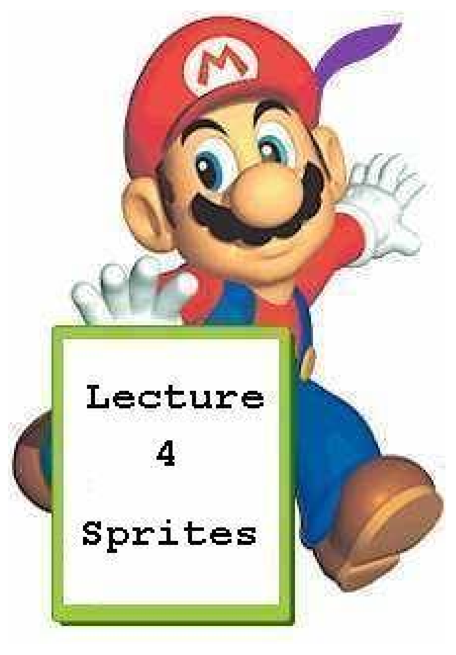
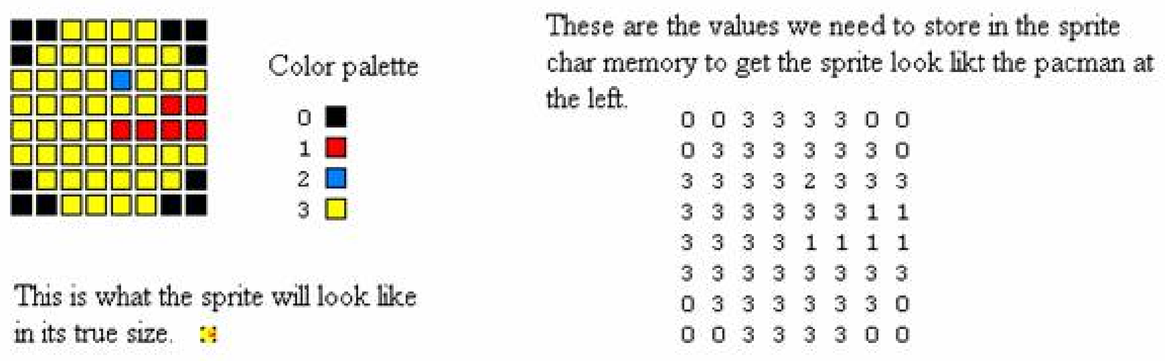


Sprites

As said earlier, a sprite is a movable character that doesn’t destroy the background when it’s moving. On the gba we can create sprites with 12 different sizes, ranging from 8x8 pixels to 64x64 pixels. We can have a total of 256 different colors for the different sprites we are making. This can be done either by making 16 palettes with 16 colors each, or making one palette with 256 colors. Each color takes as we know 2 bytes (16 bits) in memory; when we have 256 colors that makes a total of 256 colors * 2 bytes for each color = 512 bytes. Or for 16 palettes with 16 colors each, it will be 16 palettes * 16 colors * 2 bytes each color = 512 bytes. As you can see, it consumes just as much palette memory if we use 1 or 16 palettes. The memory area from 0x5000200-0x50003FF is used for storing sprite palette(s). When we are going to make sprites we always start with a 8x8 sprite. The names tile, sprite data and OBJ tile will all be used in this lesson. All of these names actually means the same thing: a character consisting of 8x8 pixels like the one in the picture under this text; these ‘tiles’ can be stored as 32 bytes (16 colors) or 64 bytes (256 colors). Tiles were also talked about briefly in lesson 1, as something you use to make backgrounds in mode 0, 1 and 2. The reason for using the word ‘tiles’ when we’re talking about sprites is that they are made up from tiles of 8x8 pixels. Sprites are usually made of several tiles; it is important to remember that there are two different memory areas for storing sprite tiles and background tiles, and that they use separate palettes.



This is what the sprite will look like in its true size.

These are the values we need to store in the sprite char memory to get the sprite look likt the pacman at the left.

```
0 0 3 3 3 3 0 0
0 3 3 3 3 3 3 0
3 3 3 3 2 3 3 3
3 3 3 3 3 3 1 1
3 3 3 3 1 1 1 1
3 3 3 3 3 3 3 3
0 3 3 3 3 3 3 0
0 0 3 3 3 3 0 0
```

Memory and sprites

There are two memory areas that are important to understand when you’re making sprites on the gba. The first is called OAM (Object Attribute Memory). And the second consists of sprite data information (OBJ tiles).

In OAM most of the information about the sprites is defined: things like size, shape, x and y coordinates and so on. OAM is a linear table with information that starts at 0x7000000 and ends at 070003FF for a total of 1KB. It divides the 1KB into 8 bytes per sprite, for a total of 128 sprites in OAM at the same time, which should be enough for anyone.

The most common method to define OAM is to make a structure.

```
typedef struct tagOAMEntry
{
        u16 attribute0;
        u16 attribute1;
        u16 attribute2;
        u16 attribute3;
}OAMEntry, *pOAMEntry;
```

Here we have four 16bit variables that we have called attribute0–3. Each of these contains two bytes; in this way we store the 8 bytes of OAM info that we need to show a sprite. This structure gives you access to a copy of OAM, and lets you copy this over to the real OAM memory each time the screen is drawn. The reason you have to wait till the screen is drawn to load the OAM info into memory is that otherwise you run the risk that only part of the sprite you want to draw is already drawn, so that the screen flickers.

Attribute : 0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Shape		C	M	Alpha		SD	R	Y-coordinate							

Bits 0-7 Y-coordinate for the sprite. The value can be between 0 and 255. (The screen is only 0-159 in y direction.)
Bit 8 Set sprite rotation.
Bit 9 Set double size. Is used when rotating sprites, to prevent parts of the sprite from disappearing when rotating.
Bits10-11 Set alpha-blending.
Bit 12 Set mosaic flag.
Bit 13 Set color palette. If this is set to '1' we will get a palette of 256 colors, otherwise set to '0' if we want 16 palettes with 16 colors each. The palette you choose is set by bit 12-15 in attribute2.
Bit 14-15 These two bits together with bits 14-15 in attribute 1 are used to choose which shape the sprite is going to have (look for different shapes in the table below).

Attribute : 1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Form		VF	HF	Rot data index			X-coordinate								

Bits 0-8 X-coordinate for the sprite, the value can be between 0 and 512. (The screen is only 0-239 in x direction.)

Bits 9-13 If rotation bit in attribute 0 bit 8 is set, these bits determine which set of rotation/scaling parameters (0-31) is being used for the specific sprite. If the rotation bit is not set these bits are not used.

Bit 12 Set horizontal flip, which means the sprite image is flipped in horizontal direction.

Bit 13 Set vertical flip, which means the sprite image is flipped in vertical direction (actually turned upside down).

Bit 14-15 These two bits together with bits 14-15 in attribute 0 are used to choose which shape the sprite is going to have (look for different shapes in the table below).

Attribute : 2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Palette nummer				**Priority**		**Sprite number**									

Bit 0-9 Decides where in memory the sprite data is stored. These memory areas are divided into 8x8 tiles, each using 32 bytes. This way index 0 means that the sprite info is stored in the first 32 bytes of sprite data, index 1 means the next 32 bytes of tile data, etc.. When using 256 color sprites each tile takes up 64 bytes. It is therefore necessary to use every other index to locate your tiles when you're using 256 color sprites.

Bit 10-11 Set priority to sprites (0-3), which determines the layering of the sprites. A sprite with priority 0 will be drawn above a sprite with priority 1. The priority of a sprite is always higher then a background with the same priority. This means that if you draw a background with priority 2 and a sprite with priority 2, the sprite will always be drawn in front of the background. When it comes to sprites with the same priority the one with the lower OAM index will be drawn on top of the other.

Bits 12-15 chooses which palette we wish to use for our sprite (0-15). This is only for when we're using 16 colors/16 palettes; if we use 256 colors there is only one palette, so these bits are not used.

This table shows the 12 possible shapes of sprites, and how much memory they require.

Shape (attribute 0)	Shape (attribute1)	Size	Total amount of sprites in 256 colors mode	Amount of bytes per sprite (256 colors)	Total amount of sprites in 16 colors mode	Amount of bytes per sprite 16 colors
SQUARE	SIZE_8	8X8	512	64	1024	32
SQUARE	SIZE_16	16X16	128	256	256	128
SQUARE	SIZE_32	32X32	32	1024	64	512
SQUARE	SIZE_64	64X64	8	4096	16	2048
TALL	SIZE_8	8X16	256	128	512	64
TALL	SIZE_16	8X32	128	256	256	128
TALL	SIZE_32	16X32	64	512	128	256
TALL	SIZE_64	32X64	16	2048	32	1024
WIDE	SIZE_8	16X8	256	128	512	64
WIDE	SIZE_16	32X8	128	256	256	128
WIDE	SIZE_32	32X16	64	512	128	256
WIDE	SIZE_64	64X32	16	2048	32	1024

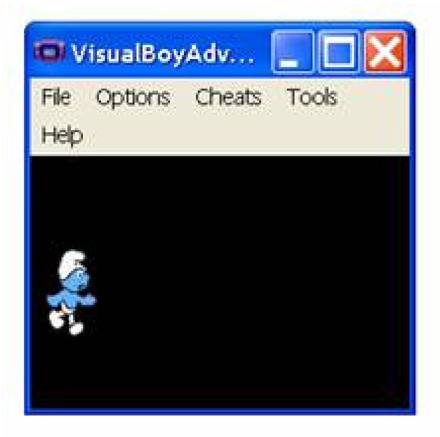
We always write 2 bytes (u16) to VRAM; it is not possible to just write one byte.

In tile modes 0, 1 and 2 we have the memory area from 0x6010000 to 0x6017FFF (32 KB) available to store sprite data (OBJ tiles). The maximum number of sprite tiles is 1024 8x8 16 color sprites (look in the table to find these numbers).

In bitmapped modes 3, 4 and 5 we only have half the available space for sprites (16 KB). The sprites are then stored in memory from 0x6014000 to 0x6017FFF (16KB). This means that if we in mode 3, 4 or 5 we have to start indexing our tiles at 512, instead of 0 like we did in modes 0, 1 and 2 (the tile modes).

Tiles can either be put into memory as a 1D (one dimensional) array or 2D (two dimensional) table in memory. It is a lot harder for the user to input sprite tiles as a 2D table than a 1D table. There is no immediate advantage to using 2D over 1D; because of this I will only use 1D tables in these lessons. By setting the 1D bit in REG_DISPCNT the gba reads all sprite data as 1D.

```
#include "gba.h"
#include "smurf.h"  //sprite data and palette made with pcx2sprite

//makes a table that contains OAM info about all 128 sprites
OAMEntry sprites[128];

//function to copy all our sprite data from the table to OAM memory
void CopyOAM()
{
        u16 loop;
        u16* temp;
        temp = (u16*)sprites;
        for(loop = 0; loop < 128*4; loop++) //2 bytes at the time (2byte * 4 * 128 = 1024 bytes)
        {
                OAM_Mem[loop] = temp[loop];
        }
}

//function that sets OAM data for all sprites, so that they are not shown on the screen.
//by setting coordinates to all sprites outside the visible area of the screen, they are not shown.
void InitializeSprites()
{
        u16 loop;
        for(loop = 0; loop < 128; loop++)
        {
                sprites[loop].attribute0 = 160;  //y to > 159
                sprites[loop].attribute1 = 240;  //x to > 239
                sprites[loop].attribute2 = 0;
        }
}

int main()
{
         u16 loop;
         s16 x = 10;
         s16 y = 50;

         //set mode 1, switch on sprites, set storing of sprite tiles as 1D
          SetMode(MODE_1 | OBJ_ENABLE | OBJ_MAP_1D);

        InitializeSprites();   //set all sprites outside the screen (removes unwanted sprites)

        sprites[0].attribute0 = COLOR_256 | TALL | y;  //set 256 colors, shape and y-coordinate
        sprites[0].attribute1 = SIZE_64 | x;  //set shape and x-coordinate
        sprites[0].attribute2 = 0;   //points to where in VRAM we get the tile data from

        for(loop = 0; loop < 256; loop++)          //transfer palette data into memory (256 colors)
        {
                OBJ_PaletteMem[loop] = palette[loop];
        }

        for(loop = 0; loop < 1024; loop++)   //transfer sprite tile data into memory (32x64)
        {
                OAM_Data[loop] = obj0[loop];
        }

        while(1)  //main loop
        {
                vsync();  //wait for screen to finish drawing.
                CopyOAM();  //copy sprite info (OAM) into OAM memory
        }
}
```



In this memory area the images/pictures of the sprites are stored. These are stored the same way as a bitmap image with a palette, except that the image will be divided into blocks of 8x8 bitmaps (tiles).

Color 0 from the palette is not drawn; this is the case whether there is a 256 color or 16 color palette. The reason for this is that by using color 0 you can make parts of the sprite transparent.

If we have 16 color sprites we can choose which 16 color palette we are going to use for each sprite (bits 12-15 attribute2).

To convert from a picture to sprite data, there are many different converter programs (pcx2sprite, for instance).

This is how you make a sprite with pcx2sprite:

First you draw a character in your favourite drawing program and store the picture as a pcx file. This pcx image has to have one of the 12 predefined shapes a sprite can have (for example: 8x8, 16x32 or any other of the shapes in the table above). You then take the image file and drag and drop it over the icon of the pcx2sprite converter. A new file will then be created (header file) with the same name as the image file. This file is given the extension .h and contains all the data we need to draw the sprite.

A sprite of size 32x16, starting at index 0 in memory, is drawn like this:

0	1	2	3
4	5	6	7

A sprite of size 16x32, starting at index 0 in memory, is drawn like this:

0	1
2	3

4	5
6	7