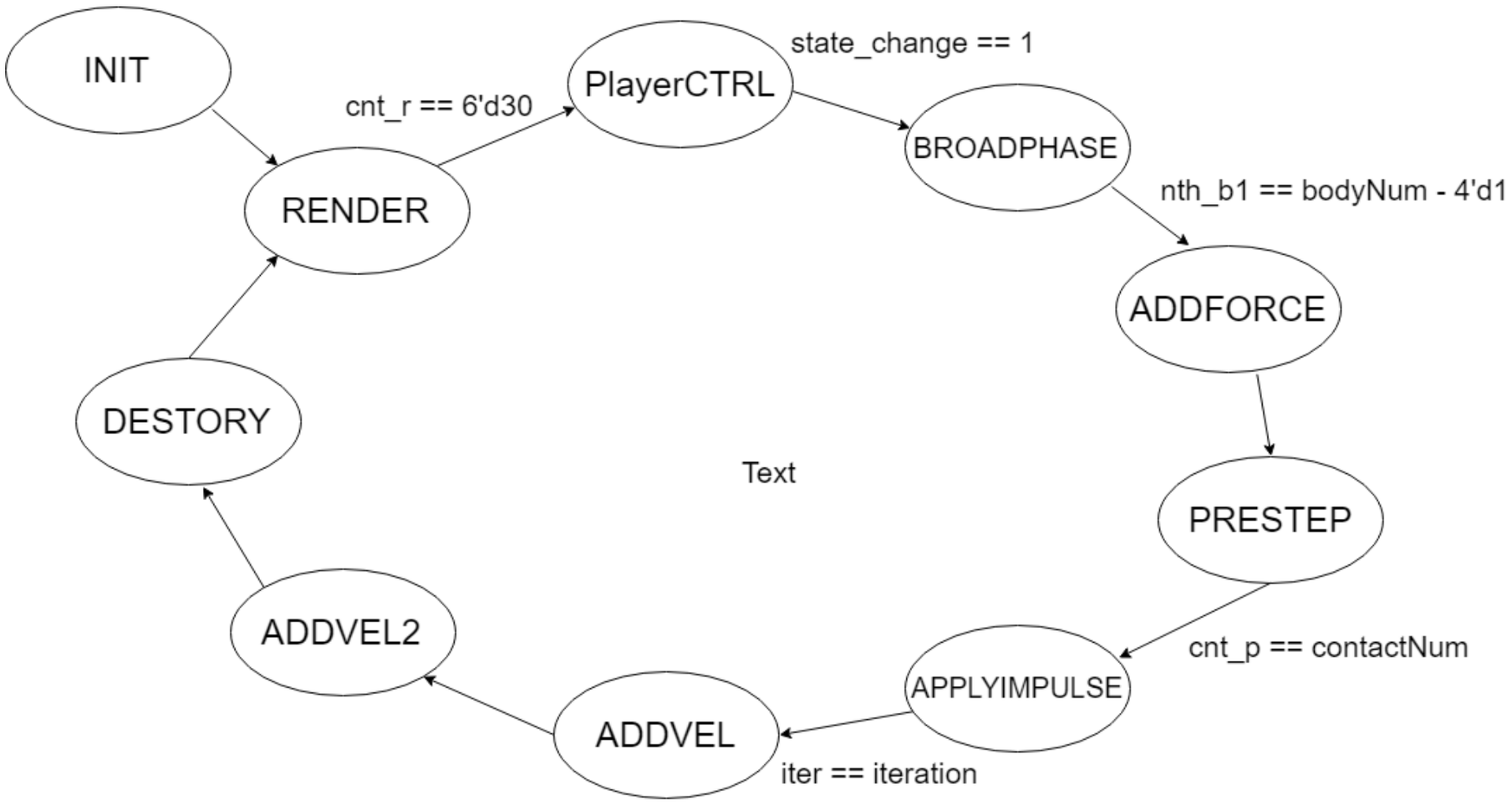
Angry Birds

Team 5 - 馮謙 史孟玄

Outline

- 在憤怒鳥裡，有運用到很多演算法，尤其是物理引擎的部分，而光是物理引擎中的其中一個小小的碰撞偵測可能就可以寫個十幾頁了，如果要再加上硬體的優化和一些合成的心路歷程，可能寫都寫不完。因此物理引擎的部分可能就只是點到為止，主要聚焦在我們是如何解決一些重要的問題上。

Game Flow



這裡的render是把四個角傳給另一台slave FPGA，由另一台來做畫圖的事情。PlayCTRL主要是做滑鼠對鳥的控制，如果滑鼠把鳥拉出一定的範圍並放開，那麼state_change就會拉成1，使state進入到一系列的物理引擎。

Init

- 在這個state裡，主要是把一些該初始化值準備好，以便之後的運算。

- 一些重要的變數如下：
  1. Position(divide in PosX, PosY)
  2. Velocity(divide in VelX, VelY)
  3. Radian(rad)
  4. Angular Velocity(angVel)
  5. Vertices(divide in verticesX, verticesY)

BroadPhase

- 在這個 state 裡面，會做碰撞的偵測和產生碰撞訊息，而這時候就會用到 Collide.v 這個 module。

- Collide.v 裡面包含了以下四個 module：

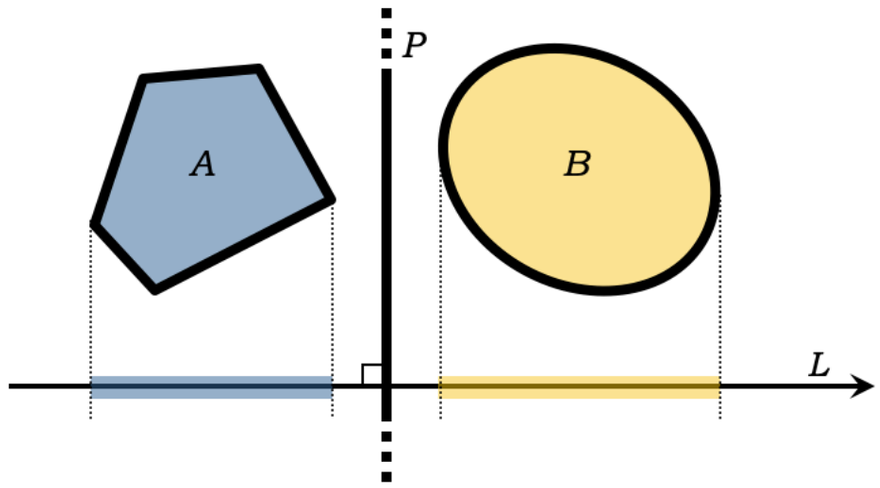
    1. Support：協助 FindLeastPenetration 找出Contact Point。
    2. FindLeastPenetration：計算出最小穿透量以及碰撞法向量。
    3. ComputeIncidentEdge：利用前面算出的法向量計算出Incident Edge。
    4. ClipSegmentToLine：把超出的點剪去，算出最終的碰撞點。

- Collide 本身也是一個FSM，透過內部的 state 控制以上四個 module 的運行。接下來會先物理引擎，在講如何在 verilog 上 implement。

Collision Detect — SAT Algorithm

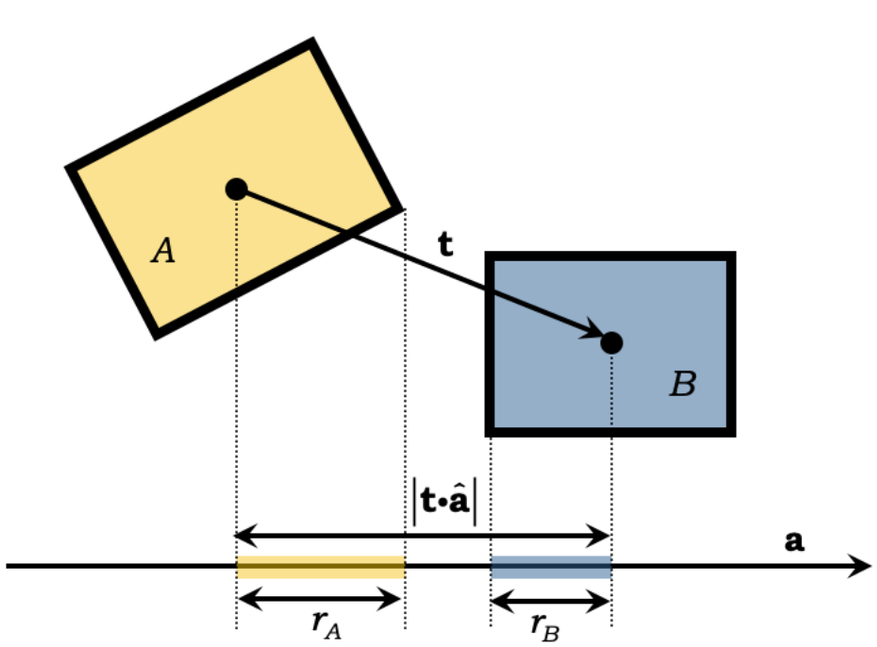
- Separating Axis Test：如果兩個物體之間沒有碰撞的話，則物體之間必定存在一條分離軸。

- 然而光是知道兩個物體是否碰撞是不得，我們最終需要三個訊息：contact point、penetration、contact normal。



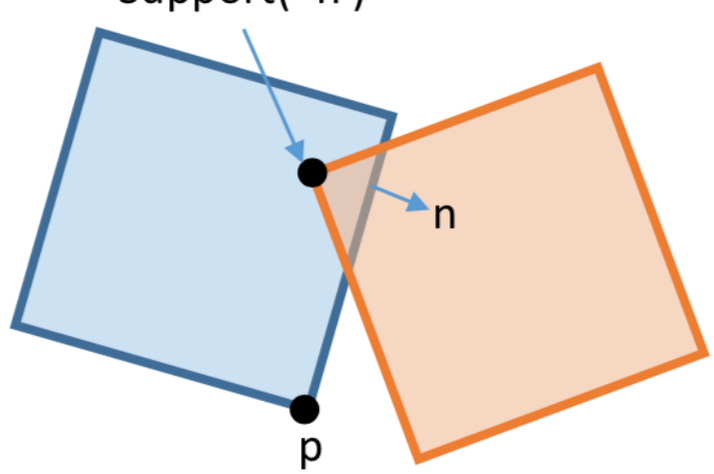
Implement SAT

- 換個方式想，只要兩個圖形彼此穿透，就代表有碰撞，因此我們改成計算穿透。而一種方法是把圖形分別投影到軸上，然後看之間的重疊，但projection的計算太耗資源，因此用另一種方法：support function



Separated if
$|\mathbf{t}\cdot\hat{\mathbf{a}}| > r_A + r_B$

use projection

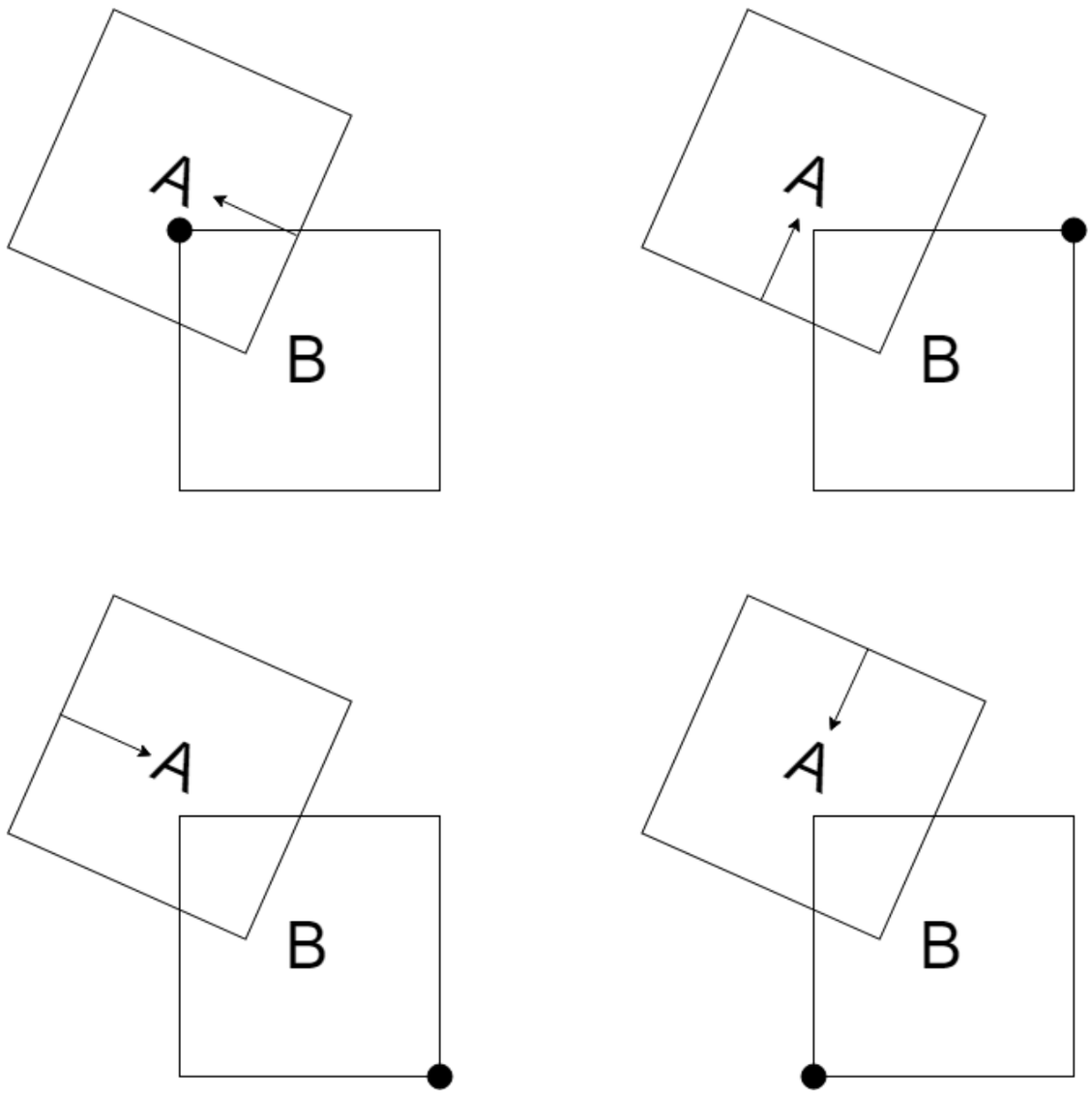


$d = n \cdot (\text{Support}(-n) - p)$

use support

Support

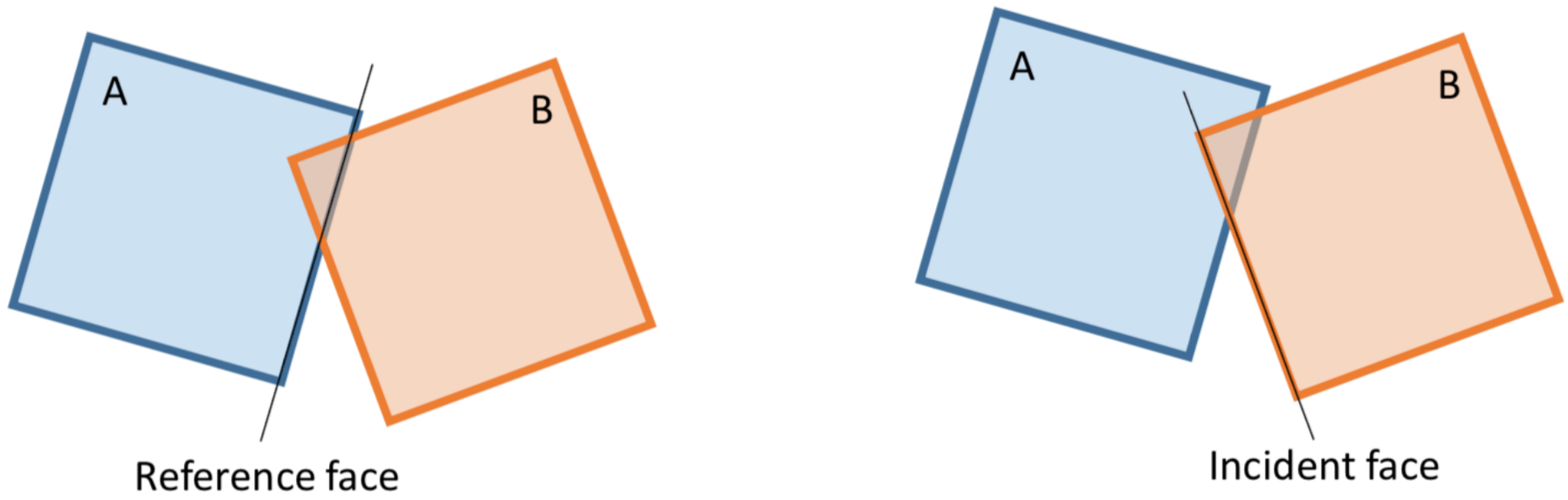
- 箭頭代表向量，通常是帶入外部圖形的向量 n*-1
- support function 會回傳那個方向中所能到達得最遠的點。
- 每個點B都代表著他可能對A穿透多深，因為只要讓它對向量做內積，就馬上可以得到穿透量。



FindLeastPenetration

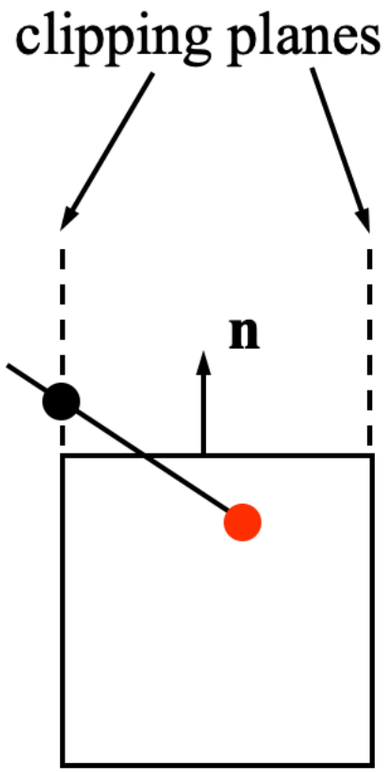
- 利用support所回傳點，做dot得出內積後存起來，最後最小的值就是我們要找的穿透量，此時碰撞法向量就是對應那個穿透量的向量。
- 比較麻煩的是，兩個圖形是互相的，不能只算 A 對 B ，也要做 B 對 A ，所以在這裡我們就好好分配FPGA的資源，然後多利用FSM來盡量縮短crtical path。

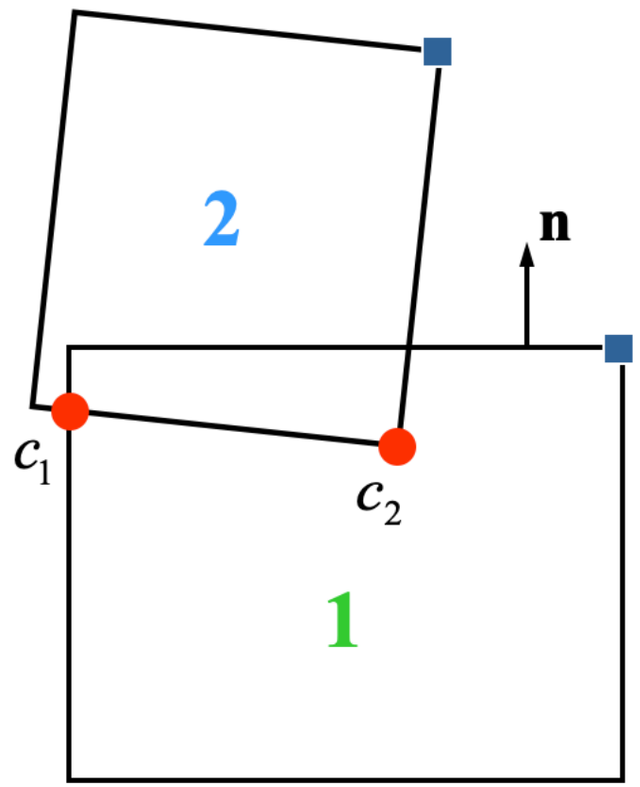
Compute Incident Edge



- 現在我們還差碰撞點，但在這之前，需要準備一些資料那就是incident face和reference face。reference face在上一個module就可以得到，incident face的話，就是找出一個向量對reference face的法向量做dot為最小的邊。

Clipping





- clipping將會產生最終的碰撞點，這部分我們是利用兩條直線方程式，算出它的交叉點。主要是將直線轉換為矩陣的形式，然後坐反矩陣求出交叉點。在對左右兩條邊做完之後，就可以得到最終的碰撞點了。collide到此就結束了，broadphase也是。

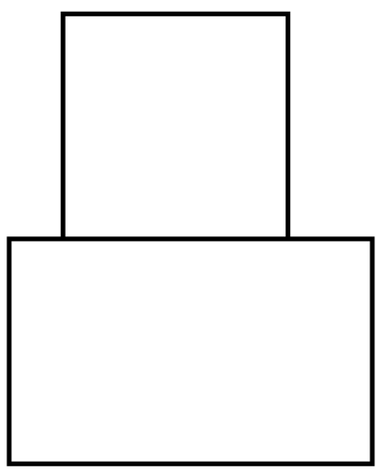
PreStep

- 這個module裡做的就是大量的運算，主要是為ApplyImpulse這個module做鋪陳。

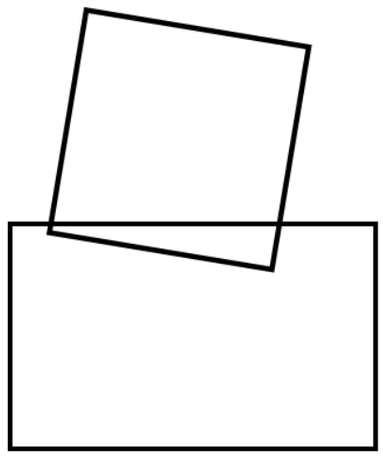
- 最終他會產生三個output：
  1. massNormal：用來和衝量做計算
  2. massTangent：用來和摩擦力做計算
  3. cBias：這個主要是來自於penetration，因為物體已經產生穿透的關係，所以需要有一個修正的係數，把物體推回正常位置，不然有可能兩個物體會產生 Jitter。

ApplyImpulse

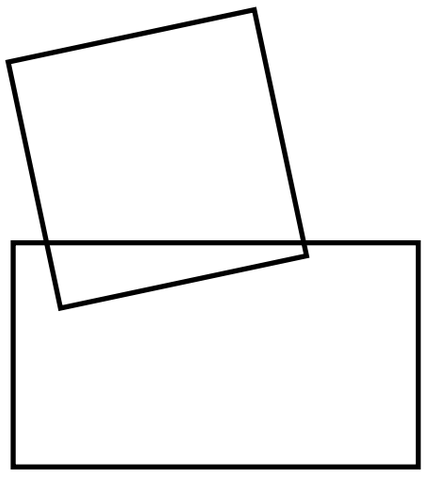
- 這個module會對每個碰撞點做力的運算，然後把力施加到對應的物體上。但是只做一遍的話物體很可能還是斜的。因此這裡會做大於30次的 iteration，理論上做越多次越準。這也是我遭遇最大的困難，因為我懷疑是這個module沒寫好，但是燒一次要15分鐘以上，然後simulation又看不出真正的樣貌，而且要看的訊號太多了，最終還是沒有找出錯誤。



What we want.



What we get.



Render

- 當所有的力都疊加，以及弧度和位置都加完後(addVel1, addVel2)，會把最新的弧度和位置丟給Render。在Render裡面，利用cordic這個ip，算出sin、cos，在做一些運算過後得出長方形的四個角。（在我們的設計中，鳥跟豬都是矩形，因為這樣就不必另外寫圓形對方型的碰撞判斷。）

- 另外，cordic算出來的誤差偏大，而且精準度不能調，一旦圖形變大，算出來的四個角誤差就會太大，影響到之後的計算，所以對於像地板這種物件，我們就是直接把值初始好。

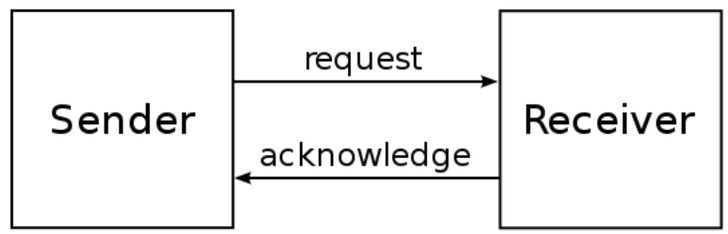
- 最後，整個Top module的output就是這些算出來的角，我們透過handshaking來將這些點傳給另一台FPGA作「真正的」Render。

HandShaking

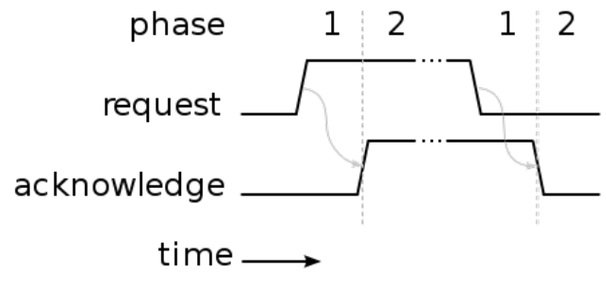
- 我們採用的是4-phase的協定，因為這樣資料會傳得更穩定。

- 實作：這裡的request相當於我們裡面的master_ready，acknowledge則是slave_ready，當master_ready拉起來後，slave_ready就跟著拉起來，然後master偵測到slave拉起來，就在拉下去，當slave偵測到master拉下去，就把自己也拉下去。

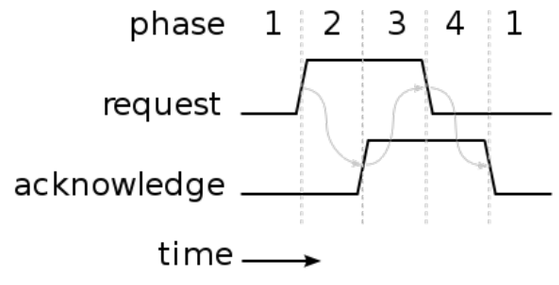
- 如果master_ready照著自己的cycle，slave_ready依照master_ready拉起來或降下，那就會變成2-phase，值的傳遞會非常不穩定。



2-phase handshake



4-phase handshake



How to rotate picture?

- Nearest Neighbor Interpolation? Bilinear Interpolation?

NO! NO! NO!

有鑒於許多用來旋轉圖片補色的演算法在FPGA上都太消耗資源，可能轉完一張圖片就掛了。為了解決這個問題，我們從SAT得到靈感，如果我們可以判斷矩形有沒有碰撞，那為何不能判斷點是否在矩形內部呢？因此運用相同的方法，利用點與四個角的差值，對分別對四個邊的法向量做內積，如果結果都是負的，代表說點存在於矩形內部，那就塗上想要的顏色，不然就塗背景的顏色。值得一提的是，如果想要有邊匡，那就設定一個slop，只要內積結果小於零，且在slop之內，就塗黑色，再來才圖想要的顏色。

IP

- 我們一共用了三個主要的IP：
    1. Divider：這個是最難搞的，因為他消耗的資源非常多，而且不管怎麼調都還是很多。
    2. Multiplier：最方便使用，資源消耗少，比起自己寫＊來的好很多。
    3. Block ram：幫了很多忙，當初為了儲存contact Info就花了不少LUT，一度爆炸到300%，用了之後直接砍半，缺點就是要算cycle。

Overview

- 從對物理引擎的陌生，開始讀很多資料，然後自己手刻python版的，刻到一半一堆bug，然後又從新回去找文件出來看，直到刻出來後發現，天啊這也太難轉到verilog了吧！接著一步一步先畫出大概的block diagram，到自己學會算fix point，然後想出各種算法來解決fpga的資源問題，最後整個東西要燒進去的時候，遇到各種critical path以及資源不夠的問題，再回去找可以優化的地方，到官網讀IP的官方文件，然後測試了一堆東西後，終於能夠燒上去，結過卻面臨de不出來bug。每一個module在接上去之前都有做tb測試，結果真正顯示出來的結果卻和python上大相徑庭。到了demo當下，真的覺得蠻遺憾的，畢竟也是拼了一整學期，最後的結果卻不如自己預期。

- 我認為 Angry Birds 並不是做不出來，但確實非常有挑戰性，跟之前的皮卡丘打排球相比，光是物理引擎的原理跟演算法的實作就需要蠻多時間吸收了，而FPGA又是考驗coding技巧的時刻，如果style不好，或是對電路的理解不夠，很容易就寫出很難改得扣。總的來說，學到了很多！