Gray code

From Wikipedia, the free encyclopedia

The **reflected binary code**, also known as **Gray code** after Frank Gray, is a binary numeral system where two successive values differ in only one digit.

The reflected binary code was originally designed to prevent spurious output from electromechanical switches. Today, Gray codes are widely used to facilitate error correction in digital communications such as digital terrestrial television and some cable TV systems.

2-bit Gray code

```
00
01
11
10
```

3-bit Gray code

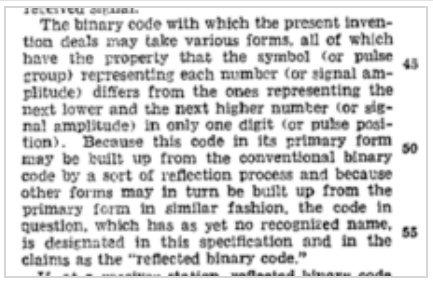
```
000
001
011
010
110
111
101
100
```

4-bit Gray code

```
0000
0001
0011
0010
0110
0111
0101
0100
1100
1101
1111
1110
1010
1011
1001
1000
```

Contents

Name



The binary code with which the present invention deals may take various forms, all of which have the property that the symbol (or pulse group) representing each number (or signal amplitude) differs from the ones representing the next lower and the next higher number (or signal amplitude) in only one digit (or pulse position). Because this code in its primary form may be built up from the conventional binary code by a sort of reflection process and because other forms may in turn be built up from the primary form in similar fashion, the code in question, which has as yet no recognized name, is designated in this specification and in the claims as the "reflected binary code."

Gray's patent introduces the term "reflected binary code"

Bell Labs researcher Frank Gray introduced the term *reflected binary code* in his 1947 patent application, remarking that the code had "as yet no recognized name."[1] He derived the name from the fact that it "may be built up from the conventional binary code by a sort of reflection process."

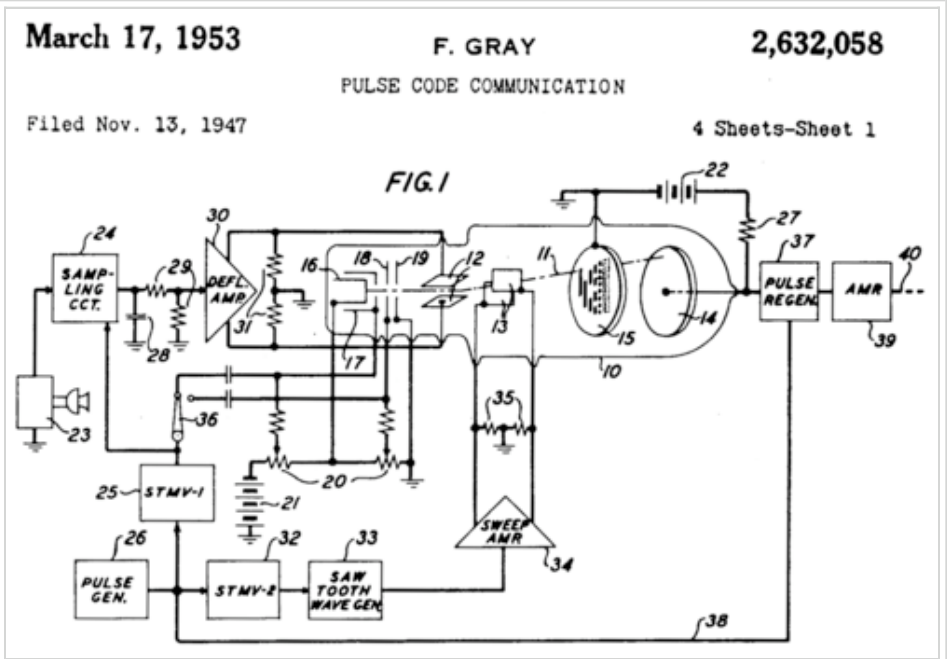
The code was later named after Gray by others who used it. Two different 1953 patent applications give "Gray code" as an alternative name for the "reflected binary code";[2][3] one of those also lists "minimum error code" and "cyclic permutation code" among the names.[3] A 1954 patent application refers to "the Bell Telephone Gray code".[4]

History and practical application

Reflected binary codes were applied to mathematical puzzles before they became known to engineers. The French engineer Émile Baudot used Gray codes in telegraphy in 1878. He received the French Legion of Honor medal for his work. The Gray code is sometimes attributed, incorrectly,[5] to Elisha Gray (in Principles of Pulse Code Modulation, K. W. Cattermole,[6] for example).

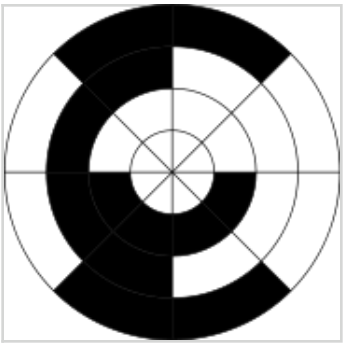
Frank Gray, who became famous for inventing the signaling method that came to be used for compatible color television, invented a method to convert analog signals to reflected binary code groups using vacuum tube-based apparatus. The method and apparatus were patented in 1953 and the name of Gray stuck to the codes. The "PCM tube" apparatus that Gray patented was made by Raymond W. Sears of Bell Labs, working with Gray and William M. Goodall, who credited Gray for the idea of the reflected binary code.[7]

The use of his eponymous codes that Gray was most interested in was to minimize the effect of error in the conversion of analog signals to digital; his codes are still used today for this purpose, and others.



Part of front page of Gray's patent, showing PCM tube (10) with reflected binary code in plate (15)

Gray codes are used in position encoders (linear encoders and rotary encoders), in preference to straightforward binary encoding. This avoids the possibility that, when several bits change in the binary representation of an angle, a misread could result from some of the bits changing before others. Rotary encoders benefit from the cyclic nature of Gray codes, because the first and last values of the sequence differ by only one bit.


Rotary encoder for angle-measuring devices marked in 3-bit binary-reflected Gray code (BRGC)

The binary-reflected Gray code can also be used to serve as a solution guide for the Tower of Hanoi problem. It also forms a Hamiltonian cycle on a hypercube, where each bit is seen as one dimension.

Due to the Hamming distance properties of Gray codes, they are sometimes used in Genetic Algorithms. They are very useful in this field, since mutations in the code allow for mostly incremental changes, but

occasionally a single bit-change can cause a big leap and lead to new properties.

Gray codes are also used in labelling the axes of Karnaugh maps.

When Gray codes are used in computers to address program memory, the computer uses less power because fewer address lines change as the program counter advances.

In modern digital communications, Gray codes play an important role in error correction. For example, in a digital modulation scheme such as QAM where data is typically transmitted in symbols of 4 bits or more, the signal's constellation diagram is arranged so that the bit patterns conveyed by adjacent constellation points differ by only one bit. By combining this with forward error correction capable of correcting single-bit errors, it is possible for a receiver to correct any transmission errors that cause a constellation point to deviate into the area of an adjacent point. This makes the transmission system less susceptible to noise.

Digital logic designers use gray codes extensively for passing multi-bit count information between synchronous logic that operates at different clock frequencies. The logic is considered operating in different "clock domains". It is fundamental to the design of large chips that operate with many different clocking frequencies.

A typical use is building a fifo (first-in, first-out data buffer) that has read and write ports that exist in different clock domains. The updated read and write pointers need to be passed between clock domains when they change, to be able to track fifo empty and full status in each domain. Each bit of the pointers is sampled non-deterministically for this clock domain transfer. So for each bit, either the old value or the new value is propagated.

Therefore, if more than one bit in the multi-bit pointer is changing at the sampling point, a "wrong" binary value (neither new nor old) can be propagated.

By guaranteeing only one bit can be changing, gray codes guarantee that the only possible sampled values are the new or old multi-bit value. Typically gray codes of power-of-two length are used.

Gray-code counters and arithmetic

Sometimes digital buses in electronic systems are used to convey quantities that can only increase or decrease by one at a time, for example the output of an event counter which is being passed between clock domains or to a digital-to-analog converter. The advantage of Gray code in these applications is that differences in the propagation delays of the many wires that represent the bits of the code cannot cause the received value to go through states that are out of the Gray code sequence. This is similar to the advantage of Gray codes in the construction of mechanical encoders, however the source of the Gray code is an electronic counter in this case. The counter itself must count in Gray code, or if the counter runs in binary then the output value from the counter must be reclocked after it has been converted to Gray code, because when a value is converted from binary to Gray code, it is possible that differences in the arrival times of the binary data bits into the binary-to-Gray conversion circuit will mean that the code could go briefly through states that are wildly out of sequence. Adding a clocked register after the circuit that converts the count value to Gray code may introduce a clock cycle of latency, so counting directly in Gray code may be advantageous. A Gray code counter was patented in 1962 US3020481 (http://patft.uspto.gov/netacgi/nph-Parser?patentnumber=3020481) , and there have been many others since. In recent times a Gray code counter can be implemented as a state machine in Verilog. In order to produce the next count value, it is necessary to have some combinational logic that will increment the current count value that is stored in Gray code. Probably the most obvious way to increment a Gray code number is to

convert it into ordinary binary code, add one to it with a standard binary adder, and then convert the result back to Gray code. This approach was discussed in a paper in 1996 [8] Some issues in gray code addressing (http://ieeexplore.ieee.org/xpls/abs_all.jsp?tp=&arnumber=497616&isnumber=10625) and then subsequently patented by someone else in 1998 US5754614 (http://patft.uspto.gov/netacgi/nph-Parser?patentnumber=5754614) . Other, potentially much faster methods of counting in Gray code are discussed in the report The Gray Code by R. W. Doran (http://www.cs.auckland.ac.nz/CDMTCS//researchreports/304bob.pdf) , including taking the output from the first latches of the master-slave flip flops in a binary ripple counter.

Motivation

Many devices indicate position by closing and opening switches. If that device uses natural binary codes, these two positions would be right next to each other:

```
...
011
100
...
```

The problem with natural binary codes is that, with real (mechanical) switches, it is very unlikely that switches will change states exactly in synchrony. In the transition between the two states shown above, all three switches change state. In the brief period while all are changing, the switches will read some spurious position. Even without keybounce, the transition might look like 011 — 001 — 101 — 100. When the switches appear to be in position 001, the observer cannot tell if that is the "real" position 001, or a transitional state between two other positions. If the output feeds into a sequential system (possibly via combinatorial logic) then the sequential system may store a false value.

The reflected binary code solves this problem by changing only one switch at a time, so there is never any ambiguity of position,

Dec	Gray	Binary
0	000	000
1	001	001
2	011	010
3	010	011
4	110	100
5	111	101
6	101	110
7	100	111

Notice that state 7 can roll over to state 0 with only one switch change. This is called the "cyclic" property of a Gray code. A good way to remember gray coding is by being aware that the least significant bit follows a repetitive pattern of 2. That is 11, 00, 11 etc. and the second digit follows a pattern of fours.

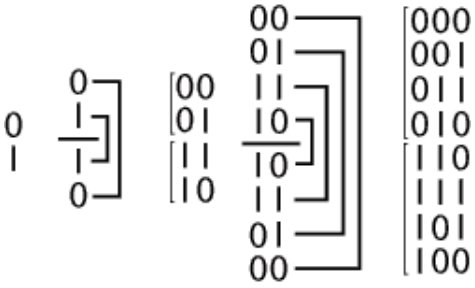
More formally, a **Gray code** is a code assigning to each of a contiguous set of integers, or to each member of a circular list, a word of symbols such that each two adjacent code words differ by one symbol. These codes are also known as *single-distance codes*, reflecting the Hamming distance of 1 between adjacent codes. There can be more than one Gray code for a given word length, but the term was first applied to a particular binary code for the non-negative integers, the *binary-reflected Gray code*, or **BRGC**, the three-bit version of which is shown above.

Constructing an n-bit gray code

The binary-reflected Gray code for *n* bits can be generated recursively by reflecting the bits (i.e. listing them in reverse order and concatenating the reverse list onto the original list), prefixing the original bits with a binary 0 and then prefixing the reflected bits with a binary 1. The base case, for *n=1* bit, is the most basic Gray code, *G = {0, 1}*. (The base case can also be thought of as a single zero-bit Gray code (*n=0*, *G = { " " }*) which is made into the one-bit code by the recursive process, as demonstrated in the Haskell example below).

The BRGC may also be constructed iteratively.

Here are the first few steps of the above-mentioned reflect-and-prefix method:



These characteristics suggest a simple and fast method of translating a binary value into the corresponding BRGC. Each bit is inverted if the next higher bit of the input value is set to one. This can be performed in parallel by a bit-shift and exclusive-or operation if they are available. A similar method can be used to perform the reverse translation, but the computation of each bit depends on the computed value of the next higher bit so it cannot be performed in parallel.

Programming algorithms

Here is an algorithm in pseudocode to convert natural binary codes to Gray code (encode):

```
Let B[n:0] be the input array of bits in the usual binary representation, [0] being LSB
Let G[n:0] be the output array of bits in Gray code
  G[n] = B[n]
  for i = n-1 downto 0
    G[i] = B[i+1] XOR B[i]
```

This algorithm can be rewritten in terms of words instead of arrays of bits:

```
G = B XOR (SHR(B))
```

For instance in C or java languages :

```
g = b ^ (b >> 1);
```

For VHDL (encoding), where G, B are std_logic_vector(3 downto 0):

```
G <= ("0" & B(3 downto 1)) xor B;
```

Here is an algorithm to convert Gray code to natural binary codes (decode):

```
Let G[n:0] be the input array of bits in Gray code
Let B[n:0] be the output array of bits in the usual binary representation
  B[n] = G[n]
  for i = n-1 downto 0
    B[i] = B[i+1] XOR G[i]
```

Here is a much faster algorithm in C/java language :

```
long inverseGray(long n) {
        long ish, ans, idiv;
        ish = 1;
        ans = n;
        while(true) {
                idiv = ans >> ish;
                ans ^= idiv;
                if (idiv <= 1 || ish == 32)
                        return ans;
                 ish <<= 1; // double number of shifts next time
        }
  }
```

Special types of Gray codes

In practice, a "Gray code" almost always refers to a binary-reflected Gray code. However, mathematicians have discovered other kinds of Gray codes. Like BRGCs, each consists of a lists of words, where each word differs from the next in only one digit (each word has a Hamming distance of 1 from the next word).

n-ary Gray code

There are many specialized types of Gray codes other than the binary-reflected Gray code. One such type of Gray code is the **n-ary Gray code**, also known as a **non-Boolean Gray code**. As the name implies, this type of Gray code uses non-Boolean values in its encodings.

For example, a 3-ary (ternary) Gray code would use the values {0, 1, 2}. The *(n,k)-Gray code* is the *n*-ary Gray code with *k* digits.[9] The sequence of elements in the *(3,2)*-Gray code is: *{00, 01, 02, 12, 11, 10, 20, 21, 22}*. The *(n,k)*-Gray code may be constructed recursively, as the BRGC, or may be constructed iteratively. An algorithm to iteratively generate the *(N,k)*-Gray code based on the work of Dah-Jyu Guan [6] is presented (in C/Java):

3-digit ternary Gray codes

```
000
001
002
012
011
010
020
021
022
122
121
120
110
111
112
102
101
100
200
201
202
212
211
210
220
221
222
```

```
int n[k+1]; // stores the maximum for each digit
int g[k+1]; // stores the Gray code
int u[k+1]; // stores +1 or -1 for each element
int i, j;

// initialize values
for(i = 0; i <= k; i++) {
        g[i] = 0;
        u[i] = 1;
        n[i] = N;
}

// generate codes
while(g[k] == 0) {
       // at this point (g[0],...,g[k-1]) hold a subsequent element of the (N,k)-Gray code
        i = 0;
        j = g[0] + u[0];
        while((j >= n[i]) || (j < 0)) {
                u[i] = -u[i];
                i++;
                j = g[i] + u[i];
        }
        g[i] = j;
}
```

It is important to note that the *(n,k)*-Gray codes produced by the above algorithm lack the cyclic property for odd n; it can be observed that in going from the last element in the sequence, *222*, and wrapping around to the first element in the sequence, *000*, three digits change, unlike in a binary Gray code, in which only one digit would change. An *(n,k)*-Gray code with even *n*, however, retains the cyclic property of the binary Gray code.

Gray codes are not uniquely defined, because a permutation of the columns of such a code is a Gray code too. The above procedure produces a code in which each digit switches faster than all digits to its right.

Beckett–Gray code

Another interesting type of Gray code is the **Beckett–Gray code**. The Beckett–Gray code is named after Samuel Beckett, an Irish playwright especially interested in symmetry. One of his plays, "Quad", was divided into sixteen time periods. At the end of each time period, Beckett wished to have one of the four actors either entering or exiting the stage; he wished the play to begin and end with an empty stage; and he wished each subset of actors to appear on stage exactly once.[10] Clearly, this meant the actors on stage could be represented by a 4-bit binary Gray code. Beckett placed an additional restriction on the scripting, however: he wished the actors to enter and exit such that the actor who had been on stage the longest would always be the one to exit. The actors could then be represented by a first in, first out queue data structure, so that the first actor to exit when a dequeue is called for is always the first actor which was enqueued into the structure.[10] Beckett was unable to find a Beckett–Gray code for his play, and indeed, an exhaustive listing of all possible sequences reveals that no such code exists for *n = 4*. Computer scientists interested in the mathematics behind Beckett–Gray codes have found these codes very difficult to work with. It is today known that codes exist for *n = {2, 5, 6, 7, 8}* and they do not exist for *n = {3, 4}*. An example of an 8-bit Beckett–Gray code can be found in [5]. According to [11], the search space for *n = 6* can be explored in 15 hours, and more than 9,500 solutions for the case *n = 7* have been found.

Snake-in-the-box codes

Snake-in-the-box codes, or *snakes*, are the sequences of nodes of induced paths in an *n*-dimensional hypercube graph, and coil-in-the-box codes, or *coils*, are the sequences of nodes of induced cycles in a hypercube. Viewed as Gray codes, these sequences have the property of being able to detect any single-bit

coding error. Codes of this type were first described by W. H. Kautz in the late 1950s;[12] since then, there has been much research on finding the code with the largest possible number of codewords for a given hypercube dimension.

Single-track Gray code

Yet another kind of Gray code is the **single-track Gray code**.

To get high angular accuracy with a BRGC, one needs lots of tracks. If one wants at least 1 degree accuracy (at least 360 distinct positions per revolution) with standard BRGC, it requires at least 9 tracks. (That actually gives 512 distinct positions).

If the manufacturer moves a contact to a different angular position (but at the same distance from the center shaft), then the corresponding "ring pattern" needs to be rotated the same angle to give the same output. If the most significant bit (the inner ring in Figure 1) is rotated enough, it exactly matches the next ring out. Since both rings are then identical, the inner ring can be cut out, and the sensor for that ring moved to the remaining, identical ring (but offset at that angle from the other sensor on that ring).

Those 2 sensors on a single ring make a quadrature encoder.

That reduces the number of tracks for a "1 degree resolution" angular encoder to 8 tracks.

Reducing the number of tracks still further can't be done with BRGC.

For many years, Torsten Sillke (http://www.mathematik.uni-bielefeld.de/~sillke/PROBLEMS/gray) and other mathematicians believed that it was impossible to encode position on a single track such that consecutive positions differed at only a single sensor, except for the 2-sensor, 1-track quadrature encoder.

So for applications where 8 tracks was too bulky, people used single-track incremental encoders (quadrature encoders) or 2-track "quadrature encoder + reference notch" encoders.

However, in 1996 Hiltgen, Paterson and Brandestini published a paper showing it was possible, with several examples.

In particular, a single-track gray code has been constructed that has exactly 360 angular positions, using only 9 sensors, the same as a BRGC with the same resolution (it would be impossible to discriminate that many positions with any fewer sensors).

The single-track Gray code was originally defined by Hiltgen, Paterson and Brandestini in "Single-track Gray codes" (1996). The STGC is a cyclical list of P unique binary encodings of length n such that two consecutive words differ in exactly one position, and when the list is examined as a $P \times n$ matrix, each column is a cyclic shift of the first column.[13] An STGC for $P = 30$ and $n = 5$ is reproduced here:

```
10000
10100
11100
11110
11010
11000
01000
01010
01110
01111
01101
01100
00100
00101
00111
10111
10110
00110
00010
10010
10011
11011
01011
00011
00001
01001
11001
11101
10101
10001
```

Note that each column is a cyclic shift of the first column, and from any row to the next row only one bit changes.[14] The single-track nature (like a code chain) is useful in the fabrication of these wheels (compared to BRGC), as only one track is needed, thus reducing their cost and size. The Gray code nature is useful (compared to chain codes), as only one sensor will change at any one time, so the uncertainty during a transition between two discrete states will only be plus or minus one unit of angular measurement the device is capable of resolving.[15]

See also

- Linear feedback shift register

Footnotes

1. **^** F. Gray. *Pulse code communication*, March 17, 1953 (filed Nov. 1947). U.S. Patent 2,632,058 (http://patft.uspto.gov/netacgi/nph-Parser?patentnumber=2632058)
2. **^** J. Breckman. *Encoding Circuit*, Jan 31, 1956 (filed Dec. 1953). U.S. Patent 2,733,432 (http://patft.uspto.gov/netacgi/nph-Parser?patentnumber=2733432)
3. ^ ***a*** ***b*** E. A. Ragland et al. *Direction-Sensitive Binary Code Position Control System*, Feb. 11, 1958 (filed Oct. 1953). U.S. Patent 2,823,345 (http://patft.uspto.gov/netacgi/nph-Parser?patentnumber=2823345)
4. **^** S. Reiner et al. *Automatic Rectification System*, Jun 24, 1958 (filed Jan. 1954). U.S. Patent 2,839,974 (http://patft.uspto.gov/netacgi/nph-Parser?patentnumber=2839974)
5. ^ ***a*** ***b*** Knuth, Donald E. "Generating all *n*-tuples." *The Art of Computer Programming, Volume 4A: Enumeration and Backtracking*, pre-fascicle 2a, October 15, 2004. [1] (http://www-cs-faculty.stanford.edu/~knuth/fasc2a.ps.gz)
6. **^** K. W. Cattermole, *Principles of Pulse Code Modulation*, American Elsevier Publishing Company, Inc., 1969, New York NY, ISBN 0-444-19747-8.
7. **^** W. M. Goodall, "Television by Pulse Code Modulation," Bell Sys. Tech. J., Vol. 30 pp. 33–49, 1951.
8. **^** Mehta, H.; Owens, R.M. & Irwin, M.J. (1996),Some issues in gray code addressing, in the Proceedings of the 6th Great Lakes Symposium on VLSI (GLSVLSI 96), IEEE Computer Society,pp. 178
9. **^** Guan, Dah-Jyu (1998). "Generalized Gray Codes with Applications". *Proc. Natl. Sci. Counc. Repub. Of China (A)* **22**: 841–848.
10. ^ ***a*** ***b*** Goddyn, Luis (1999). "MATH 343 Applied Discrete Math Supplementary Materials (http://www.math.sfu.ca/~goddyn/Courses/343/supMaterials.pdf) ". Dept. of Math, Simon Fraser U.
11. **^** J. Sawada and D. Wong, "A Fast Algorithm to generate Beckett-Gray codes" Electronic notes in Discrete

Mathematics (EuroComb 2007) Vol . 29 pp. 571–577, 2007.
12. ^ Kautz, W. H. (1958). "Unit-distance error-checking codes". *IRE Trans. Elect. Comput.* **7**: 177–180.
13. ^ Etzion, Tuvi; Moshe Schwartz (1999). "The Structure of Single-Track Gray Codes". *IEEE Transactions on Information Theory* **45**: 2383–2396. doi:10.1109/18.796379.
14. ^ "Venn Diagram Survey — Symmetric Diagrams (http://www.combinatorics.org/Surveys/ds5/VennSymmEJC.html) ". *The Electronic Journal of Combinatorics* (2001).
15. ^ Alciatore and Histand. "Introduction to Mechatronics and Measurement Systems (http://mechatronics.mech.northwestern.edu/mechatronics/design_ref/sensors/encoders.html) ".

References

- Black, Paul E. *Gray code*. 25 February 2004. NIST. [2] (http://www.nist.gov/dads/HTML/graycode.html) .
- Savage, Carla. "A Survey of Combinatorial Gray Codes." *Society of Industrial and Applied Mathematics Review* 39 (1997): 605–629. [3] (http://epubs.siam.org/sam-bin/getfile/SIREV/articles/29527.pdf) .
- Wilf, Herbert S. *Combinatorial algorithms: an update*, SIAM, 1989, ISBN 0-89871-231-9. Chapters 1-3.

External links

- Absolute Encoder Using Gray Code (http://engknowledge.com/shaft_absolute_encoder_gray_code.aspx) - Absolute encoding for rotating shaft, with a comprehensive discussion of gray code.

- "Gray Code" demonstration (http://demonstrations.wolfram.com/BinaryGrayCode/) by Michael Schreiber, The Wolfram Demonstrations Project (with Mathematica implementation). 2007.
- NIST Dictionary of Algorithms and Data Structures: Gray code (http://www.nist.gov/dads/HTML/graycode.html)
- *Numerical Recipes in C*, section 20.2 (http://www.nrbook.com/a/bookcpdf/c20-2.pdf) describing Gray codes in detail (ISBN 0-521-43108-5)
- Hitch Hiker's Guide to Evolutionary Computation, Q21: What are Gray codes, and why are they used? (http://www.aip.de/~ast/EvolCompFAQ/Q21.htm) , including C code to convert between binary and BRGC
- Subsets or Combinations (http://www.theory.cs.uvic.ca/~cos/gen/comb.html) Can generate BRGC strings
- "The Structure of Single-Track Gray Codes" (http://www.cs.technion.ac.il/users/wwwb/cgi-bin/tr-info.cgi?1998/CS/CS0937) by Moshe Schwartz, Tuvi Etzion
- Single-Track Circuit Codes (http://www.hpl.hp.com/techreports/2000/HPL-2000-81.html) by Hiltgen, Alain P.; Paterson, Kenneth G.
- Dragos A. Harabor uses Gray codes in a 3D digitizer (http://www.ugcs.caltech.edu/~dragos/3DP/coord.html) .
- single-track gray codes, binary chain codes (Lancaster 1994 (http://tinaja.com/text/chain01.html)), and linear feedback shift registers are all useful in finding one's absolute position on a single-track rotary encoder (or other position sensor).
- Computing Binary Combinatorial Gray Codes Via Exhaustive Search With SAT Solvers (http://ieeexplore.ieee.org/xpls/abs_all.jsp?isnumber=4475352&arnumber=4475394&count=44&index by Zinovik, I.; Kroening, D.; Chebiryak, Y.
- A Gray code implementation in Java to convert decimals (http://etc.manuel-breitfeld.de/gray-code-java-implementierung.html)

- Use of snake-in-the-box codes for reliable identification of tracks in servo fields of a disk drive (http://www.freepatentsonline.com/20020126407.html)
- United States Patent 6496312: Use of snake-in-the-box codes for reliable identification of tracks in servo fields of a disk drive (http://www.patentstorm.us/patents/6496312.html)

Retrieved from "http://en.wikipedia.org/wiki/Gray_code"

Categories: Data transmission | Numeration