粒子系统

一、 实验内容

1、粒子系统仿真模拟（可选雨雪、喷泉、火焰等的仿真模拟）；

2、3D 打印实验（设计一个 3D 模型，并利用 3D 打印机打印）

二、 实验环境

Window8 操作系统、VS2013、OpenGL 库、Intel HD Graphics 4000 图形卡

注：运行的话可能需要用 VS2013，并且配置一下 GLAUX 库

三、 实验方案与程序设计

✧ 目标任务

“LOVE”粒子系统的实现

✧ 所需知识

粒子结构体的定义/粒子运动轨迹的控制

✧ 设计方法

采用面向对象的设计思路，定义 Particle 类、Texture 类，通过 Main 函数声明和定义粒子系统对象，并且在显示回调函数中进行 L、O、V、E 粒子的绘制和更新，同时控制粒子速度、方向等交互操作。具体如下：

纹理类：

```
class GLTexture{
public:
    unsigned int texture[1];
    //使用纹理
    void Use();
    void BuildColorTexture(unsigned char r, unsigned char g, unsigned char b);
    //载入纹理
    void LoadBMP(char *name);
};
```

粒子结构体：

```
typedef struct {
    bool  active;                    // 激活状态
    float  life;                     // 生命周期
    float  fade;                     // 衰竭速度
    float  r;                        // 红色 R 值
    float  g;                        // 绿色 G 值
    float  b;                        // 蓝色 B 值
```

```
    float  x;                    // 坐标 X
    float  y;                    // 坐标 Y
    float  z;                    // 坐标 Z
    float  xi;                   // 方向 X 分量
    float  yi;                   // 方向 Y 分量
    float  zi;                   // 方向 Z 分量
    float  xg;                   // 重心 X
    float  yg;                   // 重心 Y
    float  zg;                   // 重心 Z
}particles;
```

粒子系统类：

```
class ParticleSystem{
public:
    ParticleSystem();            // 构造函数
    void Renovate();             // 粒子状态更新
    void InitParticles();        // 粒子初始化
private:
    particles particle[MAX_PARTICLES];
    GLuint   loop;               // 递增循环变量
    GLuint   col;                // 当前颜色选择
    float  slowdown;             // 粒子减速
    float  xspeed;               // 方向 X 速度
    float  yspeed;               // 方向 Y 速度
    float  zspeed;               // 方向 Z 速度
    float  zoom;                 // 缩小比例
};
```

✧ 模块说明

粒子初始化

定义粒子系统对象 particleSystem，通过构造函数初始化各交互参数值，同时通过粒子初始化函数设置粒子结构体参数。

构造函数：

```
ParticleSystem::ParticleSystem ()
{
    slowdown = 0.5f;
    zoom = -40.0f;
    col = 0;
    xspeed = 0;
    yspeed = 0;
    zspeed = 0;
}
```

粒子初始化：

```
void ParticleSystem::InitParticles()
{
  for (loop = 0; loop<MAX_PARTICLES; loop++)
   {
     particle[loop].active = true;                                    //粒子一开始都是活动状态
     particle[loop].life = 1.0f;                                      //所有粒子寿命取整个生命值
     particle[loop].fade = float(rand() % 100) / 1000.0f + 0.003f;          //随机衰竭速度
     particle[loop].r = colors[(loop + 1) / (MAX_PARTICLES / 12)][0];   //取红色
     particle[loop].g = colors[(loop + 1) / (MAX_PARTICLES / 12)][1];   //取绿色
     particle[loop].b = colors[(loop + 1) / (MAX_PARTICLES / 12)][2];   //取蓝色
     particle[loop].x = float(rand() % 20 - 10);                      //粒子初始位置 x 值
     particle[loop].y = float(rand() % 20 - 10);                      //粒子初始位置 y 值
     particle[loop].z = float(rand() % 20 - 10);                      //粒子初始位置 z 值
     particle[loop].xi = float((rand() % 50) - 26.0f)*10.0f;      // X 方向速度取随机的值
     particle[loop].yi = float((rand() % 50) - 25.0f)*10.0f;      // Y 方向速度取随机的值
     particle[loop].zi = float((rand() % 50) - 25.0f)*10.0f;      // Z 方向速度取随机的值
     particle[loop].xg = 0.0f;                                        //将水平拉力（加速度）设为 0
     particle[loop].yg = 0.0f;                                        //设置向下拉力（加速度为 0）
     particle[loop].zg = 18.8f;                                       //设置 Z 向的拉力为 0
   }
}
```

粒子状态更新

通过自定义 Renovate()函数在显示回调函数中更新粒子状态，从而模拟粒子系统的运动轨迹。Renovate 函数负责进行粒子绘制和粒子参数设置，将粒子总数均等地划分为四个集合，每个集合代表一个字母图案。具体如下：

“ I ”粒子：

```
for (loop = MAX_PARTICLES*0.25; loop<MAX_PARTICLES*0.5; loop++)
{
  if (particle[loop].active)                                    //如果粒子处于活动状态
  {
      float x = particle[loop].x;
      float y = particle[loop].y;
      float z1 = particle[loop].z;
      float z = particle[loop].z + zoom;
      // 根据粒子的颜色和衰减值（透明度）绘制粒子
      glColor4f(particle[loop].r, particle[loop].g, particle[loop].b, particle[loop].life);
      glBegin(GL_TRIANGLE_STRIP);
      glTexCoord2d(1, 1); glVertex3f(x + 0.5f, y + 0.5f, z);
      glTexCoord2d(0, 1); glVertex3f(x - 0.5f, y + 0.5f, z);
      glTexCoord2d(1, 0); glVertex3f(x + 0.5f, y - 0.5f, z);
      glTexCoord2d(0, 0); glVertex3f(x - 0.5f, y - 0.5f, z);
      glEnd();
```

```
        // 根据速度计算下一时刻的位置
        particle[loop].x += particle[loop].xi / (slowdown * 1000);
        particle[loop].y += particle[loop].yi / (slowdown * 1000);
        particle[loop].z += particle[loop].zi / (slowdown * 1000);
        // 根据拉力（加速度）计算下一时刻的速度
        particle[loop].xi += particle[loop].xg;
        particle[loop].yi += particle[loop].yg;
        particle[loop].zi += particle[loop].zg;
        particle[loop].life -= particle[loop].fade; '
    }
    if (particle[loop].life<0.0f)                    // 如果粒子生命值为 0（死亡），生成新粒子
    {
        particle[loop].life = 1.0f;
        particle[loop].fade = float(rand() % 100) / 1000.0f + 0.003f;
        particle[loop].x = -20;                                    //粒子初始位置 x 值
        particle[loop].y = float(rand() % 25 - 10);                //粒子初始位置 y 值
        particle[loop].z = 0;                                      //粒子初始位置 z 值
        particle[loop].xi = xspeed + float((rand() % 60) - 32.0f);
        particle[loop].yi = yspeed + float((rand() % 60) - 30.0f);
        particle[loop].zi = zspeed + float((rand() % 60) - 30.0f);
        particle[loop].zi = float((rand() % 60) - 30.0f);
        particle[loop].r = colors[col][0];
        particle[loop].g = colors[col][1];
        particle[loop].b = colors[col][2];
        col++;
        if (col>110)     col = 0;
    }
}
```

“ O ”粒子：

```
 void ParticleSystem::Renovate()
 {
for (loop = MAX_PARTICLES*0.5; loop<MAX_PARTICLES*0.75; loop++)
   // Loop Through All The Particles
{
   if (particle[loop].active)                                  //如果粒子处于活动状态
   {
        float x = particle[loop].x;
        float y = particle[loop].y;
        float z1 = particle[loop].z;
        float z = particle[loop].z + zoom;
        // 根据粒子的颜色和衰减值（透明度）绘制粒子
        glColor4f(particle[loop].r, particle[loop].g, particle[loop].b, particle[loop].life);
        glBegin(GL_TRIANGLE_STRIP);
```

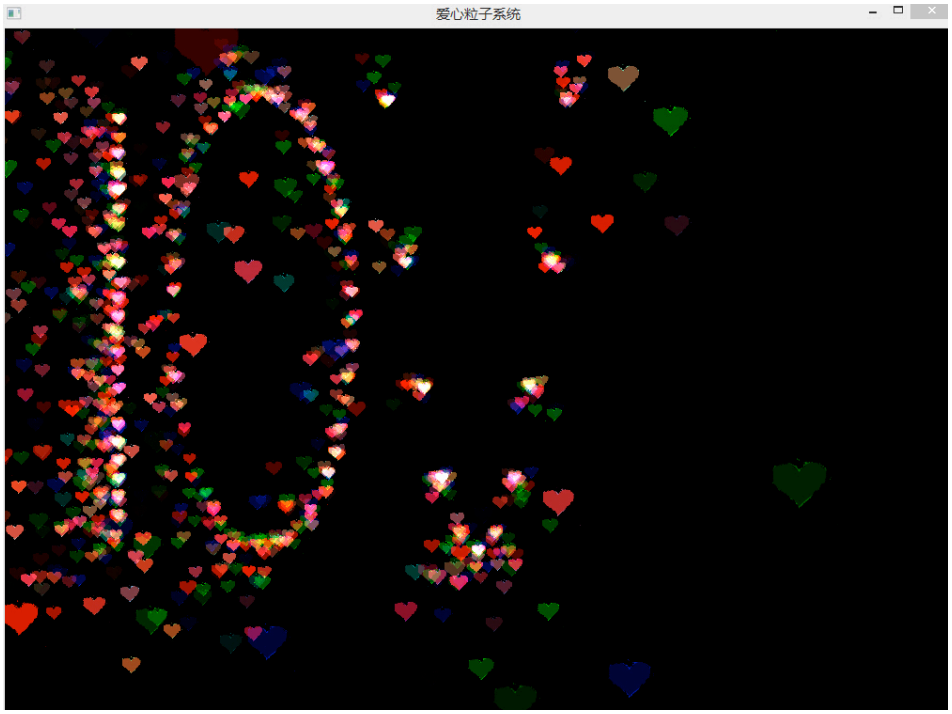
```
        glTexCoord2d(1, 1); glVertex3f(x + 0.5f, y + 0.5f, z);
        glTexCoord2d(0, 1); glVertex3f(x - 0.5f, y + 0.5f, z);
        glTexCoord2d(1, 0); glVertex3f(x + 0.5f, y - 0.5f, z);
        glTexCoord2d(0, 0); glVertex3f(x - 0.5f, y - 0.5f, z);
        glEnd();
        // 根据速度计算下一时刻的位置
        particle[loop].x += particle[loop].xi / (slowdown * 1000);
        particle[loop].y += particle[loop].yi / (slowdown * 1000);
        particle[loop].z += particle[loop].zi / (slowdown * 1000);
        // 根据拉力（加速度）计算下一时刻的速度
        particle[loop].xi += particle[loop].xg;
        particle[loop].yi += particle[loop].yg;
        particle[loop].zi += particle[loop].zg;
        particle[loop].life -= particle[loop].fade; '
    }
    if (particle[loop].life<0.0f)              // 如果粒子生命值为 0（死亡），生成新粒子
    {
        particle[loop].life = 1.0f;
        particle[loop].fade = float(rand() % 100) / 1000.0f + 0.003f;
        int m = 50;
        float angle = loop * 2 * PI / m;
        particle[loop].x = r1*cos(angle) - 12;                    //粒子初始位置 x 值
        particle[loop].y = 2.5*r1*sin(angle) + 2.8;               //粒子初始位置 y 值
        particle[loop].z = 0;                                     //粒子初始位置 z 值
        particle[loop].xi = xspeed + float((rand() % 60) - 32.0f);
        particle[loop].yi = yspeed + float((rand() % 60) - 30.0f);
        particle[loop].zi = zspeed + float((rand() % 60) - 30.0f);
        particle[loop].zi = float((rand() % 60) - 30.0f);
        particle[loop].r = colors[col][0];
        particle[loop].g = colors[col][1];
        particle[loop].b = colors[col][2];
        col++;
        if (col>110)    col = 0;
    }
  }
}
```

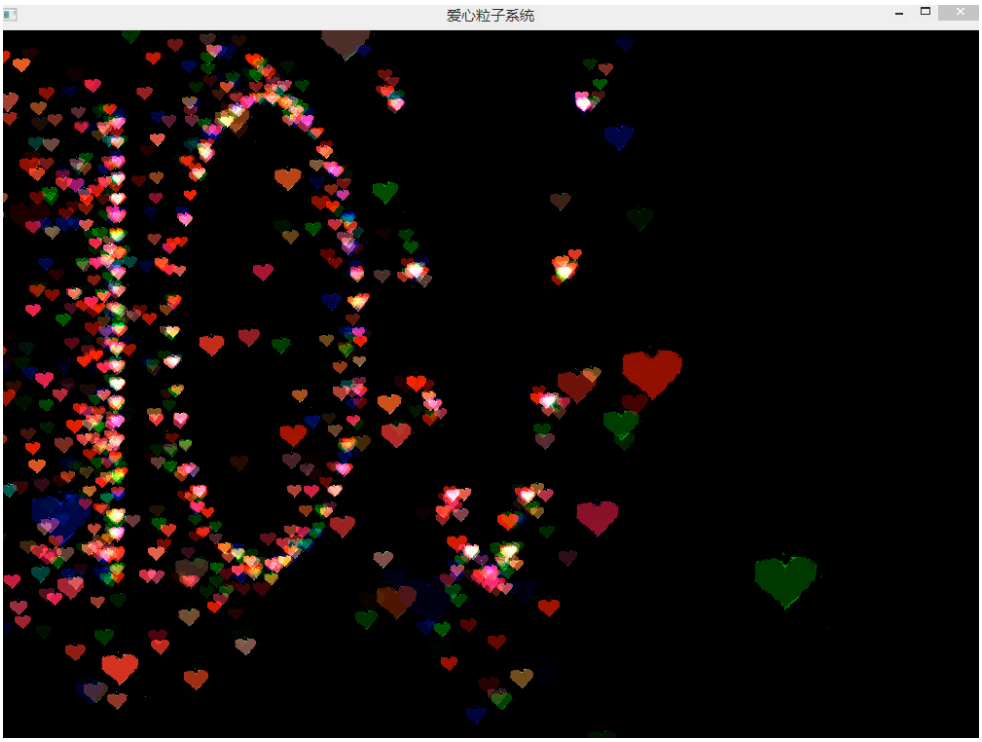
“V ”粒子：

```
 for (loop = 0; loop<MAX_PARTICLES*0.25; loop++)
{
   if (particle[loop].active)                                  //如果粒子处于活动状态
   {
       float x = particle[loop].x;
       float y = particle[loop].y;
```

```
        float z1 = particle[loop].z;
        float z = particle[loop].z + zoom;
        //根据粒子的颜色和衰减值（透明度）绘制粒子
        glColor4f(particle[loop].r, particle[loop].g, particle[loop].b, particle[loop].life);
        glBegin(GL_TRIANGLE_STRIP);
        glTexCoord2d(1, 1); glVertex3f(x + 0.5f, y + 0.5f, z);
        glTexCoord2d(0, 1); glVertex3f(x - 0.5f, y + 0.5f, z);
        glTexCoord2d(1, 0); glVertex3f(x + 0.5f, y - 0.5f, z);
        glTexCoord2d(0, 0); glVertex3f(x - 0.5f, y - 0.5f, z);
        glEnd();
        //根据速度计算下一时刻的位置
        particle[loop].x += particle[loop].xi / (slowdown * 1000);
        particle[loop].y += particle[loop].yi / (slowdown * 1000);
        particle[loop].z += particle[loop].zi / (slowdown * 1000);
        //根据拉力（加速度）计算下一时刻的速度
        particle[loop].xi += particle[loop].xg;
        particle[loop].yi += particle[loop].yg;
        particle[loop].zi += particle[loop].zg;
        particle[loop].life -= particle[loop].fade;
    }
    if (particle[loop].life<0.0f)                    // 如果粒子生命值为 0（死亡），生成新粒子
    {
        particle[loop].life = 1.0f;
        particle[loop].fade = float(rand() % 100) / 1000.0f + 0.003f;
        particle[loop].x = float(rand() % 20 - 10);                    //粒子初始位置 x 值
        particle[loop].y = particle[loop].x*particle[loop].x - 10;     //粒子初始位置 y 值
        particle[loop].z = 0;                                          //粒子初始位置 z 值
        particle[loop].xi = xspeed + float((rand() % 60) - 32.0f);
        particle[loop].yi = yspeed + float((rand() % 60) - 30.0f);
        particle[loop].zi = zspeed + float((rand() % 60) - 30.0f);
        particle[loop].zi = float((rand() % 60) - 30.0f);
        particle[loop].r = colors[col][0];
        particle[loop].g = colors[col][1];
        particle[loop].b = colors[col][2];
        col++;
        if (col>110)    col = 0;
    }
}
```

四、 实验结果与分析（包括功能说明、操作说明）





五、 实验心得（自我评价）与建议

- ✧ 在这次实验中，通过查阅大量资料了解了粒子系统的基本原理。在大二时因为课外兴趣，学习过粒子群算法，因此 CPU 版本的粒子系统实现起来碰到的问题并不多。
- ✧ 在这次实验中，碰到的主要问题是采用显示回调 display 刷新粒子的时候，刷新速度太快，导致两轮刷新间隔太短，很多粒子重叠在一起，效果不佳。因此我试探性地在显示回调函数中加入 for(int i=0;i<300;i++) printf(“%d”,i)，没想到这样可以扩大刷新间隔，实际效果也就好很多了。
- ✧ 在这次实验中，实现了 CPU 版本的粒子系统后，原本打算实现 GPU 版本的粒子系统，但是 google 后发现，对于粒子上一次的更新后得到的数据，需要通过特殊的顶点缓冲区回调 feedback 才能获取，用于新一轮的粒子更新。同时需要控制多个着色器，实现起来较为复杂，大三上的期末又是课程最多的时候，时间不够充裕。因此我在寒假的时候会将实现好的 GPU 版本发到老师邮箱，敬请见谅。