



Hitchhiker's Guide to Coordinate Systems and Spatial References

Melita Kennedy

Outline



- Coordinate systems
 - geographic/projected/vertical
- Geographic/datum transformations
- Common coordinate systems
 - UTM and State Plane
- Spatial references

Spatial reference environment



- Geometry library
- Uses SpatialReferenceEnvironment
 - Singleton object
 - Keeps track of multiple spatialrefs, coordinate systems
- SpatialReferenceFactory
 - Access existing coordinate systems
 - Create lists for UI
 - Convert between object and WKT

Initializing SpatialReferenceEnvironment



```
ISpatialReferenceFactory2 pSRF = new  
    SpatialReferenceEnvironmentClass();
```

Coordinate system

Projected coordinate system

Projection

Projection parameters

Linear Unit

Geographic coordinate system

Datum

Spheroid

Prime Meridian

Angular Unit



Projected versus geographic

- Earth is three-dimensional
- Map (screen) is 2-D
- Geographic coordinate system (datum) locates in 3-D
- Map Projection converts 3-D to 2-D
- 3-D to 2-D causes distortions

Geographic coordinate system



- Defines locations on 3-D surface
- Latitude/longitude measure angles
- Units are degrees (or grads)
- Not a projection!

Geographic Coordinate System



(gcs, geogcs)

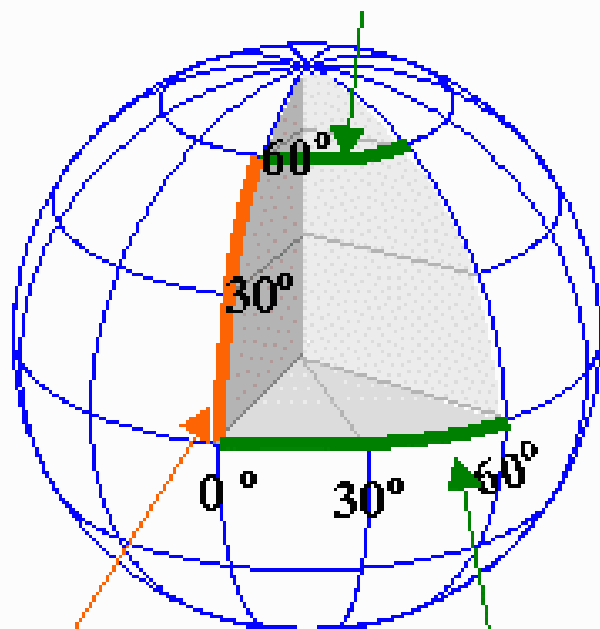
- Datum (**European Datum 1950**)
 - Spheroid (**International 1924**)
- Prime Meridian (**Greenwich**)
- Angular unit of measure (**Degrees**)

WKT version



```
GEOGCS["GCS_WGS_1984",  
  DATUM["D_WGS_1984",  
    SPHEROID["WGS_1984",6378137,  
      298.257223563]],  
  PRIMEM["Greenwich",0],  
  UNIT["Degree",0.017453292519943295]],
```

Distance of 60° longitude
 at 60° latitude



Distance of 60°
 latitude

Distance of 60°
 longitude at
 Equator

- Meridians converge

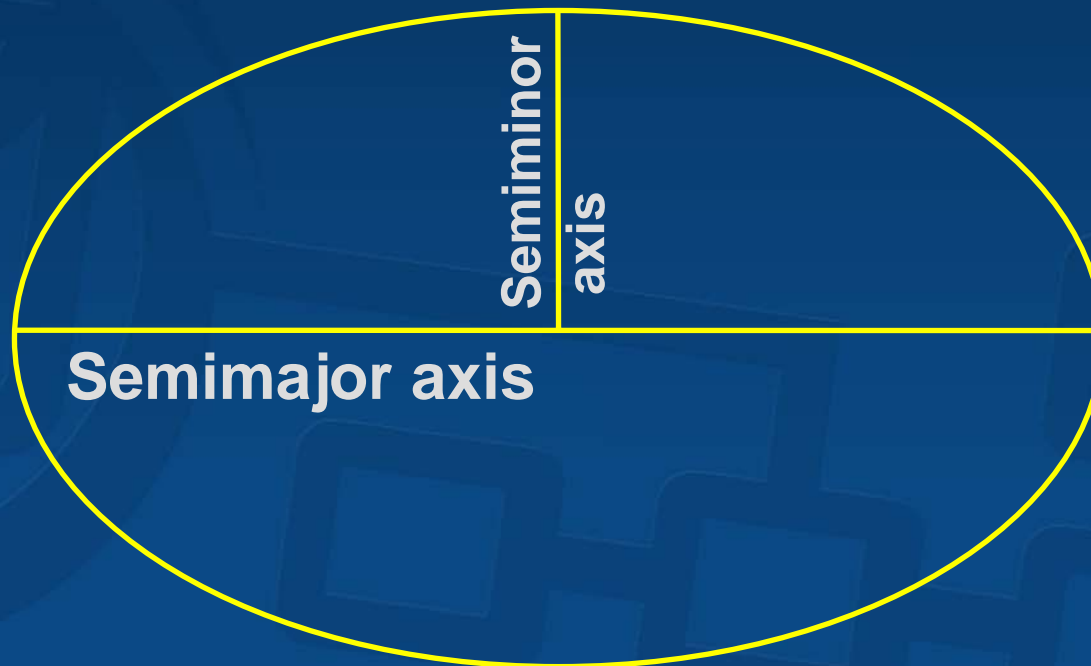
1° longitude

at Equator = 111 km
 at 60° lat. = 56 km

Background geometry



- Rotating a circle or ellipse creates a sphere or spheroid
- Defines the size and shape of the Earth



Datums



- Reference frame for locating points on Earth's surface
- Defines origin & orientation of latitude/longitude lines
- Defined by spheroid and spheroid's position relative to Earth's center

Creating a Datum

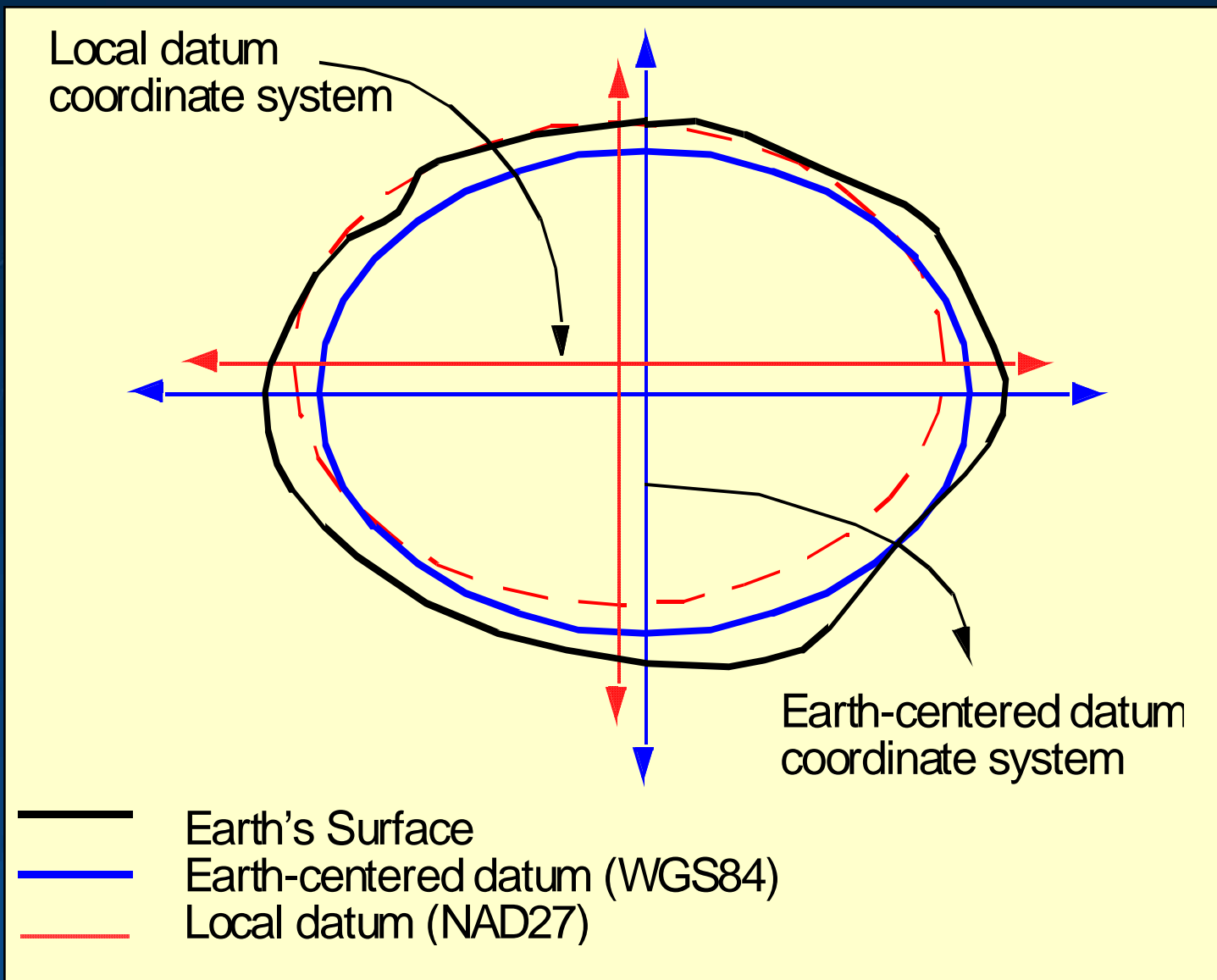


- Pick a spheroid
- Pick a point on the Earth's surface
- All other control points are located relative to the origin point
- The datum's center may not coincide with the Earth's center

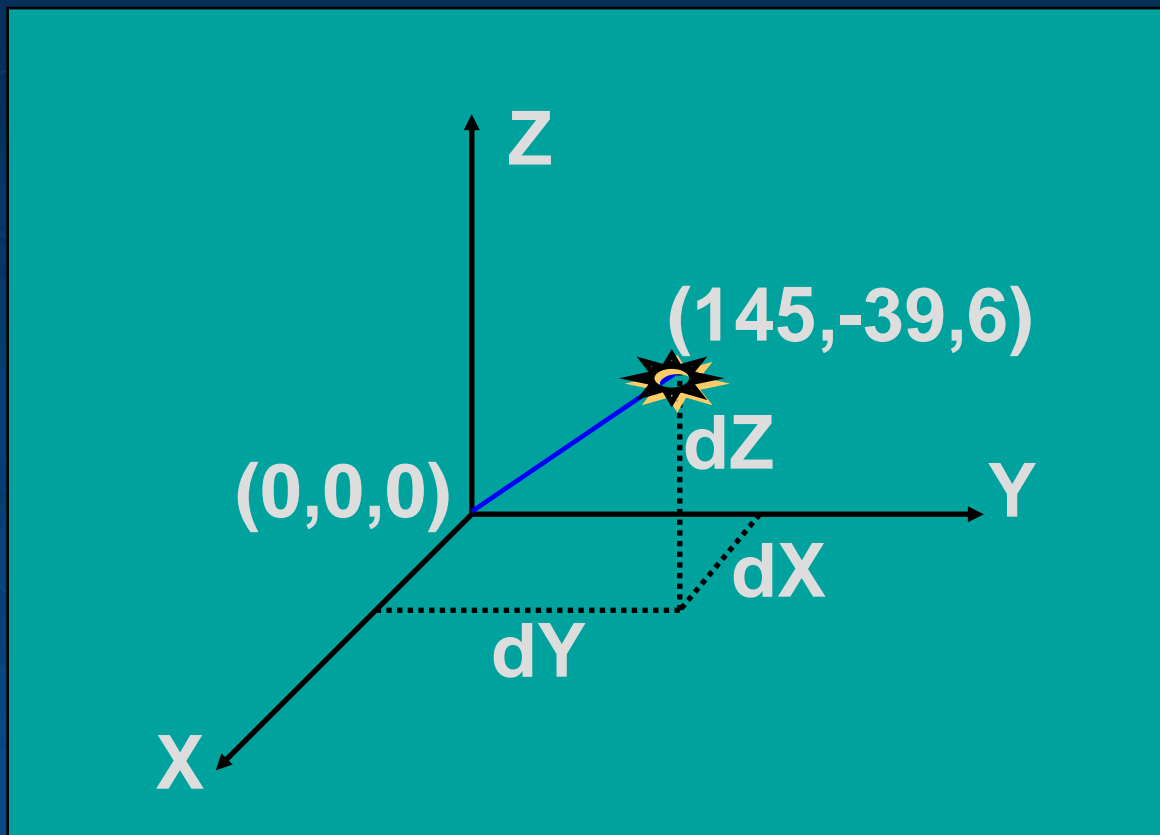
Geographic coordinate systems



- Two types
 - Earth-centered
(WGS84, NAD83)
 - Local
(NAD27, ED50)



Relationship between two gcs



Why so many GCS?



- Many estimates of Earth's size and shape
- Improved accuracy
- Designed for local regions

North American GCS



- **NAD27**
 - Clarke 1866 spheroid
 - Meades Ranch, KS
 - 1880's
- **NAD83**
 - GRS80 spheroid
 - Earth-centered datum
 - GPS-compatible

North American GCS



- **HPGN / HARN**
 - GPS readjustment of NAD83 in the US
 - Also known as ‘NAD91’ or ‘NAD93’
- **NAD27 (1976) & CGQ77**
 - Redefinitions for Ontario and Québec
- **NAD83 (CSRS98)** – GPS readjustment

International GCS



- Defined for countries, regions, or the world
- World: **WGS84, WGS72**
- Regional:
 - **ED50 (European Datum 1950)**
 - **Arc 1950 (Africa)**
- Countries:
 - **GDA 1994 (Australia)**
 - **Tokyo**



Creating a geographic coordsys

```
ISpatialReferenceFactory2 pSRF = new
    SpatialReferenceEnvironmentClass();
ISpatialReference pGCS = new
    GeographicCoordinateSystemClass();

pGCS = (IGeographicCoordinateSystem)
    pSRF.CreateGeographicCoordinateSystem(
        (int)esriSRGeoCSType.esriSRGeoCS_WGS1984);

// pGCS = (ISpatialReference)
    pSRF.CreateESRISpatialReferenceFromPRJFile
        ("C:\\WGS 1984.prj");
```

Creating a custom geographic coordsys



```
IAngularUnit pAngUnits; IPrimeMeridian pPrimeM;  
pAngUnits = (IAngularUnit)pSRF.CreateUnit(  
    (int)esriSRUnitType.esriSRUnit_Degree);  
pPrimeM = pSRF.CreatePrimeMeridian(  
    (int)esriSRPrimeMType.esriSRPrimeM_Greenwich);
```

```
ISpheroid pSpheroid; ISpheroidEdit pSpheroidEdit;  
pSpheroidEdit = new SpheroidClass();  
double a = 6371002.0; double f = 1/298.35;  
pSpheroidEdit.DefineEx("User_Defined_Spheroid", "yes", "ma  
ybe", "no", ref a, ref f);  
pSpheroid = (ISpheroid)pSpheroidEdit;
```

Creating a custom geographic coordsys



```
IDatum pDatum; IDatumEdit pDatumEdit;  
pDatumEdit = new DatumClass();  
pDatumEdit.DefineEx("D_User_Defined", "alias", "abbr", "rem",  
    pSpheroid);  
pDatum = (IDatum)pDatumEdit;
```

```
IGeographicCoordinateSystem2 pGCS;  
IGeographicCoordinateSystemEdit pGCSEdit;  
pGCSEdit = new GeographicCoordinateSystemClass();  
  
pGCSEdit.DefineEx("GCS_User_Defined",  
    "alias", "abbr", "rem", "usage", pDatum, pPrimeM, pAngUnits);  
  
pGCS = (IGeographicCoordinateSystem2)pGCSEdit;
```

Geographic (datum) transformations



- Hundreds available
- Method
- Area of use
- Accuracy

Transformation methods

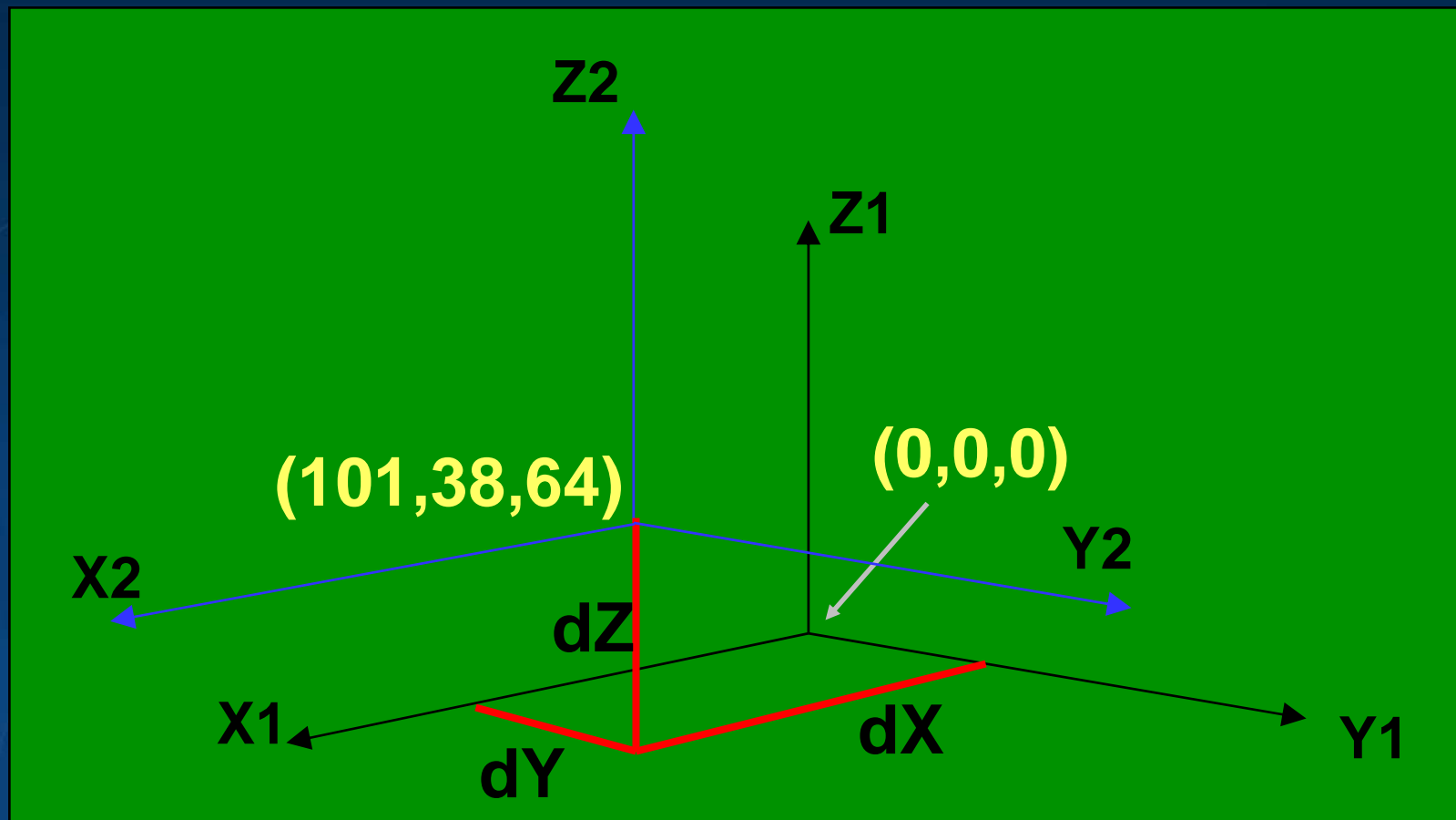


- Grid-based
 - NADCON / HARN (US),
NT v1 / NT v2 (Canada, Australia, NZ)
- Equation-based
 - Molodensky, Bursa-Wolf,
Coordinate Frame, Three Parameter, Seven
Parameter, Molodensky-Badekas

Equation-based methods



- Molodensky (3 parameter method)
 - Convert between any 2 datums (usually WGS84)
 - Origin shift in X, Y, Z
 - Similar to modified Molodensky method
 - Ex. -87, -98, -121 meters
 - Also known as Geocentric Translation

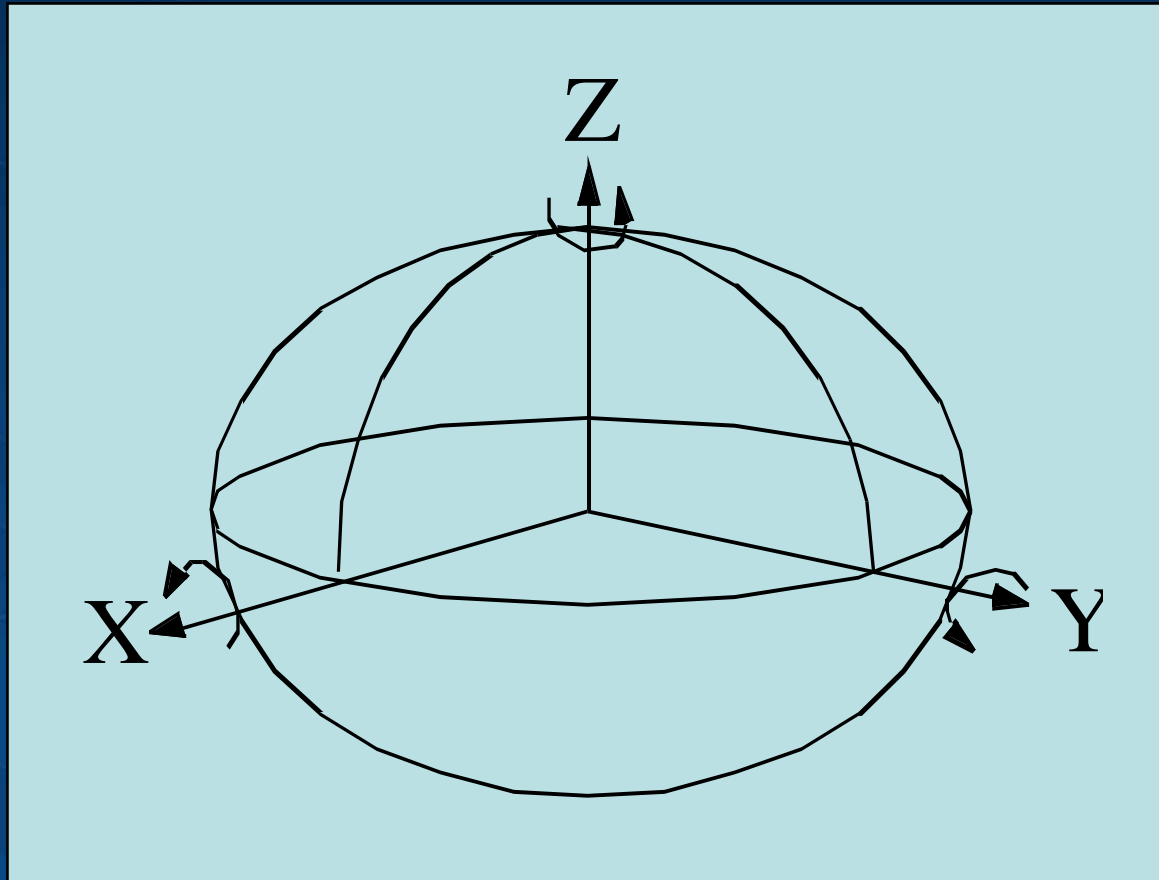


More equation-based methods



- Seven parameters
 - Convert between any 2 datums (usually WGS84)
 - Origin shifts and rotations in X, Y, Z
 - Scale difference
 - Also known as Coordinate Frame, Position Vector, Bursa-Wolf
 - Molodensky-Badekas adds the datum origin point for 3 more parameters

Seven parameter rotations



File-based methods



- NADCON
- HARN
- NTv2
- Generally most accurate—should be used if available

Method accuracies



NADCON	15 cm
HARN/HPGN	5 cm
CNT (NTv1)	10 cm
Seven parameter	1-2 m
Three parameter	4-5 m

GPS



- Uses WGS84 datum
- Other datums are transformed and not as accurate
- Know what transformation method is being used

Using transformations



- Projection Engine format
 - Name: **ED 1950 To WGS 1984 (1)**
Method: **Geocentric Translation**
From gcs: **ED 1950**
To gcs: **WGS 1984**
Parameters: **-87 -98 -121**
- Transformations are separate from a GCS / datum definition

WKT version



```
GEOGTRAN["ED_1950_To_WGS_1984_1",
GEOGCS["GCS_European_1950",DATUM["D_
European_1950",SPHEROID["International_19
24",6378388,297]],PRIMEM["Greenwich",0],U
NIT["Degree",0.017453292519943295]],
GEOGCS["GCS_WGS_1984",DATUM["D_WG
S_1984",SPHEROID["WGS_1984",6378137,2
98.257223563]],PRIMEM["Greenwich",0],UNIT
["Degree",0.017453292519943295]],
METHOD["Geocentric_Translation"],
PARAMETER["X_Axis_Translation",-87.0],
PARAMETER["Y_Axis_Translation",-98.0],
PARAMETER["Z_Axis_Translation",-121.0]]
```

Creating a geographic/datum transformation



```
IGeographicCoordinateSystem2 pGCSto;
```

```
IGeographicCoordinateSystem2 pGCSfrom;
```

```
pGCSfrom = (IGeographicCoordinateSystem2)  
            pPCSin.GeographicCoordinateSystem;  
pGCSto = (IGeographicCoordinateSystem2)  
          pPCSout.GeographicCoordinateSystem;
```

```
ICoordinateFrameTransformation pCFT;
```

```
pCFT = new CoordinateFrameTransformationClass();
```

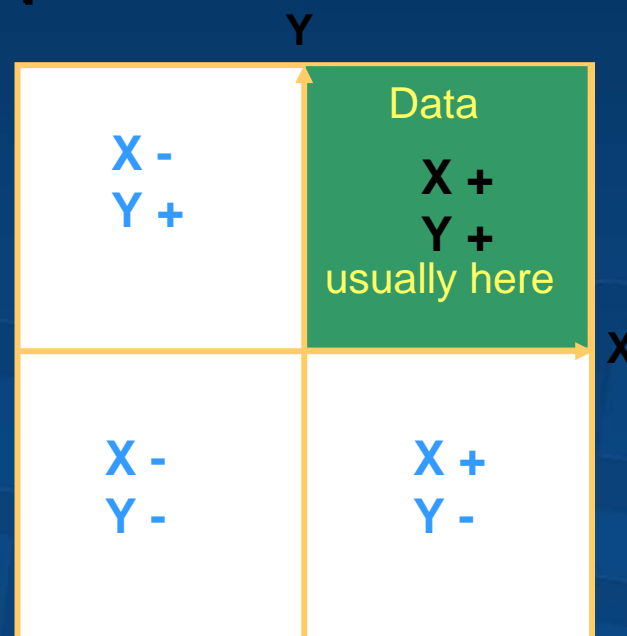


Creating a geographic/datum transformation

```
pCFT.Name = "Custom GeoTran";  
pCFT.PutParameters(1.234, -2.345, 658.3, 4.3829, -2.48591,  
    2.18943, 2.48585);  
pCFT.PutSpatialReferences(pGCSfrom, pGCSto);  
  
IGeoTransformationOperationSet pGTSet;  
  
pGTSet = pSRF.GeoTransformationDefaults;  
pGTSet.Set(esriTransformDirection.esriTransformForward,  
    pCFT);  
pGTSet.Set(esriTransformDirection.esriTransformReverse,  
    pCFT);
```

Projected or Cartesian coordinate system

- also known as projcs or pcs
- Linear units
- Lengths, angles, & areas are constant



WKT version



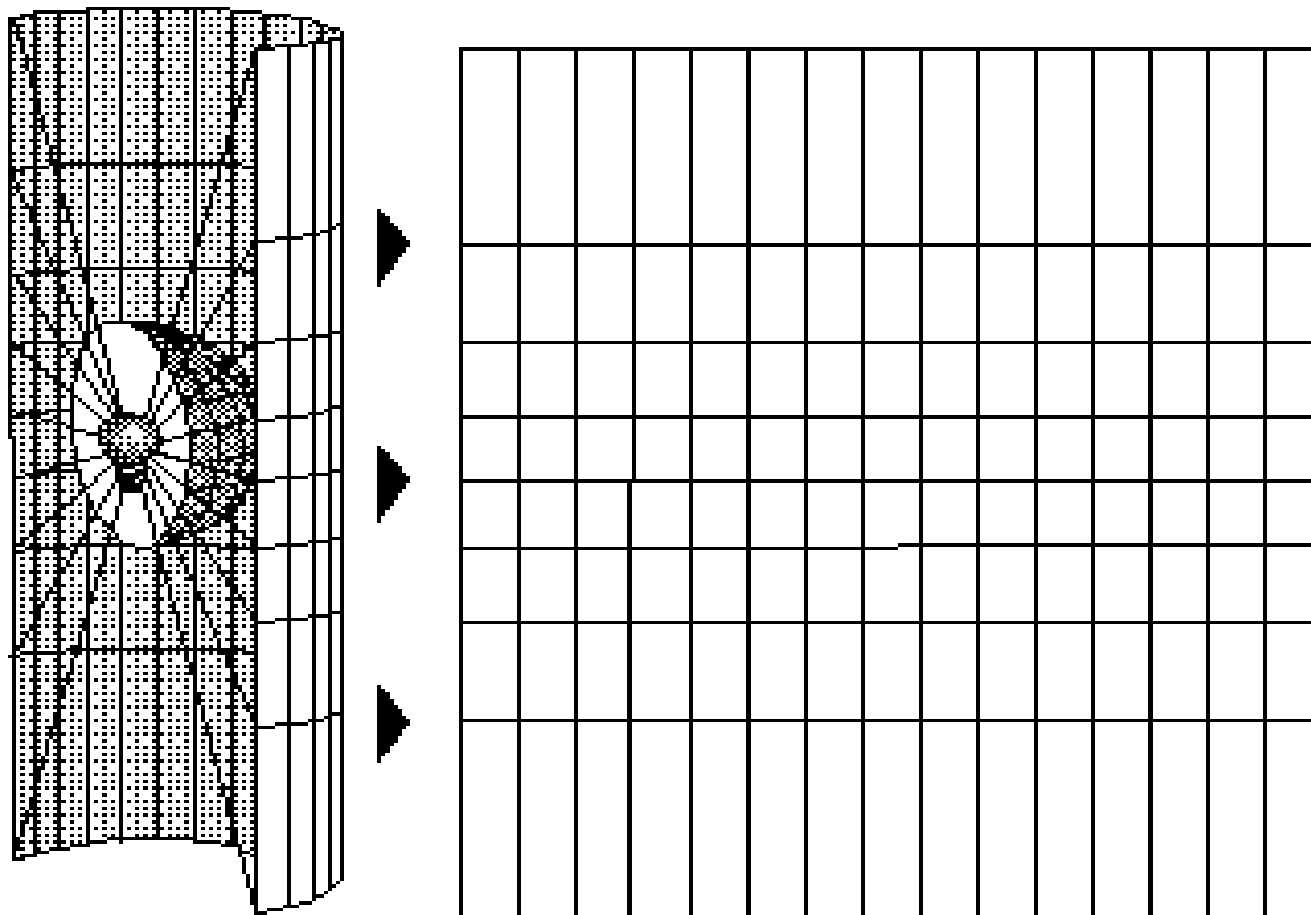
```
PROJCS [ "Test" ,  
    GEOGCS [ "GCS_WGS_1984" , ... ]  
    PROJECTION [ "Mercator" ] ,  
    PARAMETER [ "false_easting" , 500000 ] ,  
    PARAMETER [ "false_northing" , 1000000 ] ,  
    PARAMETER [ "central_meridian" , -83.0 ] ,  
    PARAMETER [ "standard_parallel_1" , 40.0 ] ,  
    UNIT [ "Foot" , 0.3048 ] ]
```

Map Projections



- mathematical conversion of 3-D Earth to a 2-D surface
 - Longitude/Latitude to XY
 - $(\lambda, \varphi) \longleftrightarrow (x, y)$

Visualize a light shining through the Earth onto a surface

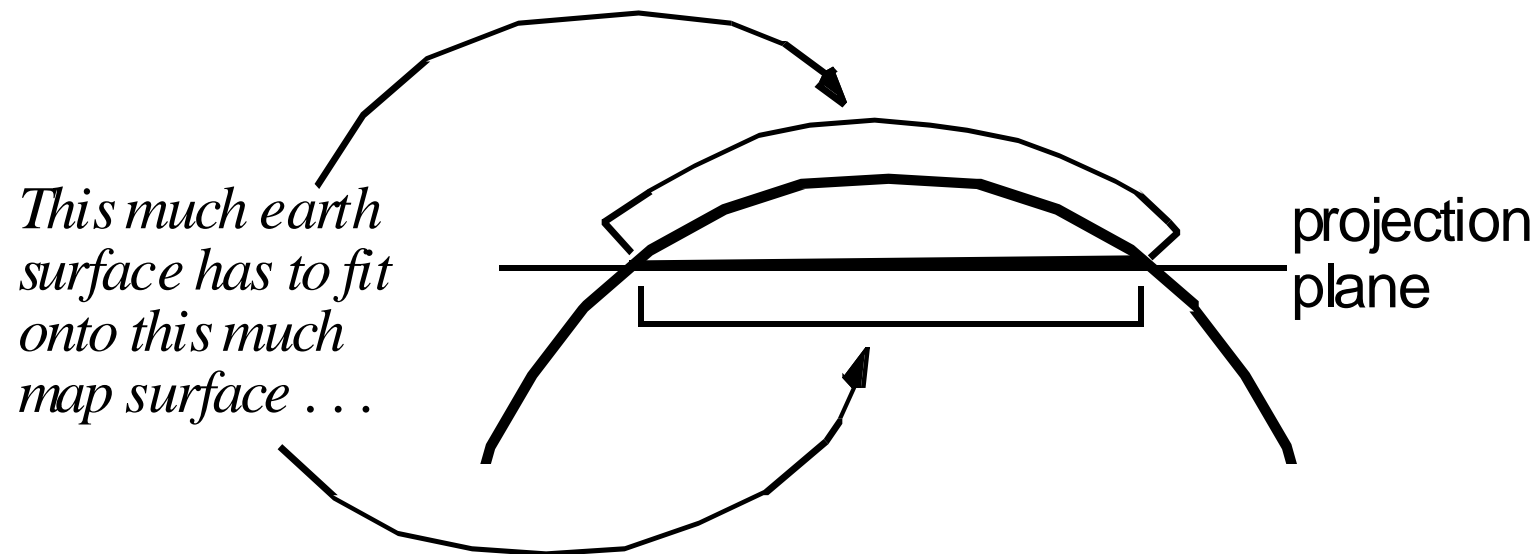


More on projections



- Different projections cause different distortions
- Different projections are useful for different applications

- Fitting sphere to plane causes stretching or shrinking of features



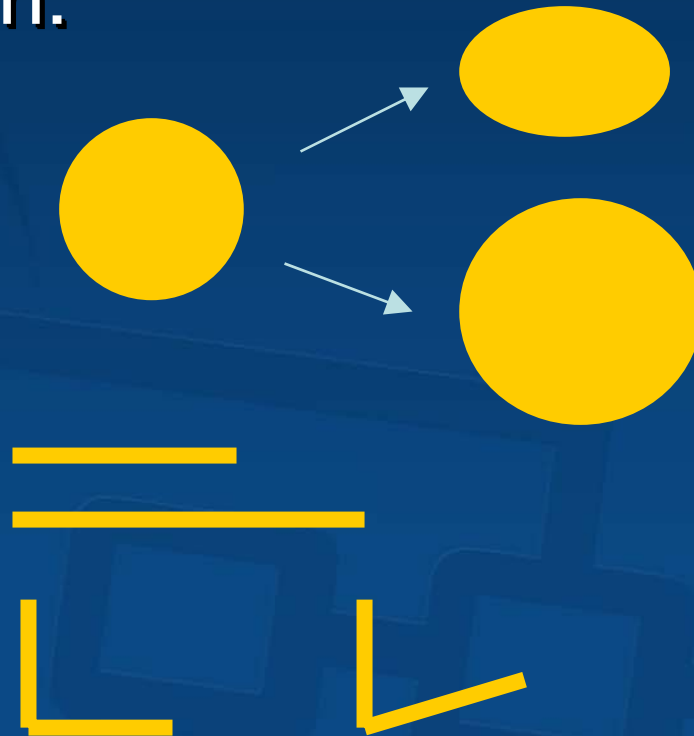
therefore, much of the Earth surface has to be represented smaller than the nominal scale.

More on projections



- Projecting Earth's surface always involves distortion:

- shape
- area
- distance
- direction



Projection properties



- **Conformal**
 - maintains shape
- **Equal-area**
 - maintains area
- **Equidistant**
 - maintains distance
- **Direction**
 - maintains some directions

Projection surfaces



- Cones, Cylinders, Planes
 - can be flattened without distortion
- A point or line of contact is created when surface is combined with a sphere

More on projection surfaces



- Tangent
 - projection surface touches sphere
- Secant
 - surface cuts through sphere
- No distortion at contact points
- Increases away from contact points

Conic Projections

- Best for mid-latitudes with an East-West orientation.
- Tangent or secant along 1 or 2 lines of latitude known as 'standard parallels'.

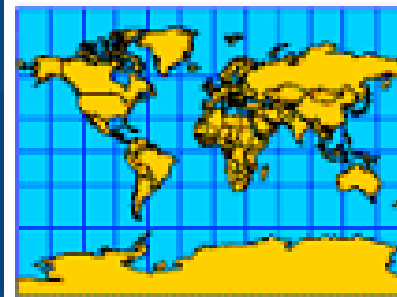


Cylindrical projections

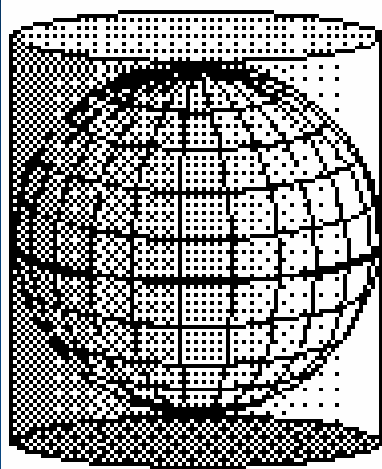


- Best for equatorial or low latitudes
- Rotate cylinder to reduce distortion along a line

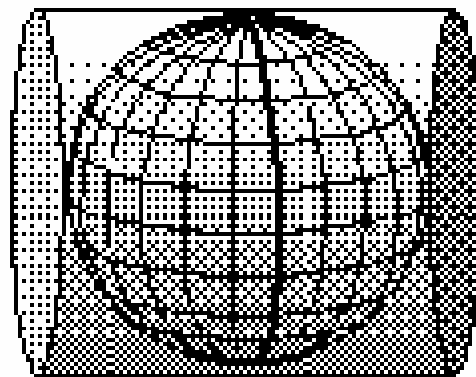
Cylinder



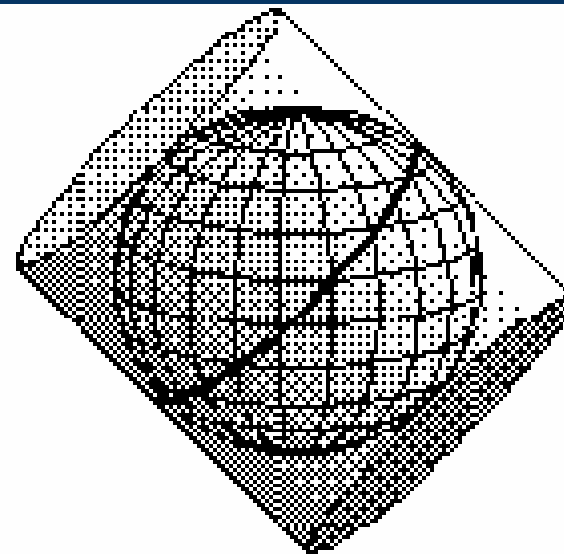
- Normal - equatorial / East-West
- Transverse - North-South regions
- Oblique - other angles



Normal



Transverse



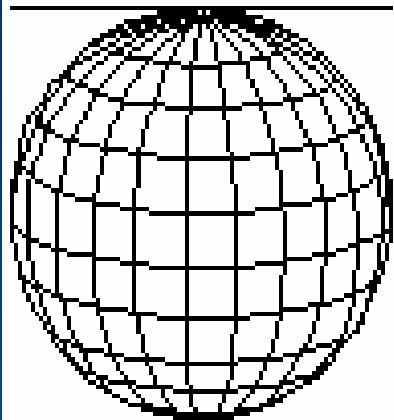
Oblique

Planar projections

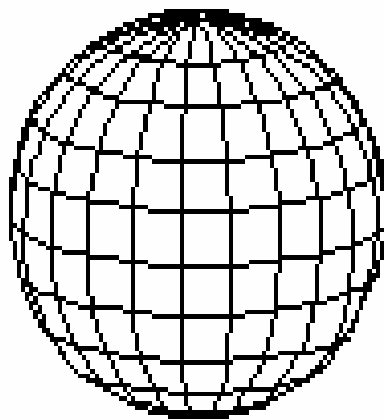


- Best for polar or circular regions
- Direction always true from center
- Shortest distance from center to another point is a straight line

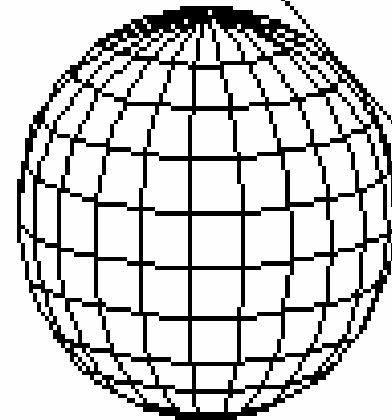
- Also called azimuthal or zenithal
- Can be any aspect



Polar



Equatorial

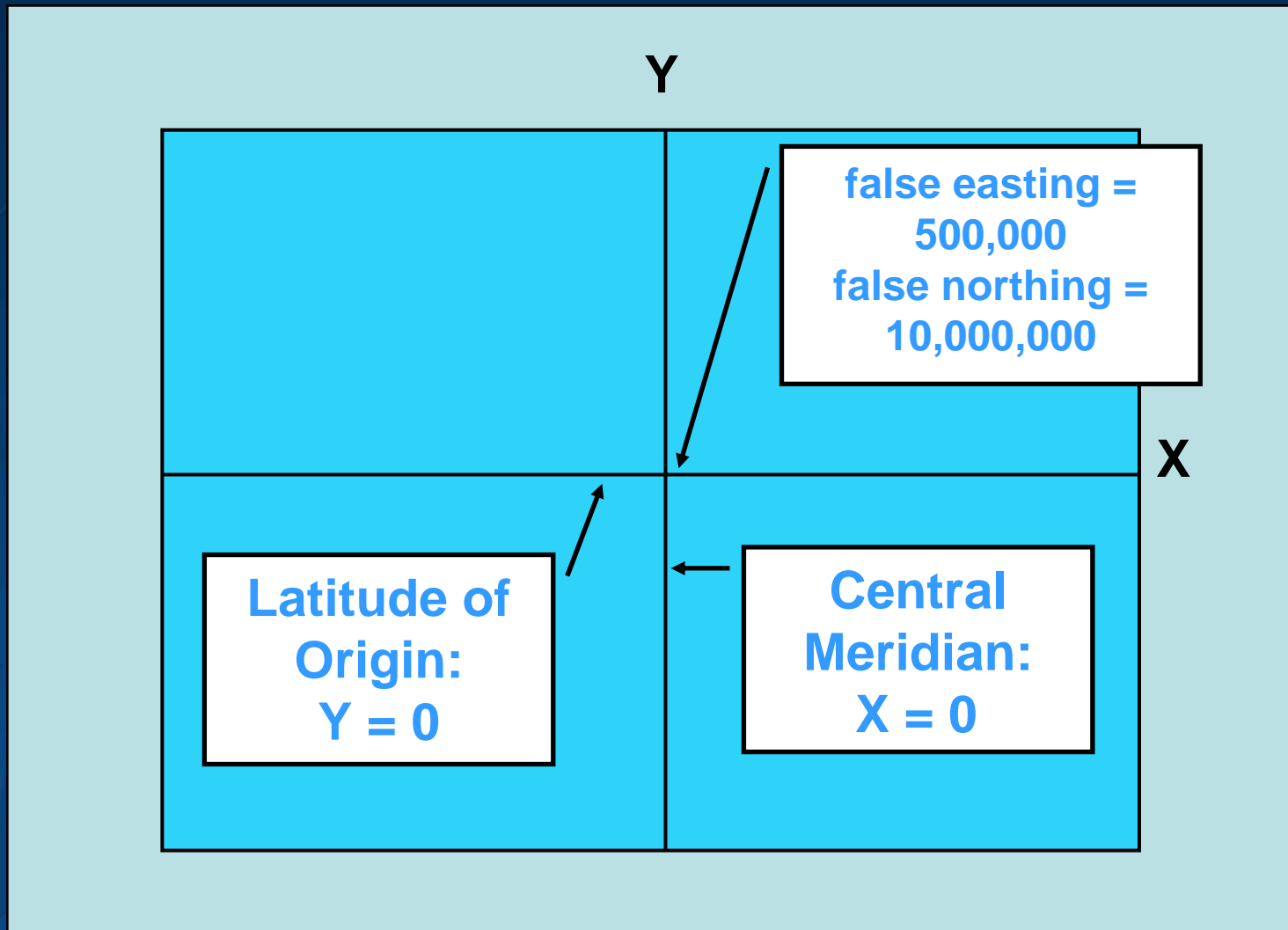


Oblique

Projection parameters



- Central meridian
Longitude of origin
Longitude of center
- Standard parallel / Latitude of center
- Latitude of origin
- False easting / False northing



Choosing a coordinate system



- Often mandated by organization
 - Thematic = equal-area
 - Presentation = conformal (also equal-area)
 - Navigation = Mercator, true direction or equidistant

More on choosing a coordinate system



- Extent
- Location
- Predominant extent
- Projection supports spheroids/datums?

Changing the coordinate system



- Know the coordinate system
 - units
 - geographic c.s. (datum)
 - map projection
 - projection parameters

UTM



- Universal Transverse Mercator
- 60 zones, 6° wide
- Transverse Mercator
- zone 1, central meridian = -177
- scale factor = 0.9996
- false easting = 500,000 m
- In Southern Hemisphere, false northing = 10,000,000 m

- Gauss Krüger
- uses zones, 6° wide
- Transverse Mercator / Gauss Krüger
- zone 1, central meridian = +3
- scale factor = 1.0
- false easting = 500,000 m (or zone number X 1,000,000 + 500,000)
ex. Zone 2 = 2,500,000 m)

SPCS



- State Plane Coordinate System
- States have 1 or more zones
- Uses either NAD27 or NAD83 datums
- Uses Lambert Conic, Transverse Mercator, and Oblique Mercator

Creating a custom projected coordinate system



```
object projection = pSRF3.CreateProjection(  
    (int)esriSRProjectionType.esriSRProjection_Albers);  
object linUnit = (ILinearUnit)pSRF3.CreateUnit(  
    (int)esriSRUnitType.esriSRUnit_Meter);  
object gGCS = (IGeographicCoordinateSystem)  
    pSRF3.CreateGeographicCoordinateSystem  
    (esriSRGeoCSType.esriSRGeoCS_NAD1983);  
  
IParameter[] parms = new IParameter[6];  
  
parms[0] = pSRF3.CreateParameter(  
    (int)esriSRParameterType.esriSRParameter_FalseEasting);  
parms[0].Value = 1000000.0;
```

Creating a custom projected coordinate system



```
parms[1] = pSRF3.CreateParameter(  
    (int)esriSRParameterType.esriSRParameter_FalseNorthing);  
parms[1].Value = 0.0;  
parms[2] = pSRF3.CreateParameter(  
    (int)esriSRParameterType.esriSRParameter_CentralMeridian);  
parms[2].Value = -126.0;  
parms[3] = pSRF3.CreateParameter(  
    (int)esriSRParameterType.esriSRParameter_StandardParallel1);  
parms[3].Value = 50.0;
```

Creating a custom projected coordinate system



```
parms[4] = pSRF3.CreateParameter(  
(int)esriSRParameterType.esriSRParameter_StandardParallel2);  
parms[4].Value = 58.5;  
parms[5] = pSRF3.CreateParameter(  
(int)esriSRParameterType.esriSRParameter_LatitudeOfOrigin);  
parms[5].Value = 45.0;  
object oParms = parms as Object;
```

Creating a custom projected coordinate system



```
IProjectedCoordinateSystem pCS;
```

```
IProjectedCoordinateSystemEdit pCSEdit = new  
    ProjectedCoordinateSystemClass();
```

```
object n = "NewPCSName";
```

```
object al = "alias"; object ab = "abbrev";
```

```
object rem = "remarks"; object u = "usage";
```

```
pCSEdit.Define(ref n, ref al, ref ab, ref rem,  
    ref u, ref gGCS, ref linUnit, ref projection, ref oParms);
```

Creating a custom projected coordinate system



```
pCS = (IProjectedCoordinateSystem)pCSEdit;  
IESRISpatialReferenceGEN esriSR =  
    (IESRISpatialReferenceGEN)pCS;
```

```
// Convert the result to the WKT version
```

```
int cb;
```

```
string wktstr;
```

```
esriSR.ExportToESRISpatialReference(out wktstr, out cb);
```

```
Debug.Print(wktstr);
```

Vertical coordinate systems



- Can define on vector data
- Transformations not supported yet
- Supports geoid-based and ellipsoid-based heights
- Heights/Depths

WKT version



```
VERTCS["NAVD_1988",  
VDATUM["North_American_Vertical_Datum_1988"],  
PARAMETER["Vertical_Shift",0.0],  
PARAMETER["Direction",1.0],  
UNIT["Meter",1.0]]
```

Creating a vertical coordinate system



```
ISpatialReferenceFactory3 pSRF3 = new  
    SpatialReferenceEnvironmentClass();
```

```
IVerticalCoordinateSystem pVCS =  
    (IVerticalCoordinateSystem)  
    pSRF3.CreateVerticalCoordinateSystem(  
        (int)esriSRVerticalCSType.esriSRVertCS_NAVD1988);
```

```
IGeographicCoordinateSystem pGCS =  
    (IGeographicCoordinateSystem)  
    pSRF3.CreateGeographicCoordinateSystem(  
        (int)esriSRGeoCSType.esriSRGeoCS_NAD1983);
```

```
ISpatialReference3 pSR3;
```

```
pSR3 = (ISpatialReference3)pGCS;
```

```
pSR3.VerticalCoordinateSystem = pVCS;
```

Georeferencing



- Required by raster and CAD data
- Internal transformation from raw to 'real' coordinates
- World files, raster header

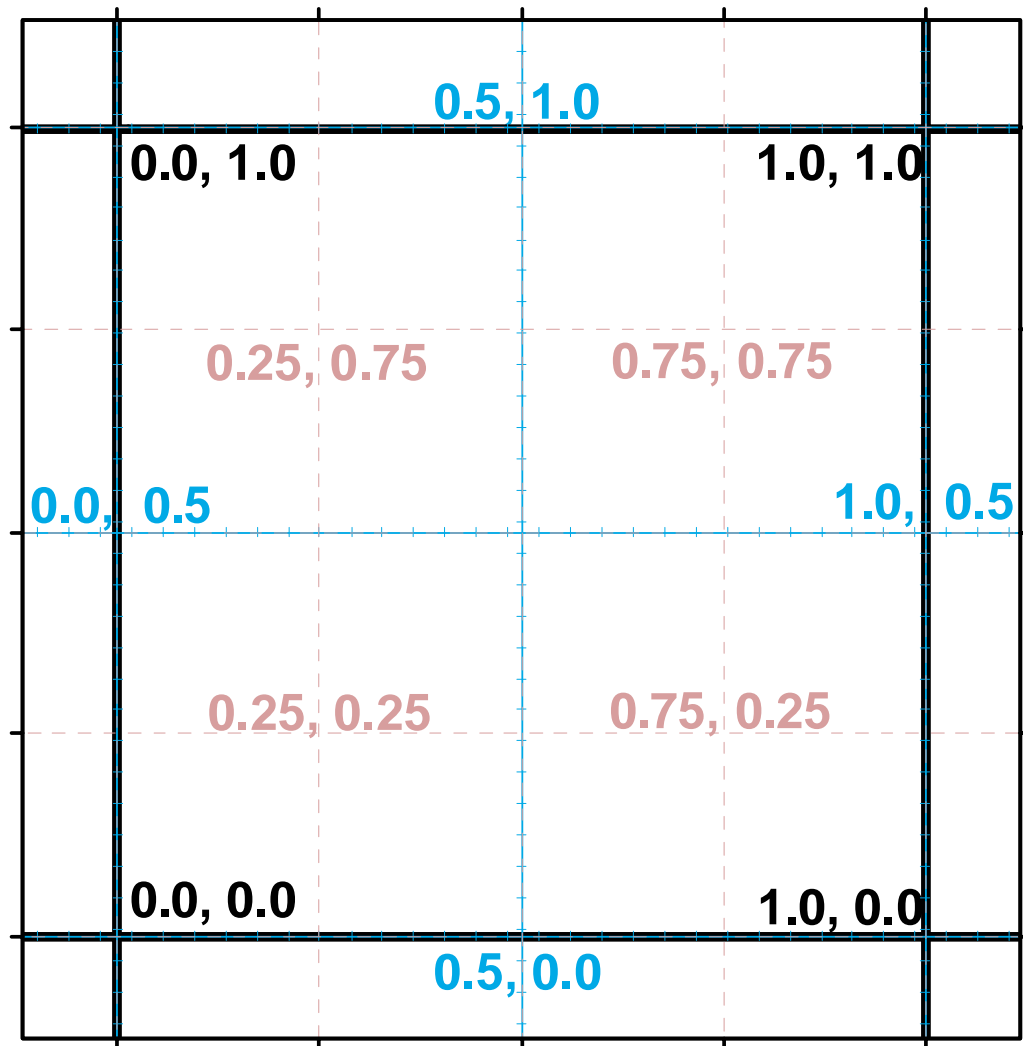
```
60.0000000000 // cell size
0.0000000000 // rotation
0.0000000000 // rotation
-60.0000000000 // negative cell size
440750.0000000000 // minimum X
3751290.0000000000 // minimum Y
```

Spatial Reference



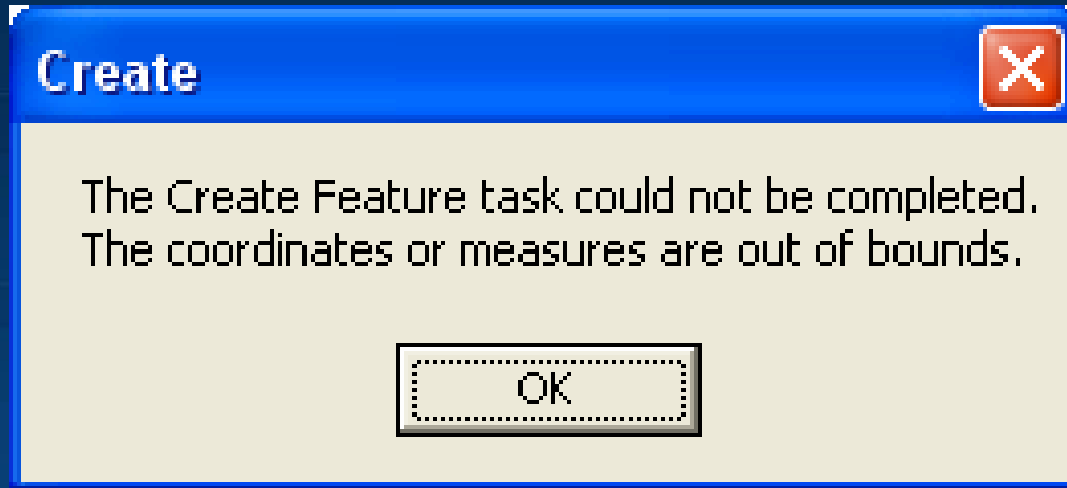
- Coordinate system
- Storage and processing parameters
 - Spatial or $xy/z/m$ domain
 - Resolution = $1/\text{precision}$
 - In the units of the coordinate system
 - Storage precision
 - Tolerance
 - Reflects data accuracy
 - Minimum: $2x$ resolution

Spatial domain



- Features are snapped to the grid corners
- Resolution = 0.25

Pre-9.2 geodatabases



- Pre-9.2 geodatabases use 32-bit integers
- Limited to 0 – 2147483645 integers

Pre-9.2 default values



Min X:	<input type="text" value="-10000"/>	Max X:	<input type="text" value="11474.83645"/>
Min Y:	<input type="text" value="-10000"/>	Max Y:	<input type="text" value="11474.83645"/>
Precision:	<input type="text" value="100000"/>		

- Spatial domain/precision defaults were useless
 - Doesn't fit geographic or projected data
- Conversion tools were better—checked data extents

High precision



- 64-bit storage
- Actually, it's 53-bits
- Range is 0 to 9007199254740990
- ArcSDE supports it already, ArcGIS couldn't use the data
- Equivalent to double precision



Integers from double precision coordinates

```
x = (int) (precision * (X - minX));  
y = (int) (precision * (Y - minY));
```

```
minX = -180.0; minY = -90.0;
```

```
precision = 5000000.0;
```

```
X = -179.0000001; Y = 0.9999999;
```

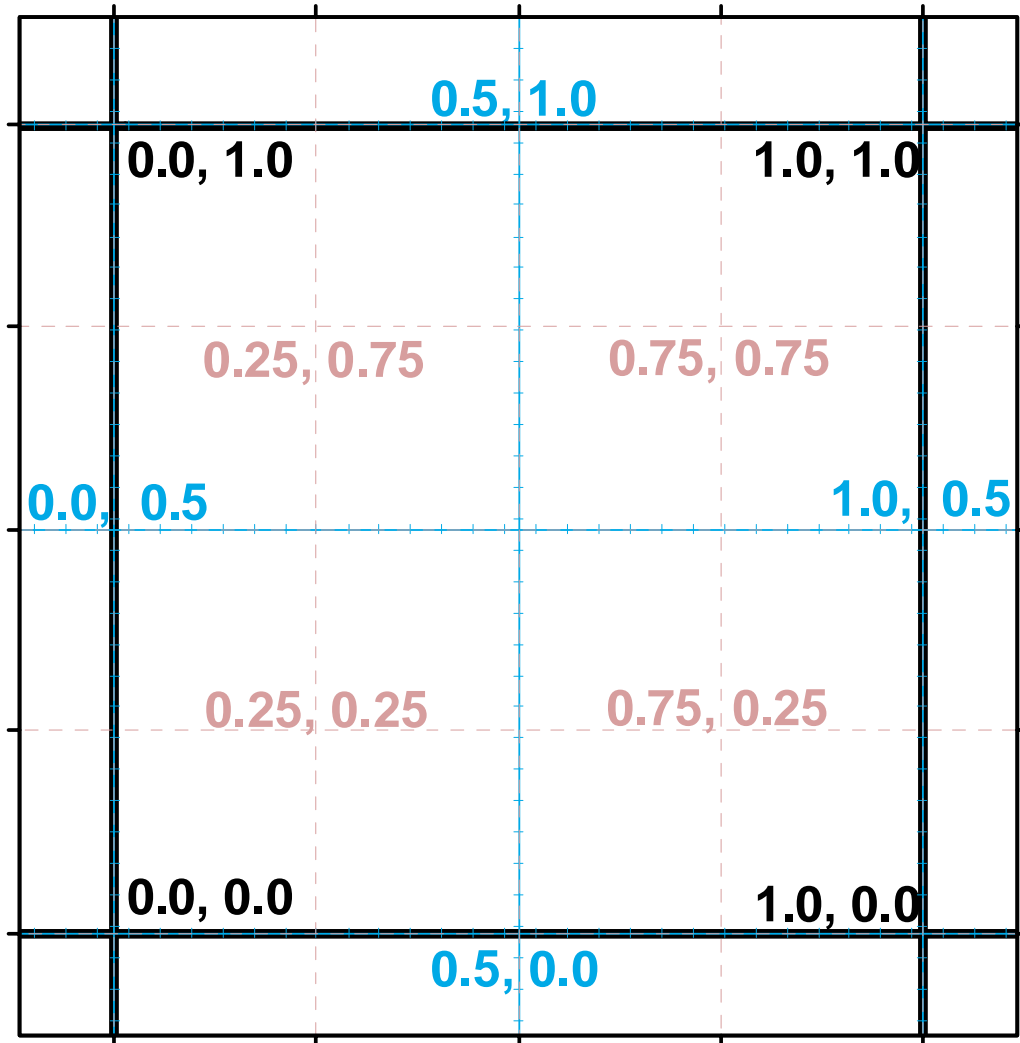
```
x = (int) (5000000.0 * (-179.0000001 - (-180.0)));
```

```
y = (int) (5000000.0 * (0.9999999 - (-90.0)));
```

```
x = 4999995
```

```
y = 445000005
```

Upgrading to high precision



- Black = original grid
- Blue = new grid, 4x
- Rose = 16x



Defining a high precision SR

```
ISpatialReference pSR;
```

```
ISpatialReferenceFactory2 pSRF = new  
    SpatialReferenceEnvironmentClass();
```

```
ISpatialReferenceResolution pSRResolution;
```

```
IControlPrecision2 pControlPrecision2;
```

```
pSR = (ISpatialReference)  
    pSRF.CreateESRISpatialReferenceFromPRJFile  
    ("C:\\WGS 1984.prj");
```

```
pControlPrecision2 = (IControlPrecision2)pSR;
```

```
pControlPrecision2.IsHighPrecision = true;
```

```
pSRResolution = (ISpatialReferenceResolution)pSR;
```

```
pSRResolution.ConstructFromHorizon();
```

```
pSRResolution.SetDefaultXYResolution();
```



Defining a low precision SR at 9.1

```
ISpatialReference pSR;
```

```
pSR = (ISpatialReference)
```

```
pSRF3.CreateESRISpatialReferenceFromPRJFile("C:\\WGS  
1984.prj");
```

```
pSR.SetDomain(-180.0, 180.0, -90.0, 270.0);
```

```
// Precision = 5965232.34722222
```

```
// or Resolution =0.00000016764
```

```
pSR.SetFalseOriginAndUnits(-180.0, -90.0, 1000000.0);
```

```
// maxX = 1967.483645
```

```
// maxY =2057.483645
```

Session Evaluations Reminder



Session Attendees:
Please turn in your session evaluations.

... Thank you