with Equations 66-68, this would not be possible because central finger translations would cause the appearance of rotation or scaling with respect to the stationary thumb or pinky. However, Equations 56-60 applied in the thumb verification process are only sensitive to symmetric rotation and scaling about a fixed point between the fingers. They approach zero if any significant whole hand translation is occurring or the finger motions are not complementary. In case the user fails to properly move the outermost finger during a rotation or scaling gesture, step **531** uses equations of the approximate form of Equations 56-60 to compute rotation and scaling velocities between the thumb and any touching fingers other than the outermost. The resulting velocities are preferably combined with the results of Equations 66-68 via a maximum operation rather than an average in case translational motion causes the fixed point rotations or scalings to be zero. Finally, decision diamond **532** orders a check for radial or rotational deceleration **534** during motions prior to finger liftoff. The method for detecting radial or rotational deceleration is the same as that detailed in the description of translation extraction.

FIG. **37** shows the details of hand translational velocity measurements referred to in process **506** of FIG. **34**. The simplest way to compute a hand translation velocity would be to simply average the lateral velocities of each finger. However, the user expects the motion or control to display gain to be constant regardless of how many fingers are being moved, even if some are resting stationary. Furthermore, if the user is simultaneously scaling or rotating the hand, a simple average is sensitive to spurious net translations caused: by uncanceled central finger motions.

Therefore, in a preferred embodiment the translational component extractor carefully assigns weightings for each finger before computing the average translation. Step **540** initializes the translation weighting Fi_{vw} of each finger to its total contact proximity, i.e., $Fi_{vw}[n]=Fi_z[n]$. This ensures that fingers not touching the surface do not dilute the average with their zero velocities and that fingers which only touch lightly have less influence since their position and velocity measurements may be less reliable. The next step **544** decreases the weightings of fingers which are relatively stationary so that the control to display gain of intentionally moving fingers is not diluted. This can be done by finding the fastest moving finger, recording its speed as a maximum finger speed and scaling each finger's translation weighting in proportion to its speed divided by the maximum of the finger speeds, as shown approximately in the formula below:

$$Fi_{vw}[n]: = Fi_{vw}[n]\times\left(\frac{Fi_{speed}[n]}{\max_j Fi_{speed}[n]}\right)^{ptw} \quad (69)$$

where the power ptw adjusts the strength of the speed dependence. Note that step **544** can be skipped for applications such as computer-aided-design in which users desire both a normal cursor motion gain mode and a low gain mode. Lower cursor motion gain is useful for fine, short range positioning, and would be accessed by moving only one or two fingers while keeping the rest stationary.

Step **546** decreases the translation weightings for the central fingers during hand scalings and rotations, though it does not prevent the central fingers from making fine translational manipulations while the thumb and pinky are stationary. The formulas below accomplish this seamlessly by downscaling the central translation weightings as the magnitudes of the rotation and scaling velocities become significant compared to $K_{polarthresh}$:

$$Fi_{vwx}[n] \approx \frac{Fi_{vw}[n]\times K_{polarthresh}}{K_{polarthresh}+|H_{vr}[n]|} \quad (70)$$

$$Fi_{vwy}[n] \approx \frac{Fi_{vw}[n]\times K_{polarthresh}}{K_{polarthresh}+|H_{vr}[n]|+|H_{vs}[n]|} \quad (71)$$

where these equations are applied only to the central fingers whose identities i are between the innermost and outermost. Note that since hand scaling does not cause much horizontal translation bias, the horizontal translation weighting $Fi_{vwx}[n]$ need not be affected by hand scaling velocity $H_{vs}[n]$, as indicated by the lack of a hand scaling term in Equation 70. The translation weightings of the innermost and outermost fingers are unchanged by the polar component speeds, i.e., $FI_{vwx}[n]=FI_{vwx}[n]=FI_{vw}[n]$ and $FO_{vwx}[n]=FO_{vwx}[n]=FO_{vw}[n]$. Step **548** finally computes the hand translation velocity vector $(H_{vx}[n],H_{vy}[n])$ from the weighted average of the finger velocities:

$$H_{vx}[n] = \frac{\sum_{i=1}^{5} Fi_{vwx} Fi_{vx}}{\sum_{i=1}^{5} Fi_{vwx}} \quad (72)$$

$$H_{vy}[n] = \frac{\sum_{i=1}^{5} Fi_{vwy} Fi_{vy}}{\sum_{i=1}^{5} F_{vwy}} \quad (73)$$

The last part of the translation calculations is to test for the lateral deceleration of the fingers before liftoff, which reliably indicates whether the user wishes cursor motion to stop at liftoff. If deceleration is not detected prior to liftoff, the user may intend cursor motion to continue after liftoff, or the user may intend a special "one-shot" command to be invoked. Decision diamond **550** only invokes the deceleration tests while finger proximities are not dropping too quickly, to prevent the perturbations in finger centroids which can accompany finger liftoff from interfering with the deceleration measurements. Step **551** computes the percentage acceleration or ratio of current translation speed $|H_{vx}[n], H_{vy}[n])|$ to a past average translation speed preferably computed by a moving window average or autoregressive filter. Decision diamond **552** causes the translation deceleration flag to be set **556** if the acceleration ratio is less than a threshold. If this threshold is set greater than one, the user will have to be accelerating the fingers just prior to liftoff for cursor motion to continue. If the threshold is set just below one, cursor motion will reliably be continued as long as the user maintains a constant lateral speed prior to liftoff, but if the user begins to slow the cursor on approach to a target area of the display the deceleration flag will be set. Decision diamond **554** can also cause the deceleration flag to be set if the current translation direction is substantially different from an average of past directions. Such change in direction indicates the hand motion trajectory is curving, in which case cursor motion should not be continued after liftoff because accurately determining the direction to the user's intended target becomes very difficult. If neither deceleration nor curved trajectories are detected, step **558** clears the translation deceleration flag. This will enable cursor motion continuation should the fingers subsequently begin liftoff. Note that decision diamond **550** prevents the state of the translation deceleration flags from changing during liftoff so that the decision after liftoff to continue cursor motion depends on the state of the deceleration flag before liftoff began. The final step **560** updates the autoregressive or moving window average of the hand translation velocity vector, which can become the velocity of continued cursor motion after liftoff. Actual generation of the continued cursor motion signals occurs in the chord motion recognizer **18** as will be discussed with FIG. **40**.

Note that this cursor motion continuation method has several advantages over motion continuation methods in related art. Since the decision to continue motion depends on a percentage acceleration which inherently normalizes to any speed range, the user can intentionally invoke motion continuation from a wide range of speeds including very low speeds. Thus the user can directly invoke slow motion continuation to auto scroll a document at readable speeds. This is not true of Watanabe's method in U.S. Pat. No. 4,734,685, which only continues motion when the user's motion exceeds a high speed threshold, nor of Logan et al.'s method in U.S. Pat. No. 5,327,161, which if enabled for low finger speeds will undesirably continue motion when a user decelerates on approach to a large target but fails to stop completely before lifting off. Percentage acceleration also captures user intent more clearly than position of a finger in a border area. Position of a finger in a border area as used in U.S. Pat. No. 5,543,591 to Gillespie et al. is ambiguous because the cursor can reach its desired target on the display just as the finger enters the border, yet the touchpad device will continue cursor motion past the target because it thinks the finger has run out of space to move. In the present invention, on the other hand, the acceleration ratio will remain near one if the fingers can slide off the edge of the sensing array without hitting a physical barrier, sensibly invoking motion continuation. But if the fingers decelerate before crossing or stop on the edge of the sensing array, the cursor will stop as desired.

The details of the differential hand pressure extraction process **508** are shown in FIG. **38**. Fingertip proximity, quickly saturates when pressure is applied through the bony tip normal to a hard surface. Unless the surface itself is highly compliant, the best dynamic range of fingertip pressure is obtained with the fingers outstretched and hand nearly flattened so that the compressible soft pulp underneath the fingertips rests on the surface. Decision diamond **562** therefore causes the tilt and roll hand pressure components to be set to zero in step **564** and pressure extraction to abort unless the hand is nearly flattened. Inherent in the test for hand flattening **562** is a finger count to ensure that most of the five fingers and both palm heels are touching the surface to maximize the precision of the hand pressure measurements, though technically only three non-collinear hand contacts arranged like a tripod are necessary to establish tilt and roll pressures. Decision diamond **562** can also require the user to explicitly enable three-dimensional manipulation with an intuitive gesture such as placing all five fingers on the surface briefly tapping the palm heels on the surface, and finally resting the palm heels on the surface. Decision diamond **566** causes step **568** to capture and store reference proximities for each contact path when the proximity of all contacts have stabilized at the end of this initiation sequence. The tilt and roll pressure components are again zeroed **564** for the sensor array scan cycle during which this calibration is performed.

However, during subsequent scan cycles the user can tilt the hand forward applying more pressure to the fingertips or backward applying more pressure to the palm heels or the user can roll the hand outward onto the pinky and outer palm heel or inward applying more pressure to the thumb, index finger and inner palm heel. Step **5170** will proceed to calculate an unweighted average of the current contact positions. Step **572** computes for each hand part still touching the surface the ratio of current proximity to the reference proximity previously stored. To make these ratios less sensitive to accidental lifting of hand parts, step **574** clips them to be greater or equal to one so only increases in proximity and pressure register in, the tilt and roll measurements. Another average contact path position is computed in step **576**, but this one is weighted by the above computed proximity ratios for each path. The difference between these weighted and unweighted contact position averages taken in step **578** produces a vector whose direction can indicate the direction of roll or tilt and whose magnitude can control the rate of roll or tilt about x and y axes.

Since the weighted and unweighted position averages are only influenced by positions of currently contacting fingers and increases in contact pressure or proximity, the method is insensitive to finger liftoffs. Computation of reference-normalized proximity ratios in step **572** rather than absolute

changes in proximity prevents the large palm heel contacts from having undue influence on the weighted average position.

Since only the current contact positions are used in the average position computations, the roll and tilt vector is independent of lateral motions such as hand translation or rotation as long as the lateral motions do not disturb finger pressure, thus once again achieving integrality. However, hand scaling and differential hand pressure are difficult to use at the same time because flexing the fingers generally causes significant decreases in fingertip contact area and thus interferes with inference of fingertip pressure changes. When this becomes a serious problem, a total hand pressure component can be used as a sixth degree of freedom in place of the hand scaling component. This total pressure component causes cursor velocity along a z-axis in proportion to deviations of the average of the contact proximity ratios from one. Alternative embodiments may include further enhancements such as adapting the reference proximities to slow variations in resting hand pressure and applying a dead zone filter to ignore pressure difference vectors with small magnitudes

Despite the care taken to measure the polar velocity, translation velocity, and hand pressure components in such a way that the resultant vectors are independent of one another, uneven finger motion during hand scaling, rotation, or translation can still cause minor perturbations in measurements of one degree of freedom while primarily attempting to move in another. Non-linear filtering applied in steps **510** and **512** of FIG. **34** removes the remaining motion leakage between dominant components and nearly stationary components. In steps **510** each component velocity is downscaled by the ratio of its average speed to the maximum of all the component speeds, the dominant component speed:

$$H_{vx}[n] := H_{vx}[n] \times \left(\frac{H_{xyspeed}[n]}{dominantspeed} \right)^{pds} \quad (74)$$

$$H_{vy}[n] := H_{vy}[n] \times \left(\frac{H_{xyspeed}[n]}{dominantspeed} \right)^{pds} \quad (75)$$

$$H_{vs}[n] := H_{vs}[n] \times \left(\frac{H_{sspeed}[n]}{dominantspeed} \right)^{pds} \quad (76)$$

$$H_{vr}[n] := H_{vr}[n] \times \left(\frac{H_{rspeed}[n]}{dominantspeed} \right)^{pds} \quad (77)$$

where $H_{xyspeed}[n]$, $H_{sspeed}[n]$, and $H_{rspeed}[n]$ are autoregressive averages over time of the translation speed, scaling speed, and rotational speed, where:

$$dominant\_speed = \max(H_{xyspeed}[n], H_{sspeed}[n], H_{rspeed}[n]) \quad (78)$$

where pds controls the strength of the filter. As pdy is adjusted towards infinity the dominant component is picked out and all components less than the dominant tend toward zero producing the orthogonal cursor effect well-known in drawing applications. As pds is adjusted towards zero the filters have no effect. Preferably, pds is set in between so that components significantly slower than the dominant are slowed further, but components close to the dominant in speed are barely affected, preserving the possibility of diagonal motion in multiple degrees of freedom at once. The autoregressive averaging helps to pick out the component or components which are dominant over the long term and suppress the others even while the dominant components are slowing to a stop.

Step **512** takes a second pass with a related filter known as a dead-zone filter. A dead-zone filter produces zero output velocity for input velocities less than a speed threshold but produces output speeds in proportion to the difference between the input speed and the threshold for input velocities that exceed the threshold. Preferably the speed threshold or width of the dead zone is set to a fraction of the maximum of current component speeds. All velocity components are filtered using this same dead zone width. The final extracted component velocities are forwarded to the chord motion recognizer module **18** which will determine what if any input events should be generated from the motions.

FIG. **39**A shows the details of the finger synchronization detector module **14**. The synchronization detection process described below is repeated for each hand independently. Step **600** fetches proximity markers and identifications for the hand's current paths. The identifications will be necessary to ignore palm paths and identify combinations of synchronized fingers, while the proximity markers record the time at which each contact path first exceeds a press proximity threshold and the time at which each contact path drops below a release proximity threshold prior to total liftoff. Setting these proximity thresholds somewhat higher than the minimum proximity considered significant by the segmentation search process **264**, produces more precise finger press and release times.

Step **603** searches for subsets of fingers which touch down at about the same time and for subsets of fingers which lift off at about the same time. This can be done by recording each finger path along with its press time in a temporally ordered list as it crosses the press proximity threshold. Since the primary function of the palms is to support the forearms while the hands are resting, palm activity is ignored by the typing **12** and chord motion recognizers **18** except during differential hand pressure extraction and palm heel presses can be excluded from this list and most other synchronization tests. To check for synchronization between the two most recent finger presses, the press times of the two most recent entries in the list are compared. If the difference between their press times is less than a temporal threshold, the two finger presses are considered synchronized. If not, the most recent finger press is considered asynchronous. Synchronization among three or more fingers up to five is found by comparing press times of the three, four, or five most recent list entries. If the press time of the most recent entry is within a temporal threshold of the nth most recent entry, synchronization among the n most recent finger presses is indicated. To accommodate imprecision in touchdown across the hand, the magnitude of the temporal threshold should increase slightly in proportion to the number of fingers being tested for synchronization. The largest set of recent finger presses found to be synchronized is recorded as the synchronized subset, and the combination of finger identities comprising this subset is stored conveniently as a finger identity bitfield. The term subset is used because the synchronized press subset may not include all fingers currently touching the surface, as happens when a finger touches down much earlier than the other fingers yet remains touching as they simultaneously touch down. An ordered list of finger release times is similarly maintained and searched separately. Alternative embodiments may require that a finger still be touching the surface to be included in the synchronized press subset.

Decision diamond **602** checks whether a synchronization marker is pending from a previous image scan cycle. If not, decision diamond **604** checks whether the search **603** found a newly synchronized press subset in the current proximity image. If so, step **606** sets the temporal synchronization

marker to the oldest press within the new synchronized subset. Additional finger presses may be added to the subset during future scan cycles without affecting the value of this temporal synchronization marker. If there is currently no finger press synchronization, decision diamond **605** determines whether three or more fingers have just been released simultaneously. Simultaneous release of three or more fingers should not occur while typing with a set of fingers but does occur when lifting fingers off the surface from rest. Therefore simultaneous release of three or more fingers reliably indicates that the released fingers are not intended as keypresses and should be deleted from the keypress queue **605**, regardless of whether these same fingers touched down synchronously. Release synchronization of two fingers is not by itself a reliable indicator of typing intent and has no effect on the keypress queue. The keypress queue is described later with FIGS. **42-43**B.

Once a press synchronization marker for the hand is pending, further processing checks the number of finger presses which are synchronized and waits for release of the synchronized fingers. If decision diamond **608** finds three or more fingers in the synchronized press subset the user cannot possibly be typing with these fingers. Therefore step **612** immediately deletes the three or more synchronized presses from the keypress queue. This way they cannot cause key symbol transmission to the host, and transmission of key symbols from subsequent asynchronous presses is not blocked waiting for the synchronized fingers to be released.

However, when the synchronization only involves two finger presses **608**, it is difficult to know whether the user intended to tap a finger pair chord or intended to type two adjacent keys and accidentally let the key presses occur simultaneously. Since such accidental simultaneous presses are usually followed by asynchronous releases of the two fingers, but finger pair chords are usually released synchronously, the decision whether the presses are asynchronous key taps or chord taps must be delayed until finger release can be checked for synchronization. In the meantime, step **610** places a hold on the keypress queue to prevent transmission of key symbols from the possible finger chord or any subsequent finger presses. To prevent long backups in key transmission, decision diamond **614** will eventually release the queue hold by having step **615** delete the synchronized presses from the keypress queue if both fingers remain touching a long time. Though this aborts the hypothesis that the presses were intended as key taps, the presses are also less likely to be key taps if the fingers are not lifted soon after touchdown.

If the synchronized fingers are not lifting, decision diamond **616** leaves the synchronization marker pending so synchronization checks can be continued with updated path parameters **600** after the next scan cycle. If the synchronized fingers are lifting, but decision diamond **618** finds with the help of the synchronization release search **603** that they are doing so asynchronously **618**, step **622** releases any holds on the keypress queue assuming any synchronized finger pair was intended to be two keypresses. Though the synchronized finger presses are not deleted from the keypress queue at this point, they may have already been deleted in step **612** if the pressed subset contained more than two. Also, step **624** clears the temporal synchronization marker, indicating that no further synchronization tests need be done for this subset.

Continuing to FIG. **39**B, if the fingers synchronized during touchdown also lift simultaneously, step **618** removes them and any holds from the keypress queue in case they were a pair awaiting a positive release synchronization test. Further tests ensue to determine whether the synchronized fingers meet additional chord tap conditions. As with single finger taps, the synchronized fingers cannot be held on the surface more than about half a second if they are to qualify, as a chord tap. Decision diamond **626** tests this by thresholding the time between the release of the last remaining synchronized finger and the temporal press synchronization marker. A chord tap should also exhibit a limited amount of lateral finger motion, measured either as an average of peak finger speeds or distance traveled since touchdown in decision diamond **628**. If the quick release and limited lateral motion conditions are not met, step **624** clears the synchronization marker with the conclusion that the synchronized fingers were either just resting fingers or part of a chord slide.

If the chord tap conditions are met, step **630** looks up, using the synchronized subset bitfield, any input events such as mouse clicks or keyboard commands assigned to the combination of fingers in the chord tap. Some chords such as those including all four fingertips may be reserved as resting chords **634**, in which case decision diamond **632** will find they have no associated input events. If the chord does have tap input events, step **636** appends these to the main outgoing event queue of the host communication interface **20**. Finally step **624** clears the synchronization marker in readiness for future finger synchronizations on the given hand.

As a further precaution against accidental generation of chord taps while typing, it is also useful for decision diamond **632** to ignore through step **634** the first chord tap which comes soon after a valid keypress without a chord slide in between. Usually after typing the user will need to reposition the mouse cursor before clicking, requiring an intervening chord slide. If the mouse cursor happens to already be in place after typing, the user may have to tap the finger chord a second time for the click to be sent, but this is less risky than having an accidental chord tap cause an unintended mouse button click in the middle of a typing session.

FIG. **40**A shows the detailed steps of the chord motion recognizer module **18**. The chord motion recognition process described below is repeated for each hand independently. Step **650** retrieves the parameters of the hand's identified paths **250** and the hand's extracted motion components from the motion extraction module **16**. If a slide of a finger chord has not already started, decision diamond **652** orders slide initiation tests **654** and **656**. To distinguish slides from glancing finger taps during typing, decision diamond **654** requires at least two fingers from a hand to be touching the surface for slide mode to start. There may be some exceptions to this rule, such as allowing a single finger to resume a previous slide within a second or so after the previous slide chord lifts off the surface.

In a preferred embodiment, the user can start a slide and specify its chord in either of two ways. In the first way, the user starts with the hand floating above the surface, places some fingers on the surface possibly asynchronously, and begins moving all of these fingers laterally. Decision diamond **656** initiates the slide mode only when significant motion is detected in all the touching fingers. Step **658** selects the chord from the combination of fingers touching when significant motion is detected, regardless of touchdown synchronization. In this case coherent initiation of motion in all the touching fingers is sufficient to distinguish the slide from resting fingers, so synchronization of touchdown is not necessary. Also, novice users may erroneously try to start a slide by placing and sliding only one finger on the surface, forgetting that multiple fingers are necessary. Tolerance of asynchronous touchdown allows them to seamlessly correct this by subsequently placing and sliding the rest of the fingers desired for

the chord. The slide chord will then initiate without forcing the user to pick up all fingers and start over with synchronized finger touchdowns.

In the second way, the user starts with multiple fingers resting on the surface, lifts a subset of these fingers, touches a subset back down on the surface synchronously to select the chord, and begins moving the subset laterally to initiate the slide. Decision diamond **656** actually initiates the slide mode when it detects significant motion in all the fingers of the synchronized subset. Whether the fingers which remained resting on the surface during this sequence begin to move does not matter since in this case the selected chord is determined in step **658** by the combination of fingers in the synchronized press subset, not from the set of all touching fingers. This second way has the advantage that the user does not have to lift the whole hand from the surface before starting the slide, but can instead leave most of the weight of the hands resting on the surface and only lift and press the two or three fingers necessary to identify the most common finger chords.

To provide greater tolerance for accidental shifts in resting finger positions, decision diamond **656** requires both that all relevant fingers are moving at significant speed and that they are moving about the same speed. This is checked either by thresholding the geometric mean of the finger speeds or by thresholding the fastest finger's speed and verifying that the slowest finger's speed is at least a minimum fraction of the fastest finger's speed. Once a chord slide is initiated, step **660** disables recognition of key or chord taps by the hand at least until either the touching fingers or the synced subset lifts off.

Once the slide initiates, the chord motion recognizer could simply begin sending raw component velocities paired with the selected combination of finger identities to the host. However, in the interest of backward compatibility with the mouse and key event formats of conventional input devices, the motion event generation steps in FIG. **40B** convert motion in any of the extracted degrees of freedom into standard mouse and key command events which depend on the identity of the selected chord. To support such motion conversion, step **658** finds a chord activity structure in a lookup table using a bitfield of the identities of either the touching fingers or the fingers in the synchronized, subset. Different finger identity combinations can refer to the same chord activity structure. In the preferred embodiment, all finger combinations with the same number of non-thumb fingertips refer to the same chord activity structure, so slide chord activities are distinguished by whether the thumb is touching and how many non-thumb fingers are touching. Basing chord action on the number of fingertips rather than their combination still provides up to seven chords per hand yet makes chords easier for the user to memorize and perform. The user has the freedom to choose and vary which fingertips are used in chords requiring only one; two or three fingertips. Given this freedom, users naturally tend to pick combinations in which all touching fingertips are adjacent rather than combinations in which a finger such as the ring finger is lifted but the surrounding fingers such as the middle and pinky must touch. One chord typing study found that users can tap these finger chords in which all pressed fingertips are adjacent twice as fast as other chords.

The events in each chord activity structure are organized into slices. Each slice contains events to be generated in response to motion in a particular range of speeds and directions within the extracted degrees of freedom. For example, a mouse cursor slice could be allocated any translational speed and direction. However, text cursor manipulation requires four slices, one for each arrow key, and each arrow's slice integrates motion in a narrow direction range of translation. Each slice can also include motion sensitivity and so-called cursor acceleration parameters for each degree of freedom. These will be used to discretize motion into the units such as arrow key clicks or mouse clicks expected by existing host computer systems.

Step **675** of chord motion conversion simply picks the first slice in the given chord activity structure for processing. Step **676** scales the current values of the extracted velocity components by the slice's motion sensitivity and acceleration parameters. Step **677** geometrically projects or clips the scaled velocity components into the slice's defined speed and direction range. For the example mouse cursor slice, this might only involve clipping the rotation and scaling components to zero. But for an arrow key slice, the translation velocity vector is projected onto the unit vector pointing in the same direction as the arrow. Step **678** integrates each scaled and projected component velocity over time in the slice's accumulators until decision diamond **680** determines at least one unit of motion has been accumulated. Step **682** looks up the slice's preferred mouse, key, or three-dimensional input event format, attaches the number of accumulated motion units to the event; and step **684** dispatches the event to the outgoing queue of the host communication interface **20**. Step **686** subtracts the sent motion events from the accumulators, and step **688** optionally clears the accumulators of other slices. If the slice is intended to generate a single key command per hand motion, decision diamond **689** will determine that it is a one-shot slice so that step **690** can disable further event generation from it until a slice with a different direction intervenes. If the given slice is the last slice, decision diamond **692** returns to step **650** to await the next scan of the sensor array. Otherwise step **694** continues to integrate and convert the current motion for other slices.

Returning to FIG. **40A**, for some applications it may be desirable to change the selected chord whenever an additional finger touches down or one of the fingers in the chord lifts off. However, in the preferred embodiment, the selected chord cannot be changed after slide initiation by asynchronous finger touch activity. This gives the user freedom to rest or lift addition fingers as may be necessary to get the best precision in a desired degree of freedom. For example, even though the finger pair chord does not include the thumb, the thumb can be set down shortly after slide initiation to access the full dynamic range of the rotation and scaling degrees of freedom. In fact, all remaining lifted fingers can always be set down after initiation of any chord to allow manipulation by the whole hand. Likewise, all fingers but one can be lifted, yet translation will continue.

Though asynchronous finger touch activity is ignored, synchronized lifting and pressing of multiple fingers subsequent to slide initiation can create a new synchronized subset and change the selected chord. Preferably this is only allowed while the hand has paused but its fingers are still resting on the surface. Decision diamond **670** will detect the new subset and commence motion testing in decision diamond **673** which is analogous to decision diamond **656**. If significant motion is found in all fingers of the newly synchronized subset, step **674** will select the new subset as the slide chord and lookup a new chord activity structure in analogy to step **658**. Thus finger synchronization again allows the user to switch to a different activity without forcing the user to lift the whole hand from the surface. Integration of velocity components resumes but the events generated from the new chord activity structure will presumably be different.

It is advantageous to provide visual or auditory feedback to the user about which chord activity structure has been selected. This can be accomplished visually by placing a row of five light emitting diodes across the top of the multi-touch

surface, with one row per hand to be used on the surface. When entering slide mode, step **658** would turn on a combination of these lights corresponding to the combination of fingers in the selected chord. Step **674** would change the combination of active lights to match the new chord activity structure should the user select a new, chord, and step **668** would turn them off. Similar lights could be emulated on the host computer display **24**. The lights could also be flashed to indicate the finger combination detected during chord taps in step **636**. The implementation for auditory feedback would be similar, except light combinations would be replaced with tone or tone burst combinations.

The accumulation and event generation process repeats for all array scan cycles until decision diamond **664** detects liftoff by all the fingers from the initiating combination. Decision diamond **666** then checks the pre-liftoff deceleration flag of the dominant motion, component. The state of this flag is determined by step **556** or **558** of translation extraction (FIG. **37**) if translation is dominant, or by corresponding flags in step **534** of polar extraction. If there has been significant deceleration, step **668** simply exits the chord slide mode, setting the selected chord to null. If the flag indicates no significant finger deceleration prior to liftoff, decision diamond **666** enables motion continuation mode for the selected chord. While in this mode, step **667** applies the pre-liftoff weighted average (**560**) of dominant component velocity to the motion accumulators (**678**) in place of the current velocities, which are presumably zero since no fingers touch the surface. Motion continuation mode does not stop until any of the remaining fingers not in the synchronized subset are lifted or more fingers newly touch down. This causes decision diamond **664** to become false and normal slide activity with the currently selected chord to resume. Though the cursor or scrolling velocity does not decay during motion continuation mode, the host computer can send a signal instructing motion continuation mode to be canceled if the cursor reaches the edge of the screen or end of a document. Similarly, if any fingers remain on the surface during motion continuation, their translations can adjust the cursor or scrolling velocity.

In the preferred embodiment, the chord motion recognizers for each hand function independently and the input events for each chord can be configured independently. This allows the system to allocate tasks between hands in many different ways and to support a variety of bimanual manipulations. For example, mouse cursor motion can be allocated to the fingertip pair chord on both hands and mouse button drag to a triple fingertip chord on both hands. This way the mouse pointer can be moved and drug with either hand on either half of the surface. Primary mouse clicks would be generated by a tap of a fingertip pair on either half of the surface, and double-clicks could be ergonomically generated by a single tap of three fingertips on the surface. Window scrolling could be allocated to slides of four fingers on either hand.

Alternatively, mouse cursor manipulations could be allocated as discussed above to the right hand and right half of the surface, while corresponding text cursor manipulations are allocated to chords on the left hand. For instance, left fingertip pair movement would generate arrow key commands corresponding to the direction of motion, and three fingertips would generate shift arrow combinations for selection of text.

For host computer systems supporting manipulations in three or more degrees of freedom, a left hand chord could be selected to pan, zoom, and rotate the display background while a corresponding chord in the right hand could translate, resize and rotate a foreground object. These chords would not have to include the thumb since the thumb can touch down anytime after initiating chord motion without changing the selected chord. The user then need add the thumb to the surface when attempting rotation or scaling.

Finger chords which initially include the thumb can be reserved for one-shot command gestures, which only generate input events once for each slide of a chord rather than repeating transmission each time an additional unit of motion is detected. For example, the common editing commands cut, copy and paste can be intuitively allocated to a pinch hand scaling, chord tap, and anti-pinch hand scaling of the thumb and an opposing fingertip.

FIG. **41** shows the steps within the key layout definition and morphing process, which is part of the typing recognition module **12**. Step **700** retrieves at system startup a key layout which has been pre-specified by the user or manufacturer. The key layout consists of a set of key region data structures. Each region has associated with it the symbol or commands which should be sent to the host computer when the region is pressed and coordinates representing the location of the center of the region on the surface. In the preferred embodiment, arrangement of those key regions containing alphanumeric and punctuation symbols roughly corresponds to either the QWERTY or the Dvorak key layouts common on mechanical keyboards.

In some embodiments of the multi-touch surface apparatus it is advantageous to be able to snap or morph the key layout to the resting positions of the hands. This is especially helpful for multi-touch surfaces which are several times larger than the standard keyboard or key layout, such as one covering an entire desk. Fixing the key layout in one small fixed area of such a surface would be inconvenient and discourage use of the whole available surface area. To provide feedback to the user about changes in the position of the key layout, the position of the key symbols in these embodiments of the multi-touch surface would not be printed permanently on the surface. Instead, the position of the key symbols would be reprogrammably displayed on the surface by light emitting polymers, liquid crystal, or other dynamic visual display means embedded in the multi-touch surface apparatus along with the proximity sensor arrays.

Given such an apparatus, step **702** retrieves the current paths from both hands and awaits what will be known as a layout homing gesture. If decision diamond **704** decides with the help of, a hand's synchronization detector that all five of the hand's fingers have just been placed on the surface synchronously, step **706** will attempt to snap the key layout to the hand such that the hand's home row keys lie under the synchronized fingertips, wherever the hand is on the surface. Step **706** retrieves the measured hand offsets from the hand position estimator and translates all key regions which are normally typed by the given hand in proportion to the measured hand offsets. Note the currently measured rather than filtered estimates of offsets can be used because when all five fingers are down there is no danger of finger misidentification corrupting the measured offsets. This procedure assumes that the untranslated locations of the home row keys are the same as the default finger locations for the hand.

Decision diamond **708** checks whether the fingers appear to be in a neutral, partially closed posture, rather closed than outstretched or pinched together. If the posture is close to neutral, step **710** may further offset the keys normally typed by each finger, which for the most part are the keys in the same column of the finger by the measured finger offsets. Temporal filtering of these finger offsets over several layout homing gestures will tend to scale the spacing between columns of keys to the user's hand size. Spacing between rows is scaled down in proportion to the scaling between columns.

With the key layout for the hand's keys morphed to fit the size and current position of the resting hand, step **712** updates

the displayed position of the symbols on the surface, so that the user will see that the key layout has snapped to the position of his hand. From this stage the user can begin to type and the typing recognizer **718** will use the morphed key region locations to decide what key regions are being pressed. The layout will remain morphed this way until either the user performs another homing gesture to move it somewhere else on the surface, or until the user takes both hands off the surface for a while. Decision diamond **714** will eventually time out so that step **716** can reset the layout to its default position in readiness for another user or usage session.

For smaller multi-touch surfaces in which the key layout is permanently printed on the surface, it is advantageous to give the user tactile feedback about the positions of key regions. However, any tactile indicators placed on the surface must be carefully designed so as not to impede smooth sliding across the surface. For example, shallow depressions made in the surface near the center of each key mimicking the shallow depressions common on mechanical keyboard keycaps would cause a vibratory washboard effect as the hand slides across the surface. To minimize such washboard effects, in the preferred embodiment the multi-touch surface provides for the fingertips of each hand a single, continuous depression running from the default index fingertip location to the default pinky fingertip location. This corresponds on the QWERTY key layout to shallow, slightly arched channels along home row from the "J" key to the ";" key for the right hand, and from the "A" key to the "F" key for the left hand. Similarly, the thumbs can each be provided with a single oval-shaped depression at their default locations, slanted slightly from vertical to match the default thumb orientation. These would preferably correspond to "Space" and "BackSpace" key regions for the right and left thumbs, respectively. Such minimal depressions can tactilely guide users' hands back to home row of the key layout without requiring users to look down at the surface and without seriously disrupting finger chord slides and manipulations on the surface.

The positions of key regions off home row can be marked by other types of tactile indicators. Simply roughening the surface at key regions does not work well. Though humans easily differentiate textures when sliding fingers over them, most textures cannot be noticed during quick taps on a textured region. Only relatively abrupt edges or protrusions can be sensed by the users' fingertips under typing conditions. Therefore, a small raised dot like a Braille dot is formed on top of the surface at the center of each key region. The user receives feedback on the accuracy of their typing strokes from where on the fingertip a dot is felt. This feedback can be used to correct finger aim during future keypresses. Since single finger slides are ignored by the chord motion recognizer, the user can also slide a finger around the surface in tactile search of a particular key region's dot and then tap the key region when the dot is found, all without looking at the surface. Each dot should be just: large enough to be felt during tapping but not so large as to impede chord slides across the surface. Even if the dots are not large enough to impede sliding, they can still corrupt proximity and fingertip centroid measurements by raising the fingertip flesh near the dot off the surface thus locally separating the flesh from the underlying proximity sensing electrode. Therefore, in the preferred embodiment, the portion of each dot above the surface dielectric is made of a conductive material. This improves capacitive coupling between the raised fingertip flesh and the underlying electrodes.

FIG. **42** shows the steps within the keypress detection loop. Step **750** retrieves from the current identified path data **250** any paths which were recently created due to hand part touchdown or the surface. Decision diamond **752** checks whether the path proximity reached a keypress proximity thresh for the first time during the current sensor array scan. If the proximity has not reached the threshold yet or has already exceeded it previously, control returns to step **750** to try keypress detection on the next recent path. If the path just crossed the keypress proximity threshold decision diamond **754** checks whether the contact path has been identified as a finger rather than a palm. To give the users the freedom rest the palms anywhere on the surface, palm presses should not normally cause keypresses, and are therefore ignored. Assuming the path is a finger, decision diamond **756** checks whether the hand the identified finger comes from is currently performing a chord slide gesture or writing via the pen grip hand configuration. Asynchronous finger presses are ignored once these activities have started, as also indicated in step **660** of FIG. **40**A. Assuming such hand activities are not ongoing, decision diamond **757** proceeds with debounce tests which check that the finger has touched the surface for at least two sensor array scan cycles and that it had been off the surface for several scan cycles before touching down. The path tracking module (FIG. **22**) facilitates such liftoff debouncing by reactivating in step **334** a finger's old path if the finger lifts off and quickly touches back down over the same spot. Upon reactivation the time stamp of the last liftoff by the old path must be preserved for comparison with the time stamp of the new touchdown.

If all of these tests are passed, step **758** looks up the current path position ($P_x[n]$,$P_y[n]$), and step **760** finds the key region whose reference position is closest to the fingertip centroid. Decision diamond **762** checks that the nearest region is within a reasonable distance of the finger, and if not causes the finger press to be ignored. Assuming a key region is close to the finger, step **764** creates a keypress element data structure containing the path, index identifier and finger identity, the closest key region, and a time stamp indicating when the finger crossed the keypress proximity threshold. Step **766** then appends this element data structure to the tail of a FIFO keypress queue. This accomplished, processing returns to step **750** to process or wait for touchdowns by other fingers.

The keypress queue effectively orders finger touchdowns by when they pass the keypress transmitted to the host. However, an element's key symbol is not assured transmission of the host once in the keypress queue. Any of a number of conditions such as being part of a synchronized subset of pressing fingers can cause it to be deleted from the queue before being transmitted to the host. In this sense the keypress queue should be considered a keypress candidate queue. Unlike the ordered lists of finger touchdowns and releases maintained for each hand separately in the synchronization detector, the keypress queue includes and orders the finger touchdowns from both hands.

FIG. **43**A shows the steps within the keypress acceptance and transmission loop. Step **770** picks the element at the head of the keypress queue, which represents the oldest finger touchdown which has neither been deleted from the queue as an invalid keypress candidate nor transmitted its associated key symbol. Decision diamond **772** checks whether the path is still identified as a finger. While waiting in the queue path proximity could have increased so much that the identification system decides the path is actually from a palm heel, in which case step **778** deletes the keypress element without transmitting to the host and step **770** advances processing to the next element. Decision diamond **774** also invalidates the element if its press happened synchronously with other fingers of the same hand. Thus decision diamond **774** follows through on deletion command steps **601**, **612**, **615**, **620** of the

synchronization detection process (FIG. **39**). Decision diamond **776** invalidates the keypress if too much lateral finger motion has occurred since touchdown, even if that lateral finger motion has not yet caused a chord slide to start. Because users may be touch typing on the surface, several millimeters of lateral motion are allowed to accommodate glancing fingertip motions which often occur when quickly reaching for keys. This is much more glancing tap motion than is tolerated by touchpads which employ a single finger slide for mouse cursor manipulation and a single finger tap for key or mouse button click emulation.

Decision diamond **780** checks whether the finger whose touchdown created the keypress element has since lifted off the surface. If so, decision diamond **782** checks whether it was lifted off soon enough to qualify as a normal key tap. If so, step **784** transmits the associated key symbol to the host and step **778** deletes it from the head of the queue. Note that a keypress is always deleted from the queue upon liftoff, but even though it may have stayed on the surface for a time exceeding the tap timeout, it may have still caused transmission as a modifier key, as an impulsive press with hand resting, or as a typematic press, as described below.

When a keypress is transmitted to the host it is advantageous for a sound generation device on the multi-touch surface apparatus or host computer to emit an audible click or beep as feedback to the user. Generation of audible click and beep feedback in response to keypresses is well known in commercial touchscreens, kiosks, appliance control panels and mechanical keyboards in which the keyswitch action is nearly silent and does not have a make force threshold which feels distinctive to the user. Feedback can also be provided as a light on the multi-touch surface apparatus which flashes each time a keypress is sent. Keypresses accompanied by modifier keypresses should cause longer flashes or tones to acknowledge that the key symbol includes modifiers.

If the finger has not yet lifted, decision diamond **786** checks whether its associated key region is a modifier such as <shift>, <ctrl>, or <alt>. If so, step **788** advances to the next element in the queue without deleting the head. Processing will continue at step **772** to see if the next element is a valid key tap. If the next element successfully reaches the transmission stage, step **784** will scan back toward the head of the queue for any modifier regions which are still pressed. Then step **784** can send the next element's key symbol along with the modifying symbols of any preceding modifier regions.

Decision diamond **782** requires that users touch the finger on the surface and lift back off within a few hundred milliseconds for a key to be sent. This liftoff timing requirement substitutes for the force activation threshold of mechanical keyswitches. Like the force threshold of mechanical keyswitches, the timing constraint provides a way for the user to rest the finger on the key surface without invoking a keypress. The synchronization detector **14** provides another way forefingers to rest on the surface without generating key symbols: they must touch down at the same time as at least one other finger. However, sometimes users will start resting by simultaneously placing the central fingertips on the surface, but then they follow asynchronously with the pinky a second later and the thumb a second after that. These latter presses are essentially asynchronous and will not be invalidated by the synchronization detector, but as long as they are not lifted within a couple hundred milliseconds, decision diamond **782** will delete them without transmission. But, while decision diamond **782** provides tolerance of asynchronous finger resting, its requirement that fingers quickly lift off, i.e., crisply tap, the surface to cause key generation makes it very difficult to keep most of the fingers resting on the surface to support the hands while tapping long sequences of symbols. This causes users to raise their hands off the surface and float them above the surface during fast typing sequences. This is acceptable typing posture except that the users arms will eventually tire if the user fails to rest the hands back on the surface between sequences.

To provide an alternative typing posture which does not encourage suspension of the hands above the surface, decision diamond **790** enables a second key acceptance mode which does not require quick finger liftoff after each press. Instead, the user must start with all five fingers of a hand resting on the surface. Then each time a finger is asynchronously raised off the surface and pressed on a key region, that key region will be transmitted regardless of subsequent liftoff timing. If the surface is hard such that fingertip proximity quickly saturates as force is applied, decision diamond **792** checks the impulsivity of the proximity profile for how quickly the finger proximity peaks. If the proximity profile increases to its peak very slowly over time, no key will be generated. This allows the user to gently set down a raised finger without generating a key in case the user lifts the finger with the intention of generating a key but then changes his mind. If the touch surface is compressible, decision diamond **792** can more directly infer finger force from the ratio of measured fingertip proximity to ellipse axis lengths. Then it can threshold the inferred force to distinguish deliberate key presses from gentle finger rests. Since when intending to generate a key the user will normally press down on the new key region quickly after lifting off the old key region, the impulsivity and force thresholds should increase with the time since the finger lifted off the surface.

Emulating typematic on a multi-touch surface presents special problems if finger resting force cannot be distinguished reliably from sustained holding force on a key region. In this case, the special touch timing sequence detected by the steps of FIG. **43**B supports reliable typematic emulation. Assuming decision diamond **798** finds that typematic has not started yet, decision diamond **794** checks whether the keypress queue element being processed represents the most recent finger touchdown on the surface. If any finger touchdowns have followed the touchdown represented by this element, typematic can never start from this queue element. Instead, decision diamond **796** checks whether the element's finger has been touching longer than the normal tap timeout. If the finger has been touching too long, step **778** should delete its keypress element because decision diamond **786** has determined it is not a modifier and decision diamond **794** has determined it can never start typematic. If decision diamond **794** determines that the keypress element does not represent the most recent touchdown, yet decision diamond **796** indicates the element has not exceeded the tap timeout, processing returns to step **770** to await either liftoff or timeout in a future sensor array scan. This allows finger taps to overlap in the sense that a new key region can be pressed by a finger before another finger lifts off the previous key region. However, either the press times or release times of such a pair of overlapping finger taps must be asynchronous to prevent the pair from being considered a chord tap.

Assuming the finger touchdown is the most recent, decision diamond **800** checks whether the finger has been touching for a typematic hold setup interval of between about half a second and a second. If not, processing returns to **770** to await either finger liftoff or the hold setup condition to be met during future scans of the sensor array. When the hold setup condition is met, decision diamond **802** checks whether all other fingers on the hand of the given finger keypress lifted off the surface more than a half second ago. If they did, step **804**

will initialize typematic for the given keypress element. The combination of decision diamonds **800** and **802** allow the user to have other fingers of the hand to be resting on the surface when a finger intended for typematic touches down. But typematic will not start unless the other fingers lift off the surface within half a second of the desired typematic finger's touchdown, and typematic will also not start until the typematic finger has a continued to touch the surface for at least half a second after the others lifted off the surface. If these stringent conditions are not met, the keypress element will not start typematic and will eventually be deleted through either tap timeout **782** when the finger lifts off or through tap timeout **796**) if another touches down after it.

Step **804** simply sets a flag which will indicate to decision diamond **798** during future scan cycles that typematic has already started for the element. Upon typematic initialization, step **810** sends out the key symbol for the first time to the host interface communication queue, along with any modifier symbols being held down by the opposite hand. Step **812** records the time the key symbol is sent for future reference by decision diamond **808**. Processing then returns to step **770** to await the next proximity image scan.

Until the finger lifts off or another taps asynchronously, processing will pass through decision diamond **798** to check whether the key symbol should be sent again. Step **806** computes the symbol repeat interval dynamically to be inversely proportional to finger proximity. Thus the key will repeat faster as the finger is pressed on the surface harder or a larger part of the fingertip touches the surface. This also reduces the chance that the user will cause more repeats than intended since as finger proximity begins to drop during liftoff the repeat interval becomes much longer. Decision diamond **808** checks whether the dynamic repeat interval since the last typematic symbol send has elapsed, and if necessary sends the symbol again in **810** and updates the typematic send time stamp **812**.

It is desirable to let the users rest the other fingers back onto the surface after typematic has initiated **804** and while typematic continues, but the user must do so without tapping. Decision diamond **805** causes typematic to be canceled and the typematic element deleted **778** if the user asynchronously taps another finger on the surface as if trying to hit another key. If this does not occur, decision diamond **182** will eventually cause deletion of the typematic element when its finger lifts off.

The typing recognition process described above thus allows the multi-touch surface to ergonomically emulate both the typing and hand resting capabilities of a standard mechanical keyboard. Crisp taps or impulsive presses on the surface generate key symbols as soon as the finger is released or decision diamond **792** verifies the impulse has peaked, ensuring prompt feedback to the user. Fingers intended to rest on the surface generate no keys as long as they are members of a synchronized finger press or release subset or are placed on the surface gently and remain there along with other fingers for a second or two. Once resting, fingers can be lifted and tapped or impulsively pressed on the surface to generate key symbols without having to lift other resting fingers. Typematic is initiated ether by impulsively pressing and maintaining distinguishable force on a key, or by holding a finger on a key while other fingers on the hand are lifted. Glancing motions of single fingers as they tap key regions are easily tolerated since most cursor manipulation must be initiated by synchronized slides of two or more fingers.

Other embodiments of the invention will be apparent to those skilled in the art from consideration of the specification and practice of the invention disclosed herein. It is intended that the specification and examples be considered as exemplary only, with a true scope and spirit of the invention being indicated by the following claims.

What is claimed is:

1. A method of processing input from a touch-sensitive surface, the method comprising:

receiving at least one proximity image representing a scan of a plurality of electrodes of the touch-sensitive surface;

segmenting each proximity image into one or more pixel groups that indicate significant proximity, each pixel group representing proximity of a distinguishable hand part or other touch object on or near the touch-sensitive surface; and

mathematically fitting an ellipse to at least one of the pixel groups.

2. The method of claim **1** further comprising transmitting one or more ellipse parameters as a control signal to an electronic or electromechanical device.

3. The method of claim **2** wherein the one or more ellipse parameters is selected from the group consisting of position, shape, size, orientation, eccentricity, major radius, minor radius, and any combination thereof.

4. The method of claim **3** wherein the one or more ellipse parameters are used to distinguish a pixel group associated with a fingertip from a pixel group associated with a thumb.

5. The method of claim **1** wherein fitting an ellipse to a group of pixels comprises computing one or more eigenvalues and one or more eigenvectors of a covariance matrix associated with the pixel group.

6. The method of claim **1** further comprising: tracking a path of at least one of the one or more pixel groups through a time-sequenced series of proximity images;

fitting an ellipse to the at least one of the one or more pixel groups in each of the time-sequenced series of proximity images; and

tracking a change in one or more ellipse parameters through the time-sequenced series of proximity images.

7. The method of claim **6** further comprising transmitting the change in the one or more ellipse parameters as a control signal to an electronic or electromechanical device.

8. The method of claim **7** wherein the change in the one or more ellipse parameters is selected from the group consisting of position, shape, size, orientation, eccentricity, major radius, minor radius, and any combination thereof.

9. The method of claim **6** wherein fitting an ellipse to the one pixel group comprises computing one or more eigenvalues and one or more eigenvectors of a covariance matrix associated with the pixel group.

10. A touch-sensing device comprising:

a substrate;

a plurality of touch-sensing electrodes arranged on the substrate;

electronic scanning hardware adapted to read the plurality of touch-sensing electrodes;

a calibration module operatively coupled to the electronic scanning hardware and adapted to construct a proximity image having a plurality of pixels corresponding to the touch-sensing electrodes; and

a contact tracking and identification module adapted to:

segment the proximity image into one or more pixel groups, each pixel group representing proximity of a distinguishable hand part or other touch object on or near the touch-sensitive surface;

and

mathematically fit an ellipse to at least one of the one or more pixel groups.

11. The touch-sensing device of claim 10 further comprising a host communication interface adapted to transmit one or more ellipse parameters as a control signal to an electronic or electromechanical device.

12. The touch-sensing device of claim 11 wherein the touch-sensing device is integral with the electronic or electromechanical device.

13. The touch-sensing device of claim 11 wherein the one or more ellipse parameters comprise one or more parameters selected from the group consisting of position, shape, size, orientation, eccentricity, major radius, minor radius, and any combination thereof.

14. The method of claim 13 wherein the one or more ellipse parameters are used to distinguish a pixel group associated with a fingertip from a pixel group associated with a thumb.

15. The touch-sensing device of claim 10 wherein the contact tracking and identification module is adapted to compute one or more eigenvalues and one or more eigenvectors to fit the ellipse.

16. The touch-sensing device of claim 10 wherein the contact tracking and identification module is further adapted to:

track a path of one or more pixel groups through a plurality of time-sequenced proximity images;

fit an ellipse to at least one of the one or more pixel groups in a first proximity image of the plurality of time-sequenced proximity images; and

track a change in one or more ellipse parameters associated with the fitted ellipse through two or more of the time-sequenced proximity images.

17. The touch-sensing device of claim 16 further comprising a host communication interface adapted to transmit the change in at least one of the one or more ellipse parameters as a control signal to an electronic or electromechanical device.

18. The touch-sensing device of claim 17 wherein the touch-sensing device is integral with the electronic or electromechanical device.

19. The touch-sensing device of claim 17 wherein the change in one or more ellipse parameters used as a control input to an electronic or electromechanical device comprises one or more parameters selected from the group consisting of position, shape, size, orientation, eccentricity, major radius, minor radius, and any combination thereof.

20. The touch-sensing device of claim 16 wherein the contact tracking and identification module is adapted to compute one or more eigenvalues and one or more eigenvectors to fit the ellipse.

21. The touch-sensing device of any one of claims 10-12 and 16-18 wherein the touch-sensing device is fabricated on or integrated with a display device.

22. The touch-sensing device of claim 21, wherein the display device comprises a liquid crystal display (LCD) or a light-emitting polymer display (LPD).

23. A computer-readable medium having embodied thereon instructions executable by a machine to perform a method according to any of claims 1-9.

24. A touch-sensing device comprising:

means for producing a proximity image representing a scan of a plurality of electrodes of a touch-sensitive surface, the proximity image having a plurality of pixels corresponding to the touch-sensing electrodes; and

means for segmenting the proximity image into one or more pixel groups, each pixel group representing a touch object on or near the touch-sensitive surface; and

means for fitting an ellipse to at least one of the pixel groups.

25. The touch-sensing device of claim 24 wherein the touch object comprises at least a portion of a hand.

26. The touch-sensing device of claim 24 wherein the touch object comprises at least a portion of one or more fingers.

27. The touch-sensing device of claim 24 wherein the touch object comprises at least a portion of a body part.

28. The touch-sensing device of claim 27 wherein the body part comprises one or more of a hand, a finger, an ear, or a cheek.

29. The touch-sensing device of claim 24 further comprising means for transmitting one or more ellipse parameters as a control signal to an electronic or electromechanical device.

30. The touch-sensing device of claim 27 wherein the touch-sensing device is integral with the electronic or electromechanical device.

31. The touch-sensing device of claim 24 further comprising:

means for tracking a path of one or more pixel groups through a plurality of time-sequenced proximity images;

means for fitting an ellipse to at least one of the pixel groups in a plurality successive proximity images; and

means for tracking a change in one or more ellipse parameters through a plurality of time-sequenced proximity images.

32. The touch-sensing device of claim 29 further comprising means for transmitting the change in the one or more ellipse parameters as a control signal to an electronic or electromechanical device.

33. The touch-sensing device of claim 32 wherein the touch-sensing device is integral with the electronic or electromechanical device.

34. The touch-sensing device of any one of claims 24 and 29-33 wherein the touch-sensing device is fabricated on or integrated with a display device.

35. The touch-sensing device of claim 34, wherein the display device comprises a liquid crystal display (LCD) or a light-emitting polymer display (LPD).

* * * * *