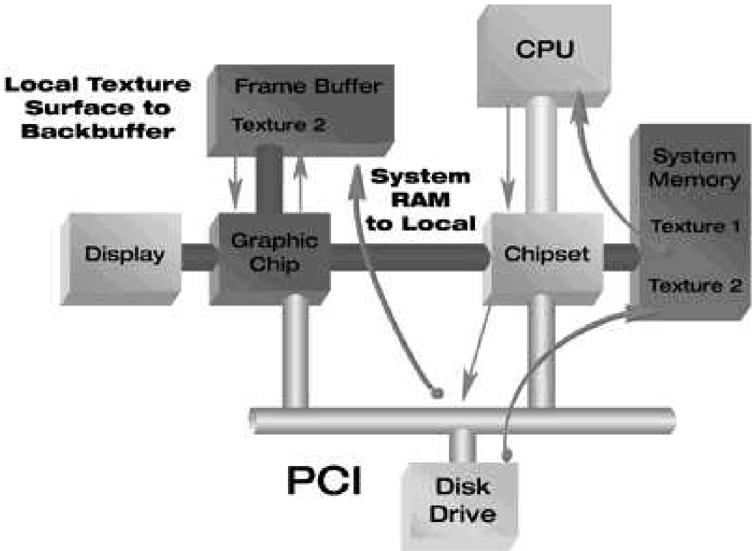
Accelerated Graphics Port tutorial

Chapter 1: 3D Graphics on Current Generation PCs

AGP is a new interface on the PC platform that dramatically improves the processing of 3D graphics and full-motion video. In order to fully understa the impact of AGP technology, its necessary to first review how 3D graphi are currently supported on the PC platform without AGP.

Lifelike, animated 3D graphics requires the performance of a continuous se of processor-intensive geometry calculations which define the position of objects in 3D space. Typically, geometry calculations are performed by the PC's processor because it is well-suited to handling the floating point operations required. At the same time, the graphics controller must process texture data in order to create lifelike surfaces and shadows within the 3D image. The most critical aspect of 3D graphics is the processing of texture maps, the bitmaps which describe in detail the surfaces of three-dimension objects. Texture map processing consists of fetching one, two, four, or eigh texels (texture elements) from a bitmap, averaging them together based on some mathematical approximation of the location in the bitmap (or multipl bitmaps) needed on the final image, and then writing the resulting pixel to t frame buffer. The texel coordinates are non-trivial functions of the 3D viewpoint and the geometry of the object onto which the bitmap is being projected.

Figure 1 shows how the processing of texture maps is currently supported o the PC. As shown, there are five basic steps involved in processing textures



1. Prior to their usage, texture maps are read from the hard drive and loade into system memory. The data travels via the IDE bus and chipset before b loaded into memory.

2. When a texture map must be used for a scene, it is read from system mem into the processor. The processor performs point-of-view transformations u the texture map then caches the results.

3. Lighting and viewpoint transforms are then applied to the cached data. T results of this operation are subsequently written back to system memory.

4. The graphics controller then reads the transformed textures from system memory and writes them in its local video memory (also called graphics controller memory, the frame buffer, or off-screen RAM). In present-day systems, this data must travel to the graphics controller over the PCI bus.

5. The graphics controller next reads the textures plus 2D color informatior from its frame buffer. This data is used to render a frame which can be displayed on the 2-D monitor screen. The result is written back into the fra buffer. The system's digital-to-analog convertor will read the frame and co it to an analog signal that drives the display.

The reader may notice a number of problems with the way texture maps ar currently handled. First, textures must be stored in both system memory an frame buffer; redundant copies are an inefficient use of memory resources. Second, storing the textures in the frame buffer, even temporarily, places a ceiling on the size of the textures. There is a demand for textures with grea and greater detail, pressuring hardware manufacturers to put more frame bu in their systems. However, this type of memory is quite expensive, thus thi not an optimal solution. Finally, the 132Mbyte/s bandwidth of the PCI bus limits the rate at which texture maps can be transferred to the graphics subsystem. Furthermore, in typical systems several I/O devices on the PCI must share the available bandwidth. The introduction of other high-speed devices, such as Ultra DMA disk drives and 100 MByte/s LAN cards make congestion even worse. It is easy to see how congestion on the PCI bus can limit 3D graphics performance on a PC.

Currently, applications employ several strategies to compensate for the limitations inherent in present-day PCs. Applications use a caching or "swapping" algorithm to decide which textures should be stored in local fra buffer memory versus system memory. Typically, applications dedicate a portion of off-screen local memory as frame-to-frame texture swapping spa while the remaining off-screen memory contains commonly used textures (fixed texture memory), for example, clouds and sea in a flight simulator.

If the hardware can only texture from local video memory, the algorithm usually attempts to pre-fetch the needed textures for each frame or scene in local video memory. Without pre-fetching, users will see a noticeable paus the scene as the software stops drawing while the needed texture is swappe into local video memory, or even worse, from disk to system memory to lo video memory. Often even more delay in initial texture loading occurs due necessary reformatting of textures into a hardware-specific compressed for

Applications may reserve part of the local memory for swapping, and leave of it permanently loaded with "fixed" commonly used textures. Depending the number of textures per frame, the algorithm may vary the proportion of memory allocated for texture swapping and fixed texture memory. Scenes which contain a large number of textures tend to have less texture reuse; th benefit from larger texture swapping space.

Chapter 2

Table of Contents