论文解读及算法框架

小组成员：崔冠宇、李泽轩、罗迪、邵宁录、王玺、张晨阳

完成时间：2020/6/7

小组分工：

- 论文翻译：崔冠宇负责LaTeX排版和校对，其他组员每人翻译一页，校对一页。
- 本文完成：李泽轩完成1-2，王玺完成3，邵宁录完成4-5，罗迪完成6-7，张晨阳完成8-9。张晨阳排版汇总。
- 程序实现及展示：崔冠宇实现程序，制作展示课件，进行课堂汇报。

本文目录如下。其中，3-9部分对应论文的Section 3-9

1 论文简介

这篇文章中研究者提出了一个适用于任何维度的刚体动力学公式。同时将碰撞检测算法扩展到n维，解决了物体之间的碰撞和接触问题。在四维情况下作者做了实现。他通过这些四维刚体的三维截面来展示他们，用户也可以实时操纵这些物体。

这篇论文的贡献在于：

- 将基于几何代数的经典三维刚体动力学公式推广到了 n 维。通过将几何代数算子表示为矩阵，以一种简单的方式构建、对角化（diagonalize）、转换任意 n 维简单网格。这样一来，就可以在 n 维中建立欧拉方程。
- 计算 n 维中的碰撞和接触处理过程。作者给出了 Minkowski 差分法和基于几何代数的分离轴定理碰撞检测方法的 n 维公式。
- 提出了一种类似于我们对现实三维空间体验的四维物体互动方法。

这篇论文的独特之处在于为我们提供了一种从侧面了解四维空间的方式。

在第三部分中作者介绍了如何定义和计算三维物体的角动量，并描述了物体在转子旋转情况下的角动量对时间微分表示形式。

$$L_t(\omega) = I(\omega_t) - \omega \times I(\omega) = \tau$$

第四部分作者将惯性张量推广到n维以便同样能在n维空间中应用同样的形式计算。刚体做的是平移+旋转的运动，在刚体坐标系中存在一个旋转矩阵R以及平移向量，作者通过计算角动量之后坐标变化使得在全局惯性坐标系中能以同样的形式表达方程式。

$$L_t(\omega) = I'\omega_t - \omega \times I'\omega = \tau$$

同时随着角速度的积分，代表物体轴向的小误差会累积在转子中。在三维情况下可以通过对四元数归一化进行消除，在更高维的情况下需要使用新的算法矫正误差。

第五部分作者介绍了如何具体计算惯性张量矩阵

$P_{jk} = \int_V jk\rho dV$

并通过选择合适的能够使协方差矩阵对角化的旋转矩阵旋转物体可以导出对角化的惯性张量。

$[r]_\star [r]_\star^T = (\sum_{i=0}^{n-1} r_i [e_i]_\star)(\sum_{j=0}^{n-1} r_j [e_j]_\star^T) = \sum_{i,j}^{n-1} r_i r_j [e_i]_\star [e_j]_\star^T$

在第六第七部分中作者提出了如何处理物体的碰撞并基于分离轴定理提出了碰撞检测的方法。

在最后两个部分中作者介绍了如何实现四维物体与用户的交互模拟和显示。

2 相关概念

惯性张量

惯性张量(Inertia tensor)是描述刚体作定点转动时的转动惯性的一组惯性量，在三维中其表现形式为由9个分量构成的对称矩阵。在坐标系A中其表示为：

$$A_I = \begin{bmatrix} I_{xx} & -I_{xy} & -Ixz \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix}$$

惯性张量描述了物体的质量分布。惯性张量矩阵的特征值是主惯性矩，其对应的特征向量是惯性主轴的方向向量。

欧拉方程

欧拉方程描述刚体绕质心的转动。刚体运动 = 质心的平动 + 绕质心的转动。其中，质心平动用牛顿方程描述；绕质心的转动用欧拉方程描述，它们都涉及到质量及其分布。欧拉方程的简写形式为：
$I\dot{\omega} + \Omega I\omega = N$

旋转向量与四元数

旋转向量通过旋转轴和旋转角来表示任意一个旋转。在三维情况下旋转向量用3个自由度描述旋转，用平移向量来描述平移，总共用6个自由度来表达一次变换。

四元数的表示形式：$q = q_0 + q_1 i + q_2 j + q_3 k$，单位四元数可以用来表示三维空间中任意一个旋转，这种表达方式和欧式变换矩阵、欧拉角是等价的。

在三维空间中用四元数表示旋转

假设一个空间三维点 P 以及一个由轴角指定的旋转 $\vec{n}, \theta$ ，经过旋转之后变为点 p'。它们之间的关系可以用下列式子来表达：

- 把三维空间点用一个纯四元数来描述：$p = [0, x, y, z]$
- 把旋转用四元数表示：$q = [cos\theta/2, \vec{n}sin\theta/2]$
- 旋转之后的点为：$p' = qpq^{-1}$

3 背景

3.1 四维转子的表示

一个普通的转子可以表示为一个标量部分，一个双向量部分 (有六个分量) 和一个四维向量 (只有一个分量)。四维转子可以作为一个连续的 8 位数数组存储在内存中。

可实现的函数：

```
void Object4D::moveVectorByRotator(){
    //根据向量x和转子计算出旋转后的向量x1
    float R[8]
    Vector x,x1
    ...
}
```

用到的公式：$x' = Rx\widetilde{R}$，其中 $\widetilde{R}$ 表示 R 的逆，类似于四元数和复数的共轭

3.2 角动量的计算

可实现的函数：

```
void Object4D::moveAngularMbyRotator(){
    //根据惯性张量I和转子计算旋转后的角动量
    float R[8]
    Bivector AngularM
    Matrix I
    ...
}
```

用到的公式： $L(\omega) = RI(\omega)\widetilde{R}$, L为角动量

对角动量求时间导数（欧拉公式）：$L_t(\omega) = I(\omega_t) - \omega \times I(\omega)$

4 n维的推广

作者在这里做的事情主要是推广了n维空间中的惯性张量的表示。

他首先定义了 $[r]_\star$ 这个 $k \times n$ 的矩阵，然后通过下面这个式子，推广了惯性张量的定义

$I(\omega) = \int_V [r]_\star [r]_\star^T dV_\omega = \int_V \Delta I dV_\omega = I\omega$

其中，I是惯性张量，它是由 $[r]_\star$ 与 $[r]_\star$ 的转置相乘并对体积做积分得出的。而这个 $[r]_\star$ 与 $[r]_\star$ 的转置相乘，应当类似于三维空间中的位置矢量。而且这里可能默认了该物体的密度为单位1，不不然这个积分中还得再乘上一个ρ。然后 ω 是角速度量，因此最后得出的 $I\omega$ 就为角动量。

然后下面定义了了一一个k×k的矩阵 $[R]_2$ ，推测该矩阵的作用应当是用于坐标变换。然后利用该坐标变换算出变换后的 I' ，在下面这个式子中用于计算作用在刚体上的力矩 $L_t(\omega) = I(\omega_t) - \omega \times I(\omega) = \tau$

5 任意n维单纯形网格的惯性张量

在这部分，作者细化了惯性张量，从整体上的计算，细化到了一个平面。

$P_{jk} = \int_V jk\rho dV$

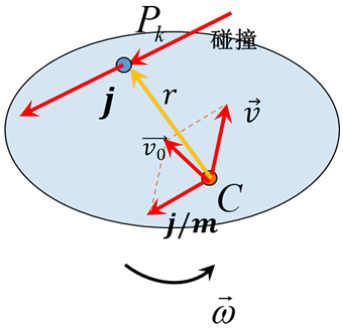
如上面这个公式所示，它算出的惯性张量，其实是上面4中讲到的 I 里的一个元素，即在 (jk) 这个平面上的。这里作者加入了对密度 ρ 的考虑。

之后下面关于协方差矩阵的部分，都是在说明如何计算出这个 P_{jk} 。在此不作过于深入的探讨。

6 碰撞解决

6.1 低维情形

一个刚体，质心为 C , 初始状态沿初速度 v 方向运动。某个时刻，其上某点 P_k 处发生了碰撞，给予了这个刚体一个冲量 j ，冲量定理，这个冲量会给质心一个沿该冲量方向的速度 j/m，合成为 v_0。同时由于冲量方向与质心到该点的位矢方向不相同，会产生一个角速度 $\omega = (r \times j)/J$ 三维情况下为叉乘，J 表示转动惯量)



即三维情形下：

$\overrightarrow{v_0} = \vec{v} + \frac{j}{m}$

$\overrightarrow{\omega_0} = \vec{\omega} + (r \times j)/J$

6.2 推广到高维

将这种情况推广到更高维，由于向量的推广性，速度公式不变，而角速度公式变化：

- r 与 j 的叉乘变成外积；
- 转动惯量 J 由前面所讲的惯性张量 I 取代，由于是矩阵，除法变为 I^{-1}

即在高维情形下：

$\overrightarrow{v_0} = \vec{v} + \frac{j}{m}$

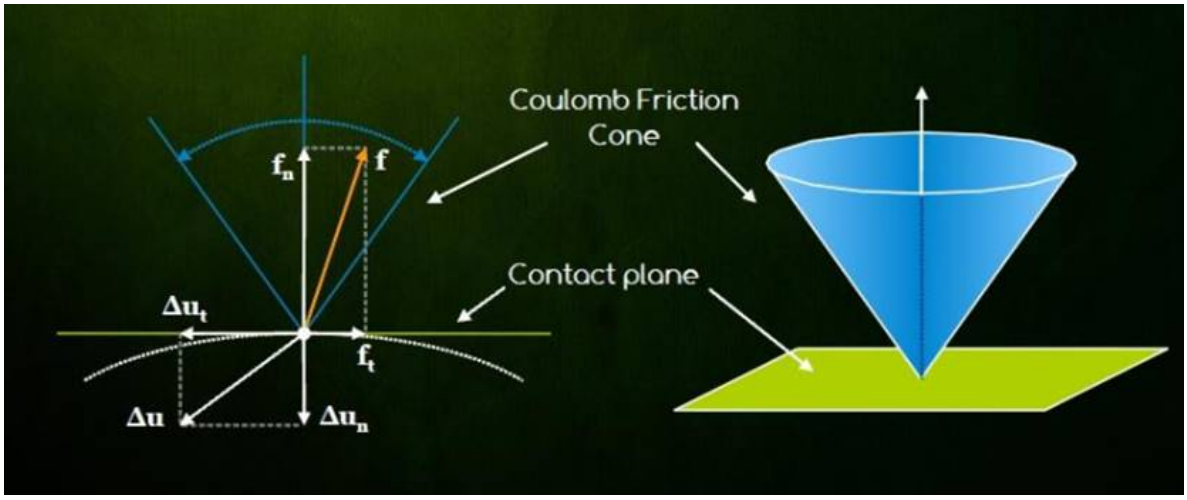
$\overrightarrow{\omega_0} = \vec{\omega} + I^{-1}(r \wedge j)$

6.3 算法设计

```
Void Object4D::collidesolution(Object4D* collisionObject){
    Vector p = this->getcollisiondot(collisionobject)
    Vector j = this->getcollisionpulse(collisionobject)
    Object4D.velocity += j/Object4D.mass
    Object4D.angularV += inv(Object4D.inertiatensor)
    *(exteriorproduct(disdance(p,Object4D.position),j))
    ...
}
```

6.4 拓展内容——碰撞过程中碰撞点的速度变化

至于碰撞点处的速度变化，既跟碰撞产生的冲量 j 有关，还跟碰撞点处的摩擦力有关，由摩擦力产生的冲量也会影响到该点的速度。

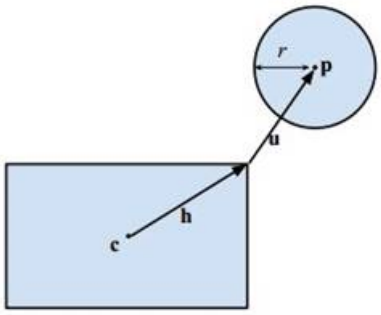


经过较为复杂的理论推导，可以得出碰撞后的新速度为：$u_0 = u + Kj$，其中，$K = \frac{\delta}{m} + [r]_\star^T I^{-1} [r]_\star$

7 碰撞检测

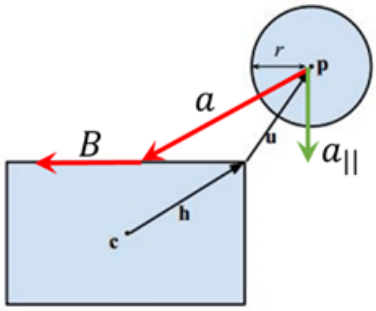
7.1 低维情形

先以低维的举例，要判断如下图所示的圆是否与矩形发生了碰撞，即要计算矩形到圆心的最短距离是否小于了圆的半径。即：$u < r$ 时碰撞。



检测方法是，构造圆心指向矩形各个表面的向量 a，也有表示矩形表面的向量 B，通过求 a 投影的方式，可以得到距离最小点。

如果这个点在表面外，再找这个点到表面上点的距离最小点即可。



7.2 高维情形

由于上述操作本身是向量操作，可以轻易推广到n维：$a_{||} = (a \cdot B)/B$

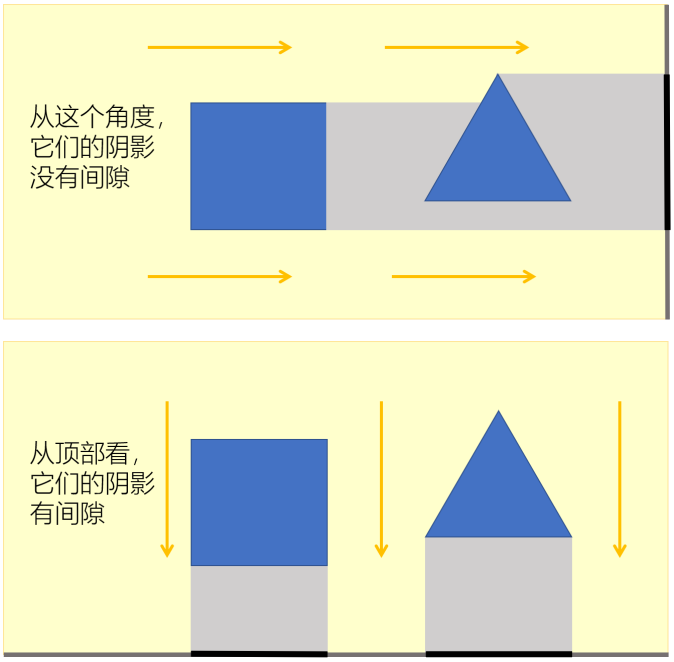
这里有一个注意点，可能会出现多个表面最小值相同的情况，这是因为这个点在公共边界上的原因。二维中就是两个边的交点；三维中就是两个面的交线或者三个面的交点；四维中就是两个三维单元的交面，三个三维单元的交线，或者四个三维单元的交点。

7.3 分离轴定理

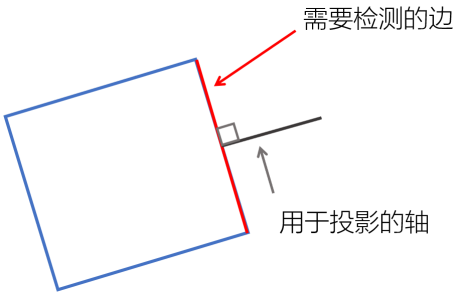
对于任意两个凸多面体是否碰撞检测的问题，文中采用到的是分离轴定理。

分离轴定理：如果能找到一条轴，使得两个物体在该轴上的投影互不重叠，那么这两个物体就是不相交的。

如下图所示，由第二个图的间隙说明投影互不重叠，即没有发生碰撞。



本来要验证两个物体是否碰撞，分离轴定理需要遍历空间中所有的方向，如果所有方向的投影都重合才能说明发生了碰撞。但是对于凸多面体而言，只需要考虑它表面低维的单元(面、线)即可，大大简化步骤。以二维为例，可能的分离轴就是该多边形所有边的法线：



- 对于三维的两个物体A和B，碰撞的可能一共有六种：

  面 - 面、面 - 边、面 - 点、边 - 边、边 - 点、点 - 点

  但是点可以看作退化的边，所以实际上只有三种：A面-B面/B边、A边-B面、A边-B边。这三种情况对应的分离轴是：物体A所有面的法向量、物体B所有面的法向量、物体A和物体B边之间公共垂面的法向量。
- 对于四维的情形，把边看作退化的面，分离轴除了考虑A的三维体与B的三维体的法向量，还需要考虑A边与B面、A面与B边的公共垂面的法向量。
- 发现规律后，作者把分离轴定理推广到了n维：n维物体A和B的低维单元只要满足 $m_a + m_b = n - 1$, 它们的外积的对偶向量（即法线方向）：$V_a^{m_a}(i) \wedge V_b^{m_b}(j)$ 都是可能的分离轴。

7.4 算法设计

```
Bool collidewith(Object4D* collisionObject){
    SeparatingAxisMethod(Object4D.edge(), Object4D.face(), Object4D.3Dcell())
    ...
}
```

8 交互式模拟与显示

8.1 四维对象的显示

4D -> 3D: 展示四维对象的三维切片，即单个三维表面和四维对象之间的交集。三维表面可以通过取一个第四维的值来确定。

3D -> 2D: 由于屏幕是二维的，所以还需要让这个三维切片显示在二维屏幕上。（这一部分崔冠宇提出直接调用相关库，我们不需要自己实现，故没有提供方法和伪代码）

需要用到的方法：

```
Object3D Projector::to3D(Object4D *){
    // 向3维子空间投影
    ...
}
```

8.2 模拟器与交互

推广重力到n维：沿法向量指向地面

推广摩擦：与3D模型相同

交互媒介：鼠标、触摸屏、VR控制器

可以做到的交互：

- 只能在当前的三维切片内点击、拖动、旋转对象
- 可以拖动滑块沿第四维的轴移动来改变三维切片
- 4D对象撞击时，VR控制器震动

需要用到的方法：

- 点击、拖动、旋转：需要改变Object4D的属性或调用Object4D的方法

```
bool InteractionMgr::userInput(operations){
    switch operations.type:
        case click:
            operations.object.showInfo()
            break
        case moveObject:
            operations.object.move(operations)
            break
        case rotate:
            operations.object.rotate(operations)
            break
        default:
            raise error("Illegal operation")
```

```
}
bool Object4D::showInfo(){
    // 在屏幕上打印对象的速度、位置信息
    ...
}
bool Object4D::move(operations){
    // 改变对象的位置
    Object4D.pos = operations.newPos
}
bool Object4D::rotate(operations){
    // 计算新的角动量，具体公式在section9给出，然后根据角动量运动
    this->calcuAngularMomentum(operations)
    this->moveByAngularMomentum()
}
```

- 使用滑块改变第四维的值 nowW：

```
void InteractionMgr::changeW()
{
    // 用户操作滑块，会改变 InteractionMgr 类内的 nowW
    ...
}
```

- VR控制器震动：碰撞时调用使其震动的方法

```
bool Object4D::collideWith(Object4D* collisionObject){
    this->interactionMgr.vibrate()
    collisionObject.interactionMgr.vibrate()
}
void InteractionMgr::vibrate(){
    // 使交互设备震动
    ...
}
```

9 4D DZHANIBEKOV 效应

这一部分详细描述4D对象旋转时的角动量如何计算，即完成函数

`void Object4D::calcuAngularMomentum(operations)`

需要用4D DZHANIBEKOV效应计算旋转的角动量变化，公式如下：

$$I_{ij}\dot{\omega}_{ij} = (I_{jk} - I_{ik})\omega_{ik}\omega_{jk} + (I_{jm} - I_{im})\omega_{im}\omega_{jm}$$

其中 I_{ij} 和 ω_{ij} 分别是平面 (ij) 的力矩、角动量， k 和 m 是除 i 和 j 之外的其他两个索引。

这说明平面 (ij) 的方程式取决于除与其垂直的平面 (km) 以外的所有其他旋转平面$(ik; jk; im; jm)$

作者说“双重旋转似乎是稳定的”，但没有给出数学证明，只是用实际例子进行了描述。