FINAL PROJECT

Rendering Milk Tea

Digital Image Synthesis 2010

B96203005 化學四 曾紀為

2010/1/22

動機

喝奶茶是一種享受，而且，不只在味覺上是，視覺上也是，尤其在剛倒入奶精的時候，在依然透明的紅茶茶面下，我們可以看到奶精先是下沉，然後以一小球小球的形狀，逐件爆裂開並浮上茶面，要過好一陣子，茶面的顏色才會擴散成均勻的材質。我覺得這段擴散的過程看起來很漂亮，卻很少在3DCG中出現，因此偶然想到，或許是個可以在 final project 中玩玩看的題目。

我計畫的實作方向，主要分為兩個部份，第一是生成奶精在紅茶中的運動，可以使用流體力學中的 Navier-Stokes 方程式來模擬，我找到 1999 年 Jos Stam 在 SIGGRAPH Proceedings 上發表了一個用以演化 Navier-Stokes 方程式，且數值穩定的演算法，作為參考[1]。第二則是生成奶精在紅茶中的形狀，可利用 metaball 製造出 implicit surface 來產生。雖然不確定這樣能不能真的能達到我想要的效果，但就姑且一試吧。

理論

1. Stable Fluid

電腦圖學中，Navier-Stokes 方程式較常見的形式為：

$$\nabla \cdot \mathbf{u} = 0$$
$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{1}{\rho}\nabla p + \nu\nabla^2\mathbf{u} + \mathbf{f}$$

其中，$\mathbf{u}$為液體的速度向量，ρ為液體密度，p為壓力，$\mathbf{f}$則為外力之向量。第一式的物理意義代表液體具有不可壓縮性(例如在正常狀況下，我們不能將兩公升的液體壓縮成一公升)，直觀來說，便是在液體內任取一個小體積，液體進與出的量必須相等。第二式則代表液體須遵守動量守恆，故液體速度的變化(左式)，可能來自於右側的任何一項，分別代表傳導力、壓力、黏滯力和外加力的貢獻。

根據 Jos Stam 的推導，上面液體速度的向量$\mathbf{u}$，事實上可以用 Helmholtz-Hodge Decomposition 看作一個新的向量$\mathbf{w}$在 divergence 為 0 空間上的投影向量，也就是說：

$$\mathbf{w} = \mathbf{u} + \nabla q$$

因此我們可以定義一個 operator $\mathbf{P}$，用以將w投影到 divergence 為 0 的空間上：

$$\mathbf{Pw} = \mathbf{u}$$

而且：

$$\mathbf{Pu} = \mathbf{u}$$

將此 operator 同時加在 Navier-Stokes 方程式中第二式的左側和右側，整理後便可推出：

$$\frac{\partial \mathbf{u}}{\partial t} = P[-(\mathbf{u} \cdot \nabla)\mathbf{u} + \nu \nabla^2 \mathbf{u} + \mathbf{f}]$$

於此，我們已將壓力與速度的關係合併。我們可用此式，對任何使用者提供的流體速度場進行演化，且任何一步的演化將分為四個階段，前三階段對應到上式右邊的三項，最後一階段則是因為系統在前三階段完全是毫無限制地自由演化，因此我們須將此時得到的結果壓回 divergence 為 0 的空間內，以符合前述 Navier-Stokes 方程式的第一式。

實作細節

1. Stable Fluid 的演化過程

我們演化的系統，是將一個$6 \times 6 \times 6$的正方體空間，每個格子點上均儲存一個

速度向量，意即$\mathbf{u}(\mathbf{x}, t)$，其中$\mathbf{x}$為空間座標，t為演化時間。且為 periodic boundary condition。四個階段依 Stam 實作的順序，分別為：

(1) 施加外力

$$\mathbf{w_1}(\mathbf{x}, t_0) = \mathbf{u}(\mathbf{x}, t_0) + \Delta t \cdot \mathbf{f}(\mathbf{x}, t_0)$$

其中外力$f(\mathbf{x}, t_0)$由使用者提供。

(2) 流體傳導

$$\mathbf{w_2}(\mathbf{x}, t_0) = \mathbf{w_1}(\mathbf{b}(\mathbf{x}, -\Delta t), t_0)$$

其中，$b(\mathbf{x}, -\Delta t)$代表現在位於$\mathbf{x}$位置的粒子，在Δt時間以前的座標位置。這個部分，Stam 採用了 second order Runge-Kutta Method 進行 particle back-tracing，我的程式也採用相同的方法，這部分我沒有利用現成的函式庫，是自己實作的。

(3) 施加黏滯力

$$(1 - \nu \Delta t \nabla^2)\mathbf{w_3}(\mathbf{x}, t_0) = \mathbf{w_2}(\mathbf{x}, t_0)$$

此式可在離散的版本中，可寫為一個 sparse matrix 的 linear system，在我的實作中，是使用 GNU 的 general scientific library (GSL)函式庫，使用 singular value decomposition 來解得$\mathbf{w_3}(\mathbf{x}, t_0)$。

(4) 限制 divergence 為 0：

$$\nabla^2 q(\mathbf{x}, t_0) = \nabla \cdot \mathbf{w_3}(\mathbf{x}, t_0)$$
$$\mathbf{u}(\mathbf{x}, t_0 + \Delta t) = \mathbf{w_3}(\mathbf{x}, t_0) - \nabla q(\mathbf{x}, t_0)$$

在此階段中，我們從$\mathbf{w_3}(\mathbf{x}, t_0)$中取出了$\nabla^2 q(\mathbf{x}, t_0)$，是$\mathbf{w_3}(\mathbf{x}, t_0)$不 divergence free 的部分，最後，我們將這部分從$\mathbf{w_3}(\mathbf{x}, t_0)$中消去，便得到了下個 frame 的速度場$\mathbf{u}(\mathbf{x}, t_0 + \Delta t)$。

留意此階段中的第一式是個 Poisson equation，在離散版本中也可寫為一個 sparse matrix 的 linear system，我同第三階段，也是使用 GSL 函式庫的 singular value decomposition 來解。

2. Metaball

速度向量場建立好了之後，我們就可以把任何我們想要的粒子放入其中，進行演化。

$$\mathbf{x_p}(t_0+\Delta t)=\mathbf{x_p}(t_0)+\mathbf{u}\big(\mathbf{x_p}(t_0),t_0\big)\Delta t$$

其中$\mathbf{x_p}$為粒子的球心位置。而速度$\mathbf{u}$則在 Navier-Stokes 的格子點之間用 cubic 取 interpolation 內插而得。由於初加入紅茶中的奶精，具有結成一球球上浮的性質，看起來像是球體彼此吸附而變形，因此我使用 metaball 作為粒子。每個 metaball 除了儲存了球心的資料之外，也儲存了一個定義域為 3D 空間的函數$f(\mathbf{x})$，而我們可以自行定義一個 threshold d。若

$$f(\mathbf{x})\geq d$$

時，代表(x,y,z)這個點位於物體的內部，反之，則位於物體的外部。也就是說，metaball 決定出一個 implicit surface，其方程式為：

$$f(\mathbf{x})=d$$

函數$f(x,y,z)$在應用上有許許多的變形，也很容易自行設計。常見的例子如[2]：

$$f\big(\mathbf{x}-\mathbf{x_p}\big)=\frac{1}{|\mathbf{x}-\mathbf{x_p}|}$$

或是含有 cutoff 部分的[3]：

$$f\big(\mathbf{x}-\mathbf{x_p}\big)=\begin{cases}a\left(1-\dfrac{3\left|\mathbf{x}-\mathbf{x_p}\right|^2}{b^2}\right), & 0\leq\left|\mathbf{x}-\mathbf{x_p}\right|\leq\dfrac{b}{3}\\ \dfrac{3a}{2}\left(1-\dfrac{\left|\mathbf{x}-\mathbf{x_p}\right|}{b}\right)^2, & \dfrac{b}{3}<\left|\mathbf{x}-\mathbf{x_p}\right|\leq b\\ 0, & |\mathbf{x}-\mathbf{x_p}|<b\end{cases}$$

我則自己測試了幾個不同的函數f，依據螢幕上呈現的 implicit surface 結果修改它，最後使用了：

$$f(\mathbf{x})\equiv\frac{0.003a^2}{|\mathbf{x}|^2}$$

而 threshold $d\equiv0.5$。a常數可自行取到合適的值。而之所以分母採用$|\mathbf{r}|^2$而非

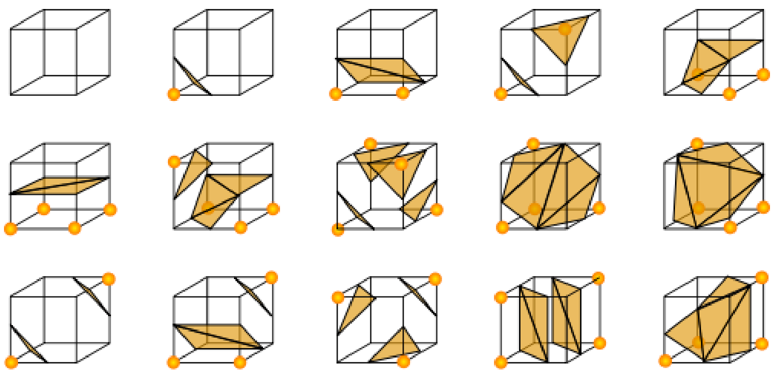
|r|，是為了避開電腦中昂貴的平方根運算。

3. Marching Cubes

將 metaball 定義的 implicit surface 轉為易於 render 的 mesh 的演算法，常用的有 brute force raycasting 和 marching cubes，我使用了後者。marching cubes 是將 3D 空間切割成許多 Voxel，每個 Voxel 在它的 8 個頂點上都儲存了前述 metaball 中物質密度的數據，我們便可以判斷這 8 個頂點，那些位於 implicit surface 的內部，哪些位在 implicit surface 的外部，並找出這個 voxel 的 12 條邊之中，哪幾條與 implicit surface 相交。

找到此 voxel 與 implicit surface 相交的一組邊之後，我們在這些邊上用線性內插估計 implicit surface 通過的點，最後對每個 voxel，把這些點連成 0 到 5 個三角形集合成的 mesh。所有 mesh 接起來，就成為我們想得到的模型了。

marching cubes 經常不透過計算，而透過查表來實作，因為每個 voxel 只有 8 個密度數據，這些密度數據，不考慮內插的話，只會被我們分為位於 implicit surface 內和外兩個類別，因此，implicit surface 在此立方體中可能的結構數，其實只有$2^8 = 256$種可能性，不算太難處理，透過查表法，又可以大幅增加程式的效能。以下是這 256 種可能性中，除去旋轉、鏡射造成之重複後，剩下的 15 個 implicit surface 在 voxel 中的結構[4]。



我所使用的 marching cubes 結構表，是利用網路上的資源，由 Cory Gene Bloyd 製作，程式部分，則參考了 Paul Bourke 的程式碼。網址列於參考資料中[5]。

程式介面與功能

我寫的程式是一個獨立於 PBRT 外的系統，使用 OpenGL 作為介面，在 Ubuntu10.10 的環境下編譯，編譯方式請見壓縮檔中附上的 readme file。

這個系統雖然獨立於 PBRT 之外，但最後可以匯出成.pbrt 的檔案格式，再 include 到 PBRT 中 render。由於這個程式的目的，純粹是實驗性地生成我想要的 3D 模型，所以我並沒有把使用者介面寫得很好，也沒有讓程式讀入 input file。有不少的參數是要在 compile-time 從 c code 中去修改的，這些參數如：液體黏度(Main.cpp)，液體所受的外力向量場(Main.cpp)，單顆 metaball 的密度函數(DensityParticle.cpp)，粒子初始位置(StableFluid.cpp)，粒子初始速度(Main.cpp)，粒子數量(Main.cpp)等等。

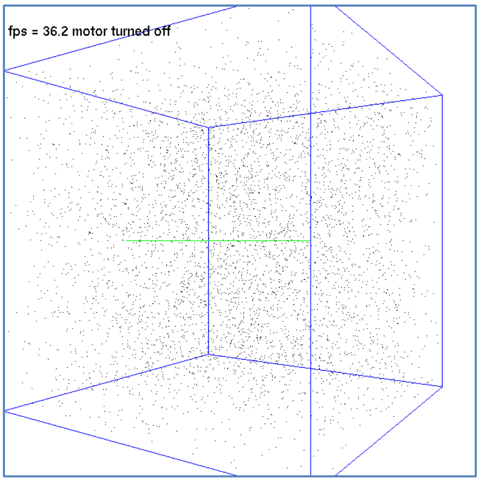
我預設粒子受到的外力場為一個延著 z 軸旋轉並往-z 方向移動的力場。粒子數量為 1000，初始速度為 0。在這些參數的調整之外，可以在 run-time 使用的操作介面為：

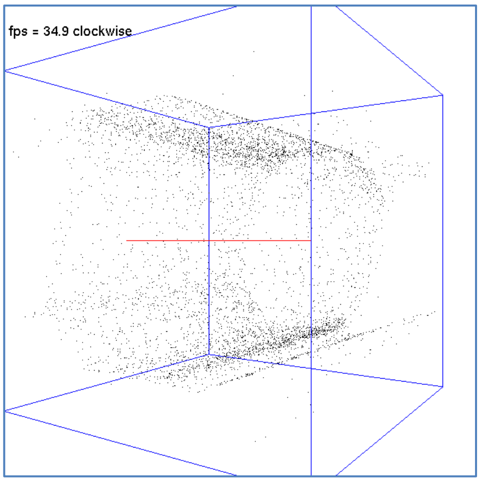
F1：控制外力場的開關。
F2：將外力場的旋轉軸回到正中央。
F3：將外力場旋轉方向顛倒。
F4：暫停動畫。
F5：將目前畫格的 implicit surface 計算出來，轉為 pbrt 的 triangle mesh。
方向鍵：移動旋轉軸的中心位置。

為了要達成系統的即時性，如實作細節中所述，我所模擬的系統大小只有 6×6×6，因此在演化時間一久，voxel 和 voxel 的邊界上就很容易因為內插的關係產生看起來不連續的粒子堆積。目前我還沒想到，要如何在不降低 fps 的條件下，避免這個問題發生。

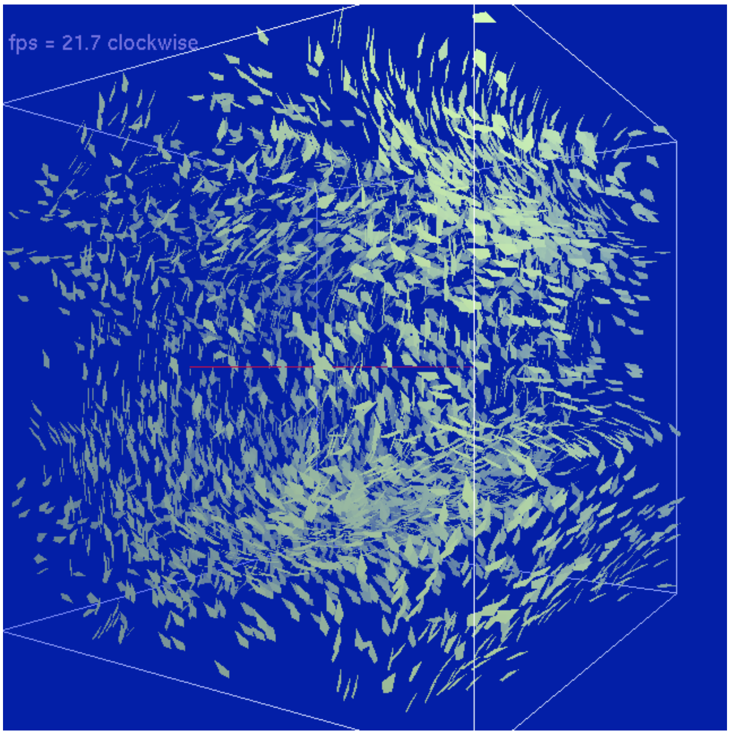
圖像生成流程

1. 測試階段，左側為初始畫面，右側為施加外力場之後：

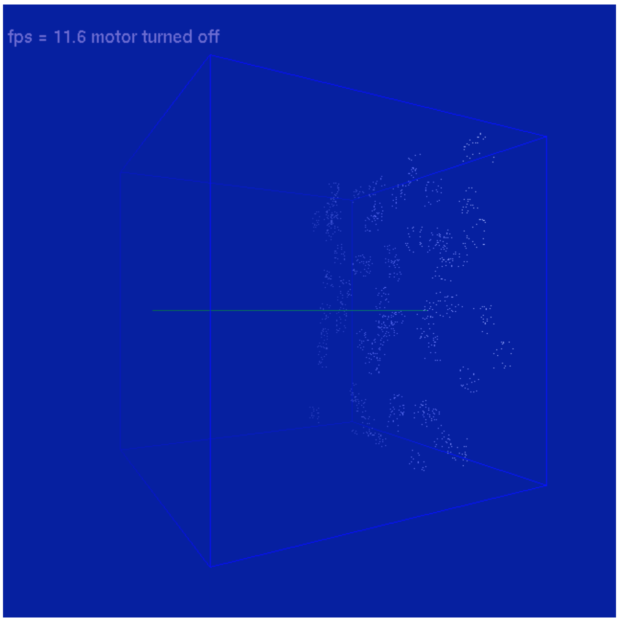


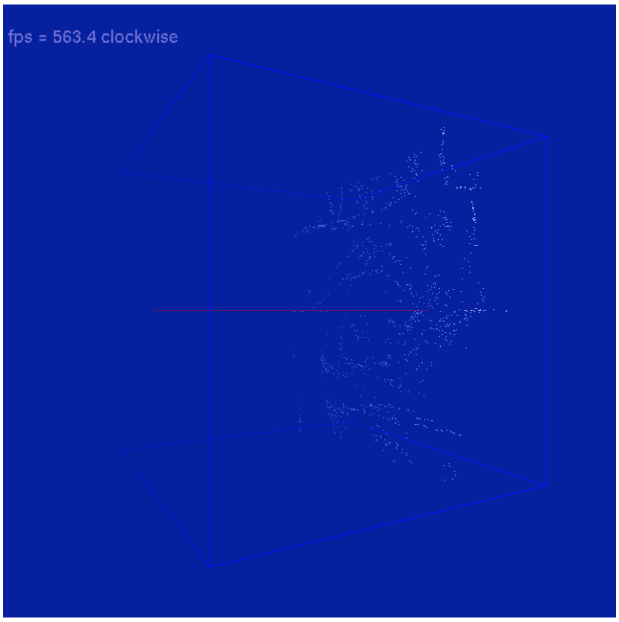


2. 將粒子改為幾何形狀後的測試畫面：

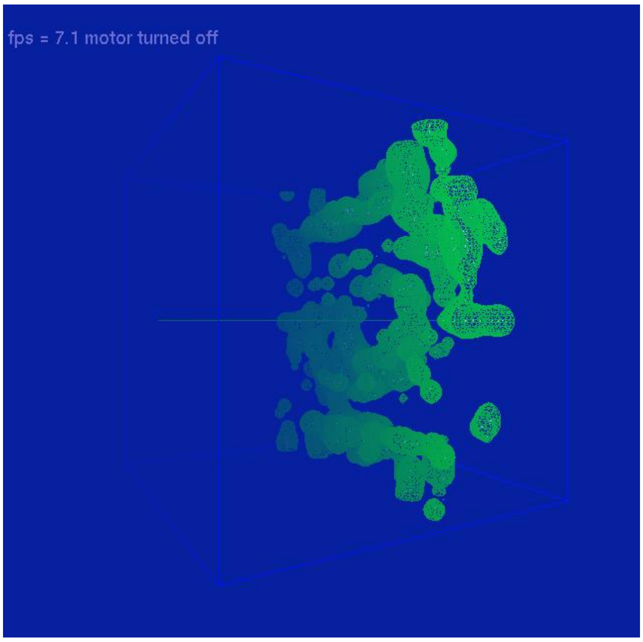


3. 改變粒子的初始分佈，並以 Navier-Stokes 方程式演化後：

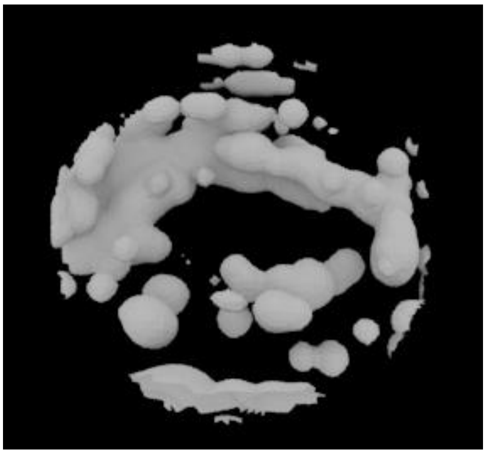




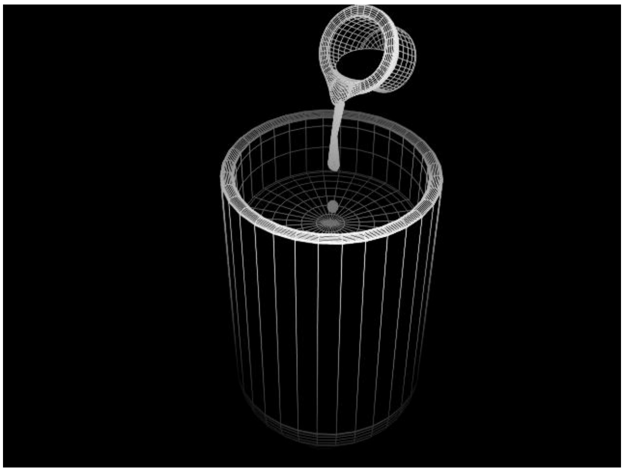
4. 粒子換為 Metaball 後，經 marching cubes 生成的 mesh：



5. 切下 mesh 的中央部份，並輸出成 pbrt 檔案後的 render：



6. 我使用其它 3D 建模軟體做出的紅茶、奶油球和玻璃杯模型：



7. 奶精的 mesh 和玻璃杯合併 render 的結果：

完成圖

(解析度： 800 × 800 pixels。計算時間：4150 秒)

參考資料

[1] Jos Stam. “Stable Fluids”. 1999. Proceedings of the 26th annual conference on Computer graphics and interactive techniques, 121-128.

[2] “Wikipedia - Metaball”. http://en.wikipedia.org/wiki/Metaball/

[3] Paul Bourke. “Implicit Surfaces”. 1997. http://local.wasp.uwa.edu.au/~pbourke/miscellaneous/implicitsurf/

[4] “Wikipedia – Marching Cubes”. http://en.wikipedia.org/wiki/Marching_cubes/

[5] Paul Bourke. “Polygonising A Scalar Field”. 1994. http://local.wasp.uwa.edu.au/~pbourke/geometry/polygonise/