HCI – Project 1: FingRRR

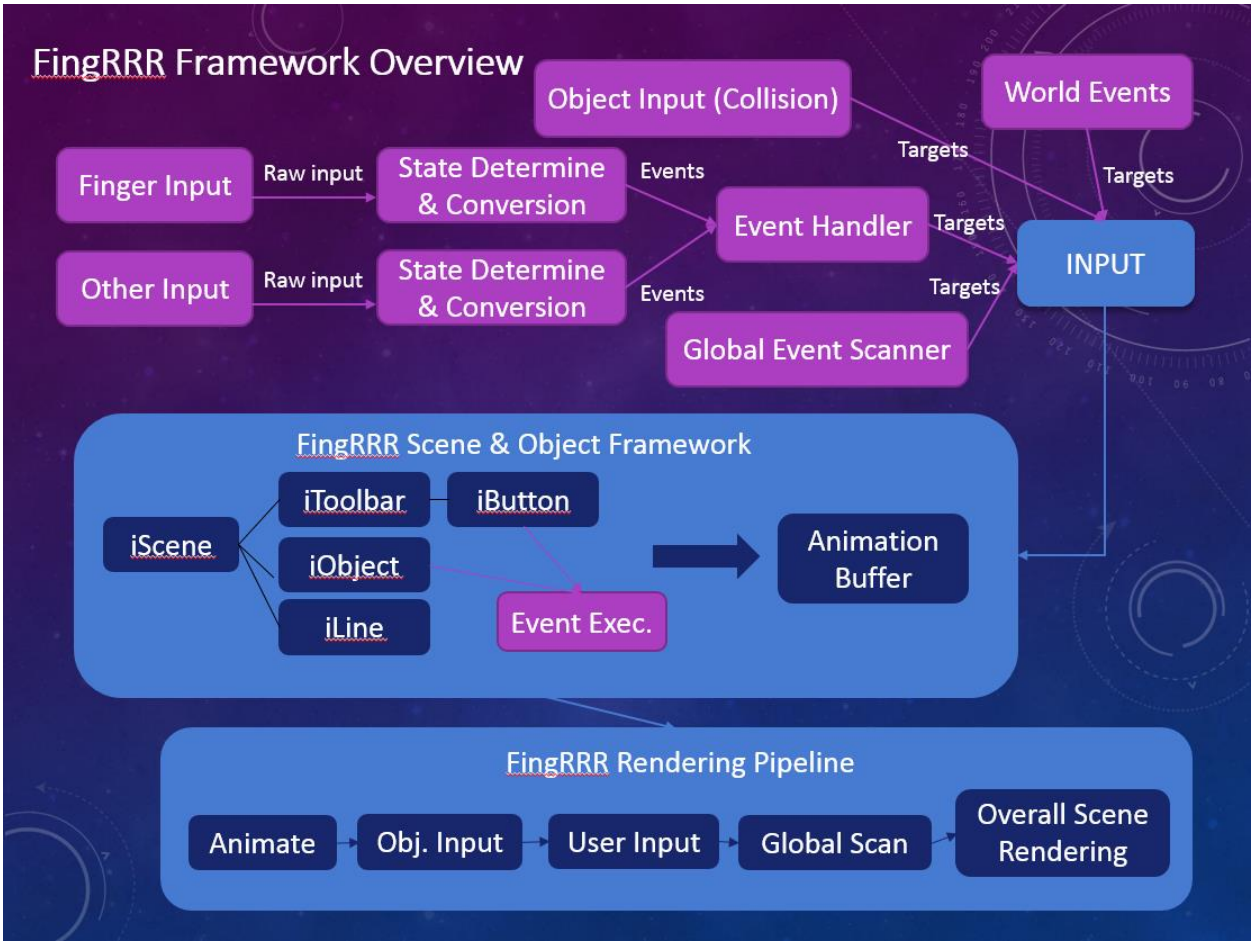
FingRRR, a 2D interaction framework structured on OpenCV

Name: 陳柏翰, Student ID: 0113110, Major: 資工系資工組二年級

Premise:

OpenCV is extremely feature-poor when it comes to UI. Therefore, building a highly extensible, reliable framework for 2D interfaces (scenes, objects) and animations is what comes to mind when I was trying to build my game.

Overview:



Concepts (Interaction):

- Mouse simulator: Using finger(s) to simulate mouse behavior, and thus allowing more advanced and intuitive interaction. (Used red ball instead, but actually more difficult than I thought)
- The interaction is done on a user-decided imaginary surface.

- Key factor: Size of red ball. (ie. red ball size gets bigger when closer to screen)
- Obtain sizes, distances, timing and issue "fingerState" (later converted to equivalent mouse events.)

Concepts (Framework):

- Reliability: The whole framework took reliability into serious consideration deep from the bottom. I spent lots of hours just to tweak the codes/methods written.
- Extensibility: Users are even able to implement custom interaction methods (keyboard, gestures, or even the real mouse … etc.) and input them into the framework with ease. Framework would still function perfectly.
- Efficiency: No matrix is stored in the classes. Matrices are only used in red ball detection and final rendering. Pointers, reference variables are used very frequently.
- Customizability: Several genres of games were being thought of during the designing phase. Most games should be able to be implemented on this framework with relative ease.

HCI-related topic – Initialization

- Goal: Detect the finger (red ball) size on a user-decided imaginary surface.
- Approach: A method similar to k-means.
- Algorithm:

---

while (inputRedBallSize = input())

find if inputRedBallSize belongs to any existing groups (<= thres_err (= 0.1))
if yes, (inputRedBallSize belongs to G(i))
calculated the new weighted average for G(i)
if number of members in G(i) > thres_init (=100)
exit, return Avg(i), algorithm done.
if not, create a new group

HCI-related topic - Issues with Red Ball Detection

Since the interaction is heavily-based on the size(area) of the red ball, the detection has to be extremely accurate.

(1) Noise

Noise is very hard to deal with. Size and Shape varies.

Size: when a user moves the red ball away from the screen, sometimes the ball becomes so small that it can be easily confused with actual noise.

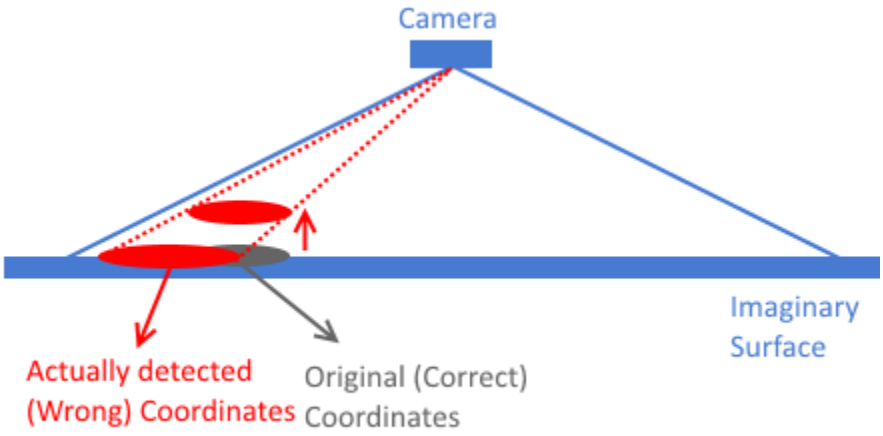
Shape: when a user moves the red ball, the shape is sometimes twisted.

Solution: Environment based. Different rule sets for each environment. Storing the previous correctly-detected coordinates also helps.

(2) Coordinates

This problem is easily noticeable especially on the edges (of screens).

When a user moves the red ball forward, due to projection issues, the center coordinates of the red ball are actually incorrect.



Solution: Similar triangles.

Finger area detected from initialization and the newly obtained area can be used to describe the ratio of distances. Then, draw a line perpendicular to the surface from the camera. The following calculation would become rather straightforward.

```
function fixCoord(int &x, int &y)

midX = camWidth / 2, midY = camHeight /2;
lFront = sqrt(fingerArea), lRear = sqrt(fingerInputArea);
x = midX + (x - midX) * (lFront / lRear);
y = midY + (y - midY) * (lFront / lRear);
```

User Aspect Coordinates: Mirror X

```
x = camWidth - x;
```

Determining Finger State.

Define diffAreaN(a1, a2) = (sqrt(a1) - sqrt(a2)) / sqrt(a2).

Newly detected area = a1.

Finger area detected from = a2.

- Finger = Up, if diffAreaN(a1, a2) < thres_finger_down (=0.2)
- Finger = Down, if if diffAreaN(a1, a2) >= thres_finger_down (=0.2)
- Finger = Hold, if Finger = Down for more than thres_hold ms(=300ms)

Converting States to Events

Current State = Down or Hold

- INPUT_DOWN, if Previous State = Up
- INPUT_DRAG, if User starts holding. (>= 300ms)
- INPUT_HOLD, if Previous State = Hold

Current State = Up

- INPUT_HOVER, if Previous State = Up
- INPUT_UP, if Previous State = Down
- INPUT_RELEASE, if Previous State = Hold

Demonstration