

Tork AGB – Technical Design Document

Copyright Realism Ltd 2002

Synopsis

This is the TDD (Technical Design Document) for the AGB version of Tork. It is designed to give the technical abilities and limitations of the AGB and their interaction with the Tork GDD (Game Design Document). This document should be read in conjunction with the GDD. Generally, the TDD will take precedence over the GDD. Realism has technical people designing the GDD, who happen to be 'hardcore' gamers, so divergence between the documents is unlikely to occur. At its simplest, the GDD says what will be done and the TDD says how it will be done.

The X-Box game

Tork the game, (also the name of the main character) is a full 3D game for X-Box with unlimited freedom of movement in all axes. The game is a mixture of action and puzzle solving with a high degree of offensive and defensive fighting. The main character can morph into other forms – three main and a few secondary, which can be sustained for varying amounts of time. Tork spends much of his time in the standard 'Tork' state. Tork is armed only with (a set of) bolas, which are used variously like oriental numchakas, midrange striking weapons and boomerang. There are four main time zone areas, which are subdivided into various numbers of sublevels (between three and seven) of differing styles. There is an underlying story to the game, which is told linearly, so that the player progresses through each time zone one after another. Some sublevels are linked so that the play order (of these linked sublevels) is irrelevant. The game is very fast, exquisitely rendered and large.

The AGB

The AGB has absolutely no hardware 3D rendering capability. The complexity and size of the 3D worlds of Tork make any attempt at 3D software rendering untenable. There are two immediately obvious alternatives, which both use the AGB hardware to separate degrees.

2D side scrolling

This is the most straightforward solution. It utilizes the sprite and scroll hardware fully. We envisage complex sections made of smaller pieces of data, which join seamlessly and permit enhanced AGB hardware effects, such as layer translucency, multiple parallax scrolling and even entire section rotation (game permitting). The benefits are simple level design, space efficient data storage and coding, relatively simple game, good hardware usage, fast (to code and run), low CPU usage, simple control system, simple collision detection and physics and little coding required. The downside is high degree of art skill required to reduce complex looking worlds to small character sets, limited gameplay and limited 'standard' puzzle sets.

Isometric pseudo 3D

This is seemingly a more complex solution. It certainly would require more creative effort, but the quality of looks and gameplay elements should make

this approach more appealing. This approach will still utilize the scroll hardware fully. Sprites although used extensively will also require the addition of software occlusion clipping, which is more processor intensive. The advantages of this system are that it is pseudo 3D, so 3D game elements translate more directly, worlds look better. Level design can be kept relatively simple, as can collision detection and physics (although both are more complex than for 2D). The downside is more difficult to program the core technology, runs slower but easily capable of sustaining 60fps, the graphics are substantially larger and initial tile design is more complex.

The Approach

So, *which system do we use?* The answer would appear to be simple, 2D, but lack of time constraints allow for a more ambitious product realization. Most of the main game will therefore use isometric technology. Two alternative pseudo 3D technologies will also be used for some sections (sublevels) of the main game. These are discussed later.

Isometric Technology

Given that a game design already exists and is 3D, using a pseudo 3D approach allows us to modify the game elements more marginally, allowing us to concentrate a little more on the core technology. The redesign of Tork to a 2D side scrolling game that still works as Tork will take longer than the saving to be made by developing a slightly simpler core technology. Either way, the design of artwork for the levels is a relatively awkward task.

Pre-existing artwork

We will be able to use much of the pre-existing artwork for a pseudo 3D approach. In any event, all of the 3D character models and textures will be required for rendering out as sprites for either side view 2D or isometric view 3D. The 3D world geometry can be used as a simple template for the layout of a tile based pseudo 3D world. The 3D world textures can be used as the basis for simpler tile textures. None of the world geometry or texturing is much use for a 2D scroller. It will be **essential** for us **to obtain existing X-Box artwork**.

The main justification for isometry

At this time, isometric games haven't been fully explored on the AGB. 2D side scrollers have been 'done to death'. To achieve the 'wow' factor necessary using 2D means that we have to push the technology a long way, certainly to the standards of Donkey Kong Country (SNES) and then become more contemporary by utilizing the extra hardware of the AGB. This isn't a problem for Realism per se, but this effort could be expended more profitably by developing a technology more closely suited to Tork. The best comparative product that we envisage Tork 'working like' is Sonic 3D Blast (Megadrive/Genesis). Once again, this needs to be brought up to a more contemporary time frame by utilizing more texturing and softening the edges of the isometric axes. (By contrast, Super Monkey Ball deliberately emphasizes the 3D axes, so we could deliberately copy this style and Tork would look very like Sonic 3D Blast). Other games that have elements that are

relevant to our implementation of isometric 3D are:- Startcraft (PC), Spyro & Spyro 2 (AGB), Populous (all formats) and Cannon Fodder (all formats).

Main problem with contemporary isometry

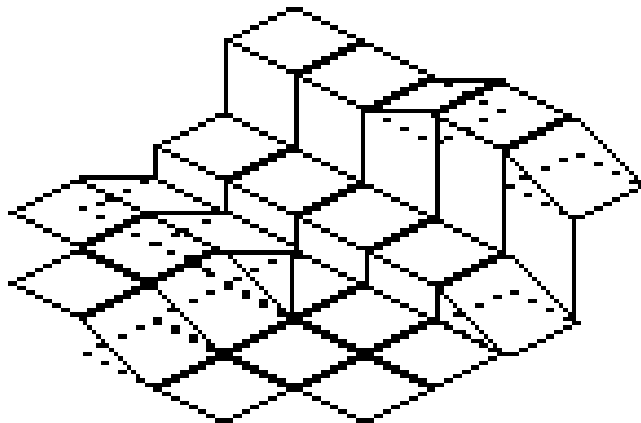
The main constraint of most isometric worlds that we see is the fact that they are mostly flat with a few vertical walls here and there (eg. Cannon Fodder). This is completely understating the ability of a true isometric system. Isometric views should encompass relatively arbitrary slope angles and directions as well as exploiting height differences all over the world. Populous is closer to this ideal.

Realism's solutions

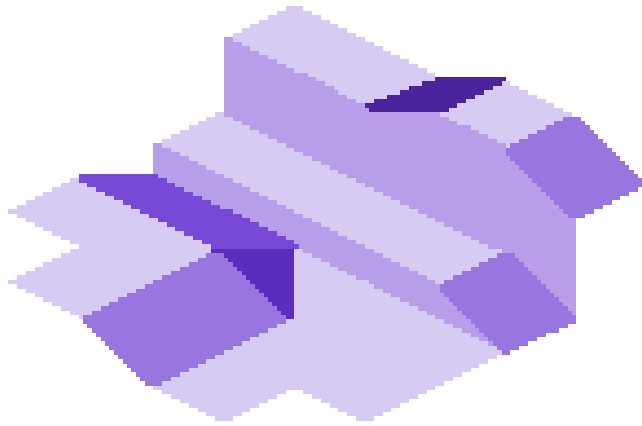
We have developed a core technology that allows us to take an isometric grid (just an array of diamonds with 2-to-1 slopes in each axis), then place a tile at any position, which has a wide variety of different vertex elevations. (Raising one vertex, creates a diagonal split and forms a 'corner' piece, raising two adjacent vertices equally creates a parallel 'slope', etc).



Furthermore, individual tiles can be raised to any elevation, creating steps, walls or cliffs.



Like Super Monkey Ball, we have initially developed a level editor, which works in a similar sort of way. Select a tile, place on grid, select groups of tiles, raise and lower them, etc). This creates the game playing world geometry. We now have a stage for the action, which looks like a colour coded (actually monochrome shaded) featureless terrain map.



We also extract a bitmap (for use as an artists template) and all manner of geometrical information about the world (for collision detection and physics mainly). The entire editor system is quite independent of the game and represents 'generic' isometric technology.

Layer art processes

Artists take the geometrical description of the world (a bitmap image) and use this as the background layer of the world in Photoshop (or similar package). The surfaces are then painted over with standard characters (8X8 pixel squares). Character repetition is encouraged but the engine technology has been developed to un-constrain the artists from the hardware limitations. Further layers are added in a similar way, up to a maximum of 3 layers. Most of the game occurs on the background layer, which as already describes is a true isometric geometry complete with height map, so that it looks 3D and plays as though it is 3D (it's both 3D and 2D at the same time). Some elements of the game are placed on either of two foreground layers (to avoid confusion from hereon, midground and foreground layers). The editor handles the additional layers and merge the layers' characters into a single character set. Parts of the game design will require that parts of the map be cut between the 3 layers. The video mode that we will be using will have a fourth and final overlay layer, which will be used for special effects, scores, information etc.

Video Mode

The video mode we use allows us to use four layers (as previously indicated) and allows us to use 1024 characters between those four layers. Generally, the background layer will use most characters and the midground and foreground layers will use fewer characters (probably in the order of a hundred or so – at any one time). The overlay layer is likely to need most of a character set, which may need to be loaded dynamically to save VRAM space.

Core isometric engine

The entire engine technology revolves around dynamic VRAM allocation (so allocating characters in VRAM dynamically won't present any special problems). The core technology retains a count (in external RAM) of the references to a hires character (in VRAM) by the current character map (also

in VRAM). When these counters hit zero, the character's VRAM position is stored into a 'free list' buffer (and is essentially discarded although stays in VRAM). When a new character is required in VRAM (i.e. it has a reference count of 0 and is now needed), a free list element is grabbed and a new character DMA'd into VRAM. Worst case, 50 characters per frame need DMA'ing into VRAM (generally every few frames, unless scroll speeds reach 8 pixels per frame). We now have no limits on characters, apart from 1024 in VRAM at any one time (difficult but theoretically possible to achieve since the entire screen only displays 600 characters, (but there are four layers). There is a lot of ROM space available for storing DMAable characters at 64 bytes per character. All characters are 256 colours. Rough calculations lead us to think that the entire character set for backgrounds for every level can be as high as 20-30,000 characters in total. This would be in the form of uncompressed characters in ROM and equates directly to between 1.2Mbyte to nearly 2Mbyte of data. The actual character split can be quite variable, so for 10 styles (unlikely to be attainable), there would be an average of 3000 characters per style. Equally, the character set can be split, so that there is a base set of characters (say 10,000), which are common to every style, with the rest being split as unique characters between each style. Finally, the palette can be split differently as well, either with one palette for all the characters, one palette for every style or a combined split palette, with fixed entries for the common style characters and variable entries for the unique style characters. Although having many characters sounds useful, there is a tradeoff between the characters against both the amount of levels that can be fitted into ROM and the number of styles of levels.

Isometric Engine Efficiency

We are allowing for the possibility of scrolling up down or left right by up to 8 pixels per frame and running the game at 60fps. This gives a fully sustainable scroll rate of 480 pixels per second, which translates to a full sideways screen scroll in half a second. This sounds very fast – and indeed it is. This is a nominal scroll speed limit we've imposed for technical reasons. It may be that Tork sometimes uses this scroll speed ability. Note that generally, vertical scroll speeds are at half the horizontal scroll speeds so that velocities work in parallel to the 2-to-1 isometric axes. The worst case scenario would be to scroll at 8 pixels per frame in both axes, which needs 50 characters updated per frame (20 on a side and 30 on top or bottom). Again, in the worst case, all 50 of these characters would need loading in to VRAM. The core engine is timed at 6% of CPU time in this extreme case. In the normal instance, a 'slow' scroll will be by two pixels horizontally (usually matched with one pixel scroll vertically at the same time). This will actually not require any DMA for 4 frames, thus further reducing the CPU hit per frame. Contrast this with our new music driver, which (although very efficient) will take approximately 15% of CPU time. It should be pointed out that we have to consider the cumulative worst case scenarios as it is conceivable that everything takes as long as it can on some frames. To avoid frame dropout, the sum of all possible worst cases has to be less than 100% CPU usage!

Map sizes

We intend producing levels which are approximately fifty screens big. The ROM space requirements are based on this assumption. We consider that this is per sublevel and there will be sixteen sublevels (using isometric technology). It will be possible to use one larger map for multiple sublevels and ROM space limitations may force us to use one (larger) map for all the sublevels using the same graphical style. We will attempt to push background sizes considerably higher where they are the same style. Where sublevel styles vary considerably, separate character maps appear to be a more logical solution. We'd strive to make each level as large as it needs to be to incorporate as much of the original puzzle gameplay as possible from the X-Box version. Maps will be compressed in ROM and be decompressed dynamically in 256 character maps (16X16 character blocks). Even allowing for caching of several blocks in external RAM, this accounts for only a very few Kbytes of RAM. Block compression will save approximately 50% of map space. A 50 screen map will require typically 30K of data in ROM (when compressed). That is, 16 maps of 50 screens big will require 1/2Mbyte. Other descriptions of each map (such as the isometric height data) will use a similar amount of ROM space.

There are obvious tradeoffs between sublevel sizes, character set sizes and the amount of art resources available to us. Larger levels will require more artwork resources, which become increasingly more constrained by the finite character set limits imposed by ROM space. All sublevels of the same style have no option but to share a single genre character set. We do not envisage big problems arising from implementations based nominally around figures presented above. This is one massively scalable area that can permit us to alter the production timescale of Tork at ease.

Sprites

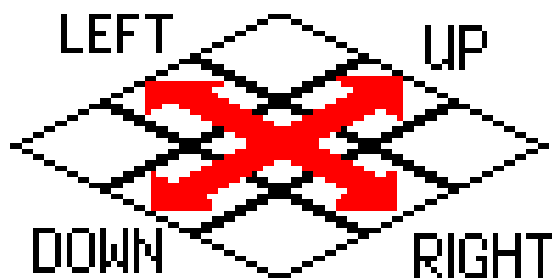
Sprites are subdivided to near optimally use as few hardware sprites as possible. Optimal mapping is computationally too expensive and produces only modest improvements (for disproportionate amounts of development CPU time). Sprites can have a priority, which places them at any depth relative to the 4 screen layers. Sprite priorities will be dynamically adjusted to allow them to pass in front and behind the mid and foreground layers. They'll always be on top of the background layer and probably always be behind the overlay layer. This priority change will be dependent on the apparent depth of a sprite within the scene. Simply put, isometric views have higher depth as you move up the screen. This is unfortunately similar to how a change in altitude appears and is one of the main causes of confusion in isometric games. Sprite shadows will be employed to give height cues to the player. When a player jumps, the sprite will move away from its shadow and then rejoin it upon landing. Sprite priority changes allows for a sprite orbiting an object, which is either part of another map or even just another sprite. Sprites orbiting one another is perfectly feasible, but involves different priority algorithms. This is any case trivial and will be employed most notably on the main character's weapon. Bolas can be seen in front of Tork and disappearing behind him.

The largest single problem with isometric games is that sprite animations have to be produced in multiple (isometric) directions. Quite simply, the main characters need at least 8 directions, which would imply 8 times as great a hit on ROM space than for 2D side scrolling sprite systems. The reality is, that in isometric games, only 5 directions are required (for an 8 direction sequence), since 3 of them are mirror images (or at least, can be) of each other. This reduces the memory consumption considerably. Keep in mind that 2D side scrollers may well use the same technique for left and right facing animations. Tork may well use 16 directions of key sequences (stand, walk, run and maybe jump), despite the control system only being capable of giving eight directions. Animations will move seamlessly through these extra directions as a player turns his character. This will improve the overall look of the main character.

Our analysis of the X-Box Tork design document and expected conversion to AGB gives the following data requirements for sprites:-

Control

We have set an isometric meter that defines the slope of the isometric axes at 2-to-1, which means that when following the isometric axes (by pressing up and down to move along the diagonal rightward axis and left or right to move along the diagonal leftward axis), any movement sideways of two pixels will also be countered with a one pixel vertical displacement. The player will also be able to move 'diagonally' by pressing more than one direction at once, which will translate naturally into movement directly up, down or across the screen.

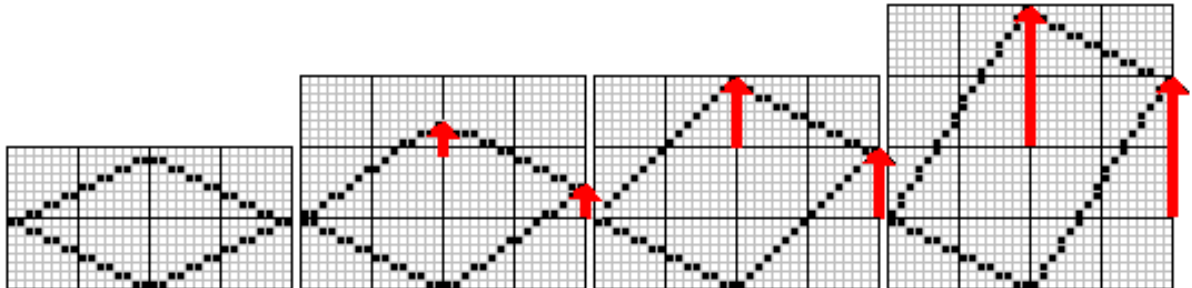


Essentially, it is as though the controls have been rotated round by 45 degrees (clockwise), then laid down onto the isometric plane.

Diamonds

Tile isometric tile 'diamonds' have a nominal area of 4 full characters (4 off, 8X8 pixels), but this diamond actually occupies 8 characters and is 4 characters wide and two characters (actually 15 pixels) high. Working in units of characters, we raise any vertex normally by one or two units (i.e. characters). The diamond then necessarily occupies more than 8 characters. It turns out that an elevation of 2 characters is unbelievably harsh (actually

producing 45 degree slopes), but this is a game and should be perfectly permissible. One unit raising of a vertex creates 30 degree slope (which although steep by skiing standards for example, actually creates good looking isometric slopes).



Even this reduced vertex elevation means that a rearward facing slope would not be visible to a viewer (it turns out to be exactly edge on to the viewer), so we have also implemented half unit raised vertices (i.e. 4 pixels). This now allows us to see rearward facing (gentle) slopes. It also gives us a very reasonable gamut of slope angles that allows us to have what appear like continuously variable smooth contoured surfaces. Note in the illustration above how some characters are the same despite the differing elevations of the tiles. This actually occurs all over isometric maps, although adjacent faces may end up being shaded with differing textures, so cancelling out many geometrically similar character optimizations.

Limitations

It's difficult to allow a landscape to occlude itself. The tools and core engine are perfectly happy to run with this type of data, but the view becomes confusing for a player. Landscapes are limited to a continuous fully viewable surface, comprising of surfaces that are flat, vertical or somewhere in between. The attributes of surfaces will be determined in the level editor, so for the most part the player character will not be able to move over vertical data – he could of course jump or fall off an edge. We will deliberately violate this limitation in other areas by putting landscape detail on either of the mid or foreground layers, but this use will be minimal so as not to overly confuse the player. It is however possible to see very gentle rearward facing surfaces as outlined earlier

Character sets for all the sublevels for any style will be restricted to about 2500 characters (some 4 times the screen area). With good art design and clever design and reuse of characters the slope angles and character constraints won't cause any apparent problems.

Scroll speed will be limited to 8 pixels per frame (480 pixels per second, which is equivalent to half second per screen) as previously discussed. In reality this limitation will never be apparent.

Isometric games are incredibly heavy on memory usage, for pretty much all of the artwork. Everything takes more room than an equivalent type of game using a 2D side scrolling view. This imposes limitations everywhere. Fortunately, an extra 4Mbyte of ROM (i.e. 8Mbyte in total) goes some way to redressing the problems associated with massive memory requirements. Main animations will be restricted to 16 directions. Most baddies will be restricted to 8 directions. Some animation will be restricted to 4 or less directions. Characters sets will be limited to a maximum of 30000 characters. Maps will be limited to no more than about 50 screens in size and will be used by on average 3 sublevels each. There is an obvious trade off between quantity of baddies and number of directions/animations of baddies, but there is enough memory to make usable compromises.

Other Main Game Technologies

Various sublevels of the main game will use two alternative technologies (instead of isometric views). The main reasons for this are that the game design requires something that although possible to achieve with isometry, could be better realized using technologies more closely aligned with the hardware. Also, isometric levels 'eat' ROM space, so changing technologies allows us to add sublevels at less ROM cost than staying with isometric sublevels. Given that isometry is a pseudo 3D technology, we thought it best to use other pseudo 3D technologies (as opposed to 'pure' 2D technologies) so that the overall feel of the game is one of harmonious technologies best suited to the various gameplay elements. The two technologies can best be described as 'Mode 7' Race Engine and Rotating Tower Engine.

Mode 7 Race Engine

This is either a 1st or 3rd person perspective 3D solution. We will always use it in 3rd party perspective, so that the player sees his character (i.e. Tork). The individual lines of the display are scaled using video hardware directly. This allows us to produce 'into-the-screen' perspective. The viewpoint can be yawed (rotated around a vertical axis), pitched (rotated about a horizontal axis) and even moved sideways or up and down, but not rolled (tilted sideways). Sprites will be scaled up and down as they come to the fore and recede into the distance. Sprites will be stored at one 'near closest approach' distance (i.e. not the closest distance) and scaled smaller (for the most part). Some scaling up will occur, but this will be limited to prevent blockiness occurring. We have produced a system where a track spline can be used to define the track, which can be relatively tortuous, certainly giving the impression of sweeping curves and hills. This engine will be used for all run, submarine and flying sequences. The nearest current equivalent technology is 'Space Harrier'.

This technology can use some of the isometric sprites from the main game (mostly the frames coming directly out of the screen (and possibly the frames either side of straight out). The sprites will generally be large enough to fit the constraints as outlined above (i.e. large, but not as large as they can get). Obviously, 'special' sprites for this level will be Tork and whatever transport he's using. These will be seen only from the rear and possibly one frame each

side (this is only two directions, since each side can be a reflection). Memory usage will be small relative to a full isometric level.

Rotating Tower Engine

This is a 2D technology, which looks pseudo 3D and in fact retains sprite sizes at any depth. It's actually quite similar to isometric technology in this respect. The entire backdrop is generated dynamically each frame. This is alleviated massively by only actually rendering a small portion and then duplicating this over the entire screen. This is in any case done trivially, by tiling the screen with repeating squares of characters, then just rendering the background column by column into one small set of tiles. All surface detail is then rendered using sprites, which will naturally have to have a variety of pre-rendered angles. We will set this up so that the same sprite data as used in the core Isometric Engine can be used. The result of this technology is a level that looks like a rotating tower (or column). The nearest direct equivalent technology is Nebulus.

The beauty of this system is that because the background characters are generated dynamically and the rest of the level shares the main game sprites, the overall memory requirements are minimal. Very few 'special' sprites will be required (mainly for platforms) and these will be required in very few directions (five when taking horizontal flips into account).

Control Systems in Alternative Engines

The alternative technologies will use more standardized control systems. Since the worlds look more 3D (than isometric 3D), the controls won't be rotated to sit along isometric axes. Basically, the controls will translate to simple 2D translations within the world. Although it is possible to use up and down as a movement into and out of the screen in the mode 7 engine with the use of a 'shift' button, this is more likely to be automated so as not to present too complex an interface to the player.

The game genre

Isometric engine technology lends itself well to collecting, puzzle solving and or fighting, so the actual game style will not unduly affect any of the systems that we have been putting in place. The mode 7 engine will lend itself well to the chase sequences and the rotating tower engine will work well for the volcano sequences.