Geospatial Texture Streaming From Slow Storage Devices

April 17th 2007
J.M.P. van Waveren

Abstract

A high speed pipeline is presented that allows streaming large amounts of geospatial texture data from a slow storage device such as a DVD player. Low level, aligned non-caching reads are used to maximize the throughput, and to avoid memory and file system cache pollution. The bandwidth requirements are significantly reduced with compression, and a texture database with an optimized layout is used to minimize seek times. Furthermore, multiple threads are implemented to improve the throughput, and to offload compression calculations to a separate CPU/core where available.

1. Introduction

Uniquely textured terrain requires large amounts of texture data that cannot all be stored in memory at the same time. The texture data needs to be streamed from some storage device into memory before it can be used for rendering. High fidelity rendering of a walk/run/drive-through over uniquely textured terrain, quickly requires a lot bandwidth to stream in the texture data.

Assume a view point is moving over flat terrain, and the texture detail is displayed in square layers around the view point. Each layer around the view point displays the same number of texels but a lower detail layer is twice the size in world units than the higher detail layer directly above it. To render the flat terrain at a resolution of 1024 x 768 or higher, a typical resolution of 2048 x 2048 texels is used for each square layer. If there are five square layers of 2048 x 2048 texels there are a total of 5 x 2048 x 2048 texels (about 21 Mega Texels) available for rendering at any time. As the view point moves, the layers have to be updated and essentially rows and columns of texels have to be updated at the side(s) of the square layers, in the direction in which the view point moves. In the worst case the view point moves along a diagonal and both a row and column of texels need to be updated. Assuming there is one texel per square inch on the highest detail layer, this amounts to 5610 texels that have to be updated per inch of movement along the diagonal.

The average walking speed for a human is about 3 miles per hour which equals 53 inches per second. At this speed about 0.3 mega texels have to be streamed per second to update the texture

detail around the view point. Someone running very fast can reach a speed of up to 20 miles per hour which equals 352 inches per second. At this speed about 2 mega texels have to be streamed per second. When driving a car at 80 miles per hour about 8 mega texels have to be streamed per second. These numbers are for rendering completely flat terrain without any elevation. For a terrain with hills and mountains the amount of texture detail that needs to be updated can be 1.5 times higher or more.

Arguably, when the view point is attached to a car going at 80 miles per hour, there is no need to render a unique texel per square inch. Rendering fewer texels may not be noticeable, or may be perceived as motion blur. However, if the car quickly comes to a stop it is desirable to immediately have a unique texel per inch visible on screen. If the streaming is not able to keep up while driving the car at high speed, the desired level of detail will not be available when the car comes to a quick stop and there will be a delay before the detail either pops in or fades in. This is perceived as a momentary loss of focus which is distracting and may cause the viewer to try to refocus his or her eyes.

Streaming large amounts of texture data from a slow storage device like a DVD player is not trivial. First of all, a good understanding of how the storage device works is required. Furthermore a variety of techniques have to be used to reduce storage and bandwidth requirements and to maximize the throughput.

2. Reading Data from DVD

The following focuses on streaming from DVD. However, when trying to improve the streaming performance, the same optimization strategies also apply to other storage devices. In particular, the throughput and block sizes may be different, but the same or similar optimizations can be used for streaming from any disk based storage device like a CD or a Blu-Ray.

The data on a DVD is stored as a continuous spiral just like on a CD. The data is laid out in sectors on this spiral where each sector can hold 2048 bytes. The sectors are grouped in 32 kB Error Correcting Code (ECC) blocks (16 sectors per block). The ECC is a code which is used to automatically detect and correct errors. A DVD player has to read and verify the whole block whenever any data is requested from an ECC block. As such the best throughput is achieved when reading whole ECC blocks with useful data at a time. Files on a DVD are aligned on sector boundaries. In other words if a file is smaller than the 2 kB sector size the rest of the sector is wasted. It is not a good idea to place many individual texture files on sector or ECC block boundaries with wasted space in-between. The wasted space in-between the files is equivalent to wasted bandwidth. Instead it is much better to use a single texture database file in which the texture data is stored as compact as possible. The texture database file should either be aligned on an ECC block boundary, or the offset to ECC block boundaries should be calculated to be able to read ECC block aligned.

A DVD drive typically has 128 kB of physical cache or more. For the best performance 128 kB is read from disk at a time in the form of four ECC blocks. The texture database file is opened with Operating System (OS) file caching disabled. (On Windows use CreateFile with FILE_FLAG_NO_BUFFERING). Non-caching reads are used to avoid memory and file system

cache pollution. On systems with tight memory constraints, an ever growing file cache due to streaming large amounts of geospatial texture data may cause other parts of the application data (for instance geometry) to be swapped out of memory, which can have a negative impact on performance. When streaming large amounts of data it is usually better for the application itself to cache only the necessary data. After all the application either knows or can anticipate the data access patterns much better than the OS. When reading whole 128 kB blocks at a time there may be some additional texture data read that is not immediately required. However, with a good database layout the 128 kB blocks contain as much useful texture data as possible. Furthermore, with an efficient caching system in place a minimal amount of memory is used to temporarily save any texture data that is not used immediately, such that, it does not need to be streamed from disk again later.

DVD players range from 1x speed typically up to 16x speed. An 1x speed DVD player has a maximum throughput of 11.08 Mbps (1.32 MB/s) and a 16x speed DVD player has a maximum throughput of 177.28 Mbps (21.13 MB/s). However, for some DVD players the maximum throughput is only achieved when reading data from the outer edge of the disk. The DVD disk diameter is 120 millimeter (mm) or 4.72 inches. The DVD data zone spans 34 mm starting at 24 mm from the center and extending to no more than 58 mm from the center. All sectors have equal length and some DVD players spin the disk at a constant angular velocity. As such, there can be up to a factor 2.4 difference in the number of sectors that pass the read head per second between the inner and outer edge of the data zone. This can results in a significant throughput difference based on where the data to be read is stored on the DVD. If there is other data on the DVD that is not streamed in real-time, then place this data close to the center of the disk. Place a texture database from which data is streamed in real-time as close as possible to the outer edge of a DVD.

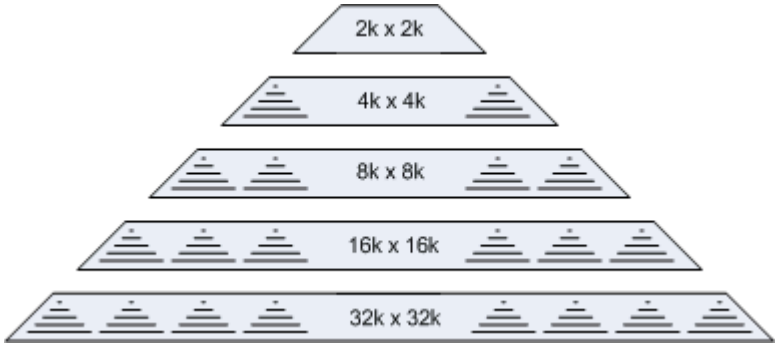
Although there are faster DVD drives out there, the average DVD seek time is typically 100 milliseconds or more. For short distance seeks the player can sometimes just skip several ECC blocks which is relatively fast. However, longer distance seeks, that require significant movement of the DVD player read head, are typically very time-consuming. It is very important to minimize seeking because any time spent seeking is equivalent to a waste of bandwidth. As such, it is important to design a texture database layout that minimizes the seeking required to load all texture data necessary for rendering any particular scene. It is typically more important to minimize the number of seeks than it is to minimize the seek distance. The seek time may grow as the read head has to move further, but any seek causes at least a significant base amount of potential bandwidth to be wasted because the read head has to refocus. A particular database layout in combination with reading 128 kB blocks at a time may result in data being read that is not useful. Just like seeking, this is also a waste of bandwidth. However, there is a balance between seeking, and potentially reading data that is not useful. In both cases bandwidth is wasted, but reading some percentage of useless data may avoid seeks, that typically result in a much more significant waste of bandwidth because the read head has to refocus.

Something much worse than seeking is a DVD layer switch when streaming from a dual-layer DVD. If possible it is best to not use a dual-layer DVD, but if the additional space is necessary an application should never read data from both layers of a dual-layer DVD while streaming texture data.

3. Texture Database Layout

To implement view-dependent texture level of detail, a hierarchical structure capable of representing different parts of the texture at different resolutions is required. There are many different approaches to rendering very large textures. However, pretty much all approaches use some form of mipmap chain or texture pyramid [5]. Mipmaps are pre-filtered collections of down-sampled textures that accompany a full resolution texture intended to reduce aliasing artifacts during rendering, by allowing quick access to filtered lower resolution versions of the texture. Each rendered pixel is derived from one or more texel samples taken from one or more levels of the mipmap hierarchy. In particular texel samples are taken from a mipmap level and it's immediate neighbors that have the closest 1:1 mapping to the rendered pixel on screen.

When dealing with very large textures the mipmap levels are typically broken up into many smaller tiles that can be streamed independently. Although the number of levels depends on the rendering solution, the individual tiles typically also have their own mip map chain to allow proper filtering on today's graphics hardware. The following image shows a texture pyramid for the situation sketched in the introduction.



While moving over the terrain the visible squares of texels around the view point are typically not updated with rows and/or columns of texels. Instead rows and/or columns of tiles are updated when the view point moves far enough in one direction. The order in which the tiles are stored on disk has a significant impact on the throughput when streaming the tiles based on the current, or changes to the view point. Optimizing the order in which the tiles are stored on disk can significantly reduce the amount of time wasted seeking from one tile to another.

The most commonly used structure for storing and ordering tiles is the quad-tree [6,7]. In a quad-tree structure a rectangular region is recursively subdivided into four uniform quadrants. The following figures show one layer of tiles in linear order and in quad-tree order. The position of each cell represents the spatial position of a tile and the number in the cell represents the tile offset on disk.

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

linear order

0	1	4	5	16	17	20	21
2	3	6	7	18	19	22	23
8	9	12	13	24	25	28	29
10	11	14	15	26	27	30	31
32	33	36	37	48	49	52	53
34	35	38	39	50	51	54	55
40	41	44	45	56	57	60	61
42	43	46	47	58	59	62	63

quad-tree order

With linear ordering of the tiles there is no seeking required between tiles when reading one row of tiles. However, when reading a column of tiles there is a long seek required for each tile. The seek distance between two tiles of a column also becomes larger as the texture size grows.

The quad-tree ordering minimizes the number of seeks and the seek distances. When reading exactly one row or one column of tiles by reading blocks of 4 contiguous tiles at a time there is one seek required for every two tiles of a row or column. In other words when reading blocks of 4 contiguous tiles there is no difference in the number of required seeks between reading a row or a column of tiles. Furthermore, when reading larger blocks at a time with many contiguous tiles the number of seeks is reduced significantly while reading as many useful tiles as possible. The structure is most efficient when reading a whole square of NxN tiles, for instance when the view point is instantly moved to a completely new location. However, the access also becomes more efficient when multiple rows and columns need to be read at one side of a visible square around the view point, for instance when for some reason the streaming was not able to keep up.

The visible squares of texels around the view point are all moving with the view point. In other words, tiles from different layers of the mipmap chain need to be streamed simultaneously, where the number of tiles streamed from one layer is twice the number of tiles streamed from the layer directly below. To reduce the amount of seeking between layers of the mipmap chain, the layers can be stored interleaved on disk.

0	5	20	25
10	15	30	35
40	45	60	65
50	55	70	75

layer N

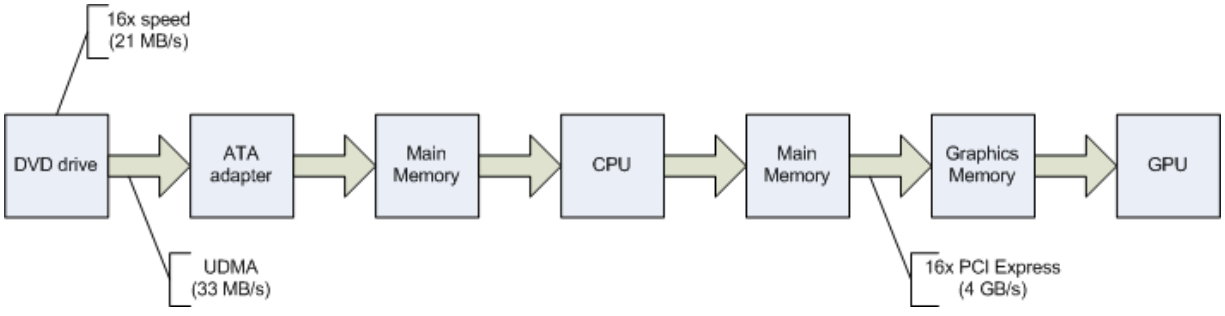
1	2	6	7	21	22	26	27
3	4	8	9	23	24	28	29
11	12	16	17	31	32	36	37
13	14	18	19	33	34	38	39
41	42	46	47	61	62	66	67
43	44	48	49	63	64	68	69
51	52	56	57	71	72	76	77
53	54	58	59	73	74	78	79

layer N+1

The above figures show how four tiles of a higher detail layer are stored with each tile of the layer directly below. In this example only the tiles from two layers are interleaved but the tiles for any number of layers can be stored in the same way.

4. Reducing Bandwidth and Storage Requirements

The figure below shows the streaming pipeline on a hardware level. The bandwidth of the hardware components may be different based on the system at hand. However, the relative bandwidth of the various components is typically different to the extent that the same optimization strategies apply to a variety of hardware combinations.



In this particular pipeline a 16x speed DVD drive is used. Ignoring seek times, the peak outer edge throughput of a 16x speed DVD drive is 21 MB per second. This is equivalent to a little over 7 MegaTexels per second (MT/s) of uncompressed RGB data. However, this throughput is never achieved in practice where a certain amount of seeking is unavoidable and bandwidth may

be wasted reading data that isn't useful. The DVD drive is connected to an ATA adapter through UDMA/33 which goes up to 33 MB/s. It's basically a clear ride once the DVD player delivers the data to the ATA adapter as the bandwidth is much larger further down the pipeline. The bandwidth between the CPU and memory is typically in the order of many GigaBytes per second (GB/s) and PCI-Express 16x goes up to 4 GB/s. The bandwidth between the GPU and the graphics memory is typically even higher.

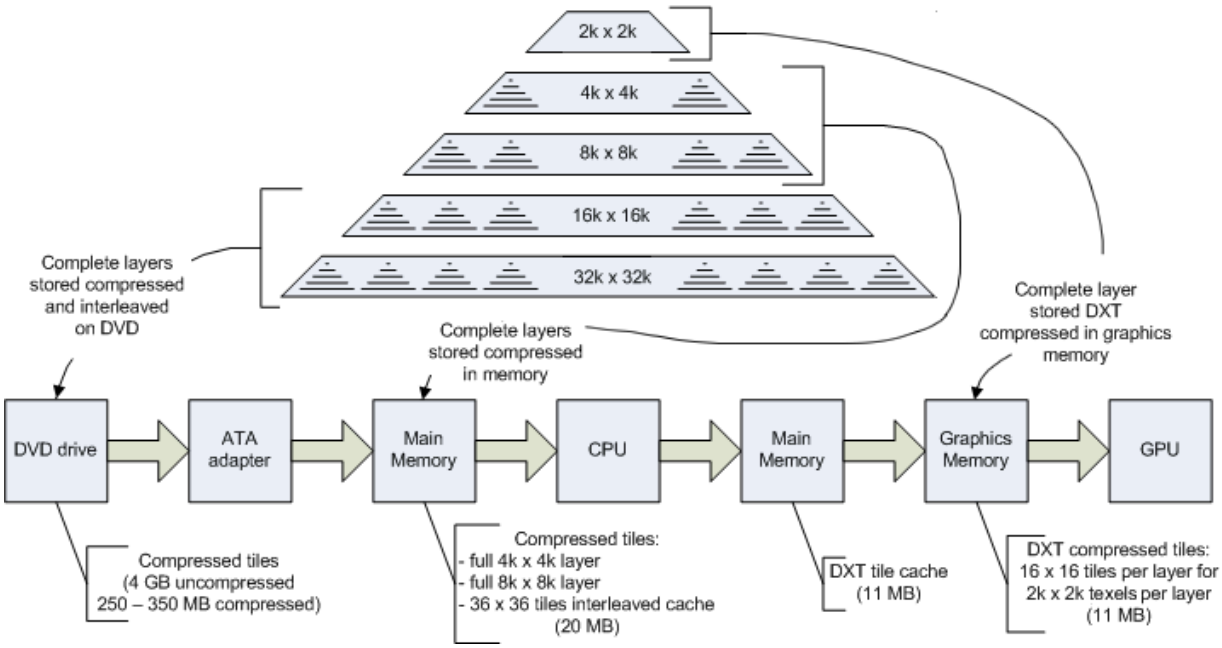
Compared to other components in the pipeline the DVD bandwidth is very low. The key to high performance streaming is to minimize bandwidth requirements at the bottlenecks in the pipeline by finding the right trade between data storage sizes, cache sizes and processing time.

The texture data is streamed from disk in the form of tiles. For proper filtering on today's graphics hardware the individual tiles typically need one or more mipmaps. To reduce the tile data streamed from DVD the mipmaps for the tiles can be generated on the fly. A simple box filter can be used to generate the mipmaps which is very fast and typically results in good quality. Using one core of an Intel Core 2 Extreme, mipmaps can be generated at speeds up to 400 MT/s.

The amount of data that needs to be streamed per tile can be reduced further by using compression. Many different compression algorithms can be used like JPEG, JPEG2000 and HD-Photo. However, the decompression needs to be very fast in order not to become a bottleneck in the pipeline. For a good trade between quality, compression ratios and decompression speeds a JPEG-like format can be used as described in [8]. The decompression from this format goes up to speeds of 190 MT/s using one core of an Intel Core 2 Extreme.

Although there is a lot of bandwidth available between the GPU and graphics memory the same texture data may be accessed many times during rasterization. As such it is still important to minimize the bandwidth requirements between the GPU and graphics memory. Most of today's graphics cards allow textures to be stored in a variety of compressed formats that are decompressed on the fly in hardware during rasterization. One such format which is supported by most graphics cards is S3TC also known as DXT compression. The reduced texture size increases rendering performance because there is less bandwidth required to fetch texture data from memory. Some quality may be lost due to the DXT compression. However, the reduced memory footprint allows higher resolution textures to be used such that there can be a significant gain in quality. As shown in [9] real-time compression to DXT format can be performed at rates of 200 MT/s using one core of an Intel Core 2 Extreme.

Some of the texture data can be cached at various points in the pipeline to significantly improve the streaming performance. The following figure shows the streaming pipeline in hardware with texture data being cached at various points in the pipeline.



The visible squares of texels around the view point are 2048 x 2048 texels. A relatively small tile size of 128 x 128 texels is used for optimal CPU cache usage (< 256 kB). In other words there are 16 x 16 tiles per visible square of texels. The tiles are stored with mipmaps, DXT compressed in graphics memory. There are 5 such squares up to a resolution of 32k x 32k and as such there is a total of 11 MB worth of DXT compressed tiles stored in graphics memory for rendering. The lowest detail layer is 2048 x 2048 texels which is the same size as a visible square of texels around the view point for that layer. As such the complete lowest detail layer is always available in graphics memory and only needs to go through the pipeline once at startup.

The small low detail layers are kept compressed in memory such that they do not have to be streamed from the DVD in real-time. The 4k x 4k layer is 48 MB without mipmaps and is stored at a 10:1 compression ratio which results in about 5 MB. The 8k x 8k layer is 192 MB without mipmaps which even at a 10:1 compression ratio consumes quite a bit of memory. To reduce memory usage, the 8k x 8k layer is stored as a luminance enhancement relative to the 4k x 4k layer. To decompress a tile from the 8k x 8k layer a quarter of a tile from the 4k x 4k layer is first scaled up 2x with a shifted bicubic filter. Next the relative luminance is decompressed and added to the luminance of the bicibuc up-filtered tile. Using one core of an Intel Core 2 Extreme, the 2x shifted bicubic upscale runs at a speed of 120 MT/s and the decompression of the luminance runs at a speed of 300 MT/s. Only storing a luminance enhancement for the 8k x 8k layer results in relative compression ratios over 20:1 and as such only about 9 MB is needed to store this layer in memory. With the luminance enhancement there is little loss of luminance detail but there may be some loss of chrominance detail. However, the texels from this lower detail layer are only visible at a distance at which the loss of chrominance detail is not or hardly noticeable.

The two highest detail layers are the only two layers that are streamed from the DVD. These layers are stored interleaved on disk to minimize the amount of time wasted seeking between tiles. With each tile from the second highest detail layer, 4 tiles from the highest detail layer are read from disk. However, only a quarter of the tiles read from the highest detail layer are needed

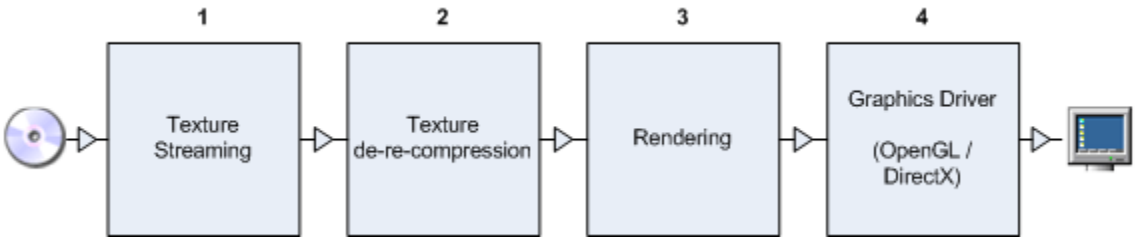
for rendering at any given time. Instead of dropping any tiles that are not immediately needed for rendering, they can be cached such that they don't have to be streamed from disk again as the view point moves. With each 16 x 16 tiles from the second highest detail layer, 32 x 32 tiles from the highest detail layer are read from disk. Only the center 16 x 16 tiles from these 32 x 32 tiles from the highest detail layer are immediately used for rendering, but all 32 x 32 tiles can be cached in compressed form. To avoid rubber banding as the cache is being re-used while the view point moves, the cache is large enough to hold 36 x 36 compressed tiles. These 36 x 36 tiles are kept compressed in memory with a 10:1 compression ratio or higher, which results in a maximum of about 6 MB worth of memory.

The end result is that only 11 MB of graphics memory is being used for rendering the large texture. There is another copy of the 11 MB worth of DXT compressed tiles in main memory, where the tiles are first updated as the view point moves, before being uploaded to the graphics card. Furthermore, there is a total of 20 MB worth of compressed tiles stored in memory to significantly improve the streaming performance.

5. Multi-Threading to Increase the Throughput

The texture streaming solution presented here reads data from disk which is then decompressed and re-compressed. If this process is serialized, the throughput is limited by the time it takes to complete all the steps in this pipeline. Even with a very fast de-compressor and compressor, the time spent in each step in the pipeline adds up quickly, and the throughput is typically not sufficient to stream in detail quickly enough for high fidelity rendering.

The texture database stores the unique texture in the form of many smaller tiles. As such, it is possible to run different steps from the texture streaming pipeline in parallel, by having each step work on a different tile independently. The following image shows the full pipeline broken up in four threads, going from streaming data from disk all the way to up to the graphics driver.



Current graphics drivers either cannot be orchestrated from multiple threads, or need to be synchronized first. As such, there is only one thread talking to the graphics driver, which is the renderer. The driver itself is typically also threaded

Threading the pipeline does not improve the latency for streaming individual tiles. However, breaking up the pipeline in threads does significantly improve the throughput when continuously streaming different tiles. The streaming, decompression and re-compression of the tiles can be broken up into two threads. With multi-threading the de-re-compression time is typically completely hidden by the time it takes to stream compressed data from disk, because the de-re-

compression is much faster than reading data from disk. While a new tile is being read from disk, the de-re-compression thread can decompress and recompress a tile that has already been read. Furthermore, the streaming thread does not do any work while it waits for the DVD drive, and as such, yields CPU time to the de-re-compression thread.

The streaming thread basically implements asynchronous reads. Memory and CPU usage can be slightly higher when not using asynchronous reads through the OS because data is being copied around. Instead of using a streaming thread, which itself can use asynchronous reads, the de-re-compression thread can also use asynchronous reads directly. However, using a separate thread allows the streaming to be easily throttled while the de-re-compression thread keeps crunching through tiles. The separate streaming thread also allows dynamic sorting of multiple data requests over time, such that seek times are minimized.

The de-compression, and re-compression could each run in a separate thread. However, current CPUs are fast enough such that the de-re-compression can run on a single CPU/core, without becoming the bottleneck in the pipeline. Furthermore, using multiple threads for the de-re-compression, does not decrease the overall CPU usage.

6. Results

Except for the lowest detail layer, which is always completely stored in memory, all tiles from the other layers need to be processed by the CPU. Tiles from the 4k x 4k layer, and tiles streamed from the DVD, are stored in a JPEG-like format. These tiles first need to be de-compressed, then mipmaps have to be generated, and finally the mipmap chain needs to be compressed to DXT format. The following table shows the steps a tile goes through before it is uploaded to graphics memory. The table also shows the input and output data per tile per step and the speed at which the texels can be generated or processed on a single core of an Intel Core 2 Extreme.

Regular tiles

step	input / tile	output / tile	MT/s
1. JPEG-like decompression	binary data	16384 texels	190
2. Mipmap generation	16384 texels	5461 texels	400
3. DXT compression of mipmap chain	21845 texels	binary data	200

If these steps are executed in sequence on a single core, this results in a throughput of 78 MT/s which is equal to 4782 tiles per second. A little bit more work is involved for tiles from the 8k x 8k layer. These tiles are stored as a luminance enhancement relative to tiles from the 4k x 4k layer. The following table shows the steps that are needed to produce a tile from the 8k x 8k layer before it is uploaded to graphics memory.

Luminance tiles

step	input / tile	output / tile	MT/s
1. JPEG-like decompression	binary data	16384 texels	190
2. Bicubic 2x upscale of quarter tile	4096 texels	16384 texels	120
3. Luminance decompression	binary data	16384 texels	300
4. Mipmap generation	16384 texels	5461 texels	400
5. DXT compression of mipmap chain	21845 texels	binary data	200

If these steps are executed in sequence on a single core, this results in a throughput of 55 MT/s which is equal to 3357 tiles per second.

Most tiles are processed at 78 MT/s, and only the tiles from the 8k x 8k layer are processed at 55 MT/s, but these tiles are updated less frequently (2 times less frequently than tiles from the 16k x 16k layer). Displaying the terrain at full detail while going 80 miles per hour, requires streaming in 8 MT/s worth of texture data. In other words only a little over 10% worth of CPU time of one core of an Intel Core 2 Extreme is used to process the texture data.

Most of the data is read from the two highest detail layers at 6 MT/s (= 275 tiles/s). At an average compression ratio of 14:1, without storing mipmaps per tile, this results in 1.25 MegaBytes worth of useful tiles that need to be streamed from the DVD per second. Furthermore, at the same compression ratio, there are about 30 to 40 tiles per 128 kB block read from the DVD. In practice about half the tiles streamed from DVD are dropped and as such wasted. This results in at least 15 to 20 useful tiles being read per seek.

7. References

1. 120 mm DVD - Read-Only Disk
   ECMA
   Standard ECMA-267, 3rd edition, April 2001
   Available Online: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-267.pdf
2. DVD FAQ
   Jim Taylor
   DVD Demystified, January 9th 2007
   Available Online: http://www.dvddemystified.com/dvdfaq.html
3. Optimizing DVD performance for Windows Games
   Microsoft Game Technology Group
   DirectX SDK, April 2006
   Available Online: http://msdn2.microsoft.com/en-us/library/bb147246.aspx
4. Accessing Multimedia Content on DVDs and CDROMs
   Rich Winterton
   Intel Software Network, July 18, 2006
   Available Online: http://www.intel.com/cd/ids/developer/asmo-na/eng/167202.htm?page=1
5. Pyramidal Parametrics
   Lance Williams
   Computer Graphics, Volume 17, No. 3. July 1983
   Available Online: http://www.cse.ucsc.edu/classes/cmps160/Fall05/papers/p1-williams.pdf
6. The quadtree and related hierarchical data structures
   Hanan Samet
   ACM Computer Surveys 16(2), pp. 187 - 260, June 1984
   Available Online: http://www.cs.umd.edu/~hjs/pubs/SameCSUR84.pdf
7. Hierarchical Spatial Data Structures
   Hanan Samet
   Proceedings of the first symposium on Design and implementation of large spatial databases, Santa Barbara, Ca, US, pp. 193 - 212, July 1989
   Available Online: http://www.cs.umd.edu/~hjs/pubs/SametSSD89.pdf
8. Real-Time Texture Streaming & Decompression
   J.M.P. van Waveren
   Intel Software Network, April 2007
   Available Online: http://softwarecommunity.intel.com/articles/eng/1221.htm
9. Real-Time DXT Compression
   J.M.P. van Waveren
   Intel Software Network, October 2006
   Available Online: http://www.intel.com/cd/ids/developer/asmo-na/eng/324337.htm