We've talked about the fact that most of the time, graphics is simply plotting pixels to memory. This may not be too dramatic, nor interesting, until you realize that plotting pixels to memory can produce quite interesting and cool looking effects. This section is not to teach you any specific algorithm, but just to show you that simply by plotting pixels into memory, a very small program can achieve very interesting results. I don't feel that what we've been doing so far is interesting enough to be inspiring; so, here's where this simple example comes in...

Have you ever played chess? Doesn't matter what your answer is (in fact, it was meant to be a rhetorical question ;-). Anyway, the examples we're about to create is going to kind of a chess board (not of standard size), as seen underwater! How complex do you think the program will be? Well, let me give you the source...

```
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include <dos.h>

int sin_x_table[640];
int sin_y_table[400];

void vid_mode(unsigned char mode){
        union REGS regs;
        regs.h.ah = 0;
        regs.h.al = mode;
        int86(0x10,&regs,&regs);
}

void make_chess(unsigned char far* where){
        int x,y,tx,ty;
        static int xcounter = 0;
        static int ycounter = 0;
        xcounter = xcounter > 311 ? 0:xcounter+8;
        ycounter = ycounter > 195 ? 0:ycounter+4;
        for(y=0;y<200;y++)
                for(x=0;x<320;x++){
                        tx = x + sin_y_table[y + ycounter]+40;
                        ty = y - sin_x_table[x + xcounter]+40;
                        if((tx % 40 >= 20) ^ (ty % 40 >= 20))
                                where[y*320+x] = 0;
                        else    where[y*320+x] = 15;
                }
}

int main(){
        int i;
        for(i=0;i<320;i++)
                sin_x_table[i] = (int)(sin(i * 6.28/320) * 10);
        for(;i<640;i++)
                sin_x_table[i] = sin_x_table[i-320];
        for(i=0;i<200;i++)
                sin_y_table[i] = (int)(sin(i * 6.28/200) * 10);
        for(;i<400;i++)
                sin_y_table[i] = sin_y_table[i-200];
        vid_mode(0x13);
        while(!kbhit())
                make_chess((unsigned char far*)0xA0000000);
        vid_mode(0x03);
        return 0;
}
```

That's it! If you compile and run this program (under Borland C++ v3.1 DOS) (easily portable anywhere, as described in previous sections) you'll see the effect. And it is very effective, considering that this program is so small.

As you see, most of this program is spent inside the `make_chess()` function. It is writing directly to video memory. But as you'll notice, the `make_chess()` function doesn't know it's writing to video memory. We can actually replace that pointer buy something else, and have it do the effect to some buffer!

Another important thing you should notice in the source is the use of `sin_x_table[]` and the `sin_y_table[]`. Most of the time, when we deal with `sin()` or `cos()` in our code, we'd like to avoid these functions (they're VERY slow!). There are many approaches to avoiding them, and the most common one is to create these *"look-up"* tables of these values. Sometimes, it is worth the effort, but sometimes, considering CPU cache speeds and FPU `sin` and `cos` speeds, it's not worth it. In this example it's definitely worth it, it even works nice under a 486 (slow FPU processor). Integer math is fast on any processor, so, try to use that most of the time, and only use floating point (FPU) when you know you can make it work in parallel with the CPU, thus, getting the FPU calculations for free. (but this subject is best left to an optimization tutorial)

I guess that's it for this example. I greatly earge you to compile and run it. A good exercise would be to make this program do the same exact effect only using some picture (not a chess board). (In fact, we'll probably end up doing it sometime in this tutorial...)