Atari ST Multi-Palette Pictures and Global Optimization

Silly Venture 2016,Gdańsk,Poland

François Galea aka Zerkman/Sector One

Nov 12th, 2016

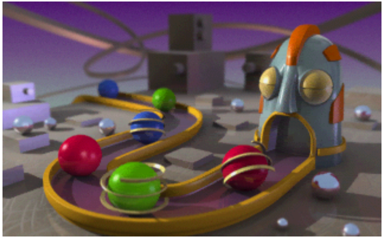
Outline

Context

Atari ST: A revolutionary personal computer (in 1985)

- 16-bit, 8 MHz microprocessor (Motorola 68000)
- Modern graphics: 320x200 screen image resolution, 16 colors !
- color palette entries from a set of 512 possible colors, then 4096 on the STe (1989)

Problem: display images with modern color standards.



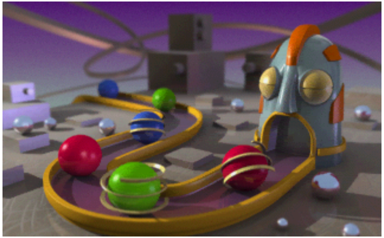
Question: is it possible to display images with more than 16 colors on the screen ?

Context

Atari ST: A revolutionary personal computer (in 1985)

- 16-bit, 8 MHz microprocessor (Motorola 68000)
- Modern graphics: 320x200 screen image resolution, 16 colors !
- color palette entries from a set of 512 possible colors, then 4096 on the STe (1989)

Problem: display images with modern color standards.



Question: is it possible to display images with more than 16 colors on the screen ?

Spoiler alert: yes!

Tweaking the display limitations

From the beginning: change the color palette between scanlines

- Use of horizontal blank or timer-B interrupt handlers to perform color palette changes
- Raster effects by changing the background color
- Neochrome Master (early 1990s)

Since then, various tools using synchronized code to increase the number of colors per line

- 1987: Spectrum 512
- 24bit.tos (Les palettes étendues) by Mathias Agopian
- Photochrome by Doug Little
- Multipalette Picture Format by François Galea (me!)
- ... and probably more

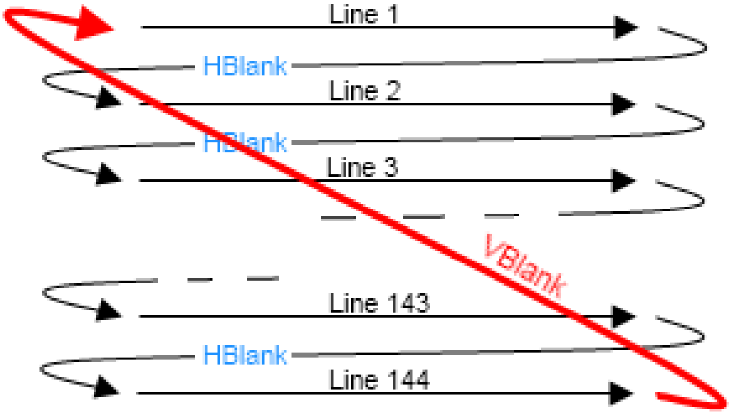
Multi-Palette Picture (MPP)

- MPP uses synchronized code to:
  - change the palette entries while the scanlines are decoded by the Shifter chip
    - Spectrum 512: 44 colors per line!
    - MPP: 56 colors per line on STe, 54 on ST
  - possibly do that in fullscreen
    - 412x272 screen resolution, 48 colors per scanline
- The image encoder uses combinatorial optimization techniques to decide the color values
- MPP file format, with tags similar to SNDH
- Free software (WTFPL), source code available on `http://github.com/zerkman/mpp`

Outline

Video display basics

In case you never knew:



How multi-palette works

Consider this video-synchronized piece of code:

```
lea     palette,a1
lea     $FF8240,a0
move.w  (a1)+,(a0)+
move.w  (a1)+,(a0)+
move.w  (a1)+,(a0)+
move.w  (a1)+,(a0)+
```

What is happening ?

- First move changes the background color (0), second move changes color 1 and so on
- On ST low resolution, displaying a pixel takes 1 CPU cycle
- Each move is executed in 12 cycles $\rightarrow$ each color change is effective 12 pixels to the right of the previous one

MPP display modes

As of today, MPP features 4 different display/screen modes:

- Mode 0: based on `move.l` instructions
  - Each `move.l` takes 20 cycles and performs two color changes
  - 48 colors per scanline, with good horizontal repartition.
- Mode 1: based on `movem.l` instructions
  - half `movem.l` instructions load color values into registers, the other half write the values into palette registers.
  - 54 colors per scanline, with irregular repartition.
- Mode 2: using the STe's blitter
  - A single blitter operation during the whole image scanning to perform writes to the color palette regiters in a cyclic way.
  - A color change is performed every 8 cycles/pixels.
  - 56 colors per line, with very regular repartition.
- Mode 3: fullscreen and `movem.l` instructions
  - 48 colors per line, irregular.

MPP display modes

- Non-fullscreen modes set all 16 palette entries before each line is displayed, then perform palette updates during image decoding
- In fullscreen, there is not enough time between scanlines to reset the whole palette $\rightarrow$ 10 colors from the above scanline are reused
- The horizontal position of each color change for each screen mode is perfectly known
  - Modulo some wakestate issues :)
- The general color model is then as this:

X position	0	1	2	...	4	...	12	....	W-1
pal. interval	0..15	0..15	0..15	0..15	1..16	1..16	2..17	2..17	N-16..N-1

MPP Extended color modes

MPP enables palettes with 1 additional bit per component, allowing to display

- 3375 (15^3) colors on ST
- 29791 (31^3) colors on STe

The additional bit is simulated by alternating the colors at each screen refresh, giving the illusion of intermediate colors.

- F.I, a 4-bit component value of 11 can be achieved on ST by alternating 3-bit component values 5 and 6.
- Alternate the use of low and high values on even/odd lines, to maintain a constant brightness level and avoid a flashing effect between frames.

Two ways of doing that:

- Transform a palette array with extra bits into two alternating palette arrays with the native color format
- Encode two pictures with alternated component values (better quality, but almost twice the memory size)

Outline

A Brief Introduction to Operations Research

A research field to solve difficult optimization problems, modelized in a mathematical way:

- A set of variables represent the unknowns of the problem
- A set of constraints on the variables define the feasible solution space
- An objective function to be optimized to get the solution quality as good as possible

A huge set of methodologies exist to solve such problems. They are divided into two major classes:

- Exact methods to find the optimal solution.
- Approximate methods to find a good enough solution. Much shorter solution times.

The Multi Palette Assignment Problem

Problem: In a specified display mode, for a scanline, being given a specific input array of true color pixels, determine the values of all palette entries.

- Once the palette values are found, finding the correct pixel values is straightforward.
- The problem is solved at each scanline.

Let's formulate things a bit:

- A color c is a vector with 3 components $c = \{c_r, c_g, c_b\}$
- The color distance function between two colors c and c' is

$$\mathrm{cdist}(c, c') = (c_r - c'_r)^2 + (c_g - c'_g)^2 + (c_b - c'_b)^2$$

The Multi Palette Assignment Problem

Problem parameters:

- N number of palette entries per line
- W number of pixels per line (320 or 412)
- p_j the color value of the j^{th} pixel in the original image line, $0 \leq j < W$
- f_j the first valid palette index for the j^{th} pixel — last one is $f_j + 15 \leq N$

Variables:

- x_i The chosen color for the i^{th} palette entry

The solution cost at pixel j is:

$$\min_{i=f_j}^{f_j+15} \text{cdist}(p_j, x_i)$$

Then our objective function to be minimized is:

$$\sum_{j=0}^{W-1} \min_{i=f_j}^{f_j+15} \text{cdist}(p_j, x_i)$$

The Multi Palette Assignment Problem

Constraints:

- Border colors (0 on the left, 32 or 48 on the right) are forced to black.
- We reduce the search space by only allowing color changes that correspond to colors in the original image in the pixel interval for palette entries
  - Pre-calculated array of possible colors for each palette entry

Exact methods for OR problems

Exact methods are used when an optimal solution is needed. Generally take a lot of time.

- Exhaustive search: brute force
- Divide and conquer: recursive search by dividing a problem into hopefully easier subproblems.
  - e.g solve the problem for all possible values for a specific x_i and take the best solution.
- Branch and bound: D&C + a bounding method to eliminate some subproblem sets.

A (buggy) B&B solver is in MPP. Relatively useless.

Approximate methods

Approximate methods to find a good enough solution. Much shorter solution times.

- Greedy algorithms: each choice made is definitive. Usually fast, with moderate solution quality.
- Local search: explore the solution set by the means of a neighborhood function, allowing to jump from one solution to another, stop when no better solution can be found (local optimum)
- Metaheuristics: more or less nature-inspired methodologies to search for good solutions while avoiding local optima.
  - genetic algorithms
  - scatter search
  - tabu search
  - ant colony
  - simulated annealing
  - ...

MPP Greedy method

A simple algorithm to find a "not too bad" solution. Inspired from 24bit.tos by Mathias Agopian

initialize all x_i values to -1
$j \leftarrow 0$
for all $j' \in [0..N-1]$ **do**
  if no x_i contains p_j, for all $i \in [f_j..f_j+15]$ **then**
    if there exists one $x_i = -1$, such as $i \in [f_j..f_j+15]$ **then**
      $x_i \leftarrow p_j$
    end if
  end if
  $j \leftarrow j+4$
  if $j \geq W$ **then**
    $j \leftarrow j - W + 1$
  end if
end for

Simulated annealing

“Simulated annealing is a probabilistic technique for approximating the global optimum of a given function. Specifically, it is a metaheuristic to approximate global optimization in a large search space.” (Wikipedia)

- A metaheuristic inspired from a technique in metallurgy
- Can be seen as an extension of local search
- Very simple to understand (and to code)

What it is not:

- Complicated (unlike many other metaheuristics)

Simulated annealing

Like local search, uses a neighborhood function which randomly generates a new solution by performing a minor change on a current solution.

- *e.g*, change one palette value
- Exploits the fact that neighbor solutions potentially are of the same level of quality
- Solution values are faster to compute, as in our case we don't have to re-compute the whole sum of minimum color distances

It is an iterative exploration process, where the solution space is explored by performing moves from one current solution to a neighbor solution.

- Makes use of a **temperature value** which decreases along time.
- The probability of accepting a new solution depends on the temperature.

Simulated annealing

If the current solution has the value v, and the temperature is T, and considering a random number $r \in [0, 1]$, the new solution of value v' is accepted if the following test is successful:

$$r \leq e^{\frac{v-v'}{T}}$$

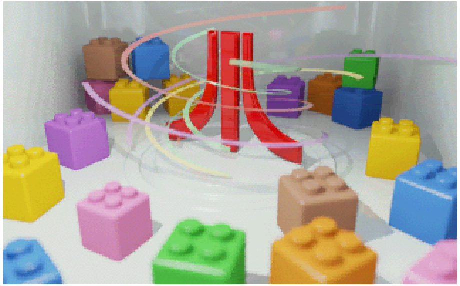
- always true if $v' \leq v$
- also often true if $v' > v$ and T is high

Temperature decrease scheme: The temperature is regularly decreased after a fixed number of iterations.

- that number depends on the optimization level specified by the user.

Stop criterion: the algorithm stops when the temperature reaches a certain value, depending on the best and worst solution values found so far (see source code for more details !)

Conclusion



- Suitable as a graphics interchange format
  - Silly Venture graphics compo ?
- Source code is available at `http://github.com/zerkman/mpp`
- Can be used in a lot of ways:
  - demos
  - import/export plugin for graphics software
  - your own silly projects
- Can be extended/adapted to specific needs