ROAD
Pseudo-3D-Racer in MMBasic

Handbuch: die Outrun-Technik Schritt für Schritt, das fertige Spiel und wie man es erweitert – für den Pico Computer 3 (RP2350) mit PicoMite V7.00.00



Teil I: die Technik · Teil II: ROAD 1.0 – Quellcode, Modding, Mitmachen – in 320 × 240 Pixeln und 16 frei belegbaren Farben (HDMI MODE 2).

Matthias (homa) mit Claude, 2026. Grundlage: die Artikelserie „How to build a racing game“ von Jake Gordon (2012, Quellcode unter MIT-Lizenz) und die Seite „Lou’s Pseudo 3d Page“. Texte, Diagramme, Pixelgrafik und Screenshots sind eigenständig erstellt. Stand: ROAD 1.0 RC1, November 2026.

Inhalt

Worum es geht **4**

1 Zielplattform: Pico Computer 3 mit PicoMite V7 **6**

1.1 Hardware und Firmware . . . 6

1.2 Welcher Bildmodus? . . . 6

1.3 Auflösung: von 1024 × 768 auf 320 × 240 . . . 6

1.4 Farben: 16 Slots, frei belegt . . . 7

1.5 Puffer: wohin wird gezeichnet? . . . 8

2 Teil 1: Die gerade Straße **8**

2.1 Projektion: ein Punkt, eine Division . . . 9

2.2 Die Straße als Kette von Segmenten . . . 11

2.3 Kamera und Spieler . . . 12

2.4 Spielschleife mit festem Zeitschritt . . . 13

2.5 Zeichnen: von vorne nach hinten . . . 13

3 Teil 2: Kurven **15**

3.1 Die Idee: Kurven werden nur vorgetäuscht . . . 15

3.2 Weiche Übergänge mit Easing . . . 15

3.3 Zeichnen mit zwei Akkumulatoren . . . 16

3.4 Fliehkraft . . . 17

3.5 Parallax-Hintergrund . . . 17

4 Teil 3: Hügel **18**

4.1 Eine Höhe pro Segment . . . 18

4.2 Die Kamera fährt mit . . . 18

4.3 Verdecken hinter Kuppen: maxy . . . 19

4.4 Hintergrund bewegt sich mit . . . 19

5 Teil 4: Das fertige Spiel **19**

5.1 Streckenbau mit Bausteinen . . . 20

5.2 Sprites: Sheet, Platzierung, Skalierung . . . 21

5.3 Clipping an der Hügelkante . . . 22

5.4 Reihenfolge: Straße vorwärts, Sprites rückwärts . . . 22

5.5 Gegner: Autos mit einfacher KI . . . 23

5.6 Kollisionen . . . 24

5.7 Spielerauto, HUD und Rundenzeiten . . . 25

6 Geschwindigkeit auf dem RP2350 **25**

6.1 Was pro Bild anfällt . . . 25

6.2 Regeln für schnelles MMBasic . . . 26

6.3 Stellschrauben im Spiel . . . 26

7 Die Alternative: Lous Raster-Straße **27**

8 Praxis: Optimierung auf dem Pico Computer 3 **27**

8.1 Messprotokoll 28

8.2 Die Lehren für MMBasic V7 auf dem RP2350 29

9 Ton: Erkenntnisse aus den Versuchen **31**

9.1 Die Audio-Systeme schließen sich gegenseitig aus 31

9.2 PLAY SOUND: Motor, Gänge, Doppler 32

9.3 MOD-Musik mit Effekt-Samples 34

9.4 Ausblick (Stand der Studie) 35

10 Vom Prototyp zum Spiel **37**

10.1 Was sich an der Technik geändert hat – und warum 38

11 Rundgang durch den Quellcode **39**

11.1 Die Hauptschleife 39

11.2 Der Zustandsautomat 40

11.3 Was in einem Segment steckt 41

11.4 Ein Bild entsteht 41

11.5 Hintergrund-Ebenen und der Kompass 41

11.6 Zeiten und Schwierigkeit 42

12 Modding: eigene Strecken, Grafik und Musik **42**

12.1 Eine neue Strecke anlegen 43

12.2 Die Track-Datei 43

12.3 Das Sprite-Sheet 44

12.4 Hintergrundbilder 44

12.5 Musik und Effekte 46

12.6 Zeiten einmessen 46

13 Mitmachen! **47**

13.1 Ideen für die Community 47

14 Anhang **48**

14.1 Konstanten: Original und Studie RACER.BAS 48

14.2 Verwendete Befehle (PicoMite V7.00.00) 48

14.3 Quellen, Lizenzen und Dank 49

Worum es geht

Ein „Pseudo-3D“-Rennspiel wie Outrun, Pole Position oder Lotus Turbo Challenge rechnet keine echte 3D-Welt. Die Straße besteht aus vielen kurzen, flachen Abschnitten (Segmenten), die nur in die Tiefe gestaffelt sind. Jedes Segment wird mit einer einzigen Division pro Eckpunkt auf den Bildschirm projiziert und als farbiges Trapez gezeichnet. Kurven entstehen durch geschicktes seitliches Verschieben, Hügel durch eine Höhe je Segment, und alles andere – Bäume, Schilder, Gegner – sind flache Bilder (Sprites), die mit der Entfernung skaliert werden. Genau diese Sparsamkeit macht die Technik ideal für einen Mikrocontroller, der in BASIC programmiert wird.

Jake Gordon hat die Technik 2012 in vier aufeinander aufbauenden Artikeln mit einem JavaScript-Spiel vorgeführt; Lou Gorenfeld hat auf seiner Seite die Tricks der Spielhallenautomaten gesammelt, aus denen sie stammt. Dieses Handbuch hat zwei Teile. **Teil I** folgt Jake Gordons Gliederung, erklärt die Technik mit den „alten“ flachen Sprites der ersten Machbarkeitsstudie `RACER.BAS` und überträgt sie Schritt für Schritt auf den Pico – dort, wo es für die Bildrate nötig ist, auch mit anderen Mitteln als im Original:

- **Teil 1 – Gerade Straße:** Projektion, Segmente, Kamera, Spielschleife.
- **Teil 2 – Kurven:** Kurvenwert je Segment, Übergänge (Easing), Fliehkraft, Parallax-Hintergrund.
- **Teil 3 – Hügel:** Höhe je Segment, Verdeckung hinter Kuppen.
- **Teil 4 – Finale:** Streckenbau, Sprites mit Clipping, computergesteuerte Autos, Kollisionen, HUD und Rundenzeiten.

Davor steht ein Kapitel zur Plattform (Bildmodus, Farben, Puffer), danach Kapitel zur Geschwindigkeit auf dem RP2350, zur Raster-Straße aus Lous Seite, zur Optimierung auf dem echten Gerät und zum Ton. **Teil II** zeigt, was daraus geworden ist: das Spiel **ROAD 1.0** mit fünf Strecken, World Tour und Bestenlisten – ein Rundgang durch den Quellcode, eine Anleitung zum Modden (Strecken, Grafik, Musik) und eine Einladung an die Community.

Woher kommen die Bilder?
Die Spielszenen in Teil I zeigen die Straße der Originalartikel nachgestellt in **320 × 240 Pixeln** und mit genau der **16-Farben-Palette** aus Kapitel 1 – so, wie sie auf dem HDMI-Ausgang des Pico Computer 3 aussieht. Die Diagramme sind neu gezeichnet und entsprechen inhaltlich denen der Originalartikel. Die Bilder in Teil II sind Aufnahmen vom echten Gerät.

Status des MMBasic-Codes
Die Code-Ausschnitte der Kapitel 1–7 stammen aus der ersten Fassung `RACER.BAS`. Sie lief auf dem Pico Computer 3 auf Anhieb. In 13 Schritten wurde sie danach auf dem echten Gerät gemessen und optimiert (Kapitel 8) und um Ton erweitert (Kapitel 9). Stand der Machbarkeitsstudie: **RACER_V13.BAS** mit **RACER.MOD** – rund 17–22 Bilder pro Sekunde, Musik und echte Effekt-Samples. Daraus entstand das Spiel ROAD (Teil II).

Zur Grafik: Das Originalspiel nutzt als Platzhalter Sprites aus der Sega-Genesis-Fassung von Outrun, die nicht frei verwendbar sind. Hier gibt es stattdessen ein eigenes, kleines Sprite-Sheet (`SPRITES.BMP`, 320 × 240), das direkt in den Bildspeicher des PicoMite passt.

Teil I

Die Technik

Pseudo-3D nach Jake Gordon und Lou Gorenfeld – übertragen auf MMBasic und den Pico Computer 3, gezeigt an der Machbarkeitsstudie RACER.BAS.

1 Zielplattform: Pico Computer 3 mit PicoMite V7

1.1 Hardware und Firmware

Der Pico Computer 3 hat einen RP2350B mit PSRAM und HDMI-Ausgang. Die passende Firmware heißt **PicoMitePC3**: Sie ist die HDMI-USB-Fassung (USB-Tastatur, Maus und Gamecontroller, alle HDMI-Auflösungen und Modi) plus WLAN und Bluetooth. Für uns zählen:

- **Takt:** Bei 640 × 480 kann der Takt mit `OPTION RESOLUTION 640x480, 378000` auf 378 MHz gesetzt werden (Standard 252 MHz). Alles läuft proportional schneller – das ist der billigste Leistungsgewinn.
- **Compiler:** V7 bringt auf dem RP2350 `OPTION COMPILE ON`. Das Programm bleibt unverändert, wird aber beim RUN in native Operationen übersetzt; einfache Rechenanweisungen werden laut Performance Guide 8- bis 14-mal schneller.
- **Speicher:** 256 KB Programmspeicher; Variablen und Arrays landen bei vorhandenem PSRAM größtenteils dort. Platz für die Streckendaten ist also reichlich.
- **Eingabe:** `KEYDOWN(n)` meldet bis zu sechs gleichzeitig gehaltene Tasten einer USB-Tastatur – wichtig für Gas + Lenken.

1.2 Welcher Bildmodus?

Bei HDMI 640 × 480 stehen fünf Modi zur Wahl. Für das Spiel wird **MODE 2** verwendet:

Modus	Auflösung	Farben	Bewertung für den Racer
MODE 1	640 × 480	2 (Kachelfarben)	ungeeignet
MODE 2	**320 × 240**	**16, frei per MAP auf RGB555**	**gewählt:** kleiner Framebuffer (38,4 KB), BLIT RESIZE, Layer L und T, MAP-Palette
MODE 3	640 × 480	16 per MAP	viermal so viele Pixel zu füllen – zu langsam für BASIC
MODE 4	320 × 240	32768 (RGB555)	schöne Farbverläufe, aber BLIT RESIZE arbeitet nur mit 4-Bit-Puffern
MODE 5	320 × 240	256 per MAP	Option für später (Nebel, mehr Sprite-Farben); BLIT RESIZE dann nicht nutzbar

Ausschlaggebend ist `BLIT RESIZE`: Der Befehl skaliert ein Rechteck aus einem Puffer per Nearest-Neighbour in ein Zielrechteck, wahlweise mit einer transparenten Farbe – exakt das, was ein Pseudo-3D-Racer für Bäume, Schilder und Autos braucht. Er arbeitet laut BLIT-Handbuch auf RGB121-gepackten 4-Bit-Puffern, also in MODE 2 und 3. Auch der Game Development Guide nennt MODE 2 den „Sweet Spot“ für Spiele.

1.3 Auflösung: von 1024 × 768 auf 320 × 240

Das Original rechnet mit einer logischen Leinwand von 1024 × 768. Beide Formate haben das Seitenverhältnis 4:3, und alle Projektionsformeln verwenden nur *width/2* und *height/2*. Es genügt also, Breite und Höhe einzusetzen – die Straße sieht identisch aus, nur gröber. Wo das Original einen Auflösungsfaktor *resolution = height/480* benutzt (Wackeln des Spielerautos, vertikaler Versatz des Hintergrunds), gilt hier **RES = 0,5**.

Eine echte Änderung betrifft die **Sichtweite**: Das Original zeichnet 300 Segmente. Bei 240 Bildzeilen schrumpfen ferne Segmente auf weniger als eine Zeile und werden ohnehin verworfen. Für den RP2350 empfehlen sich **100 bis 150 Segmente** (Kapitel 6 zeigt den Unterschied im

Bild).

1.4 Farben: 16 Slots, frei belegt

In MODE 2 hat jedes Pixel 4 Bit, also einen von 16 *Slots*. Welche Farbe ein Slot zeigt, bestimmt die Farbtabelle: `MAP(n) = rgb` setzt sie, `MAP SET` übernimmt alle Änderungen im nächsten Bildaustastintervall. Am HDMI-Ausgang wird jede Farbe auf den nächsten RGB555-Wert (5 Bit je Kanal) gerundet. Die Originalfarben von Jake Gordon wurden so auf 16 Slots verteilt:

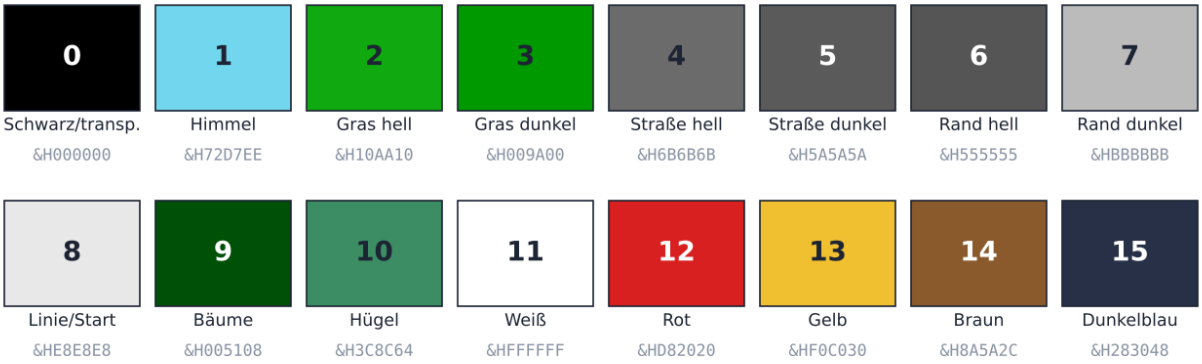


Abb. 1 – Palette des Spiels. Slot 0 ist zugleich die Transparenzfarbe für Sprites und das HUD-Layer.

- **Straße dunkel:** Das Original nutzt #6B6B6B und #696969. In RGB555 fallen beide auf denselben Wert (je 13 von 31) – der Hell/Dunkel-Wechsel wäre unsichtbar. Deshalb wurde Slot 5 auf #5A5A5A abgedunkelt.
- **Nebel:** Das Original blendet ferne Segmente halbtransparent in Waldgrün über. Ohne Alphakanal geht das in 16 Farben nicht. Der Nebel entfällt; die dunkle Waldkante am Horizont übernimmt optisch seine Aufgabe. (In MODE 5 könnte man Nebelstufen als eigene Paletteneinträge anlegen.)
- **Start/Ziel:** Startlinie nutzt den Mittelstreifen-Slot 8, die Ziellinie Slot 0 (schwarz).
- **Sprites:** Slots 9 bis 15 bedienen Bäume, Hügel, Wolken und die Autos.

Wichtig beim Zeichnen: Grafikbefehle erwarten eine 24-Bit-Farbe. Welche Farbe genau Slot *n* auswählt, liefert die Funktion `MAP(n)`. Das Programm liest diese Werte einmal in ein Array `col%()` und zeichnet nur noch damit:

RACER.BAS – Palette setzen (aus 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-Wert, der genau Slot i% waehlt
  NEXT
```

Sprite-Sheet als BMP vorbereiten
`LOAD IMAGE` ordnet jedes BMP-Pixel dem nächstgelegenen *Standard*farbwert zu, nicht der umdefinierten Palette. Die Datei `SPRITES.BMP` ist deshalb in den 16 Standardfarben gespeichert (Slot 1 = Blau, Slot 2 = Myrtle usw.). Im Spiel erscheinen die Pixel dann über MAP in den gewünschten Farben. Wer eigene Grafiken malt, arbeitet am besten indiziert und exportiert erst am Ende in die Standardfarben.

1.5 Puffer: wohin wird gezeichnet?

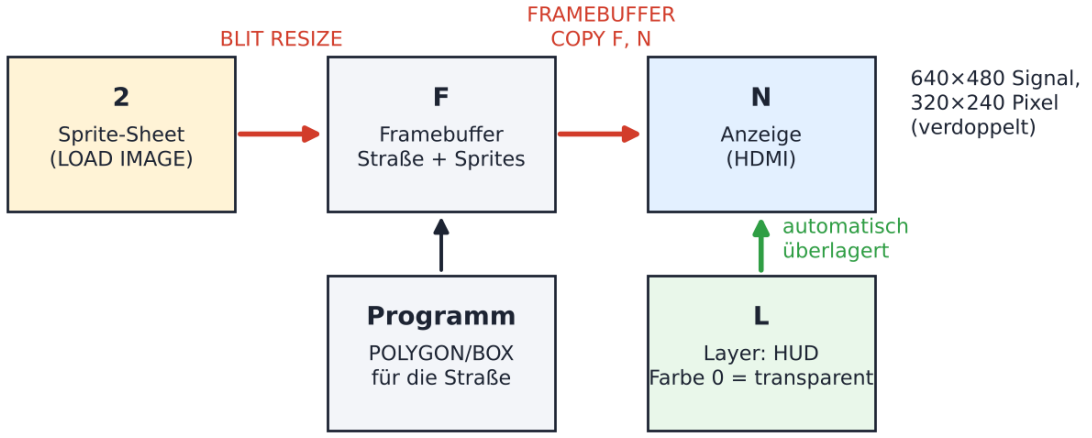


Abb. 2 – Pufferarchitektur. Gezeichnet wird immer in F; die Anzeige N bekommt das fertige Bild in einem Rutsch.

- **F (Framebuffer):** Alle Zeichenbefehle gehen hierhin. Am Ende jedes Bildes kopiert `FRAMEBUFFER COPY F, N` den Inhalt flimmerfrei auf den Bildschirm. Der Game Guide rät, die Hintergrundvariante mit drittem Parameter in der Spielschleife zu meiden.
- **L (Layer):** Wird bei HDMI in MODE 2 automatisch über das Bild gelegt; Slot 0 ist durchsichtig. Ideal für das HUD: Es muss nur neu geschrieben werden, wenn sich Geschwindigkeit oder Zeit ändern.
- **2 (zweiter Framebuffer):** Hält das Sprite-Sheet und die drei Hintergrundstreifen. Er ist unsichtbar und dient nur als Quelle für `BLIT RESIZE` und `BLIT FRAMEBUFFER`. Da L den vorab reservierten Videospeicher belegt, wird 2 aus dem Variablenspeicher genommen (beim PC3 eventuell PSRAM – das sollte man bei den Messungen im Blick behalten).

RACER.BAS – Puffer anlegen (aus SUB Init)

```
FRAMEBUFFER CREATE                ' F: hier wird gezeichnet
FRAMEBUFFER LAYER 0               ' L: HUD, Farbe 0 ist durchsichtig
FRAMEBUFFER CREATE 2              ' 2: Sprite-Sheet + Hintergrundstreifen
FRAMEBUFFER WRITE L : CLS col%(C_BLACK)
FRAMEBUFFER WRITE 2 : LOAD IMAGE "sprites.bmp"
FRAMEBUFFER WRITE F
```

2 Teil 1: Die gerade Straße

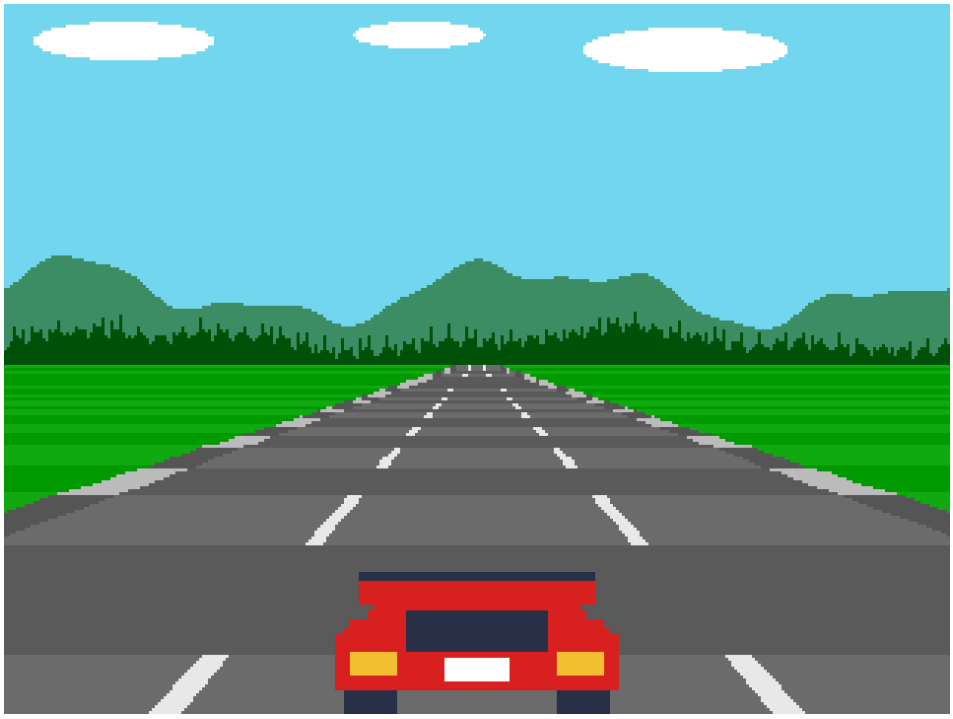


Abb. 3 – Ergebnis von Teil 1 in 320 × 240 und 16 Farben.

2.1 Projektion: ein Punkt, eine Division

Alles beruht auf ähnlichen Dreiecken. Die Kamera schwebt in der Höhe *h* über der Straße; im Abstand *d* vor ihr steht die gedachte Bildebene. Ein Straßenpunkt in der Entfernung *z* erscheint auf der Bildebene in der Höhe $y = h \cdot d / z$ unter dem Horizont. Dasselbe gilt seitlich: Ein Punkt mit dem seitlichen Abstand *w* landet bei $x = w \cdot d / z$. Der Faktor **d / z** ist der Maßstab, mit dem später auch Sprites skaliert werden.

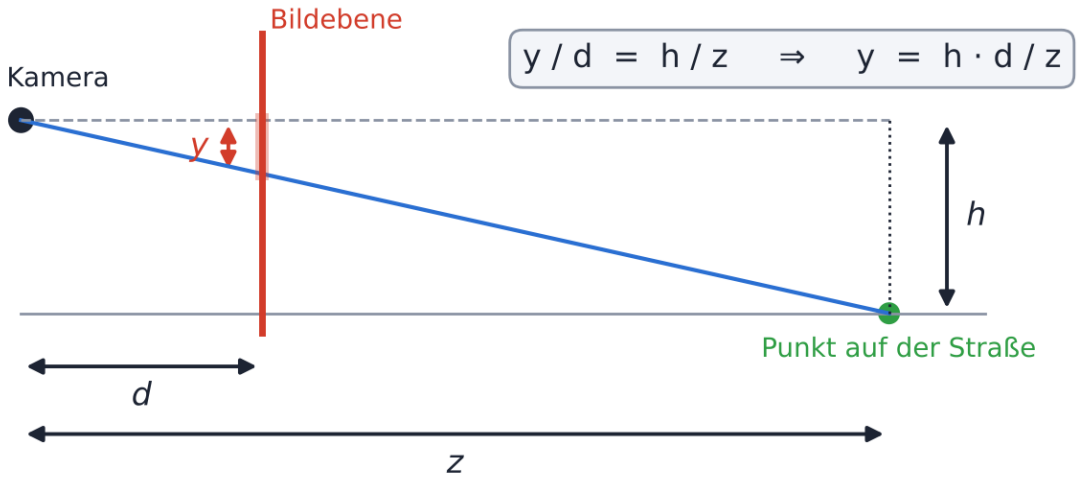


Abb. 4 – Ähnliche Dreiecke: Je weiter weg ein Punkt liegt, desto näher rückt er an den Horizont.

Damit die Formel für beliebige Punkte gilt, wird jeder Punkt in drei Schritten umgerechnet:

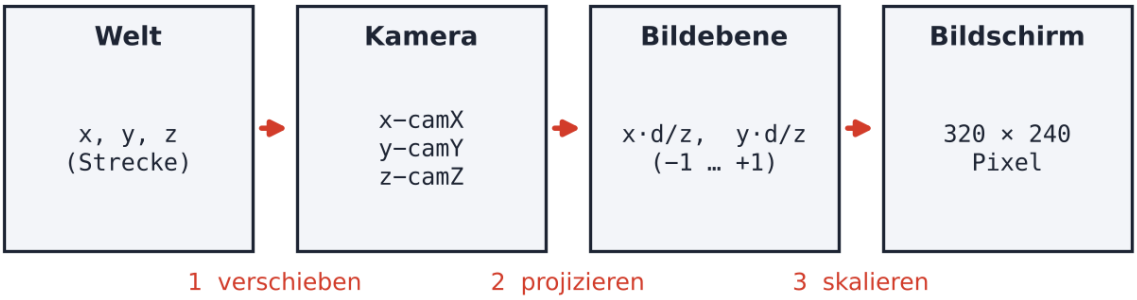


Abb. 5 – Von der Welt- zur Bildschirmkoordinate.

- **Verschieben:** Kameraposition abziehen. Die Kamera sitzt danach im Ursprung.
- **Projizieren:** x und y mit *d/z* multiplizieren. Ergebnis: Werte zwischen −1 und +1 auf der Bildebene.
- **Skalieren:** auf Pixel umrechnen. Weil die y-Achse am Bildschirm nach unten zeigt, wird y gespiegelt.

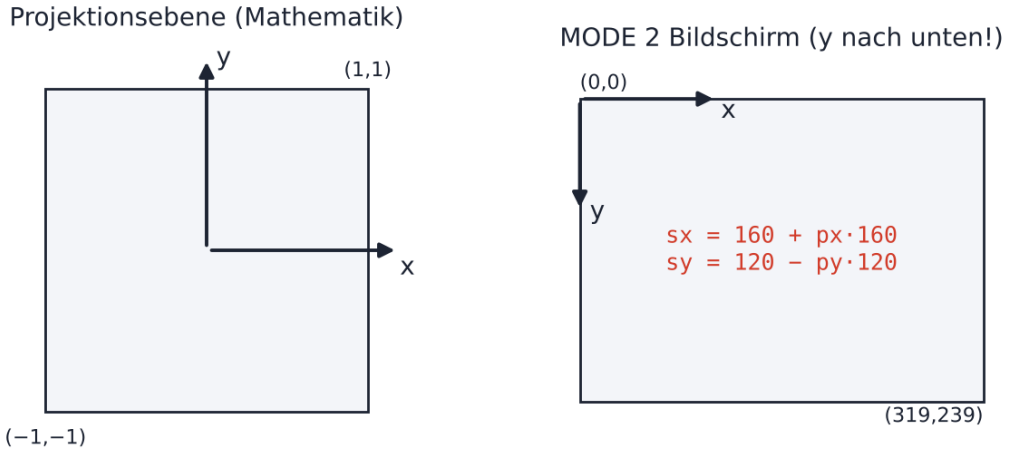


Abb. 6 – Mathematisches und Bildschirm-Koordinatensystem in MODE 2.

Der Abstand *d* folgt aus dem vertikalen Sichtwinkel: Bei einer Bildebene von −1 bis +1 gilt **d = 1 / tan(fov/2)**. Mit 100° wie im Original ergibt sich d ≈ 0,839. Größere Winkel wirken weitwinkliger und schneller, kleinere „teleartig“.

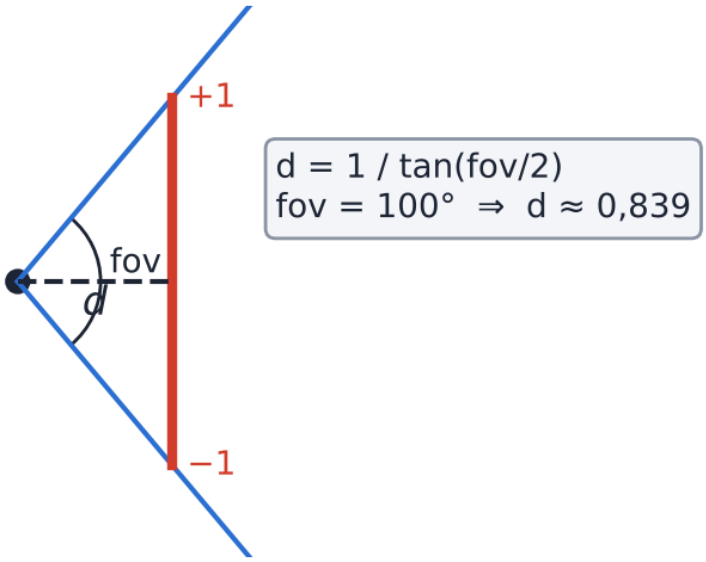


Abb. 7 – Sichtwinkel und Abstand zur Bildebene.

Zusammengefasst für einen Punkt mit Kamera-Koordinaten (cx, cy, cz):

Projektion in MMBasic-Schreibweise

```
scale = CAMD / cz
sx    = HW + scale * cx * HW          ' HW = 160  (halbe Breite)
sy    = HH - scale * cy * HH          ' HH = 120  (halbe Höhe)
sw    = scale * ROADW * HW            ' halbe Straßenbreite in Pixeln
```

2.2 Die Straße als Kette von Segmenten

Die Strecke ist eine lange Liste gleich langer Segmente (*segmentLength* = 200 Einheiten). Jedes Segment hat einen nahen Mittelpunkt **p1** und einen fernen **p2**; p2 eines Segments ist p1 des nächsten. Die halbe Straßenbreite *roadWidth* = 2000 macht die Rechnung symmetrisch (die Straße reicht von −2000 bis +2000). Alle drei Segmente (*rumbleLength*) wechselt die Farbgruppe zwischen hell und dunkel – das erzeugt die typischen Randstreifen und, viel wichtiger, das Gefühl von Geschwindigkeit. Wäre jedes Segment einzeln eingefärbt, würde das Bild flimmern.

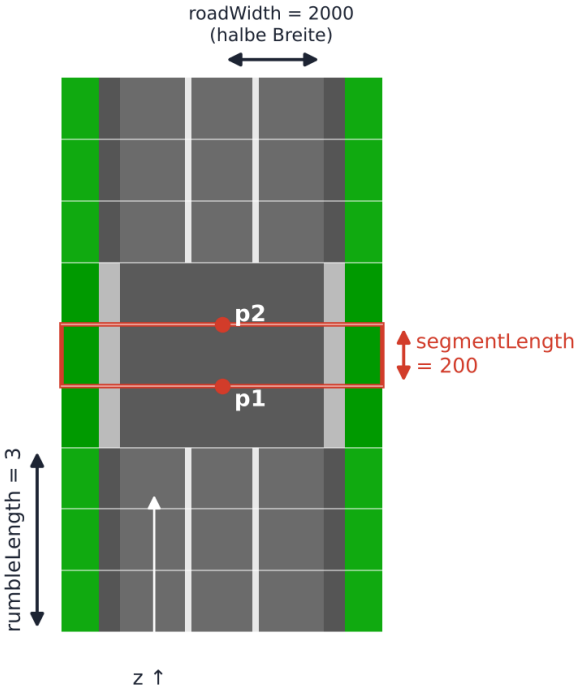


Abb. 8 – Segmente in der Draufsicht (z zeigt in Fahrtrichtung nach oben), mit Rasenfläche, Randstreifen, Fahrbahn und Fahrspurlinien.

MMBasic kennt zwar Strukturen (TYPE), der Compiler übersetzt Strukturzugriffe aber nicht. Für die heiße Schleife eignen sich darum **parallele Arrays**: ein Array je Eigenschaft, der Index ist die Segmentnummer. Von jedem Segment wird nur gespeichert, was sich nicht ausrechnen lässt:

Streckendaten (RACER.BAS)

```
DIM segY(MAXSEG), segCv(MAXSEG), segCol%(MAXSEG), sprHd%(MAXSEG)
DIM nSeg%, trackLen
' segY    Höhe am fernen Ende (p2); p1 ist segY() des Vorgängers
' segCv   Kurvenwert (Teil 2)
' segCol% 0 = hell, 1 = dunkel, 2 = Start, 3 = Ziel
' sprHd%  erstes Sprite dieses Segments (Teil 4), -1 = keins
```

Die z-Koordinaten werden gar nicht gespeichert. Weil das Segment *n* Plätze vor dem Kamerasegment liegt, ist sein Abstand zur Kamera einfach **(n − bp) · 200**, wobei *bp* der schon zurückgelegte Bruchteil des Kamerasegments ist. Das spart Speicher und löst nebenbei das Problem, dass das Original Segmente hinter dem Streckenende („looped“) gesondert behandeln muss.

2.3 Kamera und Spieler

- **position** ist die z-Position der Kamera auf der Strecke; sie läuft modulo Streckenlänge.
- **playerZ** ist der feste Abstand des Autos vor der Kamera: *cameraHeight* · *cameraDepth* ≈ 839. Genau dort hat ein Sprite den Maßstab, bei dem es am unteren Bildrand steht.
- **playerX** ist die seitliche Lage, normiert auf −1 (linker Rand) bis +1 (rechter Rand). Bei |playerX| > 1 fährt man auf dem Gras. Für die Projektion wird die Kamera um *playerX* · *roadWidth* verschoben – nicht das Auto: Das Auto bleibt in der Bildmitte, die Straße wandert.

2.4 Spielschleife mit festem Zeitschritt

Das Original trennt Logik und Darstellung: Die Logik rechnet immer in festen Schritten von 1/60 s, gezeichnet wird so oft es geht. Schafft der Pico zum Beispiel 25 Bilder pro Sekunde, laufen pro Bild zwei bis drei Logikschritte. So bleibt das Fahrgefühl unabhängig von der Bildrate, und die Höchstgeschwindigkeit von *segmentLength / step* garantiert, dass pro Schritt höchstens ein Segment übersprungen wird (wichtig für die Kollisionen).

RACER.BAS – Hauptschleife

```
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 – Tastatur (Codes 128–131 = Pfeiltasten laut Anhang des Handbuchs)

```
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       ' links  / a
    IF k% = 131 OR k% = 100 THEN kR% = 1      ' rechts / d
    IF k% = 128 OR k% = 119 THEN kF% = 1      ' hoch   / w
    IF k% = 129 OR k% = 115 THEN kS% = 1      ' runter / s
    IF k% = 27 THEN kQ% = 1                   ' ESC
  NEXT
END SUB
```

Die Physik ist bewusst einfach: Gas beschleunigt mit *maxSpeed/5* pro Sekunde, Bremsen verzögert mit *maxSpeed*, ohne Pedal rollt das Auto mit *maxSpeed/5* aus. Auf dem Gras bremst es zusätzlich mit *maxSpeed/2*, aber nie unter ein Viertel der Höchstgeschwindigkeit. Lenken versetzt playerX pro Sekunde um höchstens 2 – bei Vollgas kommt man in einer Sekunde von einem Rand zum anderen.

2.5 Zeichnen: von vorne nach hinten

Die Straße wird vom Auto weg gezeichnet. Für jedes Segment werden p1 und p2 projiziert. Es wird übersprungen, wenn p1 hinter der Kamera liegt, wenn es „von hinten“ zu sehen ist (p2 liegt tiefer als p1) oder wenn es komplett hinter bereits Gezeichnetem verschwindet (dazu mehr in Teil 3). Für jedes sichtbare Segment entstehen ein Rasenstreifen über die volle Breite und vier bis sechs Trapeze: zwei Randstreifen, die Fahrbahn und bei hellen Segmenten zwei Fahrspurlinien. Die Randstreifen sind 1/6 der halben Straßenbreite breit, die Linien 1/32.

Der wichtigste MMBasic-Trick: Polygone sammeln
Ein BASIC-Befehl kostet einige Mikrosekunden allein fürs Interpretieren. Statt 150 bis 300 einzelner Zeichenbefehle pro Bild sammelt `Quad` alle Trapeze in Arrays, und **ein einziges** `POLYGON n%(), x%(), y%(), c%(), c%()` zeichnet sie am Ende in Firmware-Geschwindigkeit. Eine 0 im Zähler-Array markiert das Ende der Liste. Die Reihenfolge bleibt erhalten, das Ergebnis ist identisch.

RACER.BAS – Trapez an die Liste anhängen

```
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 – Straßenschleife (aus SUB Render; in Teil 1 ist segCv() = 0 und segY() = 0)

```
  ' --- Strasse: vorne nach hinten, Polygone sammeln
  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     ' Kamera-z ohne Schleifenproblem
    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                                  ' Ende-Markierung
  IF np% > 0 THEN POLYGON polyN%(), polyX%(), polyY%(), polyC%(), polyC%()
```

3 Teil 2: Kurven

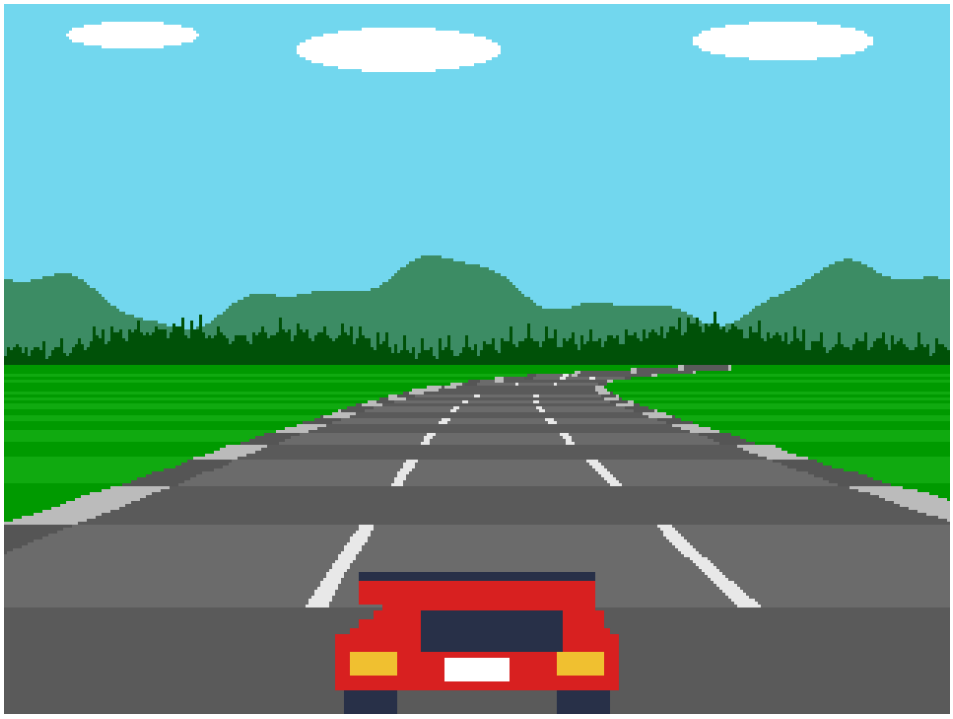


Abb. 9 – Teil 2: Eine Rechtskurve (320 × 240).

3.1 Die Idee: Kurven werden nur vorgetäuscht

Eine echte Kurve bräuchte x-Koordinaten für jedes Segment und eine gedrehte Kamera. Das Spiel macht es sich einfacher: Jedes Segment bekommt nur einen **Kurvenwert** – negativ für links, positiv für rechts, je größer, desto enger. Beim Zeichnen wird die Mittellinie der Straße von Segment zu Segment immer stärker seitlich verschoben. Die Welt selbst bleibt gerade; nur das Bild biegt sich. Diese Idee stammt von Lous Seite und funktioniert verblüffend gut.

Konstante	Werte	Bedeutung
Länge	kurz 25 · mittel 50 · lang 100	Segmente pro Phase
Kurve	leicht 2 · mittel 4 · scharf 6	Kurvenwert je Segment (nach Gefühl gewählt)
Hügel	niedrig 20 · mittel 40 · hoch 60	Höhenänderung in Segmentlängen (Teil 3)

3.2 Weiche Übergänge mit Easing

Ein abrupter Wechsel von 0 auf 4 sähe aus wie ein Knick. Jeder Streckenabschnitt hat daher drei Phasen: **Einfahrt** (Kurvenwert steigt mit easeIn), **Halten** (konstant) und **Ausfahrt** (fällt mit easeInOut auf 0). Die Hilfsfunktionen interpolieren zwischen a und b; easeIn quadratisch, easeInOut mit einer halben Kosinuswelle.

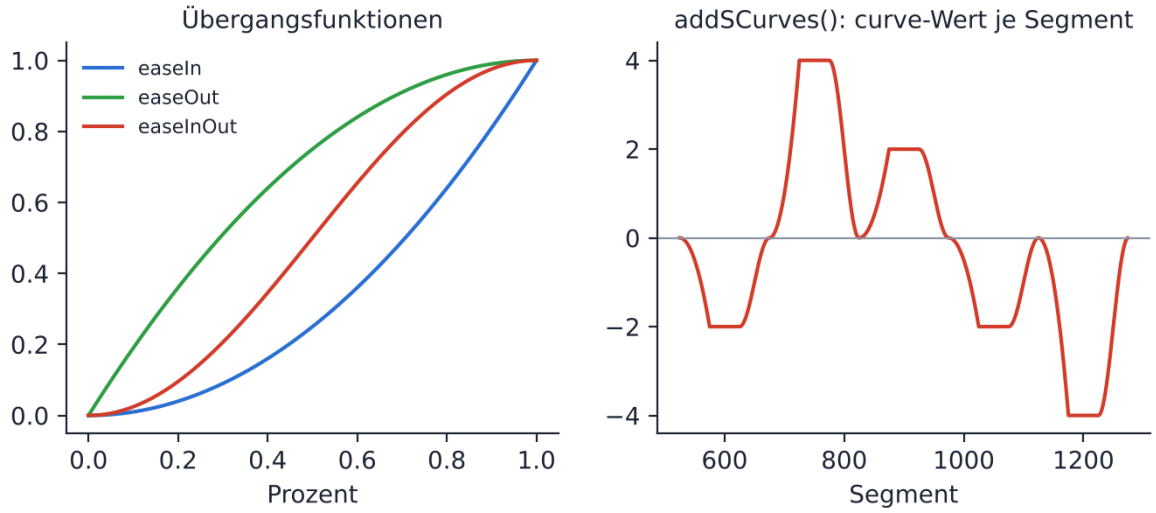


Abb. 10 – Links die Übergangsfunktionen, rechts der Kurvenverlauf von addSCurves(): fünf Abschnitte mit Ein-, Halte- und Ausfahrtphase.

RACER.BAS – Easing und der universelle Streckenbaustein

```
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 Zeichnen mit zwei Akkumulatoren

Beim Zeichnen laufen zwei Zahlen mit: **x** ist der seitliche Versatz der Mittellinie, **dx** seine Änderung pro Segment. Nach jedem Segment gilt $x = x + dx$ und $dx = dx + curve$. Weil dx selbst wächst, wächst x quadratisch – die Straße biegt sich zum Horizont hin immer stärker. p1 wird mit Versatz x projiziert, p2 mit x + dx.

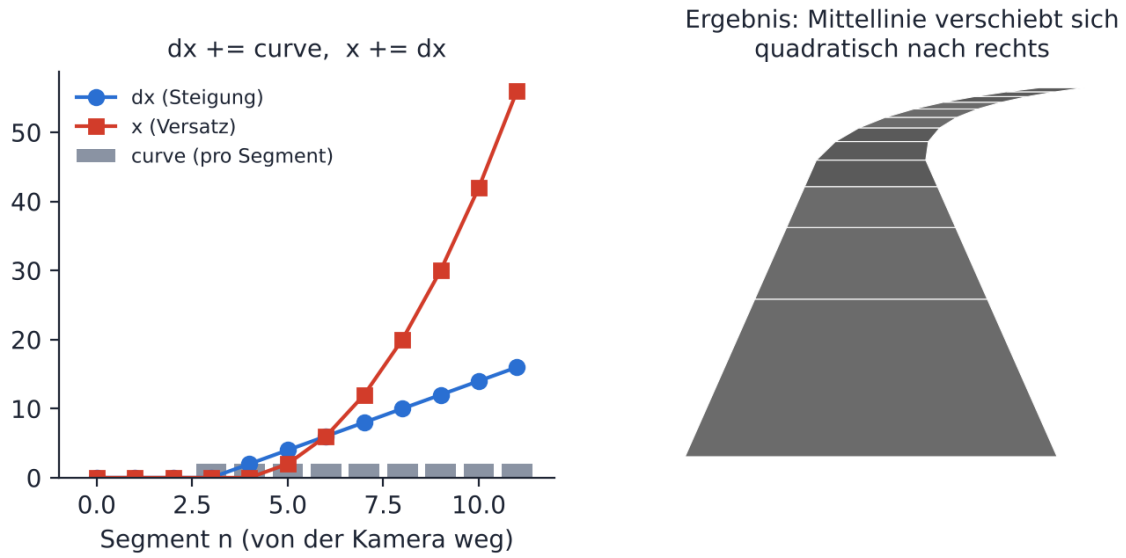


Abb. 11 – Akkumulation: Ab Segment 3 beginnt eine Kurve (curve = 2). dx steigt linear, x quadratisch.

Ein Detail verhindert Ruckeln: Steht die Kamera schon zu einem Bruchteil *bp* im aktuellen Segment, startet dx mit **−curve · bp**. Ohne diese Korrektur würde die Straße bei jedem Segmentwechsel ein Stück springen. Lou beschreibt denselben Kniff als „Scroll Fix“.

Die Kurven-Änderungen in der Straßenschleife

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

3.4 Fliehkraft

Damit Kurven sich auch so anfühlen, schiebt die Physik das Auto nach außen – proportional zu Geschwindigkeit und Kurvenwert, mit dem Faktor 0,3. Wer bei Vollgas nicht gegenlenkt, landet auf dem Gras.

```
playerX = playerX - dx * sp * segCv(ps%) * CENTRI      ' Fliehkraft (aus SUB Update)
```

3.5 Parallax-Hintergrund

Der Hintergrund besteht aus drei Ebenen – Wolken, Hügelkette, Baumreihe –, die sich in Kurven unterschiedlich schnell seitlich verschieben (Faktoren 0,001 / 0,002 / 0,003 je Segment und Kurveneinheit). Das Original malt dafür große Bilder. Für den Pico liegen die Ebenen als **320 Pixel breite, nahtlos kachelbare Streifen** im unteren Teil von Puffer 2. Pro Bild genügen dann höchstens sechs BLIT FRAMEBUFFER-Aufrufe mit Transparenzfarbe 0 – jeder Streifen wird an der Versatzstelle in zwei Teile geschnitten.

RACER.BAS – einen Hintergrundstreifen mit Versatz kopieren

```
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 Teil 3: Hügel

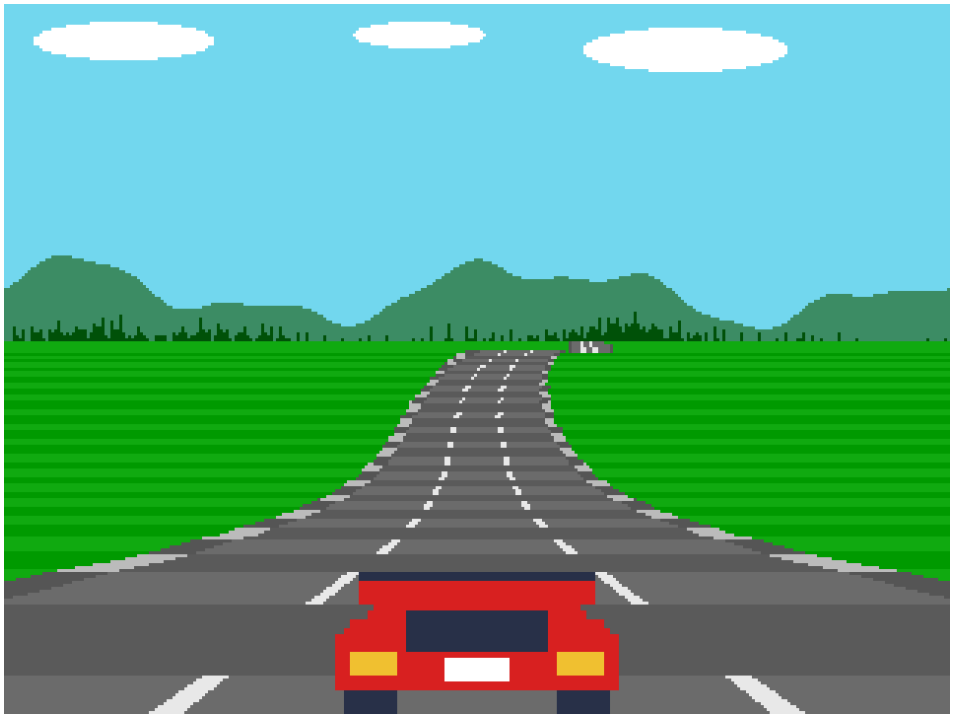


Abb. 12 – Teil 3: Kuppe mit Kurve dahinter.

4.1 Eine Höhe pro Segment

Für Hügel ist erstaunlich wenig zu tun, weil die Projektion y schon immer berücksichtigt hat – bisher war es nur überall 0. Jedes Segment speichert die Höhe seines fernen Endes; das nahe Ende übernimmt die Höhe des Vorgängers, damit keine Stufen entstehen. Beim Streckenbau wird die Höhe über den ganzen Abschnitt (Einfahrt + Halten + Ausfahrt) mit easeInOut von der Start- zur Zielhöhe geführt. Die Höhe wird in Segmentlängen angegeben: „40“ heißt 40 · 200 Einheiten höher.

RACER.BAS – Segment anhängen

```
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                    ' Hoehe am Ende (p2); p1 = Vorgaenger
  segCol%(nSeg%) = (nSeg% \ RUMBLE) MOD 2
  sprHd%(nSeg%) = -1
  INC nSeg%
END SUB
```

4.2 Die Kamera fährt mit

Die Kamera muss der Straße folgen, sonst tauchte sie in jeden Hügel ein. Dazu wird die Höhe der Straße unter dem Auto bestimmt – interpoliert zwischen p1 und p2 des Spielersegments – und die Kamera *cameraHeight* darüber gesetzt:

aus SUB Render

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

4.3 Verdecken hinter Kuppen: maxy

Hinter einer Kuppe liegende Straßenstücke dürfen nicht gezeichnet werden. Weil von vorne nach hinten gezeichnet wird, genügt eine Zahl: **maxy**, die oberste bisher gezeichnete Bildschirmzeile. Ein Segment, dessen fernes Ende nicht über maxy hinausragt, ist verdeckt und wird übersprungen. Zusätzlich werden Segmente verworfen, die man von hinten sähe (p2 tiefer als p1), was beim Bergabfahren hinter einer Kuppe vorkommt.

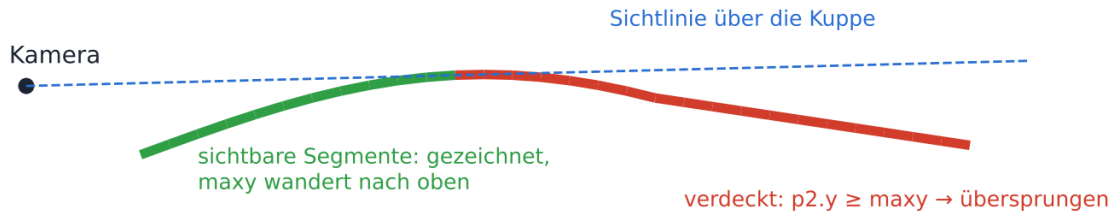


Abb. 13 – Grün: sichtbar, maxy wandert nach oben. Rot: liegt unter der Sichtlinie über die Kuppe und wird übersprungen.

Das spart nebenbei viel Arbeit: In den Auszählungen werden von 120 durchlaufenen Segmenten im Mittel nur rund 40 tatsächlich gezeichnet. Der Rasenstreifen wird in RACER.BAS zusätzlich auf maxy begrenzt, damit nichts doppelt gemalt wird.

4.4 Hintergrund bewegt sich mit

Fährt man bergauf, sinkt der Horizont. Jede Hintergrundebene wird deshalb zusätzlich um *Geschwindigkeitsfaktor · playerY · RES* nach unten verschoben (siehe `Strip`). Das ignoriert die Steigung des aktuellen Segments, wirkt aber überzeugend.

5 Teil 4: Das fertige Spiel



Abb. 14 – Kurve mit Säulenreihe, Bäumen und Laster.

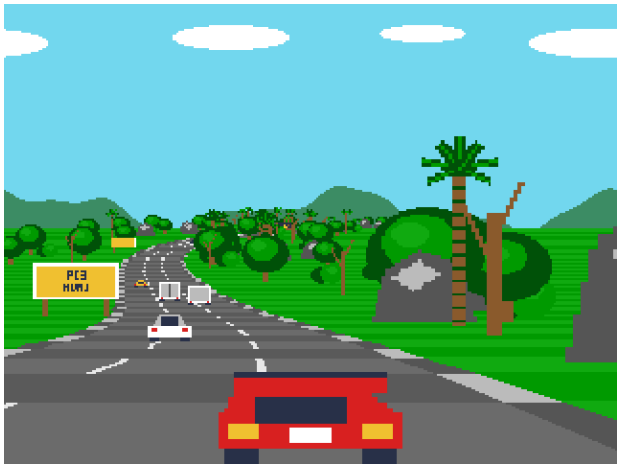


Abb. 15 – Palmenabschnitt mit Werbetafeln und Gegnern.

5.1 Streckenbau mit Bausteinen

Alle Abschnitte entstehen aus dem einen Baustein `AddRoad(einfahrt, halten, ausfahrt, kurve, höhe)`. Darauf setzen kleine Helfer auf:

Helfer (Original)	macht	MMBasic-Aufruf
addStraight(n)	gerade, flach	AddRoad n, n, n, 0, 0
addCurve(n, k, h)	Kurve, optional mit Höhenänderung	AddRoad n, n, n, k, h
addHill(n, h)	Hügel ohne Kurve	AddRoad n, n, n, 0, h
addLowRollingHills(n, h)	sechs sanfte Wellen, zwei leichte Kurven	AddLowRollingHills n, h
addSCurves()	fünf Kurven im Wechsel, mit Höhe	AddSCurves
addBumps()	acht kurze Bodenwellen	AddBumps
addDownhillToEnd(n)	führt am Ende auf Höhe 0 zurück (geschlossene Runde)	AddRoad n, n, n, −2, −LastY()/SEGLEN

RACER.BAS – die komplette Originalstrecke

```
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       ' Startlinie
  FOR n% = 1 TO RUMBLE
    segCol%(nSeg% - n%) = 3                       ' Ziellinie
  NEXT
END SUB
```

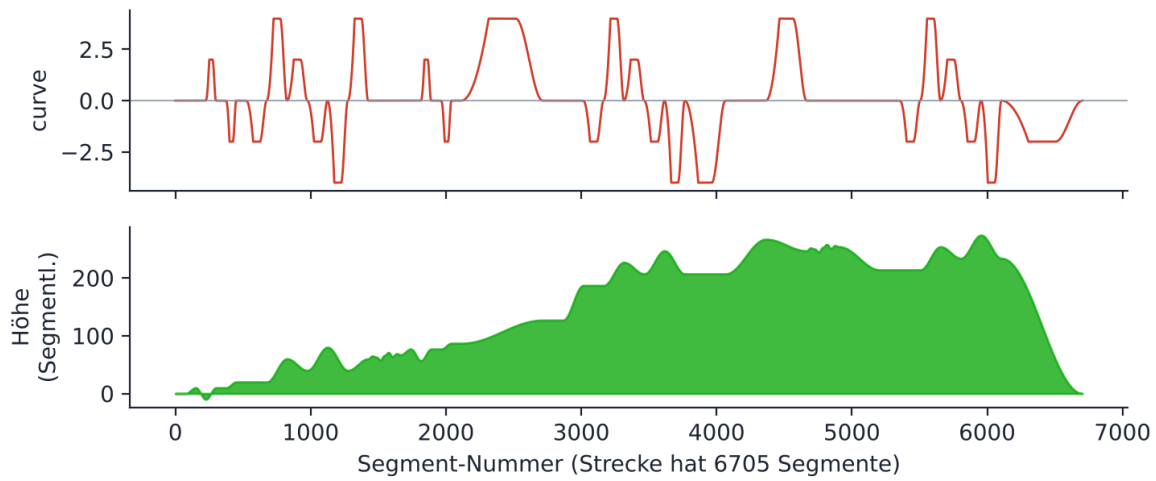


Abb. 16 – Kurvenwert und Höhenprofil der Strecke. Am Ende führt eine lange Abfahrt auf die Starthöhe zurück.

Die Strecke hat rund 6700 Segmente. Mit vier Arrays à 8 Byte sind das gut 210 KB – auf dem PC3 dank PSRAM kein Problem. Wer Speicher oder Ladezeit sparen will, baut eine kürzere Runde oder speichert Kurve und Höhe als 32-Bit-Werte.

5.2 Sprites: Sheet, Platzierung, Skalierung



Abb. 17 – SPRITES.BMP (320 × 240, hier vergrößert). Oben die Objekte, darunter die drei Hintergrundstreifen. Schwarz = transparent.

Jeder Sprite-*Typ* ist ein Rechteck im Sheet plus eine **Weltbreite** in Vielfachen von roadWidth. Das Original leitet die Weltgröße aus den Pixelmaßen seiner großen Bilder ab; weil unser Sheet klein ist, steht die Weltbreite direkt in der Tabelle. Ein Baum ist so etwa 1,35-mal so breit wie die halbe Straße, ein Auto 0,3-mal.

Am Straßenrand platzierte Objekte hängen an einem Segment, mit einem seitlichen **Offset** (−1 = linker Rand, +1 = rechter Rand, Werte darüber hinaus stehen im Gelände). Weil MMBasic keine dynamischen Listen kennt, liegen alle Objekte in einem Pool; jedes Segment zeigt auf sein erstes Objekt, jedes Objekt auf das nächste (verkettete Liste mit −1 als Ende):

RACER.BAS – Objekt an ein Segment hängen

```
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
```

Beim Zeichnen bekommt ein Sprite den Maßstab seines Segments. Breite und Höhe in Pixeln ergeben sich aus Weltbreite × Maßstab; der Ankerpunkt (unten mittig für Autos, unten außen für Randobjekte) verschiebt das Rechteck. `BLIT  RESIZE` erledigt das Skalieren samt Transparenz in Firmware.

5.3 Clipping an der Hügelkante

Steht ein Baum hinter einer Kuppe, ist sein Fuß verdeckt, seine Krone aber sichtbar. Darum merkt sich jedes Segment beim Straßendurchlauf den damaligen Wert von maxy als **clip**. Beim Zeichnen des Sprites wird alles unterhalb dieser Linie abgeschnitten – im Sheet wie im Ziel wird dieselbe anteilige Höhe weggelassen.

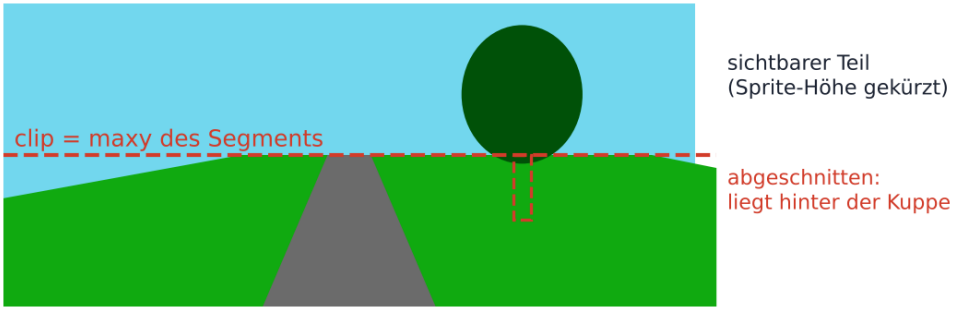


Abb. 18 – Der Baum steht hinter der Kuppe; nur der Teil oberhalb der clip-Linie wird gezeichnet.

RACER.BAS – Sprite skalieren, clippen und zeichnen

```
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 Reihenfolge: Straße vorwärts, Sprites rückwärts

Die Straße wird von vorne nach hinten gezeichnet (wegen maxy). Sprites dagegen müssen von hinten nach vorne gemalt werden, damit nahe Objekte ferne überdecken (Maleralgorithmus). Deshalb gibt es zwei Durchläufe: erst die Straße, dabei werden pro Segment die projizierten Werte und clip gespeichert; dann eine Schleife vom fernsten zum nächsten Segment, die Autos, Randobjekte und – im Segment des Spielers – das eigene Auto zeichnet.

RACER.BAS – Sprite-Durchlauf (aus SUB Render)

```
  ' --- Sprites: hinten nach vorne (Maleralgorithmus)
  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                              ' Spielerauto
      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 Gegner: Autos mit einfacher KI

Im Original fahren 200 Autos, und jedes Segment führt eine Liste seiner Autos, die beim Segmentwechsel umgehängt wird. Für MMBasic ist ein schlankeres Verfahren besser: **40 Autos** mit konstanter Geschwindigkeit (Laster langsamer), und vor jeder Verwendung werden sie in einem Durchlauf in Listen je Segment *relativ zum Spieler* einsortiert – nur für das Fenster von 20 Segmenten hinter bis DD Segmenten vor dem Spieler. Umhängen entfällt.

RACER.BAS – Autos einsortieren

```
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
```

Die Ausweich-KI schaut 20 Segmente voraus. Findet ein Auto dort ein langsameres Fahrzeug (oder den Spieler) auf seiner Spur, lenkt es weg – umso stärker, je näher das Hindernis und je größer der Geschwindigkeitsunterschied ist. Liegt das Hindernis schon am Rand, weicht es zur Mitte aus. Autos außerhalb der Sichtweite lenken gar nicht; dort darf es Überlappungen geben, die niemand sieht.

RACER.BAS – Ausweichlenkung

```
FUNCTION CarSteer(i%, r%)
  LOCAL j%, k%, q%, cw, dir
  CarSteer = 0
  IF r% > DD THEN EXIT FUNCTION              ' ausser Sicht: nicht lenken
  cw = tWF(carT%(i%))
  FOR j% = 1 TO 19
    q% = r% + j%
    IF q% >= nSeg% THEN q% = q% - nSeg%
    IF q% = 0 THEN                           ' Spieler-Segment voraus
      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 Kollisionen

- **Randobjekte:** Nur wenn das Auto neben der Straße fährt, werden die Objekte des Spielersegments geprüft. Bei Überlappung wird es auf ein Fünftel der Höchstgeschwindigkeit gebremst und an den Anfang des Segments gesetzt – es „rutscht“ am Hindernis vorbei, statt hart zu stoppen.
- **Autos:** Ist der Spieler schneller als ein überlappendes Auto in seinem Segment, verliert er Tempo und landet direkt dahinter.
- Die Überlappung ist ein einfacher 1D-Test auf der Querachse; der Prozentparameter macht die Trefferbox schmaler (0,8) oder breiter (1,2).

RACER.BAS – Überlappungstest

```
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 Spielerauto, HUD und Rundenzeiten

Das eigene Auto hat drei Bilder (gerade, links, rechts). Ein kleines, zufälliges Wackeln proportional zur Geschwindigkeit simuliert die Federung. Das Original hat zusätzlich Bergauf-Bilder; sie lassen sich ergänzen, sobald das Sheet Platz dafür hat.

Das HUD liegt im Layer L und wird – wie im Original – nur neu geschrieben, wenn sich der angezeigte Wert ändert. Eine Runde ist beendet, wenn die Kamera die Startposition überquert; die beste Zeit wird gespeichert (das Original legt sie im Browser ab, hier könnte sie per VAR SAVE die Ausschaltung überleben).

RACER.BAS – HUD auf dem 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 Geschwindigkeit auf dem RP2350

6.1 Was pro Bild anfällt

Für eine volle Runde mit 40 Gegnern wurde jedes sechste Bild ausgezählt. Die Zahlen sind unabhängig von der Hardware und zeigen, wie viel Arbeit auf den PicoMite zukommt:

Sichtweite (Segmente)	gezeichnete Segmente	Trapeze (POLYGON)	Rasen-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)

Auffällig: Die Zahl der gezeichneten Straßensegmente hängt kaum von der Sichtweite ab – ferne Segmente sind kleiner als eine Zeile und fallen weg. Die Sichtweite bestimmt vor allem, wie oft die Projektionsschleife läuft (zwei Projektionen pro Segment) und wie viele Sprites skaliert werden. Optisch fehlt bei 120 Segmenten nur das letzte Stück Straße am Horizont:

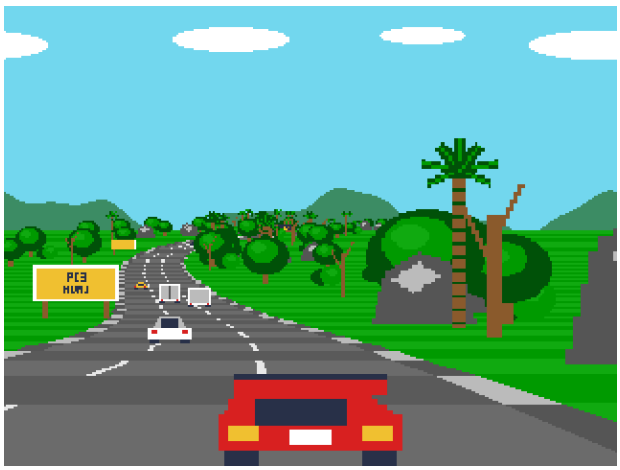


Abb. 19 – Sichtweite 300 (Original).

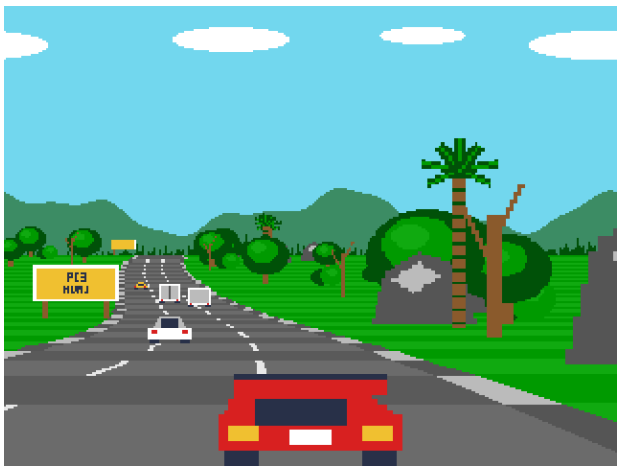


Abb. 20 – Sichtweite 120 (RACER.BAS). Die Straße endet etwas früher.

6.2 Regeln für schnelles MMBasic

Der MMBasic Performance Guide (V7) misst jede Konstruktion auf dem RP2350. Für den Racer folgen daraus:

- **Kompilieren:** `OPTION COMPILE ON`. Zuweisungen, IF, FOR, DO, SUB-Aufrufe sowie BOX, LINE, TRIANGLE und die einfachen BLIT-Formen werden übersetzt. POLYGON, MATH und BLIT RESIZE laufen interpretiert weiter – sie sind aber je ein einzelner Aufruf, der intern in C arbeitet.
- **Firmware schleifen lassen:** Array-Formen von POLYGON, BOX, LINE und PIXEL sind 20- bis 100-mal schneller als BASIC-Schleifen. Darum das Sammeln der Trapeze.
- **Ganzzahlige Koordinaten:** Gleitkommavariablen als Grafikargumente kosten bei jedem Aufruf eine Umwandlung. Koordinaten in %-Variablen halten.
- **Weniger Anweisungen:** Jede Anweisung kostet einige Mikrosekunden, bevor sie etwas tut. Konstante Teile vor die Schleife ziehen, SUB-Aufrufe in der innersten Schleife (z. B. Quad) später notfalls von Hand einsetzen.
- **FOR statt DO:** Ein leerer FOR-Durchlauf kostet laut Guide 2,3 µs, eine DO-Schleife 5 bis 11 µs (kompiliert 1,1 gegenüber 2,2 µs).
- **Messen statt raten:** `OPTION PROFILING ON` als erste Zeile zeigt die teuersten Zeilen; `PRINT MM.INFO(COMPILE)` zeigt, welcher Anteil kompiliert lief.
- **Letzte Reserve:** Das Werkzeug mmb2csub übersetzt einzelne SUBs in Maschinencode (CSUB) – Schleifen und Arithmetik laut Guide 10- bis 35-mal schneller. Erster Kandidat wäre die Straßenschleife in `Render`.

6.3 Stellschrauben im Spiel

Stellschraube	Wirkung	in RACER.BAS
Sichtweite DD	weniger Projektionen und Sprites	CONST DD = 120
Anzahl Gegner	KI-Schleife und Sprites	CONST NCAR = 40
Pflanzendichte	Sprites pro Bild (größter Posten)	ResetSprites: STEP 3 → z. B. STEP 6
Logikschritt	60 Hz Logik kostet bei 20 fps drei Updates pro Bild	CONST STP (dann MAXSPD anpassen)
Takt	378 statt 252 MHz: +50 % für alles	OPTION RESOLUTION 640x480, 378000
Fahrspurlinien	zwei Trapeze weniger je hellem Segment	Linien nur bis n% < 40 zeichnen

7 Die Alternative: Lous Raster-Straße

Lous Seite beschreibt neben dem Segment-Verfahren die noch ältere **Raster-Technik** der Spielhallenautomaten. Statt Segmente zu projizieren, wird *jede Bildschirmzeile* unterhalb des Horizonts einzeln behandelt. Für jede Zeile wird einmal vorab berechnet, wie weit sie entfernt ist – die **Z-Map**:

Prinzip der Z-Map

```
' Z-Map für eine flache Straße (einmal beim Start)
FOR y% = 1 TO 119                         ' Zeilen unter dem Horizont
  zmap(y%) = CAMH / y%                    ' Entfernung dieser Zeile
  scl(y%)  = y% / 119                     ' Maßstab relativ zur untersten Zeile
NEXT
```

- **Streifen:** Ob eine Zeile hell oder dunkel ist, folgt aus *(zmap(y) + position)* modulo der Streifenlänge.
- **Kurven:** Von unten nach oben werden ein Versatz und seine Änderung aufaddiert (ddx += dx, x += ddx) – dasselbe Prinzip wie in Teil 2, nur pro Zeile.
- **Hügel:** werden durch Überspringen oder Wiederholen von Zeilen vorgetäuscht; das ist fummeliger als die echte Höhe der Segmente.
- **Zeichnen:** Jede Zeile besteht aus höchstens fünf waagerechten Stücken (Gras, Rand, Straße, Rand, Gras). Mit der Array-Form von BOX lassen sich alle 120 Zeilen in **einem** Befehl zeichnen.

Auf dem PicoMite ist das reizvoll, weil die Arbeit pro Bild fest ist (120 Zeilen, keine Divisionen zur Laufzeit) und die BOX-Arrays nativ gezeichnet werden. Nachteil: Echte Hügel mit Verdeckung und an Segmente gebundene Sprites sind im Segment-Verfahren deutlich einfacher. **Gewählt wurde deshalb das Segment-Verfahren** – gezeichnet wird aber wie bei der Raster-Straße Zeile für Zeile mit B0X-Arrays (Kapitel 8). Diese Mischung verbindet echte Hügel und Sprites mit schnellem Füllen und ist die Grundlage von ROAD (Teil II).

8 Praxis: Optimierung auf dem Pico Computer 3

Die erste Fassung lief auf dem PC3 auf Anhieb – mit knapp 10 Bildern pro Sekunde. In einer Messreihe von 13 Versionen wurde sie mit dem eingebauten Profiler (`OPTION PROFILING ON`) und eigenen `TIMER`-Messpunkten Schritt für Schritt beschleunigt. Jede Änderung wurde einzeln gemessen. Dieses Kapitel fasst zusammen, was wirklich etwas gebracht hat – und was nicht.

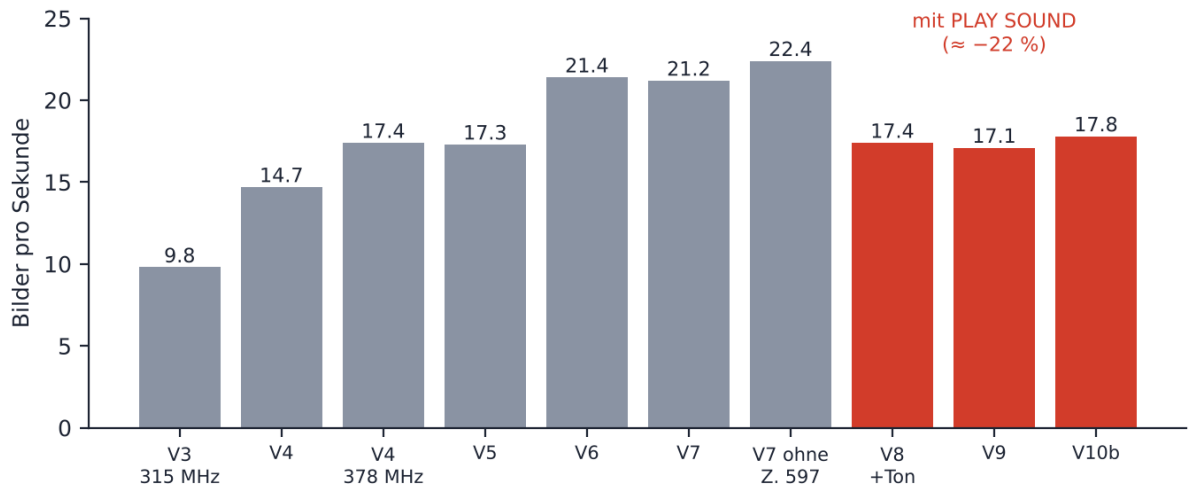


Abb. 21 – Bildrate je Version, gemessen auf dem PC3 über eine volle Runde. Rot: mit Ton über PLAY SOUND (Kapitel 9).

8.1 Messprotokoll

Version	Änderung	FPS	Erkenntnis
V1/V3	erste Fassung, 175 km/h, Takt 315 MHz	9,8	Logik läuft pro Bild fast 9-mal
V4	Auto-KI und Einsortieren 1× pro Bild statt pro Logikschritt; SegOf und Quad eingespart	14,7	Logik von ~40 auf ~6 ms
V4	Takt 378 MHz (`OPTION RESOLUTION 640x480, 378000`)	17,4	+18 %, ohne Codeänderung
V5	Puffer 2 vor dem Layer anlegen (schneller Videospeicher), HUD direkt in F	17,3	Hintergrund −30 %, Sprites unverändert
V6	Straße zeilenweise mit einem BOX-Array statt POLYGON; ferne/winzige Sprites weglassen	21,4	Füllen von ~20 auf 2 ms
V7 + Z. 597	einzeiliges `IF … THEN EXIT FOR` entfernt	22,4	lief als einzige Zeile interpretiert
V8–V10	Ton mit PLAY SOUND (Motor, Gänge, Geräusche, Doppler)	17–18	Tonerzeugung kostet ~22 %
V12/V13	MOD-Musik + Effekt-Samples, Taste M wechselt zum Motor-Modus	–	siehe Kapitel 9

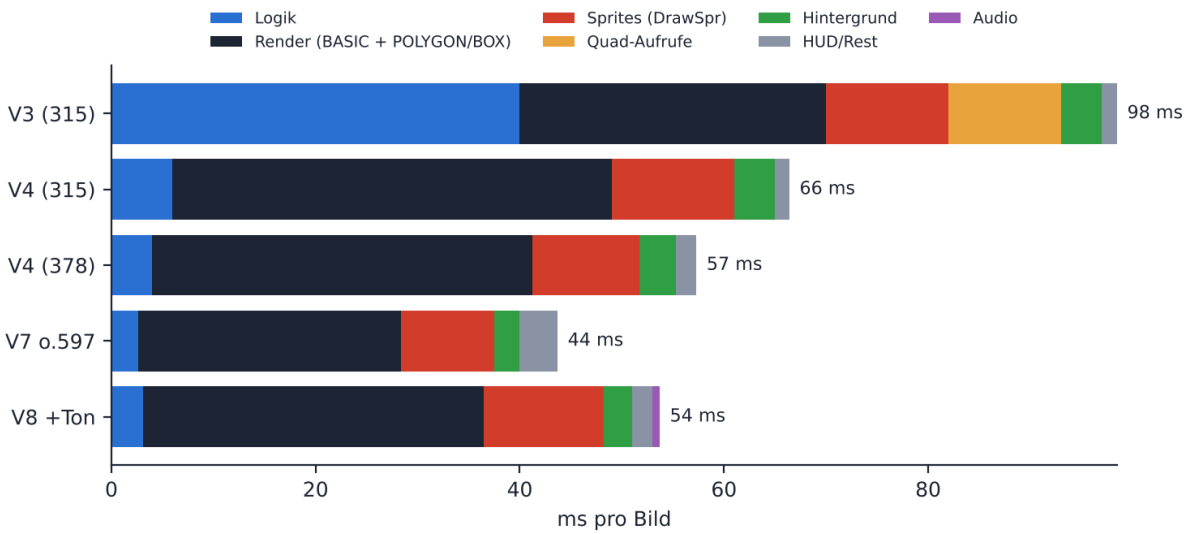


Abb. 22 – Wohin die Zeit pro Bild geht (aus den Profiler-Berichten). Die Logik schrumpft drastisch, danach dominiert das Zeichnen.

8.2 Die Lehren für MMBasic V7 auf dem RP2350

1. Der Compiler läuft – aber Aufrufe bleiben teuer

`PRINT MM.INFO(COMPILE)` meldete durchgehend über 99 % kompilierte Anweisungen. Trotzdem kostete eine Funktion mit einer einzigen Rechenzeile (`Seg0f`) rund 30 µs pro Aufruf – bei Hunderttausenden Aufrufen 13 ms pro Bild. Der Aufruf selbst ist der Posten, nicht die Rechnung. **Regel: Kleine Helfer in heißen Schleifen direkt ausschreiben.** Eine CSUB hilft hier nicht, sie wird genauso aufgerufen.

2. Fester Zeitschritt: teure Logik vervielfacht sich

Bei 10 Bildern pro Sekunde laufen pro Bild 9 Logikschritte. Alles, was im Schritt steckt, zählt neunfach – und bremst das Zeichnen weiter, das wiederum mehr Schritte erzeugt. In V3 fraß die Logik so 40 % der Zeit. **Regel: Im Logikschritt nur das, was die Physik wirklich braucht (bewegen, Kollision); KI, Einsortieren und Ton einmal pro Bild.** Die Lenkung der Gegner wird dafür mit der Zahl der Schritte des Bildes multipliziert.

3. POLYGON ist bequem, aber langsam beim Füllen

Die Messpunkte in Render zeigten: Das eine `POLYGON`-Array mit allen Trapezen kostete 17–22 ms (grob 0,7 µs pro Pixel). Dieselbe Fläche als waagerechte 1-Pixel-Rechtecke in einem `BOX`-Array: 2 ms. Dafür steigt der BASIC-Anteil, weil pro Bildzeile gerechnet wird.

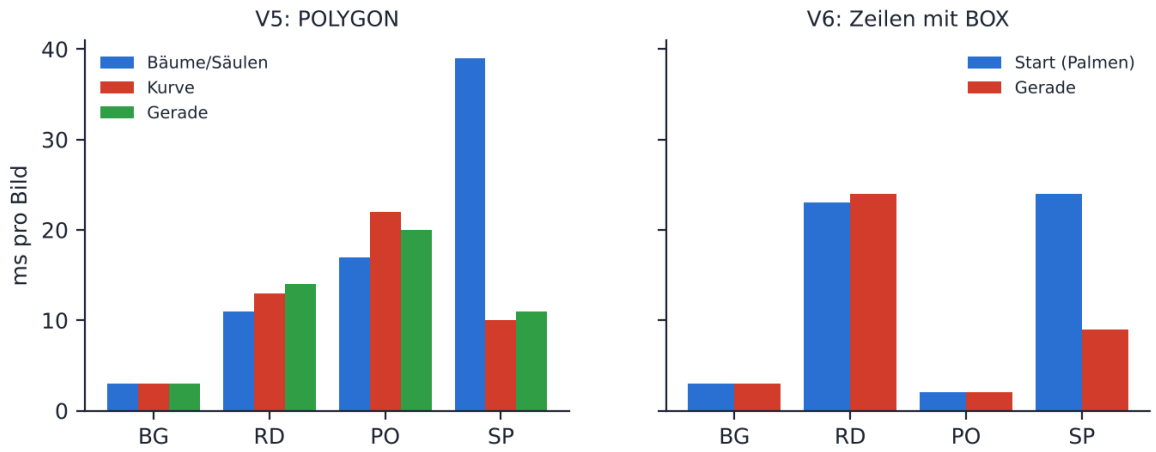


Abb. 23 – Messpunkte in Render (ms pro Bild): Hintergrund (BG), Straßenschleife (RD), Füllen (PO), Sprites (SP).

Zwei Stolpersteine beim BOX-Array
Breite wird geprüft: Ein Rechteck breiter als 320 Pixel bricht mit „Width … is invalid (valid is 1 to 320)“ ab – anders als POLYGON schneidet BOX nicht selbst zu. Jedes Stück muss vorher auf 0…319 begrenzt werden. **Keine Ende-Markierung:** BOX zeichnet immer das ganze Array. Unbenutzte Einträge werden vor jedem Bild per `MATH SET` auf ein unsichtbares 1-Pixel-Rechteck in Himmelfarbe gesetzt.

RACER_V13.BAS – Straße zeilenweise (Auszug aus Render)

```
        ' Zeilenweise: fuer jede Bildzeile Randstreifen, Fahrbahn, Linien
        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 verlangt 1 <= Breite <= 320: jedes Stueck auf den Bildschirm beschneiden
          a% = xc - ww - rr : e% = xc + ww + rr + 1          ' Randstreifen (aussen bis aussen)
          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                    ' Fahrbahn
          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                         ' Fahrspurlinie
              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. Eine einzige interpretierte Zeile kann 5 ms kosten

Der Profiler zählt unter „top commands by dispatch count“ die Befehle, die *nicht* kompiliert liefen. Dort tauchte plötzlich `If` mit 222 000 Aufrufen auf. Ursache war eine harmlose

Sicherheitszeile `IF nb% + LANES + 1 > MAXB THEN EXIT FOR` in der Zeilenschleife: Ein einzeiliges IF mit EXIT FOR übersetzt der Compiler offenbar nicht. Einzeilige IFs mit Zuweisungen oder EXIT SUB liefen dagegen kompiliert. **Regel: Nach jeder Änderung einen Blick auf die Dispatch-Liste werfen.**

5. Speicherort der Puffer zählt

Wird zuerst der Layer L angelegt, belegt er den schnellen, vorab reservierten Videospeicher, und Puffer 2 landet im Variablenspeicher – beim PC3 im PSRAM. Umgekehrt (Puffer 2 zuerst, kein Layer) wurde das Kopieren der Hintergrundstreifen 30 % schneller, und es profitierte vom höheren Takt kaum – ein Zeichen für eine Speicherbremse. `BLIT RESIZE` für die Sprites blieb dagegen bei 230–300 µs pro Aufruf: Das sind feste Kosten pro Befehl. Hebel ist dort nur die *Anzahl* der Sprites (Grenzen `SCENDD` und `SPRMIN`).

6. Kleinigkeiten mit Wirkung

- **Transparenter Text:** Bei HDMI erlaubt TEXT als Hintergrundfarbe −1 – das HUD braucht dann keinen Layer.
- **HUD-Texte zwischenspeichern:** Zeichenketten nur neu bauen, wenn sich der Wert ändert; gezeichnet wird trotzdem jedes Bild.
- **Takt:** 378 MHz statt 315 MHz (75-Hz-Modus) bringt rund 18 %.
- **Spielbalance:** Fliehkraft 0,45 bescherte 220 Zusammenstöße pro Runde, 0,38 nur noch 15. Häufige Crashs kosten auch Bildrate.

9 Ton: Erkenntnisse aus den Versuchen

Ein Rennspiel ohne Motorgeräusch fühlt sich leer an – das akustische Feedback fehlt. Der PicoMite bietet mehrere Wege zum Ton. Die Versuche haben gezeigt, was jeder davon kann, was er kostet und wo die Grenzen liegen.

9.1 Die Audio-Systeme schließen sich gegenseitig aus

System	Kann	Grenze im Spiel
`PLAY SOUND`	4 Dauertöne (Sinus, Rechteck, Dreieck, Sägezahn, Rauschen, eigene Wellenform U), Frequenz frei	kostet ~22 % CPU; Geräusche klingen synthetisch
`PLAY BBC SOUND`	4 Kanäle mit Notenwarteschlange und Hüllkurven	gut für Jingles; als Motor nicht erprobt
`PLAY MODFILE` + `PLAY MODSAMPLE`	Musik auf 4 Kanälen + 4 Effektkanäle mit echten Samples	Effekte ohne Schleife und ohne Tonhöhe
`PLAY SAMPLE`	Wellentabelle mit ADSR-Hüllkurve	startet bei jedem Aufruf die Hüllkurve neu
`PLAY MIDIFILE` / `PLAY MIDI`	MIDI-Dateien im Hintergrund; Echtzeit-MIDI über einen VS1053-Codec	nicht geprüft

Laut Handbuch blockiert `PLAY SOUND` jede andere Audioausgabe bis `PLAY STOP`. **Motor per PLAY SOUND und MOD-Musik gleichzeitig sind daher nicht möglich.** RACER_V13 löst das mit zwei Modi, Taste M wechselt.

9.2 PLAY SOUND: Motor, Gänge, Doppler

- **Kosten:** Sobald PLAY SOUND läuft, sinkt die Bildrate um ~22 % (22,4 → 17,4), gleichmäßig über alle Programmteile – unabhängig davon, ob 2 oder 4 Töne aktiv sind. Die eigene Ton-Logik kostet dagegen nur ~1 ms pro Bild.
- **Eigene Wellenform (Typ U):** `PLAY LOAD SOUND` lädt 4096 Werte. Die Frequenz gibt an, wie oft die *ganze* Tabelle pro Sekunde abgespielt wird. Mit 16 Motorzyklen in der Tabelle ist die Tonhöhe also Frequenz × 16 – in der Aufnahme exakt bestätigt (29 Hz Leerlauf, 150 Hz Vollgas).
- **Gänge:** 5 Gänge mit Drehzahlsprung beim Schalten machen den Motor lebendig; Flattern (±1–2 %) und Schiebebetrieb ohne Gas gegen Monotonie.
- **Doppler:** Der nächste Gegner bekommt einen eigenen Ton, höher beim Annähern, tiefer nach dem Überholen, links/rechts je nach Seite. Der Effekt war auf Anhieb überzeugend.

Warum der erste Motor „metallisch“ klang

Die Aufnahme vom PC3 zeigte: Die Tonhöhe stimmte, aber der 2. bis 8. Oberton war fast so laut wie der Grundton, im Leerlauf sogar der 5. Oberton der lauteste. Ursache war ein zu hohes Nachschwingen in der Wellenform (3,5- und 7-faches der Grundfrequenz). V10 schwingt mit dem 1,5- und 3-fachen nach, jeder zweite Zündzyklus ist schwächer („Blubbern“), und die Tabelle wird beim Start zweimal geglättet – ein „Auspuff“ als Tiefpass.

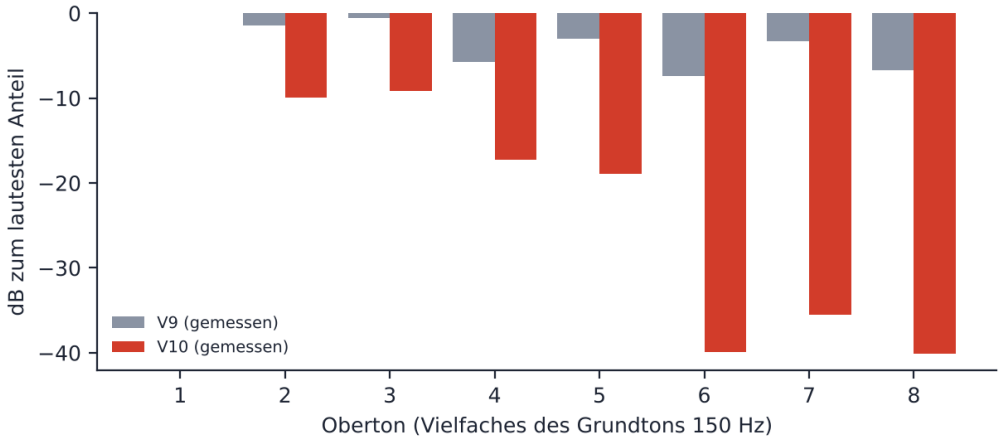


Abb. 24 – Obertöne bei Vollgas, gemessen in Aufnahmen vom PC3: V9 fast flach (metallisch), V10 mit klar dominantem Grundton.

RACER_V13.BAS – Motor-Wellenform mit Auspuff-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)                    ' Zuendung
      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)                  ' Mechanik
    NEXT
  NEXT
  ' Auspuff: zweimal gleitender Mittelwert ueber ELP Werte (ringfoermig)
  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
  ' Gleichanteil weg, auf volle Aussteuerung bringen
  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
```

Rauschtyp P klingt metallisch

Periodisches Rauschen (Typ P) erzeugte beim Schalten und auf dem Rumpelstreifen ein metallisches „Zirpen“ – in der Aufnahme als Breitband-Stöße bis über 6 kHz zu sehen. Ersatz: ein tiefer Sinus (55 Hz) als Schalt-Plopp, ein tiefes Rechteck als Rumpeln, feineres Rauschen (Typ N, 1500 Hz) fürs Gras.

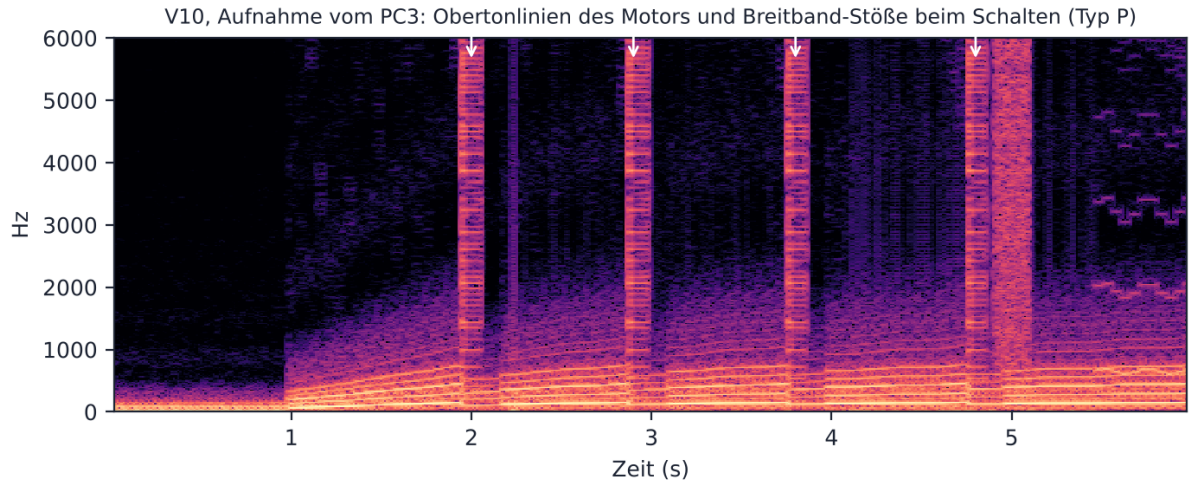


Abb. 25 – Spektrogramm der V10-Aufnahme: oben die Stöße des Schaltgeräuschs (Pfeile), unten die Obertonlinien des Motors.

9.3 MOD-Musik mit Effekt-Samples

`RACER.MOD` ist ein eigens erzeugtes 4-Kanal-ProTracker-Modul (Synthwave, A-Moll, 128 BPM, 82 s bis zur Wiederholung). In den Slots 20–27 liegen die Effekte, die das Spiel mit `PLAY MODSAMPLE nr, kanal, lautstärke` über die Musik legt:

Slot	Effekt	Grundlage	Kanal
20	Crash	Aufnahme impact.ogg + kurzes Reifenkreischen	1
21	Gras/Schotter (2 s)	Körnung aus Splittern der Aufprall-Aufnahme	3
22	Reifenquietschen (3 s)	Aufnahme skid.ogg	3
23	Schalten	tiefer Sinus-Plopp	2
24	Rumpelstreifen (2 s)	Aufprall-Anschläge, ~22 pro Sekunde	3
25	Vorbeifahrt	Motor-Aufnahme mit Doppler-Verlauf	4
26	Runden-Jingle	synthetisch	2
27	Stille	beendet einen laufenden Effekt	3

Die wichtigste Erkenntnis: MODSAMPLE kennt keine Schleifen

Der Test mit einem geloopten Sample (Bass, 0,5 s) zeigte: Nach einer halben Sekunde ist der Ton weg. Daraus folgt:

- **Dauergeräusche** (Quietschen, Gras, Rumpeln) müssen regelmäßig neu gestartet werden. Mit kurzen Samples (0,5 s) und Blenden entstand ein pumpendes „wish-wish-wish“. Die Lösung: **lange Samples** (2–3 s) mit nur 8 ms Blende, Neustart kurz vor dem Ende, und zum Beenden ein **Stille-Sample**, das den laufenden Effekt auf seinem Kanal ersetzt. Eine Nachlaufzeit von 0,25 s verhindert Zerhacken an Zustandsgrenzen.
- **Ein Motor mit variabler Drehzahl über MOD** ist mit V7.00.00 nicht sauber machbar: Überblenden zwischen Drehzahlstufen auf zwei Kanälen pulsiert hörbar. Ein Wunsch an die Firmware (TheBackShed-Forum): Loop-Unterstützung und Stopp für `PLAY MODSAMPLE` – damit wäre der Mix aus Musik und Motor lösbar.
- **Abspielrate:** Die Effekte sind für 8287 Hz (Note C-2) gebaut. Anteile über ~4 kHz fehlen, Reifenquietschen klingt etwas dumpfer als die Aufnahme.

RACER_V13.BAS – Untergrund-Effekte mit langen Samples, Nachlauf und Stopp

```
' --- Untergrund: Sample wiederholen, solange der Zustand anhaelt
t% = 0 : ax = ABS(playerX)
IF ax > 1 AND speed > 0 THEN
  t% = 21                                    ' Gras/Schotter
ELSEIF FXNOISE AND ax > 0.85 AND sp > 0.1 THEN
  t% = 24                                    ' Rumpelstreifen
ELSEIF FXNOISE AND ABS(segCv(ps%)) * sp * sp > 1.4 THEN
  t% = 22                                    ' Reifenquietschen
ENDIF
IF t% > 0 THEN
  fxOff = TIMER + 250                        ' Nachlauf: kurzes Flackern nicht hoerbar
  IF t% <> fxT% OR TIMER >= fxNext THEN
    PLAY MODSAMPLE t%, 3, FXVOL
    fxT% = t%
    fxNext = TIMER + CHOICE(t% = 22, 2880, 1880) ' kurz vor Sample-Ende (3.0 / 2.0 s) neu starten
  ENDIF
ELSEIF fxT% > 0 THEN
  IF TIMER >= fxOff THEN
    PLAY MODSAMPLE 27, 3, 1                  ' Stille-Sample beendet den Effekt
    fxT% = 0
  ENDIF
ENDIF
```

9.4 Ausblick (Stand der Studie)

- **Fahrgefühl:** Die Beschleunigung ist bisher über den ganzen Bereich gleich. Realistischer (und spannender) wird es, wenn sie mit der Geschwindigkeit abnimmt, etwa `ACCEL · (1 − 0,7 · sp²)` mit sp = speed / MAXSPD, oder je Gang gestaffelt.
- **Gegner:** 90 Autos füllen die Strecke gut; die Kosten wachsen mit der Zahl, die Messzeile zeigt es.
- **Leistung:** Größter verbliebener Hebel ist die Straßenschleife als CSUB über mmb2csub (Richtung 30 FPS).
- **Ton:** Mit Loop-Unterstützung für PLAY MODSAMPLE wären Musik und Motor gleichzeitig möglich.

Teil II

ROAD 1.0 – das Spiel

Vom Prototyp zum fertigen Spiel: was sich geändert hat und warum, ein Rundgang durch den Quellcode, eigene Strecken, Grafik und Musik – und eine Einladung zum Mitmachen.

10 Vom Prototyp zum Spiel



Abb. 26 – Titelbild mit Straßenschild (hier: Credits).



Abb. 27 – Texas: Tafelberge, Kakteen, Stierschädel.

So läuft ROAD
ROAD ist **vollständig in MMBasic** geschrieben – ohne CSUB, mit der Firmware **PicoMite V7.00.00** (PicoMitePC3) auf dem Pico Computer 3. Zwei Einstellungen machen den Unterschied und werden einmal an der Eingabeaufforderung gesetzt:
`OPTION RESOLUTION 640x480, 378000` – Takt **378 MHz** statt 252 MHz, alles läuft rund 50 % schneller.
`OPTION COMPILE ON` – der Compiler von V7 übersetzt das Programm beim RUN; über 99 % der Anweisungen laufen kompiliert.
Video: `https://youtu.be/W4qjgm6pynE`

Die Machbarkeitsstudie aus Teil I zeigte: Die Technik trägt auf dem Pico. Aus ihr wurde in gut zwanzig Schritten das Spiel **ROAD**. Jeder Schritt wurde auf dem Pico Computer 3 gefahren, gemessen (Profiler, Bildrate) und oft per Video begutachtet. Was dazukam:

Bereich	ROAD 1.0
Spielablauf	Titelbild mit Vorführfahrt, Fahrername, 3 Runden mit Zeitlimit und Checkpoints, Ausrollen nach Zeitablauf mit Rettungschance, Bestenlisten je Strecke, Freischalten der nächsten Strecke, Pause und Abbruch mit Rückfrage
Strecken	fünf Strecken als Textdateien: **Green Valley**, **Texas** (Stierschädel als Hindernisse), **California** (Küste bei Sonnenuntergang, 4 und 2 Spuren), **Circuit** (Acht mit Brücke, Tunnel, Haarnadel, Curbs), **City** (Nacht, drei Spuren, Flussbrücke, Altstadt, Unterführung)
World Tour	alle Strecken nacheinander, je 2 Runden, Zeiten addiert, eigene Bestenliste; wer sie schafft, schaltet alles frei
Optionen	ROOKIE / VETERAN (Zeitzuschlag, Verkehr, eigene Listen), PRACTICE (ohne Verkehr, zum Strecke lernen), Menümusik
Grafik	je Strecke eigenes Sprite-Sheet und eigene Palette, Hintergrund aus bis zu fünf Pixelbild-Ebenen mit Parallaxe, gezeichneter Tunnel, Küste mit Sand und Wasser, Spurwechsel mit Verjüngung, gespiegelte Sprites
Ton	je Strecke ein eigenes MOD mit Effekt-Samples; Motor-Modus (Taste E), Effekte ohne Musik (Taste M)

10.1 Was sich an der Technik geändert hat – und warum

Das Spiel macht deutlich mehr als die Studie: breitere Straßen mit bis zu vier Spuren, Küste, Tunnel, 90 Autos, Menüs. Damit es trotzdem bei **17 bis 20 Bildern pro Sekunde** bleibt, weicht ROAD an einigen Stellen bewusst von Jake Gordons Original ab. Die Grundidee – Segmente, Projektion, Kurven-Akkumulation, Sprites mit Clipping – ist unverändert.

Straße: Zeilen-Rechtecke in Blöcken

Schon die Studie zeichnete die Straße zeilenweise mit einem `BOX`-Array statt mit `POLYGON` (Kapitel 8). Mit Küste, Sand und vier Spuren wuchs das Array auf über 1700 Einträge – und `BOX` zeichnet immer das *ganze* Array. Die unbenutzten Einträge mit `MATH SET` zu füllen und zu zeichnen kostete plötzlich 5 Bilder pro Sekunde. ROAD zeichnet deshalb in **Blöcken zu 128 Rechtecken**: `MEMORY COPY INTEGER` kopiert jeden Block in ein kleines Arbeits-Array, hinter dem letzten Eintrag liegt ein Block unsichtbarer Dummy-Rechtecke. Gezeichnet wird nur, was es in diesem Bild wirklich gibt.

ROAD.BAS – Zeilen-Rechtecke blockweise zeichnen (aus 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
```

Heißer Code bleibt am Stück

Ein SUB-Aufruf kostet in MMBasic deutlich mehr als die Zeilen darin. Ein Versuch, `Render` in übersichtliche Unter-SUBs aufzuteilen, kostete messbar Bildrate und wurde zurückgenommen. Deshalb ist `Render` lang, die Straßenbreite wird inline statt per Funktion berechnet, und `DrawSpr` rechnet mit globalen Hilfsvariablen statt mit LOCALs. Die Regel aus Kapitel 8 gilt weiter: **in der innersten Schleife nichts aufrufen, was man ausschreiben kann**.

Hintergrund: Pixelbilder statt Spalten

Die Studie kopierte drei 320 Pixel breite Streifen aus dem Sprite-Puffer; spätere Fassungen berechneten Hügel und Waldkante als Spalten-Rechtecke. ROAD 1.0 legt bis zu fünf **beliebig breite Pixelbilder** in die RAM-Bildplätze des RP2350 (PSRAM) und kopiert pro Bild den sichtbaren Ausschnitt mit `BLIT FLASH`, Schwarz ist durchsichtig. Gemessen auf dem PC3: 1,3 ms für eine 320 × 70 Pixel große Ebene – schneller als eine einzige Reihe Spalten-Rechtecke (1,5 ms). Damit sind Dithering, Schnee, Fenster und Wolken je Strecke frei malbar (Kapitel 12.4).

Farben: 16 Slots, alle verschieden

Zeichenbefehle bekommen eine 24-Bit-Farbe; die Firmware sucht daraus den passenden Slot. Zwei Slots mit derselben RGB555-Farbe sind deshalb tückisch: Es wird immer der erste gefunden. Eine eigene Palette in einer Track-Datei braucht deshalb **16 verschiedene Farben**. Für den Tunnel werden nur Straße, Ränder, Linien und Autos abgedunkelt; fallen dabei zwei Slots zusammen, schiebt `MakeDark` sie um eine Stufe auseinander. Die Landschaft ist im Tunnel ohnehin verdeckt und erscheint am Ausgang in ihren normalen Farben.

Eingabe: Fahren per KEYDOWN, Menüs per INKEY$

`KEYDOWN` meldet gehaltene Tasten (Gas + Lenken gleichzeitig), `INKEY$` liefert getippte Zeichen (Namen, Menüs; funktioniert auch über das Terminal). Eine Lehre aus dem Test: Im Rennen

wartete die Rückfrage „QUIT RACE?“ zunächst auf `INKEY$` – und bekam nie eine Taste. Seitdem wird im Rennen *alles* über `KEYDOWN` und Tasten-Kanten gelesen.

Zufall mit festem Startwert

`RANDOMIZE` hat auf dem RP2350 keine Wirkung. ROAD benutzt einen eigenen kleinen Zufallsgenerator (`Rn()`) mit Startwert je Strecke – so steht jeder Baum bei jedem Laden an derselben Stelle.

11 Rundgang durch den Quellcode

`ROAD.BAS` ist eine einzige Datei mit rund 2600 Zeilen und englischen Kommentaren. Sie ist von oben nach unten so geordnet:

Abschnitt	Inhalt	wichtige SUBs
Konstanten, DIM	Bildschirm, Straße, Ton, Physik, Spielablauf, Paletten-Slots, Puffer, Zustand	–
Hauptschleife	Eingabe → Zustände → Logik im festen Zeitschritt → Ton → Zeichnen	–
Initialisierung	Bildmodus, Puffer, Palette	Init, ApplyPal
Track-Dateien	Parser für roadN.trk, Strecke abschließen, Spuren, Küste, Tunnel	LoadTrack, FinishTrack, Zone, Side, Tunnel, MakeDark
Streckenbau, Sprites	Bausteine aus Teil I, Bepflanzung, Hindernisse	AddRoad, AddSprite, Scatter, Clusters
Eingabe, Welt	Tasten, Physik, Kollisionen, Checkpoints, Autos	ReadKeys, Update, MoveCars, CarsAI, CarHit
Zeichnen	Hintergrund, Straße, Küste, Tunnel, Sprites	Render, DrawLy, DrawSpr
HUD, Overlays, Menüs	Anzeige, Titel, Optionen, Credits, Ergebnis	Hud, Overlay, Options, Credits, TourResult
Spielablauf	Zustandsautomat, Zeiten, Bestenlisten, World Tour	StateLogic, StartRace, RaceTimes, GoResult
Dateien, Karte, Ton	road.cfg, .hi, Mini-Map, Motor, MOD-Effekte	SaveCfg, BuildMap, SoundFor, AudioMod

11.1 Die Hauptschleife

Wie in Teil I läuft die Logik in festen Schritten (88 pro Sekunde), gezeichnet wird so oft es geht. Neu ist die Trennung der Eingabe nach Zustand und dass Autos, Tunnel-Palette und Ton nur einmal pro Bild behandelt werden:

ROAD.BAS – Hauptschleife (Auszug)

```
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 Der Zustandsautomat

Zustand	Bedeutung
S_TITLE / S_NAME	Titel mit Vorführfahrt, Streckenwahl, Optionen, Credits / Namenseingabe
S_START	Vorhang schließt, Strecke wird geladen, Vorhang öffnet
S_COUNT	3 – 2 – 1 – GO
S_RACE	Rennen; Checkpoints geben Zeit, Ziellinie zählt Runden
S_TOVER	Zeit abgelaufen: kein Gas, das Auto rollt – erreicht es rollend einen Checkpoint, geht es weiter
S_FINISH	im Ziel: das Auto bremst selbst (Bremslicht)
S_RESULT	Ergebnis, Bild steht bis ENTER; in der World Tour Zwischenstand

Zustand	Bedeutung
S_PAUSE	Pause oder Rückfrage „QUIT RACE?“

`StateLogic` schaltet die Zustände weiter, `Overlay` zeichnet je Zustand Schild, Meldungen und Fenster. Das Straßenschild im Titel ist eine Fläche mit wechselndem Inhalt: Streckenwahl, World-Tour-Seite, Optionen, Credits oder die Rückfrage zum Beenden.

11.3 Was in einem Segment steckt

Jedes Segment hat wie in Teil I Höhe, Kurve und Sprite-Liste. Alles Weitere ist in die Bits einer einzigen Ganzzahl `segCol%()` gepackt, damit die Straßenschleife mit einem Wert auskommt:

Bits	Bedeutung
0–2	Farbsatz: 0 hell, 1 dunkel, 2 Start, 4 Checkpoint
3 / 4	Küste rechts / links (Sand und Wasser)
5–7	Zahl der Spuren (Fahrspurlinien)
8–15	Straßenbreite × 100 (Verjüngung bei Spurwechsel)
16	Tunnel
17	Hindernis in diesem Segment (Kollision nur dort prüfen)

11.4 Ein Bild entsteht

- **Himmel:** `CLS` in Slot 1.
- **Hintergrund:** `DrawLy` kopiert die Bild-Ebenen von hinten nach vorn, darunter eine Füllfläche.
- **Straße:** von vorn nach hinten, Zeile für Zeile in die Rechteck-Listen (Rand, Fahrbahn, Linien; Küste in einer eigenen Liste). Tunnel werden als Ringe in zwei dunklen Tönen gezeichnet; hinter dem Ausgang wird nur noch durch die Öffnung gezeichnet.
- **Sprites:** von hinten nach vorn mit `BLIT RESIZE`, am Hügel abgeschnitten; dazwischen das eigene Auto mit Bremslicht, Lenkbildern und Staub.
- **HUD und Overlay,** dann `FRAMEBUFFER COPY F, N`.

11.5 Hintergrund-Ebenen und der Kompass

Jede Ebene verschiebt sich in Kurven seitlich. Damit nach einer Runde wieder genau dasselbe Bild erscheint, folgt der Versatz nicht einfach der Summe der Kurven, sondern den **ganzen Drehungen**, die die Straße wirklich macht: California dreht sich einmal herum – die Berg-Ebene ist ein 1920 Pixel breites Panorama, das pro Runde genau einmal vorbeizieht, die Sonne erscheint also genau einmal. Wo sich Links- und Rechtskurven aufheben (Texas, die Acht), kehrt jedes Bild von selbst zurück.

ROAD.BAS – Bild-Ebenen mit Umbruch und eigener Geschwindigkeit

```
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 Zeiten und Schwierigkeit

In den Track-Dateien stehen die **sauberen Fahrzeiten** je Sektor (ohne Verkehr, ohne Fehler). Das Spiel rechnet beim Start den Zuschlag dazu und gibt am ersten Checkpoint zwei, am zweiten eine Sekunde extra – zum Ende einer Runde hin wird es also knapper:

ROAD.BAS – Zeitlimits aus den sauberen Zeiten

```
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: eigene Strecken, Grafik und Musik



Abb. 28 – California: Küste, Palmen, Sonnenuntergang.



Abb. 29 – City bei Nacht.

Fast alles, was eine Strecke ausmacht, steht außerhalb des Programms: in einer Textdatei, einem Sprite-Sheet, bis zu fünf Hintergrundbildern und einem MOD. Für eine neue Strecke muss man `ROAD.BAS` nur an einer Stelle anfassen.

12.1 Eine neue Strecke anlegen

- Die Dateien einer vorhandenen Strecke kopieren und umbenennen, z. B. `road6.trk`, `road6.bmp`, `road6.mod`, `road6sky/far/near.bmp`.
- In `ROAD.BAS` `CONST NTRK = 5` auf 6 setzen. Das Menü und die World Tour nehmen jede vorhandene Datei `road1.trk` … `roadN.trk` automatisch auf.
- In `road6.trk` Namen, Strecke, Sprites, Palette und Zeiten ändern. Kommentare beginnen mit `'`; `road1.trk` enthält im Kopf die Kurzbeschreibung aller Befehle.
- Mit MODE = PRACTICE ein paar saubere Runden fahren und die Zeiten eintragen (12.6).

12.2 Die Track-Datei

Eine Anweisung pro Zeile, Werte durch Kommas getrennt. Fehlende Kopfwerte haben Vorgaben. Sprites kommen immer *nach* der Strecke.

Befehl	Werte	Wirkung
NAME	text	Name im Menü
SHEET · MUSIC · MUSICFX	datei	Sprite-Sheet, MOD mit Musik, MOD nur mit Effekten (Taste M)
TIME	start, cp1, cp2, linie	saubere Sektorzeiten in Sekunden (12.6)
CARS · CARSPD · CARTYPE	anzahl / min, max / von, bis	Verkehr: Zahl, Tempo, Fahrzeugtypen (3 = Laster)
PAL	slot, &Hrrggbb	Palettenfarbe; Slots 4, 5, 7, 8, 11–15 färben auch die Autos
SPRDEF · MIRROR	typ, x, y, b, h, weltbreite, klein / typ, neu, x, y	Sprite-Zelle neu festlegen / gespiegelte Kopie (Typ 38–43)
HAZARD	typ	Sprite-Typ ist ein Hindernis (drüber: stark, gestreift: leicht bremsen)
LAYER · BGFILL	datei, y, tempo [, wiederholungen] / slot	Hintergrundbild-Ebene (12.4) / Farbe darunter
BG · BGSTYLE · STRIP	höhen / 0, 1 / y	Hintergrund ohne Bilder (berechnet)
LANES · WIDTH · ZONE	spuren / faktor / von, bis, spuren, breite, übergang	Spuren und Breite, auch abschnittsweise
SIDE	von, bis, sandbreite, seite	Küste mit Sand und Wasser (−1 links, 1 rechts)
TUNNEL	von, bis	gezeichneter Tunnel; Portal als Sprite-Typ 37
RUMBLE · LINES · HUDCOL	slot, slot / 0, 1 / slot	Randfarben (z. B. rot/weiß), Fahrspurlinien, HUD-Farbe
MAPTURN · MAPK	grad / faktor	Form der Mini-Map: 360 Rundkurs, 0 Acht; Krümmung
STRAIGHT · CURVE · HILL	n / n, kurve, höhe / n, höhe	Bausteine aus Teil I (je 3·n Segmente)
ROLLING · SCURVES · BUMPS · DOWNHILL · ROAD	siehe road1.trk	Wellen, S-Kurven, Bodenwellen, Rückkehr auf Höhe 0, Grundbaustein
SPRITE	segment, typ, offset [, maßstab]	ein Objekt; |offset| < 1 steht auf der Straße
SCATTER · CLUSTERS	siehe road1.trk	Bepflanzung in Abschnitten, Gruppen mit Tafeln

12.3 Das Sprite-Sheet



Abb. 30 – road2.bmp (Texas) in den Farben der Strecke. Magenta = durchsichtig.

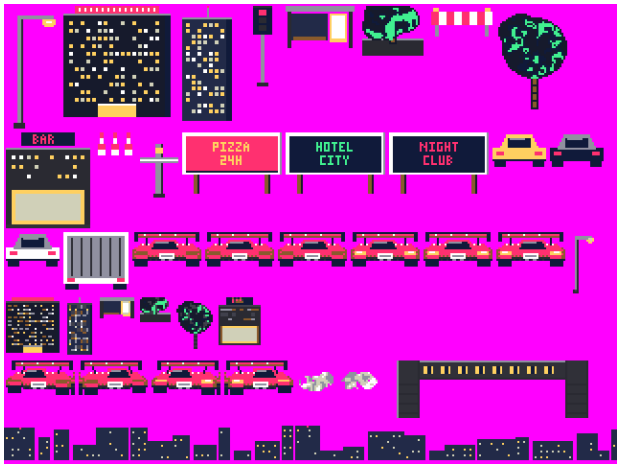


Abb. 31 – road5.bmp (City).

- **Format:** BMP mit 320 × 240 Pixeln, gemalt in den **16 Standardfarben** (Slot 0 Schwarz, 1 Blau, 2 Myrtle …). `LOAD IMAGE` legt jedes Pixel in den Slot seiner Standardfarbe; im Spiel erscheint es in der Palettenfarbe der Strecke. Schwarz ist durchsichtig.
- **Feste Zellen:** Die Autos (Typen 14–23) und die Lenk- und Staubbilder (31–36) müssen an ihrem Platz bleiben – sie werden auf allen Strecken benutzt.
- **Freie Zellen:** Pflanzen, Tafeln und Objekte (0–13, 24–30, 37) dürfen ersetzt werden; mit `SPRDEF` bekommt eine Zelle neue Maße und eine neue Weltbreite (wie breit das Objekt im Verhältnis zur halben Straße ist).
- **Spiegeln:** `MIRROR` kopiert eine Zelle beim Laden gespiegelt in freien Platz – so braucht eine Laterne für beide Straßenseiten nur eine Zeichnung.

12.4 Hintergrundbilder

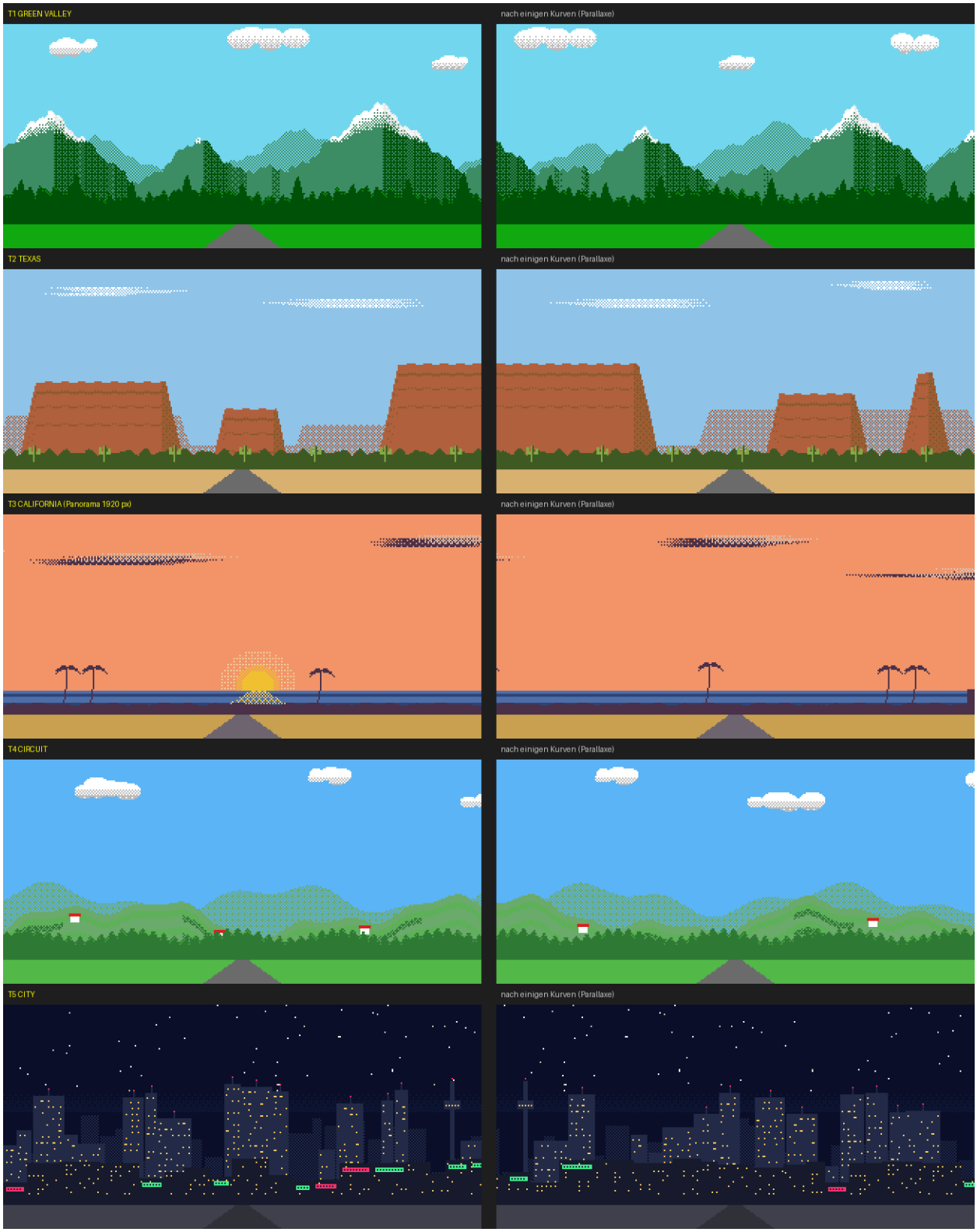


Abb. 32 – Die Hintergründe der fünf Strecken, links beim Start, rechts nach einigen Kurven: Wolken, Berge und Vordergrund wandern verschieden weit.

- **Ebenen:** bis zu fünf LAYER-Zeilen, von hinten nach vorn. Üblich sind drei: Himmel/Wolken (y 0, Tempo 0,33), Berge (y 48, Tempo 0,67), Vordergrund (y 95, Tempo 1). Der Horizont liegt bei y 128; darunter füllt BGFILL die Fläche.
- **Größe:** jede Breite ab 320 Pixeln, das Bild läuft rundum. Breitere Bilder wiederholen sich seltener. Die Höhe ist frei.

- **Farben:** wie beim Sheet die 16 Standardfarben, Schwarz ist durchsichtig. Dithering (Schachbrettmuster aus zwei Farben) wirkt aus der Entfernung wie eine Zwischenfarbe – ein Mischen mit der Himmelsfarbe ergibt Dunst.
- **Wiederholungen:** der vierte Wert von LAYER sagt, wie oft das Bild pro Runde vorbeizieht. 1 bedeutet Rundblick: ein Wahrzeichen wie Sonne, Mond oder Fernsehturm erscheint genau einmal. Ohne Angabe wählt das Spiel den Wert passend zum Tempo.

12.5 Musik und Effekte

Jede Strecke spielt ein eigenes 4-Kanal-MOD (ProTracker-Format). Die Effekte des Spiels liegen **im selben MOD** als Samples 20 bis 27 und werden mit `PLAY MODSAMPLE` über die Musik gelegt. Wer eine neue Musik baut, muss diese acht Samples übernehmen:

Sample	Effekt	Hinweis
20	Crash	kurz
21	Gras / Schotter	ca. 2 s, wird wiederholt
22	Reifenquietschen	ca. 3 s, wird wiederholt
23	Schalten / Countdown	kurz
24	Rumpelstreifen	ca. 2 s, wird wiederholt
25	Vorbeifahrt	Motor mit Doppler
26	Runden-Jingle	kurz
27	Stille	beendet einen laufenden Effekt

- **MUSICFX:** dieselben Samples mit stiller Musik – das Spiel wechselt dorthin, wenn man im Rennen die Musik abschaltet (Taste M). `road1fx.mod` passt zu allen Strecken.
- **Grenzen:** `PLAY MODSAMPLE` kennt keine Schleifen und keine Tonhöhe; lange Geräusche werden deshalb kurz vor dem Ende neu gestartet (Kapitel 9.3). Die Effekte sind für 8287 Hz gebaut.
- **Größe:** die mitgelieferten MODs haben 120–160 KB.

12.6 Zeiten einmessen

`TIME, start, cp1, cp2, linie` sind saubere Fahrzeiten in Sekunden: Sektor 1 aus dem Stand, dann die Sektoren nach Checkpoint 1, nach Checkpoint 2 und nach der Ziellinie (Sektor 1 in voller Fahrt). Eine Runde besteht aus drei gleich langen Sektoren. So misst man:

- Im Menü mit **O** die Optionen öffnen, MODE = PRACTICE: kein Verkehr, keine Rekorde.
- Zwei, drei saubere Runden fahren; die Zeiten im Ergebnis und die Restzeit an den Checkpoints zeigen, wie viel Luft bleibt.
- Die Mittelwerte eintragen. Das Spiel rechnet für ROOKIE 30 %, für VETERAN 15 % dazu (am ersten Checkpoint +2 s, am zweiten +1 s).

Strecke	TIME (sauber)	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 Mitmachen!



Abb. 33 – Circuit: Start in die Acht.

ROAD ist als Grundlage gedacht – für eigene Strecken, eigene Grafik, eigene Musik. Alles, was eine Strecke ausmacht, liegt in offenen Dateien, und dieses Handbuch beschreibt, wie sie aufgebaut sind. Jetzt seid ihr dran.

Ein Aufruf an die Spezialisten
Du bist Pixel-Künstler? Sprite-Sheets und Hintergrund-Panoramen in 16 Farben sind genau dein Revier. Kapitel 12.3 und 12.4 beschreiben Format und Regeln.

Du schreibst MOD-Musik? Ein eigener Soundtrack pro Strecke im ProTracker-Format, vielleicht sogar bessere Effekt-Samples, würde ROAD enorm aufwerten. Kapitel 12.5 beschreibt, was das Spiel von einem MOD erwartet.

Wir würden uns riesig freuen, wenn ihr euch daran versucht!

13.1 Ideen für die Community

- **Neue Strecken:** Wüste bei Nacht, Schnee und Eis, Hafen, Gebirgspass, eine Strecke durch die eigene Heimatstadt.
- **Neue Sheets und Hintergründe:** andere Jahreszeiten, Wetter, ein Sonnenaufgang.
- **Musik:** ein MOD pro Strecke im passenden Stil.
- **Spielideen:** Gamecontroller, Rückspiegel, Geister-Auto der besten Runde, Wetter, zwei Spieler.
- **Leistung:** die Straßenschleife als CSUB – Richtung 30 Bilder pro Sekunde.

Fragen, Strecken und Ergebnisse gerne im **The Back Shed Forum**. Das Video zum Spiel: `https://youtu.be/W4qjgm6pynE`

Viel Spaß beim Fahren und Bauen – have a good drive!

14 Anhang

14.1 Konstanten: Original und Studie RACER.BAS

Name (Original)	Original	RACER.BAS	Bemerkung
width × height	1024 × 768	320 × 240	MODE 2
roadWidth	2000	ROADW = 2000	halbe Straßenbreite
segmentLength	200	SEGLEN = 200	
rumbleLength	3	RUMBLE = 3	
lanes	3	fest 3	Linienbreite 1/32, Rand 1/6
fieldOfView	100°	100°	CAMD ≈ 0,839
cameraHeight	1000	CAMH = 1000	
drawDistance	300	DD = 120	Kapitel 6
fogDensity	5	–	kein Nebel in 16 Farben
maxSpeed	segmentLength/step = 12000	MAXSPD = 12000	ein Segment pro Logikschritt
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	gleich	Parallax-Faktoren
totalCars	200	NCAR = 40	Kapitel 5.5
resolution	height/480	RES = 0,5	

14.2 Verwendete Befehle (PicoMite V7.00.00)

Befehl	Einsatz im Spiel
MODE 2	320 × 240, 16 Farben
MAP(n) = rgb, MAP SET, MAP(n)	Palette setzen, Farbwert eines Slots holen
FRAMEBUFFER CREATE / CREATE 2 / LAYER / WRITE / COPY F, N	Doppelpufferung, Sprite-Puffer, HUD-Layer
LOAD IMAGE	SPRITES.BMP in Puffer 2 laden
POLYGON n%(), x%(), y%(), c%(), c%()	alle Straßen-Trapeze eines Bildes
BOX	Rasenstreifen, Waldkante
BLIT FRAMEBUFFER 2, F, …, 0	Hintergrundstreifen mit Transparenz
BLIT RESIZE 2, F, …, 0	Sprites skalieren
TEXT	HUD
KEYDOWN(n)	mehrere Tasten gleichzeitig
TIMER	fester Zeitschritt
MATH SET	Listenköpfe zurücksetzen
OPTION COMPILE ON, OPTION PROFILING ON	Geschwindigkeit, Messung
MEMORY COPY INTEGER, PEEK(VARADDR …)	ROAD: Blöcke der Zeilen-Rechtecke umkopieren

Befehl	Einsatz im Spiel
FLASH LOAD IMAGE n, f$, O · BLIT FLASH n, F, …, 0	ROAD: Hintergrundbilder in den RAM-Bildplätzen 4–8 (PSRAM)
BLIT READ / BLIT WRITE …, 1	ROAD: gespiegelte Sprite-Kopien (MIRROR)
PLAY MODFILE, PLAY MODSAMPLE	ROAD: Musik und Effekte je Strecke

14.3 Quellen, Lizenzen und Dank

- Jake Gordon: „How to build a racing game“, Teile 1–4 (straight, curves, hills, final), jakesgordon.com, 2012.
- Quellcode: github.com/jakesgordon/javascript-racer – MIT-Lizenz, Copyright © 2012–2016 Jake Gordon and contributors. RACER.BAS ist eine Übertragung dieses Codes nach MMBasic und steht entsprechend unter derselben Lizenz. Musik und Sprites des Originals sind nicht enthalten.
- Louis Gorenfeld: „Lou’s Pseudo 3d Page“, extentofthejam.com/pseudo – Ursprung der Kurven-Akkumulation und der Z-Map.
- Geoff Graham, Peter Mather u. a.: PicoMite User Manual V7.00.00 sowie Game Development Guide, BLIT User Manual, MMBasic Performance Guide und Release Notes (Supplementary Manuals 7.00.00). Firmware: github.com/UKTailwind/PicoMite, Release PicoMite-V7.00.00.
- Effekt-Aufnahmen (Reifen, Aufprall, Motor): Kenney „Starter Kit Racing“, Lizenz CC0 (Skid-Sound von Landeplage, CC0). Musik (MODs) und übrige Samples sind eigens für dieses Projekt erzeugt.
- ROAD 1.0: Code & Design Matthias (homa) mit Claude, 2026. Hardware: Pico Computer 3 von Peter Mather. MMBasic von Geoff Graham & Peter Mather. Dank an das The Back Shed Forum und die Community.