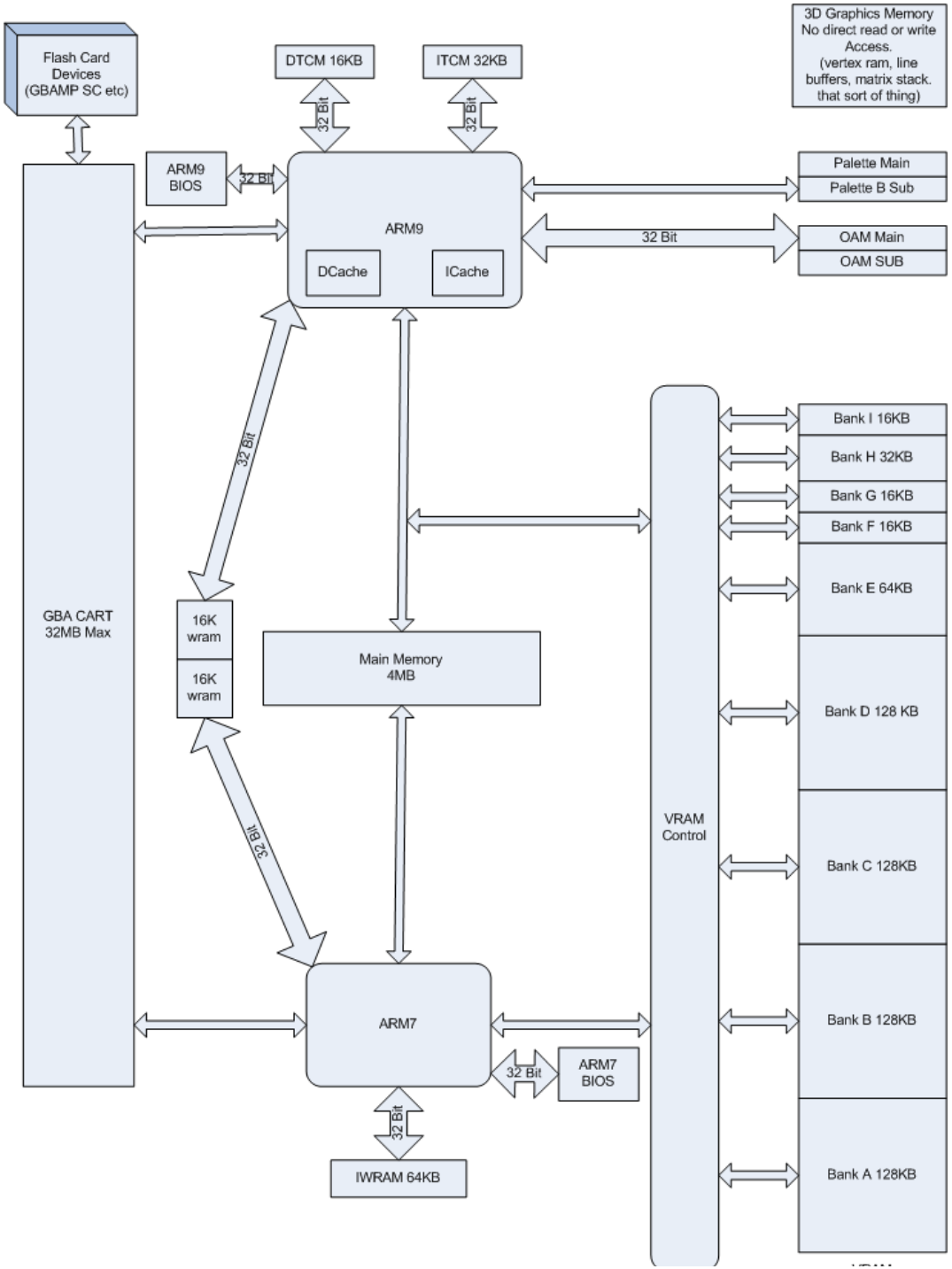
NDS Programming tutorial

Plan

- Présentation de la NDS
- Les URLs utiles
- Les microprocesseurs ARM7/ARM9
- La chaîne de compilation des applis NDS
- Plusieurs exemples

- http://lmn.mooo.com/projects/devkitpro-sh/
- http://libnds.devkitpro.org/
- http://www.patater.com/manual
- http://dev-scene.com/NDS/Tutorials


Flash Card Devices (GBAMP SC etc)
DTCM 16KB
ITCM 32KB
3D Graphics Memory No direct read or write Access. (vertex ram, line buffers, matrix stack. that sort of thing)
32 Bit
32 Bit
ARM9 BIOS
32 Bit
ARM9
DCache
ICache
Palette Main
Palette B Sub
32 Bit
OAM Main
OAM SUB
GBA CART 32MB Max
32 Bit
16K wram
16K wram
Main Memory 4MB
VRAM Control
Bank I 16KB
Bank H 32KB
Bank G 16KB
Bank F 16KB
Bank E 64KB
Bank D 128 KB
Bank C 128KB
Bank B 128KB
Bank A 128KB
32 Bit
ARM7
32 Bit
ARM7 BIOS
32 Bit
IWRAM 64KB

Inside NDS

Présentation des processeurs

Memory Map

ARM 9				
Name	Start Address	Stop Address	Size	Wait State
Main	0x02000000	0x023FFFFF	4MB	?
BIOS	0xFFFF0000	0xFFFF7FFF	32KB	?
ITCM	0x00000000	0x00007FFF	32KB	?
DTCM	0x0B000000	0x0B003FFF	16KB	?
Shared WRAM Bank 0	0x03000000	0x03003FFF	16KB	?
Shared WRAM Bank 1	0x03004000	0x03007FFF	16KB	?
ARM 7				
Main	0x02000000	0x023FFFFF	4MB	?
BIOS	0x00000000	0x00003FFF	16KB	?
IWRAM	0x03800000	0x0380FFFF	64KB	?
Shared WRAM Bank 0	0x03000000	0x03003FFF	16KB	?
Shared WRAM Bank 1	0x03004000	0x03007FFF	16KB	?
Video RAM				
Main OAM	0x07000000	0x070003FF	1KB	?
Sub OAM	0x07000400	0x070007FF	1KB	?
Main Palette	0x05000000	0x050003FF	1KB	?
Sub Palette	0x05000400	0x050007FF	1KB	?
Bank A	0x06800000	0x0681FFFF	128KB	?
Bank B	0x06820000	0x0683FFFF	128KB	?
Bank C	0x06840000	0x0685FFFF	128KB	?
Bank D	0x06860000	0x0687FFFF	128KB	?
Bank E	0x06880000	0x0688FFFF	64KB	?
Bank F	0x06890000	0x06983FFF	16KB	?
Bank G	0x06894000	0x06897FFF	16KB	?
Bank H	0x06898000	0x0689FFFF	32KB	?
Bank I	0x068A0000	0x068A3FFF	16KB	?
Virtual Video RAM				
Main Background	0x06000000	0x0607FFFF	512KB	?
Sub Background	0x06200000	0x0621FFFF	128KB	
Main Sprite	0x06400000	0x0643FFFF	256KB	?
Sub Sprite	0x06600000	0x0661FFFF	128KB	?

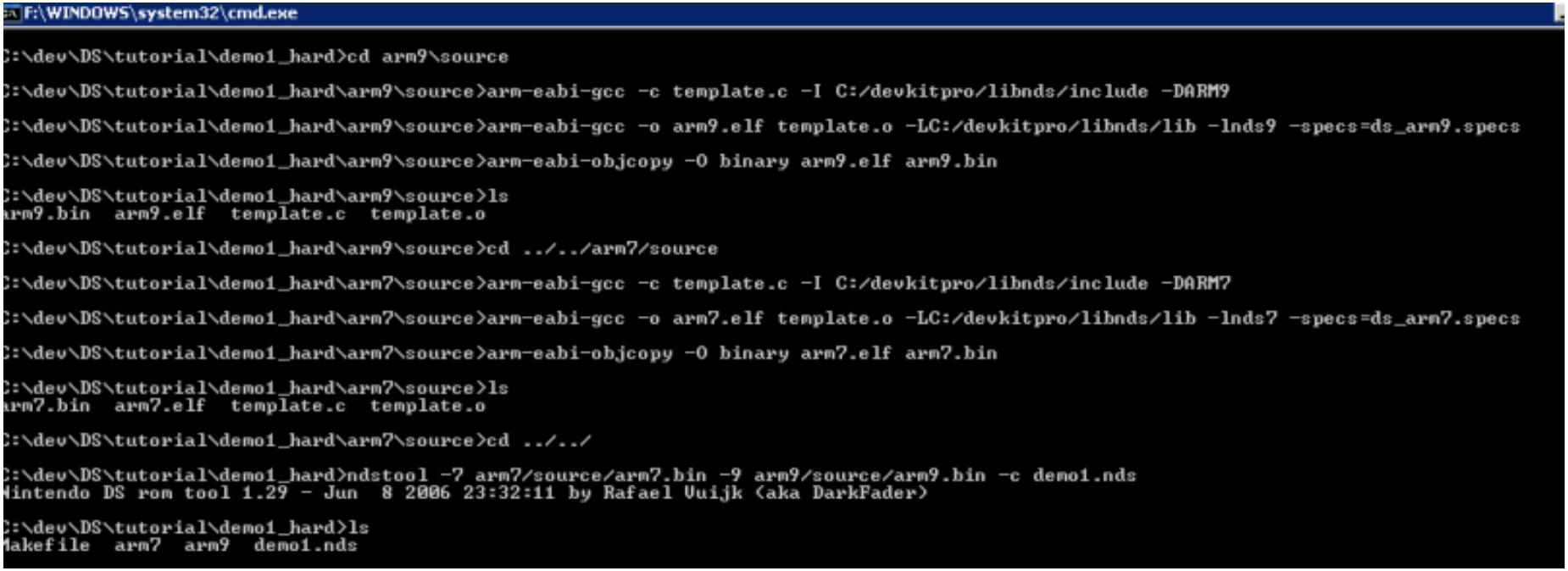
NDS Memory Map

Graphics Modes

Main 2D Engine				
Mode	BG0	BG1	BG2	BG3
Mode 0	Text/3D	Text	Text	Text
Mode 1	Text/3D	Text	Text	Rotation
Mode 2	Text/3D	Text	Rotation	Rotation
Mode 3	Text/3D	Text	Text	Extended
Mode 4	Text/3D	Text	Rotation	Extended
Mode 5	Text/3D	Text	Extended	Extended
Mode 6	3D	-	Large Bitmap	-
Frame Buffer	Direct VRAM display as a bitmap			
Sub 2D Engine				
Mode	BG0	BG1	BG2	BG3
Mode 0	Text	Text	Text	Text
Mode 1	Text	Text	Text	Rotation
Mode 2	Text	Text	Rotation	Rotation
Mode 3	Text	Text	Text	Extended
Mode 4	Text	Text	Rotation	Extended
Mode 5	Text	Text	Extended	Extended

NDS Graphics modes

Compiling NDS application



NDS Register access in C

```
unsigned int  *DISPLAY_CR = (unsigned int *) 0x4000000;
*DISPLAY_CR = somevalue;
```

```
#define DISPLAY_CR  (*(volatile unsigned int *) 0x4000000)
DISPLAY_CR = somevalue;
```

NDS Main loop, looking for Keyinput

```c
#include <nds.h>
#include <stdio.h>

int main(void)
{
	consoleDemoInit();

	while(1)
	{
		if(REG_KEYINPUT & KEY_A)
			printf("Key A is released");
		else
			printf("Key A is pressed");

		swiWaitForVBlank();

		consoleClear();
	}

	return 0;
}
```

NDS Main loop, handling input

```c
#include <nds.h>
#include <stdio.h>

int main(void)
{
	consoleDemoInit();

	while(1)
	{
		scanKeys();
		int held = keysHeld();

		if( held & KEY_A)
			printf("Key A is pressed\n");
		else
			printf("Key A is released\n");

		if( held & KEY_X)
			printf("Key X is pressed\n");
		else
			printf("Key X is released\n");

		if( held & KEY_TOUCH)
			printf("Touch pad is touched\n");
		else
			printf("Touch pad is not touched\n");

		swiWaitForVBlank();

		consoleClear();
	}

	return 0;
}
```

NDS Main loop, with some colors

```c
#include <nds.h>
#include <stdio.h>

int main(void)
{
	int i;

	//initialize the DS Dos-like functionality
	consoleDemoInit();

	//set frame buffer mode 0
	videoSetMode(MODE_FB0);

	//enable VRAM A for writing by the cpu and use
	//as a framebuffer by video hardware
	vramSetBankA(VRAM_A_LCD);

	while(1)
	{
		u16 color = RGB15(31,0,0); //red

		scanKeys();
		int held = keysHeld();

		if(held & KEY_A)
			color = RGB15(0,31,0); //green

		if (held & KEY_X)
			color = RGB15(0,0,31); //blue

		swiWaitForVBlank();

		//fill video memory with the chosen color
		for(i = 0; i < 256*192; i++)
			VRAM_A[i] = color;
	}

	return 0;
}
```

Managing colors in Framebuffer mode

```
int red = 31;
int blue = 0;
int green = 0;

unsigned short int color_red = (blue << 10) | (green << 5) | red;
```

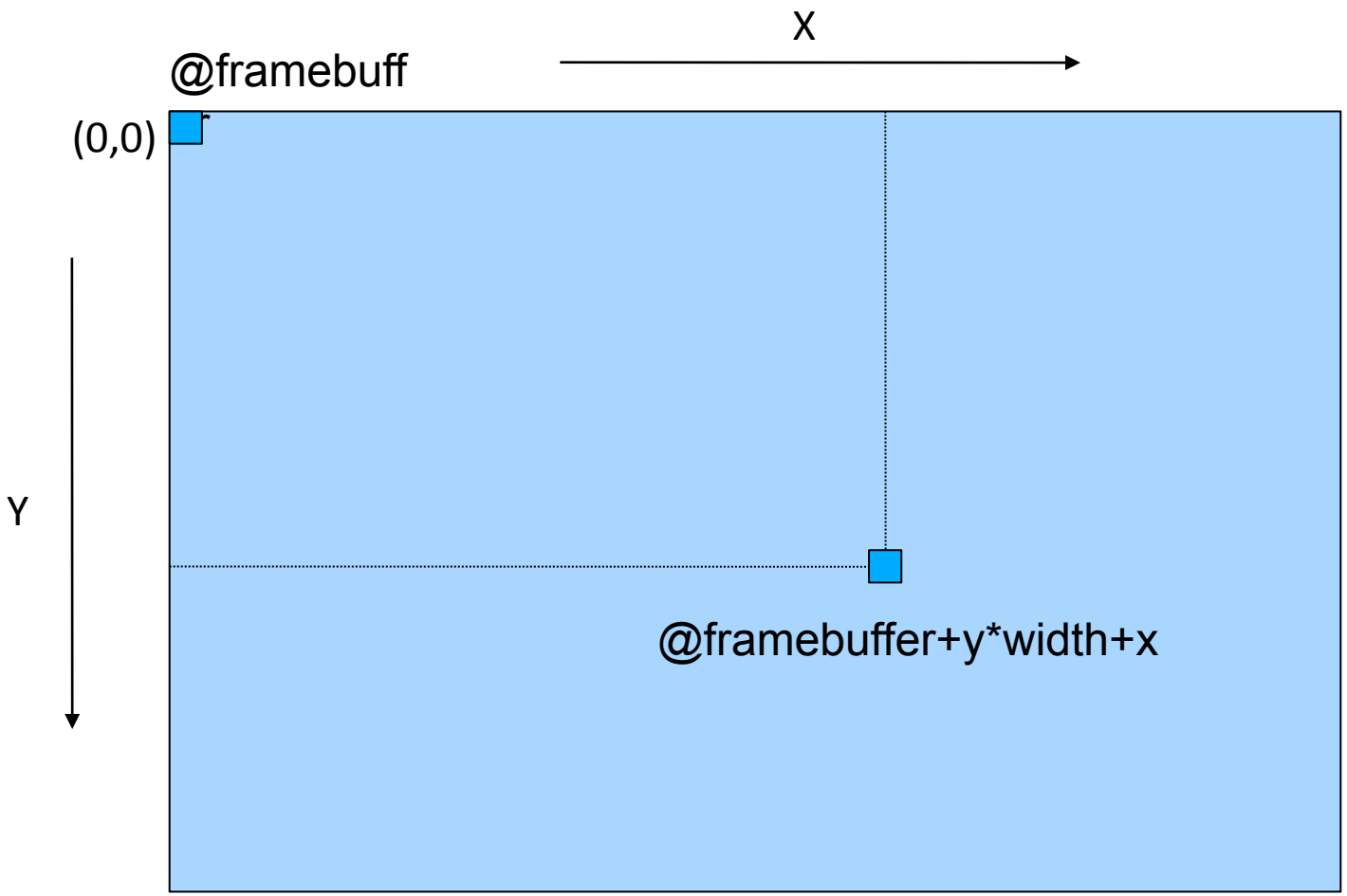
```
#define RGB15(r,g,b)  ((r)|((g)<<5)|((b)<<10))

//to use:

unsigned short int color = RGB15(red, green, blue);
```

```
//set alpha
color = color | (1<<15);

//clear alpha
color = color & ~(1<<15);
```

Acces to Framebuffer data

Image:Pixel offseting

```
unsigned short* frame_buffer = address_of_the_buffer;

frame_buffer[y * width_in_pixels + x] = color;
```



Star application (1)

```
typedef struct
{
        int x;
        int y;
        int speed;
        unsigned short color;

}Star;
```

We then need an array of stars we can track across the screen:

```
#define NUM_STARS 40

Star stars[NUM_STARS];
```

Before we start the demo, we need to clear the pixels of any color information they currently have. In other words, we are making sure we start with a black screen.

```
void ClearScreen(void)
{
     int i;

     for(i = 0; i < 256 * 192; i++)
           VRAM_A[i] = RGB15(0,0,0);
}
```

```
void InitStars(void)
{
        int i;

        for(i = 0; i < NUM_STARS; i++)
        {
                stars[i].color = RGB15(31,31,31);
                stars[i].x = rand() % 256;
                stars[i].y = rand() % 192;
                stars[i].speed = rand() % 4 + 1;
        }
}
```

Star application (2)

```
void EraseStar(Star* star)
{
        VRAM_A[star->x + star->y * SCREEN_WIDTH] = RGB15(0,0,0);
}
```

To erase just set the location of the star in the framebuffer to the background color (black in our case).

Similarly to draw the star we set the location of the star in the frame buffer to the color of the star:

```
void DrawStar(Star* star)
{
        VRAM_A[star->x + star->y * SCREEN_WIDTH] = star->color;
}
```

The final step is to move the star to its new location:

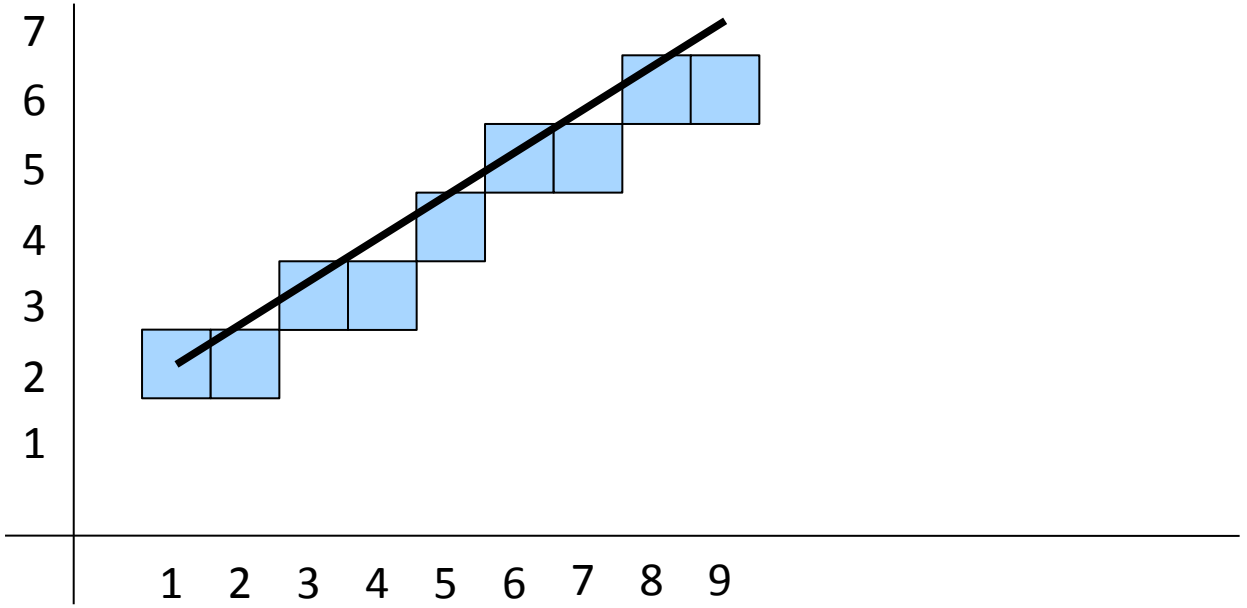
```
void MoveStar(Star* star)
{
        star->x += star->speed;

        if(star->x >= SCREEN_WIDTH)
        {
                star->color = RGB15(31,31,31);
                star->x = 0;
                star->y = rand() % 192;
                star->speed = rand() % 4 + 1;
        }
}
```

Star application (3)

```
//we like infinite loops in console dev!
        while(1)
        {
                swiWaitForVBlank();

                for(i = 0; i < NUM_STARS; i++)
                {
                        EraseStar(&stars[i]);

                        MoveStar(&stars[i]);

                        DrawStar(&stars[i]);
                }
        }
```

Bresenham algorithm

```
void DrawLine(int x1,int y1,int x2,int y2)
{
        int xStep=1;
        int yStep=10;
        int xDiff=x2-x1;
        int yDiff=y2-y1;
        int ix, iy;
        int errorTerm=0;

        // cas xDiff > yDiff

        iy=y1;
        for (ix=x1;ix<x2+1;ix++)
        {
                printf("%d %d\n",ix,iy);
                errorTerm += yDiff;
                if (errorTerm > xDiff)
                {
                        errorTerm -= xDiff;
                        iy++;
                }
        }
}
```



1 2
2 2
3 3
4 3
5 4
6 5
7 5
8 6
9 6

Bresenham algorithm

```
threshold = xdiff;

for(x = x1; x < x2; x++)
{
        //increment the error term
error += ydiff;

        //if the error gets big enough we correct by moving down one
        //y increment
        if(error > threshold)
        {
                y++;
                error = eror – threshold;
        }

        buffer[x + y * BUFFER_WIDTH] = color;
}
```

Dynamic Time Warping algorithm

```
int DTWDistance(char s[1..n], char t[1..m]) {
    declare int DTW[0..n, 0..m]
    declare int i, j, cost

    for i := 1 to m
        DTW[0, i] := infinity
    for i := 1 to n
        DTW[i, 0] := infinity
    DTW[0, 0] := 0

    for i := 1 to n
        for j := 1 to m
            cost:= d(s[i], t[j])
            DTW[i, j] := cost + minimum(DTW[i-1, j  ],    // insertion
                                        DTW[i  , j-1],    // deletion
                                        DTW[i-1, j-1])    // match

    return DTW[n, m]
}
```