0516067 曾靖驊 HW1

I. Introduction

這次的 project 實作一些常見的 2D particle system，剛體碰撞、摩擦力、彈簧阻尼力等等，但並沒有實作旋轉

II. Fundamentals

我們在每毫秒都需要算出所有物體下個毫秒之間的位置，需要在 1 毫秒裡面做完下列全部的步驟。

Step1. Generate Manifolds

利用所有剛體兩兩排列組合出 n(n-1)/2 個交互作用，每個交互作用獨立運算，Manifold 存著剛體兩兩之間的關係(有無碰撞)

Step2. Impulse resolve

利用 Step1 得出的 Manifold 找出所有的 Impulse(碰撞以及摩擦)，套用在各個剛體上，調整他們的速度向量

Step3. Positional Correction

位置校正，利用 Step1 得出的 Manifold，將重疊的物體稍微分離

Step4. Joint Constraint

這裡不需要 Manifold，不論是 DistanceJoint 還是 SpringJoint，都只需要彈簧兩邊的物體的資訊即可，利用相對位置來計算需要的速度改變量

Step5. Integrate

利用 ExplicitEulerIntegrator 或是 RungeKuttaFourthIntegrator 來計算最後所有物體應該要呈現在什麼位置

III. Implementation

Manifold:

aabb-aabb:

先利用中心點位置以及形狀判斷 x、y 方向的 overlap 各是多少，若都大於 0，再進一步選擇 overlap 比較少的那一方作為兩者接觸的方向

circle-circle:

先利用中心點位置以及形狀判斷是否有接觸，並且直接用相對位置來做為兩者接觸方向

aabb-circle:

比較麻煩

Step1. 先找出 aabb 最有可能和圓形接觸的那一角

Step2. 利用中心點位置以及形狀判斷 x、y 方向的 overlap 各是多少，若都大於 0，才*有機會*接觸，進行下一步檢查下面三種可能出現的狀況

Step3-1. 檢查圓心是否在 aabb 內部，若 true，就選擇 overlap 比較少的那一方作為兩者接觸的方向

Step3-2. 圓心不在 aabb 內部，檢查有沒有 x 或 y 過半，沒有的話才檢查有沒有包含 aabb 那一角，在用那一角和圓心作為接觸方向

Step3-3. 有 x 或 y 一方過半，視為切到 aabb 的其中一邊，用那個邊決定接觸方向

另外，所有的碰撞(isHit)必須兩物體的相對位置是拉近中的才能視為碰撞，否則把這個問題給 PositionalCorrection 解決就可以了，不然生成兩個過於貼近的物體很容易發生錯誤(速度爆表或是兩個物體來回震盪)

Impulse Resolve:

開始前就先記好切線方向是否有位移，以利於在 Step2 計算摩擦力時選擇要使用動摩擦還是靜摩擦

Step1. 非彈性碰撞

用 Manifold 提供的 Normal 方向進行碰撞，套用公式計算 I，並且更動兩物體的速度

Step2. 摩擦力

用 Step1 得到的 I 做為正向力的大小，進而算出能提供的最大摩擦力，用更動過的速度，再套用摩擦力的影響，得出最終的速度

我在這個 project 中的物體並沒有辦法真正的停下來，而是最終會達到像素無法呈現出的微小且恆定的碰撞，但這些穩定的碰撞都會被視為一個瞬間的正向力的來源，碰撞的那一瞬間就要變更物體的速度，而不是一個靜態力平衡的狀態。

Joint Constraint

Spring Joint:

利用兩物體間的相對速度及位置，即可計算彈力以及阻尼力，沒什麼特殊的

Distance Joint:

我嘗試過兩種方式，雖然兩種方式使用的速度改變量公式是相同的，都是利用相對距離以及速度，嘗試消除相對速度，同時也希望再 delta time 時間內消除相對位置的差距。

法 1: 兩者距離必須衡定，但由於初始狀態 Joint 之間有點過近，一開始速度會很快，看起來有點混亂。

法 2: 只有在兩者距離超過時才套用這個公式，允許兩點之間接近，看

起來有比較柔和一點

Integrate

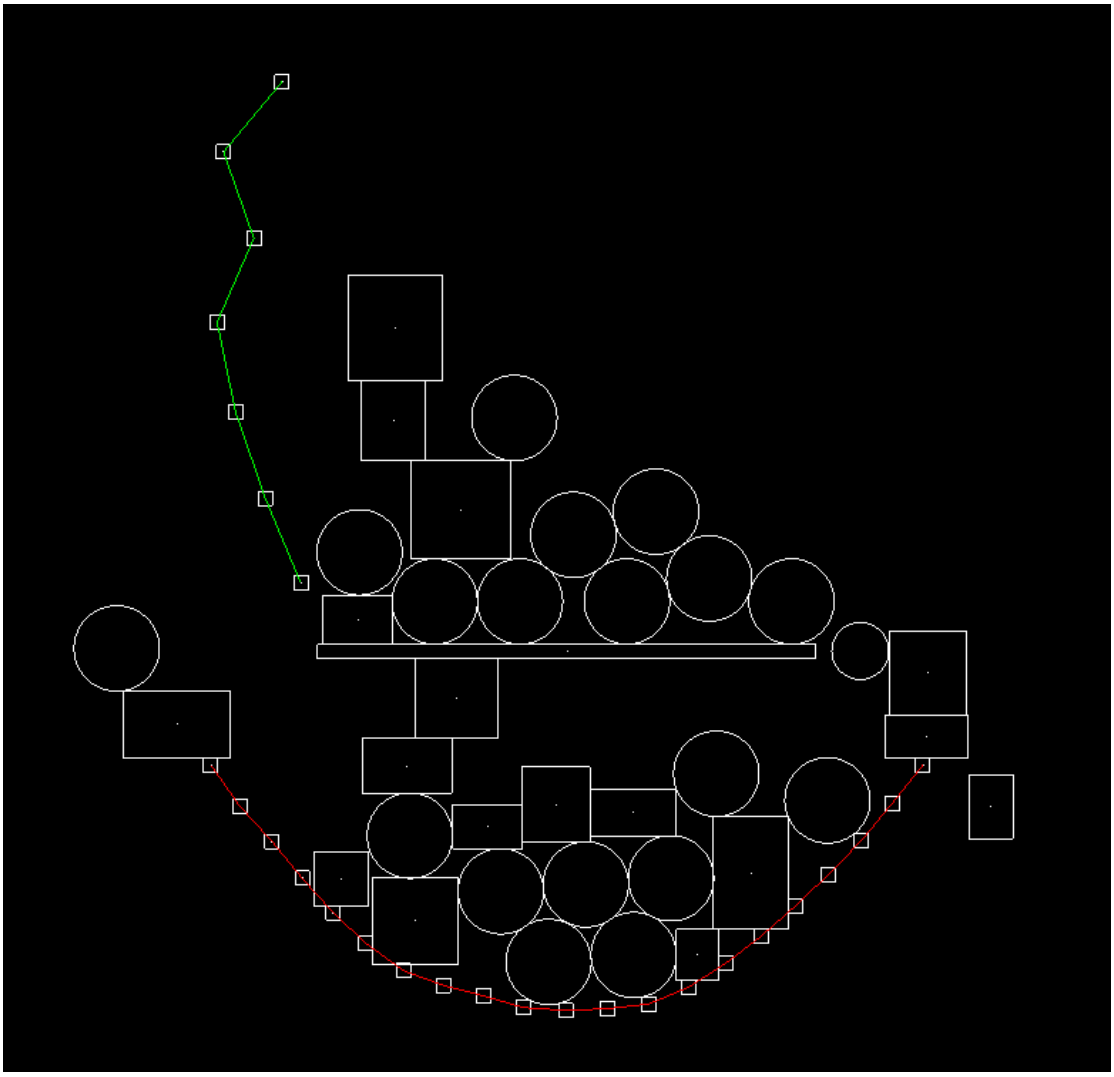
我只有嘗試 Euler's method，

Step1 先利用速度進行位移

Step2 利用受力改變速度

Step3 將受力設定為重力

IV. Result



程式跑起來應該有符合需求，雖然沒做出 RungeKuttaFourthIntegrator，但 Explicit Euler's method 搭配 Resolve 直接改變速度的效果看起來還行。

此外用這種兩兩交互計算 Manifold 的方式，物體只要超過一定數量電腦就會不堪負荷調幀，可能需要其他清除畫面外物體的機制，或是老師提過的將畫面分區減少不必要的交互計算。