Discussion of runtime:

Input(R,G,B)	Output(matrix)(Co,Y,Cg)	Runtime(matrix)(s)	Output(lifting)(Co,Y,Cg)	Runtime(lifting)(s)
255,0,0	127,63,-63	2.98e-07	127,63,-64	2.26e-07
0,255,0	0,127,127	3.05e-07	0,127,127	1.77e-07
0,0,255	-128,63,-64	3.47e-07	-128,63,-63	3.49e-07
0,0,0	0,0,0	2.57e-07	0,0,0	2.07e-07
255,255,255	0,254,0	1.87e-07	0,255,0	9.4e-08
166,25,46	60,65,-41	1.54e-07	60,65,-41	8.7e-08
193,99,113	40,125,-27	3.04e-07	40,126,-27	2.48e-07
66,133,244	-89,143,-11	3.42e-07	-89,144,-11	2.02e-07
234,67,53	90,104,-38	2.98e-07	90,105,-38	1.99e-07
254,188,5	124,158,30	3.41e-07	124,158,29	2.13e-07

Fig-1 result of sample input

The above table is the result of sample input of int version:

For direct matrix multiplication, there will be 9 multiplications and 6 additions. Some optimizations, such as canceling out the one that multiplies zero, can be performed to reduce the operations to 5 multiplications and 5 additions.

By running the algorithms many times in a loop, we can get the average running time for a single conversion. **For int version,** for different input values, the average running time of lift_based calculation is within the range of [8 x 10^{-8} s,1.4x10^{-7}s], while that of matrix multiplication is in [1.3x10^{-7} s,2.1x10^{-7} s].

As expected, the runtime of matrix multiplications is longer than that of lifting, as it requires more operations to compute the outputs.

For double version, the runtime of direct matrix multiplications is around [2.4x10 s, 3.4x10 s] while lift_based calculations run slightly faster with the average runtime falls into [1.9x10 s, 2.5x10 s].

Two interesting results has been observed for double version:

1. It runs almost five times faster than the int version. And the only implementation difference between these two versions is that the int version uses bitwise operations, such as right shift, since shifting does not work on doubles. This result is not expected since bitwise operations should run faster than the direct multiplications and int operations should also run faster than floating point operations. The reason behind this unexpected result may due to the compiler optimization, as compiler may optimize the multiplications but not bitwise operations. Another fact that may contribute to this result is where the data reside, the processor may put the double, which occupies more space, in the cache, thus a slower floating point operation will run much faster than an int operation that fetch int data from the memory.
2. Occasionally, matrix multiplications runs faster than the lift_based one, which is counter-intuitive since lift_based version has less operations and should run faster. This unexpected result may also due to the compiler optimization and CPU architecture. Also, the matrix version only requires 6 writes per iteration while the lifting version requires 7 writes per iteration, which also affects their performance.

Discussion of precision:

For double version, the output of two versions is always equal. However, for int version, some outputs of matrix multiplication may be one less/greater than that of lifting version. This may be the consequence of roundoff error, as the computed result will be rounded down to the nearest integer. But, for lifting version, it is lossless due to some math theorem S-transform, which converts the operations into integer arithmetic.