【附件 2】

國立中央大學工學院
酷派爭霸讚—程式設計菁英賽
成果報告書

以粒子系統進行擬真動畫繪製

指導教授 ： 張彙音
參賽隊員1： 陳繹帆

目錄

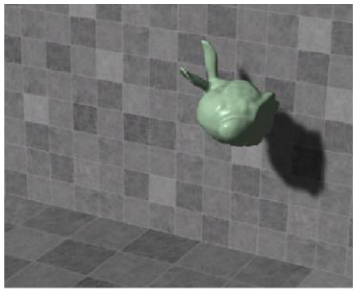
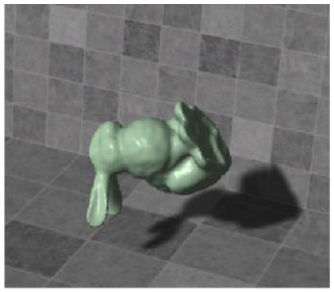
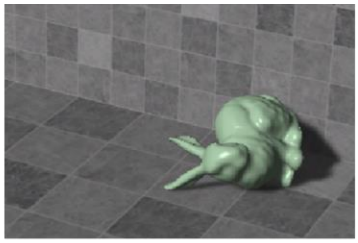
一、作品簡介(1 頁為限)

創作理念：希望使用 Java 以及 Unity 架構粒子系統對於現實事物進行模擬，製作出物理擬真的動畫，此研究對於機械、建築、或流體設計都有極大幫助。

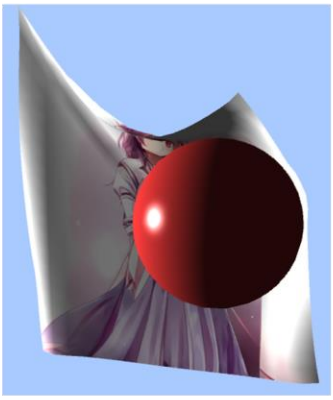
應用性：希望能夠最終應用在建築、流體模擬軟體之上，或者是 VR、擬真動畫開發，所有展示的影片，被設計成全部都可以進行實時交互並看到模擬結果。

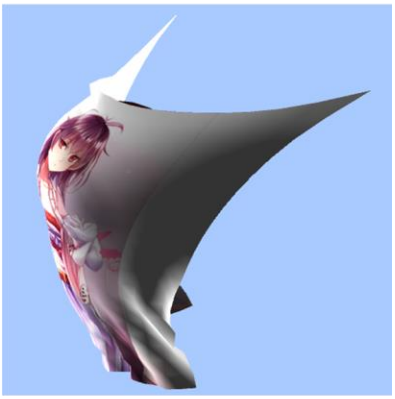
挑戰性與完成度：此題目是今年一月才開始接觸的領域，物體模擬包含(1)集群運動、(2)剛體模擬、(3)布料模擬、(4)彈性體模擬和(5)流體模擬。因為只有三個月的時間，一直很努力地希望能夠先把這五個方向的模擬完成一遍，接下來會慢慢再進行擴展。我覺得透過程式進行物體模擬為超有挑戰性的，希望到比賽前我能夠將模擬的部份做得更全面，能完美地展示給大家。

剛體模擬

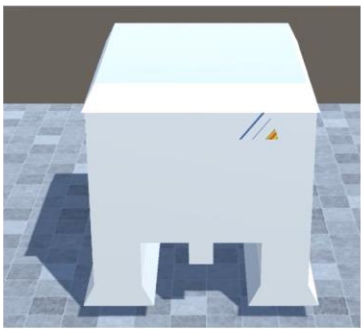
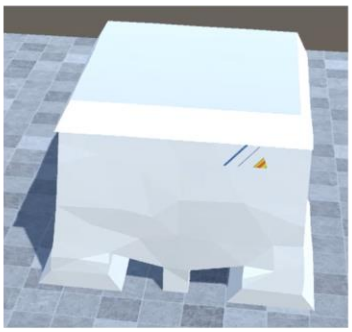
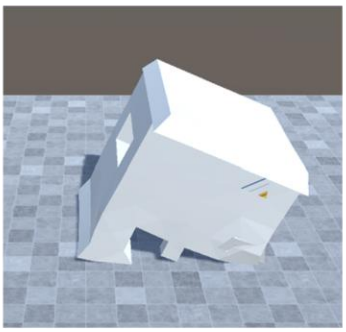




布料模擬

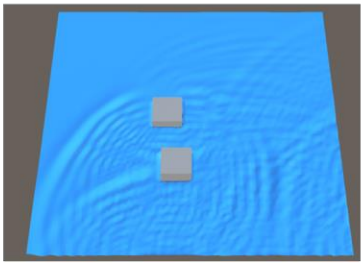
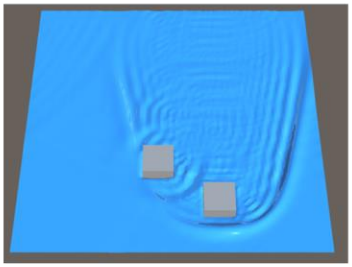
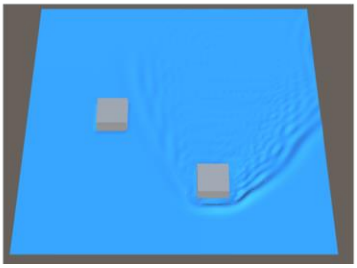




彈性體模擬

水體模擬

二、專題團隊及任務分工(1 頁為限)

請介紹團隊成員，說明任務分工方式、團員個人經歷等。

陳繹帆:

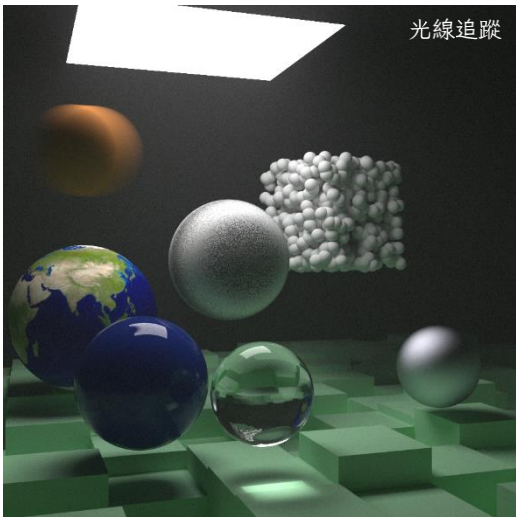
任務分工：全部

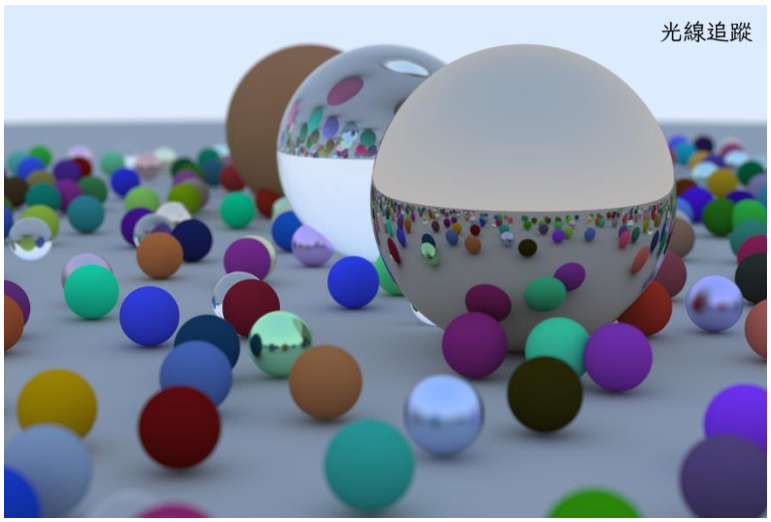
目前是工學院學士班大三，程式部分較為擅長非類神經網路的機器學習，平時喜歡亂寫一些沒什麼用處的程式，因為自己的興趣對於計算機圖學有點涉獵，就想說趁著這次比賽來學習基於物理學的動畫模擬好了，畢竟感覺起來就很棒。

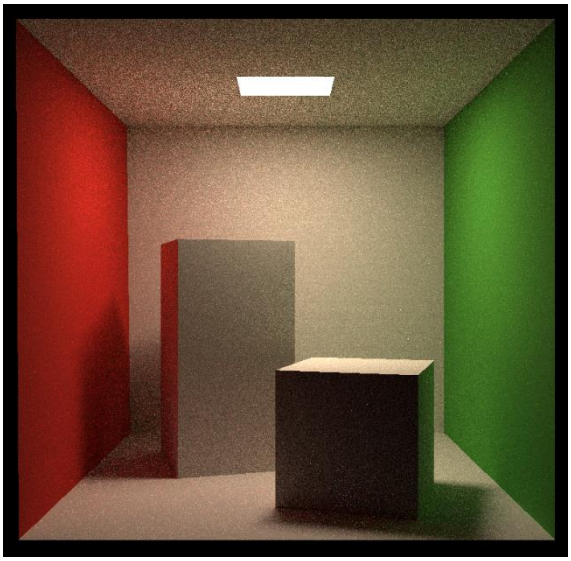
個人經歷：

1. 2021 酷派爭霸戰程式設計競賽 第二名
2. 2021-03-23 參加大學程式能力檢定(CPE)，排名 141 / 2394。
3. 2020 二上於工院學士班專題以影像拼接獲得最佳人氣獎。
4. 2020-10-20 參加大學程式能力檢定(CPE)，排名 139 / 2179。
5. 2018 成為 INFAS 全國中學生資訊年會工作人員
6. 2018 科展「為之洞容一超級電容測量及材料改良」獲得南區優等，該組該年無特優。(以 Arduino 製作可以跑循環伏安法測量電容好壞的小型儀器)
7. 2017 參加 INFAS 全國中學生資訊年會工作人員訓練

※ 經歷好像有點太少，我來放以前渲染過的圖好了(如下)，透過圖來說話也比較好玩。







雙向路徑追蹤

三、成果報告(10 頁以內)

專題名稱: 以粒子系統進行擬真動畫繪製

陳繹帆

摘要

以粒子系統分別對集群運動、剛體模擬、布料模擬、彈性體模擬、流體模擬，嘗試將計算機動畫中較為難解決的課題進行實作與分析，並將程式碼以 MIT 條款開源釋出讓大家可以隨意取用以及進行驗證，集群運動問題中使用了 PSO 演算法來對優化問題進行求解，剛體模擬中實做了 Impulse 與 Shape matching 兩種剛體運動中常使用的演算法，布料模擬則是透過 position based dynamics(PBD)來進行實現，彈性體模擬使用 Saint Venant-Kirchhoff(StVK)做為能量方程來進行計算，流體則是先使用 Shallow water wave 模型完成建置，目前已經將簡易的動畫模擬完成並可以進行簡易的互動式操作以驗證模型的正確性。

I. 動機

最初的動機是在 Lingqi Yan 教授的課程中看到別人做的海水潮汐模型，覺得真的好厲害，我也想自己寫一個，所以就開始動手自己實作，然後發現自己對於水體如何模擬一點頭緒都沒有，就想說那還是慢慢來好了，畢竟酷酷的題目哪有一下子就能完成的，由於需要足夠擬真的話，基本上都是把水體視為許多水粒子來進行模擬後再進行渲染，所以我也想說從粒子系統開始進行學習。

首先吸引我注意的是 PSO(粒子群演算法)，該演算法最初是用來描述鳥群的移動，後來被延伸成為解最佳化問題的演算法，實現完之後，我對於粒子間的交互關係更為著迷。後來有另外寫了一個太陽系，只是寫完行星之間的交互關係(只需要考慮重力)之後，發現照實際比例縮放渲染完根本看不到，就覺得自己好像有點欠缺考慮了，應該要再多上一點課，就開始上了俄亥俄州立大學的王華民教授的基於物理的計算機動畫入門，慢慢的學了一些計算機動畫的做法和如何快速進行計算的觀念，也想自己動手做能讓人眼睛為之一亮的模擬，就一頭熱的開始了不斷閱讀的學習之路。

一開始學的時候沒有想那麼多，寫完之後發現我好像對於計算模擬好像也有一些初步的了解，也可以撰寫一些基於物理的模擬程式，從中獲得了極大的成就感。

II. 研究方法

因為粒子系統可以模擬的東西有很多，所以我打算分成以下五種模擬來進行探討。除了集群行為使用 java 撰寫之外，其餘皆是使用 unity，利用自己寫的物理引擎替換掉內建的物理引擎的方式完成，避免掉需要寫渲染引擎等等的麻煩，也能更專注在程式上。

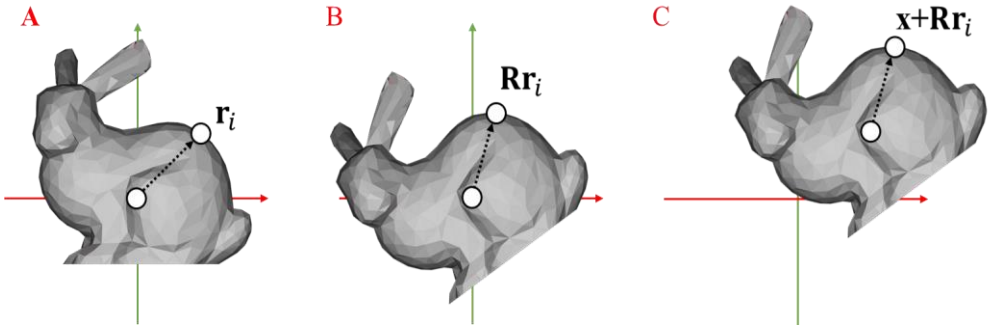
1. 集群行為

集群行為是一種在模擬上很受歡迎的課題，透過定義與約束粒子的行為，來進行集群行為的模擬，或者也可以用來做為優化算法來解最佳化問題。

PSO 演算法(粒子群演算法)：粒子群演算法是為了模擬鳥類飛行行為所發展出來的演算法，在粒子群演算法中一個粒子代表鳥群中的一隻鳥，各個粒子具有記憶性並會參考其它粒子的訊息來決定移動的方向。一群被稱為粒子的潛在解在多維度解空間中找尋最佳解位置，粒子每次移動都會參考自身過往曾找到的的最佳解位置、所有例子的過往最佳解位置，然後再決定移動方向還有距離。白話的意思是：粒子目前移動速度 = 粒子上個時間的移動速度 + 自己認為的最佳方向 + 群體認為的最佳方向，透過不斷移動一群粒子來在有限時間內得到較好的解。目前主流用途與基因演算法類似，被拿來作為解決最佳化問題的工具，因為這類問題大多都是 NP 問題或者在短時間內難以求解，所以迭代尋找相對較佳的解的做法成為主流作法，不過我想做的事情是將粒子的行為可視化，因此將能求解的問題維度降低到 2 維上。

2. 剛體模擬

剛體運動是物理模擬中最簡單的部分，使用粒子系統進行剛體模擬是將剛體所有的網格點位進行計算與碰撞檢測，來決定碰撞會獲得多少的加速度，但在實際運算上，有點多此一舉，故只實現了 Impulse (脈衝法)的情況與使用 Shape matching(基於形狀保持的算法，不包含物理特性)的情形進行比較。



圖一、ABC 為模型的實際展現過程。每個點 r_i 的位置加上一個旋轉矩陣 R 再做平移 x，模型上的每個點可以用 $x + Rr_i$ 表示，每個點的速度可以以質心速度加上角速度表示，$v_i = v + \omega \times Rr_i$，就可以快速求得模型上每個頂點的位置與速度。

i. Impulse :

先做碰撞檢測，若有發生碰撞，就直接給碰撞點一個新的速度，並假設是因為一個衝量 j 導致速度發生改變，再來推算 j 是多少，進而去改變整體的速度。

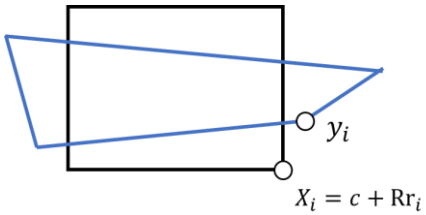
$$v^{new} \leftarrow v + \frac{1}{M} j \qquad \omega^{new} \leftarrow \omega + I^{-1}(Rr_i \times j)$$

$$v_i^{new} = v^{new} + \omega^{new} \times Rr_i = v_i + \frac{1}{M} j - (Rr_i) \times \left(I^{-1}(Rr_i \times j)\right)$$

求出衝量 j 之後直接給予剛體一個 j 的衝量就好，若多點同時碰撞就使用參與碰撞的點的位置平均值來做碰撞處理。另外因為物體掉到地板上時，在重力和彈力交互作用下會不斷抖動，所以需要在物體靜止時，將反彈係數做衰減的動作。

ii. Shape matching :

採用先讓物體形變後再還原的方式進行模擬，形變為不會展示出來的中間過程。把剛體的每個頂點都當做可以自由移動的頂點，對每個點做單獨的碰撞處理，然後通過最小化 MSE（均方誤差）的方法求出剛體的形狀，關鍵點是如何在碰撞完之後把模型拼回原本的形狀，在計算上的優勢是完全不涉及角速度，所以實現上較為簡單 (如圖二所示)。



圖二、碰撞完後示意圖。y_i是碰撞完的粒子形狀，x_i是要還原成的物體形狀，c 和 R 是欲求解的平移量和旋轉矩陣。

$\{c, R\} = \text{argmin} \sum_i \frac{1}{2} \|c + Rr_i - y_i\|^2$ ➔ $\frac{\partial E}{\partial c} = \sum_i c + Rr_i - y_i = \sum_i c - y_i = 0$

➔ $c = \frac{1}{N} \sum y_i$，求出平移量 c 之後，可藉由$\frac{\partial E}{\partial R} = \sum_i (c + Rr_i - y_i) r_i^T = 0$，得出$R = \left(\sum_i (y_i - c)\ r_i^T\right)\left(\sum_i r_i\ r_i^T\right)^{-1}$，可以快速求解得到模型應在的位置和旋轉矩陣。

3. 布料模擬

布料模擬的部分，可以分成兩個研究方向，一部分是基於物理的方法做布料模擬，一部分是基於約束的方法來做。物理方法基本上都是將布料拆成許多頂點，可以把所有頂點看做一個粒子系統，彈簧被插入到這個粒子系統中，顯式積分法的做法是透過遍歷所有的彈簧，算出每個彈簧對頂點施加的力，就可以算出每個頂點所受合力，接下來更新頂點速度和位置即可，但缺點是瞬間受力太大時會容易數值不穩定導致崩潰，主流方法為採用隱式積分法，透過推定頂點新的位置來進行所受合力的計算，不過隱式積分法這部分我還沒看懂怎麼用牛頓法迭代來做，在海報展前希望做出來。

真實世界的布料拉伸到一定程度後，對拉伸都有很強的抵抗，物理方法會需要增加彈性係數 k 來進行模擬，但這樣會讓顯式積分法和隱式積分法都出現問題，要解決也會大量增加計算量，所以基於約束的方法被提出來。

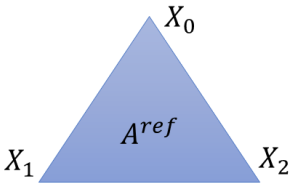
有許多實現方法，我做的是 position based dynamics(PBD)，PBD 跟 shape matching 類似，先假裝頂點間不存在彈簧，更新每個頂點的位置，再對每一個頂點進行約束並計算下一時刻頂點的速度，優點是計算快容易平行化處理，缺點是 PBD 中沒有彈性係數的概念，迭代次數多，布料的彈性就越大，並且無法獲得精確解，為了解決此問題，讓模擬更逼真，我使用了 strain limiting 方法，strain limiting 可以理解為彈簧設定有一個極限拉伸比σ。可以寫成$\sigma^{\min} \le \frac{1}{L}\left\|\mathrm{x}_i - \mathrm{x}_j\right\| \le \sigma^{\max}$，$\mathrm{x}_i$ 和x_j為彈簧兩端，L 為彈簧原長度。

在計算上可以直接以$\mathrm{x}_i^{\text{new}} \leftarrow \mathrm{x}_i - \frac{m_j}{m_i+m_j}\left(\left\|\mathrm{x}_i - \mathrm{x}_j\right\| - \sigma_0 L\right)\frac{\mathrm{x}_i-\mathrm{x}_j}{\left\|\mathrm{x}_i-\mathrm{x}_j\right\|}$、$\mathrm{x}_j^{\text{new}} \leftarrow \mathrm{x}_j + \frac{m_j}{m_i+m_j}\left(\left\|\mathrm{x}_i - \mathrm{x}_j\right\| - \sigma_0 L\right)\frac{\mathrm{x}_i-\mathrm{x}_j}{\left\|\mathrm{x}_i-\mathrm{x}_j\right\|}$來更新頂點的位置，m 為頂點質量，在模擬中因布料材質相同，故$\frac{m_j}{m_i+m_j}$我直接設為$\frac{1}{1+1} = 0.5$。

4. 彈性體模擬

彈性體與材料力學較為相關，對於彈性體計算，主流的模型有 Neo-Hookean、Mooney-Rivlin、Fung、Saint Venant-Kirchhoff(StVK)等等，我選用較容易實現的 StVK 方法和，以及使用有限體積法(finite volume method, FVM)進行實作，不過這裡 FVM 和有限元方法(FEM)是等價的。

StVK 是被用來當作能量密度函數，設 E 為能量，W(G)為能量密度，會直接決定材料性質，A^{ref}是靜止狀態算出來的面積，$E = \int W(\mathrm{G})dA = A^{\text{ref}}W(\varepsilon_{uu}, \varepsilon_{vv}, \varepsilon_{uv})$，$W(\varepsilon_{uu}, \varepsilon_{vv}, \varepsilon_{uv}) = \frac{\lambda}{2}(\varepsilon_{uu} + \varepsilon_{vv})^2 + \frac{\mu}{2}(\varepsilon_{uu}^2 + \varepsilon_{vv}^2 + \varepsilon_{uv}^2)$，W 對 G 偏微分得到 Second Piola–Kirchhoff Stress Tensor(用 S 代指)，最終可以推得$\mathrm{S} = \frac{\partial W}{\partial \mathrm{G}} = 2\mu\mathrm{G} + \lambda\,\mathrm{trace}(\mathrm{G})\mathrm{I}$，S 可以當作是力的密度，後續才能把受力算出來。

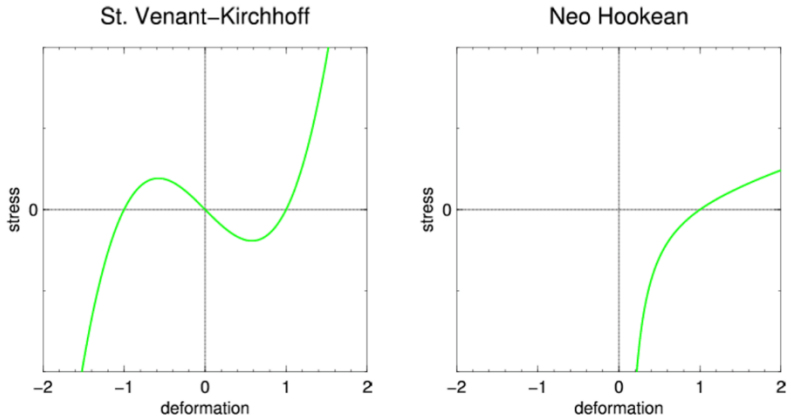


圖三、示意用的三角形。A^{ref}為未產生形變前的面積，X_0、X_1、X_2為三角形三頂點。

$$\mathrm{f}_i = -\left(\frac{\partial E}{\partial \mathrm{x}_i}\right)^{\mathrm{T}} = -A^{\mathrm{ref}}\left(\frac{\partial W}{\partial \mathrm{x}_i}\right)^{\mathrm{T}} = -A^{ref}\left(\frac{\partial W}{\partial \varepsilon_{uu}}\frac{\partial \varepsilon_{uu}}{\partial x_i} + \frac{\partial W}{\partial \varepsilon_{vv}}\frac{\partial \varepsilon_{vv}}{\partial x_i} + \frac{\partial W}{\partial \varepsilon_{uv}}\frac{\partial \varepsilon_{uv}}{\partial x_i}\right)^{T}$$

$$= -A^{\mathrm{ref}}\mathrm{FS}\begin{bmatrix} a \\ b \end{bmatrix}$$

最終會變成$[\mathrm{f}_1 \quad \mathrm{f}_2] = -A^{\mathrm{ref}}\mathrm{FS}[\mathrm{X}_{10} \quad \mathrm{X}_{20}]^{-\mathrm{T}}$，最終採用此公式就可以快速的把頂點受力算出來。在一開始模型蠻常運行二三十秒就崩潰的，原本以為原因是 StVK 模型在受力過大讓頂點間距離移動得太近，此時頂點間的張力可能會變小甚至消失的緣故。



圖四、StVK 和 Neo-Hookean 兩種模型在不同的形變量下產生的壓力。StVK 在形變過一定程度之後，產生的張力會越來越小(Irving et al. 2004.)。

後來發現是因為沒有做速度平滑所導致，讓模型穩定了下來，也可以很好的完成彈性體模擬的任務。雖然寫程式的過程當中遭遇了許多困難，但實際上核心計算用的程式並不多。

5. 流體模擬

原先預計採用 smoothed-particle hydrodynamics(SPH)方法來完成這一部分，透過將流體拆分成許多離散的粒子，透過計算不同粒子間的質量、密度、速度等等來模擬出流體的樣貌。

因能力不足，故先以 Shallow Wave Equation (模擬精確度上較低)完成此部分的撰寫並嘗試做流固耦合，本模擬以顯示歐拉法來進行計算，在模擬中也會有隨著波的傳遞，水量越來越多的問題。

Shallow Wave Equation: $\frac{d^2h}{dt^2} = \frac{h}{\rho}\frac{d^2P}{dx^2}$，離散化之後可以解出在陣列上，下一時間水柱高度和周圍水柱高的關係，$h_{i,j}^{new} < -h_{i,j} + \left(h_{i,j} - h_{i,j}^{old}\right) * damping + \left(h_{i+1,j} + h_{i-1,j} + h_{i,j+1} + h_{i,j-1} - 4 * h_{ij}\right) * rate$，依照此式子求出下一時間水柱高即可。

III. 結果討論

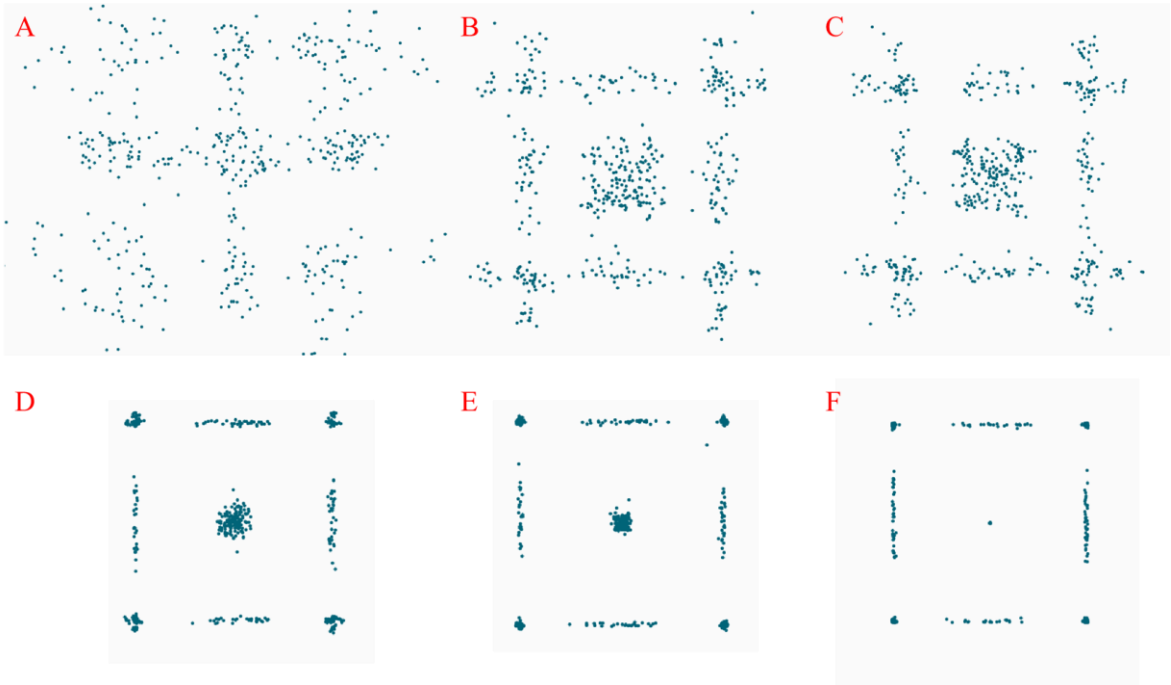
程式碼與成果影片皆開源於 https://github.com/afan0918/coding-programming，另外因為是基於物理模擬，所以還是希望以影片觀看的方式呈現。

1. 集群行為

我實現了 PSO 演算法並成功地將過程進行了視覺化，主程式碼位於 PSO.java，對 PSO 進行迭代的部分寫在 Main.java，可以透過更改欲求解的方程，來決定粒子行為，其他部分都已經封裝好了，不用進行改動。

```java
/**
 * 要求解的方程式
 * @param x 輸入粒子位置
 * @return 粒子適應程度
 */
public double func(double[] x) {
    if (x[0] == 0 && x[1] == 0) {
        return Math.exp((Math.cos(2 * Math.PI * x[0]) + Math.cos(2 * Math.PI * x[1])) / 2);
    } else {
        return Math.sin(Math.sqrt(Math.pow(x[0], 2) +
                Math.pow(x[1], 2))) / Math.sqrt(Math.pow(x[0], 2) +
                Math.pow(x[1], 2)) + Math.exp((Math.cos(2 * Math.PI * x[0]) +
                Math.cos(2 * Math.PI * x[1])) / 2);
    }
}
```

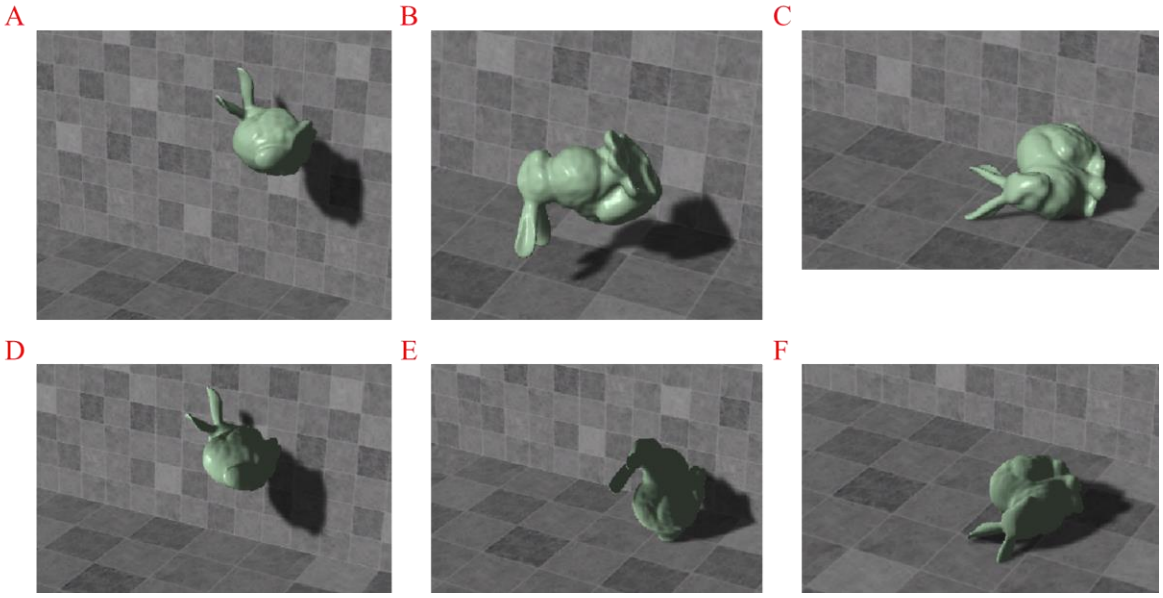
圖五、PSO 演算法的設定求解方程的函式，透過改寫此方程可以決定粒子移動規律。



圖六、PSO 演算法粒子運動狀態圖。A-F 依序為不同時間下，粒子由隨機分布開始進行運動的表現，粒子初始以隨機分布在圖上各處，經由一定次數的迭代之後會逐漸到達他所認為的最佳位置之上。

PSO 可以在極快的時間內找到適合的解，而在動畫上也可以拿來模擬不規則移動的集群，我模擬了五百個粒子來進行移動求解的過程。

2. 剛體模擬

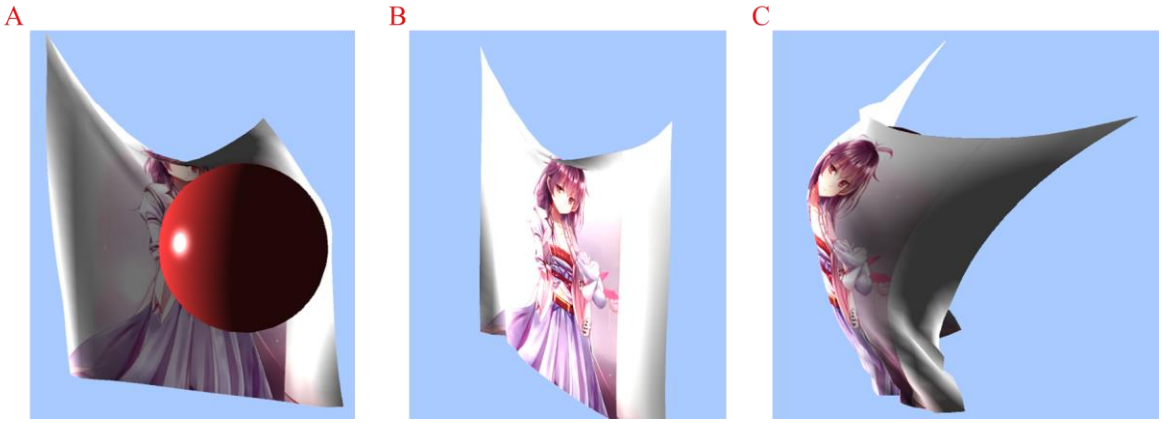


圖七、分別以 Impulse 和 Shape matching 實作的剛體模擬。ABC 為 Impulse 的做法，CDE 為 Shape matching 的做法，可以在影片中觀看，C 和 F 的耳朵可以明顯看出有長短差異，F 的兔耳有部分插入了地板內。

理論上 Impulse 和 Shape matching 兩種方法只要調好參數可以造成一樣的視覺效果，不過 Shape matching 方法因為參與碰撞的粒子越多，所能透過形變獲得的力越大，所以像是兔子尾巴、耳朵等體積較小之處，容易有明顯的陷入地面內的情況。

Impulse 法計算量較小，但需要考慮角速度計算，以及對部分情況進行單獨處理(例如需要衰減反彈係數，不然模型會不斷有微小震動)，Shape matching 法計算量較大，不過不需考慮角速度計算，在程式撰寫上較好進行實現。

3. 布料模擬

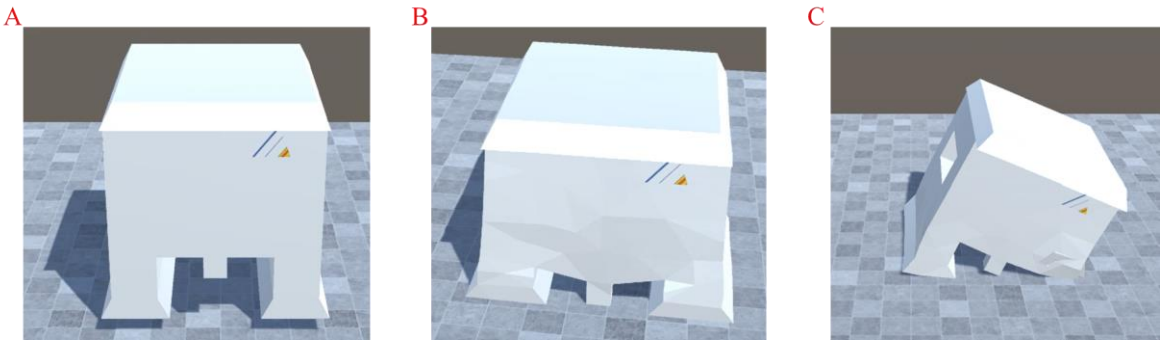


圖八、使用 PBD 方法進行的布料模擬。A 展現布料在受力下的皺褶情形，B 為布料因重力自然下垂的情形，C 是強力拉伸下的情形，可以看的出布料的拉伸、皺褶等特性。

此紋理是使用 GAN(對抗神經網路)生成出來的，較沒有版權問題，PBD 在一千個網格以下，基本上都可以進行實時渲染，在 此模擬中，僅將布料拆成 21*21 的網格就有不錯的效果，感覺上還是成功的。

4. 彈性體模擬

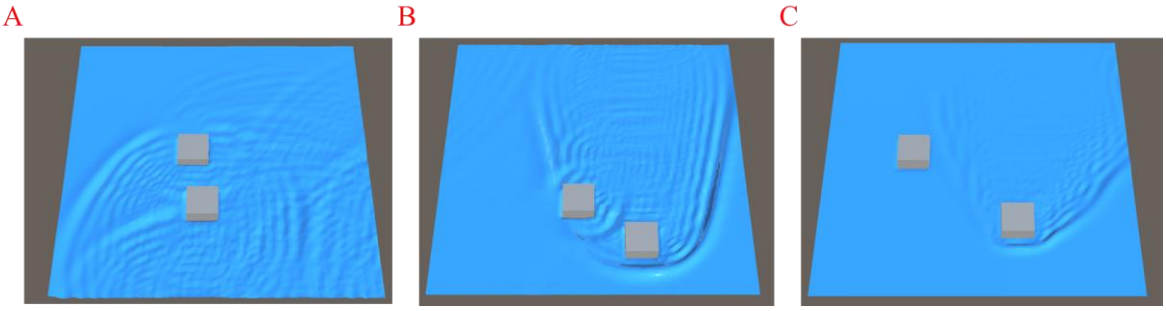
彈性體模擬是我目前為止寫最久的部分，也重寫了許多次，因為彈性體有許多計算能量的方法，所以有單獨把方法抽離出來，若要使用的話可以直接改寫 P 矩陣的計算方法即可。



圖九、使用 StVK 方法實作的彈性體模擬。A 為未發生形變時房子的形狀，B 為發生形變時的房子的情形，C 是若受力過大時，依據 StVK 實現的模型會發生崩壞的情況(無法還原)。

5. 流體模擬

以淺水模型來進行的實現，這邊比較特別的是有試著做流固耦合(實現流體和固體的交互影響)的動作，不過目前只有把固體影響流體(透過給予固體所在位置一個虛擬的水柱高，讓固體下的水會被排開)的部分完成，流體對固體的影響會難以實現很多，所以還在逐步實現當中。



圖十、使用 Shallow water wave 方法實現的流體模擬。A 為波比較小時的餘波，BC 都是方塊前進時產生的波，差異為 B 的水較深，而 C 的水較淺，造成波的大小有明顯不同。

IV. 結論

我透過了許多方法來使用粒子系統此工具，來完成不同的任務，也期許以後可以使用相關的技術來完成更大型的模擬設計，並將目前的程式碼進行開源，除了集群運動是使用 java+processing 的方式來實現外，其他都是透過 unity 來進行相關的模擬，這樣也可以避免掉寫渲染引擎的麻煩以及只需要專注於撰寫物理方法。

VR 因為近年來設備的計算能力大幅增加,也逐漸崛起,為了避免使用者有 3D 暈的情形，VR 眼鏡需要做到 90fps 的計算速度，並且不讓使用者感到突兀，各種基於物理且能快速計算的模擬方法也越發重要，使用 unity 也可以快速且簡單的在未來把模型遷移到 VR 內。在前文中可以看到多種方法的渲染效果，以及像是 PBD 中因為傳統做法並沒有物理意義，所以實作了目前比較主流物理模擬方法，加上 strain limiting 來進行約束讓動畫更加擬真，後續可能就是因應材質和特殊情況去調整參數和做法，以及擴充一些數值計算方法，畢竟目前有許多計算方法讓想進行模擬的研究者只需要根據計算開銷和適用情境去選擇想要使用什麼方式就好，我也希望讓這部分變成可以抽換的套件讓其他人可以選擇，並在不斷的實作中能夠慢慢追上目前的前沿研究和把模擬做的更好。

V. 參考文獻

剛體模擬相關

1. Muller et al. 2005. *Meshless Deformations Based on Shape Matching.* TOG (SIGGRAPH).

布料模擬相關

2. Baraff and Witkin. 1998. *Large Step in Cloth Simulation.* SIGGRAPH.
3. Bridson et al. 2003. *Simulation of Clothing with Folds and Wrinkles*. SCA.
4. Bergou et al. 2006. *A Quadratic Bending Model for Inextensible Surfaces*. SCA.
5. English and Bridson. 2008. *Animating Developable Surfaces Using Nonconforming Elements*. SIGGRAPH. (optional)

彈性體模擬相關

6. Irving et al. 2004. *Invertible Finite Elements For Robust Simulation of Large Deformation.* SCA
7. Wang. 2016. *Descent Methods for Elastic Body Simulation on the GPU.* TOG (SIGGRAPH Asia).
8. Xu et al. 2015. *Nonlinear Material Design Using Principal Stretches.* TOG (SIGGRAPH).

流體模擬相關

9. Kass and Miller. 1990. *Rapid, Stable Fluid Dynamics for Computer Graphics*. Computer Graphics.
10. Jos Stam. 1999. *Stable Fluids.* TOG (SIGGRAPH).

四、專題製作心得(2 頁以內)

隊員 1: 陳繹帆

去年參加了一次，因為有得名，所以專題不好拿來比賽，就想說既然有這個機會，那就來做一下自己喜歡的東西吧，畢竟學習應該要是一件開心的事情吧，另外因為本競賽的主旨為切磋，所以程式碼也都開源在 GitHub 之上，讓大家可以相互學習，海報展當天我也會帶筆電到現場讓大家可以玩玩看我到底在搞什麼鬼，也可以看看我放在 github 的影片，模擬的東西不看影片靠我自己人工製作過程 gif，總感覺會漏掉許多東西。

我原先比較擅長的部份應該是機器學習與類神經網路的模型，上學期在修類神經網路時還有順便開源一個用 java 寫的單層與多層感知機函式庫等等，但我其實沒有到說很喜歡，畢竟深度學習需要家底，每次比賽都要借顯卡，然後大家基本上也只能套模型對特定情況做訓練，跟許多組別的同質性可能會太高，後來就想說乾脆來做粒子系統的物理模擬好了，畢竟這樣感覺比類神經網路更符合工學院一點，不過因為去年的專題今年還在做，所以這其實是我自己做的第二個專題，因此有點趕和發生了一點難以兼顧的情況，這邊要謝謝指導教授的支持。

去年因為興趣,跑去看了加州大學聖芭芭拉分校Lingqi Yan教授開的計算機圖學入門課，覺得哇渲染景物也太厲害了吧,後續繼續學習了 Lingqi Yan 教授開設的 GPU 實時渲染課程，只是由於基礎不夠，作業寫到一半就完成不了了，變成單純只是上課聽講，但有種半途而廢的遺憾。這次最初的動機是在 Lingqi Yan 教授的課程中看到別人做的海水潮汐模型，覺得哇好厲害，我也想自己寫一個，所以就開始動手自己實作，然後發現自己對於水體如何模擬一點頭緒都沒有，就想說那還是慢慢來好了，畢竟酷酷的題目哪有一下子就能完成的，由於需要足夠擬真的話，基本上都是把水體視為許多水粒子來進行模擬後再進行渲染，所以我也想說從粒子系統開始進行學習。

首先吸引我注意的是 PSO(粒子群演算法)，該演算法最初是用來描述鳥群的移動，後來被延伸成為解最佳化問題的演算法，實現完之後，我對於粒子間的交互關係更為著迷。後來有另外寫了一個太陽系，只是寫完行星之間的交互關係(只需要考慮重力)之後，發現照實際比例縮放渲染完根本看不到，就覺得自己好像有點欠缺考慮了，應該要再多上一點課，就開始上了俄亥俄州立大學的王華民教授的基於物理的計算機動畫入門，並大量的參照了教授的投影片做了這次比賽的內容，上課的收穫真的滿大的。

過程中常常會有為什麼模型動起來不符合視覺或者是模型直接崩潰的情況，尤其是彈性體的部分，原先以為是 StVK 不太能處理兩節點之間形變量過大的情況，只好在模擬過程中不斷不斷的調參，結果最終發現是沒做好速度平滑的問題，但圖學的快樂也在於此吧，寫得不對只要看看模型就知道自己錯了。而且，模型動起來的瞬間，是真的很感動，在不同的情況下,都有許多的模擬方法都可以完成數值的計算,有各自擅長的地方和用途,真的很好玩，雖然我還是討厭材料力學(被打死)。

然後因為真的是邊學邊做的，所以難免有些粗糙的地方，尤其是流體是趁著交件延期才能勉強趕出來的，所以最後題目不敢下的太大，是「以粒子系統進行擬真動畫繪製」,流體的部分也不是以粒子系統的方式實作(因為 Shallow water wave 相對於純以粒子系統進行模擬好實作太多了，也不用顧慮到最終的渲染部分)，但也不好意思再跟系辦說要改題目了，就希望能再海報展前補上一些趕程式趕的倉促的遺憾。

最後,有點抱歉的是太多東西想寫了,但沒想到報告書寫完發現更多的東西沒有寫上去，程式碼也只挑了核心實現的數學方法來進行推導並省略了大量過程，感覺有點抱歉，然後沒想到一份報告要寫這麼久，氣死。不過好像也挺好，再把公式敘述出來的時候，又對中間的數學方法更了解了一點，又查缺補漏出了之前忽略掉的一些小細節。

剩下的，就用作品來說話吧。