Real-Time Gesture Detection for Multi-Touch Devices

Francisco R. Ortega[1], Armando Barreto[2], Naphtali Rishe[1], Malek Adjouadi[2], Fatemeh Abyarjoo[2].

School of Computer Information Sciences[1]. | Department of Electrical & Computer Engineering[2].

Abstract

We are motivated to seek a fast and accurate multi-touch gesture detection algorithm that can be utilized for 3D navigation. Our current approach tries to solve the online gesture detection for multi-touch devices for translation, rotation and zooming gestures.

Introduction

Motivation

- Find a fast gesture recognition algorithm
  - For high-demanding 3D navigation systems.
  - Perform gesture detection in real-time.
  - Easy to Implement in comparison to Neural Networks or Hidden Markov Models
- Our work is inspired by previous contributions:
  - $1 Algorithm.
  - The Rubine Algorithm.

Setup

Software

- Visual Studio 2012
- C++11
- Microsoft Parallel Patterns Library
- Win32 API
- Windows 7

Hardware

- 3M Multi-Touch Monitor 22''

Touch Definitions

Raw Touch Data

- We use Raw Touch Data and store them in a set called **trace**
- **Trace** starts when a user presses with one finger and continues until the user removes the finger.

Touch Events

- **TOUCHDOWN:** Trace has begun.
- **TOUCHMOVE:** Trace in progress. (Finger moving or static).
- **TOUCHDOWN:** Trace has ended. (Finger lifted)

TOUCHPOINT Data Structure

- 2D Coordinates **x** and **y**.
- Initial Timestamp t_0 and final timestamp t_1.
- Boolean p indicating if the touch point was already processed.

TRACE Data Structure

- Unique identifier id.
- 2D Coordinates **x** and **y**.
- Initial timestamp t_i and final timestamp t_f.
- Boolean d to see if trace is ready for deletion.

Touch System

- Multi-Threaded
- Uses Concurrent Hash-Map
- Uses Concurrent Vector

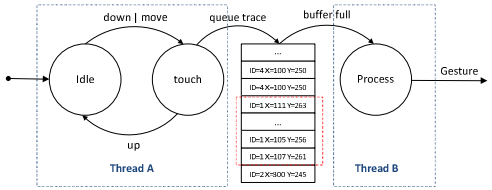

Fig.1 Touch State System

Touch Events

TOUCHDOWN

- Create a trace vector for the new trace.
- Process first point.
- Add point to trace.
- Add trace to queue.

TOUCHMOVE

- Remove noise.
- Add new touch point to trace.
- Add trace to queue.

TOUCHUP

- Remove trace from queue.

Algorithm 1 TOUCHMOVE

```
1:  trace ← TRACE(id)
2:  vtraces ← mtraces.find(id)
3:  noise ← removeNoise(trace, vtrace)
4:  if noise then
5:      trace.c+ = 1
6:      trace.t1 = trace.requestTimeStamp()
7:  else
8:      trace.t1 ← trace.requestTimeStamp()
9:      trace.t0 ← vtraces[length − 1].t0
10:     vtrace ← mtraces.getValue()
11:     vtrace.push_back(traces)
12: end if
13: mtraces.insert(id, vtrace)
```

Gesture Detection Algorithm

Gesture Detection Algorithm

- Divide window size into top and bottom.
- Obtain the grip for each top and bottom.
- Calculate the spread distance for the first trace
- For each trace
  - Calculate top and bottom difference between trace average and grip.
  - Calculate top and bottom average distance between each average trace and its grip.
  - Save Average for each spread.
  - Calculate rotation angle.
  - Save average for each rotation.
- Return gesture with highest value.

Algorithm 2 GestureDetection

```
1:  top ← traces.getTop(windowSize)
2:  bottom ← traces.getBottom(windowSize)
3:  tGrip.x ← top.getGrip.x
4:  tGrip.y ← top.getGrip.y
5:  bGrip.x ← bottom.getGrip.x
6:  bGrip.y ← bottom.getGrip.y
7:  spread.x ← iTrace[1].x − iGrip.x
8:  spread.y ← iTrace[1].y − iGrip.y
9:  swipeDistance ← sqrt(spread.x^2 + spread.y^2)
10: for t = 1 to traces.Count do
11:     i.x ← tTrace[t].x − tGrip.x
12:     i.y ← tTrace[t].y − tGrip.y
13:     f.x ← bTrace[t].x − bGrip.x
14:     f.y ← bTrace[t].y − bGrip.y
15:     di ← sqrt(i.x^2 + i.y^2)
16:     df ← sqrt(f.x^2 + f.y^2)
17:     iSpread ← iSpread + di
18:     fSpread ← fSpread + df
19:     angle ← atan2(f.y − i.y, f.x − i.x)
20:     rotAngle ← rotAngle + angle
21: end for
22: iSpread ← iSpread/traces.Count
23: fSpread ← fSpread/traces.Count
24: rotAngle ← rotAngle/traces.Count
25: zoomDistance ← fSpread − iSpread
26: rotDistance ← rotAngle/360.0 ∗ 2 ∗ π ∗ swipeDistance
27: return Gesture With Highest Distance
```

Demo

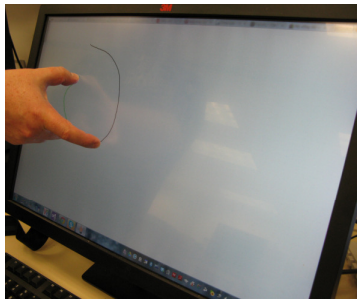

Fig.2: 3M Multi-Touch Monitor.

Visualizing the Algorithm

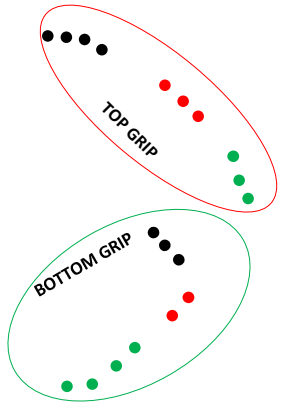

Fig.3: Top and Bottom grips.

Conclusions

Future Work

- Add additional gestures to our recognition algorithm.
- Test it with a 3D high-Demanding navigation system.
- Look into GPU computing for some of our computations.

Conclusion

- Shown how to find gestures in real-time with a simple algorithm.

Acknowledgments

This work was sponsored by NSF grants HRD-0833093,and CNS-0959985. Mr. Francisco Ortega is the recipient of a GAANN fellowship, from the US Department of Education, at Florida International University.

FLORIDA INTERNATIONAL UNIVERSITY

Email: forte007@fiu.edu