一、 实验器材（设备、元器件）：

个人电脑一台(MacOS 10.15. Xcode 11.4.1)

SOIL 外部链接库(需自行下载链接)

Framework： Opengl, GLUT

二、 实验名称： 粒子系统

三、 实验目的：

1) 掌握基本粒子系统的实现方法；
2) 掌握键盘事件响应的方法；
3) 能够在基本粒子系统基础上设计新的粒子动画。

四、 实验原理：

(一) 粒子系统框架

粒子系统基本实现步骤大致分为三步：

1. 生成（发射）粒子
2. 模拟粒子
    a) 模拟粒子运动
    b) 模拟粒子老化
    c) 模拟粒子与环境的交互（如：碰撞）
3. 渲染粒子

生成（发射）粒子通常设置粒子初始属性，包括位置、速度、加速度、生命周期、颜色、大小等信息。发射粒子的数量以及分布状态等等信息，这些通常通过可控的随机过程来处理。

接下来需要更新粒子状态。在此阶段，检查每个粒子是否已经超出了生命周期，一旦超出就将这些粒子剔出模拟过程，否则就根据粒子受力状况，更新粒子的加速度、速度以及位置信息。另外，根据渲染相关的属性更新粒子的颜色、大小等。在考虑粒子受力状态时，除了重力、摩擦力这些常见作用力外，经常需要检查与特殊三维物体的碰撞以使粒子从障碍物弹回。

在更新完成之后，需要渲染粒子，根据粒子颜色、形状、大小绘制粒子，生成一帧动画。

(二) 基本粒子系统实现

1. 定义粒子结构

首先需要定义粒子的属性，我们定义一个结构体 particles 来描述粒子属性。

- active – 指示该粒子是否消亡；
- life – 粒子的生命值；
- fade – 粒子衰老速度；

2. 创建粒子并初始化

接下来创建粒子并初始化:

1) 定义数组 particles 存储粒子信息，数组类型为定义的结构类型 particle:

2) 在函数 InitGL 中初始化粒子信息
   a) fade 粒子衰老速度范围为 0.003 – 0.102;
   b) 粒子初始颜色为红色；
   c) 粒子初始速度根据根据球坐标系设置，速度大小范围 0-99;
   d) 粒子受力为重力；
3) 球坐标系

三维空间中一点 P 的球坐标系坐标为（ρ, φ, θ），如图 3 所示:

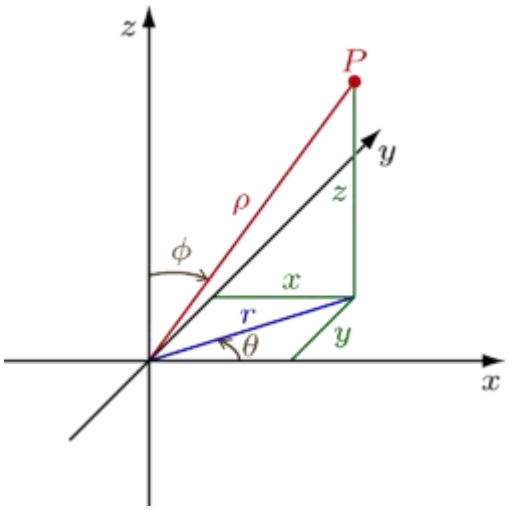


图 3

- ρ是距离球心的距离;
- φ是距离 z 轴的角度（称作余纬度或顶角，角度从 0 到 180°）；
- θ是距离 x 轴的角度（与极坐标中一样）；
- 通过以下公式，可以从直角坐标变换为球坐标:

$$x=\rho\sin\phi\cos\theta$$

$$y=\rho\sin\phi\sin\theta$$

$$z=\rho\cos\phi$$

3. 粒子状态更新

粒子状态更新在 renderScene 函数中：

1) 绘制粒子

采用画点方式绘制粒子，粒子大小设置：

```
glPointSize(4.0f);
```

我们利用粒子的生命值来控制粒子的透明度：

```
glColor4f(particles[i].r, particles[i].g, particles[i].b, particles[i].life);
```

2) 粒子状态更新

粒子位置更新时，我们将速度除以了（slowdown*1000），其中slowndown是个全局变量，可用来控制粒子速度大小：

```
float slowdown=0.25f;                    // Slow Down Particles
```

3) 粒子消亡

粒子消亡后，重新生成粒子。只需要给该粒子重新赋予生命值、衰老速度和运动速度。

4. 其他程序模块

- **主函数**
- **计时器**
- **初始化函数**
- **窗口响应函数**

（三） 添加纹理

1. 渲染原理

为了让粒子渲染效果更好，我们给粒子贴上纹理。

首先绘制正方形，然后将纹理贴在正方形上。在绘制正方形时，我们不采用传统方法，因为要绘制大量正方形速度相对较慢。OpenGL 在绘制三角形时，速度是很快的，所以我们采用绘制两个三角形的方式绘制正方形：

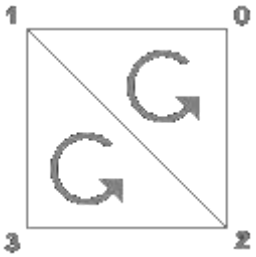


图 10

GL_TRIANGLE_STRIP 指定绘制三角形方式。如图 10 所示，通过指定顶点：

v0, v1, v2, v3

绘制 2 个三角形，分别由 v0, v1, v2 和 v1, v2, v3 构成。

在指定顶点序列同时，建立起顶点坐标和纹理坐标的对应关系。

2. 修改函数

1) 载入纹理

载入纹理，需要添加函数LoadGLTextures。首先定义全局变量texture：

3. 修改初始化函数

四、实验结果与分析（含重要数据结果分析或核心代码流程分析）

1) 实现基本粒子系统；

1. 定义粒子结构

```
typedef struct                          // Create a Structure for Particle
{
    float    life;                      // Particle Life
    float   init_life;
    float    speed_aging;              // Aging speed

    float    r;                         // Red value
    float    g;                         // Green value
    float    b;                         // Blue value

    float    x;                         // X position
    float    y;                         // Y position
    float    z;                         // Z position

    float    v_x;                      // X velocity
    float    v_y;                      // Y velocity
    float    v_z;                      // Z velocity

    float    a_x;                      // X acceleration
    float    a_y;                      // Y acceleration
    float    a_z;                      // Z acceleration
}
particle;                               // Particles structure
```

2. 创建粒子并初始化

```
particle particles[MAX_PARTICLES];


int InitPaticleSystem(void)
{
    float theta, phi, rho;

    for (int i = 0; i < MAX_PARTICLES;i++)                          // Initials all particles
    {
        particles[i].init_life = LIFE + rand() % 10 / 10.0;            // Give All The Particles Full Life
        particles[i].life = particles[i].init_life;
        particles[i].speed_aging = TIME;

        particles[i].r = 0.0f;                                      // Set color for particle
        particles[i].g = 1.0f;
        particles[i].b = 1.0f;

        particles[i].x = 0.0f;                                     // Set position for particle
        particles[i].y = 50.0f;
        particles[i].z = 0.0f;

        theta = (rand() % 90 + 45) * PI / 180;
        phi = 90 * PI / 180;
        rho = rand() % RHO;
        particles[i].v_x = float(sinf(phi) * cosf(theta) * rho);  // Set X Axis Speed And Direction for particle
        particles[i].v_y = float(sinf(phi) * sin(theta) * rho);      // Set Y Axis Speed And Direction for particle
        particles[i].v_z = float(cosf(phi) * rho);                // Set Z Axis Speed And Direction for particle

        particles[i].a_x = 0.0f;                            // Set X Axis acceleration
        particles[i].a_y = -30.0f;                            // Set Y Axis acceleration
        particles[i].a_z = 0.0f;                            // Set Z Axis acceleration
    }

    return true;
}
```

```
int SetupRC(void)
{
    if(!LoadGLTextures()){
        printf("Load texture fail");
        return false;

    }
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, texture[0]);
    return true;
}
```

3. 粒子状态更新

```
void Update(){//在glTime中，每隔一段时间调用一下
    for (int i = 0;i < MAX_PARTICLES; i++)                    // All The Particles
    {
        particles[i].x += particles[i].v_x * TIME;         // update position of particle
        particles[i].y += particles[i].v_y * TIME;
        particles[i].z += particles[i].v_z * TIME;

        particles[i].v_x += particles[i].a_x * TIME;            // update velocity
        particles[i].v_y += particles[i].a_y * TIME;
        particles[i].v_z += particles[i].a_z * TIME;

        particles[i].life -= particles[i].speed_aging;        // reduce particles life

        float theta, phi, rho;
        if (particles[i].life < 0.0f)                    // if particle has reached end of life
        {
            particles[i].init_life = LIFE + rand()%10/10.0;        // Give it new life
            particles[i].life = particles[i].init_life;

            particles[i].x = 0.0f;                        // Initialize position
            particles[i].y = 50.0f;
            particles[i].z = 0.0f;

            theta = (rand() % 360) * PI/180;
            phi = 90 * PI / 180;
            rho = rand() % RHO;
            particles[i].v_x = float(sinf(phi) * cosf(theta) * rho);    // Initialize velocity
            particles[i].v_y = float(sinf(phi) * sin(theta) * rho);
            particles[i].v_z = float(cosf(phi) * rho);
        }
    }
}

void RenderScene(void) {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);          // Clear screen and depth buffer
    for (int i = 0;i < MAX_PARTICLES; i++)                             // All particles
    {
        float x = particles[i].x;                                 // Position of particle
        float y = particles[i].y;
        float z = particles[i].z;


        // Draw particle using RGB values, alpha value based on it's life
        glColor4f(particles[i].r,particles[i].g,particles[i].b,particles[i].life);

        glPointSize(4.0f);
        glBegin(GL_TRIANGLE_STRIP);             // Build Quad From A Triangle Strip
        glTexCoord2d(1,1);
        glVertex3f(x+0.5f,y+0.5f,z); // Top Right
        glTexCoord2d(0,1); glVertex3f(x-0.5f,y+0.5f,z); // Top Left
        glTexCoord2d(1,0); glVertex3f(x+0.5f,y-0.5f,z); // Bottom Right
        glTexCoord2d(0,0); glVertex3f(x-0.5f,y-0.5f,z); // Bottom Left
        glEnd();
    }

    glutSwapBuffers();
}
```

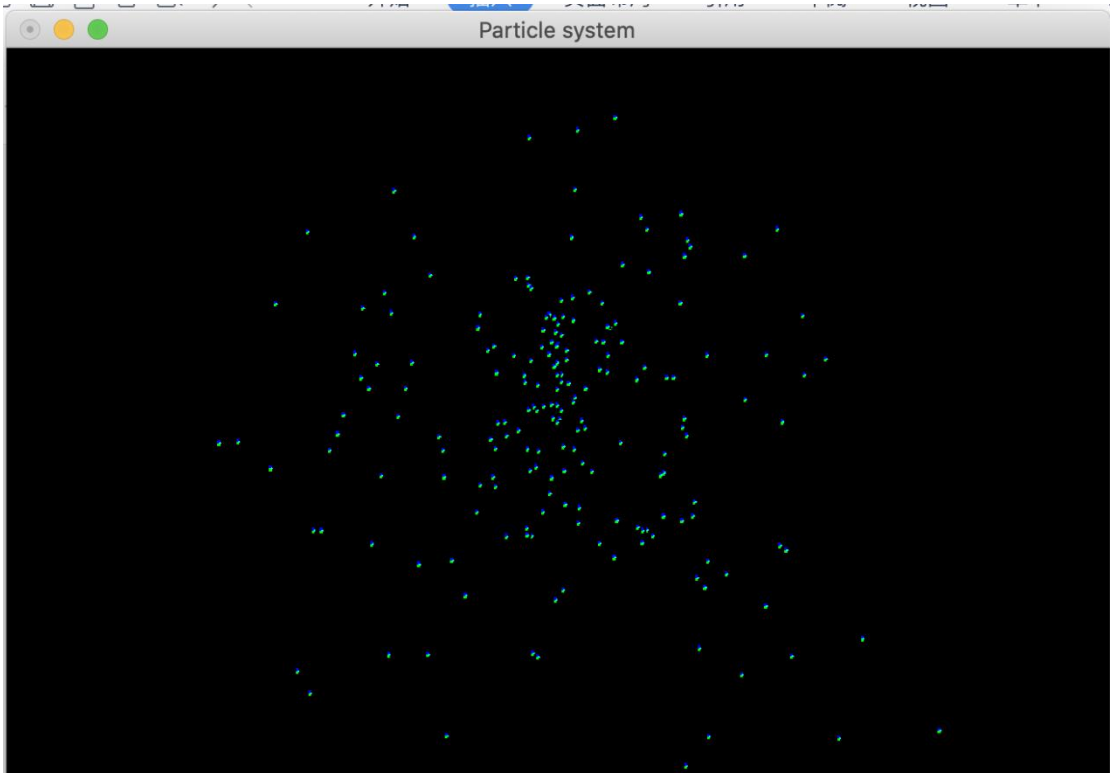
4. 时间控制器

```
void TimerFunction(int value){
    Update();
    glutPostRedisplay();
    glutTimerFunc(10,TimerFunction,1);// 10ms后执行TimerFuncion
}
```

5. 主函数

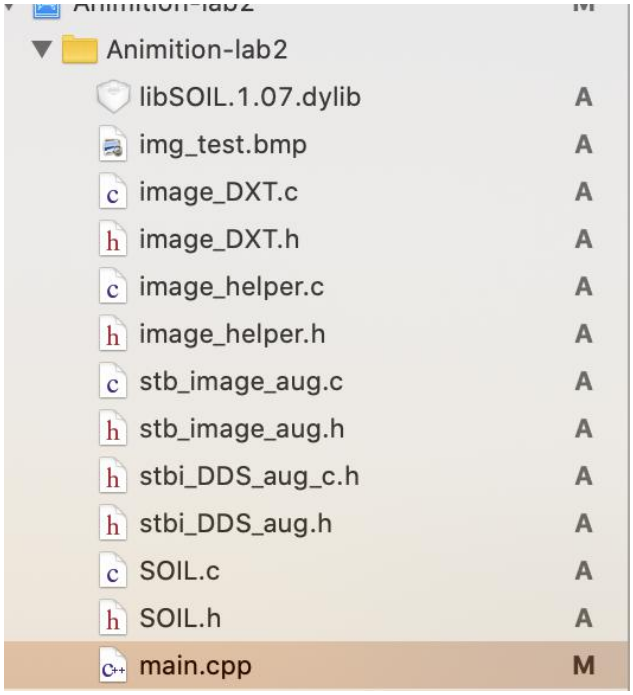
```
int main(int argc, char **argv) {

    // init GLUT and create window
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);//双缓冲区，应用于动画
    glutInitWindowPosition(100,100);
    glutInitWindowSize(640,640);
    glutCreateWindow("Particle system");

    // register callbacks
    glutDisplayFunc(RenderScene);// 绑定回调函数，传入的函数是在窗体绘制的时候调用
    glutReshapeFunc(ChangeSize);//窗口的大小
    glutTimerFunc(10, TimerFunction, 1);//每隔一段时间绘制一帧，产生连续的动画

    // Setup the rendering state

    SetupRC();//绘图初始化 设置清除色，由glClear(GL_COLOR_BUFFER_BIT)调用-在renderScene中；
    InitPaticleSystem();

    // enter GLUT event processing loop
    glutMainLoop();//处理队列，死循环的方式等待时间

    return 0;
}
```

效果：



(2) Z 为基本粒子系统添加纹理贴图；

- 使用 BOIL 库进行贴图读取：链接静态库即动态库



- 在初始化函数中设定素材

```
int SetupRC(void)
{
    if(!LoadGLTextures()){
        printf("Load texture fail");
        return false;

    }
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, texture[0]);
    return true;
}
```

- 读取素材

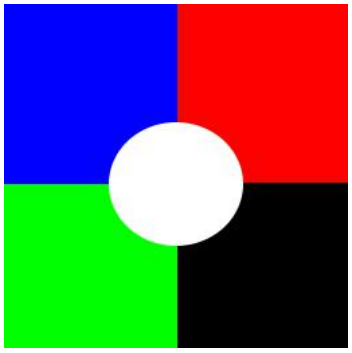
```
int LoadGLTextures(){
    char *file = "/Users/una/Library/Mobile Documents/com~apple~CloudDocs/大二下/动画/project/Animition-lab2/Animition-lab2/img_test.bmp";
    std::cout<<file<<std::endl;
    if (file == NULL) {
        printf("123");
    }
    texture[0]=SOIL_load_OGL_texture(file,
                                     SOIL_LOAD_AUTO,
                                     SOIL_CREATE_NEW_ID,
                                     SOIL_FLAG_INVERT_Y
                                     );
    if(texture[0] == 0 )
        return false;
    glBindTexture(GL_TEXTURE_2D, texture[0]);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    return true;
}
```

- 在渲染函数中贴图

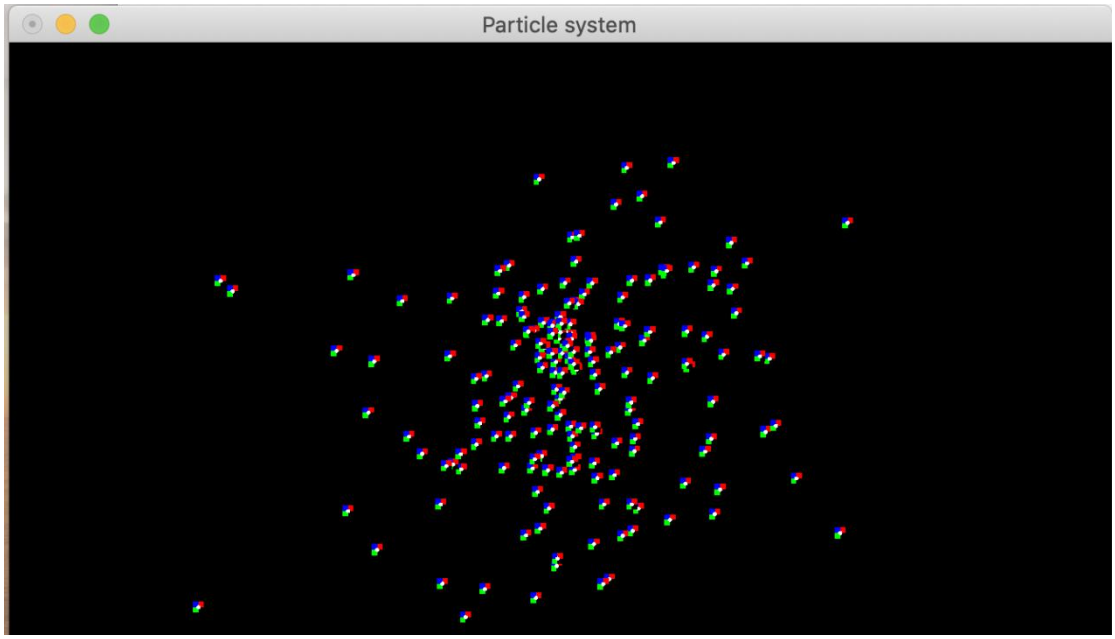
```
#define SIZE            1.0f
```

```
void RenderScene(void) {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);          // Clear screen and depth buffer //无动态模糊时
    //glClear(GL_DEPTH_BUFFER_BIT );    //动态模糊
    for (int i = 0;i < MAX_PARTICLES; i++)                              // All particles
    {
        float x = particles[i].x;                                 // Position of particle
        float y = particles[i].y;
        float z = particles[i].z;


        // Draw particle using RGB values, alpha value based on it's life
        //glColor4f(particles[i].r,particles[i].g,particles[i].b,particles[i].life);

        glPointSize(4.0f);
        glBegin(GL_TRIANGLE_STRIP);            // Build Quad From A Triangle Strip
        glTexCoord2d(1,1);glVertex3f(x+SIZE,y+SIZE,z); // Top Right
        glTexCoord2d(0,1); glVertex3f(x-SIZE,y+SIZE,z); // Top Left
        glTexCoord2d(1,0); glVertex3f(x+SIZE,y-SIZE,z); // Bottom Right
        glTexCoord2d(0,0); glVertex3f(x-SIZE,y-SIZE,z); // Bottom Left
        glEnd();
    }

    glutSwapBuffers();
}
```

- 贴图：



- 效果



3) 在基本粒子系统基础上扩展，如生成火焰、烟花、瀑布等效果，

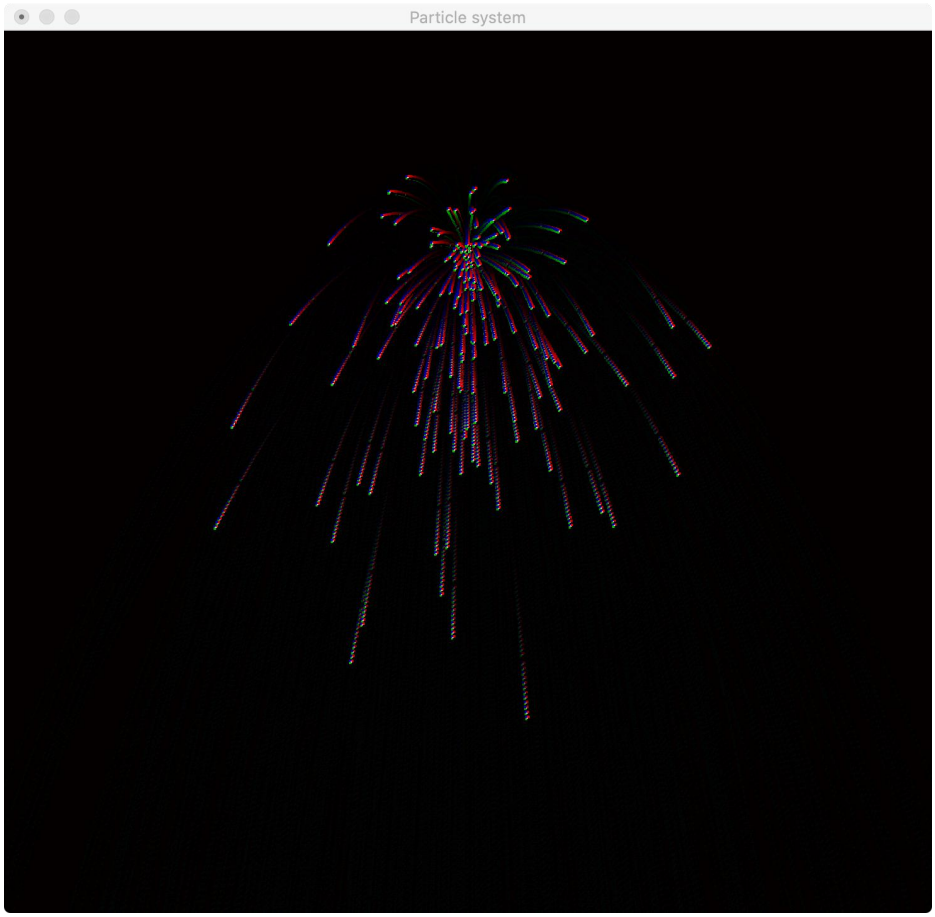
并与环境发生交互，加入碰撞检测。

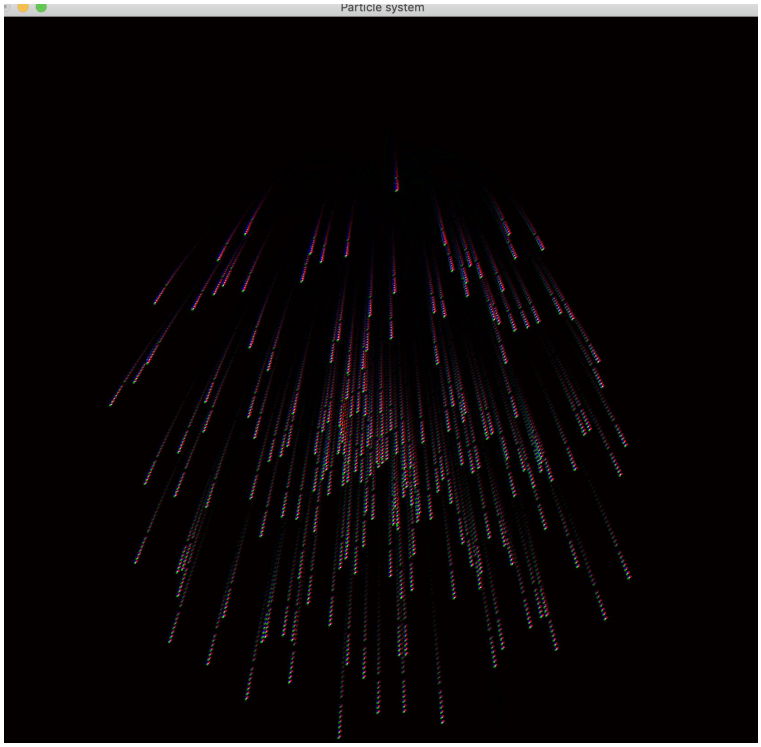
- 动态模糊

在不清空缓存区的基础上，每帧绘制一个透明度比较低的矩形，形成动态模糊效果。

```
void RenderScene(void) {
    //glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);          // Clear screen and depth buffer //无动态模糊时
    /*glClear(GL_DEPTH_BUFFER_BIT);    //动态模糊
    glColor4f(0.0f, 0.0f,0.0f, 0.1f);
    glRectf(-windowWidth, -windowHeight, windowWidth, windowHeight);*/

    //glEnable(GL_TEXTURE_2D);
    //glBindTexture(GL_TEXTURE_2D, texture[0]);

    for (int i = 0;i < MAX_PARTICLES; i++)                              // All particles
    {
        float x = particles[i].x;                                  // Position of particle
        float y = particles[i].y;
        float z = particles[i].z;
        glColor4f(particles[i].r,particles[i].g,particles[i].b,particles[i].life);
        glEnable(GL_TEXTURE_2D);
        glBindTexture(GL_TEXTURE_2D, texture[0]);
        // Draw particle using RGB values, alpha value based on it's life
        //glColor4f(particles[i].r,particles[i].g,particles[i].b,particles[i].life);

        glPointSize(4.0f);
        glBegin(GL_TRIANGLE_STRIP);            // Build Quad From A Triangle Strip
        glTexCoord2d(1,1);glVertex3f(x+SIZE,y+SIZE,z); // Top Right
        glTexCoord2d(0,1); glVertex3f(x-SIZE,y+SIZE,z); // Top Left
        glTexCoord2d(1,0); glVertex3f(x+SIZE,y-SIZE,z); // Bottom Right
        glTexCoord2d(0,0); glVertex3f(x-SIZE,y-SIZE,z); // Bottom Left
        glEnd();
    }

    glClear(GL_DEPTH_BUFFER_BIT);    //动态模糊
       glColor4f(0.0f, 0.0f,0.0f, 0.1f);
       glRectf(-windowWidth, -windowHeight, windowWidth, windowHeight);


    glutSwapBuffers();
}
```

烟花效果:





- 碰撞检测

在 update()中加入碰撞检测

```
206 void Update(){//在glTime中，每隔一段时间调用一下
207     for (int i = 0;i < MAX_PARTICLES; i++)                    // All The Particles
208     {   if(particles[i].y - SIZE < -30.0 && particles[i].x<50.0f &&particles[i].x>-50.0f || particles[i].y-SIZE > 80.0f && particles[i].x<5.0f &&particles[i].x>-5.0f)  ⚠ '&&' within '||'
209         particles[i].v_y = -particles[i].v_y;
210
```