
SPEED
101
km/h
TIME
22
LEFT
WAX
PERFECT
GOOD
POOR
BAD
R: 1'30"0 - - -
T: 0'35"6
POS.: 3/4
SURF PLANET (1997)
Mt. Cook
EASY
CPU
3

Producción

1. Sistema de captura de movimientos
   a. CAR Sant Cugat
   b. Sistema óptico, cámaras a 90^0
   c. En Candanchú
   d. Dani Fernández (campeón España snowboard)

2. Prueba con Interface físico
   a. Tabla +Lectura con potenciómetro

Arquitectura Síncrona

- **Interrupción de vídeo (VBLANK)**
  - Captura de mandos para todo el frame
  - Reset doble buffer de figuras

```
;************ INTERRUPCION VIDEO ************************************

INTVID: MOVE.W #0,CLRINT
        MOVEM.L D0-D7/A0-A6,-(A7)       ;guarda los registros

;   BSR LEVOTOP

        MOVE.L BUF68,D0         ;comprueba que ha terminado de procesar
        CMP.L PAGULT,D0         ;la pagina. Si no, no provoca int tms.
        BNE.S ESUNO

        MOVE.L BUF68,A0         ;intercambio punteros
        MOVE.L BUFTMS,BUF68
        MOVE.L A0,BUFTMS
                                ;pasa nuevo puntero al tms
        MOVE.L A0,D0
        MOVE.W D0,COMTMS+2
        SWAP D0
        MOVE.W D0,COMTMS

        MOVE.L BUF682,A0        ;intercambio punteros
        MOVE.L BUFTMS2,BUF682
        MOVE.L A0,BUFTMS2
                                ;pasa nuevo puntero al tms
        MOVE.L A0,D0
        MOVE.W D0,COMTMS2+2
        SWAP D0
        MOVE.W D0,COMTMS2

        MOVE #0,PORTINT ;provoca interrup. al TMS
        MOVE #1,PORTINT

ESUNO:
```

Arquitectura Síncrona

- Bucle principal síncrono

Funciones de instanciar objetos

Funciones de poner objetos → final de buffer

Funciones de mover objetos

```
1   GAME                                    ;GAME
2
3
4       TST FLASH
5       BNE.S ASDF
6       MOVE.L #TAB_PON,TAB_SPR ;inicializa puntero a tabla poner
7       BRA SOSDF
8   ASDF    MOVE.L #TAB_POF,TAB_SPR ;inicializa puntero a tabla poner FLASH
    SOSDF
9
10      BSR CREAR   ;decide si debe crear e inicializa estelas,etc.
11      BSR PONER   ;pone surfer, poligono borrado,camara y fig. visibles
12
13      TST FLASH
14      BNE.S NOMMO
15
16      BSR MOVER       ;mueve surfer,etc.
17  NOMMO
18
18          MOVE.L BUF68,D0
19          MOVE.L D0,PAGULT
20
21      TST LINKON
22      BNE.S NOSINC
23      SUB #1,INI_SIN
24      BPL.S NOSINC        ;mira si debe mandar secuencia de sincronizacion
25      MOVE #1000,INI_SIN      ;cada 20 seg.
26      JSR IN_COMM
27  NOSINC
28
29      JSR ENVIA   ;manda datos comunicacion entre placas
30
31      BRA BUKPRI
```

Estructuras de soporte (Rejilla 2D)

Rejilla de celdas 2D de tamaño fijo (256 x 256)

Incluye la información de la **geometría** del mapa

Info para **colisiones**

Figuras **estáticas** del mapa

Rejilla 2D

Geometría del mapa

Versión simple de “Raycast”:

Rejilla 2D: tabla de planos por celda

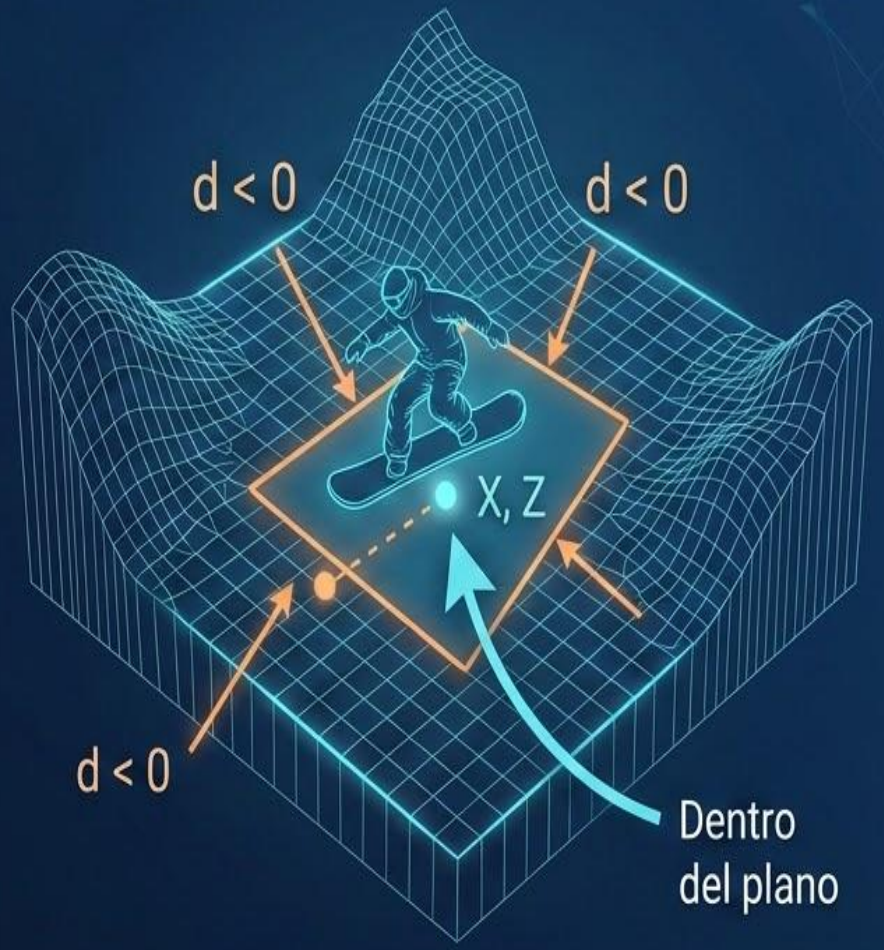
Detección del plano

Recorremos lista de polígonos / aristas, evaluando X / Z surfer

Ecuación de la recta 2D de cada arista (A·x + B·z + C); si para todas aristas d < 0 → dentro del plano

$$d = \frac{|Ax_0 + Bz_0 + C|}{\sqrt{A^2 + B^2}}$$

(*) No DIV ni ABS (vectores unitarios)!



Rejilla 2D

Físicas

- Acoplamos al suelo al surfer (vector Y surfer = N polígono). "copiando" el terreno"
- **Colisiones** con terreno:
  - Para todos planos, la 1ª arista en memoria es la de choque!
  - Se evalua producto escalar DIR y N arista
    - Si es negativo, cos < 0 → ángulo > 90°
    - Si es cero o positivo ángulo <= 90°
- **Tipos de terreno** (lago, slide, pre-salto, rozamiento, paleta sombra...)

Rejilla 2D: Culling y Figuras del Mapa

Figuras del mapa

- Ventana visible tamaño fijo (ANCHO x LARGO) desde la celda de la cámara (“far clip”)

Culling 2D:

1. Calcular Vector (Cámara → Bloque)
2. Evaluar Producto Escalar (con DIR Cámara)

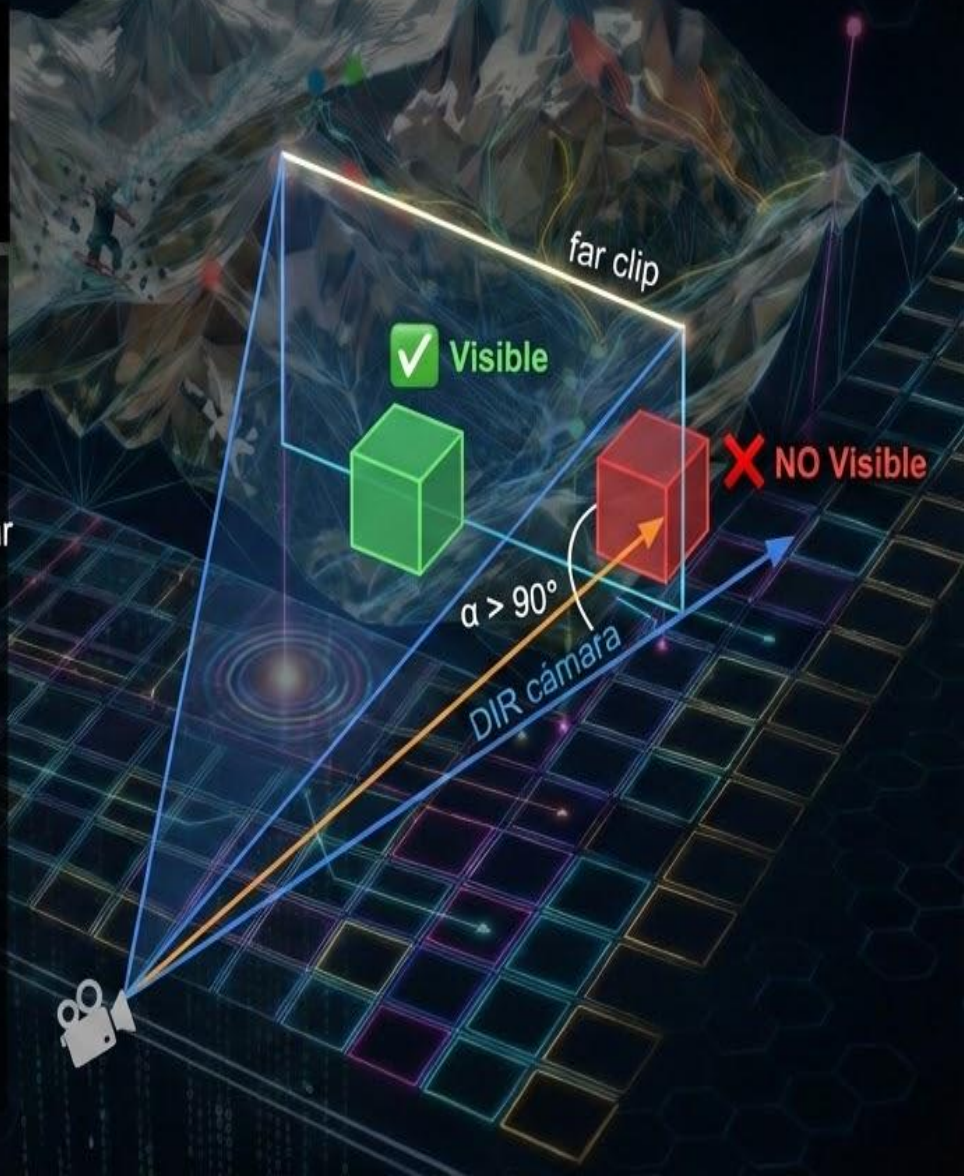
$$V_a \cdot V_b = (X_a \cdot X_b) + (Z_a \cdot Z_b)$$

Si < 0
(cos α < 0, α > 90°)

NO Visible

Si ≥ 0

Visible



Matemáticas de Punto Fijo

Formato

- Siempre vectores unitarios
- Formato punto fijo 16 bits:

Matemáticas de Punto Fijo

Precisión e indeterminación

Nunca calculamos ángulos:

- Divisiones por 0
- “Jittering” figuras

Solución: Vectores Unitarios

Las componentes vectores unitarios contienen cos / sen necesarios para transformaciones

Matemáticas de Punto Fijo

Alineamiento con Z

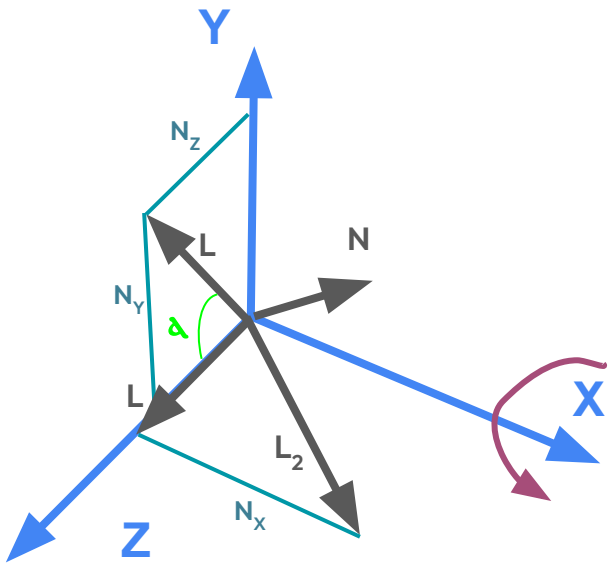
- Evitamos **rotaciones 3D**:
- Alineábamos el **vector N** con el **eje Z** → **geometría 2D** (plano XY)
- Para alinear: dos **rotaciones XY ó YX** dependiendo componentes vector
- Hacíamos las transformaciones 2D y luego devolvíamos vector al espacio 3D aplicando rotaciones en **orden inverso**

Matemáticas de Punto Fijo

Ej: Giro 2D vector DIR surfer. Alineamos su normal (N) con eje Z

- Paso 1: rotación en eje X.
  - Cálculo cos/sen α mediante el vector L = (N_y, N_z)
  - Obtenemos el vector intermedio L_2 en el plano XZ, usando la matriz de rotación en X:

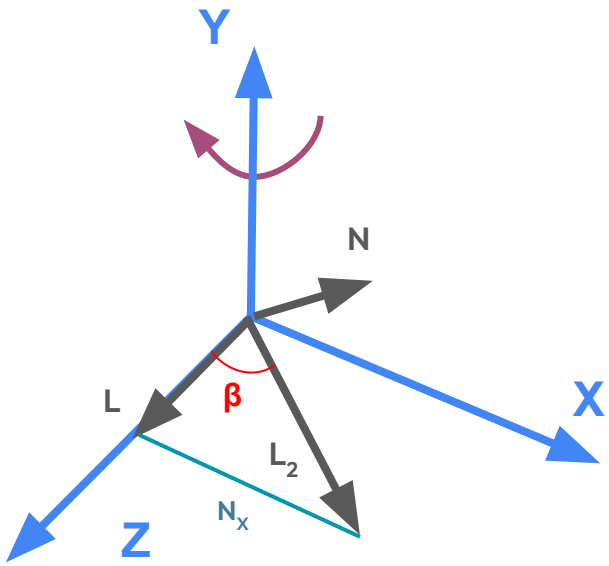
$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$



Matemáticas de Punto Fijo

- Paso 2: rotación en eje Y.
  - Cálculo cos/sen β mediante componentes del vector $L_2 = (N_x, L)$
  - Obtenemos N alineado en Z rotando L_2 con la matriz de rotación en Y:

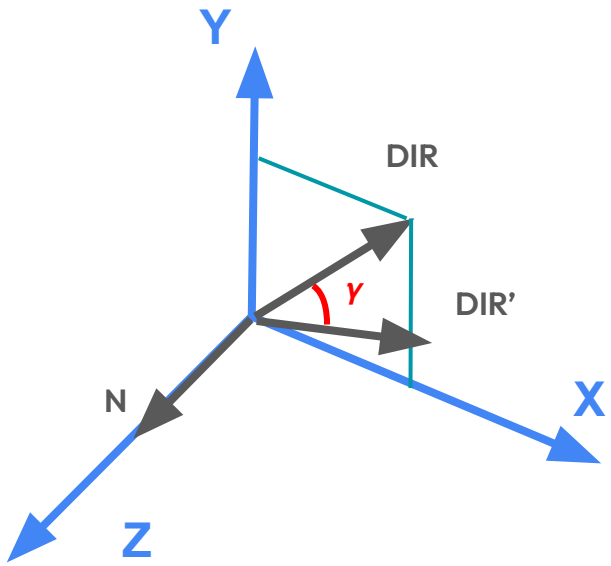
$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$



Matemáticas de Punto Fijo

- Paso 3: Giro 2D vector DIR surfer
  - Aplicamos las mismas rotaciones XY al vector DIR del surfer → lo llevamos al plano YX
  - Ahora lo giramos ***γ*** aplicando una simple rotación 2D:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos(\gamma) & \sin(\gamma) \\ -\sin(\gamma) & \cos(\gamma) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

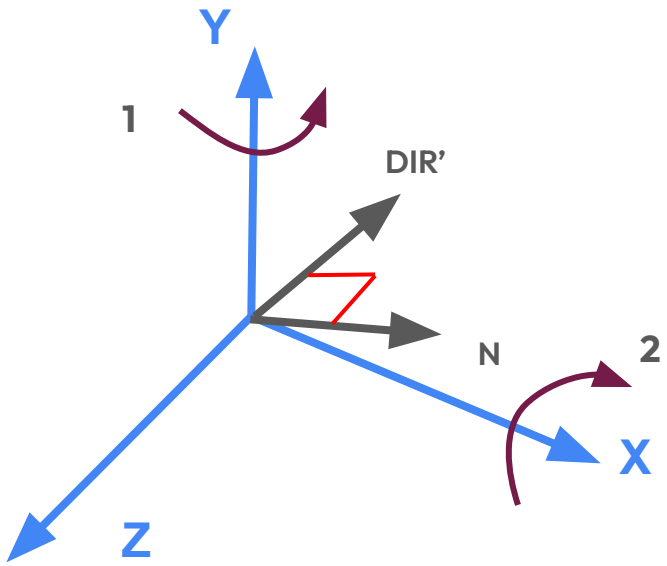


Matemáticas de Punto Fijo

- Paso 4: devolver vector DIR' a su espacio de coordenadas 3D
  - Aplicamos las rotaciones en orden inverso, YX
  - Para invertir las rotaciones se utilizan las matrices traspuestas:

$$R_y^T(\beta) = \begin{pmatrix} \cos(\beta) & 0 & -\sin(\beta) \\ 0 & 1 & 0 \\ \sin(\beta) & 0 & \cos(\beta) \end{pmatrix}$$

$$R_x^T(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & \sin(\alpha) \\ 0 & -\sin(\alpha) & \cos(\alpha) \end{pmatrix}$$



Matemáticas de Punto Fijo

Formulación moderna: **Quaternions**

- Se usa para representar rotaciones: ángulo (α) + eje rotación (**K**)
- Para rotar un vector **v** se aplica el producto de Hamilton:

$$v' = q \cdot v \cdot q^{-1}$$

- El inverso es muy sencillo (**k** unitario):

$$q^{-1} = \begin{pmatrix} \cos\left(\frac{\alpha}{2}\right) \\ -k_x \sin\left(\frac{\alpha}{2}\right) \\ -k_y \sin\left(\frac{\alpha}{2}\right) \\ -k_z \sin\left(\frac{\alpha}{2}\right) \end{pmatrix}$$

$$q = \begin{pmatrix} \cos\left(\frac{\alpha}{2}\right) \\ k_x \sin\left(\frac{\alpha}{2}\right) \\ k_y \sin\left(\frac{\alpha}{2}\right) \\ k_z \sin\left(\frac{\alpha}{2}\right) \end{pmatrix}$$

Matemáticas de Punto Fijo

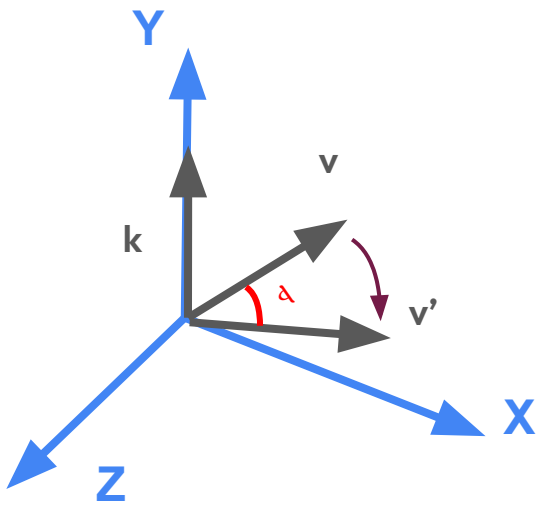
Formulación moderna: **Quaternions**

- El producto de Hamilton para rotar un vector v se puede simplificar para ahorrar operaciones:

$$v' = v + 2w(q_{vec} \times v) + 2(q_{vec} \times (q_{vec} \times v))$$

donde q_{vec} es la parte vectorial (k) y w la parte escalar, cos (∡/2).

- Es muy compacto pero requiere el cálculo previo de ∡ → problemas de precisión



Matemáticas de Punto Fijo

Comparativa de Rendimiento: Quaternions vs. Alinear en Z

Método Alinear en Z

Rotación XY
$(R_X(\alpha) + R_Y(\beta))$ → 4 + 4 = 8 mult.

↓ v

Rotación 2D
$(R_{XY}(\gamma))$ → 4 mult.

↓ v

Rotación YX
$(R_{TY}(\beta) + R_{Tx}(\alpha))$ → 4 + 4 = 8 mult.

Total: 20 Multiplicaciones

Ventajas: Más compacto que Alinear en Z

✗ Requiere cálculo del ángulo con arccos, implica problemas de precisión

Comparativa de rendimiento:

- Quaternions: 18 Multiplicaciones ⬆ Más rápido
- Alinear en Z: 20 Multiplicaciones

Nota: Rendimiento casi equivalente en la práctica.

Método Quaternions (Formulación Moderna)

$$\mathbf{v}' = \mathbf{v} + 2w(q_{\text{vec}} \times \mathbf{v}) + 2(q_{\text{vec}} \times (q_{\text{vec}} \times \mathbf{v}))$$

Parte 1: 6 mult.

Parte 2: 6 mult.

Parte 3: 6 mult.

Total: 18 Multiplicaciones

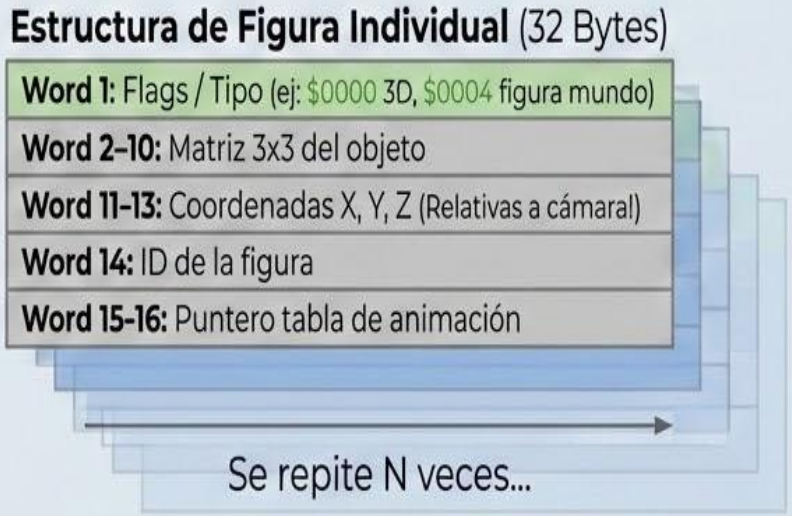
Composición Buffer de Figuras 68000 - DSP

BLOQUE 1: CABECERA GLOBAL (Fijo: 32 Bloque de 32 Bytes)

- **Bytes 0 - 1 (1 Word):** *Libre / Salto.*
- **Bytes 2 - 19 (9 Words): Matriz de la Cámara**. Las componentes X, Y, Z de la cámara
- **Bytes 20 - 29 (5 Words):** *Libre* / Relleno de alineación.
- **Bytes 30 - 31 (1 Word): Nº de Figuras (NUMTMS)**. Empieza a cero. Cada vez que una **función** mete una **figura**, se suma 1

BLOQUE 2: ARRAY DE FIGURAS (N bloques de 32 bytes)

A partir del byte 32, empezaban a concatenarse los los objetos que tenían su propia estructura:




SPEED
101
km/h
TIME
22
LEFT
WAX
PERFECT
GOOD
POOR
BAD
R: 1'30"0 - - -
T: 0'35"6
POS.: 3/4
Gracias!
CPU
3
Mt. Cook
EASY