# ANNEX 3: "MODE 7" A LA GBA

## A3.0 Objectiu

Aquest annex donarà el disseny i la implementació necessària per construir una demostració del "mode 7" a la consola de videojocs *Game Boy Advance* (GBA). El "mode 7" (nom originari de la consola SuperNintendo –SNES-) és una habilitat per hardware que disposa en algunes màquines per simular un pla en perspectiva.

La GBA no incorpora la inicialització del "mode 7" a la seva BIOS, però, basat en els modes rajola transformables de la GBA (modes 1 o 2), és possible inicialitzar l'efecte "mode 7".

El seguiment de la construcció "mode 7" serà el següent:

- Primerament, s'explicarà l'origen del "mode 7".

- Llavors, a la secció A3.2 i A3.3, es començarà amb l'estudi dels modes gràfics i registres involucrats en la inicialització/renderització de l'efecte "mode 7" a la GBA.

- Tot seguit (secció A3.4), entrarem en un estudi teòric deduint les formules involucrades en una *renderització* d'un pla 3D per-software.

- A la secció A3.5 i basant-nos amb els dos punts anteriors, s'implementarà en C el "mode 7" per la GBA.

- Per últim, s'explicarà com crear la una demostració "mode 7" a la GBA, mitjançant eines del kit HAM.

Abans de entrar en l'estudi veiem, però, una mica de historia d'aquest mode gràfic.

## A3.1 Història

En la dècada dels 90 , Nintendo va treure la continuació la NES, La Super Nintendo (SNES) o en Japó, la Super Famicom (figura A3.0). La SNES tenia una hardware gràfic 2D que permetia rotar i escalar gràfics 2D per-hardware. Els programadors de la SNES van aprofitar aquesta capacitat, configurant el hardware necessari, per tal que a cada *scanline* de pantalla disposes d'un paràmetre de escalat i rotat diferent a l'espai de mapa. Amb aquesta nova configuració va permetre a la SNES, la capacitat de simular un pla amb perspectiva 3D per-hardware; aquest l'anomenaren "**mode 7"**.

**Figura a3.0** *Super Nintendo (SNES)*

El “mode 7” podia simular pla amb perspectiva 3D des de qualsevol punt de vista i ser rotada en qualsevol angle. L’efecte “mode 7” disposà a la SNES de fer un passeig 3D pel mapa del món en un joc RPG com, per exemple, el joc *Terra Enigma*. Aquest efecte, també es va estendre bastant en els jocs de carreres com, per exemple, el joc *F-Zero*. *Axelay* també utilitzar el “mode 7”, però sense rotats a l’espai 3D. Per últim, les series *Street Fighter* utilitzava el “mode 7” per crear un efecte dissimulat de desplaçament en el terra.

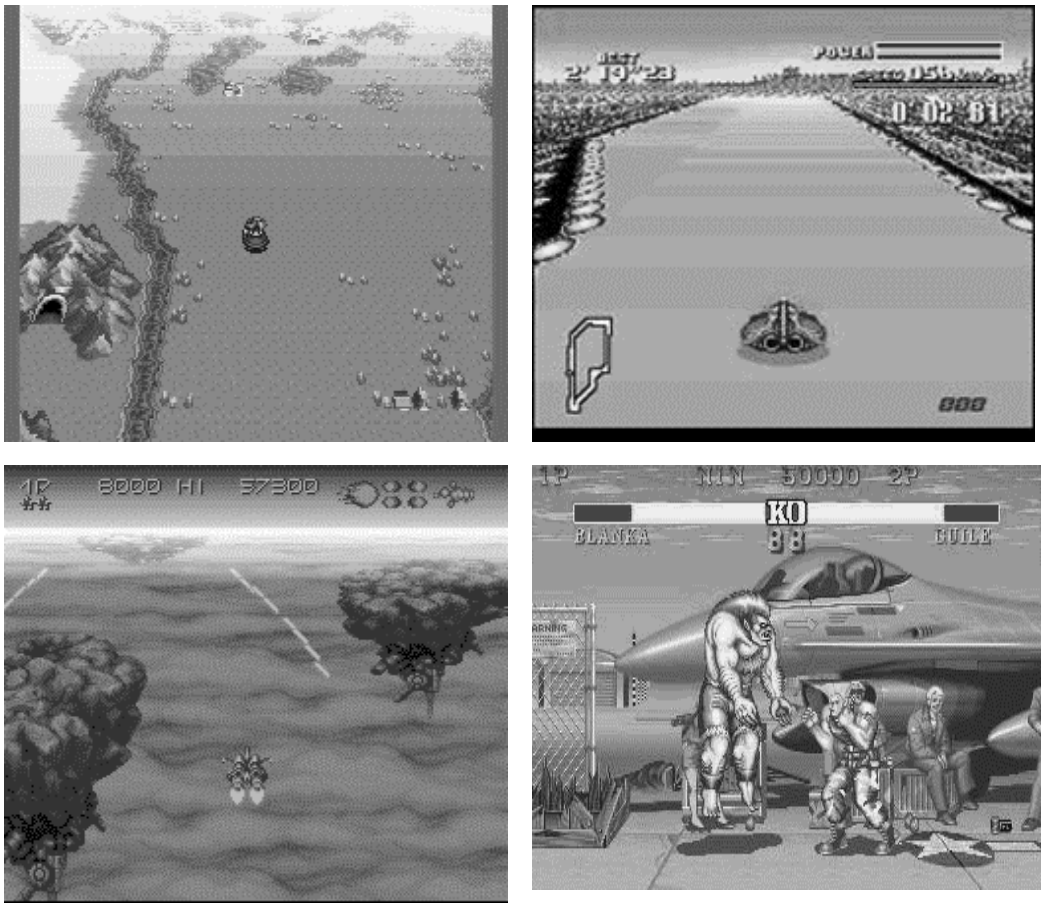



**Figura a3.1** *A l’esquerra superior, Terra Enigma, a la dreta superior F-Zero, a la l’esquerra inferior Axelay, per últim, a la dreta inferior Street Figther*

# A3.2 Modes rajola a la GBA: mode 0,1 o 2

Quan la GBA inicialitza el mode 0, 1 o 2, la pantalla es subdivideix en quadrats de 8x8 pixels (rajola), on el conjunt d’aquestes rajoles formen el fons. Per la seva inicialització és necessari col·locar les rajoles a la memòria de vídeo i un mapa, que tan sols es un vector (figura A3.3) on cada element es la referència a la textura amb uns quants atributs més. Llavors, el processador de vídeo repassa el vector de mapa (figura A3.3) i construeix la pantalla (figura A3.4) amb les rajoles que li hem especificat (figura A3.2). La grandària del fons pot ocupar des de 256x256 pixels fins a 1024x1024 pixels.

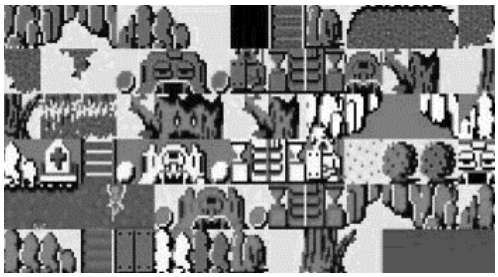


**Figura a3.2** *Col·lecció de rajoles guardades com mapes de bits de 8x8 pixels (rajola) a VRAM*

```
Map[] = {
64,64,64,64,64,64,64,6 ,6 ,64,64,64,64,64,64,64,
64,64,64,64,7 ,64,62,06,06,64,64,64,64,64,64,64,
64,64,64,62,06,06,06,06,06,64,64,64,64,64,64,64,
64,64,62,06,06,06,06,06,06,64,64,64,64,64,64,64,
64,52,06,06,06,06,06,06,06,63,64,64,64,64,64,64,
06,06,06,06,06,06,06,06,06,06,06,06,06,06,06,06,
64,67,06,06,06,06,06,06,06,06,06,06,06,06,64,64,
64,64,06,06,06,06,06,06,06,06,06,06,06,06,64,64,
64,64,06,06,06,06,06,06,06,06,06,06,06,06,64,64,
64,64,66,66,66,66,66,66,66,66,66,66,66,66,64,64,
64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64};
```

**Figura a3.3** *El vector de mapa en C. Cada posició del vector fa referència a una textura.*

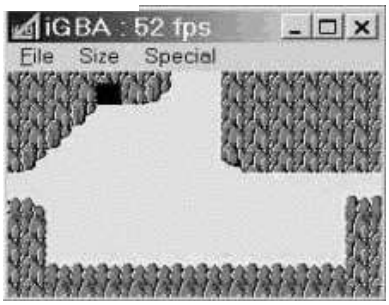



**Figura a3.4** *Resultat gràfic, amb les textures de la figura a3.2 i el vector de mapa de la figura a3.4.*

La GBA pot mostrar fins a quatre fons diferents quan opera en mode rajola, on cadascun d’ells es pot controlar independentment utilitzant diferents mapes i textures. Els seus atributs de desplaçament (scroll) també es poden controlar independentment.

Pel que fa als dibuixats de fons, el color 0 es normalment transparent permetent que els fons amb major prioritat es vegin a través dels que tenen menys. Per defecte, el dibuixat dels fons segueix l’ordre 3, 2, 1 i 0; amb el que el fons 0 apareixerà damunt del fons 3, però es pot canviar. Alguns fons també tenen la possibilitat de ser rotats y escalats (és un bon efecte per qualsevol mapa).

Pel que fa a les textures, es poden guardar com mapes de bits de 256 o de 16 colors. Ambdues utilitzen la mateixa paleta però els fons són diferents. Poden utilitzar modes de color diferents. Les paletes de 16 colors funcionen dividint la paleta de 256 colors en 16 paletes de 16 colors cada una. A més, en el mode de 16 colors pots tenir els 256 colors en la pantalla, mentre cada rajola només utilitzi una de les 16 paletes.

Les següents taules resumeixen les característiques de cadascun dels modes rajola.

| | MODE 0 | MODE 1 | MODE 2 |
|---|---|---|---|
| Fons Disponible | **Tots** | **0,1,2** | **2,3** |
| Rotació/Escalat? | **NO** | **Fons 2** | **Fons 2,3** |

**Taula A3.1** *Modes de Pantalla*

| | Numero Màxim de Rajoles | Grandària del mapa (en pixels) | Número de Colors per Rajola | Característiques Especials |
|---|---|---|---|---|
| Rotació/Escalat | **256** | **128x128, 256x256 512x512, 1024x1024** | **256** | **Rotació/Escalat** |
| Fons de text pla | **1024** | **256x256, 512x256 256x512, 512x512** | **256 o 16** | **Intercanvi horitzontal/vertical de rajoles** |

**Taula A3.2** *Característiques de fons*

# A3.3 Registres de rotat i/o escalat

Bàsicament, disposem de 4 registres de 16 bits i codificats en una Aritmètica de Punt Fixa (APF) 8.8 que expliquen els paràmetres per la rotació i escalat per un fons rotable i/o escalable. També hi ha 2 registres de 32 bits codificats en una APF 19.8 anomenats punts de referència que especifiquen el centre de rotació/escalat i/o el desplaçament d’un fons rotable i/o escalable.

## A3.4.1 REGISTRES DE PUNT DE REFERÈNCIA (REG_BG*x*X i REG_Bg*x*Y )

Per aconseguir un desplaçament d’un mapa rotable i/o escalable, disposem dels registres **REG_BG*x*X** i **REG_BG*x*Y**, on ***x*** és un numero de fons (naturalment rotable i/o esclable). Aquests registres, defineixen la coordenada del mapa que es vol coincidir a la coordenada superior esquerre de pantalla (pixel 0,0) o primer scanline.

Si per exemple, tenim un mapa de 512x512 emmagatzemat a VRAM i inicialitzem ambdós registre REG_BGxX a 272 i el registre REG_BGxY a 352, la pantalla mostraria el següent tros de mapa.

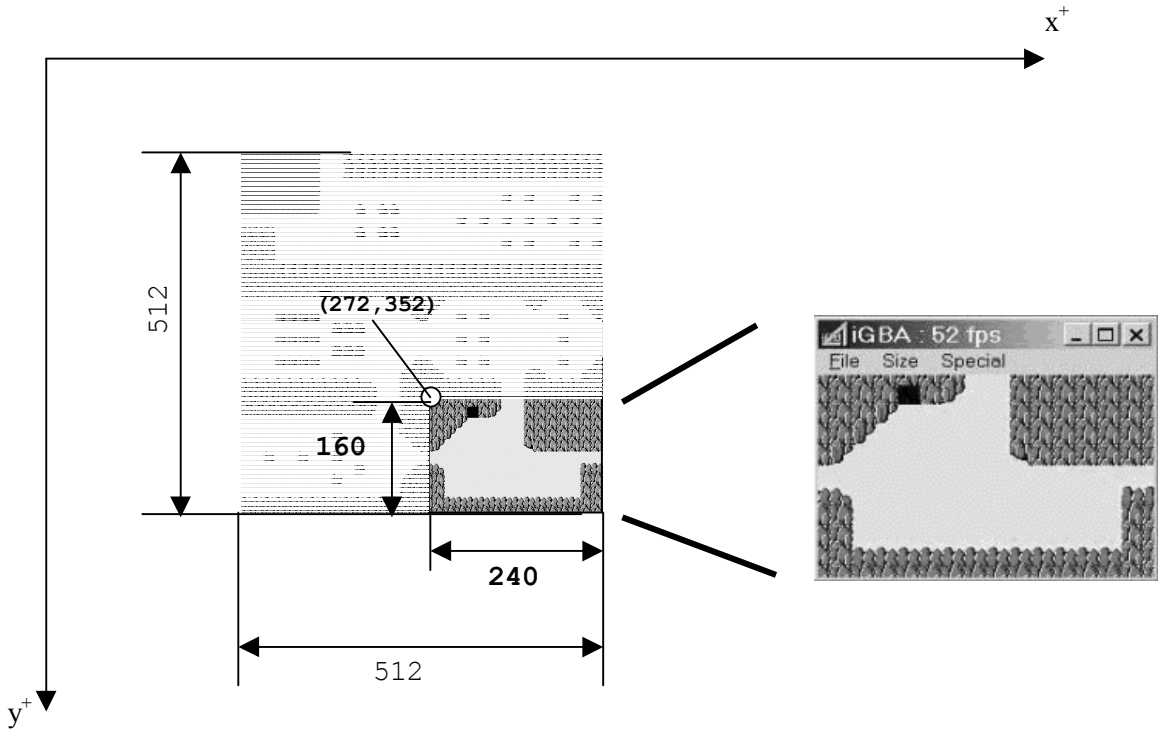



**Figura a3.5** *Mapa de 512x512 pixels*

Aquests registres estan codificats a una APF 19.8, o sigui, 19 bits per la part entera i 8 per la part fraccional.

Els registres de referència són copiats als registres interns del processador gràfic durant cada retraç horitzontal (hblank). Els registres interns són llavors incrementats per els registres BGxPB (dmx) i BGxPD (dmy) (que s'explicaran a continuació) després de renderitzar un scanline. Una important cosa a tenir en compte:

**Si s'escriu qualsevol registre de punt de referència per software fora del període hblank, s'escriurà un nou valor al corresponent registre intern del processador gràfic. Això vol dir que es podrà escriure un offset XY de mapa diferent per-scanline dins el frame actual.**

# A3.4.2 REGISTRES DE ROTACIÓ/ESCALAT (BG*x*PA, BG*x*PB, BG*x*PC i BG*x*PD )

El processador gràfic de la GBA, rota/escalar un fons a partir de 4 paràmetres que es passa als següents registres:

**BG*x*PA- Paràmetre A de rotació/escalat pel fons número *x* (alies dx)**
**BG*x*PB- Paràmetre B de rotació/escalat pel fons número *x* (alies dmx)**
**BG*x*PC- Paràmetre C de rotació/escalat pel fons número *x* (alies dy)**
**BG*x*PD- Paràmetre D de rotació/escalat pel fons número *x* (alies dmy)**

- **Registres BG*x*PA (alies dx) i BG*x*PC (alies dy)**

Quan el processador gràfic de la GBA transforma una línia horitzontal de mapa, dx i dy especifica el pendent i l'augment d'aquesta línia.

Per exemple:

Una línia horitzontal de mapa de longitud 100, amb dx = 1 i dy = 1. El pendent (dy/dx) serà 1, per tant tindrem una resultant línia dibuixada amb una inclinació de 45 º. La longitud de la línia ja no serà 100 (original), sinó que tindrà un augment de $\texttt{sqrt}(100^2 + 100^2) = 141.42$, exactament com ho explica el teorema de Pitàgoras ($a^2 + b^2 = c^2$).

- **Registres BG*x*PB (alies dmx) i BG*x*PD (alies dmy)**

  De igual forma, quan el processador gràfic de la GBA transforma una línia vertical de mapa, dmx i dmy especifiquen el pendent i l'augment d'aquesta línia.

  Si ambdós dmx i dmy configuren el pendent (dx/dy) amb els valors `dmx=-1` i `dmy=1` , amb l'exemple anterior (dx = 1 i dy = 1) significaria rotar 45 º.

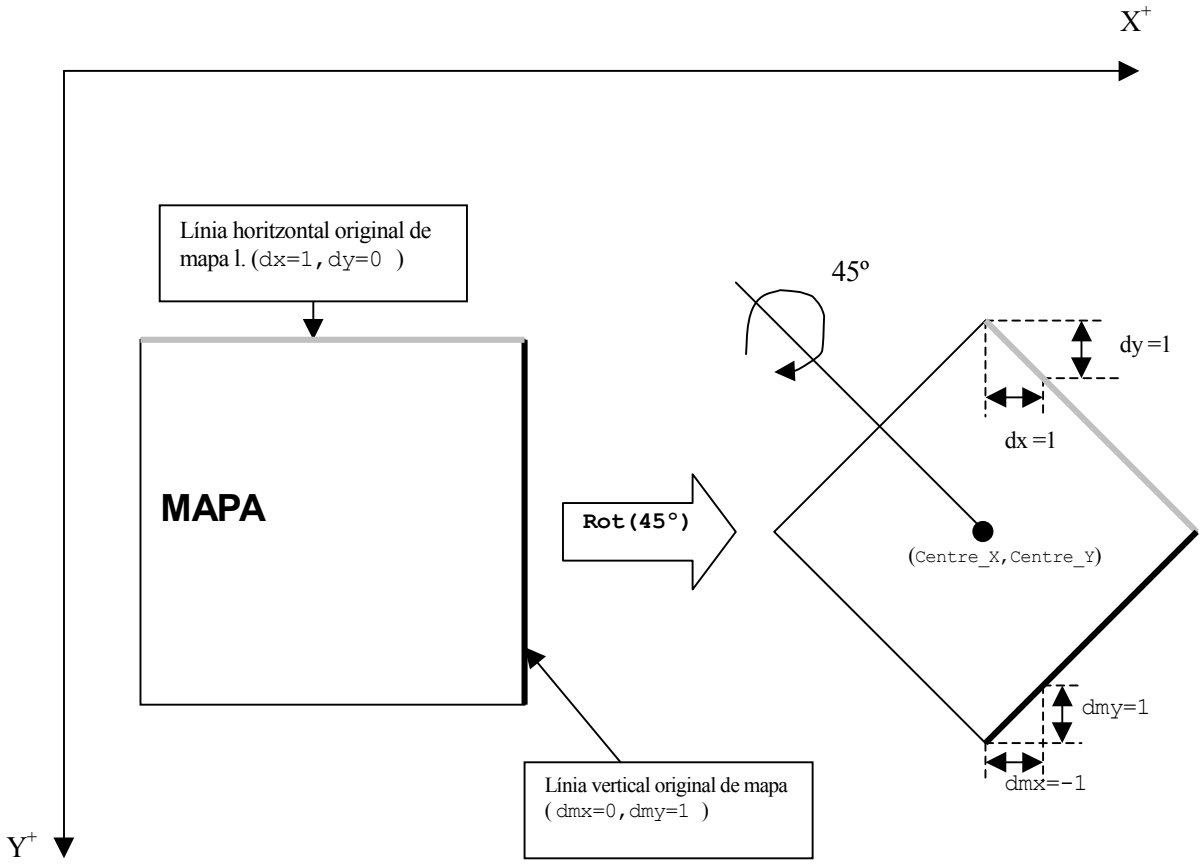



**Figura a3.6** *A l'esquerra, mapa original amb el valor de pendent per totes les línies verticals i horitzontals que el representen. A la dreta, mapa rotat 45º, amb la representació del respectius valors de pendent.*

Per rotar un mapa de forma automàtica, podem trobar el respectiu pendent directament per cada registre amb un angle de rotació α amb les següents formules:

```
PA (dx)  =  cos(α)·Escalat_X
PB (dmx) = -sin(α)·Escalat_Y
PC (dy)  =  sin(α)·Escalat_X
PD (dmy) =  cos(α)·Escalat_Y
```

Per que les funcions sinus i cosinus obtinguin el valor adequat, cal que α compleixi el sentit d'orientacions següents:

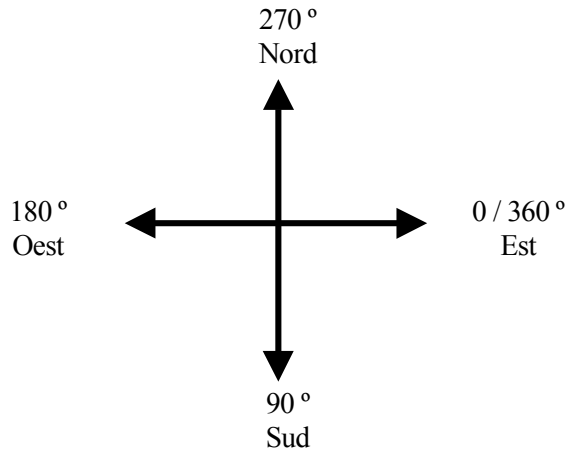



**Figura a3.7** *Orientacions de α*

**Càlcul del centre de rotació**

El centre de rotació es pot calcular aplicant la següent matriu de rotació 2D i el desplaçament del centre rotació:

$$\begin{pmatrix} X\_centre\_rot \\ Y\_centre\_rot \end{pmatrix} = \begin{pmatrix} PosX \\ PosY \end{pmatrix} - \begin{pmatrix} X\_centre \\ Y\_centre \end{pmatrix} \cdot \begin{pmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{pmatrix}$$

$$X\_centre\_rot = PosX - X\_centre \cdot \cos(\alpha) - Y\_centre \cdot \sin(\alpha)$$
$$Y\_centre\_rot = PosY + X\_centre \cdot \sin(\alpha) - Y\_centre \cdot \cos(\alpha)$$

*PosX* i *PosY* expliquen la posició X,Y de mapa que es vol coincidir al primer scanline.

*X_centre* i *Y_centre* defineixen el centre de rotat al mapa respecte el pixel ( $X_P$ i $Y_P$) de pantalla i s'obtenen de la següent forma:

$$\begin{pmatrix} X\_centre \\ Y\_centre \end{pmatrix} = \begin{pmatrix} X_p & 0 \\ 0 & Y_p \end{pmatrix} \cdot \begin{pmatrix} Escalat\_X & 0 \\ 0 & Escalat\_Y \end{pmatrix}$$

$$X\_centre = X_P \cdot Escalat\_X$$
$$Y\_centre = Y_P \cdot Escalat\_Y$$

Per últim s'assignarà els valors *X_centre_rot* i *Y_centre_rot* als registres de punt de referència. En el cas que treballéssim sobre el fons numero 2, s'assignaria les variables *X_centre_rot* i *Y_centre_rot* als registres ***REG_2X*** i ***BG_2Y***.

# A3.4. Estudi de la renderització d'un pla

## A3.4.1 La estructura del mapa

La estructura del mapa serà el vector 1D com el vist a la figura A3.2. El mapa és pot veure com una malla 2D (figura A3.8) on a cada cel·la tenim una referència a la rajola que mapeja la part corresponent al mapa.

El mapa representarà el pla XZ, equivalent al mapa XY de la GBA, per on el nostre jugador és mourà (veure secció següent). Com s'ha donat a entendre a la secció anterior, el mode 1 o 2 permet tenir un fons o espai de mapa de fins a 1024x1024 pixels o unitats. Per aquest estudi, cal tenir un mapa de grandària 512x512 unitats.

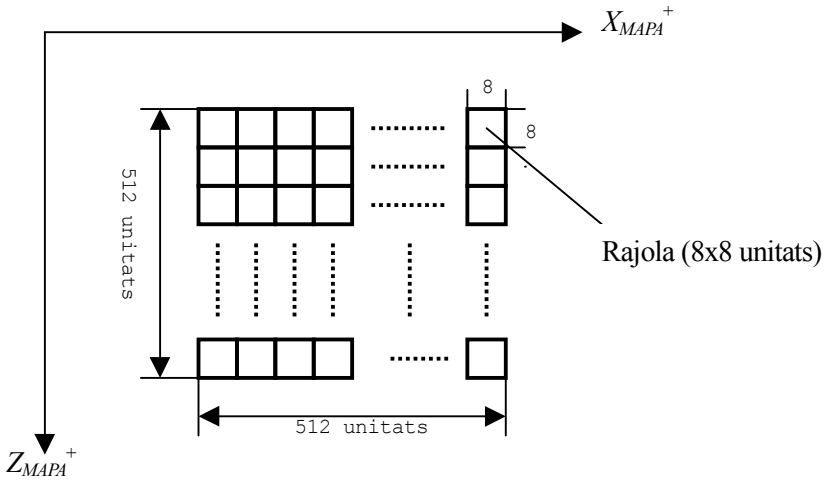
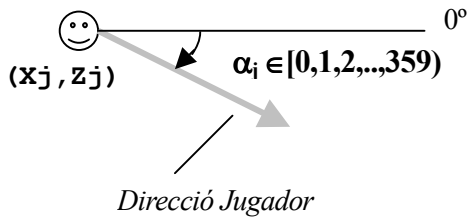



**Figura a3.7** *El mapa és representa com una malla 2D. La grandària de cel·la (rajola) és de 8x8 unitats.*

# A3.4.2 El jugador

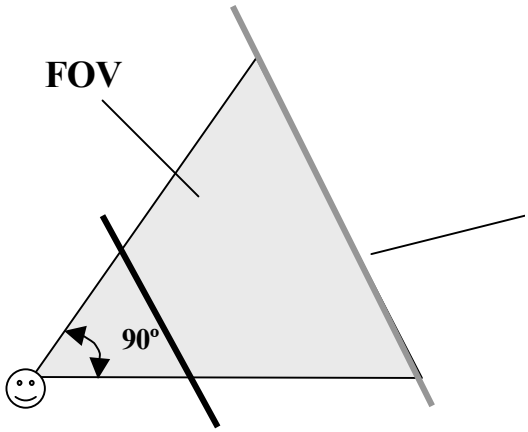
El jugador representa l'element de camera a l'espai 3D, el qual recorrerà en l'espai de mapa explicat a la secció anterior. Definim les coordenades de jugador **Xj** i **Zj** a la posició de mapa actual i $\alpha_j$ com a enter i definit a l'interval 0-359.




Podrem moure el jugador a l'espai de mapa quan l'usuari premi el control direccional endavant o enrera. La llibreria ham, disposa d'unes macros que entrega l'estat del control direccional.

```
if(F_CTRLINPUT_UP_PRESSED) // S'ha premut tecla amunt → el jugador avança
{
        Xj = Xj + Cosinus(AngleJugador)*Velocitat;
        Zj = Zj +   Sinus(AngleJugador)*Velocitat;
}else{
        if(F_CTRLINPUT_DOWN_PRESSED) // S'ha premut tecla avall → el jugador retrocedeix
        {
                        Xj = Xj - Cosinus(AngleJugador)*Velocitat
                        Zj = Zj -   Sinus(AngleJugador)*Velocitat
        }
}

if(F_CTRLINPUT_LEFT_PRESSED) // S'ha premut tecla esquerra → decrementem angle

        if(--AngleJugador < 0) AngleJugador += 360;

else

    if(F_CTRLINPUT_RIGHT_PRESSED) // S'ha premut tecla dreta → incrementem angle
          if(++AngleJugador > 359) AngleJugador -= 360;
```

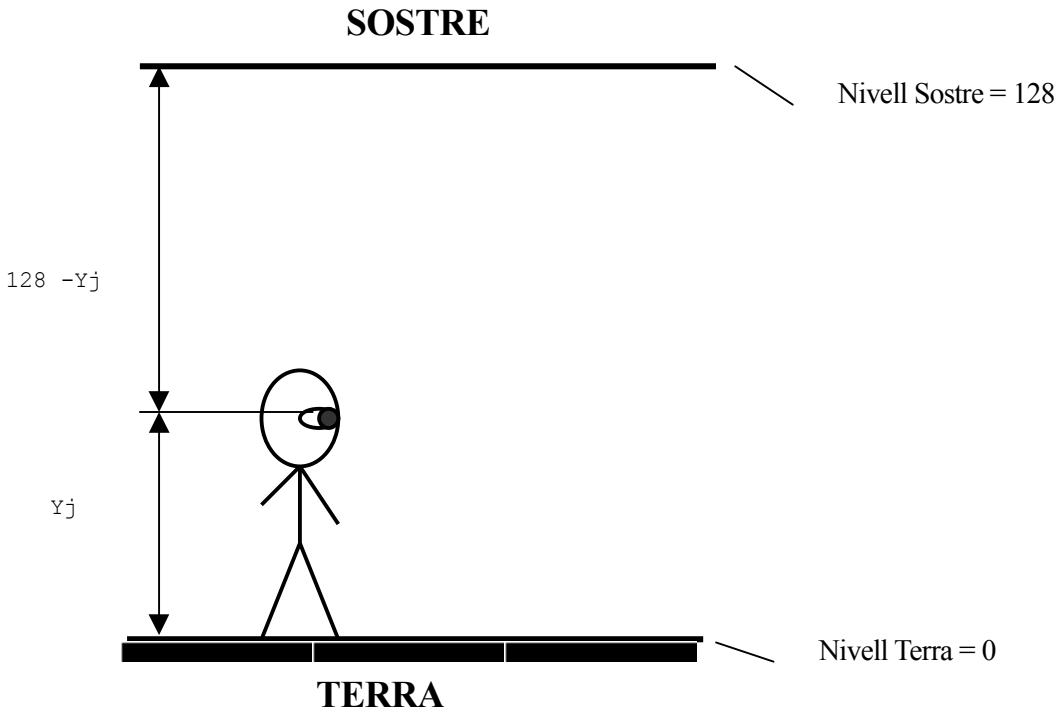
Quan el FOV (Camp de visió) de jugador, s’ha convingut de 90º




Part més llunyana de projecció (horitzó)

Part més propera de projecció (limitat per l’extrem Y del pla projecció)

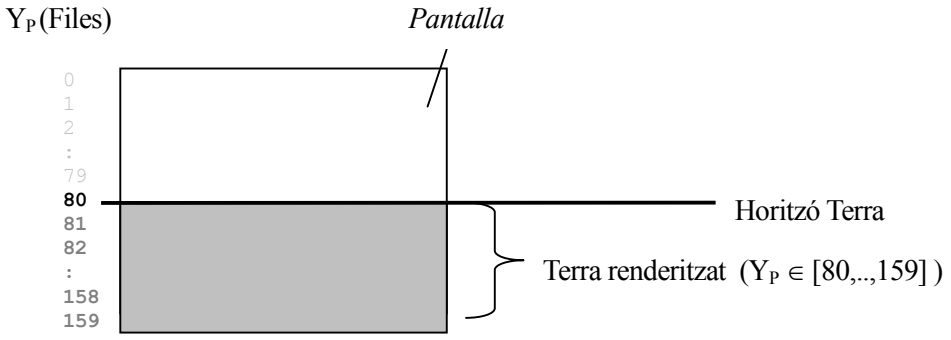
Per últim, l’altura de l’ull de jugador respecte terra que estarà representarà al valor que marqui la variable *Yj*. *Yj* no podrà ser inferior a 0 (Nivell terra) ni superior a 128 (Nivell de sostre) en el món 3D.



# A3.4.3 Càlcul de components per la renderització d’un terra

En la present secció es veurà quines són les components necessàries calcular per renderitzar un terra. La següent subsecció explicarà la obtenció de components d’un pla sense rotat de jugador a l’espai 3D, per l’objectiu d’entendre millor l’obtenció de esmentades components. Més tard, permetrem girs a l’espai 3D, però que el seu càlcul no serà res més que una derivada de les anteriors components (sense gir) calculades.

Els scanlines renderitzats d’un terra al pla de projecció, es pintaran des de el scanline que defineix l’horitzó de terra (fila 80); fins a l’extrem superior del pla projecció (fila 159).



## Càlcul sense gir a l'espai 3D ($\alpha_j = 0º$ )

Si volem renderitzar un pla sense permetre girar al jugador, fixarem la seva posició al centre amb l'angle de direcció a 0º, o també denominat espai d'ull (veure secció 2.1.1.4, del capítol 2, per més informació sobre l'espai d'ull). Com que tindrem una profunditat Z (Distància) constant, implica la renderització del pla a basat en tires de mapa. Per renderitzar una tira de mapa , només cal fer el càlcul de un valor de escalat en Z i un desplaçament a l'espai de mapa. Partim de la figura A3.8 per estudiar els càlculs que implica la renderització d'una tira de mapa,

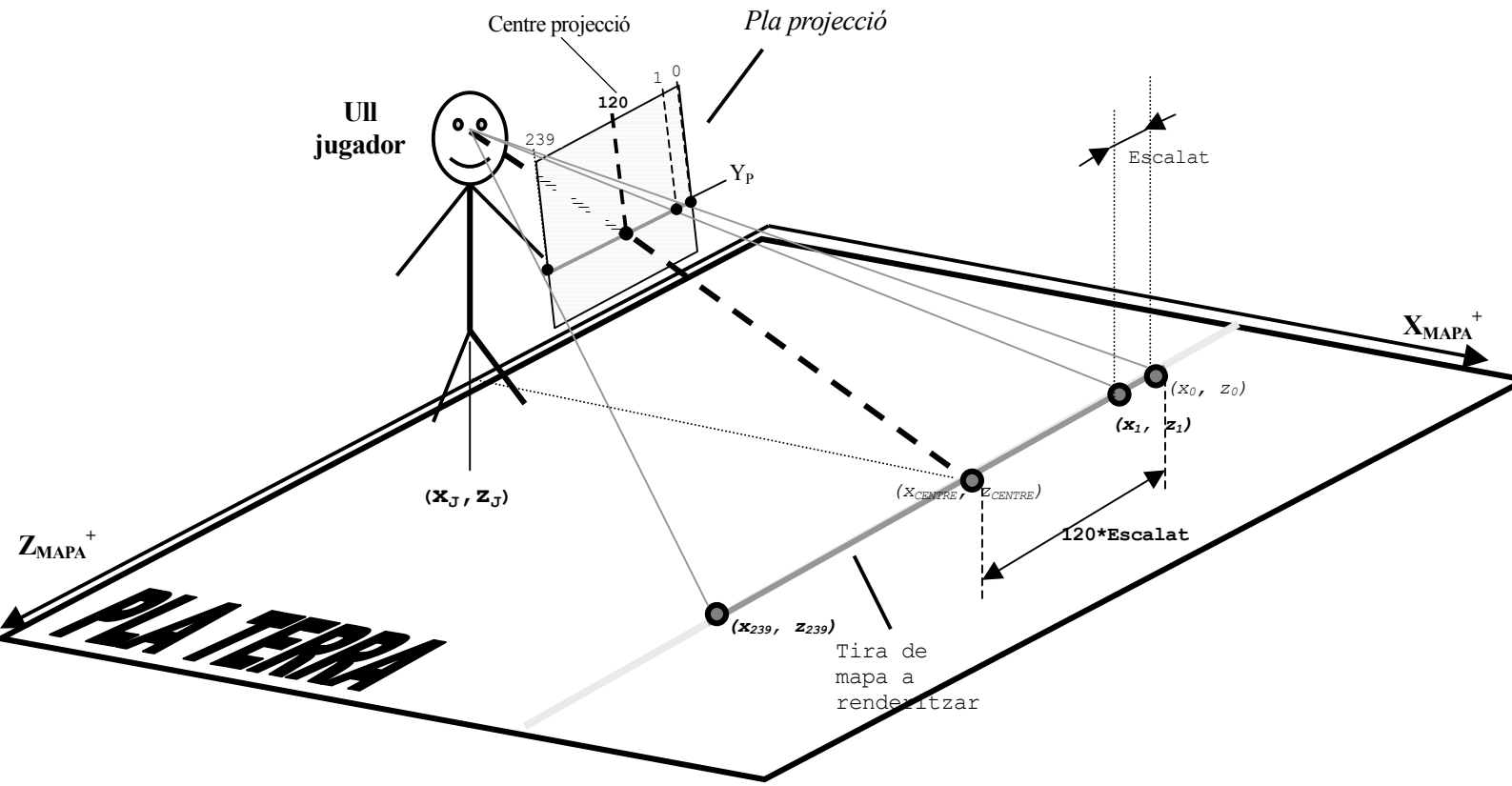



**Figura a3.8**

El nostre objectiu és trobar la coordenada inicial de mapa ( $x_0$, $z_0$) projectat al primer pixel del *scanline* ($Y_P$) i el coeficient de escalat. El procediment per trobar-lo és el següent:

1. Trobem les coordenades de mapa al centre de projecció del scanline ($X_{CENTRE}$, $Z_{CENTRE}$) respecte jugador.
2. Calculem el coeficient de variació constant (`Escalat`). L'Escalat representa l'increment a l'espai de mapa equivalent a cada increment unitari en X del pla projecció.
3. Es resta 120 vegades el coeficient de escalat per trobar les coordenades inicials a l'espai de mapa ($x_0$, $z_0$).

# 1. Càlcul de $X_{CENTRE}$, $Z_{CENTRE}$

A la figura A3.8 podem observar que $Z_{CENTRE}$ coincideix amb la coordenada de jugador, per tant, respecte el jugador $Z_{CENTRE}$ és $Z_j$.

$$Z_{CENTRE} = Z_J$$

S'estudiarà la següent figura per trobar $X_{CENTRE}$,

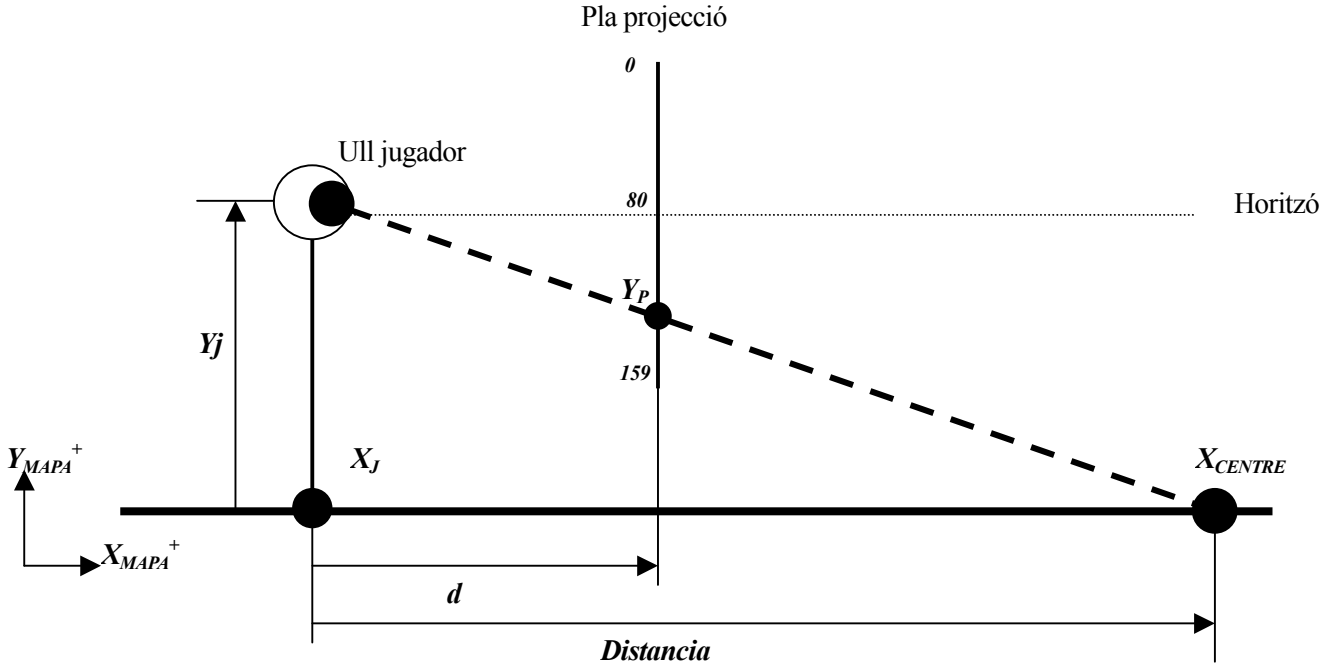



**Figura a3.9**

Observant la figura A3.9 podem trobar $X_{CENTRE}$.

$$X_{CENTRE} = X_J + Distancia$$

Per trobar el valor de *Distancia*, amb la figura A3.9 s'empra similitud de triangles,

$$\frac{Y_P}{d} = \frac{Y_J}{Distancia}$$

Aïllant,

$$Distancia = d \cdot \frac{Y_J}{Y_P}$$

*Yp* està respecte l'horitzó o centre de projecció Y (80), per tant

$$Distancia = d \cdot \frac{Y_J}{Y_P - 79}$$ **Equació a3.1**

*d* representa la distància entre jugador i el pla de projecció. Si sabem l'amplada de pantalla (240) i el camp de visió (o en anglès *Fiel Of View* –FOV-) de 90º en que es treballa, trobem *d*.

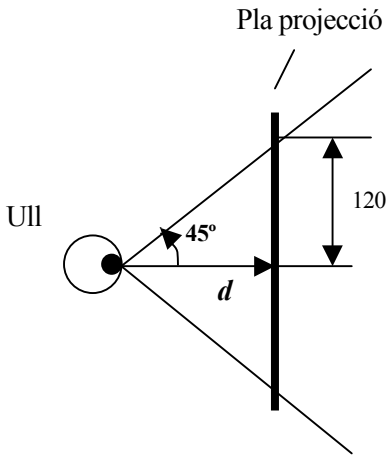



Figura a3.10

Aplicant trigonometria a la figura A3.10, la constant *d* s'aconsegueix com,

$$\tan(45°) = \frac{120}{d} \Rightarrow d = \frac{120}{\tan(45°)} = 120$$

## 2. Càlcul de l'escalat

L'escalat explica quina variació que es té a l'espai de mapa a cada increment unitari al pla projecció. Podem trobar el valor de escalat estudiant la següent figura.

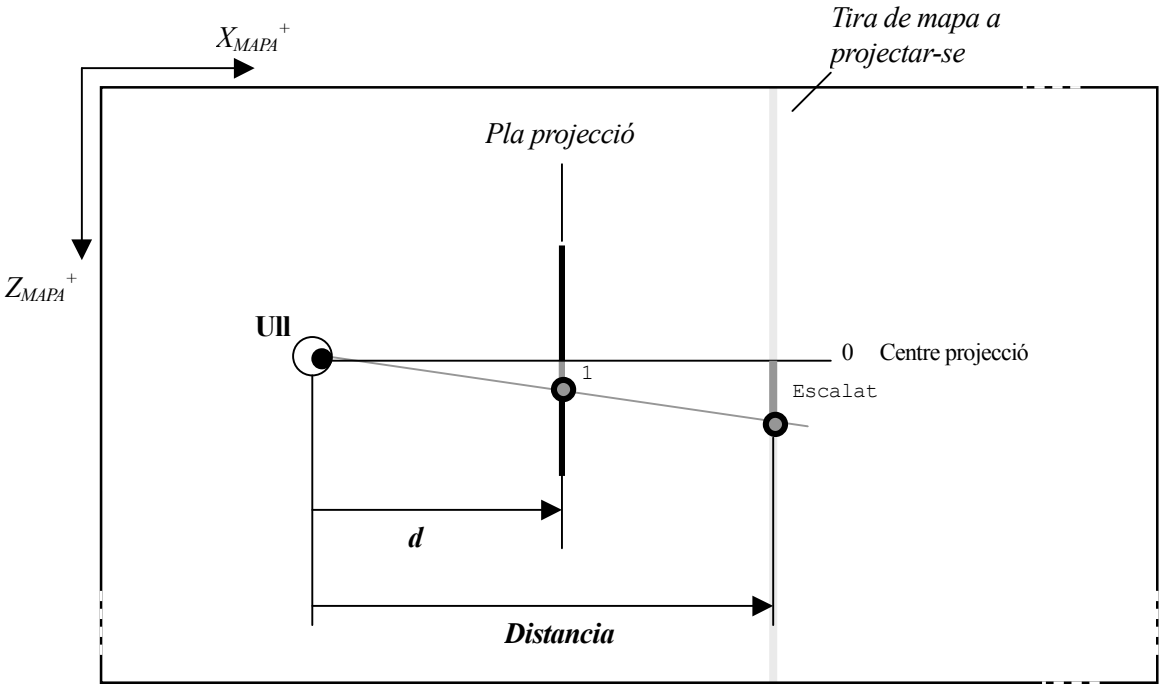



**Figura a3.11** *Mapa vista per sobre*

Amb la figura A3.11, s'emprem teoria de similitud de triangles,

$$\frac{1}{d} = \frac{\text{Escalat}}{\text{Distancia}}$$

Aïllem la variable Escalat i el trobem,

$$\text{Escalat} = \frac{\text{Distancia}}{d} \qquad \textit{Equació a3.2}$$

## 3. Càlcul de la coordenada inicial de mapa ($x_0$, $z_0$)

Sabem les coordenades centrals podem trobar l'offset inicial de mapa, restant 120 vegades el Escalat_Z a $Z_{CENTRE}$. Com que no hi ha variació en X de la tira de mapa a renderitzar, $X_0$ serà directament $X_{CENTRE}$.

$$Z_0 = Z_{CENTRE} - 120 \cdot Escalat = Z_J - 120 \cdot Escalat$$

$$X_0 = X_{CENTRE} = X_J + Distancia = X_J + d \cdot \frac{Y_J}{Y_P - 79}$$

## Càlcul amb gir $\alpha_j$ a l'espai 3D

Per renderitzar un pla permetent el gir de jugador, però igualment al seu centre, treballarem en forma similar a l'estudiat en la secció anterior. Es veu clar que ara no sempre mapejarem tires de mapa. No obstant, donat que és manté la característica de tenir una profunditat Z constant, mapejarem el mapa amb un dda's (*Digital Diferential Analizer*) a l'espai de mapa. Per detallar el que s'ha explicat partim de següent figura,

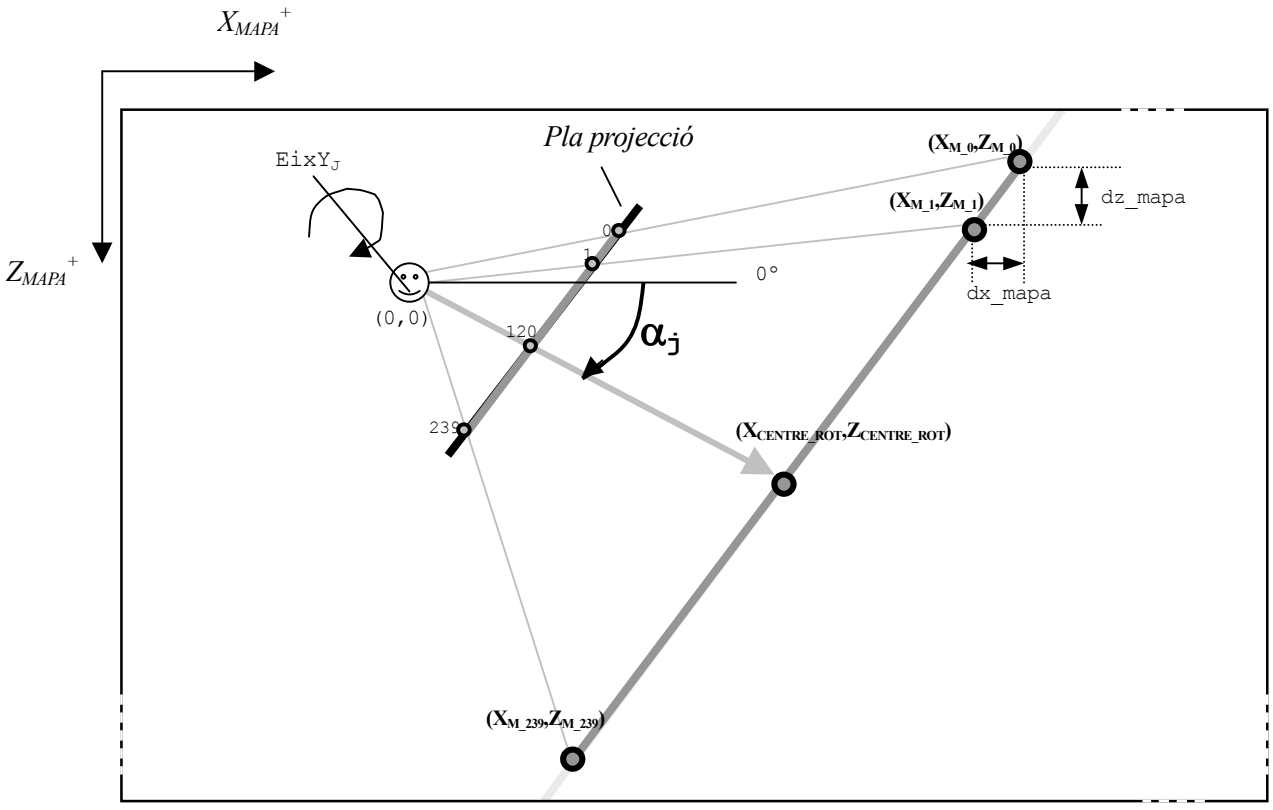



**Figura a3.12** *Mapa amb vista per sobre*

Un altre cop, el nostre objectiu està en trobar la coordenada inicial de mapa ( $x_{M\_0}$, $z_{M\_0}$) i les deltes constants XZ. El procediment per trobar-lo és el següent:

1. Havent obtingut les coordenades $X_{CENTRE}$, $Z_{CENTRE}$ a la secció anterior, s'aplicarà una rotació $\alpha_j$ respecte la coordenada Y de jugador i s'obtindran les coordenades $X_{CENTRE\_ROT}$, $Z_{CENTRE\_ROT}$.

2. El següent pas serà trobar els valors de desplaçament delta X i Z (dz_mapa i dx_mapa) constants al espai de mapa.

# 1. Càlcul de les coordenades ($X_{CENTRE\_ROT}$, $Z_{CENTRE\_ROT}$)

Per trobar les coordenades $X_{CENTRE\_ROT}$, $Z_{CENTRE\_ROT}$, cal aplicar una rotació $\alpha_j$ respecte l'eix Y de jugador a les coordenades $X_{CENTRE}$, $Z_{CENTRE}$ trobades anteriorment.

$$\begin{pmatrix} X_{CENTRE\_ROT} \\ 0 \\ Z_{CENTRE\_ROT} \end{pmatrix} = \begin{pmatrix} X_{CENTRE} \\ 0 \\ Z_{CENTRE} \end{pmatrix} \cdot Rot(EixY_J, \alpha_J)$$

La matriu de rotació Rot(Y, $\alpha_j$) és la següent.

$$Rot(EixY_J, \alpha_J) = \begin{pmatrix} \cos(\alpha_J) & 0 & -\sin(\alpha_J) \\ 0 & 1 & 0 \\ \sin(\alpha_J) & 0 & \cos(\alpha_J) \end{pmatrix}$$

Amb la qual cosa es troba els valors $X_{CENTRE\_ROT}$, $Z_{CENTRE\_ROT}$.

$$\begin{pmatrix} X_{CENTRE\_ROT} \\ 0 \\ Z_{CENTRE\_ROT} \end{pmatrix} = \begin{pmatrix} X_{CENTRE} \\ 0 \\ Z_{CENTRE} \end{pmatrix} \cdot Rot(EixY_J, \alpha_J) = \begin{pmatrix} X_{CENTRE} \\ 0 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} \cos(\alpha_J) & 0 & -\sin(\alpha_J) \\ 0 & 1 & 0 \\ \sin(\alpha_J) & 0 & \cos(\alpha_J) \end{pmatrix} = \begin{pmatrix} \cos(\alpha_J) \cdot X_{CENTRE} \\ 0 \\ \sin(\alpha_J) \cdot X_{CENTRE} \end{pmatrix}$$

$Z_{CENTRE} = 0$

$$X_{CENTRE\_ROT} = \cos(\alpha_j) \cdot X_{CENTRE} \qquad \textbf{Equació A3.3}$$

$$Z_{CENTRE\_ROT} = \sin(\alpha_j) \cdot X_{CENTRE} \qquad \textbf{Equació A3.4}$$

## 2. Càlcul dels increments Delta (`dz_mapa, dx_mapa`)

Les deltes XZ són constants i es desplacen de forma lineal pel mapa. El seu càlcul l'estudiarem amb la següent figura.

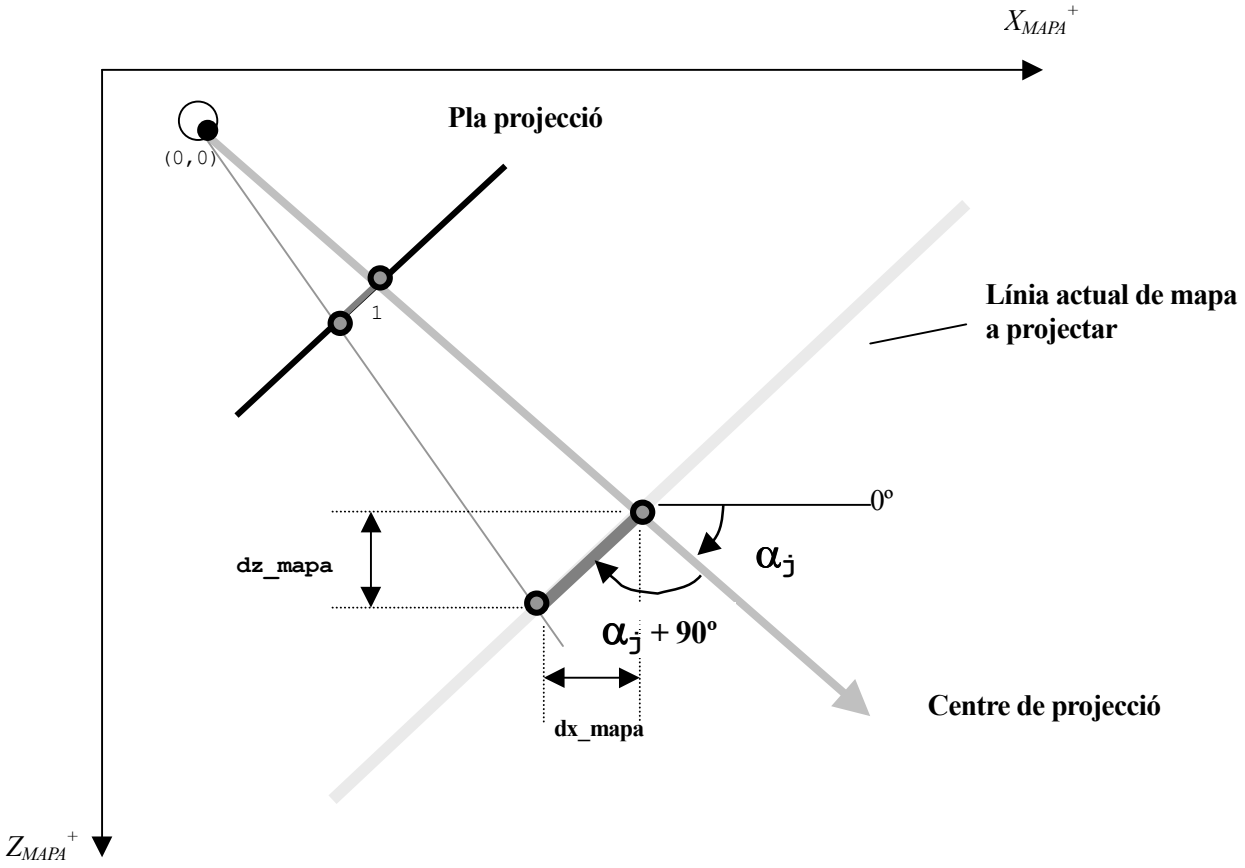



**Figura a3.13**

Per mapejar una línia de mapa com el de la figura A3.13, les deltes *dx_mapa* i *dz_mapa* són constants i per tant consten d'un pendent amb una velocitat de desplaçament. La velocitat de desplaçament ve marcada per el valor de escalat trobat a la secció anterior. A la figura A3.13, per trobar els pendents de les deltes s'ha de trobar el respectiu angle de direcció.

Es pot comprovar que l'angle direcció dels pendents és perpendicular a l'angle del jugador, per tant trobem els valors dels increments delta com:

$$dx\_mapa = cosinus(\alpha_j + 90°) \cdot Escalat = \mathbf{-sinus(\alpha_j) \cdot Escalat} \qquad \textbf{Equació a3.5}$$

$$dz\_mapa = \quad sinus(\alpha_j + 90°) \cdot Escalat = \mathbf{cosinus(\alpha_j) \cdot Escalat} \qquad \textbf{Equació a3.6}$$

## 3. Càlcul de la coordenada inicial de mapa ($x_0$, $z_0$)

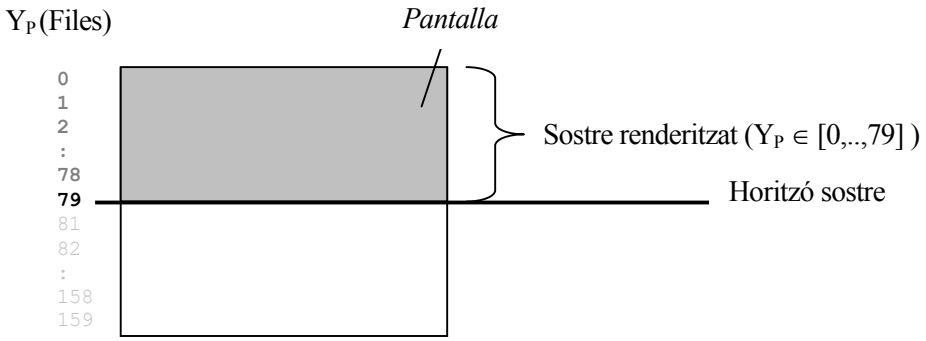
Sabem les coordenades centrals podem trobar l'offset inicial de mapa, restant 120 cops els gradients *dx_mapa* i *dz_mapa* a la coordenades $X_{CENTRE\_ROT}$ i $Z_{CENTRE\_ROT}$

$$Z_{\_M0} = X_{CENTRE\_ROT} - 120 \cdot dx\_mapa \qquad \textbf{Equació A3.7}$$

$$X_{\_M0} = Z_{CENTRE\_ROT} - 120 \cdot dz\_mapa \qquad \textbf{Equació A3.8}$$

# A3.4.4 Càlcul de components per la renderització d'un sostre

La renderització d'un sostre és semblant a la renderització d'un terra. L'únic a tenir en compte que un sostre renderitzat comprendrà el scanline primer de la pantalla (Fila 0) fins al scanline 79, que defineix el seu horitzó.




Respecte dels càlculs realitzats a la secció anterior per un terra, els càlculs seran els mateixos. L'únic a considerar és el càlcul de distància (equació A3.1), que s'haurà de modificar el denominador perquè $Y_p$ estigui respecte el centre de projecció. Com que el nivell de sostre és 128 caldrà trobar la distància respecte l'altura de jugador en el numerador.

$$Distancia = d \cdot \frac{128 - Y_J}{80 - Y_P}$$

# A3.4.5 Algorisme de renderitzat terra i sostre

Havent estudiat la manera d'obtenir càlculs que implica el renderitzat d'un pla, ja sabem l'algorisme necessari per renderitzar el terra i sostre per software.

```
Accio RenderitzaSostreTerra(e Xj,Yj,Zj,Angle:enter)
{ PRE: Xj i Zj ∈ [0,..,1024]; Angle ∈ [0°, 1°, 2°,..,360°] }
var
        X_centre,X_centre_rot, Z_centre_rot, dx_mapa, dz_mapa, Escalat_Z, x0, z0: real

fvar

        per i des de 0 a 159 pas 1 fer

                si i < 80 llavors { Calculem distancia pel terra}

                        Distancia ← d*Yj div (i-79)

                Altrament { Calculem distància pel sostre }

                        Distancia ← d*(128-Yj) div (80-i)
                Fsi


                X_centre_rot ← cos(Angle)*Distancia + Xj
                Z_centre_rot ← sin(Angle)*Distancia + Zj

                Escalat_Z ← Distancia div d
                dx_mapa ← -sin(angle)*Escalat_Z
                dz_mapa ← cos(angle)*Escalat_Z

                x0 ← X_centre_rot  - 120*dx_mapa
                z0 ← Z_centre_rot  - 120*dz_mapa

                RenderitzaScanline(0,239,i,z0,x0,dx_mapa,dz_mapa)
        fper

faccio


accio RenderitzaScanline(e x1,x2, y:enter; x0, z0,dx_mapa,dz_mapa:real)
var
        Npixels, x :enter
        u, v, x_mapa, z_mapa: real
fvar

        Npixels ← x2 – x1 + 1
        x_mapa ← x0
        z_mapa ← z0

        Si Npixels ≥ 1 llavors { Es renderitza scanline }

                Repetir
                        {recordem que la rajola és de 8x8 pixels, per tant s'ha de
                         realitzar mòdul 8 a amplada i alçada per no sortir del sus
                         límits}

                        u ← X_mapa mod 8
                        v ← Z_mapa mod 8

                        PosarPixel(x,y,textura[u][v])

                        x_mapa ← x_mapa + dx_mapa
                        z_mapa ← z_mapa + dz_mapa

                        Npixels ← Npixels – 1

                Mentre Npixels > 0
        Fsi
FAccio
```

**Llistat a3.1**

# A3.5 Adaptació del "mode 7" a la GBA

Havent estudiat la teoria 3D necessària pel renderitzat d'un pla per-software, aquesta secció s'explicarà com **enganyar** la GBA perquè el renderitzi per-hardware, basant-nos amb l'algorisme de la funció *RenderitzaSostreTerra()* anteriorment estudiada. (llistat A3.1)

La renderització tindrà lloc durant el retraçat d'un *scanline*. El retraçat d'un *scanline* consisteix d'un període de 1004 cicles de pintat (*HDraw*) seguit d'un període de 228 cicles que no es fa res (*HBlank*). Durant el període de pintat el hardware gràfic processa les dades de fons i els objectes (sprites) a la pantalla, mentre que **durant el període *HBlank* és deixa lliure perquè codi de programa pugui modificar els atributs de fons i/o d'objectes**.

Es aquí on es tindrà el cor de la renderització, es a dir, **la rutina d'interrupció *Hblank* s'encarregarà de modificar els registres de punt de referència, rotació i/o escalat** (explicats a la secció A3.3) **adequadament per-scanline, aconseguint així, mostrar el pla 3D**.

## A3.5.1 Rutina d'interrupció *HBlank*

### Implementació de la rutina *HBlank*

Fent referència a l'algorisme de funció *RenderitzarSostreTerra()* del llistat A3.1, explicarem com compatibiltzar-lo a la rutina *HBlank*, així com fer l'assignació de registres dins d'ell.

- **Compatibilització de l'algorisme a la rutina d'interrupció *hblank* registres:**

  Tindrem el mateix flux de codi a la rutina *hblank*, excepte la crida de funció `RenderitzarScanLine()`, la qual es substituirà per assignació de registres hardware. No obstant, s'ha de tenir en compte el següent.

1. Donat que en una rutina d'interrupció no s'hi passen paràmetres en mode usuari, es necessari declarar les variables de jugador (`Xj`, `Yj`, `Zj` i `AngleJugador`) de forma global. Així mateix, no les podrem modificar durant el període *HBlank*, perquè causaria una mala renderització del pla, per tant, caldrà un control on aquest serà a la rutina d'interrupció *VBlank*, la qual s'executa quan el hardware gràfic ha acabat de retraçar tota la pantalla.

2. El comptador de scanlines "i" no tindrà efecte dins l'algorisme, perquè el comptat de scanlines es fa per hardware. La llibreria HAM inclou una macro que entrega el numero de scanline que actualment s'està tractant anomenat *V_CURRENT_SCANLINE* definit a l'interval [0..228] on, d'aquest interval, [0..159] és el període de dibuixat de pantalla (*VDraw*) i [160..228] és el període *VBlank*.

- **Assignació de valors *dx_mapa* i *dz_mapa* a registres:**

A l'estudi de la renderització d'un pla s'ha demostrat que una línia de mapa renderitzada a pantalla representa un DDA a l'espai de mapa. Per tant, només caldrà utilitzar els registres *dx*, *dy* per rotar línies horitzontals de mapa o els registres *dmy*, *dmx* per rotar línies verticals de mapa, segons les deltes *dx_mapa* i *dz_mapa* anteriorment calculades. A cada *scanline*, el hardware gràfic utilitza els registres *dmx*, *dmy* per incrementar el registres de punt de referència per tant **no** els utilitzarem pels nostres propòsits. Per tant, els registres a utilitzar seran: **REG*x*_PA (*dx*) i REG*x*_PC (*dy*)**. Cal dir que no caldrà fer cap adaptació dels valors calculats de *dx_mapa* i *dz_mapa* ja que són exactament els valors estudiats a la secció A3.4.4.2 i que, aquests, assignaran els registres *dx* i *dy*.

La llibreria ham disposa de les definicions `R_BG`***x***`ROTDX` i `R_BG`***x***`ROTDY` les quals fan referència a respectius registres segons el fons ***x*** que s'utilitza.

- **Assignació de valors *x0* i *z0* (offset inici de mapa) a registres:**

La última cosa que s'haurà de fer serà l'assignació del offset inicial de mapa (x0,z0), estudiat a la secció anterior (Càlcul amb gir a l'espai 3D), del pròxim scanline a renderitzar, mitjançant els registres de punt de referència (**REG_xX** i **REG_xY**). La llibreria HAM disposa de les macros `R_BG`***x***`ROTX` i `R_BG`***x***`ROTY` que fan referència a respectius registres segons el fons ***x*** que es vol.

Havent considerat els punts anteriors, la implementació final de la rutina *Hblank* és:

```
Accio hbl_Func()
var
   X_centre,X_centre_rot, Z_centre_rot, dx_mapa, dz_mapa, Escalat_Z, x0, z0: real
fvar

        { Entrega del scanline actual }
        i← V_CURRENT_SCANLINE

        si (i ∈ [0,..,159]) llavors { efectuem els càlculs i assignació de registres}

                si i < 80 llavors                       { Calculem distancia pel terra}
                        Distancia ← d*Yj div (i-79)
                Altrament                               { Calculem distància pel sostre }
                        Distancia ← d*(128-Yj) div (80-i)
                Fsi

                X_centre_rot ← cos(Angle)*Distancia
                Z_centre_rot ← sin(Angle)*Distancia

                Escalat_Z ← Distancia div d
                dx_mapa ← -sin(angle)*Escalat_Z
                dz_mapa ← cos(angle)*Escalat_Z

                x0 ← X_centre_rot  - 120*dx_mapa
                z0 ← Z_centre_rot  - 120*dz_mapa

                { Assignació a registres (suposem que es treballa sobre el fons 2) }

                R_BG2ROTX ← enter(x0*256.0)  { donem el valor x0 a APF 24.8 }
                R_BG2ROTY ← enter(z0*256.0)  { donem el valor z0 a APF 24.8 }

                R_BG2ROTDX ← enter(dx_mapa*256.0) { donem el valor dx_mapa a APF 24.8 }
                R_BG2ROTDY ← enter(dy_mapa*256.0) { donem el valor dz_mapa a APF 24.8 }
        fsi
faccio
```

Per iniciar la rutina de interrupció *hblank* amb el codi descrit, s'ha d'executar la següent funció de la llibreria HAM:

```
ham_StartIntHandler(INT_TYPE_HBL,&hbl_Func)
```

# Optimització de la rutina *HBlank*

Donat que es disposa de només 228 cicles ($16{,}48 \cdot 10^{-6}$ segons ) per executar el codi pertinent a la rutina *HBlank*, s'ha de optimitzar al màxim l'execució de codi per no sobrepassar els 228 cicles, altrament es podria arribar a tenir el sistema inestable i, possiblement, penjar la màquina.

## 1a Optimització: Pre-Calcul de operacions trigonomètriques.

Les invocacions a funcions trigonomètriques són molt costoses en temps d'execució. Donat que la variable que representa l'angle de jugador és enter i $\in$ [0,1,2,..,360) podem precalcular els valors trigonomètrics cosinus i sinus en taules globals 360 valors.

## 2a Optimització: Operacions en Aritmètica de Punt Fixa 16.8 (APF 16.8).

La GBA no disposa de coprocessador d'aritmètica real, per tant totes les operacions en reals són emulades per software, la qual cosa li representa un gran cost a la màquina durant l'execució d'aquestes operacions. Així doncs, la segona optimització que podem donar al codi de l'interrupció és l'ús de la Aritmètica de Punt Fixa (APF) 16.8 (consultar annex 5 per més informació). Definim la variable APF (*FIXED*), la qual substituirà qualsevol variable de tipus real, per el tipus codificat APF en C com,

```
#define    FIXED      int    // int = 32 bits
```

## 3a Optimització: Eliminació de la divisió.

Durant l'execució tenim una divisió. Les divisions també són operacions costoses a temps d'execució. No obstant, com que el denominador és sempre enter i definit a l'interval [1,..,80] podem precalcular el seus recíprocs en una taula global, per així executar una multiplicació en ves de la divisió. El resultats els tindrem codificats a una APF 16.16 per tenir més precisió a la part fraccional.

```
reciprocal_div : taula [1,...,80] de FIXED { taula de division reciproques (1/y) codificades a
                                             una APF 8.16 (FIXE és un enter de 32 bits}
Accio CalcularDivisioReciproca()
Var
        i:enter
        divisio_reciproca: real
FVar
        Per i desde 1 a 80 fer
                { divisio reciproca }
                divisio_reciproca  =  (1.0/i)

                { divisió reciproca codificada a APF 8.16}
                devisio_reciproca = divisio_reciproca*65536.0

                { emmagatzament de la divisió reciproca }
                reciprocal_div[i] = enter(divisi_reciproca)
        FPer
FAccio
```

## 4a Aproximacions.

Després de les divisions, les multiplicacions són les també tenen bastant cost d'execució. Al codi tenim algunes divisions i multiplicacions pel valor 120. No obstant, si aproximem el valor a 128 podem executar 7 desplaçaments aritmètics a la dreta per la divisió o a l'esquerra quan es multipliqui.

## 5a Optimització: Portar el codi HBlank a memòria interna (IWRAM)

Per defecte el compilador HAM, situa el codi de programa a memòria externa o a la ROM el qual són regions de memòria lentes. La cinquena optimització que podem fer, és portar el codi *HBlank* a la regió de memòria interna de treball (IWRAM), aprofitant la més ràpida execució de codi en la màquina. Ho podem fer si es declara la directiva de compilador MEM_FUNC_IN_IWRAM abans de la funció ***hbl_Func()*** . En C es faria el següent,

```
MEM_FUNC_IN_IWRAM void hbl_Func()
{
        ...
}
```

## 6a Optimització: Eliminació de accessos a memòria

Els resultats d'un sinus i cosinus estaran en taules globals segons un valor d'angle de jugador. Un accés a una taula global, representa un cost de 2 cicles de tipus no continus (2N): un cicle no continu per carrega el valor de punter de la taula SIN o COS i un altre cicle no continu per retorna el resultat segons el valor d'angle. Un cicle no continu és el tipus d'accés més costós que podem trobar durant l'execució d'un programa a la GBA. Donat els accessos a les taules de SIN i COS s'efectuen amb un angle de jugador en tots els càlculs de la rutina HBlank, podem precarregar els valors SIN i COS a l'inici de programa i així evitar fer més accessos a respectives taules.

Donem la implementació final i optimitzada en C corresponent a la rutina d'interrupció *hblank*, per poder apreciar les operacions en APFs i els desplaçaments aritmètics.

```
MEM_FUNC_IN_IWRAM void hbl_Func(void)
{
       FIXED  X_centre,X_centre_rot, Z_centre_rot, dx_mapa, dz_mapa, Escalat_Z, x0, z0,
              TempC, TempS;

       i = V_CURRENT_SCANLINE;  // { Entrega del scanline actual }

       if (0 <= i && i<=159) // { efectuem els càlculs i assignació de registres}
       {
           //{Pre carrega de valors sinus i cosinus per evitar més accessos a les taules }

              TempS = SIN[AngleJugador];
              TempC = COS[AngleJugador];

              if (i < 80) { Calculem distancia pel terra}
              // Abans:
              //
              //   Distancia = Yj*120/(i-79)
              //
              // Ara:
              //
              //   Distancia = Yj*128/(i-79) = Yj*(2^7)/(i-79) =
              //   = Yj*(2^7)*reciprocal_div[i-79]/256 †   =
              //   = Yj*(2^7 + 2^-8)*reciprocal_div[i-79] =
              //   = Yj*reciprocal_div[i-79]*2^-1 → Distancia = Yj*reciprocal_div[i-79] >> 1
              //

                     Distancia = Yj*reciprocal_div[i-79] >> 1;

              else //{ Calculem distància pel sostre }

                     Distancia = (128-Yj)*reciprocal_div[80-i] >> 1;

              X_centre_rot = TempC*Distancia >> 8;
              Z_centre_rot = TempS*Distancia >> 8;

              Escalat_Z = Distancia >> 7; // Distancia/120 ≅ Distancia/128
              dx_mapa = -TempS*Escalat_Z >> 8;
              dz_mapa = TempC*Escalat_Z >> 8;

              x0 = X_centre_rot  - dx_mapa << 8;
              z0 = Z_centre_rot  - dz_mapa << 8;

              // { Assignacio a registres }

              R_BG2ROTX = x0;
              R_BG2ROTY = x0;

              R_BG2ROTDX = dx_mapa;
              R_BG2ROTDY = dz_mapa;
       fsi
faccio
```

† Cal recordar que la taula de divisions recíproques estan codificats a una APF 8.16. Nosaltres treballem a una APF 16.8 en tota la l'execució de la rutina HBlank, per tant cal dividir entre 256 si volem tenir la corresponent APF 16.8.

## A3.5.2 Rutina d'interrupció *VBlank*

Com s'ha dit, cal que les actualitzacions de jugador es facin fora del període de retraçat de pantalla, altrament es tindria una renderització del pla 3D totalment anòmala, donat que a cada interrupció *HBlank* tindríem coordenades de jugador diferents als del anterior *scanline* renderitzat.

El control de la actualització de jugador tindrà lloc durant el període vertical que no fa res (*VBlank*). El període Vblank esta disponible quan justament s'ha acabat de pintar tota la pantalla, definit a l'interval de *scanline* [160..228]. Per això, si les coordenades de jugador s'actualitzen al període *Vblank*, podem assegurar que l'usuari no podrà modificar els atributs de jugador durant el pintat de scanlines (interval [0..159]).

Una altre cosa a tenir en compte es que, la interrupció *HBlank* esdevé quan ha acabat el pintat d'un scanline. Des de la funció ***hbl_Func()*** **mai es compendrà el primer scanline (que fa referència a sostre)**. Per això, en la rutina *Vblank* s'ha de fer la posta a punt dels registres per el primer *scanline*.

```
MEM_FUNC_IN_IWRAM void vblFunc()
{
   FIXED Distancia,x_centre_rot, z_centre_rot, dx_mapa, dz_mapa, x0, z0, TempC, TempS, Escalat_Z;

    // { Actualització de jugador }

    if(F_CTRLINPUT_UP_PRESSED)   {Zj += SIN[AngleJugador] << 3; Xj += COS[AngleJugador] << 3;}
    if(F_CTRLINPUT_DOWN_PRESSED) {Zj -= SIN[AngleJugador] << 3; Xj -= COS[AngleJugador] << 3;}
    if(F_CTRLINPUT_LEFT_PRESSED) if(--Angle < 0)   Angle += 360;
    if(F_CTRLINPUT_RIGHT_PRESSED if(++Angle > 359) Angle -= 360;
    if(F_CTRLINPUT_L_PRESSED if((Yj+1) < 128)  Yj ++;
    if(F_CTRLINPUT_R_PRESSED if((Yj-1) > 0)    Yj --;

    // { Càlculs i assignació de registres corresponents al primer scanline (sostre)}

    TempC = COS[Angle];
    TempS = SIN[Angle];

    Distancia = (128-Yj)*reciprocal_div[80] >> 1;

    x_centre_rot = (TempC*Distancia >> 8) + Xj;
    z_centre_rot = (TempS*Distancia >> 8) + Zj;

    Escalat_Z = Distancia >> 7;

    dx_mapa = -TempS*Escalat_Z >> 8;
    dz_mapa =  TempC*Escalat_Z >> 8;

    x0 = x_centre_rot - (dx_mapa << 7);
    z0 = z_centre_rot - (dz_mapa << 7);

    R_BG2ROTDX = dx_mapa;
    R_BG2ROTDY = dz_mapa;

    R_BG2ROTX = x0;
    R_BG2ROTY = z0;
}
```

Per iniciar la rutina de interrupció *VBlank* amb el codi descrit, s'ha d'executar la següent funció de la llibreria HAM:

```
ham_StartIntHandler(INT_TYPE_VBL,&vbl_Func)
```

# A3.6 Preparació de una demostració

Aquesta secció pretén preparar una demostració d’un recorregut interactiu en “mode 7”. El primer pas ha estat la construcció de les següents imatges.

**Figura a3.14** *Imatge “nuvols.bmp” de 128x128 pixels i 256 colors*

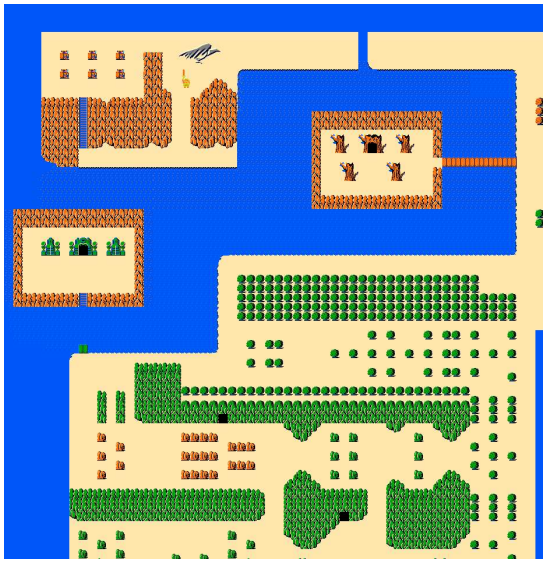


**Figura a3.15** *Imatge “terra.bmp” de 1024x1024 pixels i 256 colors.*

De les quals les imatges A3.14 i A3.15 mapejaran el sostre i terra als fons 2 i 3 respectivament, per tant, caldrà inicialitzar un mode gràfic que permeti 2 fons amb transformació (estem parlant de l’inicialització del mode 2).

Les imatges A3.14 i A3.15 han de ser prèviament convertides a un format de tipus rajola per un fons transformable, amb el vector mapa corresponent i llegible a la GBA. Tot això és possible amb l’execució de l’eina GFX2GBA (annex a2.1.3) i amb els següents paràmetres al directori on tenim les imatges “nuvols.bmp” i “terra.bmp”.

```
gfx2gba -fsrc -m -rs -t8 *.bmp
```

L’eina crearia els següents fitxers:

1. ***master.pal.c*** : conté un vector 1D en C de la paleta global representativa de les dues imatges.

2. ***nuvols.raw.c*** : conté un vector 1D en C de la informació gràfica de les rajoles disponibles i en format cru, corresponent a la imatge “nuvols.bmp”.

3. ***nuvols.map.c***: conté un vector 1D en C de la informació de mapa corresponent a la imatge “nuvols.bmp”.

4. ***terra.raw.c***: conté un vector 1D en C de la informació gràfica de les rajoles disponibles i en format cru, corresponent a la imatge “terra.bmp”.

5. ***terra.map.c***: conté un vector 1D en C de la informació de mapa corresponent a la imatge “terra.bmp”.

Els 5 fitxers els inclourem al fitxer de codi font principal.

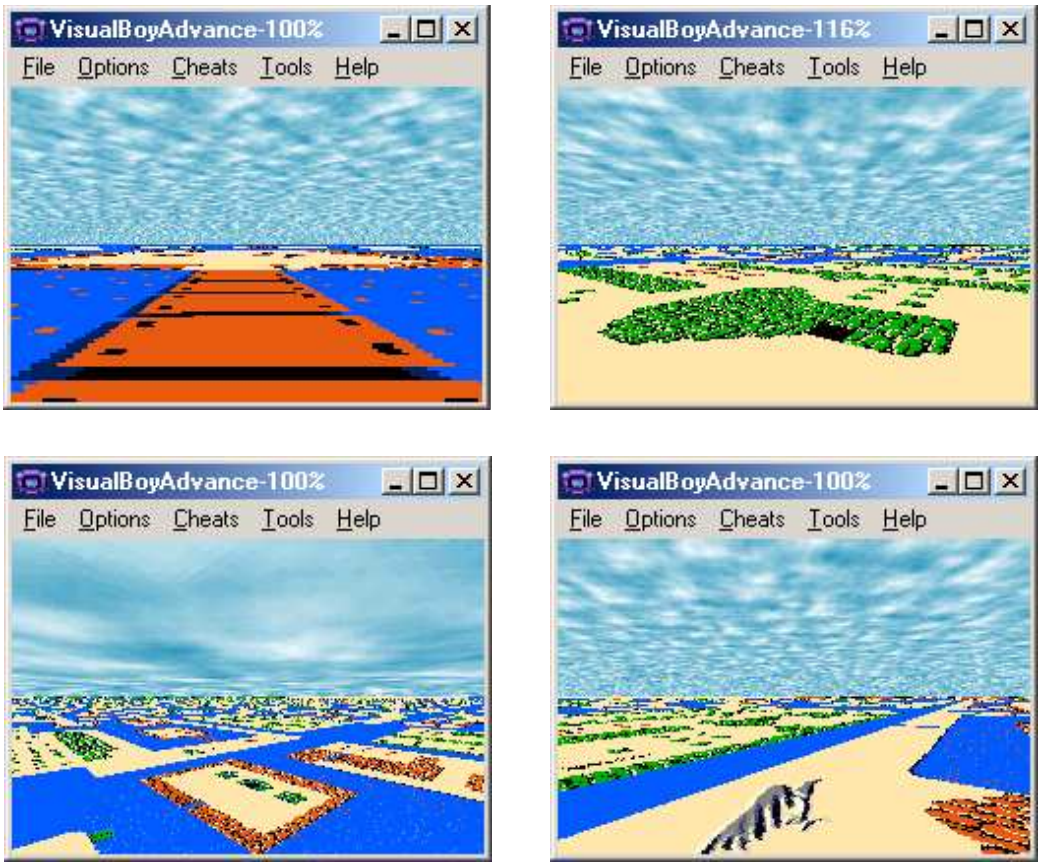
I finalment, donem el codi de la demostració ( per més informació sobre funcions de inicialització ham per un fons de tipus rajola, llegir la secció a3.6 i a3.7 de l’annex 3 ) editat a l’arxiu “main.c”.

main.c

```
#include "mygba.h"

#include "gfx/master.pal.c"  // Paleta global...
#include "gfx/nuvols.raw.c"  // Format cru de les rajoles de la imatge “nuvols.bmp”...
#include "gfx/nuvols.map.c"  // Vector mapa dels “nuvols.bmp”...
#include "gfx/terra.raw.c"   // Format cru de les rajoles de la imatge “terra.bmp”...
#include "gfx/terra.map.c"   // Vector mapa dels “terra.bmp”...
#include "LUTs.c"            // Tenim guardades les taules precalculades SIN,COS i reciprocal_div.

// Variables globals de jugador...
FIXED Xj=0;
FIXED Zj=0;
FIXED Yj = 32;
int AngleJugador = 0;


MEM_FUNC_IN_IWRAM void hblFunc()
{
    FIXED Distancia, x_centre_rot, z_centre_rot, dx_mapa, dz_mapa, x0, z0, TempC, TempS, Escalat_Z;

       int   i = F_VCNT_CURRENT_SCANLINE;

        if((0 <= i && i <= 159))
        {

        TempC = COS[AngleJugador];
        TempS = SIN[AngleJugador];

        if(i >= 80)  Distancia = Yj*reciprocal_div[i-79] >> 1;
        else Distancia = (128-Yj)*reciprocal_div[80-i] >> 1;


         x_centre_rot = (TempC*Distancia >> 8) + (Xj);
         z_centre_rot = (TempS*Distancia >> 8) + (Zj);

         Escalat_Z = Distancia >> 7;

         dx_mapa = -TempS*Escalat_Z >> 8;
         dz_mapa =  TempC*Escalat_Z >> 8;

         x0 = x_centre_rot - (dx_mapa << 7);
         z0 = z_centre_rot - (dz_mapa << 7);

         if(i >= 80){ R_BG3ROTDX = dx_mapa;R_BG3ROTDY = dz_mapa; R_BG3ROTX = x0;R_BG3ROTY = z0;}
         else{ R_BG2ROTDX = dx_mapa;R_BG2ROTDY = dz_mapa;R_BG2ROTX = x0; R_BG2ROTY = z0;}
}


MEM_FUNC_IN_IWRAM void vblFunc()
{
    FIXED Distancia, x_centre_rot, z_centre_rot, dx_mapa, dz_mapa, x0, z0, TempC, TempS, Escalat_Z;

    if(F_CTRLINPUT_UP_PRESSED) {Zj+=SIN[AngleJugador];Xj+=COS[AngleJugador];}
    if(F_CTRLINPUT_DOWN_PRESSED) {Zj-=SIN[AngleJugador];Xj-=COS[AngleJugador];}
    if(F_CTRLINPUT_LEFT_PRESSED)    if(--AngleJugador < 0) AngleJugador += 360;
    if(F_CTRLINPUT_RIGHT_PRESSED)   if(++AngleJugador > 359) AngleJugador -= 360;
    if(F_CTRLINPUT_L_PRESSED && (Yj+1) < 127)   (Yj++);
    if(F_CTRLINPUT_R_PRESSED && (Yj-1) > 0)             (Yj--);

    TempC = COS[AngleJugador];
    TempS = SIN[AngleJugador];
    Distancia = (128-Yj))*reciprocal_div[80] >> 1;

    x_centre_rot = (TempC*Distancia >> 8) + (Xj);
    z_centre_rot = (TempS*Distancia >> 8) + (Zj);

    Escalat_Z = Distancia >> 7;

    dx_mapa = -TempS*Escalat_Z >> 8;
    dz_mapa =  TempC*Escalat_Z >> 8;

    x0 = x_centre_rot - (dx_mapa << 7);
    z0 = z_centre_rot - (dz_mapa << 7);

    R_BG2ROTDX = dx_mapa;R_BG2ROTDY = dz_mapa; R_BG2ROTX = x0;  R_BG2ROTY = z0;

}
```

```
int main(void)
{
	map_fragment_info_ptr mymap2,mymap3;

	// initialize HAMlib
	ham_Init();


	// Inicialitzacio mode rajola amb dos fons rotables i/o escalables
	ham_SetBgMode(2);

	// Configura el fons 2 i 3, perquè es mapegin a pantalla de forma infinita...
	M_BG2CNT_WRAP_SET_ON;
	M_BG3CNT_WRAP_SET_ON;

	// inicialització de la paleta global dels dos fons...
	ham_LoadBGPal(&master_Palette,sizeof(master_Palette));

	// càrrega de les rajoles dels dos fons respectius...
	ham_bg[2].ti = ham_InitTileSet(&nuvols_Tiles,SIZEOF_16BIT(nuvols_Tiles),1,1);
	ham_bg[3].ti = ham_InitTileSet(&terra_Tiles,SIZEOF_16BIT(terra_Tiles),1,1);

	// inicialització de dos mapes buits...
	ham_bg[2].mi = ham_InitMapEmptySet(0,1);
	ham_bg[3].mi = ham_InitMapEmptySet(3,1);

	// Es fa una referència a les dades dels dos mapes en la ROM
	mymap2 = ham_InitMapFragment(&nuvols_Map,16,16,0,0,16,16,1);
	mymap3 = ham_InitMapFragment(&terra_Map,128,128,0,0,128,128,1);

	// Còpia (en aquest cas sencera) els mapes al fons 2 i fons 3 i a les coordenades x=0 y=0
	ham_InsertMapFragment(mymap2,2,0,0);
	ham_InsertMapFragment(mymap3,3,0,0);

	// l'habilitem els dos fons perquè es vegin a pantalla
	ham_InitBg(2,1,1,0);
	ham_InitBg(3,1,1,0);

	// Aquesta funció crea una finestra de 80x240, la qual s’indica que volem veure al seu
	// interior el fons 3 (terra) i al seu exterior el fons 2 (sostre).

	ham_CreateWin(0,0,80,240,160,WIN_BG3,WIN_BG2,0);

	// inicialització de la rutina d'interrupció VBLANK i HBLANK
	ham_StartIntHandler(INT_TYPE_VBL,&vblFunc);
	ham_StartIntHandler(INT_TYPE_HBL,&hblFunc);

    while(1)
    {

    }
}
```

# A3.7 Captures de pantalla de la demostració




# A3.8 Conclusions

Aquest annex ha donat l’estudi i la corresponent implementació d’una simulació de “mode 7” per la GBA i del qual s’ha podem destacar 2 observacions:

1. Per renderitzar un pla 3D per hardware és necessari utilitzar algun dels modes gràfics basats fons transformables (modes 1 o 2).

2. Durant la renderització de cada *scanline* es fa sempre entre les cotes x1=0 a x2=239 (tota la pantalla).

En futurs projectes, es podria aprofitar aquest estudi per un motor de jocs amb perspectiva “semi-3D” (Només plans 3D), com passava amb alguns dels jocs a la SNES (comentats a la secció A3.1).

# A3.9 Treballs futurs

- ## Plans curvats

En futurs treballs, portar l’efecte de plans curvats com l’observat a la figura A3.16. [TH01] explica la possibilitat de generar aquest efecte.

**Figura a3.16** *Captura del joc Terra Enigma on es pot observar que el terra i sostre té un efecte curvat i amb perspectiva.*

- ## Direcció lliure a l’espai YZ

Tenir una direcció 2D lliure en YZ per les renderitzacions “mode 7”, permetrà al jugador pujar abaixar el cap i moure lliurament per esmentat espai. En [TONC04B], s’explica en tota detall la manera d’aconseguir-ho.

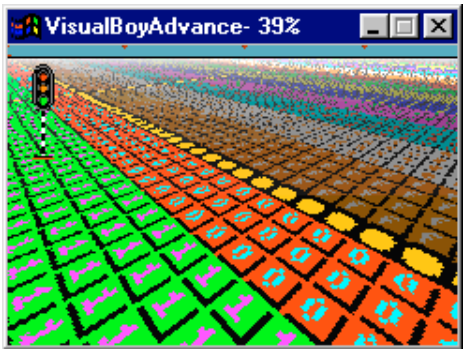



**Figura a3.17**