```
#include <avr/interrupt.h>
#include <string.h>
#define AXIS_X 1
#define AXIS_Y 2
#define AXIS_Z 3

volatile unsigned char cube[8][8];
volatile int current_layer = 0;

void setup()
{
  int i;

  for(i=0; i<14; i++)
    pinMode(i, OUTPUT);

  // pinMode(A0, OUTPUT) as specified in the arduino reference didn't work. So I accessed the registers
directly.
  DDRC = 0xff;
  PORTC = 0x00;

  // Reset any PWM configuration that the arduino may have set up automagically!
  TCCR2A = 0x00;
  TCCR2B = 0x00;

  TCCR2A |= (0x01 << WGM21); // CTC mode. clear counter on TCNT2 == OCR2A
  OCR2A = 10; // Interrupt every 25600th cpu cycle (256*100)
  TCNT2 = 0x00; // start counting at 0
  TCCR2B |= (0x01 << CS22) | (0x01 << CS21); // Start the clock with a 256 prescaler

  TIMSK2 |= (0x01 << OCIE2A);
}

ISR (TIMER2_COMPA_vect)
{
  int i;

  // all layer selects off
  PORTC = 0x00;
  PORTB &= 0x0f;

  PORTB |= 0x08; // output enable off.

  for (i=0; i<8; i++)
  {
    PORTD = cube[current_layer][i];
    PORTB = (PORTB & 0xF8) | (0x07 & (i+1));
  }

  PORTB &= 0b00110111; // Output enable on.

  if (current_layer < 6)
```

```
  {
    PORTC = (0x01 << current_layer);
  } else if (current_layer == 6)
  {
    digitalWrite(12, HIGH);
  } else
  {
    digitalWrite(13, HIGH);
  }

  current_layer++;

  if (current_layer == 8)
    current_layer = 0;
}

void loop()
{
  int i,x,y,z;

  while (true)
  {

    effect_planboing(AXIS_Z, 400);
    effect_planboing(AXIS_Y, 400);
    effect_planboing(AXIS_X, 400);

    effect_blinky2();

    effect_random_filler(75,1);
    effect_random_filler(75,0);

    effect_rain(100);

    effect_boxside_randsend_parallel (AXIS_X, 0, 150, 1);
    effect_boxside_randsend_parallel (AXIS_X, 1, 150, 1);
    effect_boxside_randsend_parallel (AXIS_Y, 0, 150, 1);
    effect_boxside_randsend_parallel (AXIS_Y, 1, 150, 1);
    effect_boxside_randsend_parallel (AXIS_Z, 0, 150, 1);
    effect_boxside_randsend_parallel (AXIS_Z, 1, 150, 1);

  }
}


//
================================================================================
===========
//   Effect functions
//
================================================================================
===========
```

```
void draw_positions_axis (char axis, unsigned char positions[64], int invert)
{
	int x, y, p;

	fill(0x00);

	for (x=0; x<8; x++)
	{
		for (y=0; y<8; y++)
		{
			if (invert)
			{
				p = (7-positions[(x*8)+y]);
			} else
			{
				p = positions[(x*8)+y];
			}

			if (axis == AXIS_Z)
				setvoxel(x,y,p);

			if (axis == AXIS_Y)
				setvoxel(x,p,y);

			if (axis == AXIS_X)
				setvoxel(p,y,x);
		}
	}

}


void effect_boxside_randsend_parallel (char axis, int origin, int delay, int mode)
{
	int i;
	int done;
	unsigned char cubepos[64];
	unsigned char pos[64];
	int notdone = 1;
	int notdone2 = 1;
	int sent = 0;

	for (i=0;i<64;i++)
	{
		pos[i] = 0;
	}

	while (notdone)
	{
		if (mode == 1)
		{
```

```
		notdone2 = 1;
		while (notdone2 && sent<64)
		{
			i = rand()%64;
			if (pos[i] == 0)
			{
				sent++;
				pos[i] += 1;
				notdone2 = 0;
			}
		}
	} else if (mode == 2)
	{
		if (sent<64)
		{
			pos[sent] += 1;
			sent++;
		}
	}

	done = 0;
	for (i=0;i<64;i++)
	{
		if (pos[i] > 0 && pos[i] <7)
		{
			pos[i] += 1;
		}

		if (pos[i] == 7)
			done++;
	}

	if (done == 64)
		notdone = 0;

	for (i=0;i<64;i++)
	{
		if (origin == 0)
		{
			cubepos[i] = pos[i];
		} else
		{
			cubepos[i] = (7-pos[i]);
		}
	}


	delay_ms(delay);
	draw_positions_axis(axis,cubepos,0);

}
```

```
}


void effect_rain (int iterations)
{
	int i, ii;
	int rnd_x;
	int rnd_y;
	int rnd_num;

	for (ii=0;ii<iterations;ii++)
	{
		rnd_num = rand()%4;

		for (i=0; i < rnd_num;i++)
		{
			rnd_x = rand()%8;
			rnd_y = rand()%8;
			setvoxel(rnd_x,rnd_y,7);
		}

		delay_ms(1000);
		shift(AXIS_Z,-1);
	}
}

// Set or clear exactly 512 voxels in a random order.
void effect_random_filler (int delay, int state)
{
	int x,y,z;
	int loop = 0;


	if (state == 1)
	{
		fill(0x00);
	} else
	{
		fill(0xff);
	}

	while (loop<511)
	{
		x = rand()%8;
		y = rand()%8;
		z = rand()%8;

		if ((state == 0 && getvoxel(x,y,z) == 0x01) || (state == 1 && getvoxel(x,y,z) == 0x00))
		{
			altervoxel(x,y,z,state);
			delay_ms(delay);
			loop++;
```

```
                }
        }
}


void effect_blinky2()
{
        int i,r;
        fill(0x00);

        for (r=0;r<2;r++)
        {
                i = 750;
                while (i>0)
                {
                        fill(0x00);
                        delay_ms(i);

                        fill(0xff);
                        delay_ms(100);

                        i = i - (15+(1000/(i/10)));
                }

                delay_ms(1000);

                i = 750;
                while (i>0)
                {
                        fill(0x00);
                        delay_ms(751-i);

                        fill(0xff);
                        delay_ms(100);

                        i = i - (15+(1000/(i/10)));
                }
        }

}

// Draw a plane on one axis and send it back and forth once.
void effect_planboing (int plane, int speed)
{
        int i;
        for (i=0;i<8;i++)
        {
                fill(0x00);
      setplane(plane, i);
                delay_ms(speed);
        }
```

```
        for (i=7;i>=0;i--)
        {
                fill(0x00);
      setplane(plane,i);
                delay_ms(speed);
        }
}

//
=============================================================================
===========
//   Draw functions
//
=============================================================================
===========


// Set a single voxel to ON
void setvoxel(int x, int y, int z)
{
        if (inrange(x,y,z))
                cube[z][y] |= (1 << x);
}


// Set a single voxel to ON
void clrvoxel(int x, int y, int z)
{
        if (inrange(x,y,z))
                cube[z][y] &= ~(1 << x);
}



// This function validates that we are drawing inside the cube.
unsigned char inrange(int x, int y, int z)
{
        if (x >= 0 && x < 8 && y >= 0 && y < 8 && z >= 0 && z < 8)
        {
                return 0x01;
        } else
        {
                // One of the coordinates was outside the cube.
                return 0x00;
        }
}

// Get the current status of a voxel
unsigned char getvoxel(int x, int y, int z)
{
        if (inrange(x,y,z))
        {
```

```
                if (cube[z][y] & (1 << x))
                {
                        return 0x01;
                } else
                {
                        return 0x00;
                }
        } else
        {
                return 0x00;
        }
}

// In some effect we want to just take bool and write it to a voxel
// this function calls the apropriate voxel manipulation function.
void altervoxel(int x, int y, int z, int state)
{
        if (state == 1)
        {
                setvoxel(x,y,z);
        } else
        {
                clrvoxel(x,y,z);
        }
}

// Flip the state of a voxel.
// If the voxel is 1, its turned into a 0, and vice versa.
void flpvoxel(int x, int y, int z)
{
        if (inrange(x, y, z))
                cube[z][y] ^= (1 << x);
}

// Makes sure x1 is alwas smaller than x2
// This is usefull for functions that uses for loops,
// to avoid infinite loops
void argorder(int ix1, int ix2, int *ox1, int *ox2)
{
        if (ix1>ix2)
        {
                int tmp;
                tmp = ix1;
                ix1= ix2;
                ix2 = tmp;
        }
        *ox1 = ix1;
        *ox2 = ix2;
}

// Sets all voxels along a X/Y plane at a given point
// on axis Z
```

```
void setplane_z (int z)
{
	int i;
	if (z>=0 && z<8)
	{
		for (i=0;i<8;i++)
			cube[z][i] = 0xff;
	}
}

// Clears voxels in the same manner as above
void clrplane_z (int z)
{
	int i;
	if (z>=0 && z<8)
	{
		for (i=0;i<8;i++)
			cube[z][i] = 0x00;
	}
}

void setplane_x (int x)
{
	int z;
	int y;
	if (x>=0 && x<8)
	{
		for (z=0;z<8;z++)
		{
			for (y=0;y<8;y++)
			{
				cube[z][y] |= (1 << x);
			}
		}
	}
}

void clrplane_x (int x)
{
	int z;
	int y;
	if (x>=0 && x<8)
	{
		for (z=0;z<8;z++)
		{
			for (y=0;y<8;y++)
			{
				cube[z][y] &= ~(1 << x);
			}
		}
	}
}
```

```
void setplane_y (int y)
{
	int z;
	if (y>=0 && y<8)
	{
		for (z=0;z<8;z++)
			cube[z][y] = 0xff;
	}
}

void clrplane_y (int y)
{
	int z;
	if (y>=0 && y<8)
	{
		for (z=0;z<8;z++)
			cube[z][y] = 0x00;
	}
}

void setplane (char axis, unsigned char i)
{
   switch (axis)
   {
      case AXIS_X:
         setplane_x(i);
         break;

      case AXIS_Y:
         setplane_y(i);
         break;

      case AXIS_Z:
         setplane_z(i);
         break;
   }
}

void clrplane (char axis, unsigned char i)
{
   switch (axis)
   {
      case AXIS_X:
         clrplane_x(i);
         break;

      case AXIS_Y:
         clrplane_y(i);
         break;

      case AXIS_Z:
```

```
        clrplane_z(i);
        break;
    }
}

// Fill a value into all 64 byts of the cube buffer
// Mostly used for clearing. fill(0x00)
// or setting all on. fill(0xff)
void fill (unsigned char pattern)
{
        int z;
        int y;
        for (z=0;z<8;z++)
        {
                for (y=0;y<8;y++)
                {
                        cube[z][y] = pattern;
                }
        }
}



// Draw a box with all walls drawn and all voxels inside set
void box_filled(int x1, int y1, int z1, int x2, int y2, int z2)
{
        int iy;
        int iz;

        argorder(x1, x2, &x1, &x2);
        argorder(y1, y2, &y1, &y2);
        argorder(z1, z2, &z1, &z2);

        for (iz=z1;iz<=z2;iz++)
        {
                for (iy=y1;iy<=y2;iy++)
                {
                        cube[iz][iy] |= byteline(x1,x2);
                }
        }

}

// Darw a hollow box with side walls.
void box_walls(int x1, int y1, int z1, int x2, int y2, int z2)
{
        int iy;
        int iz;

        argorder(x1, x2, &x1, &x2);
        argorder(y1, y2, &y1, &y2);
        argorder(z1, z2, &z1, &z2);
```

```
	for (iz=z1;iz<=z2;iz++)
	{
		for (iy=y1;iy<=y2;iy++)
		{
			if (iy == y1 || iy == y2 || iz == z1 || iz == z2)
			{
				cube[iz][iy] = byteline(x1,x2);
			} else
			{
				cube[iz][iy] |= ((0x01 << x1) | (0x01 << x2));
			}
		}
	}

}

// Draw a wireframe box. This only draws the corners and edges,
// no walls.
void box_wireframe(int x1, int y1, int z1, int x2, int y2, int z2)
{
	int iy;
	int iz;

	argorder(x1, x2, &x1, &x2);
	argorder(y1, y2, &y1, &y2);
	argorder(z1, z2, &z1, &z2);

	// Lines along X axis
	cube[z1][y1] = byteline(x1,x2);
	cube[z1][y2] = byteline(x1,x2);
	cube[z2][y1] = byteline(x1,x2);
	cube[z2][y2] = byteline(x1,x2);

	// Lines along Y axis
	for (iy=y1;iy<=y2;iy++)
	{
		setvoxel(x1,iy,z1);
		setvoxel(x1,iy,z2);
		setvoxel(x2,iy,z1);
		setvoxel(x2,iy,z2);
	}

	// Lines along Z axis
	for (iz=z1;iz<=z2;iz++)
	{
		setvoxel(x1,y1,iz);
		setvoxel(x1,y2,iz);
		setvoxel(x2,y1,iz);
		setvoxel(x2,y2,iz);
	}
```

```
}

// Returns a byte with a row of 1's drawn in it.
// byteline(2,5) gives 0b00111100
char byteline (int start, int end)
{
	return ((0xff<<start) & ~(0xff<<(end+1)));
}

// Flips a byte 180 degrees.
// MSB becomes LSB, LSB becomes MSB.
char flipbyte (char byte)
{
	char flop = 0x00;

	flop = (flop & 0b11111110) | (0b00000001 & (byte >> 7));
	flop = (flop & 0b11111101) | (0b00000010 & (byte >> 5));
	flop = (flop & 0b11111011) | (0b00000100 & (byte >> 3));
	flop = (flop & 0b11110111) | (0b00001000 & (byte >> 1));
	flop = (flop & 0b11101111) | (0b00010000 & (byte << 1));
	flop = (flop & 0b11011111) | (0b00100000 & (byte << 3));
	flop = (flop & 0b10111111) | (0b01000000 & (byte << 5));
	flop = (flop & 0b01111111) | (0b10000000 & (byte << 7));
	return flop;
}

// Draw a line between any coordinates in 3d space.
// Uses integer values for input, so dont expect smooth animations.
void line(int x1, int y1, int z1, int x2, int y2, int z2)
{
	float xy;		// how many voxels do we move on the y axis for each step on the x axis
	float xz;		// how many voxels do we move on the y axis for each step on the x axis
	unsigned char x,y,z;
	unsigned char lasty,lastz;

	// We always want to draw the line from x=0 to x=7.
	// If x1 is bigget than x2, we need to flip all the values.
	if (x1>x2)
	{
		int tmp;
		tmp = x2; x2 = x1; x1 = tmp;
		tmp = y2; y2 = y1; y1 = tmp;
		tmp = z2; z2 = z1; z1 = tmp;
	}


	if (y1>y2)
	{
		xy = (float)(y1-y2)/(float)(x2-x1);
		lasty = y2;
	} else
	{
```

```
                xy = (float)(y2-y1)/(float)(x2-x1);
                lasty = y1;
        }

        if (z1>z2)
        {
                xz = (float)(z1-z2)/(float)(x2-x1);
                lastz = z2;
        } else
        {
                xz = (float)(z2-z1)/(float)(x2-x1);
                lastz = z1;
        }



        // For each step of x, y increments by:
        for (x = x1; x<=x2;x++)
        {
                y = (xy*(x-x1))+y1;
                z = (xz*(x-x1))+z1;
                setvoxel(x,y,z);
        }

}

// Delay loop.
// This is not calibrated to milliseconds,
// but we had allready made to many effects using this
// calibration when we figured it might be a good idea
// to calibrate it.
void delay_ms(uint16_t x)
{
  uint8_t y, z;
  for ( ; x > 0 ; x--){
    for ( y = 0 ; y < 90 ; y++){
      for ( z = 0 ; z < 6 ; z++){
        asm volatile ("nop");
      }
    }
  }
}



// Shift the entire contents of the cube along an axis
// This is great for effects where you want to draw something
// on one side of the cube and have it flow towards the other
// side. Like rain flowing down the Z axiz.
void shift (char axis, int direction)
{
        int i, x ,y;
```

```
int ii, iii;
int state;

for (i = 0; i < 8; i++)
{
	if (direction == -1)
	{
		ii = i;
	} else
	{
		ii = (7-i);
	}


	for (x = 0; x < 8; x++)
	{
		for (y = 0; y < 8; y++)
		{
			if (direction == -1)
			{
				iii = ii+1;
			} else
			{
				iii = ii-1;
			}

			if (axis == AXIS_Z)
			{
				state = getvoxel(x,y,iii);
				altervoxel(x,y,ii,state);
			}

			if (axis == AXIS_Y)
			{
				state = getvoxel(x,iii,y);
				altervoxel(x,ii,y,state);
			}

			if (axis == AXIS_X)
			{
				state = getvoxel(iii,y,x);
				altervoxel(ii,y,x,state);
			}
		}
	}
}

if (direction == -1)
{
	i = 7;
} else
{
```

```
                i = 0;
        }

        for (x = 0; x < 8; x++)
        {
                for (y = 0; y < 8; y++)
                {
                        if (axis == AXIS_Z)
                                clrvoxel(x,y,i);

                        if (axis == AXIS_Y)
                                clrvoxel(x,i,y);

                        if (axis == AXIS_X)
                                clrvoxel(i,y,x);
                }
        }
}
```