MANU & KIT

Travel Through Time — In Detail

A book about how we made the game **Travel Through Time Volume 1: Northern Lights** *for the ZX Spectrum computer*

1

Creating computer games is a devilishly fascinating process. When working on a large project, an enormous number of questions and tasks inevitably arise—questions whose answers cannot be found in books like *"How to Write a Game in Assembly."* Our book will not teach you how to program, but—we are sure—it will be very interesting for everyone who played our games and wondered: *"How did they do that?"*

The facts in this book are presented in no particular order. Some other games related to the **Travel Through Time** series (hereinafter sometimes referred to as **TTT**) are mentioned, as well as details that did not make it into the final version of the game.

2

The original game project was entirely devoted to one well-known automobile brand. In the end, we removed all mentions of car names (and will not mention them in this book either) in order to avoid claims from rights holders, and also added other cars and various vehicles.

The very first version of the title screen has survived, along with a rather pompous description of the original project:

> "The first racing game for the ZX Spectrum with a photorealistically depicted road. It is time to forget about striped plowed fields and track curbs, about monotonous levels and roadside objects growing like mushrooms.
>
> The tracks in this game are truly alive. Forest areas and open spaces, hills, cliffs and rock walls, tunnels, intersections, functioning gas stations, populated areas with different types of buildings, bridges, and even… a working railway crossing!
>
> And what else have you not yet seen in racing games on the ZX Spectrum? A wide variety of competitions are offered: time trial, duel, mass start, and even orienteering—based on real maps of regions in Sweden. And yes, in this type of competition, intersections are real.
>
> Points earned in races are used to upgrade the car. All parts influence handling, acceleration, and top speed to some extent. In addition, a separate upgrade procedure is implemented for the turbocharger, with an even greater number of parameters. When a new car appears, the turbo is transferred to it and its parameters are preserved."

As you can see, not all of the original ideas were implemented (some of them—for example, upgrades—did not fit into the final concept), but many other interesting details appeared instead.

3

The game uses more than a dozen graphical formats and rendering procedures. For each element, the most rational methods of output and data storage are used.

For example, rendering the player’s car and rendering roadside poles use the same procedure, while opponent cars use a completely different one, because they employ automatic masks, per-pixel positioning, clipping, and slight vertical scaling. Separate procedures are used for panoramic backgrounds, objects “emerging” from beyond the horizon, mid-ground trees, color sprites in the interface and cutscenes, large character portraits, and so on.

4

As in the case of **DRIFT!**, all player cars and some opponent vehicles were first prepared as three-dimensional models and then drawn from different angles. The detailing of the models was done with consideration for ease of subsequent pixel drawing.

5

When drawing cars, we used our own utility for converting 3D models into pixel graphics. This program, called **Ze3dex**, had already been used by us during the development of **DRIFT!**. It allows virtually any source model to be displayed in pixels, but works much better with specially prepared low-polygon models that emphasize the necessary structural elements.

This method fundamentally differs from the typical approach, where not the model itself but its rendered image is converted. Below is shown how an image would look with a typical graphic conversion, and how it looks when converting the model directly.

Nevertheless, while large images the size of a ZX Spectrum screen initially come out at an acceptable quality using this program, sprites like those in *Travel Through Time* require extensive manual refinement. In this game, they were essentially redrawn from scratch.

6

A very interesting graphics format is used for the player’s car and roadside objects. It was originally developed specifically for this game, and therefore is called **Travel Through Time Sprite** (.tts). Before the game was released, we had already used it in other games—**DRIFT!** and **Maureen Miles** (known as *Just a gal* in the ZOSYA release). A sprite editor was written for this format.

The essence of this format is that sprites are assembled from separate parts of different sizes, which may or may not use a mask. Thanks to this, sprites of complex shapes are stored in the most compact form, and during rendering a set of procedures is used that is optimal for each sprite fragment. There is a subroutine for outputting a fragment one character cell wide without a mask; the same with a mask; then the same two procedures but mirrored; then all of the above for a width of two character cells, and so on.

It looks overly cumbersome, but it works very fast. There are limitations: for example, in TTT we use fragment widths of one, two, or four character cells, and the height is always a multiple of two lines, as is the vertical positioning of the sprite in the game.

It is also possible to agree that we never use some fragment variant in the game (for example, width 2 with mask and mirroring), and then remove unnecessary procedures from the code.

But that is not all the features of the format.

It also allows storing sprite fragments as “columns” consisting of a pair of repeating bytes—and this is not only very compact, but during sprite output the height of such a column can be changed. As one might guess, this is an ideal option for rendering roadside poles. Moreover, for subsequent sprite fragments it is possible to use positioning relative to the top of the last column, or with an additional offset.

An interesting format overall. But its drawbacks are very significant: horizontal positioning is only by character cells, and it is difficult to organize sprite clipping.

7

As mentioned above, roadside objects use the same sprite format as the player’s cars. It does not support horizontal shifting “on the fly” (it should be said that procedural shifting is in any case an extremely poor option for these objects due to slow rendering speed), so the offsets are drawn in advance.

For the most distant sprites, four variants of horizontal offset are drawn, then two, and a single variant for the nearest sprite. This is due to the fact that horizontal movement of poles in the distance is much slower than nearby and requires more precise positioning. Such a solution can also be found in other racing games.

The middle part of the poles is made from a pair of repeating pixels (the principle described above). When rendering the sprite, the original height is adjusted depending on the distance to the object.

It can also be noticed that for some shifted sprites, certain small details are simply ignored so as not to use an extra character cell. This is also a kind of optimization—in the game it is completely unnoticeable.

Masks are used where necessary.

Left and right variants are used for lamps and turn direction signs.

All other roadside objects on the left and right sides of the road are identical. In the games **Rubinho Cucaracha** and **Travel Unlimited**, lamp posts also use a single variant for both sides of the road.

8

To make everything fit into memory, many data are stored in compressed form and decompressed as needed.

Both a standard packer and our own packing methods for various data types are used. For example, a token system is used for texts.

9

Development of the game began with this mock-up of the game screen.

It was noted how the depiction of roads and surrounding objects in ZX Spectrum racing games is usually far from reality and generally very conditional. The basis for the mock-up was this hastily converted photograph.

Subsequent work on the game was carried out with this mock-up in mind, which represented a completely different atmosphere compared to all existing games. We tried to transfer this atmosphere into the game, taking into account the existing technical limitations.

For example, we depicted dark silhouettes of trees slowly drifting in the mid-ground, a rather “ragged” roadside (in this respect the capabilities of our engine matched the source image perfectly), and overall the picture in our game—with its black fills—gravitates toward enclosed spaces, which is almost never found in ZX Spectrum racing games. This makes the panoramic views that occasionally open up on the horizon look especially striking.

Of course, not everything could be implemented in full accordance with the original mock-up.

For example, it would have been very difficult to create hilly terrain beyond the roadside with houses and trees on it (all of this in motion and at a decent speed). We carried out some work on simplifying the picture:

...Well, or at least stone boulders at the very beginning of the race? So as not to draw a large number of variants at different scales.

Alas, in the final version of the game there remained only one dark rocky ridge along the road, which for some reason is present only on a single, 45th level.

10

So, in the **TRAVEL** engine, in addition to typical roadside objects, a so-called “mid-ground” is used. This is a unique element not found in other games: distant objects that move more slowly than roadside poles, but nevertheless appear and leave the screen, unlike a permanent background.

There are two types of such objects: trees that almost merge with the background, and buildings and structures “emerging” from beyond the horizon.

Another element unique to this game, which has already been mentioned, is panoramic backgrounds. These are sets of finely detailed images that occasionally appear from beyond the horizon instead of standard backgrounds. Unlike the latter, they unfortunately do not move horizontally, but they create an incomparable atmosphere.

Some other objects, for example overpasses under which cars drive, although not entirely unique, also have interesting features. All such objects will be discussed further.

11

To display opponent cars (as well as in cutscenes and for other purposes), a simple linear sprite format without masks is used. The main advantage lies in the sprite rendering procedure itself: it uses precise per-pixel positioning both horizontally and vertically, automatic masking, clipping at the screen edges, and allows vertical sprite scaling. Considering all this functionality and the ability to scroll sprites of any width “on the fly,” it is obvious that the procedure cannot be ultra-fast. Nevertheless, it copes perfectly with the tasks of our engine. As a rule, only one opponent is seen at a large scale, while the remaining sprites, if present, are much smaller and therefore rendered quickly.

As for clipping at the screen edges, it can be either automatic or forced. The latter is needed for cases where the center of the car is so far to the left on the screen that it is already on the right. But the engine still knows that the car is actually on the left and that it must be clipped from the left side.

In addition, if necessary, the granularity of horizontal positioning can be adjusted.

12

Vertical scaling of sprites is implemented as follows: with each line of the sprite, a counter is increased, and if it reaches a specified value, it is reset to zero and the line is repeated once more. Thus, maximum scaling (two times) is achieved when the scale value is equal to 1.

With a value of 2, the sprite becomes one and a half times taller, since every second line of the original sprite is duplicated. The game uses moderate values, greater than 8. Scaling is applied only to the “near,” largest sprites.

A value of −1 (255), or any value greater than the sprite height, disables scaling.

Stretching a sprite looks unattractive if duplicated lines fall on parts of the sprite with horizontal lines.

Knowing in advance the scaling values used, one can try to draw the sprite in a more suitable way.

13

In the first working versions of the engine, this format was not yet used, and the coarse movement of opponent cars on the screen (to the extent allowed by the .tts format) became a stumbling block for further development. Imagine cars leaving the starting line “like stairs.” And keeping in memory pre-prepared offsets for each scale of each car would have been excessively expensive.

14

To create visual variety in the game, a set of mid-ground objects emerging from beyond the horizon is used. There are 16 of them in total.

Some appear only once in the game, others are repeated. The object index is specified in the level header if the object is present on the track.

These objects always appear on an uphill section, when the road smoothly turns to the right, and accordingly they always move to the left and upward, eventually disappearing behind the left curb. They are drawn with a perspective suitable for their placement. They are not scaled.

If, by the time work on these objects began, we had already been using the procedure for sprites with automatic masks and per-pixel positioning, we would certainly have rendered them with its help. Instead, we set ourselves the ambitious task of implementing very fast rendering with smooth movement. As a result, a large buffer is used for them, where copies of the sprite with offsets are prepared in advance. Moreover, separate procedures are used for their output, which include the necessary clipping capabilities.

Today it can be said with certainty that this approach was extremely suboptimal from the very beginning. Even if rendering a sprite with an automatic mask and on-the-fly shifting is slower, for such a rare and short episode this would be unnoticeable. On the other hand, a lot of memory could have been saved, and rendering capabilities would only have increased.

It would have been possible to draw objects on the right as well, and also apply slight scaling.

In general, the objects are good, but the procedures are not optimal.

15

Initially, the speedometer needle was planned to be done in raster graphics. We changed our minds when we estimated the required number of sprites.

As a result, the needle is drawn using vector graphics. A special utility was written to calculate the coordinates of the line segments. The resulting table is embedded directly into the program code. The needle has 62 positions, which is more granular than the change in the speed value itself. It is redrawn directly on the screen, and the previous position is erased. To reduce the load on the engine, the procedure is executed every other frame, and of course only if the needle position has changed.

Interestingly, the coordinates of the inner point, which changes within a small range, are stored in the table more compactly—using a single byte. The gain from this is small, since such a record still needs to be interpreted by the program.

16

The tachometer is implemented in a much simpler way. It is essentially a bar that fills from left to right depending on engine speed. The bar itself is pre-drawn, and only its visible length is changed during rendering. This approach is extremely economical in terms of both memory and CPU time.

The update of the tachometer, like that of the speedometer needle, does not occur every frame, but only when the displayed value changes. This is more than enough for a smooth visual perception and significantly reduces the load on the engine.

17

The gear indicator is drawn using ordinary character graphics. Each digit is a separate symbol, which allows the indicator to be updated with minimal overhead. No animation is used here, since gear changes are already well perceived through sound and changes in acceleration.

From the very beginning, we deliberately avoided overloading the interface with animated elements. On the ZX Spectrum, every additional dynamic detail has a price, and in a racing game stable frame timing is far more important than visual excess.

18

Sound effects and music in the game are handled by a unified subsystem. Depending on the hardware configuration, sound can be output either through the standard beeper or via the AY-3-8910 sound chip.

Priority is always given to sound effects related to gameplay: engine noise, collisions, and other important events. Music can be temporarily muted or simplified so as not to interfere with them. In practice, this behavior is almost unnoticeable, but it allows the game to sound much richer overall.

19

The engine sound itself is generated procedurally. Its pitch depends on the current engine speed, while volume changes slightly depending on acceleration and deceleration. This creates a convincing illusion of a “living” engine without the need to store multiple sound samples.

Naturally, the sound is far from realistic in the modern sense, but within the limitations of the platform it performs its function perfectly and provides important feedback to the player.

20

Collision detection in the game is deliberately simplified. We do not calculate precise intersections of shapes; instead, a set of logical conditions is checked based on the position of the car relative to the road, roadside objects, and other vehicles.

This approach may seem primitive, but it is more than sufficient for a game of this type. Moreover, it allows collisions to be processed very quickly, which is critical at high speeds.

21

Opponents in the game are controlled by a very simple algorithm. Each opponent has a target speed and a permissible lateral deviation from the center of the road. Depending on the current situation, the AI slightly accelerates or decelerates and shifts left or right within the allowed corridor.

Despite its simplicity, this approach works surprisingly well. Opponents behave believably, rarely collide with each other, and create enough pressure on the player without resorting to cheating or rubber-banding.

22

When an opponent car leaves the visible area, it is not immediately removed from processing. Instead, it continues to exist logically for some time, which prevents sudden changes in race conditions and allows for smoother re-entries onto the screen.

This also simplifies the handling of overtakes and collisions near the screen edges, where purely visual logic would often lead to errors.

23

Traffic cars are implemented even more simply. They move at a fixed speed and strictly follow predefined lanes. Their main purpose is not competition, but creating additional obstacles and increasing the density of events on the road.

To save resources, traffic cars use fewer animation states and simpler collision logic. Nevertheless, their presence significantly affects the dynamics of the race, especially at high speeds.

24

The road itself is described by a compact set of parameters. These include curvature, elevation changes, width, and the placement of roadside objects.

All these parameters are packed very tightly and interpreted on the fly by the engine. This allows long and diverse tracks to be stored using a relatively small amount of memory.

25

Curves are implemented by gradually shifting the center of the road horizontally. The rate of this shift determines the sharpness of the turn.

Because the ZX Spectrum has no hardware support for perspective transformations, all such effects are achieved through careful precomputation and well-chosen constants.

26

Elevation changes are handled in a similar way. The horizon line is moved up or down, and the scaling of roadside objects is adjusted accordingly.

Small changes in elevation greatly affect the perception of speed and space, even though the actual calculations behind them are very simple.

27

Each level has its own set of parameters that define lighting, background selection, and the types of objects used.

This makes it possible to reuse the same engine code while creating locations that feel very different from each other.

28

The transition between levels is deliberately unobtrusive. Wherever possible, changes in scenery happen gradually, so as not to break immersion.

Sharp transitions are used only when they are justified from a gameplay or narrative point of view.

29

Loading between stages is hidden behind brief pauses and interface updates. No separate loading screens are used.

This was done both to save memory and to maintain a sense of continuous travel, which is important for the overall concept of the game.

30

A great deal of testing was devoted to balancing speed, visibility, and difficulty. Even small changes in parameters could dramatically affect the feel of the game.

Many values were adjusted repeatedly, sometimes returning to earlier variants, until the desired balance was achieved.

31

The system of penalties in the game is intentionally mild. Collisions and driving off the road do not immediately end the race, but they noticeably reduce speed and time, which is often more frustrating for the player than a simple restart.

This approach encourages careful driving without making the game overly punishing.

32

Checkpoints are placed in such a way as to maintain tension throughout the race. Their spacing was adjusted many times during testing, since even a small change could drastically affect the perceived difficulty.

In some cases, checkpoints also serve as visual landmarks, helping the player to better memorize the track.

33

The time system in the game is unified for all modes. Internally, time is counted in ticks, which are then converted into seconds and fractions thereof for display.

This makes it possible to reuse the same code for different competitions and simplifies comparisons of results.

34

Records and best times are stored in a compact form. Only the most essential data are kept, which allows several records to be saved without consuming excessive memory.

The presentation of records is intentionally minimalistic, in keeping with the overall style of the game.

35

The game contains several modes of competition. Although they share the same core mechanics, each mode emphasizes different aspects of driving.

Some modes reward precision and consistency, while others encourage risk and aggressive maneuvers.

36

In the time trial mode, the track is completely empty. This allows the player to focus entirely on driving technique and optimal trajectories.

It was this mode that proved to be the most useful during internal testing of the physics and controls.

37

The duel mode pits the player against a single opponent. In this case, the behavior of the opponent is tuned more carefully, since all attention is focused on this one rival.

Small adjustments in speed and positioning logic can significantly change the perceived difficulty of such races.

38

In races with a mass start, the density of cars on the track is much higher. This increases the number of collisions and unpredictable situations.

From a technical point of view, this mode is also the most demanding, since many objects must be processed simultaneously.

39

Orienteering mode differs from the others in that the player must make decisions at intersections. The correct route is not always obvious, and mistakes can be costly.

This mode required additional logic for handling intersections and route validation.

40

Real maps of Swedish regions were used as a basis for orienteering tracks. Of course, they were heavily simplified and adapted to the capabilities of the engine.

Nevertheless, they add a distinctive flavor to this mode and set it apart from typical racing game challenges.

41

During development, we paid special attention to input handling. The game supports several control schemes, including keyboard and joystick, and all of them had to feel equally responsive.

To achieve this, input is read at a fixed point in the frame and processed using a small amount of filtering, which helps to avoid jitter and accidental rapid changes.

42

Acceleration and braking are not binary actions. Their effect depends on the current speed, gear, and engine state.

This makes control slightly more complex, but also more expressive, allowing experienced players to better feel the car.

43

Steering sensitivity changes with speed. At low speeds, the car responds more sharply, while at high speeds steering becomes smoother and less aggressive.

This is essential for maintaining control during fast sections of the track and prevents sudden, unrealistic movements.

44

The physics model used in the game is intentionally abstract. It does not attempt to simulate real-world forces, but instead focuses on producing predictable and enjoyable behavior.

Many parameters were tuned by feel rather than by formula, based on extensive playtesting.

45

Crashes and off-road events are accompanied by brief visual and sound feedback. These effects are deliberately short so as not to interrupt the flow of the race.

The goal is to inform the player about a mistake, not to punish them excessively.

46

The game engine is built around a strict frame budget. Each subsystem is allocated a specific amount of time, and exceeding this budget is not allowed.

This discipline was necessary to ensure stable performance on all supported machines.

47

If some feature turned out to be too expensive in terms of CPU time, it was either simplified or removed entirely.

In several cases, visually attractive ideas had to be abandoned in favor of smoother gameplay.

48

Debugging tools were extremely limited. In most cases, errors had to be identified by observing visual glitches or incorrect behavior on screen.

This significantly slowed down development, but also encouraged writing very careful and defensive code.

49

A number of internal test modes were built into the game. They allowed individual subsystems to be exercised in isolation.

These modes were never intended for players, but they were invaluable during development.

50

As the project approached completion, more and more time was spent on polishing rather than adding new features.

Small improvements in timing, visuals, and sound often had a greater impact than large structural changes.