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 bit (binary 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.

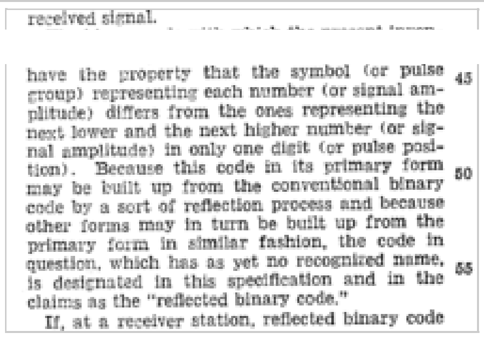
Gray code by bit width

2-bit	4-bit
00	0000
01	0001
11	0011
10	0010
	0110
	0111
3-bit	0101
000	0100
001	1100
011	1101
010	1111
110	1110
111	1010
101	1011
100	1001
	1000

Contents

Name

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".


received signal.

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."

If, at a receiver station, reflected binary code

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

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]

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 combinational 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
```

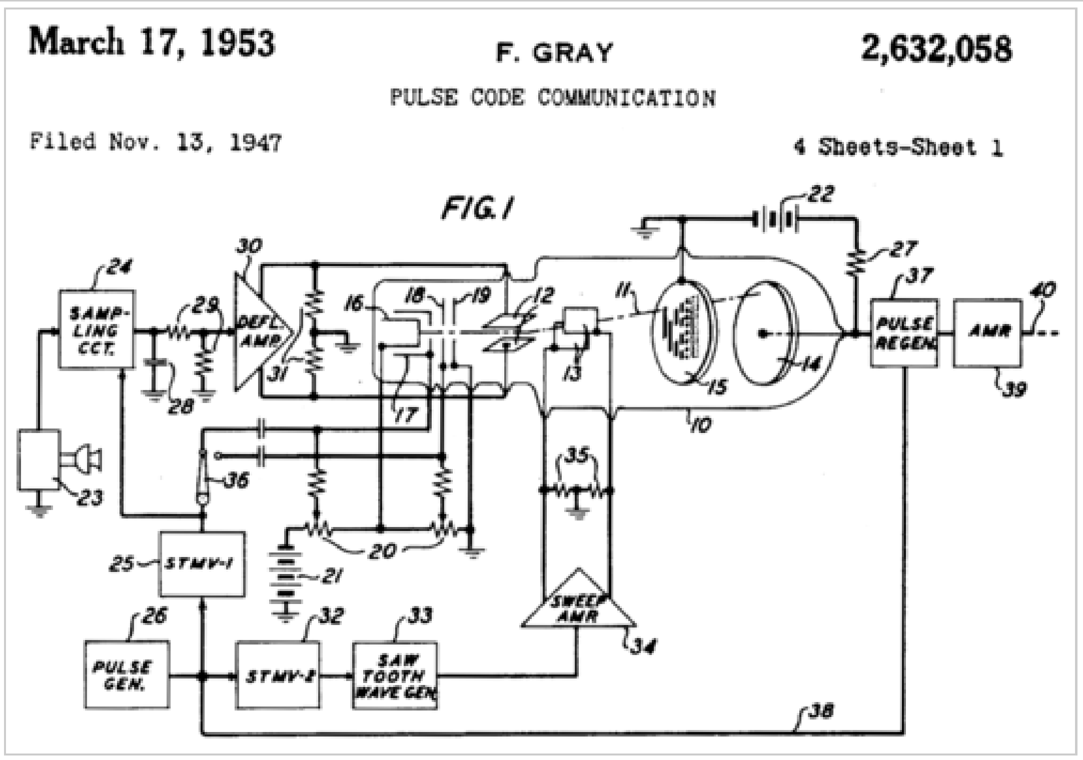
Notice that state 7 can roll over to state 0 with only one switch change. This is called the "cyclic" property of a Gray code. In the standard Gray coding the least significant bit follows a repetitive pattern of 2 on, 2 off (⋯ 11001100 ⋯); the next digit a pattern of 4 on, 4 off; and so forth.

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.

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]



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

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.

Position encoders

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 will result from some of the bits changing before others. Originally, the code pattern was electrically conductive, supported (in a rotary encoder) by an insulating disk. Each track had its own stationary metal spring contact; one more contact made the connection to the pattern. That common contact was connected by the pattern to whichever of the track contacts were resting on the conductive pattern. However, sliding contacts wear out and need maintenance, which favors optical encoders.

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

Regardless of the care in aligning the contacts, and accuracy of the pattern, a natural-binary code would have errors at specific disk positions, because it is impossible to make all bits change at exactly the same time as the disk rotates. The same is true of an optical encoder; transitions between opaque and transparent cannot be made to happen simultaneously for certain exact positions. Rotary encoders benefit from the cyclic nature of Gray codes, because consecutive positions of the sequence differ by only one bit. This means that, for a transition from state A to state B, timing mismatches can only affect when the A→B transition occurs, rather than inserting one or more (up to N-1 for an N-bit codeword) false intermediate states, as would occur if a standard binary code were used.

Towers of Hanoi

The binary-reflected Gray code can also be used to serve as a solution guide for the Towers of Hanoi problem, as well as the classical Chinese rings puzzle, a sequential mechanical puzzle mechanism.[5] It also forms a Hamiltonian cycle on a hypercube, where each bit is seen as one dimension.

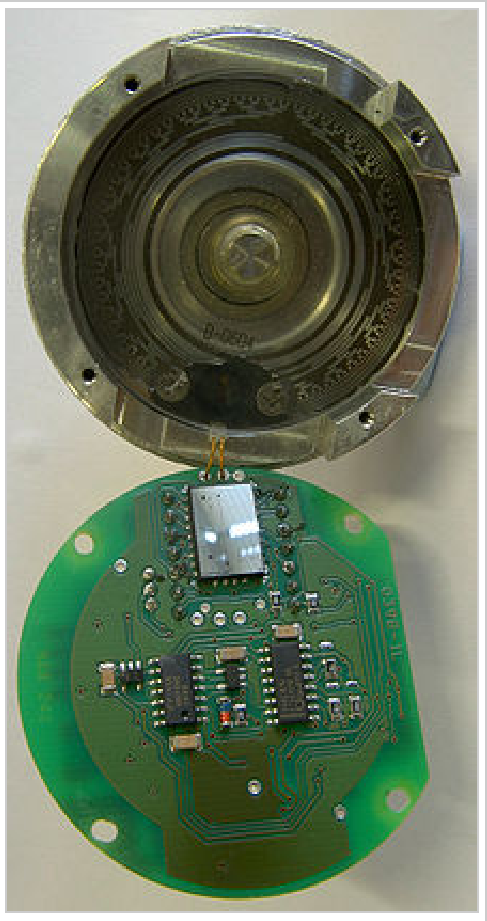
Genetic algorithms

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.

Karnaugh maps

Gray codes are also used in labelling the axes of Karnaugh maps.[8]

Error correction


A Gray code absolute rotary encoder with 13 tracks. At the top can be seen the housing, interrupter disk, and light source; at the bottom can be seen the sensing element and support components.

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.

Communication between clock domains

Main article: clock domain crossing

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.

Gray code counters and arithmetic

A typical use of Gray code counters is building a FIFO (first-in, first-out) data buffer that has read and write ports that exist in different clock domains. The input and output counters inside such a dual-port FIFO are often stored using Gray code to prevent invalid transient states from being captured when the count crosses clock domains.[9] 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.

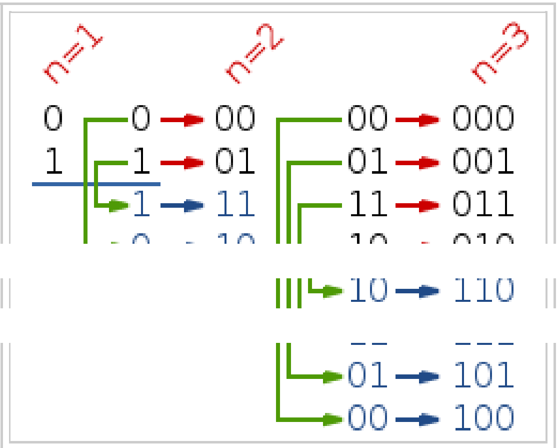
Sometimes digital buses in electronic systems are used to convey quantities that can only increase or decrease by one at a time, for example the output of an event counter which is being passed between clock domains or to a digital-to-analog converter. The advantage of Gray codes in these applications 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://www.google.com/patents/US3020481), 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 [10] and then subsequently patented by someone else in 1998 US5754614 (http://www.google.com/patents/US5754614). Other methods of counting in Gray code are discussed in a report by R. W. Doran, including taking the output from the first latches of the master-slave flip flops in a binary ripple counter.[11]

Perhaps the most common electronic counter with the "only one bit changes at a time" property is the Johnson counter.

Constructing an *n*-bit Gray code

The binary-reflected Gray code list for *n* bits can be generated recursively from the list for *n*−*1* bits by reflecting the list (i.e. listing the entries in reverse order), concatenating the original list with the reversed list, prefixing the entries in the original list with a binary 0, and then prefixing the entries in the reflected list with a binary 1.[5] For example, generating the *n* = 3 list from the *n* = 2 list:

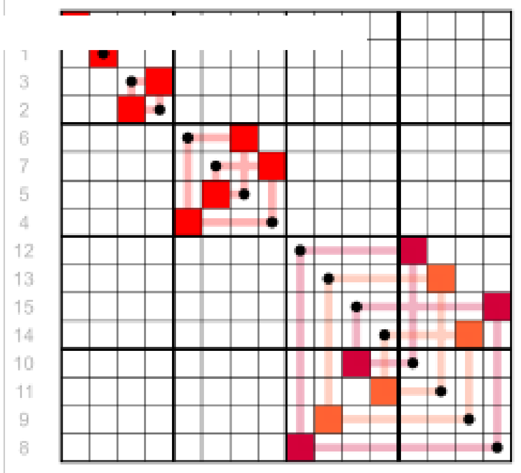
2-bit list:	00, 01, 11, 10	
Reflected:		10, 11, 01, 00
Prefix old entries with *0*:	000, 001, 011, 010,	
Prefix new entries with *1*:		110, 111, 101, 100
Concatenated:	000, 001, 011, 010,	110, 111, 101, 100



The first few steps of the reflect-and-prefix method.

The one-bit Gray code is G_1 = (0, 1). This can be thought of as built recursively as above from a zero-bit Gray code G_0 = { Λ } consisting of a single entry of zero length. This iterative process of generating G_{n+1} from G_n makes the following properties of the standard reflecting code clear:

- G_n is a permutation of the numbers 0, ..., 2^n-1. (Each number appears exactly once in the list.)
- G_n is embedded as the first half of G_{n+1}.
- Therefore the coding is *stable*, in the sense that once a binary number appears in G_n it appears in the same position in all longer lists; so it makes sense to talk about *the* reflective Gray code value of a number: G(*m*) = the *m*-th reflecting Gray code, counting from 0.
- Each entry in G_n differs by only one bit from the previous entry. (The Hamming distance is 1.)
- The last entry in G_n differs by only one bit from the first entry. (The code is cyclic.)



4-bit Gray code permutation

These characteristics suggest a simple and fast method of translating a binary value into the corresponding Gray code. 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: the *n*th Gray code is obtained by computing $n \oplus \lfloor n/2 \rfloor$

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. Assuming g_i is the ith gray-coded bit (g_0 being the most significant bit), and b_i is the ith binary-coded bit (b_0 being the most-significant bit), the reverse translation can be given recursively: $b_0 = g_0$, and $b_i = g_i \oplus b_{i-1}$. Alternatively, decoding a Gray code into a binary number can be described as a prefix sum of the bits in the Gray code, where each individual summation operation in the prefix sum is performed modulo two.

To construct the binary-reflected Gray code iteratively, at step 0 start with the , and at step find the bit position of the least significant 1 in the binary representation of and flip the bit at that position in the previous code to get the next code . The bit positions start 0, 1, 0, 2, 0, 1, 0, 3, ... (sequence A007814 in OEIS). See find first set for efficient algorithms to compute these values.

Converting to and from Gray code

The following functions in C convert between binary numbers and their associated Gray codes. While it may seem that gray-to-binary conversion requires each bit to be handled one at a time, faster algorithms exist.[12]

```
/*
        The purpose of this function is to convert an unsigned
        binary number to reflected binary Gray code.

        The operator >> is shift right. The operator ^ is exclusive or.
*/
unsigned int binaryToGray(unsigned int num)
{
        return (num >> 1) ^ num;
}

/*
        The purpose of this function is to convert a reflected binary
        Gray code number to a binary number.
*/
unsigned int grayToBinary(unsigned int num)
{
    unsigned int mask;
    for (mask = num >> 1; mask != 0; mask = mask >> 1)
    {
        num = num ^ mask;
    }
    return num;
}
```

Special types of Gray codes

In practice, a "Gray code" almost always refers to a binary-reflected Gray code (BRGC). 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.

Ternary number → ternary Gray code

0	000
1	001
2	002
10	012
11	010
12	011
20	021
21	022
22	020
100	120
101	121
102	122
110	102
111	100
112	101
120	111
121	112
122	110
200	210
201	211
202	212
210	222
211	220
212	221
220	201
221	202
222	200

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.[13] The sequence of elements in the (3, 2)-Gray code is: {00, 01, 02, 12, 10, 11, 21, 22, 20}. 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 is presented (in C):

```c
// inputs: base, digits, value
// output: gray
// Convert a value to a graycode with the given base and digits.
// Iterating through a sequence of values would result in a sequence
// of Gray codes in which only one digit changes at a time.
void to_gray(unsigned base, unsigned digits, unsigned value, unsigned gray[digits])
{
        unsigned baseN[digits]; // Stores the ordinary base-N number, one digit per entry
        unsigned i;             // The loop variable

        // Put the normal baseN number into the baseN array. For base 10, 109
        // would be stored as [9,0,1]
        for (i = 0; i < digits; i++) {
                baseN[i] = value % base;
                value    = value / base;
        }

        // Convert the normal baseN number into the graycode equivalent. Note that
        // the loop starts at the most significant digit and goes down.
        unsigned shift = 0;
        while (i--) {
                // The gray digit gets shifted down by the sum of the higher
                // digits.
                gray[i] = (baseN[i] + shift) % base;
                shift = shift + base - gray[i]; // Subtract from base so shift is positive
        }
}
// EXAMPLES
// input: value = 1899, base = 10, digits = 4
// output: baseN[] = [9,9,8,1], gray[] = [0,1,7,1]
// input: value = 1900, base = 10, digits = 4
// output: baseN[] = [0,0,9,1], gray[] = [0,1,8,1]
```

There are other graycode algorithms for *(n,k)*-Gray codes. It is important to note that the *(n,k)*-Gray codes produced by the above algorithm is always cyclical; some algorithms, such as that by Guan,[13] lack this property when k is odd. On the other hand, while only one digit at a time changes with this method, it can change by wrapping (looping from n-1 to 0). In

Guan's algorithm, the count alternately rises and falls, so that the numeric difference between two graycode digits is always one.

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 the lower the significance of a digit, the more often it changes, making it similar to normal counting methods.

Balanced Gray code

Although the binary reflected Gray code is useful in many scenarios, it is not optimal in certain cases because of a lack of "uniformity".[14] In balanced Gray codes, the number of changes in different coordinate positions are as close as possible. To make this more precise, let G be an *R*-ary complete Gray cycle having transition sequence (δ_k); the *transition counts (spectrum)* of G are the collection of integers defined by

$$\lambda_k = |\{j \in \mathbb{Z}_{R^n} : \delta_j = k\}|\,,\ \text{for } k \in \mathbb{Z}_R$$

A Gray code is *uniform* or *uniformly balanced* if its transition counts are all equal, in which case we have $\lambda_k = R^n/n$ for all *k*. Clearly, when $R = 2$, such codes exist only if *n* is a power of 2. Otherwise, if *n* does not divide R^n evenly, it is possible to construct *well-balanced* codes where every transition count is either $\lfloor R^n/n \rfloor$ or $\lceil R^n/n \rceil$. Gray codes can also be *exponentially balanced* if all of their transition counts are adjacent powers of two, and such codes exist for every power of two.[15]

For example, a balanced 4-bit Gray code has 16 transitions, which can be evenly distributed among all four positions (four transitions per position), making it uniformly balanced:[14]

```
0111111000000110
0011110011110000
0000111110011100
0001100000111111
```

whereas a balanced 5-bit Gray code has a total of 32 transitions, which cannot be evenly distributed among the positions. In this example, four positions have six transitions each, and one has eight:[14]

```
11111000011111100111110000000000
00011111111000000011111100011000
11001110000001110001111110000011
10000000111111000000111111110001
11111100001100000000011000111111
```

We will now show a construction for well-balanced binary Gray codes which allows us to generate an *n*-digit balanced Gray code for every *n*.[16] The main principle is to inductively construct an $(n+2)$-digit Gray code given an *n*-digit Gray code G in such a way that the balanced property is preserved. To do this, we consider partitions of into an even number *L* of non-empty blocks of the form

where , and (mod). This partition induces an -digit Gray code given by

If we define the *transition multiplicities* to be the number of times the digit in position *i* changes between consecutive blocks in a partition, then for the -digit Gray code induced by this partition the transition spectrum is

The delicate part of this construction is to find an adequate partitioning of a balanced *n*-digit Gray code such that the code induced by it remains balanced. Uniform codes can be found when and , and this construction can be extended to the *R*-ary case as well.[16]

Monotonic Gray codes

Monotonic codes are useful in the theory of interconnection networks, especially for minimizing dilation for linear arrays of processors.[17] If we define the *weight* of a binary string to be the number of 1's in the string, then although we clearly cannot have a Gray code with strictly increasing weight, we may want to approximate this by having the code run through two adjacent weights before reaching the next one.

We can formalize the concept of monotone Gray codes as follows: consider the partition of the hypercube $Q_n = (V_n, E_n)$ into *levels* of vertices that have equal weight, i.e.

$$V_n(i) = \{v \in V_n : v \text{ has weight } i\}$$

for $0 \leq i \leq n$. These levels satisfy $|V_n(i)| = \binom{n}{i}$. Let $Q_n(i)$ be the subgraph of Q_n induced by $V_n(i) \cup V_n(i+1)$, and let $E_n(i)$ be the edges in $Q_n(i)$. A monotonic Gray code is then a Hamiltonian path in Q_n such that whenever $\delta_1 \in E_n(i)$ comes before $\delta_2 \in E_n(j)$ in the path, then $i \leq j$.

An elegant construction of monotonic *n*-digit Gray codes for any *n* is based on the idea of recursively building subpaths $P_{n,j}$ of length $2\binom{n}{j}$ having edges in $E_n(j)$.[17] We define $P_{1,0} = (0, 1)$, $P_{n,j} = \emptyset$ whenever $j < 0$ or $j \geq n$, and

$$P_{n+1,j} = 1P^{\pi_n}_{n,j-1}, 0P_{n,j}$$

otherwise. Here, π_n is a suitably defined permutation and P^{π} refers to the path *P* with its coordinates permuted by π. These paths give rise to two monotonic *n*-digit Gray codes $G^{(1)}_n$ and $G^{(2)}_n$ given by

$$G^{(1)}_n = P_{n,0}P^R_{n,1}P_{n,2}P^R_{n,3}\cdots \text{ and } G^{(2)}_n = P^R_{n,0}P_{n,1}P^R_{n,2}P_{n,3}\cdots$$

The choice of π_n which ensures that these codes are indeed Gray codes turns out to be $\pi_n = E^{-1}(\pi^2_{n-1})$. The first few values of $P_{n,j}$ are shown in the table below.

These monotonic Gray codes can be efficiently implemented in such a way that each subsequent element can be generated in O(*n*) time. The algorithm is most easily described using coroutines.

Subpaths in the Savage-Winkler algorithm

$P_{n,j}$	*j* = 0	*j* = 1	*j* = 2	*j* = 3
n = 1	0, 1			
n = 2	00, 01	10, 11		
n = 3	000, 001	100, 110, 010, 011	101, 111	
n = 4	0000, 0001	1000, 1100, 0100, 0110, 0010, 0011	1010, 1011, 1001, 1101, 0101, 0111	1110, 1111

Monotonic codes have an interesting connection to the Lovász conjecture, which states that every connected vertex-transitive graph contains a Hamiltonian path. The "middle-level" subgraph $Q_{2n+1}(n)$ is vertex-transitive (that is, its automorphism group is transitive, so that each vertex has the same "local environment"" and cannot be differentiated from the others, since we can relabel the coordinates as well as the binary digits to obtain an automorphism) and the problem of finding a Hamiltonian path in this subgraph is called the "middle-levels problem", which can provide insights into the more general conjecture. The question has been answered affirmatively for $n \leq 15$, and the preceding construction for monotonic codes ensures a Hamiltonian path of length at least 0.839*N* where *N* is the number of vertices in the middle-level subgraph.[18]

Beckett–Gray code

Another type of Gray code, the Beckett–Gray code, is named for Irish playwright Samuel Beckett, who was interested in symmetry. His play "Quad" features four actors and is divided into sixteen time periods. Each period ends with one of the four actors entering or leaving the stage. The play begins with an empty stage, and Beckett wanted each subset of actors to appear on stage exactly once.[19] Clearly the set of actors currently on stage can be represented by a 4-bit binary Gray code. Beckett, however, placed an additional restriction on the script: he wished the actors to enter and exit so 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, so that (of the actors onstage) the actor being dequeued is always the one who was enqueued first.[19] 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. It is known today that such codes do exist for *n* = 2, 5, 6, 7, and 8, and do not exist for *n* = 3 or 4. An example of an 8-bit Beckett–Gray code can be found in Knuth's *Art of Computer Programming*.[5] According to Sawada and Wong, 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.[20]

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;[21] 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 (STGC) developed by N. B. Spedding (NZ Patent 264738 - October 28, 1994)[22] and refined by Hiltgen, Paterson and Brandestini in "Single-track Gray codes" (1996).[23] 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 x n* matrix, each column is a cyclic shift of the first column.[24]

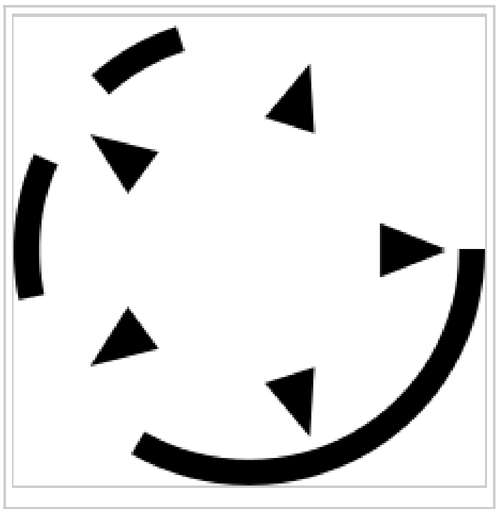
The name comes from their use with rotary encoders, where a number of tracks are being sensed by contacts, resulting for each in an output of 0 or 1. To reduce noise due to different contacts not switching at exactly the same moment in time, one preferably sets up the tracks so that the data output by the contacts are in Gray code. To get high angular accuracy, one needs lots of contacts; in order to achieve at least 1 degree accuracy, one needs at least 360 distinct positions per revolution, which requires a minimum of 9 bits of data, and thus the same number of contacts.

If all contacts are placed at the same angular position, then 9 tracks are needed to get a standard BRGC with at least 1 degree accuracy. However, 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 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 were too bulky, people used single-track incremental encoders (quadrature encoders) or 2-track "quadrature encoder + reference notch" encoders.

N. B. Spedding, however, registered a patent in 1994 with several examples showing that it was possible.[22] Although it is not possible to distinguish 2^n positions with n sensors on a single track, it *is* possible to distinguish close to that many. For example, when n is itself a power of 2, n sensors can distinguish 2^n-2n positions. Hiltgen and Paterson published a paper in 2001 exhibiting a single-track gray code with exactly 360 angular positions, constructed using 9 sensors.[25] Since this number is larger than $2^8 = 256$, more than 8 sensors are required by any code, although a BRGC could distinguish 512 positions with 9 sensors. 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.[26] 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, also called De Bruijn sequences), 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.[27]

See also

- Linear feedback shift register
- De Bruijn sequence
- Gillham code
- Steinhaus–Johnson–Trotter algorithm, an algorithm that generates Gray codes for the factorial number system

Notes

1. ^ F. Gray. *Pulse code communication*, March 17, 1953 (filed Nov. 1947). U.S. Patent 2,632,058 (http://www.google.com/patents/US2632058)
2. ^ J. Breckman. *Encoding Circuit*, Jan 31, 1956 (filed Dec. 1953). U.S. Patent 2,733,432 (http://www.google.com/patents/US2733432)
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://www.google.com/patents/US2823345)
4. ^ S. Reiner et al. *Automatic Rectification System*, Jun 24, 1958 (filed Jan. 1954). U.S. Patent 2,839,974 (http://www.google.com/patents/US2839974)
5. ^ *a* *b* *c* *d* 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. ^ Cattermole, K. W. (1969). *Principles of Pulse Code Modulation*. New York: American Elsevier. ISBN 0-444-19747-8.

6. ^ Cattermole, K. W. (1969). *Principles of Pulse Code Modulation*. New York: American Elsevier. ISBN 0-444-19747-8.
7. ^ Goodall, W. M. (1951). "Television by Pulse Code Modulation". *Bell Sys. Tech. J.* 30: 33–49.
8. ^ Wakerly, John F (1994). *Digital Design: Principles & Practices*. New Jersey: Prentice Hall. pp. 222, 48–49. ISBN 0-13-211459-3. Note that the two page sections taken together say that K-maps are labeled with Gray code. The first section says that they are labeled with a code that changes only one bit between entries and the second section says that such a code is called Gray code.
9. ^ "Synchronization in Digital Logic Circuits (http://www.stanford.edu/class/ee183/handouts_spr2003/synchronization_pres.pdf) by Ryan Donohue
10. ^ Mehta, H.; Owens, R.M. & Irwin, M.J. (1996), Some issues in gray code addressing (http://ieeexplore.ieee.org/xpls/abs_all.jsp?tp=&arnumber=497616&isnumber=10625), in the Proceedings of the 6th Great Lakes Symposium on VLSI (GLSVLSI 96), IEEE Computer Society,pp. 178
11. ^ The Gray Code by R. W. Doran (http://www.cs.auckland.ac.nz/CDMTCS//researchreports/304bob.pdf)
12. ^ Henry Gordon Dietz. "The Aggregate Magic Algorithms: Gray Code Conversion" (http://aggregate.org/MAGIC/#Gray%20Code%20Conversion)
13. ^ *a b* Guan, Dah-Jyh (1998). "Generalized Gray Codes with Applications" (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.119.1344&rep=rep1&type=pdf) (PDF). *Proc. Natl. Sci. Counc. Repub. Of China (A)* 22: 841–848.
14. ^ *a b c* Bhat, Girish S.; Savage, Carla D. (1996). "Balanced Gray codes" (http://www.combinatorics.org/Volume_3/Abstracts/v3i1r25.html). *Electronic Journal of Combinatorics* 3 (1): R25.
15. ^ Suparta, I. N. (2005). "A simple proof for the existence of exponentially balanced Gray codes". *Electronic Journal of Combinatorics* 12.
16. ^ *a b* M. Flahive and B. Bose (2007). "Balancing cyclic R-ary Gray codes". *Electronic Journal of Combinatorics* 14.
17. ^ *a b* C. D Savage and P. Winkler (1995). "Monotone Gray codes and the middle levels problem". *Journal of Combinatorial Theory, Series A* 70 (2): 230–248. doi:10.1016/0097-3165(95)90091-8 (http://dx.doi.org/10.1016%2F0097-3165%2895%2990091-8). ISSN 0097-3165 (//www.worldcat.org/issn/0097-3165).
18. ^ C. D Savage (1997). *Long cycles in the middle two levels of the Boolean lattice*.
19. ^ *a b* Goddyn, Luis (1999). "MATH 343 Applied Discrete Math Supplementary Materials" (http://www.math.sfu.ca/~goddyn/Courses/343/supMaterials.pdf) (PDF). Dept. of Math, Simon Fraser U.
20. ^ Wong, J. (2007). "A Fast Algorithm to generate Beckett-Gray codes". *Electronic Notes in Discrete Mathematics* 29: 571–577. doi:10.1016/j.endm.2007.07.091 (http://dx.doi.org/10.1016%2Fj.endm.2007.07.091).
21. ^ Kautz, W. H. (1958). "Unit-distance error-checking codes". *IRE Trans. Elect. Comput.* 7: 177–180.
22. ^ *a b* ([http://www.winzurf.co.nz/Single_Track_Grey_Code_Patent/Single_track_Grey_code_encoder_patent.pdf "Single track grey code" (http://worldwide.espacenet.com/textdoc?DB=EPODOC&IDX=NZ264738). winzurf.co.nz) NZ A method of creating a single track absolute position rotary or linear encoder of varying resolutions using equi-spaced detectors. 264738 ("Single track grey code" (http://www.winzurf.co.nz/Single_Track_Grey_Code_Patent/Single_track_Grey_code_encoder_patent.pdf). winzurf.co.nz)], [|Spedding, Norman Bruce (http://www.iponz.govt.nz/cms/patents/banner_template/IPPATENT)], "A position encoder", published 1994
23. ^ Hiltgen, Alain P.; Kenneth G. Paterson; Marco Brandestini (1996). "Single-Track Gray Codes" (http://ieeexplore.ieee.org/iel1/18/11236/00532900.pdf) (PDF). *IEEE Transactions on Information Theory* 42 (5): 1555–1561. doi:10.1109/18.532900 (http://dx.doi.org/10.1109%2F18.532900).
24. ^ Etzion, Tuvi; Moshe Schwartz (1999). "The Structure of Single-Track Gray Codes" (http://www.cs.technion.ac.il/~etzion/PUB/Gray2.pdf) (PDF). *IEEE Transactions on Information Theory* 45 (7): 2383–2396. doi:10.1109/18.796379 (http://dx.doi.org/10.1109%2F18.796379).
25. ^ Hiltgen, Alain P.; Kenneth G. Paterson (2001). "Single-Track Circuit Codes" (http://www.hpl.hp.com/techreports/2000/HPL-2000-81.pdf) (PDF). *IEEE Transactions on Information Theory* 47 (6): 2587–2595. doi:10.1109/18.945274 (http://dx.doi.org/10.1109%2F18.945274).
26. ^ "Venn Diagram Survey — Symmetric Diagrams" (http://www.combinatorics.org/Surveys/ds5/VennSymmEJC.html). *Electronic Journal of Combinatorics*. 2001.
27. ^ Alciatore, David G.; Michael B. Histand (1999). *Mechatronics* (http://mechatronics.colostate.edu/). McGraw-Hill Education - Europe. ISBN 978-0-07-131444-2.

References

- Black, Paul E. *Gray code* (http://www.nist.gov/dads/HTML/graycode.html). 25 February 2004. NIST.
- Press, WH; Teukolsky, SA; Vetterling, WT; Flannery, BP (2007). "Section 22.3. Gray Codes" (http://apps.nrbook.com/empanel/index.html#pg=1166). *Numerical Recipes: The Art of Scientific Computing* (3rd ed.). New York: Cambridge University Press. ISBN 978-0-521-88068-8.
- Savage, Carla (1997). "A Survey of Combinatorial Gray Codes" (http://www.csc.ncsu.edu/faculty/savage/AVAILABLE_FOR_MAILING/survey.ps). *SIAM Rev.* 39 (4): 605–629. doi:10.1137/S0036144595295272 (http://dx.doi.org/10.1137%2FS0036144595295272). JSTOR 2132693 (//www.jstor.org/stable/2132693).
- Wilf, Herbert S. (1989). "Chapters 1-3". *Combinatorial algorithms: an update*. SIAM. ISBN 0-89871-231-9.

External links

- "Gray Code" demonstration (http://demonstrations.wolfram.com/BinaryGrayCode/) by Michael Schreiber, Wolfram Demonstrations Project (with Mathematica implementation). 2007.
- NIST Dictionary of Algorithms and Data Structures: Gray code (http://www.nist.gov/dads/HTML/graycode.html)
- 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=39) by Zinovik, I.; Kroening, D.; Chebiryak, Y.
- AMS Column: Gray codes (http://www.ams.org/featurecolumn/archive/gray.html)
- Optical Encoder Wheel Generator (http://www.bushytails.net/~randyg/encoder/encoderwheel.html)
- ProtoTalk.net - Understanding Quadrature Encoding (http://prototalk.net/forums/showthread.php?t=78) - Covers quadrature encoding in more detail with a focus on robotic applications