PowerVR™
The Future of 3D Graphics Technology

Technical Backgrounder

5/98 V.10

Video games have been around for several years and improve substantially with each generation. Early video game machines and their associated games titles were restricted by what is known as two-dimensional graphics. The images that such machines or games provide are generally "flat" and lack any notion of depth (*z*-dimension). For example, in a two-dimensional game, the player can only move in *x* and *y* dimensions and cannot look "behind" objects in a scene. Later machines added the ability to process three-dimensional graphics, where the player has total freedom of movement and can go "around" and "behind" objects, much like in real life.

The processing power and graphics capabilities needed to migrate from two-dimensional to three-dimensional graphics is substantial. In a two-dimensional game, two-dimensional images called *bitmaps* are moved from one area of memory (the source) to another (the destination). Such bitmaps may be combined with other bitmaps or with the destination bitmap to create special effects. Bitmap moves, often referred to as *bitblit*, can also be made conditional on a pixel-by-pixel basis, depending on the destination and source pixels and their relative values.

In a three-dimensional environment, the entire database representing the modeled "world" is processed for every frame displayed. An observer can be looking at and manipulating the world from a new position and direction, but the relative position and orientation of each object is recalculated in every frame. This operation is called *geometry transformation*.

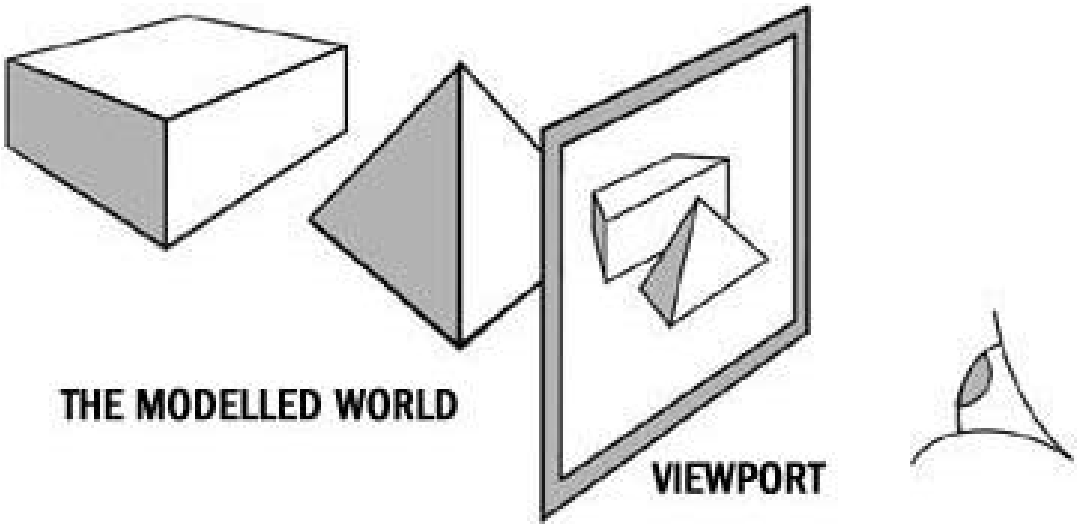

Figure 1. The Basics of 3D Graphical Processing

Once the new coordinate of every object is known, the effective view as seen by the observer must be generated, often by projecting the three-dimensional world onto a viewport that constitutes the image seen by the observer. To produce the correct image, it must be known which object or part of each object is visible in the viewport. This operation is generally known as *hidden surface removal*.

Conventional Hidden-Surface Removal Techniques

Many three-dimensional rendering techniques use a z-buffer concept, whereby objects used to construct the three-dimensional world are normally composed of triangles, or *polygons*. For example, the pyramid (tetrahedron) in Figure 1 consists of four polygons. The cube as shown in Figure 1 can be divided into 12 triangles by separating each quad side into two triangles. Each polygon is then drawn, pixel by pixel, into the display buffer, and then displayed on the screen.

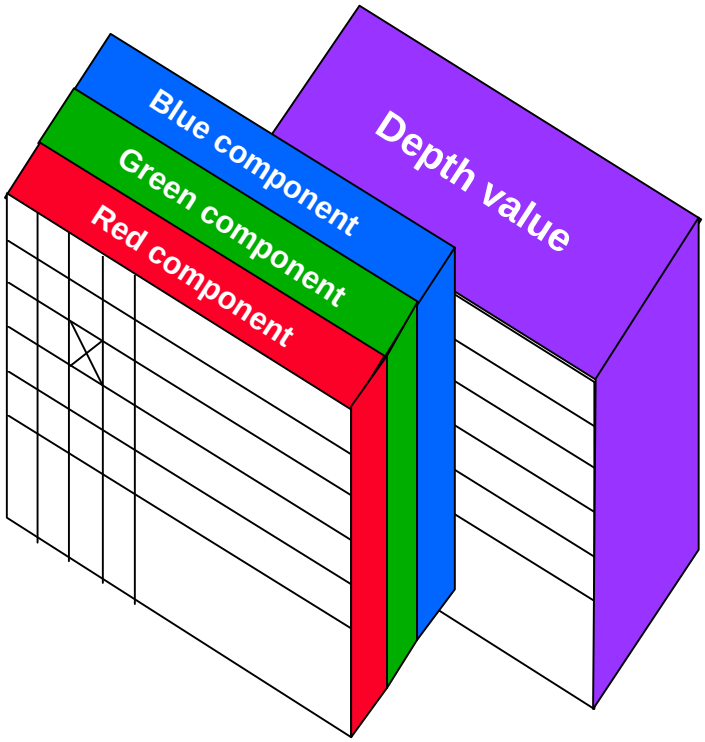

Figure 2. Representation of Z Buffer

During this process, polygons or portions of polygons drawn on to the screen are overwritten if follow-on polygons are closer to the observer. Similarly, if polygons or a portion of the polygons to be drawn are behind previously drawn polygons, the associated pixels are ignored and not drawn. To achieve hidden-surface removal, many applications store a z value in addition to a color value for each pixel on the screen. This z value represents depth and is used to determine whether pixels from a newly drawn polygon are in front of or behind the current pixel. Figure 2 shows how each pixel is expressed in red, green and blue components and also in a z value stored in the z-buffer.

Some systems avoid using a dedicated z-buffer by breaking down all intersecting polygons first, allowing intersecting polygons to be simply drawn according to depth order. Compared to systems using a z-buffer, these systems have reduced performance because the CPU must perform the additional polygon partitioning and presorting.

Conventional Texturing and Shading

In addition to hidden-surface removal, when polygons are written into the frame buffer, pixel values must be modified to take account of texture, shading and lighting characteristics.

Textures are stored in memory and relevant pixels from the texture map, called *texels*, are retrieved and used to texture each pixel before it is written into the frame buffer. Depending on the texturing technique, each output pixel requires a different number of texels. The simplest technique, called *point sample texturing*, uses a single texel from the texture map to best approximate a pixel's texture, depending on the orientation of the associated polygon. Point sampling can result in the same texel value being used for several adjacent output pixels, depending on the orientation of the polygon. This technique sometimes causes “blockiness” of the resulting textured polygons.

More advanced features interpolate between many texels to more accurately estimate the texture for the output pixel. For example *bilinear texturing* uses four adjacent texels to interpolate the output pixel value, resulting in a smoother-textured polygon as the interpolation smoothes the blockiness associated with point sampling. The disadvantage of bilinear texturing is that it increases the memory bandwidth required to retrieve texture data by up to four times.

Another advanced texturing technique is *MIP mapping*, where low- and high-resolution versions of the same texture are mixed to produce the final value. This technique is particularly useful in removing the aliasing problems that occur when textures are mapped onto acute-angled polygons that disappear into the distance. Aliasing occurs when there are insufficient pixels to faithfully represent sharp variations in the image. Anti-aliasing is important for modeling sky or flat landscapes in which a single resolution is used and image discontinuities occur due to aliasing at certain depths. Varying the mix of low- and high-resolution textures, depending on depth, greatly minimizes this effect.

It is possible to combine point sampling and bilinear techniques within a MIP map approach. *Linear MIP mapping* refers to a case where point sampling is used for both MIP maps when two texels, one from each MIP map, are accessed and combined for every output pixel. T*rilinear texturing* is where bilinear interpolation is used for both MIP maps, resulting in eight texel accesses per output pixel. Even more advanced techniques are now appearing e.g. anisotropic texturing where typically sixteen texels are sampled depending on the polygon orientation to achieve a very high level of image sharpness without introducing aliasing.

Figure 3 shows a basic three-dimensional rendering pipeline, with particular emphasis on memory access requirements. Front-end geometry transformation is omitted for simplicity.

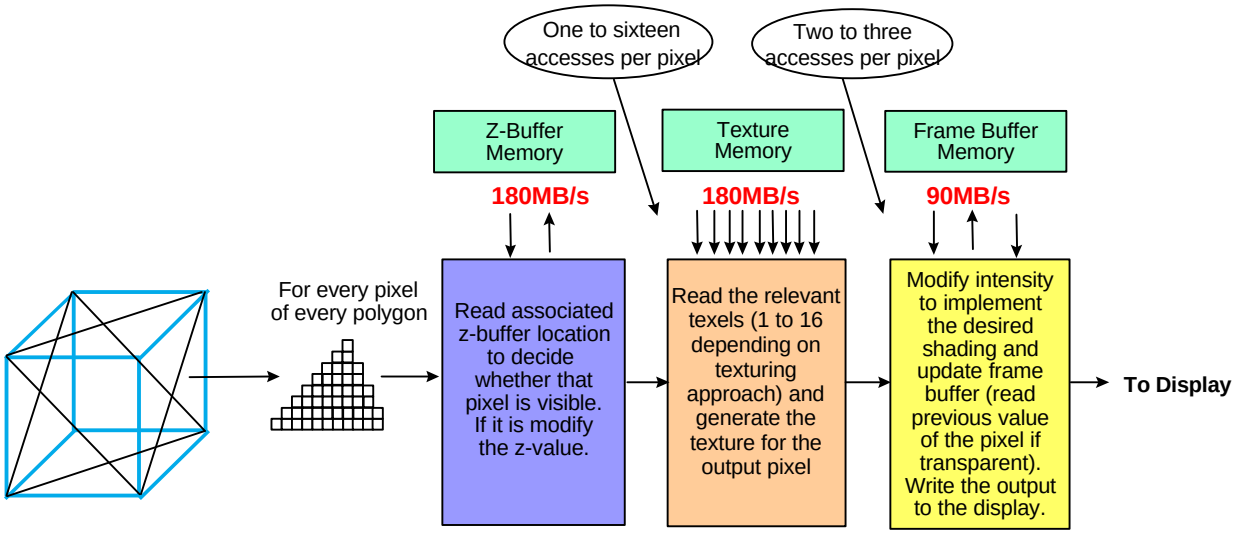

Figure 3. Data Flow in a Simplified Rendering Pipeline at 500K polygons/sec

In addition to adding textures to polygons, it is often desirable to model surfaces that appear smooth and curved. Processing curved surfaces is very complicated; modifying the pixels' colour so that the polygons appear curved and smooth to the observer is a simpler approach that takes into account the position of light sources in the scene relative to the polygon surface. This technique is used to modify the colour of reflected light at each pixel on the surface, as if the surface normals at each pixel were those of a curved surface passing through vertices of the polygon.

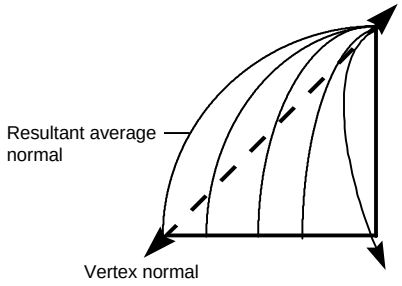

Figure 4. Colour of Reflected Light at Each Pixel (Smooth Shading)

Figure 4 illustrates the basics of smooth shading, where surface normals at vertices are defined and are linearly interpolated (averaged) across each edge and along each display line. The resulting “simulated” curved surface is used in calculating the colour of reflected light across the polygon. Unlike z-buffering and texturing, the shading operation does not require memory accesses.

Once the output pixel is shaded and textured, it is written into the display frame buffer. If the polygon has any degree of transparency, then the previous value of the same pixel is read from the frame buffer and combined with the new value before being written.

Once all the polygons are processed, the newly generated image is output to the display. This operation is often done in a double-buffered manner to overlap the image display and rendering of the next frame. Double buffering eliminates visual cut lines generated because of the difference between display refresh and rendering frame rate. Figure 3 illustrates the concept of a standard approach to three-dimensional rendering operations including z-buffering, texturing, shading, translucency and frame buffer management.

Limitations of Today's Approaches

Three-dimensional systems using conventional approaches suffer from a number of limitations.

Massive Memory Bandwidth Requirement

There are many accesses into the z-buffer and the texture and frame buffers for each polygon (Figure 3). Assuming a 24-bit z-buffer, 24-bit textures, 24-bit pixels, and a 800x600x72Hz refresh rate, these formulas can be used to calculate required bandwidth.

Z-Buffer Bandwidth (Bytes/Second)	Texture Bandwidth (Bytes/Second)	Frame Buffer Bandwidth (Bytes/Second)	Total Bandwidth Required
6xPPxPZ	3xPPxPZ (note 3) 6xPPxPZ (note 4) 12xPPxPZ (note 5) 24xPPxPZ (note 6) 48xPPxPZ (note 7)	3xPPxPZ + 104 MB/sec	12xPPxPZ+104 MB/sec 15xPPxPZ+104 MB/sec 21xPPxPZ+104 MB/sec 33xPPxPZ+104 MB/sec 57xPPxPZ+104 MB/sec

Table 1. Bandwidth Requirements for Conventional Three-Dimensional Systems

Notes:
(1) PZ = polygon size in pixels (average polygon size of 100 pixels)
(2) PP = polygon-per-second performance
(3) Point sampling
(4) Linear MIP mapping
(5) Bilinear interpolation
(6) Trilinear texturing
(7) Anisotropic texturing

Table 2 shows the results of these calculations.

Case	500k Polygons/Second	1M Polygons/Second	2M Polygons/Second
Point sample	704 MB/sec	1304 MB/sec	2504 MB/sec
Linear MIP mapping	854 MB/sec	1604 MB/sec	3104 MB/sec
Bilinear interpolation	1154 MB/sec	2204 MB/sec	4304 MB/sec
Trilinear texturing	1754 MB/sec	3404 MB/sec	6704 MB/sec
Anisotropic texturing	2954 MB/sec	5804 MB/sec	11504 MB/sec

Table 2. Examples of Bandwidth Requirements (Average Polygon Size of 100 Pixels)

It is clear from these examples that the bandwidth requirement for conventional three-dimensional rendering systems is substantial. Furthermore, this requirement increases as the performance level or quality of texturing technique improves.

Memory Consumption

Conventional three-dimensional solutions require memory systems for the z-buffer, texture storage and frame buffer. In addition, many implementations use multi-bank memory systems to optimize data transfer since the bandwidth needed for each one of these memories, particularly texture storage and the z-buffer, is very high and increases with performance. Memory subsystems have necessarily become the most costly part of today's high-performance, three-dimensional rendering systems. In fact, the cost of processing devices is often only a fraction of the cost for required memory components. To achieve a cost-effective three-dimensional graphics solution, therefore, it is essential to eliminate memory requirements wherever possible.

Performance for Cost Sensitive systems Limited by Memory Bandwidth

Given their memory bandwidth and cost, conventional three-dimensional systems cannot deliver both low cost and high performance. Low-cost systems must use cost-effective DRAM memories and combine z-buffer, texture memory and frame buffer memory wherever possible to share the expensive resources. It is therefore not surprising that systems making such compromises do not deliver the required performance.

Poor Platform Scalability

Given the nature of the algorithms used in conventional three-dimensional rendering systems, and the hugely different memory bandwidth requirements, none of today's conventional approaches are scalable. It is therefore impossible to use the same solution for a low-end consumer application and a high-end arcade platform. Such scalability is highly desirable because the very best titles are often developed on arcade platforms and then migrated to a consumer PC implementation

Limited Special Effects and Realism

Some of the very powerful and important special effects need a fundamentally different approach in order to achieve low overhead and cost effective implementations. An example of such capabilities is the ability to modify an object's appearance on screen dynamically dependant on it's interaction with other objects in the scene e.g. a car being cast into shadow when passing under a tree. Such features bring a new dimension to games and cannot be effectively implemented using the conventional polygon based approach. Mechanisms that can better model the real world are needed to address these issues.

PowerVR Breaks All Conventional 3D Barriers

A traditional three-dimensional approach cannot provide both low cost and high performance. PowerVR technology, a joint development between VideoLogic and NEC Corporation, minimizes the amount of memory used *and* the bandwidth of the memory system. As shown in Figure 3, in conventional three-dimensional approaches, two memory subsystems (texture memory and frame buffer memory) hold *real* image data, while the third (z-buffer) temporarily holds depth history during rendering of each frame.

Architecture and Design Objectives

To minimize memory requirements, PowerVR was designed to eliminate the requirement for z-buffer memory. To ensure that maximum performance is maintained, hidden-surface removal is implemented entirely in hardware. This saves z-buffer costs and removes the bottleneck associated with accessing off-chip memory. Furthermore, since hidden-surface removal is performed fully on-chip, calculations are performed at the clock speed of the device and can benefit from continuous improvements to silicon technology. The first generation PowerVR chips (NEC PCX1 and PCX2) were the first devices to use this innovative architecture. Recently the second generation PowerVR technology was introduced which included future significant enhancements to the basic PowerVR architecture.

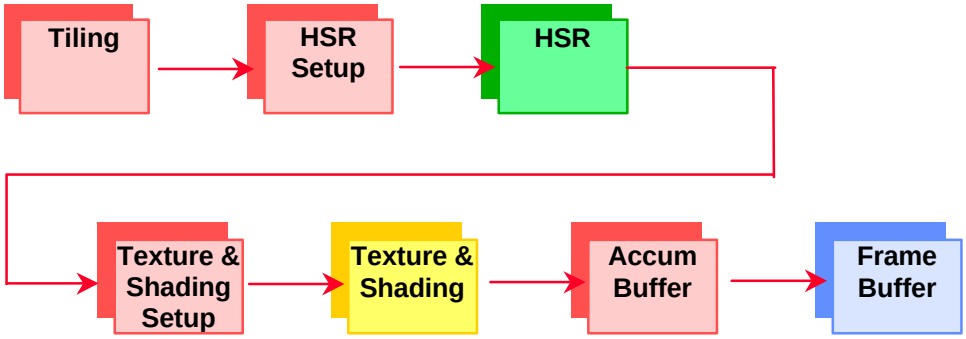

Figure 5. PowerVR Second Generation Rendering Pipeline

PowerVR also substantially reduces the bandwidth requirements of the texture and frame buffers. Texture bandwidth is minimized using *deferred texturing*. Unlike a conventional rendering system where every polygon is textured and then potentially overwritten, the PowerVR approach only textures finally visible pixels (polygons or portions of polygons).

This deferral is possible through innovative hidden-surface removal techniques which outputs pixel groups (tiles) fully resolved in depth and constituting visible pixels only. During the rendering of each frame, every pixel on the screen is only textured once and each output pixel is written to the frame buffer only once. Conversely, in a conventional system, every pixel on every rendered polygon potentially must be written into the frame buffer. First generation chips processed 32 pixels in a group in a single clock cycle. Second generation chips process a 32 x16 pixel matrix (or tile) in a single clock cycle.

Table 3 uses the same parameters and conditions defined earlier to calculate bandwidth requirements for each of the three main operations in PowerVR.

z-Buffer Bandwidth (Bytes/Second)	Texture Bandwidth (Bytes/Second)	Frame Buffer Bandwidth (Bytes/Second)	Total Bandwidth Required
0	3xFRxRES (note 3)		6xFRxRES+104 MB
0	6xFRxRES (note 4)	3xFRxRES	9xFRxRES+104 MB
0	12xFRxRES (note 5)	+	15xFRxRES+104 MB
0	24xFRxRES (note 6)	104 MB/sec	27xFRxRES+104 MB
0	48xFRxRES (note 7)		51xFRxRES+104 MB

Table 3. Bandwidth Requirement (Sustained) For PowerVR

Notes:
(1) RES = screen resolution (800x600)
(2) FR = rendering frames rate (60)
(3) Point sampling
(4) Linear MIP mapping
(5) Bilinear interpolation
(6) Trilinear texturing
(7) Anisotropic texturing

Table 4 shows the results obtained by applying these equations to 500k, 1M and 2M polygons for a 60fps rendering frame rate.

Case	500k Polygons	1M Polygons	2M Polygons
Point sampling	276.8 MB/sec	276.8 MB/sec	276.8 MB/sec
MIPmapping	363.2 MB/sec	363.2 MB/sec	363.2 MB/sec
Bilinear interpolation	536 MB/sec	536 MB/sec	536 MB/sec
Trilinear texturing	881.6 MB/sec	881.6 MB/sec	881.6 MB/sec
Anisotropic filtering	1572.8 MB/sec	1572.8 MB/sec	1572.8 MB/sec

Table 4. Example of Bandwidth Requirements (Average Polygon Size of 100 Pixels)

The comparison between the results of Table 2 and Table 4 clearly show two points:

- PowerVR's memory bandwidth requirements are between three to ten times lower than traditional three-dimensional systems
- PowerVR's memory bandwidth requirement for any given texturing approach is independent of polygon-per-second performance figure.

Note that second generation PowerVR implementations can improve the bandwidth further by using compressed textures, a 16-bit frame buffer with the texturing and shading operations all calculated in 24-bit, and then dithering the output. Also the Z compare is 32-bit which is more accurate than the example conditions without any increase in bandwidth requirement.

Scalable Architecture

Another key feature of the PowerVR technology is its true scalability. PowerVR's display list architecture is inherently scalable. The screen can be segmented and the work shared between PowerVR chips, providing linear performance increases, allowing system designers to match three-dimensional performance to user requirements.

Hidden surface removal is also key to PowerVR scalability. Hidden surface removal is performed by multiple on-chip processing elements (PEs which operate concurrently and process a tile of pixels. By adding more PEs (in an integrated device or by using multiple devices) it is possible to process more pixels simultaneously.

Summary

In summary, traditional 3D rendering approaches rely on large memory systems and performance is limited by memory access speed (bandwidth). This means conventional 3D systems cannot fundamentally deliver both low cost and high performance. PowerVR's reduced memory approach eliminates expensive memory and associated performance limiting bandwidth. This means the performance of PowerVR chips is determined by the number of processing elements simultaneously active in a chip (or group of chips) and the clock speed of the chip (or group of chips). Therefore PowerVR provides high performance with low cost and ensures scalability as performance is not limited by memory bandwidth and continues to increase as silicon technology improves.