第六章：几何变换

类层次结构

```
classDiagram
    class Transform~Scalar,Dim~{
        <<interface>>
        +matrix() Matrix
        +inverse() Transform
        +operator*(Transform) Transform
    }

    class Translation~Scalar,Dim~ {
        -vector_ Vector
        +Translation(Vector)
        +vector() Vector
    }

    class Rotation2D~Scalar~ {
        -angle_ Scalar
        +Rotation2D(Scalar angle)
        +angle() Scalar
    }

    class AngleAxis~Scalar~ {
        -angle_ Scalar
        -axis_ Vector3
        +AngleAxis(Scalar angle, Vector3 axis)
        +angle() Scalar
        +axis() Vector3
    }

    class Quaternion~Scalar~ {
        -w_ Scalar
        -x_ Scalar
        -y_ Scalar
        -z_ Scalar
        +Quaternion(w,x,y,z)
        +matrix() Matrix3
    }

    class Scaling~Scalar,Dim~ {
        -factors_ Vector
        +Scaling(Vector factors)
        +factors() Vector
    }
```

```
class Isometry3d {
    +rotate(Quaternion)
    +translate(Vector3d)
    +matrix() Matrix4d
}

class Affine3d {
    +scale(Vector3d)
    +rotate(Matrix3d)
    +translate(Vector3d)
}

Transform <|-- Translation
Transform <|-- Rotation2D
Transform <|-- AngleAxis
Transform <|-- Quaternion
Transform <|-- Scaling
Transform <|-- Isometry3d
Transform <|-- Affine3d
```

类说明

1. Transform：所有变换的基类
   - 提供基本的矩阵操作接口
   - 支持变换组合和求逆
2. 基本变换类：
   - Translation：平移变换
   - Rotation2D：2D 平面旋转
   - AngleAxis：3D 轴角旋转
   - Quaternion：四元数旋转
   - Scaling：缩放变换
3. 复合变换类：
   - Isometry3d：刚体变换（保持欧氏距离）
   - Affine3d：仿射变换（保持平行关系）

6.1 基本变换

Eigen 提供了多种几何变换类型，包括平移、旋转、缩放等。

主要的变换类型包括：- Translation3d：平移变换 - AngleAxisd：轴角旋转 - Quaterniond：四元数旋转 - Scaling3d：缩放变换

6.1.1 平移

```
Vector3d translation(1, 2, 3);
Translation3d t(translation);
```

```
// 或者直接使用向量
Translation3d t(1, 2, 3);
```

6.1.2 旋转

Eigen 提供了多种旋转表示方法，每种方法都有其特点和适用场景：

1. Matrix3d（旋转矩阵）
   - 3x3 双精度矩阵
   - 直观但参数冗余（9 个参数表示 3 自由度）
   - 成员函数：
     - eulerAngles(i,j,k)：提取欧拉角，i,j,k 指定轴的顺序
2. AngleAxisd（轴角表示）
   - 最简单的旋转表示方法
   - 构造函数：AngleAxisd(angle, axis)
   - 成员函数：
     - angle()：获取旋转角度
     - axis()：获取旋转轴
     - matrix()：转换为矩阵形式
   - 可以直接相乘组合多个旋转
3. Quaterniond（四元数）
   - 最稳定的旋转表示方法
   - 构造函数：
     - Quaterniond(w,x,y,z)：直接指定四元数分量
     - Quaterniond(Matrix3d)：从旋转矩阵构造
     - Quaterniond(AngleAxisd)：从轴角构造
   - 成员函数：
     - w(),x(),y(),z()：获取四元数分量
     - matrix()：转换为矩阵形式
     - normalized()：归一化

示例代码：

```
// 使用欧拉角（ZYX 顺序）
Matrix3d R = AngleAxisd(yaw, Vector3d::UnitZ())
           * AngleAxisd(pitch, Vector3d::UnitY())
           * AngleAxisd(roll, Vector3d::UnitX());

// 使用四元数
Quaterniond q = AngleAxisd(angle, axis);
```

6.1.3 缩放

```
Vector3d scale(2, 2, 2);
Scaling3d s(scale);
// 或者直接使用标量
Scaling3d s(2.0);
```

6.2 三维旋转表示

在三维空间中，旋转的表示方式有多种，包括欧拉角、旋转矩阵、四元数等。

重要的辅助类

1. Vector3d
   - 3 维向量
   - 静态成员：
     - UnitX(): (1,0,0)
     - UnitY(): (0,1,0)
     - UnitZ(): (0,0,1)
   - 成员函数：
     - transpose(): 转置
     - normalized(): 归一化

注意事项

1. 旋转顺序很重要，不同的顺序会得到不同的结果
2. 欧拉角可能存在万向节死锁问题
3. 四元数提供了最稳定的旋转插值
4. 所有角度计算都使用弧度制
5. 组合多个旋转时要注意顺序，旋转不是可交换的

6.2.1 欧拉角

```
// 从旋转矩阵获取欧拉角
Vector3d euler = R.eulerAngles(2, 1, 0); // ZYX 顺序
```

6.2.2 旋转矩阵

```
// 从欧拉角构造
Matrix3d R;
R = AngleAxisd(roll, Vector3d::UnitX())
  * AngleAxisd(pitch, Vector3d::UnitY())
  * AngleAxisd(yaw, Vector3d::UnitZ());
```

6.2.3 四元数

```
// 从旋转矩阵构造
Quaterniond q(R);

// 从欧拉角构造
Quaterniond q = AngleAxisd(yaw, Vector3d::UnitZ())
              * AngleAxisd(pitch, Vector3d::UnitY())
              * AngleAxisd(roll, Vector3d::UnitX());
```

6.3 变换组合

在实际应用中，我们经常需要将多个变换组合在一起，比如先旋转再平移。Eigen 提供了多种变换组合的方式。

6.3.1 仿射变换

```
// 创建变换矩阵
Affine3d transform = Translation3d(1, 2, 3)
                   * AngleAxisd(M_PI/2, Vector3d::UnitZ())
                   * Scaling3d(2.0);

// 应用变换
Vector3d point(1, 0, 0);
Vector3d transformed = transform * point;
```

6.3.2 刚体变换

```
// 使用 Isometry3d 表示刚体变换（旋转 + 平移）
Isometry3d T = Isometry3d::Identity();
T.rotate(q);
T.pretranslate(Vector3d(1, 2, 3));
```

6.4 代码示例说明

6.4.1 transforms.cpp 详解

这个示例展示了如何使用 Eigen 进行基本的几何变换。让我们逐行分析代码：

```
// 创建一个点和变换
Vector3d point(1, 0, 0);  // 在 x 轴上的点
cout << " 原始点: " << point.transpose() << endl;

// 1. 平移变换
Translation3d translation(1, 2, 3);  // 在 x,y,z 方向分别平移 1,2,3 单位
Vector3d translated = translation * point;
cout << " 平移后: " << translated.transpose() << endl;
```

- `Vector3d` 用于表示 3D 点或向量
- `Translation3d` 专门用于表示平移变换
- `.transpose()` 用于横向打印向量

```
// 2. 旋转变换
AngleAxisd rotation(M_PI/2, Vector3d::UnitZ());  // 绕 Z 轴旋转 90 度
Vector3d rotated = rotation * point;
cout << " 旋转后: " << rotated.transpose() << endl;
```

- `AngleAxisd` 使用轴角表示旋转
- `M_PI` 是数学常数

- `Vector3d::UnitZ()` 表示 Z 轴单位向量 $(0,0,1)$

```
// 3. 缩放变换
Scaling3d scale(2.0, 2.0, 2.0);  // 各方向放大 2 倍
Vector3d scaled = scale * point;
cout << " 缩放后: " << scaled.transpose() << endl;
```

- `Scaling3d` 用于表示缩放变换
- 可以在不同方向设置不同的缩放因子

```
// 4. 组合变换
Affine3d transform = translation * rotation * scale;
Vector3d transformed = transform * point;
cout << " 组合变换后: " << transformed.transpose() << endl;
```

- `Affine3d` 可以表示任意仿射变换
- 变换的组合顺序很重要：先缩放，再旋转，最后平移

6.4.2 rotations.cpp 详解

这个示例展示了旋转的不同表示方法及其转换。让我们详细分析代码：

```
// 定义欧拉角（弧度制）
double yaw = M_PI / 4;    // 绕 Z 轴旋转 45 度
double pitch = M_PI / 6;  // 绕 Y 轴旋转 30 度
double roll = M_PI / 3;   // 绕 X 轴旋转 60 度
```

- 使用弧度制表示角度
- 遵循 ZYX 顺序（偏航-俯仰-滚转）
- 这是航空航天中常用的欧拉角顺序

```
// 从欧拉角创建旋转矩阵
Matrix3d R = (AngleAxisd(yaw, Vector3d::UnitZ()) *
              AngleAxisd(pitch, Vector3d::UnitY()) *
              AngleAxisd(roll, Vector3d::UnitX())).matrix();
```

- 使用轴角表示创建基本旋转
- 按照 Z-Y-X 顺序组合旋转
- `.matrix()` 将旋转转换为矩阵形式

```
// 从旋转矩阵提取欧拉角
Vector3d euler = R.eulerAngles(2, 1, 0);  // ZYX 顺序
```

- `eulerAngles(2,1,0)` 指定提取顺序：2=Z 轴，1=Y 轴，0=X 轴
- 返回的角度范围：
  - Z 轴：[- ,]
  - Y 轴：[- /2, /2]
  - X 轴：[- ,]

```
// 四元数表示
Quaterniond q(R);  // 从旋转矩阵构造四元数
```

```
cout << " 四元数:\n"
     << "w: " << q.w() << "\n"   // 实部
     << "x: " << q.x() << "\n"   // 虚部 i
     << "y: " << q.y() << "\n"   // 虚部 j
     << "z: " << q.z() << "\n"; // 虚部 k
```

- 四元数形式：$w + xi + yj + zk$
- w 是实部，(x,y,z) 是虚部
- 四元数自动归一化：$w^2 + x^2 + y^2 + z^2 = 1$

```
// 轴角表示
AngleAxisd aa(R);  // 从旋转矩阵构造轴角
cout << " 轴角表示:\n"
     << " 角度: " << aa.angle() * 180 / M_PI << " 度\n"
     << " 轴: " << aa.axis().transpose() << "\n";
```

- 轴角表示：绕固定轴旋转特定角度
- `angle()` 返回旋转角度（弧度）
- `axis()` 返回旋转轴的单位向量

重要概念说明

1. 旋转表示的选择：
   - 欧拉角：直观但有万向节死锁问题
   - 旋转矩阵：计算方便但参数冗余
   - 四元数：紧凑且稳定，适合插值
   - 轴角：最符合人类直觉
2. 常见应用场景：
   - 机器人运动学：描述关节旋转
   - 计算机图形学：相机视角变换
   - 姿态估计：传感器数据融合
3. 性能考虑：
   - 存储：四元数最省内存
   - 计算：旋转矩阵乘法最快
   - 插值：四元数球面线性插值最平滑

6.5 实践建议

1. 选择合适的变换类型
   - Isometry3d：刚体变换
   - Affine3d：一般仿射变换
   - Projective3d：投影变换
2. 避免万向节死锁
   - 注意欧拉角的使用顺序
   - 考虑使用四元数
3. 性能优化
   - 缓存变换矩阵
   - 使用适当的数据类型