BRIDGES
Mathematical Connections
in Art, Music, and Science

The Reflected Binary Gray Code Transform

Steve Whealton
1725 Kilbourne Place, NW
Washington, DC 20010-2605, USA
E-mail: swhealton@ctsmd.com

Abstract

The "Reflected Binary Gray Code" is a system of representing numbers using zeroes and ones. It is closely allied to the familiar binary system for representing numbers, also using zeroes and ones. It is easy to switch back and forth between one system and the other. This paper examines this "transform" that switches back and forth, and focuses especially upon the uses to which this transform can be put.

1. What is a Gray Code?

1.1. General Definition. A Gray Code is a system where similar patterns are represented similarly. More precisely, any two adjacent numbers (numbers whose difference is one) must have their Gray-Code patterns differ in only one place. Even there, those two patterns may differ only by one unit.

1.2. Background. Frank Gray worked for Bell Labs during the 1930s through the 1950s. He received U.S. patent 2 632 058 for the Gray Code in 1953 [1]. Legend has it that it was George Stibitz (1904-1995), also at Bell Labs, and not Gray, who first "invented" the Gray Code. The French engineer Émile Baudot had used Gray-code-like ideas in telegraphy in 1878. He received the French Legion of Honor medal for his work.

2. *The* Gray Code

2.1. Reflected Binary Gray Code. There are myriads, even infinities, of legitimate Gray Codes. Out of this multitude, one particular realization of the Gray Code criterion has come to be the most important and useful. Its proper full name is the "binary reflected gray code" (BRGC). For most people who use it or know about it, it's simply "The Gray Code."

The word, "binary" acknowledges the fact that The Binary Reflected Gray Code is a system of representing numbers using ones and zeroes, and also that it is a cousin of the familiar system of representing numbers using ones and zeroes that computers regularly utilize. It is a cousin so close that one can easily switch back and forth between regular base-two bit-patterns and BRGC bit-patterns for any given integer.

What the Binary Reflected Gray Code transformation (BRGCT) does is to take any positive integer and find that number whose regular bit-pattern is the BRGC bit-pattern for the original integer. Except for the numbers one and zero, this second number is always different from the first one.

Besides meeting the Gray Code criterion about representing numbers near one another similarly, the BRGCT has many other useful and beautiful properties.

2.2. Programming. The BRGCT is almost embarrassingly easy to program. This is the case because many programming languages have the two necessary procedures built right into them. At code level, too, the entire procedure is simple and straightforward.

One of the two procedures is what one might call "fixed-point division." In BASIC, this operation has even been given its own symbol, the "\". Thus, "**a = b \ c**" would mean to divide "b" by "c," to select "a" as the quotient, and then finally to throw away the remainder.

The other procedure is designated by "XOR" in BASIC and by the "^" symbol in C and in C++. Its name comes from logic, and from the phrase "exclusive or." In everyday English, one thinks first of taking one thing or another, but not both.

From its beginnings in the realms of logic, the "XOR" notion has evolved into an operator that can be used between two whole numbers. This operator considers comparable bits, applying the "one or the other but not both" idea to produce a result. It is equivalent to bitwise addition, modulo two, and also to Nim Addition.

To find the BRGC Disguise (equivalent) for any given integer, one must first halve the number, throwing away the remainder. One must make sure that the original number is also retained. The final step is simply to perform the XOR operation between the original number and the halved-and-trimmed number.

The result is the BRGC equivalent.

BRGC(x) = x XOR (x \ 2) [BASIC]

brgc = x ^ trunk(x/2); [C++, Java, &c.]

3. Using *The* Gray Code

3.1. Transforms. In looking for ways to disguise numbers usefully, one hopes to find a few interesting combinations of regularity and irregularity. With the BRGCT, such combinations abound.

For one thing, the BRGCT equivalent of a number is usually just a little bit smaller or a little bit larger than the original number itself. It can never be less than half of the parent number, nor more than double it. Except in the cases of zero and one, it also can never be the number, itself.

What happens when the BRGCT were used over and over, taking the output of one application of the BRGC and treating it as the input for an additional BRGC transformation?

If the BRGCT algorithm is applied again and again, the original number eventually returns. The question of how soon it will return leads to perhaps the single most fascinating and useful of the BRGC's qualities.

It happens that "10" is the BRGC-equivalent bit-pattern for three, whereas "11" is the BRGC bit-pattern for two. So one could say, using BRGC notation, that **BRGC(2) = 3** and also that **BRGC(3) = 2**. Similarly, **BRGC(0) = 0** and **BRGC(1) = 1**. Moving through the next few numbers, we find that **BRGC(4) = 6**, that **BRGC(5) = 7**, that **BRGC(6) = 5**, and that **BRGC(7) = 4**.

At this point, we can also see that 0 and 1 exist in cycles of length one; that 2 and 3 belong to a cycle of length two; and that 4, 5, 6, and 7 belong to a cycle of length four.

It happens that all the numbers between 4 and 15 are members of cycles of size four, while all the numbers between 16 and 255 are members of cycles of size 8. That is as high as I've investigated

exhaustively. But my intuition, along with a few random testings, suggest that for all numbers between 256 and 65535 the cycles are of size 16.

Before listing cycles for numbers larger than 7, a few terms will be defined. The length of a cycle is one more than the number of transformations necessary before encountering a number already in the given cycle. The head of a cycle is the smallest number in it. The tail of a cycle is the final number in the cycle begun with the head. Note that the tail is not necessarily the largest number in a cycle, though sometimes it will be. Note that BRGC(tail) = head, for any cycle.

Next, the cycles already described, using the above terms.

The length of the first cycle is one. Its head and tail are both zero. In the second cycle, the length is again one, whereas this time the head and tail are both one. In the third cycle, the length is two, the head is two, and the tail is three.In the fourth cycle, the length is four, the head is four, and the tail is seven.

But the tail is seven not because it is the largest number in the cycle, but rather because it is the number that immediately precedes four, which is the smallest number.

Below are listed the cycles for all integers between 0 and 255, inclusive. Each list begins with the cycle's head, and ends with its tail.

Cycles of Length One	
0	**1**

Cycle of Length Two
2 3

Cycles of Length Four		
4 6 5 7	**8 12 10 15**	**9 13 11 14**

Cycles of Length Eight Set One – Two Cycles	
16 24 20 30 17 25 21 31	**18 27 22 29 19 26 23 28**

Cycles of Length Eight Set Two – Four Cycles	
32 48 40 60 34 51 42 63	**33 49 41 61 35 50 43 62**
36 54 45 59 38 53 47 56	**37 55 44 58 39 52 46 57**

Cycles of Length Eight Set Three – Four Cycles	
64 96 80 120 68 102 85 127	**65 97 81 121 69 103 84 126**
66 99 82 123 70 101 87 124	**67 98 83 122 72 100 86 125**

Cycles of Length Eight Set Four – Four Cycles	
72 108 90 119 76 106 95 112	**73 109 91 118 77 107 94 113**
74 111 88 116 78 105 93 115	**75 110 89 117 79 104 92 114**

Cycles of Length Eight Set Five – Eight Cycles	
128 192 160 240 136 204 170 255	**129 193 161 241 137 205 171 254**
130 195 163 243 138 207 168 252	**131 194 163 242 139 206 169 253**
132 198 165 247 140 202 175 248	**133 199 164 246 141 203 174 249**
134 197 167 244 142 201 173 251	**135 196 166 245 143 200 172 250**

Cycles of Length Eight Set Six – Eight Cycles	
144 216 180 238 153 213 191 224	**145 217 181 239 152 212 190 225**
146 219 182 237 155 214 189 227	**147 218 183 236 154 215 188 226**
148 222 177 233 157 211 186 231	**149 223 176 232 156 210 187 230**
150 221 179 234 159 208 184 228	**151 220 178 235 158 209 185 229**

3.2. Using the BRGCT. Taken together, the diverse features of the BRGCT make it a useful tool. Equally important, when a simple BRGC disguise is used with one or more numbers being fed into a simple formula, the musical or visual result is quite often an interesting one. Let us now examine a few of the ways that it can be used.

3.3. Designing a Good Transform. First, a brief discussion of criteria and for designing transformations. One of these criteria is what one might informally call a certain kind of neatness.

Suppose one is inventing a transformative algorithm to operate only upon the integers from 0 through 255. It seems somehow much cleaner and in many ways also much more useful if this transformation of ours will be one-to-one. That is to say that if one makes a chart showing all 256 of the integers from 0 up to 255 along with their transformed equivalents, each number will appear once and only once on both sides of the chart. There are basic terms for these qualities, some of them being "into," "onto," and "isomorphic."

Most of the time, one must insist that the transformation amount to an isomorphism.

What will happen when the transformation is applied over and over again (iteration)? There are 256! Permutations of the integers between 0 and 255, and therefore there are that same vast number of isomorphic transformations.

3.3. Brevity and Ease. Of these isomorphic transformations, only those that can be defined with ease and brevity are useful. After all, any one of the 256! transformations can be "defined" simply by stating the equivalent for each of the 256 numbers. This could be called a "hard-wiring" method.

Useful transformations strive to be briefer than this. To achieve such brevity, one must make use of the tools available to us in one's programming language of choice, along with whatever mathematical notions can be devised.

3.4. Cycles with the BRGCT. The BRGC transformation does not treat each number similarly in the cycles that it produces. Recall that the BRGCT creates two cycles of length one, one cycle of length two, three cycles of length four, and thirty cycles of length eight.

One might characterize this breakdown as "diversity, but not chaos."

Why is such a balance between regularity and irregularity pleasing? There are two answers to this.

Deeper, perhaps, is the fact that one might prefer a balance between the two extremes of over-regularity and total randomness. But on a more practical level, there is also an advantage in having cycles of varying lengths. This advantage can be seen by examining a few of the ways in which transformations can be used.

3.6. Changing Color Index Numbers. Many pictures are made by recoloring images that have been created outside the computer. Scanned, 8-bit paletted bitmaps of photographs or paintings are often the starting point. A piece of paint, metal, leaf, or bark might also be used as a beginning.

When using picture-making software that can only transform 8-bit imagery, one is forced to render all bitmaps into the 8-bit, paletted, format. In more directly mathematical terms, this means that each image can be thought of as a vast array of integers, each of one of which is eight bits in length, between 0 and 255.

Before a picture is submitted for mathematical pixel-by-pixel transformation, it must first have been rendered into the requisite eight-bit format. Typically, one uses one of the well-known software packages (e.g. Adobe PhotoShop, Corel PhotoPaint, JASC Paint Shop Pro.) to reduce the original 24-bit "true-color" images produced by my camera or scanner to the 8-bit paletted format required by my algorithmic software.

Basically, the 256 colors that these applications create are chosen in such a way that the resulting picture will resemble the 24-bit original image as closely as possible. So the original image's most common colors, broadly speaking, appear in the palette that is created, while the rarer colors have to be left out.

Once the 256 most representative colors are cobbled together by the software, all of the image's pixels are fudged to whichever color in the palette is deemed closest in some useful sense to the pixel's original color.

Illustrations

Figure 0: *Small interior portion of my scanned painting, "Buffalo #5."*

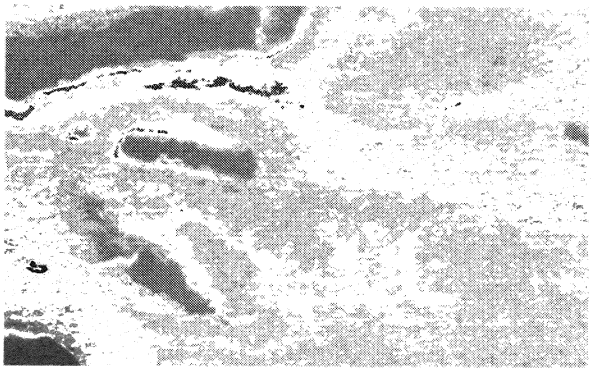

Figure 1: *Buffalo #5, as transformed by 1 application of BRGCT to Power Numbers."*

Figure 2: *Buffalo #5, as transformed by 2 iterations of BRGCT, using Power Numbers."*

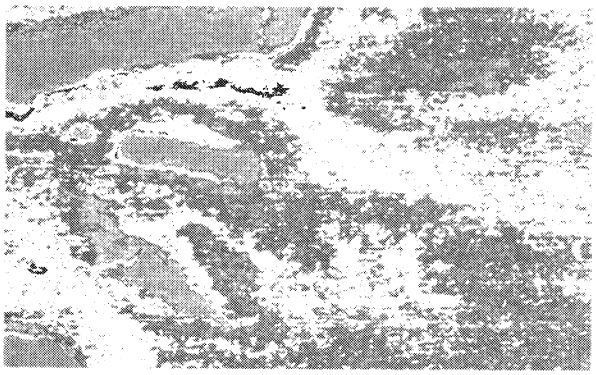

Figure 3: *Buffalo #5, as transformed by 3 iterations of BRGCT using Power Numbers."*

Figure 4: *Buffalo #5, as transformed by 4 iterations of BRGCT, using Power Numbers."*

Figure 5: *Buffalo #5, as transformed by 5 iterations of BRGCT, using Power Numbers."*

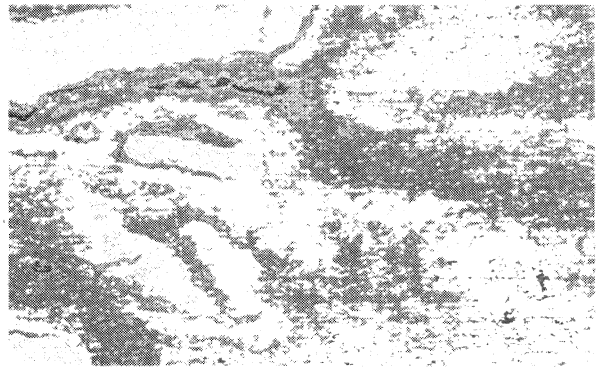

Figure 6: *Buffalo #5, as transformed by 6 iterations of BRGCT, using Power Numbers."*

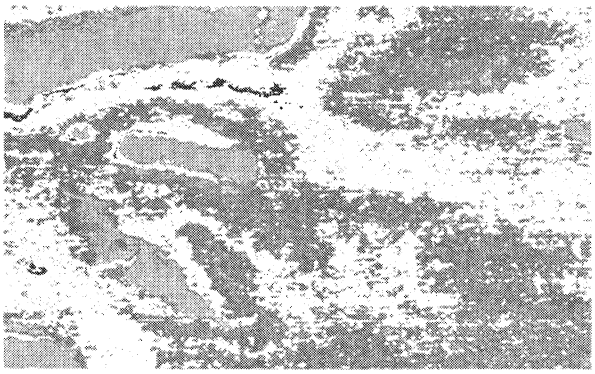

Figure 7: *Buffalo #5, as transformed by 7 iterations of BRGCT using Power Numbers."*

References

[1] F. Gray, *Pulse Code Communication*, U. S. Patent 2 632 058, March 17, 1953.

[2] F. G. Heath, "Origins of the Binary Code", *Scientific American* v.227,n.2; August, 1972, p.76.

[3] Martin Gardner, "Mathematical Games", *Scientific American* v.227,n.2; August, 1972, p.106.

[4] William H. Press, et al., *Numerical Recipes in C*, Second Edition; Cambridge University Press, 1992.

[5] Paul Horowitz and Winfield Hill, *The Art of Electronics*, Second Edition; Cambridge University Press, 1989.

[6] Dexter Kozen, *The Design and Analysis of Algorithms;* Springer-Verlag, New York, NY, 1992.

[7] Edward M. Reingold, et al., *Combinatorial Algorithms* Prentice Hall, Englewood Cliffs, NJ, 1977.

[8] David E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*; Addison-Wesley, Reading, MA, 1989.

[9] R. B. Hollstien, *Artificial Genetic Adaptation in Computer Control Systems* PhD thesis, University of Michigan, 1971.

[10] Albert Nijenhuis and Herbert S. Wilf, *Combinatorial Algorithms*, Academic Press, Inc., New York, San Francisco, London 1975.