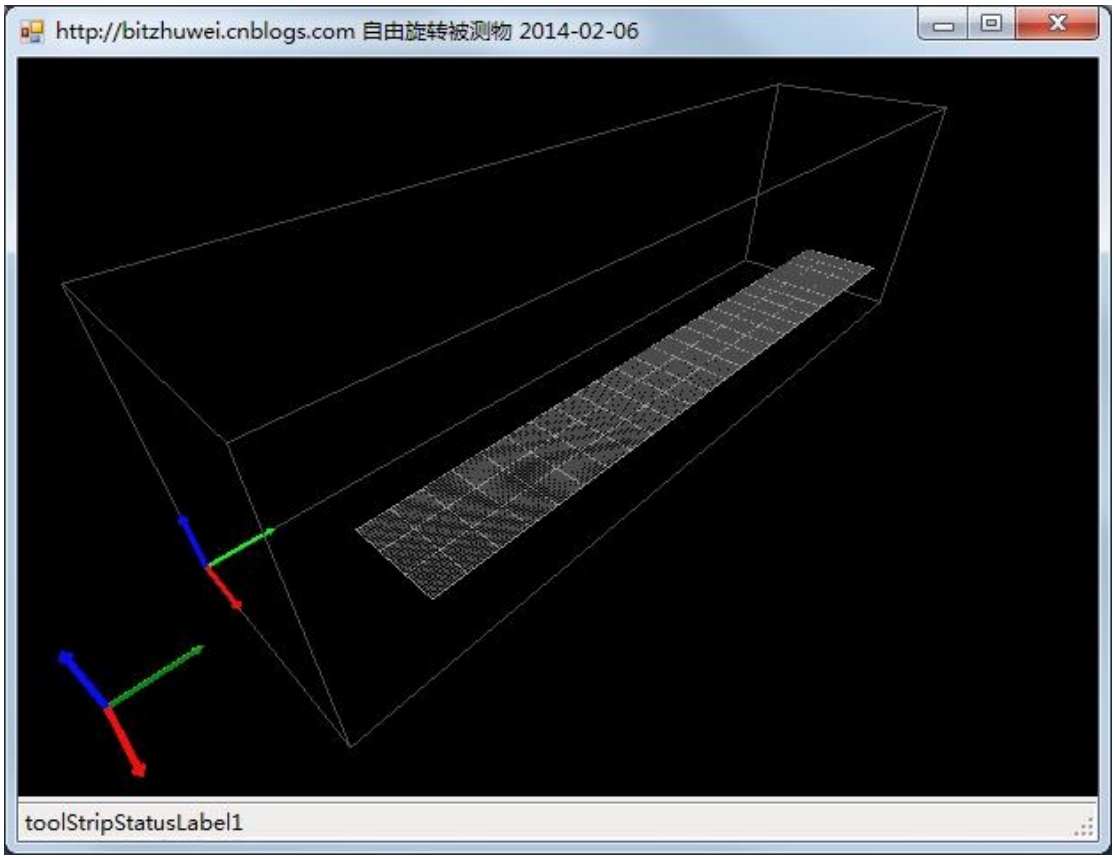
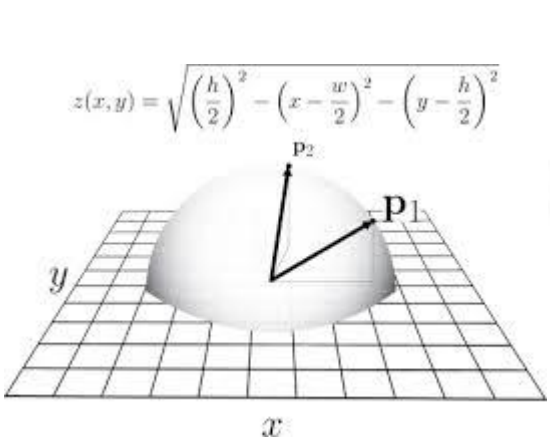
【OpenGL(SharpGL)】支持任意相机可平移缩放的轨迹球

在 3D 程序中，轨迹球（ArcBall）可以让你只用鼠标来控制模型（旋转），便于观察。在这里（http://www.yakergong.net/nehe/ ）有 nehe 的轨迹球教程。

本文提供一个本人编写的轨迹球类（ArcBall.cs），它可以直接应用到**任何 camera** 下，还可以同时实现**缩放**和**平移**。工程源代码在文末。



1. 轨迹球原理



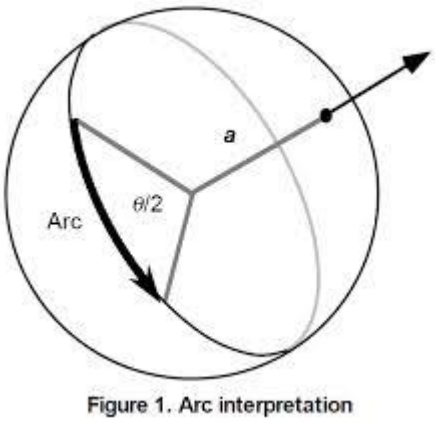


Figure 1. Arc interpretation

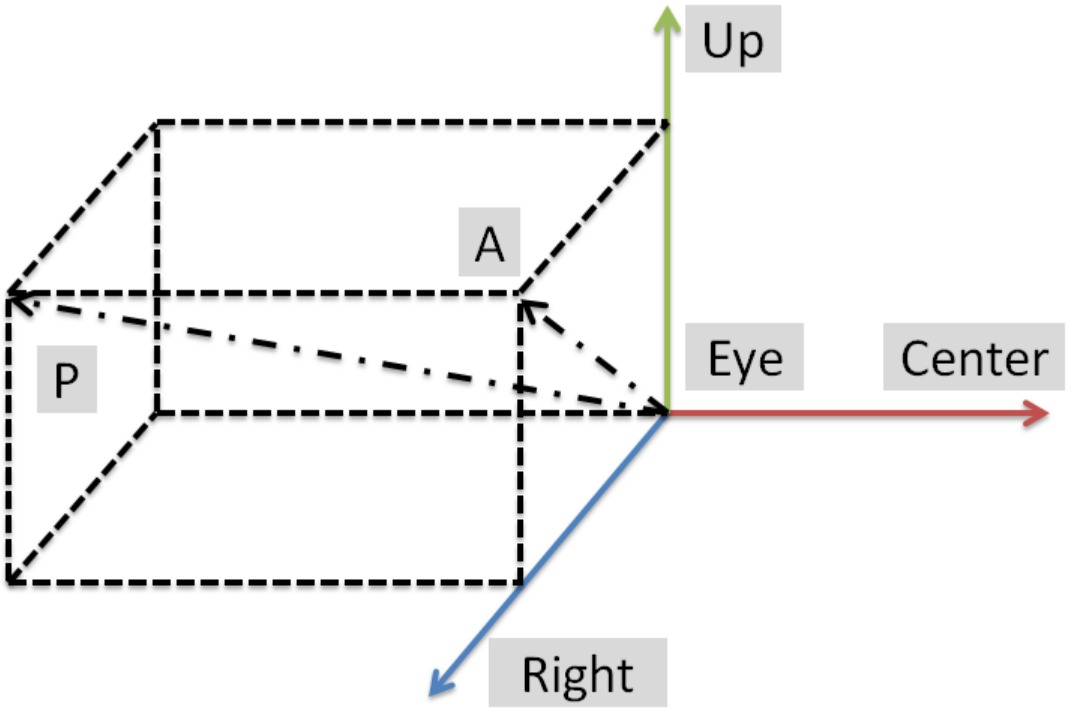
上面是我黑来的两张图，拿来说明轨迹球的原理。
看左边这个，网格代表绘制 3D 模型的窗口，上面放了个半球，这个球就是轨迹球。假设鼠标在网格上的某点 A，过 A 点作网格所在平面的垂线，与半球相交于点 P，P 就是 A 在轨迹球上的投影。鼠标从 A1 点沿直线移动到 A2 点，对应着轨迹球上的点 P1 沿球面移动到了 P2。那么，从球心 O 到 P1 和 P2 分别有两个向量 OP1 和 OP2。OP1 旋转到了 OP2，我们就认为是模型也按照这个方式作同样的旋转。这就是轨迹球的旋转思路。
右边这个图没用上…

2. 轨迹球实现

实现轨迹球，首先要求出鼠标点 A1、A2 投影到轨迹球上的点 P1、P2 的坐标，然后计算两个向量 A1P1 和 A2P2 之间的夹角以及旋转轴，最后让模型按照求出的夹角和旋转轴，调用 glRotate 就可以了。

1) 计算投影点

在摄像机上应用轨迹球，才能实现适应任意位置摄像机的 ArcBall 类。



如图所示，红绿蓝三色箭头的交点是摄像机 eye 的位置，红色箭头指向 center 的位置，绿色箭头指向 up 的位置，蓝色箭头指向右侧。
说明：1.Up 是可能在蓝色 Right 箭头的垂面内的任意方向的，这里我们要把它调整为与红色视线垂直的 Up，即上图所示的 Up。2.绿色和蓝色箭头组成的平面即为程序窗口所在位置，因为 Eye 就在这里嘛。而且 Up 指的就是屏幕正上方，Right 指的就是屏幕正右方。3.显然轨迹球的半球在图中矩形所在这一侧，球心就是 Eye。

鼠标在 Up 和 Right 所在的平面移动，当它位于 A 点时，投影到轨迹球的点 P。现在已知的是 Eye、Center、原始 Up、A 点在屏幕上的坐标、向量 Eye-P 的长度、向量 AP 的长度。现在要求 P 点的坐标，只不过是一个数学问题了。

当然，开始的时候要设置相机位置。

初始化 – 设置相机位置

```
        public void SetCamera(float eyex, float eyey, float eyez,
            float centerx, float centery, float centerz,
            float upx, float upy, float upz)
        {
            _vectorCenterEye = new Vertex(eyex - centerx, eyey - centery,
eyez - centerz);
            _vectorCenterEye.Normalize();
            _vectorUp = new Vertex(upx, upy, upz);
            _vectorRight = _vectorUp.VectorProduct(_vectorCenterEye);
            _vectorRight.Normalize();
            _vectorUp = _vectorCenterEye.VectorProduct(_vectorRight);
            _vectorUp.Normalize();
        }
```

根据鼠标在屏幕上的位置投影点的计算方法如下。

根据鼠标在屏幕上的位置投影点

```
        private Vertex GetArcBallPosition(int x, int y)
        {
            var rx = (x - _width / 2) / _length;
            var ry = (_height / 2 - y) / _length;
            var zz = _radiusRadius - rx * rx - ry * ry;
            var rz = (zz > 0 ? Math.Sqrt(zz) : 0);
            var result = new Vertex(
                (float)(rx * _vectorRight.X + ry * _vectorUp.X + rz *
_vectorCenterEye.X),
                (float)(rx * _vectorRight.Y + ry * _vectorUp.Y + rz *
_vectorCenterEye.Y),
                (float)(rx * _vectorRight.Z + ry * _vectorUp.Z + rz *
_vectorCenterEye.Z)
                );
            return result;
        }
```

这里主要应用了向量的思想，向量(Eye-P) = 向量(Eye-A) + 向量(A-P)。而向量(Eye-A)和向量(A-P)都是可以通过单位长度的 Up、Center-Eye 和 Right 向量求得的。

2) 计算夹角和旋转轴

首先，设置鼠标按下事件

设置鼠标按下事件

```
        public void MouseDown(int x, int y)
        {
            this._startPosition = GetArcBallPosition(x, y);

            mouseDownFlag = true;
        }
```

然后，设置鼠标移动事件。此时 P1P2 两个点都有了，旋转轴和夹角就都可以计算了。

设置鼠标移动事件

```
        public void MouseMove(int x, int y)
        {
            if (mouseDownFlag)
            {
                this._endPosition = GetArcBallPosition(x, y);
                var cosAngle = _startPosition.ScalarProduct(_endPosition)
/ (_startPosition.Magnitude() * _endPosition.Magnitude());
                if (cosAngle > 1) { cosAngle = 1; }
                else if (cosAngle < -1) { cosAngle = -1; }
                var angle = 10 * (float)(Math.Acos(cosAngle) / Math.PI *
180);
                System.Threading.Interlocked.Exchange(ref _angle,
angle);
                _normalVector =
_startPosition.VectorProduct(_endPosition);
                _startPosition = _endPosition;
            }
        }
```

然后，设置鼠标弹起的事件。

```
        public void MouseUp(int x, int y)
        {
            mouseDownFlag = false;
        }
```

在使用 opengl（sharpgl）绘制的时候，调用

```
        public void TransformMatrix(OpenGL gl)
        {
```

```
            gl.PushMatrix();
            gl.LoadIdentity();
            gl.Rotate(2 * _angle, _normalVector.X, _normalVector.Y,
_normalVector.Z);
            System.Threading.Interlocked.Exchange(ref _angle, 0);
            gl.MultMatrix(_lastTransform);
            gl.GetDouble(Enumerations.GetTarget.ModelviewMatix,
_lastTransform);
            gl.PopMatrix();
            gl.Translate(_translateX, _translateY, _translateZ);
            gl.MultMatrix(_lastTransform);
            gl.Scale(Scale, Scale, Scale);
        }
```

3. 额外功能实现

缩放很容易实现，直接设置 Scale 属性即可。
沿着屏幕上下左右前后地移动，则需要参照着 camera 的方向动了。

使模型向上移动

```
        public void GoUp(float interval)
        {
            this._translateX += this._vectorUp.X * interval;
            this._translateY += this._vectorUp.Y * interval;
            this._translateZ += this._vectorUp.Z * interval;
        }
```

其余方向与此类似，不再浪费篇幅。

源代码请在此文末下载。（本文链接无法更新，所以放到网页上，可以下到最新版程序。）
链接：（http://www.cnblogs.com/bitzhuwei/p/arcball_4_all_camera.html ）