经纬高坐标系（BLH）到站心坐标系（ENU）转换

一、术语和定义

1、地球椭球earth ellipsoid

近似表示地球的形状和大小，并且其表面为等位面的旋转椭球。

2、参考椭球reference ellipsoid

最符合一定区域的大地水准面，具有一定大小和定位参数的旋转地球椭球。

3、地心地固坐标系（空间直角坐标系）ECEF

空间直角坐标系，其xy平面与地球赤道面重合，x轴指向0°经度方向，y轴指向东经90°方向，z轴为与赤道平面垂直指向地理北极，构成右手坐标系。xyz轴随着地球一起旋转，在惯性空间中不再描述固定的方向。

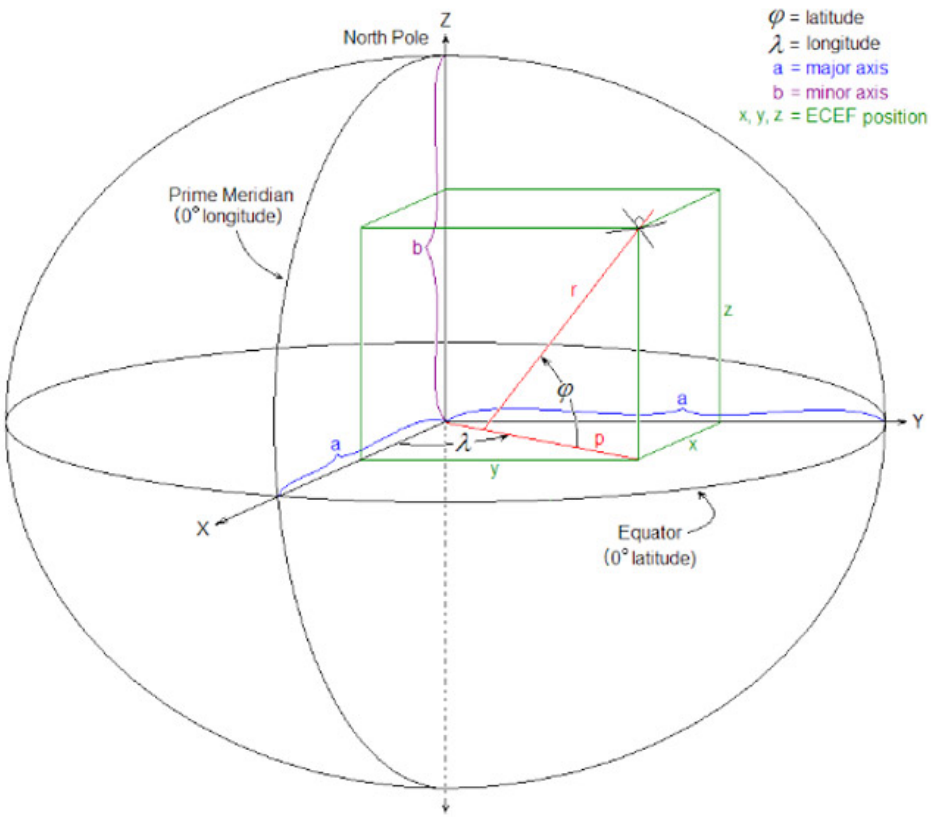


图1-1 地心固坐标系示意图

4、大地坐标系geodetic coordinate system

以地球椭球中心为原点、起始子午面和赤道面为基准面的地球坐标系。由于地球的形状不规则，在GPS/北斗中使用一个标准椭球体来近似描述，这样可以方便计算接收机的经度、纬度和高度。椭球体可以用两个参数来表示，长半轴和扁率，其他参数如短半轴、偏心率、第二偏心率等都可以由这两个参数导出。

坐标系	说明
1984 世界大地坐标系 (World Geodetic System 1984,WGS84)	美国军用大地坐标系统，GPS 系统使用的坐标系。
2000 国家大地坐标系 (CGCS2000)	北斗系统使用的坐标系。给定点位在某一框架和某一历元下的空间直角坐标,投影到 CGCS2000 椭球和 WGS84 椭球上所得的经度相同，纬度方向最大差值 0.11mm。
1954 年北京坐标系 (Beijing Geodetic Coordinate System 1954)	将我国大地控制网与苏联 1942 普尔科沃大地坐标系联结后建立的我国过渡性大地坐标系。
1980 西安坐标系 (Xian Geodetic Coordinate System 1980)	采用 IAG 1975 国际椭球，以 JYD1968.0 系统为椭球定向基准，大地原点设在陕西泾阳县永乐镇，采用多点定位所建立的大地坐标系。

5、椭球高/海拔高/大地水准面高(高程异常) ellipsoid/orthometric/geoid height

如图3-2所示，橙色曲线为参考椭球面，绿色曲线为地表（或GPS天线位置），h为椭球高，GPS定位解算结果（ECEF）在WGS84椭球下可转换为经纬度和椭球高h。蓝色曲线为大地水准面Geoid，对应

于最小二乘下的全球平均海平面（Mean Sea Level），恒重力势平面，由于地球密度分布不均，各地的重力势存在差异，因而大地水准面不是规则的曲面（图33）。相对于大地水准面的高度H称为海拔高（orthometric height），或者平均海平面高（MSL）。大地水准面的高度N是大地水准面相对于椭球面的高度，也称高程异常。椭球高h、海拔高H和高程异常N三者之间的关系为h=H+N。

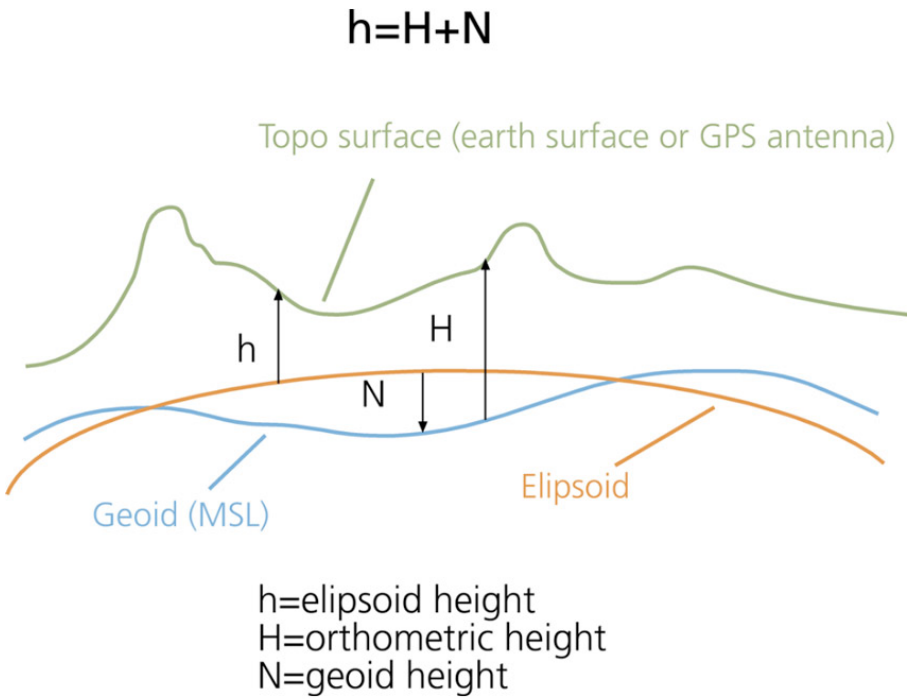


图1-2 椭球高/海拔高/大地水准面高

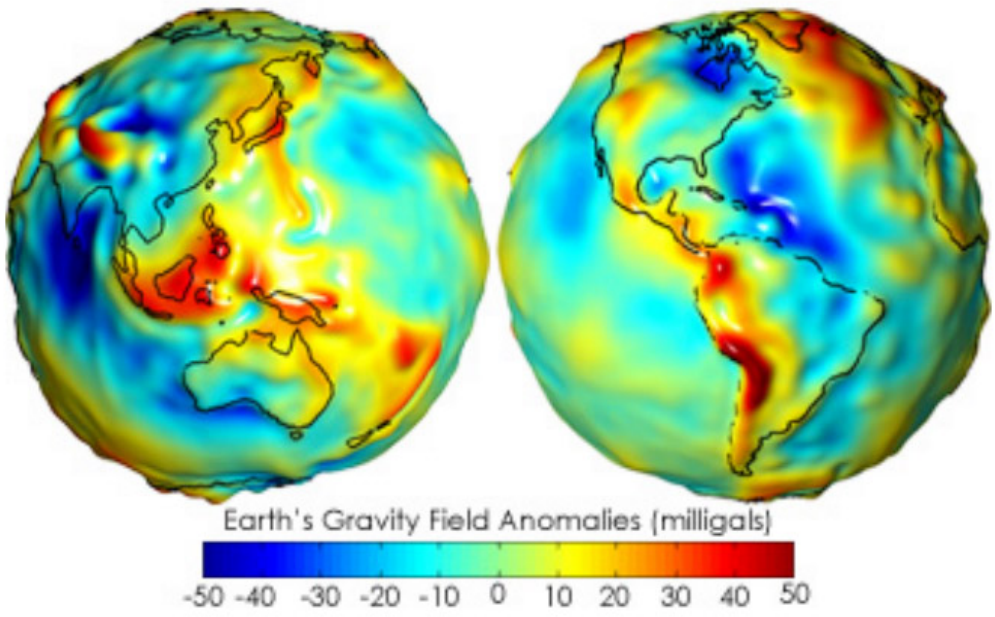


图1-3 大地重力势

6、站心坐标系（东北天坐标系）

站心坐标系也叫东北天坐标系ENU，用于表示以观察者为中心的其他物体的运动规律。以站心为坐标系原点，z轴与椭球法线重合，向上为正，y与椭球短半轴重合（北向），x轴与椭球长半轴重合（东向）。

7、高斯投影Gauss projection

由于地球是球体，为了在地图上表示，将椭球面沿子午线划分成若干个经差相等的狭窄地带，然后各带分别投影到平面上。一般以赤道上某一经度的点为原点，计算投影后的坐标，为了使投影后的坐标为正数，通常在横轴加500km的常数。

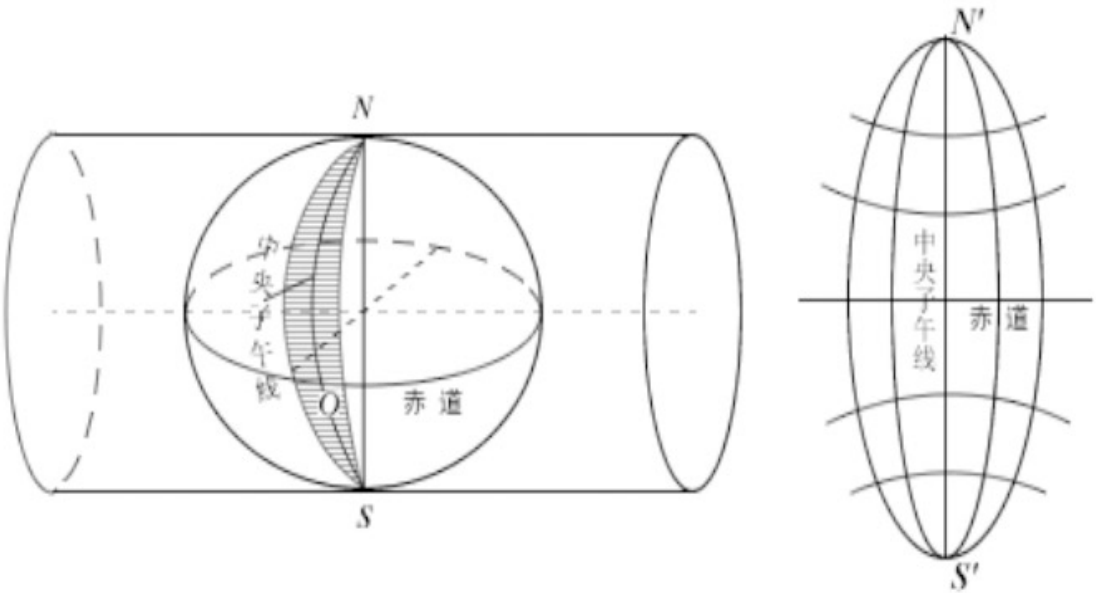
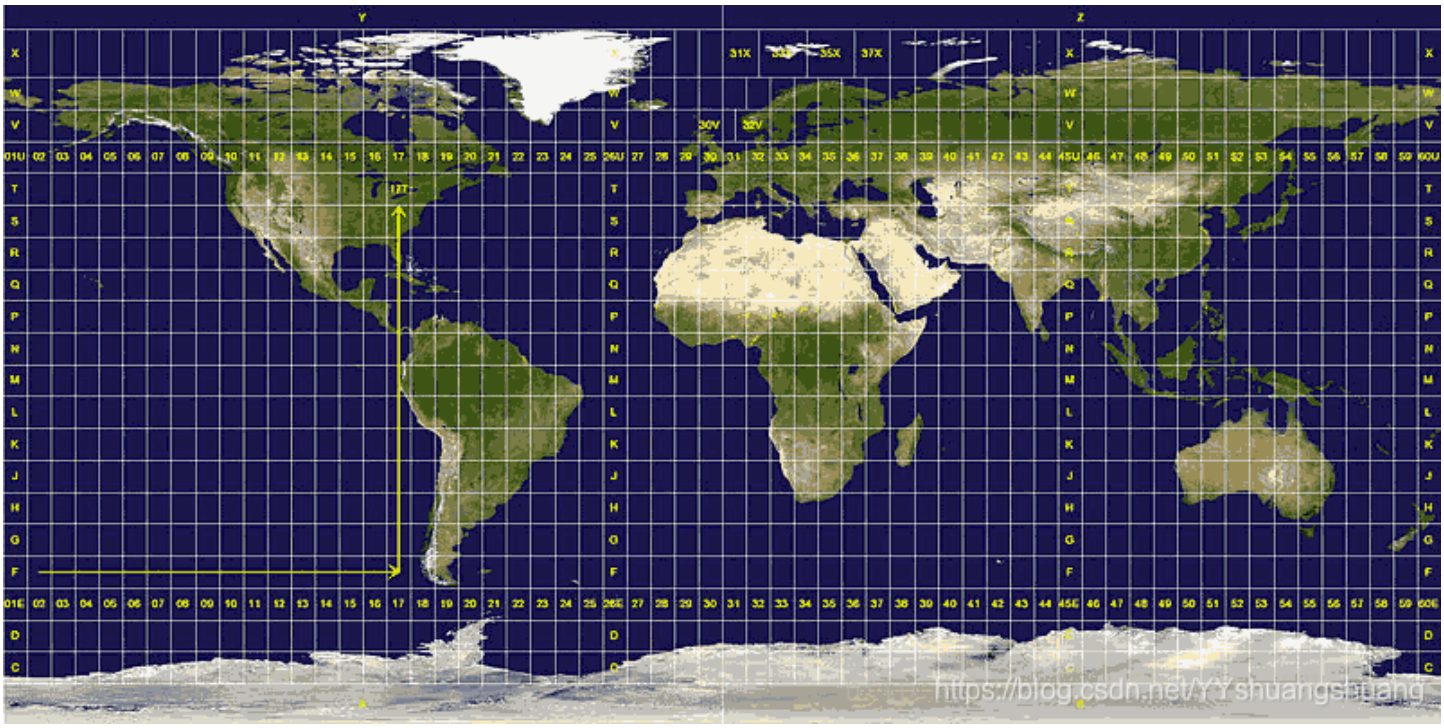


图1-4 高斯投影

也叫站心坐标系以用户所在位置P为坐标原点。坐标系定义为： X轴：指向东边 Y轴：指向北边 Z轴：指向天顶

ENU局部坐标系采用三维直角坐标系来描述地球表面，实际应用较为困难，因此一般使用简化后的二维投影坐标系来描述。在众多二维投影坐标系中，统一横轴墨卡托（The Universal Transverse Mercator，UTM）坐标系是一种应用较为广泛的一种。UTM 坐标系统使用基于网格的方法表示坐标，它将地球分为 60 个经度区，每个区包含6度的经度范围，每个区内的坐标均基于横轴墨卡托投影，如下图所示：



8、坐标转换coordinate transformation

包含坐标系变换和椭球基准变换两层含义。在测量数据处理过程中，采用适用的转换模型和转换方法，空间点从某一参考椭球基准下的坐标转换到另一坐标系统下的坐标，如WGS84转为北京54。坐标转换过程就是转换参数的求解过程。

9、坐标系变换coordinate conversion

同一椭球基准下，空间点的不同坐标表示形式间进行变换。包括大地坐标系和空间直角坐标系的互相转换、空间直角坐标系与站心坐标系间的转换和高斯投影坐标正反算。

10、椭球变换ellipsoid conversion

空间点在不同的参考椭球间的坐标变换。

11、平移参数translation parameters

两坐标系转换时，新坐标系原点在原坐标系中的坐标分量。

12、旋转参数rotation parameters

两坐标系转换时，把原坐标系中的各坐标轴左旋转到与新坐标系相应的坐标轴重合或平行时坐标系各轴依次转过的角度。

13、车身坐标系—右-前-天坐标（RFU）

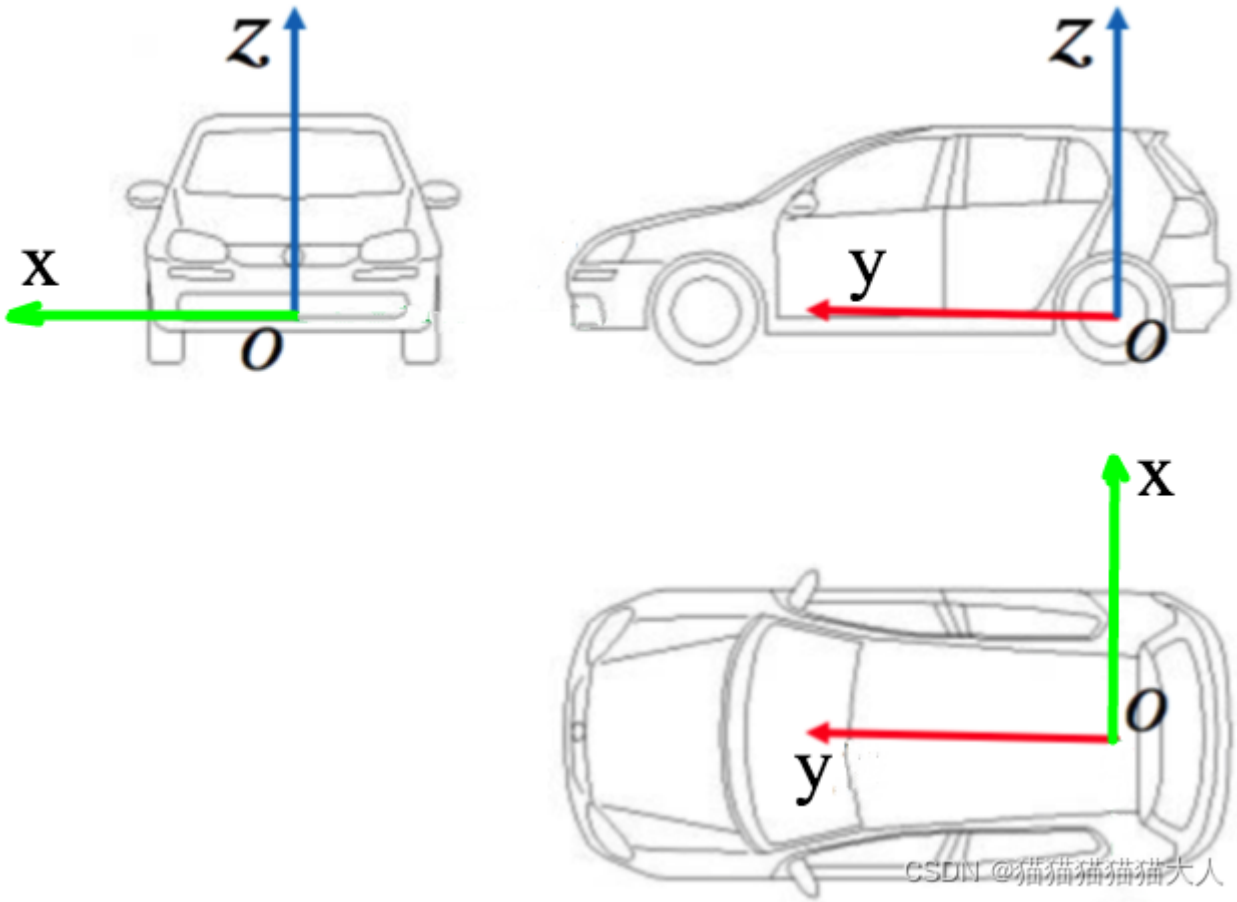
车身坐标系—右-前-天坐标（RFU）的定义如下：

X轴：面向车辆前方，右手所指方向

Y轴：车辆前进方向

Z轴：与地面垂直，指向车顶方向

注意：车辆参考点为后轴中心。该坐标系一般用于感知模块。如下图所示：



14、 车身坐标系—前-左-天坐标（FLU）

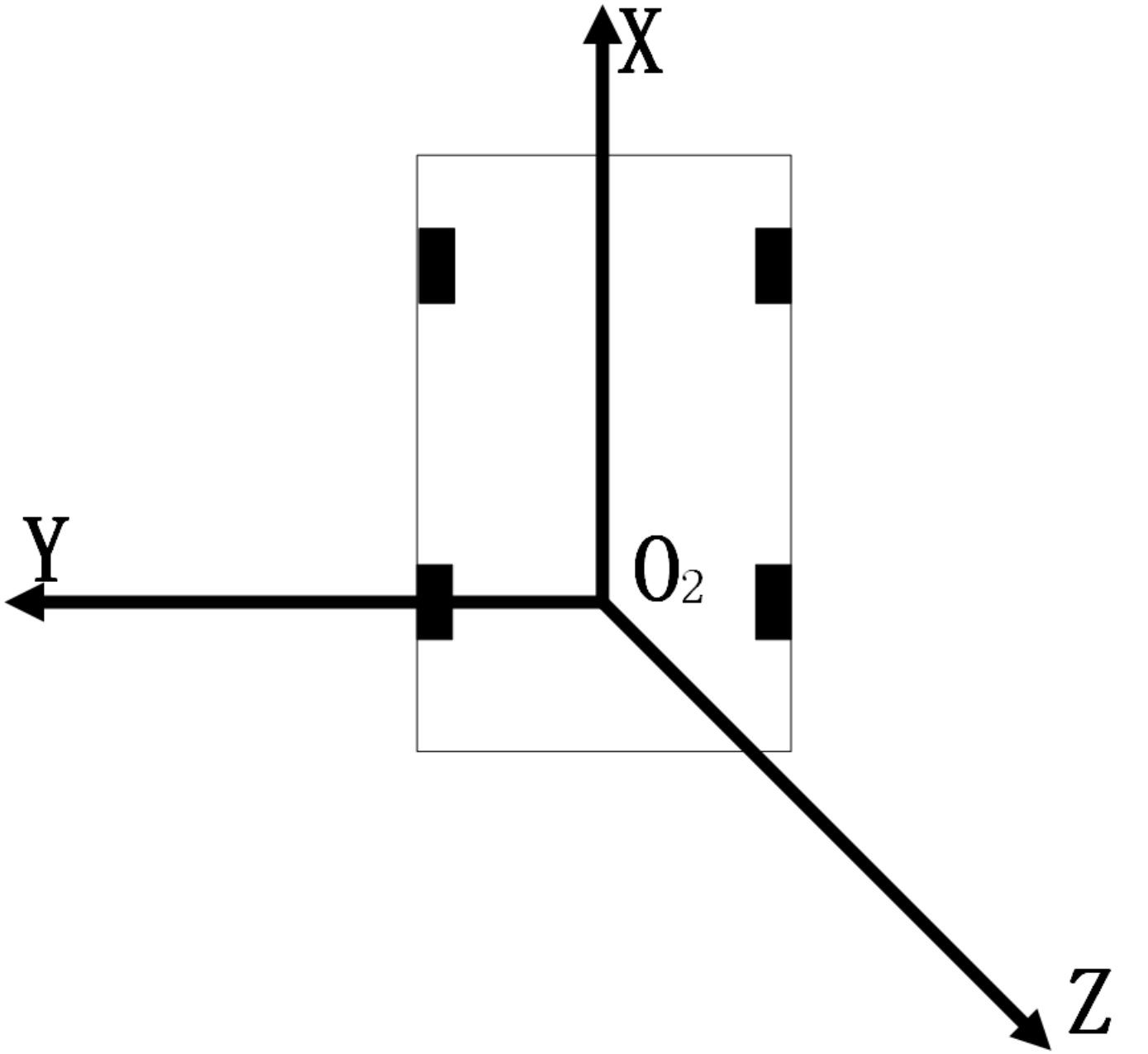
车身坐标系—前-左-天坐标（FLU）的定义如下：

X轴：车辆前进方向

Y轴：面向车辆前方，左手所指方向

Z轴：与地面垂直，指向车顶方向

注意：车辆参考点为后轴中心。该坐标系常用于实时相对地图模块。将RFU坐标系向左旋转90度即可得到FLU坐标系。

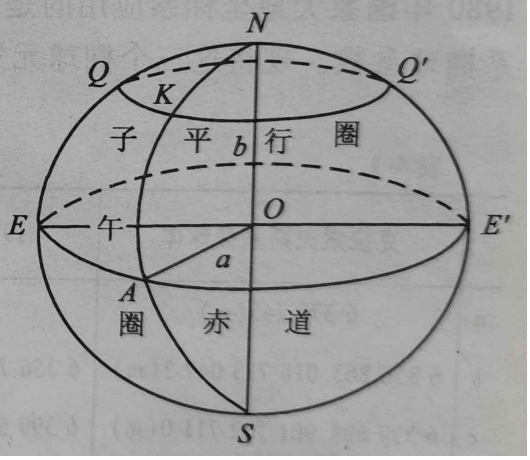


二、大地经纬度坐标（BLH）与地心地固坐标（ECEF）的的转换

1、概述

要解决这个问题首先得理解地球椭球这个概念，这里直接用武汉大学《大地测量学基础》（孔详元、郭际明、刘宗全）的解释吧：

地球椭球是经过适当选择的旋转椭球。旋转椭球是椭圆绕其短轴旋转而成的几何形体。在图 4-1 中，O 是椭球中心，NS 为旋转轴，a 为长半轴，b 为短半轴。包含旋转轴的平面与椭球面相截所得的椭圆，叫子午圈（或经圈，或子午椭圆），如 $NKAS$。旋转椭球面上所有的子午圈的大小都是一样的。垂直于旋转轴的平面与椭球面相截所得的圆，叫平行圈（或纬圈），如 QKQ'。通过椭球中心的平行圈，叫赤道，如 EAE'。赤道是最大的平行圈，而南极点、北极点是最小的平行圈。



大地经纬度坐标系是地理坐标系的一种，也就是我们常说的经纬度坐标+高度。经纬度坐标用的虽然多，但是很多人并没有理解经纬度的几何意义：纬度是一种线面角度，是坐标点P的法线与赤道面的夹角（注意这个法线不一定经过球心）；经度是面面角，是坐标点P所在的的子午面与本初子午面的夹角。这也是为什么经度范围是-180 ~ +180，纬度范围却是-90 ~ +90：

1. 大地坐标系

如图 4-2 所示，P 点的子午面 NPS 与起始子午面 NGS 所构成的二面角 L，叫做 P 点的大地经度。由起始子午面起算，向东为正，叫东经（$0° \sim 180°$）；向西为负，叫西经（$0° \sim 180°$）。P 点的法线 Pn 与赤道面的夹角 B，叫做 P 点的大地纬度。由赤道面起算，向北为正，叫北纬（$0° \sim 90°$）；向南为负，叫南纬（$0° \sim 90°$）。在该坐标系中，P 点的位置用 L, B 表示。如果点不在椭球面上，表示点的位置除 L, B 外，还要附加另一参数——大地高 H，它同正常高 $H_{正常}$ 及正高 $H_{正}$ 有如下关系

$$\begin{cases} H = H_{正常} + \zeta \text{（高程异常）} \\ H = H_{正} + N \text{（大地水准面差距）} \end{cases} \tag{4-10}$$

显然，如果点在椭球面上，$H = 0$。

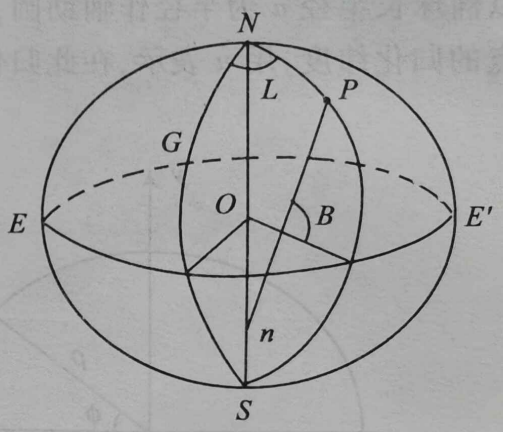
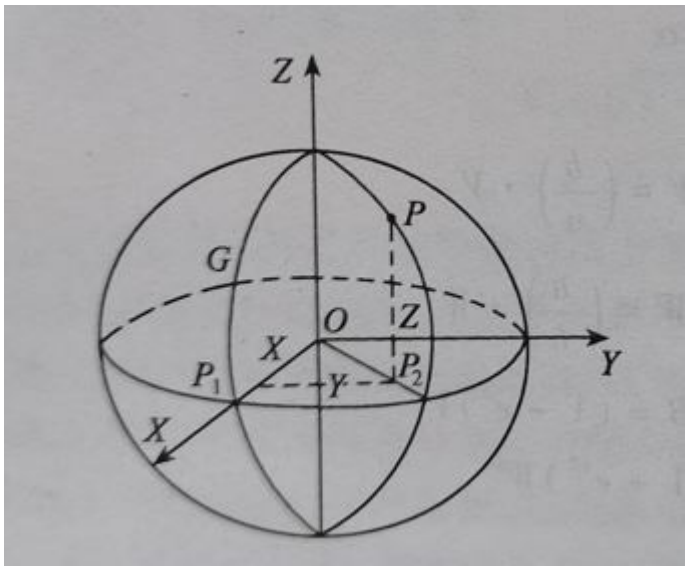


图 4-2

地心地固坐标系就是我们常用的笛卡尔空间直角坐标系了。这个坐标系以椭球球心为原点，本初子午面与赤道交线为X轴，赤道面上与X轴正交方向为Y轴，椭球的旋转轴(南北极直线)为Z轴。显然，这是个右手坐标系：

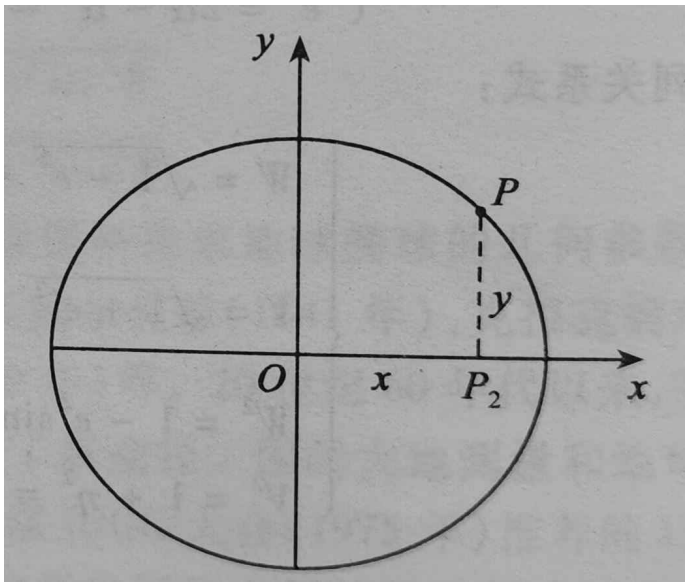


显然，两者都是表达的都是空间中某点P，只不过一个是经纬度坐标（BLH），一个是笛卡尔坐标（XYZ）；两者是可以相互转换的

2、推导

BLH->XYZ

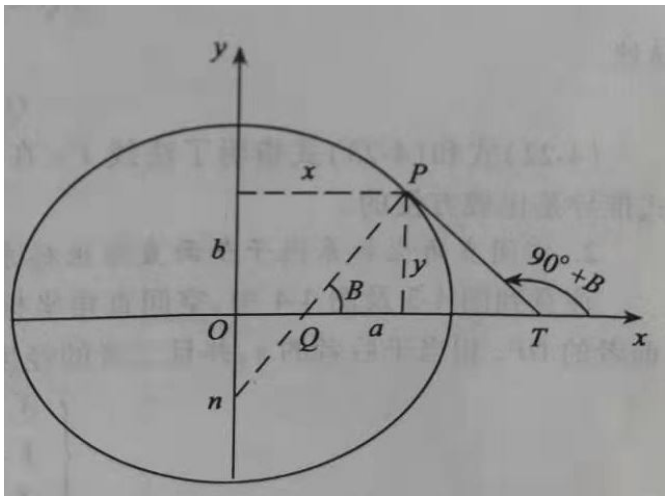
将P点所在的子午椭圆放在平面上，以圆心为坐标原点，建立平面直角坐标系：



对照地心地固坐标系，很容易得出：

$$\begin{cases} Z = y \\ X = x \cdot cosL \\ Y = x \cdot sinL \end{cases}$$

那么，关键问题在于求子午面直角坐标系的x,y。过P点作原椭球的法线Pn，他与子午面直角坐标系X轴的夹角为B；过P点作子午椭圆的切线，它与X轴的夹角为(90°+B):


y
x
P
b
y
90° +B
B
O
Q
a
T
x
n

根据椭圆的方程，位于椭圆的P点满足：

$$\frac{x^2}{a^2}+\frac{y^2}{b^2}=1 \tag{1.2}$$

对x求导，有：

$$\frac{dy}{dx}=-\frac{b^2}{a^2}\cdot\frac{x}{y} \tag{2}$$

又根据解析几何可知，函数曲线（椭圆）某一点（就是P点）的倒数为该点切线的斜率，也就是正切值：

$$\frac{dy}{dx}=tan(90^o+B)=-cotB \tag{3}$$

联立公式（2）(3)，可得：

$$y=x(1-e^2)tanB \tag{4}$$

其中，e为椭圆第一偏心率：

$$e=-\frac{\sqrt{a^2-b^2}}{a}$$

令Pn的距离为N，那么显然有：

$$x=NcosB \tag{4-2}$$

根据（4）式可得：

$$y=N(1-e^2)sinB \tag{4-3}$$

将其带入（1）式，可得到椭球上P点的坐标为：

$$\begin{cases}X=NcosBcosL\\Y=NcosBsinL\\Z=N(1-e^2)sinB\end{cases} \tag{5}$$

那么唯一的未知量就是Pn的长度N了，将（4）式带入到椭圆方程式(1.2)：

$$\frac{x^2}{a^2}+\frac{x^2(1-e^2)^2tan^2B}{b^2}=1$$

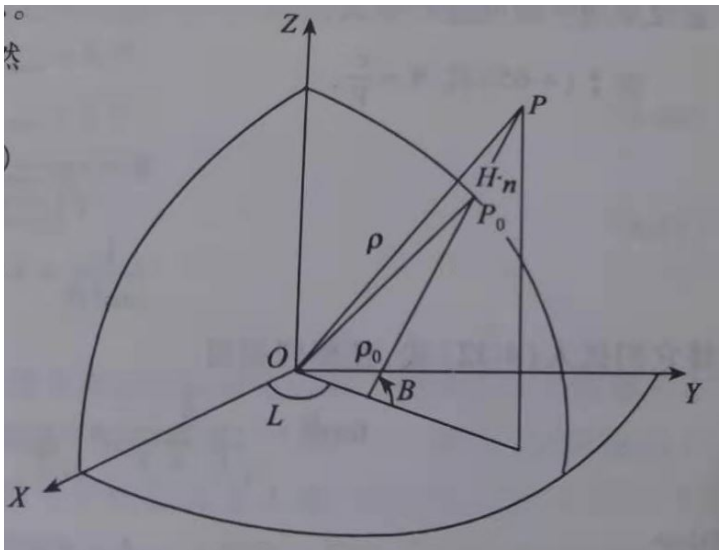
化简，得：

$$x=\frac{acosB}{\sqrt{1-e^2sin^2B}} \tag{6}$$

联立式（5）式（6），得：

$$N=\frac{a}{\sqrt{1-e^2sin^2B}} \tag{6}$$

通过式（5）式（6），可以计算椭球上某一点的坐标。但这个点并不是我们真正要求的点，我们要求的点P(B,L,H)是椭球面沿法向量向上H高度的点：



P点在椭球面上的点为P_0，那么根据矢量相加的性质，有：

$$P = P_0 + H \cdot n \tag{6}$$

其中，P_0也就是式(5)，而n是P_0在椭球面的法线单位矢量。

矢量在任意位置的方向都是一样的，那么我们可以假设存在一个单位球(球的半径为单位1)，将法线单位矢量移动到球心位置，可得法线单位矢量为：

$$n = \begin{bmatrix} cosBcosL \\ cosBsinL \\ sinB \end{bmatrix} \tag{7}$$

因此有：

$$P = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} (N+H)cosBcosL \\ (N+H)cosBsinL \\ [N(1-e^2)+H]sinB \end{bmatrix} \tag{8}$$

其中：

$$N = \frac{a}{\sqrt{1 - e^2 sin^2 B}} \tag{9}$$

XYZ->BLH

根据式（8），可知：

$$\frac{Y}{X}=\frac{(N+H)cosBsinL}{(N+H)cosBcosL}=tanL$$

因此有：

$$L=arctan(\frac{Y}{X}) \tag{10}$$

不过纬度B就不是那么好算了，首先需要计算法线Pn在赤道两侧的长度。根据图1，有：

$$y=PQsinB$$

与式（4-3）比较可得：

$$PQ=N(1-e^2)$$

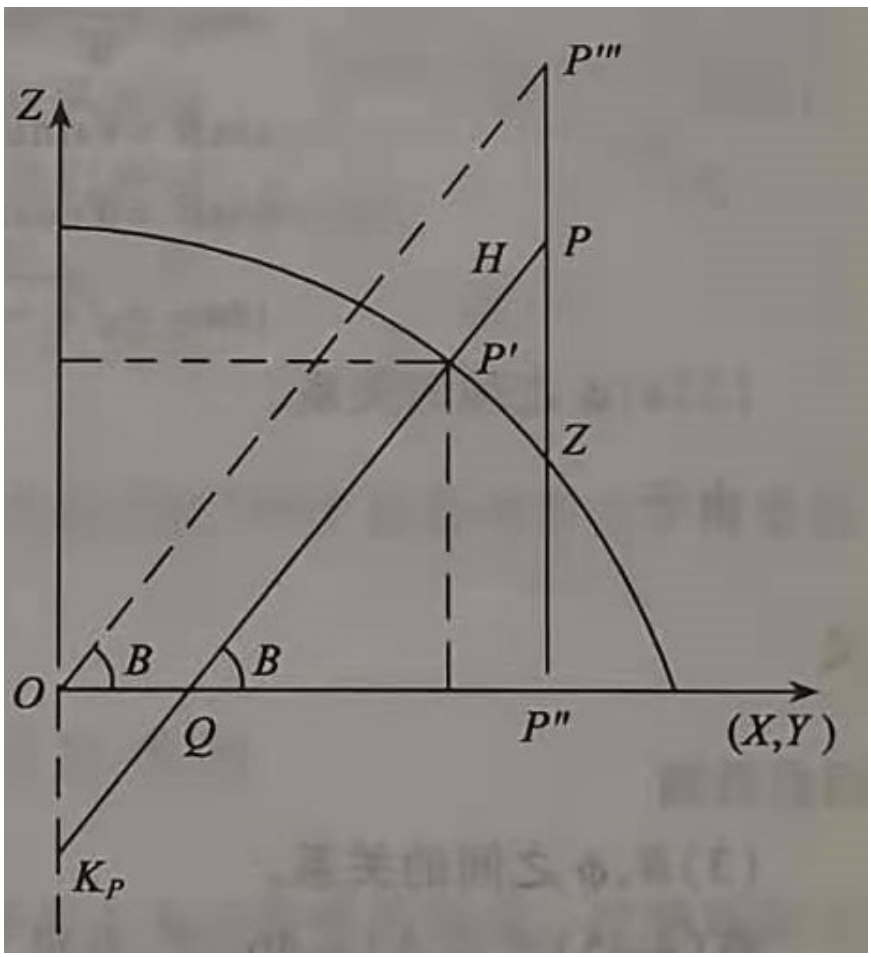
显然，由于：

$$Pn=N=PQ+Qn$$

有：

$$Qn=Ne^2$$

接下来如下图所示，对图1做辅助线：



有：

$$\begin{cases} PP'' = Z \\ OP'' = \sqrt{x^2 + y^2} \\ PP''' = OK_p = QK_p sinB = Ne^2 sinB \\ P''P''' = PP''' + PP'' \end{cases}$$

因而可得：

$$tanB = \frac{Z + Ne^2 sinB}{\sqrt{x^2 + y^2}} \tag{11}$$

这个式子两边都有待定量B，需要用迭代法进行求值。具体可参看代码实现，初始的待定值可取 $tanB = \frac{z}{\sqrt{x^2+y^2}}$。

大地纬度B已知，那么求高度H就非常简单了，直接根据式（8）中的第三式逆推可得：

$$H = \frac{Z}{sinB} - N(1 - e^2) \tag{12}$$

汇总三式，可得：

$$\begin{cases} L = arctan(\frac{Y}{X}) \\ tanB = \frac{Z+Ne^2 sinB}{\sqrt{x^2+y^2}} \\ H = \frac{Z}{sinB} - N(1 - e^2) \end{cases}$$

3、实现

根据前面的推导过程，具体的C/C++代码实现如下：

其最关键的还是计算大地纬度B时的迭代过程，其余的计算都只是套公式。数值计算中的很多算法都是采用迭代趋近的方法来趋近一个最佳解。

```
#include <iostream>

using namespace std;

const double epsilon = 0.000000000000001;
const double pi = 3.14159265358979323846;
const double d2r = pi / 180;
const double r2d = 180 / pi;

const double a = 6378137.0;                 //椭球长半轴
const double f_inverse = 298.257223563;                         //扁率倒数
const double b = a - a / f_inverse;
//const double b = 6356752.314245;                          //椭球短半轴

const double e = sqrt(a * a - b * b) / a;
```

```
16
17 void Blh2Xyz(double &x, double &y, double &z)//经纬高转地心地固坐标系
18 {
19         double L = x * d2r;
20         double B = y * d2r;
21         double H = z;
22
23         double N = a / sqrt(1 - e * e * sin(B) * sin(B));
24         x = (N + H) * cos(B) * cos(L);
25         y = (N + H) * cos(B) * sin(L);
26         z = (N * (1 - e * e) + H) * sin(B);
27 }
28
29 void Xyz2Blh(double &x, double &y, double &z)
30 {
31         double tmpX =  x;
32         double temY = y ;
33         double temZ = z;
34
35         double curB = 0;
36         double N = 0;
37         double calB = atan2(temZ, sqrt(tmpX * tmpX + temY * temY));
38
39         int counter = 0;
40         while (abs(curB - calB) * r2d > epsilon  && counter < 25)
41         {
42                 curB = calB;
43                 N = a / sqrt(1 - e * e * sin(curB) * sin(curB));
44                 calB = atan2(temZ + N * e * e * sin(curB), sqrt(tmpX * tmpX + te
45                 counter++;
46         }
47
48         x = atan2(temY, tmpX) * r2d;
49         y = curB * r2d;
50         z = temZ / sin(curB) - N * (1 - e * e);
51 }
52
53 int main()
54 {
55         double x = 113.6;
56         double y = 38.8;
57         double z = 100;
58
59         printf("原大地经纬度坐标：%.10lf\t%.10lf\t%.10lf\n", x, y, z);
60         Blh2Xyz(x, y, z);
61
62         printf("地心地固直角坐标：%.10lf\t%.10lf\t%.10lf\n", x, y, z);
```

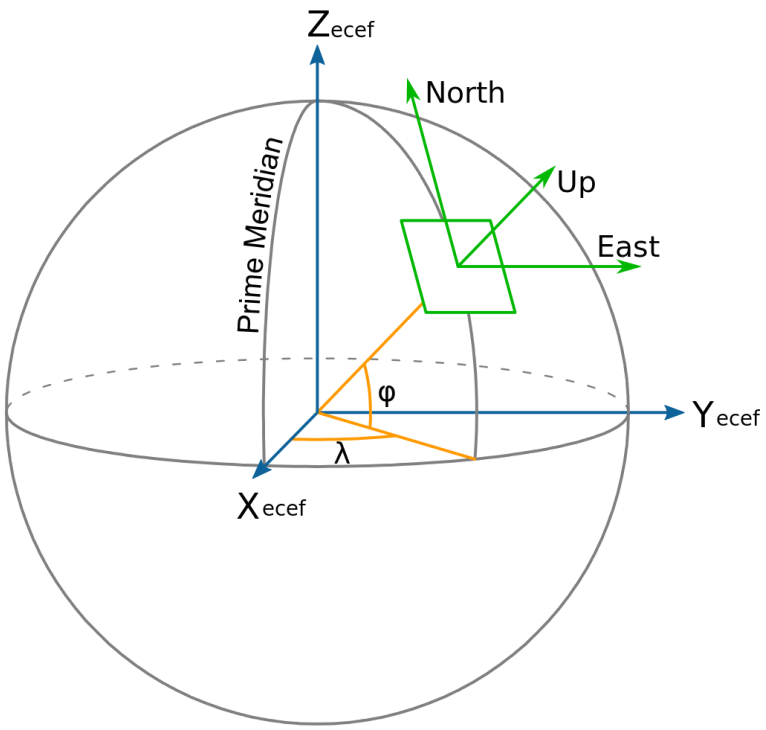
```
63          Xyz2Blh(x, y, z);
64          printf("转回大地经纬度坐标：%.10lf\t%.10lf\t%.10lf\n", x, y, z);
65  }
66
```

三、地心地固坐标系(ECEF)与站心坐标系(ENU)的转换

1、概述

我们知道，基于地心坐标系的坐标都是很大的值，这样的值是不太方便进行空间计算的，所以很多时候可以选取一个站心点，将这个很大的值变换成一个较小的值。以图形学的观点来看，地心坐标可以看作是世界坐标，站心坐标可以看作局部坐标。

站心坐标系以一个站心点为坐标原点，当把坐标系定义为X轴指东、Y轴指北，Z轴指天，就是ENU（东北天）站心坐标系。这样，从地心地固坐标系转换成的站心坐标系，就会成为一个符合常人对地理位置认知的局部坐标系。同时，只要站心点位置选的合理（通常可选取地理表达区域的中心点），表达的地理坐标都会是很小的值，非常便于空间计算。



注意站心天向(法向量)与赤道面相交不一定会经过球心

2、原理

WebGL简易教程(五)：图形变换(模型、视图、投影变换)

令选取的站心点为P，其大地经纬度坐标为 $(B_p,\ L_p,\ H_p)$ ，对应的地心地固坐标系为 $(X_p,\ Y_p,\ Z_p)$ 。地心地固坐标系简称为ECEF，站心坐标系简称为ENU。

2.1. 平移

通过第一节的图可以看出，ENU要转换到ECEF，一个很明显的图形操作是平移变换，将站心移动到地心。根据站心点P在地心坐标系下的坐标（X_p，Y_p，Z_p），可以很容易推出ENU转到ECEF的平移矩阵：

$$T = \begin{bmatrix} 1 & 0 & 0 & X_p \\ 0 & 1 & 0 & Y_p \\ 0 & 0 & 1 & Z_p \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

反推之，ECEF转换到ENU的平移矩阵就是T的逆矩阵：

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 & -X_p \\ 0 & 1 & 0 & -Y_p \\ 0 & 0 & 1 & -Z_p \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2.2. 旋转

另外一个需要进行的图形变换是旋转变换，其旋转变换矩阵根据P点所在的经度L和纬度B确定。这个旋转变换有点难以理解，需要一定的空间想象能力，但是可以直接给出如下结论：

1. 当从ENU转换到ECEF时，需要先旋转再平移，旋转是先绕X轴旋转$(\frac{pi}{2} - B)$，再绕Z轴旋转$(\frac{pi}{2} + L)$
2. 当从ECEF转换到ENU时，需要先平移再旋转，旋转是先绕Z轴旋转$-(\frac{pi}{2} + L)$，再绕X轴旋转$-(\frac{pi}{2} - B)$

根据我在《WebGL简易教程(五)：图形变换(模型、视图、投影变换)》提到的旋转变换，绕X轴旋转矩阵为：

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sin\theta & 0 \\ 0 & sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

绕Z轴旋转矩阵为：

$$R_z = \begin{bmatrix} cos\theta & -sin\theta & 0 & 0 \\ sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

从ENU转换到ECEF的旋转矩阵为：

$$pi \qquad pi$$

$$R = R_z(\frac{\pi}{2} + L) \cdot R_x(\frac{\pi}{2} - B) \tag{1}$$

根据三角函数公式：

$$\begin{aligned} sin(\pi/2 + \alpha) &= cos\alpha \\ sin(\pi/2 - \alpha) &= cos\alpha \\ cos(\pi/2 + \alpha) &= -sin\alpha \\ cos(\pi/2 - \alpha) &= sin\alpha \end{aligned}$$

有：

$$R_z(\frac{pi}{2} + L) = \begin{bmatrix} -sinL & -cosL & 0 & 0 \\ cosL & -sinL & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

$$R_x(\frac{pi}{2} - B) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & sinB & -cosB & 0 \\ 0 & cosB & sinB & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

将(2)、(3)带入(1)中，则有：

$$R = \begin{bmatrix} -sinL & -sinBcosL & cosBcosL & 0 \\ cosL & -sinBsinL & cosBsinL & 0 \\ 0 & cosB & sinB & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

而从ECEF转换到ENU的旋转矩阵为：

$$R^{-1} = R_x(-(\frac{pi}{2} - B)) \cdot R_z(-(\frac{pi}{2} + L)) \tag{5}$$

旋转矩阵是正交矩阵，根据正交矩阵的性质:正交矩阵的逆矩阵等于其转置矩阵，那么可直接得：

$$R^{-1} = \begin{bmatrix} -sinL & cosL & 0 & 0 \\ -sinBcosL & -sinBsinL & cosB & 0 \\ cosBcosL & cosBsinL & sinB & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

2.3. 总结

将上述公式展开，可得从ENU转换到ECEF的图形变换矩阵为：

$$M = T \cdot R = \begin{bmatrix} 1 & 0 & 0 & X_p \\ 0 & 1 & 0 & Y_p \\ 0 & 0 & 1 & Z_p \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -sinL & -sinBcosL & cosBcosL & 0 \\ cosL & -sinBsinL & cosBsinL & 0 \\ 0 & cosB & sinB & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

而从ECEF转换到ENU的图形变换矩阵为：

$$M^{-1} = R^{-1} * T^{-1} = \begin{bmatrix} -sinL & cosL & 0 & 0 \\ -sinBcosL & -sinBsinL & cosB & 0 \\ cosBcosL & cosBsinL & sinB & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -X_p \\ 0 & 1 & 0 & -Y_p \\ 0 & 0 & 1 & -Z_p \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3、实现

接下来用代码实现这个坐标转换，选取一个站心点，以这个站心点为原点，获取某个点在这个站心坐标系下的坐标：

```cpp
#include <iostream>
#include <eigen3/Eigen/Eigen>

#include <osgEarth/GeoData>

using namespace std;

const double epsilon = 0.000000000000001;
const double pi = 3.14159265358979323846;
const double d2r = pi / 180;
const double r2d = 180 / pi;

const double a = 6378137.0;                 //椭球长半轴
const double f_inverse = 298.257223563;                         //扁率倒数
const double b = a - a / f_inverse;
//const double b = 6356752.314245;                          //椭球短半轴

const double e = sqrt(a * a - b * b) / a;

void Blh2Xyz(double &x, double &y, double &z)
{
        double L = x * d2r;
        double B = y * d2r;
        double H = z;
```

```
25
26          double N = a / sqrt(1 - e * e * sin(B) * sin(B));
27          x = (N + H) * cos(B) * cos(L);
28          y = (N + H) * cos(B) * sin(L);
29          z = (N * (1 - e * e) + H) * sin(B);
30  }
31
32  void Xyz2Blh(double &x, double &y, double &z)
33  {
34          double tmpX =  x;
35          double temY = y ;
36          double temZ = z;
37
38          double curB = 0;
39          double N = 0;
40          double calB = atan2(temZ, sqrt(tmpX * tmpX + temY * temY));
41
42          int counter = 0;
43          while (abs(curB - calB) * r2d > epsilon  && counter < 25)
44          {
45                  curB = calB;
46                  N = a / sqrt(1 - e * e * sin(curB) * sin(curB));
47                  calB = atan2(temZ + N * e * e * sin(curB), sqrt(tmpX * tmpX + te
48                  counter++;
49          }
50
51          x = atan2(temY, tmpX) * r2d;
52          y = curB * r2d;
53          z = temZ / sin(curB) - N * (1 - e * e);
54  }
55
56  void TestBLH2XYZ()
57  {
58          //double x = 113.6;
59  //double y = 38.8;
60  //double z = 100;
61  //
62  //printf("原大地经纬度坐标: %.10lf\t%.10lf\t%.10lf\n", x, y, z);
63  //Blh2Xyz(x, y, z);
64
65  //printf("地心地固直角坐标: %.10lf\t%.10lf\t%.10lf\n", x, y, z);
66  //Xyz2Blh(x, y, z);
67  //printf("转回大地经纬度坐标: %.10lf\t%.10lf\t%.10lf\n", x, y, z);
68
69          double x = -2318400.6045575836;
70          double y = 4562004.801366804;
71          double z = 3794303.054150639;
```

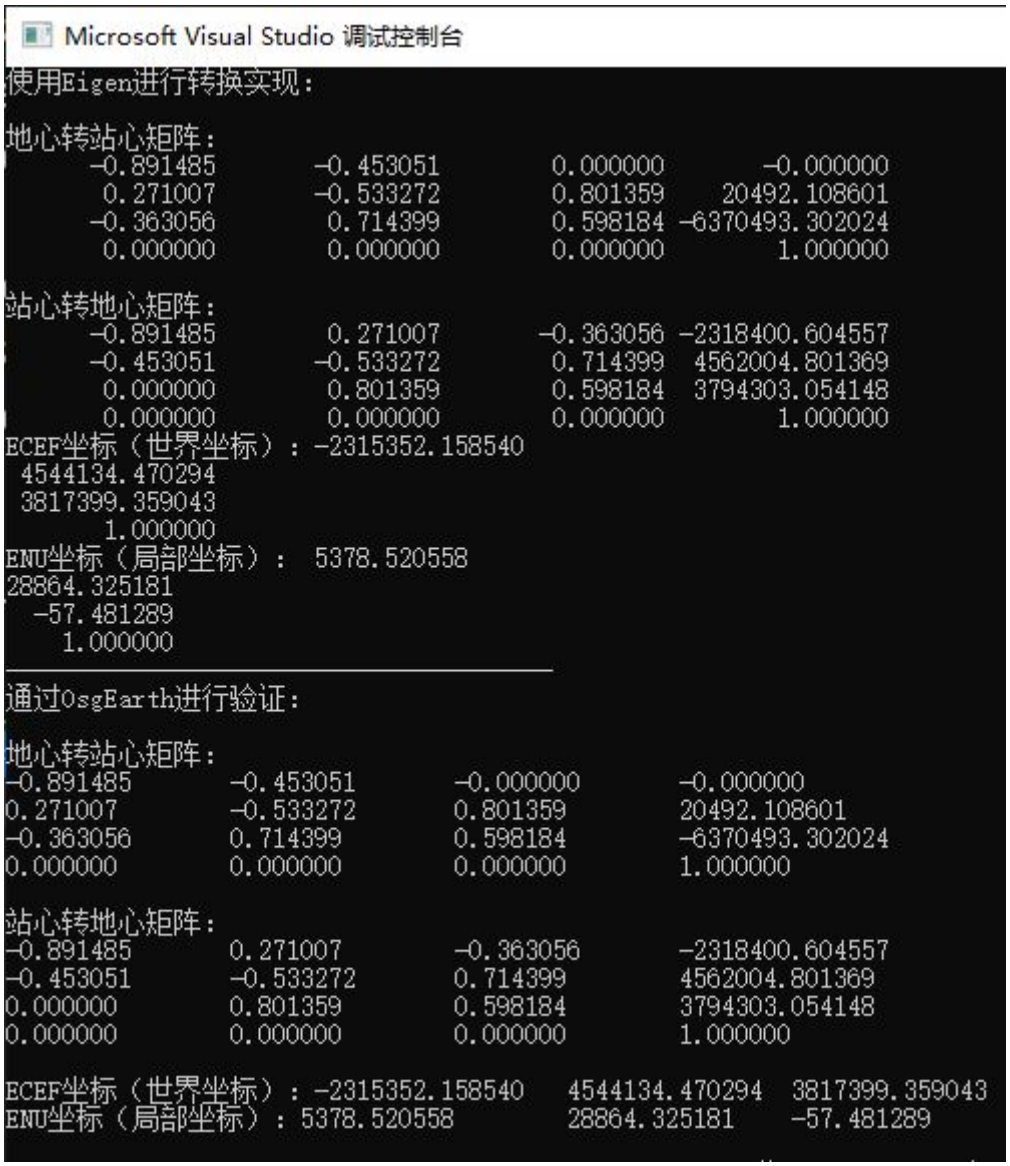
```
72
73          //116.9395751953      36.7399177551
74
75          printf("地心地固直角坐标：%.10lf\t%.10lf\t%.10lf\n", x, y, z);
76          Xyz2Blh(x, y, z);
77          printf("转回大地经纬度坐标：%.10lf\t%.10lf\t%.10lf\n", x, y, z);
78  }
79
80  void CalEcef2Enu(Eigen::Vector3d& topocentricOrigin, Eigen::Matrix4d& resultMat)
81  {
82          double rzAngle = -(topocentricOrigin.x() * d2r + pi / 2);
83          Eigen::AngleAxisd rzAngleAxis(rzAngle, Eigen::Vector3d(0, 0, 1));
84          Eigen::Matrix3d rZ = rzAngleAxis.matrix();
85
86          double rxAngle = -(pi / 2 - topocentricOrigin.y() * d2r);
87          Eigen::AngleAxisd rxAngleAxis(rxAngle, Eigen::Vector3d(1, 0, 0));
88          Eigen::Matrix3d rX = rxAngleAxis.matrix();
89
90          Eigen::Matrix4d rotation;
91          rotation.setIdentity();
92          rotation.block<3, 3>(0, 0) = (rX * rZ);
93          //cout << rotation << endl;
94
95          double tx = topocentricOrigin.x();
96          double ty = topocentricOrigin.y();
97          double tz = topocentricOrigin.z();
98          Blh2Xyz(tx, ty, tz);
99          Eigen::Matrix4d translation;
100         translation.setIdentity();
101         translation(0, 3) = -tx;
102         translation(1, 3) = -ty;
103         translation(2, 3) = -tz;
104
105         resultMat = rotation * translation;
106 }
107
108 void CalEnu2Ecef(Eigen::Vector3d& topocentricOrigin, Eigen::Matrix4d& resultMat)
109 {
110         double rzAngle = (topocentricOrigin.x() * d2r + pi / 2);
111         Eigen::AngleAxisd rzAngleAxis(rzAngle, Eigen::Vector3d(0, 0, 1));
112         Eigen::Matrix3d rZ = rzAngleAxis.matrix();
113
114         double rxAngle = (pi / 2 - topocentricOrigin.y() * d2r);
115         Eigen::AngleAxisd rxAngleAxis(rxAngle, Eigen::Vector3d(1, 0, 0));
116         Eigen::Matrix3d rX = rxAngleAxis.matrix();
117
118         Eigen::Matrix4d rotation;
```

```
        rotation.setIdentity();
        rotation.block<3, 3>(0, 0) = (rZ * rX);
        //cout << rotation << endl;

        double tx = topocentricOrigin.x();
        double ty = topocentricOrigin.y();
        double tz = topocentricOrigin.z();
        Blh2Xyz(tx, ty, tz);
        Eigen::Matrix4d translation;
        translation.setIdentity();
        translation(0, 3) = tx;
        translation(1, 3) = ty;
        translation(2, 3) = tz;

        resultMat = translation * rotation;
}

void TestXYZ2ENU()
{
        double L = 116.9395751953;
        double B = 36.7399177551;
        double H = 0;

        cout << fixed << endl;
        Eigen::Vector3d topocentricOrigin(L, B, H);
        Eigen::Matrix4d wolrd2localMatrix;
        CalEcef2Enu(topocentricOrigin, wolrd2localMatrix);
        cout << "地心转站心矩阵: " << endl;
        cout << wolrd2localMatrix << endl<<endl;

        cout << "站心转地心矩阵: " << endl;
        Eigen::Matrix4d local2WolrdMatrix;
        CalEnu2Ecef(topocentricOrigin, local2WolrdMatrix);
        cout << local2WolrdMatrix << endl;

        double x = 117;
        double y = 37;
        double z = 10.3;
        Blh2Xyz(x, y, z);

        cout << "ECEF坐标（世界坐标）: ";
        Eigen::Vector4d xyz(x, y, z, 1);
        cout << xyz << endl;

        cout << "ENU坐标（局部坐标）: ";
        Eigen::Vector4d enu = wolrd2localMatrix * xyz;
        cout << enu << endl;
```

```
166 }
167
168 void TestOE()
169 {
170         double L = 116.9395751953;
171         double B = 36.7399177551;
172         double H = 0;
173
174         osgEarth::SpatialReference *spatialReference = osgEarth::SpatialReferenc
175         osgEarth::GeoPoint centerPoint(spatialReference, L, B, H);
176
177         osg::Matrixd worldToLocal;
178         centerPoint.createWorldToLocal(worldToLocal);
179
180         cout << fixed << endl;
181         cout << "地心转站心矩阵: " << endl;
182         for (int i = 0; i < 4; i++)
183         {
184                 for (int j = 0; j < 4; j++)
185                 {
186                         printf("%lf\t", worldToLocal.ptr()[j * 4 + i]);
187                 }
188                 cout << endl;
189         }
190         cout << endl;
191
192         osg::Matrixd localToWorld;
193         centerPoint.createLocalToWorld(localToWorld);
194
195         cout << "站心转地心矩阵: " << endl;
196         for (int i = 0; i < 4; i++)
197         {
198                 for (int j = 0; j < 4; j++)
199                 {
200                         printf("%lf\t", localToWorld.ptr()[j * 4 + i]);
201                 }
202                 cout << endl;
203         }
204         cout << endl;
205
206         double x = 117;
207         double y = 37;
208         double z = 10.3;
209         osgEarth::GeoPoint geoPoint(spatialReference, x, y, z);
210
211         cout << "ECEF坐标（世界坐标）: ";
212         osg::Vec3d out_world;
```

```
        geoPoint.toWorld(out_world);
        cout << out_world.x() <<'\t'<< out_world.y() << '\t' << out_world.z() <<

        cout << "ENU坐标（局部坐标）：";
        osg::Vec3d localCoord = worldToLocal.preMult(out_world);
        cout << localCoord.x() << '\t' << localCoord.y() << '\t' << localCoord.z
}

int main()
{
        //TestBLH2XYZ();

        cout << "使用Eigen进行转换实现：" << endl;
        TestXYZ2ENU();

        cout <<"--------------------------------------"<< endl;
        cout << "通过OsgEarth进行验证：" << endl;
        TestOE();
}

```

这个示例先用Eigen矩阵库，计算了坐标转换需要的矩阵和转换结果；然后通过osgEarth进行了验证，两者的结果基本一致。运行结果如下：


Microsoft Visual Studio 调试控制台
使用Eigen进行转换实现：
地心转站心矩阵：
-0.891485 -0.453051 0.000000 -0.000000
0.271007 -0.533272 0.801359 20492.108601
-0.363056 0.714399 0.598184 -6370493.302024
0.000000 0.000000 0.000000 1.000000
站心转地心矩阵：
-0.891485 0.271007 -0.363056 -2318400.604557
-0.453051 -0.533272 0.714399 4562004.801369
0.000000 0.801359 0.598184 3794303.054148
0.000000 0.000000 0.000000 1.000000
ECEF坐标（世界坐标）：-2315352.158540
4544134.470294
3817399.359043
1.000000
ENU坐标（局部坐标）： 5378.520558
28864.325181
-57.481289
1.000000
通过OsgEarth进行验证：
地心转站心矩阵：
-0.891485 -0.453051 -0.000000 -0.000000
0.271007 -0.533272 0.801359 20492.108601
-0.363056 0.714399 0.598184 -6370493.302024
0.000000 0.000000 0.000000 1.000000
站心转地心矩阵：
-0.891485 0.271007 -0.363056 -2318400.604557
-0.453051 -0.533272 0.714399 4562004.801369
0.000000 0.801359 0.598184 3794303.054148
0.000000 0.000000 0.000000 1.000000
ECEF坐标（世界坐标）：-2315352.158540 4544134.470294 3817399.359043
ENU坐标（局部坐标）：5378.520558 28864.325181 -57.481289

四、坐标系介绍

4.1 惯性坐标系与地球坐标系

惯性坐标系（或空固坐标系）在空间静止或匀速直线运动的坐标系。实际操作中很难建立惯性坐标系，方便用于天文学中对于描述星系中的卫星运行轨道。

地球坐标系（或地固坐标系）是固定在地球上而随地球一起在空间做自转和公转运动的坐标系。

4.2 常用地球坐标系类型

建立地球坐标系需要确定的地理基准，如直角坐标系中的赤道、北极，大地坐标系的地球椭球模型。地球的北极点并不是固定不变的，地球的形状也不是完整的椭球，所以基于不同时间段不同测量方式确定的各国的坐标系参数会有微小的偏差。各国坐标系参数可在国家测绘局或宇航局的公开文件中查询。而常用的地球坐标系种类有以下几种：

4.2.1 直角坐标系（ECEF）

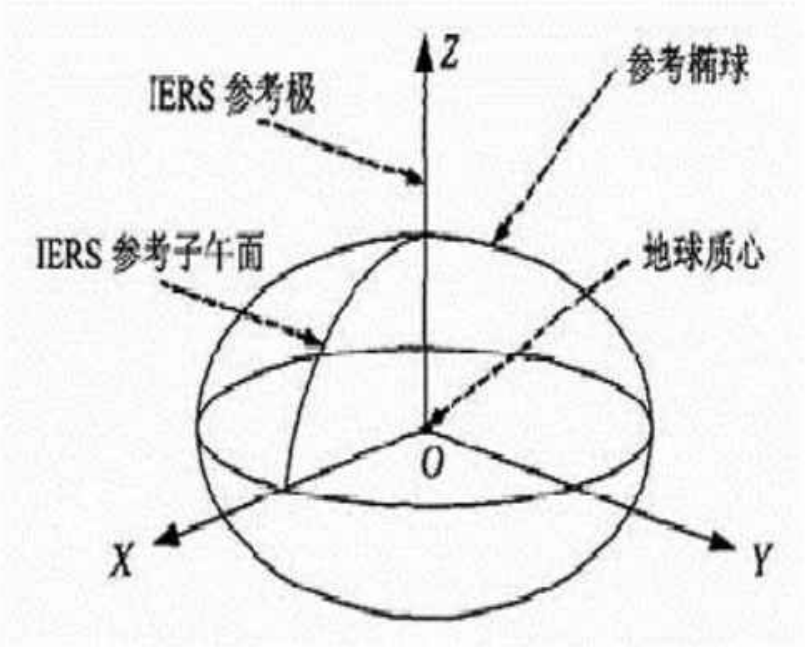
地心地固直角坐标系构建方式：

以地心作为坐标原点；

Z 轴指向地球北极点；

X 轴指向参考子午面（英国伦敦格林尼治子午面）与赤道的一个交点；

X、Y、Z 三轴构成右手直角坐标系。



受地球自转和公转的影响，地球南北两极点并不是固定的；而不同测绘组织测量的地球南北两极点也存在位置偏差。按照不同时期或不同组织发布的地球极

点位置建立的地心地固直角坐标系框架一致，但有微小的偏差。如 GPS-ICD 文件中说明 GPS 的直角坐标系是以 IERS（国际地球自转服务组织）WGS84 椭球体对应的地级为基准的；如 BDS-ICD 文件中说明 BDS 的直角坐标系是以 IERS（国际地球自转服务组织）定义的地级为基准的；两者定义的直角坐标系，框架一致，但有微小的偏差。

4.2.2 大地坐标系（LLA）

首先需要先建立与地球表面最吻合的地球基准面，要求是以地球质心为原点建立一个与地球水准面（假想海平面）之间的高度差的平方和最小的基准椭球体。由于测量或建模方式的不同，不同国家或地区建立的地心地固大地坐标系之间框

坐标系统	坐标原点	长半轴(m)	地球引力常数(含大气层)(m^3/s^2)	扁率	自转角速度(rad/s)
WGS-84	地球质心	6 378 137.0	$398\ 600.5 \times 10^9$	1/298.257 223 563	$7.2921\ 15 \times 10^{-5}$
CGCS2000	地球质心	6 378 137.0	$398\ 600.441\ 8 \times 10^9$	1/298.257 222 101	$7.2921\ 15 \times 10^{-5}$
PZ-90	地球质心	6 378 136.0	$398\ 600.44 \times 10^9$	1/298.257 839 303	$7.292\ 115 \times 10^{-5}$

架一致，但椭球参数有微小的偏差。

建立地球基准面后，

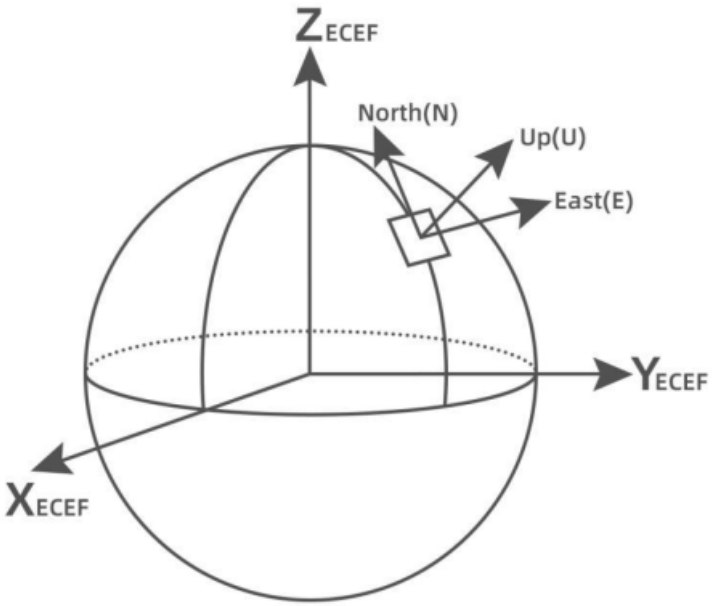
定义大地纬度（简称纬度）是所在位置的基准椭球面法线与赤道面之间的夹角，赤道以北为正，赤道以南为负；

定义大地经度（简称经度）是所在位置的子午面与参考子午面的夹角，参考子午面以东为正，参考子午面以西为负；

定义大地高度（简称大地高）是所在位置到基准托球面法线的距离。查看 3.5 大地高/海拔高/高程异常值。

4.2.3 站心坐标系（ENU）

站心坐标系（或用户坐标系或东北天坐标系）是以用户所在位置为坐标原点，坐标轴分别垂直的指向东向、北向和天向的坐标系。可通过平移和旋转变换成地心直角坐标系。

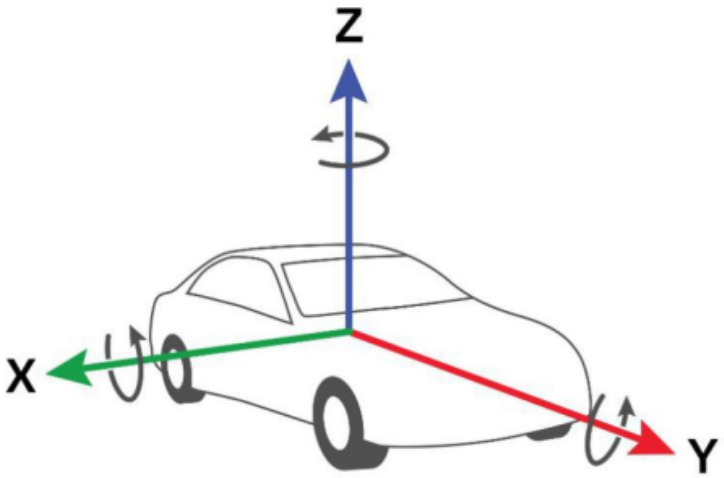


4.2.4 特殊站心坐标系——组合导航载体坐标系（Body Frame）

载体坐标系(body frame)，又称为运动载体坐标系，由导航中要解算的导航对象的原点和姿态确定。载体坐标系的原点即为导航结果所描述的对象，坐标轴却与载体固联。

常用的整机坐标系（Body）和车体坐标系（Vehicle）都可理解为载体坐标系的一种，因定义或应用场景的不同而有所差别，在此主要介绍我们目前常用的定义。

通常定义 Y 轴为前向（即正常航行的前进方向），Z 轴为天向（即向下方向），X 轴为右向，并由三个坐标轴组成正交坐标系。对于角运动来说，载体坐标系的轴也被称为滚动、俯仰和偏航轴，其中 Y 轴方向为滚动轴，X 轴方向为俯仰轴，而 Z 轴方向为偏航轴，均符合右手螺旋法则规定。



4.3 坐标系转换

在定位计算中经常涉及到直角坐标和大地坐标之间的相互转换。

4.3.1 lla2ecef

从大地坐标（φ，λ，h）到直角坐标（x，y，z）的转换，需要先计算参数，使用基准椭球体的长半径 a 和短半径 b，计算椭球偏心率 e，$e=\frac{a^2-b^2}{a^2}$，再计算基准椭球体的曲率半径 N，$N=\frac{a}{\sqrt{1-e^2\sin\varphi^2}}$，再使用公式如下：

$$\begin{cases} x=(N+h)\cos\varphi\cos\lambda \\ y=(N+h)\cos\varphi\sin\lambda \\ z=(N(1-e^2)+h)\sin\varphi \end{cases}$$

4.3.2 ecef2lla

从直角坐标（x，y，z）到大地坐标（φ，λ，h）的转换，需要先计算参数，使用基准椭球体的长半径 a 和短半径 b，计算椭球偏心率 e，$e=\frac{a^2-b^2}{a^2}$，再计算基准椭球体的曲率半径 N，$N=\frac{a}{\sqrt{1-e^2\sin\varphi^2}}$，中间变量 p，$p=\sqrt{x^2+y^2}$，再使用公式如下：

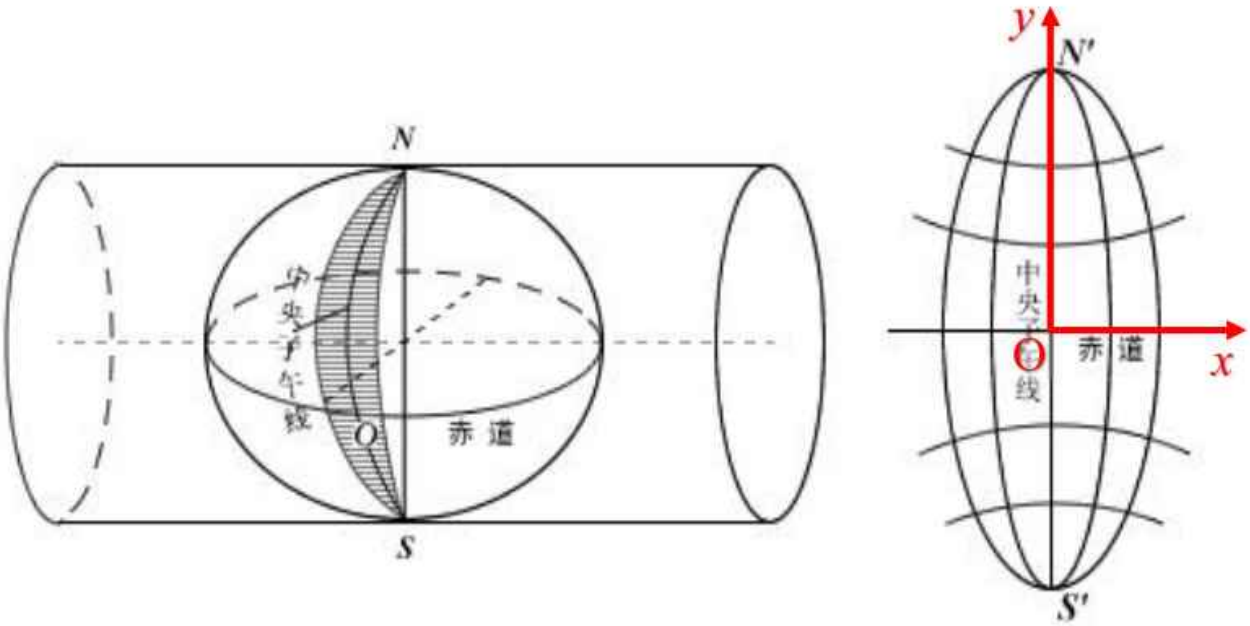
$$\begin{cases} \lambda=\tan^{-1}\left(\dfrac{y}{x}\right) \\ h=\dfrac{p}{\cos\varphi}-N \\ \varphi=\tan^{-1}\left[\dfrac{z}{p}\left(1-e^2\dfrac{N}{N+h}\right)^{-1}\right] \end{cases}$$

4.4 投影坐标系

投影坐标系是在二维平面中进行定义的。基于地理坐标系，将其三维曲面以数学转换的方式创建平面坐标系的过程是投影，投影会导致形状、距离或方向发生变形，不同的投影方式会引起不同类型的变形。

4.4.1 高斯投影

高斯投影的三个条件：正形条件，中央子午线投影为一直线，中央子午线投影后长度不变。为了有效控制长度变形，投影成若干个以中央子午线为中心的狭长带。高斯平面直角坐标的 x 轴为中央子午线的投影，Y 轴为赤道的投影，原点是中央子午线与赤道的焦点。



球体位置　　投影⟶　　高斯坐标系

4.4.2 横轴墨卡托投影

横轴墨卡托投影是沿经线的圆柱投影，中央经线经过的区域变形较少，适于南北分布的地区作图。经线等距平行，纬线也彼此平行，但理赤道越近，间距越小，常用于 Web 制图站点。

五、导远INS570#系列车载组合导航定位系统输出说明

受控编号：AG2.508.002

组合导航安装及配置步骤

INS Installation and Configuration Instructions

适用INS570D、INS570L系列产品

For INS570D 、INS570L series products

INS570#ç³»åⵈçº§åⵈå¯¼èªåºèªåⵈéⵈç½®æ¥éª¤ V1.6.pdf

受控编号：AG2.508.011SS

INS570#系列车载组合导航定位系统

使用手册

手册适用：INS570D 高精度车载组合导航定位系统、

INS570L-6DAA 标准车载组合导航定位系统

INS570#ç³»åⵈè½¦è½½çº§åⵈå¯¼èªåºèä½ⵈç³»çº§ ä½¿ç¨¨æⵈⵈåⵈⵈV1.01.pdf