0.1 Building layer for layer

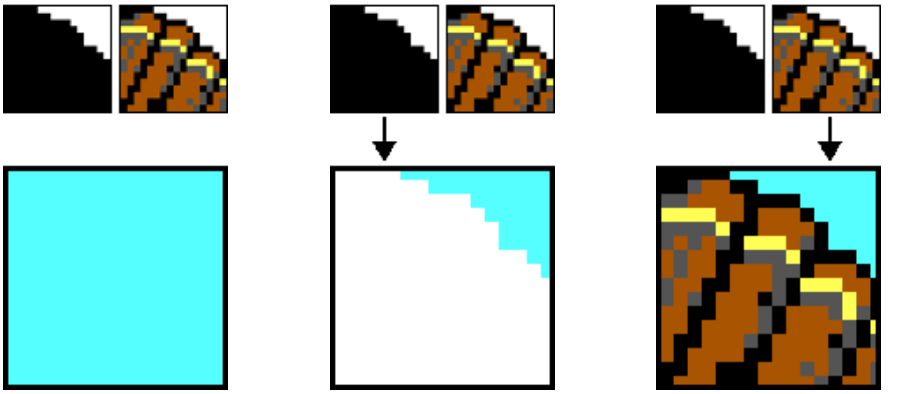
Each tile on the screen can contain up to three layers: the background tile, the foreground tile, and a sprite layer. It requires multiple redrawing on the same tile:

1. Draw the background tile or a combined background and foreground tile.
2. Draw the sprites.
3. Re-draw the foreground tile if the sprite should not appear on top.

In the best case, the engine must read and write 128 bytes (2 * 16 bytes * 4 memory banks) to VRAM for a background tile only. In the worst case, up to 512 bytes are needed (background, foreground, sprite, and foreground again), with several bitwise operations involved. For the sake of speed, all code is written in assembly, as explained in the following sections.

0.1.1 Draw background and foreground tiles

Drawing the background tiles is straightforward, as it only involves copying 128 bytes to VRAM. However, drawing foreground tiles on top of the background requires an additional mask layer, which defines which pixels are overwritten by the foreground tile.



(a) Background tile. ***(b)*** `AND` foreground mask. ***(c)*** `OR` foreground bitmap.

Figure 1: Merge foreground with background tile.

Combining the background and foreground tile is handled by the `RFL_NewTile` function, using the `AND` bitwise operator to clear the background and the `OR` bitwise operator to write the foreground tile.

```
PUBLIC  RFL_NewTile

[...]
  es,[mapsegs+2]          ;foreground plane
  mov bx,[es:si]
  mov es,[mapsegs]        ;background plane
  mov si,[es:si]

[...]
  or  bx,bx               ;do we have foreground tile?
  jz  @@singletile        ;draw background tile only
  jmp @@maskeddraw        ;draw both together

[...]
@@maskeddraw:
  shl bx,1
  mov ss,[grsegs+STARTTILE16M*2+bx]
  shl si,1
  mov ds,[grsegs+STARTTILE16*2+si]

  xor si,si               ;first word of tile data

  mov ax,SC_MAPMASK+0001b*256 ;map mask for plane 0

  mov di,[cs:screenstartcs]

@@planeloopm:
  WORDOUT
tileofs = 0
lineoffset  = 0
REPT  16
  mov bx,[si+tileofs]    ;background tile
  and bx,[ss:tileofs]    ;mask
  or  bx,[ss:si+tileofs+32] ;masked data
  mov [es:di+lineoffset],bx
tileofs   = tileofs + 2
lineoffset  = lineoffset + SCREENWIDTH
ENDM
```

0.1.2 Drawing sprites

The next step is to render sprites on the screen. Most home computers of that era had built-in sprite functionality on the video card. For example, on a MSX computer, one could simply enter

```
PUT SPRITE <SpriteNumber>, <X>, <Y>, Color
```

to display a sprite on the screen. Updating the `(X, Y)` coordinates would move the sprite, with the display adapter handling everything else. However, the concept of sprites did not exist on EGA cards, so game developers had to implement their own solution.

A challenge arises from the fact that sprites can move freely across the screen and are not byte-aligned. To address this, the bit-shifting technique described in section **??** is used. When caching a sprite into memory, each sprite is copied four times, with each copy shifted by two or more pixels using this technique. The property `*spr->shifts` determines the number of bit shifts applied to each of the four copies.

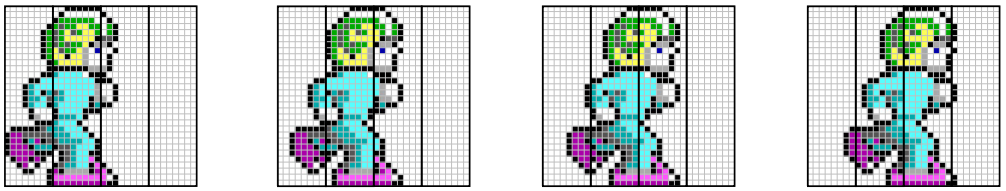


Figure 2: Sprite shifted in 4 steps.

Displaying the correct shifted sprite is as simple as

```
//Set x,y to top-left corner of sprite
y+=spr->orgy>>G_P_SHIFT;
x+=spr->orgx>>G_P_SHIFT;

shift = (x&7)/2;  // Set sprite shift
```

```
void CAL_CacheSprite (int chunk, char far *compressed)
{
[...]
//
// make the shifts!
//
  switch (spr->shifts)
  {
  case  1:  // no shifts
    [...]

  case  2:  // one shift of 4 pixels
    for (i=0;i<2;i++)
    {
      dest->sourceoffset[i] = shiftstarts[0];
      dest->planesize[i] = smallplane;
      dest->width[i] = spr->width;
    }
    for (i=2;i<4;i++)
    {
      dest->sourceoffset[i] = shiftstarts[1];
      dest->planesize[i] = bigplane;
      dest->width[i] = spr->width+1;
    }
    CAL_ShiftSprite ((unsigned)grsegs[chunk],dest->
   sourceoffset[0],
      dest->sourceoffset[2],spr->width,spr->height,4);
    break;

  case  4:  // four shifts of 2 pixels
    [...]
  default:
    Quit ("CAL_CacheSprite: Bad shifts number!");
  }
}
```

If multiple sprites are displayed on the same tile, each sprite is assigned a priority from 0 to 3 to determine the drawing order. A sprite with a higher priority number is always drawn on top of sprites with a lower priority. Since sprites are always drawn on top of tiles, this can create unnatural situations, such as when Commander Keen is climbing through a hole, as illustrated in Figure 3.

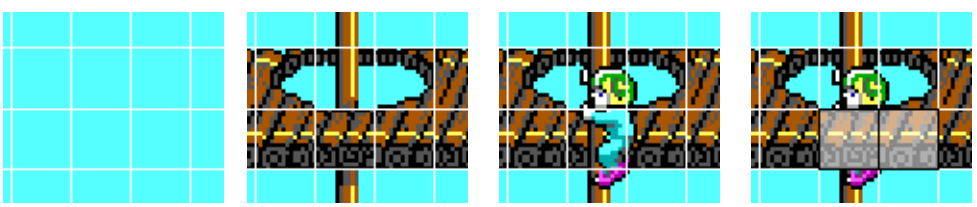
(a) Background tile. ***(b)*** Foreground tile. ***(c)*** Sprite on top.

Figure 3: Unnatural situation where Commander Keen is in front of a hole.

To draw sprites 'inside' a foreground tile, a small trick is used by introducing a priority foreground tile. As explained in section **??**, each foreground tile is enriched with INTILE ("inside tile") information. If the highest bit (`80h`) of INTILE is set, that foreground tile has a higher priority than sprites with priority 0, 1, or 2. Therefore, the following drawing order is applied:

1. Draw the background tile and the masked foreground tile.
2. Draw sprites with priority 0, 1, and 2 (in that order), and mark the corresponding tile in the tile buffer array with a '3', as illustrated in Figure **??** on page **??**.
3. Scan the tile buffer array for tiles marked with '3'. If the corresponding foreground INTILE high bit (`80h`) is set, redraw the masked foreground tile.
4. Finally, draw sprites with priority 3. These sprites are always drawn on top of everything.

The priority foreground tiles are updated in the `RFL_MaskForegroundTiles()` function.



(a) Background tile. ***(b)*** Foreground tile. ***(c)*** Sprite on top. ***(d)*** Redraw masked foreground tile.

Figure 4: Draw sprite inside a tile, by redrawing foreground tile.

```
  jmp SHORT @@realstart   ; start the scan
@@done:
;=============
; all tiles have been scanned
;=============
  ret

@@realstart:
  mov di,[updateptr]
  mov bp,(TILESWIDE+1)*TILESHIGH+2
  add bp,di           ; when di = bx,
  push di             ; all tiles have been scanned
  mov cx,-1           ; definately scan the entire thing
;=============
; scan for a 3 in the update list
;=============
@@findtile:
  mov ax,ss
  mov es,ax           ; scan in the data segment
  mov al,3            ; check for tiles marked as '3's
  pop di              ; place to continue scanning from
  repne scasb
  cmp di,bp
  je  @@done
;=============
; found a tile, see if it needs to be masked on
;=============
  push  di
  sub di,[updateptr]
  shl di,1
  mov si,[updatemapofs-2+di]  ; offset from originmap
  add si,[originmap]
  mov es,[mapsegs+2]      ; foreground map plane segment
  mov si,[es:si]          ; foreground tile number
  or  si,si
  jz  @@findtile          ; 0 = no foreground tile
  mov bx,si
  add bx,INTILE           ; INTILE tile info table
  mov es,[tinf]
  test [BYTE PTR es:bx],80h ; high bit = masked tile
  jz  @@findtile

; mask the tile
```