計算機圖學

學號：B0829059
姓名：呂欣玲

1. (5%) 請以 Ch9 (Programmable Shaders), Ch10 (Modeling and Procedural Methos), Ch11 (Scene Graphs and Real Time), Ch12 (Curves and Surfaces) 之任一或多個主題內容設計一個「具有故事性」的計算機繪圖作品。請提出你的作品題目(5%)，題目必須與第一次作品有直接相關，可以用子標題顯示增強之功能，若與第一次作品無關，則必須說明重新製作的理由。

   [Ans]

   我的題目是：3D CUBE:渲染分裂和物體碰撞(加強前兩次作品的功能)

2. (15%) 請依據你所提出的題目，列出它所要實現的內容，至少三項。

   [Ans]

   (1) 物體碰撞到邊界或粒子之間碰撞時，必須做出正確的反應
   (2) 按滑鼠右鍵時，必須跳出 menu 且可以選擇欲操作的動作
   (3) 可以調整粒子的參數，包含個數、邊界及速度等等

3. (15%) 請具體說明上述作品內容，以及它與目前課程主題之相關性。

   [Ans]

   Ch10 (Modeling and Procedural Methos): 粒子系統可以定義為一個粒子集合的陣列，在沒有碰撞檢測的情況下，粒子沿著這個反射的距離等於它透多邊形的距離。

   Ch11 (Scene Graphs and Real Time): 對於要符合現實世界的模擬模型，基礎過程進行建模，且圖形的驅動一定符合物理理論。保持兩個物體的碰撞，並跟蹤它們的變化。

4. (20%) 請列舉上述實作項目中主要使用的 OpenGL 函數，以及程式設計邏輯。

[Ans]

(1) 粒子生成初始化

以陣列 particles 來儲存粒子的質量、顏色、目前座標及速度

i. particles.color: 粒子顏色

ii. particles.velocity: 粒子速度

iii. particles.position: 粒子位置

iv. platform: 控制碰撞初始邊界範圍

*細部解釋如註解

```
for (i = 0; i < num_particles; i++)
{ // 初始化粒子質量, 顏色, 座標, 速度
    particles[i].mass = 2.0;
    particles[i].color = i % 8;
    for (j = 0; j < 3; j++)
    {
        //位置以速度更新
        //速度通過計算該粒子上的力更新
        particles[i].position[j] = 0.5 * platform * ((float)rand() / RAND_MAX) - 0.5 * platform;
        particles[i].velocity[j] = speed * 1.0 * ((float)rand() / RAND_MAX) - 1.0;
    }
}
glPointSize(point_size); // 設定粒子大小
glEnable(GL_DEPTH_TEST);
```

(2) 粒子狀態更新

粒子的位置是用速度來更新的，而速度是通過計算該粒子上的力來更新的，其中更新因素包含作用在粒子上的力，將計算結果存回一個陣列(particles)中，將使用碰撞函數來保持粒子在一個固定邊界中。*細部解釋如註解

i. 若是兩粒子碰撞(repulsion)，根據兩粒子之間的位置做更新

ii. 開啟重力(gravity)向下 1 單位時，會模擬在有重力的情況下的碰撞情況。

iii. 一次碰撞需要改變法線方向上的方向速度。如果回復係數(coef)小於 1.0，那麼粒子在碰到盒子的一個側面時速度將會減慢，

*細部解釋如註解

```
        for (j = 0; j < 3; j++)
        {
            //根據作用在粒子上的力更新狀態
            particles[i].position[j] += dt * particles[i].velocity[j];
            particles[i].velocity[j] += dt * forces(i, j) / particles[i].mass;
        }
        collision(i);
    }
    if (repulsion)  //粒子之間碰撞
        for (i = 0; i < num_particles; i++)
            for (k = 0; k < i; k++)
            {
                d2[i][k] = 0.0;
                //碰撞後狀態更新
                for (j = 0; j < 3; j++)
                    d2[i][k] += (particles[i].position[j] - particles[k].position[j]) *
                                (particles[i].position[j] - particles[k].position[j]);
                d2[k][i] = d2[i][k];
if (gravity && j == 1)
    force = -1.0; // 方向向下1單位的重力
if (repulsion)
    for (k = 0; k < num_particles; k++)
    { // 計算粒子之間的碰撞
        if (k != i)
            force += 0.001 * (particles[i].position[j] - particles[k].position[j]) / (0.001 + d2[i][k]);
    }
```

(3) 操作選單

在 main function 中設定每個操作的名字和對應的變數，再利用 switch 根據變數做出相應的操作。

*細部解釋如註解

```
glutCreateMenu(main_menu); // 建立操作選單
glutAddMenuEntry("more particles", 1);
glutAddMenuEntry("fewer particles", 2);
glutAddMenuEntry("faster", 3);
glutAddMenuEntry("slower", 4);
glutAddMenuEntry("larger particles", 5);
glutAddMenuEntry("smaller particles", 6);
glutAddMenuEntry("toggle gravity", 7);
glutAddMenuEntry("toggle restitution", 8);
glutAddMenuEntry("toggle repulsion", 9);
glutAddMenuEntry("[C mode] mask texture bigger", 10);
glutAddMenuEntry("[C mode] mask texture smaller", 11);
glutAddMenuEntry("[M mode] object space bigger", 12);
glutAddMenuEntry("[M mode] object space smaller", 13);
glutAddMenuEntry("quit", 14);
glutAttachMenu(GLUT_RIGHT_BUTTON);
```

```
case (1): // 2倍粒子數量
    num_particles = 2 * num_particles;
    init();
    break;
case (2): // 1/2倍粒子數量
    num_particles = num_particles / 2;
    init();
    break;
case (3): // 2倍粒子速度
    speed = 2.0 * speed;
    init();
    break;
case (4): // 1/2倍粒子速度
    speed = speed / 2.0;
    init();
    break;
case (5): // 2倍粒子大小
    point_size = 2.0 * point_size;
    init();
    break;
case (6): // 1/2倍粒子大小
    point_size = point_size / 2.0;
    if (point_size < 1.0)
        point_size = 1.0;
    init();
```

```
case (7): //重力切換
    gravity = !gravity;
    init();
    break;
case (8): // 粒子非彈性碰撞切換
    elastic = !elastic;
    if (elastic)
        coef = 0.9;
    else
        coef = 1.0;
    init();
    break;
case (9): // 粒子間碰撞切換
    repulsion = !repulsion;
    init();
    break;
case (10): // 2倍遮罩大小
    scale = scale * 2;
    break;
case (11):  //1/2倍遮罩大小
    if (scale == 0)
        scale = 2;
    scale = scale / 2;
    break;
```

5. (15%) 請擷圖說明實作成果。

[Ans] *本次作業增加的部分

操作說明:

[a, d, w, s] -> 控制視角,[x]- > 以 x 軸旋轉, [y] -> 以 y 軸旋轉, [z] -> 以 z 軸旋轉

[r] -> 控制自動旋轉 , [q] -> reset, [c] -> colorful, [m] -> Mult,

[o] -> Normal ,*[l] ->開啟燈光

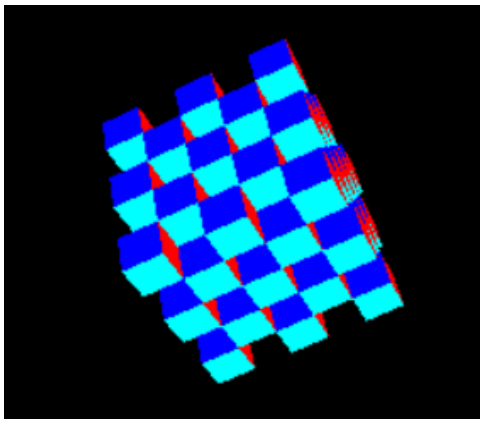
(需在 c 的模式下操控)

[<, >] -> 控制色彩渲染方向

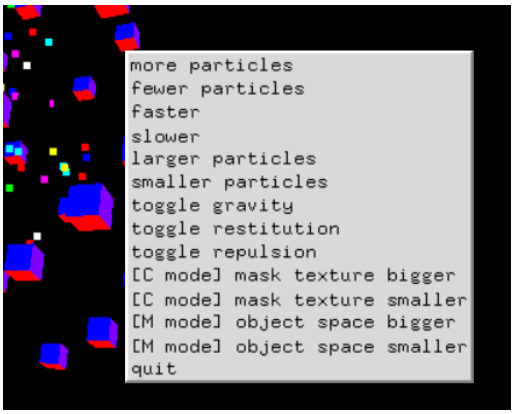
*[滑鼠右鍵] -> 開啟操作選單

[滑鼠滾輪] -> 控制遠近視角

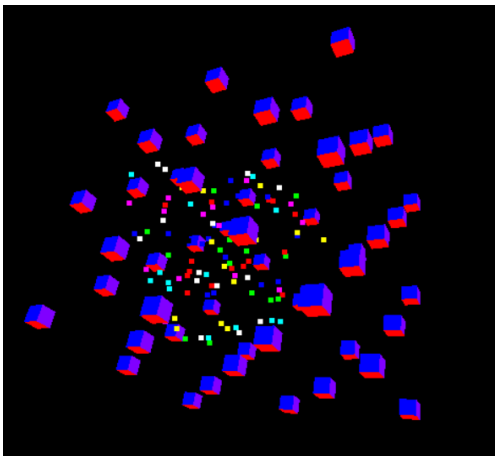
[m] -> Mult，立方體分裂為多個



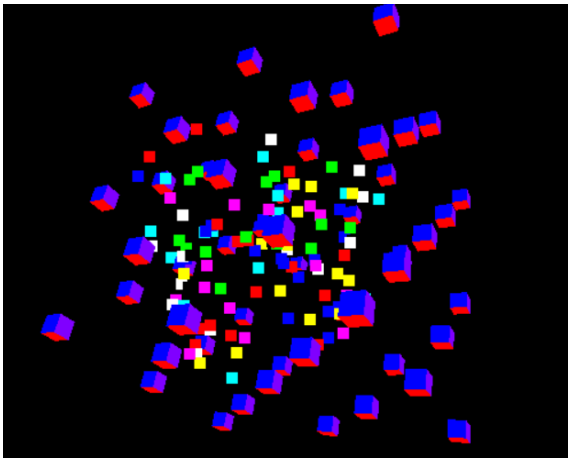
[滑鼠右鍵]　開啟操作選單



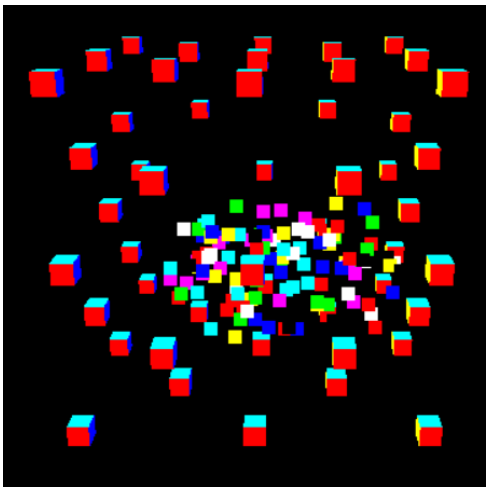
[[M mode] object space bigger]
增加物體間距，可看到碰撞中的粒子



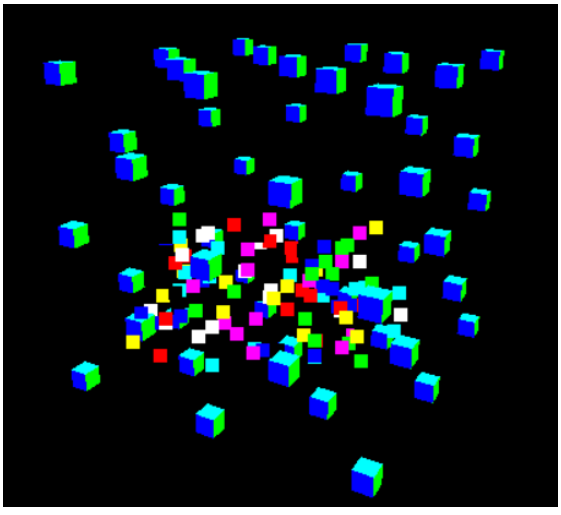
[larger particles, smaller particles]
調整粒子大小



[toggle repulsion] 預設為關閉
粒子之間的碰撞切換
->開啟狀態 粒子碰撞容易集中



[toggle gravity] 預設為關閉
系統重力切換
->開啟狀態 因重力作用粒子集中於下方



[faster,slower, toggle restitution]無法以截圖展示成果，故無截圖

6. (10%) 請說明在此成品中你最滿意的部份以及理由。

[Ans]

使用碰撞函數來保持粒子在邊界內。遞增每個粒子的位置，然後檢查該粒子是已經越過了盒子邊界的一個邊。如果它已經越過了一個邊，可以把它視為一次反射，完成後很有成就感。

7. (10%) 請說明你所遭遇的困難以及解決方法。

[Ans]

讓粒子直接穿過平面比起回彈要容易許多，但為了符合正常的物理現象，當例子碰到邊界時，要如何做出正確的反彈回應和正確的回彈路徑，以及當預設的邊界移動時，粒子是否也會擴大或縮小移動範圍。

解決方法為當邊界移動時，重新生成每個粒子的限制邊界，若在所小範圍的當下粒子位於縮小的邊界外，則重新生成粒子位置，其餘保持原有狀態。

```
for (i = 0; i < 3; i++)
{
    if (particles[n].position[i] >= 0.5 * platform)
    {
        particles[n].velocity[i] = -coef * particles[n].velocity[i];
        particles[n].position[i] = 0.5 * platform - coef * (particles[n].position[i] - 0.5 * platform);
    }
    if (particles[n].position[i] <= -0.5 * platform)
    {
        particles[n].velocity[i] = -coef * particles[n].velocity[i];
        particles[n].position[i] = -0.5 * platform - coef * (particles[n].position[i] + 0.5 * platform);
    }
}
```

8. (10%) 請說明此作品可以改進之處。

[Ans]

這次粒子和邊界都是採用光滑平面的方式，模擬不受材質影響的碰撞情況。但實際上符合物理現象的物體碰撞還包含，當一個粒子在空間中與另一個粒子碰撞需要根據不同的平面材質和各種力，來推導出正確的碰撞路徑及反應。原則上，應該利用動力學和連續力學的知識來推導出所需的演算法。