Chaotix's 32X Graphics Drawing System

Discussion in 'Engineering & Reverse Engineering' started by Devon, Nov 8, 2023.

Nov 8, 2023 #1

Devon

I think I'm paranoid

1,158
1,296
93
your mom

I've dug a bit into Chaotix, and I have managed to document how a good chunk of its graphics rendering system on the 32X side works. Of course, some of this is already documented, but I think it'd still be nice to have it explained here.

Sprite Data

The main index table for the sprite sets is located at $A0000 in the ROM. Each sprite set starts with another index table for each sprite in the set. Blank entries have the pointer set to -1.

The sprite format is quite interesting, as it is designed to avoid storing unnecessary blank pixels. First, it starts off with this header that defines the sprite boundaries:

Code (Text):

```
1. .w Left boundary
2. .w Right boundary
3. .w Top boundary << 8
4. .w Bottom boundary << 8
```

The best way I can describe these values is with this example: let's say you wanted to have a 32x32px sprite with its pivot set in the middle. You'd set the left and top boundaries to -16, and the right and bottom boundaries to 16. Basically, a sprite's pivot point is always at (0, 0), and the pixels are centered around that.

After this header comes the actual pixel data. Chaotix stores pixels as individual rows. Each row starts off with its own header that

defines its positioning in the sprite:

Code (Text):

```
1. .b Left position
2. .b Right position
3. .b Y position
4. .b 0
```

The same rules for the positioning being centered around the pivot point at (0, 0) in the sprite header apply here as well. After that is an array of palette indices for each pixel in the row.

The end of the sprite data is found when the left and right values are the same.

So, that's the main sprite format. However, I'm not done talking about it, because sprites can actually be set to be compressed. The compression basically involves removing unneeded bits from the sprite data by subtracting a minimum value for each data value and determining the max number of bits that needs to be supported. Decompressing a compressed sprite will result in the regular sprite format.

A compressed sprite starts off with this header:

Code (Text):

```
1. .w Uncompressed size
2. .b $42
3. .b Left boundary
4. .b Bits per X position value
5. .b Top boundary
6. .b Bits per Y position value
7. .b Base pixel value
8. .b Bits per pixel value
9.
```

That "$42" is basically the magic number that defines a sprite as compressed. The left and top boundaries get defined here, alongside how many bits that an X position value, Y position value, and pixel value take up in the compressed data, and the base pixel value that gets added to each opaque pixel value. The left and top boundaries also act as the base values that get added to each X and Y position value.

The compressed data bitstream start off with the right and bottom

boundary values, and then the pixel rows. Decompressing is really just a matter of reading a number of bits, depending on the type of value, adding its associated base value, and then storing it in the decompression buffer.

When drawing sprites, they can be drawn anywhere on the screen, and can be scaled in either the X and Y axis. Unfortunately, this is no support for rotating sprites. This is why Chaotix still has separate sprites for angled character sprites, for example.

Sprite Lists

Chaotix stores many different lists that define what sprite sets to load. The format is simple: each entry contains 2 bytes: the sprite set ID, and its palette index offset. When it loads a sprite set, it'll read/decompress each of its frames into a buffer, and offset each pixel value, so you can basically choose which part of the palette that the sprite should use.

The end of a sprite list is found when the sprite set ID is set to -1.

When going into a stage, Chaotix will load 2 sprite lists: a global list and a stage specific list. The global sprite list is located at $A0200 in the ROM. The index table for stage specific sprite lists is located at $A0210 in the ROM, with each entry being a 16-bit relative offset.

Player Sprites

The player sprites are special. The way Chaotix handles this involves allocating the first 16 sprite sets as player sprites, with the first 8 being the main sprites and the other 8 being the limbs. The sprite data MUST be uncompressed, because with how many frames a character has, it really can't be pre-loaded. So, the game just loads in the currently displayed frame for a character as the graphics are rendered.

Polygons

Alongside sprites, Chaotix can render basic polygons. You can define different kinds of shapes to be drawn and have their vertices placed anywhere on the screen. You can also draw 3D polygons that can be placed and rotated in the context of a 3D space. For that, Chaotix uses a buffer that contains a table of polygons that can be picked from and drawn.

Boundaries

You can define a set of boundaries in which graphics will be visible in. For 3D polygons and scaled sprites, you can also define 3D boundaries in which they'll be visible in as well. By default, graphics will be visible on the entire screen, and 3D polygons and scaled sprite can be drawn at range 1-$7FFF. For 3D polygons, this applies to its Z position, and for scaled sprites,this applies to its scale values.

Palettes and Priority

On the 32X, graphics can either be drawn behind the Genesis graphics, or in front of them. The way Chaotix handles being able to determine a sprite's priority is to basically contain 2 copies of the palette in the 32X's CRAM. The first 128 colors contain the regular palette data, and the last 128 colors contain those same colors, but with their priority flag flipped. Chaotix will then choose which half of CRAM to use for drawing a sprite.

Palettes are actually managed by the 68000 sides for regular stages. When performing graphics updates, the FM bit is set to 0 so that palette updates can be made, and then it's set back to 1 so that the graphics rendering can begin.

Commands

To send a command to the 32X, you set the command ID in communication register 0 (word). While a command is being processed, it will remain at that ID, and once it's done, it will be cleared. Here's a list of the commands:

- 01 - Reset renderer
- 02 - Render graphics (clears frame buffer)
- 03 - Initialize stage graphics
- 04 - Initialize character palette index offsets for character selection menu
- 05 - Nothing
- 06 - Nothing
- 07 - Load a sprite set
- 08 - Handle special stage
- 09 - Render graphics (doesn't clear frame buffer)

I've not looked into the special stages much at all, so I won't be covering that command. The focus will be for rendering during regular stages and other screens, so I will cover the other commands.

Sending Data

Chaotix uses DREQ for data being sent from by the 68000 side. The way this is handled is that the Master SH-2's command interrupt is set to accept a DREQ request and copy data into a double buffered area. When the 68000 wants to send data, it simply sets up the DREQ parameters, sends a Master SH-2 command interrupt signal, and then sends the data over. If I recall, DREQ data must be sent in units of 4 words, due to a hardware thing.

Initialization

Generally, this is just a matter of running commands 01, then 03, and then running command 02 twice to make sure that things are all flushed properly. Running command 01 will reset the renderer and running command 03 will prepare for rendering stage graphics.

Command 01 requires no additional parameters or data to be sent, but command 03 does. It expects the following data to be sent via DREQ:

Code (Text):

```
1. .b Attraction ID
2. .b Level ID
3. .b Time zone ID
4. .b Player 1 character ID
5. .b Player2 character ID
```

The first 3 are used as indices for the stage specific sprite list index table, and the last 2 are used to define the character palette index offsets to $12 and $24 respectively.

Running this command will also reset the visible space, and set the near and far range to 4-$4000. You can find the parameters set at $6000FE4 in the SH-2 program ($787E4 in the ROM). The 3D polygons used for the title card also get defined. You can find that data at $6000FF0 in the SH-2 program ($787F0 in the ROM), where 12 4-sided polygons are defined. After that, the palette index offsets

are set, and the sprites are loaded.

Rendering

Now let's get into actually getting stuff drawn. The way this is done is that you send in a list of draw commands via DREQ, and then call either command 02 or 09. It should be noted that Chaotix opts to not wait for the 32X to fully process the command due to how slow the process can take. So instead, it just runs the command whenever the 32X is able to every frame.

Chaotix builds the draw command list via a buffer that is built in RAM first, and then sent over via DREQ. Here's a list of the commands that I was able to figure out:

- 00 - End of list
- 01 - Draw sprite
- 02 - Draw trailing dot at velocity
- 03 - Draw trailing dot at angle
- 04 - Draw trailing dot at angle using global acceleration
- 05 - Set global trailing dot acceleration
- 06 - Set dissolve and trailing dot velocity
- 07 - Define 2D boundaries
- 08 - Define 3D boundaries
- 09 - Draw line
- 0A - Draw polygon
- 0B - Define 3D polygon shape
- 0C - Draw 3D polygon shape
- 0D - Draw top of sprite dissolving
- 0E - Draw sprite dissolving from the bottom
- 0F - Draw a stack of bars
- 10 - Draw the distortion on the Sega logo
- 11 - Draw the HUD (not fully sure what it's doing so far)
- 12 - Draw bonus stage item
- 13 - Load new stage specific sprite list
- 14 - ? (Something to do with the HUD)
- 15 - Set player palette index offsets
- 16 - Draw final boss arena
- 17 - Draw zoomed sprite
- 18 - Set zoom value
- 19 - Draw stage end/game over circle transition
- 1A - Draw power-up icons

Let's go over each of them now and see how they are structured.

Draw Command 00 - End of list

Self explanatory. It defines the end of the list.

Draw Command 01 - Draw sprite

Code (Text):

```
1. .b $01
2. .b Sprite set ID
3. .b Sprite frame ID
4. .b Draw flags
5. .w X
6. .w Y
7. .w X scale
8. .w Y scale
```

Pretty self explanatory for the most part. For the default 100% scaling, set a scale value to $100. Lower values will scale up, higher values will scale down.

The draw flags are in this format in binary:

Code (Text):

```
1. 0000 mmyx
```

- x - X flip flag
- y - Y flip flag
- m - Mode
  - 00 - Normal
  - 01 - Priority
  - 02 - "Transparency" via vertical dithering (only if both X and Y scale are set to $100)

Draw Command 02 - Draw trailing dot at velocity

Code (Text):

```
1. .b $02
2. .b $00
3. .b Palette index
4. .b Trail flags
5. .w Draw area length (only when flag is set)
6. .w X
7. .w Y
```

```
 8. .w X velocity
 9. .w Y velocity
10. .w X acceleration
11. .w Y acceleration
```

This will draw a moving dot that leaves a trail.

The trail flags are as follows:

Code (Text):

```
1. 00ae 000t
```

- a - Set draw area length relative to position (override the "e" flag)
- e - Set draw area up to end of screen
- t - Thickness
  - 0 - 1px
  - 1 - 2px

If there's a draw area that's defined, then when the dot moves past the end of that area, it will reset itself back at its starting position. **Note that there seems to be a bug with this functionality where it it will wrap to the top of the screen after going past the bottom, and then reset itself when it goes back to its initial Y position.**

Draw Command 03 - Draw trailing dot at angle

Code (Text):

```
 1. .b $03
 2. .b $00
 3. .b Palette index
 4. .b Trail flags
 5. .w Draw area length (only when flag is set)
 6. .w X
 7. .w Y
 8. .w Angle
 9. .w Velocity
10. .w Acceleration
```

Draw Command 04 - Draw trailing dot at angle using global acceleration

Code (Text):

```
1. .b $04
2. .b $00
3. .b Palette index
4. .b Trail flags
5. .w Draw area length (only when flag is set)
6. .w X
7. .w Y
8. .w Angle
9. .w Velocity
```

Draw Command 05 - Set global trailing dot acceleration

Code (Text):

```
1. .b $05
2. .b $00
3. .l Global X acceleration
4. .l Global Y acceleration
```

Draw Command 06 - Set dissolve and trailing dot velocity

Code (Text):

```
1. .b $06
2. .b $00
3. .w X velocity
4. .w Y velocity
```

Chaotix uses this to move the dissolved particles along with the camera.

Draw Command 07 - Define 2D boundaries

Code (Text):

```
1. .b $07
2. .b $00
3. .w Left
4. .w Right
5. .w Top
```

```
6. .w Bottom
```

Draw Command 08 - Define 3D boundaries

Code (Text):

```
1. .b $08
2. .b $00
3. .w Near
4. .w Far
```

Draw Command 09 - Draw line

Code (Text):

```
1. .b $09
2. .b Palette index
3. .w Point 1 X
4. .w Point 1 Y
5. .w Point 2 X
6. .w Point 2 Y
```

Draw Command 0A - Draw polygon

Code (Text):

```
1. .b $0A
2. .b Palette index
3. .b Draw mode
4. .b Number of vertices
5. [For each vertex]
6.     .w X
7.     .w Y
```

The draw modes are as follows:

- 0 - Corners
- 1 - Wireframe
- 2 - Solid

Draw Command 0B - Define 3D polygon shape

Code (Text):

```
1. .b $0B
2. .b $00
3. .b Shape ID
4. .b Number of vertices
5. [For each vertex]
6.     .w X
7.     .w Y
```

Draw Command 0C - Draw 3D polygon shape

Code (Text):

```
1. .b $0C
2. .b $00
3. .b Palette index
4. .b Draw mode
5. .b Shape ID
6. .b Pitch rotation
7. .b Yaw rotation
8. .b Roll rotation
9. .w X
10. .w Y
11. .w Z
```

Note that there is no Z sorting involved, so you gotta do that yourself.

Draw Command 0D - Draw top of sprite dissolving

Code (Text):

```
1. .b $0D
2. .b Sprite set ID
3. .b Sprite frame ID
4. .b Draw flags
5. .w X
6. .w Y
```

Draw Command 0E - Draw sprite dissolving from the bottom

Code (Text):

```
1. .b $0E
2. .b Sprite set ID
3. .b Sprite frame ID
4. .b Draw flags
5. .w X
6. .w Y
```

Draw Command 0F - Draw a stack of bars

Code (Text):

```
1. .b $0F
2. .b Bar count
3. [For each bar]
4.     .w Height
5.     .b Even pixel palette index
6.     .b Odd pixel palette index
```

Draw Command 10 - Draw the distortion on the Sega logo

Code (Text):

```
 1. .b $10
 2. .b Sprite set ID
 3. .b Sprite frame ID
 4. .b Draw flags
 5. .w X
 6. .w Y
 7. .w X scale
 8. .w Left distortion
 9. .w Right distortion
10. .w X distortion intensity
11. .w Y scale
12. .w Top distortion
13. .w Bottom distortion
14. .w Y distortion intensity
```

Draw Command 11 - Draw the HUD (not fully sure what it's doing so far)

Code (Text):

```
1. .b $11
2. .b $00
3. .w ?
```

```
4. .w ?
5. .w ?
6. .w ?
```

or

Code (Text):

```
1. .b $11
2. .b $01
3. .l Score
4. .w Ring count
```

Draw Command 12 - Draw bonus stage item

Code (Text):

```
 1. .b $12
 2. .b Sprite set ID
 3. .b Sprite frame ID
 4. .b Draw flags
 5. .w X
 6. .w Y
 7. .w Z
 8. .w Length
 9. .b Palette index (top side)
10. .b Palette index (bottom side)
11. .b Palette index (left side)
12. .b Palette index (right side)
```

Draw Command 13 - Load new stage specific sprite list

Code (Text):

```
1. .b $13
2. [For each entry]
3.     .b Sprite set ID
4.     .b Palette index offset
5. .b $FF
```

This overwrites the currently loaded stage specific sprites.

Draw Command 14 - ? (Something to do with the

HUD)

Code (Text):

```
1. .b $14
2. .b ?
3. .w ?
4. .w ?
5. .w ?
6. .w ?
```

Draw Command 15 - Set player palette index offsets

Code (Text):

```
1. .b $15
2. .b $00
3. .b Player 1 character ID
4. .b Player 1 palette index offset
5. .b Player 2 character ID
6. .b Player 2 palette index offset
```

Draw Command 16 - Draw final boss arena

Code (Text):

```
1. .b $16
2. .b Angle
3. .w X
4. .w Y
5. .w Z
```

Draw Command 17 - Draw zoomed sprite

Code (Text):

```
1. .b $17
2. .b Sprite set ID
3. .b Sprite frame ID
4. .b Draw flags
5. .w X
6. .w Y
7. .w X scale
8. .w Y scale
```

Draw Command 18 - Set zoom value

Code (Text):

```
1. .b $18
2. .b $00
3. .w Zoom value
```

Draw Command 19 - Draw stage end/game over circle transition

Code (Text):

```
1. .b $19
2. .b Scale
```

You can find the parameters for the sprites that gets drawn for this at $600347C in the SH-2 program ($7AC7C in the ROM), which is as follows:

Code (Text):

```
1. ; Sprite 1
2. dc.w $0050      ; X
3. dc.w $0070      ; Y
4. dc.w %00        ; Flip flags (no flip)
5. dc.b $51        ; Sprite set ID
6. dc.b $00        ; Sprite frame ID
7.
8. ; Sprite 2
9. dc.w $00F0      ; X
10. dc.w $0070     ; Y
11. dc.w %11       ; Flip flags (X and Y flip)
12. dc.b $51       ; Sprite set ID
13. dc.b $00       ; Sprite frame ID
14.
15. ; Sprite 3
16. dc.w $00A0     ; X
```

```
17. dc.w $0070      ; Y
18. dc.w %00        ; Flip flags (no flip)
19. dc.b $51        ; Sprite set ID
20. dc.b $01        ; Sprite frame ID
```

The code defines the number of sprites in this list at $6003436 in the SH-2 program ($7AC36 in the ROM).

Draw Command 1A - Draw power-up icons

Code (Text):

```
1. .b $1A
2. .b Player 1 power-up flags
3. .b Player 2 power-up flags
4. .b Shared power-up flags
```

The Other 32X Commands

Not a whole lot to say. Command 04 simply sets every character's palette offset for use with character selection menus. Command 07 takes in a parameter via communication register 2 (word), where the upper byte is the sprite list ID and the lower byte is the palette index offset. I have nothing to say regarding the special stage command, because that's a completely different beast, and I've not looked into it yet.

Conclusion

So yeah, this is the gist of how graphics are rendered in Chaotix. I hope this might be of some use for hacking and documentation stuff. I know it's not exactly complete, but it's still pretty extensive.

Last edited: Nov 11, 2023

Informative x **12** Like x **5** Useful x **1** List

Nov 9, 2023 #2

I have updated the post. I found out how the missing draw commands (the ones that used to be marked with "?") worked. They are used for drawing trailing dots.

Devon

I think I'm paranoid

1,158
1,296
93
your mom

Here's a sample of what I'm talking about.

[IMG]

And this is the vertical wrap bug that I mentioned.

[IMG]

Last edited: Nov 9, 2023

Like x **8** List

Nov 9, 2023 #3

Chimes

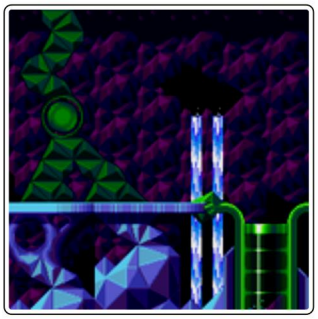
The One SSG-EG Maniac

572
459
63

Getting violent math class reminders here

Like x **1** List

Nov 11, 2023 #4



Goodness, hacks using 32x capabilities are going to be going wild with all this new stuff… heck, looking at this makes me think a proof-of-concept Mania port to the SEGA CD+32x isn't impossible!

SyntaxTsu

285
130
43
Neo Metropolis Zone

(You must log in or sign up to reply here.)