# The Design and Analysis of a Cache Architecture for Texture Mapping

Ziyad S. Hakura and Anoop Gupta

Computer Systems Laboratory
Stanford University
Stanford, CA 94305

## Abstract

The effectiveness of texture mapping in enhancing the realism of computer generated imagery has made support for real-time texture mapping a critical part of graphics pipelines. Despite a recent surge in interest in three-dimensional graphics from computer architects, high-quality high-speed texture mapping has so far been confined to costly hardware systems that use brute-force techniques to achieve high performance. One obstacle faced by designers of texture mapping systems is the requirement of extremely high bandwidth to texture memory. High bandwidth is necessary since there are typically tens to hundreds of millions of accesses to texture memory per second. In addition, to achieve the high clock rates required in graphics pipelines, low-latency access to texture memory is needed. In this paper, we propose the use of texture image caches to alleviate the above bottlenecks, and evaluate various tradeoffs that arise in such designs.

We find that the factors important to cache behavior are (i) the representation of texture images in memory, (ii) the rasterization order on screen and (iii) the cache organization. Through a detailed investigation of these issues, we explore the best way to exploit locality of reference and determine whether this technique is robust with respect to different scenes and different amounts of texture. Overall, we observe that there is a significant amount of temporal and spatial locality and that the working set sizes are relatively small (at most 16KB) across all cases that we studied. Consequently, the memory bandwidth requirements of a texture cache system are substantially lower (at least three times and as much as fifteen times) than the memory bandwidth requirements of a system which achieves equivalent performance but does not utilize a cache. These results are very encouraging and indicate that caching is a promising approach to designing memory systems for texture mapping.

## 1 Introduction

Computer graphics is becoming an increasingly important application. Consequently, there has been much interest from designers of general-purpose microprocessors, media processors, and specialized hardware to provide cost-effective real-time computer graphics capabilities. Examples of recent developments in this area are the Visual Instruction Set (VIS) in UltraSPARC™ [16] and FBRAM [18] from Sun Microsystems, MMX™ Technology [19] and Accelerated Graphics Port (AGP) [1] from Intel Corporation, Magic Carpet [12] from MIPS Technologies, and Talisman [13] from Microsoft Corporation. In this paper, we focus on one challenging aspect of graphics architecture, the design of a memory system for texture mapping.

The mapping of images onto the surfaces of three-dimensional objects is known as texture mapping. Texture mapping has earned the role of a fundamental drawing primitive for its ability to substantially enhance the realism and visual complexity of computer-generated imagery [23, 22]. Examples of this technique include the mapping of a 2D color image of a wooden texture to the surface of a guitar, a road image to a highway, or a grassy plot image to a mountain. In addition to the mapping of surface color [5], texture mapping has been used for mapping a myriad of other surface parameters including reflection of the environment [21], bumps [9], transparency [7], and shadows [20, 25].

One characteristic of texture mapping is that texture images often require large amounts of memory (typically in the range of a few megabytes to tens of megabytes). The amount of memory is dependent upon the number of textures in a scene and the size of each texture. It is generally accepted that the usefulness of texture mapping hardware increases with the amount of memory dedicated for textures. Another characteristic of texture mapping is that it requires many calculations and texture lookups. This characteristic causes it to be the main performance bottleneck in graphics pipelines. For each screen pixel (fragment) that is textured, the calculations consist of generating texture addresses, filtering multiple texture samples to avoid aliasing artifacts, and modulating the texture color with the pixel color. (Throughout this paper we will use the terms *fragment* and *screen pixel* interchangeably.) Since the number of fragments that are textured can be quite large (typically tens to hundreds of millions per second), and each textured fragment requires multiple texture lookups (usually 8), the memory bandwidth requirements to texture memory can be very large (typically several gigabytes per second). In addition, to achieve the high clock rates required in graphics pipelines, low-latency access to texture memory is needed.

The Silicon Graphics' RealityEngine [14] is an example of a high-end parallel graphics architecture that can support real-time texture mapping. This system uses multiple engines (also called fragment generators) for texture mapping fragments. Each engine has an 8-way banked DRAM memory system that is dedicated for textures (total 16 MB), and can perform eight independent texture lookups in parallel. Since there are multiple fragment generators (between 5 and 20) and each one has its own dedicated memory, the texture images must be replicated in each memory. One problem with this architecture is that an application running on a 20 fragment generator system is limited to 16 MB of unique memory for textures even though there is a total of 320 MB of texture memory in the system. Therefore, the domain of applications that can utilize the texture mapping hardware is constrained by the limited amount of unique memory.

An alternative approach for providing fast texture accesses and high bandwidths is to use an SRAM cache with each fragment generator instead of a dedicated DRAM memory. The premise of this idea is that there is a substantial amount of locality of refer-

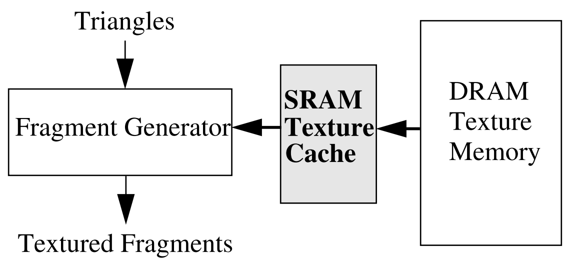



**FIGURE 2.1. Block diagram of texture mapping hardware.**

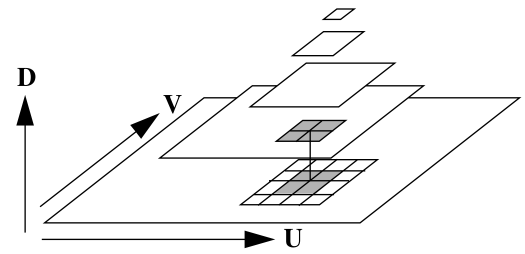



**FIGURE 2.2. Illustration of a Mip Map. Eight texels (shaded) from two adjacent levels of the pyramid are used in a Trilinear Interpolation.**

ence in texture mapped scenes. Thus, the cache size can be relatively small compared to a dedicated memory. The cache would be backed by a shared pool of DRAM. One advantage of this solution is that there is no need for texture replication for each fragment generator and the amount of unique memory for textures can be much larger. Another advantage is that the lower cost of a small cache compared to a much larger dedicated memory makes this architecture more scalable. A final advantage is that the latency of access to texture memory can be much lower and the access bandwidth can be substantially higher due to the use of SRAM.

In this paper, we focus on a system that uses a single fragment generator and study the implications of a texture cache on memory bandwidth and rendering performance. Particularly important to the cache behavior are the representation of textures in memory, the rasterization order on screen and the cache organization. Through a detailed investigation of these three issues, we explore the best way to exploit temporal and spatial locality and determine whether this technique is robust with respect to different scenes and different amounts of texture. It is interesting to note that current graphics pipelines do not use caches for texture mapping and no serious evaluations have been published. Our simulation indicates that there is a significant amount of temporal and spatial locality and that the working set sizes are relatively small (at most 16KB) across all cases that we studied. As a result, we find that even small caches of 32KB can offer substantial cost/performance benefits. With respect to the representation of textures in memory, we find that a blocked representation is required to fully exploit spatial locality and to avoid any dependency on the orientation of textures as they appear on the screen. When the texture images are large, the image arrays must be either padded or blocked at a coarser level to avoid conflicts between texture blocks. Finally, we find that a tiled rasterization order is useful for reducing the working set size and for avoiding conflicts between texture blocks.

The remainder of the paper is structured as follows. The next section provides background information on texture mapping. The motivation for a texture cache and its potential benefits are described in Section 3. Section 4 explains our experimental methodology and describes the benchmarks used in the study. Section 5 introduces two representations of textures in memory and evaluates their miss rate characteristics and interactions with the cache organization. Section 6 introduces tiled rasterization and evaluates its effect on working set size and conflict misses. Section 7 relates the miss rate results from Section 5 and Section 6 to memory bandwidth and rendering performance. Finally, Section 8 summarizes the results and presents conclusions.

## 2 Background

To understand where texture mapping fits in a real-time rendering system, we briefly describe the four major functions of a traditional graphics pipeline [11]: geometry processing, fragment generation, hidden surface removal, and framebuffer display. The 3D geometries of scenes are commonly defined in terms of triangles. In the first stage, matrix transformations are applied to the triangle vertices, resulting in a perspective mapping of the triangles to the 2D display. The second stage is fragment generation consisting of rasterization, shading, and texture mapping of fragments. The third stage is hidden surface removal, accomplished using a z-buffer algorithm and the final stage is the display of the rendered image stored in the framebuffer.

Of the four stages in a graphics pipeline, we are primarily interested in the second stage. A block diagram of the second stage is shown in Figure 2.1. The fragment generator is responsible for rasterizing the input triangles into fragments and performing the shading and texture mapping calculations. Rasterization involves interpolating screen coordinates, depth, texture coordinates and shading color across the surface of each triangle, and identifying the screen pixels (also called fragments) that lie inside the triangles. The color applied to the fragments is usually a function of both the shading and texture mapping calculations. Before we can describe how the fragments are texture mapped, we must first discuss the concept of Mip Mapping.

The goal of Mip Mapping, introduced by Williams [15], is to efficiently avoid aliasing artifacts by quickly filtering the texture image. It involves representing a texture as an image pyramid as illustrated in Figure 2.2. The bottommost level of the pyramid is the original texture image and each subsequent level is a filtered and down-sampled version of the previous level. Consider an example in which a texture mapped triangle is being viewed from far away, so that there is low resolution of image details. In this case, an area of texture can be represented by one screen pixel. Instead of re-filtering the image based on the level-of-detail required, we can use the pre-computed image from the appropriate level of the Mip Map. Each level of the Mip Map corresponds to a particular screen pixel to texture pixel ratio, where a screen pixel refers to the pixel on the screen itself and texture pixel, or *texel*, refers to the pixel in the texture image being depicted on the screen.

When applying texture, we compute the texture coordinates *(u,v)* and the screen pixel to texel ratio *d*. In practice, the u,v, and d coordinates do not exactly map to a single texel. Consequently, we must take the weighted average of the eight texels closest to the (u,v,d) coordinates, four texels each from the two levels whose d values most closely approximate the desired d value. This is known as trilinear interpolation. A special case arises if the screen pixel to texel ratio is less than one. This occurs when a texture mapped triangle is being viewed from very close and causes the original texture image to be magnified. In this case, we take the weighted average of the four texels closest to the (u,v) coordinates in the bottommost level of the image pyramid. This is known as bilinear interpolation. The result of the trilinear or bilinear interpolation is usually modulated with the color computed from the shading calculations to obtain the final color for the textured fragment.

In summary, the fragment generator is responsible for rasteriz-

| Phase of Fragment Generator Computation | Precision | Add/Subtract/ Shift | Multiply | Divide | Number of Texture Memory Accesses |
|---|---|---|---|---|---|
| Per Triangle Setup | Fixed/Floating | 89 | 64 | 1 | - |
| Per Fragment Rasterization and Shading | Fixed/Floating | 11 | | 1 | - |
| Level-of-detail, d | Fixed/Floating | 9 | 9 | | - |
| Texel coordinates nearest (u,v,d) | Fixed<br>Integer | 5<br>14 | 5 | | - |
| Texel address calculation | Integer | Dependent upon Memory Representation. See Section 5. | | | - |
| Trilinear Interpolation/ Bilinear Interpolation | Fixed | 56<br>24 | 28<br>12 | | 8<br>4 |
| Modulation with fragment color | Fixed | 8 | 4 | | - |

**TABLE 2.1. Computational costs of the operations of a fragment generator. Apart from triangle setup, all the costs are on a per fragment basis.**

ing triangles into fragments, computing level-of-detail and Mip Map texture addresses for each fragment, filtering the texels that are accessed using either trilinear or bilinear interpolation, and finally applying the texture color to the fragments. Typical unoptimized computational costs for each of the operations of a fragment generator are shown in Table 2.1. Since these costs are quite high, fragment generators often exploit deep pipelining and parallelism. In fact, most of the costs are incurred by texture mapping.

# 3 Texture Cache: Motivation and Benefits

In this section, we identify the types of locality of reference that are present in texture accesses and discuss the benefits of adding an SRAM texture cache between the fragment generator and DRAM texture memory.

## 3.1 Locality of Reference in Texture Mapping

The effectiveness of a memory hierarchy depends on locality of reference in data accesses. Both spatial and temporal locality are present in texture mapping.

### 3.1.1 Spatial Locality

The representation of textures as Mip Maps contributes to spatial locality in texture accesses. The Mip Map accesses have a high degree of spatial locality since the level of the map is selected to closely match the level-of-detail that is being drawn on the screen. In essence, this means that movements of one pixel in screen space roughly correspond to movements of one texel in texture space, hence the spatial locality in texture space. The spatial locality in Mip Map accesses is thus present irrespective of the scene.

### 3.1.2 Temporal Locality

Temporal locality in accesses to texture data is present when rendering a single frame and between consecutive frames. We generally do not expect our caches to exploit temporal locality between consecutive frames because the cache sizes that we consider are much smaller than the amount of texture data that is typically used by a single frame. Between memory and disk, however, this kind of temporal locality is of interest. Within a single frame, two types of temporal locality are present: locality arising from overlap of accesses needed for filtering between neighboring fragments and locality arising from repeated texture. Below, we discuss each of these two types of temporal locality in more detail.

**Overlap of Accesses between Neighboring Fragments**

Temporal locality is present between the texel accesses of neighboring fragments. Since each textured fragment must access multiple texels for trilinear or bilinear interpolation, it is expected that some of these accesses will overlap with the texel accesses of neighboring fragments. We measured the average number of accesses per texel made by a spatially contiguous group of fragments across all four benchmark scenes, which are described in Section 4. The results for trilinear interpolation (lower level), trilinear interpolation (upper level) and bilinear interpolation are 4, 14 and 18, respectively. For trilinear interpolation, we distinguish between texel accesses that are to the more detailed level, lower level, and the less detailed level, upper level, of the two adjacent levels of the Mip Map that are used in the interpolation. These results clearly show that the texels in the upper level of a trilinear interpolation are accessed a larger number of times than the texels in the lower level. Since the lower level is more detailed than the upper level, it is traversed more rapidly and this leads to fewer accesses per texel. In general, we expect texels in the lower level of a trilinear interpolation to be accessed an average of four times and the texels in the upper level to be accessed an average of sixteen times. For bilinear interpolation, the number of accesses per texel is directly related to the amount of texture magnification and this can vary widely depending on the scene.

**Repeated Texture**

Temporal locality is also present when a texture is repeated across the surface of an object. An example application is a wall that is textured by repeating a 2D texture image of an individual brick. Since the amount of repetition is a function of the texture coordinates defined in the high-level geometric description of the scene, this type of temporal locality is very scene dependent. We measured the average number of times a texel is repeated in the benchmark scenes. The results for the Town, Guitar, Goblet and Flight scenes are 2.9, 1.7, 1.1 and 1.0 times, respectively.

## 3.2 Texture Cache Benefits

The evolution of DRAM technology motivates the use of an SRAM texture cache. The rapid growth in DRAM density has meant that fewer DRAM chips are needed to construct a memory of a fixed size. The use of higher density memory chips has led to a decline in bandwidth per Megabyte of memory [24]. An SRAM texture cache can exploit locality of reference to hide most of the texture accesses from reaching memory, thus lowering the bandwidth requirements.

Another reason for adding an SRAM cache is that block transfers of cache lines between the cache and memory make it possible to get the most bandwidth out of the memory. Present-day DRAM architectures are optimized for long burst transfers to microprocessor caches since this amortizes the setup costs of the transfer over many bytes and leads to the most efficient memory bus utilization.

A third reason for having an SRAM texture cache is that static memories typically have shorter access latencies than dynamic memories. The SRAM cache can be tightly coupled with the fragment generator which makes it possible to run the fragment generator at higher clock rates. This is an incentive for integrating the SRAM cache onto the same chip as the fragment generator.

Finally, a recent trend in computer graphics has been the use of rendered images as textures [3]. As a result, it has become desirable to unify the framebuffer and texture memories to avoid copy-

| Scene | Image Resolution (Pixels) | Number of Triangles | Average Triangle Area (Pixels) | Average Triangle Width (Pixels) | Average Triangle Height (Pixels) | Number of Textures | Texture Storage (MB) | Texture Used (MB) | Texture Used (%) | Pixels Textured (millions) |
|---|---|---|---|---|---|---|---|---|---|---|
| Flight | 1280x1024 | 9152 | 294 | 38 | 20 | 15 | 56 | 6.3 | 11% | 1.4 |
| Town | 1280x1024 | 5317 | 1149 | 67 | 23 | 51 | 4.7 | 1.8 | 38% | 2.1 |
| Guitar | 800x800 | 719 | 1867 | 72 | 94 | 8 | 4.9 | 1.1 | 23% | 0.7 |
| Goblet | 800x800 | 7200 | 41 | 25 | 14 | 1 | 1.4 | 0.78 | 56% | 0.3 |

**TABLE 4.1. Texture Mapping Benchmarks.**

ing data between the two. A fragment generator connected to an SRAM texture cache does not necessarily require a dedicated texture memory. This makes it possible to fetch the texture data as it is needed from whichever DRAM buffer it is stored in. In fact, multiple fragment generators can share the same DRAM memory system and no cache coherence is needed since the texture data is mostly read-only. The caches can be flushed if necessary when the textures change.

The organization of the texture cache is characterized by three parameters: cache size, line size and associativity. The choice of cache size is related to the working set sizes of texture mapped scenes. The line size depends on the amount of spatial locality that can be exploited. As noted earlier, larger line sizes can elicit a larger fraction of the peak memory bandwidth. Finally, the cache associativity is related to the kinds of conflicts that can occur.

# 4 Methodology

Doing performance studies for graphics architectures is a difficult endeavour. The primary reasons are the lack of well-established benchmark programs and the lack of tools for obtaining traces of graphics commands. In this section, we discuss how we address these and additional issues.

## 4.1 Simulation Environment

The simulation environment consists of three components. The first component is a software implementation of a three-dimensional polygonal graphics pipeline. This is responsible for geometry, clipping, lighting of vertices, rasterization, shading, texture mapping and finally Z-buffering. The pipeline is similar to the one described in [14]. Specifically, the texture mapping implementation is based on the OpenGL specification document [17]. Since the pipeline is implemented in software, we can easily experiment with different representations of texture in memory and can rasterize the triangles in either the horizontal or vertical direction. The textures are assigned memory using the malloc() system call and we allocate 32 bits per texel. The triangles are rasterized in the same order that they are specified in the input.

The second component is a capability to trace the GL calls that are made by a graphics application while it is running in real-time on a hardware-based renderer. This was done using a standard utility program called gldebug, intended for debugging GL programs, and a parser that parses the GL calls while the application is running. This technique can be applied to any binary executable that makes GL calls. The trace is then fed to our software implementation of the graphics pipeline which executes equivalent procedures and generates images as output. The images allow us to verify that the interpretation of the trace is accurate.

The third component is a trace-driven cache simulator that can model different cache sizes, line sizes and associativities. Whenever the software-based fragment generator accesses a texel from memory, it also makes a call to the cache simulator passing the address of the texel as a parameter. The cache simulator runs concurrently with the graphics pipeline.

## 4.2 Benchmarks

We study four benchmarks that are applications of color texture mapping. The characteristics of these benchmarks are summarized in Table 4.1. Because the cache sizes that we consider are much smaller than the amount of texture used in each benchmark, we do not expect to exploit temporal locality between consecutive frames. For this reason, we have selected a single frame per benchmark.

The first two benchmarks, shown in Figure 4.1 and Figure 4.2, are taken from the Silicon Graphics Reality Engine Demo Suite and are representative of present-day applications of texture mapping. The Flight scene, shown in Figure 4.1, uses several 1024x1024 pixel satellite images as textures and maps these textures onto a geometric model of the terrain. An important characteristic of the Flight scene is that it has large variations in level-of-detail as a result of the mountainous terrain. In comparison, the Town scene, shown in Figure 4.2, maps many smaller textures onto flat surfaces and these textures appear upright in the image of the scene. The Guitar scene, shown in Figure 4.3, is another application where textures are mapped onto flat surfaces. It differs from the Town scene in that the textures are larger and they do not appear uniformly oriented in the image of the scene. Finally, the fourth benchmark, shown in Figure 4.4, consists of a single texture wrapped around a goblet. The Goblet benchmark is characterized by its use of small triangles to make up the curved surface and by the variations in level-of-detail that occur when the surface becomes 90 degrees to the viewing angle. In all four benchmarks, the textures are stored as Mip Maps and trilinear interpolation is used for filtering the images.

# 5 Representation of Texture Maps in Memory

The representation of texture maps in memory is important to the cache behavior because it effects where texture data is placed in the cache. In this section, we study the interaction between the texture representation and the parameters of the cache organization. We examine two representations: a Nonblocked representation and a Blocked representation.

## 5.1 Previous Work

In [15], Williams also described a clever memory organization and addressing scheme for two-dimensional textures which is illustrated in Figure 5.1(a). At each level of the pyramid, the image is separated into its red, green, and blue color components. The filtered and down-sampled levels of the pyramid are stored above and to the left of their predecessor levels. Once the u, v, and d coordinates are calculated, indexing the Mip Map is a simple matter that involves binary operations since the individual images are sampled at powers of two.

Although this representation makes addressing very inexpensive, it is prone to several problems from a caching perspective. The most serious problem is that the individual color components

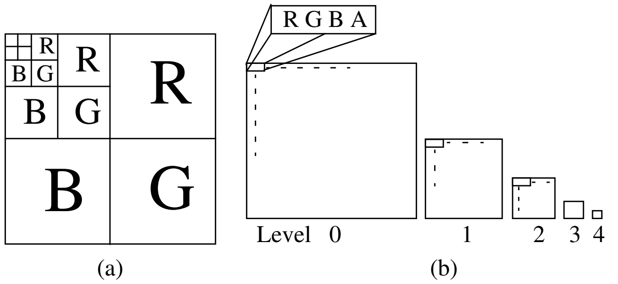



**FIGURE 5.1. (a) Mip Map representation proposed by Williams. (b) Base nonblocked representation.**

of a texel are always separated by powers of two bytes in memory since the texture image dimensions are powers of two. (Note that in most graphics libraries including OpenGL, the texture image dimensions are either restricted to be powers of two or the images are padded so that the dimensions are powers of two.) Thus, it is very likely that the individual color components will map to the same line in the cache resulting in conflicts. Another problem is that the representation does not exploit spatial locality that would be present if the color components were stored contiguously in the same cache line. Finally, since the image is separated by component, reading a texel from the cache requires three separate accesses with tag comparisons for each access.

## 5.2 Base Nonblocked Representation

An alternative representation is illustrated in Figure 5.1(b). In this representation, the red, green, blue and alpha components are stored contiguously. Each level of the pyramid is represented by its own two-dimensional array and each array is stored in row-major order. We will consider this representation as the base representation for the remainder of this paper since it is not prone to the problems mentioned previously.

### 5.2.1 Texel Addressing

The main advantage of the base representation is that it requires the fewest number of addressing calculations. The addressing calculations, shown below, assume that the dimensions of the texture arrays are powers of two. The texel addressing calculations must be performed eight times per fragment since a trilinear interpolation requires eight texture samples.

All variables are a function of the Mip Map level.
**tu**, **tv**: texel u- and v-coordinates
**base**: starting address of 2D texture array
**lw**: $\log_2$(width of 2D texture array in pixels)

Texel address = base + (tv << lw) + tu

### 5.2.2 Cold Misses

Cold misses cannot be avoided and so they represent the lower bound for miss rates in the cache. Figure 5.2 shows graphs of miss rate versus cache size measured for fully associative caches with an LRU replacement policy. These graphs assume a line size of 32 bytes and fully associative caches so that we can ignore conflict misses. We study the effect of line size on cold misses and defer the discussion of conflict misses to Section 5.3.3.

Cold misses typically occur in two places: along triangle edges where the textures are first accessed, and in the interior of large triangles when the texture accesses cross cache line boundaries. The cold miss rates can be deduced from the miss rates for the large caches since they do not include capacity misses. (Note that the cold miss rates are the same regardless of whether rasterization is horizontal or vertical; we discuss the effect of rasterization direction on working set size in Section 5.2.3). The cold miss rates for the Town, Guitar, Goblet and Flight scenes are 0.55%, 0.87%, 1.5%, and 2.8% respectively. These miss rates are quite low considering that the line size of 32 bytes holds just eight texels.

One factor that contributes to differences in cold miss rate is the amount of temporal locality in the form of repeated texture. As previously noted in Section 3.1.2, the amount of repetition of textures is very scene dependent. Another factor is the frequency of changes in level-of-detail across the surfaces that are textured. In the Town and Guitar scenes, the surfaces are predominantly flat causing the variations in level-of-detail to be gradual. As a result, we find that a large fraction of each line of texture data that is fetched into the cache is used. In comparison, the mountainous terrain in the Flight scene causes frequent changes in level-of-detail. Consequently, the accesses are fragmented across different levels of the Mip Map and a smaller fraction of each line of texture data that is fetched into the cache is used. Since more lines are needed for texturing, the cold miss rates are higher.

We also measured the cold miss rates for caches with a larger 128 byte line size. The results for the Town, Guitar, Goblet and Flight scenes are 0.15%, 0.25%, 0.42%, and 1.1% respectively. We see that the cold miss rates are much reduced with increasing line size, indicating the presence of substantial spatial locality.

### 5.2.3 Working Set Size

The working set size is a measure of the amount of data that is actively in use at a particular time. Most applications have a hierarchy of working sets [6]. In a graph of miss rate versus cache size, the different levels of the working set hierarchy can be seen as plateaus followed by sharp reductions in miss rate at particular cache sizes.

One can consider the coarsest level of the working set hierarchy in texture mapping to consist of all the texture data required to render an image. The *Principle of Texture Thrift* [4] defines the minimum amount of data required to render an image:

*Given a scene consisting of textured 3D surfaces, the amount of texture information minimally required to render an image of the scene is proportional to the resolution of the image and is independent of the number of surfaces and the size of the textures.*

This principle is a consequence of the fact that each pixel of the image can represent at most one filtered sample of texture. The advantage of representing textures in the form of Mip Maps is that the amount of data used in filtering for each pixel is fixed regardless of the level-of-detail.

We are interested in understanding the makeup of the first significant working set because it corresponds to the smallest cache size that one would consider using for caching texture images. We are also interested in making a worst-case estimate of the size of the first significant working set to generalize our analysis beyond the benchmarks that we study. We define the first significant working set as the first level of the working set hierarchy and begin by noting that a triangle is rasterized one scan line at a time, where a scan line consists of either a horizontal or vertical span of pixels in screen space. We assume for our worst-case analysis that the triangle being rasterized is large and spans the entire area of the screen. We have shown that there is spatial and temporal locality in the accesses to texture data, especially by accesses from adjacent scan lines. A cache that is large enough to hold the texture data needed for an entire scan line can exploit such localities of reference to significantly reduce miss rates. Thus, we see that the first level working set consists of the texture data needed for an entire scan line, and a significant reduction in miss rate occurs when the cache is large enough to hold this working set. Two parameters that can place a bound on the worst-case working set size are the texture image size and the screen size. If the texture image size is less than

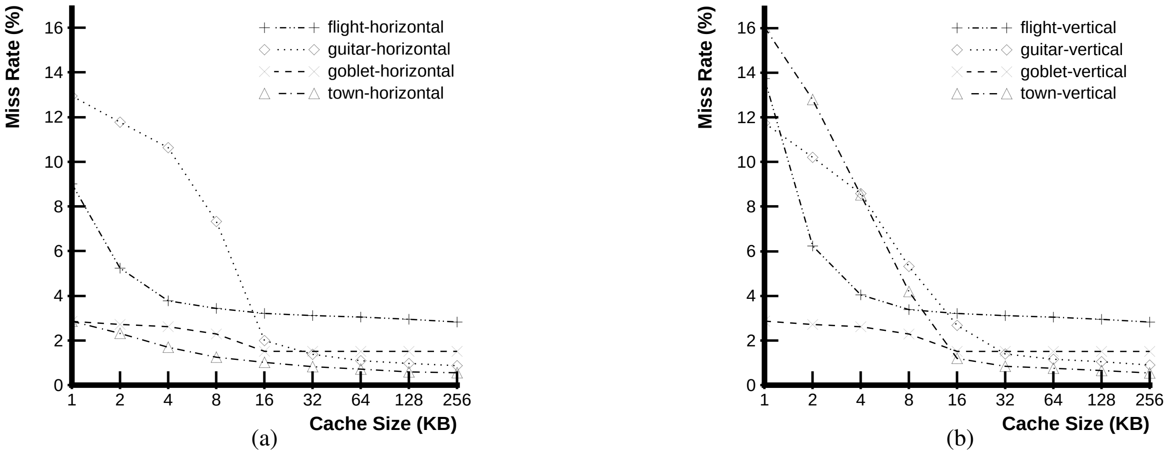



**FIGURE 5.2. Results for the base representation using fully associative caches. The line size is 32 bytes (8 texels). (a) Horizontal rasterization (row major order). (b) Vertical rasterization (column major order).**

the screen size, the texture is wrapped around and repeated. In this case, the worst case working set size is bounded by the cache line size multiplied by the length of the diagonal of the texture image, since this is the maximum length through the texture and the texture can appear in an arbitrary orientation on the screen. On the other hand, if the texture image size is greater than the screen size, the worst case working set size is bounded by the cache line size multiplied by the maximum width or height of the screen, depending upon whether we rasterize horizontally or vertically[1].

We have measured the average runlength of texel accesses from the same texture. The results for the Town, Guitar and Flight scenes, which are applications of more than one texture, are 223,629, 553,745 and 562,154, respectively. An underlying assumption is that the triangles are processed in the same order that they are specified in the high-level geometric description of the scene. These long runlengths demonstrate that the working set is limited to one texture at any point in time.

When rendering a span of pixels in screen space, texels (texture pixels) may be traversed in arbitrary orientations on the screen. In the worst case, the texture accesses can be streaming vertically through the texture causing only a fraction of each cache line that is brought into the cache to be actively used. The net effect is that the working set size is larger than it needs to be. This is a shortcoming of the base representation that we have chosen.

Figure 5.2(a) shows the miss rate results versus cache size when the scenes are rasterized horizontally. The first level working set sizes for the Flight, Town, Guitar and Goblet scenes are 4KB, 8KB, 16KB, and 16KB respectively. These working set sizes are a very small fraction of the total amount of texture that is used to render each scene (please see Table 4.1 for the full texture content used) and indicate that texture caching can be effective with relatively small cache sizes.

Figure 5.2(b) shows the miss rates versus cache size when the scenes are rasterized vertically. Compared to Figure 5.2(a), the miss rates for the Town scene have substantially increased for small cache sizes, whereas the miss rates for the other scenes have not changed quite as much. The first level working set size for the Town scene has grown from 8KB to 16KB because of a mismatch between the base representation and the rasterization direction. In fact, since most of the textures appear upright in the image of the Town scene, vertical rasterization causes the direction in which the texels are accessed to be perpendicular to the direction in which the texels are stored leading to worst-case behavior. In the Flight scene, the effect is less pronounced because the triangles are moderately sized. The Goblet scene results do not change because the triangles are relatively small. The results for the Guitar scene do not change very much because the textures in this scene are not uniformly oriented in any particular direction. From an architectural perspective, the important point is that the base representation is sensitive to the direction of texture accesses. For the remainder of this paper, we report results using vertical rasterization for the Town scene since this leads to worst-case behavior, and report results using horizontal rasterization for the Flight, Guitar and Goblet scenes.

In summary, we have shown that the cold miss rates are very low and can be further reduced with larger line sizes. The long texture runlengths, measured with the same rendering sequence as that taken by actual hardware engines that do texture mapping, indicate that the working set is limited to one texture. In addition, the working set sizes are very small compared with the amount of texture that is used to render each scene, justifying the use of relatively small texture caches. Finally, we found that the base representation is sensitive to the orientation of textures on the screen.

## 5.3 Blocked Representation

A blocked representation can be used to reduce the dependency on the orientation of textures, as seen by the viewer, and to exploit more spatial locality. A blocked representation of a 5-level Mip Map is illustrated in Figure 5.3. In this representation, also commonly known as Tiled, texels that are within a square region (block) of a two-dimensional image are ordered consecutively in memory. The idea of texture blocking is previously discussed in [4] and [10]. There are several issues that are raised by this representation. First, what is the addressing overhead associated with blocking? Second, how do we select the block size and is it related to the cache line size? Third, what are the improvements in miss rate as we increase the cache line size? Finally, what is the effect of blocking on conflict misses? In the following subsections, we discuss each of these issues in turn.

### 5.3.1 Texel Addressing Overhead

The blocked representation converts two-dimensional texture arrays into four-dimensional arrays. Therefore, the texel addressing calculations must be done in a two-step process which is illustrated below. We assume that the blocks are square and have dimensions that are powers of two. Furthermore, we assume that the blocks have the same dimensions across all Mip Map levels.

1. The screen rasterization path that would lead to the smallest working set would follow a Peano-Hilbert order since this would traverse a region of the texture in a spatially contiguous manner.

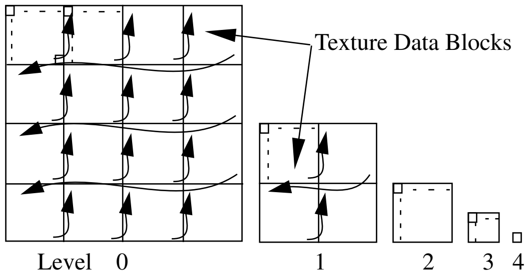



**FIGURE 5.3. An illustration of the blocked representation for a texture Mip Map.**

**bw**, **bh**: block width and height in texels. These are equal and are powers of two.
**lbw**, **lbh**: $\log_2$(bw or bh)
**bs**: $\log_2$(bw * bh)

The following variables are a function of the Mip Map level.
**rs**: $\log_2$(width of texture array in texels * bh)
**tu**, **tv**: texel u- and v-coordinates
**bx**, **by**: block coordinates
**sx**, **sy**: sub-block coordinates in texels
**base**: starting address of 4D texture array

bx = tu >> lbw
by = tv >> lbh
Block address = base + (by << rs) + (bx << bs)

sx = tu & (bw - 1)
sy = tv & (bh - 1)
Texel address = Block address + (sy << lbw) + sx

The variables bx, by, sx, and sy are simply bit-fields of the u- and v- texel coordinates, tu and tv. Furthermore, two of the shift operations shown above, involving the variables bs and lbw, have constant shift amounts assuming that the block dimensions remain fixed. Hence, the aggregate hardware overhead of the blocked representation compared to the base representation simply consists of two additions. These operations are incurred in the calculation of the block address.

### 5.3.2 Selecting a Block Size and Cache Line Size

The next question we would like to answer is whether there is any interaction between the block size used in the representation and cache line size. Figure 5.4 shows miss rate versus cache line size for a variety of block sizes. We chose a 32KB cache for our measurements because this cache size is larger than the working set sizes which we noted earlier. The graph in Figure 5.4(a) is for the Town scene. It shows that the lowest miss rates occur when the block size most closely matches the cache line size. For example, the lowest miss rate for a 64 byte cache line is for a 4x4 block which has a block size of 64 bytes. This effect can be explained by observing that individual cache lines that hold square regions of texture are most effective at exploiting spatial locality. When the block sizes greatly differ from the cache line sizes, we find that the working set sizes can become unnecessarily large leading to many capacity misses. The results for the Guitar scene, shown in Figure 5.4(b), are very similar.

We would like to quantify the extent of reduction in miss rate as the line size is increased. Figure 5.5 shows the effects of line and block size on miss rate for a fully associative 32KB cache. At 32KB, the primary misses that remain are cold misses and these misses are independent of the orientation of textures as viewed on the screen. The miss rates for the Flight, Goblet, Guitar and Town scenes at a line size of 32 bytes are 2.8%, 1.5%, 1.2% and 0.8% respectively. The miss rates at a line size of 128 bytes are 0.87%,

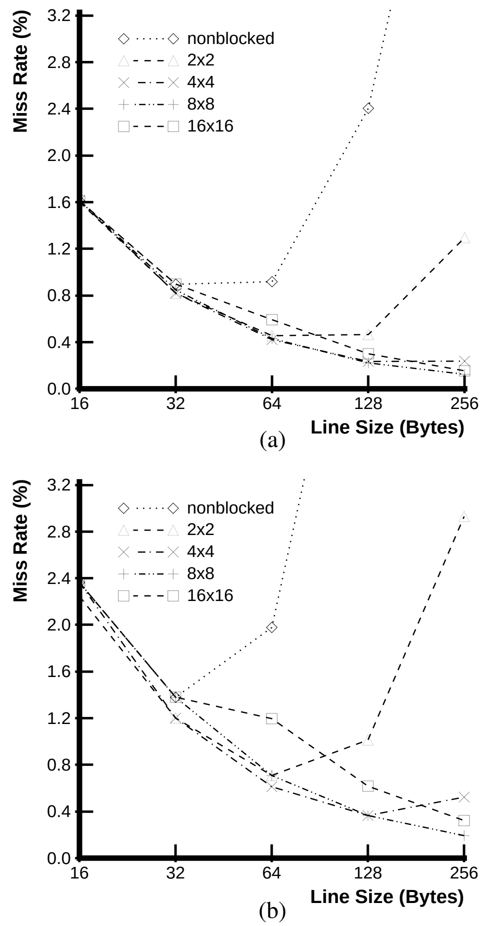



**FIGURE 5.4. The interaction between block size and cache line size. These results were measured using fully associative 32KB caches. The block dimensions are given in texels. (a) Town-vertical. (b) Guitar-horizontal.**

0.41%, 0.36% and 0.21%. There appears to be a significant reduction in miss rate as the line and block sizes are increased.

Thus far, we have presented results for a 32KB cache and we have found that this cache size is adequate for holding the working sets of all four scenes. In Figure 5.6, we show results specific to the Guitar scene for different cache sizes. These graphs demonstrate that the blocked representation when coupled with larger line and block sizes, leads to a reduction in the number of capacity misses for cache sizes that are smaller than the working set size. In Figure 5.4 we saw that increasing the line size alone, without blocking, leads to worse miss rates. We can conclude that the blocked representation is an essential component for reducing the frequency of capacity misses. We have found that the other benchmark scenes experience similar reductions in capacity misses (although not shown here due to space reasons) when the cache sizes are smaller than the working set sizes.

In summary, we can conclude that the best block size to use for texturing corresponds to when the memory required to store one block of texture is the same as the cache line size. Furthermore, for scenes that we evaluate, there is enough spatial locality that larger line sizes can be used.

### 5.3.3 Conflict Misses

Thus far, we have ignored conflict misses and have only shown results for fully associative caches. In this section, we identify the

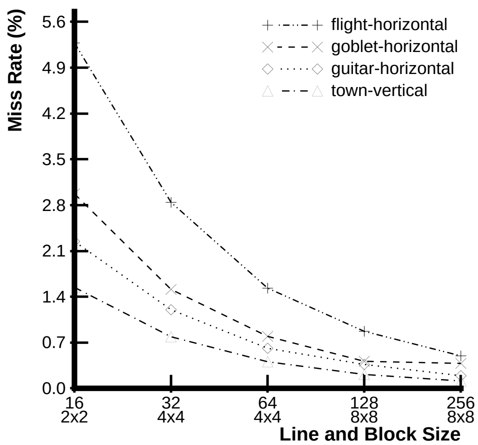



**FIGURE 5.5. Effect of changes in line and block size on miss rates for all four scenes. These results were measured using fully associative 32KB caches. The block dimensions are given in texels.**

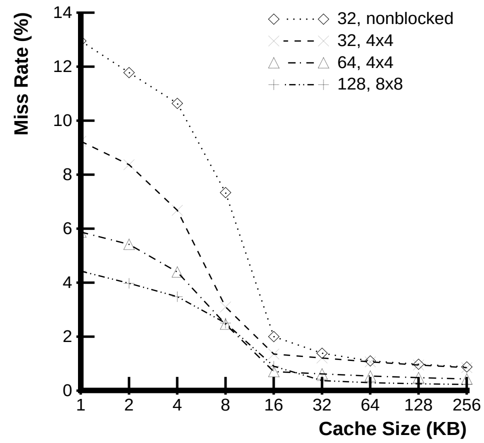



**FIGURE 5.6. Effect of blocked representation on miss rates for different cache sizes. These results were measured for the Guitar scene using fully associative caches. The lines are labeled with the line size and block dimensions given in texels.**

kinds of conflict misses that can occur and discuss how they can be avoided.

In Figure 5.1(b), we showed that in the base nonblocked representation, the levels of a Mip Map are stored separately in memory as two-dimensional arrays. Since these arrays are usually restricted to have dimensions that are powers of two, conflicts are prone to occur between neighboring texels that are in the same column. This kind of conflict miss can be avoided with the use of a blocked representation. The texels that lie within a block are guaranteed not to conflict in the cache since they are stored consecutively in memory. Moreover, since the block size is selected to be the same as the cache line size, the group of texels that lie within a block would be held by the same cache line.

Conflicts can also occur between blocks which are in different levels of a Mip Map. These conflicts are likely to be avoided with a two-way set-associative cache since trilinear interpolations simultaneously access at most two levels of a Mip Map.

Figure 5.7 shows graphs of miss rate versus cache size for different cache associativities. A relatively large line size of 128 bytes was selected for these experiments because conflict misses are more likely to occur when there are fewer lines in the cache. Thus, these results are indicative of the worst-case effect of conflicts on miss rate. The results in Figure 5.7(a) are for the Goblet scene and show that there is a significant difference in miss rates between the direct-mapped caches and the two-way set-associative caches. Since the triangles in the Goblet scene have fairly small areas, it is unlikely that conflicts will occur between blocks that are in the same level of a Mip Map. Hence, we attribute this difference in miss rates to conflicts between blocks that are in different levels of a Mip Map. The graph also shows that the two-way set-associative caches have the same miss rates as the fully associative caches indicating that increasing the associativity beyond two-way does not lead to any further improvement in miss rates.

The results in Figure 5.7(a) for the Goblet scene assumed a blocked representation. Had the representation been nonblocked, an eight-way associative cache would have been required to achieve the same miss rates as a fully associative cache among the small cache sizes. This result demonstrates that the blocked representation is useful for avoiding conflicts between neighboring texels in different rows of a 2D array.

The results in Figure 5.7(b) are for the Town scene. In this graph, we also find that two-way set-associativity is useful for eliminating conflicts between adjacent levels of a Mip Map. However, unlike Figure 5.7(a), there is a notable difference between the miss rates for the two-way set-associative caches and the fully associative caches. Vertical rasterization through the upright textures in the Town scene leads to conflicts between multiple blocks which lie within the same 2D array. The graph for the Town scene also shows that limited cache associativities beyond two-way are beneficial for small cache sizes, but are ineffective at avoiding this kind of block conflict miss among large caches. The results for the Guitar scene are similar since the textures are accessed along a variety of paths and the triangles are large. The results for the Flight scene are also similar except that most of the conflicts are avoided when the caches are eight-way set-associative and this is attributed to the fact that the triangles are moderately sized.

In summary, the blocked representation prevents conflicts from occurring between neighboring texels. Conflicts are also prone to occur between blocks. Conflicts between blocks at adjacent levels of the Mip Map can be successfully eliminated with two-way set-associative caches. Conflict misses that occur within the same 2D array are harder to prevent because the textures can be accessed along any path. In the next section, we discuss a tiled rasterization order that has the effect of reducing the number of blocks in the working set that can conflict with each other.

# 6 Rasterization Order and Tiling in Screen Space

The order in which screen pixels are traversed to determine whether they lie within the boundary of a triangle is the rasterization order. The rasterization order effects the texture access pattern and consequently, it can influence the cache behavior. As can be seen in Figure 6.1(a), the rasterization path of row major order spans the entire width of the triangle. As a result, the working set size for row major order is proportional to the amount of texture needed for an entire row of fragments; thus the working set size is related to the dimensions of the triangles that are mapped onto the screen. We propose reducing this variance, and thus the cache size needed for textures, by the use of a tiled rasterization order.

In a tiled order, a spatially contiguous block of screen pixels is traversed consecutively. Figure 6.1(b) illustrates a tiled rasterization path where the screen is statically decomposed into tiles. The cost associated with tiling depends on the rasterization algorithm. Additional setup computations for each tile may be required if the rasterization, shading and texture parameters are computed incre-

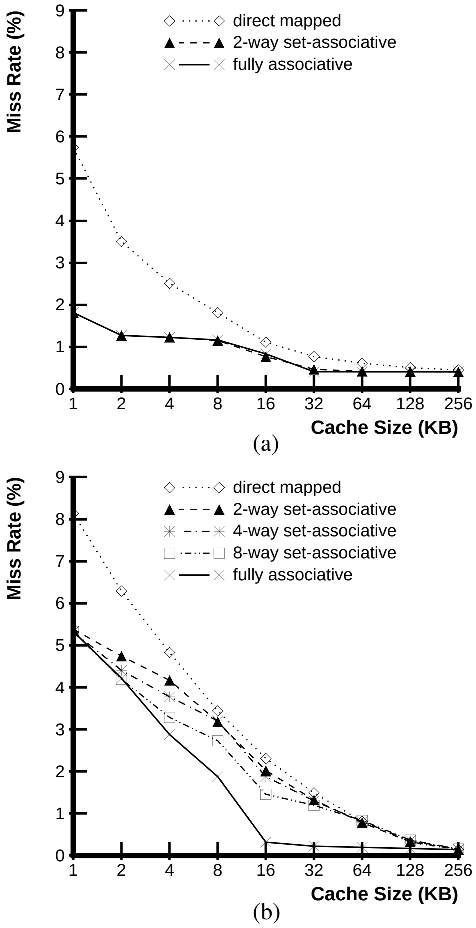


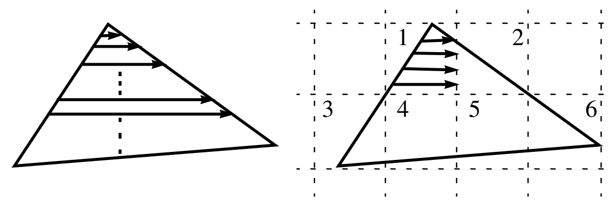

**FIGURE 5.7. Effect of cache associativity on conflict misses. The textures are stored in blocks of 8x8 texels and the cache line size is 128 bytes. (a) Goblet-horizontal. (b) Town-vertical.**

mentally across the surface of a triangle. One factor that has considerable influence on the texture access pattern is the tile size. In the following two sections, we address the issues of selecting a tile size and study the implications of tiled rasterization with respect to texture representation and cache organization.

## 6.1 Selecting a Tile Size

To understand the interaction between the tile dimensions and the working set size, we show miss rate results versus cache size in Figure 6.2 for the Guitar scene. The measurements were taken with a blocked texture representation and a relatively large line size of 128 bytes to take advantage of spatial locality. In Figure 6.2(a), we find that as we progress from very small tiles to medium tiles, there are significant reductions in miss rate amongst cache sizes that previously did not fit the working set. It is evident that tiling causes the working set size to be reduced and this shows up as fewer capacity misses. Figure 6.2(b) shows that the opposite effect occurs as we progress from medium tiles to very large tiles for the same scene. When the tiles are very small, the texture access patterns converge towards the access pattern of a nontiled rasterization order. On the other hand, when the tiles are very large, the working sets are larger than the cache size and capacity misses dominate the cache behavior.

We have found that tiled rasterization has a similar effect on the results of the Town scene. One characteristic that is common to both the Guitar and Town scenes is that they include large triangles that span significant areas of the screen. When the triangles are




(a) (b)

**FIGURE 6.1. (a) Illustration of nontiled horizontal rasterization, (b) Illustration of tiled rasterization. The arrows indicate the rasterization path within a tile and the numbers indicate the order in which the tiles are rasterized.**

moderately sized, as in the Flight scene, the effect of tiled rasterization on the working set size is less pronounced, and when the triangles are small, as in the Goblet scene, the working set size becomes completely unaffected by the tile dimensions. For scenes with small triangles, it is advantageous to render neighboring triangles that lie in the same tile consecutively to exploit spatial locality and thus ensure that the working set size is minimized. We can conclude that tiled rasterization does not hurt when the triangles are small (and is thus robust), while in frequently occurring situations where the triangles are large it helps substantially in reducing working set size.

## 6.2 Effect of Tiling on Conflict Misses

In Section 5.3.3, we found that conflicts between blocks that are in the same level of a Mip Map are difficult to avoid because the direction of texture accesses can be arbitrary. Assuming that neighboring blocks do not map to the same line in the cache, tiled rasterization can reduce the occurrence of this kind of block conflict since it confines the working set to a spatially contiguous region of texture. Unfortunately, because the texture image dimensions are powers of two and the memory required for a single row of texture blocks can be a multiple of the cache size, it is likely that conflicts will arise between neighboring blocks in the same column. One way to ensure that neighboring blocks do not conflict is to place a number of unused pad blocks at the end of each row of blocks as illustrated in Figure 6.3(a). This scheme requires additional texel addressing calculations which are shown below.

**bw**, **bh**: block width and height in texels. These are equal and are powers of two.

**ps**: $\log_2$(bw * bh * number of pad blocks). Number of pad blocks is a power of two.

**by**: block row coordinate

Texel address = Texel address computed for blocked representation + (by << ps)

The shift operation shown above has a constant shift amount assuming that the block dimensions and the number of pad blocks remain fixed. Hence, the additional hardware overhead of padding is just one addition per texel addressing calculation. Padding also incurs a memory overhead, though this overhead tends to be negligible for large textures where padding is most needed.

Another way to ensure that neighboring blocks do not conflict is to use another level of blocking in the representation of the texture images as illustrated in Figure 6.3(b). In this scheme, the two-dimensional texture arrays would effectively be stored as six-dimensional arrays. The size of the coarser blocks should be equal to the cache size since this ensures that a square region of blocks can be mapped into the cache without any conflicts. The additional hardware overhead of another level of blocking is two additions per texel addressing calculation.

Figure 6.4 shows the effect of tiled rasterization on conflict

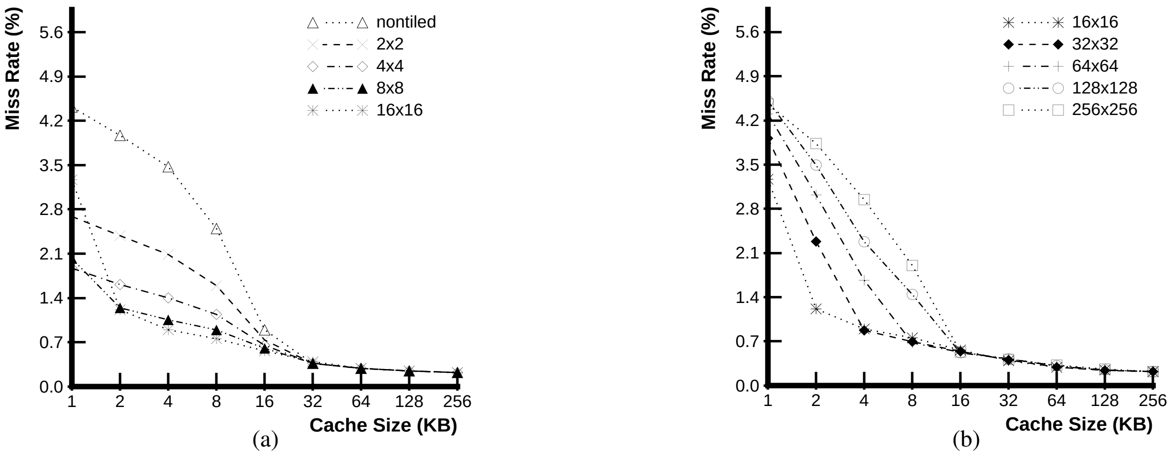



**FIGURE 6.2. Effect of tiled rasterization on the working set size. These results were measured for the Guitar scene using fully associative caches. The textures are stored in blocks of 8x8 texels and the cache line size is 128 bytes. The tile dimensions are given in pixels. (a) Small to medium tile sizes. (b) Medium to large tile sizes.**

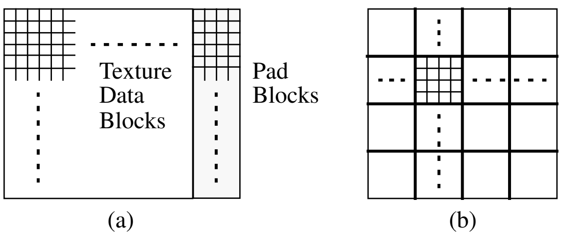



**FIGURE 6.3. (a) Illustration of a 4D blocked and padded representation for one level of a Mip Map. We use the same number of pad blocks at the end of each row of blocks for all levels of a Mip Map. (b) Illustration of a 6D blocked representation for one level of a Mip Map. The size of the finer 2D blocks is selected to be equal to the cache line size and the size of the coarser 4D blocks is selected to be equal to the cache size.**

misses for the Town and Flight scenes. Comparing the tiled rasterization results for the Town scene in Figure 6.4(a) with the nontiled rasterization results previously shown in Figure 5.7(b), we find that the rate of conflicts is quite diminished with tiled rasterization. As expected, the effect of tiled rasterization on block conflict misses in the Guitar scene is very similar. Even though the Flight scene uses moderately sized triangles, it is highly prone to conflicts between neighboring blocks that lie in the same column because of the relatively large textures used for the terrain images. Indeed, Figure 6.4(b) for the Flight scene shows that tiled rasterization by itself is not sufficient for avoiding block conflict misses. However, when tiled rasterization is combined with either padding or 6D blocking, the rate of block conflict misses is significantly reduced. We can conclude that tiled rasterization effectively limits the number of blocks in the working set that can conflict with each other and that either padding or 6D blocking is needed to ensure that conflicts do not occur between neighboring blocks.

# 7 Memory Bandwidth and Rendering Performance

Perhaps the two most important metrics that characterize a texture mapping system are rendering performance and memory bandwidth. In this section, we discuss the issues in texture caching that effect rendering performance and after making some assumptions about the machine model, we relate the miss rates to memory bandwidth.

## 7.1 Machine Model

As previously shown in Figure 2.1, the texture mapping hardware consists of two components: a fragment generator and an SRAM texture cache. We discuss the details of each of these components separately.

### 7.1.1 Fragment Generator

The fragment generator is responsible for performing a fairly large number of calculations. As mentioned earlier, these include rasterizing triangles into fragments, computing level-of-detail and texture addresses for each fragment, trilinear interpolation of texels accessed from a Mip Map, and finally, applying the filtered texture to the fragments. To accomplish all of these tasks, we assume that the fragment generator exploits parallelism and pipelining to a great extent. We also assume that the clock frequency is 100 MHz since this is representative of present-day ASIC technology.

An appropriate measure of rendering performance is the number of textured fragments per second. One factor that can limit the maximum bandwidth achieved is the number of texels that can be accessed from the cache per cycle. Considering that a trilinear interpolation requires eight distinct texels to be accessed from the cache, a system that accessed just one texel per cycle would be limited to 12.5 million textured fragments per second. It is apparent that to achieve higher performance, more than one texel must be accessed per cycle. This issue is discussed further in Section 7.1.2. For now, we assume that the machine can read four texels per cycle, leading to a maximum bandwidth of 50 million textured fragments per second. The computation time is factored out by the pipelined nature of the system.

Another factor that can effect the performance of the system is the latency associated with filling a cache line from memory when a cache miss occurs. Even though the memory latency tends to be very long (roughly fifty 10ns cycles for a 128 byte cache line), it still must be completely hidden to achieve the maximum rate of fragments textured per second. Note that the memory latency would constrain the performance of the system if it were not hidden and that this reduction of fragment bandwidth becomes more pronounced as we increase the clock rate or the number of texel accesses per cycle. Another incentive for hiding the memory latency is the notion of robustness of performance with respect to different scenes. Some applications of real-time graphics require a high level of performance even under the worst-case conditions. A texturing system can sustain the maximum rate if the memory

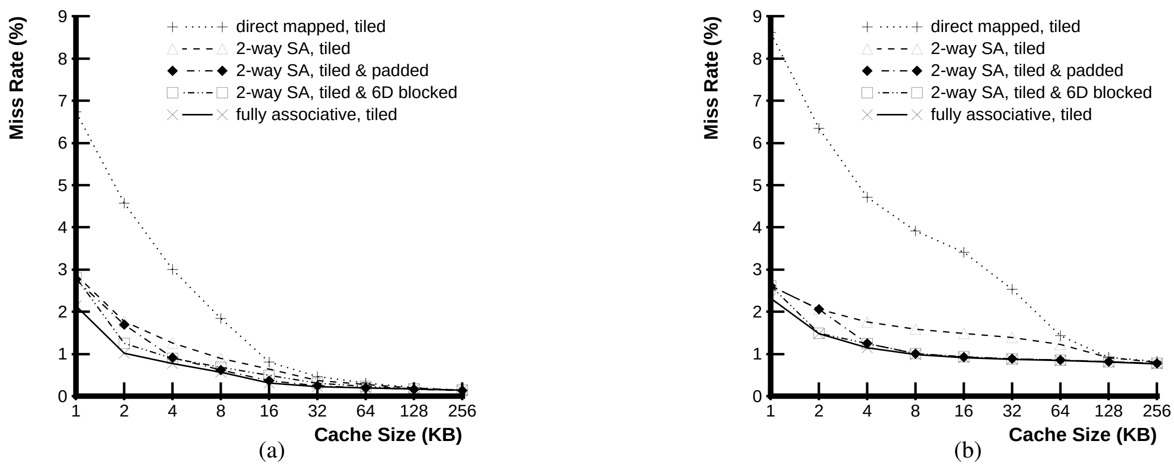



**FIGURE 6.4. Effect of tiled rasterization on conflict misses. The textures are stored in blocks of 8x8 texels and the cache line size is 128 bytes. The tiles are 8x8 pixels. The padded results use a pad of four blocks at the end of each row of texture blocks. For the 6D blocked results, we use the largest block size for the coarser blocks that is less than or equal to the cache size.**
**(a) Town scene (rasterization is in column major order within and between tiles). (b) Flight scene.**

latency is completely hidden and the memory bandwidth is met.

One solution for hiding the memory latency is proposed in [13]. The basic idea is to compute the texel addresses far in advance of the cache accesses by rasterizing the triangles twice: the first time to compute the texel addresses and to prefetch the lines that are missing in the cache and the second time to perform fragment texturing and shading. Since the texel addresses are needed by both rasterizers, they can either be computed independently by each rasterizer or they can be passed from the first rasterizer to the second by means of a FIFO buffer. The former approach is likely to be more costly because of the large number of calculations needed to compute the texel addresses (please refer to Table 2.1).

### 7.1.2 SRAM Texture Cache

As mentioned in the previous section, it is important to be able to access more than one texel from the cache in the same cycle. A common way of designing a multi-ported cache is to interleave the cache lines across multiple independently addressed banks [8]. Since a trilinear interpolation involves accessing neighboring texels, the interleaving across banks in a texture cache must be at the granularity of a texel rather than a cache line. A conflict-free address distribution which allows up to four texels to be accessed in parallel is possible if the texels are stored in a morton order within the cache lines [3]. Morton order implies that the texels are stored in 2x2 blocks. The texels within each 2x2 block are interleaved across the four banks and the same interleaving pattern is used for all 2x2 blocks that lie within a cache line to ensure that adjacent texels in abutting blocks are assigned to different banks.

While it may seem from Table 2.1 that the number of calculations for texture mapping is sufficiently high that the computation rather than cache bandwidth is the bottleneck, in practice this is not the case. Comparing the number of calculations required in each phase of texture mapping, we note that the trilinear interpolation portion is the most computationally-intensive. The nature of this computation is such that cache bandwidth is critical to performance. Consider the core of a trilinear interpolation, which must be performed separately for each R,G,B,A color component:

Interpolated value = Texel(n) + Weight * ( Texel(n+1) - Texel(n) )

We observe that this calculation requires pairs of texel values before it can proceed. Limiting the number of texel accesses to one per cycle constrains the rate of subtractions to one every other cycle. In contrast, a two-ported cache would allow subtractions every cycle. One might also consider adding more than two cache ports so that this calculation can be performed in parallel on different pairs of texels. Since the trilinear interpolation phase is the bottleneck in the fragment generator pipeline, improving its performance would directly impact the overall performance of a fragment generator. While we have been mostly focussing on special-purpose hardware, recent additions of visualization instructions such as MMX [19] and VIS [16] and deep pipelines have shifted the bottleneck away from computation in microprocessors.

## 7.2 Memory Bandwidth

The memory bandwidth results assuming that the system operates at peak bandwidth of 50 million textured fragments per second are shown in Table 7.1. We report results for three different cache sizes: 4KB, 32KB, and 128KB. The 4KB cache is representative of a very small on-chip cache. The 32KB cache is large enough to hold the working sets, yet it can also be placed on-chip. Both the 4KB and 32KB caches are two-way set-associative. The 128KB cache size provides a measure of how low the miss rates and memory bandwidths can become if a very large cache is used. This cache is direct mapped since associativity makes little difference in miss rates at this size. The results were measured for the blocked representation since we previously found this texture representation to be most suitable. In addition, the texture arrays are padded and the rasterization in screen space is tiled to reduce conflict misses and working set size.

One trend that can be seen in these results is that the memory bandwidth requirements are much reduced in the transition from the 4KB cache to the 32KB cache for all scenes except the Flight scene where the reductions are more modest. The transition to the 128KB cache leads to less drastic reductions in memory bandwidths demonstrating that fairly small cache sizes are adequate for texture mapping. Another trend is that the memory bandwidth requirements increase very slightly with increasing line size for the 32KB and 128KB caches. This result is encouraging because larger line sizes can elicit a larger fraction of the peak memory bandwidth.

We would like to compare the memory bandwidth requirements discussed above for a fragment generator that uses an SRAM texture cache with the memory bandwidth requirement of a fragment generator of equivalent performance that directly accesses a dedicated DRAM memory system for all texel lookups. For the latter system, which does not use a cache, the memory bandwidth

| Cache Size Associativity | | 4KB 2-way Set-Associative | | | 32KB 2-way Set-Associative | | | 128KB Direct Mapped | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Line Size Block Size | | 32 4x4 | 64 4x4 | 128 8x8 | 32 4x4 | 64 4x4 | 128 8x8 | 32 4x4 | 64 4x4 | 128 8x8 |
| Scene | Flight | **396** (3.24) | **447** (1.83) | **610** (1.25) | **355** (2.91) | **386** (1.58) | **435** (0.89) | **339** (2.78) | **366** (1.50) | **425** (0.87) |
| | Town | **233** (1.91) | **271** (1.11) | **444** (0.91) | **99** (0.81) | **103** (0.42) | **122** (0.25) | **77** (0.63) | **78** (0.32) | **88** (0.18) |
| | Guitar | **319** (2.61) | **371** (1.52) | **552** (1.13) | **154** (1.26) | **161** (0.66) | **215** (0.44) | **120** (0.98) | **125** (0.51) | **137** (0.28) |
| | Goblet | **385** (3.15) | **566** (2.32) | **596** (1.22) | **189** (1.55) | **212** (0.87) | **225** (0.46) | **194** (1.59) | **215** (0.88) | **229** (0.47) |

**TABLE 7.1. Memory bandwidth requirements in MBytes/second for different cache and line sizes. The performance of the system is 50 million textured fragments/second. The miss rates are also shown in parentheses. The texture maps are stored in a blocked and padded representation in memory. The block dimensions are given in texels and the pad is four blocks at the end of each row of texture blocks. The rasterization is tiled in screen space using 8x8 pixel tiles. Texels are 32-bits.**

requirement is 1.5 GBytes/second (4 bytes/texel * 8 texels/fragment * 50M fragments/second). On the other hand, the memory bandwidth requirement for a system that uses a 32KB cache is between 100 and 450 MBytes/second. We see that there is a three to fifteen times reduction in memory bandwidth requirements due to caching.

Although the main focus of this paper has been on a graphics rendering system with a single fragment generator, it is interesting to note that the memory bandwidths are low enough that a parallel system could be built with multiple fragment generators sharing a single texture memory, each with their own cache. As mentioned earlier, this makes it possible to avoid replicating the textures in each fragment generator memory as is done in the RealityEngine.

# 8 Conclusions and Future Work

As we move into the era of the Internet with 3-D virtual chat groups, realistic visualization of scientific phenomenon, photorealistic computer games, the need to provide high performance yet cheap computer graphics is becoming critical. While much attention has been given in the past to design of memory hierarchies for general-purpose computers using the SPEC benchmarks and transaction processing benchmarks, so far there is little published data available on core graphics algorithms like texture mapping. This paper attempts to fill that gap.

We have built a sophisticated environment in which computer graphics workloads can be rendered, and our simulation results show that caches can be highly effective for texture-mapped graphics. Traditionally, caches have not been used in computer graphics systems where the philosophy has been to provide guaranteed performance under worst-case conditions, although this philosophy is beginning to change [13]. By using techniques such as (i) block-based representation of Mip-Mapped textures, (ii) tiled rendering in the screen space itself, and (iii) padded or six-dimensionally blocked texture arrays, we can robustly reduce the miss rates. We study the relationship between block size in texture representations and cache line size, and also discuss which tile sizes are appropriate. We observe that caches as small as 16 KBytes with 2-way associativity (to reduce conflict miss rates for Mip Map levels) can reduce the effective bandwidth needed from the memory system by a factor of three to fifteen, while also reducing the latency of access. Because of the above tiling techniques, the performance can be made robust regardless of the scenes that are being texture mapped.

A promising approach for rendering directly from compressed textures has been proposed in the literature [2]. In future work, it would be interesting to study the interaction between compressed representations of textures and cache architectures. Another area which we have not studied in detail is the use of texture caching in parallel architectures. One of the interesting questions that must be addressed in this area is how to balance the work among multiple fragment generators without reducing the spatial locality in each reference stream.

# Acknowledgments

We would like to thank Vijayaraghavan Soundararajan, Gordon Stoll, and especially Pat Hanrahan for helpful discussions and valuable comments on earlier drafts of this paper. We are also grateful to John Gerth for his help and to Homan Igehy for contributing the Guitar scene. The Flight and Town scenes appear courtesy of Silicon Graphics and Paradigm Simulation. This work was partly supported by DARPA under grant No. DABT63-95-C-0085-P00000.

# References

[1] Accelerated Graphics Port (AGP). *Accelerated Graphics Port Interface Specification*, Revision 1.0. Intel Corporation, 1996.

[2] A. C. Beers, M. Agrawala, and N. Chaddha. Rendering from Compressed Textures. In *Proceedings of SIGGRAPH'96*, pages 373-378, August 1996.

[3] A. Schilling, G. Knittel, and W. Strasser. Texram: A Smart Memory for Texturing. In *Computer Graphics and Applications*, pages 32-41, May 1996.

[4] Darwyn Peachey. Texture on Demand. Technical Report, Pixar, 1990.

[5] Ed Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, University of Utah, Dec. 1974.

[6] E. Rothberg, J. P. Singh, and A. Gupta. Working Sets, Cache Sizes, and Node Granularity Issues for Large-Scale Multiprocessors. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pages 14-25, May 1993.

[7] Geoffrey Y. Gardner. Visual Simulation of Clouds. In *Proceedings of SIGGRAPH'85*, pages 297-303, July 1985.

[8] Gurindar S. Sohi and Manoj Franklin. High-Bandwidth Data Memory Systems for Superscalar Processors, In *Fourth Int'l. Conference on Architectural Support for Programming Languages and Operating Systems*, pages 53-62, April 1991.

[9] James F. Blinn. Simulation of Wrinkled Surfaces. In *Proceedings of SIGGRAPH'78*, pages 286-292, August 1978.

[10] James F. Blinn. The Truth About Texture Mapping. In *Computer Graphics and Applications*, pages 78-83, March, 1990.

[11] J. Foley, A. van Dam, S. Feiner, J. Hughes. *Computer Graphics Principles and Practice*, Second Edition. Addison-Wesley Publishing Company, Inc., 1990.

[12] James L. Turley. Action-Packed Fray for Newest Video Games. *Microprocessor Report*, pages 15-20, May 30, 1995.

[13] Jay Torborg, and James T. Kajiya. Talisman: Commodity Realtime 3D Graphics for the PC. In *Proceedings of SIGGRAPH'96*, pages 353-363, August 1996.

[14] Kurt Akeley. RealityEngine Graphics. In *Proceedings of SIGGRAPH'93*, pages 109-116, August 1993.

[15] Lance Williams. Pyramidal Parametrics. In *Proceedings of SIGGRAPH'83*, pages 1-11, July 1983.

[16] L. Kohn, G. Maturana, M. Tremblay, A. Prabhu, and G. Zyner. Visual Instruction Set (VIS) in UltraSPARC™. In *Proceedings of COMPCON'95*, pages 462-469, March 1995.

[17] Mark Segal, and Kurt Akeley. *The OpenGL Graphics System: A Specification*, Version 1.0. Silicon Graphics, Inc., 1992.

[18] M. F. Deering, S. A. Schlapp, and M. G. Lavelle. FBRAM: A new Form of Memory Optimized for 3D Graphics. In *Proceedings of SIGGRAPH'94*, pages 167-174, July 1994.

[19] MMX™ Technology. *Intel Architecture MMX Technology Programmer's Reference Manual*. Intel Corporation, March 1996.

[20] M. Segal, C. Korobkin, R. van Widenfelt, J. Foran, and P. Haeberli. Fast Shadows and Lighting Effects Using Texture Mapping. In *Proceedings of SIGGRAPH'92*, pages 249-252, July 1992.

[21] Ned Greene. Applications of World Projections. In *Proceedings of Graphics Interface '86*, pages 108-114, May 1986.

[22] Paul Haeberli and Mark Segal. Texture Mapping as a Fundamental Drawing Primitive. In *Proceedings Fourth Eurographics Workshop on Rendering*, pages 259-266, June 1993.

[23] Paul S. Heckbert. Survey of Texture Mapping. In *Proceedings of Graphics Interface '86*, pages 207-212, May 1986.

[24] Steven A. Przybylski. *New DRAM Technologies: A Comprehensive Analysis of the New Architectures*, Second Edition. MicroDesign Resources, 1996.

[25] W. T. Reeves, D. H. Salesin, and R. L. Cook. Rendering Antialiased Shadows with Depth Maps. In *Proceedings of SIGGRAPH'87*, pages 283-291, July 1987.

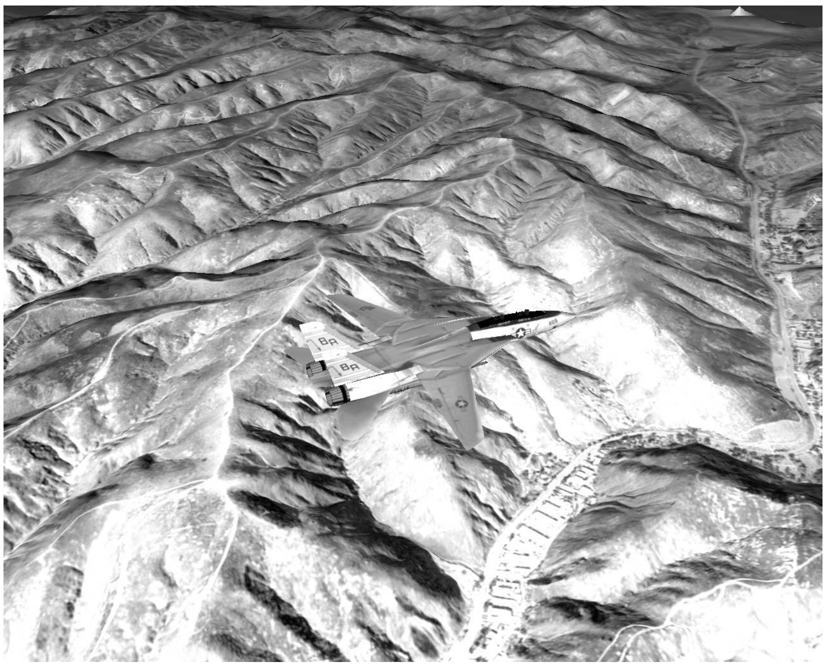


**FIGURE 4.1. Flight scene**

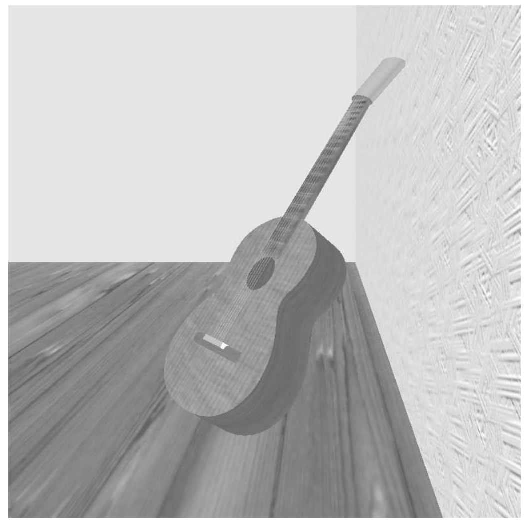


**FIGURE 4.3. Guitar scene**

**FIGURE 4.2. Town scene**

**FIGURE 4.4. Goblet scene**