Natural Area Coding System

Xinhang Shen
1608-45 Huntingdale Blvd., Toronto, ON M1W 2N8, Canada
NAC: 8CNJ Q8ZG

Summary

Natural Area Coding System is a new geodetic system to unify the very concepts of areas and locations to generate highly efficient representation called Natural Area Code (NAC) for any location and areas in the world or three dimension region in the universe. Eight or ten character two-dimensional Natural Area Codes can uniquely specify every building, house, gate, door, fire hydrant, parking meter, manhole, street light, tree, and other fixed object or spot in the world. They can be directly used as geographic coordinates, area codes, addresses, postcodes and property identifiers. The grid generating NACs is called Universal Map Grid which can be applied on all kinds of maps. When NACs are used as addresses, they can shorten the representations by 80%, and make them language independent and geographic coverage complete. When a NAC is used to specify an area, it tells both the location and size of the area with very few characters. Thus, NACs will be very useful in all location related services and applications, especially useful for drones and self-driving cars which need exact locations of their destinations.

Introduction

The efficiency in dealing with geodetic datums, geographic coordinates, areas, addresses, postal codes and map grids influences a variety of applications: navigations, postal/courier services, emergency services, GIS applications, etc.

Now all smartphones have GPS capability to measure locations to sub-meter accuracy, but a pair of longitude/latitude coordinates are too long for people to efficiently use. For example, when an emergency happens, a person still uses ambiguous description of their location: "it's on Highway 400 , about one hundred kilometers north of Toronto." Recently, drone deliveries and self-driving taxis are getting popular, which need an efficient and reliable way to specify their destinations, but longitude/latitude coordinates and street addresses can't meet these requirements.

All these needs lead to the birth of Natural Area Coding System.

Specification

Natural Area Coding System employs revolutionary approaches:

- Unify the concepts of geodetic points, areas, and three-dimensional regions.
- Employ 30 world most popular characters instead of ten digits to produce efficient codes;
- Be defined only on the datum of WGS-84 to avoid datum variations;

Natural Area Coding System is a three dimensional grid system with its origin at the earth gravitation center and three directions: longitudinal direction and latitudinal direction on geopotential surfaces and radial direction along the gravitation lines extending to the infinitely distant universe. The system divides the whole range of longitude (-180^o to 180^o), latitude (-90^o to 90^o) and radius (0 to infinite) into 30 divisions respectively and assign a character to each division respectively. Each character is selected from a character set consisting of digits 0 to 9 and 20 English capital consonants representing an integer ranging from 0 to 29, as shown in the following table:

Table of the NAC Character and Integer Correspondences

Character	Integer	Character	Integer	Character	Integer
0	0	B	10	N	20
1	1	C	11	P	21
2	2	D	12	Q	22
3	3	F	13	R	23
4	4	G	14	S	24
5	5	H	15	T	25
6	6	J	16	V	26
7	7	K	17	W	27
8	8	L	18	X	28
9	9	M	19	Z	29

Each resulting division can be further divided into 30 subdivisions, represented by one character respectively too. The division process can continue to the third, fourth, and other levels. The final code is called Natural Area Code (NAC) starts with "NAC: " followed by the three character strings separated by blank spaces, representing longitude, latitude and altitude respectively. The resulting divisions in three dimensions are called NAC blocks. Therefore, a first level NAC block can be represented by a NAC of three characters separated by blank spaces, each of which represents the character string for longitude, latitude and altitude respectively, for example, NAC: 5 6 7. A second level NAC block can be represented by a NAC of six characters to form three character strings: the first two characters form the longitudinal string, the third and fourth characters form the latitudinal string, and the last two characters form the altitudinal string. A blank space is placed between these strings, for example, NAC: JB KH LN represents a NAC block at the second level, in which the characters J, K and L represent coordinates of a first level NAC block which contains the second level NAC block, and the characters B, H and N are the relative coordinates of the second level NAC block in the first level NAC block.

If the third string of a NAC is omitted, the resulting NAC represents an area on the earth surface, called NAC cell. A NAC can represent any point or any area on the earth, any point or any 3D region in the universe.

When longitude, latitude and altitude are given, the following algorithm is used to find the NAC containing the location:

LONG = (longitude + 180)/360
x_1 = Integer part of(LONG*30)
x_2 = Integer part of((LONG*30-x_1)*30)
x_3 = Integer part of(((LONG*30-x_1)*30-x_2)*30)
x_4 = Integer part of((((LONG*30-x_1)*30-x_2)*30-x_3)*30)
...

LAT = (latitude + 90)/180
y_1 = Integer part of(LAT*30)

y_2 = Integer part of((LAT*30-y_1)*30)
y_3 = Integer part of(((LAT*30-y_1)*30-y_2)*30)
y_4 = Integer part of((((LAT*30-y_1)*30-y_2)*30-y_3)*30)
...

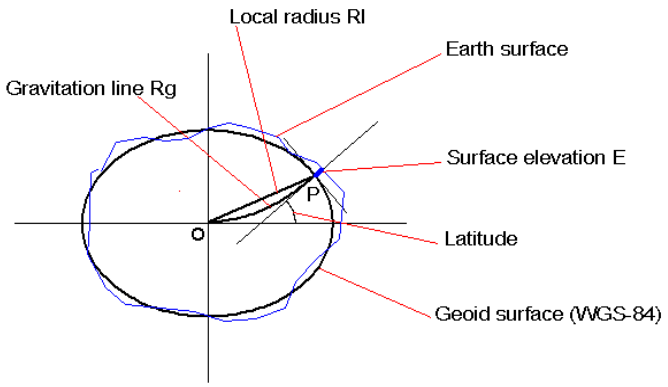
ALT = Arctan(altitude/R)/90
z_1 = Integer part of(ALT*30)
z_2 = Integer part of((ALT*30-z_1)*30)
z_3 = Integer part of(((ALT*30-z_1)*30-z_2)*30)
z_4 = Integer part of((((ALT*30-z_1)*30-z_2)*30-z_3)*30)
...

where longitude is positive in the eastern hemisphere but negative in the western; latitude is positive in the northern hemisphere but negative in the southern; both longitude and latitude are in degrees plus decimals; altitude is measured along the gravitational force line from the center of the geoid of the earth in kilometers; symbol * is the multiplication sign; x_1, x_2, x_3, x_4, ..., y_1, y_2, y_3, y_4, ..., z_1, z_2, z_3, z_4, ... are integers ranging from 0 to 29 here; Arctan( ) is the arctangent function with value in degrees; R is in km the distance from the earth center along the gravitational force line to the geoid surface and can be approximated by the local earth radius at the location:

$$R = Rl = \text{sqrt}\{[a^4\cos^2(\text{latitude}) + b^4\sin^2(\text{latitude})]/[a^2\cos^2(\text{latitude}) + b^2\sin^2(\text{latitude})]\}$$

or

$$R = R1 = \text{sqrt}\{b^2 + (a^2 - b^2)/[1 + (b^2/a^2)*\tan^2(\text{latitude})]\}$$



Surface altitude = Rg + E ~ Rl + E

where a is the semi-major earth axis (ellipsoid equatorial radius) equal to 6378.1370 km; b is the semi- minor earth axis (ellipsoid polar radius) equal to 6356.7523 km; E is the elevation above the geoid surface (approximately the sea level); sqrt( ) is the square root function; tan( ) is a triangular tangent function; symbol / is the division sign.

Once x_1, x_2, x_3, x_4, ..., y_1, y_2, y_3, y_4, ..., z_1, z_2, z_3, z_4, ... are calculated, the corresponding characters can be found from the Table of the NAC Character and Integer Correspondences: X_1, X_2, X_3, X_4, ..., Y_1, Y_2, Y_3, Y_4, ..., Z_1, Z_2, Z_3, Z_4, ... Then, the Natural Area Code of the region is written as:

NAC: $X_1X_2X_3X_4$... Y1Y2Y3Y4... $Z_1Z_2Z_3Z_4$...

with a blank space between any two character strings. The first character string of a NAC represents longitude, the second string represents latitude, and the third represents altitude.

If a NAC has only two character strings, then the NAC represents an area on the earth surface and the two character strings represent the longitude and latitude respectively. For example, NAC: 8KD8 PGGK represents a 25 by 37 meter area in White House; NAC: 8KD8 PGGK H000 represents a three-dimensional region 25 meters wide, 37 meters long and 25 meters high measured from the geoid surface under White House.

The number of characters to be used in a character string of a NAC representing the geodetic point is determined by the required resolution or the resolution of the original coordinates of the longitude, latitude and altitude. A NAC using more characters represents a smaller area or region. The smallest area or region containing the geodetic point is the one of the size equal to the error range of the coordinates. Therefore, when a NAC is used to represent a geodetic point, it contains the information of both the location and its error range.

When a NAC is given, the longitude, latitude and altitude of the southwestern bottom corner of the NAC block can be calculated by the following procedure:

First, convert all characters X_1, X_2, X_3, X_4, ... Y_1, Y_2, Y_3, Y_4, ... Z_1, Z_2, Z_3, Z_4, ... into integers x_1, x_2, x_3, x_4, ... y_1, y_2, y_3, y_4, ... z_1, z_2, z_3, z_4, ... according to the Table of the NAC Character and Integer Correspondences.

Then use the following formulae to calculate coordinates:

$$\text{longitude} = (x_1/30 + x_2/30^2 + x_3/30^3 + x_4/30^4 + ...)*360 - 180$$
$$\text{latitude} = (y_1/30 + y_2/30^2 + y_3/30^3 + y_4/30^4 + ...)*180 - 90$$
$$\text{altitude} = R*\tan((z_1/30 + z_2/30^2 + z_3/30^3 + z_4/30^4 + ...)*90)$$

where R is the distance from the earth center to the geoid surface along gravitational line. The northeastern upper corner of the region can be calculated by repeating the same procedure with the same integers adding 1 to the integer corresponding to the last character of each string of the NAC.

When a three-dimensional region defined by the NAC of its southwest lower corner and the NAC of its northeast upper corner, the region can be represented by a single complex NAC which consists of each string of the southwest lower NAC if the corresponding strings of the two NACs are the same, otherwise the string of the southwest lower NAC plus a hyphen plus the corresponding string of the northeast upper NAC with the shared prefix characters dropped. For example, a region with the southwest lower NAC: 8CHJ Q8KL HJP9 and northeast upper NAC: 8CHL Q8ZS HPKG can be represented by a single complex NAC:

NAC: 8CHJ Q8KL HJP9 + NAC: 8CNL Q8ZS HPKG = NAC: 8CHJ-L Q8KL-ZS HJP9-PKG.

When a complex NAC with 0-Z at the end of any of its character strings, these three characters can be omitted provided that there are some characters left in the character string, for example:

NAC: JJ0-Z KKL HG = NAC: JJ KKL HG

NAC: JJ0-Z KKL0-Z HG0-Z = NAC: JJ KKL HG

but NAC: 0-Z K L can't be simplified to NAC: K L

If the first string of the southwest lower corner NAC represents a longitude larger than the first string of the northeast upper NAC, the resulted complex NAC represents a region across the -180^o and 180^o meridian of the earth while other strings of the southwest lower NAC must represent values smaller than the corresponding

strings of the northeast upper NAC such as: NAC: W-2 H K represents all cells from NAC: W H K to NAC: 2 H K across the 180^{o} meridian, i.e.

NAC: W-2 H K = NAC: W H K + NAC: Z H K + NAC: 0 H K + NAC: 1 H K + NAC: 2 H K

Therefore, a valid NAC only have two or three character strings separated by single spaces. Each character string of a NAC must consist of characters from the character set "0123456789BCDFGHJKLMNPQRSTVWXZ-", and has at most one hyphen which should never be the leading or ending character of the string.

When a complex NAC is given, the southwest lower NAC and the northeast upper NAC can be derived according to the following procedure: The southwest lower corner NAC consists of the corresponding string of the complex NAC if it does not contain a hyphen or the string in front of the hyphen if it contains a hyphen; the northeast upper corner NAC consists of the corresponding string of the complex NAC if it does not contain a hyphen or the prefix of the string in front of the hyphen plus the string behind the hyphen to form a string with the same length of the string in front of the hyphen if it contains a hyphen. For example, NAC: 8CHJ-L Q8HG-KL has the southwest corner NAC: 8CHJ Q8HG and the northeast corner NAC: 8CHL Q8KL

Applications

Natural Area Codes can be used in all geographic location and area related applications such as car navigations, postal/courier services and GIS and online maps.

When a NAC is used to specify a location, it is highly efficient, language independent and geographic coverage complete. For example, NAC: 8CNJ Q8ZG can directly pinpoint our building on all maps and for all location based services, compared with the street address: "45 Huntingdale Blvd, Toronto, ON M1W 2N8, Canada".

When a NAC is used to retrieve a map on a GIS or an online map, it will be even more efficient than specifying a location. For example, you need to input only "8C Q8" to get the map of the city of Toronto instead of "Toronto, Ontario, Canada".

When a NAC is used as a postcode, it can help postal services to sort all mail from the world level to final mail boxes automatically without any missing locations anywhere in the world.

When a NAC is used as a property/object identifier, it will not only make all such identifiers worldwide unique, but also represent their exact locations.

Three-dimensional NACs can be efficiently used to specify locations of mines anywhere in the earth and satellites anywhere in the space.

JavaScript

Generated by Gemini on March 18, 2025 (not verified yet)

```
const nacCharacters = "0123456789BCDFGHJKLMNPQRSTVWXZ";

function isValidNAC(nac) {
  if (typeof nac !== 'string') {
    return false;
  }

  const strings = nac.split(' ');

  if (strings.length < 2 || strings.length > 3) {
    return false;
  }

  for (const str of strings) {
    if (str.length === 0) {
      return false;
    }
    if (str.includes('-')) {
      if (str.indexOf('-') === 0 || str.indexOf('-') === str.length - 1 ||
str.split('-').length > 2)
      {
        return false;
      }
      const parts = str.split('-');
      for (const part of parts){
        if (!/^[0-9BCDFGHJKLMNPQRSTVWXZ]+$/.test(part)){
          return false;
        }
      }
    } else if (!/^[0-9BCDFGHJKLMNPQRSTVWXZ]+$/.test(str)) {
      return false;
    }
  }

  return true;
}

function charToInt(char) {
  return nacCharacters.indexOf(char);
}

function intToChar(int) {
  return nacCharacters[int];
}

function calculateCoordinate(chars, isLatitude = false) {
  let coordinate = 0;
  let multiplier = isLatitude ? 180 : 360;
  let offset = isLatitude ? 90 : 180;

  for (let i = 0; i < chars.length; i++) {
    coordinate += charToInt(chars[i]) / Math.pow(30, i + 1);
  }

  return coordinate * multiplier - offset;
}
```

```
function calculateAltitude(altitudeChars, latitude) {
    let altitudePart = 0;

    for (let i = 0; i < altitudeChars.length; i++) {
        altitudePart += charToInt(altitudeChars[i]) / Math.pow(30, i + 1);
    }

// Accurate altitude calculation
    const a = 6378.1370; // semi-major earth axis in km [cite: 30, 31]
    const b = 6356.7523; // semi-minor earth axis in km [cite: 30, 31]

    const R = Math.sqrt(
      (a**4 * Math.cos(latitude * Math.PI / 180)**2 + b**4 * Math.sin(latitude * Math.PI
/ 180)**2) /
        (a**2 * Math.cos(latitude * Math.PI / 180)**2 + b**2 * Math.sin(latitude * Math.PI
/ 180)**2)
    );

    return R * Math.tan(altitudePart * 90 * Math.PI / 180);
}

function GetNAC(level, longitude, latitude, altitude) {
// Input validations
    if (level < 1) {
        throw new Error("Level must be at least 1.");
    }
    if (longitude < -180 || longitude > 180) {
        throw new Error("Longitude must be between -180 and 180 degrees.");
    }
    if (latitude < -90 || latitude > 90) {
        throw new Error("Latitude must be between -90 and 90 degrees.");
    }

    let longChars = "";
    let latChars = "";
    let altChars = "";

    const LONG = (longitude + 180) / 360;
    const LAT = (latitude + 90) / 180;
    let ALT;

    if (altitude !== undefined) {
        // Accurate ALT calculation
        const a = 6378.1370; // semi-major earth axis in km [cite: 30, 31]
        const b = 6356.7523; // semi-minor earth axis in km [cite: 30, 31]

        const R = Math.sqrt(
          (a**4 * Math.cos(latitude * Math.PI / 180)**2 + b**4 * Math.sin(latitude * Math.PI
/ 180)**2) /
            (a**2 * Math.cos(latitude * Math.PI / 180)**2 + b**2 * Math.sin(latitude *
Math.PI / 180)**2)
        );

        ALT = Math.atan(altitude / R) / 90;
    }

    for (let i = 0; i < level; i++) {
        let x = Math.floor(LONG * Math.pow(30, i + 1)) % 30;
        longChars += intToChar(x);

        let y = Math.floor(LAT * Math.pow(30, i + 1)) % 30;
```

```
        latChars += intToChar(y);

        if (altitude !== undefined) {
            let z = Math.floor(ALT * Math.pow(30, i + 1)) % 30;
            altChars += intToChar(z);
        }
    }

    let nac = longChars + " " + latChars;
    if (altitude !== undefined) {
        nac += " " + altChars;
    }

    return nac;
}

function GetSouthwest(nac) {
    if (!isValidNAC(nac)) {
        throw new Error("Invalid NAC format.");
    }

    const strings = nac.split(' ');
    const longitudeChars = strings[0];
    const latitudeChars = strings[1];
    const altitudeChars = strings.length > 2 ? strings[2] : null;

    const longitude = calculateCoordinate(longitudeChars);
    const latitude = calculateCoordinate(latitudeChars, true);
    let altitude = null;

    if (altitudeChars) {
        altitude = calculateAltitude(altitudeChars, latitude);
    }

    return altitude !== null ? { longitude, latitude, altitude } : { longitude, latitude };
}

function GetCenter(nac) {
    if (!isValidNAC(nac)) {
        throw new Error("Invalid NAC format.");
    }

    const strings = nac.split(' ');
    const longitudeChars = strings[0];
    const latitudeChars = strings[1];
    const altitudeChars = strings.length > 2 ? strings[2] : null;

    let southwest = GetSouthwest(nac);

    let longLength = 360 / Math.pow(30, longitudeChars.length);
    let latLength = 180 / Math.pow(30, latitudeChars.length);

    let centerLong = southwest.longitude + longLength / 2;
    let centerLat = southwest.latitude + latLength / 2;

    if (altitudeChars) {

        let altSouthwest = GetSouthwest(nac).altitude;
        let altDiff;

// Calculate the altitude of the "northeast upper" corner.
```

```
        let upperAltChars = altitudeChars.substring(0, altitudeChars.length - 1) +
intToChar(charToInt(altitudeChars.slice(-1)) + 1);

        if (upperAltChars.slice(-1) === nacCharacters[0]){
          upperAltChars = altitudeChars.substring(0, altitudeChars.length - 2) +
intToChar(charToInt(altitudeChars.slice(-2, -1)) + 1) + nacCharacters[1];
        }

        if (upperAltChars.includes(NaN)){
          upperAltChars = altitudeChars;
        }

        let altNortheast = calculateAltitude(upperAltChars, southwest.latitude);

        altDiff = altNortheast - altSouthwest;

        let centerAlt = altSouthwest + altDiff / 2;
        return {longitude: centerLong, latitude: centerLat, altitude: centerAlt};
    }

    return { longitude: centerLong, latitude: centerLat };
}

function GetBoundingBox(nac) {
    if (!isValidNAC(nac)) {
        throw new Error("Invalid NAC format.");
    }

    let southwestNAC, northeastNAC;

    if (nac.includes('-')) {
// Complex NAC handling [cite: 46, 47, 48, 49, 50, 51, 52, 53, 54]
        const strings = nac.split(' ');
        let southwestStrings =;
        let northeastStrings =;

        for (const str of strings) {
            if (str.includes('-')) {
                const parts = str.split('-');
                southwestStrings.push(parts[0]);
// Ensure northeast part has same length as southwest part
                northeastStrings.push(parts[0].slice(0, parts[0].length -
parts[1].length) + parts[1]);
            } else {
                southwestStrings.push(str);
                northeastStrings.push(str);
            }
        }
        southwestNAC = southwestStrings.join(' ');
        northeastNAC = northeastStrings.join(' ');

    } else {
        southwestNAC = nac;
        northeastNAC = nac;
    }

    const southwest = GetSouthwest(southwestNAC);
    const strings = northeastNAC.split(' ');
    const longitudeChars = strings[0];
    const latitudeChars = strings[1];
    const altitudeChars = strings.length > 2 ? strings[2] : null;
```

```
    let longLength = 360 / Math.pow(30, longitudeChars.length);
    let latLength = 180 / Math.pow(30, latitudeChars.length);

    const northeastLongitude = southwest.longitude + longLength;
    const northeastLatitude = southwest.latitude + latLength;
    let northeastAltitude = null;

    if (altitudeChars) {
// Calculate the altitude of the "northeast upper" corner.
        let upperAltChars = altitudeChars.substring(0, altitudeChars.length - 1) +
intToChar(charToInt(altitudeChars.slice(-1)) + 1);

// if last character is Z, then we need to increment the second to last character
        if (upperAltChars.slice(-1) === nacCharacters.slice(-1)){
          upperAltChars = altitudeChars.substring(0, altitudeChars.length - 2) +
intToChar(charToInt(altitudeChars.slice(-2, -1)) + 1) + nacCharacters[0];
        }

        // if upperAltChars has NaN, then it means we are at the maximum altitude, so we
just use the southwest altitude
        if (upperAltChars.includes(NaN)){
          northeastAltitude = southwest.altitude;
        }
        else {
          northeastAltitude = calculateAltitude(upperAltChars, southwest.latitude);
        }

        return {
            southwest: southwest,
            northeast: {
                longitude: northeastLongitude,
                latitude: northeastLatitude,
                altitude: northeastAltitude,
            },
        };
    }

    return {
        southwest: southwest,
        northeast: {
            longitude: northeastLongitude,
            latitude: northeastLatitude,
        },
    };
}
```