High Powered Computer Graphics in C (Space Harrier)

Dale Wick

AdamCon 21, June 27, 2009.

Grand Rapids Michigan

- different forms of graphics display loops
- interrupt based graphics versus main loop graphics
- displaying graphics at different update frequencies.

Outline

- Graphics in Space Harrier arcade game
- Sprite conversion for hero of Space Harrier
- Adding perspective to the land
- Clouds and mountains add parallax
- Simulating 3D dragons with character based graphics
- Adding Explosions

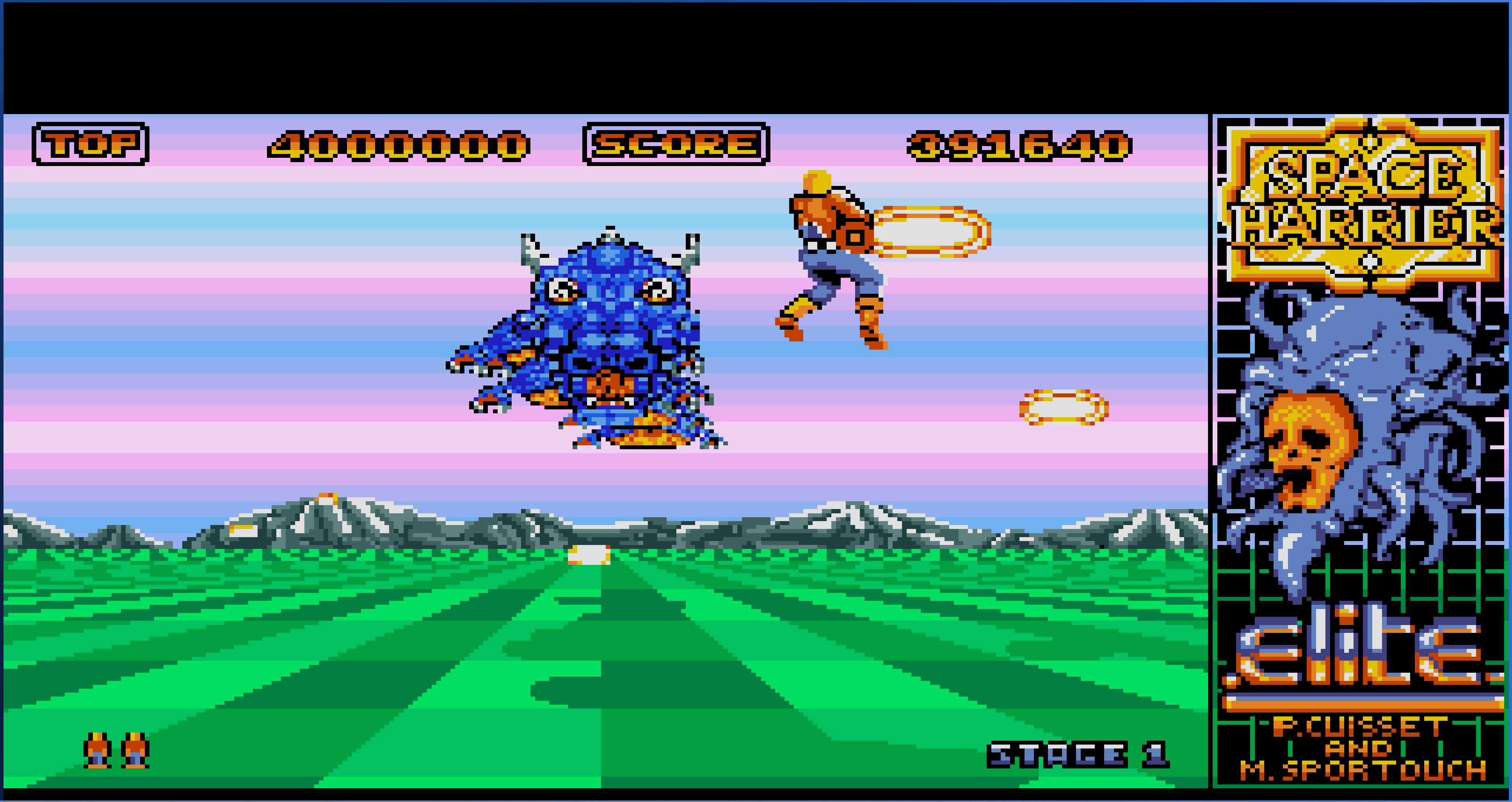
Outline

- Graphics in Space Harrier arcade game
- Sprite conversion for hero of Space Harrier
- Adding perspective to the land
- Clouds and mountains add parallax
- Simulating 3D dragons with character based graphics
- Adding Explosions

Graphics in Space Harrier

- Motivation
  - Daniel made the theme song, title screen and some sound samples.
- Arcade Game
  - The arcade unit has double buffered graphics, powered by two 68000s plus a Z80 which is used for sounds.

Screenshot from Arcade Game



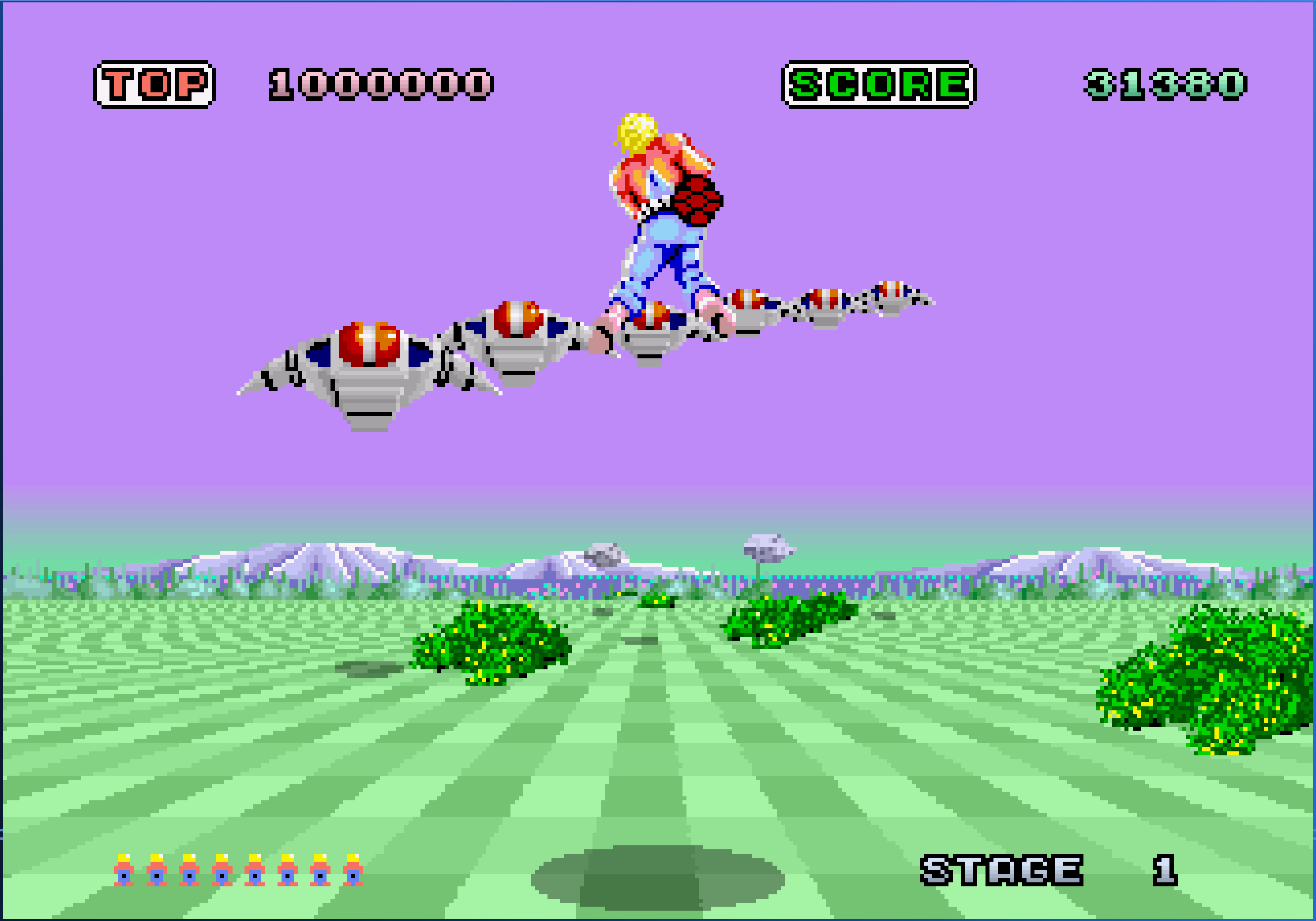
Outline

- Graphics in Space Harrier arcade game
- Sprite conversion for hero of Space Harrier
- Adding perspective to the land
- Clouds and mountains add parallax
- Simulating 3D dragons with character based graphics
- Adding Explosions

Sprite Conversion of hero

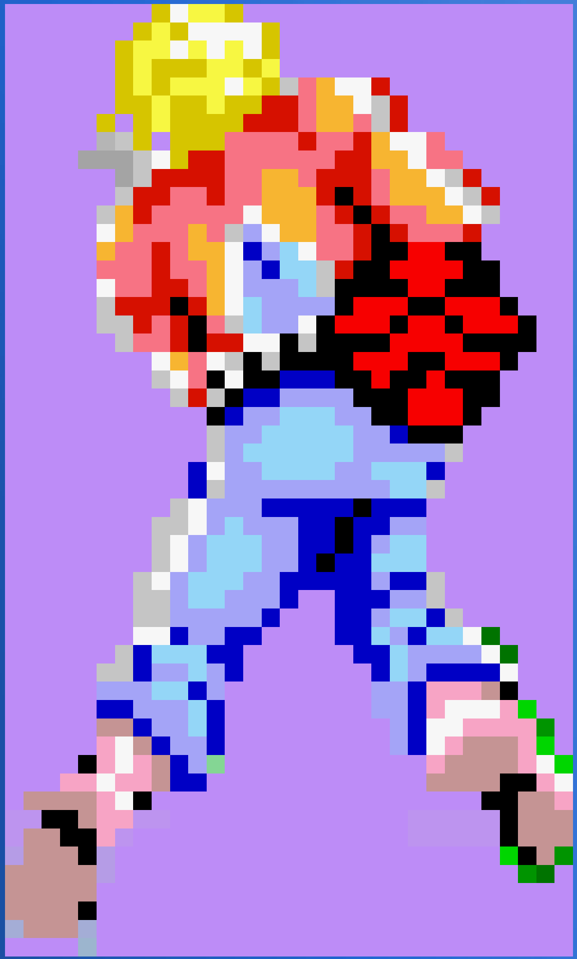
- Sprite Mode on the Adam/ColecoVision video chip (the TMS9918 video display processor)
  - the graphics mode allows up to 32 active sprites that are either 8x8 or 16x16 pixels (or dots).
  - The sprites are overlayed over the regular graphics.
  - Each sprite has one of the 15 colours, and is transparent everywhere else.
  - There is a restriction that the TMS9918 displays at most 4 sprites on a line.

Extract sprite from screen shot



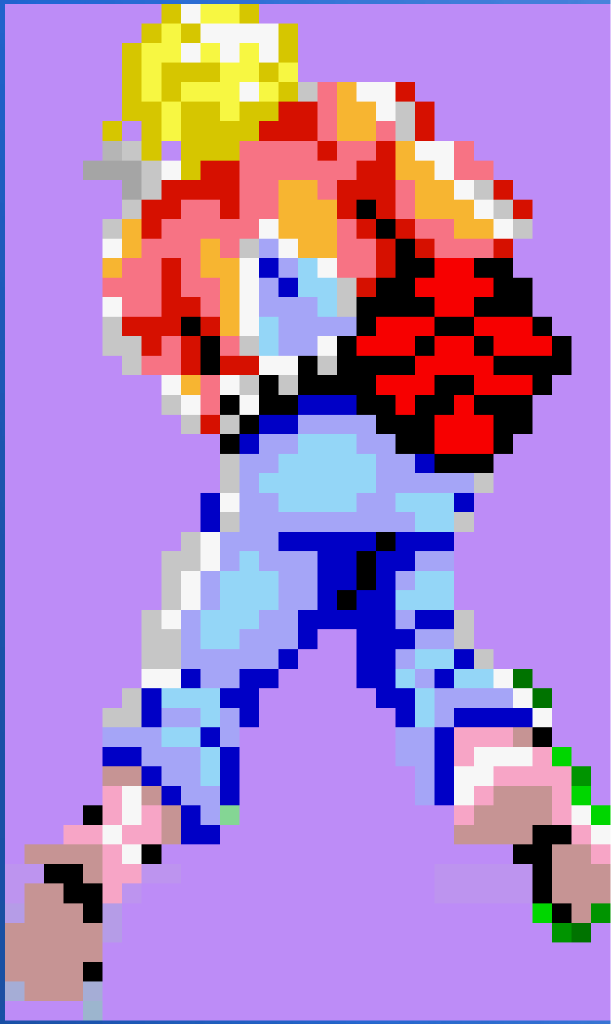
Cut out the sprite

- Remove just the sprite and erase the background
- Scale it so that it'll fit in two sprites wide, with two colours per 16x16 region
- This image is 31x56



Colour reduce sprite

- Using the GIMP trace the sprite shape with a new layer, which colour reduces it, with a TMS9918 palette.
- Export the result in XPM format for conversion to the sprite format.



Export in XPM format

- The XPM file is a graphics file written as C code. It is basically like ASCII art.

```
/* XPM */
static char * space_harrier_sprite_xpm[] = {
"31 52 9 1",
"          c None",
".         c #FFFFFF",
"+         c #D6C500",
"@         c #F7F742",
"#         c #F70000",
"$         c #000000",
"%         c #A4A4F7",
"&         c #94D6F7",
```

```
"........+@@@+..................",
".......+@+@@@@+................",
"......+@@@@@@@+................",
"......+@+++@@+@.#####..........",
"......+@+@@@@@+.#####..........",
"......++@++@++@#######.........",
"......@+@++++@@#######.........",
".......+@+++@@##########.......",
"........@+@@@############......",
"........##################.....",
"......############$########....",
".....##############$#######....",
".....###############$######....",
```

Read and make sprites

- Search for all images of one colour in the XPM
- Calculate the size of each solid color area
    - The colours used are: light blue, cyan, dark red, light red, gray, dark yellow, black, and brownish-red
- Areas are as follows:

Colour area sizes

- color 15 (gray) Sprite at 6,0 -> 14,8 (8 x 8) sprite count=1 (1 x 1)
- color 10 (light yellow) Sprite at 6,0 -> 14,8 (8 x 8) sprite count=1 (1 x 1)
- color 8 (bright red) Sprite at 5,3 -> 27,21 (22 x 18) sprite count=4 (2 x 2)
- color 1 (black) Sprite at 13,10 -> 28,23 (15 x 13) sprite count=1 (1 x 1)
- color 4 (dark blue) Sprite at 5,21 -> 27,42 (22 x 21) sprite count=4 (2 x 2)
- color 5 (light blue) Sprite at 8,22 -> 24,37 (16 x 15) sprite count=1 (1 x 1)
- color 6 (brownish red) Sprite at 0,38 -> 30,50 (30 x 12) sprite count=2 (2 x 1)

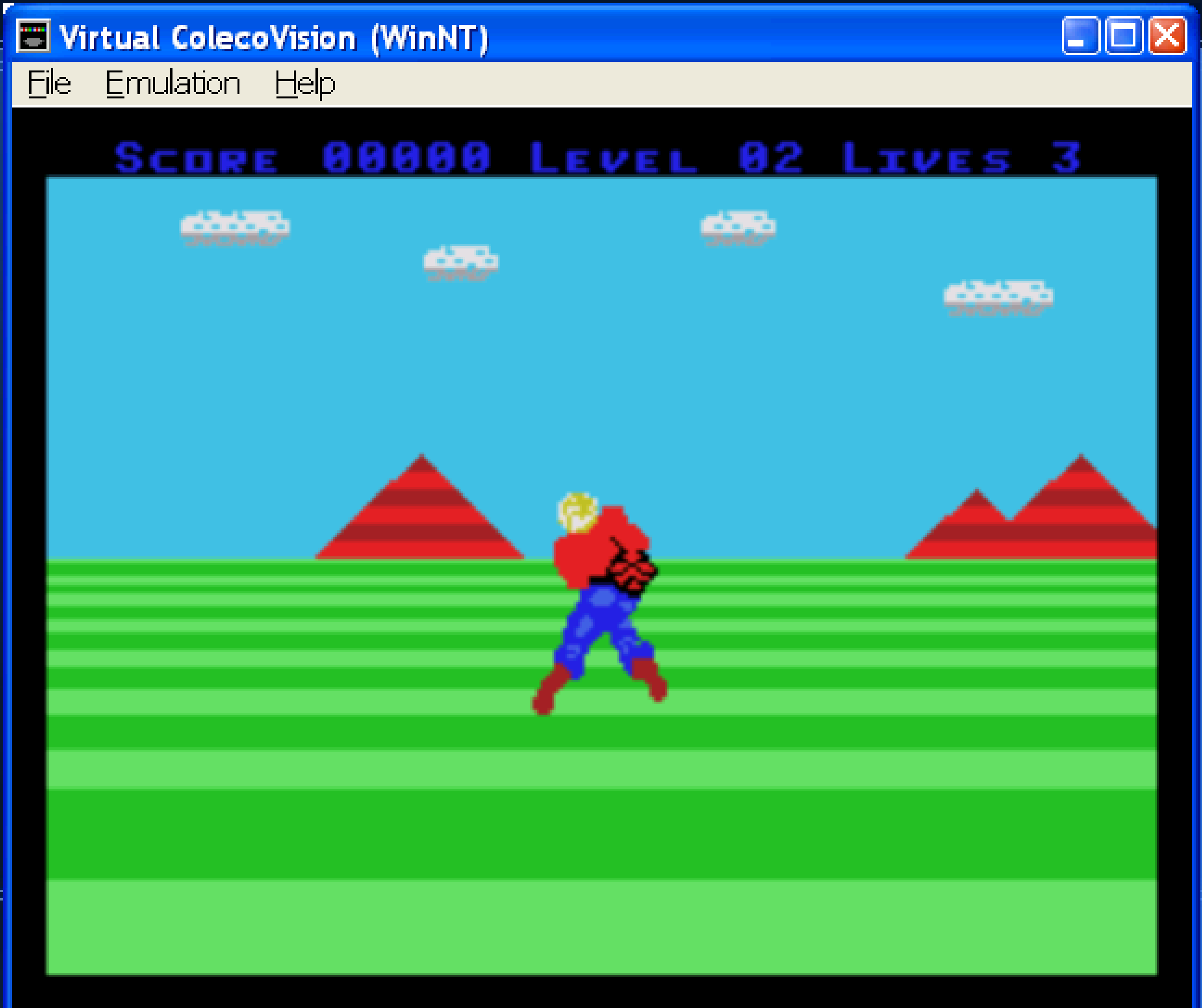
Sample of output as sprites

```
byte SPATT[]={
/* Output color 15 */
/* Sprite at 6,0 -> 14,8 (8 x 8) */
/* sprite count=1 (1 x 1) */
0x22,0x50,0x80,0xb9,0xa0,0xdb,0x5e,0x5c,0x10,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x80,0x80,0x00,0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
/* Output color 10 */
/* Sprite at 6,0 -> 14,8 (8 x 8) */
/* sprite count=1 (1 x 1) */
0x1c,0x2f,0x7f,0x46,0x5f,0x24,0xa1,0x23,0x2e,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x80,0x00,0x80,0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
```

Output as sprites continued

```
/* Output color 8 */

/* Sprite at 5,3 -> 27,21 (22 x 18) */

/* sprite count=4 (2 x 2) */

0x00,0x00,0x00,0x00,0x00,0x00,0x1f,0x7f,0xff,0xff,0xff,0xff,0xff,0xff,0x7f,0x7f,

0x1f,0x1f,0x3f,0x3f,0x7f,0xff,0xff,0xfb,0xfd,0xfe,0xfe,0xfc,0xf8,0xfb,0xf7,0xf0,

0x00,0x00,0x80,0x80,0xe0,0xf0,0xf8,0xfc,0xfc,0xfc,0x60,0xf0,0x60,0x9c,0x6e,0xf0,

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,

0x3f,0x1f,0x1f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,

0x03,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,

0x9c,0x20,0x70,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
```

Screenshot in Virtual Coleco



Update based on user input

- Updates at full speed (60 fps)
  - We set the video display processor (VDP) to give a Non-Maskable Interrupt (NMI) every 60th of a second
  - The NMI must complete (and tell the VDP it is done) before the next NMI is triggered
  - This is easy when just updating the sprite table
- Moves in 2 pixel increments
  - Makes it seem quite responsive

Outline

- Graphics in Space Harrier arcade game
- Sprite conversion for hero of Space Harrier
- Adding perspective to the land
- Clouds and mountains add parallax
- Simulating 3D dragons with character based graphics
- Adding Explosions

Moving ground and parallax

- The idea is to have the feeling of moving forward on a plane.
  - When the player moves up and down we will move the skyline up and down in 8 pixel units
  - When the player moves left and right, the clouds and mountains will move in 2 pixel units
  - The ground will update at 20 fps (every 3rd NMI)

Format of the ground

- TMS9918 (the VDP) mode
  - Create a column of 12 character tiles in the character set
  - Each character is an 8x8 tile with a pattern and colours.
  - The colours are either light green or dark green
  - The patterns loop after 8 frames
  - Height is up to 96 pixels or half the screen

Calculate one frame of the ground

```
bar=0;
distance=6;
for(y=95;y>=0;y--) {
    if(bar<128 || (bar>256 && bar<384)) ground[y]=0xff; else ground[y]=0;
    bar=bar+distance;
    if(bar>=256) {
        bar-=256;
        distance+=9;
    }
}
```

How does it look?

distance	bar	increment	solid	line
33	9	9	255	26
33	42	9	255	25
33	75	9	255	24
33	108	9	255	23
33	141	9	0	22
33	174	9	0	21
33	207	9	0	20
33	240	9	0	19
42	17	9	255	18
42	59	9	255	17
42	101	9	255	16
42	143	9	0	15
42	185	9	0	14
42	227	9	0	13

Add in the offset

```
bar=0;
distance=6;
for(y=95;y>=0;y--) {
    if(bar<128-off*32 || (bar>256-off*32 && bar<384-off*32))
                ground[y]=0xff;
        else
                ground[y]=0;
    bar=bar+distance;
    if(bar>=256) {
        bar-=256;
        distance+=9;
    }
}
```

Play the animation

- Reprogram the 12 characters every 3rd NMI (at 20 fps)
- Switch between the 8 different scrolling frames
- Repeat the 12 characters all across the screen
  - See sharr3.rom

Outline

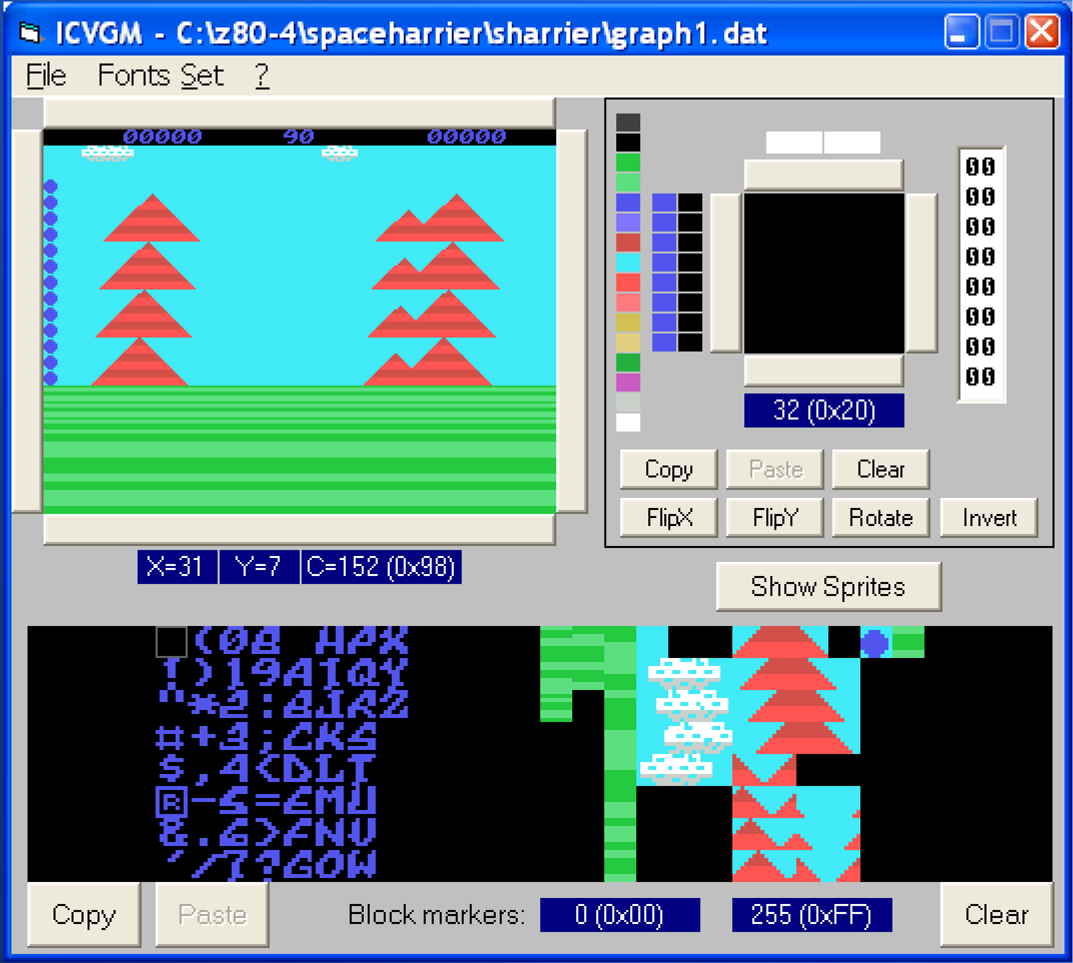
- Graphics in Space Harrier arcade game
- Sprite conversion for hero of Space Harrier
- Adding perspective to the land
- Clouds and mountains add parallax
- Simulating 3D dragons with character based graphics
- Adding Explosions

Clouds and mountains

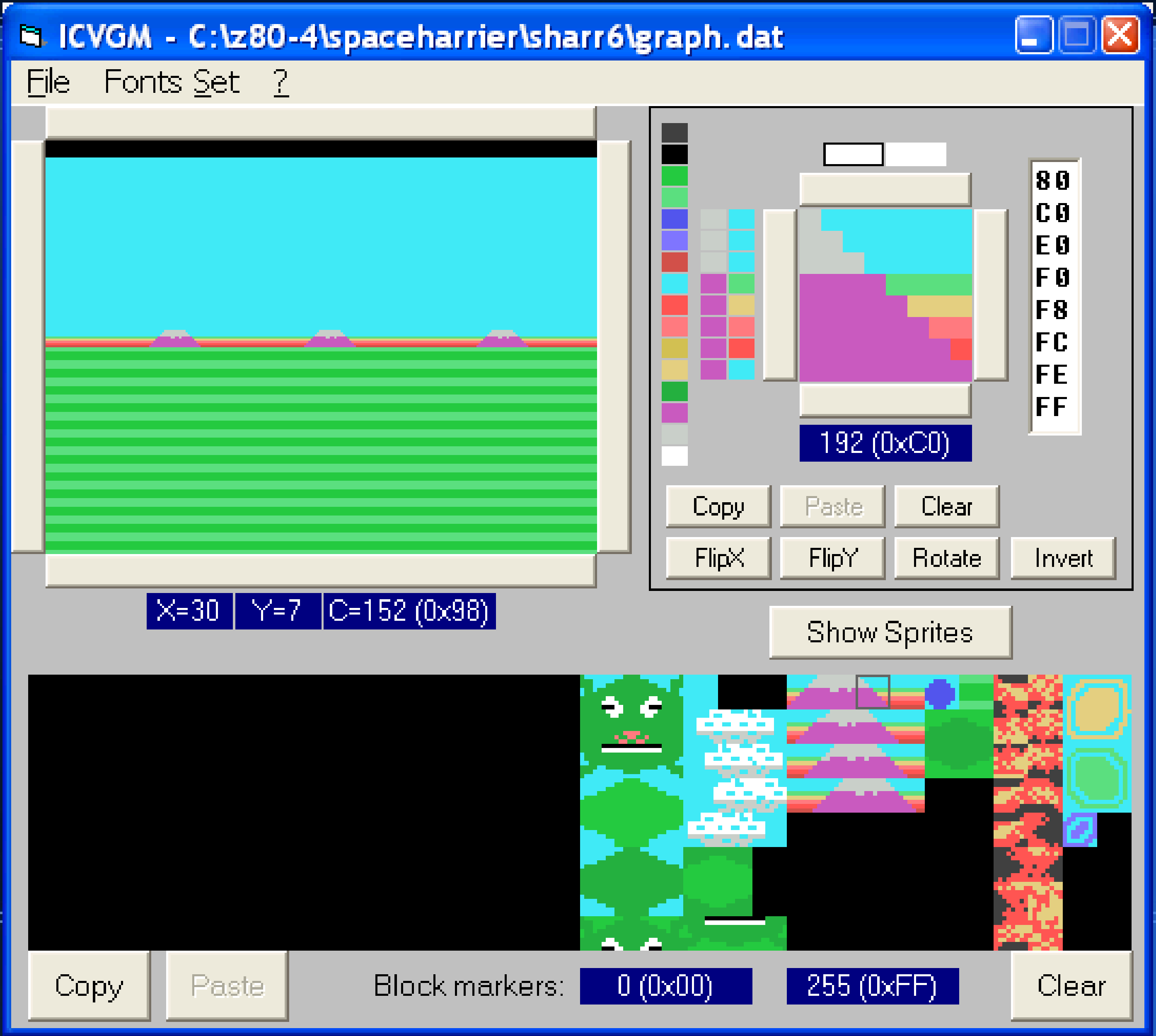
- Use the movement of the clouds and mountains to add a parallax effect to enhance the feeling of depth
- Move in 2 pixel increments, so we need to repeat the characters 4 times in different postions

ICVGM Character Map

- Character Set
  - Mountains are made of triangles with two red colours plus sky
  - Clouds use two colours plus the sky colour



Refinement of mountains



Outline

- Graphics in Space Harrier arcade game
- Sprite conversion for hero of Space Harrier
- Adding perspective to the land
- Clouds and mountains add parallax
- Simulating 3D dragons with character based graphics
- Adding Explosions

Simulating 3D dragon

- The dragon path is based on the formula:

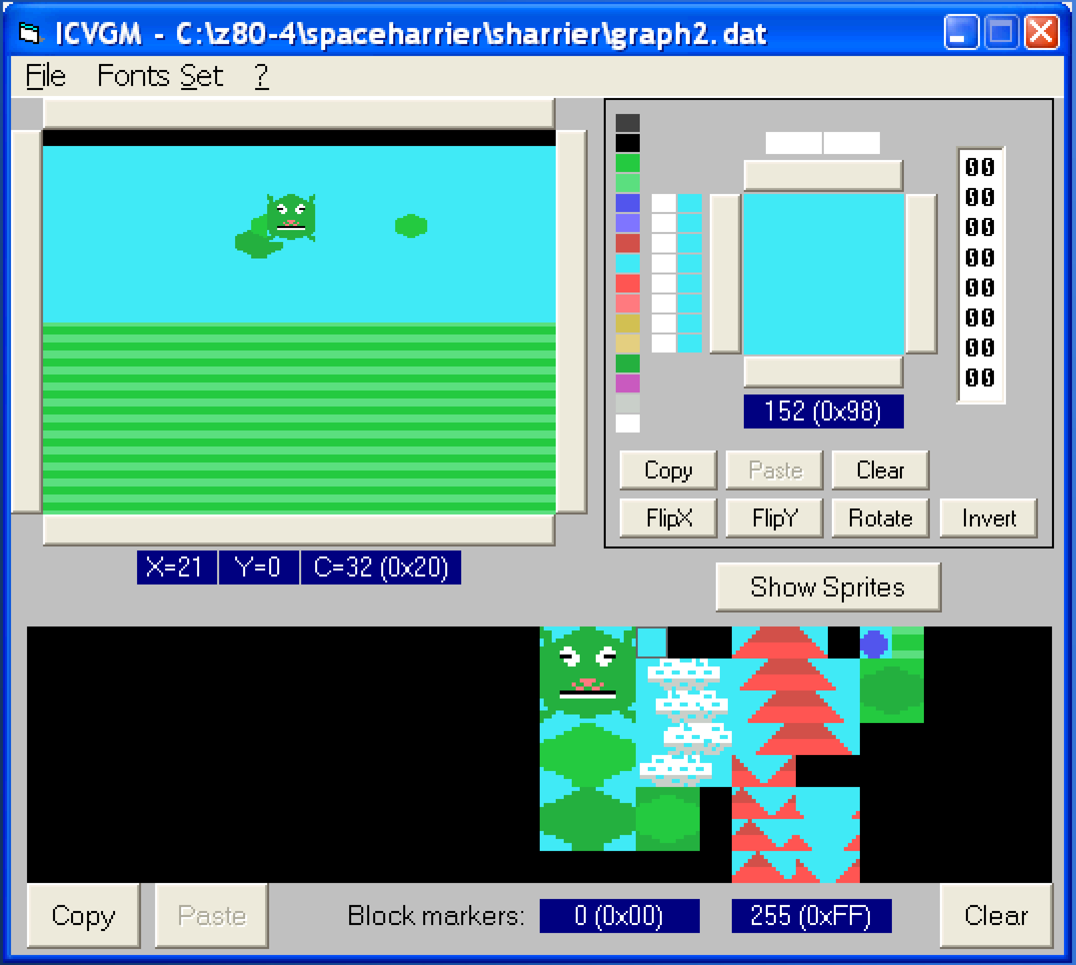
```
for(i=0;i<MAX;i++) {
    cz=cosf(i*M_PI/64.0f)*(i<128?128-i:i-128)*radius/128;
    x=cx+cosf(i*M_PI/16.0f)*radius;
    z=cz+sinf(i*M_PI/16.0f)*radius;
    y=cy;
    sx=(x-16)+16;
    sy=(y-12)+12+z/4;
    sz=z;
    printf("{%d,%d,%d}, /* %.4f,%.4f,%.4f */\n",sx,sy,sz,x,y,z);
}
```

Dragon with size cues

- The dragon is drawn into a 512 byte buffer in the ColecoVision's 1k of RAM
  - That excludes the bottom of the ground, and the clouds at the top (4 lines at the top, and 4 lines at the bottom)
  - There are 3 versions each of the head, dark body and light body. These version are 1x1, 2x2 and 3x2 character tiles.

The dragon character set

- When the layers are overlapping there are double green versions
- When they are over the sky there are versions with cyan sky



The Greenness grid

```
const unsigned char
    greenness[256]={
    /* 0 1 2 3 4 5 6 7 8 9 a b c d e f */
0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0, /* 0 */
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, /* 1 */
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, /* 2 */
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, /* 3 */
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, /* 4 */
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, /* 5 */
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, /* 6 */
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, /* 7 */
1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,0, /* 8 */
1,1,1,1,1,1,1,1,0,0,0,0,0,1,1,0, /* 9 */
0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0, /* a */
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, /* b */
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, /* c */
0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0, /* d */
0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0, /* e */
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, /* f */
};
```

Check the depth and draw

```
#define DRAGONFAR -7

#define DRAGONNEAR 3

void drawDragonBody(unsigned char *buff,char x,char y,char z,unsigned char
    color) {

   unsigned char *cell=buff+x+y*32;

   if(z<DRAGONFAR) {      // small

        cell[0]=0x8B+color*3;

   } else if(z<DRAGONNEAR) {      // medium
```

Check depth and draw medium

```
if(color) {
    cell[0]=greenness[cell[0]]?0x9d:0x83;
    cell[1]=greenness[cell[1]]?0xa5:0x93;
    cell[32]=greenness[cell[32]]?0x9e:0x84;
    cell[33]=greenness[cell[33]]?0xa6:0x94;
 } else {
     cell[0]=greenness[cell[0]]?0xd1:0x85;
     cell[1]=greenness[cell[1]]?0xd9:0x95;
     cell[32]=greenness[cell[32]]?0xd2:0x86;
     cell[33]=greenness[cell[33]]?0xda:0x96;
}
```

Check depth and draw far

```
} else {      // large
    if(color) {
        cell[-33]=greenness[cell[-33]]?0x9d:0x83;
        cell[-32]=0x8b;
        cell[-31]=greenness[cell[-31]]?0xa5:0x93;
        cell[-1]=0x8c;
        cell[0]=0x8c;
        cell[1]=0x8c;
        cell[31]=greenness[cell[31]]?0x9e:0x84;
```

And so on...

Drawing the overall body

```
// Panter's algorithm, back to front.
start=minz;  end=maxz;
for(j=start;j<=end;j++) {
    for(i=0;i<MAXDRAGONCELL;i++) {
        if(dragonCell[i].z!=j) continue;
        if(i!=0 && i!=MAXDRAGONCELL-1)
                drawDragonBody(buff,dragonCell[i].x,dragonCell[i].y-miny,
    dragonCell[i].z,i&1);
       else
                drawDragonHead(buff,dragonCell[i].x,dragonCell[i].y-
    miny,dragonCell[i].z);
    }
}
```

Different ways to draw the dragon

- Method 1: In the main loop draw the dragon to a 512 byte buffer, and write it all to the VRAM
- Method 2: In the NMI draw the dragon in the first NMI, write it to VRAM in the second NMI
- Method 3: draw the dragon in the main loop, then flag the NMI that it is ready. The NMI writes it in two 256 byte blocks over two NMIs and signals when it is done

Draw it all in the main loop

- With a slow main loop (10 to 30 fps) the sprite for the player moves at an uneven speed
- The ground display update also moves at an uneven speed, and is too sluggish or too jerky to look right

Break up drawing over several NMIs

- Sprite position updated at 60 fps, and feels very responsive
- The 96 bytes of ground movement update every 3rd NMI and also feel neither too slow nor too fast
- Sometimes the drawing of the dragon takes too long, so an NMI is missed, causing inconsistent response

Signaling architecture

- VDP Choice
  - You can't send information to the VDP from both the NMI function and the main line. It has to be one or the other.
  - The main line can do heavy calculations and will be interrupted by the NMI to update the display
- A variable will tell both who is updating the render buffer.

What does the NMI do?

- The NMI routine keeps things moving
  - Check the joystick and update the sprite position table
  - Update the music
  - Send the new sprite position table to the VDP
  - Every third NMI update the ground animation table
- When signaled it will update the render buffer

NMI screen buffer updating

- With all of the other things being updated, the full 768 byte name table can't be updated in one NMI
- The solution is to use double buffering, so that there are two name tables.
  - draw the top half of the screen in one NMI and the bottom half in the next one
  - Clouds are still updated more often for a parallax cue

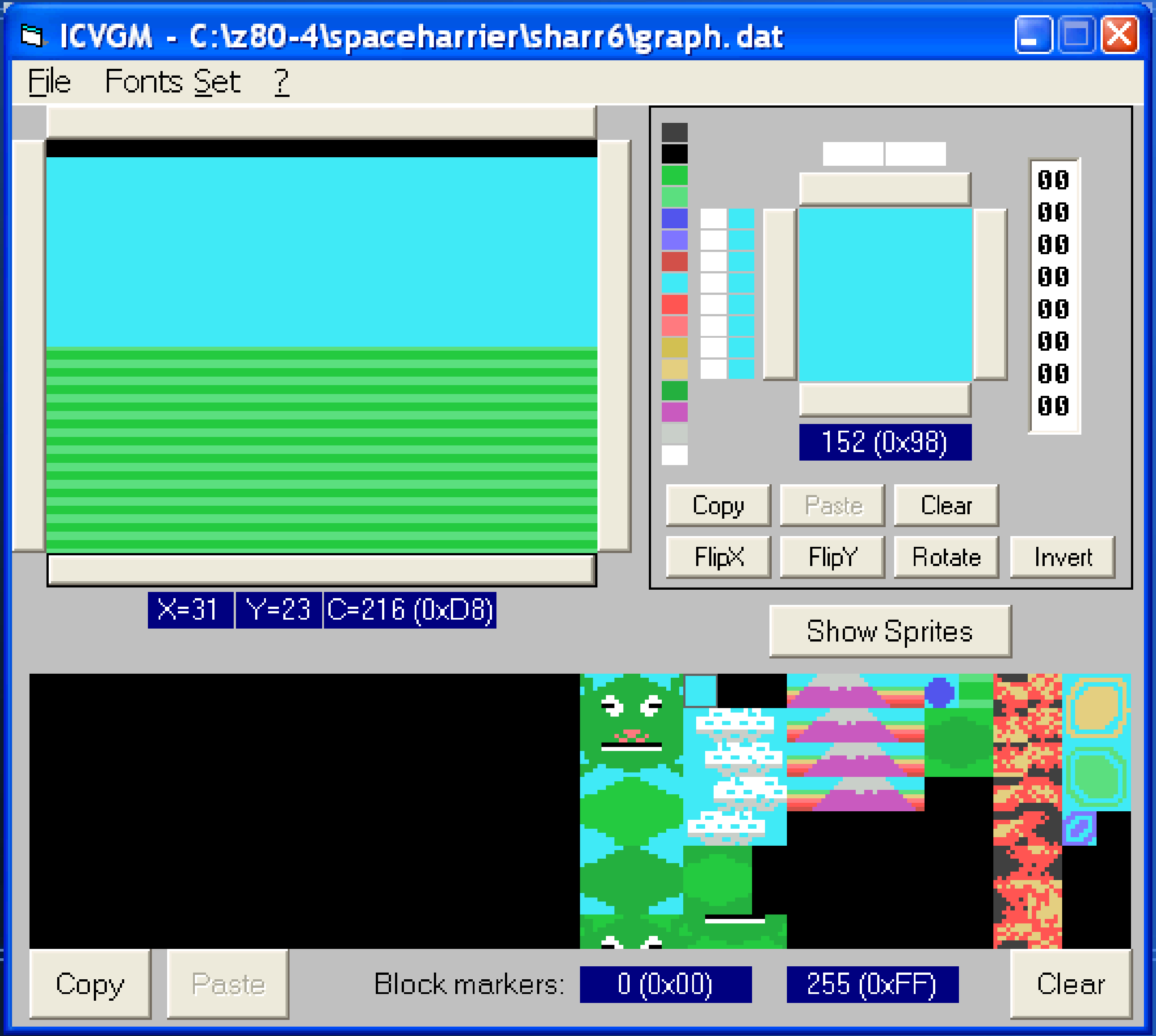
Outline

- Graphics in Space Harrier arcade game
- Sprite conversion for hero of Space Harrier
- Adding perspective to the land
- Clouds and mountains add parallax
- Simulating 3D dragons with character based graphics
- Adding Explosions

Fireballs are needed

- Visual Cue
  - When the Space Harrier fires, the missile glows
  - a fireball explodes when it hits the target
  - The fireball gradually grows in size, and is generally round in shape

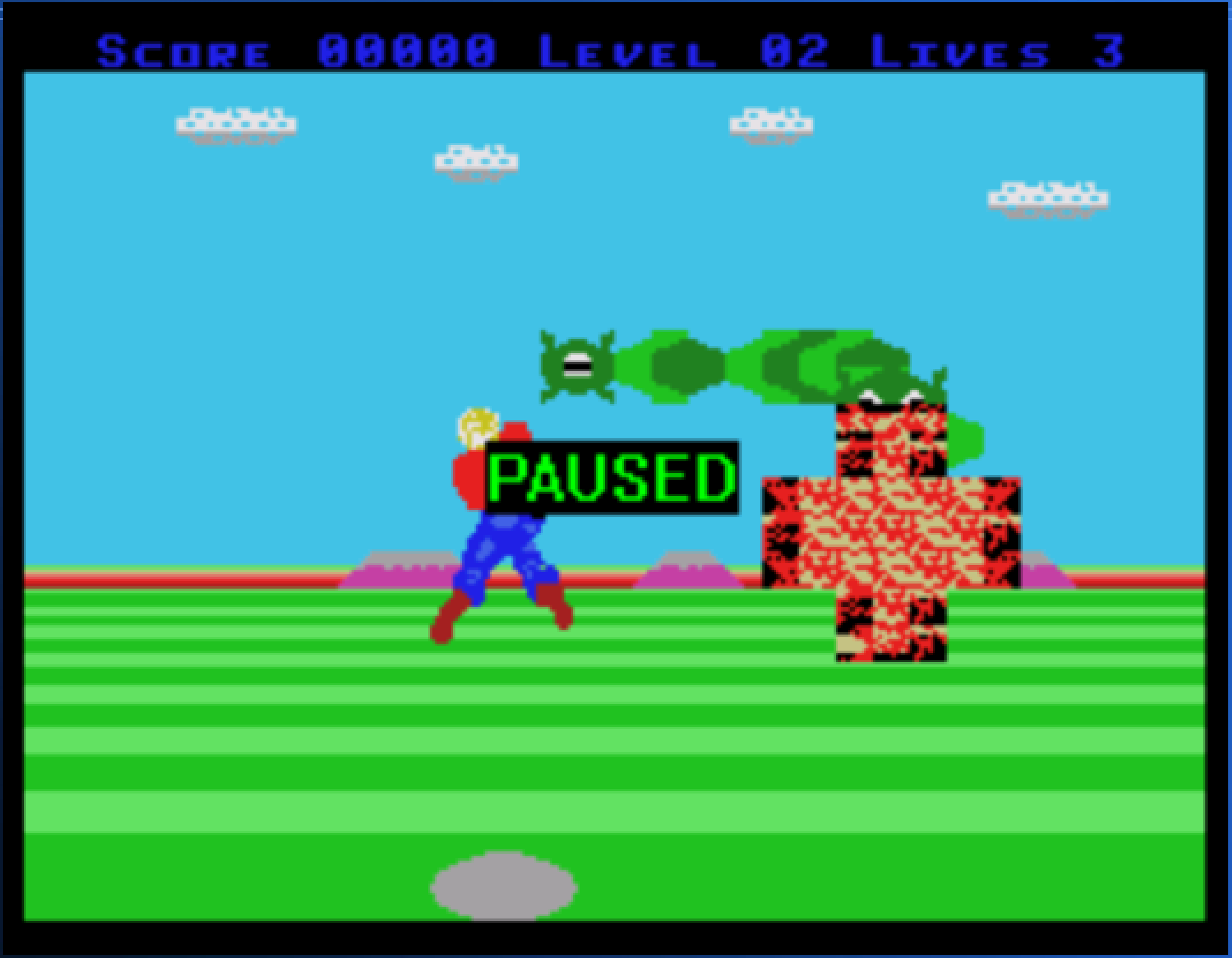
Screenshot of char set



Drawing a circle with chars

```
odd=((size+1)&1)+1;
for(j=0;j<size;j++) {
    char jj=j-size/2;
    if(size==2) odd=2; else if(j>size/2) odd-=2; else if(j>0) odd+=2;
    if(odd==1) width=3; else if(odd==size-2) width=size; else width=odd;
    if(y+jj<4 || y+jj>19) continue;// clip above and below
    for(i=0;i<width;i++) {
        int pos=jj*32+i-width/2;
        if(i==0 && j==0) buff[pos]=0xe0;
And so on
```

Screenshot of explosion



Animating the explosion

- Without animating, the slow update rate (5 to 10 fps) of the explosion didn't seem right
  - The pattern table for the explosion is 128 bytes long.
  - by repeating the first entries at the end, we can choose from 4 different pattern sets

Explosion pattern table

```
const unsigned char explosionPattern[0x98]={
0x80, 0xC0, 0x8F, 0xE1, 0xD0, 0xB8, 0x3F, 0x1F, 0x38, 0x1C, 0xF3, 0xE6, 0x73, 0x29,
  0x40, 0x10, 0x10, 0xBD, 0xF6, 0x78, 0xD0, 0x68, 0xFE, 0x01, 0x01, 0x03, 0xF1, 0x87, 0x0B, 0x1D,
  0xFC, 0xF8, 0x01, 0x03, 0xF1, 0x87, 0x0B, 0x1D, 0xFC, 0xF8, 0x0F, 0x17, 0x0F, 0x03, 0x01, 0x03,
  0x0F, 0x5F, 0xF0, 0xA9, 0x7F, 0x3F, 0x1D, 0x0F, 0x16, 0x3F, 0x1F, 0x3F, 0xE8, 0x90, 0xED, 0x9B,
  0xC0, 0x80, 0x32, 0x4C, 0x8B, 0x53, 0xE6, 0xF3, 0x1C, 0x38, 0x21, 0xD6, 0x63, 0xC4, 0x7B, 0xF6,
  0xBD, 0x10, 0x7F, 0x16, 0x43, 0x9E, 0x6F, 0xBD, 0x08, 0x00, 0xF8, 0xFC, 0x97, 0x43, 0xB5, 0xD9,
  0x03, 0x01, 0xF0, 0xE8, 0xF0, 0xC0, 0x80, 0xC0, 0xF0, 0xFA, 0x0F, 0x95, 0xFE, 0xFC, 0xB8, 0xF0,
  0x68, 0xFC, 0x18, 0x84, 0xCB, 0x79, 0x38, 0x28, 0x60, 0x40, 0x42, 0xEE, 0x64, 0x7F, 0x9E, 0x1B,
  0x01, 0xE7, 0x99, 0x80, 0xC0, 0x8F, 0xE1, 0xD0, 0xB8, 0x3F, 0x1F, 0x38, 0x1C, 0xF3, 0xE6, 0x73, 0x29,
  0x40, 0x10,
};
```

Alternate explosion

- Runs every 3rd NMI

```
char explosionFrame;
void alternateExplosion()
{
    unsigned char *patt=explosionPattern;
    explosionFrame=(explosionFrame+1)&3;
    patt+=((explosionFrame&1)<<3)+((explosionFrame&2)<<1);
    put_vram(8*224,patt,128);
}
```

Outline

- Graphics in Space Harrier arcade game
- Sprite conversion for hero of Space Harrier
- Adding perspective to the land
- Clouds and mountains add parallax
- Simulating 3D dragons with character based graphics
- Adding Explosions

Questions?

- Future directions
  - Add in trees and bushes
  - Add in flying tiki heads
  - Add in different colour schemes for new levels
  - Add in bonus level with the white dragon
- Questions?