Cart-pole Render Code

Overview

This is the code used to draw the cart-pole pixels for the screen at each frame. At first, we adopted the old-sequential style of drawing each pixel: row by row like the cathode ray tube. After the implementation of VRAM, we were free to draw the pixels using (x, y) coordinates, buffering in VRAM. Then the VRAM is dumped as a frame.

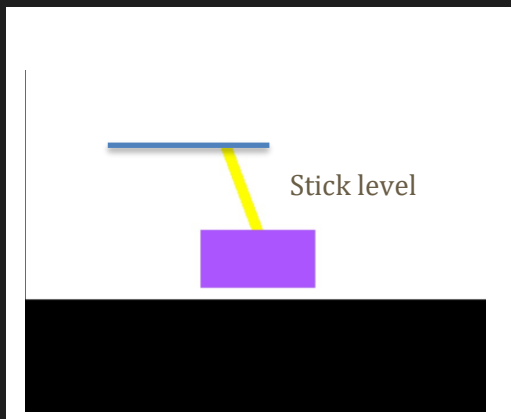
Code Visualized

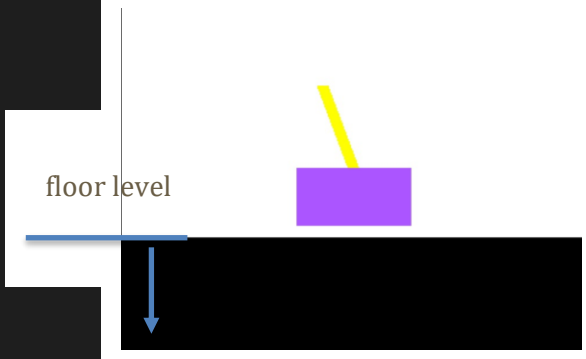
```
@@@@@@    Colors   @@@@@@

ivar floorColor = 0 ;
ivar wheelColor = 18688 ;
ivar cartColor = 17792 ;
ivar stickColor = 60416 ;
ivar skyColor = 65408 ;

@@@@@@    Logic   @@@@@@

@@@ Monitor dimensions
fvar sw = 399.0 ;
fvar sh = 299.0 ;
```

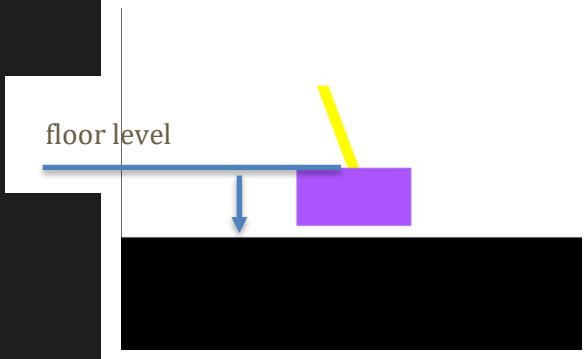
```
fvar pi = 22.0 / 7.0 ;


@@ Cart-pole and floor levels
fvar floorHeight = 100.0 ;
fvar cartWidth = 100.0 ;
fvar cartHeight = 50.0 ;
fvar wheelRadius = 10.0 ;
fvar wheelsSeparation = 20.0 ;
fvar wheelsSeperationX2 = 40.0 ;
fvar stickLength = 80.0 ;
fvar stickWidth = 10.0 ;


fvar floorLevel = sh - floorHeight ;
fvar wheelCenterLevel = floorLevel - wheelRadius ;
fvar wheelLevel = wheelCenterLevel - wheelRadius ;
fvar cartLevel = wheelCenterLevel - cartHeight ;


@@@ Inputs from game
fvar x = 200.0 ;
fvar theta = 120.0 ;
theta = theta / 180.0 ;
theta = theta * pi ;
@@ Helping edge coordinates
fvar cartHalfWidth = cartWidth / 2.0 ;
fvar cartLeft = x - cartHalfWidth ;
fvar cartRight = x + cartHalfWidth ;


fvar stickXS = cos theta ;
fvar stickYS = sin theta ;
fvar stickHeight = stickLength * stickYS ;
```

```
fvar stickLevel = cartLevel - stickHeight ;
fvar stickSlope = stickXS / stickYS ;
fvar stickX = stickLength * stickXS ;
stickX = stickX + x ;

fvar dxWheel = 0.0 ;
fvar firstWheelLeft = 0.0 ;
fvar firstWheelRight = 0.0 ;
fvar secondWheelLeft = 0.0 ;
fvar secondWheelRight = 0.0 ;

@@@ Axes counters
fvar shc = 0.0 ;
fvar swc = 0.0 ;

@@ Conditional values containers
bvar y_loop = 1 ;
bvar x_end = 1 ;
bvar inStickLevel = 1 ;
fvar temp_condition = 0.0 ;
bvar tempCondition1 = 1 ;
bvar tempCondition2 = 1 ;
bvar condition = 1 ;

while y_loop {

  x_end = swc == sw ;
  if x_end {
    swc = 0.0 ;
    shc = shc + 1.0 ;
```

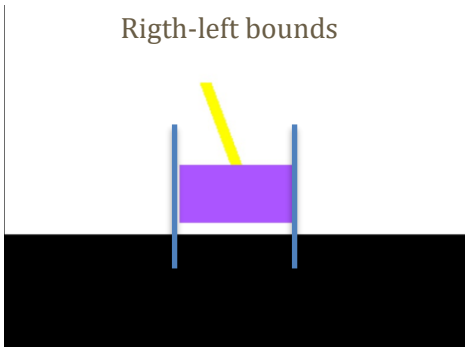
```
    y_loop = shc == sh ;
    y_loop = not y_loop ;

    inStickLevel = shc > stickLevel ;
    if inStickLevel {
      @@ Shift x coordinate using stick's
slope
      stickX = stickX - stickSlope ;
    } else {

    }

  } else {

  }

  tempCondition1 = 1 ;
  tempCondition2 = 1 ;
  condition = shc > floorLevel ;

  if condition {
    intf shc ;
    intf swc ;
    VRAM_Save shc swc floorColor ;
    fint shc ;
    fint swc ;
  } else {
    condition = shc > cartLevel ;
    if condition {
      condition = shc > wheelCenterLevel ;
```

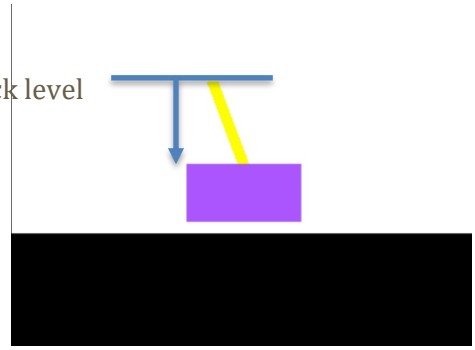






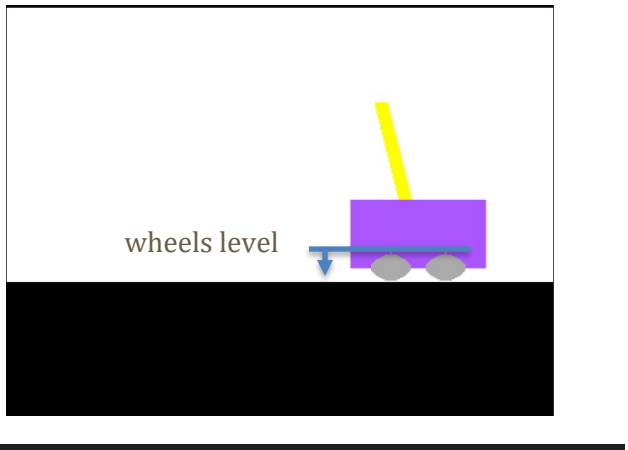
```
    if condition {
      intf shc ;
      intf swc ;
      VRAM_Save shc swc skyColor ;
      fint shc ;
      fint swc ;
    } else {
      tempCondition1 = swc > cartLeft ;
      tempCondition2 = swc < cartRight ;
      condition = tempCondition1 and tempCondition2 ;
      if condition {
        intf shc ;
        intf swc ;
        VRAM_Save shc swc cartColor ;
        fint shc ;
        fint swc ;
      } else {
        intf shc ;
        intf swc ;
        VRAM_Save shc swc skyColor ;
        fint shc ;
        fint swc ;
      }
    }
  } else {
    condition = shc > stickLevel ;
    if condition {
      fvar stickXWithWidth = stickX + stickWidth ;
      tempCondition1 = swc > stickX ;
      tempCondition2 = swc < stickXWithWidth ;
```





```
        condition = tempCondition1 and tempCondition2 ;
        if condition {
          intf shc ;
          intf swc ;
          VRAM_Save shc swc stickColor ;
          fint shc ;
          fint swc ;
        } else {
          intf shc ;
          intf swc ;
          VRAM_Save shc swc skyColor ;
          fint shc ;
          fint swc ;
        }
      } else {
        intf shc ;
        intf swc ;
        VRAM_Save shc swc skyColor ;
        fint shc ;
        fint swc ;
      }
    }
  }

  swc = swc + 1.0 ;
}

fvar wheelRadiusSquared = 0.0 ;
fvar dyFlipped = 0.0 ;
fvar dy = 0.0 ;
```

```
fvar dySquared = 0.0 ;
bvar inWheelLevel = 1 ;


y_loop = 1 ;
shc = wheelLevel ;
swc = 0.0 ;


while y_loop {


  x_end = swc == sw ;
  if x_end {
    swc = 0.0 ;
    shc = shc + 1.0 ;
    y_loop = shc == floorLevel ;
    y_loop = not y_loop ;


    wheelRadiusSquared = wheelRadius * wheelRadius ;
    dyFlipped = shc - wheelLevel ;
    dy = wheelRadius - dyFlipped ;
    dySquared = dy * dy ;
    temp_condition = wheelRadiusSquared - dySquared ;
    dxWheel = sqrt temp_condition ;
    firstWheelLeft = x - dxWheel ;
    firstWheelLeft = firstWheelLeft - wheelsSeparation ;
    secondWheelLeft = firstWheelLeft + wheelsSeperationX2 ;
    firstWheelRight = firstWheelLeft + dxWheel ;
    firstWheelRight = firstWheelRight + dxWheel ;
    secondWheelRight = firstWheelRight + wheelsSeperationX2 ;
  } else {
```



```
  }


  tempCondition1 = swc > firstWheelLeft ;
  tempCondition2 = swc < firstWheelRight ;
  condition = tempCondition1 and tempCondition2 ;
  tempCondition1 = swc > secondWheelLeft ;
  tempCondition2 = swc < secondWheelRight ;
  tempCondition1 = tempCondition1 and tempCondition2 ;
  condition = condition or tempCondition1 ;
  if condition {
    intf shc ;
    intf swc ;
    VRAM_Save shc swc wheelColor ;
    fint shc ;
    fint swc ;
  } else {


  }
  swc = swc + 1.0 ;
}
bvar hfgk = 1 ;
```