《可视化程序设计》结课程序设计报告

数字虚空

Digital Void

专业名称	物联网工程
班级学号	4111613
学生姓名	唐启鸣
指导教师	鲁宁
设计时间	2014.3.25~2014.5.5

一、　需求分析

在信息技术高度发展的今天，计算机模拟已经成为当下科学研究中的一个重要环节。从数学推演、物理计算、生化模拟到经融投资、气象预测，计算机模拟无处不在。纯形式化的逻辑理论是高度抽象的，而现实的各种问题要求又是高度具象的，所以计算机模拟就能在这两者之中起到一个良好的过渡和桥接作用。我们以某个抽象的规律或者形式理论为基础，建立一个计算机的模拟程序，来推演“假定以此为基础的事实情况”，然后再将计算机的模拟结果与真正的现实情况相对比。以此来进一步加深我们对现实问题的认识，对未来的掌握。可以看出，计算机模拟在科学研究中扮演的重要角色。

本程序命名为数字虚空（Digital Void），系一运动模拟程序，其总目标为实现对现实世界中复数个实体运动的模拟演示。现在实现了类星体运动的模拟（太阳系运动），其具体的功能有：

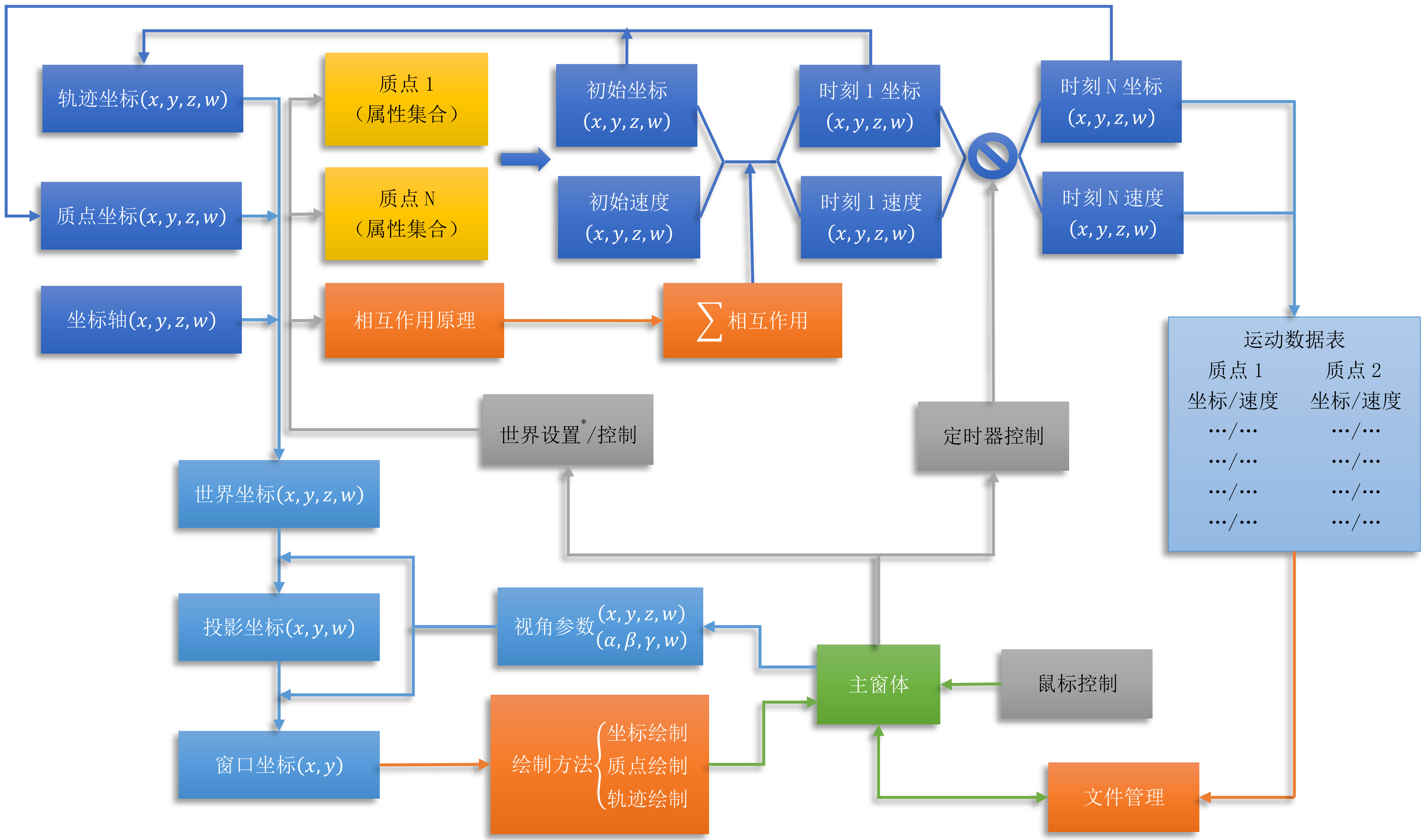
——三维坐标的投影及图形绘制

——实体运动的模拟计算

——计算数据的存储管理

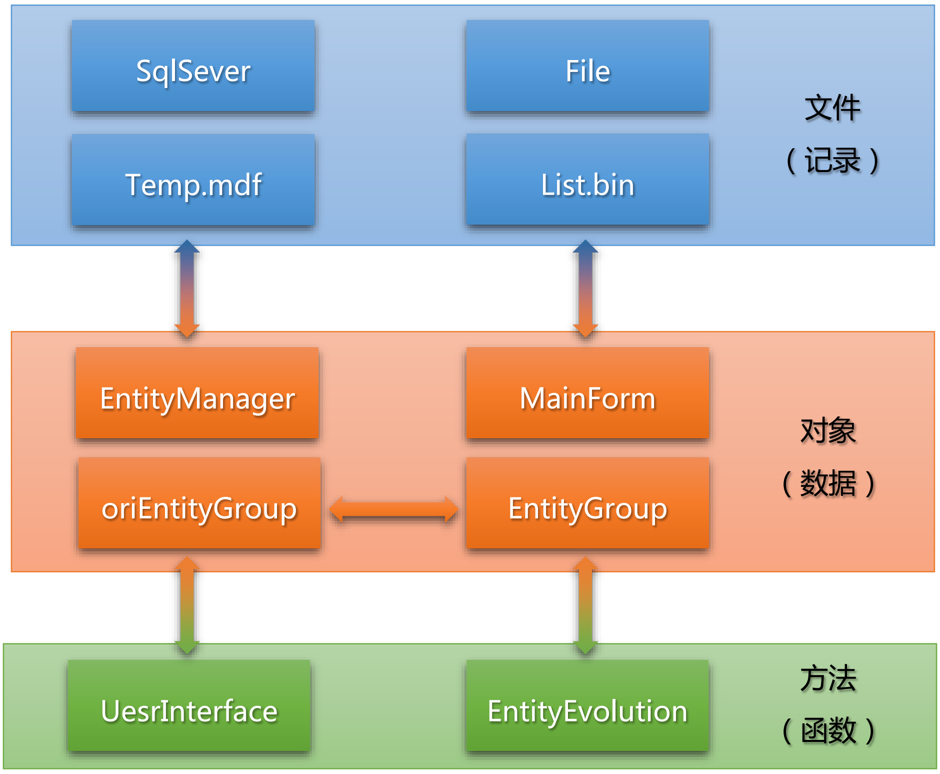
二、　设计方案

整体的程序结构如图所示：

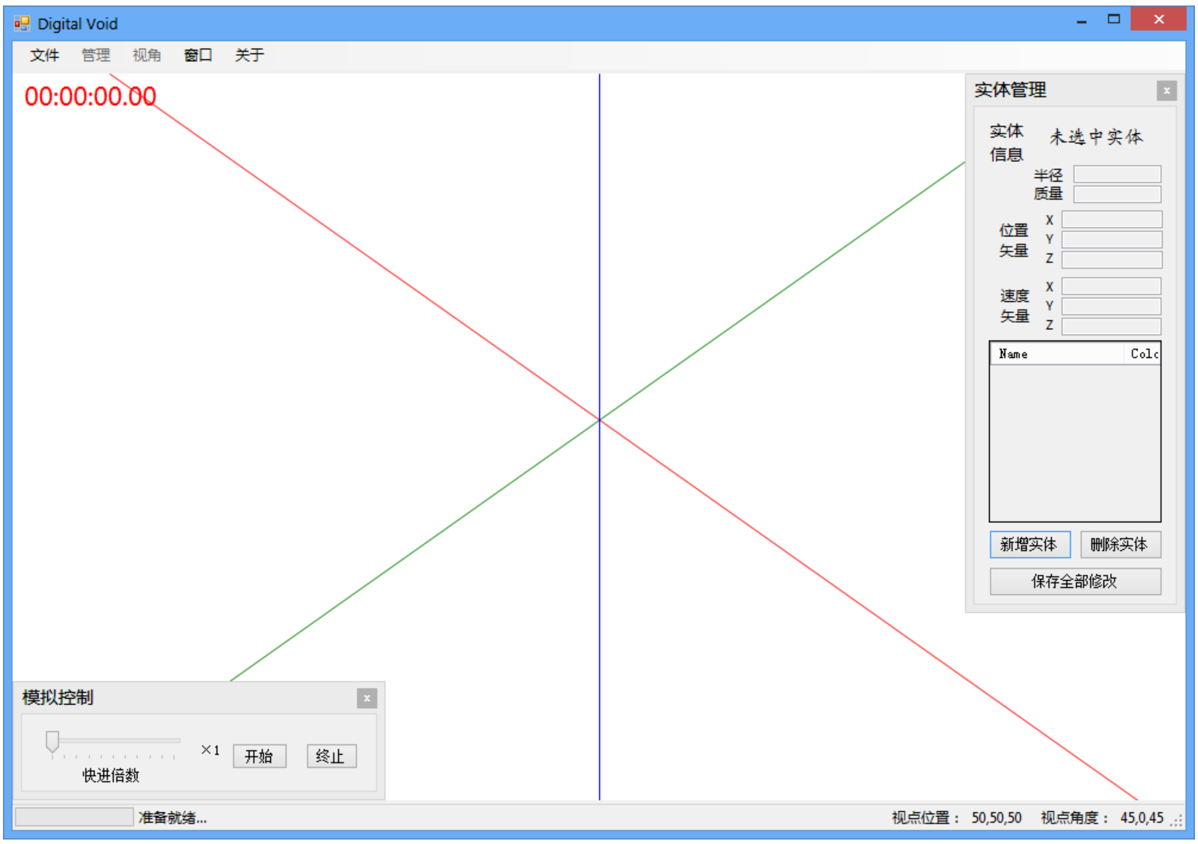

轨迹坐标(x,y,z,w)
质点坐标(x,y,z,w)
坐标轴(x,y,z,w)
质点 1
（属性集合）
质点 N
（属性集合）
相互作用原理
初始坐标
(x,y,z,w)
初始速度
(x,y,z,w)
∑相互作用
时刻 1 坐标
(x,y,z,w)
时刻 1 速度
(x,y,z,w)
时刻 N 坐标
(x,y,z,w)
时刻 N 速度
(x,y,z,w)
运动数据表
质点 1 质点 2
坐标/速度 坐标/速度
…/… …/…
…/… …/…
…/… …/…
…/… …/…
世界设置*/控制
定时器控制
世界坐标(x,y,z,w)
投影坐标(x,y,w)
窗口坐标(x,y)
视角参数 (x,y,z,w) (α,β,γ,w)
绘制方法 坐标绘制 质点绘制 轨迹绘制
主窗体
鼠标控制
文件管理

其中关于相互作用以及三维投影在代码实现和附录中有详细的叙述。

数据流向的设计结构则如下图所示：



整体 UI 的设计：



三、 代码实现

整体代码结构：

方法类：	窗口(Form)类：
Vector3（齐次坐标、三维投影）	Main Form（主窗体）
Martix3（矩阵的乘法）	Play Control（模拟控制窗体）
Entity（实体）	Entity Manager（实体管理窗体）
Entity Evolution（实体演化）	About（关于对话框）
Draw Method（绘图方法）	

三维投影部分的实现代码：

```
/// <summary>
/// 该方法将一个世界坐标矢量转换为一个窗口坐标
/// </summary>
/// <param name="WorldVector">世界坐标矢量</param>
/// <param name="MainForm">窗体实例对象</param>
/// <returns>窗口坐标</returns>
        public static Vector3 Transform3D_View(Vector3 WorldVector, MainForm MainForm)
        {
            Vector3 ViewVector = new Vector3();
            Vector3 ShowVector = new Vector3();
            Vector3 ViewAngle = MainForm.ViewAngle;
            float PanelHeight = MainForm.PlotPanel.Height;
            float PanelWidth = MainForm.PlotPanel.Width;
            Vector3 Distance=new Vector3();
            Distance = WorldVector - MainForm.ViewPosition;
            ViewVector = Transform3D_2D(ViewAngle,Distance);
            if (Math.Abs(ViewVector.X) < (PanelWidth / 2) && Math.Abs(ViewVector.Y) < (PanelHeight /
2) && ViewVector.Z <= 0)
            {
                ShowVector.X = ViewVector.X  + (PanelWidth / 2);
                ShowVector.Y = ViewVector.Y  + (PanelHeight / 2);
            }
            else
            {
                ShowVector.X = ViewVector.X + (PanelWidth / 2);
                ShowVector.Y = ViewVector.Y + (PanelHeight / 2);
                ShowVector.W = 0;
            }
            return ShowVector;
        }
```

运动模拟部分的实现代码：

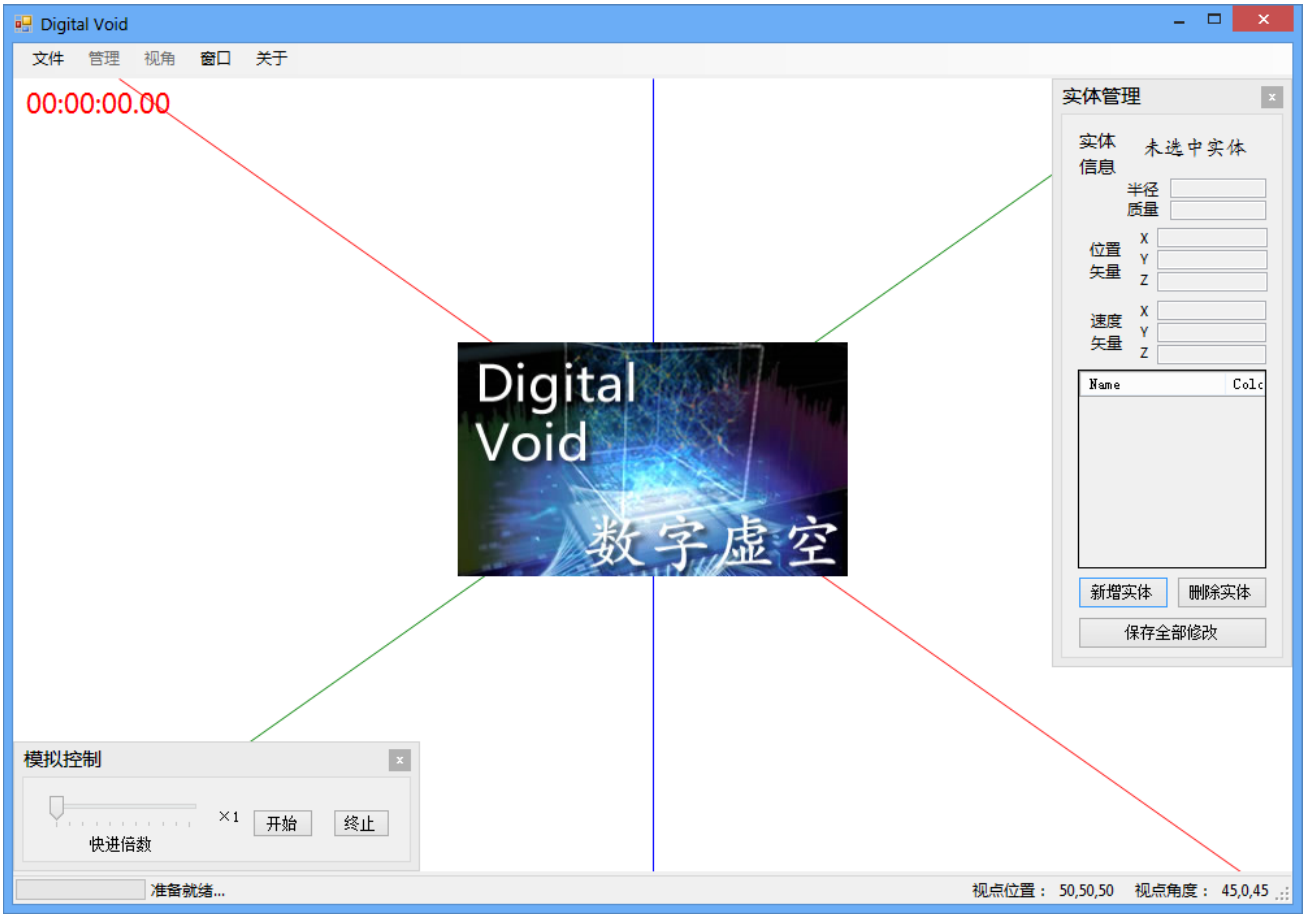
```
/// <summary>
/// 对整个实体集合进行演化模拟
/// </summary>
/// <param name="EntityGroup">实体数组</param>
        public void AtomicEvolution(Entity[] EntityGroup)
        {
            int Sum = EntityGroup.Length;
            Vector3[,] Force = new Vector3[Sum, Sum];
            Vector3[] TotalForce = new Vector3[Sum];
            Vector3[] Acceleration = new Vector3[Sum];
            for(int i=0;i<Sum;i++)//对于第i个实体来说
            {
                TotalForce[i] = new Vector3();
                Acceleration[i] = new Vector3();
                for(int j=0;j<Sum;j++)//第j个实体对于它（第i个实体）的作用力
                {
                    if (j != i)
                    {
                        Force[i, j] = WorldInteraction(EntityGroup[i], EntityGroup[j]);
                    }
                    else
                        Force[i,j] = new Vector3(0,0,0,1);
                    TotalForce[i] += Force[i, j];//其余所有实体对于它（第i个实体）的合力矢量
                }
                Acceleration[i].X = TotalForce[i].X / EntityGroup[i].Mass;
                Acceleration[i].Y = TotalForce[i].Y / EntityGroup[i].Mass;
                Acceleration[i].Z = TotalForce[i].Z / EntityGroup[i].Mass; //得加速度矢量
            }
            for(int i=0;i<Sum;i++)
            {
                EntityGroup[i].Position.X += EntityGroup[i].Velocity.X * DeltaTime + Acceleration[i].X
* DeltaTime * DeltaTime / 2;
                EntityGroup[i].Position.Y += EntityGroup[i].Velocity.Y * DeltaTime + Acceleration[i].Y
* DeltaTime * DeltaTime / 2;
                EntityGroup[i].Position.Z += EntityGroup[i].Velocity.Z * DeltaTime + Acceleration[i].Z
* DeltaTime * DeltaTime / 2;
                EntityGroup[i].Velocity.X += Acceleration[i].X * DeltaTime;
                EntityGroup[i].Velocity.Y += Acceleration[i].Y * DeltaTime;
                EntityGroup[i].Velocity.Z += Acceleration[i].Z * DeltaTime;
            }
        }
```

精确计时部分的实现代码：

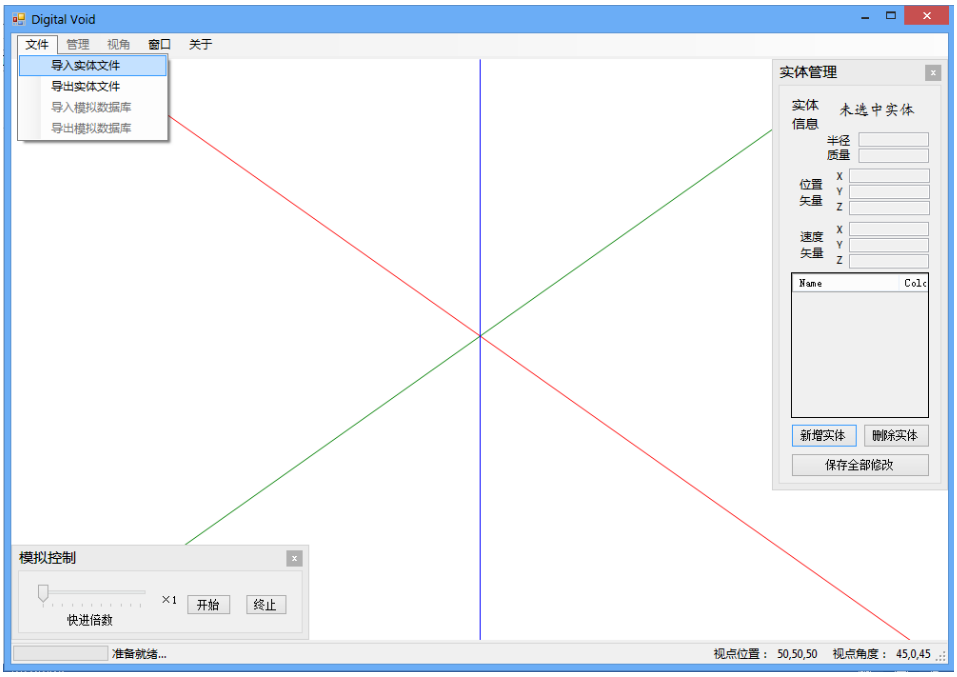
```
namespace ConsoleTimer
{
    class Program
    {
        static void Main(string[] args)
        {
            MainForm MainForm = new MainForm();
            PlayControl PC = new PlayControl(MainForm);
            PC.PlayPress();
            Thread.Sleep(1000);
            PC.PlayPress();
            Thread.Sleep(1000);
            PC.StopPress();
        }
    }
}

namespace Digital_Void
{
    public partial class MainForm : Form
    {
    (略)……
    private void timer_Tick(object sender, EventArgs e)
        {
            (略)……
            Span = (DateTime.Now - StartTime).Add(this.TimePauseSpan);
            SpanDateTime = new DateTime(Span.Ticks);
            this.TimeLabel.Text = SpanDateTime.ToString("HH:mm:ss.ff");
        }
     }
```

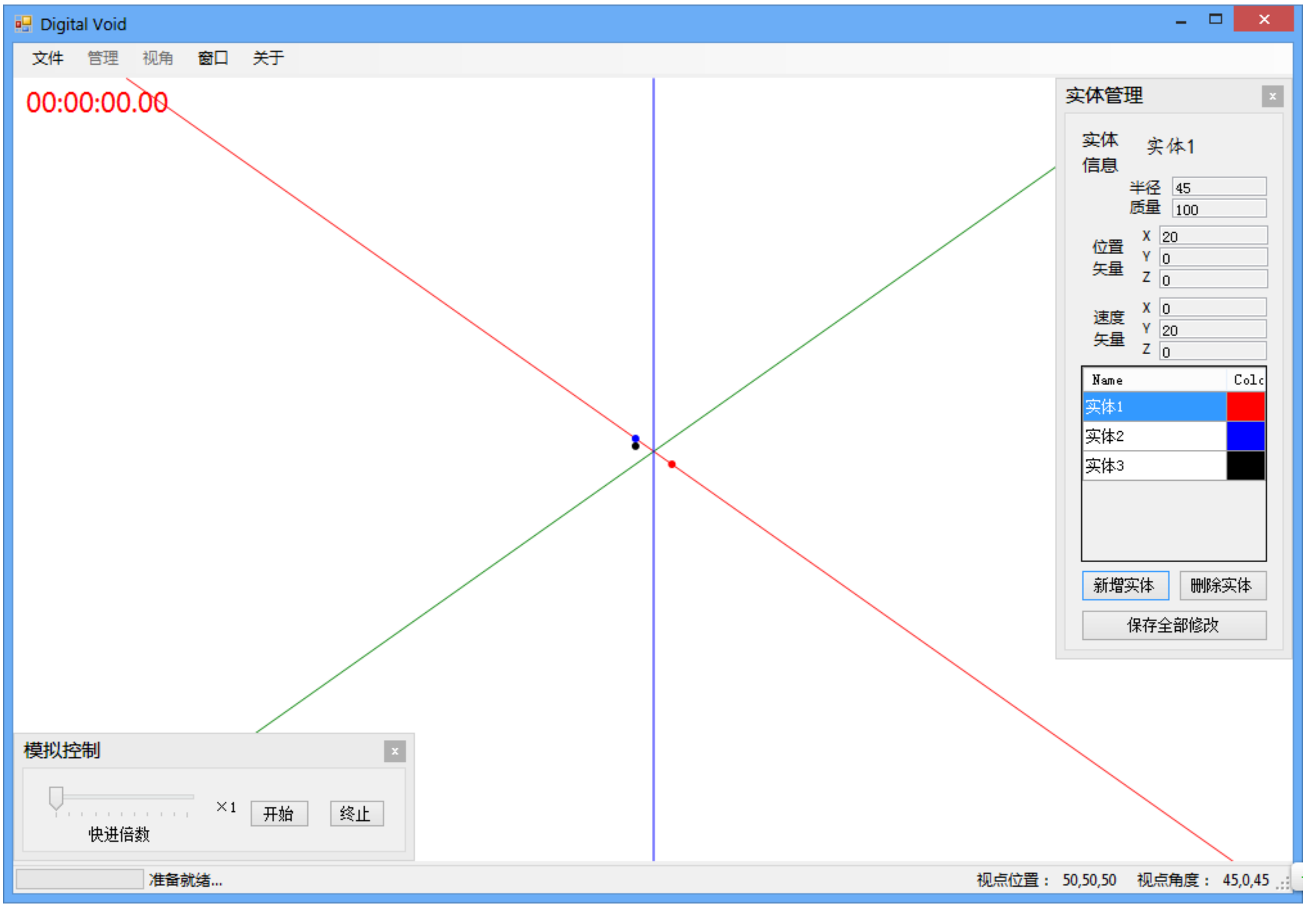
四、 项目测试

程序启动运行：

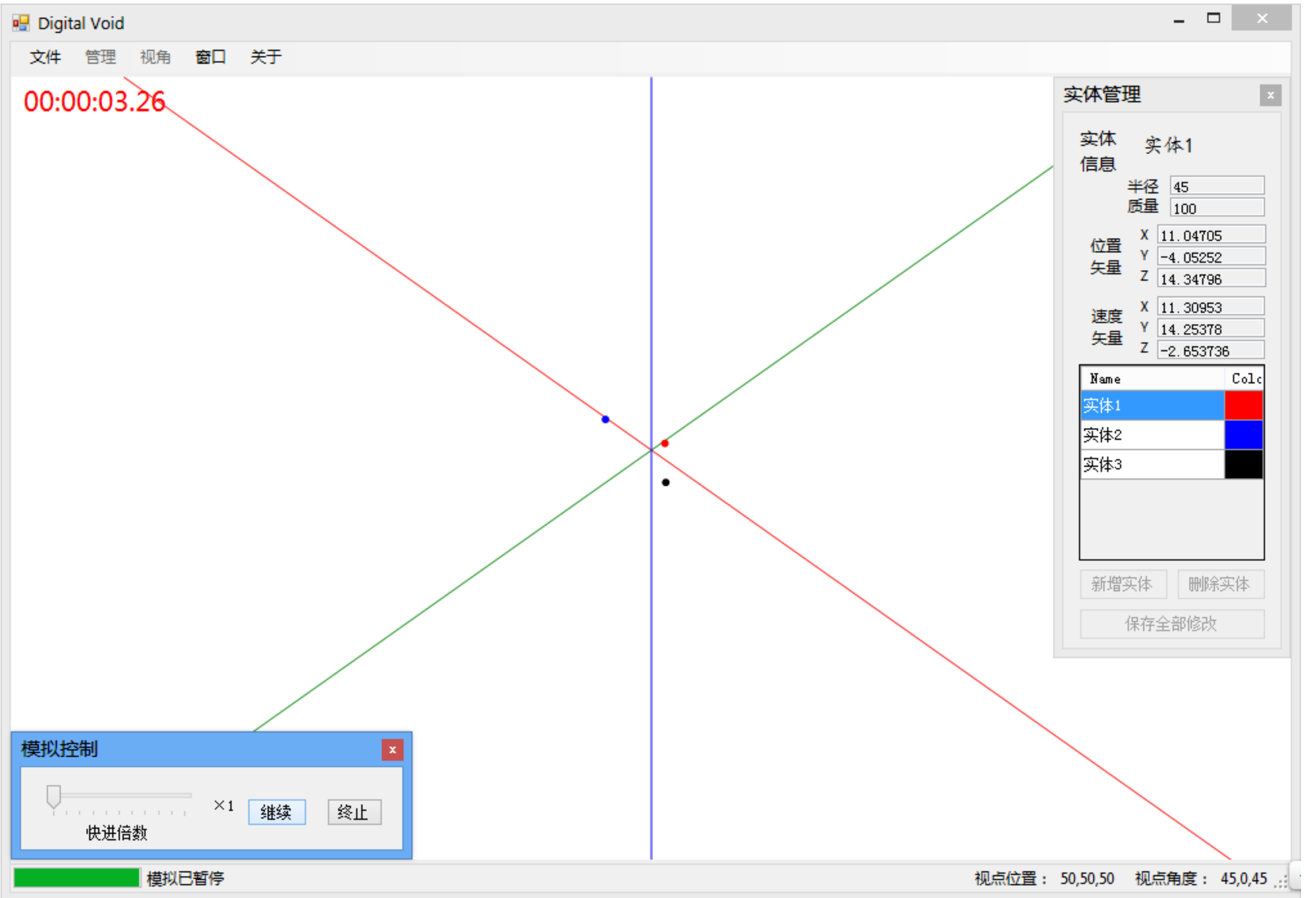


菜单栏单击文件-导入实体数据，也可以在右侧实体管理窗口自行创建实体、设置初始位置与速度信息。





单击左下角模拟控制窗口中的开始按钮，开始模拟。



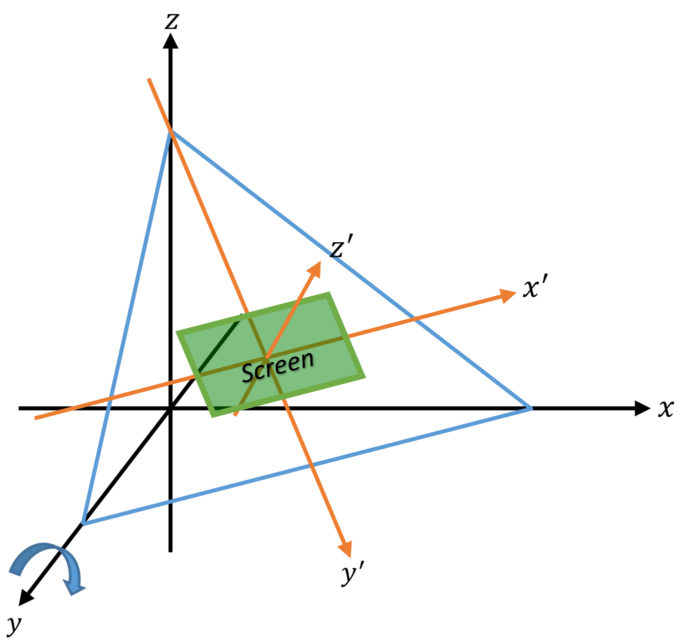
五、 结论与心得体会

本程序现阶段实现了预定的基本功能，可以顺利地模拟多个实体的类星体运动。但是仍有很多未完成的功能与有待优化的地方。

总的来说，这次程序的编写充分锻炼了我 C#的编程能力，GDI+、ADO 以及多线程都在程序中得到了使用。同时也让我对正交投影、线性变换等数学技巧有了更深一步的掌握。而这些都必然地为今后的进一步学习研究打下坚实的基础。

六、 附录

- 正交投影：



正交投影是指对于 xyz 坐标系中的任意一点，其转换为在另一坐标系 x'y'z'中的坐标的变换方法。如右图，xyz 坐标系为世界坐标，x'y'z'坐标系中的 x'0'y'部分为显示屏所在位置（具体部分用绿色矩形标出），其变换方法由下式表述：

$$\begin{bmatrix} d_x \\ d_y \\ d_z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x & 0 \\ 0 & \sin\theta_x & \cos\theta_x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 & 0 \\ \sin\theta_z & \cos\theta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \left(\begin{bmatrix} a_x \\ a_y \\ a_z \\ 1 \end{bmatrix} - \begin{bmatrix} c_x \\ c_y \\ c_z \\ 1 \end{bmatrix} \right)$$

- 经典多体运动：

以万有引力为相互作用力的多体运动由下式表达：

$$m_j \ddot{q}_j = \gamma \sum_{k \neq j}^{n} \frac{m_j m_k (q_j - q_k)}{|q_j - q_k|^3} \quad j = 1,2,\dots,n$$

然而，$n \geq 3$时，方程组陷入混沌而无法找到通解。一般来说我们都会采用数值解法（线性化）来求得一个近似的情况，其基本步骤为：

T_i时刻，q_j确定，随即确定$\ddot{q}_j$；

$T_{i+1} = T_i + \delta_T$时刻，$q_j{}' - q_j = \dot{q}_j \delta_T + \frac{1}{2} \ddot{q}_j {\delta_T}^2$，随即进一步确定$\ddot{q}_j{}'$。