FINGRRR

A FRAMEWORK BASED UPON OPENCV

WHY "FINGRRR" ?

- FINGRRR = FINGER + RRRRRRRRRRRRRRRRRRRRRRRRR …
- Interaction with finger(s) - *used a red ball for this Prj.
- RRRRRRRRRRRRRRRRRRRRRRRRRR

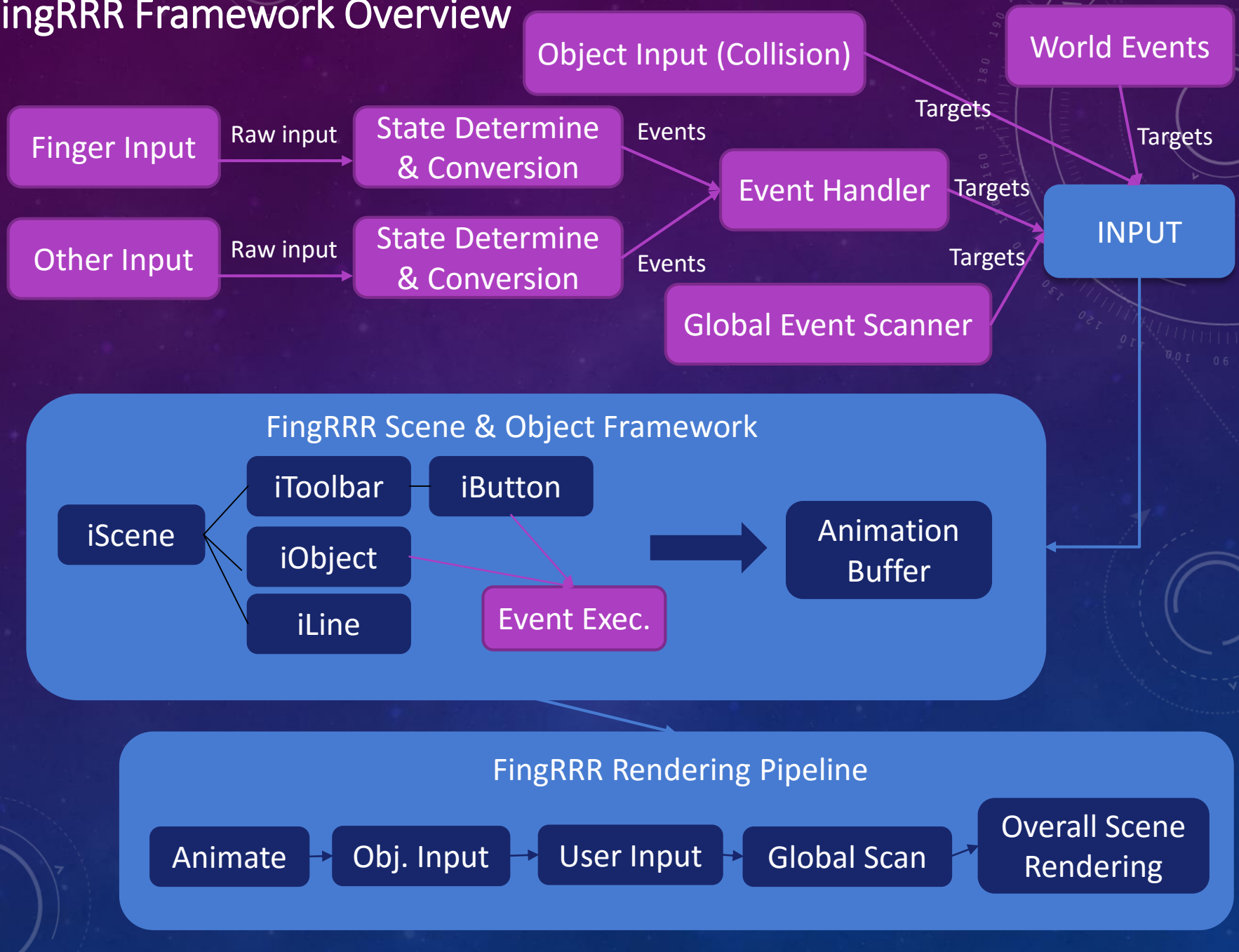
- Framework: Modularization + Extensibility (It's highly extensible)
- * Issues: Single-threaded program, Smoothness is bound with camera FPS.

CORE CONCEPTS

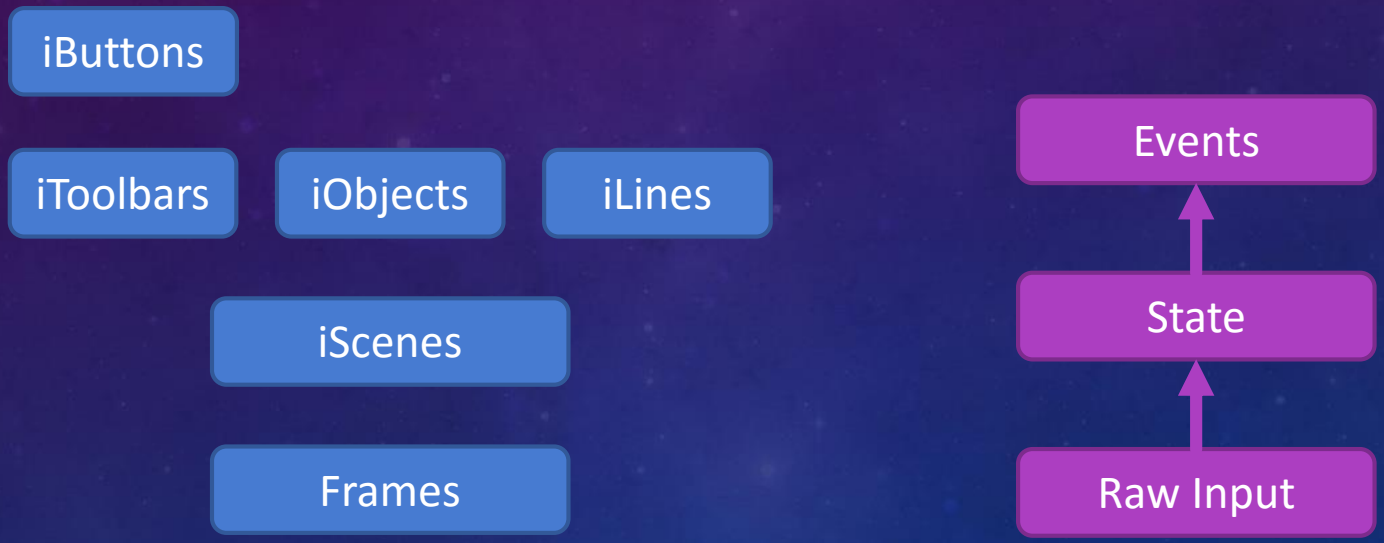
- Mouse simulator: Using finger(s) to simulate mouse behavior
- Key factor: Size of red ball. (ie. red ball size gets bigger when closer to screen)
- Obtain sizes, distances, timing and issue "fingerState" (later converted to equivalent mouse events.)

- … As it turns out, It isn't as easy as I thought it would be. (RRRRRR ….)

FingRRR Framework Overview



FINGRRR FRAMEWORK ITEM HIERACHY



USAGE

- Steps:
- 1. Initialization
- 2. Create Scenes, Toolbars(buttons) and Objects.
- 3. Specify active Scene and active Frame.
- 4. While-loop to input camera frames, execute FingRRR rendering pipeline.
- 5. Switch between Scenes and Frames, dynamically modify scenes.

STEP 1: 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
```

STEPS 2, 3: FRAMEWORK USAGE

```
iScene *main = new iScene(0, 0, camWidth, camHeight, 255, 255, 0, &bgImgScene);
sceneActive = main;
frameActive = &frameGame;
iToolbar *mainTB = main->createToolbar(0, 0, 640, 150);
mainTB->createButton(255, 0, 0, &bgImgBtn, onBtnHover, onBtnClick, NULL, NULL, NULL, NULL, onBtnOut);
mainTB->createButton(0, 255, 0, &bgImgBtn, onBtnHover, clickMoveTest);
mainTB->createButton(0, 0, 255, &bgImgBtn, onBtnHover, onBtnClick);

test = sceneActive->createObject(200, 200, 50, 50, 0, 0, 255, NULL, true, true);
test->onDrag = &onDrag;

sceneActive->render(frameGame);
imshow(windowNameGame, frameGame);
```

iButtons

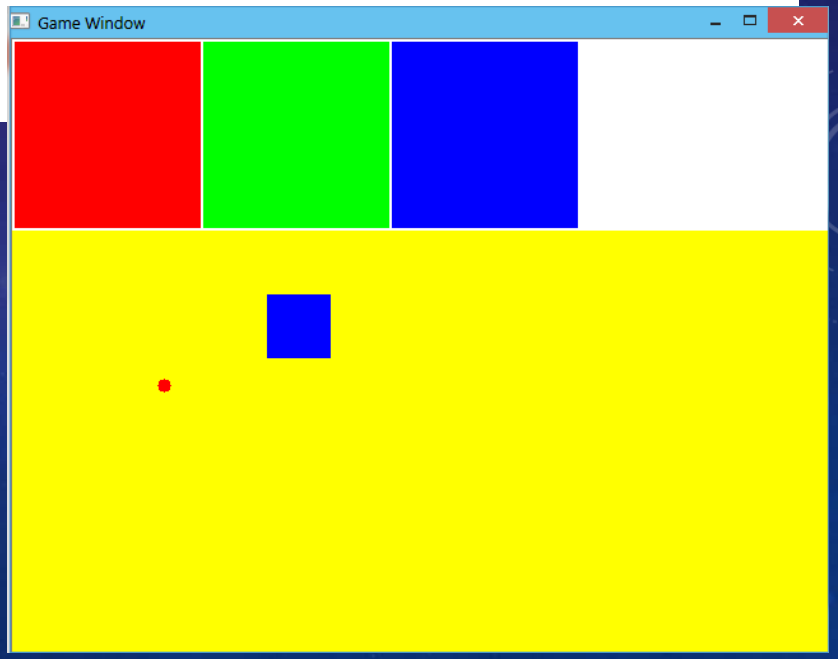
iToolbars

iObjects

iLines

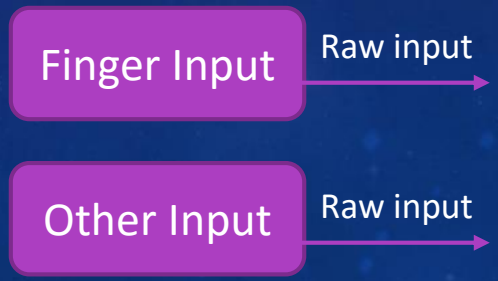
iScenes

Frames

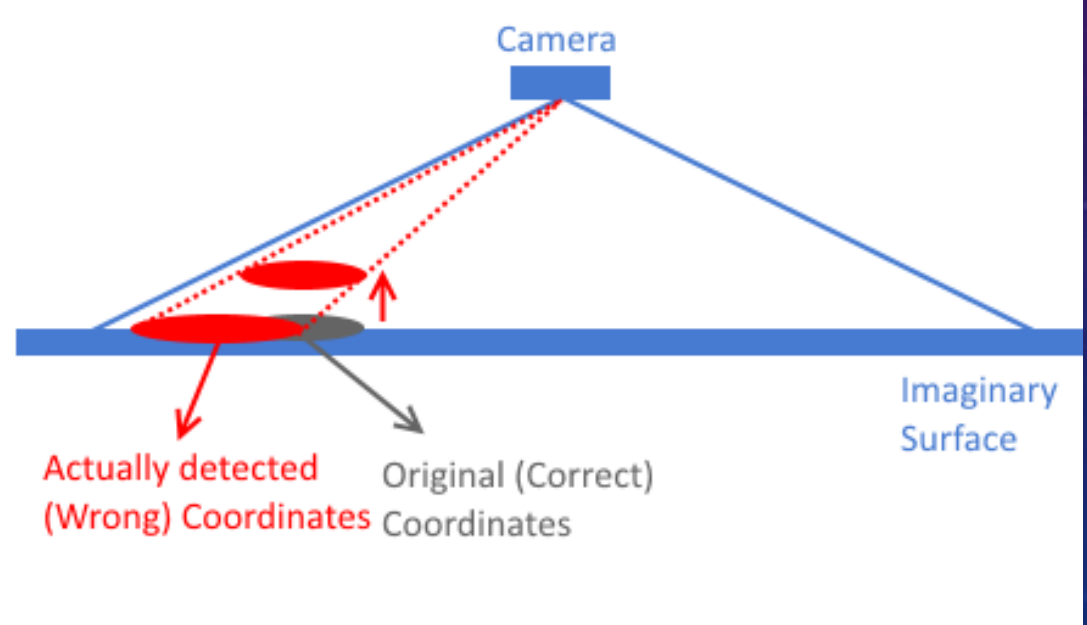


STEP 4: ISSUES WITH RED BALL DETECTION

- Since the interaction is heavily-based on the size(area) of the red ball, the detection has to be extremely accurate.
  - Noise is very difficult to deal with. (size and shape varies)
  - Coordinates issues (elaborate later)
- Unsolvable issues



STEP 4: ISSUES WITH RED BALL DETECTION

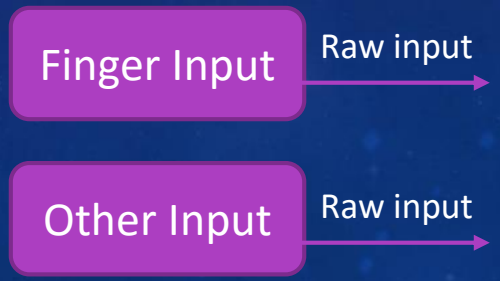


Finger Input → Raw input

Other Input → Raw input

STEP 4: FIXING COORDINATES

- Define function fixCoord(int &x, int &y)
- Fixing incorrect coordinates due to distances
  - 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;



STEP 4: 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)

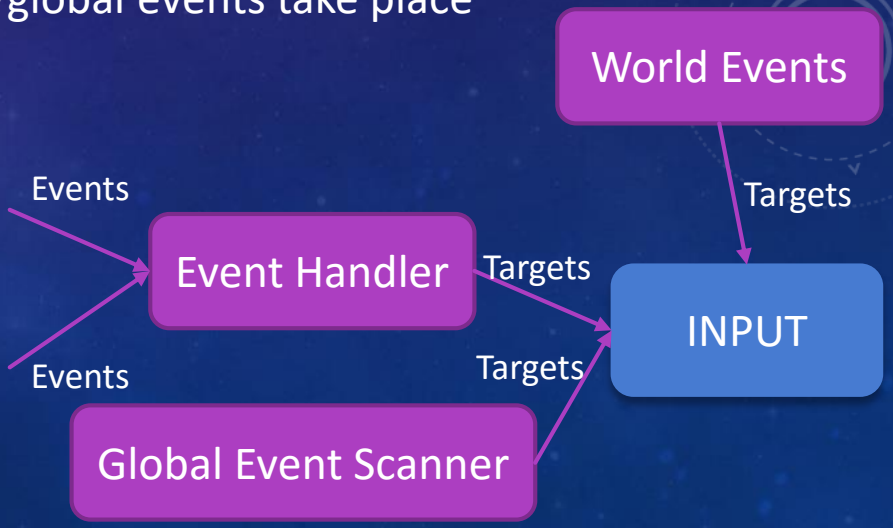
STEP 4: CONVERTING STATES TO EVENTS

- #define INPUT_HOVER 0
- #define INPUT_DOWN 1
- #define INPUT_UP 2
- #define INPUT_DRAG 3
- #define INPUT_HOLD 4
- #define INPUT_RELEASE 5
- #define INPUT_NONE 10 // global event
- #define INPUT_OUT 11 // global event

```
void determineFingerState()
{
    int prevState = fingerState;
    if (isFingerDown(fingerInputArea))
    {
        if (prevState == FINGER_UP) // up
        {
            fingerState = FINGER_DOWN;
            fingerDownStart = clock();
            inputHandler(sceneActive, INPUT_DOWN, fingerP
        }
        else if (prevState == FINGER_DOWN) // down
        {
            if (clock() - fingerDownStart >= thres_hold)
            {
                fingerState = FINGER_HOLD;
                inputHandler(sceneActive, INPUT_DRAG, fin
            }
            /* else
                fingerState = FINGER_DOWN; */
        }
        else if (prevState == FINGER_HOLD)
            inputHandler(sceneActive, INPUT_HOLD, fingerP
    }
    else
    {
        fingerState = FINGER_UP;
        if (prevState == FINGER_UP)
            inputHandler(sceneActive, INPUT_HOVER, finger
        else if (prevState == FINGER_DOWN)
            inputHandler(sceneActive, INPUT_UP, fingerPos
        else if (prevState == FINGER_HOLD)
            inputHandler(sceneActive, INPUT_RELEASE, fing
    }
    return ;
}
```

STEP 4: FINALLY, REGISTERING EVENTS

- Event Handler deals with (Local) events
- Local events = events triggered on objects.
- Global events = events triggered based on objects. (possibly not on objects)
- World events = triggered whenever local/global events take place



Mercy Please