A method for depth-based hand tracing

Khoa Ha

University of Maryland, College Park
khoaha@umd.edu

Abstract

An algorithm for natural human-computer interaction via in-air drawing is detailed. We discuss using depth information to control the drawing state and methods to handle multiple hand interactions.

I. Background

A system which traces in-air finger movements would provide a natural input for schematics, figures, and artwork. It streamlines user-interaction by eliminating physical inputs without relying on special gestures to trigger actions. This system requires only a stereo camera, making it compact and robust.

Similar existing systems use different techniques for hand-tracking which make interaction less natural. Some require markers to be worn on the fingers to locate their spatial position [1]. Other methods for detecting hands rely on identifying the finger contours when the hand is open [2] or fitting a 3D model to the hand [3]. The former breaks down when the hand is pointed at the camera most of the time, and the latter is unable to track an arbitrary number of hands, which is desirable. To actually trace the finger existing systems require the user to gesture with extra fingers to trigger a drawing action [4] [5], which is unnecessarily limiting.

A more natural solution would be to track near points in the depth matrix, automatically detect the desired depth the user wants to draw at based on the motion of the hand, and trace its path as it intersects with the plane. The analogue of a surface removes the need for trigger gestures and increases the ease of use.

II. Related work

Kinect Paint [6], developed by IdentityMine, is a finger-painting simulator that traces in-air movements. It determines the position of the hands using skeletal mapping and uses gestures mapped to the left hand to trigger drawing with the right hand. The reliance on a skeleton requires the user to be within a set distance, and reserving the left hand for gestures prevents users from drawing with it.

Point 2 Paint [7] is a Kinect-based art installation created by Hatchd and Adapptor. It appears to use a drawing plane for input, but its position seems fixed rather than specified by the user. Additionally, it looks like all points which intersect the drawing plane trigger drawing. Since the installation was a controlled environment, the program also does not have to handle interfering foreground objects.

III. Overview

The algorithm can be divided into five main parts for each frame:

1. **Measure depth changes** — determine the change in depth from the previous frame for every pixel in the depth matrix

2. **Segment** — separate all points which lies within a certain depth of the minimum peak into blobs

3. **Select hands** — classify whether these blobs belong to hands

4. **State check** — switch in/out of drawing mode as appropriate

5. **Associate points** — match points from the current frame to previous paths

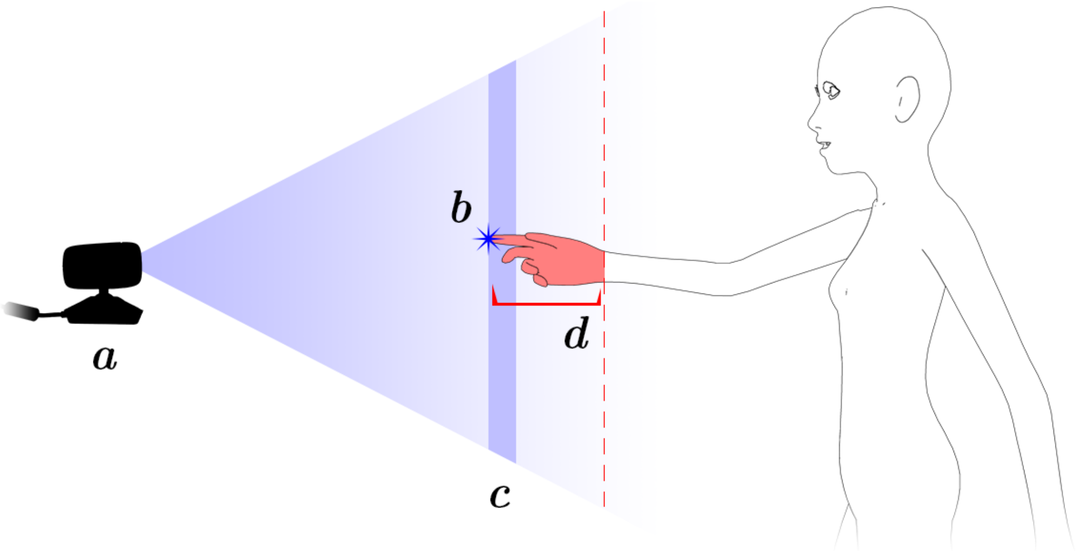

Figure 1: Side view of user interaction. a) Kinect sensor. b) The user's finger, corresponding to the minimum peak in the depth array. c) The depth threshold, where active points register as traces. d) The hand buffer, a space a fixed distance from the drawing depth where points are extracted to form blobs (red).

Once per frame we extract the closest moving point in the depth matrix, the "minimum peak", which is assumed to be on the user's hand. Drawing is activated when the minimum peak's distance to the camera stops decreasing, indicating that the user wants to draw at that depth. This depth becomes the "drawing depth" and afterwards if any hands move within a certain distance or "depth threshold" from this plane we register it as an interaction and trace its path. The minimum points which intersect with the depth threshold are "active points". Drawing ceases when the minimum peak leaves the depth threshold, indicating there are no active points left. If afterwards the user approaches and stops again within a certain "registration buffer" of the original drawing depth, it is registered as a second contact and drawing resumes with the new drawing depth set to the minimum peak.

IV. Segment

Measure depth changes

To minimize interference from stationary objects, we take the depth difference between the current frame and the previous frame and only consider points which move in-between frames. Since there is naturally noise in the depth array, we classify any changes below a certain threshold as static. The minimum peak is taken to be the closest point out of those that moved to prevent foreground objects from blocking user interaction.

Derive blobs

After taking the minimum point from the set of points with changed depths, we extract all points which lie within the "hand buffer", a region behind the minimum point spanning roughly the length of a hand. Contiguous points form discrete candidate blobs which we classify based on their likeliness to be a hand.

V. Select hands

To keep the system robust, the only qualification for a blob to be classified as a hand is for its pixel area to depth ratio to be within δ units of a *targetValue*. Specifically, this must hold true:

$$\left|\sqrt{blobArea} \cdot blobDepth - targetSize\right| < \delta$$

This allows all moving objects about the size of a hand to provide input. Afterwards, the minimum depths for each hand are found to form the active points for the frame.

VI. State check

We estimate the discrete velocity of the minimum peak by taking its depth difference between frames. If the velocity changes from negative (approaching the camera) to non-negative (moving away or stopping) and no previous contact was established, we register that the user is touching. If the drawing depth has already been established via a prior interaction, the minimum peak must stop within the registration buffer to trigger a new action, after which the drawing depth is recreated at the minimum peak. This makes it easier for the user to stop and resume drawing without having to be at the exact depth as before. Conversely, if the user is already in contact and the minimum peak leaves the depth threshold we register that the user is no longer interacting with the drawing plane, since any potential active points would have to have depth greater than or equal to the minimum peak.

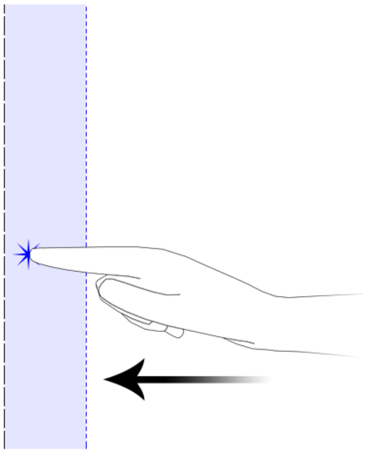

Figure 2: Hand enters depth threshold and triggers drawing.

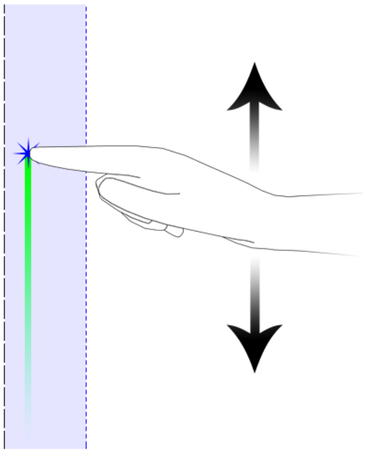

Figure 3: User moves hand around inside threshold, tracing a path.

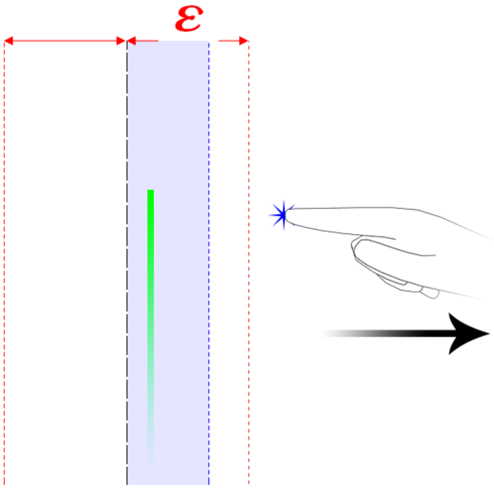

Figure 4: Hand exits threshold and registration buffer is extended ϵ units around the drawing depth.

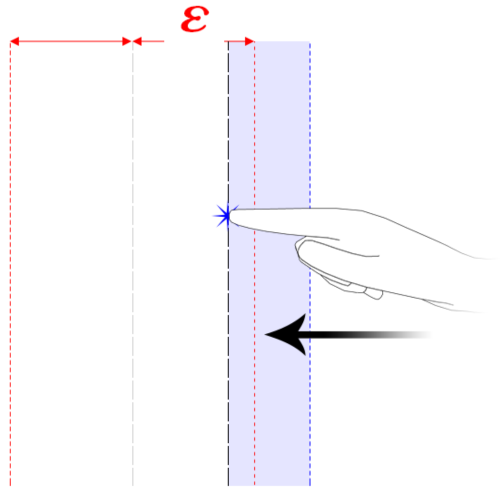

Figure 5: Hand enters registration buffer and new drawing depth is established at the minimum peak.

VII. Associate points

For the current active points, greedy matching is performed in an attempt to connect each point with their nearest match from the previous frame. [8] Only active points which show motion are appended to paths.

```
for $point \in activePoints$ do
    if depthDifference($point$) $> 0$ then
        continue
    end if
    $nearest \leftarrow None$
    $minDist \leftarrow Infinity$
    for $other \in prevPoints$ do
        $dist \leftarrow |point - other|$
        if $dist < minDist$ then
            $nearest \leftarrow other$
            $minDist \leftarrow dist$
        end if
    end for
    connect $point$, $nearest$
    $prevPoints \leftarrow prevPoints \setminus nearest$
end for
```

If a match is found, the point is considered a continuation of a previous path. Otherwise, it is regarded as the starting point of a new path.

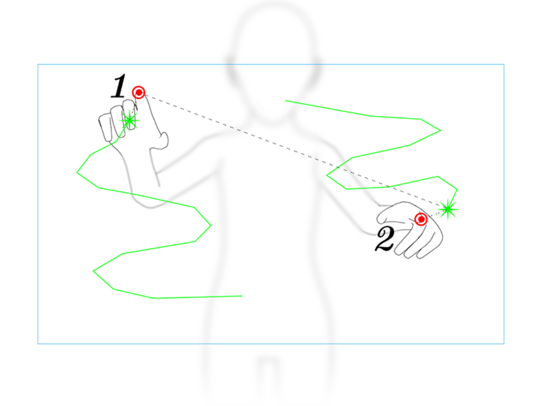

Figure 6: Frontal view of user interaction. Active points are highlighted in red. Point 1 is checked first and compared to both leading points of the previous paths. It is matched with the left one since it is closer. Since there is only one leading point left when checking Point 2, it extends the right path.

VIII. Suppressing noise

The depth matrix used is prone to error since it is calculated from the images received by the Kinect camera and these cameras have a relatively low resolution of 640x480. Because the minimum peak is usually a small point, like a finger, its depth is easily lost to the camera. In addition, there are some fluctuations in the depth data due to noise, particularly at further distances.

To prevent spikes in the depth matrix from registering as false minimum points, a Gaussian filter is applied to smooth the depth data. Additionally, when we observe the change in depth between frames we check for each pixel:

$$|pixel.z - prevPixel.z| < noiseTolerance \cdot pixel.z$$

If this is false, we classify the change as background noise and ignore it. The cutoff scales with depth since noise affects further points to a greater degree.

Sometimes smoothing is not enough, and hands may stop being registered between frames due to rapid motion, etc. When this occurs it can lead to false matches in the greedy tracking method for connecting paths. To combat this, a maximum cutoff is established for the distance a hand can jump between frames and an active point cannot extend an existing path if its distance to the path's last point is greater than the cutoff.

IX. Results

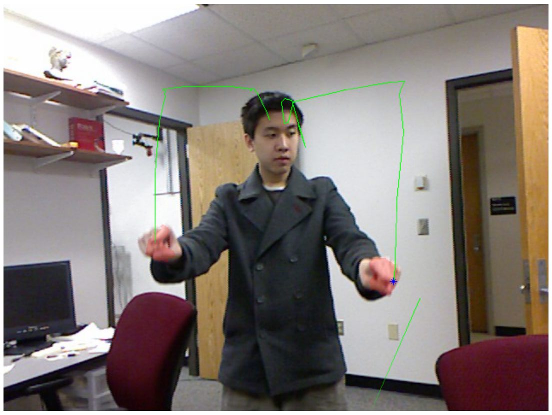

Figure 7: Screenshot from the program showing a two-handed trace of a square in progress. The blue point is the minimum peak, the red areas are detected hands and the green lines are the trace.

We introduced various parameters for noise-suppression, however if the amount of suppression is too high the system becomes insensitive to desired user input. If the sigma used for depth smoothing is high, the peaks may become so dull that it might not catch when the user interacts with the drawing plane. If we increase the percentage below which depth changes will be considered background noise, subtle movements of the hand may be discarded. Likewise, decreasing the maximum allowed movement for a hand between frames will exclude legitimate quick motions.

In its current state the system does not have a problem recognizing smooth, continuous movements. Point traces, as in when the user pokes the drawing plane, present a problem as the timeframe for execution is very short so there aren't many frames capture the intent. Unrelated paths may also converge between the leading points of two nearby hands when an active point "flickers" in and out of the program's consideration, either due to lack of motion or being close to the depth cutoff. Blobs can also become divided if the hand is ill-posed. For example, the inside of the hand can read inaccurate depths if the hand is cupped. In these cases the hand may not be recognized. Foreground objects are mostly ignored due to motion detection, but can sometimes leave traces if they have a hand-sized section within the drawing buffer while other objects are tracing.

X. Future Work

We want to work on improving tracking between frames, which would be easier with higher frame rate data. The method for classifying hands could also be made more rigorous by tracking the user's skeleton to find the area to segment, predicting their trajectories, etc. Lastly, some parameters such as the maximum travel distance per frame and the depth threshold need to be adjusted for best results depending on the physical setup. It would be nice to auto-configure these settings in the future.

References

[1] S. Agravat and P. G. Pandey, "Finger painting using computer vision," *International Journal of Societal Applications of Computer Science*, vol. 2, pp. 402–406, 2013.

[2] F. T. Cerezo, "3d hand and finger recognition using kinect."

[3] N. Kyriazis, I. Oikonomidis, and A. Argyros, "Giving a hand to kinect," 2012.

[4] M. Isard and J. MacCormick, "Hand tracking for vision-based drawing."

[5] O. Lavi, L. Kagan, H. Yeshurun, and G. Kimmel, "Workshop in computer vision."

[6] IdentityMine, "Kinect paint."

[7] A. Hatchd, "Point 2 paint."

[8] F. Wang, X. Ren, and Z. Liu, “A robust blob recognition and tracking method in vision-based multitouch technique,” pp. 971–974, 2008.