ROAD
Pseudo-3D Racer in MMBasic

Manual: the Outrun technique step by step, the finished game and how to extend it – for the Pico Computer 3 (RP2350) with PicoMite V7.00.00



Part I: the technique · Part II: ROAD 1.0 – source code, modding, joining in – in 320 × 240 pixels and 16 freely assignable colours (HDMI MODE 2).

Matthias (homa) with Claude, 2026. Based on the article series “How to build a racing game” by Jake Gordon (2012, source code under the MIT licence) and on “Lou’s Pseudo 3d Page”. Texts, diagrams, pixel art and screenshots were made for this document. Status: ROAD 1.0 RC1, November 2026.

Contents

What this is about **4**

1 Target platform: Pico Computer 3 with PicoMite V7 **6**

1.1 Hardware and firmware . . . 6

1.2 Which screen mode? . . . 6

1.3 Resolution: from 1024 × 768 to 320 × 240 . . . 6

1.4 Colours: 16 slots, freely assigned . . . 7

1.5 Buffers: where is drawn? . . . 8

2 Part 1: The straight road **8**

2.1 Projection: one point, one division . . . 9

2.2 The road as a chain of segments . . . 11

2.3 Camera and player . . . 12

2.4 Game loop with a fixed time step . . . 12

2.5 Drawing: front to back . . . 13

3 Part 2: Curves **14**

3.1 The idea: curves are only faked . . . 14

3.2 Smooth transitions with easing . . . 14

3.3 Drawing with two accumulators . . . 15

3.4 Centrifugal force . . . 16

3.5 Parallax background . . . 16

4 Part 3: Hills **17**

4.1 One height per segment . . . 17

4.2 The camera follows . . . 17

4.3 Hiding behind crests: maxy . . . 18

4.4 The background moves too . . . 18

5 Part 4: The finished game **18**

5.1 Track building with building blocks . . . 18

5.2 Sprites: sheet, placement, scaling . . . 20

5.3 Clipping at the hill edge . . . 20

5.4 Order: road forwards, sprites backwards . . . 21

5.5 Opponents: cars with a simple AI . . . 22

5.6 Collisions . . . 23

5.7 Player car, HUD and lap times . . . 24

6 Speed on the RP2350 **24**

6.1 What each frame costs . . . 24

6.2 Rules for fast MMBasic . . . 25

6.3 Knobs in the game . . . 25

7 The alternative: Lou’s raster road **26**

8 In practice: optimising on the Pico Computer 3 **26**

8.1 Measurement log 27

8.2 Lessons for MMBasic V7 on the RP2350 27

9 Sound: what the experiments showed **29**

9.1 The audio systems exclude each other 29

9.2 PLAY SOUND: engine, gears, Doppler 30

9.3 MOD music with effect samples 32

9.4 Outlook (state of the study) 33

10 From prototype to game **35**

10.1 What changed in the technique – and why 36

11 A tour through the source code **37**

11.1 The main loop 37

11.2 The state machine 38

11.3 What is inside a segment 39

11.4 How a frame is made 39

11.5 Background layers and the compass 39

11.6 Times and difficulty 40

12 Modding: your own tracks, graphics and music **40**

12.1 Creating a new track 41

12.2 The track file 41

12.3 The sprite sheet 42

12.4 Background pictures 42

12.5 Music and effects 44

12.6 Measuring the times 44

13 Join in! **45**

13.1 Ideas for the community 45

14 Appendix **46**

14.1 Constants: original and study RACER.BAS 46

14.2 Commands used (PicoMite V7.00.00) 46

14.3 Sources, licences and thanks 47

What this is about

A “pseudo-3D” racing game like Outrun, Pole Position or Lotus Turbo Challenge does not compute a real 3D world. The road consists of many short, flat pieces (segments) that are only staggered in depth. Each segment is projected onto the screen with a single division per corner and drawn as a coloured trapezoid. Curves come from clever sideways shifting, hills from a height per segment, and everything else – trees, signs, opponents – are flat pictures (sprites) scaled with the distance. Exactly this thrift makes the technique ideal for a microcontroller programmed in BASIC.

Jake Gordon demonstrated the technique in 2012 in four articles that build on each other, with a JavaScript game; Lou Gorenfeld collected the tricks of the arcade machines it comes from on his page. This manual has two parts. **Part I** follows Jake Gordon’s structure, explains the technique with the “old” flat sprites of the first feasibility study `RACER.BAS` and carries it over to the Pico step by step – where the frame rate needs it, with other means than the original:

- **Part 1 – Straight road:** projection, segments, camera, game loop.
- **Part 2 – Curves:** curve value per segment, transitions (easing), centrifugal force, parallax background.
- **Part 3 – Hills:** height per segment, hiding behind crests.
- **Part 4 – Final:** track building, sprites with clipping, computer cars, collisions, HUD and lap times.

Before that comes a chapter on the platform (screen mode, colours, buffers), afterwards chapters on speed on the RP2350, on Lou’s raster road, on optimising on the real device and on sound. **Part II** shows what came of it: the game **ROAD 1.0** with five tracks, world tour and records – a tour through the source code, a guide to modding (tracks, graphics, music) and an invitation to the community.

Where do the pictures come from?
The game scenes in Part I show the road of the original articles recreated in **320 × 240 pixels** and with exactly the **16-colour palette** of chapter 1 – the way it looks on the HDMI output of the Pico Computer 3. The diagrams are redrawn and match those of the original articles. The pictures in Part II are captures from the real device.

Status of the MMBasic code
The code excerpts of chapters 1–7 come from the first version `RACER.BAS`. It ran on the Pico Computer 3 straight away. In 13 steps it was then measured and optimised on the real device (chapter 8) and given sound (chapter 9). State of the feasibility study: **RACER_V13.BAS** with **RACER.MOD** – about 17–22 frames per second, music and real effect samples. The game ROAD grew out of it (Part II).

About the graphics: the original game uses sprites from the Sega Genesis version of Outrun as placeholders, which may not be used freely. Here there is a small sprite sheet of its own instead (`SPRITES.BMP`, 320 × 240) that fits straight into the PicoMite’s image buffer.

Part I

The technique

Pseudo-3D after Jake Gordon and Lou Gorenfeld – carried over to MMBasic and the Pico Computer 3, shown with the feasibility study RACER.BAS.

1 Target platform: Pico Computer 3 with PicoMite V7

1.1 Hardware and firmware

The Pico Computer 3 has an RP2350B with PSRAM and HDMI output. The matching firmware is called **PicoMitePC3**: it is the HDMI-USB version (USB keyboard, mouse and game controller, all HDMI resolutions and modes) plus WiFi and Bluetooth. What counts for us:

- **Clock:** at 640 × 480 the clock can be set to 378 MHz with `OPTION RESOLUTION 640x480, 378000` (default 252 MHz). Everything runs proportionally faster – the cheapest gain there is.
- **Compiler:** V7 brings `OPTION COMPILE ON` on the RP2350. The program stays unchanged but is translated into native operations on RUN; simple arithmetic statements become 8 to 14 times faster according to the Performance Guide.
- **Memory:** 256 KB program memory; variables and arrays mostly go into PSRAM if present. Plenty of room for the track data.
- **Input:** `KEYDOWN(n)` reports up to six keys of a USB keyboard held at the same time – important for throttle + steering.

1.2 Which screen mode?

At HDMI 640 × 480 there are five modes. The game uses **MODE 2**:

Mode	Resolution	Colours	Rating for the racer
MODE 1	640 × 480	2 (tile colours)	unsuitable
MODE 2	**320 × 240**	**16, freely via MAP to RGB555**	**chosen:** small framebuffer (38.4 KB), BLIT RESIZE, layers L and T, MAP palette
MODE 3	640 × 480	16 via MAP	four times as many pixels to fill – too slow for BASIC
MODE 4	320 × 240	32768 (RGB555)	nice gradients, but BLIT RESIZE only works with 4-bit buffers
MODE 5	320 × 240	256 via MAP	option for later (fog, more sprite colours); BLIT RESIZE then not usable

The deciding command is `BLIT RESIZE`: it scales a rectangle from a buffer into a target rectangle (nearest neighbour), optionally with a transparent colour – exactly what a pseudo-3D racer needs for trees, signs and cars. According to the BLIT manual it works on RGB121-packed 4-bit buffers, i.e. in MODE 2 and 3. The Game Development Guide also calls MODE 2 the “sweet spot” for games.

1.3 Resolution: from 1024 × 768 to 320 × 240

The original computes with a logical canvas of 1024 × 768. Both formats have the aspect ratio 4:3, and all projection formulas only use *width/2* and *height/2*. So it is enough to plug in width and height – the road looks the same, only coarser. Where the original uses a resolution factor *resolution = height/480* (wobble of the player car, vertical offset of the background), **RES = 0.5** applies here.

One real change concerns the **draw distance**: the original draws 300 segments. With 240 screen rows, far segments shrink to less than one row and are dropped anyway. For the RP2350 **100 to 150 segments** are recommended (chapter 6 shows the difference).

1.4 Colours: 16 slots, freely assigned

In MODE 2 every pixel has 4 bits, i.e. one of 16 *slots*. Which colour a slot shows is set by the colour table: `MAP(n) = rgb` sets it, `MAP SET` applies all changes in the next blanking interval. On the HDMI output every colour is rounded to the nearest RGB555 value (5 bits per channel). Jake Gordon's original colours were spread over 16 slots like this:



Fig. 1 – Palette of the game. Slot 0 is also the transparent colour for sprites and the HUD layer.

- **Dark road:** the original uses #6B6B6B and #696969. In RGB555 both fall onto the same value (13 of 31 each) – the light/dark change would be invisible. That is why slot 5 was darkened to #5A5A5A.
- **Fog:** the original blends far segments semi-transparently into forest green. Without an alpha channel this is not possible in 16 colours. The fog is dropped; the dark tree line on the horizon does its job visually. (In MODE 5 fog steps could be separate palette entries.)
- **Start/finish:** the start line uses the lane slot 8, the finish line slot 0 (black).
- **Sprites:** slots 9 to 15 serve trees, hills, clouds and the cars.

Important when drawing: graphics commands expect a 24-bit colour. Which colour selects slot *n* exactly is returned by the function `MAP(n)`. The program reads these values once into an array `col%()` and only draws with it:

RACER.BAS – setting the palette (from SUB Init)

```
SUB Init
  LOCAL i%
  MODE 2
  FOR i% = 0 TO 15
    READ pal%(i%)
    MAP(i%) = pal%(i%)
  NEXT
  MAP SET
  FOR i% = 0 TO 15
    col%(i%) = MAP(i%)              ' RGB value that selects exactly slot i%
  NEXT
```

Preparing the sprite sheet as BMP
`LOAD IMAGE` maps every BMP pixel to the nearest *standard* colour, not to the redefined palette. The file `SPRITES.BMP` is therefore saved in the 16 standard colours (slot 1 = blue, slot 2 = myrtle, etc.). In the game the pixels then appear in the wanted colours via MAP. If you paint your own graphics, work indexed and export to the standard colours at the end.

1.5 Buffers: where is drawn?

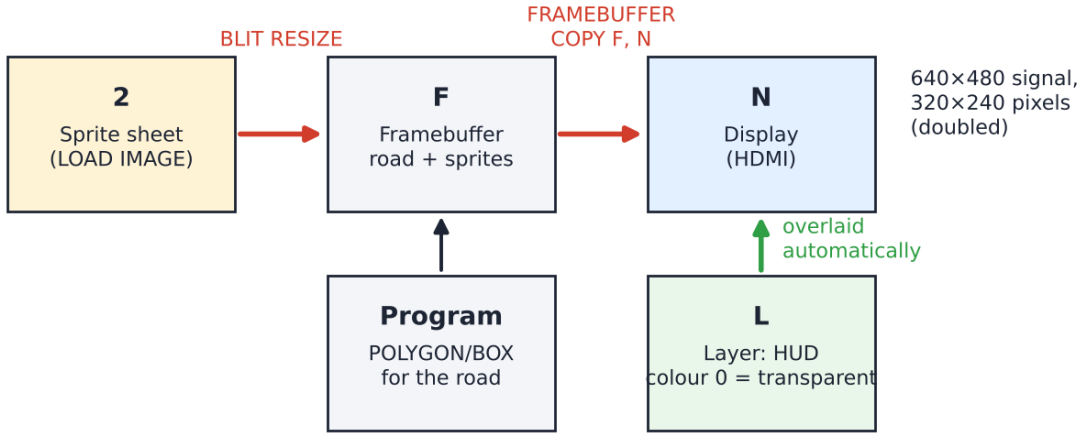

Fig. 2 – Buffer architecture. Drawing always goes to F; the display N gets the finished picture in one go.

- **F (framebuffer):** all drawing commands go here. At the end of each frame `FRAMEBUFFER COPY F, N` copies the content to the screen without flicker. The Game Guide advises against the background variant with a third parameter in the game loop.
- **L (layer):** with HDMI in MODE 2 it is laid over the picture automatically; slot 0 is transparent. Ideal for the HUD: it only has to be rewritten when speed or time change.
- **2 (second framebuffer):** holds the sprite sheet and the three background strips. It is invisible and only serves as the source for `BLIT RESIZE` and `BLIT FRAMEBUFFER`. Because L uses the pre-allocated video memory, 2 is taken from variable memory (on the PC3 possibly PSRAM – keep that in mind when measuring).

RACER.BAS – creating the buffers (from SUB Init)

```
FRAMEBUFFER CREATE                ' F: everything is drawn here
FRAMEBUFFER LAYER 0               ' L: HUD, colour 0 is transparent
FRAMEBUFFER CREATE 2              ' 2: sprite sheet + background strips
FRAMEBUFFER WRITE L : CLS col%(C_BLACK)
FRAMEBUFFER WRITE 2 : LOAD IMAGE "sprites.bmp"
FRAMEBUFFER WRITE F
```

2 Part 1: The straight road

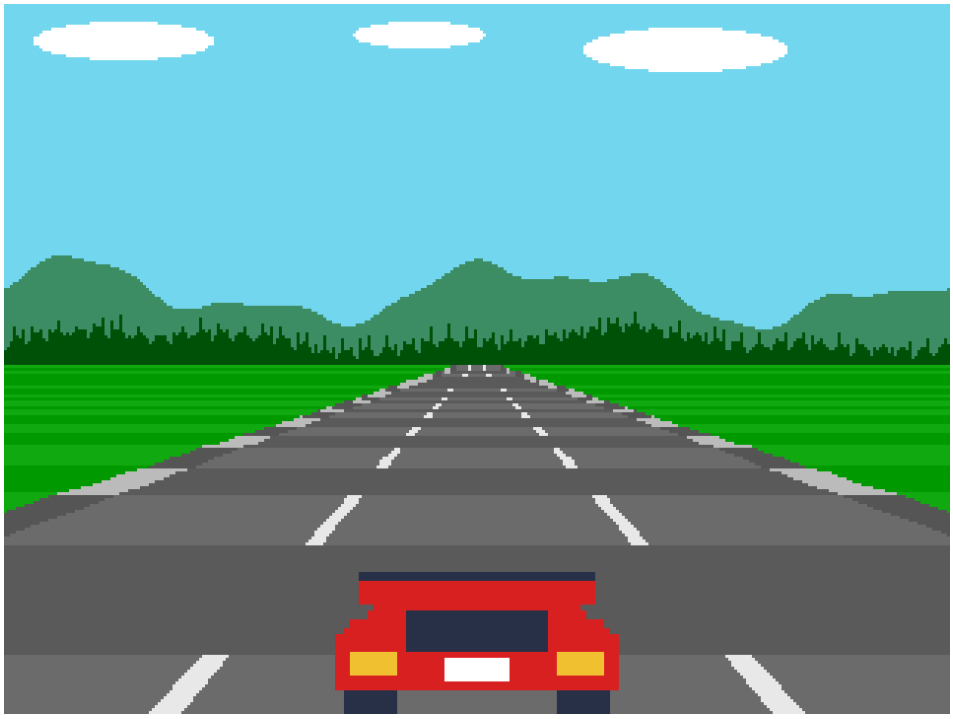

Fig. 3 – Result of part 1 in 320 × 240 and 16 colours.

2.1 Projection: one point, one division

Everything rests on similar triangles. The camera hovers at height *h* above the road; at distance *d* in front of it stands the imaginary image plane. A road point at distance *z* appears on the image plane at height $y = h \cdot d / z$ below the horizon. The same applies sideways: a point with the lateral distance *w* lands at $x = w \cdot d / z$. The factor **d / z** is the scale with which sprites are scaled later too.

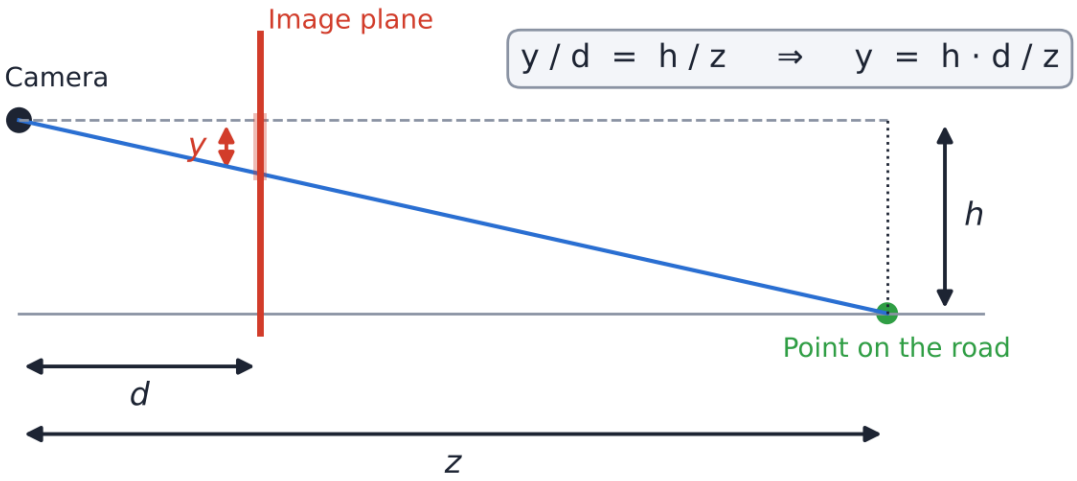

Fig. 4 – Similar triangles: the farther away a point is, the closer it moves to the horizon.

For the formula to work for any point, each point is converted in three steps:

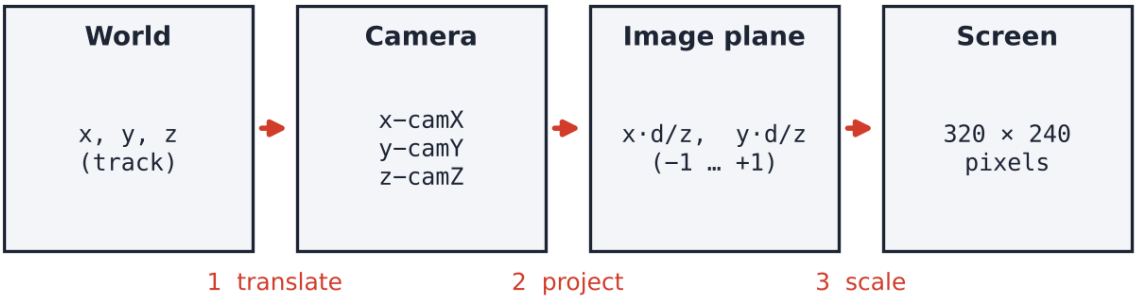

Fig. 5 – From world to screen coordinates.

- **Translate:** subtract the camera position. The camera then sits at the origin.
- **Project:** multiply x and y by *d/z*. Result: values between −1 and +1 on the image plane.
- **Scale:** convert to pixels. Because the y axis points down on the screen, y is flipped.

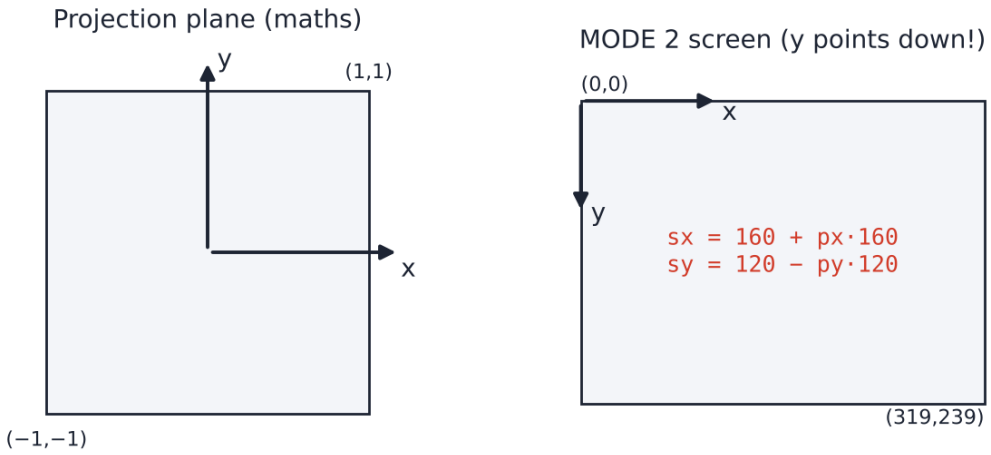

Fig. 6 – Mathematical and screen coordinate systems in MODE 2.

The distance *d* follows from the vertical field of view: with an image plane from −1 to +1, **d = 1 / tan(fov/2)**. With 100° as in the original this gives d ≈ 0.839. Bigger angles look wider and faster, smaller ones like a telephoto lens.

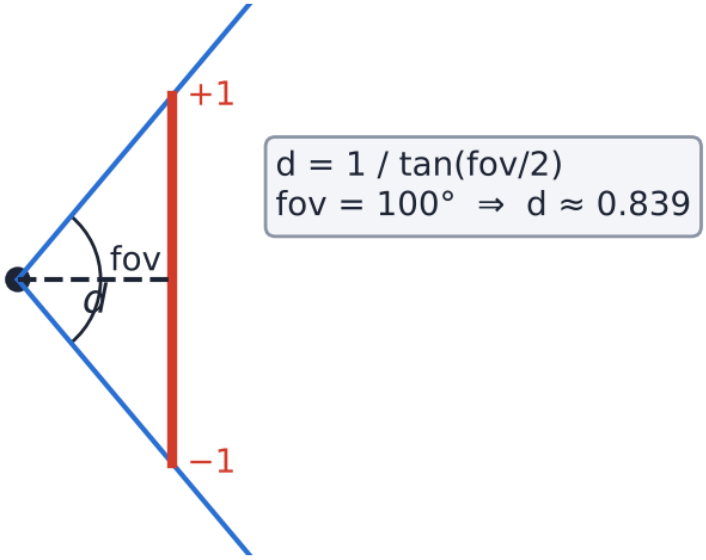

Fig. 7 – Field of view and distance to the image plane.

In short, for a point with camera coordinates (cx, cy, cz):

Projection in MMBasic notation

```
scale = CAMD / cz
sx    = HW + scale * cx * HW          ' HW = 160  (half width)
sy    = HH - scale * cy * HH          ' HH = 120  (half height)
sw    = scale * ROADW * HW            ' half road width in pixels
```

2.2 The road as a chain of segments

The track is a long list of segments of equal length (*segmentLength* = 200 units). Each segment has a near centre point **p1** and a far one **p2**; p2 of one segment is p1 of the next. Half the road width *roadWidth* = 2000 makes the maths symmetric (the road reaches from −2000 to +2000). Every three segments (*rumbleLength*) the colour group switches between light and dark – this creates the typical rumble strips and, much more important, the feeling of speed. If every segment were coloured on its own, the picture would flicker.

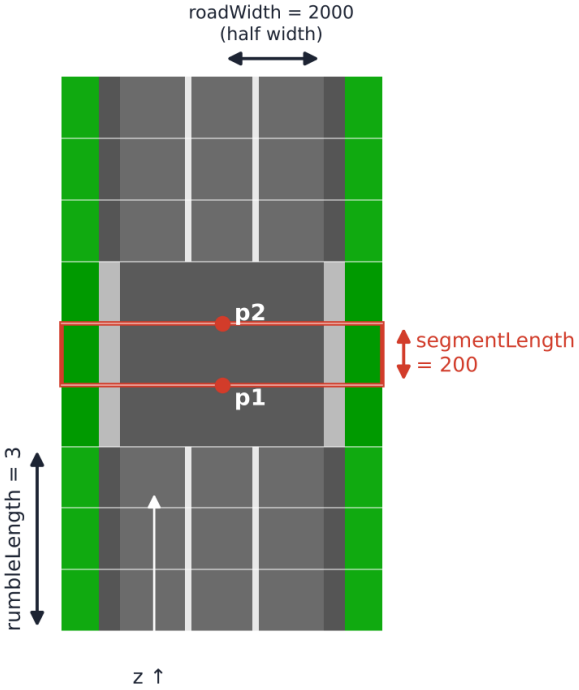

Fig. 8 – Segments seen from above (z points up in the driving direction), with grass, rumble strips, road and lane lines.

MMBasic knows structures (TYPE), but the compiler does not translate structure access. For the hot loop **parallel arrays** are better: one array per property, the index is the segment number. Only what cannot be computed is stored per segment:

Track data (RACER.BAS)

```
DIM segY(MAXSEG), segCv(MAXSEG), segCol%(MAXSEG), sprHd%(MAXSEG)
DIM nSeg%, trackLen
' segY    height at the far end (p2); p1 is segY() of the previous one
' segCv   curve value (part 2)
' segCol% 0 = light, 1 = dark, 2 = start, 3 = finish
' sprHd%  first sprite of this segment (part 4), -1 = none
```

The *z* coordinates are not stored at all. Because segment *n* lies places in front of the camera segment, its distance to the camera is simply **(n − bp) · 200**, where *bp* is the part of the camera segment already covered. This saves memory and also solves the problem that the original has to treat segments beyond the end of the track (“looped”) separately.

2.3 Camera and player

- **position** is the z position of the camera on the track; it runs modulo the track length.
- **playerZ** is the fixed distance of the car in front of the camera: *cameraHeight* · *cameraDepth* ≈ 839. Exactly there a sprite has the scale at which it stands at the bottom edge of the screen.
- **playerX** is the lateral position, normalised to −1 (left edge) to +1 (right edge). With |playerX| > 1 you drive on the grass. For the projection the camera is shifted by *playerX* · *roadWidth* – not the car: the car stays in the middle of the picture, the road moves.

2.4 Game loop with a fixed time step

The original separates logic and drawing: the logic always computes in fixed steps of 1/60 s, drawing happens as often as possible. If the Pico manages, say, 25 frames per second, two to three logic steps run per frame. This keeps the handling independent of the frame rate, and the top speed of *segmentLength / step* guarantees that at most one segment is skipped per step (important for collisions).

RACER.BAS – main loop

```
Init
tLast = TIMER : acc = 0
DO
  ReadKeys
  tNow = TIMER
  dt = MIN(1000, tNow - tLast) / 1000
  tLast = tNow
  acc = acc + dt
  DO WHILE acc > STP
    acc = acc - STP
    Update STP
  LOOP
  Render
  Hud
  FRAMEBUFFER COPY F, N
LOOP UNTIL kQ%
```

RACER.BAS – keyboard (codes 128–131 = arrow keys according to the appendix of the manual)

```
SUB ReadKeys
  LOCAL i%, k%, n%
  kL% = 0 : kR% = 0 : kF% = 0 : kS% = 0
  n% = KEYDOWN(0)
  FOR i% = 1 TO n%
    k% = KEYDOWN(i%)
    IF k% = 130 OR k% = 97 THEN kL% = 1       ' left  / a
    IF k% = 131 OR k% = 100 THEN kR% = 1      ' right / d
    IF k% = 128 OR k% = 119 THEN kF% = 1      ' up    / w
    IF k% = 129 OR k% = 115 THEN kS% = 1      ' down  / s
    IF k% = 27 THEN kQ% = 1                   ' ESC
  NEXT
END SUB
```

The physics is deliberately simple: throttle accelerates by *maxSpeed/5* per second, braking decelerates by *maxSpeed*, without a pedal the car rolls out at *maxSpeed/5*. On the grass it brakes additionally by *maxSpeed/2*, but never below a quarter of the top speed. Steering moves playerX by at most 2 per second – at full throttle you get from one edge to the other in one second.

2.5 Drawing: front to back

The road is drawn away from the car. For every segment p1 and p2 are projected. It is skipped if p1 lies behind the camera, if it would be seen “from behind” (p2 lies lower than p1) or if it disappears completely behind what has been drawn (more on that in part 3). Every visible segment produces a grass band across the full width and four to six trapezoids: two rumble strips, the road and, on light segments, two lane lines. The rumble strips are 1/6 of half the road width wide, the lines 1/32.

The most important MMBasic trick: collect polygons
A BASIC command costs a few microseconds just for being interpreted. Instead of 150 to 300 single drawing commands per frame, Quad collects all trapezoids in arrays, and **one single** `POLYGON n%(), x%(), y%(), c%(), c%()` draws them at the end at firmware speed. A 0 in the count array marks the end of the list. The order is kept, the result is identical.

RACER.BAS – append a trapezoid to the list

```
SUB Quad(xa, ya, xb, yb, xc, yc, xd, yd, c%)
  polyN%(np%) = 4 : polyC%(np%) = c%
  polyX%(ip%) = xa : polyY%(ip%) = ya
  polyX%(ip% + 1) = xb : polyY%(ip% + 1) = yb
  polyX%(ip% + 2) = xc : polyY%(ip% + 2) = yc
  polyX%(ip% + 3) = xd : polyY%(ip% + 3) = yd
  INC np% : INC ip%, 4
END SUB
```

RACER.BAS – road loop (from SUB Render; in part 1 segCv() = 0 and segY() = 0)

```
  ' --- road: front to back, collect polygons
  maxy% = SH : x = 0 : dx = -segCv(base%) * bp
  camY = playerY + CAMH : px = playerX * ROADW
  np% = 0 : ip% = 0
  FOR n% = 0 TO DD - 1
    s% = base% + n% : IF s% >= nSeg% THEN s% = s% - nSeg%
    IF s% = 0 THEN y1w = 0 ELSE y1w = segY(s% - 1)
    z1 = (n% - bp) * SEGLEN : z2 = z1 + SEGLEN     ' camera z without loop problems
    sc1 = CAMD / MAX(z1, 1) : sc2 = CAMD / MAX(z2, 1)
    x1% = CINT(HW + sc1 * (x - px) * HW)
    y1% = CINT(HH - sc1 * (y1w - camY) * HH)
    w1% = CINT(sc1 * ROADW * HW)
    x2% = CINT(HW + sc2 * (x + dx - px) * HW)
    y2% = CINT(HH - sc2 * (segY(s%) - camY) * HH)
    w2% = CINT(sc2 * ROADW * HW)
    rX1%(n%) = x1% : rY1%(n%) = y1% : rS1(n%) = sc1
    rX2%(n%) = x2% : rY2%(n%) = y2% : rS2(n%) = sc2
    clip%(n%) = maxy%
    x = x + dx : dx = dx + segCv(s%)
    IF z1 > CAMD AND y2% < y1% AND y2% < maxy% THEN
      g% = segCol%(s%)
      BOX 0, y2%, SW, MIN(y1%, maxy%) - y2%, 0, grC%(g%), grC%(g%)
      r1 = w1% / 6 : r2 = w2% / 6
      Quad x1% - w1% - r1, y1%, x1% - w1%, y1%, x2% - w2%, y2%, x2% - w2% - r2, y2%, rbC%(g%)
      Quad x1% + w1% + r1, y1%, x1% + w1%, y1%, x2% + w2%, y2%, x2% + w2% + r2, y2%, rbC%(g%)
      Quad x1% - w1%, y1%, x1% + w1%, y1%, x2% + w2%, y2%, x2% - w2%, y2%, rdC%(g%)
      IF lnC%(g%) <> -1 THEN
        l1 = w1% / 32 : l2 = w2% / 32
        lw1 = w1% * 2 / 3 : lw2 = w2% * 2 / 3
        lx1 = x1% - w1% + lw1 : lx2 = x2% - w2% + lw2
        FOR ln% = 1 TO 2
          Quad lx1 - l1 / 2, y1%, lx1 + l1 / 2, y1%, lx2 + l2 / 2, y2%, lx2 - l2 / 2, y2%, lnC%(g%)
          lx1 = lx1 + lw1 : lx2 = lx2 + lw2
        NEXT
      ENDIF
      maxy% = y1%
    ENDIF
  NEXT
  polyN%(np%) = 0                                  ' end marker
  IF np% > 0 THEN POLYGON polyN%(), polyX%(), polyY%(), polyC%(), polyC%()
```

3 Part 2: Curves

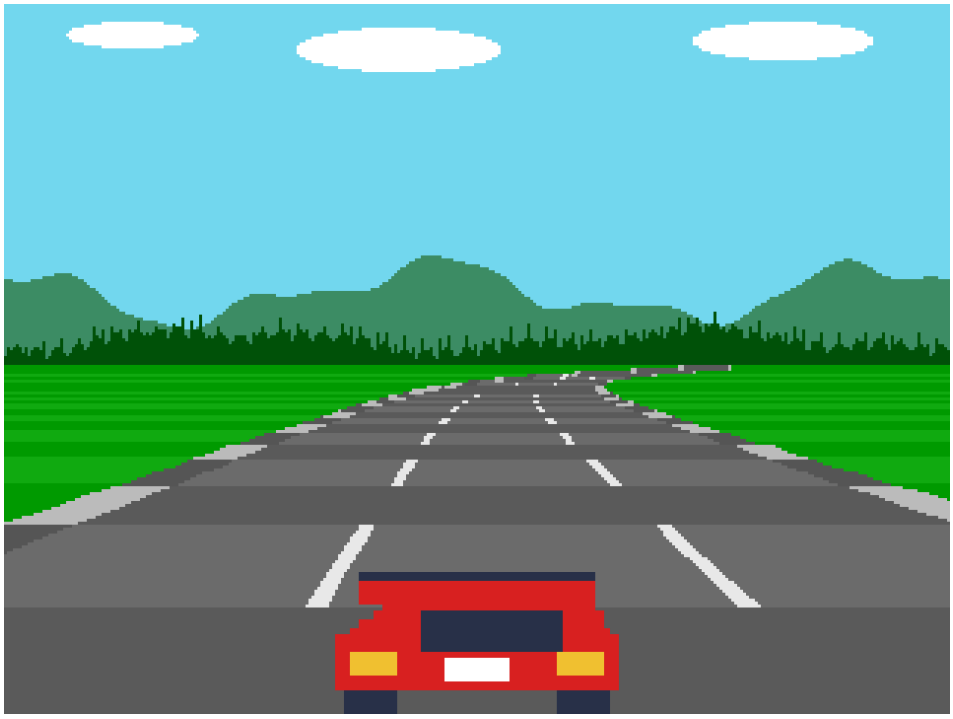

Fig. 9 – Part 2: a right-hand curve (320 × 240).

3.1 The idea: curves are only faked

A real curve would need x coordinates for every segment and a rotated camera. The game makes it easier: each segment only gets a **curve value** – negative for left, positive for right, the bigger, the tighter. When drawing, the centre line of the road is shifted sideways more and more from segment to segment. The world itself stays straight; only the picture bends. This idea comes from Lou’s page and works amazingly well.

Constant	Values	Meaning
Length	short 25 · medium 50 · long 100	segments per phase
Curve	easy 2 · medium 4 · hard 6	curve value per segment (chosen by feel)
Hill	low 20 · medium 40 · high 60	height change in segment lengths (part 3)

3.2 Smooth transitions with easing

An abrupt change from 0 to 4 would look like a kink. Every road section therefore has three phases: **enter** (curve value rises with easeIn), **hold** (constant) and **leave** (falls to 0 with easeInOut). The helper functions interpolate between a and b; easeIn quadratically, easeInOut with half a cosine wave.

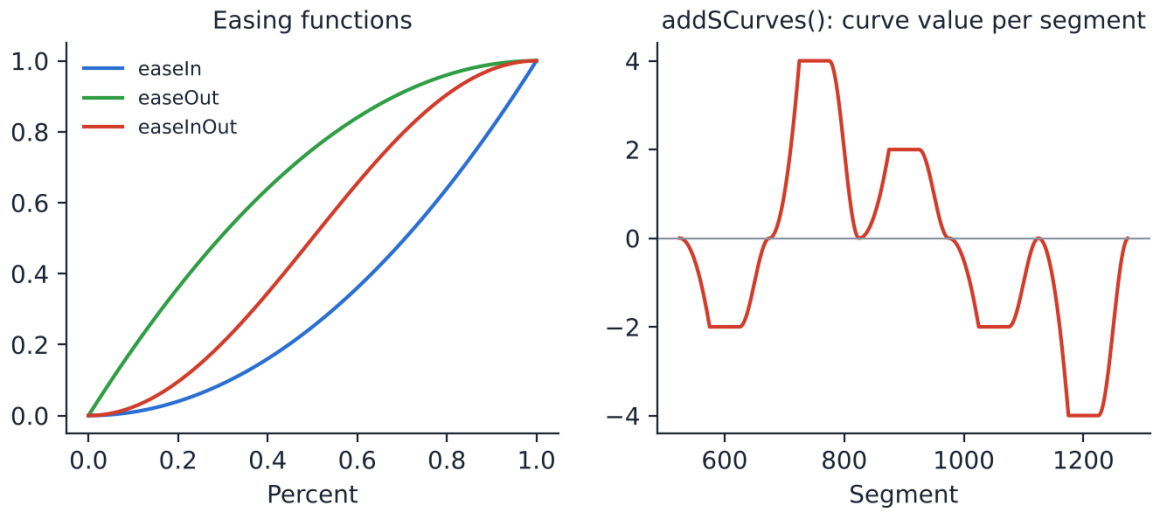

Fig. 10 – Left the easing functions, right the curve profile of addSCurves(): five sections with enter, hold and leave phases.

RACER.BAS – easing and the universal road building block

```
FUNCTION EaseIn(a, b, p)
  EaseIn = a + (b - a) * p * p
END FUNCTION

FUNCTION EaseInOut(a, b, p)
  EaseInOut = a + (b - a) * ((-COS(p * PI) / 2) + 0.5)
END FUNCTION

SUB AddRoad(en%, ho%, le%, cv, h)
  LOCAL sy, ey, n%, tot%
  sy = LastY()
  ey = sy + h * SEGLEN
  tot% = en% + ho% + le%
  FOR n% = 0 TO en% - 1
    AddSegment EaseIn(0, cv, n% / en%), EaseInOut(sy, ey, n% / tot%)
  NEXT
  FOR n% = 0 TO ho% - 1
    AddSegment cv, EaseInOut(sy, ey, (en% + n%) / tot%)
  NEXT
  FOR n% = 0 TO le% - 1
    AddSegment EaseInOut(cv, 0, n% / le%), EaseInOut(sy, ey, (en% + ho% + n%) / tot%)
  NEXT
END SUB
```

3.3 Drawing with two accumulators

While drawing, two numbers run along: **x** is the lateral offset of the centre line, **dx** its change per segment. After every segment $x = x + dx$ and $dx = dx + curve$. Because dx itself grows, x grows quadratically – the road bends more and more towards the horizon. p1 is projected with offset x, p2 with x + dx.

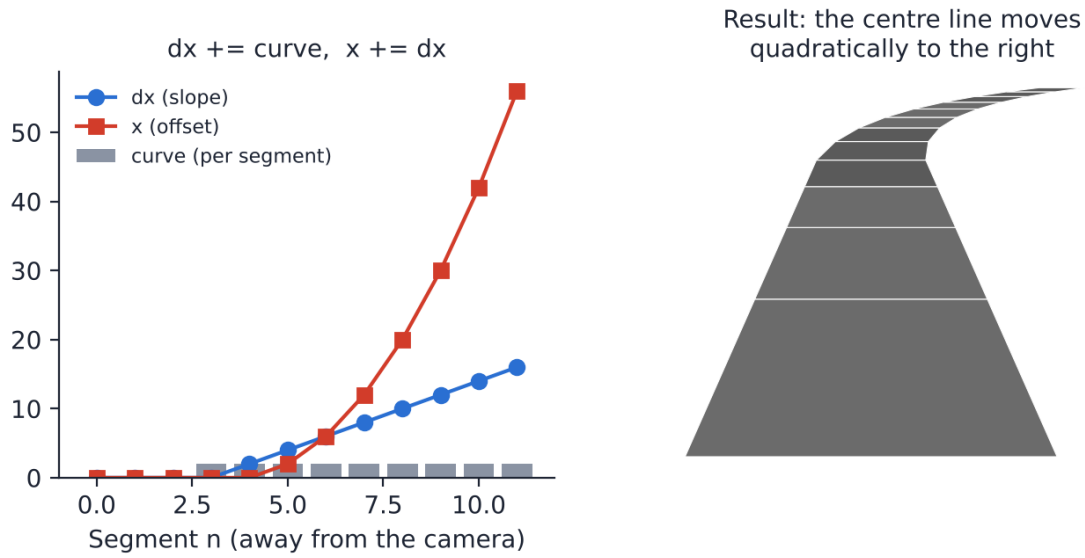

Fig. 11 – Accumulation: from segment 3 a curve starts (curve = 2). dx rises linearly, x quadratically.

One detail prevents jerking: if the camera is already a fraction *bp* into the current segment, dx starts with **−curve · bp**. Without this correction the road would jump a bit at every segment change. Lou describes the same trick as “scroll fix”.

The curve changes in the road loop

```
maxy% = SH : x = 0 : dx = -segCv(base%) * bp
...
x1% = CINT(HW + sc1 * (x - px) * HW)             ' p1 with offset x
x2% = CINT(HW + sc2 * (x + dx - px) * HW)        ' p2 with offset x + dx
...
x = x + dx : dx = dx + segCv(s%)
```

3.4 Centrifugal force

So that curves also feel like curves, the physics pushes the car outwards – proportional to speed and curve value, with the factor 0.3. If you do not steer against it at full throttle, you end up on the grass.

```
playerX = playerX - dx * sp * segCv(ps%) * CENTRI      ' centrifugal force (from SUB Update)
```

3.5 Parallax background

The background consists of three layers – clouds, hills, tree line – that move sideways at different speeds in curves (factors 0.001 / 0.002 / 0.003 per segment and curve unit). The original paints big pictures for this. For the Pico the layers lie as **320 pixel wide, seamlessly tileable strips** in the lower part of buffer 2. Per frame at most six BLIT FRAMEBUFFER calls with transparent colour 0 are then enough – each strip is cut into two pieces at the offset.

RACER.BAS – copy a background strip with an offset

```
SUB Strip(sy%, h%, off, spd, dy%)
  LOCAL o%, y%, hh%
  o% = INT(off * SW) MOD SW
  y% = dy% + INT(spd * playerY * RES)
  hh% = MIN(h%, SH - y%)
  IF y% < 0 OR hh% < 1 THEN EXIT SUB
  BLIT FRAMEBUFFER 2, F, o%, sy%, 0, y%, SW - o%, hh%, 0
  IF o% > 0 THEN BLIT FRAMEBUFFER 2, F, 0, sy%, SW - o%, y%, o%, hh%, 0
END SUB
```

4 Part 3: Hills

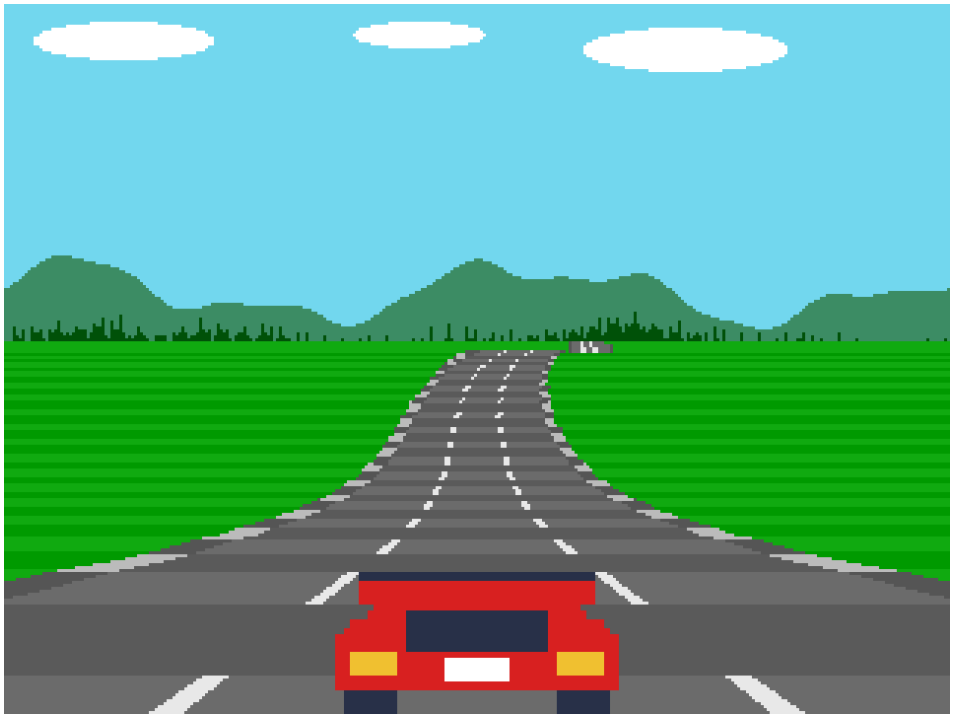

Fig. 12 – Part 3: crest with a curve behind it.

4.1 One height per segment

For hills surprisingly little has to be done, because the projection has always taken y into account – so far it was just 0 everywhere. Each segment stores the height of its far end; the near end takes the height of the previous one, so there are no steps. While building the track, the height is led from the start to the end height over the whole section (enter + hold + leave) with easeInOut. The height is given in segment lengths: “40” means 40 · 200 units higher.

RACER.BAS – appending a segment

```
FUNCTION LastY()
  IF nSeg% = 0 THEN LastY = 0 ELSE LastY = segY(nSeg% - 1)
END FUNCTION

SUB AddSegment(cv, y)
  segCv(nSeg%) = cv
  segY(nSeg%) = y                     ' height at the far end (p2); p1 = previous segment
  segCol%(nSeg%) = (nSeg% \ RUMBLE) MOD 2
  sprHd%(nSeg%) = -1
  INC nSeg%
END SUB
```

4.2 The camera follows

The camera has to follow the road, otherwise it would dive into every hill. For this the height of the road under the car is determined – interpolated between p1 and p2 of the player segment – and the camera is set *cameraHeight* above it:

from SUB Render

```
IF ps% = 0 THEN y1w = 0 ELSE y1w = segY(ps% - 1)
playerY = Lerp(y1w, segY(ps%), pp)
camY = playerY + CAMH
```

4.3 Hiding behind crests: maxy

Road pieces behind a crest must not be drawn. Because drawing goes from front to back, one number is enough: **maxy**, the topmost screen row drawn so far. A segment whose far end does not reach above maxy is hidden and skipped. In addition, segments that would be seen from behind (p2 lower than p1) are dropped, which happens when driving downhill behind a crest.

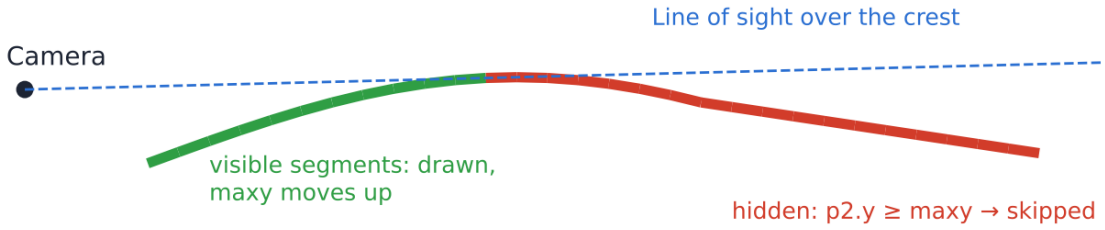

Fig. 13 – Green: visible, maxy moves up. Red: below the line of sight over the crest, skipped.

This saves a lot of work too: in the counts, on average only about 40 of 120 segments looked at are actually drawn. In RACER.BAS the grass band is also limited to maxy, so nothing is painted twice.

4.4 The background moves too

When driving uphill the horizon sinks. Every background layer is therefore also moved down by *speed factor · playerY · RES* (see `Strip`). This ignores the slope of the current segment but looks convincing.

5 Part 4: The finished game



Fig. 14 – Curve with columns, trees and a truck.

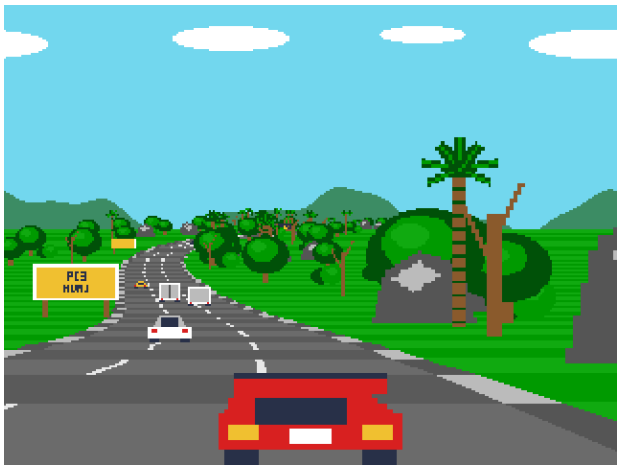

Fig. 15 – Palm section with billboards and opponents.

5.1 Track building with building blocks

All sections come from the one building block `AddRoad(enter, hold, leave, curve, height)`. Small helpers build on it:

Helper (original)	does	MMBasic call
addStraight(n)	straight, flat	AddRoad n, n, n, 0, 0
addCurve(n, k, h)	curve, optionally with height change	AddRoad n, n, n, k, h

Helper (original)	does	MMBasic call
addHill(n, h)	hill without curve	AddRoad n, n, n, 0, h
addLowRollingHills(n, h)	six gentle waves, two easy curves	AddLowRollingHills n, h
addSCurves()	five alternating curves, with height	AddSCurves
addBumps()	eight short bumps	AddBumps
addDownhillToEnd(n)	leads back to height 0 at the end (closed lap)	AddRoad n, n, n, −2, −LastY()/SEGLEN

RACER.BAS – the complete original track

```
SUB BuildTrack
  LOCAL n%, s%
  nSeg% = 0
  AddRoad 25, 25, 25, 0, 0                  ' addStraight(SHORT)
  AddLowRollingHills 25, 20
  AddSCurves
  AddRoad 50, 50, 50, 4, 20                 ' addCurve(MEDIUM, MEDIUM, LOW)
  AddBumps
  AddLowRollingHills 25, 20
  AddRoad 200, 200, 200, 4, 40              ' addCurve(LONG*2, MEDIUM, MEDIUM)
  AddRoad 50, 50, 50, 0, 0                  ' addStraight()
  AddRoad 50, 50, 50, 0, 60                 ' addHill(MEDIUM, HIGH)
  AddSCurves
  AddRoad 100, 100, 100, -4, 0              ' addCurve(LONG, -MEDIUM, NONE)
  AddRoad 100, 100, 100, 0, 60              ' addHill(LONG, HIGH)
  AddRoad 100, 100, 100, 4, -20             ' addCurve(LONG, MEDIUM, -LOW)
  AddBumps
  AddRoad 100, 100, 100, 0, -40             ' addHill(LONG, -MEDIUM)
  AddRoad 50, 50, 50, 0, 0
  AddSCurves
  AddRoad 200, 200, 200, -2, -LastY() / SEGLEN   ' addDownhillToEnd
  trackLen = nSeg% * SEGLEN
  s% = SegOf%(PLAYERZ)
  segCol%(s% + 2) = 2 : segCol%(s% + 3) = 2       ' start line
  FOR n% = 1 TO RUMBLE
    segCol%(nSeg% - n%) = 3                       ' finish line
  NEXT
END SUB
```

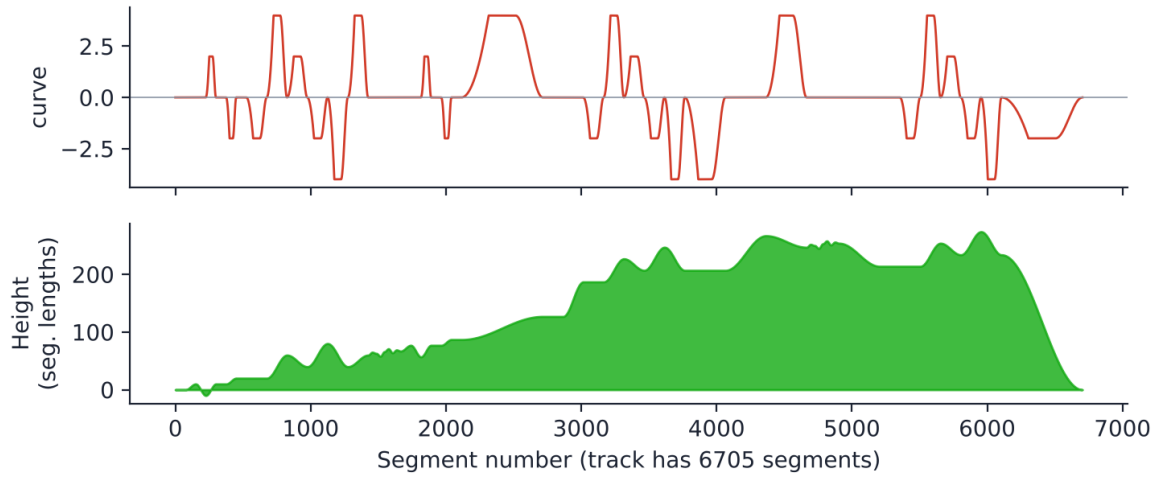

Fig. 16 – Curve value and height profile of the track. At the end a long descent leads back to the start height.

The track has about 6700 segments. With four arrays of 8 bytes each that is a good 210 KB – no problem on the PC3 thanks to PSRAM. If you want to save memory or loading time, build a shorter lap or store curve and height as 32-bit values.

5.2 Sprites: sheet, placement, scaling



Fig. 17 – SPRITES.BMP (320 × 240, enlarged). Objects on top, the three background strips below. Black = transparent.

Every sprite *type* is a rectangle in the sheet plus a **world width** in multiples of roadWidth. The original derives the world size from the pixel size of its big pictures; because our sheet is small, the world width is in the table directly. A tree is about 1.35 times as wide as half the road, a car 0.3 times.

Objects placed at the roadside hang on a segment, with a lateral **offset** (−1 = left edge, +1 = right edge, values beyond stand in the terrain). Because MMBasic has no dynamic lists, all objects lie in a pool; every segment points to its first object, every object to the next (linked list with −1 as the end):

RACER.BAS – hang an object on a segment

```
SUB AddSprite(s%, t%, o)
  IF s% >= nSeg% OR nSpr% > MAXSPR THEN EXIT SUB
  sprT%(nSpr%) = t% : sprO(nSpr%) = o
  sprNx%(nSpr%) = sprHd%(s%) : sprHd%(s%) = nSpr%
  INC nSpr%
END SUB
```

When drawing, a sprite gets the scale of its segment. Width and height in pixels come from world width × scale; the anchor (bottom centre for cars, bottom outside for roadside objects) shifts the rectangle. `BLIT RESIZE` does the scaling including transparency in firmware.

5.3 Clipping at the hill edge

If a tree stands behind a crest, its foot is hidden but its crown is visible. That is why every segment remembers the value of maxy at that time as **clip** during the road pass. When drawing the sprite, everything below this line is cut off – the same share of the height is left out in the sheet and in the target.

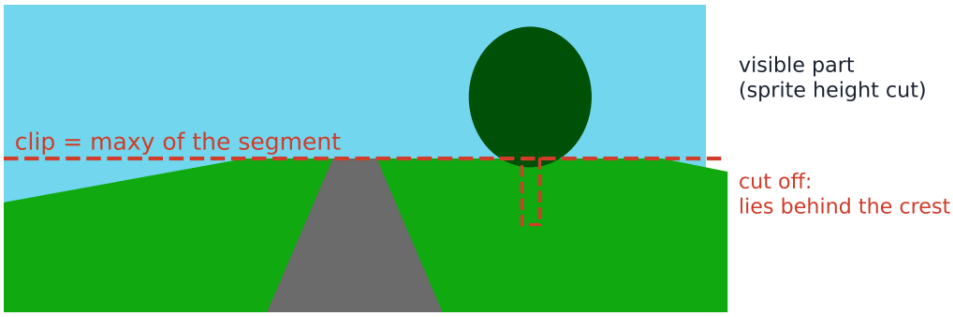

Fig. 18 – The tree stands behind the crest; only the part above the clip line is drawn.

RACER.BAS – scale, clip and draw a sprite

```
SUB DrawSpr(t%, sc, xx, yy, ox, oy, clip%)
  LOCAL dw, dh, ch, x, y, srch%
  dw = tWF(t%) * ROADW * sc * HW
  IF dw < 1 THEN EXIT SUB
  dh = dw * tH%(t%) / tW%(t%)
  x = xx + dw * ox : y = yy + dh * oy
  IF x >= SW OR x + dw < 0 OR y >= SH THEN EXIT SUB
  ch = MAX(0, y + dh - clip%)
  IF ch >= dh - 0.5 THEN EXIT SUB
  srch% = CINT(tH%(t%) * (1 - ch / dh))
  IF srch% < 1 THEN EXIT SUB
  BLIT RESIZE 2, F, tX%(t%), tY%(t%), tW%(t%), srch%, CINT(x), CINT(y), CINT(dw), CINT(dh - ch), 0
END SUB
```

5.4 Order: road forwards, sprites backwards

The road is drawn from front to back (because of maxy). Sprites, however, must be painted from back to front so that near objects cover far ones (painter’s algorithm). Hence two passes: first the road, storing the projected values and clip per segment; then a loop from the farthest to the nearest segment that draws cars, roadside objects and – in the player segment – the own car.

RACER.BAS – sprite pass (from SUB Render)

```
  ' --- sprites: back to front (painter's algorithm)
  BucketCars ps%
  off% = ps% - base% : IF off% < 0 THEN off% = off% + nSeg%
  FOR n% = DD - 1 TO 1 STEP -1
    s% = base% + n% : IF s% >= nSeg% THEN s% = s% - nSeg%
    r% = n% - off%
    IF r% >= -20 AND r% <= DD THEN
      k% = carHd%(r% + 20)
      DO WHILE k% >= 0
        sc = Lerp(rS1(n%), rS2(n%), carP(k%))
        sx = Lerp(rX1%(n%), rX2%(n%), carP(k%)) + sc * carO(k%) * ROADW * HW
        sy = Lerp(rY1%(n%), rY2%(n%), carP(k%))
        DrawSpr carT%(k%), sc, sx, sy, -0.5, -1, clip%(n%)
        k% = carNx%(k%)
      LOOP
    ENDIF
    k% = sprHd%(s%)
    DO WHILE k% >= 0
      sc = rS1(n%)
      sx = rX1%(n%) + sc * sprO(k%) * ROADW * HW
      DrawSpr sprT%(k%), sc, sx, rY1%(n%), CHOICE(sprO(k%) < 0, -1, 0), -1, clip%(n%)
      k% = sprNx%(k%)
    LOOP
    IF n% = off% THEN                              ' player car
      sc = CAMD / PLAYERZ
      IF ps% = 0 THEN y1w = 0 ELSE y1w = segY(ps% - 1)
      cy = Lerp(y1w, segY(ps%), pp) - camY
      t% = 15
      IF speed > 0 AND kL% THEN t% = 14
      IF speed > 0 AND kR% THEN t% = 16
      bounce = 1.5 * RND * (speed / MAXSPD) * RES * CHOICE(RND < 0.5, -1, 1)
      DrawSpr t%, sc, HW, HH - sc * cy * HH + bounce, -0.5, -1, SH
    ENDIF
  NEXT
```

5.5 Opponents: cars with a simple AI

In the original 200 cars drive, and every segment keeps a list of its cars that is re-linked when a car changes segment. For MMBasic a leaner way is better: **40 cars** at constant speed (trucks slower), and before every use they are sorted into lists per segment *relative to the player* in one pass – only for the window from 20 segments behind to DD segments in front of the player. No re-linking.

RACER.BAS – sorting the cars

```
SUB BucketCars(ref%)
  LOCAL i%, r%
  MATH SET -1, carHd%()
  FOR i% = 0 TO NCAR - 1
    r% = SegOf%(carZ(i%)) - ref%
    IF r% < -nSeg% \ 2 THEN r% = r% + nSeg%
    IF r% > nSeg% \ 2 THEN r% = r% - nSeg%
    IF r% >= -20 AND r% <= DD THEN
      carNx%(i%) = carHd%(r% + 20) : carHd%(r% + 20) = i%
    ENDIF
  NEXT
END SUB
```

The evasive AI looks 20 segments ahead. If a car finds a slower vehicle (or the player) on its lane there, it steers away – the harder, the closer the obstacle and the bigger the speed difference. If the obstacle is already at the edge, it moves towards the middle. Cars out of sight do not steer at all; overlaps nobody sees are allowed there.

RACER.BAS – evasive steering

```
FUNCTION CarSteer(i%, r%)
  LOCAL j%, k%, q%, cw, dir
  CarSteer = 0
  IF r% > DD THEN EXIT FUNCTION              ' out of sight: no steering
  cw = tWF(carT%(i%))
  FOR j% = 1 TO 19
    q% = r% + j%
    IF q% >= nSeg% THEN q% = q% - nSeg%
    IF q% = 0 THEN                           ' player segment ahead
      IF carV(i%) > speed THEN
        IF Overlap%(playerX, PLAYERW, carO(i%), cw, 1.2) THEN
          IF playerX > 0.5 THEN
            dir = -1
          ELSEIF playerX < -0.5 THEN
            dir = 1
          ELSE
            dir = CHOICE(carO(i%) > playerX, 1, -1)
          ENDIF
          CarSteer = dir / j% * (carV(i%) - speed) / MAXSPD
          EXIT FUNCTION
        ENDIF
      ENDIF
    ENDIF
    IF q% <= DD THEN
      k% = carHd%(q% + 20)
      DO WHILE k% >= 0
        IF carV(i%) > carV(k%) THEN
          IF Overlap%(carO(i%), cw, carO(k%), tWF(carT%(k%)), 1.2) THEN
            IF carO(k%) > 0.5 THEN
              dir = -1
            ELSEIF carO(k%) < -0.5 THEN
              dir = 1
            ELSE
              dir = CHOICE(carO(i%) > carO(k%), 1, -1)
            ENDIF
            CarSteer = dir / j% * (carV(i%) - carV(k%)) / MAXSPD
            EXIT FUNCTION
          ENDIF
        ENDIF
        k% = carNx%(k%)
      LOOP
    ENDIF
  NEXT
  IF carO(i%) < -0.9 THEN CarSteer = 0.1
  IF carO(i%) > 0.9 THEN CarSteer = -0.1
END FUNCTION
```

5.6 Collisions

- **Roadside objects:** only when the car drives beside the road are the objects of the player segment checked. On an overlap it is slowed to a fifth of the top speed and put at the start of the segment – it “slides” past the obstacle instead of stopping hard.
- **Cars:** if the player is faster than an overlapping car in his segment, he loses speed and ends up right behind it.
- The overlap is a simple 1D test across the road; the percentage parameter makes the hit box narrower (0.8) or wider (1.2).

RACER.BAS – overlap test

```
FUNCTION Overlap%(x1, w1, x2, w2, pc)
  LOCAL h
  h = pc / 2
  Overlap% = 1
  IF x1 + w1 * h < x2 - w2 * h THEN Overlap% = 0
  IF x1 - w1 * h > x2 + w2 * h THEN Overlap% = 0
END FUNCTION
```

5.7 Player car, HUD and lap times

The own car has three pictures (straight, left, right). A small random wobble proportional to the speed simulates the suspension. The original also has uphill pictures; they can be added as soon as the sheet has room for them.

The HUD lies in layer L and – as in the original – is only rewritten when the shown value changes. A lap ends when the camera crosses the start position; the best time is kept (the original stores it in the browser, here it could survive power-off via VAR SAVE).

RACER.BAS – HUD on the layer

```
SUB Hud
  LOCAL v%, t%, a$
  v% = 5 * CINT(speed / 500)
  t% = INT(curLap * 10)
  IF v% = hudV% AND t% = hudT% THEN EXIT SUB
  hudV% = v% : hudT% = t%
  FRAMEBUFFER WRITE L
  a$ = "ZEIT " + FmtTime$(curLap) + "  BESTE " + FmtTime$(bestLap) + "  "
  TEXT 4, 2, a$, "LT", 1, 1, col%(C_WHITE), col%(C_BLACK)
  TEXT SW - 4, 2, STR$(v%, 3) + " km/h", "RT", 1, 1, col%(C_YEL), col%(C_BLACK)
  FRAMEBUFFER WRITE F
END SUB
```

6 Speed on the RP2350

6.1 What each frame costs

For a full lap with 40 opponents every sixth frame was counted. The numbers do not depend on the hardware and show how much work the PicoMite faces:

Draw distance (segments)	segments drawn	trapezoids (POLYGON)	grass BOX	sprites (BLIT RESIZE)
300 (original)	46 (max. 80)	183 (max. 330)	47 (max. 81)	123 (max. 267)
150	42 (max. 77)	166 (max. 317)	43 (max. 78)	68 (max. 126)
120 (RACER.BAS)	40 (max. 77)	159 (max. 317)	41 (max. 78)	53 (max. 97)
100	38 (max. 75)	153 (max. 307)	39 (max. 76)	42 (max. 79)

Notably, the number of road segments drawn hardly depends on the draw distance – far segments are smaller than a row and drop out. The draw distance mainly sets how often the projection loop runs (two projections per segment) and how many sprites are scaled. Visually, at 120 segments only the last piece of road on the horizon is missing:

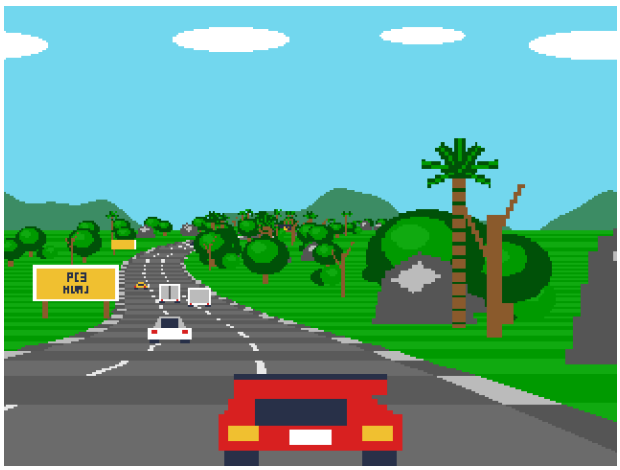

Fig. 19 – Draw distance 300 (original).

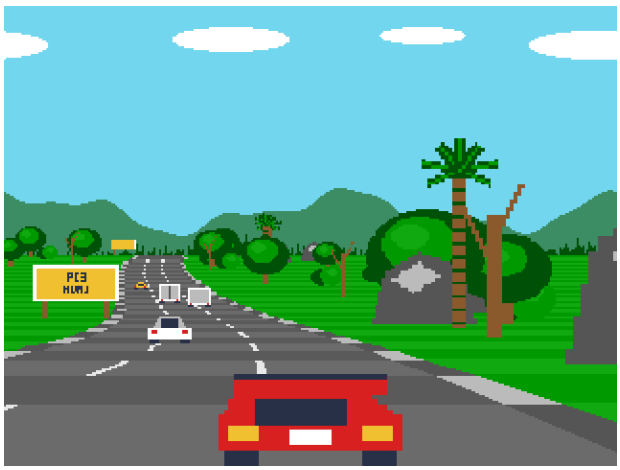

Fig. 20 – Draw distance 120 (RACER.BAS). The road ends a little earlier.

6.2 Rules for fast MMBasic

The MMBasic Performance Guide (V7) measures every construct on the RP2350. For the racer it follows:

- **Compile:** `OPTION COMPILE ON`. Assignments, IF, FOR, DO, SUB calls as well as BOX, LINE, TRIANGLE and the simple BLIT forms are translated. POLYGON, MATH and BLIT RESIZE still run interpreted – but each is a single call that works in C internally.
- **Let the firmware loop:** array forms of POLYGON, BOX, LINE and PIXEL are 20 to 100 times faster than BASIC loops. Hence collecting the trapezoids.
- **Integer coordinates:** float variables as graphics arguments cost a conversion on every call. Keep coordinates in % variables.
- **Fewer statements:** every statement costs a few microseconds before it does anything. Move constant parts out of the loop; if needed, inline SUB calls in the innermost loop (e.g. Quad) by hand later.
- **FOR instead of DO:** an empty FOR pass costs 2.3 µs according to the guide, a DO loop 5 to 11 µs (compiled 1.1 against 2.2 µs).
- **Measure, do not guess:** `OPTION PROFILING ON` as the first line shows the most expensive lines; `PRINT MM.INFO(COMPILE)` shows which share ran compiled.
- **Last reserve:** the tool mmb2csub translates single SUBs into machine code (CSUB) – loops and arithmetic 10 to 35 times faster according to the guide. First candidate would be the road loop in `Render`.

6.3 Knobs in the game

Knob	Effect	in RACER.BAS
draw distance DD	fewer projections and sprites	CONST DD = 120
number of opponents	AI loop and sprites	CONST NCAR = 40
plant density	sprites per frame (biggest item)	ResetSprites: STEP 3 → e.g. STEP 6
logic step	60 Hz logic costs three updates per frame at 20 fps	CONST STP (then adjust MAXSPD)
clock	378 instead of 252 MHz: +50 % for everything	OPTION RESOLUTION 640x480, 378000
lane lines	two trapezoids less per light segment	draw lines only up to n% < 40

7 The alternative: Lou’s raster road

Besides the segment method, Lou’s page describes the even older **raster technique** of the arcade machines. Instead of projecting segments, *every screen row* below the horizon is handled on its own. For every row it is computed once in advance how far away it is – the **Z map**:

Principle of the Z map

```
' Z map for a flat road (once at the start)
FOR y% = 1 TO 119                          ' rows below the horizon
  zmap(y%) = CAMH / y%                     ' distance of this row
  scl(y%)  = y% / 119                      ' scale relative to the bottom row
NEXT
```

- **Stripes:** whether a row is light or dark follows from *(zmap(y) + position)* modulo the stripe length.
- **Curves:** from bottom to top an offset and its change are added up (ddx += dx, x += ddx) – the same principle as in part 2, only per row.
- **Hills:** are faked by skipping or repeating rows; this is fiddlier than the real height of the segments.
- **Drawing:** every row consists of at most five horizontal pieces (grass, rumble, road, rumble, grass). With the array form of BOX all 120 rows can be drawn in **one** command.

On the PicoMite this is appealing because the work per frame is fixed (120 rows, no divisions at runtime) and the BOX arrays are drawn natively. Drawback: real hills with hiding and sprites tied to segments are much easier with the segment method. **So the segment method was chosen** – but drawing happens row by row with B0X arrays, as with the raster road (chapter 8). This mix combines real hills and sprites with fast filling and is the base of ROAD (Part II).

8 In practice: optimising on the Pico Computer 3

The first version ran on the PC3 straight away – at just under 10 frames per second. In a series of 13 versions it was sped up step by step with the built-in profiler (`OPTION PROFILING ON`) and own `TIMER` measuring points. Every change was measured on its own. This chapter sums up what really helped – and what did not.

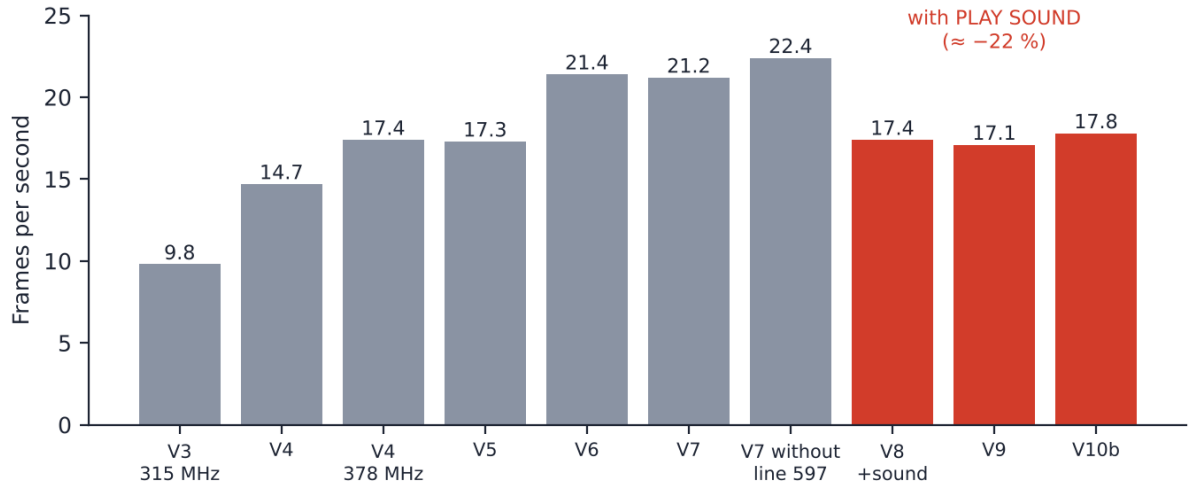

Fig. 21 – Frame rate per version, measured on the PC3 over a full lap. Red: with sound via PLAY SOUND (chapter 9).

8.1 Measurement log

Version	Change	FPS	Finding
V1/V3	first version, 175 km/h, clock 315 MHz	9.8	logic runs almost 9 times per frame
V4	car AI and sorting once per frame instead of per logic step; SegOf and Quad saved	14.7	logic from ~40 to ~6 ms
V4	clock 378 MHz (`OPTION RESOLUTION 640x480, 378000`)	17.4	+18 %, without changing code
V5	create buffer 2 before the layer (fast video memory), HUD straight into F	17.3	background −30 %, sprites unchanged
V6	road row by row with one BOX array instead of POLYGON; leave out far/tiny sprites	21.4	filling from ~20 to 2 ms
V7 + l. 597	one-line `IF … THEN EXIT FOR` removed	22.4	was the only line running interpreted
V8–V10	sound with PLAY SOUND (engine, gears, noises, Doppler)	17–18	sound generation costs ~22 %
V12/V13	MOD music + effect samples, key M switches to engine mode	–	see chapter 9

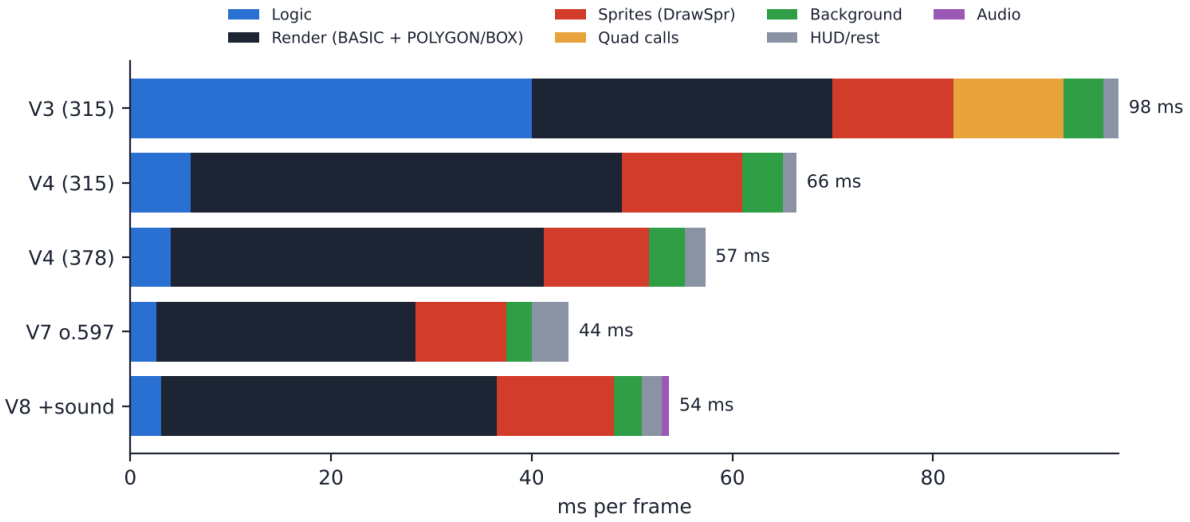

Fig. 22 – Where the time per frame goes (from the profiler reports). The logic shrinks drastically, then drawing dominates.

8.2 Lessons for MMBasic V7 on the RP2350

1. The compiler works – but calls stay expensive

`PRINT MM.INFO(COMPILE)` reported over 99 % compiled statements throughout. Still, a function with a single line of arithmetic (`Seg0f`) cost about 30 µs per call – with hundreds of thousands of calls 13 ms per frame. The call itself is the cost, not the arithmetic. **Rule: write small helpers out in hot loops.** A CSUB does not help here, it is called the same way.

2. Fixed time step: expensive logic multiplies

At 10 frames per second, 9 logic steps run per frame. Everything in the step counts nine times – and slows drawing further, which in turn creates more steps. In V3 the logic ate 40 % of the time this way. **Rule: only what the physics really needs (moving, collision) in the logic step; AI, sorting and sound once per frame.** The steering of the opponents is multiplied by the number of steps of the frame for that.

3. POLYGON is convenient but slow at filling

The measuring points in Render showed: the one POLYGON array with all trapezoids cost 17–22 ms (roughly 0.7 µs per pixel). The same area as horizontal 1-pixel rectangles in a BOX array: 2 ms. In return the BASIC share rises, because computing happens per row.

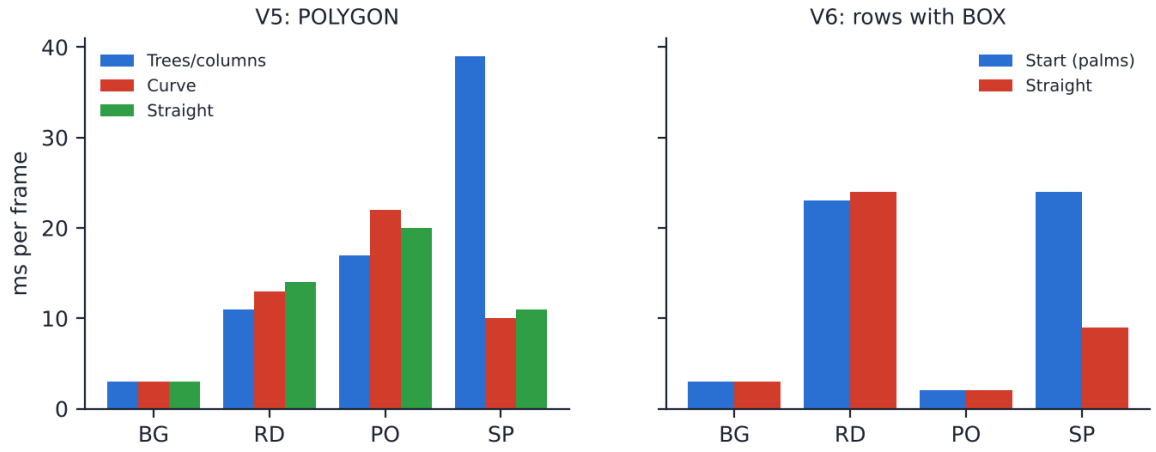

Fig. 23 – Measuring points in Render (ms per frame): background (BG), road loop (RD), filling (PO), sprites (SP).

Two pitfalls with the BOX array

The width is checked: a rectangle wider than 320 pixels stops with “Width … is invalid (valid is 1 to 320)” – unlike POLYGON, BOX does not clip by itself. Every piece must be limited to 0…319 first. **No end marker:** BOX always draws the whole array. Unused entries are set to an invisible 1-pixel rectangle in sky colour with MATH SET before every frame.

RACER_V13.BAS – road row by row (excerpt from Render)

```
        ' row by row: for every screen row rumble strip, road, lines
        yb% = MIN(MIN(y1%, maxy%), SH) - 1
        fx = (x2% - x1%) / (y1% - y2%) : fw = (w2% - w1%) / (y1% - y2%)
        FOR yy% = MAX(y2%, 0) TO yb%
          f = y1% - yy%
          xc = x1% + fx * f : ww = w1% + fw * f : rr = ww / MAX(6, 2 * LANES)
          ' BOX needs 1 <= width <= 320: clip every piece to the screen
          a% = xc - ww - rr : e% = xc + ww + rr + 1          ' rumble strip (outside to outside)
          IF a% < 0 THEN a% = 0
          IF e% > SW THEN e% = SW
          IF e% > a% THEN
            bx%(nb%) = a% : by%(nb%) = yy% : bw%(nb%) = e% - a% : bc%(nb%) = rbC%(g%)
            INC nb%
          ENDIF
          a% = xc - ww : e% = xc + ww + 1                    ' road
          IF a% < 0 THEN a% = 0
          IF e% > SW THEN e% = SW
          IF e% > a% THEN
            bx%(nb%) = a% : by%(nb%) = yy% : bw%(nb%) = e% - a% : bc%(nb%) = rdC%(g%)
            INC nb%
          ENDIF
          IF lnC%(g%) <> -1 THEN
            ll = MAX(1, ww / MAX(32, 8 * LANES)) : lw = ww * 2 / LANES : lx = xc - ww + lw - ll / 2
            FOR ln% = 1 TO LANES - 1
              a% = lx : e% = lx + ll                         ' lane line
              IF a% < 0 THEN a% = 0
              IF e% > SW THEN e% = SW
              IF e% > a% THEN
                bx%(nb%) = a% : by%(nb%) = yy% : bw%(nb%) = e% - a% : bc%(nb%) = lnC%(g%)
                INC nb%
              ENDIF
              lx = lx + lw
            NEXT
          ENDIF
        NEXT
      ENDIF
```

4. A single interpreted line can cost 5 ms

Under “top commands by dispatch count” the profiler counts the commands that did *not* run compiled. Suddenly `If` showed up there with 222,000 calls. The cause was a harmless safety line `IF nb% + LANES + 1 > MAXB THEN EXIT FOR` in the row loop: the compiler apparently does not translate a one-line IF with EXIT FOR. One-line IFs with assignments or EXIT SUB did run compiled. **Rule: look at the dispatch list after every change.**

5. Where the buffers live matters

If layer L is created first, it takes the fast pre-allocated video memory, and buffer 2 ends up in variable memory – on the PC3 in PSRAM. The other way round (buffer 2 first, no layer) copying the background strips became 30 % faster, and it hardly profited from the higher clock – a sign of a memory brake. `BLIT RESIZE` for the sprites stayed at 230–300 µs per call: a fixed cost per command. The only lever there is the *number* of sprites (limits `SCENDD` and `SPRMIN`).

6. Small things with an effect

- **Transparent text:** with HDMI, TEXT allows −1 as background colour – the HUD then needs no layer.
- **Cache HUD texts:** only rebuild strings when the value changes; they are drawn every frame anyway.
- **Clock:** 378 MHz instead of 315 MHz (75 Hz mode) brings about 18 %.
- **Game balance:** centrifugal force 0.45 caused 220 collisions per lap, 0.38 only 15. Frequent crashes also cost frame rate.

9 Sound: what the experiments showed

A racing game without engine noise feels empty – the acoustic feedback is missing. The PicoMite offers several ways to make sound. The experiments showed what each of them can do, what it costs and where the limits are.

9.1 The audio systems exclude each other

System	Can	Limit in the game
`PLAY SOUND`	4 continuous tones (sine, square, triangle, saw, noise, own waveform U), any frequency	costs ~22 % CPU; noises sound synthetic
`PLAY BBC SOUND`	4 channels with note queue and envelopes	good for jingles; not tried as an engine
`PLAY MODFILE` + `PLAY MODSAMPLE`	music on 4 channels + 4 effect channels with real samples	effects without loop and without pitch
`PLAY SAMPLE`	wave table with ADSR envelope	restarts the envelope on every call
`PLAY MIDIFILE` / `PLAY MIDI`	MIDI files in the background; real-time MIDI via a VS1053 codec	not tested

According to the manual `PLAY SOUND` blocks every other audio output until `PLAY STOP`. **Engine via PLAY SOUND and MOD music at the same time are therefore not possible.** RACER_V13 solves this with two modes; key M switches.

9.2 PLAY SOUND: engine, gears, Doppler

- **Cost:** as soon as PLAY SOUND runs, the frame rate drops by ~22 % (22.4 → 17.4), evenly over all parts of the program – no matter whether 2 or 4 tones are active. The own sound logic only costs ~1 ms per frame.
- **Own waveform (type U):** `PLAY LOAD SOUND` loads 4096 values. The frequency says how often the *whole* table is played per second. With 16 engine cycles in the table the pitch is frequency × 16 – confirmed exactly in the recording (29 Hz idle, 150 Hz full throttle).
- **Gears:** 5 gears with a jump in revs when shifting make the engine lively; flutter (±1–2 %) and overrun without throttle against monotony.
- **Doppler:** the nearest opponent gets its own tone, higher when approaching, lower after overtaking, left/right depending on the side. The effect was convincing straight away.

Why the first engine sounded “metallic”

The recording from the PC3 showed: the pitch was right, but the 2nd to 8th harmonics were almost as loud as the base tone, at idle the 5th harmonic was even the loudest. The cause was too much ringing in the waveform (3.5 and 7 times the base frequency). V10 rings at 1.5 and 3 times, every second firing cycle is weaker (“burble”), and the table is smoothed twice at the start – an “exhaust” as a low-pass filter.

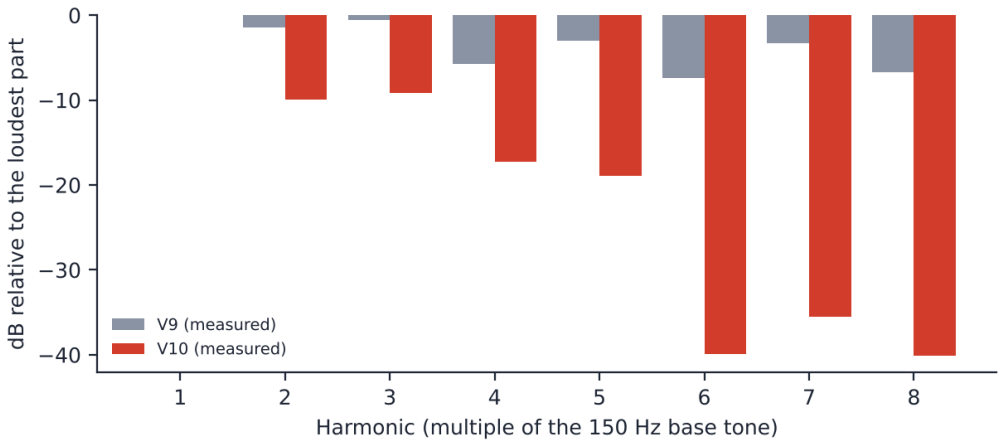

Fig. 24 – Harmonics at full throttle, measured in recordings from the PC3: V9 almost flat (metallic), V10 with a clearly dominant base tone.

RACER_V13.BAS – engine waveform with exhaust filter

```
SUB MakeEngine
  LOCAL c%, k%, n%, i%, p%, t, a, a2, sm, mx
  n% = 4096 \ ECYC
  FOR c% = 0 TO ECYC - 1
    a = CHOICE(c% MOD 2 = 0, 1, EODD) * (0.85 + 0.3 * RND)
    a2 = 0.15 + 0.2 * RND
    FOR k% = 0 TO n% - 1
      t = k% / n%
      sm = a * EXP(-t * 5) * SIN(2 * PI * t * EH1)                    ' firing
      sm = sm + a2 * EXP(-(t - 0.55) * (t - 0.55) * 40) * SIN(2 * PI * t * EH2)
      engR(c% * n% + k%) = sm + ENOISE * (RND - 0.5)                  ' mechanics
    NEXT
  NEXT
  ' exhaust: moving average over ELP values, twice (wrapping)
  FOR p% = 1 TO 2
    sm = 0
    FOR k% = 0 TO ELP - 1
      sm = sm + engR(k%)
    NEXT
    FOR i% = 0 TO 4095
      engS(i%) = sm / ELP
      sm = sm - engR(i%) + engR((i% + ELP) MOD 4096)
    NEXT
    MATH SCALE engS(), 1, engR()
  NEXT
  ' remove DC, scale to full level
  sm = MATH(MEAN engR())
  mx = MAX(MATH(MAX engR()) - sm, sm - MATH(MIN engR()))
  FOR i% = 0 TO 4095
    engT%(i%) = 2048 + 2000 * (engR(i%) - sm) / mx
  NEXT
  MEMORY PACK engT%(), engW%(), 4096, 16
  PLAY LOAD SOUND engW%()
END SUB
```

Noise type P sounds metallic

Periodic noise (type P) produced a metallic “chirp” when shifting and on the rumble strip – visible in the recording as broadband bursts up to over 6 kHz. Replacement: a low sine (55 Hz) as shift pop, a low square wave as rumble, finer noise (type N, 1500 Hz) for the grass.

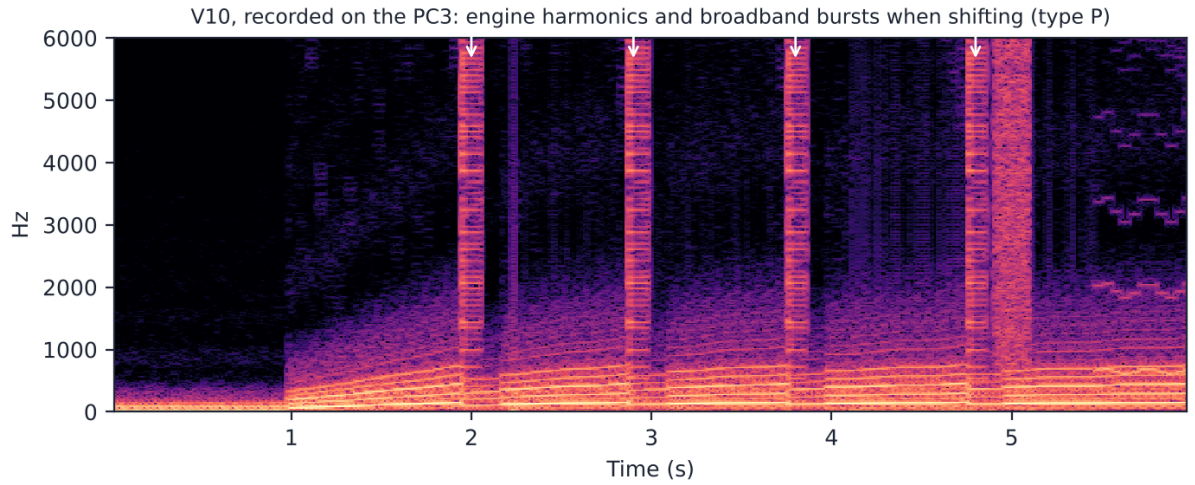

Fig. 25 – Spectrogram of the V10 recording: on top the bursts of the shift noise (arrows), below the harmonic lines of the engine.

9.3 MOD music with effect samples

`RACER.MOD` is a 4-channel ProTracker module made specially (synthwave, A minor, 128 BPM, 82 s until it repeats). Slots 20–27 hold the effects the game lays over the music with `PLAY MODSAMPLE nr, channel, volume`:

Slot	Effect	Based on	Channel
20	crash	recording impact.ogg + short tyre screech	1
21	grass/gravel (2 s)	grain from pieces of the impact recording	3
22	tyre squeal (3 s)	recording skid.ogg	3
23	shifting	low sine pop	2
24	rumble strip (2 s)	impact hits, ~22 per second	3
25	pass-by	engine recording with Doppler	4
26	lap jingle	synthetic	2
27	silence	ends a running effect	3

The most important finding: MODSAMPLE knows no loops

The test with a looped sample (bass, 0.5 s) showed: after half a second the tone is gone. It follows:

- **Continuous noises** (squeal, grass, rumble) must be restarted regularly. With short samples (0.5 s) and fades a pumping “wish-wish-wish” appeared. The solution: **long samples** (2–3 s) with only an 8 ms fade, restart shortly before the end, and to stop a **silence sample** that replaces the running effect on its channel. A run-on of 0.25 s prevents chopping at state borders.
- **An engine with variable revs via MOD** is not cleanly possible with V7.00.00: crossfading between rev steps on two channels pulses audibly. A wish for the firmware (TheBackShed forum): loop support and stop for `PLAY MODSAMPLE` – that would make mixing music and engine possible.
- **Playback rate:** the effects are built for 8287 Hz (note C-2). Content above ~4 kHz is missing; the tyre squeal sounds a bit duller than the recording.

RACER_V13.BAS – ground effects with long samples, run-on and stop

```
' --- ground: repeat the sample while the state lasts
t% = 0 : ax = ABS(playerX)
IF ax > 1 AND speed > 0 THEN
  t% = 21                                    ' grass/gravel
ELSEIF FXNOISE AND ax > 0.85 AND sp > 0.1 THEN
  t% = 24                                    ' rumble strip
ELSEIF FXNOISE AND ABS(segCv(ps%)) * sp * sp > 1.4 THEN
  t% = 22                                    ' tyre squeal
ENDIF
IF t% > 0 THEN
  fxOff = TIMER + 250                        ' run-on: short flicker not audible
  IF t% <> fxT% OR TIMER >= fxNext THEN
    PLAY MODSAMPLE t%, 3, FXVOL
    fxT% = t%
    fxNext = TIMER + CHOICE(t% = 22, 2880, 1880) ' restart shortly before the sample ends (3.0 / 2.0
    s)
  ENDIF
ELSEIF fxT% > 0 THEN
  IF TIMER >= fxOff THEN
    PLAY MODSAMPLE 27, 3, 1                  ' silence sample ends the effect
    fxT% = 0
  ENDIF
ENDIF
```

9.4 Outlook (state of the study)

- **Handling:** so far the acceleration is the same over the whole range. It becomes more realistic (and more exciting) if it falls with speed, e.g. `ACCEL · (1 − 0.7 · sp²)` with sp = speed / MAXSPD, or graded per gear.
- **Opponents:** 90 cars fill the track well; the cost grows with their number, the info line shows it.
- **Performance:** the biggest lever left is the road loop as a CSUB via mmb2csub (towards 30 FPS).
- **Sound:** with loop support for PLAY MODSAMPLE, music and engine would be possible at the same time.

Part II

ROAD 1.0 – the game

From prototype to finished game: what changed and why, a tour through the source code, your own tracks, graphics and music – and an invitation to join in.

10 From prototype to game

How ROAD runs
ROAD is written **entirely in MMBasic** – without a CSUB, using the firmware **PicoMite V7.00.00** (PicoMitePC3) on the Pico Computer 3. Two settings make the difference; set them once at the prompt:
`OPTION RESOLUTION 640x480, 378000` – clock **378 MHz** instead of 252 MHz, everything runs about 50 % faster.
`OPTION COMPILE ON` – the V7 compiler translates the program on RUN; more than 99 % of the statements run compiled.
Video: `https://youtu.be/W4qjgm6pynE`



Fig. 26 – Title screen with the road sign (here: credits).



Fig. 27 – Texas: mesas, cacti, bull skulls.

The feasibility study of Part I showed that the technique works on the Pico. In a good twenty steps it became the game **ROAD**. Every step was driven on the Pico Computer 3, measured (profiler, frame rate) and often checked on video. What was added:

Area	ROAD 1.0
Game flow	title with demo lap, driver name, 3 laps with time limit and checkpoints, rolling out after time-out with a chance to be saved, records per track, unlocking the next track, pause and quit with a question
Tracks	five tracks as text files: **Green Valley**, **Texas** (bull skulls as obstacles), **California** (coast at sunset, 4 and 2 lanes), **Circuit** (figure of eight with bridge, tunnel, hairpin, curbs), **City** (night, three lanes, river bridge, old town, underpass)
World tour	all tracks in a row, 2 laps each, times added up, own record list; finishing it unlocks everything
Options	ROOKIE / VETERAN (time margin, traffic, own lists), PRACTICE (no traffic, to learn a track), menu music
Graphics	own sprite sheet and palette per track, background of up to five pixel-art layers with parallax, drawn tunnel, coast with sand and water, lane changes with tapering, mirrored sprites
Sound	own MOD with effect samples per track; engine mode (key E), effects without music (key M)

10.1 What changed in the technique – and why

The game does much more than the study: wider roads with up to four lanes, coast, tunnel, 90 cars, menus. To stay at **17 to 20 frames per second** anyway, ROAD departs from Jake Gordon's original on purpose in a few places. The basic idea – segments, projection, curve accumulation, sprites with clipping – is unchanged.

Road: row rectangles in chunks

The study already drew the road row by row with a `BOX` array instead of `POLYGON` (chapter 8). With coast, sand and four lanes the array grew to more than 1700 entries – and `BOX` always draws the *whole* array. Filling and drawing the unused entries with `MATH SET` suddenly cost 5 frames per second. ROAD therefore draws in **chunks of 128 rectangles**: `MEMORY COPY INTEGER` copies each chunk into a small work array, and after the last entry comes one chunk of invisible dummy rectangles. Only what really exists in this frame is drawn.

ROAD.BAS – drawing the row rectangles in chunks (from Render)

```
' draw only the nb% used entries: pad with one dummy chunk, then BOX in chunks of CHB
MEMORY COPY INTEGER pkAX3%, pkAX% + 8 * nb%, CHB
MEMORY COPY INTEGER pkAY3%, pkAY% + 8 * nb%, CHB
MEMORY COPY INTEGER pkAW3%, pkAW% + 8 * nb%, CHB
MEMORY COPY INTEGER pkAC3%, pkAC% + 8 * nb%, CHB
FOR k% = 0 TO nb% - 1 STEP CHB
  MEMORY COPY INTEGER pkAX% + 8 * k%, pkAX2%, CHB
  MEMORY COPY INTEGER pkAY% + 8 * k%, pkAY2%, CHB
  MEMORY COPY INTEGER pkAW% + 8 * k%, pkAW2%, CHB
  MEMORY COPY INTEGER pkAC% + 8 * k%, pkAC2%, CHB
  BOX pkX%(), pkY%(), pkW%(), 1, 0, pkC%(), pkC%()
NEXT
```

Hot code stays in one piece

A SUB call costs much more in MMBasic than the lines in it. An attempt to split `Render` into clear sub-SUBs cost measurable frame rate and was reverted. That is why `Render` is long, the road width is computed inline instead of by a function, and `DrawSpr` works with global helper variables instead of LOCALs. The rule of chapter 8 still applies: **in the innermost loop, call nothing you can write out**.

Background: pictures instead of columns

The study copied three 320 pixel wide strips from the sprite buffer; later versions computed hills and tree line as column rectangles. ROAD 1.0 puts up to five **pixel pictures of any width** into the RAM image slots of the RP2350 (PSRAM) and copies the visible window with `BLIT FLASH` every frame, black is transparent. Measured on the PC3: 1.3 ms for a 320 × 70 pixel layer – faster than a single row of column rectangles (1.5 ms). Dithering, snow, windows and clouds per track can now be painted freely (chapter 12.4).

Colours: 16 slots, all different

Drawing commands get a 24-bit colour; the firmware looks up the matching slot from it. Two slots with the same RGB555 colour are therefore tricky: the first one is always found. A palette of your own in a track file needs **16 different colours**. For the tunnel only road, rumble strips, lines and cars are darkened; if two slots fall together, `MakeDark` moves them one step apart. The landscape is hidden in the tunnel anyway and appears in its normal colours at the exit.

Input: driving via KEYDOWN, menus via INKEY$

`KEYDOWN` reports held keys (throttle + steering at once), `INKEY$` delivers typed characters (names, menus; also works over the terminal). A lesson from testing: in the race, the question "QUIT RACE?" first waited for `INKEY$` – and never got a key. Since then *everything* in the race is read via `KEYDOWN` and key edges.

Random numbers with a fixed seed

`RANDOMIZE` has no effect on the RP2350. ROAD uses a small random generator of its own (`Rn()`) with a seed per track – so every tree stands in the same place at every load.

11 A tour through the source code

`ROAD.BAS` is a single file with about 2600 lines and English comments. From top to bottom it is ordered like this:

Section	Content	important SUBs
Constants, DIM	screen, road, sound, physics, game flow, palette slots, buffers, state	–
Main loop	input → states → logic in a fixed time step → sound → drawing	–
Initialisation	screen mode, buffers, palette	Init, ApplyPal
Track files	parser for roadN.trk, finishing the track, lanes, coast, tunnel	LoadTrack, FinishTrack, Zone, Side, Tunnel, MakeDark
Road building, sprites	building blocks from Part I, planting, obstacles	AddRoad, AddSprite, Scatter, Clusters
Input, world	keys, physics, collisions, checkpoints, cars	ReadKeys, Update, MoveCars, CarsAI, CarHit
Drawing	background, road, coast, tunnel, sprites	Render, DrawLy, DrawSpr
HUD, overlays, menus	display, title, options, credits, result	Hud, Overlay, Options, Credits, TourResult
Game flow	state machine, times, records, world tour	StateLogic, StartRace, RaceTimes, GoResult
Files, map, sound	road.cfg, .hi, mini map, engine, MOD effects	SaveCfg, BuildMap, SoundFor, AudioMod

11.1 The main loop

As in Part I, the logic runs in fixed steps (88 per second), drawing happens as often as possible. New is that input is split by state and that cars, tunnel palette and sound are handled only once per frame:

ROAD.BAS – main loop (excerpt)

```
tLast = TIMER : acc = 0 : fpsT = TIMER
DO
  tNow = TIMER
  dt = MIN(250, tNow - tLast) / 1000
  tLast = tNow
  ' --- input: menus via INKEY$ (also works over the terminal), driving via KEYDOWN
  IF st% = S_TITLE OR st% = S_NAME OR st% = S_RESULT THEN
    MenuKeys
    IF st% <> S_RESULT THEN Autopilot
  ELSE
    ReadKeys
    GameKeys
  ENDIF
  StateLogic
  ' --- world
  IF st% <> S_PAUSE AND st% <> S_RESULT THEN      ' result screen: picture stands until ENTER
    acc = acc + dt
    nUpd% = 0
    ps0% = INT((position + PLAYERZ) / SEGLEN) MOD nSeg%
    DO WHILE acc > STP
      acc = acc - STP
      Update STP
      INC nUpd%
    LOOP
    MoveCars nUpd% * STP
    IF hasTun% THEN                               ' tunnel: dark palette while the car is inside
      tnK% = INT((position + PLAYERZ) / SEGLEN) MOD nSeg%                ' dark while inside, normal
      TUNOUT before the exit
      tnK% = ((segCol%(tnK%) AND 65536) <> 0) AND ((segCol%((tnK% + TUNOUT) MOD nSeg%) AND 65536) <> 0)
      IF tnK% <> inTun% THEN
        inTun% = tnK%
        IF inTun% THEN MATH ADD palD%(), 0, pal%() ELSE MATH ADD palN%(), 0, pal%()
        ApplyPal
      ENDIF
    ENDIF
    CarHit ps0%
    CarsAI nUpd%
  ENDIF
  IF GEARS THEN Gearbox speed / MAXSPD
  IF SOUND THEN
    SoundFor
    IF st% >= S_COUNT AND st% <> S_RESULT AND st% <> S_PAUSE THEN   ' no car effects in the menu
      IF mMod% THEN
        AudioMod
      ELSE
        Audio
      ENDIF
    ENDIF
  ENDIF
  Render
  Hud
  Overlay
  FRAMEBUFFER COPY F, N
```

11.2 The state machine

State	Meaning
S_TITLE / S_NAME	title with demo lap, track choice, options, credits / name entry
S_START	curtain closes, track is loaded, curtain opens
S_COUNT	3 – 2 – 1 – GO
S_RACE	race; checkpoints give time, the finish line counts laps
S_TOVER	time is up: no throttle, the car rolls – if it reaches a checkpoint while rolling, the race goes on
S_FINISH	at the finish: the car brakes by itself (brake light)
S_RESULT	result, the picture stands until ENTER; in the world tour the interim standings

State	Meaning
S_PAUSE	pause or the question “QUIT RACE?”

`StateLogic` moves the states on, `Overlay` draws sign, messages and panels per state. The road sign on the title is one area with changing content: track choice, world tour page, options, credits or the quit question.

11.3 What is inside a segment

As in Part I, every segment has height, curve and sprite list. Everything else is packed into the bits of a single integer `segCol%()` so that the road loop needs only one value:

Bits	Meaning
0–2	colour set: 0 light, 1 dark, 2 start, 4 checkpoint
3 / 4	coast right / left (sand and water)
5–7	number of lanes (lane lines)
8–15	road width × 100 (tapering on lane changes)
16	tunnel
17	obstacle in this segment (collision is only checked there)

11.4 How a frame is made

- **Sky:** `CLS` in slot 1.
- **Background:** `DrawLy` copies the picture layers from back to front, with a filled area below.
- **Road:** from front to back, row by row into the rectangle lists (rumble, road, lines; coast in a list of its own). Tunnels are drawn as rings in two dark tones; behind the exit, drawing only continues through the opening.
- **Sprites:** from back to front with `BLIT RESIZE`, cut off at hills; in between the own car with brake light, steering pictures and dust.
- **HUD and overlay,** then `FRAMEBUFFER COPY F, N`.

11.5 Background layers and the compass

Every layer moves sideways in curves. So that exactly the same picture appears again after a lap, the offset does not simply follow the sum of the curves but the **whole turns** the road really makes: California turns around once – its mountain layer is a 1920 pixel wide panorama that passes exactly once per lap, so the sun appears exactly once. Where left and right curves cancel out (Texas, the figure of eight), every picture comes back by itself.

ROAD.BAS – picture layers with wrap-around and their own speed

```
SUB DrawLy
  LOCAL i%, o%, y%, h%, yi%, w%, v
  v = TREESPD * playerY * RES
  FOR i% = 0 TO nLy% - 1
    y% = lyY%(i%) + INT(lyS(i%) * v) : h% = lyH%(i%) : yi% = 0
    IF y% < 0 THEN yi% = -y% : h% = h% + y% : y% = 0
    IF y% + h% > SH THEN h% = SH - y%
    IF h% > 0 THEN
      w% = lyW%(i%) : o% = INT(bgP * lyR(i%)) : o% = o% - w% * INT(o% / w%)
      IF w% - o% >= SW THEN
        BLIT FLASH 4 + i%, F, o%, yi%, 0, y%, SW, h%, 0
      ELSE
        BLIT FLASH 4 + i%, F, o%, yi%, 0, y%, w% - o%, h%, 0
        BLIT FLASH 4 + i%, F, 0, yi%, w% - o%, y%, SW - w% + o%, h%, 0
      ENDIF
    ENDIF
  NEXT
  i% = nLy% - 1 : y% = lyY%(i%) + lyH%(i%) + INT(lyS(i%) * v)
  IF y% < SH THEN BOX 0, y%, SW, SH - y%, 0, col%(lyFill%), col%(lyFill%)
END SUB
```

11.6 Times and difficulty

The track files hold the **clean driving times** per sector (no traffic, no mistakes). At the start the game adds the margin and gives two extra seconds at the first checkpoint and one at the second – so it gets tighter towards the end of a lap:

ROAD.BAS – time limits from the clean times

```
SUB RaceTimes
  LOCAL i%, f
  f = CHOICE(optDf%, VETF, ROOKF)
  timeStart = CINT(tBase(0) * f)
  FOR i% = 1 TO 3 : tBon(i%) = CINT(tBase(i%) * f) : NEXT
  tBon(1) = tBon(1) + 2 : tBon(2) = tBon(2) + 1     ' more slack early, more stress towards the end
END SUB
```

12 Modding: your own tracks, graphics and music



Fig. 28 – California: coast, palms, sunset.



Fig. 29 – City at night.

Almost everything that makes a track is outside the program: in a text file, a sprite sheet, up to five background pictures and a MOD. For a new track you only touch `ROAD.BAS` in one place.

12.1 Creating a new track

- Copy and rename the files of an existing track, e.g. `road6.trk`, `road6.bmp`, `road6.mod`, `road6sky/far/near.bmp`.
- In `ROAD.BAS` set `CONST NTRK = 5` to 6. The menu and the world tour pick up every existing file `road1.trk` … `roadN.trk` automatically.
- In `road6.trk` change name, road, sprites, palette and times. Comments start with `'`; the header of `road1.trk` has a short description of every command.
- Drive a few clean laps with MODE = PRACTICE and enter the times (12.6).

12.2 The track file

One command per line, values separated by commas. Missing header values have defaults. Sprites always come *after* the road.

Command	Values	Effect
NAME	text	name in the menu
SHEET · MUSIC · MUSICFX	file	sprite sheet, MOD with music, MOD with effects only (key M)
TIME	start, cp1, cp2, line	clean sector times in seconds (12.6)
CARS · CARSPD · CARTYPE	number / min, max / from, to	traffic: number, speed, vehicle types (3 = truck)
PAL	slot, &Hrrggbb	palette colour; slots 4, 5, 7, 8, 11–15 also colour the cars
SPRDEF · MIRROR	type, x, y, w, h, world width, small / type, new, x, y	redefine a sprite cell / mirrored copy (type 38–43)
HAZARD	type	sprite type is an obstacle (over it: heavy, touched: light slowdown)
LAYER · BGFILL	file, y, speed [, repeats] / slot	background picture layer (12.4) / colour below
BG · BGSTYLE · STRIP	heights / 0, 1 / y	background without pictures (computed)
LANES · WIDTH · ZONE	lanes / factor / from, to, lanes, width, taper	lanes and width, also per section
SIDE	from, to, sand width, side	coast with sand and water (−1 left, 1 right)
TUNNEL	from, to	drawn tunnel; portal as sprite type 37
RUMBLE · LINES · HUDCOL	slot, slot / 0, 1 / slot	rumble colours (e.g. red/white), lane lines, HUD colour
MAPTURN · MAPK	degrees / factor	shape of the mini map: 360 loop, 0 figure of eight; curvature
STRAIGHT · CURVE · HILL	n / n, curve, height / n, height	building blocks from Part I (3·n segments each)
ROLLING · SCURVES · BUMPS · DOWNHILL · ROAD	see road1.trk	waves, S-curves, bumps, back to height 0, basic block
SPRITE	segment, type, offset [, scale]	one object; |offset| < 1 stands on the road
SCATTER · CLUSTERS	see road1.trk	planting in sections, groups with boards

12.3 The sprite sheet



Fig. 30 – road2.bmp (Texas) in the colours of the track. Magenta = transparent.

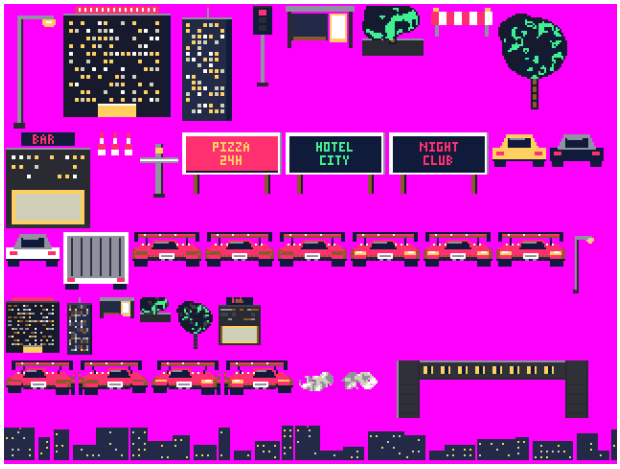

Fig. 31 – road5.bmp (City).

- **Format:** BMP with 320 × 240 pixels, painted in the **16 standard colours** (slot 0 black, 1 blue, 2 myrtle …). `LOAD IMAGE` puts every pixel into the slot of its standard colour; in the game it appears in the palette colour of the track. Black is transparent.
- **Fixed cells:** the cars (types 14–23) and the steering and dust pictures (31–36) must stay in place – they are used on all tracks.
- **Free cells:** plants, boards and objects (0–13, 24–30, 37) may be replaced; with `SPRDEF` a cell gets new dimensions and a new world width (how wide the object is compared to half the road).
- **Mirroring:** `MIRROR` copies a cell mirrored into free space at load time – so a street lamp needs only one drawing for both sides of the road.

12.4 Background pictures

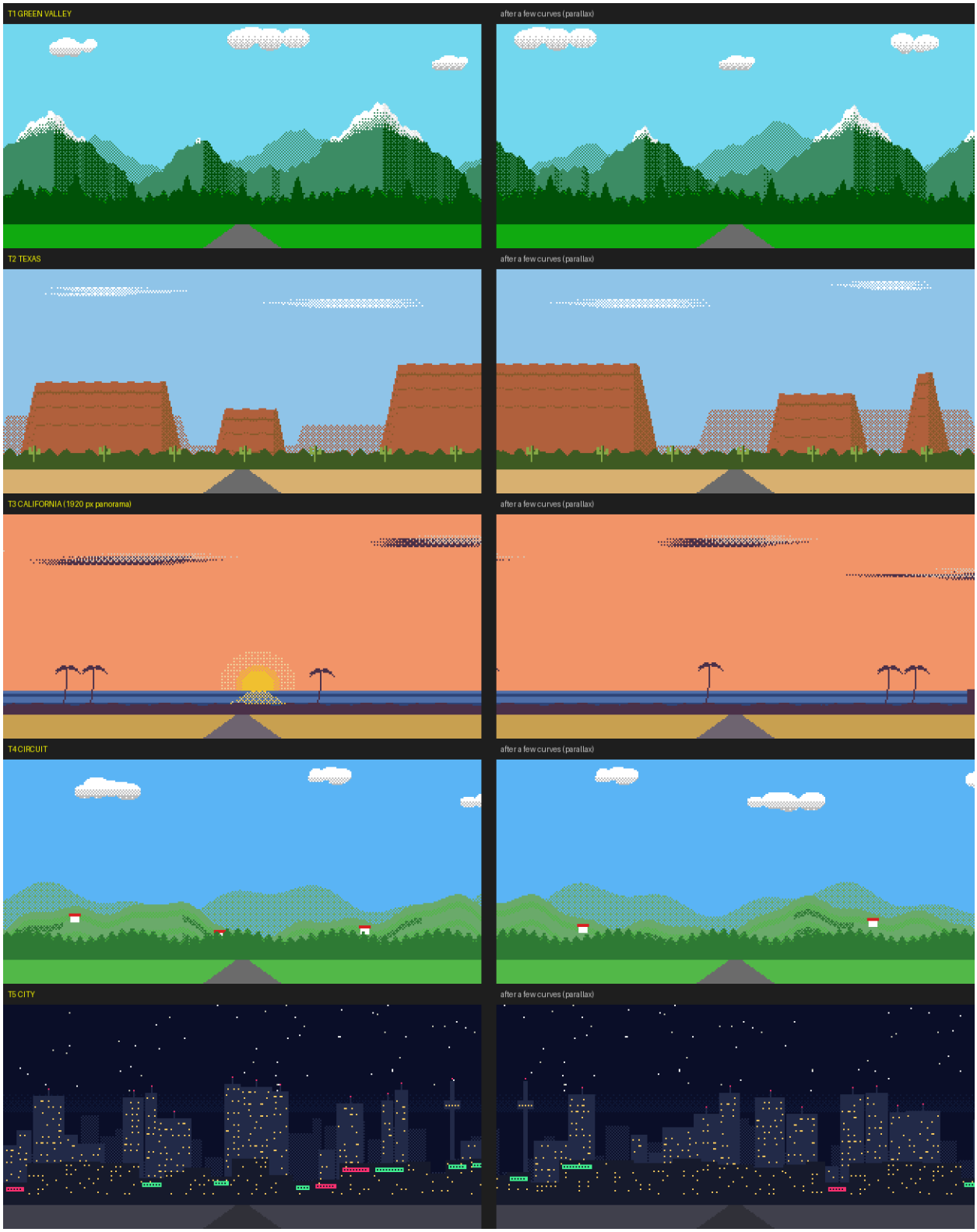

Fig. 32 – The backgrounds of the five tracks, left at the start, right after a few curves: clouds, mountains and foreground move different distances.

- **Layers:** up to five LAYER lines, back to front. Usually three: sky/clouds (y 0, speed 0.33), mountains (y 48, speed 0.67), foreground (y 95, speed 1). The horizon is at y 128; below it `BGFILL` fills the area.
- **Size:** any width from 320 pixels, the picture wraps around. Wider pictures repeat less often. The height is free.

- **Colours:** as with the sheet, the 16 standard colours, black is transparent. Dithering (a checkerboard of two colours) looks like an in-between colour from a distance – mixing with the sky colour gives haze.
- **Repeats:** the fourth value of LAYER says how often the picture passes per lap. 1 means a panorama: a landmark like the sun, the moon or a TV tower appears exactly once. Without it the game chooses the value to match the speed.

12.5 Music and effects

Every track plays its own 4-channel MOD (ProTracker format). The effects of the game sit **in the same MOD** as samples 20 to 27 and are laid over the music with `PLAY MODSAMPLE`. If you build new music, you must take over these eight samples:

Sample	Effect	Note
20	crash	short
21	grass / gravel	about 2 s, repeated
22	tyre squeal	about 3 s, repeated
23	shifting / countdown	short
24	rumble strip	about 2 s, repeated
25	pass-by	engine with Doppler
26	lap jingle	short
27	silence	ends a running effect

- **MUSICFX:** the same samples with silent music – the game switches to it when you turn the music off in the race (key M). `road1fx.mod` fits all tracks.
- **Limits:** `PLAY MODSAMPLE` knows no loops and no pitch; long noises are therefore restarted shortly before their end (chapter 9.3). The effects are built for 8287 Hz.
- **Size:** the MODs included are 120–160 KB.

12.6 Measuring the times

`TIME, start, cp1, cp2, line` are clean driving times in seconds: sector 1 from the standing start, then the sectors after checkpoint 1, after checkpoint 2 and after the finish line (sector 1 at full speed). A lap consists of three sectors of equal length. How to measure:

- In the menu press **O** for the options, MODE = PRACTICE: no traffic, no records.
- Drive two or three clean laps; the times in the result and the time left at the checkpoints show how much slack there is.
- Enter the averages. The game adds 30 % for ROOKIE and 15 % for VETERAN (+2 s at the first checkpoint, +1 s at the second).

Track	TIME (clean)	ROOKIE	VETERAN
Green Valley	32.5 · 26.6 · 26.9 · 27.9	42 · 37 · 36 · 36	37 · 33 · 32 · 32
Texas	28.3 · 27.2 · 27.3 · 25.3	37 · 37 · 36 · 33	33 · 33 · 32 · 29
California	25.2 · 21.1 · 21.3 · 21.4	33 · 29 · 29 · 28	29 · 26 · 25 · 25
Circuit	17.6 · 19.8 · 15.1 · 13.8	23 · 28 · 21 · 18	20 · 25 · 18 · 16
City	18.3 · 16.5 · 15.0 · 14.3	24 · 23 · 21 · 19	21 · 21 · 18 · 16

13 Join in!



Fig. 33 – Circuit: start into the figure of eight.

ROAD is meant as a base – for your own tracks, graphics and music. Everything that makes a track lies in open files, and this manual describes how they are built. Now it is your turn.

A call to the specialists
You are a pixel artist? Sprite sheets and background panoramas in 16 colours are exactly your field. Chapters 12.3 and 12.4 describe the format and the rules.

You write MOD music? A soundtrack of its own per track in ProTracker format, maybe even better effect samples, would lift ROAD enormously. Chapter 12.5 describes what the game expects from a MOD.

We would be delighted if you gave it a try!

13.1 Ideas for the community

- **New tracks:** desert at night, snow and ice, harbour, mountain pass, a track through your home town.
- **New sheets and backgrounds:** other seasons, weather, a sunrise.
- **Music:** one MOD per track in a fitting style.
- **Game ideas:** game controller, rear-view mirror, ghost car of the best lap, weather, two players.
- **Performance:** the road loop as a CSUB – towards 30 frames per second.

Questions, tracks and results are welcome in **The Back Shed forum**. The video of the game: `https://youtu.be/W4qjgm6pynE`

Have fun driving and building – have a good drive!

14 Appendix

14.1 Constants: original and study RACER.BAS

Name (original)	Original	RACER.BAS	Remark
width × height	1024 × 768	320 × 240	MODE 2
roadWidth	2000	ROADW = 2000	half road width
segmentLength	200	SEGLEN = 200	
rumbleLength	3	RUMBLE = 3	
lanes	3	fixed 3	line width 1/32, rumble 1/6
fieldOfView	100°	100°	CAMD ≈ 0.839
cameraHeight	1000	CAMH = 1000	
drawDistance	300	DD = 120	chapter 6
fogDensity	5	–	no fog in 16 colours
maxSpeed	segmentLength/step = 12000	MAXSPD = 12000	one segment per logic step
accel / breaking / decel	max/5 · −max · −max/5	2400 · −12000 · −2400	
offRoadDecel / offRoadLimit	−max/2 · max/4	−6000 · 3000	
centrifugal	0.3	CENTRI = 0.3	
sky/hill/treeSpeed	0.001 · 0.002 · 0.003	same	parallax factors
totalCars	200	NCAR = 40	chapter 5.5
resolution	height/480	RES = 0.5	

14.2 Commands used (PicoMite V7.00.00)

Command	Use in the game
MODE 2	320 × 240, 16 colours
MAP(n) = rgb, MAP SET, MAP(n)	set the palette, get the colour value of a slot
FRAMEBUFFER CREATE / CREATE 2 / LAYER / WRITE / COPY F, N	double buffering, sprite buffer, HUD layer
LOAD IMAGE	load the sprite sheet into buffer 2
POLYGON n%(), x%(), y%(), c%(), c%()	all road trapezoids of a frame (study)
BOX	road rows, grass, tree line
BLIT FRAMEBUFFER 2, F, …, 0	background strips with transparency
BLIT RESIZE 2, F, …, 0	scale sprites
TEXT	HUD
KEYDOWN(n)	several keys at the same time
TIMER	fixed time step
MATH SET	reset list heads
OPTION COMPILE ON, OPTION PROFILING ON	speed, measuring
MEMORY COPY INTEGER, PEEK(VARADDR …)	ROAD: copy chunks of the row rectangles

Command	Use in the game
FLASH LOAD IMAGE n, f$, O · BLIT FLASH n, F, …, 0	ROAD: background pictures in the RAM image slots 4–8 (PSRAM)
BLIT READ / BLIT WRITE …, 1	ROAD: mirrored sprite copies (MIRROR)
PLAY MODFILE, PLAY MODSAMPLE	ROAD: music and effects per track

14.3 Sources, licences and thanks

- Jake Gordon: “How to build a racing game”, parts 1–4 (straight, curves, hills, final), jakesgordon.com, 2012.
- Source code: github.com/jakesgordon/javascript-racer – MIT licence, copyright © 2012–2016 Jake Gordon and contributors. RACER.BAS is a port of this code to MMBasic and is under the same licence. Music and sprites of the original are not included.
- Louis Gorenfeld: “Lou’s Pseudo 3d Page”, extentofthejam.com/pseudo – origin of the curve accumulation and the Z map.
- Geoff Graham, Peter Mather et al.: PicoMite User Manual V7.00.00 and Game Development Guide, BLIT User Manual, MMBasic Performance Guide and Release Notes (Supplementary Manuals 7.00.00). Firmware: github.com/UKTailwind/PicoMite, release PicoMite-V7.00.00.
- Effect recordings (tyres, impact, engine): Kenney “Starter Kit Racing”, licence CC0 (skid sound by Landeplage, CC0). Music (MODs) and the other samples were made specially for this project.
- ROAD 1.0: code & design Matthias (homa) with Claude, 2026. Hardware: Pico Computer 3 by Peter Mather. MMBasic by Geoff Graham & Peter Mather. Thanks to The Back Shed forum and the community.