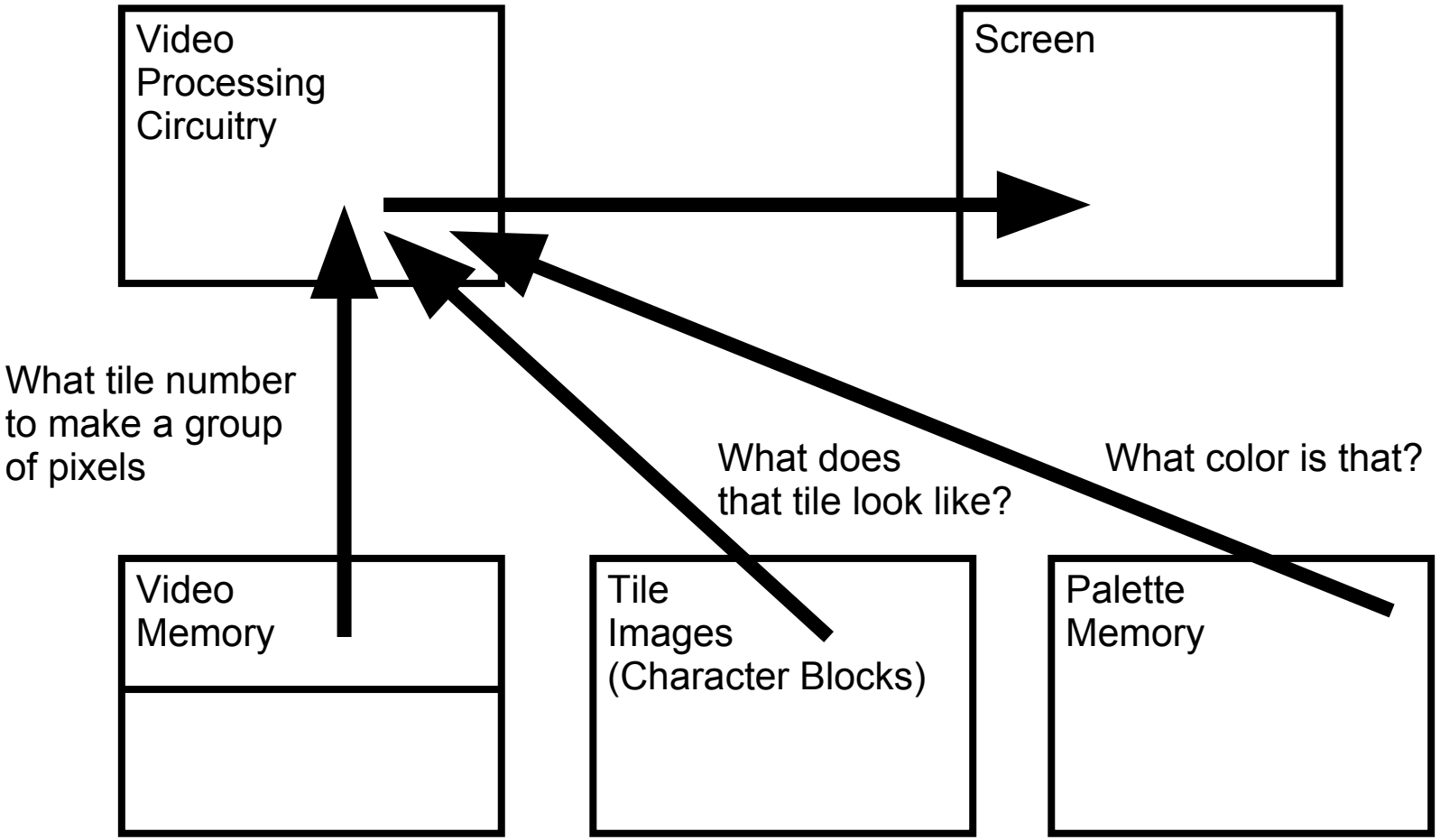
CS 2261: Media Device Architecture - Week 9 - part 2

Overview

- Some clarifications and a demo on Sprites
  - 1D vs 2D mapping modes (2D assumes 256px wide image!)
  - 4bpp vs 8bpp
  - The interplay of both
- Mode 0

Mode 0 Tilemaps



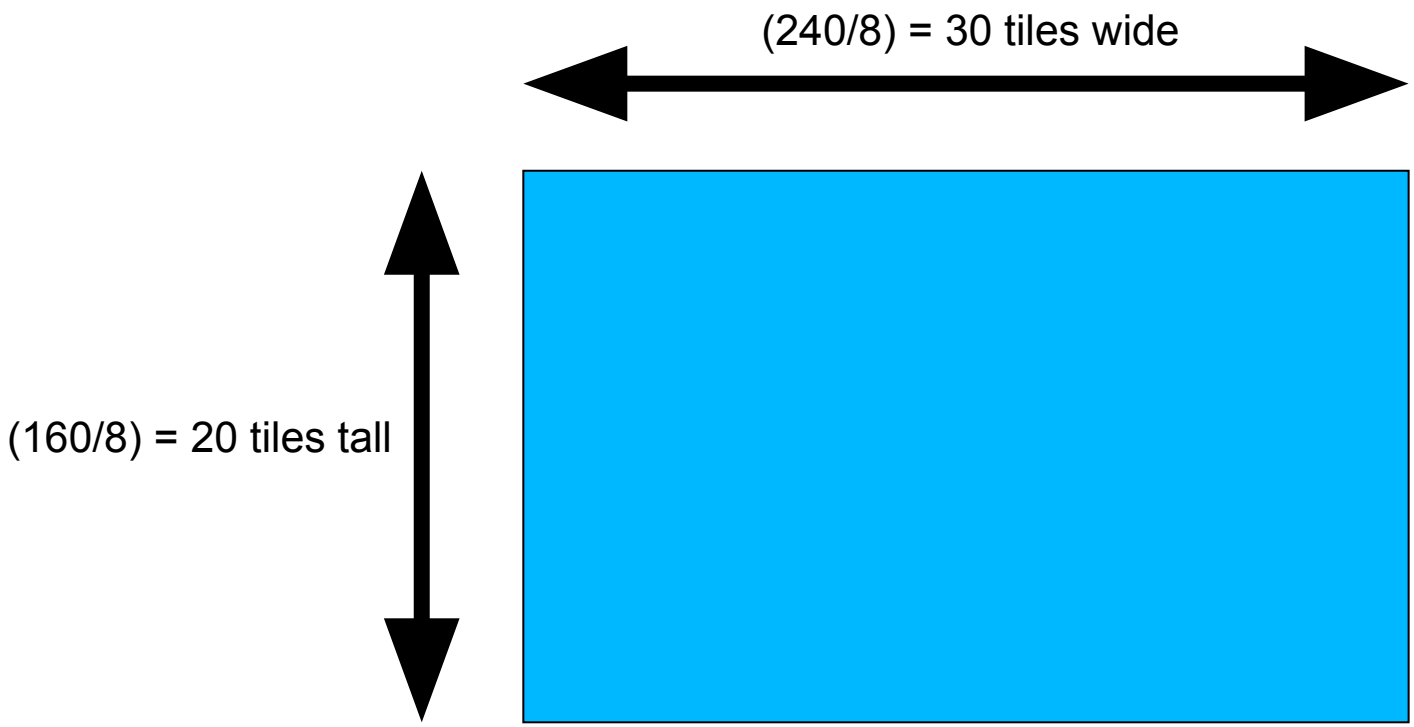
Tiles

- These basically the same as Sprite tiles, though they go in a slightly different place in memory, and working with them is a little different

- Each tile is an 8x8 pixel bitmap image
  - Treated like big pixels, as the building block of a larger image
  - 16 or 256 colors (4bpp or 8bpp)

Tiles on Screen

- Since each tile is 8x8:



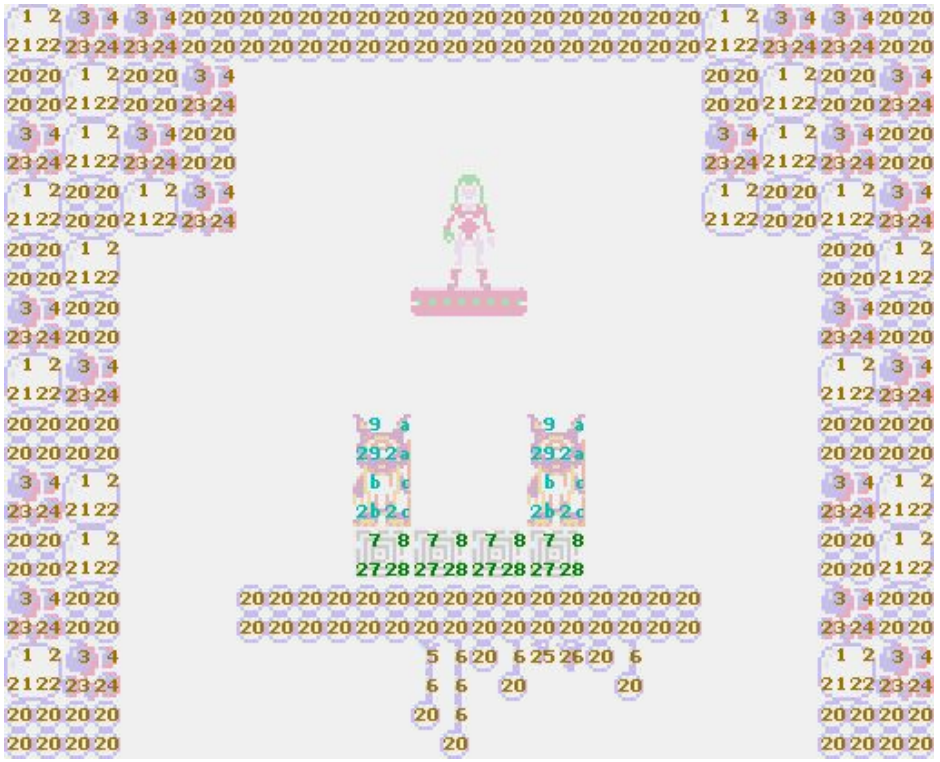
Tilemap

- 2D array of tile indexes
- Tile indexes in a Tilemap are just like mode4 pixels in the VRAM bitmap
  - Tile : Tilemap :: Pixel : Bitmap

Illustrations from Tonc

Tiles (with a map of indices below):

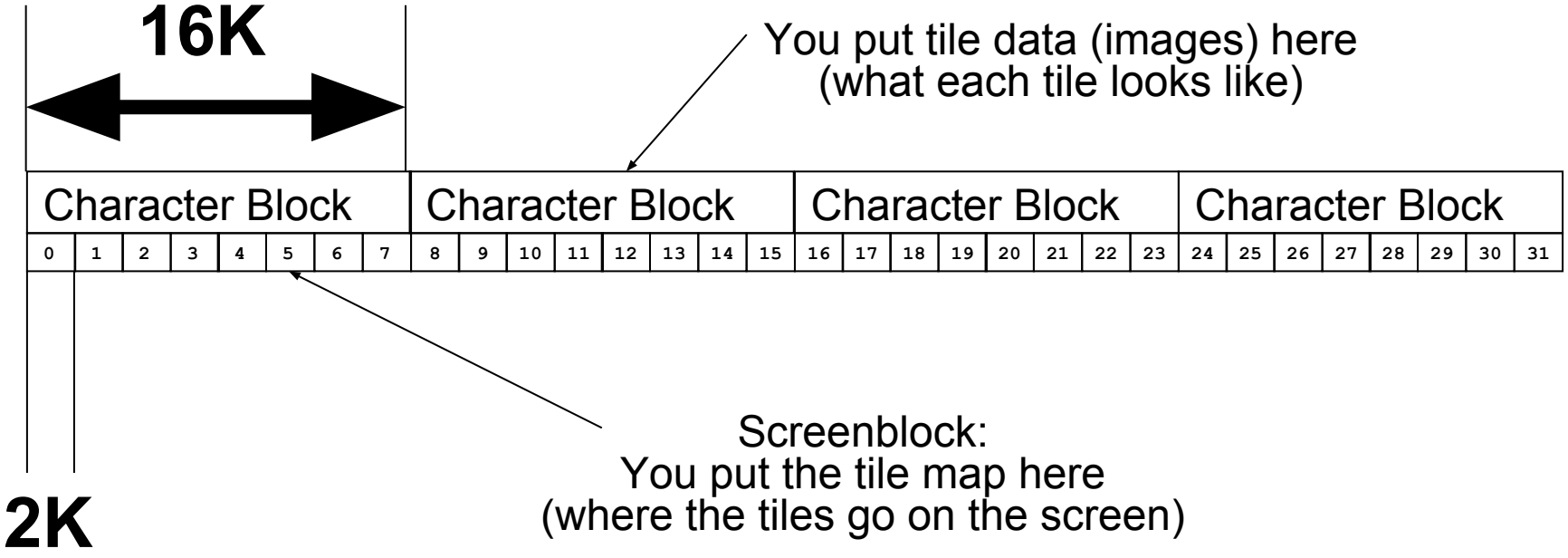
Tilemap (with zeros omitted for brevity):



GBA Tile Mode Features

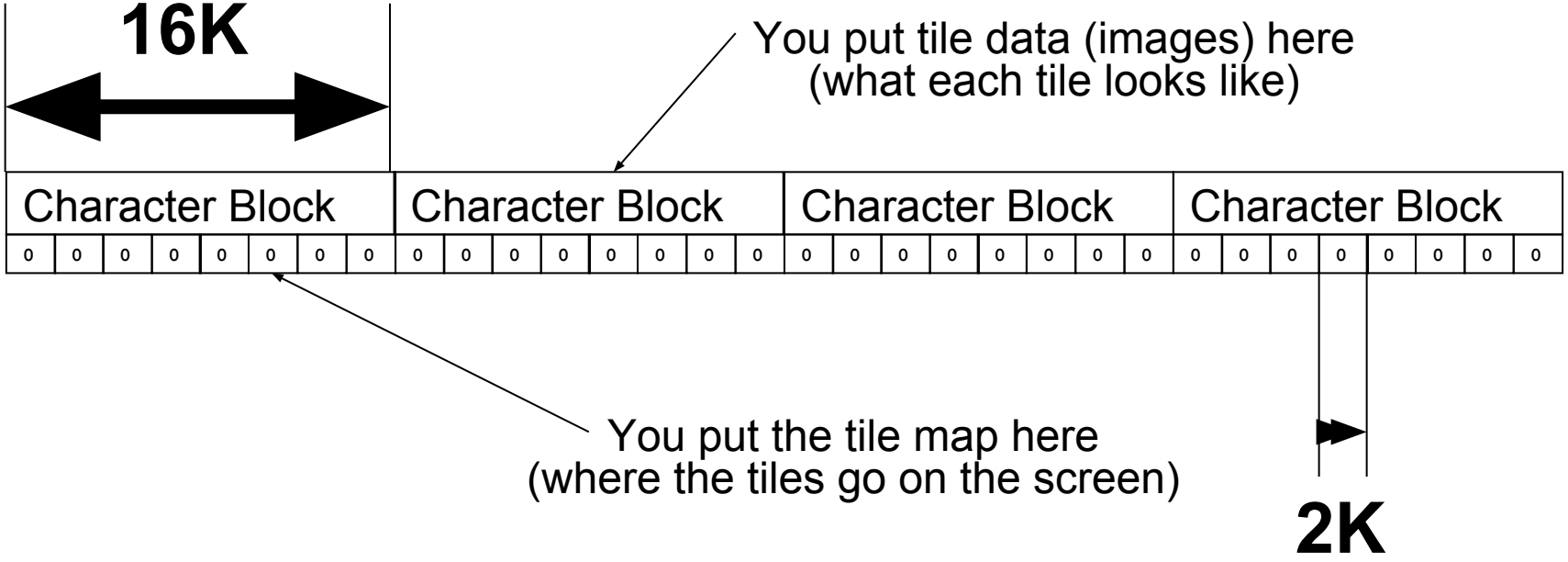
- Up to 4 tiled background layers
  - Not just BG2 anymore!
  - As small as 128x128 pixels
  - As large as 1024x1024 pixels!
- Hardware scrolling
  - Maps can be larger than the screen
- Hardware Parallax
  - Layers scroll at different speeds to simulate depth
- Affine transformations
  - Rotation and scaling effects on the background itself
  - We'll cover these shortly when we do the same for Sprites.

Video Buffer Layout for Tiles



Sprite Data	Sprite Data

Video Buffer Layout for Tiles & Sprites



Sprite Data	Sprite Data

Tile Memory

- **Double the sprite memory compared with bitmap modes**
- Tile data video in video buffer
  - From 0x6000000 to 0x600FFFF
  - On 16k boundary (any of 4 spots)
- Tile map also stored in video buffer
  - On 2k boundary (any of 32 spots)
- Tile data and Tile map are in same space but normally would not overlap.

Tile Data and Tilemap

- The tilemap is stored in the same location as the video buffer (in the bitmap video modes), an array of numbers that point to the tile images.
- In text backgrounds (0 and 1) the tile map is comprised of 16-bit numbers, while the rotation backgrounds (2 and 3) store 8-bit numbers in the tile map.
- When working with tile-based modes, video memory is divided into 4 logical char base blocks, which are made up of 32 smaller screen base

Tilemap Entries

Screen entry format for regular backgrounds

F E D C	B	A	9 8 7 6 5 4 3 2 1 0
PB	VF	HF	TID

bits	name	define	description
0-9	TID	*SE_ID#*	**Tile-index** of the SE.
A-B	HF, VF	SE_HFLIP, SE_VFLIP. *SE_FLIP#*	**Horizontal/vertical flipping** flags.
C-F	PB	*SE_PALBANK#*	**Palette bank** to use when in 16-color mode. Has no effect for 256-color bgs (`REG_BGxCNT{6}` is set).

REG_GBxCNT (0x04000008, 0x0400000A, etc.)

REG_BGxCNT @ 0400:0008 + 2x

F E	D	C B A 9 8	7	6	5 4	3 2	1 0
Sz	Wr	SBB	CM	Mos	-	CBB	Pr

bits	name	define	description
0-1	Pr	*BG_PRIO#*	**Priority**. Determines drawing order of backgrounds.
2-3	CBB	*BG_CBB#*	**Character Base Block**. Sets the charblock that serves as the base for character/tile indexing. Values: 0-3.
6	Mos	BG_MOSAIC	**Mosaic** flag. Enables mosaic effect.
7	CM	BG_4BPP, BG_8BPP	**Color Mode**. 16 colors (4bpp) if cleared; 256 colors (8bpp) if set.
8-C	SBB	*BG_SBB#*	**Screen Base Block**. Sets the screenblock that serves as the base for screen-entry/map indexing. Values: 0-31.
D	Wr	BG_WRAP	**Affine Wrapping** flag. If set, affine background wrap around at their edges. Has no effect on regular backgrounds as they wrap around by default.
E-F	Sz	*BG_SIZE#*, *see below*	**Background Size**. Regular and affine backgrounds have different sizes available to them. The sizes, in tiles and in pixels, can be found in table 9.5.

Sz-flag	(tiles)	(pixels)
00	32x32	256x256
01	64x32	512x256
10	32x64	256x512
11	64x64	512x512

Table 9.5a

Steps to Tiles

- Create tile images
- Create screen image
- Create a palette
- Store tile images in a character block
- Store screen image in one or more screen blocks
- Store palette in palette memory area
- Set image control buffer and Mode

Steps to Tiles

- Create tile images
- Create screen image
- Create a palette -- Usenti is your friend for these
- Store tile images in a character block
- Store screen image in one or more screen blocks
- Store palette in palette memory area
- Set image control buffer and Mode

Mode & BG Specifics

Mode	Backgrounds	Rotation/Scaling
0	0, 1, 2, 3	No
1	0, 1, 2	Yes (only background 2)
2	2, 3	Yes (both)

Background	Resolution	Rotation/Scaling
0	512 x 512	No
1	512 x 512	No
2	128 to 1024	Yes
3	128 to 1024	Yes

Bg Layers

- 0 & 1
  - Max Resolution 512x512
  - Tile map of 16 bit numbers
- 2 & 3
  - Resolution from 128x128 to 1024x1024
  - Rotation & Scaling
  - Tile map of 8 bit numbers

REG_BGxHOFS 2 0400:0010h + 4·x
REG_BGxVOFS 2 0400:0012h + 4·x

Two write-only 16-bit registers for setting the offset

These control where the screen is on the background

(not the other way around!)

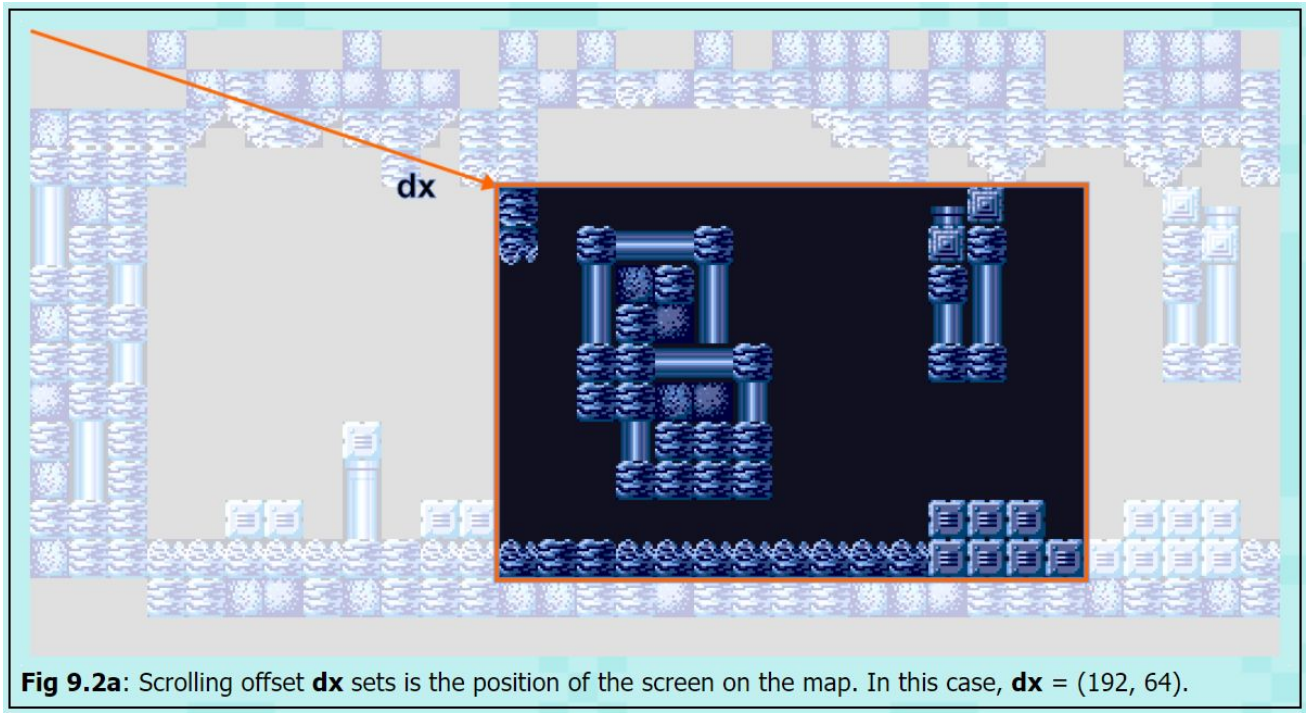


Fig 9.2a: Scrolling offset **dx** sets is the position of the screen on the map. In this case, **dx** = (192, 64).

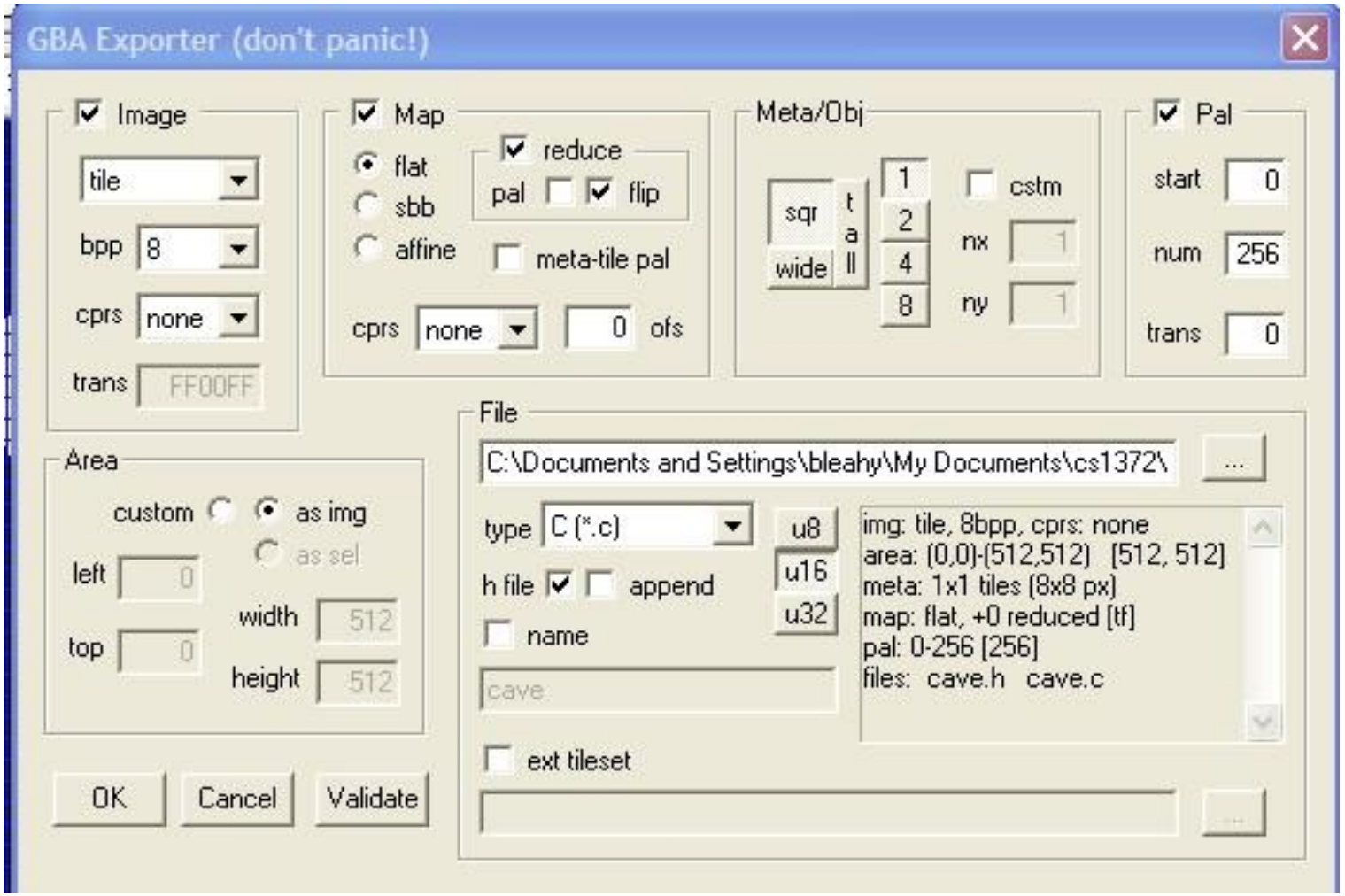
Color

- 256 color tiles
  - 8 bits per pixel
  - Tiles share one 256 color palette
- 16 color tiles
  - 4 bits per pixel
  - Tiles use one of 16 palettes, 16 colors each

Correct

Incorrect: 256 color tiles being displayed as 16 color tiles

usenti


GBA Exporter (don't panic!)
Image
tile
bpp 8
cprs none
trans FF00FF
Map
flat
sbb
affine
reduce
pal
flip
meta-tile pal
cprs none
0 ofs
Meta/Obj
sqr
wide
tall
1
2
4
8
cstm
nx 1
ny 1
Pal
start 0
num 256
trans 0
Area
custom
as img
as sel
left 0
top 0
width 512
height 512
File
C:\Documents and Settings\bleahy\My Documents\cs1372\
...
type C (*.c)
u8
u16
u32
h file
append
name
cave
img: tile, 8bpp, cprs: none
area: (0,0)-(512,512) [512, 512]
meta: 1x1 tiles (8x8 px)
map: flat, +0 reduced [tf]
pal: 0-256 [256]
files: cave.h cave.c
ext tileset
OK
Cancel
Validate