EPFL

EMBEDDED SYSTEMS LABORATORY

Topic 3: (Part B)

I/O and Peripheral Devices Management

GRAPHICS in the Nintendo DS

Systèmes Embarqués Microprogrammés

Prof. David Atienza Alonso, SEL-STI

Content of Session

- Use of backgrounds in the NDS: tiled/text mode
  - NDS video modes for tiled mode
  - Concepts: tiles and their effects
  - Multi-palettes with tiles
  - Background representations with tiles
  - Review of the (V)RAM structure
  - Copying large data sets: direct memory address (DMA) controller
- Drawing graphics in tiled/text mode in the NDS
  - Example of custom background definition in tiled mode
  - Converting images to tiles: revisit the grit tool
  - Implementation of effects using multiple backgrounds

NDS screen modes: 2D engines configuration
Tiled modes

- From the four 2D engines modes, select appropriate one
  - Tiled
  - Rotation or rotoscale
  - Extended rotation
  - Framebuffer

Main engine: 7 modes and framebuffer

Mode	BG0	BG1	BG2	BG3
0	Tiled/3D	Tiled	Tiled	Tiled
1	Tiled/3D	Tiled	Tiled	Rotoscale
2	Tiled/3D	Tiled	Rotoscale	Rotoscale
3	Tiled/3D	Tiled	Tiled	Ext. Rotoscale
4	Tiled/3D	Tiled	Rotoscale	Ext. Rotoscale
5	Tiled/3D	Tiled	Ext. Rotoscale	Ext. Rotoscale
6	3D	N/A	Large Bitmap	N/A
FrameBuf.	Direct VRAM display as a bitmap			

Sub engine: 6 modes

Mode	BG0	BG1	BG2	BG3
0	Tiled	Tiled	Tiled	Tiled
1	Tiled	Tiled	Tiled	Rotoscale
2	Tiled	Tiled	Rotoscale	Rotoscale
3	Tiled	Tiled	Tiled	Ext. Rotoscale
4	Tiled	Tiled	Rotoscale	Ext. Rotoscale
5	Tiled	Tiled	Ext. Rotoscale	Ext. Rotoscale

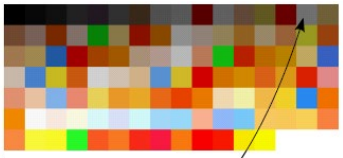
- In the case of tiled in different backgrounds?
  - All modes are valid, just configure ***REG_DISPCNT*** with BG0: Modes 0 to 5
  - Example: activate mode 0 and background 0 (BG0) to plot tiles first

```
REG_DISPCNT = MODE_0_2D | DISPLAY_BG0_ACTIVE;
```

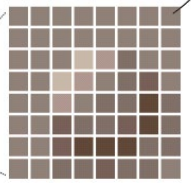
The tiled/text mode: Tiles and drawing options

- Backgrounds are composed of smaller blocks
  - Tile/text: minimum unit to compose tiled backgrounds, made of blocks of 8x8 pixels

Game: Yoshi Island



Palette

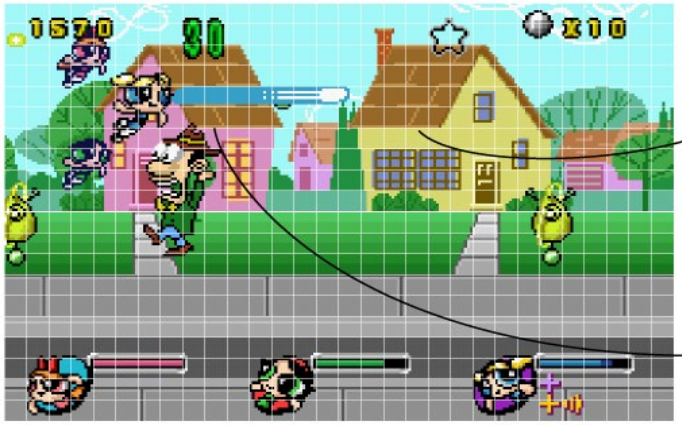


Tile

Tile for letter 'A'

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

- Two components to create effects in tiled backgrounds
  - Palettes: different colours sets
  - Transformations: tiles rotations
    - Horizontal , vertical or both

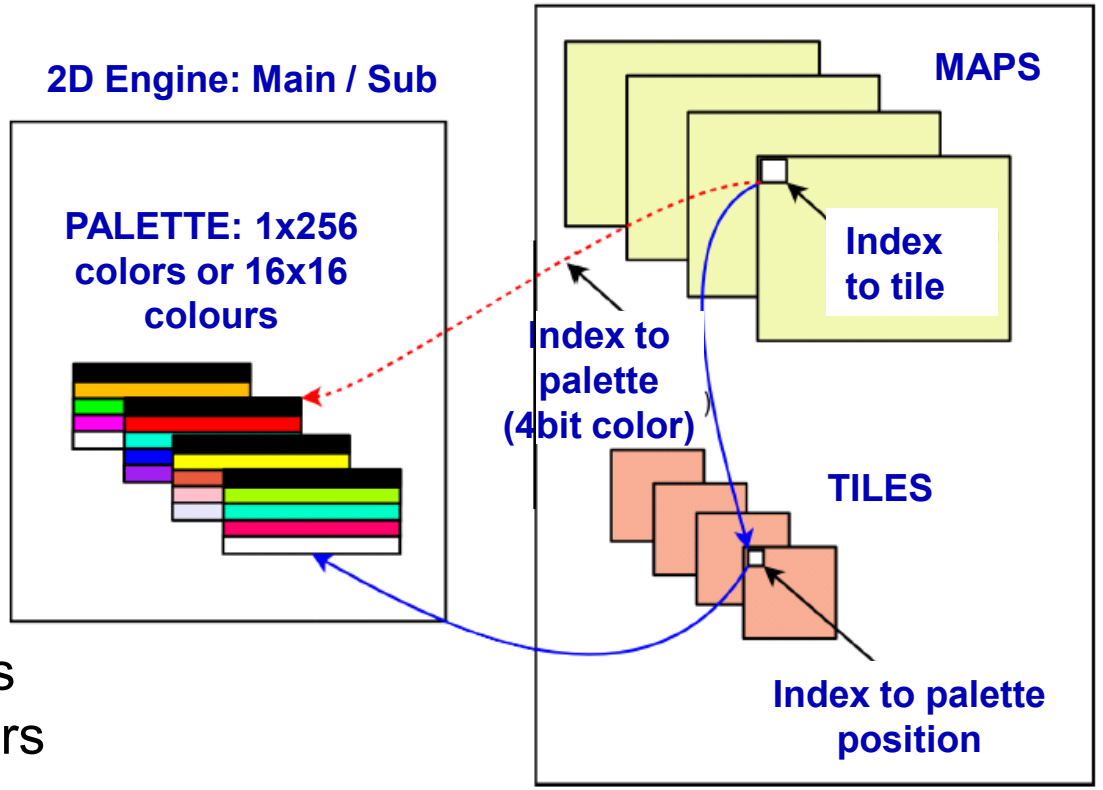


Game: Power Puff Girls

Horizontal mirror effect

Complete display configuration tiled mode: Three elements to store and use

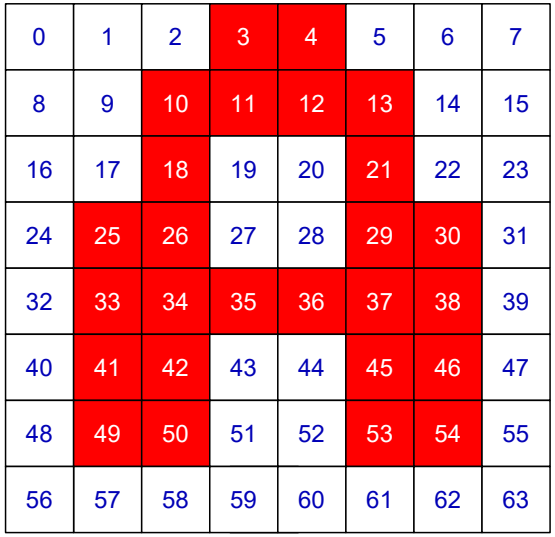
- Maps of tiles for background: references to tiles
  - If multiple palettes are used, sets which one to use
- Definitions of tiles that can be used
  - Up to 1024 tiles as index map is 10 bits
- Palettes: trade-offs of memory use
  - Resolution: 1x256 colors
  - Little memory: 16 palettes of 16 colors



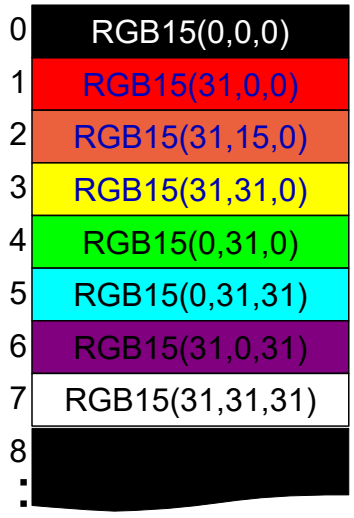
- We need to configure the engine background: ***BGCTRL[x]***

The tiled/text mode: Representations options for tiles

- Two possibilities of RGB palettes for tiles
  - 1 palette of 256 colors each: 8 bits per pixel
  - 16 palettes of 16 colors each: 4 bits per pixel

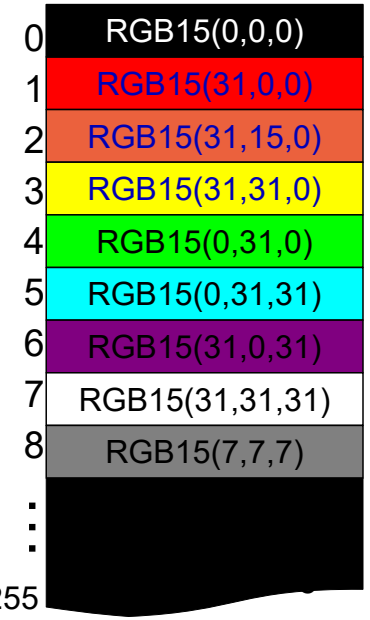


- Tile: matrix of color indexes in a palette
  - Little endian: least significant nibble is left pixel




```
// With 16-color palette
unsigned char A16[] = {
    0x00,0x10,0x01,0x00,
    0x00,0x11,0x11,0x00,
    0x00,0x01,0x10,0x00,
    0x10,0x01,0x10,0x01,
    0x10,0x11,0x11,0x01,
    0x10,0x01,0x10,0x01,
    0x10,0x01,0x10,0x01,
    0x00,0x00,0x00,0x00,
};
```

```
// With 256-color palette
unsigned char A256[] =
{
        0,0,0,1,1,0,0,0,
        0,0,1,1,1,1,0,0,
        0,0,1,0,0,1,0,0,
        0,1,1,0,0,1,1,0,
        0,1,1,1,1,1,1,0,
        0,1,1,0,0,1,1,0,
        0,1,1,0,0,1,1,0,
        0,0,0,0,0,0,0,0,
};
```



The tiled/text mode: Example of color representation

- Define an 8-colour tile
  - Each color in the palette is 15-bit RGB pixel
    - 5 bits per color, the transparence bit is ignored
- Two steps
  1. Choose palette in 15-bit RGB pixel
  2. Implement color index table in tile
     - **Index 0 represents 'transparency'**: it displays the color 0 of the palette if there is no opaque pixel behind (e.g. from other background)

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

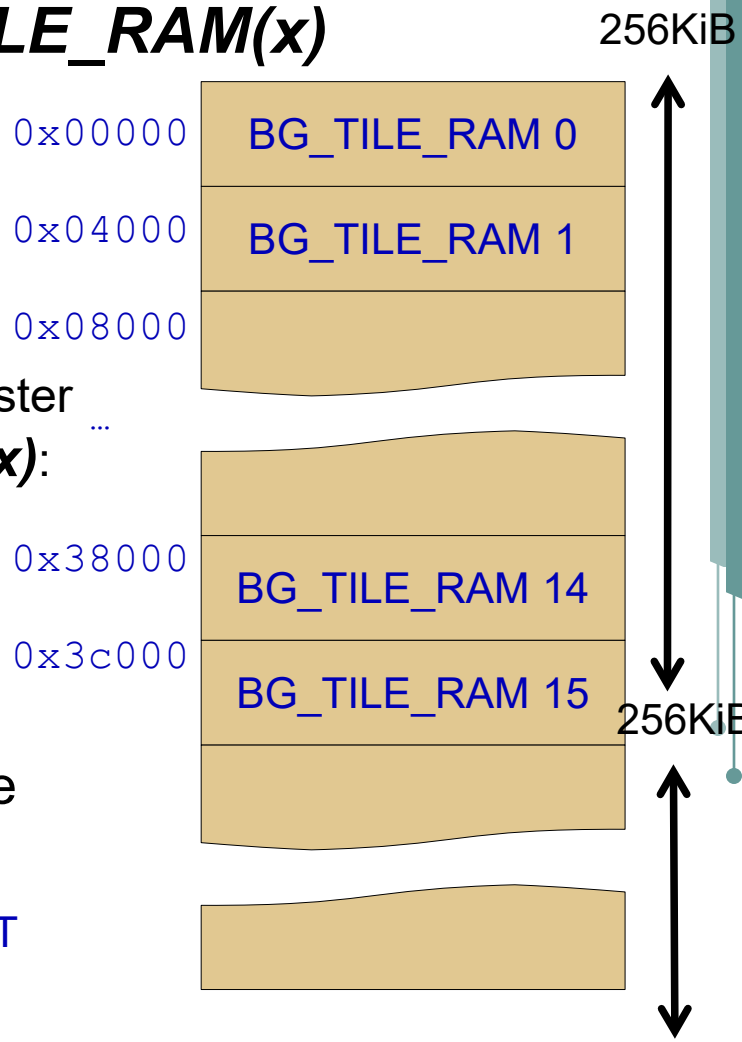
Palette

0	RGB15(0,0,0)
1	RGB15(31,0,0)
2	RGB15(31,15,0)
3	RGB15(31,31,0)
4	RGB15(0,31,0)
5	RGB15(0,31,31)
6	RGB15(31,0,31)
7	RGB15(31,31,31)
8	RGB15(7,7,7)
⋮	

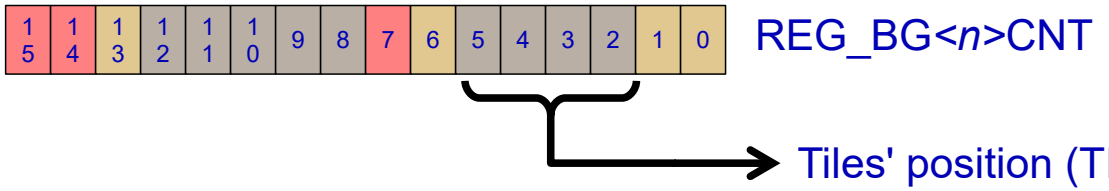
```
unsigned long my8-colorTile[] =
{
        0x12345678,
        0x12345678,
        0x12345678,
        0x12345678,
        0x12345678,
        0x12345678,
        0x12345678,
        0x12345678,
};
```

Tiles access and storage in memory

- Tiles stored from base address: ***BG_TILE_RAM(x)***
  - 2D graphic engines only allow tile base addresses as multiple of 16KB (0x4000)
  - The maximum size is 256KB
- Libnds reference them with form:
  - ***BG_TILE_BASE(x)***: configure BG control register
  - ***BG_TILE_RAM(x)*** / ***BG_TILE_RAM_SUB(x)***: access the tile set or modify it
  - Up to 16 values of base address: x=0..15

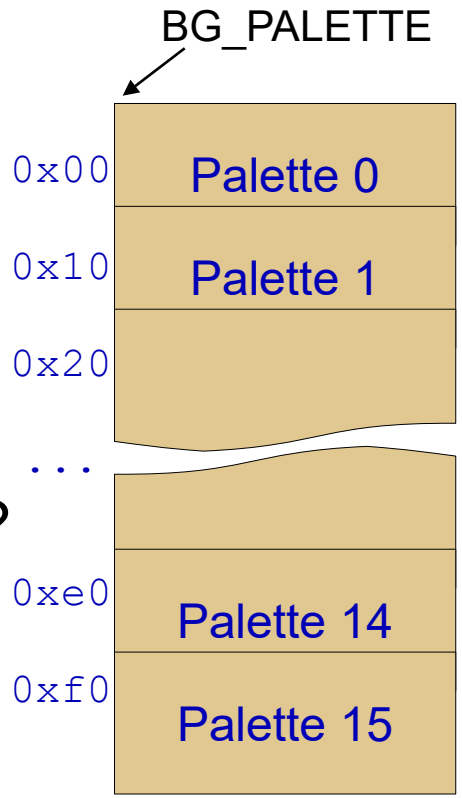


- Limit due ***BGCTRL[n]***: 4 bits reserved to store displacement from ***TILE_BASE*** address



Color palettes access and storage in memory

- Referenced in libnds as background palette: ***BG_PALETTE***, access using consecutive positions in memory with 8 bits
  - 1 palette with 256 colors: colors from 0x00 to 0xff
  - 16 palettes with 16 colors: two steps
    1. Palette from 0x0 to 0xf
    2. Color from 0x0 to 0xf

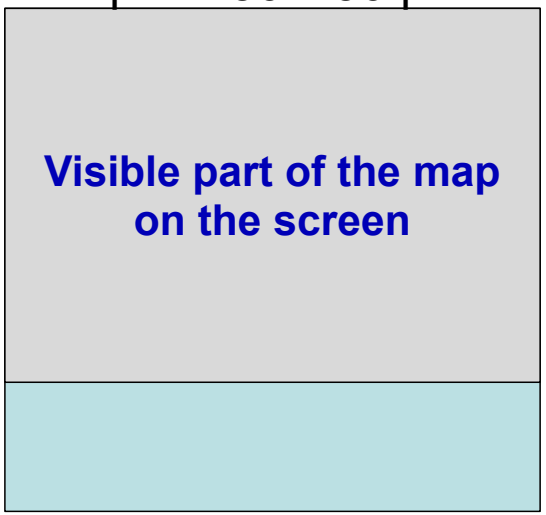


- Example: Which color is ***BG_PALETTE[0x25]***?
  - 256-palette: color 37 (0x25)
  - 16-bit palettes: 5th color in Palette 2

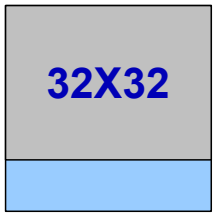
Screen representation: Tiled maps and scroll

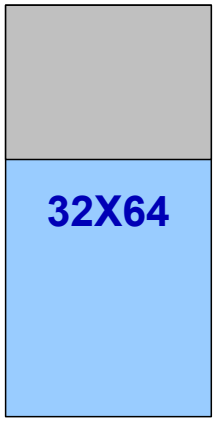
- Screen: map of tiles, but it can be larger than the actual screen
  - Not all the tiles can be visible
  - ***Scrolling effect***: visible area is adjusted

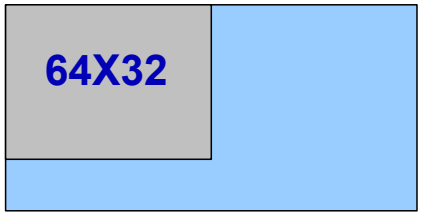
Example: 32x32 tiles create a map of 256x256 pixels

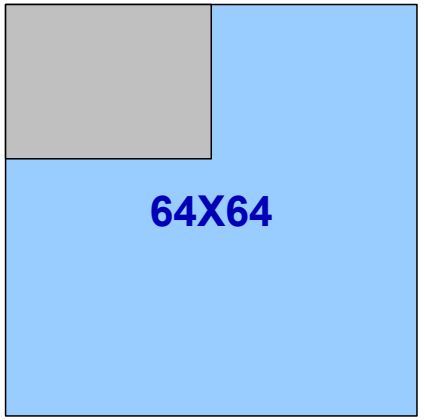


- 4 possibilities for the map on the NDS:
  - 32x32 tiles, 32x64 tiles, 64x32 tiles, 64x64 tiles



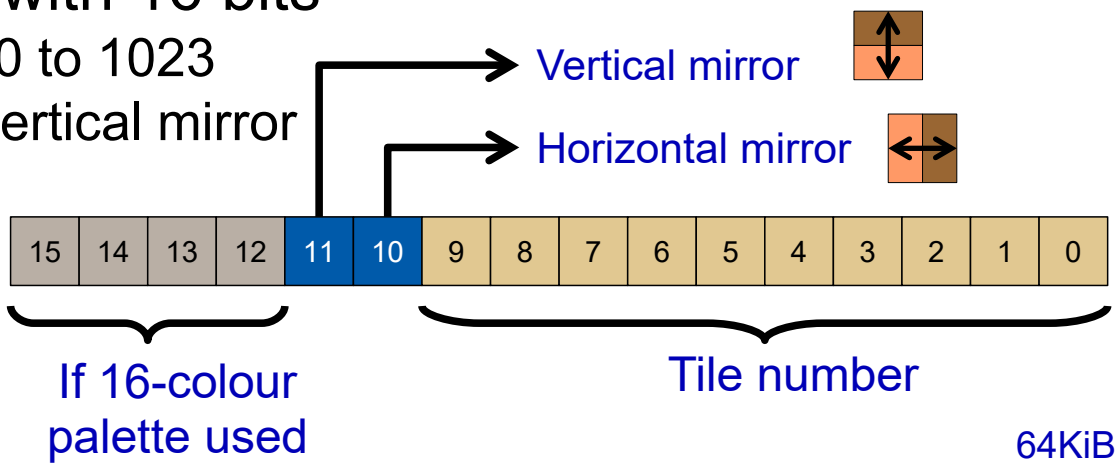




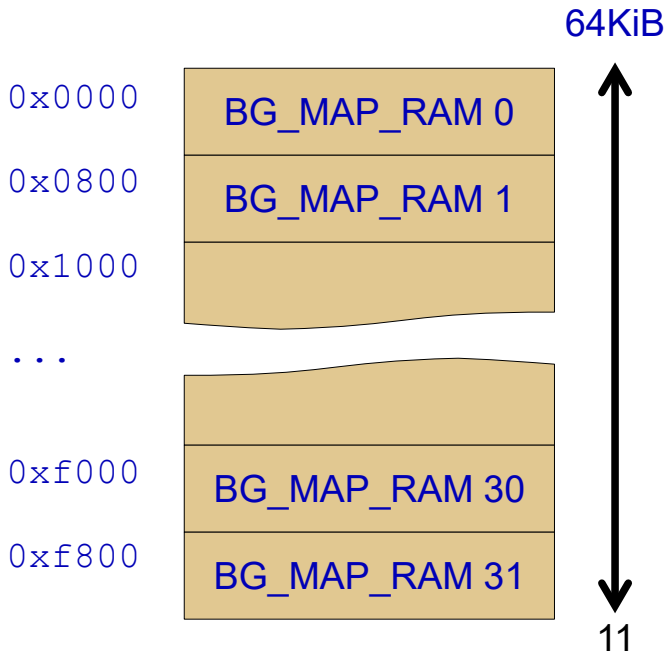


Tiles map format and storage

- Each tile is represented with 16 bits
  - 10 bits: Tile number from 0 to 1023
  - 2 bits: Horizontal and/or vertical mirror
  - 4 bits: Palette if 16-color
    one used by the tile

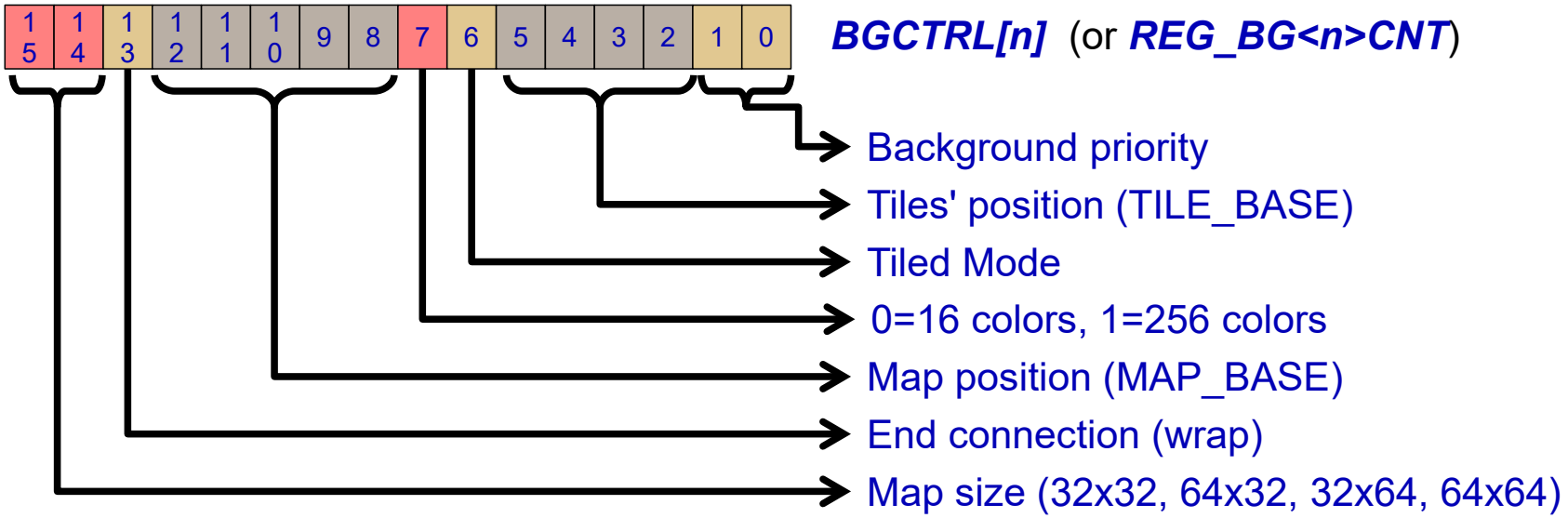


- A map can start in an address multiple of 2KB (0x800)
  - Most used BG: 32x32 tiles (2B/tile) = 2KB
- Libnds allows referencing them with:
  - ***BG_MAP_BASE(x)***: config. BG reg. Controller
  - ***BG_MAP_RAM(x)*** / ***BG_TILE_RAM_SUB(x)*** : access the tile set or modify it
  - Up to 31 values of base address: x=0..31



Configuring a tiled background

- Three options to configure
  - Number of tiles: {32 or 64}x{32 or 64}
  - Number of colours: 16 or 256
  - Memory positions: For each map and for tiles



- Example: Config. background 0 in SUB engine: 32x32 tiles, use 1 palette of 256 colors, tiles map stored in MAP_BASE0, and tiles at 2KB from TILE_BASE

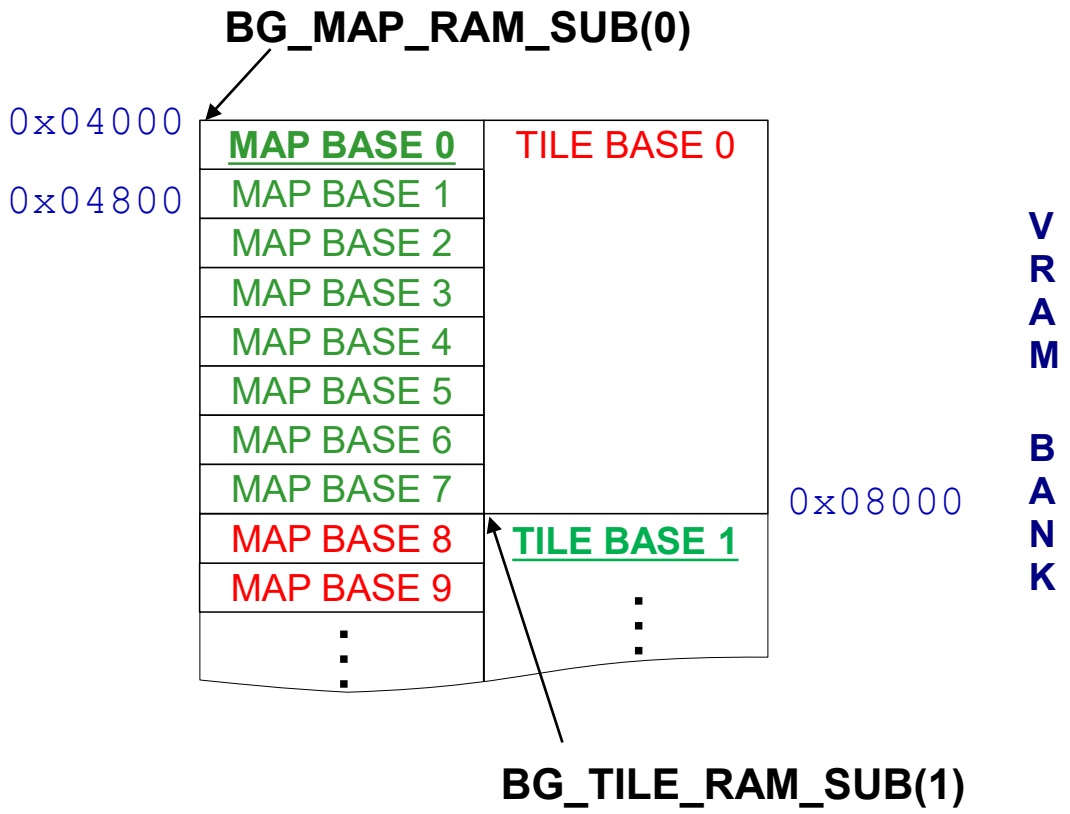
```
BGCTRL_SUB[0] = BG_32x32 | BG_COLOR_256
              | BG_MAP_BASE(0) | BG_TILE_BASE(1);
```

Example of Memory Use of BG_TILE_BASE and BG_MAP_BASE

- Background configuration example

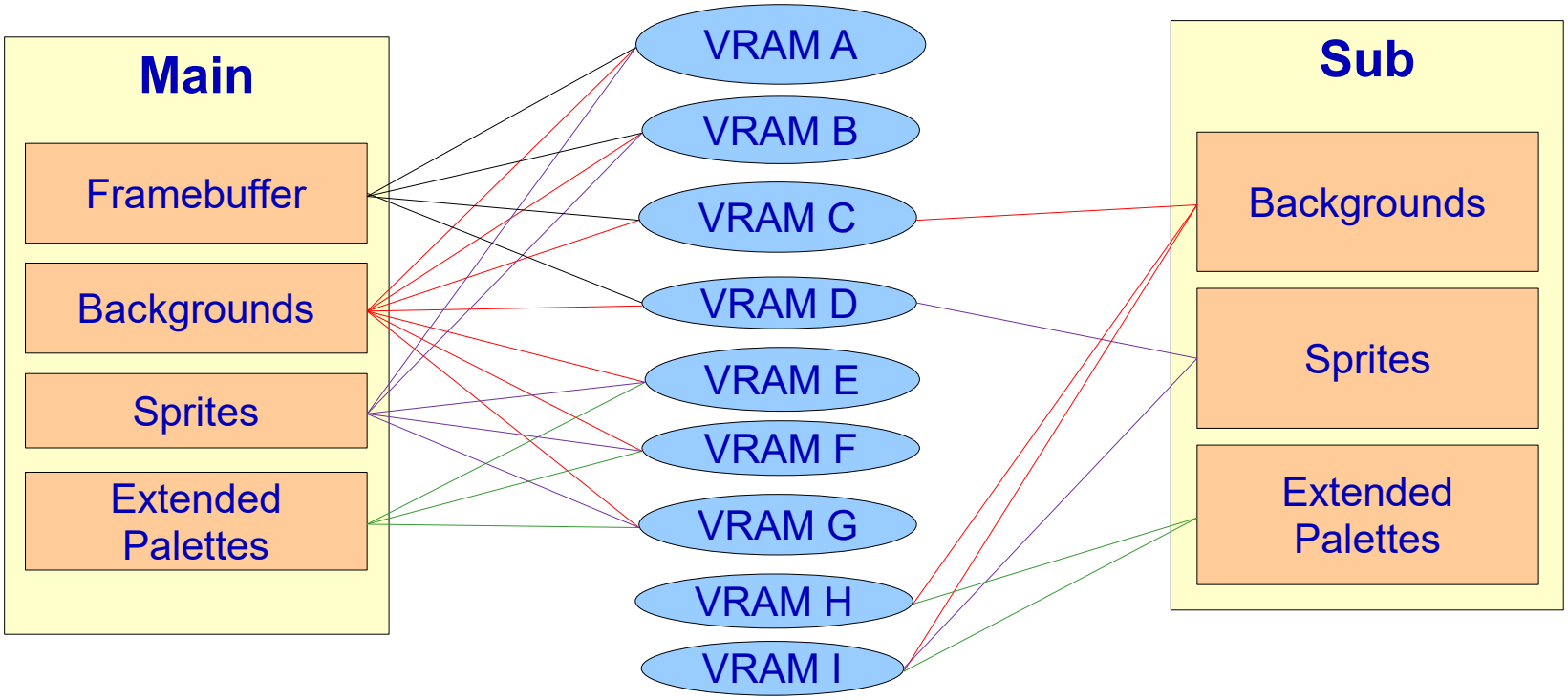
```
// 3) Configure the background
BGCTRL_SUB[0] = BG_MAP_BASE(0) | BG_TILE_BASE(1) | BG_32x32 | BG_COLOR_256;
```

- Location of BG_TILE_BASE and BG_MAP_BASE overlap at *0x4000*



Structure of tiled background in VRAM banks mapping

- Fixed location in VRAM banks for each type of data

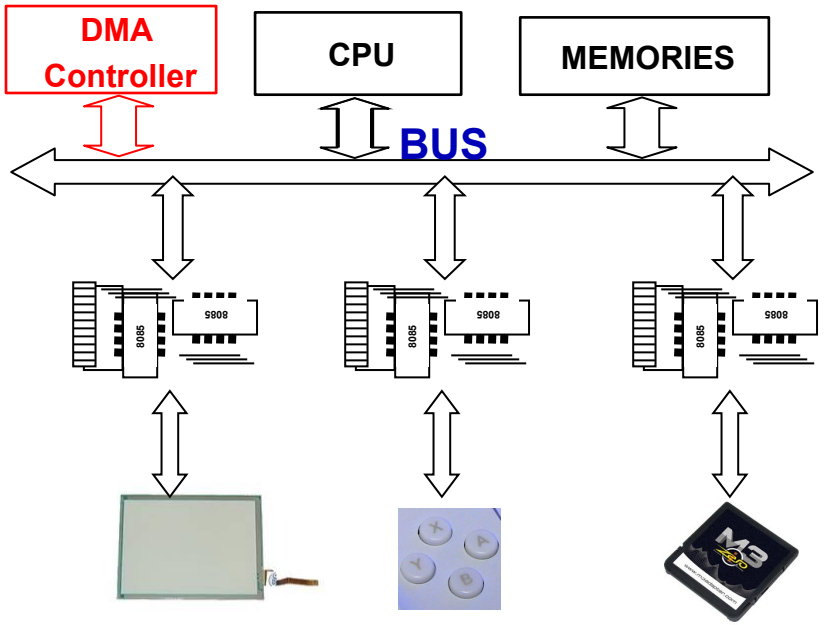


- Enable the necessary VRAM banks: **VRAM_?_CR**
  - Example: store 32 tiles, 1 palette of 16 colors, 1 map of 32x32 tiles
    ~4KB needed: enough with one bank (*bank A*), assign to *main 2D core*

VRAM_A_CR = VRAM_ENABLE | VRAM_A_MAIN_BG;

Efficient data transfer between CPU and I/O subsystem: Direct Memory Address (DMA)

- Mechanism to perform data transfers without CPU intervention, use of specialized module: **DMA Controller**
  1. CPU configures DMA controller to make data transfer between peripherals
     - Base address
     - Destination address
     - Data size to transfer
  2. DMA controller performs I/O operation autonomously when bus is available

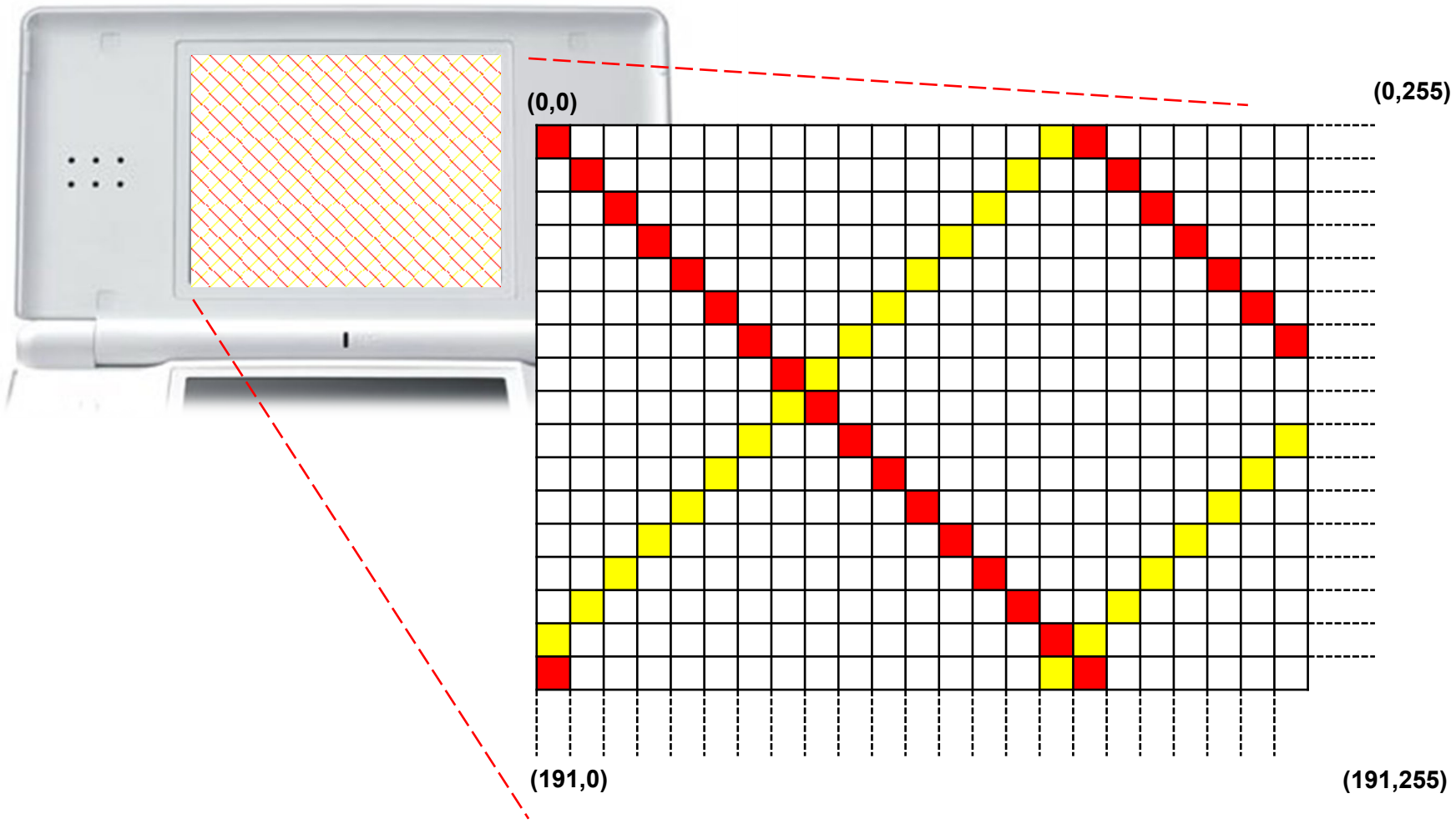


- Several function in libnds: */opt/devkitpro/libnds/include/nds/dma.h*
  - CPU configures DMA controller to make data transfer between peripherals

    *void dmaCopy (const void *source, void *dest, uint32 size)*
    - **source/dest**: Origin /Destination memory address of data to transfer
    - **size**: Data size in bytes to transfer
  - Example: copy map data in tiled mode: **uint8 myMap[256];**

```
dmaCopy(myMap, mymemory, 256);    // Copy 128 map components (16b/comp.)
```

Example: Full stripped screen with 2-color crossing lines

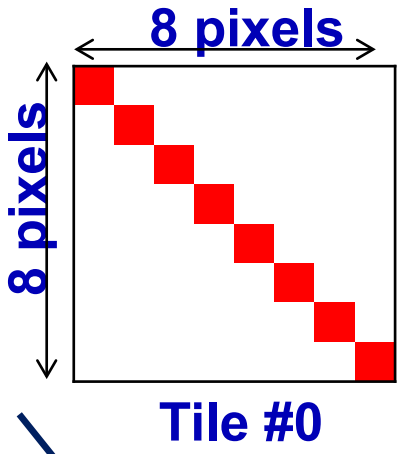


Example: Full stripped screen with 2-color crossing lines

- Define the palette and colors map of first tile: red diagonal

BG_PALETTE

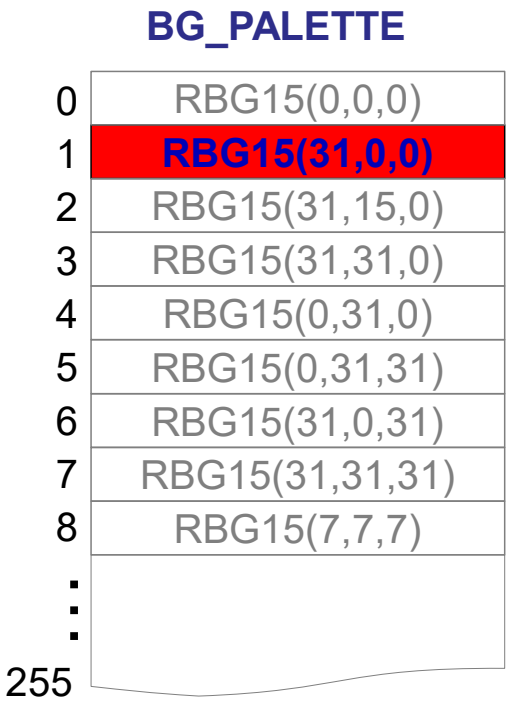
0	RBG15(0,0,0)
1	RBG15(31,0,0)
2	RBG15(31,15,0)
3	RBG15(31,31,0)
4	RBG15(0,31,0)
5	RBG15(0,31,31)
6	RBG15(31,0,31)
7	RBG15(31,31,31)
8	RBG15(7,7,7)
...	
255	

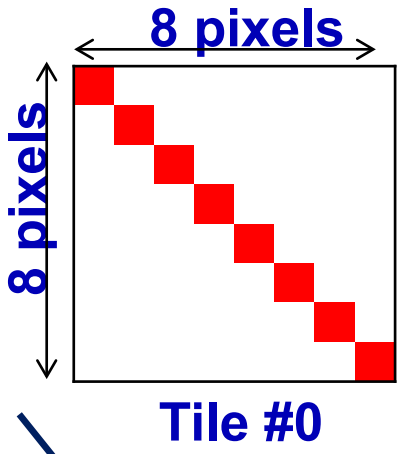


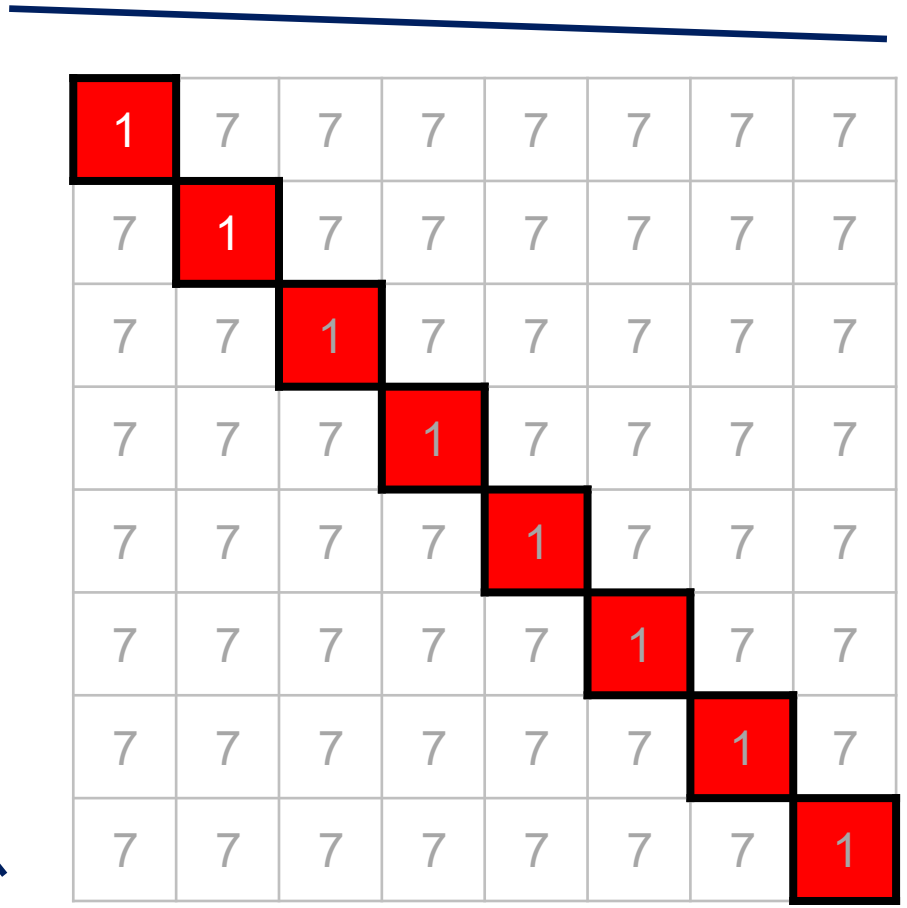
1	7	7	7	7	7	7	7
7	1	7	7	7	7	7	7
7	7	1	7	7	7	7	7
7	7	7	1	7	7	7	7
7	7	7	7	1	7	7	7
7	7	7	7	7	1	7	7
7	7	7	7	7	7	1	7
7	7	7	7	7	7	7	1

Example: Full stripped screen with 2-color crossing lines

- The elements of the diagonal take color index 1 (red)





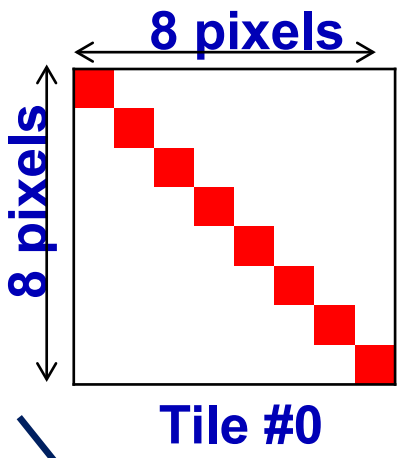


Example: Full stripped screen with 2-color crossing lines

- Assign a white color for the rest of the tiles components

BG_PALETTE

0	RBG15(0,0,0)
1	**RBG15(31,0,0)**
2	RBG15(31,15,0)
3	RBG15(31,31,0)
4	RBG15(0,31,0)
5	RBG15(0,31,31)
6	RBG15(31,0,31)
7	**RBG15(31,31,31)**
8	RBG15(7,7,7)
⋮	
255	



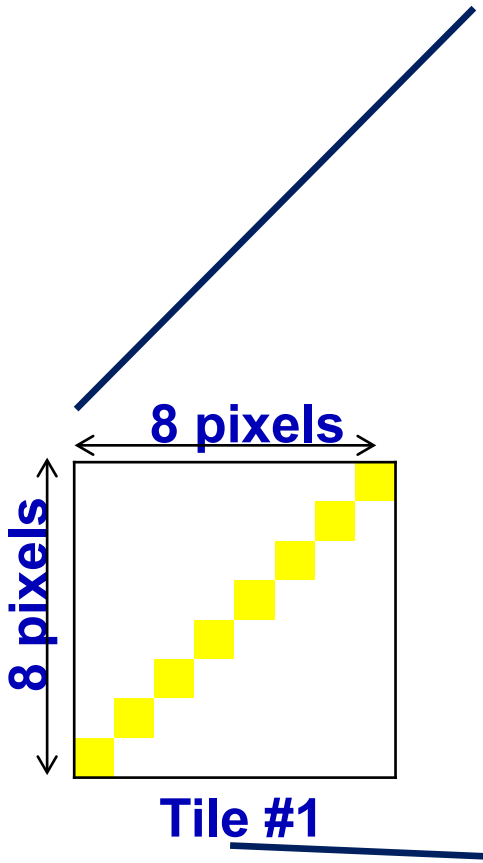
1	7	7	7	7	7	7	7
7	1	7	7	7	7	7	7
7	7	1	7	7	7	7	7
7	7	7	1	7	7	7	7
7	7	7	7	1	7	7	7
7	7	7	7	7	1	7	7
7	7	7	7	7	7	1	7
7	7	7	7	7	7	7	1

Example: Full stripped screen with 2-color crossing lines

- Define the second tile: yellow diagonal and the rest of the points with white color

BG_PALETTE

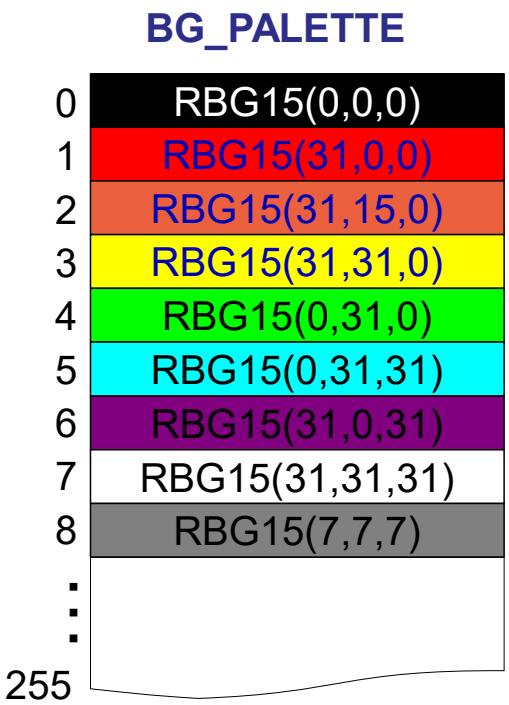
0	RBG15(0,0,0)
1	RBG15(31,0,0)
2	RBG15(31,15,0)
3	RBG15(31,31,0)
4	RBG15(0,31,0)
5	RBG15(0,31,31)
6	RBG15(31,0,31)
7	RBG15(31,31,31)
8	RBG15(7,7,7)
...	
255	

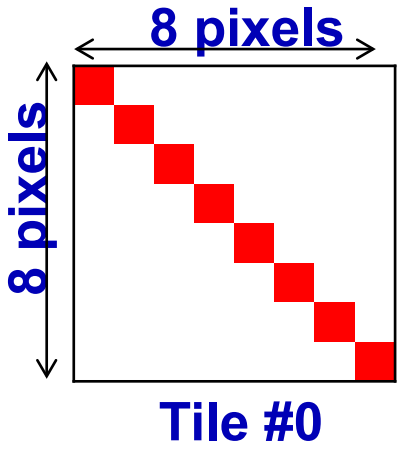


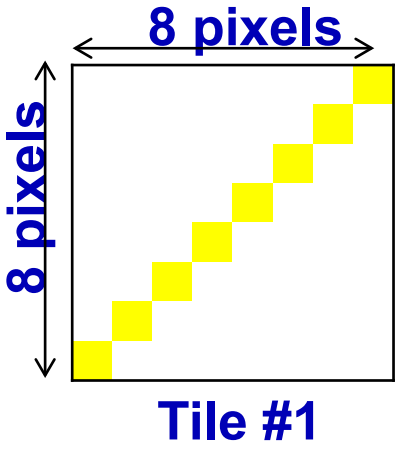
7	7	7	7	7	7	7	3
7	7	7	7	7	7	3	7
7	7	7	7	7	3	7	7
7	7	7	7	3	7	7	7
7	7	7	3	7	7	7	7
7	7	3	7	7	7	7	7
7	3	7	7	7	7	7	7
3	7	7	7	7	7	7	7

Example: Full stripped screen with 2-color crossing lines

- C declaration of the two tiles for the background





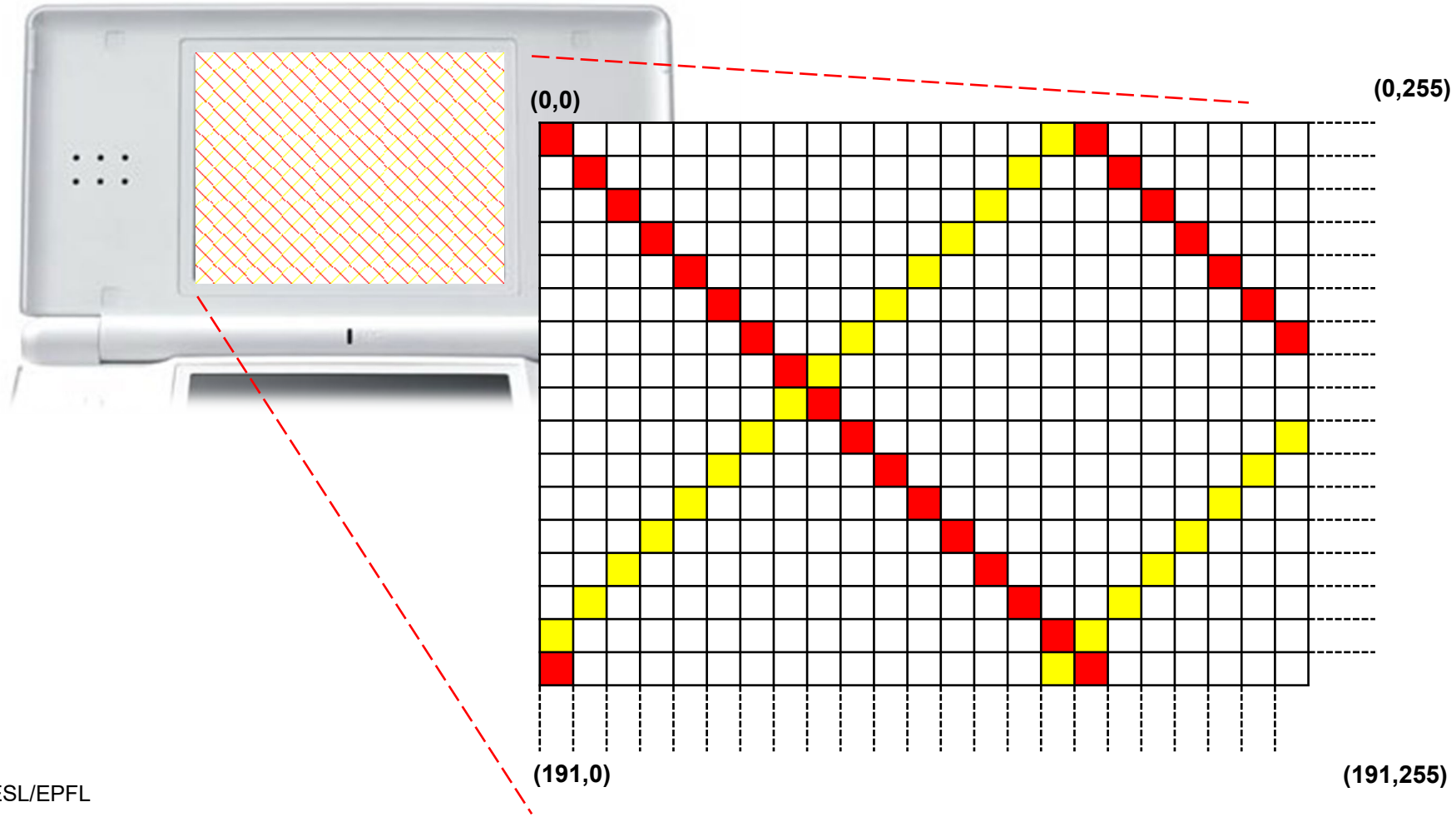


Two arrays

```
//Tile#0
u8 tile0[64] = {
    1, 7, 7, 7, 7, 7, 7, 7,
    7, 1, 7, 7, 7, 7, 7, 7,
    7, 7, 1, 7, 7, 7, 7, 7,
    7, 7, 7, 1, 7, 7, 7, 7,
    7, 7, 7, 7, 1, 7, 7, 7,
    7, 7, 7, 7, 7, 1, 7, 7,
    7, 7, 7, 7, 7, 7, 1, 7,
    7, 7, 7, 7, 7, 7, 7, 1
};

//Tile#1
u8 tile1[64] = {
    7, 7, 7, 7, 7, 7, 7, 3,
    7, 7, 7, 7, 7, 7, 3, 7,
    7, 7, 7, 7, 7, 3, 7, 7,
    7, 7, 7, 7, 3, 7, 7, 7,
    7, 7, 7, 3, 7, 7, 7, 7,
    7, 7, 3, 7, 7, 7, 7, 7,
    7, 3, 7, 7, 7, 7, 7, 7,
    3, 7, 7, 7, 7, 7, 7, 7
};
```

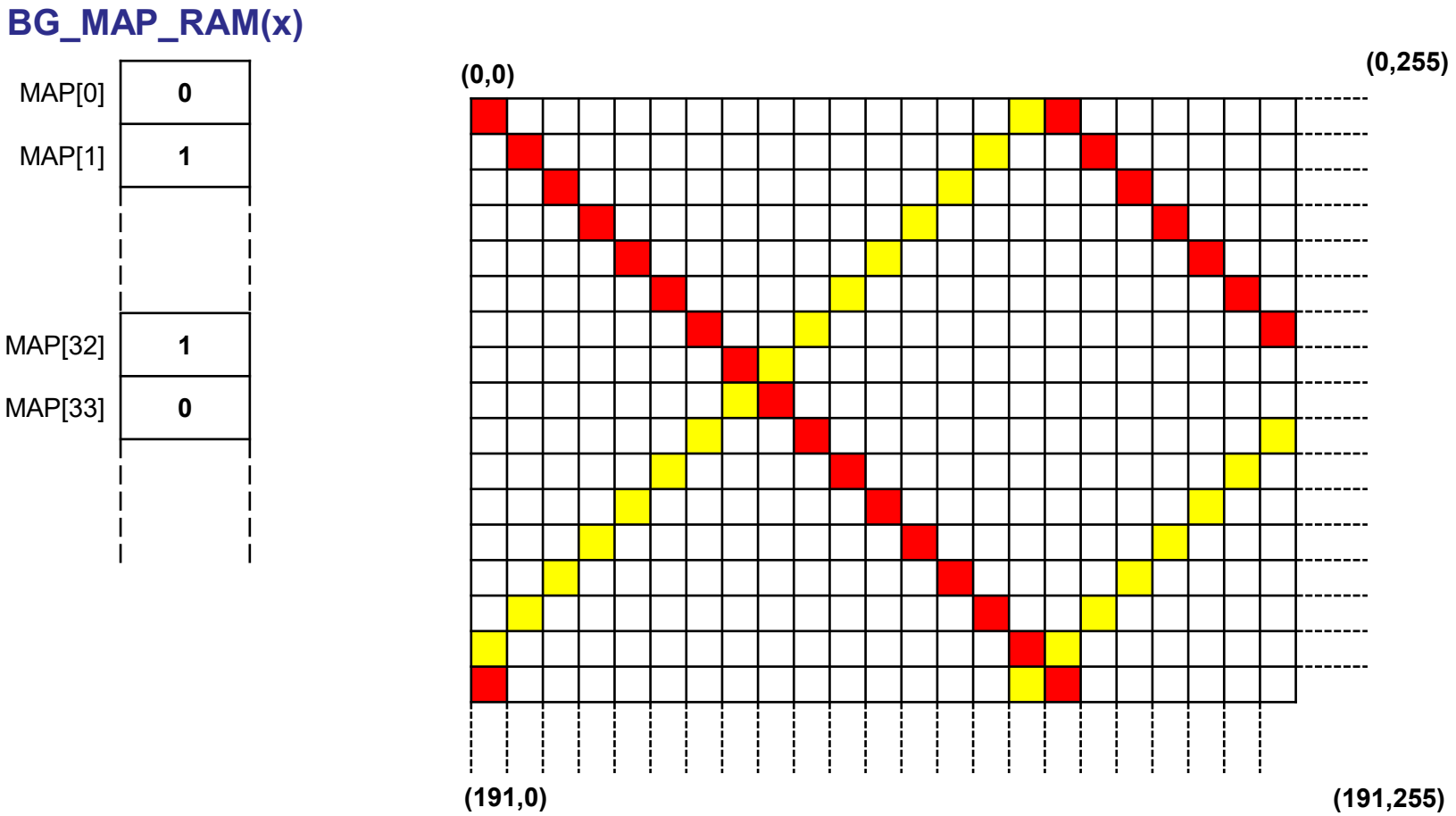
Example: Full stripped screen with 2-color crossing lines

- Define background map after creating tiles and palette



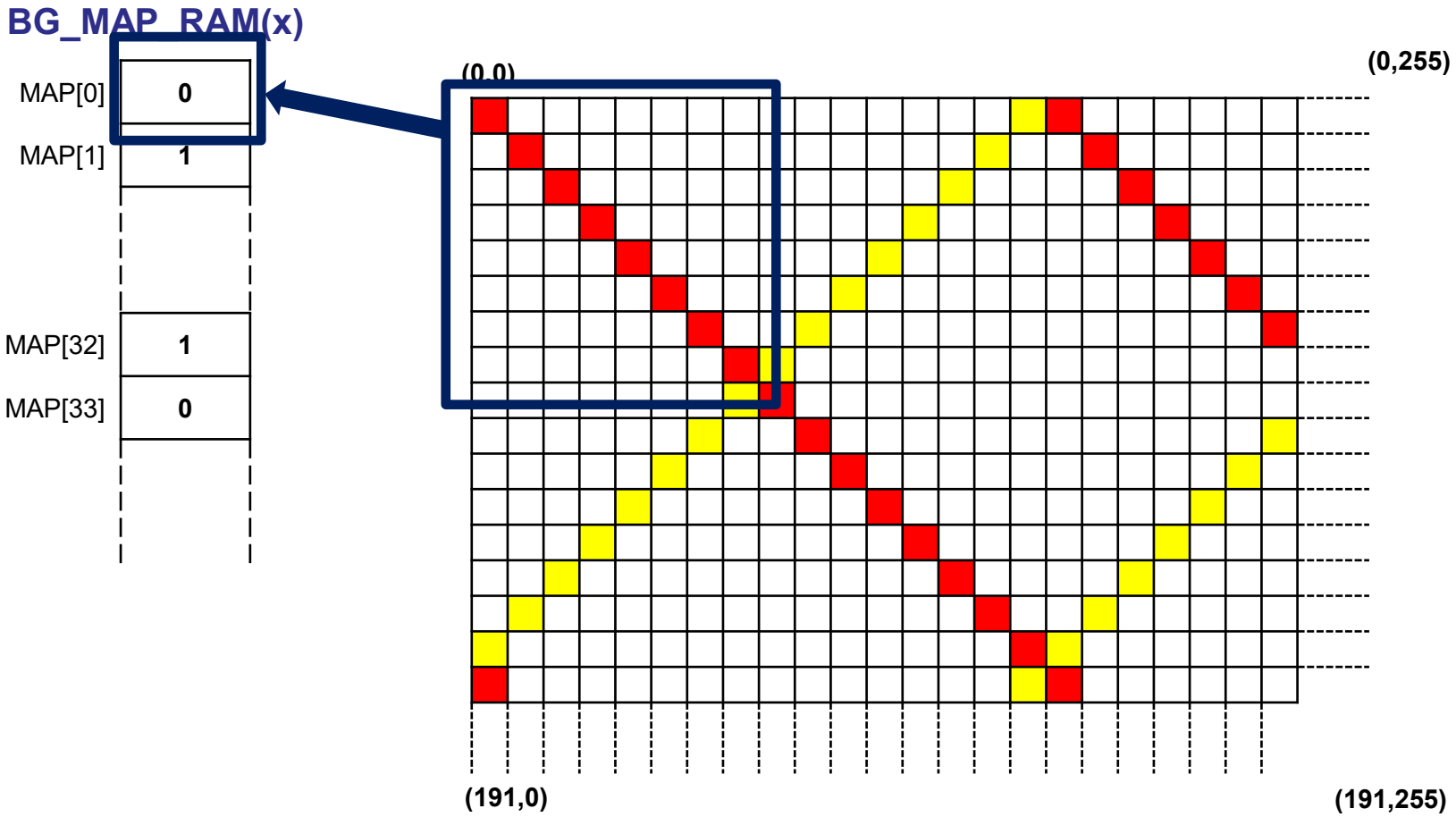
Example: Full stripped screen with 2-color crossing lines

- Define background map after creating tiles and palette



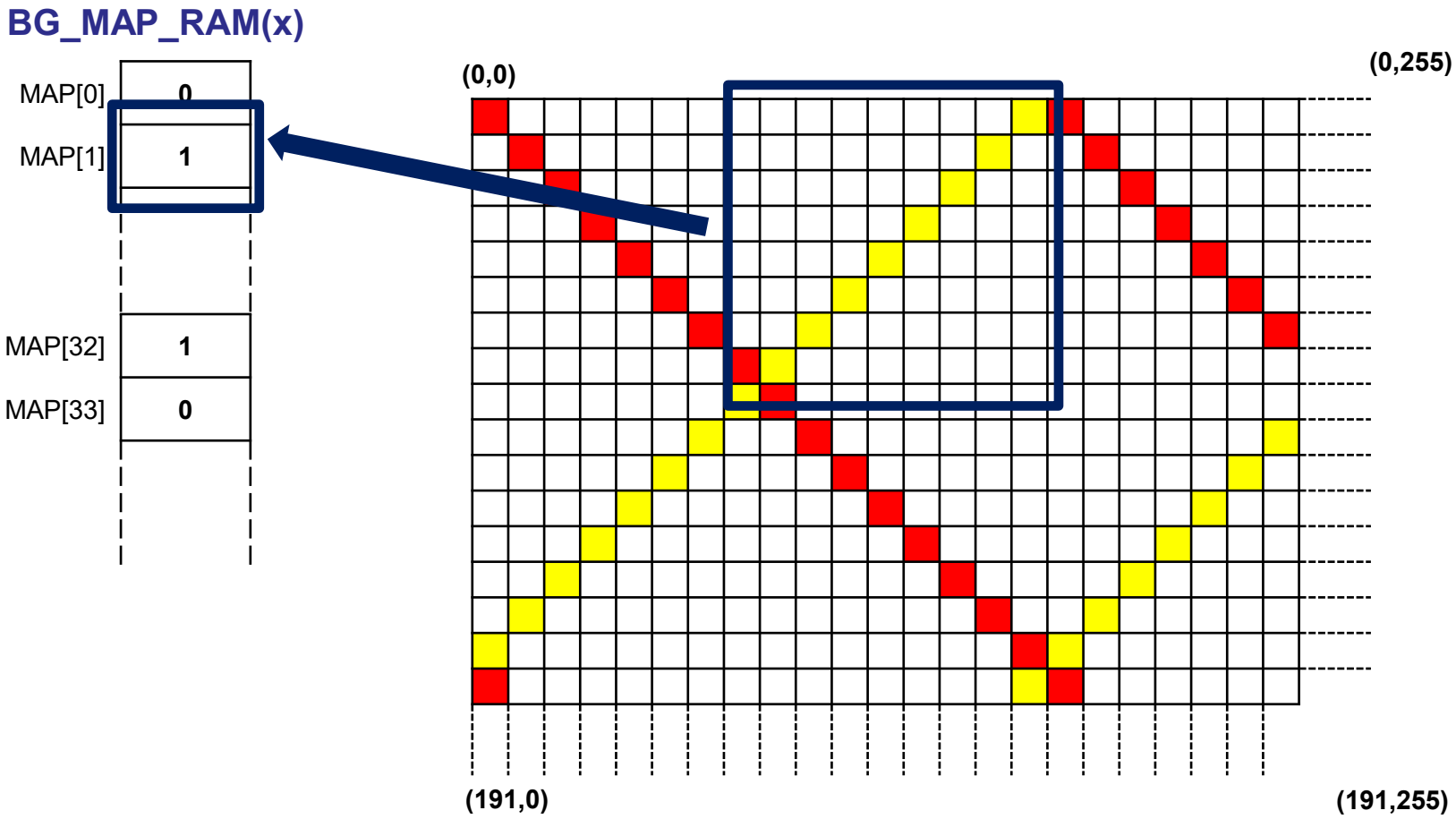
Example: Full stripped screen with 2-color crossing lines

- Define background map after creating tiles and palette



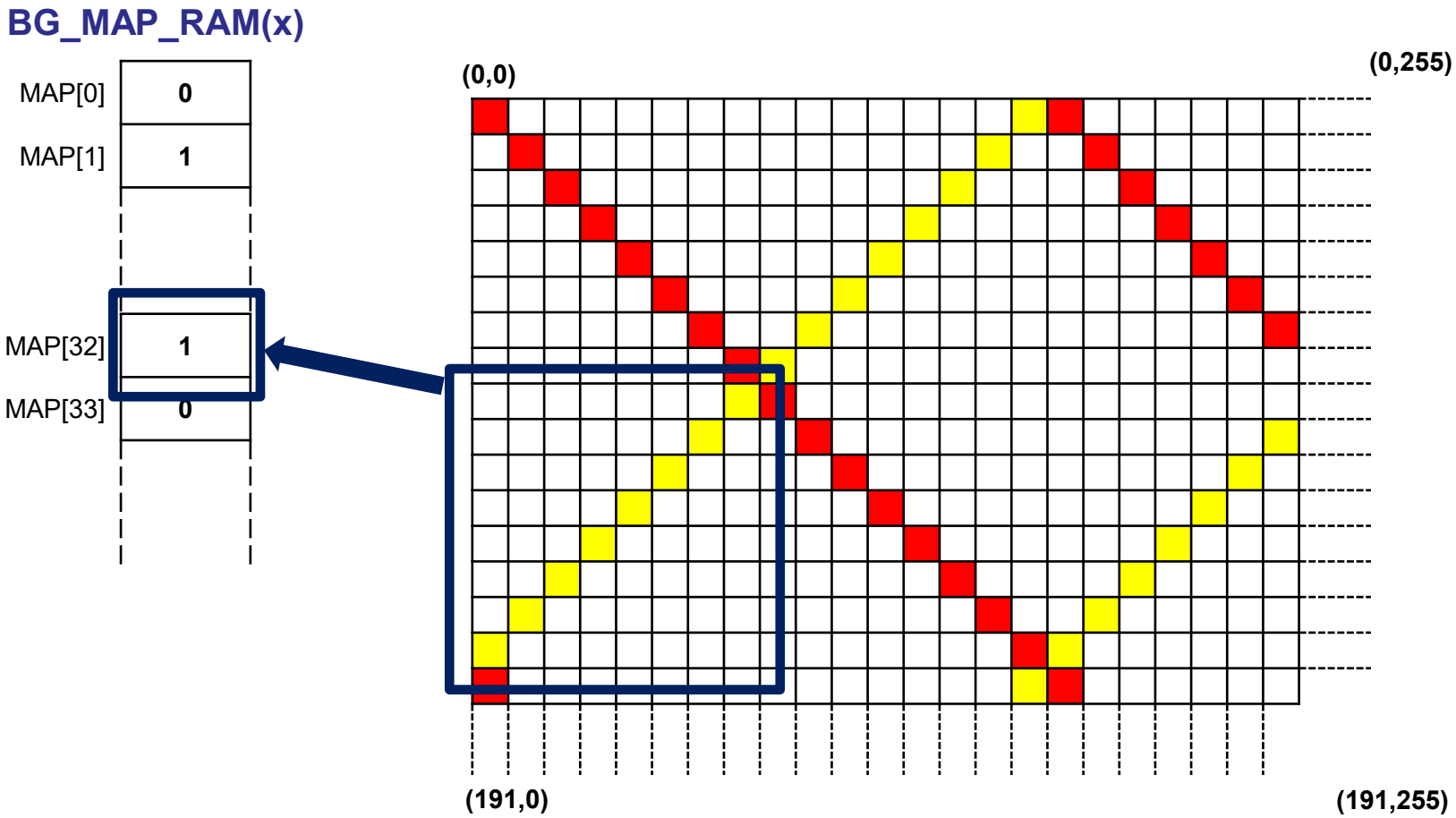
Example: Full stripped screen with 2-color crossing lines

- Define background map after creating tiles and palette



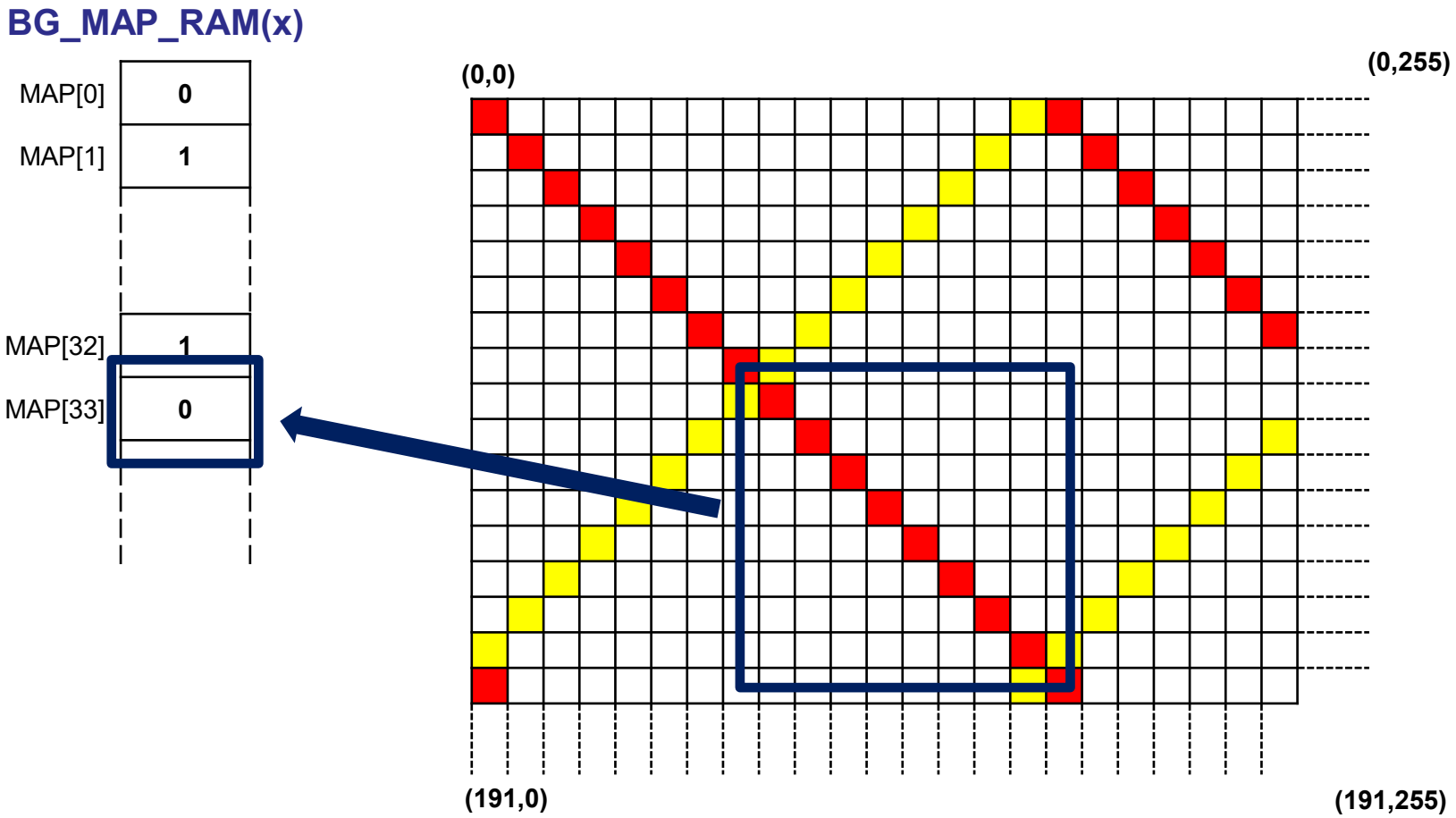
Example: Full stripped screen with 2-color crossing lines

- Define background map after creating tiles and palette



Example: Full stripped screen with 2-color crossing lines

- Define background map after creating tiles and palette



Example: Full stripped screen with 2-color crossing lines

- Four steps to configure engine with initialized tiled background

1. Configure the engine in Mode 0 and the VRAM block
2. Use Background 0
3. Transfer the tiles to the Tile memory using a DMA transfer
4. Create the map by alternating the tiles and the lines to create the grid

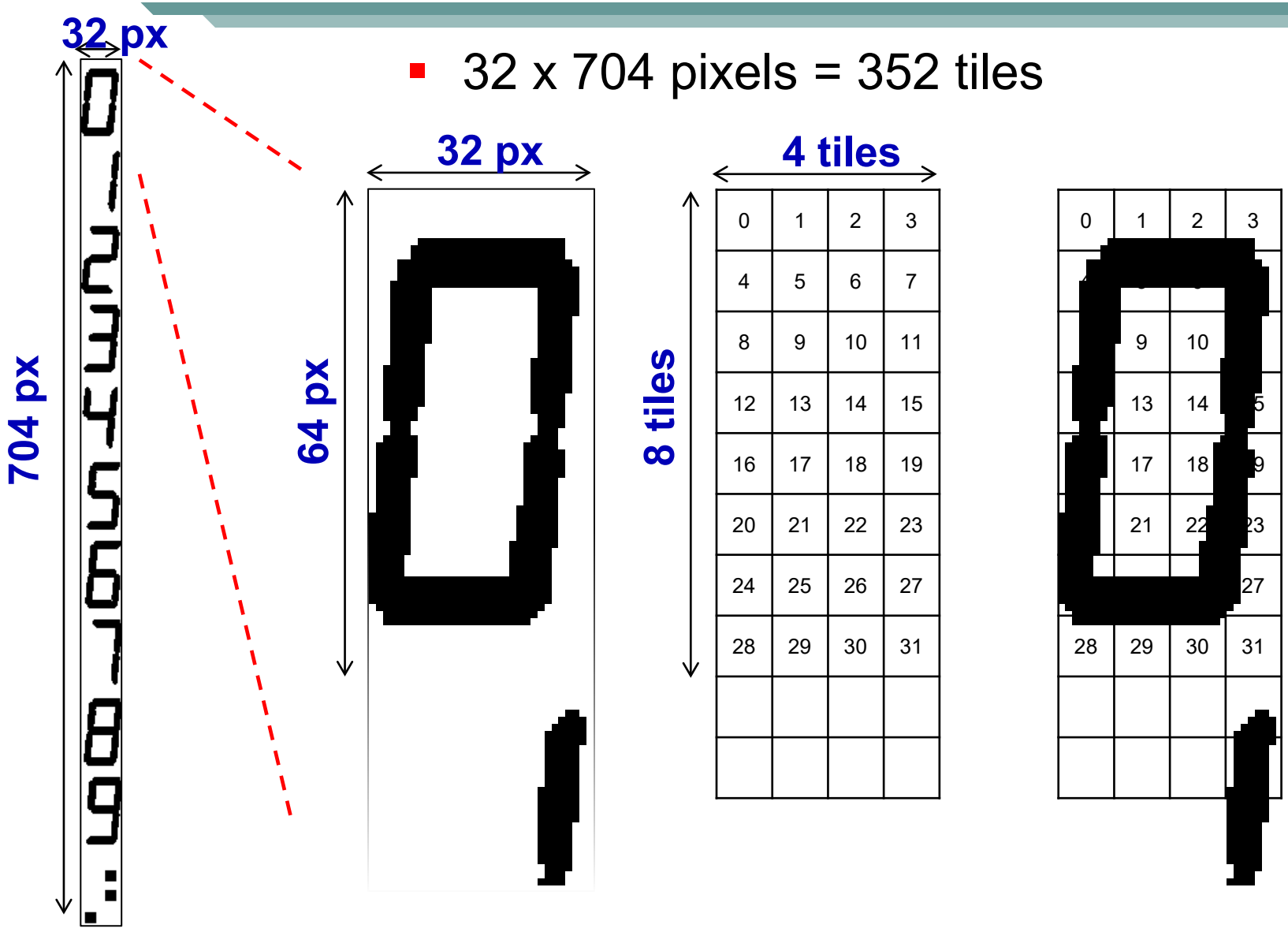
```
//Configure the Engine and the VRAM bank
REG_DISPCNT = MODE_0_2D | DISPLAY_BG0_ACTIVE;
VRAM_A_CR = VRAM_ENABLE | VRAM_A_MAIN_BG;

//Configure the activated background BG0
BGCTRL[0] = BG_32x32 | BG_COLOR_256
        | BG_MAP_BASE(1) | BG_TILE_BASE(0);

//Copy the tiles
dmaCopy(tile0, &BG_TILE_RAM(0)[0], 64);
dmaCopy(tile1, &BG_TILE_RAM(0)[32], 64);

//Generate the map
for(i = 0;i<32; i+=2)
    for(j=0; j<32; j+=2)
    {
        BG_MAP_RAM(1)[32*(i)+j] = 0;
        BG_MAP_RAM(1)[32*(i+1)+j] = 1;
        BG_MAP_RAM(1)[32*(i)+j+1] = 1;
        BG_MAP_RAM(1)[32*(i+1)+j+1] = 0;
    }
```

Creating images with tiles using «grit»

- grit can also transform an image into a tiles input for NDS
  - -gt : option used to generate tiles (it can be combined with other options)
  - Recommended to generate the palette (option –p)
- The image to transform should have sizes multiple of 8
- The map can be generated if the full image is going to be displayed (option –m)
- The tiles, the palette and/or the map will be created in an assembly file and will be declared in a header file
  - Link with the C code

Creating tiles using «grit»: Example

- 32 x 704 pixels = 352 tiles



NDS screen modes: 2D engines configuration
Using multiple backgrounds

- Each engine has four backgrounds (or layers): BG0, BG1, BG2 and BG3
  - Final view on the screen will be their combination based on used graphic mode
  - Typically BG0 is the most external one and BG3 is the most internal one
- Multiple modes possible in 2D engines, select the most appropriate one
  - Main engine: 7 modes and framebuffer

Mode	BG0	BG1	BG2	BG3
0	Tiled/3D	Tiled	Tiled	Tiled
1	Tiled/3D	Tiled	Tiled	Rotoscale
2	Tiled/3D	Tiled	Rotoscale	Rotoscale
3	Tiled/3D	Tiled	Tiled	Ext. Rotoscale
4	Tiled/3D	Tiled	Rotoscale	Ext. Rotoscale
5	Tiled/3D	Tiled	Ext. Rotoscale	Ext. Rotoscale
6	3D	N/A	Large Bitmap	N/A
FrameBuf.	Direct VRAM display as a bitmap			

  - Sub engine: 6 modes

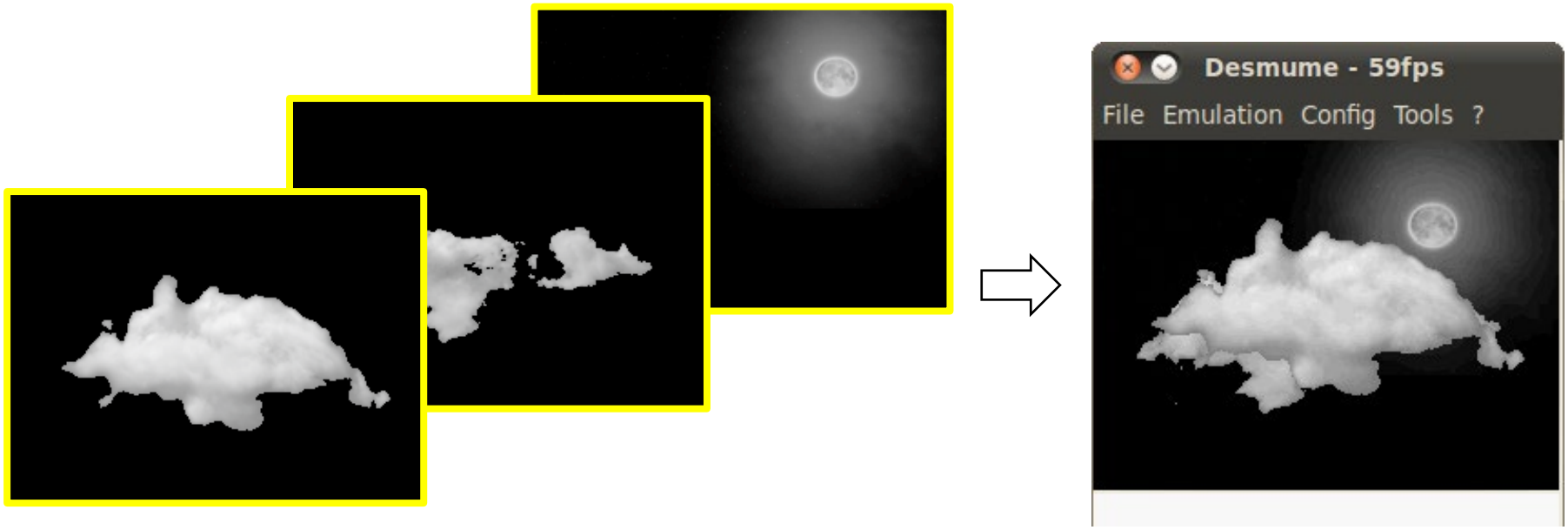
Mode	BG0	BG1	BG2	BG3
0	Tiled	Tiled	Tiled	Tiled
1	Tiled	Tiled	Tiled	Rotoscale
2	Tiled	Tiled	Rotoscale	Rotoscale
3	Tiled	Tiled	Tiled	Ext. Rotoscale
4	Tiled	Tiled	Rotoscale	Ext. Rotoscale
5	Tiled	Tiled	Ext. Rotoscale	Ext. Rotoscale

- How to use tiled mode and extended rotoscale backgrounds in one mode?
  - Tiled mode always possible in BG0
  - Configure REG_DISPCNT with one of the possible modes: 3, 4 and 5 (Extended)
  - Example: activate mode 5 and background 1 (BG1) and 3 (BG3);

```
REG_DISPCNT = MODE_5_2D | DISPLAY_BG1_ACTIVE | DISPLAY_BG3_ACTIVE;
```

Use of multiple backgrounds concurrently

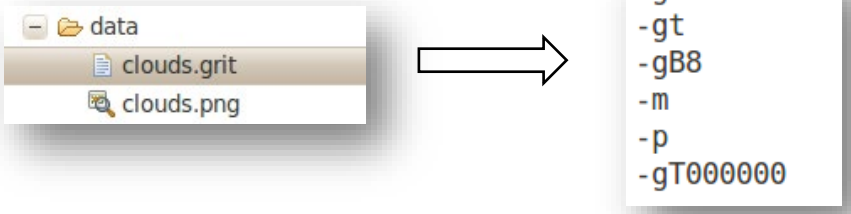
- Up to 4 backgrounds can be used at a time in each engine
  - Superposition of backgrounds from BG3 to BG0
- One color can be set as transparent in the palette
  - In the next example the black color is set to transparent



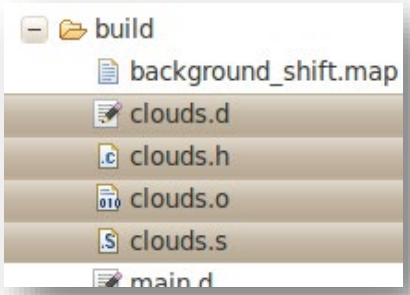
- Background X can be shifted using **write-only** registers REG_BGXHOFS and REG_BGXVFS (where X is the number of the background

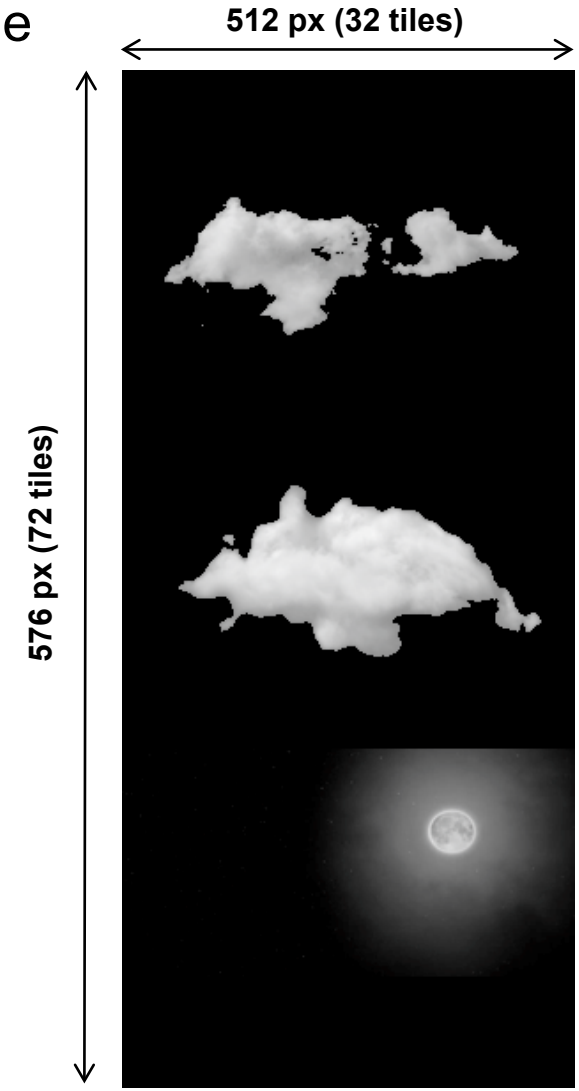
Example: Use of overlapping backgrounds Grit configuration

- A single image (“clouds.png”) combining all the backgrounds has been used
  - Multiple images can also be used instead
- Configuration of grit to use tiled mode
  - 4bit pixel depth can also be used
  - Black color is set as ‘transparent’ color (-gT flag)



- Grit creates the necessary files storing the palette, the map and the tiles





Example: Use of overlapping backgrounds Backgrounds configuration

- Configuration of VRAM bank, engine and background registers
  - BG0, BG1 and BG2 are activated using Mode 0 (all tiled mode)
  - Different parts of the same image are used to render the different backgrounds
    → Different map bases for each background but shared tile base

```
// Activate main engine and backgrounds 0, 1 and 2 in standard tiled mode
VRAM_A_CR = VRAM_ENABLE | VRAM_A_MAIN_BG;
REG_DISPCNT = MODE_0_2D | DISPLAY_BG0_ACTIVE | DISPLAY_BG1_ACTIVE | DISPLAY_BG2_ACTIVE;
BGCTRL[0] = BG_COLOR_256 | BG_MAP_BASE(0) | BG_TILE_BASE(1) | BG_32x32;
BGCTRL[1] = BG_COLOR_256 | BG_MAP_BASE(1) | BG_TILE_BASE(1) | BG_32x32;
BGCTRL[2] = BG_COLOR_256 | BG_MAP_BASE(2) | BG_TILE_BASE(1) | BG_32x32;
```

- Transfer grit information to the proper locations
  - Palette and tiles are shared by the 3 backgrounds
  - Maps are different for each of them → select different part of the map

```
//Copy tiles and palette (shared by all backgrounds in this case)
swiCopy(cloudsTiles, BG_TILE_RAM(1), cloudsTilesLen/2);
swiCopy(cloudsPal, BG_PALETTE, cloudsPalLen/2);
//Copy maps (32x24 components of 2 bytes for each of the map bases)
swiCopy(&cloudsMap[0], BG_MAP_RAM(1), 32*24);
swiCopy(&cloudsMap[32*24], BG_MAP_RAM(0), 32*24);
swiCopy(&cloudsMap[32*48], BG_MAP_RAM(2), 32*24);
```

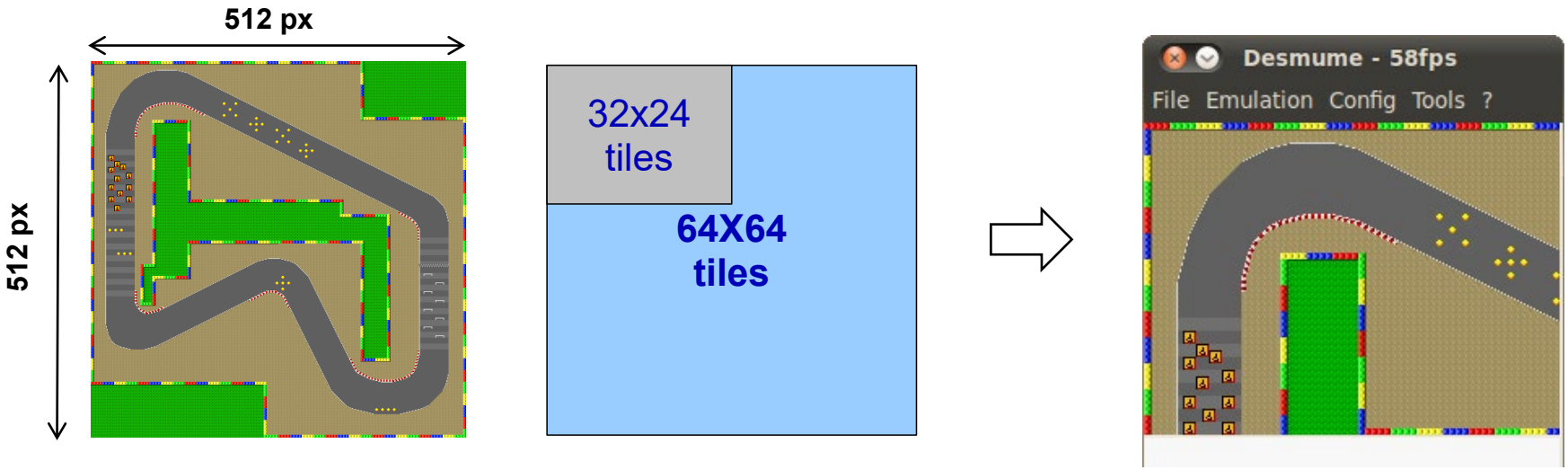
Example: Use of overlapping backgrounds Shifting backgrounds

- Backgrounds are shifted horizontally creating an effect of moving clouds
  - Shift registers are not readable → need of a variable to track previous assigned value
  - BG0 is shifted to the right decreasing by one pixel the horizontal shift register
  - BG1 is shifted to the left increasing by one pixel the horizontal shift register
  - BG2 is still (the moon does not move)

```
//Local variables to track the shifting
int bg0 = 0, bg1 = 0;

//Shifting background
while(1) {
    //Assign shift registers (they are not readable!)
    REG_BG0HOFS = bg0;
    REG_BG1HOFS = bg1;
    //Update local variables that track the shifting
    if(--bg0 < 0) bg0 = 255;
    if(++bg1 > 255) bg1 = 0;
    swiWaitForVBlank();
}
```

Use of Big Maps in Tiled Mode

- Screen size is limited to 32x24 tiles (256x192 pixels)
  - Screen is only the visible part of the map stored in memory
  - Normally 32x32 maps are used in tiled mode
- Different map sizes are available
  - 32x32, 32x64, 64x32 and 64x64

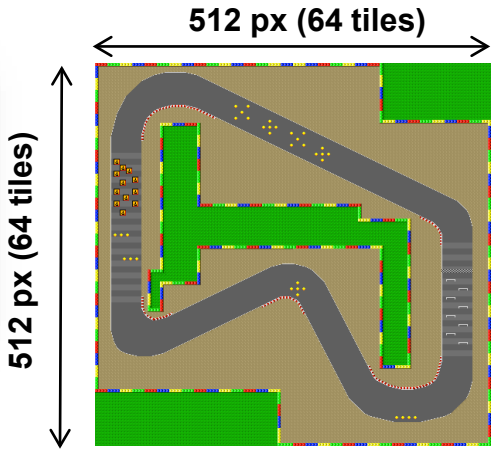


- Background can be rotated changing the visible part of the map
  - Horizontal and Vertical Shifting using rotation registers

Example: Use of Big Maps in Tiled Mode
Grit file and transfer of palette and tiles

- A big image (“background.png”) has been used in this project
  - Grit is configured in the standard way in order to generate the tiles, maps and palette and the corresponding files are generated

```
-g
-gt
-gB8
-m
-p
```



- Engine, VRAM and background configuration
  - 64x64 grid size instead of standard 32x32
  - Since the image is 4x bigger, a block of 4 consecutive map bases are necessary
    → only the first one is set in the background control register

```
// Activate main engine and background 0 in tiled mode using 64x64 map
VRAM_A_CR = VRAM_ENABLE | VRAM_A_MAIN_BG;
REG_DISPCNT = MODE_5_2D | DISPLAY_BG0_ACTIVE;
BGCTRL[0] = BG_COLOR_256 | BG_MAP_BASE(0) | BG_TILE_BASE(1) | BG_64x64;
```

- Tiles and palette transfer is done in the standard way

```
// Copy tiles and palette to the corresponding place
swiCopy(backgroundTiles, BG_TILE_RAM(1), backgroundTilesLen/2);
swiCopy(backgroundPal, BG_PALETTE, backgroundPalLen/2);
```

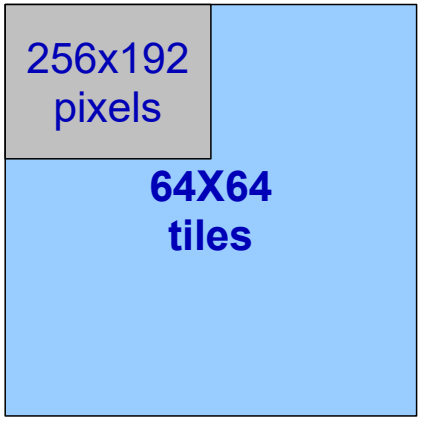
Example: Use of Big Maps in Tiled Mode Transfer map to map bases

- A 64x64 tiles map requires more space in the VRAM memory
  - 64x64 tiles * 2 Bytes/tile = 8 KB → **4 Map bases of 2 KB**
  - Placement in memory is not linear! → Divided in quadrants

```
// TOP LEFT quadrant of the image in first map base
for(i=0; i<32; i++)
    dmaCopy(&backgroundMap[i*64], &BG_MAP_RAM(0)[i*32], 64);

// TOP RIGHT quadrant of the image in second map base
for(i=0; i<32; i++)
    dmaCopy(&backgroundMap[i*64+32], &BG_MAP_RAM(1)[i*32], 64);

// BOTTOM LEFT quadrant of the image in third map base
for(i=0; i<32; i++)
    dmaCopy(&backgroundMap[(i+32)*64], &BG_MAP_RAM(2)[i*32], 64);

// BOTTOM RIGHT quadrant of the image in fourth map base
for(i=0; i<32; i++)
    dmaCopy(&backgroundMap[(i+32)*64+32], &BG_MAP_RAM(3)[i*32], 64);
```

Example: Use of Big Maps in Tiled Mode Shifting the background

- The visible part of the background (256x192 pixels) is limited by the screen size and depends on its horizontal and vertical shift
  - Shift background in order to “move” the visible part in the




```
//Shift background
while(1) {
    //shifting horizontally from left to right
    for(i=0; i<=512-256; i++){
        swiWaitForVBlank();
        REG_BG0HOFS = i;
    }
    //shifting vertically from up to down
    for(i=0; i<=512-192; i++){
        swiWaitForVBlank();
        REG_BG0VOFS = i;
    }
    //shifting horizontally from right to left
    for(i=512-256; i>=0; i--){
        swiWaitForVBlank();
        REG_BG0HOFS = i;
    }
    //shifting vertically from down to up
    for(i=512-192; i>=0; i--){
        swiWaitForVBlank();
        REG_BG0VOFS = i;
    }
}
```

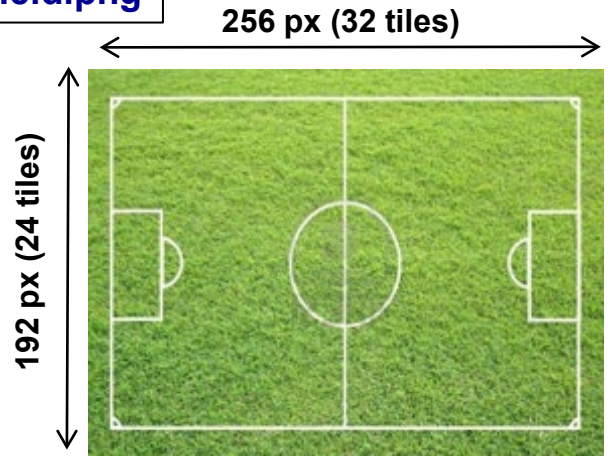
Advance graphics: Sprites

- Sprites are small graphic objects that can be rendered on top of the backgrounds and provide extended features
  - Different sizes (8x8, 16x16, 32x32, 64x64, 16x8, 32x8, 32x16, 64x32, 8x32, 16x32)
  - Free position: they can be rendered in any point of the screen (even outside)
  - Transformation capabilities: they can be rotated, scaled or flipped in 2D
  - Different modes (bitmap, tiled…)
- Graphic engine features and limitations
  - The amount of sprites is fixed to 128 and they are hidden by default
  - A special mode has to be configured in a VRAM mode: this bank cannot be used for anything else (e.g.: for backgrounds)
  - A sprite is associated to a graphic stored in memory and several sprites can be associated to the same graphic (i.e.: we can have several sprites with a single graphic loaded in memory)
  - They use a special palette (different from the one used for backgrounds) and it can use extended palettes
- libNDS support through means of an API
  - See more information in file **/opt/devkitPro/libnds/include/nds/arm9/sprite.h**
  - Some examples can be found at **http://sourceforge.net/projects/devkitpro/files/examples/nds/**

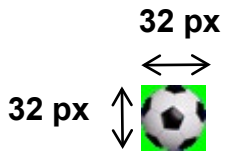
Example: Simple Sprite Usage grit configuration

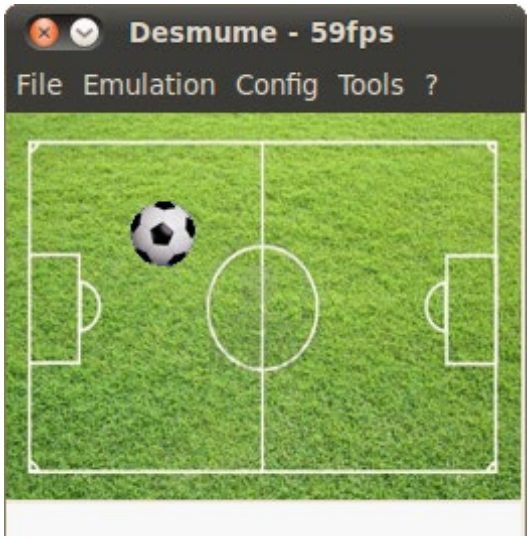
- The sprite of a ball (image “ball.png”) will be displayed over a background of a football field (image “field.png”)

field.png



ball.png





- Grit has been configured to generate the tiled version of both images
  - Green has been selected as transparent color in the ball graphic (-gT flag)

field.grit

```
-g
-gt
-gB8
-m
-p
```

ball.grit

```
-g
-gt
-gB8
-gT00FF00
```

Example: Simple Sprite Usage
Engine and background configuration

- Engine is configured normally in mode 0 activating one background
  - Any other configuration could be possible

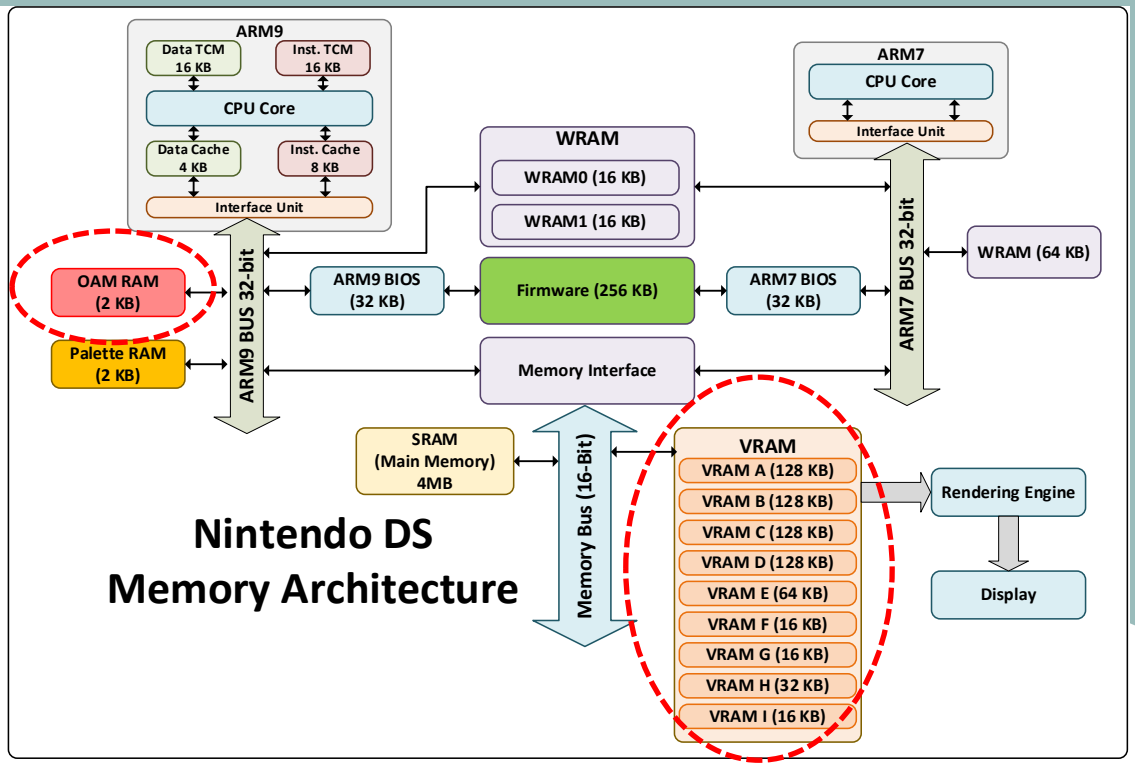
```
//MAIN engine
REG_DISPCNT = MODE_0_2D | DISPLAY_BG0_ACTIVE;
```

- Background is configured in the standard way
  - VRAM bank A enabled in background mode
  - Tiled mode using 32x32 grid and 256 colors

```
void configureBG0() {
    //Activate and configure VRAM bank to work in background mode
    VRAM_A_CR = VRAM_ENABLE | VRAM_A_MAIN_BG;

    //BG0 configuration for the background
    BGCTRL[0] = BG_COLOR_256 | BG_MAP_BASE(0) | BG_TILE_BASE(1) | BG_32x32;

    //Copy data to display background (tiles, palette and map)
    swiCopy(fieldTiles, BG_TILE_RAM(1), fieldTilesLen/2);
    swiCopy(fieldPal, BG_PALETTE, fieldPalLen/2);
    swiCopy(fieldMap, BG_MAP_RAM(0), fieldMapLen/2);
}
```

Nintendo DS memory range: Storage of sprites in video mode data

- Dedicated fast memories for caching
  - Object Attribute Memory (OAM)
    - Sprites or 2D images animations integrated into larger scene



- Background memory: VRAM portions with pixels and palettes
  - Enable the necessary VRAM banks: **VRAM_?_CR**
    - Example: store sprites in VRAM B, OAM used for caching the visible ones **few KBs needed for all the sprites (limit of 128 sprites)**: enough with one bank (*bank B*), assign to *main 2D core for sprites*

```
VRAM_B_CR = VRAM_ENABLE | VRAM_B_MAIN_SPRITE_0x06400000;
```

Example: Simple Sprite Usage
Sprite configuration

- Sprite configuration
  - VRAM bank B initialization in sprite mode and offset (since A is also used)
  - Sprite manager initialization: constant **oamMain** is declared in libnds as the sprite manager of the system
  - Allocate space for the graphic (size and format necessary)
  - Copy graphic bitmap into the allocated space and palette in the sprite palette

```
void configureSprites() {
    //Set up memory bank to work in sprite mode (offset since we are using VRAM A for backgrounds)
    VRAM_B_CR = VRAM_ENABLE | VRAM_B_MAIN_SPRITE_0x06400000;

    //Initialize sprite manager and the engine
    oamInit(&oamMain, SpriteMapping_1D_32, false);

    //Allocate space for the graphic to show in the sprite
    gfx = oamAllocateGfx(&oamMain, SpriteSize_32x32, SpriteColorFormat_256Color);

    //Copy data for the graphic (palette and bitmap)
    swiCopy(ballPal, SPRITE_PALETTE, ballPalLen/2);
    swiCopy(ballTiles, gfx, ballTilesLen/2);
}
```

- **Important:** The sprite manager has been initialized and the graphic loaded in memory but the sprite has not been displayed yet!

Example: Simple Sprite Usage
Sprite configuration

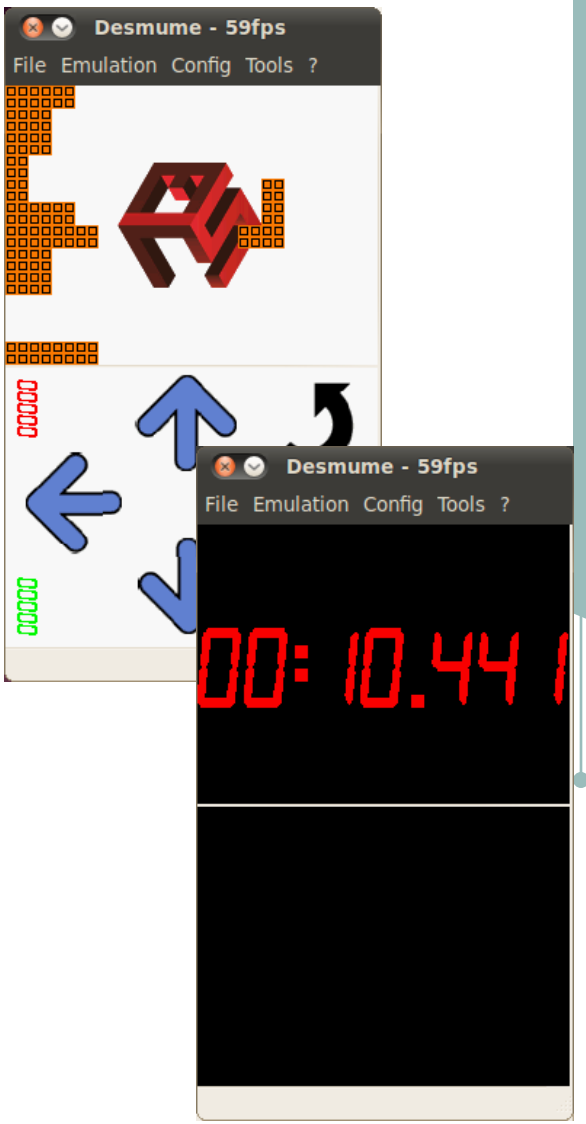
- Keys tracking to modify coordinates
  - **x** and **y** represent the position of the top left corner of the ball and are modified within the screen
- Sprite set-up:
  - coordinates
  - size
  - format
  - associated graphic
  - rotation
  - …
- Refresh screen
  - Renders all the not hidden sprites

```
//Position
int x = 0, y = 0, keys;
while(1){
    //Read held keys
    scanKeys();
    keys = keysHeld();

    //Modify position of the sprite accordingly
    if((keys & KEY_RIGHT) && (x < (SCREEN_WIDTH - SPRITE_WIDTH))) x+=2;
    if((keys & KEY_DOWN) && (y < (SCREEN_HEIGHT - SPRITE_HEIGHT))) y+=2;
    if((keys & KEY_LEFT) && (x  > 0)) x-=2;
    if((keys & KEY_UP) && (y  > 0)) y-=2;

    oamSet(&oamMain,    // oam handler
        0,              // Number of sprite
        x, y,           // Coordinates
        0,              // Priority
        0,              // Palette to use
        SpriteSize_32x32,           // Sprite size
        SpriteColorFormat_256Color, // Color format
        gfx,            // Loaded graphic to display
        -1,             // Affine rotation to use (-1 none)
        false,          // Double size if rotating
        false,          // Hide this sprite
        false, false,   // Horizontal or vertical flip
        false           // Mosaic
        );
    swiWaitForVBlank();
    //Update the sprites
    oamUpdate(&oamMain);
}
```

Practical Work 8: Graphics (Part 3): Rotoscale and tiled mode

- Exercises (and homework)
  - Exercise 1 – Configure the engine
  - Exercise 2 – Load the background image in the main engine
  - Exercise 3 – Create user-define tiles for the Tetris blocks
  - Exercise 4 – Use multiple backgrounds
  - Exercise 5 – Plot background image using tiles created with grit
  - Exercise 6 - Sprites
  - *Exercise 7 – Chrono display: Plot a number
  - *Exercise 8 – Chrono display: Plot the value of the chrono
  - *Exercise 9 – Chrono display: Change the color of the chrono display

* Additional exercises



Questions?

Let’s use the screen with tiles and multi-background in the NDS!