湖南大學

HUNAN UNIVERSITY

本科生毕业设计(论文)

设计（论文）题目： 基于基尔霍夫理论的水下单刚体运动仿真算法研究

学生姓名：	王艺菲
学生学号：	201526010204
专业班级：	数字媒体技术 1501
学院名称：	信息科学与工程学院
指导老师：	周世哲
学院院长：	李肯立

2019 年 5 月 26 日

毕业设计（论文）原创性声明

本人郑重声明：所呈交的设计（论文）是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写的成果作品。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律后果由本人承担。

学生签名：　　日期：20　　年　月　日

毕业设计（论文）版权使用授权书

本毕业设计（论文）作者完全了解学校有关保留、使用设计（论文）的规定，同意学校保留并向国家有关部门或机构送交设计（论文）的复印件和电子版，允许设计（论文）被查阅和借阅。本人授权湖南大学可以将本设计（论文）的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本设计（论文）。

本设计（论文）属于

1、保　密□，在_____年解密后适用本授权书。

2、不保密□。

（请在以上相应方框内打“√”）

学生签名：　　日期：20　　年　月　日

导师签名：　　日期：20　　年　月　日

基于基尔霍夫理论的水下单刚体运动仿真算法研究

摘　　要

流体与物体的相互作用在计算机图形学文献中得到了广泛的研究。本文研究内容是的是刚体在流体中的运动。本文实现的算法基于基尔霍夫理论。基尔霍夫模型有如下基本假设：刚体单独存在于无限体积的无粘性和不可压缩流体中。除了满足流体不会穿过刚体的边界条件所需的最小流量，整个流体处于静止状态。流体速度场是无旋的（即无涡度或湍流），且无水动力耦合。移动的刚体不会在下游产生任何尾流，一旦物体停止移动，液体就会完全静止。用基尔霍夫张量代替惯量张量和标量质量，可以比较真实地模拟刚体在水中的运动。这使得我们可以在不模拟周围流体的情况下模拟流体与物体的相互作用。本文对比了一些水下刚体与真空中刚体运动的实验现象，并做出了一些合理的解释。

关键词:　基尔霍夫张量；附加质量张量；水下；刚体动力学

Research on underwater single rigid body motion simulation algorithm based on kirchhoff theory

Abstract

Fluid-body interactions have been extensively studied in the computer graphics literature.The research content of this paper is the motion of rigid body in fluid.The algorithm implemented in this paper is based on kirchhoff's theory.Kirchhoff model has the following basic assumptions: rigid body exists alone in infinite volume of non-viscous and incompressible fluid.The entire fluid is at rest except for the minimum flow required to satisfy the boundary condition that the fluid does not cross the rigid body.The fluid velocity field is rotation-free (i.e., no vorticity or turbulence), and there is no hydrodynamic coupling.A moving rigid body does not create any wake downstream, and once the object stops moving, the liquid is completely stationary.The motion of rigid body in water can be simulated by replacing inertia tensor and scalar mass with kirchhoff tensor.This allows us to simulate the interaction of a fluid with an object without simulating the surrounding fluid.In this paper, some experimental phenomena of underwater rigid body and vacuum rigid body motion are compared, and some reasonable explanations are given.

Key words: **kirchhoff tensor, additional mass tensor, underwater, rigid body dynamics**

插图索引

附表索引

1 绪论

1.1 研究背景及意义

计算机图形学（Computer Graphics，简称 CG）是一种使用数学算法将二维或三维图形转化为计算机显示器的栅格形式的科学。基于物理的物体仿真是计算机图形学的一个重要分支。

计算机动画的历史早在 20 世纪四五十年代就开始了，那时人们开始尝试计算机图形学——最著名的是约翰·惠特尼。直到 20 世纪 60 年代早期，数字计算机才得到广泛应用，创新计算机图形学的新途径才得以蓬勃发展。最初，主要用于科学、工程和其他研究目的，艺术实验在 20 世纪 60 年代中期开始出现。到 20 世纪 70 年代中期，许多这样的努力开始进入公共媒体。当时的许多计算机图形学都涉及二维图像，尽管随着计算机能力的提高，实现三维真实感的努力越来越受到重视。到 20 世纪 80 年代末，真实感 3D 开始出现在电影中，到 90 年代中期，3D 动画已经发展到可以用于整个故事片制作的地步。

基于物理的计算机动画的发展源于人们对游戏，电影等模拟世界真实性的需要，这不但减轻了程序开发人员的劳动强度，还减轻了设计师们的劳动强度。其中包括：粒子系统，关节与弹簧系统，布料系统。流体，刚体动力学与碰撞检测，物体材料属性（表面粗糙度，弹性，硬度等等）。

本文研究的主体——刚体是一种理想的模型，不会发生形变且内部各点相对位置不会改变。在真空中运动的刚体广泛应用于电影和游戏的物理模拟中。在设计相应的算法时所面临的主要挑战来自于用高效和鲁邦的方法处理碰撞和其他约束。在水下运动的刚体同样值得被人们所关注，比如去年年末热映的电影《海王》由于其主要场景均在海底，其实就需要运用到水下运动的物体动力学内容。

本文的研究意义在于能够用更高的效率得到更好的视觉效果，水下刚体的运动如果不用到工业，医疗仿真及 CAD 等等领域，物体的运动精度就不是那么重要。

1.2 国内外研究状况

流体与物体的相互作用在计算机图形学文献中得到了广泛的研究。但很多研究只模

拟单相流，一种是只考虑固体对流体的作用，已知固体运动以及某些属性，这种方法不考虑流体对固体的作用。而另一种只考虑流体对固体的作用。这两种方法忽略了两者相互的作用，许多学者转而研究流固耦合的问题。如 Carlson 等人在 2004 年提出的 Rigid Fluid 法[1]，在整个的计算过程中都把刚体看作是流体的一部分。用拉格朗日因子来结合刚体与流体的相互作用，是一个不错的方法。Takahashi[2]等人利用压强在固体表面的积分作为物体所受的外力，还将固体的速度作为 N-S 方程组的一个边界条件，这是对固液耦合模拟的一种简单的方法。

Treuille 等人在 2006 年[3]以强烈的预计算和过多的内存需求为代价实现了对可变形物体进行了模拟实时性能。所有这些方法都依赖于对周围流体的模拟，使得计算成本非常昂贵。Ozgen 等人在 2010 年[4]不模拟周围水体的情况下模拟了水下运动的布料。Baraff 等人在 1993 年[5]于基尔霍夫的方法也不需要对流体进行模拟，而是对真空中刚体动力学的运动方程进行了修改，以考虑流体流动对物体运动的影响。

Kobilarov 等人在 2009 年[6]提出了基尔霍夫方程已经在 CG（计算机图形学）文献中应用于具有非完整约束的车辆动力学，但没有转动和平动的耦合。Holmes 等人在 1998 年[7]这个案例也从可整合性的角度进行了研究。Bou-Rabee 在 2007 年[8]Nordkvist and Sanyal 2010[9]为系统开发了不同的李群积分器。Vankerschaver 等人在 2009[10]用基尔霍夫张量研究流体-物体与点涡的耦合；Shashikanth 等人在 2008 年[11]用基尔霍夫张量研究漩涡细丝。Nair 和 Kanso 在 2007 年[12]也考虑了几个物体之间的水动力耦合。Newman 在 1977 年[13]提出了在军舰架构上的应用，Kanso 等人在 2005 年[14]以及 Lee 等人在 2009 年[15]提出了军舰移动上的应用。

Field 等人在 1997 年[16]，Pesavento 和 Wang 两人在 2004[17]年以及 Zhong 等人在 2011 年[18]进行了不同条件下的落体行为的实验研究，但是没有一项关于基尔霍夫运动方程的数值模拟能真实再现水下物体行为的研究。于是 Steffen 等人在 2012 年[19]提出了基尔霍夫应用于水下刚体运动的研究。

1.3 本文结构安排

全文总共七章，第一章是绪论部分，主要讲解了水下刚体动力学研究背景及意义，流体与物体相互作用的国内外研究现状以及本文的结构安排。

第二章讲述了本文算法的理论基础，包括基本假设，算法核心思想，预计算求解基

尔霍夫张量的过程以及仿真的过程。这一章是本文重点。

第三章讲述了刚体运动过程中所受外力的计算。

第四章展示了对一些模型的仿真结果。

第五章对仿真结果进行了分析并进行了一些合理的解释。

第六章对本文的水下刚体动力学算法做了总结，并进行了展望，对未来的研究方向进行了一些思考。

2 理论基础

2.1 基本假设

假设每个物体在无限体积的无粘性和不可压缩流体中是单独存在的。整个流体处于静止状态，除了满足流体不会穿过物体的边界条件所需的最小流量。流体速度场是无旋的（即无涡度或湍流），且无水动力耦合:即使是两个相邻的物体也不会影响彼此的运动。移动的物体不会在下游产生任何尾流，一旦物体停止移动，液体就会完全静止。

根据基本假设我们得到如下三个公式，公式（1）表示无穷远处速度势为 0，（2）是纽依曼问题边界条件的限制，（3）表示流体不可压缩，其中 B 表示刚体，∂B 则表示刚体的边界，φ表示流体的速度势，$grad\,\varphi(z)$ 表示流体在 z 点处的速度，n(z)表示的曲面法线∂B。

$$\phi(z)=0,\|z\|\to\infty, \tag{1}$$

$$\langle grad\phi(z)-(\omega\times z+v),n(z)\rangle=0,z\in\partial B, \tag{2}$$

$$\triangle\phi(z)=0,z\in R^3\setminus B. \tag{3}$$

2.2 算法核心思想

刚体下落和在碰撞之间自由旋转的情况对于许多情况（如碎片四处乱飞）很重要。在这种情况下，欧拉已经知道产生的运动:刚体 B 的惯量告诉我们使物体运动所涉及的动能。如果运动是平移速度 v，动能是 $1/2MV^2$。如果除此之外我们也有一个旋转角速度 ω 的质心能量是：

$$E=\frac{1}{2}m|v|^2+\frac{1}{2}\langle\omega,J_B\omega\rangle$$

J_B 是物体的惯性张量。由此可见，在没有重力的情况下，质心 c 的运动和相对于 c 的运动（由时间相关的正交矩阵 R 描述）是完全解耦的。

如果我们考虑物体在液体中运动，整个画面就会发生巨大的变化。现在物体的任何运动都必然涉及到周围流体的运动，这增加了运动的动能。为了解决这个问题，我们原则上必须模拟包含流体和浸入体的整个系统，这会导致一个更难计算的问题。Kirchhoff 在 1870 年[20]提出了一个巧妙的想法来简化这个问题，并且只修改惯性张量来包含周围

流体运动的动能。用基尔霍夫张量代替质量惯性张量，就可以真实地模拟刚体在水中的运动。基尔霍夫张量是真空中物体的质量惯性张量(K_B)与附加质量张量 K_F 之和。J_B 是标准的 3×3 惯性张量，I 是 3×3 单位矩阵，m 是刚体的质量。

$$K = K_B + K_F,\ K_B = \begin{pmatrix} J_B & 0 \\ 0 & mI \end{pmatrix} \tag{4}$$

l 表示角动量 p 表示动量

$$\begin{pmatrix} l \\ p \end{pmatrix} = \underbrace{\left(K_B + K_F\right)}_{K} \begin{pmatrix} \omega \\ v \end{pmatrix} = \begin{pmatrix} l_B \\ p_B \end{pmatrix} + \begin{pmatrix} l_F \\ p_F \end{pmatrix} \tag{5}$$

根据公式（5）我们注意到基尔霍夫张量不依赖于物体的运动姿态。因此，基尔霍夫张量是在预计算步骤中，即只需要计算一次。

2.3 预计算过程

我们接下来计算附加质量张量 K_F 和质量-惯性张量矩阵 K_B ，然后由公式（4）将两项相加得到基尔霍夫张量 K

2.3.1 求解附加质量张量 KF

选取六组 $(\omega, v)^T$ 向量，组成 6*6 的单位阵，即

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

计算出物体周围流体的 $(l, p)^T$（6 组，即 6*6 的矩阵），根据（5），计算得到的 lp 矩阵即为 K_F 矩阵。流体总角动量的定义为流体内单位体积每一点到中心点的距离叉乘该点的动量之和，总动量的定义为流体内单位体积的动量之和，写出如下公式：

$$l_F = \rho \iiint_{R^3 \setminus B} z \times grad\phi(z) dA$$

$$p_F = \rho \iiint_{R^3 \setminus B} grad\phi(z) dA$$

其中 $R^3 \setminus B$ 表示的三维区域为空间中无限大的区域除去刚体所在的区域。上述公式再通

过散度定理（高斯公式），将在无限域上的体积积分转化到刚体表面的面积分，得到

$$l_F = \rho\iiint_{R^3\setminus B} z\times grad\phi(z)dA = \rho\int_{\partial B}\phi(z)z\times n(z)dA, \tag{6}$$

$$p_F = \rho\iiint_{R^3\setminus B} grad\phi(z)dA = \rho\int_{\partial B}\varphi(z)n(z)dA \tag{7}$$

刚体表面∂B 的速度势Φ(z)目前是未知的，需要进行计算。这里运用势能理论进行计算：通过选取位于 sj 位置，有着强度为σj 的点源的集合来近似求解速度势Φ。

$$\phi(z) = \sum_j \frac{\sigma_j}{\|z-s_j\|}, grad\phi(z) = -\sum_j \sigma_j \frac{z-sj}{\|z-s_j\|^3} \tag{8}$$

势能理论中 sj 位于物体 B 的内部，在这里，我们简单地使用顶点位置在物体表面法线反方向的偏移作为 sj。然后条件（1）和（3）就已经自动满足了，通过限定边界条件（2），得到公式（9）

$$\left\langle \sum_j \sigma_j \frac{z-s_j}{\|z-s_j\|^3}, n(z) \right\rangle = \langle \omega\times z+v, n(z)\rangle, z\in\partial B. \tag{9}$$

当刚体表面是离散的用面Γi 表示时，把（9）写成面积分的形式

$$\sum_j \sigma_j \int_{\Gamma_i} \left\langle \frac{z-s_j}{\|z-s_j\|^3}, n(z) \right\rangle dA = \int_{\Gamma_i} \langle \omega\times z+v, n(z)\rangle dA. \tag{10}$$

对全部刚体面片，可以写成这种形式

$$M\sigma = f, M_{ij} = \int_{\Gamma_i} \left\langle \frac{z-s_j}{\|z-s_j\|^3}, n(z) \right\rangle dA \tag{11}$$

其中，M 是一个矩阵面片数*源点数的矩阵，σ是源点数*1 的向量，f 是面片数*1 的向量。根据公式（11），我们可以看出 Mij 是以 sj 为源点，Γi 为面的立体角（立体角的定义：任意定向曲面 S 相对于某一个点 P 的立体角，即为该曲面投影到以 P 为球心的单位球面上的面积。另向量 r 为该单位球面上以 P 为原点的极小面积的位置向量，可得公式：

$$\Omega = \iint_S \frac{dA}{r^2} = \iint_S \frac{\vec{r}\bullet d\vec{S}}{|\vec{r}|r^2} = \iint_S \frac{\vec{r}\bullet d\vec{S}}{r^3}\ )$$

然后 Oosterom 等人在 1983 年[21]提出了立体角的加速计算方法，这里运用它们的方法求立体角矩阵（M 矩阵），公式如下

$$\tan\left(\frac{1}{2}\Omega\right)=\frac{[R_1R_2R_3]}{R_1R_2R_3+(R_1\bullet R_2)R_3+(R_1\bullet R_3)R_2+(R_2\bullet R_3)R_1}.$$

其中 R 是从源点到面片上各个顶点的向量，[R1R2R3]表示由三个向量 R1R2R3 组成矩阵的行列式。

然后求解 f 矩阵（由于选取了六组 ω，v 向量，这是一个 face*6 的矩阵）

$$\int_{\Gamma_i}\langle\omega\times z+v,n(z)\rangle dA.$$

矩阵的前三列 v=0，可化为混合积，进而转为求行列式求解

$$\int_{\Gamma_i}\omega\times z\bullet n(z)dA=-\omega\times n(z)\bullet\int_{\Gamma_i}zdA=-\omega\times n(z)\bullet(\mathrm{p}_1+\mathrm{p}_2+\mathrm{p}_3)/3.$$

其中 p1p2p3 为面 i 的三个顶点。

矩阵的后三列ω=0，就是面积向量（面积*面的法向）

然后通过求解六个线性方程组成的线性方程组得到 σ 矩阵

想要求解出 Φ (z)还需要计算矩阵 $\frac{1}{\|z-s_j\|}$ (single_layer)，然后由（8）计算出 φ(z) (single_layer*σ)

将公式（6）（7）离散，并写成一点积分的形式，即面 i 上每个点处的函数值都相等，用在面的中点处计算出来的值乘以该面面积代替在这个面上的面积分

$$l_F=\rho\sum_{i=0}^{n}\phi(z)z\times n(z)vol(\Gamma_i)$$

$$p_F=\rho\sum_{i=0}^{n}\phi(z)n(z)vol(\Gamma_i)$$

求出上式不包含φ项的 Q 矩阵（这个矩阵的大小是 6*面片数，前三行是 l 矩阵，后三行是 p 矩阵），然后用φ*Q 求出 LP 矩阵，即得到附加质量惯性张量 K_F 矩阵。

2.3.2 求解质量-惯性张量矩阵 K_B

K_B 的形式见（3），需要计算惯性张量矩阵 J_B 和刚体的质量。polyhedral 在 2002 年[21]提出了多面体的质量特性，接下来的讲述的计算方法主要基于这篇论文。

惯性张量由相同的惯性矩和关于三个坐标轴的惯性积组成。惯性张量由九个分量张量构成。表达式定义如下：

$$\begin{pmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{pmatrix}$$

$$\begin{pmatrix} I_{xx}=\iiint_B (y^2+z^2)dm & I_{xy}=-\iiint_B xydm & I_{xz}=-\iiint_B xzdm \\ I_{yx=}I_{xy} & I_{yy}=\iiint_B (z^2+x^2)dm & I_{yz}=-\iiint_B yzdm \\ I_{zx=}I_{xz} & I_{zy=}I_{yz} & I_{zz}=\iiint_B (x^2+y^2)dm \end{pmatrix}$$

质量、质心（第 4 章会用到）和惯性张量需要计算这种类型的体积积分

$$\int_V p(x,y,z)dV$$

其中 V 是积分的体积区域，dV 是体积的无穷小度量。函数 p(x,y,z)是从 1、x、y、z、x^2、y^2、z^2、xy、xz 和 yz 中选出来的多项式，体积积分可以通过微积分中的散度定理转化为曲面积分:

$$\int_V p(x,y,z)dV=\int_V \nabla\bullet FdV=\int_S N\bullet FdS$$

：其中 S 是三角形面的并集。是多面体的边界，dS 是表面的无穷小量。函数 F (x, y, z)是选择这∇·F = p。向量 N 表示方向向外的单位长度表面法线。p 与 F 的对应关系给出如下

表 2. 1 p 函数与 F 函数对应关系表

p	F	p	F
1	(x,0,0)	y^2	$(0,y^3/3,0)$
x	$(x^2/2,0,0)$	z^2	$(0,0,z^3/3)$
y	$(0,y^2/2,0)$	xy	$(x^2y/2,0,0)$
z	$(0,0,z^2/2)$	xz	$(0,0,z^2x/2)$

x^2	$(x^3/2,0,0)$	yz	$(0,y^2z/2,0)$

那么现在只需要计算积分$\int_S N\bullet FdS$。边界 S 是多面体面Γ的并集，N_Γ是Γ面的单位外法向。曲面积分分解为

$$\int_S N\bullet FdS=\sum_{\Gamma\in S}\int_\Gamma N_\Gamma\bullet FdS$$

积分现在被简化为

$$\int_V dV=\sum_{\Gamma\in S}(N_\Gamma\bullet i)\int_\Gamma xdS$$
$$\int_V xdV=\frac{1}{2}\sum_{\Gamma\in S}(N_\Gamma\bullet i)\int_\Gamma x^2dS$$
$$\int_V ydV=\frac{1}{2}\sum_{\Gamma\in S}(N_\Gamma\bullet j)\int_\Gamma y^2dS$$
$$\int_V zdV=\frac{1}{2}\sum_{\Gamma\in S}(N_\Gamma\bullet k)\int_\Gamma z^2dS$$
$$\int_V x^2dV=\frac{1}{3}\sum_{\Gamma\in S}(N_\Gamma\bullet i)\int_\Gamma x^3dS$$
$$\int_V y^2dV=\frac{1}{3}\sum_{\Gamma\in S}(N_\Gamma\bullet j)\int_\Gamma y^3dS$$
$$\int_V z^2dV=\frac{1}{3}\sum_{\Gamma\in S}(N_\Gamma\bullet k)\int_\Gamma z^3dS$$
$$\int_V xydV=\frac{1}{2}\sum_{\Gamma\in S}(N_\Gamma\bullet i)\int_\Gamma x^2ydS$$
$$\int_V yzdV=\frac{1}{2}\sum_{\Gamma\in S}(N_\Gamma\bullet j)\int_\Gamma y^2zdS$$
$$\int_V xzdV=\frac{1}{2}\sum_{\Gamma\in S}(N_\Gamma\bullet k)\int_\Gamma z^2xdS$$

现在我们需要计算这种形式的积分

$$(N_\Gamma\bullet\ell)\int_\Gamma q(x,y,z)dS \qquad (12)$$

ℓ是 i，j，k 中的一个，q 是 x, x2, y2, z2, x3, y3, z3, x2y, y2z, z2x 中的一个

让三角形的面，按逆时针顺序排列，并有顶点 $P_i=(x_i,y_i,z_i),0\le i\le 2$，两个边是 $E_i=P_i-P_0=(x_i-x_0,y_i-y_0,z_i-z_0)=(\alpha_1,\beta_i,\gamma_i),1\le i\le 2$，面的参数方程如下

$$\begin{aligned}P(u,v)&=P_0+uE_1+vE_2=(x_0+\alpha_1u+\alpha_2v,y_0+\beta_1u+\beta_2v,z_0+\gamma_1u+\gamma_2v)\\&=(x(u,v),y(u,v),z(u,v)),u\ge 0,v\ge 0,u+v\le 1.\end{aligned} \qquad (13)$$

u 面积的无穷小量 dS 计算如下

$$dS = \left|\frac{\partial P}{\partial u} \times \frac{\partial P}{\partial v}\right| dudv = \left|E_1 \times E_2\right| dudv$$

面的单位外法向计算如下

$$N_\Gamma = \frac{E_1 \times E_2}{\left|E_1 \times E_2\right|} = \frac{(\beta_1\gamma_2 - \beta_2\gamma_1, \alpha_2\gamma_1 - \alpha_1\gamma_2, \alpha_1\beta_2 - \alpha_1\beta_1)}{\left|E_1 \times E_2\right|} = \frac{(\delta_0, \delta_1, \delta_2)}{\left|E_1 \times E_2\right|}$$

积分（12）可以写作

$$(N_\Gamma \bullet \ell)\int_\Gamma q(x,y,z)dS = (E_1 \times E_2 \bullet \ell)\int_0^1 \int_0^{1-v} q(x(u,v), y(u,v), z(u,v))dudv \qquad (14)$$

其中 x(u,v)，y(u,v)和 z(u,v)是式(13)中参数化的分量。方程(14)右边的积分可以用符号计算，可以用手工计算，也可以用符号代数包计算。后面列出的公式是使用 Mathematica 计算的。

一般的子表达式可以通过一些额外的因式分解得到。定义

$$s_n(w) = \sum_{i=0}^{n} w_0^{n-i} w_1^{i}, f_0(w) = 1, f_n(w) = s_n(w) + w_2 f_{n-1}(w) \quad n \geq 1$$

把这些看作宏，其中输入参数用函数替换。还需要定义宏：

$$g_i(w) = f_2(w) + w_i f_1(w) + w_i^2$$

下面以 w 的形式列出曲面积分中需要的具体表达式。每个宏展开三次，分别针对 x、y 和 z 展开一次。

$$f_1(w) = w_0 + w_1 + w_2 = [w_0 + w_1] + w_2$$
$$f_2(w) = w_0^{\,2} + w_0 w_1 + w_1^{\,2} + w_2 f_1(w) = [[w_0^{\,2}] + w_1 \{w_0 + w_1\}] + w_2 \{f_1(w)\}$$
$$f_3(w) = w_0^{\,3} + w_0^{\,2} w_1 + w_0^{\,2} w_1 + w_1^{\,3} + w_2 f_2(w) = w_0 \{w_0^{\,2}\} + w_1 \{w_0^{\,2} + w_0 w_1 + w_1^{\,2}\} + w_2 \{f_2(w)\}$$
$$g_i(w) = \{f_2(w)\} + w_i (\{f_1(w)\} + w_i)$$

方括号[・]表示子表达式被计算并保存在临时变量中，以供以后使用。大括号{・}表示子表达式是在前面计算的，可以从临时变量访问。每次必须存储的子表达式的数量很少，所以当有足够的浮点寄存器可用来存储子表达式时，缓存一致性应该不是问题(请参阅本文后面的附录 A 的伪代码)。

最后的积分表达如下

$$
\begin{aligned}
&\int_V dV = \sum_{\Gamma\in S}(N_\Gamma \bullet i)\int_\Gamma x dS = \frac{\delta_0}{6} f_1(x) \\
&\int_V x dV = \frac{1}{2}\sum_{\Gamma\in S}(N_\Gamma \bullet i)\int_\Gamma x^2 dS = \frac{\delta_0}{12} f_2(x) \\
&\int_V y dV = \frac{1}{2}\sum_{\Gamma\in S}(N_\Gamma \bullet j)\int_\Gamma y^2 dS = \frac{\delta_1}{12} f_2(y) \\
&\int_V z dV = \frac{1}{2}\sum_{\Gamma\in S}(N_\Gamma \bullet k)\int_\Gamma z^2 dS = \frac{\delta_2}{12} f_2(z) \\
&\int_V x^2 dV = \frac{1}{3}\sum_{\Gamma\in S}(N_\Gamma \bullet i)\int_\Gamma x^3 dS = \frac{\delta_0}{20} f_3(x) \\
&\int_V y^2 dV = \frac{1}{3}\sum_{\Gamma\in S}(N_\Gamma \bullet j)\int_\Gamma y^3 dS = \frac{\delta_1}{20} f_3(y) \\
&\int_V z^2 dV = \frac{1}{3}\sum_{\Gamma\in S}(N_\Gamma \bullet k)\int_\Gamma z^3 dS = \frac{\delta_2}{20} f_3(z) \\
&\int_V xy dV = \frac{1}{2}\sum_{\Gamma\in S}(N_\Gamma \bullet i)\int_\Gamma x^2 y dS = \frac{\delta_0}{60}(y_0 g_0(x) + y_1 g_1(x) + y_2 g_2(x)) \\
&\int_V yz dV = \frac{1}{2}\sum_{\Gamma\in S}(N_\Gamma \bullet j)\int_\Gamma y^2 z dS = \frac{\delta_1}{60}(z_0 g_0(y) + z_1 g_1(y) + z_2 g_2(y)) \\
&\int_V xz dV = \frac{1}{2}\sum_{\Gamma\in S}(N_\Gamma \bullet k)\int_\Gamma z^2 x dS = \frac{\delta_2}{60}(x_0 g_0(z) + x_1 g_1(z) + x_2 g_2(z))
\end{aligned}
\tag{15}
$$

（15）的第一个公式即模型的体积，乘以材料密度可以得到刚体的质量 M（第 4 章会用到）。

2.4 仿真过程

我们将刚体 B 在三维空间中的运动描述为欧几里得运动群中的曲线 g(t) = (R(t)，y(t))。3×3 的矩阵 R (t)描述了刚体的旋转矩阵，y (t)是刚体中心的位置。

首先要注意到坐标变换的问题，运动方程只是说线性动量/角动量的时间导数等于施加的力/力矩。然而，这个公式中的动量在世界坐标中表示，而基尔霍夫张量是在物体坐标系中给出的。

设 q 为物体坐标系下刚体表面上一点，在欧拉运动下，q 点从物体坐标系到世界坐标系的变换为 q-> $q_S = R\,q + y$ ，q 点在物体坐标系下的无穷小欧几里德运动的向量场为 u_q=w×q+v，q 在空间坐标系下的向量场直接左乘 R,，即

Ru_q=R(w×q+v)=R(w×R^T(q_S-y)+v)=(Rw)×q+YRw+Rv

，这里 Y 是斜伴随映射，作用在矩阵上等价于 y 叉乘的结果，因此欧几里德运动作用于速度状态如下

$$\begin{pmatrix}\omega\\ v\end{pmatrix}->\underbrace{\begin{pmatrix}\mathrm{R} & 0\\ YR & R\end{pmatrix}}_{g}\begin{pmatrix}\omega\\ v\end{pmatrix}$$

那么角动量与动量应该如何变换呢？假设一个映射 $\hat{g}$，使得(l,p)变换为 $\hat{g}$ (l,p),使它与标量积与速度状态（ ω，v）的点积保持不变（点积的结果表示动量，动量是一个标量，标量在任何坐标系下的值都是一样的），即

$$\left\langle \hat{g}\begin{pmatrix}l\\ p\end{pmatrix}, g\begin{pmatrix}\omega\\ v\end{pmatrix}\right\rangle = \left\langle \begin{pmatrix}l\\ p\end{pmatrix}, \begin{pmatrix}\omega\\ v\end{pmatrix}\right\rangle$$

这样就可以得到

$$\hat{g}=(\mathrm{g}^{t})^{-1}=\begin{pmatrix}R & YR\\ 0 & R\end{pmatrix}$$

可以得到

$$\begin{pmatrix}l_s\\ p_s\end{pmatrix}=\begin{pmatrix}R & YR\\ 0 & R\end{pmatrix}\begin{pmatrix}l\\ p\end{pmatrix}$$

或者同样的可以写作

$$\begin{pmatrix}l\\ p\end{pmatrix}=\begin{pmatrix}R^{t} & -R^{t}Y\\ 0 & R^{t}\end{pmatrix}\begin{pmatrix}l_s\\ p_s\end{pmatrix} \quad (16)$$

(ls,ps)的对时间的导数等于施加于物体的外部力矩和力

$$\begin{pmatrix}\dot{l}_s\\ \dot{p}_s\end{pmatrix}=\begin{pmatrix}t_s\\ f_s\end{pmatrix}$$

通过对公式(16)求导可得

$$\begin{pmatrix}\dot{l}\\ \dot{p}\end{pmatrix}=\begin{pmatrix}l\times\omega+p\times v\\ p\times\omega\end{pmatrix}+\begin{pmatrix}t\\ f\end{pmatrix} \quad (17)$$

合外力矩与合外力的坐标表示如下

$$\begin{pmatrix}t\\ f\end{pmatrix}=\begin{pmatrix}R^{t} & -R^{t}Y\\ 0 & R^{t}\end{pmatrix}\begin{pmatrix}t_s\\ f_s\end{pmatrix} \quad (18)$$

我们模拟水下刚体的方法发生在欧氏运动 SE(3)的李群中。因此，几何李群积分器是它的完美实现，正如 BouRabee 在 2007 年[7]所讲述的。在 Kobilarov 等人在 2009 的论

文[5]Section 4.2 中，作者为我们的场景详细描述了一个动量保持积分器，我们已经为我们的模拟实现了它，接下来我们讲述算法的主要流程。

李群方法是目前结构保持积分的一种重要方法，直接用旋转矩阵和坐标表示物体位姿，构成的矩阵是 4*4 的，属于矩阵李群，其在单位元处的切矢量称为对应的李代数。基于李群表达的动力学方程非常简单，即求解先在李代数空间上进行，然后借助重构方程（reconstruction equations）恢复到李群空间。

在物体坐标系下。角速度和速度表示为$\omega \in R^3, v \in R^3$，向量$(\omega,v) \in R^6$通过下式与李代数元素$\xi \in$ se(3)对应

$$\xi = \begin{pmatrix} \hat{\omega} & v \\ 0 & 1 \end{pmatrix}$$

映射$\hat{\cdot}$：R^3->so(3)定义如下

$$\hat{\omega} = \begin{pmatrix} 0 & -\omega^3 & \omega^2 \\ \omega^3 & 0 & \omega^1 \\ -\omega^2 & -\omega^1 & 0 \end{pmatrix}$$

这个系统的拉格朗日函数ℓ:R^6->R，如下表示

$$\ell(\omega,v) = K\begin{pmatrix} \omega \\ v \end{pmatrix}$$

由论文[]，得到以下两个公式

$$\begin{pmatrix} R_{k+1} & y_{k+1} \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} R_k & y_k \\ 0 & 1 \end{pmatrix}\begin{pmatrix} \tau(h\omega_k) & hB_\tau(h\omega_k)v_k \\ 0 & 1 \end{pmatrix} \tag{19}$$

$$C_\tau^T(h(\omega_k,v_k))\bullet K\begin{pmatrix} \omega_k \\ v_k \end{pmatrix} - C_\tau^T(-h(\omega_k,v_k))\bullet K\begin{pmatrix} \omega_{k-1} \\ v_{k-1} \end{pmatrix} = hf_{ext}((R_k,y_k),(\omega_{k-1},v_{k-1}) \tag{20}$$

这里 k=1,2，…，N-1，在这种表达中，映射τ可以是指数映射也可以是凯利映射，如下（其中 I_k 表示维度为 k 的单位阵）：

$$\exp(\omega) = \begin{cases} I_3, if\ \omega = 0 \\ I_3 + \dfrac{\sin\|\omega\|}{\|\omega\|}\hat{\omega} + \dfrac{1-\cos\|\omega\|}{\|\omega\|^2}\hat{\omega}^2, if\ \omega \neq 0 \end{cases} \tag{21}$$

$$\mathrm{cay}(\omega)=I_3+\frac{4}{4+\|\omega\|^2}\left(\hat{\omega}+\frac{\hat{\omega}^2}{2}\right) \tag{22}$$

映射 B_τ：R^3->$L(R^3,R^3)$,取决于映射 τ 的选取

$$B_{\exp}(\omega)=\begin{cases} I_3, if\ \omega=0 \\ I_3+(\frac{1-\cos\|\omega\|}{\|\omega\|})\frac{\hat{\omega}}{\|\omega\|}+(1-\frac{\sin\|\omega\|}{\|\omega\|})\frac{\hat{\omega}^2}{\|\omega\|^2}, if\ \omega\neq 0 \end{cases} \tag{23}$$

$$\mathrm{B}_{\mathrm{cay}}(\omega)=\frac{2}{4+\|\omega\|^2}\left(2I_3+\hat{\omega}\right) \tag{24}$$

最后，映射 C_τ：R^6->$L(R^6,R^6)$,取决于组差异映射 τ 的选取

$$\mathrm{C}_{\exp}((\omega,v))=I_6-\frac{1}{2}\left[ad_{(\omega,v)}\right]+\frac{1}{12}\left[ad_{(\omega,v)}\right]^2 \tag{25}$$

或者

$$\mathrm{C}_{\mathrm{cay}}((\omega,v))=\begin{pmatrix} I_3-\frac{1}{2}\hat{\omega}+\frac{1}{4}\omega\omega^T & 0_3 \\ -\frac{1}{2}(I_3-\frac{1}{2}\hat{\omega})\hat{v} & I_3-\frac{1}{2}\hat{\omega} \end{pmatrix} \tag{26}$$

这里

$$[ad_{(\omega,v)}]=\begin{pmatrix} \hat{\omega} & 0_3 \\ \hat{v} & \hat{\omega} \end{pmatrix}$$

为了提高效率，我们建议忽略后面这些矩阵中的二次项，从而生成映射

$$\mathrm{C}_{\mathrm{TLN}}=I_6-\frac{1}{2}\left[ad_{(\omega,v)}\right] \tag{27}$$

这对应于 Boui-rabee 和 Marsden 在 2009 年提出的梯形李-牛顿方案[18]。这种映射既可以代替 C_{exp}，也可以代替 C_{cay}，而不损失离散欧拉-庞加莱方程的二阶精度。积分器更新步骤在表 2.4 中进行了总结，并在附录 B 的伪代码中进行了描述

表 2. 2 积分器更新步骤

给出(R_k; y_k; ω_{k-1}; v_{k-1})

选择映射τ为指数映射或者凯利映射
计算矩阵B_τ和C_τ
使用迭代法求解公式（23），得到角速度速度向量（w_k,v_k）
使用公式（22）更新旋转矩阵和位置坐标(R_{k+1},y_{K+1})

将公式（23）改写为函数的形式，方便下述说明

$$f(\omega_k,v_k)=C_\tau^T(h(\omega_k,v_k))\bullet K\begin{pmatrix}\omega_k\\ v_k\end{pmatrix}-C_\tau^T(-h(\omega_k,v_k))\bullet K\begin{pmatrix}\omega_{k-1}\\ v_{k-1}\end{pmatrix}-hf_{ext}((R_k,y_k),(\omega_{k-1},v_{k-1})$$

下面给出牛顿迭代求解非线性方程组的步骤

a 选定近似初始值ξ_0，计算ξ_0，本算法先执行欧拉步显示计算;

b 进行修正解，计算$\xi_{t+1}=\xi_t$-Jacobian*f(ξ_t)，（Jacobian 为雅克比矩阵，即对每个变量求偏导的结果，这个方程包括 6 个变量，由于角速度和速度都是三维的变量，这个求导的过程是计算机不能完成的，附录 C 给出了雅克比矩阵的计算结果），迭代一次得到新的近似值ξ_{t+1}，并计算 f(ξ_{t+1});

c 如果 f(ξ_{t+1})<ε（ε为预先给出的精度），则过程收敛，终止迭代，并取$\xi=\xi_{t+1}$为所求根的近似值，否则 t 增加 1 再转 b 步骤继续计算，如果迭代次数超过预先指定的次数 N。但仍达不到精度要求，则认为牛顿法解方程失败，可以根据需要选择仿真结束与否。

将整个算法写成如图 2.1 的流程图。

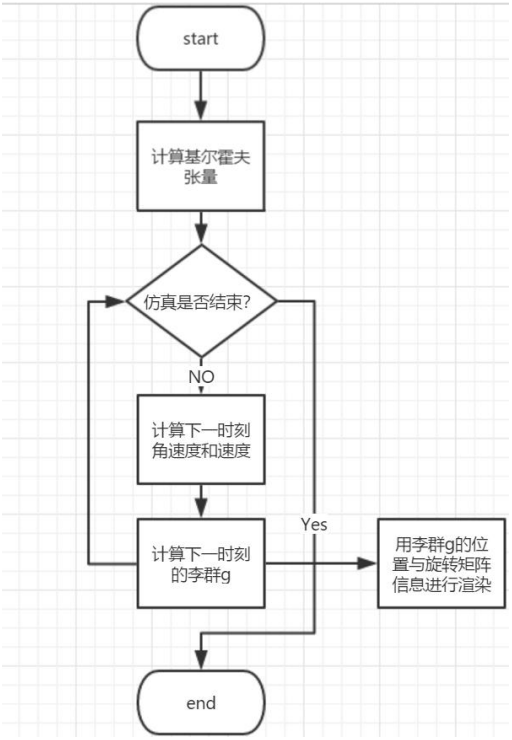


图 2. 1 主要算法的流程图

2.5 本章小结

本章主要对单刚体运动仿真的理论基础进行详细介绍，包括基本假设，算法核心思想，预计算过程及仿真过程，为下章介绍刚体外力的计算做了铺垫。

3 外力计算

在仿真运动求解过程中（2.4），需要知道刚体所受合外力以及合外力矩，下面对刚体所受的几种力进行讨论。

3.1 重力与浮力

重力重力加速度 g 作用于物体质量（重力）和流体质量（浮力）。这两个重力加速度 g 作用于物体质量（重力）和流体质量（浮力）。力都平行于 g，但有不同的作用点：浮力作用于流体 c_{ρ} 流离失的中心，而重力作用于物体的质心 C_m。如果物体材料不均匀分布，它们可能会不同。这两种力是:

$$f_{\text{gravity}} = mg, f_{bouyancy} = -\rho volBg$$

，其中 volB 是刚体的体积。

为了计算在物体坐标系下产生的转矩和力（通过式（13）），我们需要在空间坐标系下表达两个中心

$$\mathrm{c}_{\text{gravity}} = R\mathrm{c}_m + y, c_{bouyancy} = Rc_{\rho} + y$$

因此我们可得物体坐标系下所受重力和浮力的作用

$$\begin{pmatrix} t_g \\ f_g \end{pmatrix} = \begin{pmatrix} (m_{\rho}c_{\rho} - mc_m) \times R^t g \\ (m - m_{\rho}) R^t g \end{pmatrix}$$

这使我们不仅能够模拟水下的运动，还能够模拟许多有趣的物体的运动，例如像降落伞一样的物体或气球。

3.2 阻力

在某些情况下，可能需要应用某种流体阻力。由于涡流或涡度的粘性扩散，物体会失去一部分能量给向周围的流体。更复杂的是，涡流意味着我们不仅要知道物体的流速状态，还要知道物体运动的整个历史以及它对周围流体的影响。因此，即使仿真过程有

一定的“时间延迟”，也可以从流体中得到一些反馈。

在这里，我们建立了一个非常简单的阻力模型，它只考虑固定厚度边界层中的粘性摩擦：在物体边界上的每个点上，我们施加一个力，这个力与从流体到物体内部的速度差成正比。然后施加到物体上，力矩 t_v 和阻力 d_v 表示如下

$$t_v = \nu \int_{\partial B} z \times (grad\phi(z) - (\omega \times z + v))dA \quad (25)$$
$$d_v = \nu \int_{\partial B} grad\phi(z) - (\omega \times z + v)dA$$

t_v 和阻力 d_v 与角速度和速度线性相关

$$\begin{pmatrix} t_v \\ d_v \end{pmatrix} = \mathrm{D} \begin{pmatrix} \omega \\ v \end{pmatrix}$$

3.3 本章小结

本章主要介绍了单刚体所受合外力的计算，包括重力与浮力以及阻力。在下一章节中，将通过实验说明该算法在不同模型的运行结果，论述该算法的可行性。

4　仿真结果

选取了很多模型进行了实验，对真空中和水下刚体运动进行了对照，这里列举了两个典型模型。

4.1 圆盘仿真结果

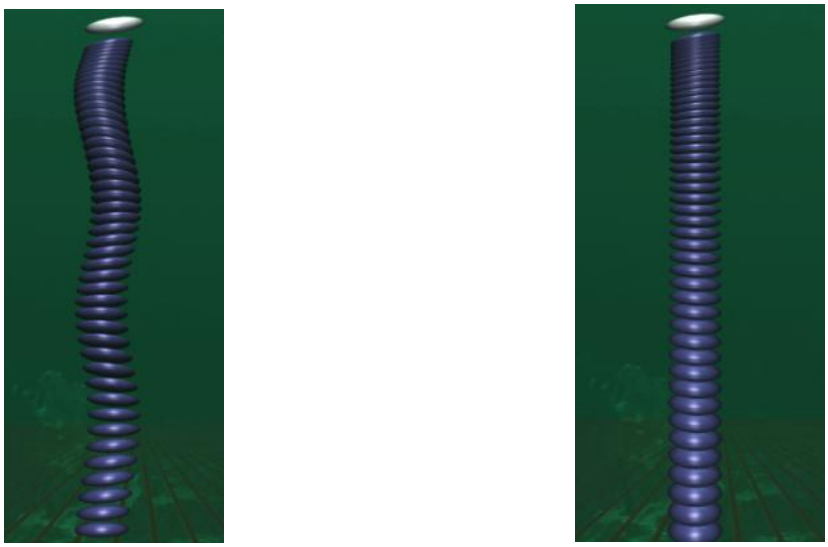


图 4. 1 水平静止的圆盘水下运动仿真结果（左）水平静止的圆盘真空中运动仿真结果（右）

4.2 螺旋桨模型仿真结果

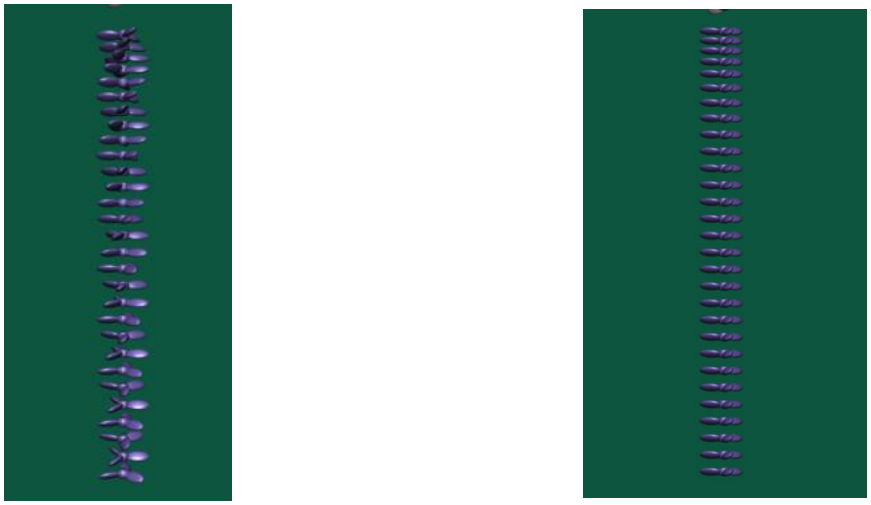


图 4. 2 螺旋桨模型水下运动仿真结果（左）螺旋桨模型真空中运动仿真结果（左）

4.3 斯坦福兔子模型仿真结果

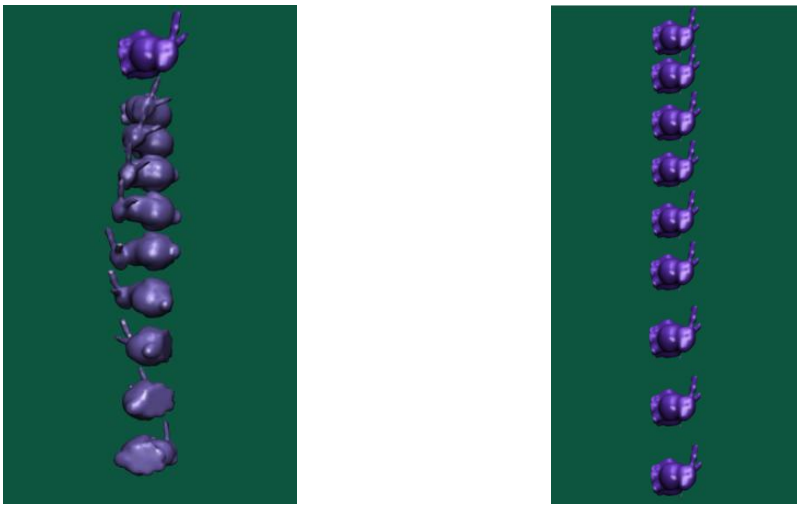


图 4. 3 斯坦福兔子模型水下运动仿真结果（左）斯坦福兔子模型真空中运动仿真结果（左）

4.4 结论

可以看出水下刚体的运动与真空中刚体的运动有着明显的区别，水下运动的圆盘开始晃动，而真空中运动的圆盘不会受到任何扰动垂直地下落；更有趣的是，水下的螺旋桨模型开始旋转，而这种情况在真空中根本不会发生，真空中螺旋桨模型也是垂直地下落。

4.5 本章小结

本章主要介绍了介绍仿真结果，证明了算法的正确性。下一章将对基尔霍夫张量进行一些解释。

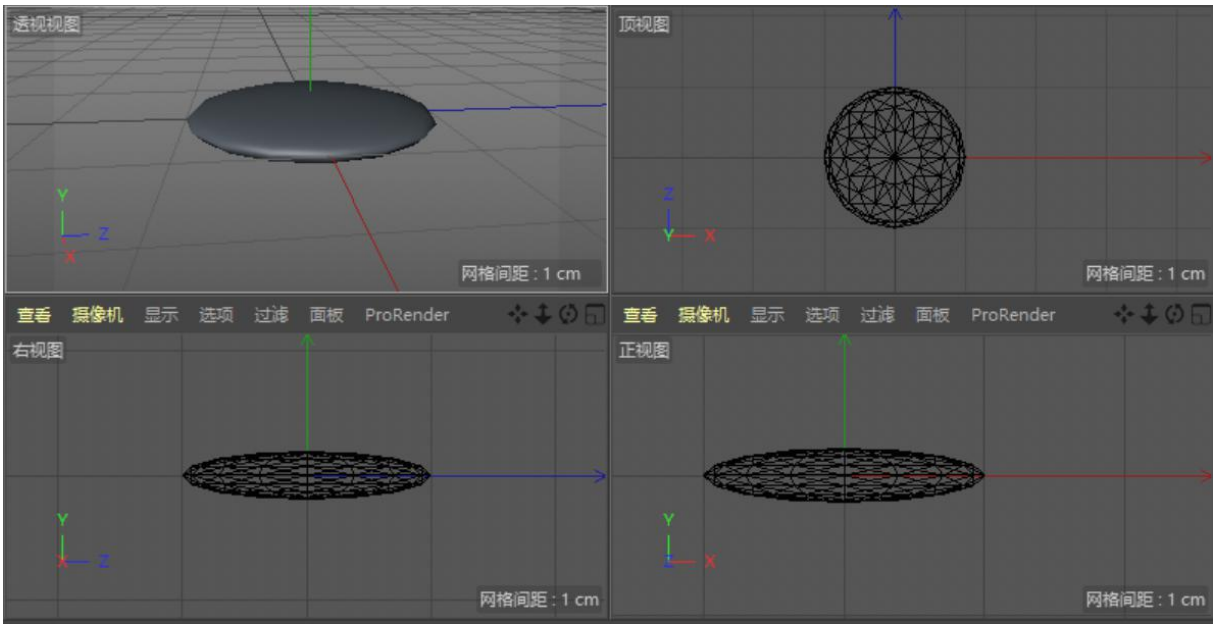
5　基尔霍夫张量的解释

与标准质量惯性张量相比，基尔霍夫张量是一种更一般的形式，体现了各向异性质量和螺旋桨张量在直线运动与转动运动之间的耦合。

5.1 各向异性质量

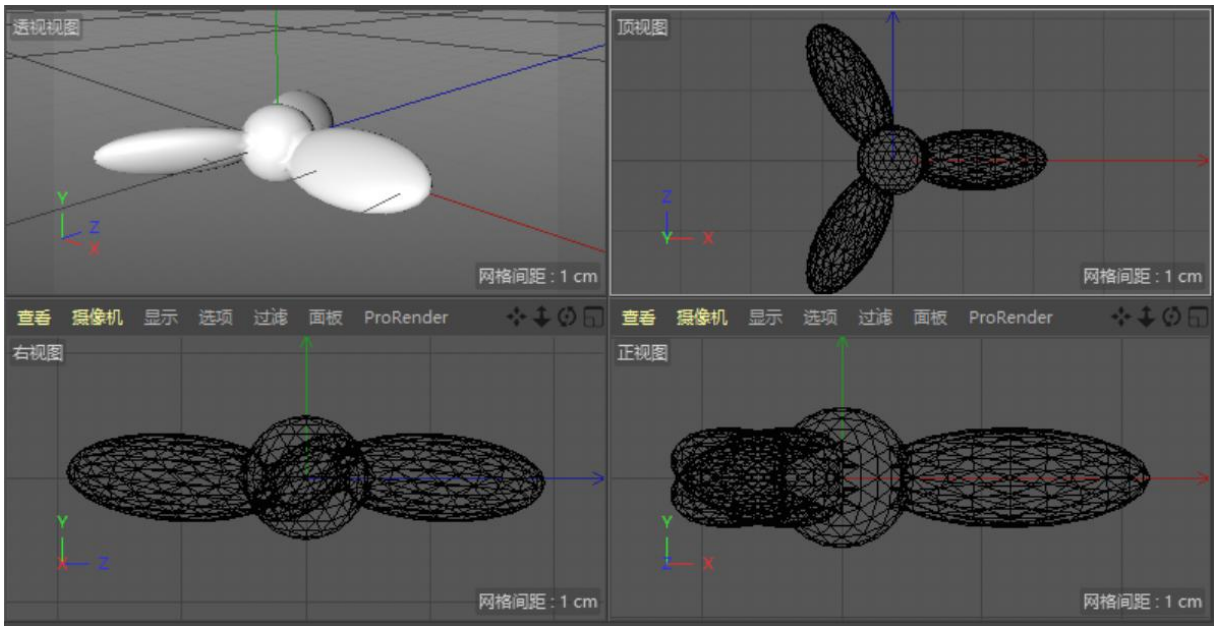
浸入液体中的物体的质量张量不再像真空中的物体一样是单位矩阵的倍数。这意味着以相同的速度在不同的方向进行平移运动可以导致不同的动能值。例如，考虑一个绕垂直轴旋转对称的平面椭球圆盘（如图 5.1）。这样，圆盘的垂直平移就会产生比水平平移引起更多的流体运动（因此动能更大）。这种效果会导致下沉物体的典型摆动，而在真空中则完全不存在这种摆动，如图 4.1（左）所示。

图 5. 1 平面椭球圆盘模型



5.2 螺旋桨张量

图 5. 2 螺旋桨模型



考虑我们实验的螺旋桨模型（见图 5.1），即使我们静止地把它放入水中，依旧可以旋转。这个螺旋桨是一个三倍对称的模型，那么重力和浮力对它的力矩为 0。根据公式 4 和公式 5，子矩阵 P 一定不为 0，P 是引发模型旋转的根本原因，所以我们把它称为螺旋桨张量。

相对于真空中的物体，浸入液体中的物体的平移和旋转运动是不可分离（耦合）的。下面解释了为什么我们把它称作螺旋桨张量，我们看看一个形状如图 4 所示的物体。我们假设物体的质量分布符合三角对称，假设开始的运动状态对称轴以及线速度和角速度是垂直的。设 a 表示沿 y 轴负方向的单位向量，这意味着ω=ωa， v = va。那么这些条件在整个运动中都保持不变，虽然 ω 和 v 可能取决于时间。由于对称的原因，a 是 J 和 P 的特征向量：$J_a = \mu J_a$， $P_a = \mu P_a$。因为重力和浮力对物体的扭矩消失了，所以角动量保持不变。如果我们从静止开始，角动量一直保持为 0，由运动方程我们可以获得 $\frac{\omega}{v} = -\frac{\mu_P}{\mu_J}$。

因此，螺旋桨的每个点都会以螺旋形的速度运动，并且速度会越来越快。这是我们在实验和模拟中观察到的（图 5.2，左）。对于不太特定的形状，螺旋桨张量 P 可以是非零的。比如图 4.3 的斯坦福兔子模型即使在一开始是静止的，后面开始旋转。

5.3 本章小结

本章对基尔霍夫张量做了一些解释，包括各向异性质量以及螺旋桨张量。下一章将

对本论文进行总结并进行展望。

6 结论与展望

本论文证明了在低雷诺数流动的层流中基尔霍夫方程是对刚体进行真实模拟的有力工具。该方法在中等速度的物体浸入一个近似静止的流体状态下的模拟中非常突出，因为它的效率较为理想（与真空中物体运动有着相同的时间复杂度）而且与实际实验的有着大视觉一致性。

由于时间和精力有限，我还没用把基尔霍夫张量集成到物理引擎中。在以后的时间里，我们想把它集成到普遍使用的开源物理引擎 bullet 中。由于合并后的动能（包括物体和流体运动）关于物体速度（ω，v）仍然是一个恒定的二次型函数，只需要对现有的物理引擎轻微地进行修改，很容易地就可以扩展我们的方法，实时模拟浸入流体的物体。

另外，在物理仿真领域中，实际的代码运行大都是用 GPU 来进行计算的，本论文还没有实现该算法在 GPU 的运算，这也是未来的一个工作目标。

基尔霍夫的方法在原则上也可以扩展到多刚体运动。然而，每个物体的基尔霍夫张量的流体部分 K_F 不再是恒定的，而是依赖于所有其他物体的姿态。在交互式速率下实现这样的模拟将具有挑战性。由于周围的流体的存在，我们建议忽略物体之间的长期相互作用，并像在真空中一样处理碰撞。因此，我们根本不修改求解器代码。相反，我们只让碰撞求解器看到基尔霍夫张量的惯性部分和平均质量部分。

在未来的研究中，我们想探讨当基尔霍夫方程也用于约束求解器时，多体系统是如何工作的。这将需要一些努力，因为现有的代码通常依赖于质量是标量的事实，而旋转运动和线性运动不是耦合的。

另一个有趣的问题是如何在运动方程中加入背景速度场的影响。与力场相比，这将允许对整体运动特性进行更复杂的控制。

参考文献

[1] CARLSON, M., MUCHA, P. J., AND TURK, G. 2004. Rigid fluid[A]. ACM Trans. Graph. 23, 3

[2] Takahashi T.,Fujii H.,Kunimatsu A.,et al.Realistic Animation of Fluid with Splash and foam[J].computer graphics forum，2003,22（3）：391-400

[3] TREUILLE, A., LEWIS, A., AND POPOVI´ C, Z. 2006. Model reduction for real-time fluids[C]. Proc. ACM/SIGGRAPH Conf. 25, 3, 826–834.

[4] OZGEN, O., KALLMANN, M., RAMIREZ, L. E., AND COIMBRA, C. F. 2010. Underwater cloth simulation with fractional derivatives[A]. ACM Trans. Graph. 29, 3.

[5] BARAFF, D. 1993. Non-penetrating rigid body simulation[C]. In State of the Art Reports, Eurographics.

[6] KOBILAROV, M., CRANE, K., AND DESBRUN, M. 2009. Lie group integrators for animation and control of vehicles[A]. ACM Trans. Graph. 28, 2.

[7] HOLMES, P., JENKINS, J., AND LEONARD, N. E. 1998. Dynamics of the Kirchhoff Equations I : Coincident Centers of Gravity and Buoyancy[A]. Physica D 118.

[8] BOU-RABEE, N. 2007. Hamilton-Pontryagin Integrators on Lie Groups[D]. PhD thesis, Caltech. CARLSON, M., MUCHA, P. J., AND TURK, G. 2004. Rigid fluid. ACM Trans. Graph. 23, 3.

[9] NORDKVIST, N., AND SANYAL, A. K. 2010. A Lie group variational integrator for rigid body motion in SE(3) with applications to underwater vehicle dynamics[C]. In Proc. IEEE CDC.

[10] VANKERSCHAVER, J., KANSO, E., AND MARSDEN, J. 2009. The geometry and dynamics of interacting rigid bodies and point vortices[J]. The Journal of Geometric Mechanics 1, 2, 223–266.

[11] SHASHIKANTH, B. N., SHESHMANI, A., DAVID, S., AND JERROLD, K. 2008. Hamiltonian structure for a neutrally buoyant rigid body interacting with N vortex rings of arbitrary shape : the case of arbitrary smooth body shape[J]. 37–64.

[12] NAIR, S., AND KANSO, E. 2007. Hydrodynamically coupled rigid bodies[J]. Journal of Fluid Mechanics 592, 393–411.

[13] NEWMAN, J. N. 1977. Marine hydrodynamics[M]. MIT Press.

[14] KANSO, E., MARSDEN, J. E., ROWLEY, C. W., MELLI-HUBER, J. B., AND

NEWTON, C. P. K. 2005. Locomotion of articulated bodies in a perfect fluid[J]. Nonlinear Science 15, 255–289.

[15] LEE, T., LEOK, M., AND MCCLAMROCH, N. H. 2009. Dynamics of connected rigid bodies in a perfect fluid. In Proc. ACC 2009, [Online]. Available: http://arxiv.org/abs/0809.1488.

[16] FIELD, S., KLAUS, M., AND MOORE, M. 1997. Chaotic dynamics of falling disks. Nature 388, July.

[17] PESAVENTO, U., AND WANG, Z. J. 2004. Falling paper: Navierstokes solutions, model of fluid forces, and center of mass elevation. Phys. Rev. Lett. 93, 14, 144501.

[18] ZHONG, H., CHEN, S., AND LEE, C. 2011. Experimental study of freely falling thin disks: Transition from planar zigzag to spiral. Physics of Fluids 23, 1.

[19] Steffen, W. 2012. Underwater Rigid Body Dynamics[A]. in ACM TOG 31(4).

[20] KIRCHHOFF, G. R. 1870. [On the motion of a solid of revolution in a fluid][M]. Journal fur die Reine and Angewandte Mathematik 71. (in German).

[21] VAN OOSTEROM, A., AND STRACKEE, J. 1983. The Solid Angle of a Plane Triangle[J]. IEEE Trans. Biomed. Eng. BME-30, 2.

[22] David E.2002. Polyhedral Mass Properties (Revisited)[J].Geometric Tools, LLC

[23] BOU-RABEE, N. AND MARSDEN, J. E. 2009. Hamilton-Pontryagin integrators on Lie groups Part I: Introduction and structure-preserving properties[J]. To appear in Foundations of Computational Mathematics

致 谢

在这之前，我对计算机图形学的物理仿真领域还是比较陌生，通过对课题的研究，我收获了科学的研究态度，接触到了前沿的研究工作。我能获得今天的成绩一方面归因于自己的努力，更大程度上和许多人的帮助是不可分的。

在这里，首先要感谢我的导师周世哲副教授，本文是在他的指导下完成的。感谢中科院软件所何小伟副研究员，是他让我接触到了新的研究方向，给我全新的课题，并提供给我优良的工作环境，让我能顺利完成毕设工作。

感谢大学期间教过我的其他老师，杨科华老师，赵欢老师，黎文伟老师，彭鹏老师，肖懿老师，谭光华老师，周四望老师，潘华伟老师等等。我是一个爱问问题的学生，感谢他们在课间时间不厌其烦地给我解答疑惑，这是我在大学四年期间最为感动的事情。感谢他们无私的奉献，让我打下了坚实的理论基础。

感谢我的朋友马文锐同学，从她身上我学到了踏实认真，勤奋刻苦的学习态度，让我改正了许多学习中的不良习惯，这对我以后的学习研究很有帮助。感谢我的室友许靓，王卓娅，吴宇晗同学，她们在学习和生活上都给了我许多帮助。感谢数媒 1501 班全体同学，他们给了我一个融洽上进的班集体，使我有了一个难忘的大学回忆。

感谢我的父母，他们一直以来都在背后默默地支持我，鼓励我。是他们无时无刻的关心与支持使我一直饱含自信，不畏困难与挫折，我只有更加努力才能回报他们的恩情。

附录

附录 A 计算惯性张量

```
MACRO Subexpressions(w0,w1,w2,f1,f2,f3,g0,g1,g2) {
    temp0 = w0+w1; f1 = temp0+w2; temp1 = w0*w0; temp2 = temp1+w1*temp0;
     f2 = temp2+w2*f1; f3 = w0*temp1+w1*temp2+w2*f2;
     g0 = f2+w0*(f1+w0); g1 = f2+w1*(f1+w1); g2 = f2+w2*(f1+w2);
}
void Compute (Point p[], int tmax, int index[], Real& mass, Point& cm, Matrix& inertia) {
constant Real mult[10] = {1/6,1/24,1/24,1/24,1/60,1/60,1/60,1/120,1/120,1/120}; Real intg[10]
= {0,0,0,0,0,0,0,0,0,0}; // order: 1, x, y, z, x^2, y^2, z^2, xy, yz, zx
for (t = 0; t < tmax; t++) {
// get vertices of triangle t
    i0 = index[3*t]; i1 = index[3*t+1]; i2 = index[3*t+2];
    x0 = p[i0].x; y0 = p[i0].y; z0 = p[i0].z;
    x1 = p[i1].x; y1 = p[i1].y; z1 = p[i1].z;
    x2 = p[i2].x; y2 = p[i2].y; z2 = p[i2].z;
// get edges and cross product of edges
    a1 = x1-x0; b1 = y1-y0; c1 = z1-z0; a2 = x2-x0; b2 = y2-y0; c2 = z2-z0;
    d0 = b1*c2-b2*c1; d1 = a2*c1-a1*c2; d2 = a1*b2-a2*b1;
// compute integral terms
    Subexpressions(x0,x1,x2,f1x,f2x,f3x,g0x,g1x,g2x);
    Subexpressions(y0,y1,y2,f1y,f2y,f3y,g0y,g1y,g2y);
    Subexpressions(z0,z1,z2,f1z,f2z,f3z,g0z,g1z,g2z);
// update integrals
    intg[0] += d0*f1x; intg[1] += d0*f2x; intg[2] += d1*f2y; intg[3] += d2*f2z;   intg[4] +=
d0*f3x; intg[5] += d1*f3y; intg[6] += d2*f3z;
    intg[7] += d0*(y0*g0x+y1*g1x+y2*g2x); intg[8] += d1*(z0*g0y+z1*g1y+z2*g2y);
    intg[9] += d2*(x0*g0z+x1*g1z+x2*g2z);
}
for (i = 0; i < 10; i++)
    intg[i] *= mult[i];
```

```
mass = intg[0];
// center of mass
cm.x = intg[1]/mass; cm.y = intg[2]/mass; cm.z = intg[3]/mass;
// inertia tensor relative to center of mass
inertia.xx = intg[5]+intg[6]-mass*(cm.y*cm.y+cm.z*cm.z);
inertia.yy = intg[4]+intg[6]-mass*(cm.z*cm.z+cm.x*cm.x);
inertia.zz = intg[4]+intg[5]-mass*(cm.x*cm.x+cm.y*cm.y);
inertia.xy = -(intg[7]-mass*cm.x*cm.y);
inertia.yz = -(intg[8]-mass*cm.y*cm.z);
inertia.xz = -(intg[9]-mass*cm.z*cm.x);
}
```

附录 B 结构保持积分器和动量保持积分器伪代码

积分器是隐式的，通过求解动力学方程 Unconstr_Dyn 获得下一时刻的角速度速度向量，外力通过 f_{ext} 函数求解

Function [ξ_k; g_{k+1}] = HolInt(ξ_{k-1}; g_k)

% 基尔霍夫张量矩阵

$K = \begin{matrix} J & P^t \\ P & M \end{matrix}$

% 初始步骤(例如：欧拉法)

$\xi_k = \xi_{k-1} + h \cdot \text{inv}(K) \cdot (\text{se3\_ad}(h \cdot \xi_k)t \cdot K \cdot \xi_{k-1} + f_{ext}(\xi_{k-1}; g_k) + F(\gamma_k)u_k)$

% 隐式动力学求解（例如：牛顿迭代法)

$\xi_k = \text{fsolve}(\text{Unconstr\_Dyn}; \xi_k; \xi_{k-1}; g_k)$

% 显示更新物体的姿态矩阵

$g_{k+1} = g_k \cdot \text{se3\_cay}(h\xi_k)$

Function f = Unconstr_Dyn(ξ_k; ξ_{k-1}; g_k)

f = se3_DEP(ξ_k; ξ_{k-1}; g_k) - $hF(\gamma_k)u_k$

Function f = se3_DEP(ξ_k; ξ_{k-1}; g_k)

f = se3_CTLN($h \cdot \xi_k)^t \cdot K \cdot \xi_k$- se3 CTLN($-h \cdot \xi_{k-1})t \cdot K \cdot \xi_{k-1}$-$h \cdot f_{ext}(g_k; \xi_{k-1})$

Function f = so3_cay(ω)

$\hat{\omega}$ = so3_ad(ω)

f = eye(3) + 4=(4 + norm(ω)2) · $(\hat{\omega} + \hat{\omega} \cdot \hat{\omega}/2)$

Function f = se3_cay(ξ)

ω= ξ(1 : 3)

v =ξ(4 : 6)

B = 2/(4 + norm(h ω)²) · (2 · eye(3) + so3_ad(h ω))

f = so3_cay(h ω) hB· v

0 1

Function f = se3_CTLN (ξ)

f = eye(6) -0.5 · se3 ad (ξ)

Function f = se3 ad(ξ)

ω = ξ(1 : 3)

v = ξ(4 : 6)

f = so3_ad(ω) zeros(3)

so3_ad(v) so3_ad(ω)

Function f = so3_ad(ω)

f =

0 - ω(3) ω(2)

ω(3) 0 - ω(1)

- ω(2) ω(1) 0

附录 C 雅克比矩阵的计算

```
Jacobian(0, 0) = K(0, 0) + delta_t * 0.5*w(2)*K(1, 0) - delta_t * 0.5*w(1)*K(2, 0);
    Jacobian(0, 1) = -delta_t * 0.5*w(0)*K(2, 0) - delta_t * 0.5*w(2)*K(2, 2) //第一行第三行对 w（1）求导
        - delta_t * 0.5*K(2, 1)*w(1)+//前导后不导
        K(0, 1) + delta_t * 0.5*w(2)*K(1, 1) - delta_t * 0.5*w(1)*K(2, 1);//前不导后导
    Jacobian(0, 2) = delta_t * 0.5*w(0)*K(1, 0)+ delta_t * 0.5*w(1)*K(1, 1)//第一行第二行对 w（2）求导
        + delta_t * 0.5*K(1, 2)*w(2) +//前导后不导
        K(0, 2) + delta_t * 0.5*w(2)*K(1, 2) - delta_t * 0.5*w(1)*K(2, 2);//前不导后导
    Jacobian(0, 3) = 0;
    Jacobian(0, 4) = 0;
```

```
Jacobian(0, 5) = 0;

Jacobian(1, 0) = delta_t * 0.5*K(2, 0)*w(0) //前导后不导
    -delta_t * 0.5*w(2)*K(0, 0) + K(1, 0) + delta_t * 0.5*w(0)*K(2, 0)//前不导后导
    +0.5*delta_t*K(2,1)*w(1)+0.5*delta_t*K(2, 2)*w(1);//第二行第三行对 w（0）求导
Jacobian(1, 1) = -delta_t * 0.5*w(2)*K(0, 1) + K(1, 1) + delta_t * 0.5*w(0)*K(2, 1);
Jacobian(1, 2) = -delta_t * 0.5* K(0,2) *w(2)   //前导后不导
    -delta_t * 0.5*w(2)*K(0, 2) + K(1, 2) + delta_t * 0.5*w(0)*K(2, 2)//前不导后导
    -delta_t * 0.5*w(0)*K(0, 0)-delta_t * 0.5*w(1)*K(0, 1);//第 1 行第二行对 w（2）求
导
Jacobian(1, 3) = 0;
Jacobian(1, 4) = 0;
Jacobian(1, 5) = 0;

Jacobian(2, 0) = -delta_t * 0.5*K(1, 0)*w(0)//前导后不导
    +delta_t * 0.5*w(1)*K(0, 0) - delta_t * 0.5*w(0)*K(1, 0) + K(2, 0) //前不导后导
    - delta_t * 0.5*w(1)*K(1, 1) - delta_t * 0.5*w(2)*K(1, 2);//后两行对 w（0）
Jacobian(2, 1) = delta_t * 0.5*K(0, 1)*w(1)//前导后不到
    + delta_t * 0.5*w(1)*K(0, 1) - delta_t * 0.5*w(0)*K(1, 1) + K(2, 1)//前不导后导
    + delta_t * 0.5*K(0, 0)*w(0)+ delta_t * 0.5*K(0, 2)*w(2);//一三行对 w(1)
Jacobian(2, 2) = delta_t * 0.5*w(1)*K(0, 2) - delta_t * 0.5*w(0)*K(1, 2) + K(2, 2);
Jacobian(2, 3) = 0;
Jacobian(2, 4) = 0;
Jacobian(2, 5) = 0;

Jacobian(3, 0) = delta_t * 0.5*v(2)*K(1, 0) - delta_t * 0.5*v(1)*K(2, 0) + K(3, 0) +
    delta_t * 0.5*w(2)*K(4, 0) - delta_t * 0.5*w(1)*K(5, 0);
Jacobian(3, 1) = -delta_t * 0.5*w(0)*K(5, 0) - delta_t * 0.5*w(2)*K(5, 2)
    - delta_t * 0.5*v(0)*K(5, 3) - delta_t * 0.5*v(1)*K(5, 4) - delta_t * 0.5*v(2)*K(5, 5)
```

```
        //0，2，3，4，5 对 w(1)求导
        + delta_t * 0.5*v(2)*K(1, 1) - delta_t * 0.5*v(1)*K(2, 1) + K(3, 1) +
        delta_t * 0.5*w(2)*K(4, 1) - delta_t * 0.5*w(1)*K(5, 1)//前不导后导
        - 0.5*delta_t*K(5, 1)*w(1);
        //前导后不导
    Jacobian(3, 2) = delta_t * 0.5*w(0)*K(4, 0)+ delta_t * 0.5*w(1)*K(4, 1)
        + delta_t * 0.5*v(0)*K(4, 3) + delta_t * 0.5*v(1)*K(4, 4) + delta_t * 0.5*v(2)*K(4,
5)
        //0，1，3，4，5 对 w(2)求导
        +delta_t * 0.5*v(2)*K(1, 2) - delta_t * 0.5*v(1)*K(2, 2) + K(3, 2) +
        delta_t * 0.5*w(2)*K(4, 2) - delta_t * 0.5*w(1)*K(5, 2)//前不导后导
        + delta_t * 0.5*K(4, 2)*w(2)
        ;//前导后不导
    Jacobian(3, 3) =
        //0，1，2，4，5 对 v(0)求导为 0
        +delta_t * 0.5*v(2)*K(1, 3) - delta_t * 0.5*v(1)*K(2, 3) + K(3, 3) +
        delta_t * 0.5*w(2)*K(4, 3) - delta_t * 0.5*w(1)*K(5, 3);//前不导后导
    //前导后不导没有 v(0
    Jacobian(3, 4) = -delta_t * 0.5*K(2, 0)*w(0) - delta_t * 0.5*K(2, 1)*w(1)
        - delta_t * 0.5*K(2, 2)*w(2) - delta_t * 0.5*K(2, 3)*v(0) - delta_t * 0.5*K(2, 5)*v(2)
        //0，1，2，3，5 对 v(1)求导
        + delta_t * 0.5*v(2)*K(1, 4) - delta_t * 0.5*v(1)*K(2, 4) + K(3, 4) +
        delta_t * 0.5*w(2)*K(4, 4) - delta_t * 0.5*w(1)*K(5, 4)//前不导后导
        - delta_t * 0.5*K(2, 4)*v(1);
    //前导后不导
    Jacobian(3, 5) = delta_t * 0.5*K(1, 0)*w(0)+ delta_t * 0.5*K(1, 1)*w(1)
        + delta_t * 0.5*K(1, 2)*w(2) + delta_t * 0.5*K(1, 3)*v(0) + delta_t * 0.5*K(1,
4)*v(1)
        //0，1，2，3，4 对 v(2)求导
```

```
        +delta_t * 0.5*v(2)*K(1, 5) - delta_t * 0.5*v(1)*K(2, 5) + K(3, 5) +
        delta_t * 0.5*w(2)*K(4, 5) - delta_t * 0.5*w(1)*K(5, 5)//前不导后导
        +delta_t * 0.5*K(1, 5)*v(2);
    //前导后不导

    Jacobian(4, 0) = delta_t * 0.5*K(5, 1)*w(1) + delta_t * 0.5*K(5, 2)*w(2)
        + delta_t * 0.5*K(5, 3)*v(0) + delta_t * 0.5*K(5, 4)*v(1) + delta_t * 0.5*K(5,
5)*v(2)
        //1，2，3，4，5 对 w(0)求导
        - delta_t * 0.5*v(2)*K(0, 0) + delta_t * 0.5*v(0)*K(2, 0) -
        delta_t * 0.5*w(2)*K(3, 0) + K(4, 0) + delta_t * 0.5*w(0)*K(5, 0)//前不导后导
        + delta_t * 0.5*K(5, 5)*w(0);
        //前导后不导
    Jacobian(4, 1) =
        //0，2，3，4，5 对 w(1)求导为 0
        -delta_t * 0.5*v(2)*K(0, 1) + delta_t * 0.5*v(0)*K(2, 1) -
        delta_t * 0.5*w(2)*K(3, 1) + K(4, 1) + delta_t * 0.5*w(0)*K(5, 1);//前不导后导
        //前导后不导求导为 0
    Jacobian(4, 2) = -delta_t * 0.5*K(3, 0)*w(0) - delta_t * 0.5*K(3, 1)*w(1)
        - delta_t * 0.5*K(3, 3)*v(0) - delta_t * 0.5*K(3, 4)*v(1) - delta_t * 0.5*K(3, 5)*v(2)
        //0，1，3，4，5 对 w(2)求导
        - delta_t * 0.5*v(2)*K(0, 2) + delta_t * 0.5*v(0)*K(2, 2) -
        delta_t * 0.5*w(2)*K(3, 2) + K(4, 2) + delta_t * 0.5*w(0)*K(5, 2)//前不导后导
        - delta_t * 0.5*K(3, 2)*w(0);
        //前导后不导
    Jacobian(4, 3) = delta_t * 0.5*K(2,0)*w(0)+ delta_t * 0.5*K(2, 1)*w(1)
        + delta_t * 0.5*K(2, 2)*w(2) + delta_t * 0.5*K(2, 4)*v(1) + delta_t * 0.5*K(2,
5)*v(2)
        //0，1，2，4，5 对 v(0)求导
```

-delta_t * 0.5*v(2)*K(0, 3) + delta_t * 0.5*v(0)*K(2, 3) -

delta_t * 0.5*w(2)*K(3, 3) + K(4, 3) + delta_t * 0.5*w(0)*K(5, 3)//前不导后导

+delta_t * 0.5*K(2, 3)*v(0);

//前导后不导

Jacobian(4, 4) =

//0，1，2，3，5 对 v(1)求导为 0

-delta_t * 0.5*v(2)*K(0, 4) + delta_t * 0.5*v(0)*K(2, 4) -

delta_t * 0.5*w(2)*K(3, 4) + K(4, 4) + delta_t * 0.5*w(0)*K(5, 4)//前不导后导

;

////前导后不导为 0

Jacobian(4, 5) = -delta_t * 0.5*K(0, 0)*w(0) - delta_t * 0.5*K(0, 1)*w(1)

- delta_t * 0.5*K(0, 2)*w(2) - delta_t * 0.5*K(0, 3)*v(0) - delta_t * 0.5*K(0, 4)*v(1)

//0，1，2，3，4 对 v(2)求导

- delta_t * 0.5*v(2)*K(0, 5) + delta_t * 0.5*v(0)*K(2, 5) -

delta_t * 0.5*w(2)*K(3, 5) + K(4, 5) + delta_t * 0.5*w(0)*K(5, 5)//前不导后导

- delta_t * 0.5*K(0, 5)*v(2);

//前导后不导

Jacobian(5, 0) = -delta_t * 0.5*K(4, 1)*w(1) - delta_t * 0.5*K(4, 2)*w(2)

- delta_t * 0.5*K(4, 3)*v(0) - delta_t * 0.5*K(4, 4)*v(1) - delta_t * 0.5*K(4, 5)*v(2)

//1，2，3，4 ,5 对 w(0)求导

+ delta_t * 0.5*v(1)*K(0, 0) - delta_t * 0.5*v(0)*K(1, 0) +

delta_t * 0.5*w(1)*K(3, 0) - delta_t * 0.5*w(0)*K(4, 0) + K(5, 0)//前不导后导

- delta_t * 0.5*K(4, 0)*w(0);

//前导后不到

Jacobian(5, 1) = delta_t * 0.5*w(1)*K(3, 0)*w(0) + delta_t * 0.5*w(1)*K(3, 2)*w(2)

+ delta_t * 0.5*w(1)*K(3, 3)*v(0) + delta_t * 0.5*w(1)*K(3, 4)*v(1) + delta_t *

0.5*w(1)*K(3, 5)*v(2)

```
        //0，2，3，4 ,5 行对 w(1)求导
        + delta_t * 0.5*v(1)*K(0, 1) - delta_t * 0.5*v(0)*K(1, 1) +
        delta_t * 0.5*w(1)*K(3, 1) - delta_t * 0.5*w(0)*K(4, 1) + K(5, 1)//前不导后导
        + delta_t * 0.5*w(1)*K(3, 1)*w(1);
    //前导后不导

    Jacobian(5, 2) =
        //0，1，3，4 ,5 行对 w(2)求导为 0
        delta_t * 0.5*v(1)*K(0, 2) - delta_t * 0.5*v(0)*K(1, 2) +
        delta_t * 0.5*w(1)*K(3, 2) - delta_t * 0.5*w(0)*K(4, 2) + K(5, 2);//前不导后导
    //前导后不导为 0
    Jacobian(5, 3) = -delta_t * 0.5*K(1, 0)*w(0) - delta_t * 0.5*K(1, 1)*w(1)
        - delta_t * 0.5*K(1, 2)*w(2) - delta_t * 0.5*K(1, 4)*v(1) - delta_t * 0.5*K(1,5)*v(2)
        //0，1，2，4 ,5 行对 v(0)求导为 0
        +delta_t * 0.5*v(1)*K(0, 3) - delta_t * 0.5*v(0)*K(1, 3) +
        delta_t * 0.5*w(1)*K(3, 3) - delta_t * 0.5*w(0)*K(4, 3) + K(5, 3)//前不导后导
    -delta_t * 0.5*K(1, 3)*v(0);
    //前导后不导
    Jacobian(5, 4) = delta_t * 0.5*K(0, 0)*w(0)+ delta_t * 0.5*K(0, 1)*w(1)
        + delta_t * 0.5*K(0, 2)*w(2) + delta_t * 0.5*K(0, 3)*v(0) + delta_t * 0.5*K(0,
4)*v(2)
        //0，1，2，3 ,5 行对 v(1)求导
        +delta_t * 0.5*v(1)*K(0, 4) - delta_t * 0.5*v(0)*K(1, 4) +
        delta_t * 0.5*w(1)*K(3, 4) - delta_t * 0.5*w(0)*K(4, 4) + K(5, 4);//前不导后导
        delta_t * 0.5*K(0, 4)*v(1);
    //前导后不导
    Jacobian(5, 5) = //0，1，2，3 ,4 行对 v(2)求导为 0
        delta_t * 0.5*v(1)*K(0, 5) - delta_t * 0.5*v(0)*K(1, 5) +
        delta_t * 0.5*w(1)*K(3, 5) - delta_t * 0.5*w(0)*K(4, 5) + K(5, 5);//前不导后导
```