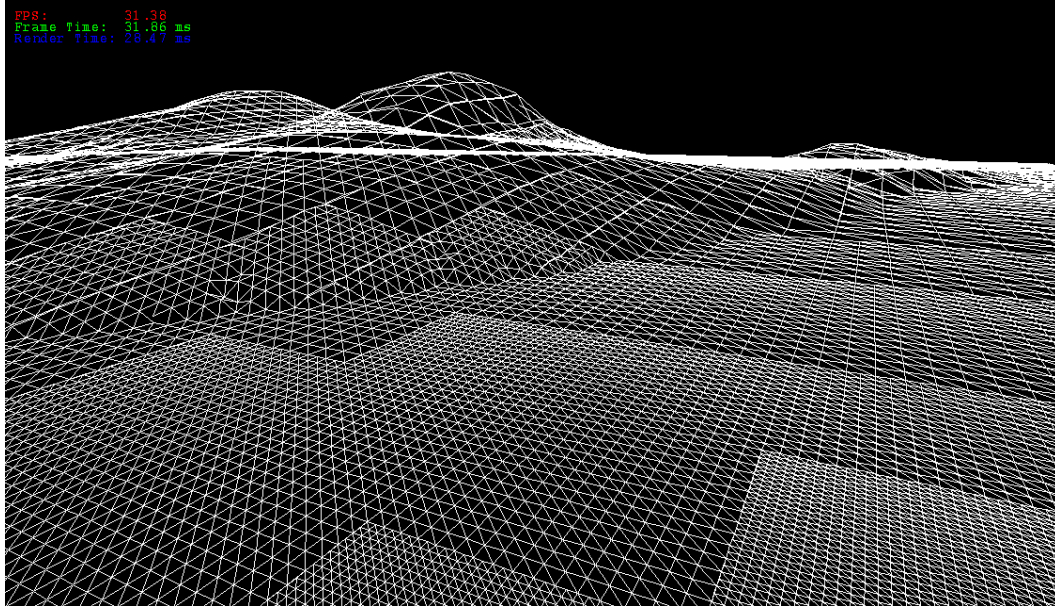


# Terrain Streaming on the Playstation<sup>®</sup> 3



A Project Presented to the Faculty of The Guildhall  
at Southern Methodist University

By Tyler Robertson

(B.S. Electrical & Computer Engineering, University of Missouri - Columbia,  
2007)

In Partial Fulfillment of the Requirements for a Masters of Interactive  
Technology in Digital Game Development with a Specialization in Software  
Development

May 11, 2009

To the Graduate Faculty:

I am submitting herewith a project written by Tyler Robertson entitled “Terrain Streaming on the Playstation® 3.” I recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Interactive Technology in Digital Game Development, with Specialization in Software Development.

---

Professor Gary Brubaker, Supervisor

I have read this Project  
and recommend its acceptance:

---

Professor Gary Brubaker, Advisor

---

Professor Wouter van Oortmerssen, Reader

---

Professor Jeff Wofford, Reader

Accepted for the Faculty:

---

Dr. Peter Raad, Executive Director  
The Guildhall at SMU

## ACKNOWLEDGEMENTS

This research project would not have been possible without the support of many people. I would like to thank my friends and family for their unwavering support in all my decisions and who have always been there for me through good times and bad. I would also like to thank my supervisor Professor Gary Brubaker for his support and advice in all stages of this project and paper.

Tyler Robertson

M.I.T., The Guildhall at SMU, 2009

Terrain Streaming on the Playstation® 3

Supervisor: Professor Gary Brubaker

Master of Interactive Technology degree conferred December 12, 2008

Thesis / Project completed May 11, 2009

This project explores issues that come along with streaming heightmap data from the DVD on the Playstation 3. By maximizing streaming on a console, games can minimize load-times and avoid lengthy pre-installs. This project looks at two different ways terrain data can be laid out on the disc, two methods for reading level-of-detail data and three algorithms to schedule read requests. The end result is a set of systems that can support camera velocities of up to 108,000 km/hr while flying over a heightmap of the Big Island of Hawaii with 30 meter resolution.

Because Sony does not allow access to the graphics card in Linux, a software wireframe renderer needed to be created so that the terrain could be visualized on the screen. This project uses two Synergetic Processing Units in pipeline to transform, clip, and rasterize lines. The software renderer is able to draw approximately 1 million triangles per second at a 1280 x 720 resolution.

The project converts a standard binary terrain file into a chunked format. During this preprocessing step the terrain is split into chunks and level-of-detail data is generated. The result is a single file holding two copies of the terrain data. One copy has columns contiguous in the file and the other copy lays chunks out according to a space-filling Hilbert curve.

An input system reads data from the DVD asynchronously and the terrain system takes care of requesting terrain chunks and recycling vertex and index buffers. The systems support changing algorithms at run-time so the user can switch data layout, level-of-detail source and request scheduling on the fly.

For data collection, a camera travels a pre-recorded path so that the same chunks of terrain would be requested. Several statistics were written to file and the results compared. In general the configuration which provided the best results was a data layout using Hilbert curves, a level-of-detail source of per-level streams and a scanning scheduling algorithm.

# Table of Contents

List of Figures .....	vi
List of Tables .....	vii
List of Tables .....	vii
Nomenclature .....	viii
Nomenclature .....	ix
Chapter 1 : Introduction .....	1
Chapter 2 : Field Review .....	2
Chapter 3 : Methodology .....	5
The Product .....	5
3:1 Asynchronous Input System .....	5
3:1:1 Optical Media .....	6
3:1:2 Scheduling Policies .....	7
3:1:3 Architecture .....	8
3:2 Terrain System .....	9
3:2:1 Data Layouts .....	9
3:2:2 Compression .....	12
3:2:3 Level of Detail .....	12
3:2:4 Data Layout on Disc .....	14
3:2:5 Managing the Terrain Chunks .....	14
3:3 Rendering System .....	15
3:3:1 Architecture .....	15
3:3:2 Managing the SPE Stages .....	16
3:3:3 The Transform and Clip Stage .....	17
3:3:4 The Rasterization Stage .....	18
Chapter 4 : Results and Analysis .....	20
4:1 Data Collection .....	20
4:1:1 Source Data .....	20
4:1:2 Testing Method .....	21
4:1:3 Information Gathered .....	22
4:2 Results and Analysis .....	24
4:2:1 Raw DVD Performance .....	24
4:2:2 Streaming Methods .....	25
Chapter 5 : Conclusion .....	31
Articles .....	33
Games .....	33
Appendix .....	35

# List of Figures

Figure 2-1: SSX3 .....	2
Figure 2-2: Picture of geometry clipmaps .....	3
Figure 3-1: The typical layout of an 8x8 grid of terrain .....	10
Figure 3-2: a) The same terrain, but using Hilbert curves b) Shows how each type of quadrant splits into another four sets of quadrants. ....	11
Figure 3-3: Level-of-detail is computed logarithmically .....	13
Figure 3-4: Model of the rendering system.....	16
Figure 3-5: Shows how lines are packed for the rasterization stage.....	18
Figure 3-6: Shows how chunks of the frame buffer are transferred only as their referenced.	19
Figure 4-1: The path of the camera during each test .....	21
Figure 4-2: Throughput.....	24
Figure 4-3: DVD Access time.....	25
Figure 4-4: Offsets issued .....	30
Figure A-0-1: Number of reads.....	37
Figure A-0-2: Bytes read .....	37
Figure A-0-3: Time to complete .....	38
Figure A-0-4: Average throughput .....	38
Figure A-0-5: Seeks per second.....	39
Figure A-0-6: Average seek time.....	39
Figure A-0-7: Average distance the OPU traveled.....	40
Figure A-0-8: Total distance the OPU traveled .....	40
Figure A-0-9: Maximum distance the OPU traveled.....	41
Figure A-0-10: Percentage of reading vs. seeking.....	41

# List of Tables

Table 4-1: Variables tracked..... 23

# Nomenclature

## Definitions

**Access Time:** The sum of the three types of latency (rotation, seek and transfer) associated with reading data from a disk drive.

**Hilbert Curve:** A space-filling curve that maps two-dimensional space to one-dimensional space.

**Hypervisor:** The security mechanism on the PS3 to prevent unauthorized access to the hardware.

**Rotational Latency:** The delay incurred while waiting for the disk to rotate so the read head is at the starting angular position of the data.

**Seek Latency:** The delay incurred while waiting for the read head to move radially so that reading can begin.

**Transfer Latency:** The time it takes to transfer the data from the disk into memory.

## Acronyms

**CELL [BE]:** The Cell Broadband Engine. The single-chip multiprocessor on the PS3 with eight SPE's and one PPE.

**DMA:** Direct Memory Access. A fast way to transfer data between two memory locations without CPU monitoring.

**DVD:** Digital Versatile Disc. The optical media used for this project.

**EA:** Effective Address.

**LOD:** Level of Detail.

**LS:** The 256 KB memory (Local Store) on the SPE's.

**MFC:** Memory Flow Controller. Interface between the SPU and PPE.

**OPU:** Optical Pickup Unit

**OS:** Operating System

**PPE:** Power PC Processor Element. The main dual-core processor on the PS3.

**PS3:** Playstation<sup>®</sup> 3

**SIMD:** Single Instruction Multiple Data

# Nomenclature

**SPE:** Synergistic Processor Element. Contains the SPU, MFC, and LS.

**SPU:** Synergistic Processing Unit. Contains the execution units and local store.

# Chapter 1: Introduction

The Playstation® 3 is one of the most powerful next generation consoles on the market. It has a massive parallel architecture holding seven, 128-bit Synergistic Processing Elements (SPE) and a 64-bit dual core Power Processing Element (PPE). All processors have SIMD capabilities and all run at 3.2 GHz. Although this is a large amount of raw processing power, like all consoles, the transfer speed of the optical drive is still a common bottleneck when streaming data for games. This project focuses on streaming height map data from a DVD and explores some problems with current implementations as well as a way to overcome these problems.

The primary difficulty in streaming terrain is dealing with the limited bandwidth and access latency from the DVD. To overcome this developers have compressed the data to increase bandwidth and placed data together on the disc. Another difficulty with terrain rendering is displaying realistic terrains at real-time frame rates. To solve this problem developers use level-of-detail schemes to reduce the amount of data while sacrificing some visual detail in the distance. This project experimented with data placement and designed an input system that uses disc scheduling algorithms and compression to allow for a reading rate of over 43 M height values from disc per second. The terrain method is a chunk based level-of-detail system that, when combined with the input system, will support camera velocities up to 108,000 km/hr in a terrain map of 122.88 km x 122.88 km with 30 meter resolution. The system has a throughput of reading and rendering approximately 1 Million triangles/s without lagging behind the camera.

## Chapter 2: Field Review

There are many engineering problems to terrain streaming. One solution is to load the entire world in the beginning of the level. While this can be fine for small games, it is just simply impossible for most modern 3D games that have large worlds. In the past there have been two main approaches: blocking and non-blocking. One common blocking way is to separate the world into different “areas.” When the player crosses from one area into the next, the user is presented with a very short, two to three second loading screen and must wait until the new area loads. An example of this is seen in the game *Elder Scrolls IV: Oblivion* [Bethesda Game Studios. 2006]. Although the load time is short, this method breaks emersion and can be an annoyance. This method also confines the level designers to create the world as a group of small areas.



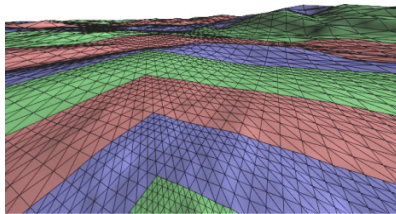
Figure 2-1: SSX3

The non-blocking streaming solutions branch out into two different methods. The first method uses an easy form of location prediction to load the necessary terrain data asynchronously. Typically designers use an invisible trigger or have some sort of environmental object block the viewer of the transition. In this way the data is visible when the player arrives at the predicted location. The classic example is the ubiquitous dog-leg tunnel seen in many first person shooters. This method is also common to several linear based games where location prediction is well defined such as in the snowboarding game *SSX3* [EA Canada. 2003].

The other method uses a distance from the camera to predict upcoming data. This allows players to roam freely such as in open world games like any MMOs or *Grand Theft*

*Auto IV* [Rockstar Games. 2008]. This lifts the burden of forcing designers to certain dimensions but can make engineering more difficult.

In all cases where content is streamed there can be issue of memory fragmentation. The constant cycle of allocating and de-allocating space for data means care must be taken to prevent both internal and external fragmentation when designing the streaming system. Some solutions use a large memory pool or have garbage collection. The game *Just Cause* [Avalanche Studios. 2006] solves this problem by using static buffers that are large enough for the biggest data chunk in the world. Additionally they are sector aligned. This means no memory needs to be dynamically allocated. Combining this with other design decisions have allowed their developers to enable the player to do everything from walking around on foot to flying jets through 32x32 km game environments, all using data streamed from the DVD [Lönn, Fredrik. 2006.].



**Figure 2-2: Picture of geometry clipmaps**

In both methods for streaming there are ways to increase performance. Typically some sort of level of detail (LOD) effect is calculated. This means chunks of terrain further away from you can display less detailed data, lowering the overall data that needs to be streamed. To get more perceived bandwidth, developers can compress the data in a preprocessing step resulting in more data transferred in the same amount of time. Once in memory the necessary data can be decompressed and displayed as normal. Ideally, the time required for decompression is smaller than the amount of time required to read the data. Hugues Hoppe of Microsoft Research has demonstrated great success with compression and reduced a height-map of the continental United States with a raw size of 40GB down to 396 MB using a compressed image pyramid [Hoppe. 2006].

An additional method for increasing efficiency is through visibility calculations. One common calculation is frustum culling. This method uses a spatial data structure – typically

a quad tree or binary space partitioning (BSP) – to ignore streaming data that is not visible from the camera. In worlds that are highly partitioned such as indoor first person shooter levels like *Unreal 3* [Epic Games. 2007.], one can go further and pre-calculate room to room visibility information. This can be used at run-time to quickly determine which rooms are visible from the current location. Other systems compute volume intersections and project the results to the ground, which then return the visible data.

There is also the question of how the original data is laid out on the source disk. On optical discs the most computationally expensive operation is seek latency – time to move the optical read head from one location to the next. With that in mind one may want to group neighboring chunks of terrain data physically near each other, and also possibly have duplicate data on the disk. Another solution stores everything in one large file that is opened at load time. When using this method, a table of contents is always kept in memory with offsets and size information. This keeps the system from wasting time to open files or query attribute information such as texture size or vertex normals.

On drives that spin with a constant angular velocity like the Playstation3 DVD drive, disc location can play a key role in data read performance. This means data on the outer edge can be read at a faster rate than data on the inside edge. Because of this, large files tend to be placed at the outer tracks to reduce load times and stream time.

# Chapter 3: Methodology

## The Product

This project experimented with two different methods for data layout, two different methods for reading level-of-detail and three methods for scheduling requests while streaming heightmap data from the DVD on the Playstation 3. The goal was to compare these configurations to determine which method provided the best results. The end product is a system which supports a camera flying over the Big Island of Hawaii, streaming the data from the DVD using one of two different methods for level-of-detail and also using one of two different methods for terrain layout. The method that provides the best results use a terrain layout that follow a Hilbert curve, use separate streams for each level of detail and use the shortest-seek-time-first scheduling algorithm. This configuration provides the highest throughput without wasting any data, the highest seeks/s and results in the ability to fly through the terrain at 108,000 km/hr without significant visual “pop-in”.

This project has been split into three sections. The input section details DVD storage and how to efficiently service requests for reads. The terrain system details my implementation of chunk-based level-of-detail. The rendering section discusses the architecture of the software renderer and how it will be able to handle the polygon draw rate requested by the terrain system.

### 3:1 Asynchronous Input System

One of the main goals of this project is maximize the optical bandwidth and determine what amount of detailed terrain can be streamed directly from the optical disc. The optical media brings with it many challenges to approach the advertised bandwidth. Knowledge of how data is stored and read from the optical disk can give the developer a significant performance gain for the entire project. This section discusses the details of the asynchronous data input system including optical issues, architectural design, and resource usage.

### 3:1:1 Optical Media

Although the Playstation<sup>®</sup> 3 (PS3) comes with Sony's Blu-ray Disc technology as its primary method of optical storage, the drive can also support reading of DVD media. DVD's are cheap to buy and the equipment to create custom discs are easily accessible today. Because of this, DVD's were used for the optical media in this project. The actual resulting performance will be different due to the differences in the DVD and Blu-ray drive properties, but the outcome of the results should remain true across all types of optical media.

The read speeds of media are referred to in multiples of 'X', which is different depending on the type of media. For a DVD the base read speed is 1,385,000 bytes/s (1.32 MB/s). The DVD drive spins its discs at a constant angular velocity (CAV). This means that the rotational speed is constant, no matter where the optical pickup unit (OPU) is located. This has the result that the surface of the disc on the outer edge moves faster than the inner edge, meaning in any given unit of time more data is accessible on the outside than inside. Thus seek times and transfer times increase, but the rotational latency is constant. The listed read speed of the DVD is the speed at its outer edge, and on the PS3 it is 8X [Playstation 3 Secrets]. So the theoretical limit of data transfer rates is 10.567 MB/s. The data in Figure 4-2 shows these facts.

Seek time is a common difficulty of efficient data access when reading data on any optical media. The average seek time for a typical DVD drive is about 160 ms [Pioneer. 2008], with a small percentage of that due to rotational latency. Assuming the OPU is reading from the middle of the disk, this is equivalent to losing approximately 1 MB<sup>1</sup> because no reading can be done while seeking. This potentially large loss of data shows that access to data needs to be controlled and organized in such a way to decrease the number of large seeks. For specifics of data layout for the terrain see section 3:2:1 Data Layouts.

Low level knowledge of how optical media operates can have an impact on the performance. The logical block of information on a disk is a sector and on the DVD is 2048 bytes. All files must be aligned on a sector boundary [Microsoft. 2006]. Thus a large amount of files smaller than 2KB will result in wasted space known as internal fragmentation. The best solution for this is to pack everything into one large file on the disc. This virtually

---

<sup>1</sup> The middle of the disk has a throughput of approximately 6.2 MB/s.  $0.160s * 6.2 MB/s = 0.992 MB$

eliminates all internal fragmentation and saves time wasted by having to open all the files. By including a small table of contents at the beginning of the disk, each file location is known. The table of contents can be kept in memory to avoid repeated seeks for file information.

The physical block size of a DVD is 64 KB and is known as the error correction code (ECC) block. Its purpose is to allow data to be rebuilt if there are small areas missing or was damaged. This means there are thirty-two 2 KB sectors spread across one ECC block. When data is accessed, the drive must read in the entire 64 KB into the drive's cache so that data recovery algorithms can be computed [Microsoft. 2006]. If data is referenced that is not inside the cache, the entire cache is flushed and the next ECC block read. Without knowing this, a user could run a naive shortest-seek-time-first algorithm on several read requests and waste 62 KB of data on every read.

Originally the data was to be read directly, meaning the data was transferred from the drive cache to the user space buffers, bypassing the OS buffers. However, the PS3 restricts this type of access to the DVD drive so all data read from the DVD is buffered through the OS. There are some benefits and drawbacks of this. The non-direct approach saves work in exchange for more memory and time for a double copy of all the data. Additionally, reading data already in the OS cache is much faster than reading the same data on the optical drive. However, in a console environment developers may want to have more control over what memory is used for caching. For this project, using the OS buffers has a slight performance hit on the system. Also care must be taken to avoid reading duplicate data so that measured read times more accurately reflect the optical disc speeds rather than hard drive speeds.

### **3:1:2 Scheduling Policies**

The following scheduling algorithms were implemented for the data input system. In each of the scheduling algorithms, any requests that are inside the ECC block are issued first.

*Shortest Seek Time First (SSTF)* – The next read request is one that is the closest to the current read location and thus incur the smallest seek delay. This policy can lead to data starvation given a heavy and random workload.

*First Come First Serve (FCFS)* – This is a simple scheduling policy and has been shown to result in poor performance [Orthington. 1994]. Although expected results were sub-optimal, this algorithm was implemented for comparison.

*Elevator (ELEV)* – This policy provides a lower response time variance than SSTF with only slightly higher average response time [Orthington. 1994]. The idea for this algorithm is to continue reads in a single direction, then change directions at the innermost and outermost requested offset.

### **3:1:3 Architecture**

Using the information presented above a system was created that will queue read requests, read them asynchronously and be able to switch scheduling algorithms on the fly. To eliminate the internal fragmentation and delay for opening files all the data will be packed into a single file.

The SSTF disc scheduling algorithm is the default policy for servicing read requests because it provides the best results for this project. Because the terrain does not quickly request the same data from the DVD, starvation associated with SSTF is not a problem. Other disc algorithms are implemented, and available if desired, but they are used for comparison only.

The asynchronous data system runs on a separate thread on the PowerPC Processing Element (PPE), the main processor. This is needed since the Linux kernel running on the PS3 does not support non-blocking reading from true files on disk. Communication between the read thread and the request thread is the selected scheduling queue. This is a typical producer-consumer problem and so a mutex was used to control access the given queue. The read thread transfers the raw data from the disc into a single buffer which is able to hold the largest possible data chunk. Once the read is completed, the raw data is DMA'd to the decompression SPE and the next request is issued. Before sending the chunk to be decompressed, time statistics are gathered for seek and transfer times.

The main data structure used across all systems was chunk pointers. This allowed each system to reference, and operate on, a single chunk. Originally a separate structure was used that contained the all the necessary information to complete requests. This meant, however, that much of this information would need to be duplicated for this to work. Doing

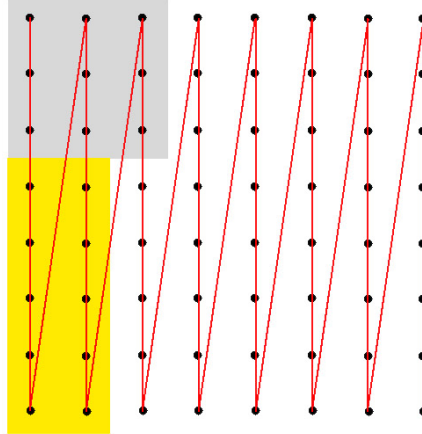
this would increase memory and also increase the CPU load from copying the data – however trivial it may be. Additionally, having the statistical data in a single location means easier synchronization and also the results are able to be used from other systems like the visual metrics display.

## 3:2 Terrain System

The terrain system uses a simple height map for the data to display on screen. Height maps usually have an odd number of values in either direction because displaying the data as an  $N \times N$  polygon grid requires having  $(N+1) \times (N+1)$  height values. The renderer cannot display the terrain in one large chunk however so the terrain is split into small chunks of  $33 \times 33$  vertices which make managing the terrain easier. The list of chunks is the main container for all the systems. This section discusses the file formats of the terrain used for this project.

### 3:2:1 Data Layouts

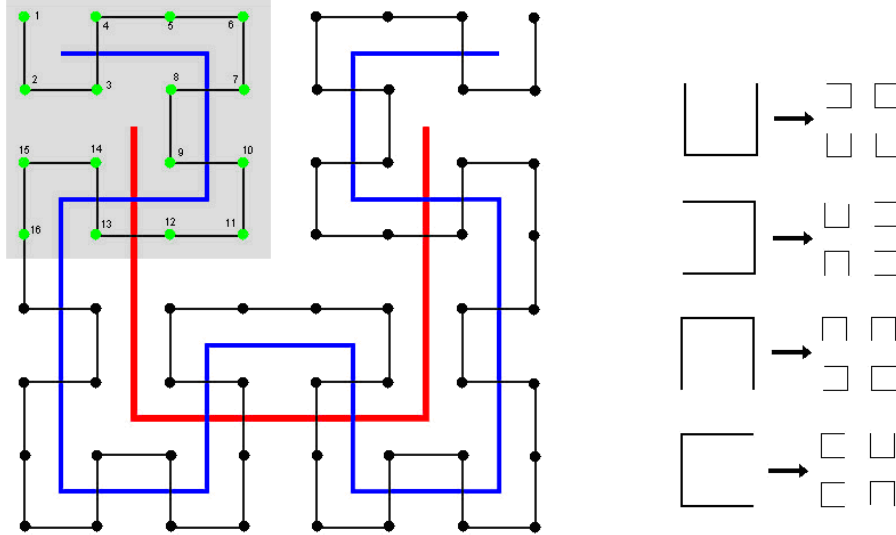
Managing the order of which chunks get transferred in can improve performance through minimizing seeks, however this is minimal when compared to how the terrain layout on the DVD. Typical data layout orders the vertices by columns, meaning column 2 is found directly after column 1 in memory. Since we know everything will be in chunks then we can gather all the vertices in a given chunk and place that entire chunk as one contiguous block of data on the disk. This repeats the edges of every chunk, but speeds things up a bit and makes it easier to process when reading in. Even if we gather all the vertices in the chunks together we still waste seeks at read time. Look at the figure below.



**Figure 3-1: The typical layout of an 8x8 grid of terrain**

The problem is when we want to read a given set of chunks around the camera. Let's say we want to read in the chunks in the grey area (the points are the centers of the chunk). If the OPU is at the top left corner then reading the first three chunks are fast. But in order to get to the next three chunks the OPU has to seek past all the chunks in the yellow, wasting time.

This is a problem of mapping a 2 dimensional space to a one dimensional space. There are numerous ways to do this by using, but a common way is to use the fractal space-filling Hilbert curve because it has better geometric locality-preserving behavior. Hilbert curves recursively divide 2D space into four quadrants so it can be thought of like a quad tree. In the Hilbert curve, however, the four sets of children of a given space have a different ordering depending on the ordering of the parent. It sounds very complex, but it is really quite simple.



**Figure 3-2: a) The same terrain, but using Hilbert curves b) Shows how each type of quadrant splits into another four sets of quadrants.**

The figure above shows terrain with 64 chunks, eight chunks on each side. The different colored lines represent different order of approximations to the curve. For a first order curve (the red line) the terrain is split into four squares with an order of top left, bottom left, bottom right, then top right. The second approximation (the blue line) splits each of those previous four areas into four more, but with a different order. The final line that connects all the chunks is the third order approximation.

Hilbert curves are only valid for N dimensions to 1 dimensional mapping, but the length of the source dimensions must be a power of two. To find the order of curve that fits a given grid one can use the simple equation

$$\text{Order Approximation} = \log_2 N$$

where N is the length of one side of the source dimension.

To see the advantage of using Hilbert curves we can look at the previous problem. This time when the chunks in the grey area are requested they all can be read without any extra seeks since they are next to each other on the curve. With Hilbert curves it can be shown that the distance between any two neighboring points is minimized along the curve, which turns into minimal seeks. In this project both methods were used to place data on the disc. A comparison of the two methods can be seen in the results section.

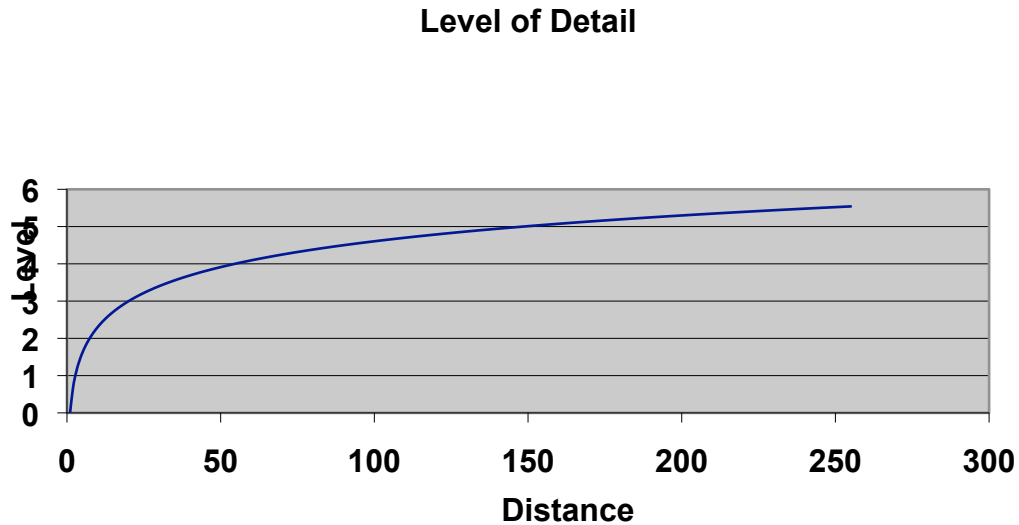
### **3:2:2 Compression**

To maximize the amount of data transferred from the DVD, each chunk is compressed before it is stored. This project used a simple delta compression. This works nicely because each vertex in the chunk of terrain will statistically have a similar height than its neighbor. Additionally the height values are quantized to minimize storage.

Because the height values are so large, each value is divided by 64 before any compression begins. For the compression, the minimum and maximum in the chunk are found and stored as the first two floats in the chunk on disc. The every height value is then quantized between these extremes with 16-bits giving 65536 possible values. This helps transfer speeds because reading smaller chunks requires less time and also reduces seek times since the chunks are closer together on disk. Compression is used for both types of terrain layout and both types of level-of-detail layout.

### **3:2:3 Level of Detail**

To reduce the amount of triangles rendered to the screen a simple distance based level-of-detail was used. The next level has half the amount of data than the level before it, stopping at a minimum of 5 by 5 vertices (see above for why the odd number). Since the terrain system is confined to a power of two, possible level of detail is guaranteed for levels greater 5 by 5 height values. The distance at which the levels are triggered to increase or decrease in resolution can be adjusted in real-time or altered in the config.cfg file. To ensure screen-space triangles sizes are uniform, each level's distance is not linear but rather a multiple of the natural log as shown by the graph below.



**Figure 3-3: Level-of-detail is computed logarithmically**

There are several options when reading in the displayed data in a level-of-detail system. The first option is to have a large file with every chunk at the highest resolution. This has the benefit of minimal storage on the DVD since there virtually no duplicated data. But this comes with a price of reading in the largest amount of data which increases transfer times. This works because each lowest level is a subset of its parent, so every level can still be generated, but a large majority of the transferred data must be thrown away. This is especially true since the majority of chunks in memory are at their lowest detail.

A second approach is to generate and store each LOD beforehand and only read the data that is needed. Each level's data is then appended to the level before it, much like a MIP map in texturing. In doing this no data is wasted, however the OPU must do a large amount of seeking to the end of the largest data set just to reach the other types of data. This is the opposite tradeoff than the approach given above. Since each level is half the dimensions of the previous, the size can be computed as the sum of a geometric series. The increase in storage size is  $1/3$ , again similar to MIP maps.

$$\text{Extra space needed} = \sum_{i=1}^{\infty} \frac{1}{4^i} = \frac{1}{4} + \frac{1}{16} + \frac{1}{256} + \dots = \frac{1}{3}$$

**Equation 3-1: The extra space needed for using per-level streams**

Like the data layout, both types of storage were used on the same data set. The results can be seen in the results section.

### **3:2:4 Data Layout on Disc**

The source terrain for this project is a binary terrain (.bt) file of the main island (“the Big Island”) of Hawaii. This data contains 2049 samples in each direction, each sample being 30 meters apart. This was not large enough virtual real estate for the project so it has been mirrored twice in each direction.

The source file stores these height values in column order. However in this project a chunk system was used for reading and rendering. Because of this a preprocess step was made to reorder the source file such that full chunks were contiguous in the file. Although the data was reordered into chunks, this project compared the performance of two ways to order the chunks either by column or by following the Hilbert curve. Additionally, two methods of reading in LOD data were compared. One using the highest level to produce all the sub-levels per frame, or have each level’s data pre-computed and in its own “stream.”

All of this is taken care of by a two pre-processed programs. One will mirror any binary terrain file a desired number of times, the result being a valid binary terrain file. The other program that takes as input a source binary terrain file and generates all the data needed for this project into a large, single, proprietary file.

- Table of Contents (Header)
- Hilbert Layout – Level 0 Chunks
- Hilbert Layout – Level 1 Chunks
- Hilbert Layout – Level 2 Chunks
- Hilbert Layout – Level 3 Chunks
- Hilbert Layout – Level 4 Chunks
- Hilbert Layout – Level 5 Chunks
- Standard Layout – Level 0 Chunks
- Standard Layout – Level 1 Chunks
- Standard Layout – Level 2 Chunks
- Standard Layout – Level 3 Chunks
- Standard Layout – Level 4 Chunks
- Standard Layout – Level 5 Chunks

### **3:2:5 Managing the Terrain Chunks**

Because the entire terrain file cannot fit into memory a system was created for managing the chunks of terrain that are in memory. The LOD system requires different levels of data depending on the distance from the camera, each level having one-fourth the data of the previous. This naturally fit into a bucket list of vertex buffers. Each bucket holding the

set of vertex buffers for that LOD. When a new chunk is requested, a new buffer is created and added to the list. Then the input system fills that buffer with the necessary height data.

In addition to this, another bucket list is used to maintain the “free” buffers in each level. When a visible chunk is no longer in range, the buffer is marked as “free” and can be reused when the next chunk of the same size is requested.

This system has two advantages. Since the chunks of each level are the same size, there is no memory fragmentation when reading new chunks. Also, the number of chunks is dynamic. With this system the minimum number of memory is allocated for all levels of data, and there is no cost of having to constantly allocate memory for new chunks.

## 3:3 Rendering System

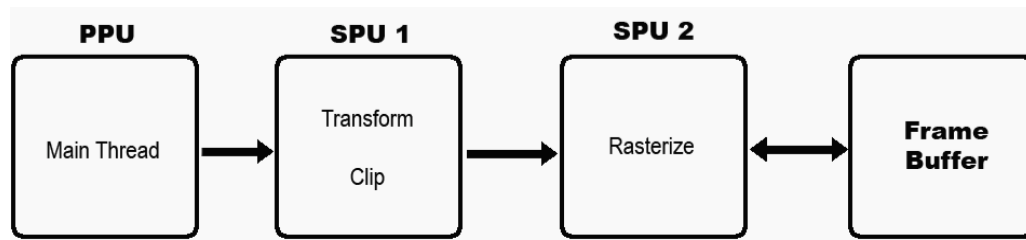
Sony has allowed Linux developers to access to most of the Cell B.E.<sup>2</sup> on the Playstation 3 through a modified Linux kernel. This allows developers to experiment with and learn how to program the cell processor. In order to hinder homebrew video games, Sony has denied direct access to the video hardware. Sony does, however, provide software access to a virtual frame buffer which can be used to draw pixels to the screen. In order to display the terrain data to the screen, all rendering must be done in software. Software rendering was a bottleneck in the system so the rendering system was designed to minimize this performance bottleneck. This section discusses the details of the rendering system.

### 3:3:1 Architecture

In an attempt to maximize performance in software rendering, this project used two SPE's setup in pipeline to mimic the graphics pipeline on standard GPU's. One SPE is used for transforming the vertices into screen space and clipping the lines on the screen. The second SPE takes the results of the first stage and rasterizes the lines and manages chunks of the frame buffer.

---

<sup>2</sup> Developers have access to 6 of the 8 SPE's. One SPE is disabled by the manufacturer to increase yield and the other SPE is used in protected mode by the kernel.



**Figure 3-4: Model of the rendering system.**

From the perspective of the main thread, there is only one SPE. Splitting the rendering process over two SPE's means that two things can be done in parallel, a set of lines can get rasterized while the next set of vertices get transformed and clipped. Additionally, the main thread can do useful work while waiting for the transform stage to complete. All this requires a way to ensure each stage works in lock step and no data gets overwritten. These mechanisms are discussed in the next section.

The rendering system can render polygons through vertex and index buffers and also support arbitrary transformations. When a user requests a vertex buffer with a given set of data the rendering system copies the data to a static buffer that is aligned on a 128-byte address. The alignment allows data to be cache aligned and allows for the fastest transfer to the transform stage via DMA. Also the original buffer can then be used again for other data. Once created, the user references the buffer via an ID.

### 3:3:2 Managing the SPE Stages

Since there is a high degree of parallelism and multiple threads executing at once, there needs to be a well designed system to control everything. The basic synchronization mechanism for all these actions is SPE signals and mailboxes. Both SPE signals and mailboxes are interfaces into the SPU's. What makes these good for synchronization is that they are fast and also halt the SPU when no messages or signals are present during a read request. This means the SPU can wait to execute further until receiving a message from an external source.

The PPE has built-in functions for sending messages and signals to a given SPE. However the goal was for the pipeline was to operate on its own. So managing each stage from the PPE was to be avoided. The solution was to map each SPE's problem state (mailbox registers, signal registers, etc.) to the main process' address space. Once each SPE's

variables have a defined effective address (EA), they can be DMA'd to every SPE in the pipeline. This allows each SPE to send signals and messages directly to any other SPE via a simple DMA. With this setup each stage in the pipeline can communicate and manage transfers with only the next and previous stages, alleviating the global management needed on the PPE. This is also how each stage knows the destination and source address to fetch and retrieve buffers from one stage to another.

### **3:3:3 The Transform and Clip Stage**

At the start of the rendering process the PPE requests a draw call via the user created vertex and index buffer. To alleviate the load on the main thread the DMA transfer is initiated from the transform SPE. Since the vertex buffer is static, there is no chance of getting unexpected data during the transfer. The chunk of data sent to the transform SPE is already in homogeneous form. This wastes 4 bytes of space, but is faster because it is already in the form to be transformed and a single point can be represented in a natural 128-bit vector data type.

This stage also holds matrices to transform the vertices into screen space, so a camera matrix, object matrix, and perspective matrix. This stage took advantage of the vector processing capabilities on the SPU when possible. This includes matrix-matrix multiplication and matrix-vector multiplication.

For clipping lines the Southerland-Hodgman clipping algorithm was used which calculates the clipped line by clipping the line on each frustum plane and accumulating the results. In order to speed up rendering zero-area and back-facing triangles were ignored. Both of these tests were computed with vertices in clipping space.

Finally this stage calculates the start and end positions of the lines to be rasterized in screen space. The output buffer is packed in a specific way so that the maximum amount of data can be sent to the rasterizer. The rasterizer expects the source and destination points of a line in two separate 8-element vectors, each vector containing four points consisting of an x and y position. See the figure below.

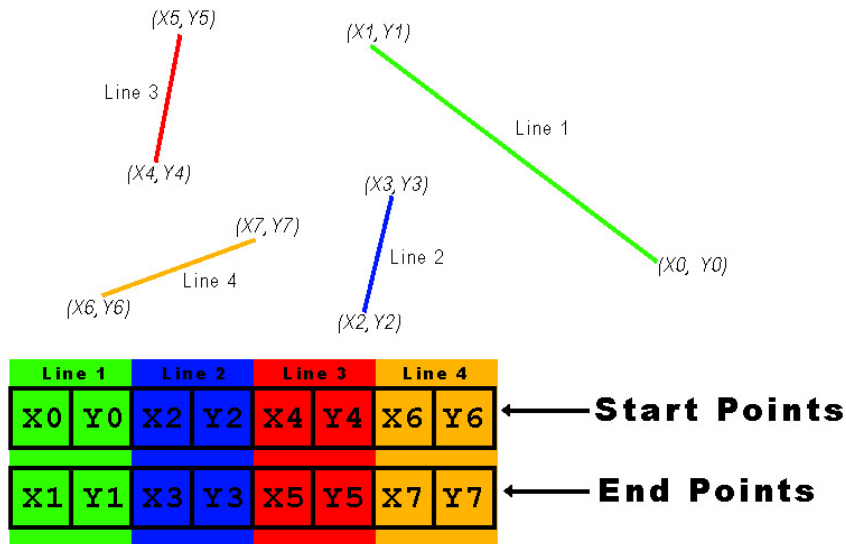
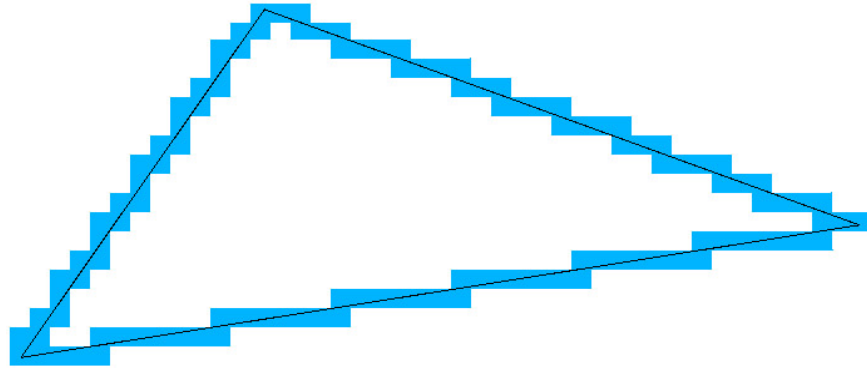


Figure 3-5: Shows how lines are packed for the rasterization stage

### 3:3:4 The Rasterization Stage

The slowest stage in the rendering pipeline is the rasterization stage. The biggest problem is that the SPE local store (LS) is only 256 KB for code and data while the single frame from the frame buffer is over 3.5 MB. To deal with this problem a system similar to a direct-mapped hardware cache was used. The frame buffer is split into chunks with 16x16 pixels in each chunk. Then frame buffer chunks are swapped in and out of the LS when needed. Having a level of indirection between the frame buffer and the line drawing algorithm meant code could assume the chunk is already in memory. If the chunk is not in memory, the paged system will evict the current chunk and DMA in the new chunk transparently. Below is a screenshot showing how the chunks around the line are actually brought into the LS. The black line triangle is the output line and the blue area is the portion of the frame buffer in the LS. By keeping the necessary chunks in LS the data needed to DMA is minimized and common reference areas stay in memory. Jack Breseham's line drawing algorithm was used for the rasterizing lines.



**Figure 3-6: Shows how chunks of the frame buffer are transferred only as their referenced.**

# Chapter 4: Results and Analysis

The goal of this project was to determine the best way to maximize performance when streaming terrain data from the DVD. As with many areas of engineering there is no absolute best way for everything in every situation. Each configuration has advantages and disadvantages for each property measured. For a mutex-locked scheduled queue, which this project uses, the configuration which provides the highest seeks per second is a scanning scheduling algorithm with a data layout of Hilbert curves and a level-of-detail method of per-level streams. This section discussed how testing was done, the data that was collected and an analysis of the different configuration highlighting advantages and disadvantages of each.

## 4:1 Data Collection

### 4:1:1 Source Data

As discussed in the methodologies section, differences in data layout on disc, how LOD was laid out and used, and types of disc scheduling queues had to be tested to get good results. The prediction was that the combination which provided the best performance would be to use Hilbert curves for data layout and to use a scheduling algorithm of shortest-seek-time first. The benefit of using per-level-of-detail streams was not predicted.

The terrain data used for the project is a 2k x 2k height map of the Big Island Hawaii. The 2k map was processed before the test to tile 2 times in each dimension as discussed in the File Format Section. Thus the final map size is 122.88 km x 122.88 km which is mapped to 4096 x 4096 units in the virtual world. After being compressed the total file size for the source data is approximately 95 MB (99682304 bytes). This large data set attempts to fill as much of the OS file IO buffers as possible and decreases the chance of cache hits in the OS buffers. By thrashing the OS cache on purpose, the transfer times more accurately reflect the true speed of the disc. An additional step to minimize caching effects was to physically eject the DVD before each test run.

Updating the terrain system is done with a constant frequency of 20 times a second on the same thread as the rendering. This includes querying the mouse and keyboard from the

user or reading the mouse/keyboard recordings during playback. A constant update time helps maintain an exact path when replaying recorded data ensuring the same data is requested across all configurations. The camera has a velocity of 1000 world units per second as to provide a good “load” on the all systems. This means the terrain cannot visually keep-up with the camera in any configuration. This camera velocity translates to approximately 108,000 km/hr when maintaining an approximately horizontal orientation.

### 4:1:2 Testing Method

With several combinations to test, there needed to be some common ground with which to compare all the results with. In this project the camera follows pre-recorded path over the terrain. Following this path from start to end is repeatable with the press of a button. This would ensure that although different algorithms or data layout was used, the same data was being requested and processed every time, making all variances a direct result of the variable change (eg. different data layout). Below is a top down view of the path taken by the camera. Initially the camera starts at a far enough distance in the negative direction on each axis so that on start-up the terrain is outside the view frustum and thus no terrain data is requested.

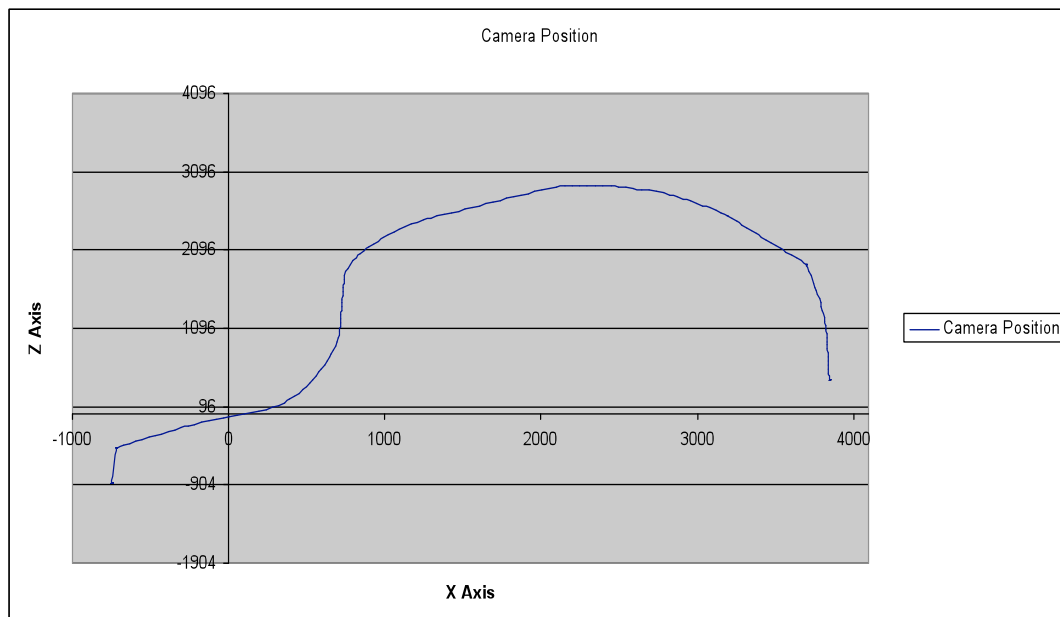


Figure 4-1: The path of the camera during each test

### **4:1:3 Information Gathered**

There are several things which are tracked during each run. Most is detailed information about disc performance such as seek time, read time, and distance the OPU has traveled. In addition to this, frame time is also tracked, as well as render time. These numbers are tracked each frame and are saved to the hard disk. The frame time, render time and read time can be visualized in real-time using a screen-space graph. Below is a list of the variables tracked. These variables are analyzed in Excel after the test and the results help to quantify how much better one configuration is when it is compared to another.

<b>Variables Tracked</b>	
Frame time / frames per second	With frame time, the total throughput on the renderer in terms of triangles per second can be measured. This is used to determine system load related metrics such as if the level-of-detail is up too high.
Render time	The render time is measured separately to determine how much of the frame was spent on rendering versus how much time was spent reading input, culling geometry, and inserting requests into the queue.
Total transfer time	This is the amount of time it takes to transfer a single chunk. The transfer time starts when the chunk is requested and ends when the chunk is in system memory. This does not include decompression time.
Total number of requests	This metric shows the number of requests processed during the simulation. A higher number is desired here. The number is a result of the scheduling algorithm efficiency, data layout and a product of terrain system implementation. See the results section for more details.
Seek time / seeks per second	This shows the average amount of time between each scheduled chunk. This is a combination of the scheduled queue efficiency and the data layout.
Average/maximum time in request queue	These times shows the general efficiency of the scheduling algorithm when compared to other scheduling algorithms given the same data layout.
Maximum time in request queue	This metric, like the average time in request queue, measures the efficiency of the scheduling algorithm.
Average/maximum/total distance the OPU traveled	Since the camera path is fixed, this metric should be relatively similar during each run. The differences show the efficiency of the data layout and the scheduling algorithm.
Average DVD read throughput	Changes in this metric show how differences in data layout and scheduling queue efficiency affect DVD throughput

Table 4-1: Variables tracked

## 4:2 Results and Analysis

### 4:2:1 Raw DVD Performance

Programs were written to profile the actual performance of the DVD drive in the Playstation 3. Since DVD's rotate at a constant rate the access time and throughput should increase as the read head moves at the outer section of the disc. The testing program samples 50 points along the disc<sup>3</sup>. After each sample, the media was ejected to ensure no caching is done. Below are the results.

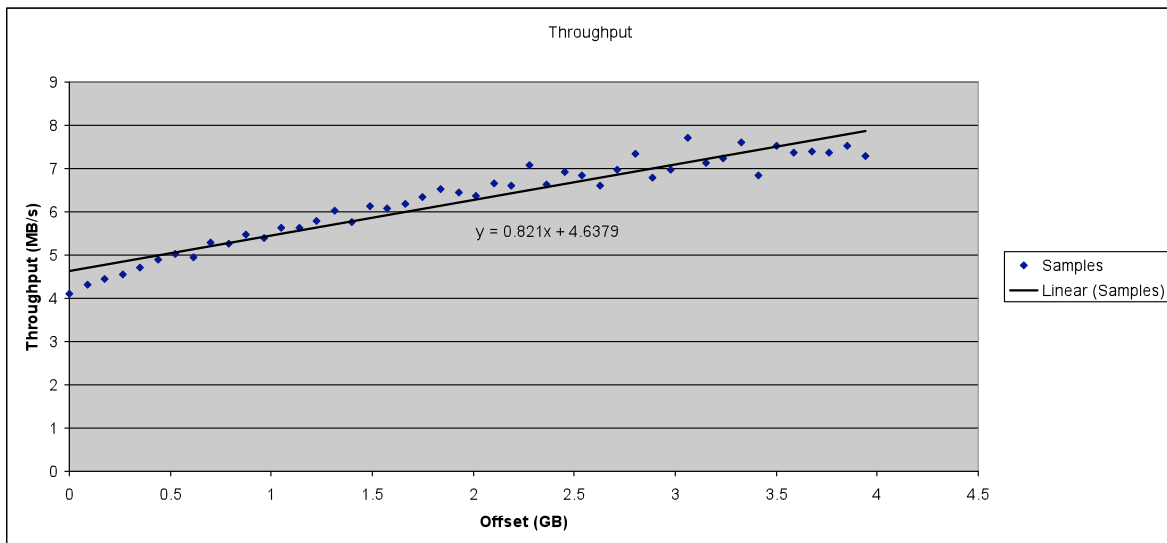


Figure 4-2: Throughput

The graph of the throughput shows an expected rate of transfer. Near the outer edge the throughput approaches a throughput of approximately 8.5 MB/s<sup>3</sup>. This is approximately 2 MB/s slower than the theoretical maximum (see 3:1:1 Optical Media). Also the inner track throughput should be roughly half of the outer track speed, which here remains true. There are three possible reasons for a slower read speed than expected. The DVD drives in the PS3 could be cheap and just don't spin as fast as the specification. Or the manufacturer

<sup>3</sup> The maximum valid offset on any file (even opening with the O\_LARGE flag) is  $2^{32}-1$ . This is an apparent security method in the Linux kernel for the PS3. Because of this only 47 out of the 50 samples were valid.

specification could also be overly optimistic. There is always the possibility of something more complex going on during the simulation.

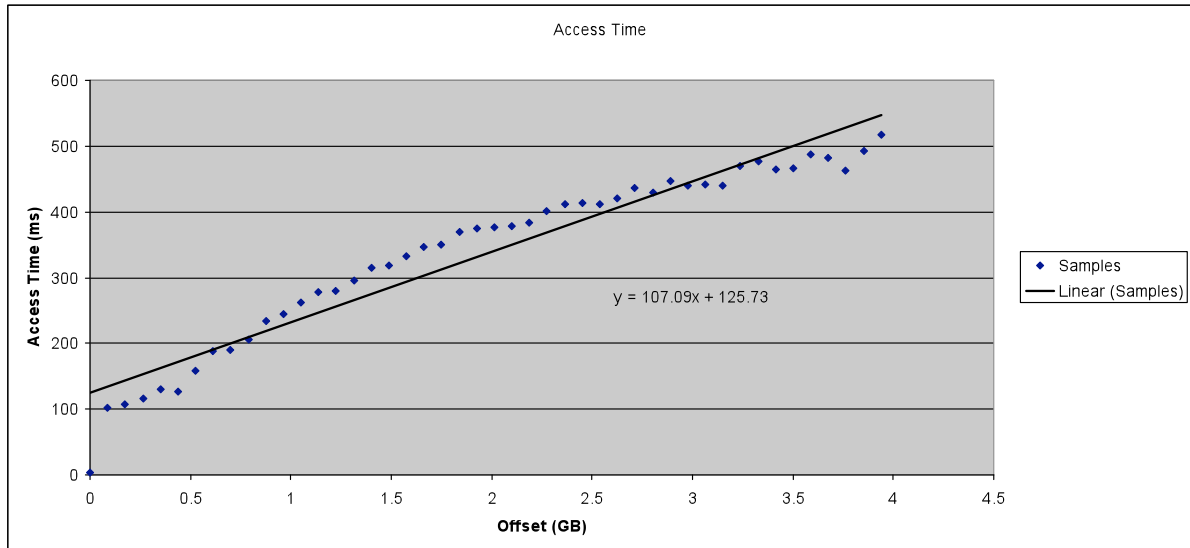


Figure 4-3: DVD Access time

As expected the access time increases as the OPU continues outward. The numbers, however, are not typical of a DVD. Access time involves two types of latency: seek and rotational. Seek latency is the amount of time needed to move the OPU to the desired track. Rotational latency is the time waiting for the disc to rotate to the needed sector.

The results show that the average access time is approximately 350 ms, over double the expected 160 ms. Also the average rotational latency for a typical 8X DVD drive (4600 RPM) is approximately 13 ms while the results show the PS3 average rotational latency is approximately 125 ms.

The results of the access time are much slower than expected, but the results of the throughput show that the drive does indeed spin at approximate 8X speeds. The conclusion is that the extra time found in the rotational latency is due to the time required to get the DVD to the maximum rotational speed. Subtracting the spin-up time results in an average seek time of approximately 200 ms.

### 4:2:2 Streaming Methods

Each type of scheduling algorithm was tested for each type of terrain layout and each type of LOD layout. The result is 12 different configurations that need to be compared. Each configuration was run 10 times and the average was used as the final result. Additionally, a

test was run to determine the performance of scheduling requests inside the ECC (See 3:1:1 Optical Media). The scheduled queue for this additional test was a minimum FIFO where iterating over each item to check if it lies within the ECC block was omitted. The final results can be seen in the Appendix.

Figure A-0-1 shows the number of requests made to the DVD. The first thing to notice is each configuration requests a different number of chunks, although each set of configurations follow a pattern. The differences in the number of requests can be explained by how the terrain system is implemented.

While the camera moves along the terrain it iterates through every chunk and makes several tests to determine what data, if any, needs to be requested and in what LOD. One of those tests is whether or not the chunk has already been requested (in any LOD). Thus a chunk will never be in the queue twice at two different resolutions. This was designed on purpose to reduce the complexity of the system and reduce the load on the queue.

Configurations that service chunks quickly will be able to pump more chunks at different resolutions through the system and thus have a higher number of requests. For configurations that are not able to keep up with the camera's speed, chunks are consistently only requested in the lowest LOD thus the number of requests is smaller.

Next we look at the amount of data read for each different configuration. This can be seen in Figure A-0-2. Using a stream for each level of detail drastically reduces the amount of data needed to be read from disk. Here, when using a single stream for all the LOD's means wasting, on average, approximately 95% of the data read in. Although reading less data seems more attractive for throughput and speed reasons, the next chart shows some interesting results.

The data in Figure A-0-3 shows the total time to service all the requests in each configuration. At first glance one can notice the FIFO algorithms take the longest amount of time. This was expected. But the other algorithms are largely the same across different configurations when compared to the FIFO schedule. This was not expected. The biggest surprise was the speed at which single stream and standard layout configurations performed compared to per-level and Hilbert layout. Without looking closely one can see they are close to the same even though the amount of data read in is quite a bit more.

Before continuing further, the results of scheduling for blocks inside the ECC chunk need to be discussed. The test for determining the advantages of ECC scheduling was the same as the others but with a different queue. The queue is the exact same as the FIFO queue, but without the ECC check. When looking at Figure A-0-3 it's best to compare the FIFO numbers with the FIFO-No-ECC numbers. The data shows in every case it is better to not issue requests inside the ECC block.

This unexpected result is a consequence of two things: OS caching and the terrain system architecture. The terrain system and the asynchronous input system operate separate threads, communicating by the shared request queue. The shared queue solves contention by a mutex and the entire queue is locked when executing any scheduling algorithm. The terrain system cannot continue issuing requests until the queue is unlocked. Thus the less complex the scheduling algorithm, the faster chunks can be inserted. To find chunks inside the ECC block means iterating over all the requests in the queue to issue any ECC chunks. In this case it hurts performance since the kernel buffers the area around the last requests and perhaps even does some amount of prediction. If the file were to be opened in direct mode one would expect to see a performance gain from this trick.

The main problem is from locking the entire queue for inserting and deleting. If developing in a Windows environment Critical Sections provide faster results than a mutex [Paquet. 2005]. Using a lockless queue would also increase performance as long as inserting, deleting, and searching occurred in a true lockless fashion.

Although the ECC trick hurts performance in this system that does not mean no scheduling should be done. Clearly the results show that the FIFO queue is the worst of them all and anything else will most likely perform better.

But looking at the details there are some things to note. In almost every case the Hilbert layout was faster than the standard layout<sup>4</sup>. The same is true for using per-level streams rather than one stream. Additionally, in all cases the ELEV queue proved to have the fastest time. With this in mind it comes as no surprise the fastest configuration was a combination of the above three facts: Hilbert curves, per-level streams and the ELEV scheduling algorithm.

---

<sup>4</sup> The standard layout had better results when a single stream was used with the ELEV scheduling algorithm.

The fact that the ELEV schedule provided faster times than the SSTF schedule came as a bit of a surprise. The answer to why this is lies has to do with implementation of the two queues used inside the ELEV queue. Because the ELEV algorithm is a scanning algorithm the implementation was naturally sorted as requests were made. This makes finding the next request to issue fast and also finding requests inside the ECC fast. In this queue insertions are  $O(\log N)$  and issues are  $O(1)$  complexity.

The SSTF algorithm, however, is not sorted at all. Thus to find the closest chunk or the ECC chunks the algorithm must iterate over the entire queue. This makes insertions  $O(1)$  and issuing  $O(N)$  complexity. Because of the complexity the algorithm takes longer thus stalling the terrain system to request chunks as fast as the ELEV queue. Also when viewing the algorithms in action, it appears that there is some starvation happening which is causing the algorithm to not reach it's maximum performance.

Figure A-0-4 shows the average throughput of the different configuration. The test file starts at approximately the 4 GB offset and looking at the Raw DVD Performance the maximum throughput at this location is approximately 8 MB/s. The results show this is not to be expected when streaming terrain however. The standard layout consistently provides higher throughput. But as Figure A-0-3: Time to complete shows, they provide the worse performance. The standard layout must read a larger amount of data so the throughput should be higher, but the maximum is still only approximately 3 MB/s – a little more than 37% of the maximum. The ELEV provides the maximum throughput of all the layouts rather than the SSTF. This is the same reason why ELEV provides faster results in the Time to complete chart. The other scheduling algorithms have a much lower throughput, averaging approximately 0.155 MB/s or less than 2% of the DVD maximum.

The per-level streams method also has a much lower throughput. Interestingly they provide the best performance. Again the ELEV provided the best throughput when compared to the other algorithms. The FIFO proved to have the worst. As with the other statistics, using the Hilbert layout provides better throughput than the standard layout in both LOD configurations.

The seeking statistics can be found in Figure A-0-5 and Figure A-0-6. The first chart displays the seeks per second of each configuration. The second shows the time per seek and is basically the inverse of the first. The expected results we the SSTF would provide the

most seeks/s while the FIFO would provide the worst. The actual results did show the FIFO offering the least seeks/s, however, the best performance was found in using the ELEV algorithm. Also not the FIFO-No-ECC had more seeks/s than the FIFO. Both of these support the theory that queue contention and algorithmic complexity play an extremely large role in the performance.

When comparing the configurations against one another, the data shows that using Hilbert curves provide better performance than the standard layout. Additionally using per-level streams allow more seeks/s than the single stream LOD method.

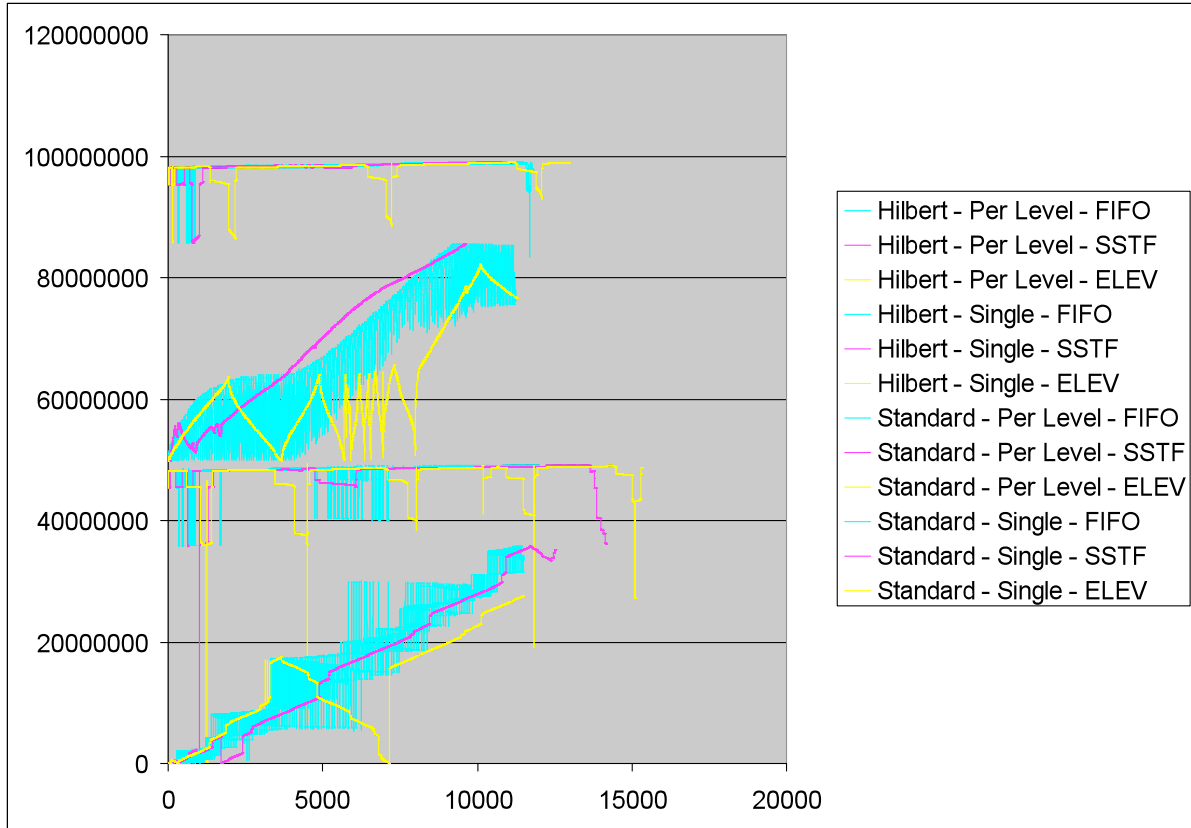
The data in Figure A-0-7 shows the average distance the OPU theoretically moved during the test<sup>5</sup>. The results show that the algorithms work as expected. On average the FIFO algorithm caused the OPU to travel the farthest in all configurations. The shortest distance is can be found in the SSTF algorithm. This shows the algorithm is correct in producing the shortest seek even though the implementation could be improved. Like the other statistics, when comparing the two space-filling methods Hilbert curves has a slightly better performance over the standard layout.

The final statistic can be seen in Figure A-0-10 which shows the time spent reading versus the time spent accessing. By first glance one can see on all configurations the majority of time was spent just waiting for the OPU to reach the desired offset. In most cases the time spent reading is less than 0.5%. The maximum is only 26%. The model used predicted a seek time of 90% and read time of 10%. This model assumed an identical read and seek speeds. Rotational and seek latency were also not included in the model. The model assumed per-level streams with a single transition between each level every second. Although good disc scheduling will help minimize stream level transitions, the true simulation inevitably cannot reach this ideal model. Even though some error was expected with the prediction, a seek time of 99% is surprisingly high and shows the importance of efficient use of the OPU.

Despite the low read percentage, the results do show that using Hilbert curves increases time spent reading and decreases time spent accessing. Also, as expected, using a single stream for LOD causes a higher percentage of reading versus accessing although keep in mind 95% of the data read is wasted so one cannot judge performance by this metric alone.

---

<sup>5</sup> Due to caching the OPU may have moved less than the calculated result.



**Figure 4-4: Offsets issued**

The graph in Figure 4-4 shows the offsets issued during each test. The colors represent different algorithms. Cyan shows the FIFO, yellow shows the ELEV and magenta shows the SSTF. Each set of configurations are grouped together with the SSTF plotting the general slope. Here one can see that the algorithms are correct and also show why the algorithms perform as they do.

In general one can see that by using per-level streams the OPU do not have to travel very far from the beginning of the first read. This is because the majority of the data is at the lowest resolution. Occasionally, higher resolution data must be fetched which explains the dips in the graphs. By using a single stream, the OPU has to traverse a larger data set. Because the single stream is entirely made of the highest resolution, that stream spans the greatest distance.

## Chapter 5: Conclusion

This project looked at the issues that come along with streaming terrain from the DVD on the Playstation® 3. Specifically it looked at two methods to map terrain onto disc, two methods to read in level-of-detail and three methods to schedule requests to the disc. The goal was to compare these methods to determine which configuration would provide the best performance.

The expected result was that using a stream of data for each level-of-detail provided better results over a single stream method. Additionally, storing data on the disc using a Hilbert curve rather than the standard per-column method was expected to have better performance. These data collected shows the expectations to be true in this project.

An unexpected find was that although the shortest-seek-time-first algorithm indeed provided the shortest seeks, it was the scanning algorithm that provided the best performance. This is believed to be because of algorithmic implementation, shared queue contention, and starvation associated with the shortest-seek-time-first algorithm.

The final results show the best performance used Hilbert curves for data layout, per-level streams for level-of-detail methods and a scanning scheduling algorithm. With this configuration the system could support a throughput of 230 KB/s with 1616 seeks/s. With the system described in this paper this translates into streaming approximately 1 Million triangles/s with a camera speed of 108,000 km/hr across a 122.88 km x 122.88 km terrain file having 30 meter resolution.

It is important to realize these results are true for this specific implementation of streaming terrain. In this particular project there are a lot of complex things happening under the hood that were unable to be controlled. Multiple levels of caching and additional kernel scheduling can have a large effect on the overall performance. Managing a custom caching system and bypassing the kernel I/O buffers can also help performance. But for this project the metric that determines which configuration would be the “best” is the number of chunks issued during the simulation since this will automatically translate into the best visual results.

Future projects that use streaming can use several things from this one. Reducing contention between the shared resources will drastically improve performance. Also, reading data in chunks smaller than the ECC block will likely reduce performance without proper

scheduling. Additionally, better compression schemes will provide better triangles/s and will decrease the number of reads needed.

These areas can also be researched further. Determining the optimal chunk size for the best reading performance is not something this project explored. Because of the limitations in size when making a single draw call this project used very small chunks. As a result the chunks on disc were much smaller than the ECC block and thus could have better performance [Microsoft. 2006]. Additionally further research could be done on a system which allows direct access to the optical media. The expected result would be scheduling the ECC blocks would provide faster performance. Finally this research should be compared against systems which use other types of media to ensure the results are universal.

## References

## Articles

Losasso, Frank; Hoppe, Hugues. 2004. "Geometry Clipmaps: Terrain Rendering Using Nested Regular Grids". SIGGRAPH.

<<http://research.microsoft.com/~hoppe/geomclipmap.pdf>>. (Accessed June 3, 2008)

Orthington, Bruce; Ganger, Gregory; Patt, Yale. 1994. "Scheduling Algorithms for Modern Disk Drives". ACM Sigmetrics Conference..

<<http://www.ece.cmu.edu/~ganger/papers/sigmetrics94.pdf>> (Accessed on July 3, 2008)

Lönn, Fredrik. 2006. "Streaming for Next Generation Games".

<[http://www.gamasutra.com/view/feature/1769/streaming\\_for\\_next\\_generation\\_games.php](http://www.gamasutra.com/view/feature/1769/streaming_for_next_generation_games.php)>. (Accessed on June 3, 2008).

Microsoft. 2006. Optimizing DVD Performance for Windows Games.

<<http://msdn.microsoft.com/en-us/library/bb147246.aspx>>. (Accessed on June 30, 2008)

Pioneer 2008. BDC-202BK Blu-ray Drive Specifications.

<<http://www.pioneer.eu/eur/products/45/104/442/BDC-202BK/index.html>>. (Accessed July 5, 2008)

"Playstation 3 Secrets – Blu-ray Drive". <<http://www.edepot.com/playstation3.html>>. (Accessed on February 24, 2009).

Paquet, Philippe. 2005. "Programming Muti-Threaded Architectures – Mutex Objects and Critical Sections".

<[http://www.gamasutra.com/view/feature/2476/programming\\_multithreaded\\_.php](http://www.gamasutra.com/view/feature/2476/programming_multithreaded_.php)>. (Accessed April 2008)

## Games

Criterion Gmes. 2008. "Burnout Paradise"; Xbox 360, Playstation® 3. Electronic Arts

Rockstar Games. 2008. "Grand Theft Auto IV"; Xbox 360, Playstation® 3 Game. 2k Games.

Avalanche Studios. 2006. "Just Cause". Xbox, Xbox 360, Playstation 2, PC Game. Eidos Interactive.

Bethesda Game Studios. 2006. "The Elder Scrolls IV: Oblivion"; Xbox 360, Playstation® 3, PC Game. 2k Games.

EA Canada. 2003. "SSX 3"; Xbox, Playstation 2, GameCube, Game Boy Advance, Gizmondo Game. EA Sports BIG.

Epic Games. 2007. "Unreal Tournament 3". Xbox 360, Playstation® 3, PC

# Appendix

This appendix shows the results of the 16 tests which are discussed in the : Results and Analysis chapter. The table is given as well as charts showing the rows individually.











