



Intel740 Graphics Accelerator

Software Developer's Manual

February 1998

Order Number: 290617-001





Information in this document is provided in connection with Intel products. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. Except as provided in Intel's Terms and Conditions of Sale for such products, Intel assumes no liability whatsoever, and Intel disclaims any express or implied warranty, relating to sale and/or use of Intel products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright or other intellectual property right. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The Intel740 may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

MPEG is an international standard for video compression/decompression promoted by ISO. Implementations of MPEG CODECs, or MPEG enabled platforms may require licenses from various entities, including Intel Corporation.

I²C is a two-wire communications bus/protocol developed by Philips. SMBus is a subset of the I²C bus/protocol and was developed by Intel. Implementations of the I²C bus/protocol or the SMBus bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature, may be obtained from:

<http://www.intel.com>
or call 1-800-548-4725

Copyright © Intel Corporation, 1997-1998

*Third-party brands and names are the property of their respective owners.



Contents

1	Introduction	1-1
1.1	About This Manual	1-1
1.2	Intel740 Features	1-2
1.3	Related Documents	1-3
2	Hardware Capabilities	2-1
2.1	Architectural Overview	2-2
2.2	3D Capabilities.....	2-7
2.2.1	3D Pipeline.....	2-7
2.2.2	3D Primitives	2-10
2.2.3	Data Formats	2-16
2.2.4	Surface Color Attributes	2-16
2.2.4.1	Fogging	2-16
2.2.4.2	Specular Highlighting	2-18
2.2.4.3	Alpha Blending.....	2-19
2.2.4.4	Alpha Testing	2-22
2.2.4.5	Color Dithering	2-22
2.2.4.6	Shading.....	2-23
2.2.4.7	Stippled Pattern	2-24
2.2.5	Texture Map Attributes.....	2-24
2.2.5.1	Texture Map Formats.....	2-26
2.2.5.2	Texture Map Blending	2-28
2.2.5.3	Texture Map Chroma Keying and Color Keying	2-30
2.2.5.4	Texture Wrapping Formats	2-32
2.2.5.5	Texture Map Filtering	2-33
2.2.5.6	Texture Mipmapping	2-34
2.2.6	Drawing Formats.....	2-37
2.2.7	Buffers.....	2-37
2.2.7.1	Double and Triple Buffering	2-38
2.2.7.2	Z-Buffering	2-38
2.2.8	Antialiasing.....	2-39
2.2.9	Back Face Culling	2-40
2.3	2D Capabilities.....	2-41
2.3.1	BitBLT Engine	2-41
2.3.2	Stretch BLT Engine	2-41
2.3.3	Color Expansion.....	2-41
2.3.4	Hardware Cursor.....	2-42
2.3.5	Video Display Resolutions	2-42
2.4	Video, VBI, and Intercast Capabilities.....	2-43
2.4.1	Video Capture Port	2-43
2.4.2	Video Overlay	2-44
2.4.3	VBI and Intercast	2-44
2.4.3.1	VBI Data Format	2-45
2.4.3.2	VBI Data Flow	2-47
2.4.3.3	CC and EDS	2-47
2.4.3.4	Direct CC and EDS Capture	2-47
2.5	DVD Capabilities.....	2-47
2.5.1	Hardware DVD/MPEG-2 Movie Playback.....	2-47
2.6	TV Out Interface.....	2-48

2.7	AGP Interface	2-48
2.7.1	AGP Primer	2-48
2.7.2	AGP Software Architecture	2-49
2.8	BIOS Interface	2-50
2.9	Local Memory	2-50
3	Programming Environment	3-1
3.1	OpenGL Programming Environment	3-1
3.1.1	Model	3-1
3.1.2	Supported OpenGL MCD States	3-2
3.1.3	Supported OpenGL MCD Primitives	3-5
3.1.4	Supported OpenGL MCD Texture States	3-6
3.1.5	Supported OpenGL MCD Texture Environment States	3-6
3.2	DirectX Programming Environment	3-7
3.3	Mini Interface Drivers	3-8
3.3.1	Mini Display Driver	3-8
3.3.1.1	Structures Exported to GDI	3-8
3.4	DirectDraw Display Driver Interface	3-10
3.4.1	Directdraw Hal Capabilities	3-10
3.5	Direct3D Interface	3-14
3.5.1	Supported Direct3D Capabilities	3-14
3.5.2	Supported RenderState	3-18
3.5.3	Supported RenderPrimitives	3-20
3.6	Video Interface	3-21
3.7	GDI Escape Interface	3-22
4	Performance Considerations	4-1
4.1	Performance Strategies And Measurements	4-1
4.1.1	Intel740 Performance Capabilities	4-1
4.1.2	Using CPU/Intel740 Concurrency	4-2
4.1.3	Performance Test Results	4-3
4.1.3.1	Raster Speed Test Method	4-3
4.1.3.2	Implications and Analysis	4-9
4.1.4	Special Performance Considerations	4-11
4.1.4.1	Direct3D DrawPrimitive vs. Execute Buffers	4-11
4.1.4.2	Triangle Packet Size	4-12
4.1.4.3	Texture Sizes	4-14
4.1.4.4	Palette Changes	4-15
4.1.4.5	Untiled Textures for Procedural Texture Animation	4-15
4.1.4.6	High Performance Transparency	4-16
4.1.4.7	Screen Resolutions	4-17
4.1.5	Budgeting CPU Clock Cycles	4-18
4.1.6	Video Performance	4-18
4.2	Other Programming Tips	4-20
4.2.1	Texture and Surface Effects	4-20
4.2.1.1	Texture Formats	4-20
4.2.1.2	Texture Sizes	4-21
4.2.1.3	Texture Storage	4-21
4.2.1.4	Animated Texture Effects	4-21
4.2.1.5	Multi-pass Texture Effects	4-22
4.2.2	Software Strategies	4-22
4.2.2.1	Z-Buffering	4-23
4.2.2.2	Antialiasing	4-23



	4.2.2.3	Minimizing State Transitions	4-23
	4.2.2.4	Dynamic AGP Buffer Placement	4-23
	4.2.2.5	Texture Palettes	4-24
	4.2.2.6	Using Mipmapping	4-24
	4.2.2.7	Designing For Sub Pixel Accuracy.....	4-25
	4.2.2.8	Using Color/chroma Keying Over Alpha Blended Textures.....	4-25
	4.2.2.9	Avoiding Stippling Errors.....	4-25
	4.2.2.10	Avoiding Flipping Errors.....	4-25
	4.2.2.11	Texture Sorting Is Not Required	4-25
4.3		OpenGL Performance Guidelines.....	4-26
	4.3.1	OpenGL Feature Classification	4-26
	4.3.2	Feature Overview.....	4-27
	4.3.3	OpenGL Operation.....	4-29
	4.3.3.1	Begin/End Paradigm	4-29
	4.3.3.2	Vertex Specification	4-29
	4.3.3.3	Vertex Arrays	4-29
	4.3.3.4	Rectangles	4-29
	4.3.3.5	Coordinate Transformation	4-29
	4.3.3.6	Clipping	4-29
	4.3.3.7	Current Raster Position.....	4-29
	4.3.3.8	Colors and Coloring	4-30
	4.3.4	Rasterization	4-30
	4.3.4.1	Antialiasing.....	4-30
	4.3.4.2	Points	4-30
	4.3.4.3	Line Segments	4-30
	4.3.4.4	Polygons	4-30
	4.3.4.5	Pixel Rectangles	4-30
	4.3.4.6	Bitmaps	4-30
	4.3.4.7	Texturing	4-30
	4.3.4.8	Fog.....	4-31
	4.3.4.9	Antialiasing Application	4-31
	4.3.5	Fragments And The Frame Buffer	4-31
	4.3.5.1	Per-Fragment Operations	4-31
	4.3.5.2	Whole Framebuffer Operations.....	4-31
	4.3.5.3	Drawing, Reading, and Copying Pixels.....	4-31
	4.3.6	Special Functions.....	4-32
	4.3.7	State And State Requests.....	4-32
	4.3.8	GL Command Summary	4-32
		Glossary	Glossary-1

Figures

2-1	System Block Diagram with Intel740	2-2
2-2	Intel740 Architectural Interfaces	2-3
2-3	The Intel740 Implementation of Sideband Addressing	2-4
2-4	Batch Processing on the Intel740 - A Conceptual View	2-5
2-5	The Intel740's Ability to Execute Textures Directly From AGP Memory	2-6
2-6	The Intel740 Functioning as Two Memory Controllers	2-6
2-7	3D Pipeline for DirectX	2-8
2-8	3D Pipeline for OpenGL.....	2-9
2-9	Triangle as the Intel740 Driver Sees It	2-11
2-10	Effects of Fogging Off vs Fogging On.....	2-17
2-11	Effects of Using Specular Highlighting.....	2-18
2-12	Effects of Using Alpha Blending	2-20
2-13	Effects of Flat Shading vs. Smooth Shading	2-23
2-14	Getting 1.3 Gbytes of Concurrent Throughput with the Intel740	2-25
2-15	A Color Keyed Splash.....	2-31
2-16	Point Filtering VS. Bilinear Filtering	2-34
2-17	An Example of Five Levels of Mipmapped Texture	2-35
2-18	Z-Buffering Off vs. Z-Buffering On	2-38
2-19	Effects of Antialiasing	2-40
2-20	Content of an NTSC Video Frame	2-45
2-21	Configuration of Video Capture Memory with VBI	2-46
2-22	VBI Buffering Scheme	2-46
2-23	Intel740 Connects to System Memory Over AGP	2-48
2-24	New Services in Windows Work with DirectDraw to Support AGP Applications	2-49
3-1	MCD Architecture	3-1
3-2	Intel740 Software Architecture.....	3-7
4-1	Intel740/CPU Usage Model	4-2
4-2	Improper Usage Model	4-2
4-3	RasM Intel740/CPU Usage Model.....	4-3
4-4	RasM Pseudo-Code	4-4
4-5	Basic Feature Sweeps.....	4-6
4-6	Advanced Feature Sweeps.....	4-7
4-7	Full Feature Sweeps.....	4-8
4-8	Performance vs. Percent Z Occlusion	4-10
4-9	Performance of DrawPrimitive vs. Execute Buffer	4-11
4-10	Performance vs. Buffer Size (Duty Cycle)	4-13
4-11	Performance vs. Total Packet Size.....	4-14
4-12	Performance vs. Texture Size	4-14
4-13	Performance vs. Palette Changes	4-15
4-14	Performance with Untiled Textures.....	4-16
4-15	Performance vs. Transparency.....	4-17
4-16	Performance vs. Screen Resolution	4-17
4-17	Available Memory Bandwidth on a Pentium® II Processor System	4-18
4-18	Dynamic AGP Buffer Placement.....	4-24



Tables

1-1	Intel740 Feature Summary	1-2
2-1	Data Formats	2-16
2-2	Alpha Blend Functions for OpenGL & DirectX	2-21
2-3	DirectX Texture Map Blending Functions	2-29
2-4	OpenGL Texture Blend Modes and Equations	2-30
2-5	Supported DirectX Texture Wrap Formats.....	2-32
2-6	Supported OpenGL Texture Wrap Formats.....	2-32
2-7	Pixel Formats and Buffers.....	2-37
2-8	Display Resolutions	2-42
3-1	Supported OpenGL MCD Enables.....	3-2
3-2	Supported OpenGL MCD States	3-3
3-3	Supported OpenGL MCD Primitives	3-5
3-4	Supported OpenGL MCD Texture States	3-6
3-5	Supported OpenGL MCD Texture Environment States	3-6
3-6	Device Technology—dpTechnology	3-8
3-7	dwCaps—Specifies Driver-Specific Capabilities	3-10
3-8	dwCaps2—Specifies More Driver-Specific Capabilities.....	3-11
3-9	dwCKeyCaps—Color Key Capabilities	3-11
3-10	dwFXCaps—Specifies Driver-Specific Stretching and Effects Capabilities	3-12
3-11	dwPalCaps—Specifies Palette Capabilities.....	3-12
3-12	ddsCaps.dwCaps—Specifies The Capabilities Of The Surface	3-13
3-13	General Device Capabilities.....	3-14
3-14	Texture Capabilities	3-15
3-15	Primitive Capabilities Supported	3-15
3-16	DIRECT3D RenderState Hardware / Software Support	3-18
3-17	DIRECT3D RenderPrimitive Hardware / Software Support	3-20
3-18	VfW Capture Driver Capability	3-21
3-19	Functionality Control	3-22
3-20	Device Driver Debugging Control	3-22
4-1	Result Summary	4-4
4-2	Symbol Key.....	4-5
4-3	CPU Cycle Targets	4-18
4-4	Typical Video/Data Capture Applications.....	4-19
4-5	CPU Usage for Some Typical Applications.....	4-19
4-6	Rating OpenGL Features.....	4-27
4-7	Included and Excluded Pre-Fragment Operations.....	4-31
4-8	Command Performance Ratings.....	4-32

The Intel740 is a graphics hardware accelerator providing a variety of features which can enhance the speed and visual quality of 2D and 3D applications. The Intel740 feature set also includes DVD, video capture, VBI and intercast programming capabilities. The Intel740 works with the OpenGL*, Microsoft DirectX*, and Win32* programming interfaces. Both the OpenGL and the DirectX APIs give graphics applications a standard way to invoke 2D, 3D and video graphics rendering functions and allow a software application to be hardware independent.

The Intel740 OpenGL driver set runs on personal computers that are based on the Intel Architecture with Accelerated Graphics Port (AGP) support and have Microsoft WindowsNT* 4.0 or newer operating system with the OpenGL 1.1 application programming interface (API). The Intel740 DirectX driver set runs on personal computers that are based on the Intel Architecture with AGP support and have the Microsoft Windows98*, Windows95* with USB support, or WindowsNT 5.0 operating system with DirectX 5.0 (or newer) and Win32 programming interfaces. This manual presents the Intel740's accelerated functions that are callable from OpenGL, DirectX and Win32 application programs.

1.1 About This Manual

This manual is intended for graphics tool or application programmers who are experienced with writing 2D, 3D, or video graphics applications. The manual assumes that the programmer has a working knowledge of the vocabulary and principles of graphics applications. It is intended for programmers who plan to use the DirectX, OpenGL and Win32 software API interfaces.

This chapter introduces the Intel740 features and API support.

Chapter 2, "Hardware Capabilities" — provides a hardware system overview and reviews the hardware functionality of the Intel740. This chapter describes in detail the 3D rendering, 2D display and video capabilities.

Chapter 3, "Programming Environment" — describes the OpenGL and DirectX APIs for the Windows95, Windows98, and WindowsNT operating environments.

Chapter 4, "Performance Considerations" — discusses programming approaches to maximize performance. Throughput, duty cycle, and memory bandwidth sensitivities on performance are addressed. Programming tips and strategies for using the Intel740 are provided. OpenGL performance guidelines are also discussed.

1.2 Intel740 Features

This section offers a brief overview of the most prominent Intel740 features. The Intel740 graphics controller may contain design defects or errors known as errata. Current characterized errata are available on request.

Table 1-1. Intel740 Feature Summary

HYPER PIPELINED ARCHITECTURE	2D & DISPLAY FEATURES
<ul style="list-style-type: none"> • Direct Memory Execution (DME) 	<ul style="list-style-type: none"> • Display Resolution: 640x480x8 up to 1280x1024x16 @ 48 Hz - 85 Hz Refresh Rate
<ul style="list-style-type: none"> • 0.85 Mega-Triangles/Second Peak[†] 	<ul style="list-style-type: none"> • Hardware Cursor
<ul style="list-style-type: none"> • 425-500K Triangles/Second Full Featured Sustained 3D Performance[†] 	<ul style="list-style-type: none"> • Hardware Overlay
<ul style="list-style-type: none"> • 45-55 Mega-Pixels/Second Full Features (>140 Pixel Triangles) Sustained 3D Performance[†] 	<ul style="list-style-type: none"> • Blitter Engine
<ul style="list-style-type: none"> • Full Sideband Accelerated Graphics Port 	<ul style="list-style-type: none"> • Stretch Blitter Engine
<ul style="list-style-type: none"> • Parallel Execution 	<ul style="list-style-type: none"> • Color Expansion
<ul style="list-style-type: none"> • Optimized for 440LX Intel AGP Sets 	
3D FEATURES	VIDEO IN/OUT FEATURES
<ul style="list-style-type: none"> • Z-Buffering 	<ul style="list-style-type: none"> • Programmable Video Output Characteristics (VGA, SVGA, NTSC, PAL)
<ul style="list-style-type: none"> • Back Face Culling 	<ul style="list-style-type: none"> • Video Capture Support (16- or 8- bit Uni-Directional Capture Port)
<ul style="list-style-type: none"> • Antialiasing 	<ul style="list-style-type: none"> • Scaling of the Full Motion Video Data
<ul style="list-style-type: none"> • Flat and Gouraud Shading 	<ul style="list-style-type: none"> • Full Motion Video Overlaid with Frame Buffer
<ul style="list-style-type: none"> • Specular Highlighting 	<ul style="list-style-type: none"> • Intercast & VBI Support
<ul style="list-style-type: none"> • Fog with RGB Components 	<ul style="list-style-type: none"> • MPEGII DVD Capability
<ul style="list-style-type: none"> • Color Alpha Blending 	
<ul style="list-style-type: none"> • Color Dithering 	
<ul style="list-style-type: none"> • Stippling or "Screen Door" transparency 	
<ul style="list-style-type: none"> • Texture Color Keying and Chroma Keying 	
<ul style="list-style-type: none"> • Per Pixel Perspective Correct Texture Mapping 	
<ul style="list-style-type: none"> • Mipmapping with Trilinear Filtering 1024x1024 to 1x1 	
<ul style="list-style-type: none"> • Texture Formats: 1, 2, 4 or 8-bit palettized; ARGB 1555 0565 4444; Compressed AYUV 0422 0555 1544. 	
<ul style="list-style-type: none"> • Texture Memory Limited Only by System RAM 	
<ul style="list-style-type: none"> • Optimized for 800x600x16 and 640x480x16 Display Resolution 	

[†] See "Performance Strategies And Measurements" on page 4-1 for the system configuration used to generate these performance statistics.

1.3 Related Documents

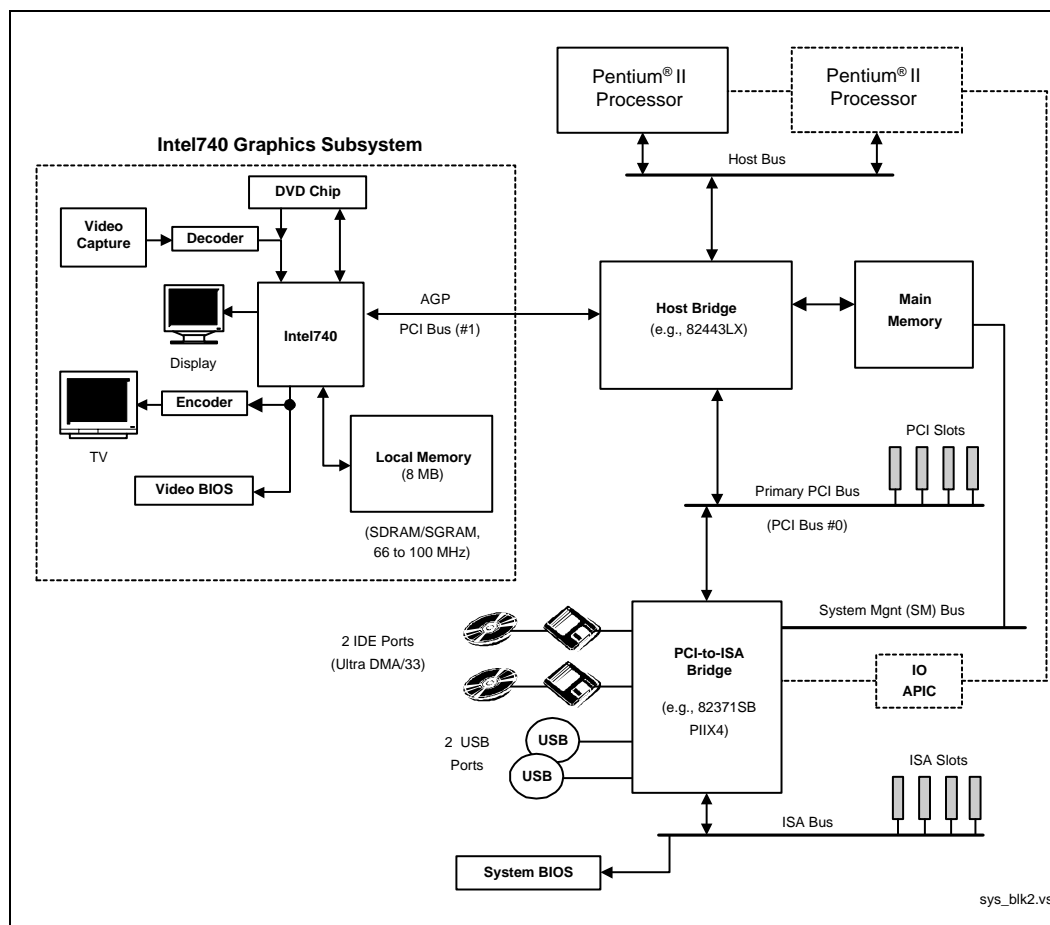
Refer to the following materials for information outside the scope of this document.

- Intel740 Specification Update
- Silicon Graphics OpenGL* SDK
- *OpenGL Programming Guidelines*, Second Edition; Woo, Mason; Neider, Jackie; Davis, Tom; Addison-Wesley Developer Press; 1997.
- Microsoft DirectX* Media 5.0 SDK
- Win32 SDK

Optimized for the new Accelerated Graphics Port (AGP), Intel740 delivers high performance in 2D and 3D graphics rasterization. In addition, Intel740 has a video capture port that allows easy hookup to video conferencing systems such as POTS (Plain Old Telephone Set) video conferencing applications and Intericast technology. Each hardware feature is discussed in the following sections:

- “Architectural Overview” on page 2-2
- “3D Capabilities” on page 2-7
- “2D Capabilities” on page 2-41
- “Video, VBI, and Intericast Capabilities” on page 2-43
- “DVD Capabilities” on page 2-47
- “TV Out Interface” on page 2-48
- “AGP Interface” on page 2-48
- “BIOS Interface” on page 2-50
- “Local Memory” on page 2-50

Figure 2-1. System Block Diagram with Intel740

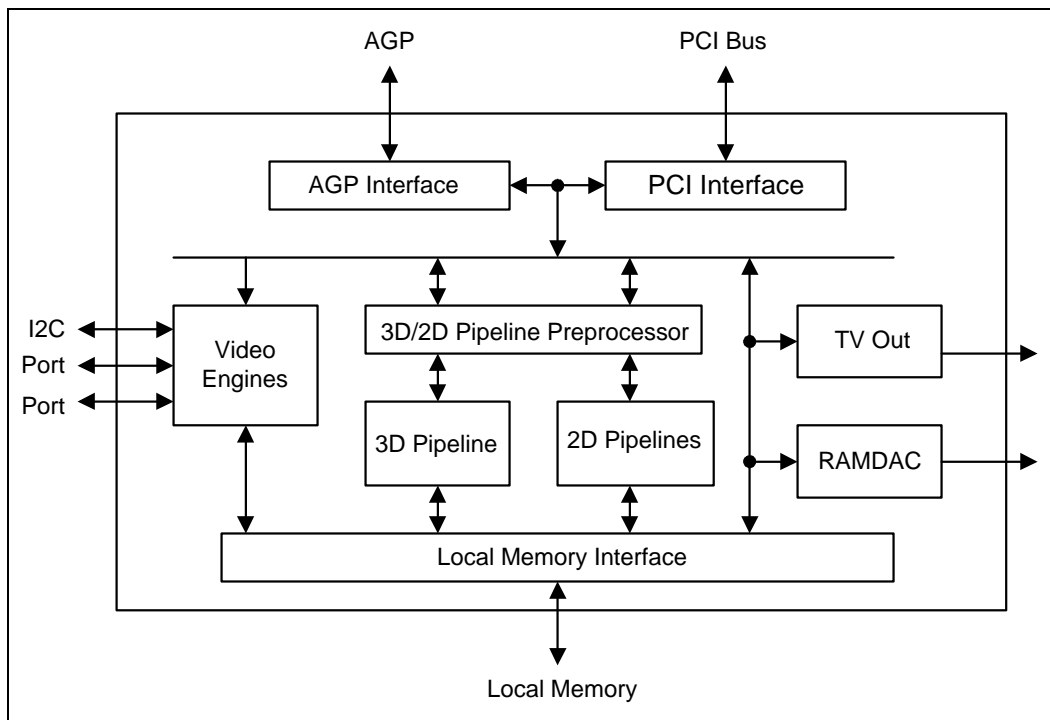


2.1 Architectural Overview

The Intel740 is a highly integrated graphics accelerator designed for the Accelerated Graphics Port (AGP). Its architecture consists of dedicated multimedia engines executing in parallel to deliver high performance 3D, 2D and video capabilities. The 3D and 2D engines are managed by the 3D/2D pipeline preprocessor ensuring them a sustained flow of graphics data. The Intel740 also includes dedicated video engines for support of video conferencing and other video applications.

The Intel740 is capable of delivering a high rate of sustained 3D graphics performance with full 3D feature set functionality. This constant high level of performance is delivered through the Intel740's Direct Memory Execution (DME) architecture and the incorporation of specific graphics architectural enhancements. Through the use of DME, the Intel740 fully utilizes the entire bandwidth of the AGP and memory, which improves the performance when processing the heavy data demands of 3D. Architectural enhancements within the 3D pipeline ensure that the Intel740 uses this data in the most efficient way possible. Figure 2-1 shows a block diagram of the Intel740 architecture.

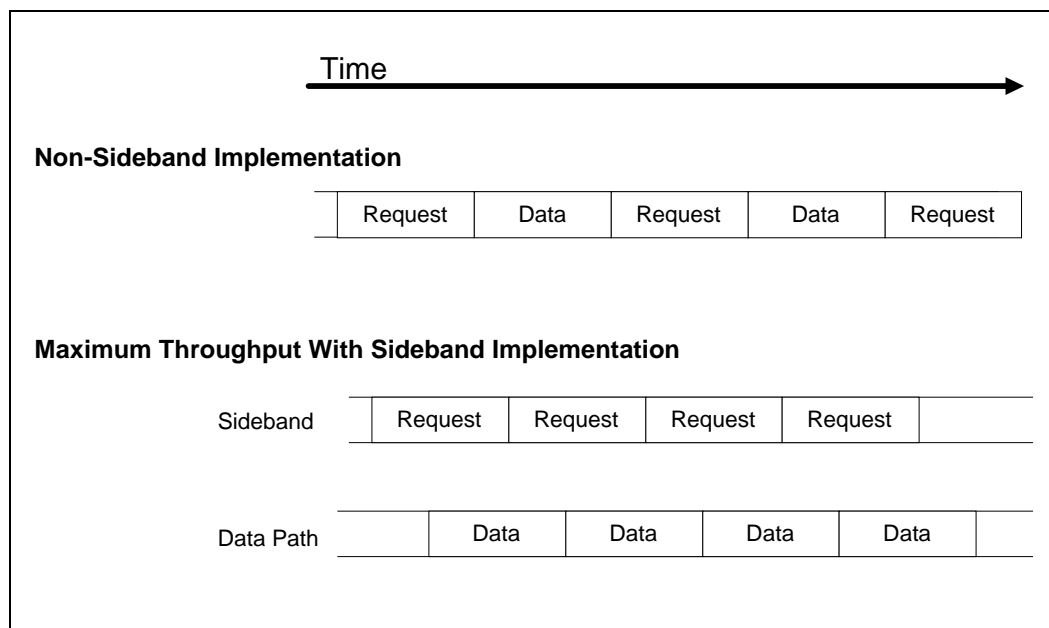
Figure 2-2. Intel740 Architectural Interfaces



The DME architecture means that full AGP implementation is integrated into the Intel740 with 2X sideband support, allowing up to 533 Mbyte/s data transfers. Deep buffering allows the Intel740 to receive data at this high rate and handle any latencies associated with AGP transactions. Figure 2-3 illustrates the Intel740 implementation of sideband addressing.

Further utilization of AGP is achieved using 2X sideband signaling. Sideband addressing gives the Intel740 the ability to issue multiple requests without having to wait for data to be returned. This allows the Intel740 to achieve the highest possible sustained data transfer rates across the AGP, and makes DME possible.

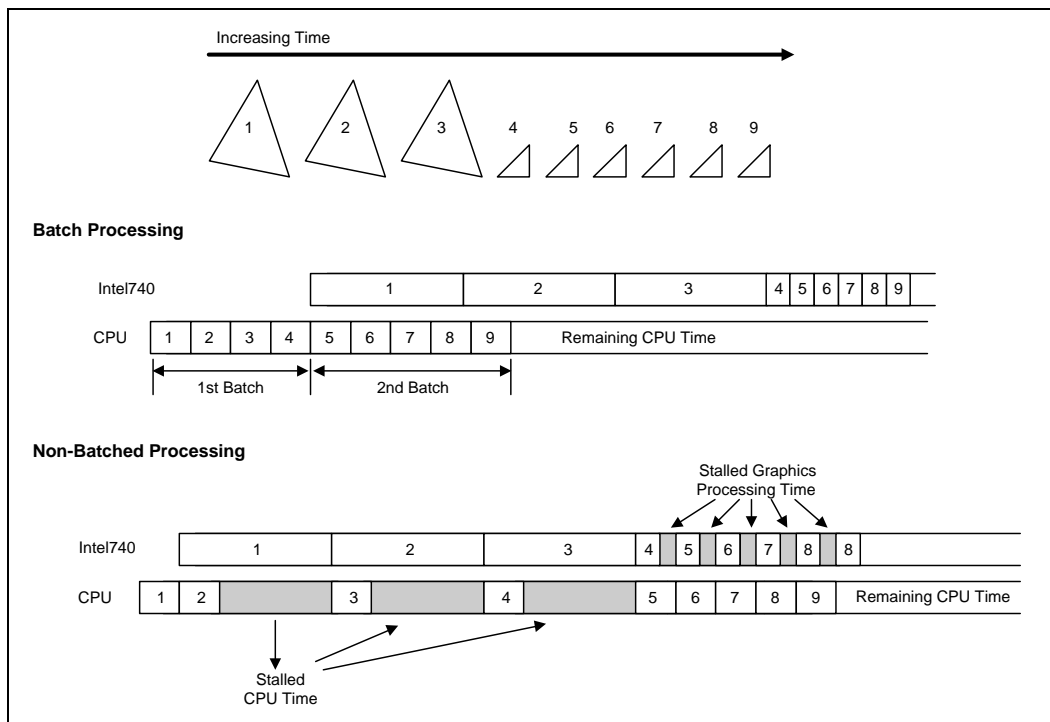
Figure 2-3. The Intel740 Implementation of Sideband Addressing



Deep buffering, 2X mode, and sideband signaling provide sustained 3D performance by delivering data at a constant rate regardless of other system activities.

To provide the highest level of system concurrency and performance the Intel740 is optimized for a batch processing mode of triangle delivery. Batch processing frees up the CPU for intelligent 3D gaming and more complex geometry processing. This batch processing allows the CPU to place a “batch” of triangles in memory and begin on another batch of triangles without needing to perform handshaking with the Intel740. Figure 2-4 illustrates a conceptual view of the Intel740’s batch triangle processing.

Figure 2-4. Batch Processing on the Intel740 - A Conceptual View



The DME capabilities of the Intel740 maximize the amount of memory available for rendering. The Intel740 is capable of executing directly from AGP memory. This “direct execution” avoids the “thrashing” of local memory associated with an architecture that must load local memory from AGP or system memory. Textures can be executed directly from AGP memory, allowing performance to be sustained even when texture complexity increases. Figure 2-5 shows conceptually how the Intel740 executes textures directly from AGP memory.

The Intel740 is capable of rendering from local memory while textures are being executed from AGP memory through parallel arbitration. This arbitration allows a combined memory peak bandwidth of 1.3 Gbyte/s. The capability to support two open pages in local memory coupled with an open page in AGP memory supports the 3D rendering model of front and back buffers, Z buffer, and textures. 2D rendering through the use of three raster operands (pattern, source and destination) is supported. By maintaining virtually three open pages, the Intel740 sustains graphics performance with fully textured and Depth Buffered scenes. Figure 2-6 illustrates the dual memory utilization of the Intel740.

Figure 2-5. The Intel740's Ability to Execute Textures Directly From AGP Memory

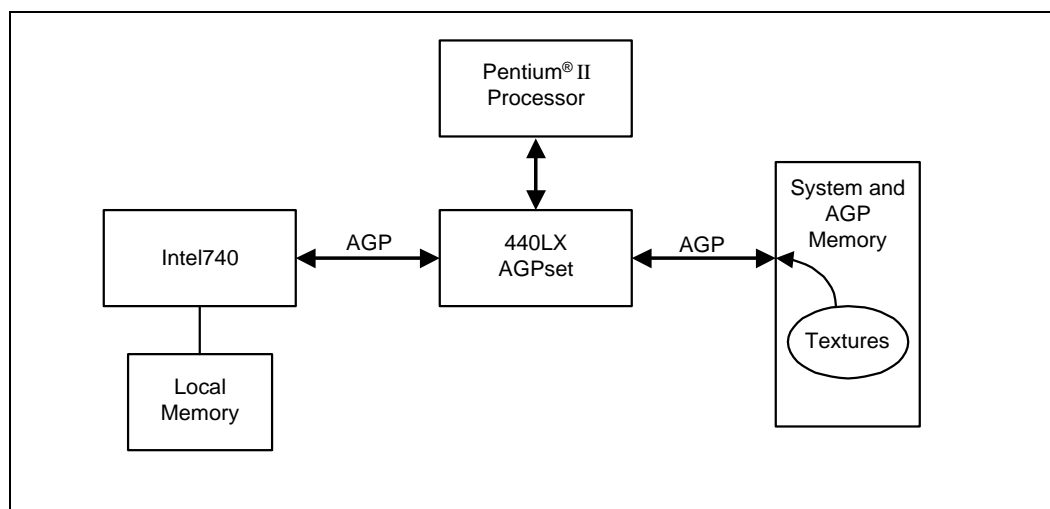
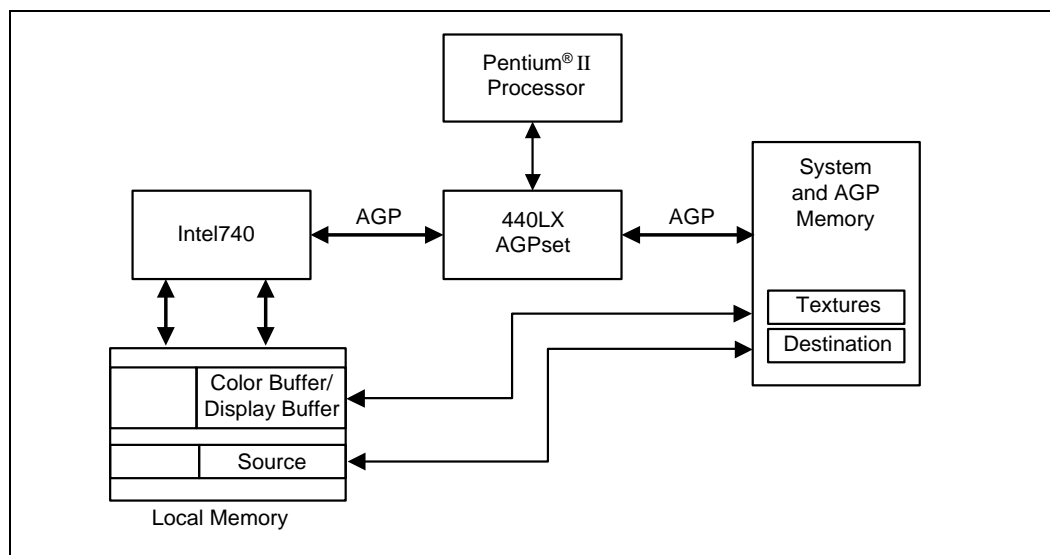


Figure 2-6. The Intel740 Functioning as Two Memory Controllers



Dedicated 3D pipeline enhancements are included in the Intel740's architecture. These enhancements are designed to manage the way in which 3D data is requested from memory and then used within the compute engine. While parallelism is employed among each of the Intel740's engines, the 3D pipeline calculates 3D data in a highly parallel fashion. With this architecture, the 3D rasterizer is able to compute four fully textured, shaded, fogged and Z Buffered pixels per clock. In addition to using data efficiently, the 3D pipeline requests data from memory so that memory locality is maximized regardless of triangle size or orientation. This results in fewer page misses and a highly sustained 3D graphics output independent of the complexity of the 3D scene being rendered. To further improve memory accesses, an efficient texture cache is implemented, avoiding the need for redundant fetches when texturing triangles. By combining memory efficiencies and processing data efficiencies, the Intel740 is capable of a high rate of sustained 3D performance.

2.2 3D Capabilities

While the API or software application takes care of the geometry and lighting stages of the 3D pipeline, the Intel740 enables hardware acceleration of the rendering stages. In the DirectX and OpenGL 3D Pipeline diagrams (Figure 2-7 and Figure 2-8), the rasterization stage of the 3D pipeline consists of the Setup Engine, Scan Converter, Texture Pipeline, and Color Calculator/Depth Buffer Test. These four modules comprise the rendering engine and this section discusses all of the rendering features associated with the 3D hardware including the following subsections for both OpenGL and DirectX:

- “3D Pipeline” (below)
- “3D Primitives” on page 2-10
- “Data Formats” on page 2-16
- “Surface Color Attributes” on page 2-16
- “Texture Map Attributes” on page 2-24
- “Drawing Formats” on page 2-37
- “Buffers” on page 2-37
- “Antialiasing” on page 2-39
- “Back Face Culling” on page 2-40

2.2.1 3D Pipeline

The 3D pipeline unit in the Intel740 offers advantages over the traditional graphics accelerators by performing 3D setup locally rather than within the CPU. This difference allows the processor to perform more geometry calculations while the Intel740 performs set-up and rendering. 3D features supported include perspective correct texture mapping, trilinear mipmapping, Gouraud shading, alpha-blending, stippling, and Z-buffering. Depending on the application, each feature can be independently enabled or disabled for various levels of performance. The Intel740 allows for high performance when all 3D features are enabled for the entire run of the application with the only exception being antialiasing. The Intel740 is optimized for high throughput when textures are stored in AGP memory, otherwise known as non-local video memory. Relocating textures in main memory is also supported. Locating texture information in the AGP non-local video memory frees up the Intel740 local frame buffer memory for graphics execution. Textures cannot be put in local video memory. Polygon antialiasing is hardware assisted by Intel740.

Figure 2-7 and Figure 2-8 illustrates the DirectX and OpenGL API function calls, respectively, as they are used in the 3D rasterization pipeline of the Intel740 architecture.

Figure 2-7. 3D Pipeline for DirectX

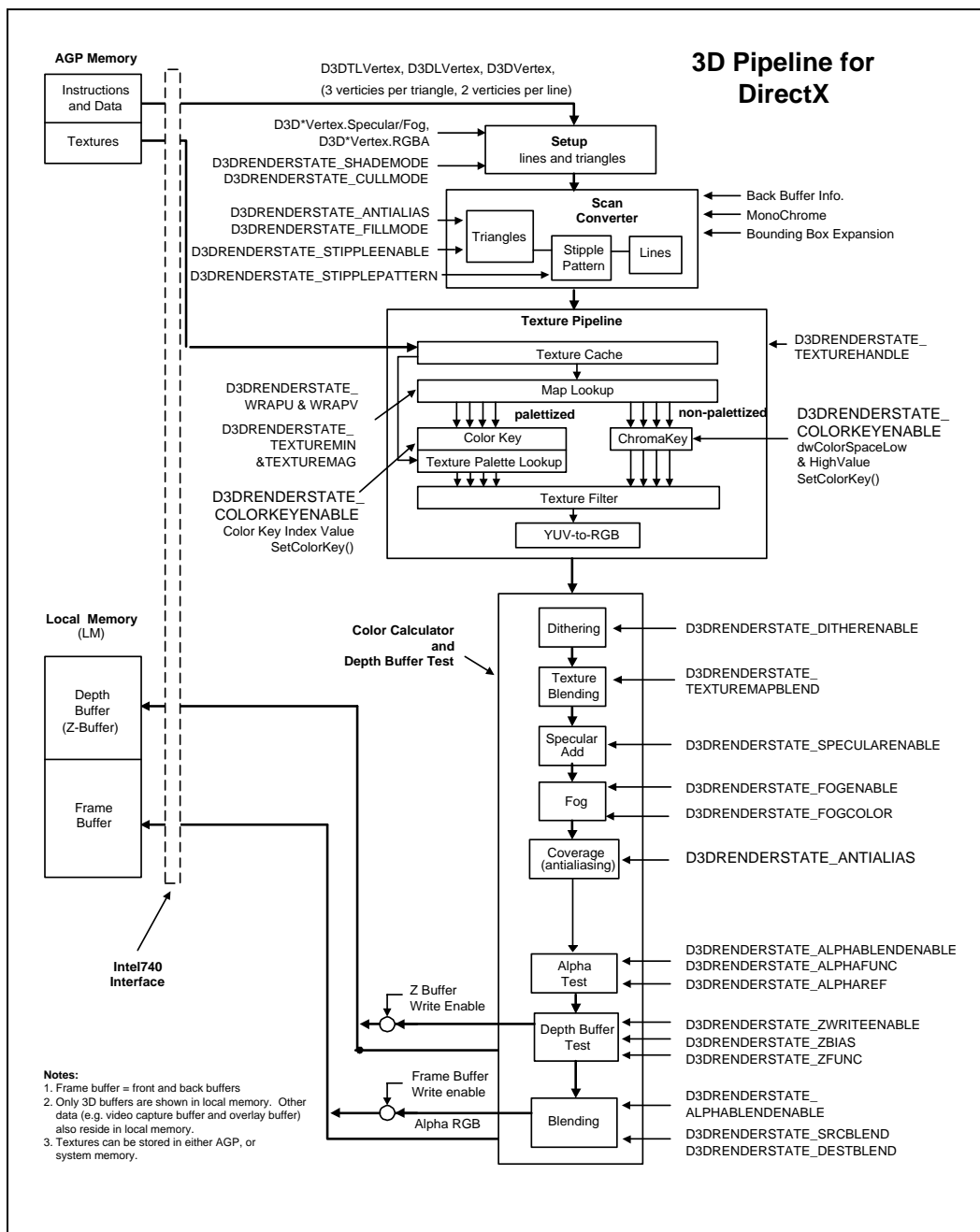
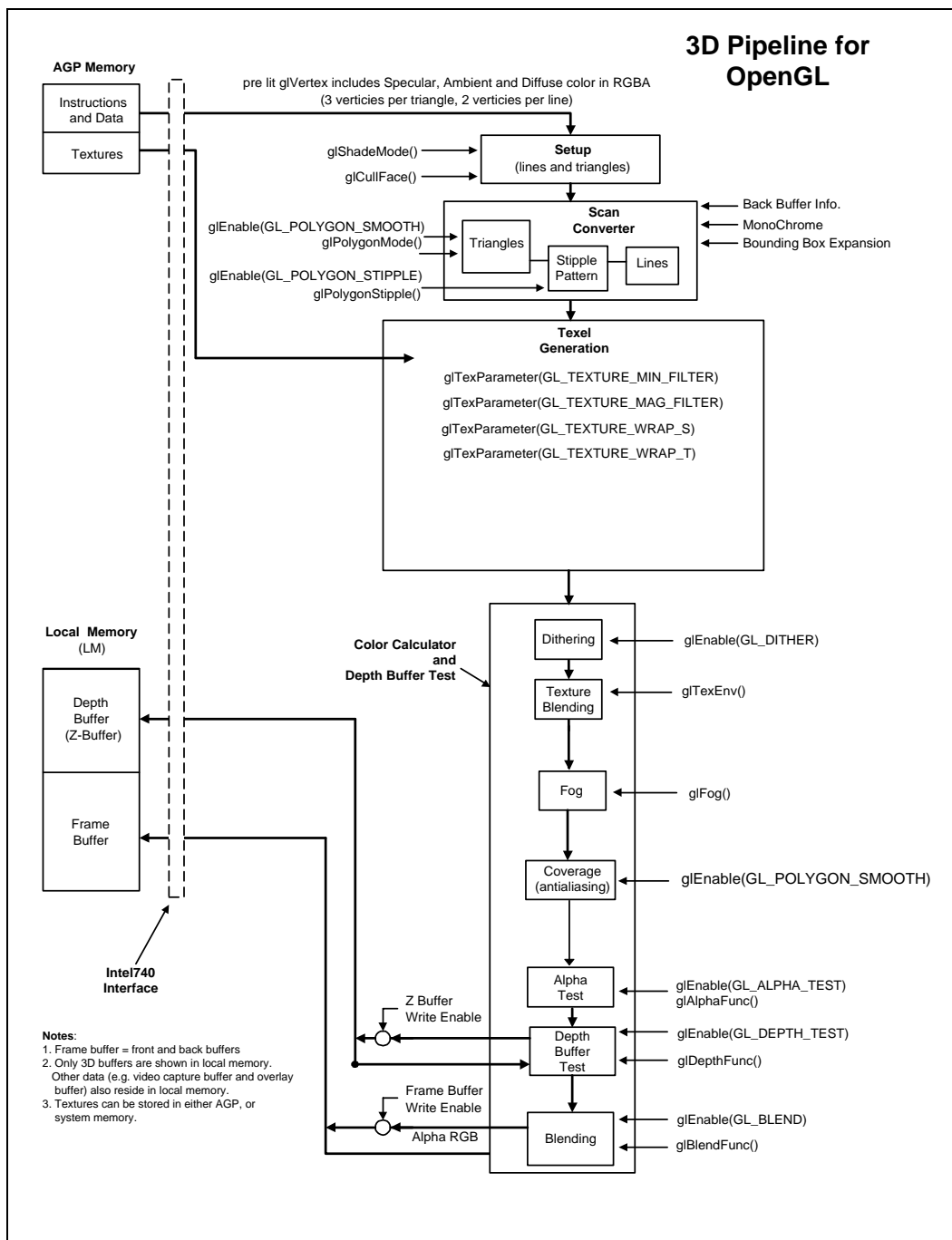


Figure 2-8. 3D Pipeline for OpenGL



The four main modules within the 3D Pipeline are:

Setup Engine

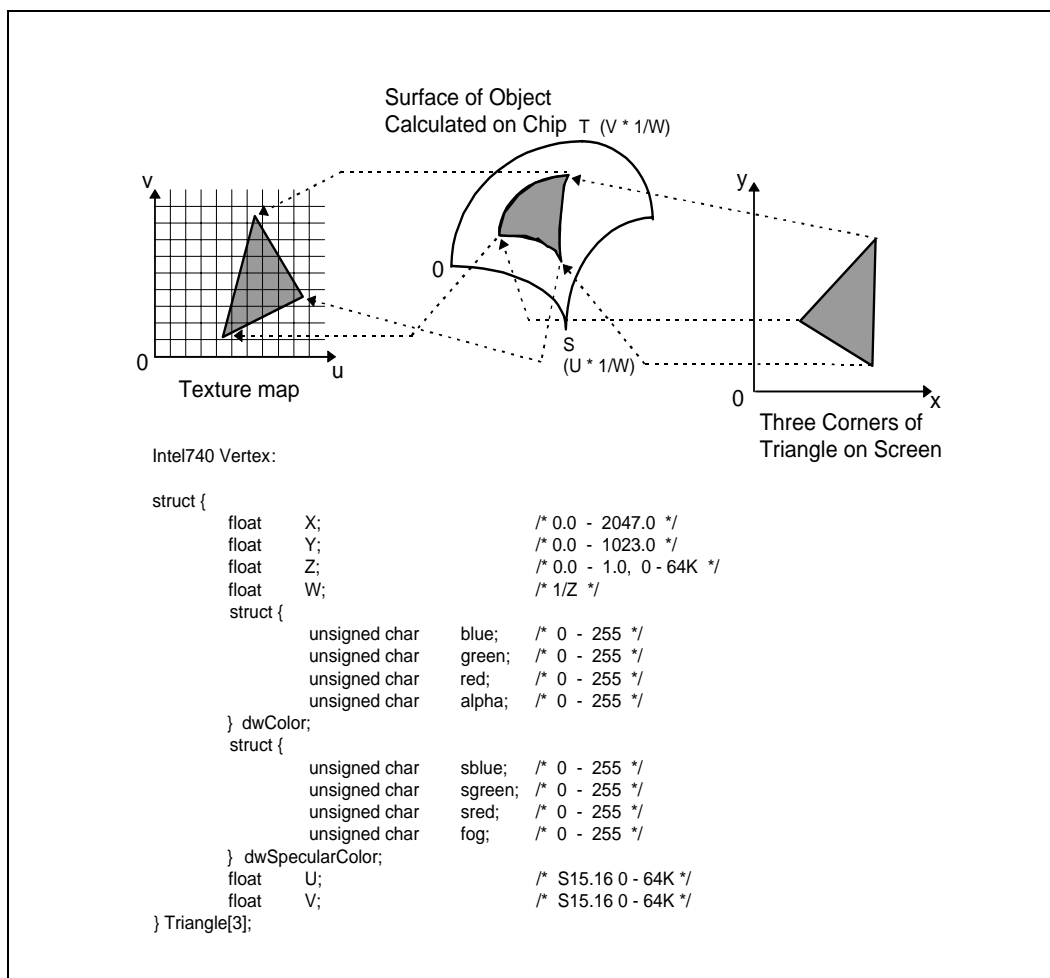
The Setup Engine performs the necessary calculations to make the geometry data useful for the rest of the pipeline. Some of the functions include culling, and perspective correct calculation of texture coordinates as they correspond to pieces of the geometry.

Scan Converter	The Scan Converter performs functions in parallel with the Setup Engine to read vital information such as fog, specular RGB, and blending data and sends it on to the Texture Pipeline so that the Texture Pipeline does not have to stop the flow of the pipeline in order to wait for this data.
Texture Pipeline	The Texture Pipeline receives the texture coordinate data information from the Setup Engine and texture blend information from the Scan Converter and stores this information in the texture cache. It performs texture chroma and color key match, texture bilinear interpolation, and YUV to RGB conversions.
Color Calc./Depth Test	The Color Calculator/Depth Test is where the color data such as fogging, specular RGB, texture blend, and alpha blend is processed. The Color Calculator computes the resulting color of a pixel. The red, green, blue, and alpha are combined with the corresponding components resulting from the Texture Pipeline unit. These textured pixels are then modified by the specular and fog parameters to create specular highlighted, fogged, textured pixels which are color blended with the existing values in the frame buffer. Alpha and depth buffer tests are conducted which will determine whether the frame and depth buffers will be updated with new pixel values.

2.2.2 3D Primitives

The 3D primitives are lines, triangles, and state variables. Pipeline flushes occur when updating the palette and stipple memories, since these are too large to allow pipelining of their data. In either case, all primitives rendered after a change in state variables will reflect the new state. Figure 2-9 shows the triangle data structure which is handled by the Intel740 drivers and also shows how the texture is mapped from the texture coordinate U, V space to the normalized S, T object space where perspective correction is applied to the texture as well as simulated curvature before being mapped to the object in X, Y screen coordinates. The triangle data structure is passed to the Intel740 drivers by either the DirectX or the OpenGL API call backs.

Figure 2-9. Triangle as the Intel740 Driver Sees It



Example 2-1. Sending Data to the Intel740 Using DirectX

When using DirectX, the data format for a vertex which can be sent to the Intel740 driver via a DirectX execute buffer, or by using the DrawPrimitive or DrawIndexedPrimitive command is a D3DTLVERTEX, D3DLVERTEX, or D3DVERTEX data structure. The Intel740 does the rasterization or rendering portion of the 3D pipe. The user must set up the appropriate lighting and transforms regardless of vertex type. The difference is that the DirectX API will know to perform lighting and transforms as preset by the user when a D3DVERTEX is sent, or just transforms when the D3DLVERTEX is sent. Lighting and transformation is not done by the Intel740, but will be done by the API software in these instances. See the Microsoft DirectX 5.0 documentation for more information on how to set up the lighting and transformations. The D3DTLVERTEX data structure is illustrated below.

```
D3DTLVERTEX TYPE
typedef struct _D3DTLVERTEX {
    union {
        D3DVALUE sx; // sx is the screen coordinate of the x position of the vertex
        D3DVALUE dvSX;
    };
    union {
        D3DVALUE sy // sy is the screen coordinate of the y position of the vertex
        D3DVALUE dvSY;
    };
    union {
        D3DVALUE sz; // sz is the z position of the vertex used for z compares
        D3DVALUE dvSZ;
    };
    union {
        D3DVALUE rhw; // rhw is the 1/z value for the vertex or the reciprocal
                       //of homogeneous
        D3DVALUE dvRHw; // w. This value is 1 divided by the distance from the
                       //origin to the object
    };
    // along the z-axis.
    };
    union {
        D3DCOLOR color; // color corresponds to the vertex color components of red,
                       //green, blue, and alpha.
    };
D3DCOLOR dcColor;
    };
    union {
        D3DCOLOR specular; // specular corresponds to the vertex specular color
                       //component
        D3DCOLOR dcSpecular; // consisting of sred, sgreen, and sblue. The alpha of
                       //the specular color is used for the fog density value.
    };
    union {
        D3DVALUE tu; // tu corresponds to the texture map horizontal component.
        D3DVALUE dvTU;
    };
    };
    union {
        D3DVALUE tv; // tv corresponds to the texture map vertical component.
        D3DVALUE dvTV;
    };
    };
} D3DTLVERTEX, *LPD3DTLVERTEX;
```


The Intel740 supports the following different D3DPRIMITIVETYPES for DrawPrimitive:

D3DPT_POINTLIST	Renders a collection of isolated points
D3DPT_LINELIST	Renders a list of isolated straight line segments
D3DPT_LINESTRIP	Renders a single polyline
D3DPT_TRIANGLELIST	Renders a sequence of isolated triangles
D3DPT_TRIANGLESTRIP	Renders a triangle strip
D3DPT_TRIANGLEFAN	Renders a triangle fan

Below is the DirectX function prototype for DrawIndexedPrimitive which is used to call the Intel740 driver to take the triangle data and begin the hardware rasterization process.

```
HRESULT IDirect3DDevice2::DrawIndexedPrimitive(
    D3DPRIMITIVETYPE type,
    D3DTLVERTEXTYPE D3DTLVertex,
    LPVOID VertextsListPointer,
    DWORD VertextsCount,
    LPWORD VertextsIndexList,
    DWORD VertextsIndexCount,
    DWORD DrawIndexedPrimitiveFlags);
```

The following code segment illustrates using DrawIndexedPrimitive to send the vertex data to the Intel740, assuming that the triangle information is ready for rendering:

```
HRESULT ddval
LPDIRECT3DDEVICE lpDev;
TransformVerticesTo3DView();
LightVertices();
TransformVerticesTo2DScreen();
if ((ddrval = lpDev->BeginScene()) != D3D_OK)
    return FALSE;
//beginning of atomic block for Direct 3D rendering
ddrval=lpDev->DrawIndexedPrimitive(D3DPT_TRIANGLELIST,
    D3DVT_TLVERTEX,
    (LPVOID)pvTLVertex,
    iNumVertex,
    (LPWORD)pdwIndex,
    iNumFaces*3,
    0) ;
if (ddrval != DD_OK)
    return FALSE;
//end of atomic block for Direct 3D rendering
if ((ddrval = lpDev->EndScene()) != D3D_OK)
    return FALSE;
```

It is best to do the transformations and lighting for the entire scene before the rendering, as implied in the code segment above. Multiple triangle lists can be sent within the BeginScene() and EndScene() call without hampering the performance. A triangle list larger than 85 triangles is recommended while a list of 512 triangles is optimal. See Chapter 4 for in-depth triangle list performance information.

Example 2-2. Sending Data to the Intel740 Using OpenGL

The three ways to send rendering information to the Intel740 using OpenGL are immediate method, vertex arrays, and display lists. This document first shows the immediate method, which is straightforward and which helps to understand the second and preferred vertex array method. The display list method is not discussed in this document; it can be found in the *OpenGL Programming Guide*. This document is concerned with showing the user how to implement OpenGL calls which will utilize the features of the Intel740, therefore this manual will not discuss overall OpenGL programming methods. It should be noted that the OpenGL vertex information sent to the Intel740 will be pre-lit, which means that the RGBA component will have already included the specular, diffuse and ambient lighting for the vertex.

OpenGL describes vertex information a little bit differently than DirectX. For instance, to specify an OpenGL vertex and its surface and texture attributes the following code could be used:

```
glBegin();
    glColor*();// Set current color
    glTexCoor*();// Set texture coordinates
    glEdgeFlag*();// Control drawing of edges
    glVertex*();// Set vertex coordinates
glEnd();
```

“*” specifies the type of arguments the function call will pass in the function parameters. For glVertex, the types conform to the following:

```
void glVertex{234}{sifd}[v](TYPE coords);
```

Where “(234)” specifies the number of coordinates from as few as two for (x,y) to as many as four for (x,y,z,w). Then the “{sifd}” portion describes the data type as either “short”, “int”, “float”, or “double.” The next portion of the function, “{v}” is used to specify that a pointer to a vector (or array) will be past in the parameter rather than a series of individual arguments.

It is important to send the glVertex() command last, because the information sent previously will be used to describe the vertex at this point.

To describe all of the component information of a vertex including the texture coordinates, color, and edge flags, each of the functions between the glBegin() and glEnd() may be called. Before making the glColor call, other calls to set the specular lighting, fogging and antialiasing methods should be called. These calls are discussed in the 3D features section of this document where for each feature of the Intel740 such as fogging, an OpenGL implementation is provided. The glBegin() and glEnd() are used to specify the beginning and end of an atomic primitive. There are different types of primitives which can be passed as arguments to glBegin(). They are as follows:

GL_POINTS	Renders a collection of isolated points
GL_LINES	Renders a list of isolated straight line segments
GL_TRIANGLES	Renders a sequence of isolated triangles
GL_LINE_STRIP	Renders a single polyline
GL_TRIANGLE_STRIP	Renders a triangle strip
GL_TRIANGLE_FAN	Renders a triangle fan
GL_QUAD	Renders a quad triangulated into individual triangles
GL_QUAD_STRIP	Renders quadrilateral strips triangulated into individual triangles
GL_POLYGON	Renders polygons triangulated into individual triangles

When using OpenGL, the best way to send vertex data to the driver is to use vertex arrays, which minimize the number of function calls required for one geometric object. Vertex arrays are a new feature of OpenGL 1.1. For the Intel740 it is best to minimize these function calls to improve performance and to reduce the redundant processing of shared vertices. The way to use the vertex arrays is as follows:

1. Enable each array type to be used:

```
void glEnableClientState(GLenum array);
```

Where array is one of the following symbolic constants: GL_VERTEX_ARRAY, GL_COLOR_ARRAY, GL_INDEX_ARRAY, GL_NORMAL_ARRAY, GL_TEXTURE_COORD_ARRAY, GL_EDGE_FLAG_ARRAY.

2. Point to each array to be rendered:

```
void glColorPointer(GLint size, GLenum type, GLsizei stride, const GLvoid *pointer);
```

```
void glTexCoordPointer(GLint size, GLenum type, GLsizei stride, const GLvoid *pointer);
```

```
void glEdgeFlagPointer(GLsizei stride, const GLvoid *pointer);
```

```
void glVertexPointer(GLint size, GLenum type, GLsizei stride, const GLvoid *pointer);
```

GLint size: is the number of coordinates per vertex, which must be 2, 3, or 4.

GLenum type: is the data type (GL_SHORT, GL_INT, GL_FLOAT, GL_DOUBLE).

GLsizei stride: is the byte offset between consecutive vertices (or other type).

GLvoid *pointer: points to the storage array for the vertices (or other type).

Note: There is such a thing as “intertwined” arrays where multiple types can be stored in a single array and therefore can be “pointed to” using the stride variable to indicate the offset from the beginning of the first group to the beginning of the next group of the type to be pointed to. For example, an intertwined array of x, y, z vertices and RGB color could be created and pointed to in this way:

```
static GLfloat intertwined[] =
{2.0, 0.3, 2.0, 200.0, 100.0, 0.0,
 2.0, 0.3, 0.0, 100.0, 100.0, 0.0,
 2.0, 2.0, 0.3, 100.0, 300.0, 0.0};
glEnableClientState(GL_COLOR_ARRAY);
glEnableClientState(GL_VERTEX_ARRAY);
glColorPointer(3, GL_FLOAT, 6 * sizeof(GLfloat), intertwined);
glVertexPointer(3, GL_FLOAT, 6 * sizeof(GLfloat), &intertwined[3]);
```

3. Render the data. The above calls remain on the application side of the graphics pipeline. In order to send the data to the Intel740 for rendering the user needs to “dereference” the arrays which cause them to be sent down the graphics processing pipeline. This can be done by either de-referencing a single array element from a sequence of array elements or from an ordered list of array elements. The following call is used to render an ordered list of array elements:

```
void glDrawArrays(GLenum mode, GLint first, GLsizei count);
```

GLenum mode: The primitive type.

GLint first: The start of the array to be processed

GLsizei count: The number of elements to be rendered.

```
glEnableClientState(GL_COLOR_ARRAY);
```

```
glEnableClientState(GL_VERTEX_ARRAY);
```

```
.
```

```
.
```

```
glEnableClientState(otherarray);
```

```
glColorPointer(3, GL_FLOAT, 6 * sizeof(GLfloat), intertwined);
```

```
glVertexPointer(3, GL_FLOAT, 6 * sizeof(GLfloat), &intertwined[3]);
```

```
.
```

```
.
```

```

gl*Pointer(...);
glDrawArrays(GL_TRIANGLES, 0, vertexs_count);
glDisableClientState(Glenum array);

```

The above call would render all of the arrays which have been enabled and pointed to.

2.2.3 Data Formats

The data value ranges are independent of the API. Table 2-1 gives a listing of each data format and the corresponding domain and range values.

Table 2-1. Data Formats

Parameters	Input Format	Domain	Range
Vertex X, Y	32-bit Floating Point	0.0–2048	x: 0–2047
y: 0–1023	Depth (Z)	32-bit Floating Point	0.0–1.0
0–64K	Texture U, V	32-bit Floating Point	0–64K
0–64K (32K)	Texture W	32-bit Floating Point	0.0–1.0
1/z	Color R, G, B, A	Fixed 0.8	0–255
0–255	Specular Color R, G, B	Fixed 0.8	0–255
0–255	Fog Factor	Fixed 0.8	0–255
0–255			

2.2.4 Surface Color Attributes

Surface attributes are those items which allow the user to define the object’s visual quality and which can be combined in a number of ways to create different atmospheric and lighting effects. The surface attributes which the Intel740 supports are discussed in the following subsections:

- “Fogging” (below)
- “Specular Highlighting” on page 2-18
- “Alpha Testing” on page 2-22
- “Color Dithering” on page 2-22
- “Shading” on page 2-23
- “Stippled Pattern” on page 2-24

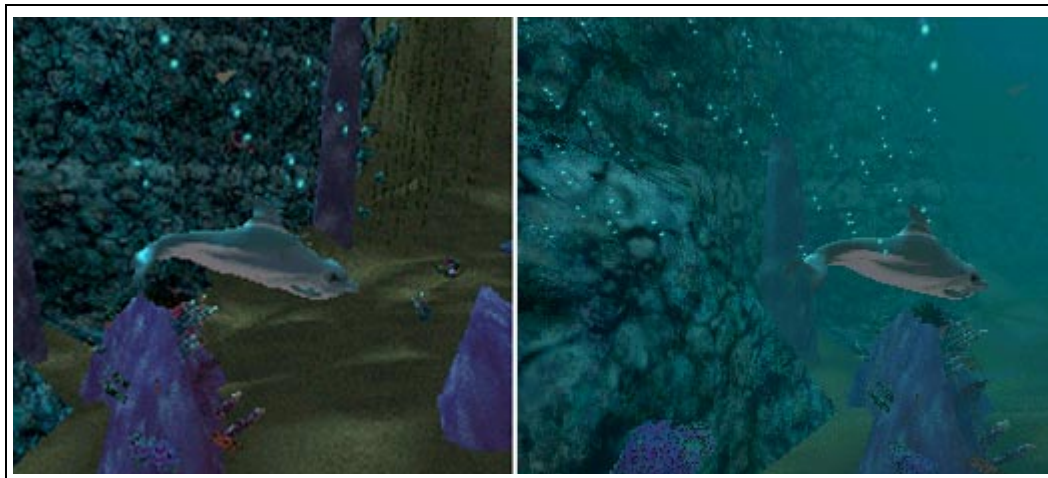
2.2.4.1 Fogging

Fogging adds the effect of density to the atmosphere. As an object goes farther away from the viewer, it appears to become more “cloudy” or “foggy” than closer objects. Fogging is specified at each vertex and is interpolated to each pixel center. If fog is disabled, the incoming color intensities are passed unchanged. Fog is linearly interpolative, with the pixel color determined by the following equation:

$$C = f * C_p + (1 - f) * C_f$$

where f is the fog coefficient per pixel, C_p is the polygon color, and C_f is the fog color.

Figure 2-10. Effects of Fogging Off vs Fogging On

**Example 2-3. Enabling Fogging with DirectX**

The following code shows how to enable fogging using the DirectX API. The first step is to turn fogging on by setting the “D3DRENDERSTATE_FOGENABLE” state to “TRUE”. The second step is to set the color of the fog as shown below where D3DCOLOR has a red, green and blue value that will correspond to the color of the fog.

```
SetRenderState(D3DRENDERSTATE_FOGENABLE, TRUE);
SetRenderState(D3DRENDERSTATE_FOGCOLOR, <D3DCOLOR>);
```

The density of the fog is specified by setting the alpha component of the specular value of a vertex as shown below using the D3DLVERTEX data type:

```
D3DLVERTEX pLVertex;
pLVertex.specular = RGBA_MAKE( sred, sgreen, sblue, FOG_DENSITY);
```

The density of the fog value is between 0 and 255, where 0 is dense, completely opaque fog and 255 completely clear or no fog.

Example 2-4. Enabling Fogging with OpenGL

There are several steps and many choices when implementing fogging through the OpenGL API. The following code shows how to set the multiple fogging values:

```
glEnable(GL_FOG) { ... };
```

Enables fogging; other values corresponding to the fog can be set within the braces.

```
glFogi(GL_FOG_MODE, <MODE>);
```

Where <MODE> is either GL_LINEAR, GL_EXP, or GL_EXP2. The GL_LINEAR flag is hardware accelerated with the Intel740.

```
GLfloat fogColor[4] = {0.5, 0.5, 0.5, 1.0};
glFogfv(GL_FOG_COLOR, fogColor);
```

Sets the fog color from the values set in the fogColor array. Fog color can be set as RGB values or from a color index.

```
glFogf(GL_FOG_DENSITY, <VALUE>);
```

Sets the fog density to <VALUE> which can be a floating point number from 0.0 to 1.0. The fog density is used when calculating GL_EXP or GL_EXP2 fog values.

```
glFogf(GL_FOG_START, <START_VALUE>);
```

Sets the start of the fog in the view. The <START_VALUE> corresponds to a “z” value in the view and can be any floating point value within the view volume z range.

```
glFogf(GL_FOG_END, <END_VALUE>);
```

Sets the end of the fog in the view. The <END_VALUE> corresponds to the point in the view where the user wants fogging to end and can be a floating point value with the view volume z range.

```
glHint(GL_FOG_HINT, <HINT_VALUE>);
```

Specifies how the fog is calculated where <HINT_VALUE> is either GL_NICEST or calculated per pixel, or GL_FASTEST, calculated per vertex. The Intel740 accelerates GL_FASTEST.

For OpenGL, the fog equations are as follows:

$$f = e^{-(\text{density} * z)} \quad (\text{GL_EXP})$$

$$f = e^{-(\text{density} * z)^2} \quad (\text{GL_EXP2})$$

$$f = \text{end} - z / \text{end} - \text{start} \quad (\text{GL_LINEAR})$$

2.2.4.2 Specular Highlighting

Specular highlighting adds the effect of a “hot spot” on an object which corresponds to the shininess of the material. The specular highlight can be varied by the amount specified for each red, green, and blue component. The Intel740 has the capability to utilize colored specular highlights which adds to the realism of a scene. For instance, if you have a blue light shining on a red apple, the specular highlight would be blue in real life. With the Intel740, it is possible to create a specular highlight of any color.

Figure 2-11. Effects of Using Specular Highlighting



Example 2-5. Enabling Specular Highlighting with DirectX

The specular color of a vertex is set to red as illustrated with the following DirectX code:

```
D3DLVERTEX pLVertex;  
pLVertex.specular = RGBA_MAKE( 255, 0, 0, FOG_DENSITY);
```

In order to enable the specular highlights with DirectX so that they are visible, the following D3DRENDERSTATE is set to true:

```
SetRenderState(D3DRENDERSTATE_SPECULARENABLE, TRUE);
```

Example 2-6. Enabling Specular Highlighting with OpenGL

Specular highlighting is added in to the color equation at the application's lighting stage which formulates the RGBA color sent to the driver. To set the specular lighting component in OpenGL the following code may be used:

```
GLfloat mat_specular[] = {1.0, 1.0, 1.0, 1.0}  
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
```

2.2.4.3 Alpha Blending

Alpha blending adds the material property of transparency or opacity to an object. Alpha blending requires a source red, green, blue, and alpha component and a destination red, green, blue and alpha component. This is so that a glass surface on top (source) of a red surface (destination) would allow much of the red base color to show through. The Intel740 blends the source R_s , G_s , B_s , A_s component with the destination R_d , G_d , B_d , A_d component by the following formula:

$$(R', G', B', A') = (R_s S_r + R_d D_r, G_s S_g + G_d D_g, B_s S_b + B_d D_b, A_s S_a + A_d D_a)$$

Where S_r , S_g , S_b , S_a is a blending factor for the source and D_r , D_g , D_b , D_a is a blending factor for the destination.

Figure 2-12. Effects of Using Alpha Blending

**Example 2-7. Enabling Alpha Blending with DirectX**

To enable alpha blending with DirectX, the ALPHABLENDENABLE flag must be set to TRUE, and then a SRCBLEND and DESTBLEND flag must be specified as shown below:

```
SetRenderState(D3DRENDERSTATE_ALPHABLENDENABLE, TRUE);
SetRenderState(D3DRENDERSTATE_SRCBLEND, <D3DBLEND FLAG>);
SetRenderState(D3DRENDERSTATE_DESTBLEND, <D3DBLEND FLAG>);
```

The D3DBLEND FLAG is ZERO, ONE, SRCCOLOR, INVSRCOLOR, DESTCOLOR, INVDESTCOLOR, BOTHSRCALPHA, or BOTHINVSRCALPHA. The blending factors are calculated depending on the D3DBLEND FLAG according to the formulas shown in Table 2-2. A common implementation is to set the source flag to SRCCOLOR and the destination flag to INVSRCOLOR.

Example 2-8. Enabling Alpha Blending with OpenGL

To enable alpha blending with OpenGL, the following function call is made:

```
glEnable(GL_BLEND);
```

To set the source and destination blending factors, the following call is made:

```
glBlendFunc(<SOURCE_FLAG>, <DESTINATION_FLAG>);
```

The <SOURCE_FLAG> and <DEST_FLAG> can be set to any of the flags in the chart below and the resulting blend will be what the corresponding values equate to when plugged into the Intel740 equation above.

Table 2-2. Alpha Blend Functions for OpenGL & DirectX

FLAG	Source Blend Factor	Destination Blend Factor
GL_ZERO D3DBLEND_ZERO	Sr = 0 Sg = 0 Sb = 0 Sa = 0	Dr = 0 Dg = 0 Db = 0 Da = 0
GL_ONE D3DBLEND_ONE	Sr = 1 Sg = 1 Sb = 1 Sa = 1	Dr = 1 Dg = 1 Db = 1 Da = 1
GL_SRC_COLOR D3DBLEND_SRCOLOR	Sr = Rs Sg = Gs Sb = Bs Sa = As	
GL_DST_COLOR D3DBLEND_DESTCOLOR		Dr = Rd Dg = Gd Db = Bd Da = Ad
GL_ONE_MINUS_SRC_COLOR D3DBLEND_INVSRCOLOR	Sr = 1-Rs Sg = 1-Gs Sb = 1-Bs Sa = 1-As	
GL_ONE_MINUS_DST_COLOR D3DBLEND_INVDESTCOLOR		Dr = 1-Rd Dg = 1-Gd Db = 1-Bd Da = 1-Ad
GL_SRC_ALPHA D3DBLEND_SRCALPHA	Sr = As Sg = As Sb = As Sa = As	Dr = As Dg = As Db = As Da = As
GL_ONE_MINUS_SRC_ALPHA D3DBLEND_INVSRALPHA	Sr = 1-As Sg = 1-As Sb = 1-As Sa = 1-As	Dr = 1-As Dg = 1-As Db = 1-As Da = 1-As
D3DBLEND_BOTHSRALPHA	Sr = As Sg = As Sb = As Sa = As	Dr = 1-As Dg = 1-As Db = 1-As Da = 1-As
D3DBLEND_BOTHINVSRCALPHA	Sr = 1-As Sg = 1-As Sb = 1-As Sa = 1-As	Dr = As Dg = As Db = As Da = As

2.2.4.4 Alpha Testing

The Intel740 supports the use of alpha blend testing functions. This allows the user to control how objects in the scene are alpha blended. When using source alpha blending the user does not need to create an alpha buffer. When using source alpha blending, the alpha channel of the textures are used for the blending formulas and there is no need for an alpha buffer. The user must remember to sort from back to front, so that the blending is performed correctly.

Example 2-9. Enabling Alpha Testing Functions With DirectX

To enable alpha testing functions with DirectX, the following render states are set:

```
SetRenderState(D3DRENDERSTATE_ALPHABLENDENABLE, TRUE);
SetRenderState(D3DRENDERSTATE_ALPHATESTENABLE, TRUE);
SetRenderState(D3DRENDERSTATE_ALPHAFUNC, <D3DCMPFUNC>);
SetRenderState(D3DRENDERSTATE_ALPHAREF, <ALPHA REF> );
```

Where <D3DCMPFUNC> can be set to D3DCMP_NEVER, D3DCMP_LESS, D3DCMP_EQUAL, D3DCMP_LESSEQUAL, D3DCMP_GREATER, D3DCMP_NOTEQUAL, D3DCMP_GREATEREQUAL, or D3DCMP_ALWAYS. And where <ALPHA REF> is a value specifying a reference alpha value against which pixels are tested when alpha-testing is enabled. This value is in the range of 0 to 1 and must be 8 bits or less for the Intel740. The default value is 0.

Example 2-10. Enabling Alpha Testing Functions With OpenGL

To enable alpha testing functions with OpenGL, the following render states are set:

```
glEnable(GL_ALPHA_TEST);
glAlphaFunc(<GLFUNC>, <GLREF>);
```

Where <GLFUNC> is GL_NEVER, GL_ALWAYS, GL_LESS, GL_LEQUAL, GL_EQUAL, GL_GEQUAL, GL_GREATER, or GL_NOTEQUAL. <GLREF> must be between 0 and 1.

2.2.4.5 Color Dithering

Color dithering is created by a pattern of pixels which are more than one color. When looked at from a distance, the combined effect is a new color. In this manner, many different colors can be simulated by combining a few colors. Color dithering takes advantage of the human eye's propensity to "average" the colors in a small area. With limited color fidelity available, large areas of "flat" colors can exist. Color dithering takes the input of color, alpha, and fog components and converts them from 8 bits to five- or six-bit components. Color dithering simulates 256-level color resolution by an ordered pattern of 32- or 64-level color pixels. A four-bit dither value is obtained by addressing a 4x4 matrix with the pixel's x and y (2 LSBs of each). The matrix repeats every four pixels in both directions. The value obtained is appropriately shifted to align with (what would be otherwise) truncated bits of the component being dithered. It is then added with the component and the result is truncated to the five (six for green) MSBs.

Example 2-11. Enabling Color Dithering with DirectX

To enable color dithering with DirectX do the following:

```
SetRenderState(D3DRENDERSTATE_DITHERENABLE, TRUE);
```

Example 2-12. Enabling Color Dithering with OpenGL

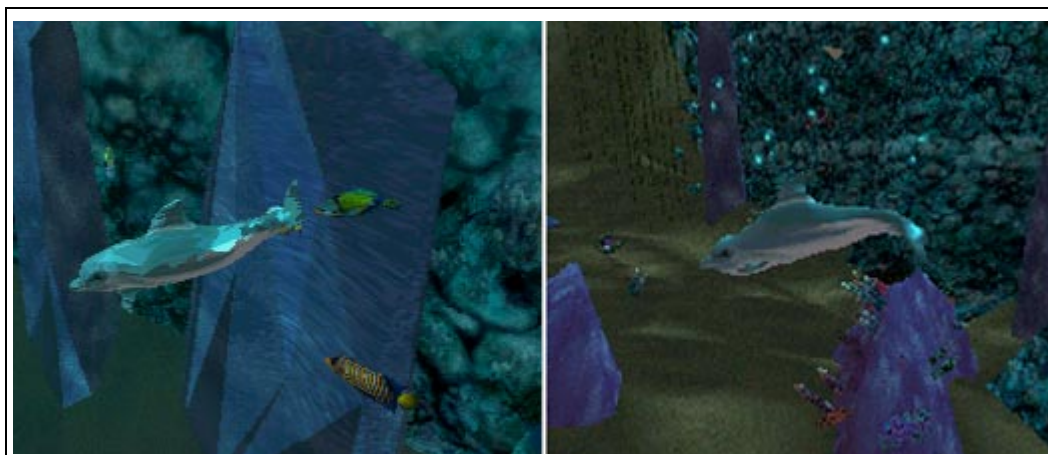
To enable color dithering with OpenGL do the following:

```
glEnable(GL_DITHER);
```

2.2.4.6 Shading

The Intel740 shading attributes determine how the colors of the polygons (triangles) are interpolated for each pixel in a surface. The Intel740 allows each of the alpha, fog, specular, and color attributes to be shaded individually. There are two types of shading performed by the Intel740: flat shading and Gouraud shading. Flat shading makes objects appear blocky, since each polygon (triangle) face is denoted by a solid color. This is because flat shading takes a specified attribute from the first passed vertex and uses this attribute to cover every pixel in the polygon. Gouraud shading smooths the appearance of adjacent polygons (triangles) so that a sphere which looked blocky flat shaded can be made to look more rounded. This is because Gouraud shading takes the three vertices of the triangle and interpolates over the entire surface to blend the vertex colors and attributes such as fog, specularity and transparency (alpha).

Figure 2-13. Effects of Flat Shading vs. Smooth Shading



Example 2-13. Enabling Shading with DirectX

In order to enable either flat or Gouraud shading using DirectX, the following render state is set:

```
SetRenderState(D3DRENDERSTATE_SHADEMODE, <D3DSHADEMODE>);
```

Where the shade mode is either D3D_GOURAUD or D3D_FLAT.

Example 2-14. Enabling Shading with OpenGL

In order to enable either flat or Gouraud shading using OpenGL, the following call can be made:

```
glShadeModel (<GLSHADEMODE>)
```

Where the shade mode is either GL_SMOOTH, for Gouraud shading, or GL_FLAT for flat shading.

2.2.4.7 Stippled Pattern

The stipple pattern feature of the Intel740 is used to set values in a 32x32 pixel matrix to be either 1 or 0, where 0 means that the corresponding portion of the pattern will be rendered as a black pixel. Stippled patterns can be used when the application wants the screen to fade to black by changing the pattern to have more zeros set for each frame rendered.

Example 2-15. Enabling Stippled Patterns with DirectX

To enable stippled pattern for the Intel740 using DirectX, do the following:

```
SetRenderState(D3DRENDERSTATE_STIPPLEENABLE, TRUE);
SetRenderState(D3DRENDERSTATE_STIPPLEPATTERN00, 1 OR 0);
.
.
SetRenderState(D3DRENDERSTATE_STIPPLEPATTERN31, 1 OR 0);
```

The default value for all of the stipple patterns is 0. When a stippled pattern is enabled and no stipple pattern is set, the result is a black screen.

Example 2-16. Enabling Stippled Patterns with OpenGL

To enable stippled pattern for the Intel740 using OpenGL do the following:

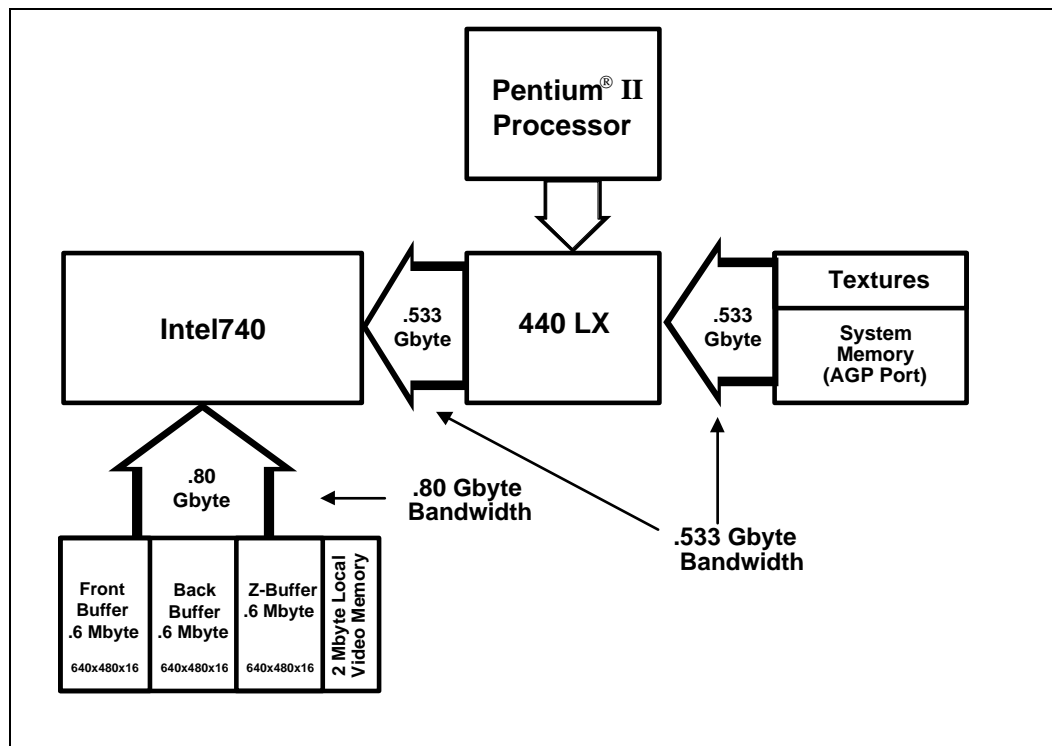
```
glEnable(GL_POLYGON_STIPPLE);
glPolygonStipple(const Glubyte *StippleMatrix);
```

Where StippleMatrix is a pointer to a 32x32 pixel bitmap interpreted as a mask of 0s and 1s.

2.2.5 Texture Map Attributes

The Intel740 allows virtually unlimited texture usage. This is because textures can be stored in the AGP system memory otherwise known as non-local video memory. The amount of AGP memory available for the application is limited by the amount of system RAM which can be allocated. Therefore, if a system has 32 Mbytes of RAM available, 20 Mbytes could be used for textures. Using AGP for texture memory complements the performance of the Intel740, since textures can be mapped directly from AGP memory to the Intel740 without using the CPU. This mapping is done in parallel with the Intel740 local video memory transfers for frame buffers. The total bandwidth enabled by the parallel throughput is up to 1.3 Gbytes per second. The Intel740 also “tiles” textures in AGP memory in order to minimize page faults and storage overhead which increases both performance and texture space. Textures can not be put in local video memory.

Figure 2-14. Getting 1.3 Gbytes of Concurrent Throughput with the Intel740



There are many ways to manipulate surface textures with the many Intel740 Texture Map Attributes. The categories are described in the following subsections:

- “Texture Map Formats” on page 2-26
- “Texture Map Blending” on page 2-28
- “Texture Map Chroma Keying and Color Keying” on page 2-30
- “Texture Wrapping Formats” on page 2-32
- “Texture Map Filtering” on page 2-33
- “Texture Mipmapping” on page 2-34

2.2.5.1 Texture Map Formats

The Intel740 supports up to 16 bits of color in various texture formats. There are three ways to catalog texture types: ARGB, AYUV, or YUV. All the texture formats listed below are supported as either palettized or non-palettized. When the amount of bits per texel in a texture is less than 16, the color information is stored in a palette consisting of 256 16-bit entries. The texture cache is used to store previously accessed texels needed for blending or other purposes, so that additional reads from memory are not needed. The Intel740 supports images whose dimensions are a power of two. The dimensions do not have to be square.

DirectX Texture Map Formats supported:

- 1555ARGB
- 0565ARGB (DirectX default for palettized)
- 4444ARGB (DirectX default for palettized with alpha)
- 422YUV (UV is 2's complement) (YUY2 FOURCC)
- 422YUV (UV is excess 128) (YUY2 FOURCC)
- 0555AYUV (texture data compression)
- 1544AYUV (texture data compression)
- Palettized 1, 2, 4, and 8 bit.

OpenGL Texture Map Formats supported:

- RGB5 (0555ARGB)
- RGBA4(4444ARGB)
- RGB5_A1(1555ARGB)

Example 2-17. Creating a Texture Surface with DirectX

The following DirectX example shows how to create a 4444 ARGB texture surface in AGP non-local video memory:

First set the pixel format for the 4444 ARGB:

```
DDPIXELFORMAT ddpf;
DDSURFACEDESC ddsd;
ddpf.dwSize = sizeof(ddpf);
ddsd.dwSize = sizeof(ddsd);
ddpf.dwRGBBitCount = 16 //Total number of bits including alpha
ddpf.dwRBitMask = 0x0F00; //Specify the masks for color components
ddpf.dwGBitMask = 0x00F0;
ddpf.dwBBitMask = 0x000F;
ddpf.dwRGBAlphaBitMask = 0xF000;
ddpf.dwFlags = DDPF_RGB; //specify the pixel format is valid
ddsd.dwFlags = DDSD_PIXELFORMAT;
```

Next set the correct direct draw surface capability flags and creates the surface:

```
IDIRECTDRAW*lpdd;
IDIRECTDRAWSURFACE*lpTextureSurface;
HRESULT ddrval;
ddsd.dwSize = sizeof(ddsd);
ddsd.dwHeight = 128;
ddsd.dwWidth = 128;
```

```

ddsds.wFlags = DDS_DCAPS | DDS_HEIGHT | DDS_WIDTH;
ddsds.ddsCaps = DDSCAPS_TEXTURE | DDSCAPS_ALLOCONLOAD | DDSCAPS_VIDEMEMORY |
                DDSCAPS_NONLOCALVIDMEM;
ddrval = lpdd->CreateSurface(&ddsds, &lpTextureSurface, NULL);

```

Once a texture surface has been created, a palette and texture can be loaded onto the surface using the DirectDraw sample functions `DDLLoadPalette` and `DDReLoadBitmap` from the `ddutil.cpp` file included in the DirectX 5.0 SDK.

```

IDIRECTDRAWPALETTE *lpDDPal;
lpDDPal = DDLLoadPalette(lpDD, "MyTexture.bmp");
ddrval = lpTextureSurface->SetPalette(lpDDPal);
ddrval = DDReLoadBitmap(lpTextureSurface, "MyTexture.bmp");

```

To enable the texture for rendering, the following state change is made where the texture handle which points to a texture surface is enabled so that a particular texture surface will be rendered:

```

D3DTEXTUREHANDLE HTex;
lpTextureSurface->GetHandle(lpD3DDevice, &HTex);
SetRenderState(D3DRENDERSTATE_TEXTUREHANDLE, &HTex);

```

The texture handle assigned to the texture surface is enabled.

Example 2-18. Creating a Texture Surface with OpenGL

In OpenGL 1.1, it is recommended to use texture objects. Texture objects are beneficial because they allow the programmer to specify which texture is active with one simple call after these three steps are taken:

1. Generate texture names; a texture name can be any nonzero unsigned integer. The following call should be used when generating a texture name to ensure that a unique texture name is created.

```
glGenTextures(GLsizei n, GLuint *TextureName);
```

This call returns a texture object pointed to through `TextureName`. When using an array of texture names, `n` corresponds to the number of unused texture names in the array of texture names.

2. The next step is to bind texture objects to texture data. The following call is used:

```
glBindTexture(GLenum target, GLuint *TextureName);
```

This causes the texture specified by `TextureName` to become active where `target` is either `GL_TEXTURE_1D`, or `GL_TEXTURE_2D` and `TextureName` is the same pointer used in `glGenTextures`.

3. The next step creates the texture surface which will from then on, correspond to the `TextureName` pointer:

```
glTexImage2D(GLenum <TARGET>, GLint <LEVEL>, GLint <INTERNALFORMAT>,
             GLsizei <WIDTH>, GLsizei <HEIGHT>, GLint <BORDER>, GLenum <FORMAT>, GLenum
             <TYPE>, GLvoid <PIXELS>);
```

<TARGET> is either `GL_TEXTURE_2D`, or `GL_PROXY_TEXTURE_2D`;

<LEVEL> is 0 or the number of texture resolutions to be used

<INTERNALFORMAT> is the texture format supported by the Intel740 and is `GL_RGB5` or `GL_RGBA4`, or `GL_RGB5_A1`

<WIDTH> and <HEIGHT> correspond to the dimensions of the texture; <BORDER> indicates the width of the border which is either 0 (if there is no border) or 1

<FORMAT> and <TYPE> describe the format and data type of the texture image data

<PIXELS> is a pointer to the texture image data. This data describes the texture image itself as well as its border.

When put together, creating and enabling a texture surface is done by the following:

```
glEnable(GL_TEXTURE_2D);
glGenTextures(1, &texture_name);
glBindTexture(GL_TEXTURE_2D, texture_name);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA16, width, height, 0, GL_RGBA16,
GL_UNSIGNED_BYTE, image_pointer);
```

The variable `image_pointer` points to the memory location where the image data is currently stored. Subsequent uses of the same image data need only use the `glEnable()` and the `glBindTexture()` calls.

2.2.5.2 Texture Map Blending

The Intel740 supports texture map blending modes that can be used to modify the pixel color by blending a textured surface with the underlying vertex color.

Example 2-19. Enabling Texture Blending with DirectX

DirectX texture blending modes are shown in Table 2-3. Each mode's behavior depends on whether a texture alpha is provided (RGB or RGBA). The color and alpha equations are given for each case. These equations employ the following definitions:

Cf	intrinsic (flat or Gouraud Interpolated) color of feature
Af	intrinsic (flat or Gouraud Interpolated) alpha of feature
Ct	color from texture data
At	alpha from texture data
Am	lsb of nearest-neighbor alpha from texture data
Co	color output of texture blend function
Ao	alpha output of texture blend function

Some of the modes degenerate to the same function if a texture alpha is not provided.

Table 2-3. DirectX Texture Map Blending Functions

Mode	Texture Mode	Pixel Color	Alpha	D3D Texture Modes (D3DBLEND_)
Decal	RGB	$Co = Ct$	$Ao = Af$	DECAL
Decal	RGBA	$Co = Ct$	$Ao = At$	
Modulate	RGB	$Co = Cf * Ct$	$Ao = Af$	MODULATE
Modulate	RGBA	$Co = Cf * Ct$	$Ao = At$	
Decal Alpha	RGB	$Co = Ct$	$Ao = Af$	DECALALPHA
Decal Alpha	RGBA	$Co = (1-At)*Cf + At*Ct$	$Ao = Af$	
Modulate Alpha	RGB	$Co = Cf * Ct$	$Ao = Af$	MODULATEALPHA
Modulate Alpha	RGBA	$Co = Cf * Ct$	$Ao = Af * At$	
Decal Mask	RGB	$Co = Ct$	$Ao = Af$	DECALMASK
Decal Mask	RGBA	If (Am) $Co = Ct$ Else $Co = Cf$	$Ao = Af$	
Modulate Mask	RGB	$Co = Cf * Ct$	$Ao = Af$	MODULATEMASK
Modulate Mask	RGBA	If (Am) $Co = Cf * Ct$ Else $Co = Cf$	$Ao = Af$	

Each of the DirectX texture blend states is described in detail below:

DECAL

In the Decal state, the output color is the texture color. The output alpha is the feature alpha with an RGB texel format and the texture alpha with an RGBA texel format.

MODULATE

In the Modulate state, the output color is the product of the texture color and the feature color. The output alpha is the feature alpha with an RGB texel format and is the texture alpha with an RGBA texel format.

DECALALPHA

In the Decal Alpha state, the output color is the texture color with an RGB texel format and is a texture alpha blended combination of the feature color and the texture color with an RGBA texel format. The output alpha is the feature alpha.

MODULATEALPHA

In the Modulate Alpha state, the output color is the product of the texture color and the feature color. The output alpha is the feature alpha with an RGB texel format and is the product of the feature alpha and the texture alpha, with an RGBA texel format.

DECALMASK

In the Decal Mask state, the output color is the texture color with an RGB texel format. With an RGBA texel format, the output color is the texture color if the nearest neighbor texel alpha lsb is 1 and is the feature color if the nearest neighbor texel alpha lsb is 0. The output alpha is the feature alpha.

MODULATEMASK

In the Modulate Mask state, the output color is the product of the feature color and the texture color with an RGB texel format. With an RGBA texel format, the output color is the product of the feature color and the texture color if the nearest neighbor texel alpha lsb is 1 and is the feature color if the nearest neighbor texel alpha lsb is 0. The output alpha is the feature alpha.

In order to use the texture map blending features with DirectX, first obtain a handle to the texture surface to be used for blending:

```
D3DTEXTUREHANDLE HTex;
lpTextureSurface->GetHandle(lpD3DDevice, HTex);
SetRenderState(D3DRENDERSTATE_TEXTUREHANDLE, &HTex);
SetRenderState(D3DRENDERSTATE_TEXTUREMAPBLEND, <D3DTEXTUREBLEND>);
```

Where the D3DTEXTUREBLEND values are (D3DTBLEND_) DECAL, DECALALPHA, DECALMASK, MODULATE, MODULATEALPHA, MODULATEMASK, or COPY.

Example 2-20. Enabling Texture Blending with OpenGL

Table 2-4 states the texture blend functions for OpenGL which the Intel740 supports.

Table 2-4. OpenGL Texture Blend Modes and Equations

Mode	Texture Mode	Pixel Color	Alpha	OpenGL Mode
DECAL	RGB	$Co = Ct$	$Ao = Af$	GL_DECAL
DECAL	RGBA	$Co = Cf(1-At)+Ct*At$	$Ao = Af$	GL_DECAL
MODULATE	RGB	$Co = Cf * Ct$	$Ao = Af$	GL_MODULATE
MODULATE	RGBA	$Co = Cf * Ct$	$Ao = Af*At$	GL_MODULATE

To enable texture map blending in OpenGL, the following code is used:

```
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, textureName);
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, <MODE>);
```

where <MODE> is stated as being either GL_DECAL or GL_MODULATE.

2.2.5.3 Texture Map Chroma Keying and Color Keying

Chroma Keying is analogous to the Hollywood “blue screen” effect whereby a color can be selected in the destination texture through which the source can be made visible on top. The source texture defines a color to be made transparent so that the destination texture appears to be visible through where that color is defined in the source texture. Both texture map chroma keying and color keying were designed to give the user the ability to make portions of a texture invisible so that the underlying scene can show through. The difference between texture map chroma keying and color keying is that chroma keying is used for non-palettized textures. Texture map color keying is used with palettized texture where the user specifies one value from the color palette for both the high and low range. Texture map chroma keying is used with RGB(A) textures where the user specifies a 16 bit value which represents the transparent color (usually black) for both the hi and low range.

Figure 2-15. A Color Keyed Splash



Example 2-21. Enabling Texture Map Chroma Keying and Color Keying with DirectX

To enable chroma/color keying with DirectX, the user fills in a D3DCOLORKEY structure with either a value range for chroma keying or a single palette value for color keying as follows:

```
typedef struct D3DCOLORKEY{
    DWORD   dwColorSpaceLowValue;
    DWORD   dwColorSpaceHighValue;
} DDCOLORKEY;
DDCOLORKEY ColorKeyInfo;
// for non-palettized textures
ColorKeyInfo.dwColorSpaceLowValue = 0x0000;
ColorKeyInfo.dwColorSpaceHighValue = 0x0000;
// for palettized textures
ColorKeyInfo.dwColorSpaceLowValue = 0;
ColorKeyInfo.dwColorSpaceHighValue = 0;
lpTextureSurface->SetColorKey(<dwFlags>, &ColorKeyInfo);
```

Where the <dwFlags> are either, DDCKEY_DESTBLT, DDCKEY_DESTOVERLAY, or DDCKEY_SRCBLT. The SetColorKey function takes as its first parameter a DWORD flag which can specify whether the color key is for a source blit, a destination blit, or a destination overlay.

To enable the color/chroma keying, the user needs to set the appropriate render state:

```
SetRenderState(D3DRENDERSTATE_COLORKEYENABLE, TRUE);
```

To actually see color keying, use one of the DirectX Blt functions as shown:

```
lpBackBuffer->BltFast(Xpos, Ypos, lpOffscreenSurface, &Rectangle,
    DBLTFAST_SRCOLORKEY);
```

2.2.5.4 Texture Wrapping Formats

Applications can specify different texture-wrapping formats for either or both of the U and V directions.

Example 2-22. Enabling Texture Wrapping with DirectX

The Intel740 supports the following DirectX texture wrap formats:

Table 2-5. Supported DirectX Texture Wrap Formats

Texture Wrap U	Texture Wrap V	D3DTEXTUREADDRESS
Wrap	Wrap	D3DTEXTUREADDRESS_WRAP
Mirror	Mirror	D3DTEXTUREADDRESS_MIRROR
Clamp	Clamp	D3DTEXTUREADDRESS_CLAMP

WRAP

The wrap mode creates an effect in which the texture map looks like it is repeated over and over in the selected region. In wrap mode, textures appear to be tiled. If either WRAPU or WRAPV is set, the texture is an infinite cylinder with a circumference of 1.0. Texture coordinates greater than 1.0 are valid only in the dimension that is not wrapped.

MIRROR

The mirror mode creates an effect where the texture map looks flipped or “mirrored.” It is equivalent to the wrap mode’s “tiling” effect except that the texture is flipped at every integer junction. For instance, between 0 and 1 the texture is normal, then between 1 and 2 the texture is flipped, and between 2 and 3 it is normal, then between 3 and 4 it is flipped, etc.

CLAMP

In clamp mode, the texture coordinates greater than or equal to 1.0 are set to (impasses - 1)/mapsize, and values less than 0.0 are set to 0.0.

In DirectX, the default texture wrap format is D3DTEXTUREADDRESS_WRAP. To change the texture map format with DirectX API, first set the appropriate texture address type:

```
SetRenderState(D3DRENDERSTATE_TEXTUREADDRESS, <D3DTEXTUREADDRESS>);
```

Where the D3DTEXTUREADDRESS is either D3DTEXTUREADDRESS_WRAP, D3DTEXTUREADDRESS_MIRROR, or D3DTEXTUREADDRESS_CLAMP.

Then enable texture wrapping in either the U or V direction by setting the following:

```
SetRenderState(D3DRENDERSTATE_WRAPU, TRUE);
SetRenderState(D3DRENDERSTATE_WRAPV, TRUE);
```

Example 2-23. Enabling Texture Wrapping with OpenGL

The Intel740 supports the following OpenGL texture wrap formats:

Table 2-6. Supported OpenGL Texture Wrap Formats

GL_TEXTURE_WRAP_S	GL_TEXTURE_WRAP_T	VALUE
Clamp	Clamp	GL_CLAMP
Repeat	Repeat	GL_REPEAT

In OpenGL, the texture wrap methods are defined as follows:

CLAMP

Any values greater than 1.0 are set to 1.0, and any values less than 0.0 are set to 0.0. Clamping is useful for applications where you want a single copy of the texture to appear on a large surface. If the surface-texture coordinates range from 0.0 to 10.0 in both directions, one copy of the texture appears in the lower corner of the surface.

REPEAT

Any values outside the range of [0,1] will be repeated in the texture map. With repeating textures, if you have a large texture surface with coordinates from 0.0 to 10.0 in both directions, then 100 copies of the texture will be tiled on the screen.

To enable a texture mapping method, the following calls should be made:

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, <WRAP_MODE>);  
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, <WRAP_MODE>);
```

Where <WRAP_MODE> is either GL_CLAMP or GL_REPEAT.

2.2.5.5 Texture Map Filtering

Texture map filtering enables the user to choose the method the hardware uses to calculate the output pixel color as it corresponds to the texture's texel color at the mapped location. Factors which determine the user's screen pixel color include the distance of the object from the viewer and the size of the texture map in relation to the size of the object. In some applications where texture filtering is not used, a close up object can cause a texture to look blocky because each texel is repeated over a square range of pixels.

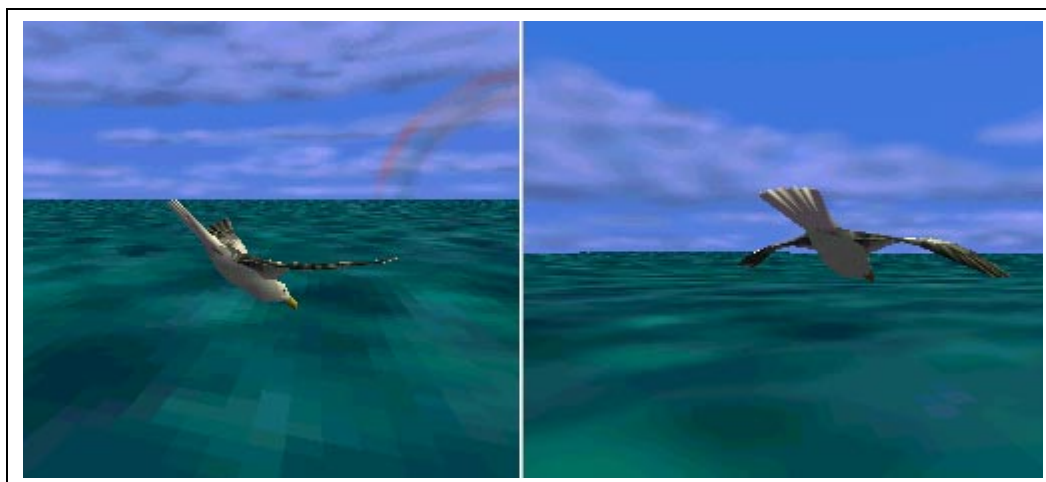
The Intel740 supports the following texture filtering modes for both DirectX and OpenGL: Nearest, Linear, Mip Nearest, Mip Linear, Linear Mip Nearest and Linear Mip Linear. The Mip modes will be discussed in the Texture Mipmapping section.

NEAREST

The nearest texture filtering mode is also known as "point filtering." In this mode, the texel with coordinates nearest to the desired pixel value are used. The output can result in blocky textures as the object becomes larger to the viewer.

LINEAR

The linear texture filtering mode is also known as "bilinear filtering." In this mode, a weighted average of a 2-by-2 area of texels surrounding the desired pixel is used. The output results in a smoother representation of the texture without blockiness.

Figure 2-16. Point Filtering VS. Bilinear Filtering**Example 2-24. Enabling Texture Map Filtering with DirectX**

To enable texture filtering with DirectX, there are two cases which must be addressed. First is when the texture map is minified because the texel is smaller than one pixel. The second case is when the texture map is magnified and a texel is larger than one pixel. To enable texture filtering with DirectX, the following render states must be set:

```
SetRenderState(D3DRENDERSTATE_TEXTUREMIN, <D3DTEXTUREFILTER>);
SetRenderState(D3DRENDERSTATE_TEXTUREMAG, <D3DTEXTUREFILTER>);
```

Where the D3DTEXTUREFILTER can be set to either D3DFILTER_NEAREST, D3DFILTER_LINEAR, D3DFILTER_MIPNEAREST, D3DFILTER_MIPLINEAR, D3DFILTER_LINEARMIPNEAREST, or D3DFILTER_LINEARMIPLINEAR.

Example 2-25. Enabling Texture Map Filtering with OpenGL

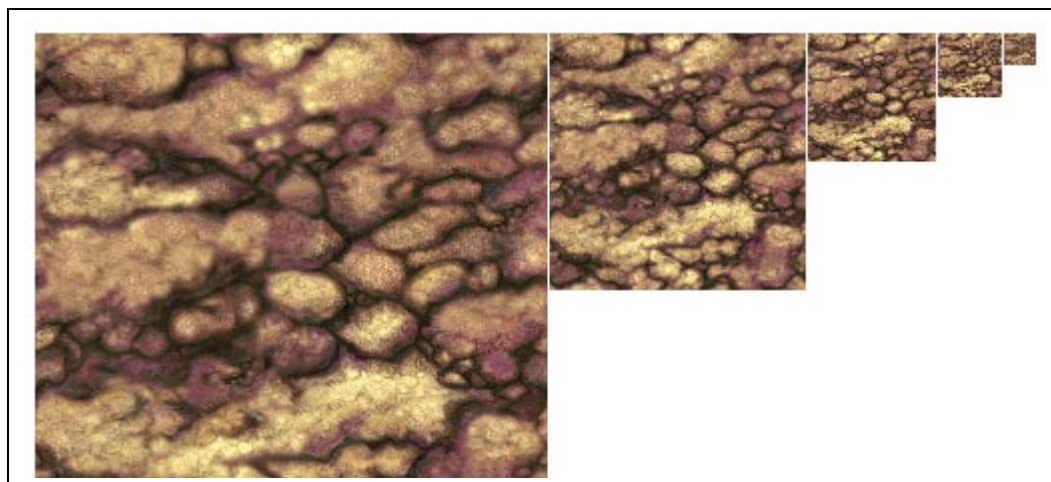
To enable texture filtering with OpenGL, the following calls are made:

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, <FILTER_MODE>);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, <FILTER_MODE>);
```

Where FILTER_MODE is either GL_NEAREST, GL_LINEAR, GL_MIPMAP_NEAREST, GL_MIPMAP_LINEAR, GL_LINEAR_MIPMAP_NEAREST, or GL_LINEAR_MIPMAP_LINEAR.

2.2.5.6 Texture Mipmapping

Because textured objects can be viewed at different distances from the viewer in 3D space, it is possible for a texture object to become smaller than the texture image. This occurrence will cause the texture map to be under-sampled during rasterization. As a result, the texture mapping may display artifacts or “noise.” The purpose of trilinear interpolating and mipmapping is to minimize this effect. With mipmapping, software provides a series of pre-filtered texture maps of decreasing resolutions, called “mipmaps” and stores them in memory. When a 3D object is larger because of its close proximity to the viewer, a corresponding texture map is used. As the object moves farther away from the viewer, Intel740 determines which mipmap to use and switches to a smaller texture size.

Figure 2-17. An Example of Five Levels of Mipmapped Texture

Intel740 supports 11 mipmaps ranging from 1024 x 1024 down to a 1 X 1 texel map. Each successive level has 1/2 the resolution of the previous level in the U and V directions until a 1x1 texture is reached. Both dimensions of the mipmap must be a power of 2 although they do not have to be square. Two forms of mipmap texture filtering can be selected in either DirectX or OpenGL they are:

MIP NEAREST

Similar to the texture filtering Nearest form except that Mip Nearest uses the appropriate mipmap for texel selection.

MIP LINEAR

Similar to the texture filtering Linear form except that Mip Linear uses the appropriate mipmap for texel selection.

LINEAR MIP NEAREST

The two closest mipmap levels are chosen and then a linear blend is used between point filtered samples of each level.

LINEAR MIP LINEAR

The two closest mipmap levels are chosen and then combined using a bilinear filter.

Example 2-26. Mipmap Enabling with DirectX

To enable texture mipmapping using DirectX, a mipmapped surface must first be created and loaded with the appropriate texture maps. To do this with DirectX Immediate Mode, specify that the surface is a TEXTURE surface and also a MIPMAP surface. The user can specify the mipmap count, but this is not necessary. When the “CreateSurface” call is made, DirectX generates all the levels on its own, down to 1x1.

Start by creating the mipmap surfaces:

```
HRESULT          ddrres;
DDSURFACEDESC   ddsd;
LPDIRECTDRAW3   lpDDMipMap;
ZeroMemory(&ddsd, sizeof(ddsd));
ddsd.dwSize = sizeof(ddsd);
ddsd.dwFlags = DDSURFACEDESC_DDSURFACE_TEXTURE | DDSURFACEDESC_DDSURFACE_MIPMAP;
ddsd.ddsCaps.dwCaps = DDSCAPS_TEXTURE | DDSCAPS_MIPMAP | DDSCAPS_COMPLEX
                    | DDSCAPS_VIDEOMEMORY | DDSCAPS_NONLOCALVIDMEM;
ddsd.dwWidth = 256;
ddsd.dwHeight = 256;
```

Then call the CreateSurface function to build the mipmap chain of surfaces:

```
ddres = lpDD->CreateSurface(&ddsd, &lpDDMipMap);
```

Now five subsequent mipmapped surfaces have been created. The next step is to load an image onto each surface. This can be done by traversing the surfaces with the DirectX GetAttachedSurface call and then copying a bitmap which has already been loaded to the current mipmap level surface using the DDCopyBitmap function. See the DirectX SDK manuals and on-line help for more in-depth information.

Finally, enable the mipmap filtering mode by setting the following render state in DirectX:

```
SetRenderState(D3DRENDERSTATE_TEXTUREMIN, <D3DTEXTUREFILTER>);
SetRenderState(D3DRENDERSTATE_TEXTUREMAG, <D3DTEXTUREFILTER>);
```

Where D3DTEXTUREFILTER is D3DFILTER_MIPNEAREST, D3DFILTER_MIPLINEAR, D3DFILTER_LINEAR_MIPNEAREST, or D3DFILTER_LINEAR_MIPLINEAR.

Example 2-27. Enabling Mipmapping with OpenGL

OpenGL has a function which generates all the mipmaps from the dimensions of the mipmap specified down to 1x1. The dimensions of the mipmap can be any power of 2. The following call is used:

```
gluBuild2DMipmaps(GLenum target, GLint components, GLint width, GLint height,
                  GLenum format, GLenum type, void *data);
```

This function is like the glTexImage2D() which creates a texture map surface as mentioned in the section above.

Then, enable either the mip-nearest or mip-linear filtering mode with the following function call:

```
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, <FILTER_MODE>);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, <FILTER_MODE>);
```

Where FILTER_MODE is GL_MIPMAP_NEAREST or GL_MIPMAP_LINEAR, GL_LINEAR_MIPMAP_NEAREST, or GL_LINEAR_MIPMAP_LINEAR.

2.2.6 Drawing Formats

The Intel740 supports the following Drawing Formats:

- Solid The output to the screen is a triangle, either solid color or patterned by a texture map.
- Wire-frame The output to the screen is a line drawing, either solid color or patterned by a texture map.

Example 2-28. Enabling Drawing Formats with DirectX

To enable drawing formats with DirectX, the following render state call is used:

```
SetRenderState(D3DRENDERSTATE_FILLMODE, <D3DFILLMODE>);
```

Where D3DFILLMODE is either D3DFILL_WIREFRAME or D3DFILL_SOLID.

Example 2-29. Enabling Drawing Formats with OpenGL

To enable drawing formats with OpenGL, the following call is made:

```
glPolygonMode(<FACE>, <MODE>);
```

where FACE is GL_FRONT_AND_BACK, GL_FRONT or GL_BACK and MODE is either GL_LINE, or GL_FILL.

2.2.7 Buffers

The Intel740 supports many buffer types including:

- A back buffer, which can be placed in either local video memory or AGP memory
- A front buffer, which should be placed in local video memory
- A Z-buffer, which must be placed in local video memory

The Intel740 also supports two back buffer surfaces needed for triple buffering.

In OpenGL, the buffers are created by selecting the proper pixel format. The pixel formats and the corresponding buffers they create are as follows:

Table 2-7. Pixel Formats and Buffers

Back Buffer	Depth (Z) Buffer
NO	NO
NO	YES
YES	NO
YES	YES

When creating buffers with the DirectX API, the user uses the “CreateSurface” call and sets appropriate DDSD flags and capabilities.

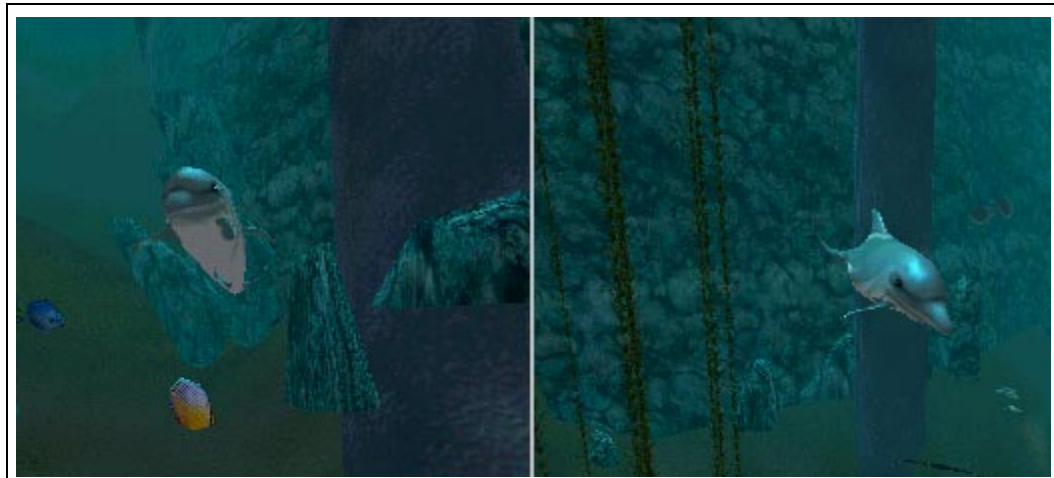
2.2.7.1 Double and Triple Buffering

Intel740 permits the use of both double and triple buffering, where one buffer is the primary buffer used for display and one or two are the back buffer(s) used for rendering. With double buffering, an application typically constructs a scene in the back buffer while the front buffer is being displayed. With triple buffering, a flipping chain of buffers is used which gives added buffering between drawing to the back buffer and rendering which can help increase performance. For double buffering, when the scene in the back buffer is complete and it is time to display, the application flips the two buffers or rather, switches the roles of the two buffers so that the drawn-to buffer becomes the rendering buffer and vice versa. In the case of triple buffering, when flipping of the buffers is performed, the application makes the second to last drawn-to buffer the rendering (primary) buffer and draws to the last buffer used for rendering.

2.2.7.2 Z-Buffering

The Z-buffer contains 16 bits of depth information for each pixel in the display buffer. The use of the Z-buffer is optional. Figure 2-18 below shows the use of the Z-buffer.

Figure 2-18. Z-Buffering Off vs. Z-Buffering On



When enabled, the Z-buffer function performs a depth compare between the pixel Z (known as source Z or ZS) and the Z value read from the Z-buffer at the current pixel address (known as destination Z or ZD). If the test is not enabled, it is assumed the Z test always passes. The Z value is only written to the Z-buffer when the results of the Z test are true. It is always necessary to clear the Z-buffer before each new frame is drawn.

The Intel740 uses a logarithmic method for Z-buffering. The logarithmic approach makes those objects closer to the viewer look better than does the linear approach.

Example 2-30. Enabling Z-Buffering with DirectX

To Create a Z-buffer with DirectX the following surface must be created:

```
DDSURFACEDESC ddsd;
IDIRECTDRAW*lpdd;
IDIRECTDRAW*lpZSurface;
HRESULT ddrval;
ddsd.dwSize = sizeof(ddsd);
ddsd.dwHeight = window_height;
ddsd.dwWidth = window_width;
ddsd.dwZBufferBitDepth = 16;
ddsd.wFlags = DDSD_CAPS | DDSD_HEIGHT | DDSD_WIDTH | DDSD_ZBUFFERBITDEPTH;
ddsd.ddsCaps = DDSCAPS_ZBUFFER | DDSCAPS_VIDEMEMORY | DDSCAPS_LOCALVIDMEM;
ddrval = lpdd->CreateSurface(&ddsd, &lpZSurface, NULL);
```

To enable Z-buffering with DirectX, the following render states must be set:

```
SetRenderState(D3DRENDERSTATE_ZENABLE, TRUE);
SetRenderState(D3DRENDERSTATE_ZWRITEENABLE, TRUE);
SetRenderState(D3DRENDERSTATE_ZFUNC, <D3DCMPFUNC>);
```

D3DCMPFUNC is D3DCMP_NEVER, D3DCMP_LESS, D3DCMP_EQUAL, D3DCMP_GREATEREQUAL, D3DCMP_LESSEQUAL, D3DCMP_GREATER, D3DCMP_NOTEQUAL, or D3DCMP_ALWAYS.

The application also must clear the Z-Buffer using the following DirectX function call:

```
lpZSurface->Blt(lpDestRect, lpDDSrcSurface, lpSrcRect, DDBLT_DEPTHFILL, dwFillDepth);
```

Example 2-31. Enabling Z-Buffering with OpenGL

In order to enable Z-Buffering with OpenGL, the following code is used:

```
glEnable(GL_DEPTH_TEST);
glDepthFunc(<FUNCTION>);
```

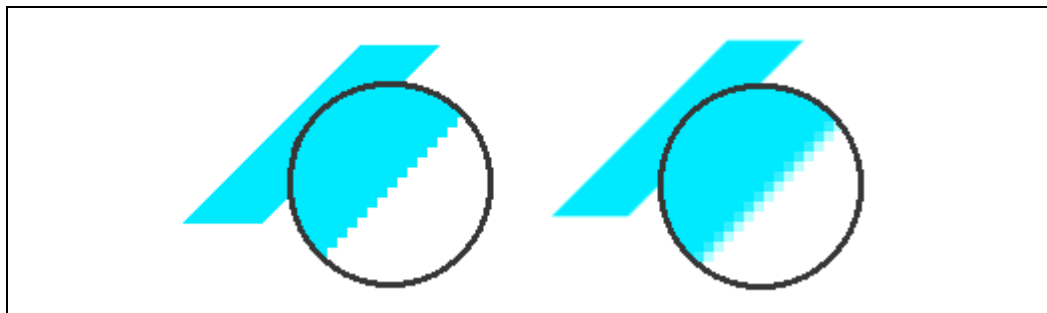
FUNCTION is GL_NEVER, GL_ALWAYS, GL_LESS, GL_LEQUAL, GL_EQUAL, GL_GEQUAL, GL_GREATER, or GL_NOTEQUAL.

2.2.8 Antialiasing

Aliasing is one of the artifacts that degrades image quality. In its simplest manifestation, aliasing causes the jagged staircase effects on sloped lines and polygon edges. More subtle effects are observed in animation, where very small primitives blink in and out of view.

One of the possible solutions to the aliasing problem, area sampling, treats primitives (e.g., points and lines) as primitives with an area, rather than as zero-dimensional geometric entities. This method permits rasterizers to compute the fraction of a pixel that a primitive covers and blend the area-weighted color or the pixel with the color in the frame buffer.

For antialiasing to work properly on the Intel740, polygons (triangles) must be sorted from back to front. Antialiasing can be used with Z-buffering.

Figure 2-19. Effects of Antialiasing**Example 2-32. Enabling Antialiasing with DirectX**

To enable antialiasing with DirectX, the following render state is enabled:

```
SetRenderState(D3DRENDERSTATE_ANTIALIAS, SORTDEPENDENT);
```

When using execute buffers, an edge flag can be set to enable edge antialiasing.

Example 2-33. Enabling Antialiasing with OpenGL

To enable antialiasing with OpenGL, the user must first enable alpha blending and set the source and destination blends properly:

```
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glEnable(GL_POLYGON_SMOOTH);
```

2.2.9 Back Face Culling

One of the stages in the 3D Pipeline which can be performed in either the software geometry stage or in the hardware rendering stage is that of back face culling which consists of the removal of surfaces of 3D objects which cannot be seen from the user's viewpoint. The Intel740 supports back face culling. Because every surface has a surface normal which is a vector perpendicular to its surface, the normals of each surface can be tested to see if they point backwards away from the viewer. Back face culling saves processing time since culled surfaces will not need to be rendered. When using color alpha blending, be sure to disable back face culling because alpha blending looks better when the back facing polygons are also rendered and are visible through the translucent alpha blended portions.

Example 2-34. Enabling Back Face Culling with DirectX

To enable back face culling with DirectX, the following renderstate is set:

```
SetRenderState(D3DRENDERSTATE_CULLMODE, <MODE>);
```

MODE is D3DCULL_CCW for counter clockwise culling, or D3DCULL_CW for clockwise culling.

Example 2-35. Enabling Back Face Culling with OpenGL

To enable back face culling with OpenGL:

```
glEnable(GL_CULL_FACE);
glCullFace(<MODE>);
```

MODE is GL_FRONT, GL_BACK, or GL_FRONT_AND_BACK.

2.3 2D Capabilities

In this section the 2D capabilities of the Intel740 are discussed:

- “BitBLT Engine” (below)
- “Stretch BLT Engine” (below)
- “Color Expansion” (below)
- “Hardware Cursor” on page 2-42
- “Video Display Resolutions” on page 2-42

2.3.1 BitBLT Engine

The term BitBLT refers to block transfers of pixel data between memory locations. Intel740’s high performance 64-bit BitBLT engine provides hardware acceleration for many common Windows operations. To facilitate these, there are two primary BitBLT functions in the Intel740: regular BitBLT and stretch BitBLT. Regular BitBLT involves transferring blocks of data from one memory location to another. The capability of performing raster operations on the data using a pattern is also included. Stretch BitBLT can stretch source data in the X and Y directions to a destination larger or smaller than the source. Stretch BitBLT functionality expands a region of memory into a larger or smaller region using replication and interpolation.

If required, Intel740 will expand monochrome data into color data. This new data will be destination aligned. The main feature of the BitBLT is to take a stored pattern and expand it to the destination color space while destination aligning it. Intel740’s raster opcode engine supports all 256 Microsoft-defined raster operations (ROPs) including transparent BitBLT.

2.3.2 Stretch BLT Engine

The Stretch BLT Engine allows a source memory region to be blitted to a destination region which is larger, smaller or the same size as the source region by replacing or removing pixels. Expansion and shrinking can occur in both the horizontal and vertical directions.

An additional feature of the Stretch BLT Engine is the ability to transparently place the source data over some destination data by masking. This is useful for sprites in 3D games.

2.3.3 Color Expansion

During a BLT operation, source color depth may not be the same as destination color depth. The Intel740 supports monochrome data as well as 8, 16, and 24 bit color data. The BLT engine has the ability to expand source monochrome data into a color depth of 8, 16, or 24. Color expansion can be either opaque or transparent. When opaque, a foreground and background color are both transferred to the destination in the new color depth. When transparent, only the foreground color is specified. This is very useful for text data. Text data can be stored as one bit per pixel color (monochrome), and expanded to the correct color later.

2.3.4 Hardware Cursor

The Intel740 allows a total of 16 cursor patterns to be stored in 4 Kbytes. Six modes are provided for the cursor:

- 32x32 2 bpp 2-plane mode (Solid Color, Inverted Solid Color, Transparent, Inverted)
- 128x128 1 bpp 2-color mode
- 128x128 1 bpp 1-color and transparency mode
- 64x64 2 bpp 3-color and transparency mode
- 64x64 2 bpp 2-plane mode (Solid Color, Inverted Solid Color, Transparent, Inverted)
- 64x64 2 bpp 4-color mode

2.3.5 Video Display Resolutions

The Intel740's video function provides analog output for use with a monitor or a 8/12-bit digital output to interface to a TV output chip. Integrated into the Intel740 is an I²C interface to facilitate this capability. Video synchs and timings are fully programmable. Any overlays are merged with data from the frame buffer during output and can be scaled in the X and Y directions. Gamma correction can be applied on the video output. Resolutions supported for display ranges are shown in Table 2-8. In addition to the standard VGA modes, the Intel740 also supports the following extended modes with the stated memory and refresh timings:

Table 2-8. Display Resolutions

Resolution	Colors/Color Bits per Pixel	Minimum Memory Configuration (64-Bit SDRAM)	Vertical Refresh Rates (MHz)
640x480	256/8-bit palettized	1 Mbyte	60, 75, 85
	64K/16	1 Mbyte	
	16M/24	1 Mbyte	
800x600	16/4	1 Mbyte	56, 60, 75, 85
	256/8-bit palettized	1 Mbyte	
	64K/16	1 Mbyte	
	16M/24	2 Mbyte	
1024x768	256/8-bit palettized	1 Mbyte	43 ⁽¹⁾ , 60, 75, 85
	64K/16	2 Mbyte	
	16M/24	4 Mbyte	
1280x1024	256/8-bit palettized	2 Mbyte	43 ⁽¹⁾ , 60, 75
	64K/16	4 Mbyte	

NOTE:

1. The 3D hardware only rasterizes to 16-bit surface

The video display controller is responsible for the horizontal and vertical timings of the monitor, accessing data from memory, preparing data for display, and presenting the results to the monitor or TV. The Intel740 can convert YUV(4:2:2) to RGB format. An I²C Bus is provided for easier connection to some chips.

The display engine also contains an overlay unit. The overlay (full motion video) unit is capable of converting from YUV4:2:2 format to 24 bpp RGB. Line widths to 720 pixels are supported. X,Y interpolation can be performed on the overlay window if the source is smaller or larger than the destination display size. Intel740 performs filtering/smoothing when interpolating in the horizontal and vertical directions. The data may be scaled in both the horizontal or vertical direction using a six bit expansion value. On output, the data is scaled up. The image is increased in size only. This expansion is smoothed/filtered before being passed to the display.

When stretching is performed, the horizontal filter is 1-1. The vertical interpolation is either deblocking (average on change only) or 1-2-1 running average. Chroma keying is performed so that pixels of a selected color are transparent. (This editing effect is sometimes known as “blue screening.”)

The Intel740 contains a separate hardware cursor for Windows. The cursor information is not stored within the frame buffer but is combined with the screen image immediately before the image is displayed. Functionality built into the cursor allows it to be enabled or disabled. Up to 16 cursor patterns (depending on size) may be stored in separate cursor data space. Cursor modes supported are: 32x32x2bpp and 64x64x2bpp pixel plane modes (for Microsoft Windows), 64x64x2bpp four color mode, 64x64x2bpp three color transparency mode, 128x128x1bpp two color mode, and 128x128x1bpp one color transparency mode.

The combined result from the hardware cursor, overlay, and primary display is performed by the RAMDACs. There are three 8-bit DACs (one for controlling red, one for green, and one for blue). Each DAC has a 256x8 palette RAM is responsible for storing information about the colors to be displayed. The Intel740 is optimized for a 2D output resolution of 1024x768 and a 3D display resolution of 640x480. Within the 2D section, the horizontal sync, vertical sync, and blanking signals are fully programmable.

2.4 Video, VBI, and Intercast Capabilities

The Intel740's Video, VBI and Intercast capabilities are discussed in the following subsections:

- “Video Capture Port” (below)
- “Video Overlay” on page 2-44
- “VBI and Intercast” on page 2-44

2.4.1 Video Capture Port

The PC video interface to Intel740 is a uni-directional digital input port that accepts 16-bit wide data, two synchronizing signals (HREF, VFREF), and a pixel rate clock (VCLK). The video capture port can be configured as a VMI/VAFC interface. Taking the digital video data from this video port, Intel740 can perform video functions such as color space conversion, scaling, zooming, interpolation, and video playback. The data input to this port can be in RGB-15, RGB-16, or YUV 4:2:2 format.

Devices that output an analog signal can be connected to the video capture port through a third party chip that provides digital output. Digital camera video conferencing applications are supported permitting the user to have an unflipped/mirrored view. This port provides support for Intercast technology and POTS (Plain Old Telephone Set) video conferencing. Note that an external third party VBI decoder chip is needed for Intercast technology. For POTS video conferencing, the port interfaces to a camera.

To provide smooth overlay, the captured frame can be assembled before being displayed. The Intel740 also allows incoming video frames to be copied to disk for later playback (via hardware and software). Because software can write to the video capture buffer directly, playback can occur directly from CD-ROM, or disk. During playback, software can use status bits to indicate to the Intel740 when the frame is ready to be displayed.

To facilitate digital camera applications, the Intel740 can perform backward writes. This allows the user to see a mirrored or non-mirrored view on screen.

Gamma correction is also provided. When in 8-bit-per-pixel mode or smaller, the graphics data is expanded by a palette. If analog to digital conversion is needed, an external chip creates the digital signal sent to the Intel740.

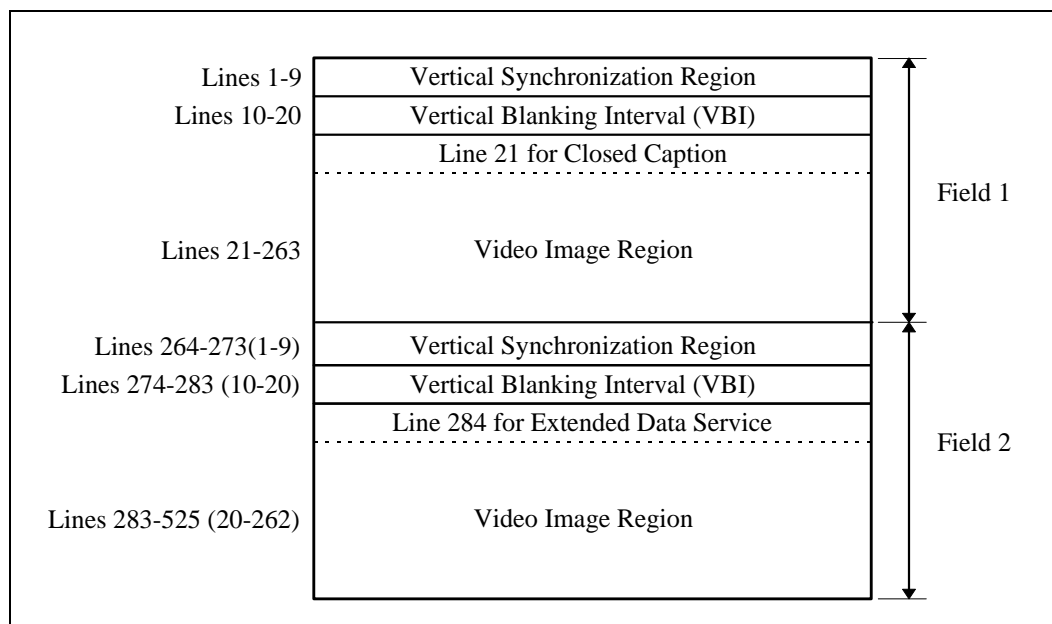
2.4.2 Video Overlay

The maximum line width of the overlay is 720 pixels wide. However, the overlay engine can enlarge a picture using a 6-bit expansion value. The data in the overlay can use a 1-2-1 FIR filter in the horizontal direction and either line replication, a smoothing at line boundaries, or a continuous running average filter in the vertical direction. Using the overlay, the data in the capture buffer can be displayed while capture is occurring. Overlay data does not necessarily have to originate from the capture port. The data can come from AGP memory. Unlike the capture port, the overlay engine can support 4-bit indexed, 8-bit indexed, 15-bit RGB, 16-bit RGB, and 24-bit RGB color formats. The capability to do color keying is included in the overlay.

2.4.3 VBI and Intercast

Television has two main formats: NTSC and PAL/SECAM. The NTSC signal is composed of two fields made up of 720 pixels and 525 lines every 1/30 of a second. PAL/SECAM has two fields of 720 pixels and 625 lines every 1/25 of a second. Lines 1-22 of the NTSC signal are used for VBI data and the other lines 23 - 525 are used for the actual video data. The Intel740 has the ability to capture the NTSC data and place it in a buffer where the video data can be displayed and the VBI data can be sent to the ISV's Intercast software to be interpreted. An ISV can then create an application which displays both the video and the VBI data which is sent by some television cable channels as HTML files which complement the currently viewed program. One example is the cable TV music video channel, MTV 2, which sends HTML files in the VBI stream showing poetry, photos, and other information along with the currently broadcasting artist which can be viewed in an HTML browser side by side with the video.

Figure 2-20. Content of an NTSC Video Frame



2.4.3.1 VBI Data Format

VBI enabled video decoders, like Brooktree Bt829, are capable of passing through the ancillary data in the VBI region for later processing by software. Specifically, the raw digitized VBI data stream bypasses the decimation filter, Y/C separation filters, and the interpolation filter of the video decoder, and are sent to the Intel740 video capture port along with the rest of the active video lines.

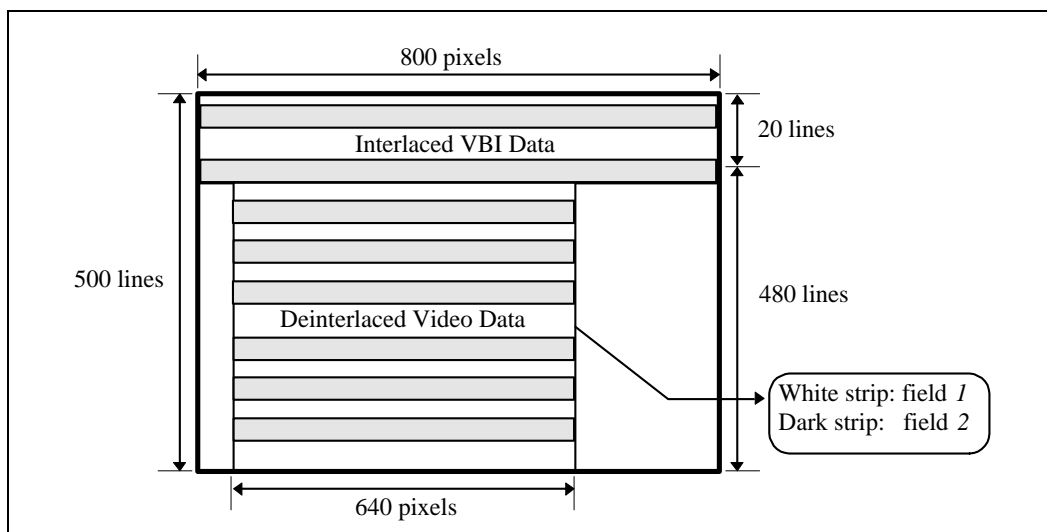
The Intel740 video capture engine views the incoming VBI and video data lines as the same for each frame, fetching the same amount of pixels per line and storing to the same local memory locations. As shown in Figure 2-21, one complete video frame captured in the local memory starts with about 10 lines of full VBI data (say 800 pixels), followed by active video lines with smaller amounts of valid video pixels (640 for full resolution and 320 for half resolution).

Note: Since DirectDraw does not allow the width of DD surface in secondary buffer bigger than primary buffer, the graphics resolution has to be at least 800x600. Also the video decoder has to be able to deliver 800-pixel or less wide VBI lines.

Since the VBI data stream is not pipeline-delayed to match the YUV4:2:2 video output data with respect to horizontal timing (HSYNC signal), it arrives earlier than valid video data related to the HSYNC signal. Therefore, the leading pixels in active video lines are invalid. The tail pixels are also invalid, since VBI lines are wider than video lines. The boundary of valid video data can be calculated from the scaling factors of the video decoder. The memory pointer for the active video frame can be calculated from the number of VBI lines and the video capture span.

The video capture VSYNC is field based (60 per second for NTSC). The VBI Capture VxD maintains a counter to indicate the field number. VBI Capture VxD handles the VBI data only for odd field VSYNC.

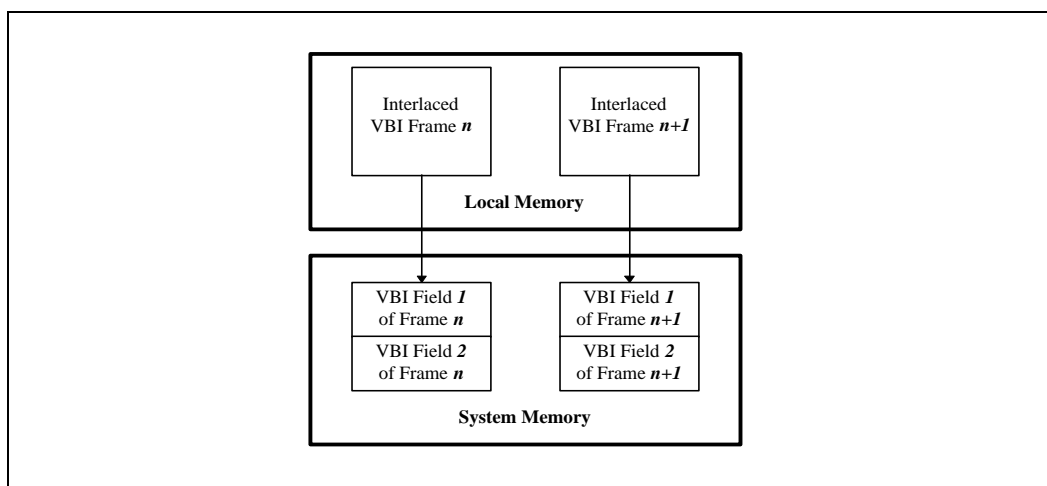
Figure 2-21. Configuration of Video Capture Memory with VBI



The Intel740 VBI Capture VxD is responsible for capturing raw VBI data into system memory (regular cacheable system memory). Allocation of system memory is the responsibility of The Intel740 VBI Capture VxD. The format of raw VBI data in system memory is per InterCast specification. InterCast VBI decoder requires VBI data to be organized into fields (each field corresponds to one and only one television field) in system memory. However, the Intel740 Video Capture Engine is frame-based (i.e., the odd and even fields of a video frame, including VBI data, are automatically de-interlaced before storing in local memory). To avoid data overrun, hardware double buffering is activated. Four VBI field buffers are allocated in system memory by the Intel740 VBI Capture VxD. Two for frame n and another two for frame $n+1$. As shown in Figure 2-22, there are two fields of VBI data ready for decoding for frame n , and two for frame $n+1$. Note that the two fields of a frame can be allocated together as long as the correct pointer is sent to the VBI decoder.

Alternatively, the interlaced VBI frame data can also be directly copied to system memory without reordering. In this case, a correct pointer and a correct line span (twice the input line span) needs to be sent to the VBI decoder.

Figure 2-22. VBI Buffering Scheme



2.4.3.2 VBI Data Flow

The Intel740 VBI Capture VxD moves raw VBI data from local memory to system memory. No format conversion or data processing is done. There are about 34 Kbytes of VBI data per frame second (~800 pixels/line * 2 bytes/pixel * ~22 lines/frame) and the total VBI data bandwidth is about 1 Mbyte (30 frames/second).

2.4.3.3 CC and EDS

According to the EIA-608 standard, two bytes of information are presented on line 21 (field 1) for Closed Captioning (CC) and an additional two bytes are presented on line 284 (field 2) for Extended Data Services (EDS). The Intel740 VBI Capture VxD delivers raw data for CC/EDS lines as other VBI lines to the Intercast VBI decoder VxD.

2.4.3.4 Direct CC and EDS Capture

Some video decoders (Brooktree Bt829, Philips 7111A) can provide decoded CC/EDS data through the I²C interface. When valid data is detected, the video decoder stores the data in an I²C data register. An I²C status register provides data valid and data ready flags.

An application can access CC/EDS data from the video decoder through asynchronous reading of the I²C bus. The application polls the Close Caption Valid bit in the video decoder's I²C status register. The polling frequency has to be slightly higher than the field frequency of 60 Hz for NTSC. Since there are only two bytes of CC or EDS data per frame and a video decoder can provide a FIFO for storing the bytes, there is typically sufficient time to fetch this data.

The direct CC/EDS capture is convenient for TV viewing applications that do not require full VBI decoding. The low data-rate CC/EDS data needs to be delivered to system memory.

2.5 DVD Capabilities

DVD (Digital Video Disk) discs are the media for next generation laserdisc players. DVD format specifies MPEG-2 standard for video compression, and AC-3 (NTSC) or MPEG (PAL) for audio compression. MPEG-2 uses the YUV4:2:0 format, also known as YUV12; U and V are subsampled by 2 in both horizontal and vertical directions. A YUV12 macroblock consists of a 16x16 block of Y data (256 bytes) and two 8x8 blocks (2x64 bytes) for the U and V data. Each macroblock stores 256 pixels in 384 bytes, or 12 bits/pixel. Backward compatible with the CD, DVD allows a single side storage of 8.5 Gbytes. DVD is used to play movies, interactive games, and training on the PC using Dolby* AC-3 (3D sound) surround sound audio. DVD uses MPEG-2 encoding and special encryption. Once the encryption and MPEG-2 are decoded, the output can be sent directly to the Intel740 using the YUV 4:2:2 encoding.

2.5.1 Hardware DVD/MPEG-2 Movie Playback

A DVD decoder chip located on the PCI bus receives the raw DVD stream (including compressed video and audio data) and decompresses the data. The decompressed audio stream is sent to the audio subsystem through a dedicated digital audio port. The decoded video stream is sent from the DVD decoder chip to the Intel740 through the video capture port. The incoming video stream to the Intel740 is in YUV2 format with resolution of 720x480 at 30 frames per second following the CCIR601 standard. The Intel740 drivers, DirectDraw HAL and DDVPE HAL, handle video display for DVD.

2.6 TV Out Interface

The Intel740 has a digital TV out interface multiplexed with the BIOS address lines. When using the TV out interface, normal VGA display cannot be used. The 12-bit digital interface is designed to interface with an external TV encoder, which incorporates a high quality flicker filter and performs overscan compensation.

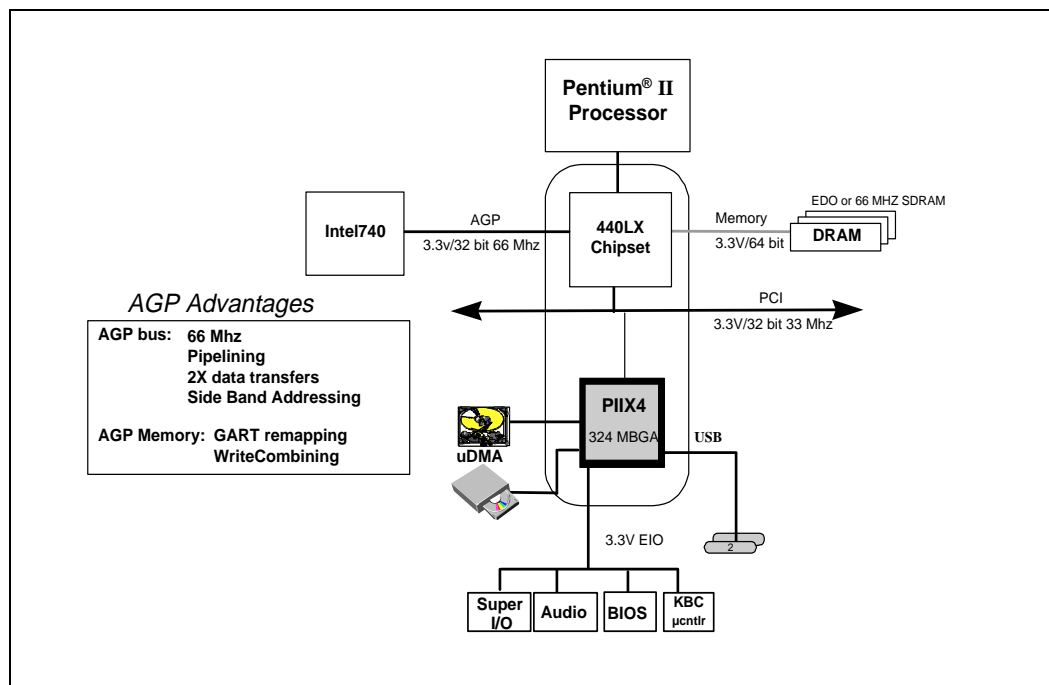
2.7 AGP Interface

The Intel740 is AGP 1.0 and PCI 2.1 compliant. Optimized for AGP, the Intel740 runs effectively at 133 MHz on a 32-bit bus, allowing 533 Mbyte/s peak data throughput. The Intel740 supports AGP sideband extensions, permitting demultiplexed address/data transfers. The Intel740's AGP interface allows memory reads to equal the throughput of memory writes. The use of this graphics port overcomes the read latency of PCI by making reads zero wait state, just like writes. Because of this dedicated high bandwidth port, the Intel740 is able to use system memory for graphics purposes more effectively than PCI.

2.7.1 AGP Primer

The Accelerated Graphics Port (AGP) brings new levels of performance and realism to next-generation 3D graphics accelerators. The principal benefit comes from the graphics accelerator having high speed access to surface textures and other graphics surfaces in main system memory. Special performance oriented AGP features allow much faster read/write access to these surfaces than has been possible in the past. The basic memory architecture of an AGP system is illustrated in Figure 2-23 below.

Figure 2-23. Intel740 Connects to System Memory Over AGP



Graphics software infrastructure requires that AGP memory be contiguous, which means a page based system memory must have a graphics address remapping table (GART) capability. This is because the operating system ordinarily allocates randomly located pages of memory whereas graphics software requires its memory to be contiguous.

The translation facility gives each memory page a second aliased address. All the addresses are adjacent, making this part of system memory closely resemble conventional video memory. Memory accessible through the GART is referred to as non-local video memory, meaning video memory that is not local to the Intel740.

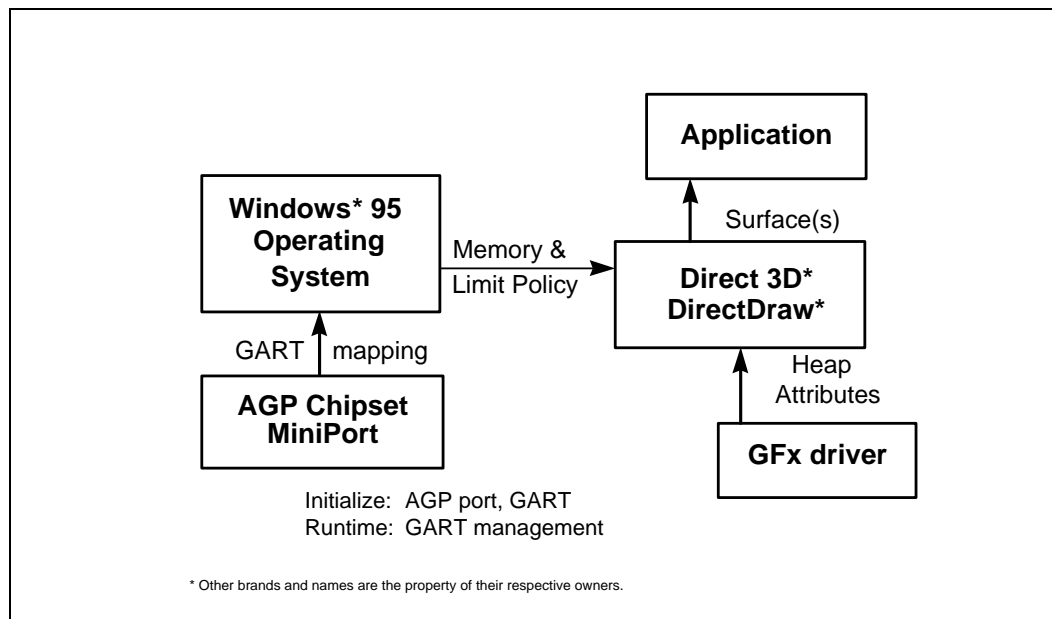
Non-local memory can be accessed by the host processor, by the Intel740, and in current AGP systems by other PCI devices. In future systems, the GART translation will only be used by AGP graphics devices and the host processor will perform a corresponding address translation.

2.7.2 AGP Software Architecture

DirectDraw applications request space for graphics surfaces by calling the DirectDraw function “CreateSurface.” Space for the surface is obtained from heaps defined by the graphics driver. Memory for non-local memory heaps is obtained from the operating system. When more non-local video memory is needed, DirectDraw can obtain additional memory from the operating system. Memory is locked in place and mapped into the proper GART address range. The surface is aligned and its memory type established as specified in the graphics heap template. Requests to expand AGP memory are honored so long as the total amount of AGP memory does not exceed a limit set by the operating system.

Initialization details are attended to at the time the operating system is loaded. The operating system calls the chipset miniport which initializes AGP port parameters, allocates space for the GART translation table and initializes the GART hardware. The interaction of these functions is summarized below.

Figure 2-24. New Services in Windows Work with DirectDraw to Support AGP Applications



2.8 BIOS Interface

The Intel740 supports a maximum video BIOS size of 256K x 8. Flash can be used.

2.9 Local Memory

The Intel740 uses SDRAM technology and can interface to SGRAM through its 32-bit or 64-bit memory interface. Memory Bus speeds range from 66 MHz, 83 MHz, and 100 MHz while configurations of 1, 2, 4 and 8 Mbytes are supported. Using a 64-bit interface, up to 800 Mbytes/s peak bandwidth is supported. The Intel740 allows operands to be placed in either local video memory or AGP memory. It is recommended that the Z, display, and Render buffers, video capture and MPeg overlay be located in local video memory, however when space becomes limited, the Render buffer should be relocated into AGP memory.

3.1 OpenGL Programming Environment

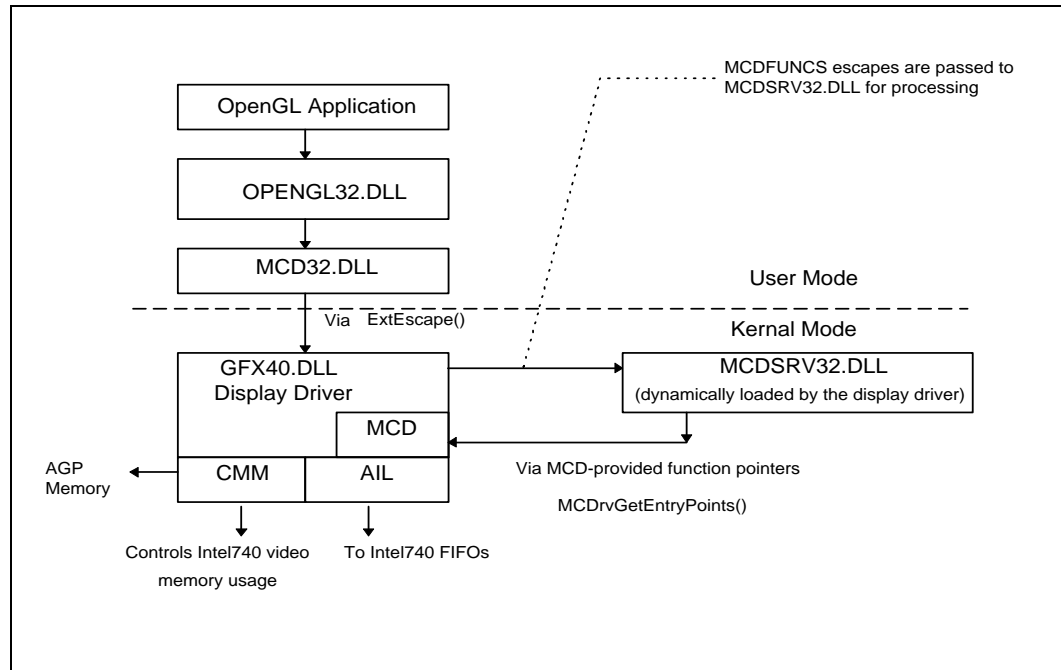
OpenGL is an application programming interface (API) which is used by a software application to interface with the graphics hardware. OpenGL consists of approximately 120 different commands which are used to specify graphical objects and the operations applied to the objects which are required by 3D applications. OpenGL is a streamlined, hardware-independent interface designed to make applications portable from one hardware platform to another. For more information on the OpenGL function commands, see the OpenGL Specification document which can be obtained from the SGI web site at <http://www.sgi.com>. Also see Section 4.3, “OpenGL Performance Guidelines” on page 4-26, for the Intel740-specific OpenGL performance information.

3.1.1 Model

The MCD interfaces with the following external entities: Microsoft Mini-Client Driver Interface, Intel740 2D display driver, WIN32 GDI Escape Mechanism, Microsoft Windows Registry, and the Intel740 Interface Language (AIL). The Configuration Applet, along with any Diagnostic/Test application, will interface with the MCD through the GDI device-dependent escapes defined by the driver. Figure 3-1 demonstrates the MCD architecture context diagram.

Note: In this chapter, the terms *pass* and *punt* have special meaning: “pass” control or data to a lower level on the software chain; “punt” back to a higher level on the software chain.

Figure 3-1. MCD Architecture



3.1.2 Supported OpenGL MCD States

Table 3-1. Supported OpenGL MCD Enables (Sheet 1 of 2)

MCD Enables	Intel740 State(s)	Notes
ALPHA_TEST	ALPHA_TEST_ENABLE	
BLEND	BLEND_ENABLE	
INDEX_LOGIC_OP	n/a	color index not supported
DITHER	DITHER_ENABLE	
DEPTH_TEST	Z_ENABLE	
FOG	FOG_ENABLE	
LIGHTING [†]	n/a	
COLOR_MATERIAL [†]	n/a	
LINE_STIPPLE	n/a	Intel740 hardware acceleration cannot be used for line stipple except in the case where the stipple pattern is solid
LINE_SMOOTH	EDGE_ANTIALIAS LINE_AA_REGION BOUNDING_BOX_EXPANSION	
POINT_SMOOTH	n/a	Antialiased points are punted to software
POLYGON_SMOOTH	EDGE_ANTIALIAS POLY_AA_REGION BOUNDING_BOX_EXPANSION	Antialiased lines wider than three are punted to software
CULL_FACE	CULL_MODE	Also uses the cullFaceMode and frontFace items in the MCDRENDERSTATE structure
POLYGON_STIPPLE	STIPPLE	Intel740 buffer stipple will be used to mimic polygon stipple. Hardware stipple will be disabled when lines and points are rasterized.
SCISSOR_TEST	n/a	Scissor rectangles that are smaller than the window size are punted to software
STENCIL_TEST		MCD will punt unless the stencil test is a trivial operation
TEXTURE_1D	See Notes	textureEnabled Boolean in the MCDRENDERSTATE structure controls whether textures are to be used during rasterization
TEXTURE_2D	See Notes	See Notes above
TEXTURE_GEN_S [†]	n/a	
TEXTURE_GEN_T [†]	n/a	
TEXTURE_GEN_R [†]	n/a	
TEXTURE_GEN_Z [†]	n/a	
NORMALIZE [†]	n/a	
AUTO_NORMAL [†]	n/a	

Table 3-1. Supported OpenGL MCD Enables (Sheet 2 of 2)

MCD Enables	Intel740 State(s)	Notes
POLYGON_OFFSET_POINT	n/a	MCD will calculate the appropriate offset and apply to the vertices before passing to the Intel740 hardware
POLYGON_OFFSET_LINE	n/a	See above
POLYGON_OFFSET_FILL	n/a	See above
COLOR_LOGIC_OP	n/a	MCD will punt the rendering to the software implementation unless the logical operation is GL_COPY

† Not currently used by MCD.

Table 3-2. Supported OpenGL MCD States (Sheet 1 of 2)

MCD State	Intel740 State	Notes
textureEnabled	TEXTURE_ENABLE	
fogColor	FOG_COLOR	
fogIndex	n/a	
fogDensity	n/a	
fogStart	n/a	The fog component of the MCDVERTEX structure takes into account the fogStart and fogEnd components.
fogEnd	n/a	
fogMode		GL_LINEAR fog will be accelerated; GL_EXP and GL_EXP2 exponential fog will not be accelerated if FOG_HINT is set to GL_NICEST.
shadeModel	SHADE_MODE	
pointSize		GFXTLTRIANGLES will be used to simulate GL_POINTS with pointSize greater than one.
lineWidth	LINE_WIDTH	If lineWidth is greater than three, GFXTLTRIANGLES will be used to simulate.
lineStipplePattern	n/a	MCD will punt to software implementation if pattern is not solid
lineStippleRepeat	n/a	See above
cullFaceMode	CULL_MODE	Also depends on frontFace
frontFace	CULL_MODE	Also depends on cullFaceMode
polygonModeFront	FILL_MODE	
polygonModeBack	FILL_MODE	
polygonStipple	GFXPALSTIPPROC_STIPPLE_PATTERN	
zOffsetFactor	n/a	The zOffsetFactor and zOffsetUnits will be used to calculate an offset to apply to the z component of each primitive
ZOffsetUnits	n/a	see zOffsetFactor above
StencilTestFunc	n/a	MCD will punt unless trivial operation

Table 3-2. Supported OpenGL MCD States (Sheet 2 of 2)

MCD State	Intel740 State	Notes
stencilMask	n/a	
stencilRef	n/a	
stencilFail	n/a	
stencilDepthFail	n/a	
stencilDepthPass	n/a	
alphaTestFunc	ALPHA_FUNC	
alphaTestRef	ALPHA_REF	
depthTestFunc	Z_FUNC	
blendSrc	SRC_BLEND	
blendDst	DST_BLEND	
logicOpMode	n/a	
drawBuffer	n/a	
indexMask	n/a	
colorWriteMask	n/a	
depthWriteMask	Z_WRITE_ENABLE	
stencilWriteMask	n/a	
colorClearValue	n/a	used to clear the back and/or front buffers
indexClearValue	n/a	
depthClearValue	n/a	used to clear the depth buffer
stencilClearValue	n/a	
TwoSided	n/a	A multi-pass rendering will be used for two sided polygon rendering
userClipPlanes	n/a	
perspectiveCorrectionHint	n/a	All textures are perspective-corrected
pointSmoothHint		Antialiasing-related
lineSmoothHint		Antialiasing-related
polygonSmoothHint		Antialiasing-related
fogHint		If fogHint is GL_NICEST and the fogMode is GL_EXP or GL_EXP2 the MCD will punt the rendering to the software implementation

3.1.3 Supported OpenGL MCD Primitives

Table 3-3. Supported OpenGL MCD Primitives

MCD Render Primitive	Intel740 Primitive	Notes
GL_POINTS	LINE or TRIANGLE	Points are converted to sub-pixel length lines. Wide points are converted to two triangles.
GL_LINES	LINE	
GL_LINE_STRIP	LINE	Line strips are converted to individual lines
GL_LINE_LOOP	LINE	The MCD Helper library automatically converts line loops to line strips. The MCD will never actually receive line loops.
GL_TRIANGLES	TRIANGLE	
GL_TRIANGLE_STRIP	TRIANGLE	Triangle strips converted to individual triangles
GL_TRIANGLE_FAN	TRIANGLE	Triangle fans converted to individual triangles
GL_QUADS	TRIANGLE	Quads are triangulated into individual triangles
GL_QUAD_STRIP	TRIANGLE	Quad strips are triangulated into individual triangles
GL_POLYGON	TRIANGLE	Polygons are triangulated into individual triangles

3.1.4 Supported OpenGL MCD Texture States

Table 3-4. Supported OpenGL MCD Texture States

MCD Texture State	Intel740 Texture State Supported	Notes
sWrapMode GL_CLAMP GL_REPEAT	WRAP_U CLAMP WRAP	The Intel740 wrap states MIRROR, CLAMP_TRANSPARENT, and WRAP_SHORTEST aren't used by the MCD.
tWrapMode GL_CLAMP GL_REPEAT	WRAP_V CLAMP WRAP	The Intel740 wrap states MIRROR, CLAMP_TRANSPARENT, and WRAP_SHORTEST aren't by the MCD.
minFilter GL_NEAREST GL_LINEAR GL_NEAREST_MIPMAP_NEAREST GL_NEAREST_MIPMAP_LINEAR GL_LINEAR_MIPMAP_NEAREST GL_LINEAR_MIPMAP_LINEAR	TEXTURE_MIN NEAREST GLINEAR MIP_NEAREST MIP_LINEAR LINEAR_MIPMAP_NEAREST LINEAR_MIPMAP_LINEAR	
magFilter GL_NEAREST GL_LINEAR	TEXTURE_MAG NEAREST LINEAR	
borderColor	n/a	Texture border colors are not supported

3.1.5 Supported OpenGL MCD Texture Environment States

Table 3-5. Supported OpenGL MCD Texture Environment States

MCD Texture Environment State	Intel740 Texture State Supported	Notes
texEnvMode GL_REPLACE GL_DECAL GL_MODULATE GL_BLEND	MAP_BLEND DECAL DECAL_ALPHA MODULATE_ALPHA Not Supported (see note)	The Intel740 Texture map blend modes MODULATE, DECAL_MASK, MODULATE_MASK and the other more obscure blend modes aren't used in the MCD. GL_BLEND mode is punted
texEnvColor	n/a (see note)	The texEnvColor is used when the MCD mimics GL_BLEND

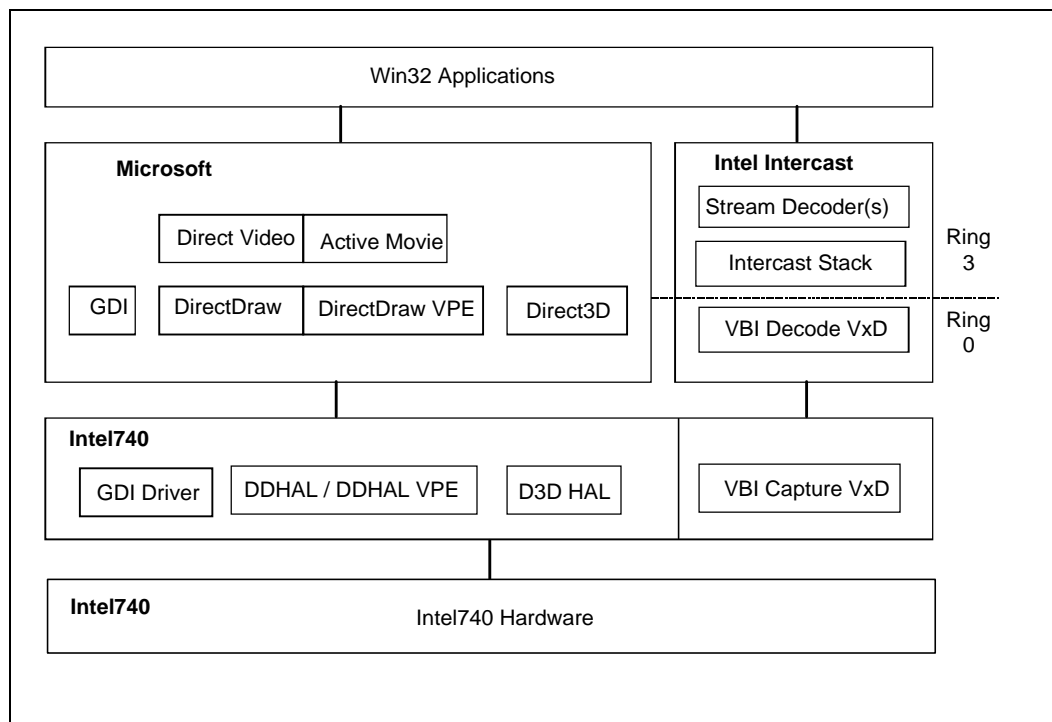
3.2 DirectX Programming Environment

This chapter explains the relationship between the Intel740 API and the Microsoft Windows* support driver environment (Microsoft Windows95*/Windows98*/WindowsNT* 5.0). References are made to existing standards documents. Intel740 extensions or behaviors that differ from the standard are described in detail.

The Intel740 video support drivers include DirectDraw* (Overlay) driver, DirectDraw VPE driver, and VBI Capture VxD. The Intel740 DirectDraw Driver (DDHAL/DDHAL VPE) interfaces with the following external entities: Microsoft DirectX* API, and AGP Memory driver. The Intel740 VBI Capture VxD interfaces with the Intel VBI Decoder VxD, DDHAL VPE driver, AGP Memory driver. Table 3-2 shows the Intel740 driver architecture.

The Intel740 Direct3D device driver interfaces with the following external entities: Microsoft DirectX API, Intel740 2D display driver, WIN32 GDI Escape Mechanism, Windows 95 Registry, and AGP Memory driver. The Configuration Applet along with any Diagnostic/Test applications will interface with the Intel740 Direct3D device driver through the GDI device-dependent graphics escapes defined by the driver. Figure 3-2 shows the Intel740 Direct3D driver architecture.

Figure 3-2. Intel740 Software Architecture



3.3 Mini Interface Drivers

3.3.1 Mini Display Driver

3.3.1.1 Structures Exported to GDI

Table 3-6. Device Technology—dpTechnology (Sheet 1 of 2)

Function	Supported
DT_PLOTTER(0)	
DT_RASDISPLAY(1)	✓
DT_RASPRINTER (2)	
Raster Capabilities—dpRaster	
RC_BITBLT (0001h)	✓ (8 BPP, 16 BPP, 24BPP)
RC_BANDING (0002h)	
RC_SCALING (0004h)	
RC_SAVEBITMAP (0040h)	
RC_PALETTE (0100h)	✓ (8 BPP)
RC_DIBTODEV (0200h)	✓ (8 BPP, 16 BPP, 24BPP)
RC_BIGFONT (0400h)	✓ (8 BPP, 16 BPP, 24BPP)
RC_STRETCHBLT (0800h)	✓ (8 BPP, 16 BPP, 24BPP)
RC_FLOODFILL (1000h)	
RC_STRETCHDIB (2000h)	✓ (8 BPP, 16 BPP, 24BPP)
RC_DEVBITS (8000h)	✓ (8 BPP, 16 BPP, 24BPP)
Level of text support the device driver provides—dpText	
TC_OP_CHARACTER (0001h)	
TC_OP_STROKE (0002h)	
TC_CP_STROKE (0004h)	✓
TC_CR_90 (0008h)	
TC_CR_ANY (0010h)	
TC_SF_X_YINDEP (0020h)	
TC_SA_DOUBLE (0040h)	
TC_SA_INTEGER (0080h)	
TC_SA_CONTIN (0100h)	
TC_EA_DOUBLE (0200h)	
TC_IA_ABLE (0400h)	
TC_UA_ABLE (0800h)	
TC_SO_ABLE (1000h)	
TC_RA_ABLE (2000h)	✓
TC_VA_ABLE (4000h)	

Table 3-6. Device Technology—dpTechnology (Sheet 2 of 2)

Additional raster abilities—dpCaps1	
C1_TRANSPARENT (0001h)	
TC_TT_ABLE (0002h)	
C1_TT_CR_ANY (0004h)	
C1_EMF_COMPLIANT (0008h)	
C1_DIBENGINE (0010h)	✓
C1_GAMMA_RAMP (0020h)	✓
C1_ICM (0040h)	
C1_REINIT_ABLE (0080h)	
C1_GLYPH_INDEX (0100h)	✓
C1_BIT_PACKED (0200h)	
C1_BYTE_PACKED (0400h)	✓
C1_COLORCURSOR (0800h)	✓
C1_CMYK_ABLE (1000h)	
C1_SLOW_CARD (2000h)	
Polyline and line-drawing capabilities—dpLines	
LC_POLYGONSCANLINE (0001h)	✓
LC_POLYLINE (0002h)	✓
LC_WIDE (0010h)	
LC_STYLED (0020h)	✓
LC_WIDESTYLED (0040h)	
LC_INTERIORS (0080h)	
Polygon-, rectangle-, and scan-line drawing capabilities- dpPolygons	
PC_ALTPOLYGON (0001h)	✓
PC_RECTANGLE (0002h)	
PC_WINDPOLYGON (0004h)	
PC_SCANLINE (0008h)	✓
PC_WIDE (0010h)	
PC_STYLED (0020h)	
PC_WIDESTYLED (0040h)	
PC_INTERIORS (0080h)	✓
PC_POLYPOLYGON (0100h)	
PC_PATHS (0200h)	

3.4 DirectDraw Display Driver Interface

This section explains the interfaces of Intel740 2D drivers. It does not cover the whole 2D driver interface, since it is already defined by Microsoft in the Windows95 or Windows98 DDK. This section specifies the interfaces of display driver, mini-VDD, DirectDraw HAL, DirectDraw VPE HAL and version information.

3.4.1 Directdraw Hal Capabilities

Table 3-7. dwCaps—Specifies Driver-Specific Capabilities

Function	Supported
DDCAPS_3D	✓
DDCAPS_ALIGNBOUNDARYDEST	
DDCAPS_ALIGNBOUNDARYSRC	
DDCAPS_ALIGNSIZEDEST	
DDCAPS_ALIGNSIZESRC	
DDCAPS_ALIGNSTRIDE	
DDCAPS_ALPHA	
DDCAPS_BANKSWITCHED	
DDCAPS_BLT	✓
DDCAPS_BLTCOLORFILL	✓
DDCAPS_BLTDEPTHFILL	✓
DDCAPS_BLTFOURCC	
DDCAPS_BLTQUEUE	
DDCAPS_BLTSTRETCH	
DDCAPS_CANBLTSYSTEMEM	✓
DDCAPS_CANCLIP	
DDCAPS_CANCLIPSTRETCHED	
DDCAPS_COLORKEY	✓
DDCAPS_COLORKEYHWASSIST	
DDCAPS_GDI	✓
DDCAPS_NOHARDWARE	
DDCAPS_OVERLAY	✓
DDCAPS_OVERLAYCANTCLIP	✓
DDCAPS_OVERLAYFOURCC	✓ (YUV4:2:2, RBG555 and RGB565)
DDCAPS_OVERLAYSTRETCH	✓
DDCAPS_PALETTE	
DDCAPS_PALETTEVSYNC	
DDCAPS_READSCANLINE	✓
DDCAPS_STEREOVIEW	
DDCAPS_VBI	
DDCAPS_ZBLTS	
DDCAPS_ZOVERLAYS	
DDCAPS_ZOVERLAYS	

Table 3-8. dwCaps2—Specifies More Driver-Specific Capabilities

Function	Supported
DDCAPS2_CERTIFIED	
DDCAPS2_NO2DDURING3DSCENE	
DDCAPS2_VIDEOPORT	✓
DDCAPS2_AUTOFLIPOVERLAY	✓
DDCAPS2_CANBOBINTERLEAVED	✓
DDCAPS2_WIDESURFACES	✓
DDCAPS2_NOPAGELOCKREQUIRED	

Table 3-9. dwCKeyCaps—Color Key Capabilities

Function	Supported
DDCKEYCAPS_DESTBLT	✓
DDCKEYCAPS_DESTBLTCLRSPACE	
DDCKEYCAPS_DESTBLTCLRSPACEYUV	
DDCKEYCAPS_DESTBLTYUV	
DDCKEYCAPS_DESTOVERLAY	✓
DDCKEYCAPS_DESTOVERLAYCLRSPACE	
DDCKEYCAPS_DESTOVERLAYCLRSPACEYUV	
DDCKEYCAPS_DESTOVERLAYONEACTIVE	✓
DDCKEYCAPS_DESTOVERLAYYUV	✓
DDCKEYCAPS_NOCOSTOVERLAY	✓
DDCKEYCAPS_SRCBLT	✓
DDCKEYCAPS_SRCBLTCLRSPACE	
DDCKEYCAPS_SRCBLTCLRSPACEYUV	
DDCKEYCAPS_SRCBLTYUV	
DDCKEYCAPS_SRCOVERLAY	
DDCKEYCAPS_SRCOVERLAYCLRSPACE	
DDCKEYCAPS_SRCOVERLAYCLRSPACEYUV	
DDCKEYCAPS_SRCOVERLAYONEACTIVE	
DDCKEYCAPS_SRCOVERLAYYUV	

Table 3-10. dwFXCaps—Specifies Driver-Specific Stretching and Effects Capabilities

Function	Supported
DDFXCAPS_BLTARITHSTRETCHY	
DDFXCAPS_BLTARITHSTRETCHYN	
DDFXCAPS_BLMIRRORLEFTRIGHT	
DDFXCAPS_BLMIRRORUPDOWN	
DDFXCAPS_BLTROTATION	
DDFXCAPS_BLTROTATION90	
DDFXCAPS_BLTSHRINKX	
DDFXCAPS_BLTSHRINKXN	
DDFXCAPS_BLTSHRINKY	
DDFXCAPS_BLTSHRINKYN	
DDFXCAPS_BLTSTRETCHX	✓
DDFXCAPS_BLTSTRETCHXN	
DDFXCAPS_BLTSTRETCHY	✓
DDFXCAPS_BLTSTRETCHYN	
DDFXCAPS_OVERLAYARITHSTRETCHY	✓
DDFXCAPS_OVERLAYARITHSTRETCHYN	
DDFXCAPS_OVERLAYMIRRORLEFTRIGHT	
DDFXCAPS_OVERLAYMIRRORUPDOWN	
DDFXCAPS_OVERLAYSHRINKX	
DDFXCAPS_OVERLAYSHRINKXN	
DDFXCAPS_OVERLAYSHRINKY	
DDFXCAPS_OVERLAYSHRINKYN	
DDFXCAPS_OVERLAYSTRETCHX	✓
DDFXCAPS_OVERLAYSTRETCHXN	
DDFXCAPS_OVERLAYSTRETCHY	✓
DDFXCAPS_OVERLAYSTRETCHYN	

Table 3-11. dwPalCaps—Specifies Palette Capabilities

Function	Supported
DDPCAPS_1BIT	✓
DDPCAPS_2BIT	✓
DDPCAPS_4BIT	✓
DDPCAPS_8BIT	✓
DDPCAPS_8BITENTRIES	
DDPCAPS_ALLOW256	
DDPCAPS_PRIMARYSURFACE	
DDPCAPS_PRIMARYSURFACELEFT	
DDPCAPS_VSYNC	

Table 3-12. ddsCaps.dwCaps—Specifies The Capabilities Of The Surface

Function	Supported
DDSCAPS_3D	✓ (Enabled if 3D is detected)
DDSCAPS_3DDEVICE	✓
DDSCAPS_ALLOCONLOAD	✓
DDSCAPS_ALPHA	
DDSCAPS_BACKBUFFER	✓
DDSCAPS_COMPLEX	✓
DDSCAPS_FLIP	✓
DDSCAPS_FRONTBUFFER	✓
DDSCAPS_HWCODEC	
DDSCAPS_LIVEVIDEO	✓
DDSCAPS_MIPMAP	✓
DDSCAPS_MODEX	✓
DDSCAPS_OFFSCREENPLAIN	✓
DDSCAPS_OVERLAY	✓
DDSCAPS_OWDC	
DDSCAPS_PALETTE	✓
DDSCAPS_PRIMARYSURFACE	✓
DDSCAPS_PRIMARYSURFACELEFT	
DDSCAPS_SYSTEMMEMORY	✓
DDSCAPS_TEXTURE	✓
DDSCAPS_VIDEMEMORY	✓
DDSCAPS_VISIBLE	✓
DDSCAPS_WRITEONLY	
DDSCAPS_ZBUFFER	✓
DDSCAPS_NONLOCALVIDMEM	✓

3.5 Direct3D Interface

3.5.1 Supported Direct3D Capabilities

Table 3-13. General Device Capabilities

Function	Supported
Device Color Model	
RGB	✓
Mono	✓
Device Capabilities	
FloatTLVertex	✓
SortIncreasingZ	
SortDecreasingZ	
SortExact	
ExecuteSystemMemory	
ExecuteVideoMemory	
TLVertexSystemMemory	
TLVertexVideoMemory	
TextureSystemMemory	
TextureVideoMemory	✓
Transform Capabilities	
Clip	
Lighting Capabilities	
RGBModel	
MonoModel	
Point	
Spot	
Directional	
ParallelPoint	
GLSpot	
Clipping	
True	
False	✓
Render Bit Depth	
16-bit	✓
Z Buffer Bit Depth	
16-bit	✓

Table 3-14. Texture Capabilities

Format	Width	Height	Bits Per Texel	R/Y Mask	G/U Mask	B/V Mask	Alpha Mask
RGB 565	1024	1024	16	F800h	07E0h	001Fh	0000h
RGBa 5551	1024	1024	16	7C00h	03E0h	001Fh	8000h
RGBa 4444	1024	1024	16	0F00h	00F0h	000Fh	F000h
YUV 422	1024	1024	8	F0h	0Ch	03h	00h
Palette Indexed 1	1024	1024	1				
Palette Indexed 2	1024	1024	2				
Palette Indexed 4	1024	1024	4				
Palette Indexed 8	1024	1024	8				

Table 3-15. Primitive Capabilities Supported (Sheet 1 of 3)

Capability	Lines	Triangles
Misc. Capabilities		
MaskPlanes		
MaskZ	✓	✓
LinePatternRep		
Conformant		
CullNone		✓
CullCW		✓
CullCCW		✓
Raster Capabilities		
Dither	✓	✓
Rop2		
Xor		
Pat		
Ztest	✓	✓
Subpixel	✓	✓
SubpixelX		
FogVertex	✓	✓
FogTable		
Stipple	✓	✓
Z/AlphaCompare Capabilities		
	Z / Alpha	Z / Alpha
Never	✓ / ✓	✓ / ✓
Less	✓ / ✓	✓ / ✓
Equal	✓ / ✓	✓ / ✓
LessEqual	✓ / ✓	✓ / ✓
Greater	✓ / ✓	✓ / ✓

Table 3-15. Primitive Capabilities Supported (Sheet 2 of 3)

Capability	Lines	Triangles
NotEqual	✓/✓	✓/✓
GreaterEqual	✓/✓	✓/✓
Always	✓/✓	✓/✓
Source/Destination Blend Capabilities	Src / Dst	Src / Dst
Zero	✓/✓	✓/✓
One	✓/✓	✓/✓
SrcColor	✓/✓	✓/✓
InvSrcColor	✓/✓	✓/✓
SrcAlpha	✓/✓	✓/✓
IncSrcAlpha	✓/✓	✓/✓
DestAlpha		
InvDestAlpha		
IncSrcAlpha	✓/✓	✓/✓
InvDestColor	✓/✓	✓/✓
SrcAlphaSat		
BothSrcAlpha	✓/✓	✓/✓
BothInvSrcAlpha	✓/✓	✓/✓
Shade Capabilities		
ColorFlatMono	✓	✓
ColorFlatRGB	✓	✓
ColorGouraudMono	✓	✓
ColorGouraudRGB	✓	✓
ColorPhongMono		
ColorPhongRGB		
SpecularFlatMono	✓	✓
SpecularFlatRGB	✓	✓
SpecularGouraudMono	✓	✓
SpecularGouraudRGB	✓	✓
SpecularPhongMono		
SpecularPhongRGB		
AlphaFlatBlend	✓	✓
AlphaFlatStippled	✓	✓
AlphaGouraudBlend	✓	✓
AlphaGouraudStippled		
AlphaPhongBlend		
AlphaPhongStippled		
FogFlat	✓	✓
FogGouraud	✓	✓

Table 3-15. Primitive Capabilities Supported (Sheet 3 of 3)

Capability	Lines	Triangles
FogPhong		
Texture Capabilities		
Perspective	✓	✓
Pow2	✓	✓
Alpha	✓	✓
Transparency	✓	✓
Border		
SquareOnly		
Texture Filter Capabilities		
Nearest	✓	✓
Linear	✓	✓
MipNearest	✓	✓
MipLinear	✓	✓
LinearMipNearest	✓	✓
LinearMipLinear	✓	✓
Texture Blend Capabilities		
Decal	✓	✓
Modulate	✓	✓
DecalAlpha	✓	✓
ModulateAlpha	✓	✓
DecalMask	✓	✓
ModulateMask		
Copy	✓	✓
Texture Address Capabilities		
Wrap	✓	✓
Mirror	✓	✓
Clamp	✓	✓

3.5.2 Supported RenderState

Table 3-16. DIRECT3D RenderState Hardware / Software Support (Sheet 1 of 3)

State	Supported in SW	Supported in HW	Values	Notes
ALPHAFUNC	✓	✓	NEVER LESS EQUAL LESEQUAL GREATER NOTEQUAL GREATEREQUAL ALWAYS	
ALPHAREF	✓	✓	8-bit value	
ALPHATESTENABLE	✓	✓	TRUE / FALSE	
ANTIALIAS	✓	✓	SORTDEPENDENT / NONE	
ALPHABLENDENABLE	✓	✓	TRUE / FALSE	
CULLMODE	✓	✓	NONE CW CCW	
DESTBLEND	✓	✓	ZERO ONE SRCCOLOR INVSRCOLOR SRCALPHA INVSRCALPHA DESTCOLOR INVDESTCOLOR BOTHSRCALPHA BOTHINVSRCALPHA	
DITHERENABLE	✓	✓	TRUE / FALSE	
FILLMODE	✓	✓	WIREFRAME - SOLID	
FOGENABLE	✓	✓	TRUE / FALSE	
FOGCOLOR	✓	✓	lower 24-bits of a 32-bit value	
FOGTABLEDENSITY	NO	NO		
FOGTABLEEND	NO	NO		
FOGTABLEMODE	NO	NO		
FOGTABLESTART	NO	NO		
LASTPIXEL	NO	NO	TRUE / FALSE	
LINEPATTERN	NO	NO	32-bit value	
MONOENABLE	✓	✓	TRUE / FALSE	
PLANEMASK	NO	NO	32-bit value	

Table 3-16. DIRECT3D RenderState Hardware / Software Support (Sheet 2 of 3)

State	Supported in SW	Supported in HW	Values	Notes
ROP2	NO	NO		
SHADEMODE	✓	✓	FLAT GOURAUD	
SPECULARENABLE	✓	✓	TRUE / FALSE	
SRCBLEND	✓	✓	ZERO ONE SRCCOLOR INVSRCOLOR SRCALPHA INVSRCALPHA DESTCOLOR INVDESTCOLOR BOTHSRCALPHA BOTHINVSRCALPHA	
STIPPLEDALPHA	NO	NO		
STIPPLEENABLE	✓	✓	TRUE / FALSE	
STIPPLEPATTERN00-31	✓	✓	32-bit values	
SUBPIXEL	NO	NO		
SUBPIXELX	NO	NO		
TEXTUREADDRESS	✓	✓	WRAP MIRROR CLAMP	
TEXTUREHANDLE	✓	✓	32-bit value	
TEXTUREMAG	✓	✓	NEAREST LINEAR MIPNEAREST MIPLINEAR LINEARMIPNEAREST LINEARMIPLINEAR	
TEXTUREMAPBLEND	✓	✓	DECAL MODULATE DECALALPHA MODULATEALPHA DECALMASK COPY	
TEXTUREMIN	✓	✓	NEAREST LINEAR MIPNEAREST MIPLINEAR LINEARMIPNEAREST LINEARMIPLINEAR	
TEXTURE PERSPECTIVE	✓	✓	TRUE	

Table 3-16. DIRECT3D RenderState Hardware / Software Support (Sheet 3 of 3)

State	Supported in SW	Supported in HW	Values	Notes
WRAPUV	✓	✓	TRUE / FALSE	
WRAPV	✓	✓	TRUE / FALSE	
ZENABLE	✓	✓	TRUE / FALSE	
ZFUNC	✓	✓	NEVER LESS EQUAL LESSEQUAL GREATER NOTEQUAL GREATEREQUAL ALWAYS	
ZVISIBLE	NO	NO	TRUE / FALSE	
ZWRITEENABLE	✓	✓	TRUE / FALSE	

3.5.3 Supported RenderPrimitives

Table 3-17. DIRECT3D RenderPrimitive Hardware / Software Support

Primitive	Supported in SW	Supported in HW	Notes
POINT	✓	NO	Implemented as a 0 length line
LINE	✓	✓	
TRIANGLE	✓	✓	
SPAN	✓	NO	Implemented with a line
STRIP	✓	NO	Implemented with a triangle
FAN	✓	NO	Implemented with a triangle

3.6 Video Interface

All VfW Capture Messages are supported by the Intel740 video capture driver.

Table 3-18. VfW Capture Driver Capability

VfW Capture Message	Supported
DRV_LOAD	✓
DRV_FREE	✓
DRV_OPEN	✓
DRV_CLOSE	✓
DRV_ENABLE	✓
DRV_DISABLE	✓
DRV_QUERYCONFIGURE	✓
DRV_CONFIGURE	✓
DRV_INSTALL	✓
DRV_REMOVE	✓
DRV_GETVIDEOAPIVER	✓
DVM_GETERRORTEXT	✓
DVM_DIALOG	✓
DVM_PALETTE	✓
DVM_FORMAT	✓
DVM_PALETTERGB555	✓
DVM_SRC_RECT	✓
DVM_DST_RECT	✓
DVM_UPDATE	✓
DVM_CONFIGURE_STORAGE	✓
DVM_FRAME	✓
DVM_GET_CHANNEL_CAPS	✓
DVM_STREAM_INIT	✓
DVM_STREAM_FINI	✓
DVM_STREAM_GETERROR	✓
DVM_STREAM_GETPOSITION	✓
DVM_STREAM_ADDBUFFER	✓
DVM_STREAM_PREPAREHEADER	✓
DMV_STREAM_UNPREPAREHEADER	✓
DVM_STREAM_RESET	✓
DVM_STREAM_START	✓
DVM_STREAM_STOP	✓

3.7 GDI Escape Interface

The Intel740 Direct3D Driver supports the GDI Escape interface that allows dynamic alterations of operational parameters as well as debugging and performance monitoring. Access to these device capabilities which are specific to Intel740 3D functionality is achieved using the following function call:

```
ExtEscape( HDC, //handle to Windows device context
          int,  //Intel740 3D escape function number (1234h)
          int,  //number of bytes in input structure
          LPCSTR, //pointer to input structure
          //typedef struct AubControlInBuffer
          // { DWord EscapeNumber;
          //   DWordSubFunction;
          //   DWordDataPointer;
          // }
          int,  //number of bytes in output structure
          LPSTR); //pointer to output structure
          // typedef struct AubControlOutBuffer
          // { DWordEscapeNumber;
          //   DWordSubFunction;
          //   DWordDataPointer;
          // }
```

The following sections define the available subfunctions along with a definition for each DataPointer associated with the input and/or output structures. Data types which are in bold italic text are defined by Microsoft in the DirectX documentation.

Table 3-19. Functionality Control

Sub-function	Description	AubControlInBuffer Data	AubControlOutBuffer Data
101h	Set State Variable	DWord StateNumber 01h-FFh - As defined by D3DRENDERSTATETYPE 100h - Texture LOD Bias 101h - Texture LOD Dither weight 102h - Alpha in Z buffer 103h - QWord fetch mode DWord StateValue	void
102h	Set Capabilities	D3DDEVICEDESC D3Dcapabilities	void
103h	Get Capabilities	void	D3DDEVICEDESC D3Dcapabilities
10Ah	Get AGP Config Registers	void	DWord Reg[3]

Table 3-20. Device Driver Debugging Control

Sub-function	Description	AubControlInBuffer Data	AubControlOutBuffer Data
200h	Set Debug Logging Level	DWord Level 0..MaxDebugLevel	void

This chapter describes programming approaches to maximize performance, report Intel740 performance test results, and introduce creative programming techniques which take advantage of the Intel740 features.

4.1 Performance Strategies And Measurements

All performance statistics outlined in this section were gathered using Intel's RasM (Raster Metric) 2.0 software. RasM, a raster speed measurement tool, measures the rasterization speed of a hardware accelerator vs. the scene complexity of an application. The system configuration used for gathering the data shown in this document is as follows:

- 300 MHz Pentium® II processor with MMX™ technology
- Atlanta motherboard with PhoenixBIOS*
- 440LX Chipset
- Intel740 AGP graphics card with 200 BIOS
- Windows95 operating system (OSR2.1)
- 64 Mbytes system memory (SDRAM, 66 MHz)
- 4 Mbyte local video memory (SDRAM, 100 MHz)
- 640x480x16 bits per pixel screen resolution
- 60 Hz refresh rate

4.1.1 Intel740 Performance Capabilities

The Intel740 supports the next generation of high-content applications. 3D games will use more realistic models with more triangles of smaller size. The Intel740 provides its peak performance for these types of games.

The recommended game detail target for the Intel740 is 10,000 triangles per frame, between 75 and 175 pixels per triangle, at 30 frames per second. 10,000 triangles per scene requires a triangle rate of approximately 300,000. The Intel740 can render 366,000 full featured triangles per second with an average of 105 pixels per triangle.

$$\text{Required_Tri_Per_Sec} = \text{Tri_Per_Scene} / (1/\text{Frames_per_Second} - \text{Tover_head})$$

The following sections include Intel740 performance results along with descriptions of how the results can be used to predict frame rates for particular applications and scene complexities.

4.1.2 Using CPU/Intel740 Concurrency

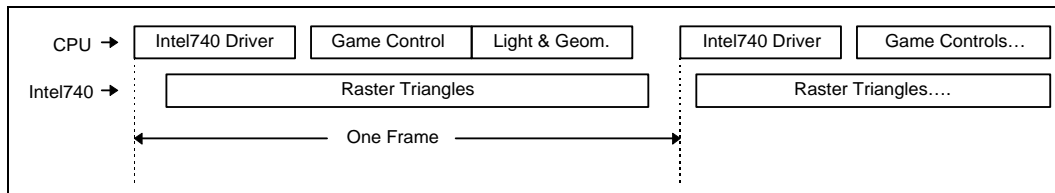
Applications should be designed to take advantage of the concurrency allowed by the Intel740 and AGP system architecture. The Intel740 can be thought of as a second processor for rasterization, optimized for maximum parallelism with the CPU. The benefit given to the application is that the CPU is free to do more AI, physics, lighting, and geometry. The Intel740 drivers minimize CPU overhead, balance the system, and allow for maximum system concurrency.

Many of the performance results included in this chapter report the driver duty cycle for the CPU. The duty cycle is the ratio of CPU time used by the Intel740 driver divided by the length of time the Intel740 requires to render the scene. It is a measure of how much time an application can spend on lighting, geometry, and game controls while not causing the CPU to limit performance.

In systems with software rasterization only, a typical application used 90% of CPU cycles for rasterization alone. Because the Intel740 renders much faster than software engines and because of the system's available concurrency, a system with an Intel740 gains a tremendous performance advantage.

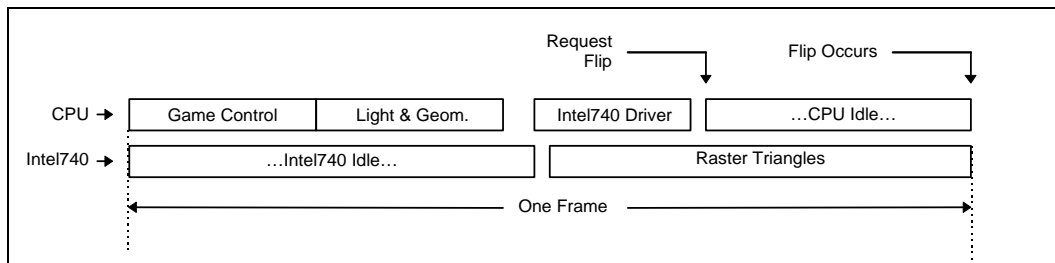
Figure 4-1 shows the usage model for the Intel740 and the CPU during one and a half frames of a typical application cycle.

Figure 4-1. Intel740/CPU Usage Model



Applications should be structured such that CPU cycles are not wasted waiting for synchronization with the Intel740. Forcing flips or blits to surfaces being rendered cause the CPU to sit idle until rendering has completed. Figure 4-2 illustrates how an improperly placed flip or blit can drastically reduce frame rate.

Figure 4-2. Improper Usage Model



In this case, only minimal concurrency is achieved. The problem can be alleviated by simply rearranging the flow so that the CPU processing for the following frame is completed before the render target is flipped. Similar problems will be seen by code that issues blit commands for 2D effects directly after sending a 3D scene.

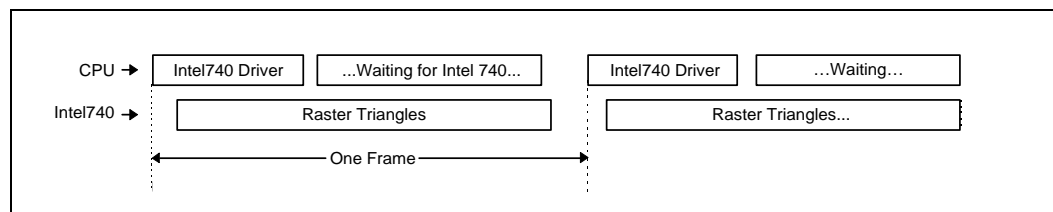
4.1.3 Performance Test Results

4.1.3.1 Raster Speed Test Method

This section describes the tests used to measure the performance numbers reported in this document.

Figure 4-3 shows the system usage while RasM is running. The time that RasM waits for the Intel740 to complete will be used for AI, game control, lighting, geometry, and anything else the application needs to do before sending the next frame off to be rendered. To attain the maximum frame rate, applications should be optimized to finish all computations during this time.

Figure 4-3. RasM Intel740/CPU Usage Model



The program execution can be broken down into two phases called consecutively by a loop that sequences through all the triangle sizes to be tested:

```

Loop (for all triangle sizes do)
    Phase 1: Build execute buffers
    Phase 2: Execute the buffers and time the hardware
    
```

The first phase creates and fills execute buffers with 512 triangles each. The total number of triangles depends on triangle size, depth complexity (DC) goal, and percent Z-buffering (%Z) goal. Unless otherwise stated, the sweeps reported in this document have a constant DC of 2.5 and 50% Z across the triangle size sweeps. For example, the 120 pixel/triangle data point contains about 13,300 randomly distributed triangles per scene:

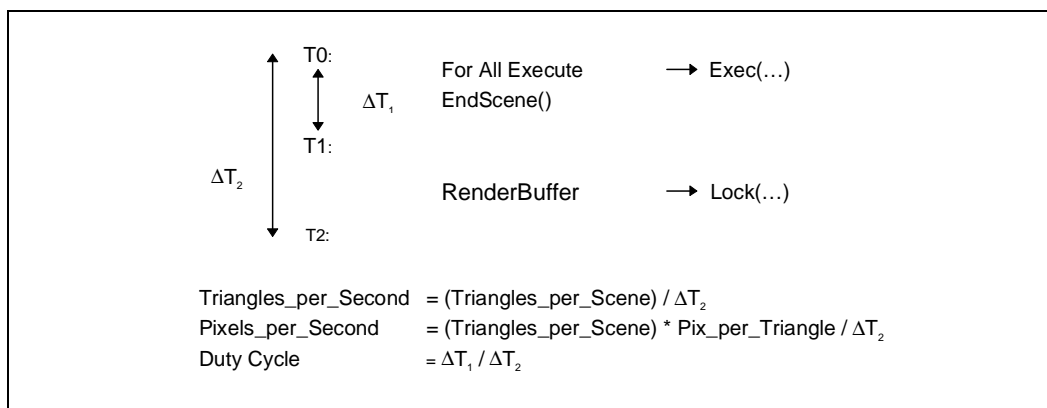
$$\text{Triangles_per_Scene} = (\text{Screen.W} * \text{Screen.H} * \text{Avg_DC_Goal} / \text{Percent_Z_Goal}) / \text{Pix_per_Tri}$$

To achieve a predefined DC and %Z goal, a “survival of the fittest” algorithm is implemented: 10 randomly placed and oriented triangles are generated for each required triangle, and the triangle that brings the scene the closest to its DC and %Z goals is selected.

Game scenes often have some percentage of triangles use specular and alpha blend. Many of the sweeps in this document test scenes with specular and blend enabled for only a fraction of the triangles. RasM always puts the triangles with specular and blend in the last portion of the scene. The rationale here is that games using only a small percent specular or alpha blend will be applying highlights to the scene near the end of their triangle lists. Unless otherwise stated, textures are mipmapped with 16-bit color. When multiple textures are used in a scene, they are distributed equally throughout the scene. A scene with 10,000 triangles, three textures, 30% specular, and 20% alpha blend would generate 20 execute buffers, 3,400 triangles per texture; the last 2,000 triangles would use specular and alpha blend, and the preceding 1,000 would use just specular.

The second phase of the loop executes the buffers created in the first phase, and then clocks the driver and hardware raster speed. The scene is clocked, displayed, and recorded 15 times; the middle five times are averaged to get the final result. Figure 4-4 shows pseudo-code from the timing/display loop.

Figure 4-4. RasM Pseudo-Code



The reported results are divided into sections: Result Summary, Basic Sweeps, Advanced Sweeps, and Full Sweeps. The Result Summary contains data taken directly from the set of sweeps. It is intended to be used as just a summary or for quick reference. Section 4.1.3.2 contains more detailed information that can be used to predict application frame rates.

The basic sweep compares Gouraud only to Gouraud with Z-Buffer and, finally, Gouraud with Z-Buffer and Textures (GZT). The advanced sweeps takes the GZT features from the last basic sweep and tests the sensitivity to fog, alpha blending, specular, and anti-aliasing. The full sweeps combine all features.

Table 4-1. Result Summary

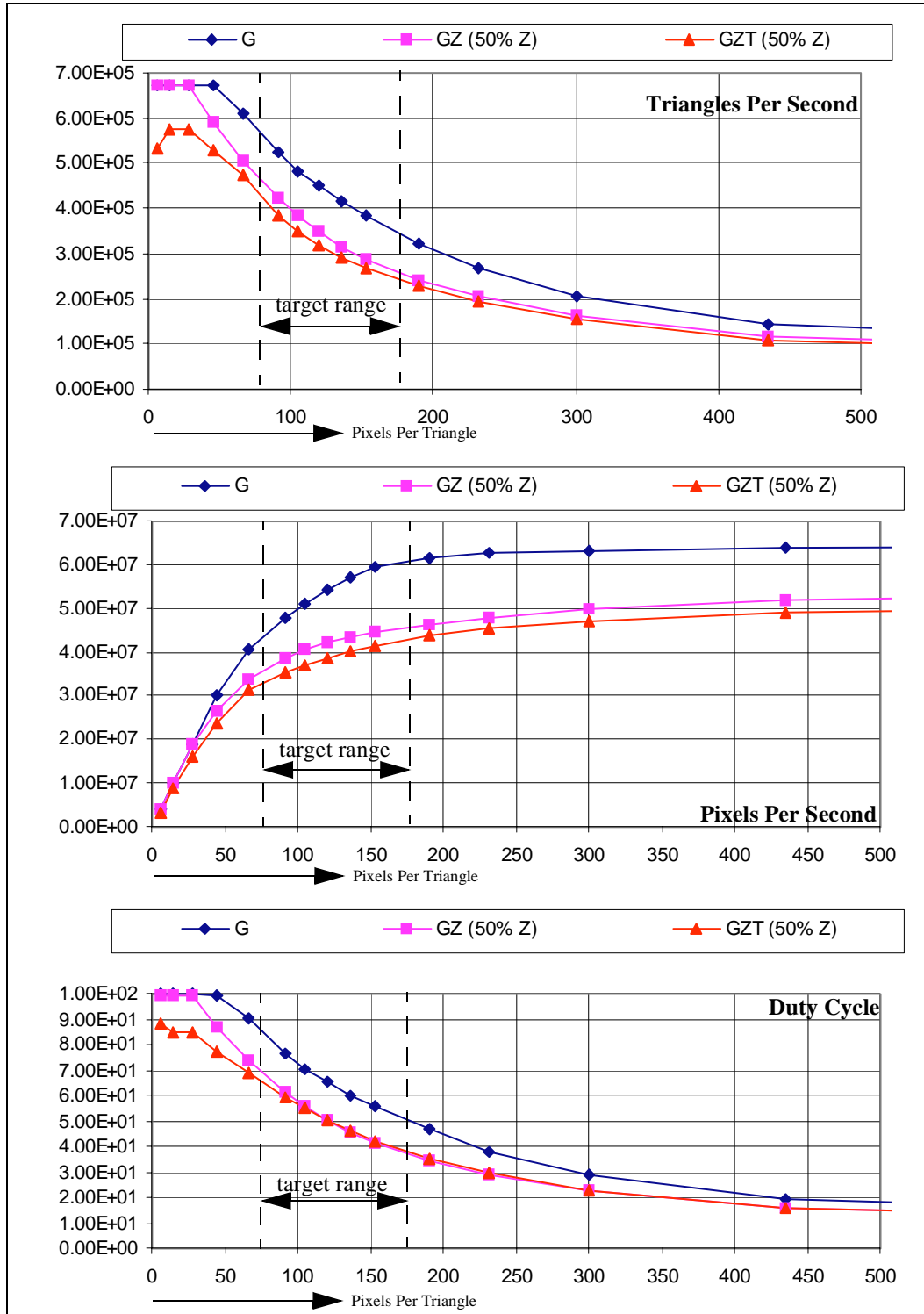
Gouraud	Fog	Blend	Spec	Anti-Alias	Z-Buff 50% Z	MipMap 16 BBP	Set-up Limited (Triangles per Second)	105 PixTri (Triangles per Second)	Fill Rate (Pixels per Second)
X							675k	482k	65.7M
X					X		672k	385k	59.1M
X					X	X	577k	349k	54.2M
X	X				X	X	535k	349k	53.8M
X		20%			X	X	568k	340k	53.8M
X		100%			X	X	535k	300k	48.2M
X			30%		X	X	539k	348k	53.7M
X			100%		X	X	468k	347k	54.2M
X				20%	X	X	372k	253k	51.4M
X	X	20%	30%		X	X	496k	338k	53.2M
X	X	100%	100%		X	X	415k	298k	48.0M

Table 4-2. Symbol Key

Symbol	Definition
G	Gouraud Shading
Fg	Vertex Fog Enabled 100% of Scene
A20 and A100	Alpha Blend Enabled for 20% and 100% of Scene
S30 and S100	Specular Highlights Enabled for 30% and 100% of Scene
Aa20	Anti-Aliasing enabled for 20% of Scene
Z	Z-Buffer enabled. Cleared at beginning of Scene
T	256X256 MipMap BiLinear Filter, ARGB 0565 Format (unless otherwise stated)

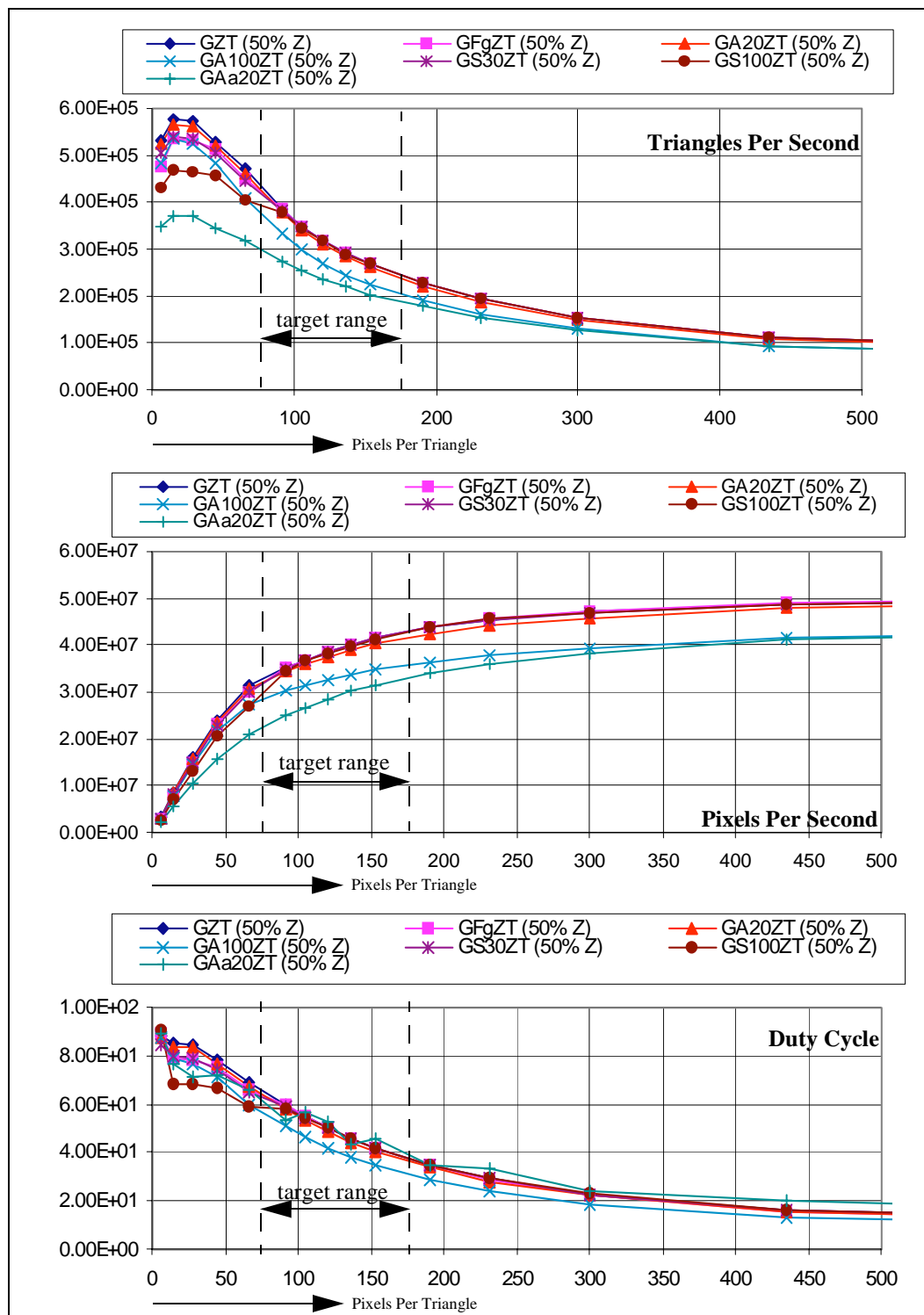
The graphs in Figure 4-5 show triangles per second, pixels per second, and duty cycle for Gouraud only, Gouraud with Z-Buffer and Gouraud with Z-Buffer and Textures.

Figure 4-5. Basic Feature Sweeps



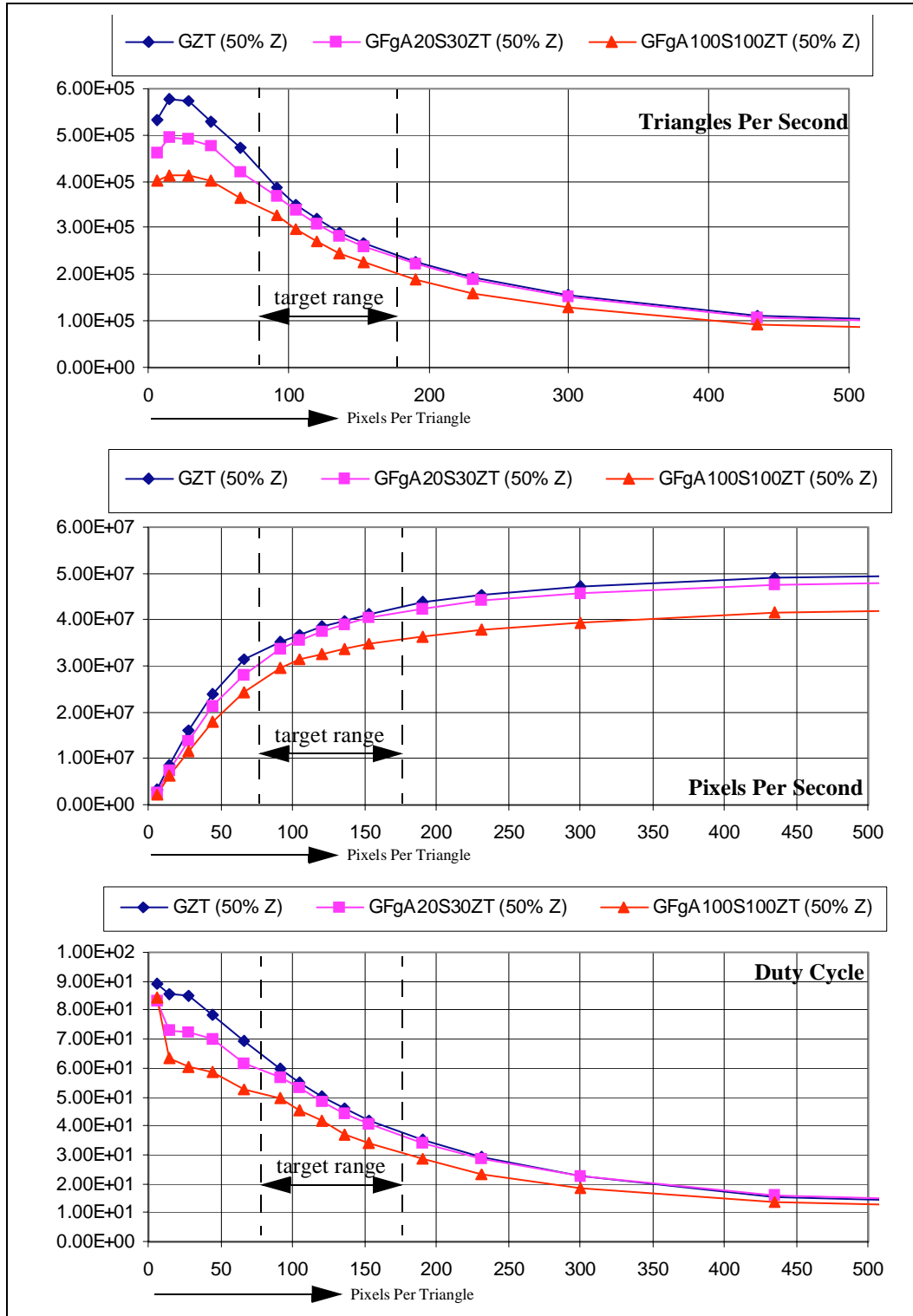
The graphs in Figure 4-6 show triangles per second, pixels per second, and duty cycle. The feature sets start with the GZT features set from the last basic sweep and display the sensitivity to fog, alpha blending, specular, and anti-aliasing.

Figure 4-6. Advanced Feature Sweeps



The graphs in Figure 4-7 show triangles per second, pixels per second, and duty cycle with full feature sets.

Figure 4-7. Full Feature Sweeps



4.1.3.2 Implications and Analysis

This section suggests how the reported results can be translated into performance for individual applications. The tests are raster speed only. Because of system concurrency, if the application code executed between scenes preserves the duty cycle, the stated triangle and fill rates will be achieved.

Average and percent Z are a good measure of scene complexity from the graphics card's point of view. They actually define the number of pixels that will be processed by the graphics accelerator, per scene. Pixels per scene and desired frames per second give the fill rate that is required of the graphics accelerator to hit that frame rate.

$$\text{Pixels_per_Scene} = (\text{Screen.W} * \text{Screen.H} * \text{Avg_DC_Goal} / \text{Percent_Z_Goal})$$

$$\text{Required_Fill_Rate} = \text{Pixels_per_Scene} / (1/\text{Frames_per_Second} - \text{Tover_head})$$

Tover_head is the overhead time which may be required to clear the Z-buffer, render buffers, or blit a background. The number and size of triangles per scene may be more convenient for a game designer to work with, but it is not a difficult conversion between the two.

$$\text{Avg_DC_Goal} = \text{Triangles_per_Scene} * \text{Pix_per_Tri} * \text{Percent_Z_Goal} / (\text{Screen.W} * \text{Screen.H})$$

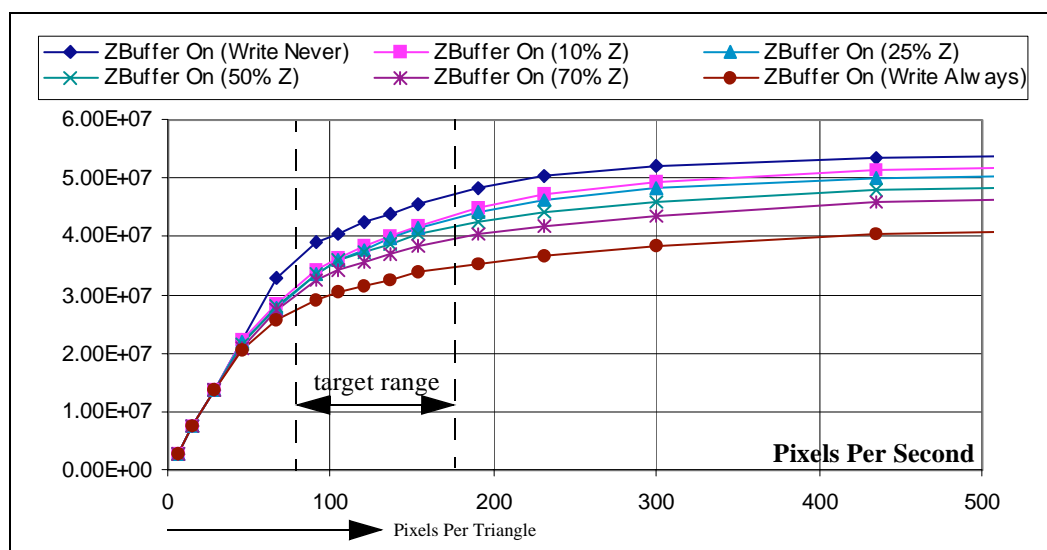
The %Z goal is really just how well the triangles are ordered before being sent to the Intel740 graphics accelerator. 50% Z assumes that half of the pixels contained in the processed triangles will actually not be written to the screen because they are behind the previous pixel in the z-order. Note that for a constant number of pixels per scene, if %Z goes up (a higher number of Z-values are written) then the DC also goes up. Even though the pixels per scene remains the same, the fill rate will change because it is a function of %Z.

A scene complexity of 2.5 DC and 50% Z was chosen because it is predicted that typical games will have a similar complexity. However, not all games will follow this pattern.

Depth complexity is really a measure of pixels per scene. Increasing DC does not affect the triangle rate or the actual fill rate, but will affect the pixels per scene and the required fill rate according to the equations mentioned above.

Percent Z occlusion does affect the triangle and fill rates. Basically, decreasing %Z increases fill rate, and vice versa. Sorting triangles from front to back produces higher graphic card performance. Implementing a sorting algorithm is only recommended when the Intel740 fill rate becomes the system performance bottleneck. The following graph illustrates the performance with changing scene %Z occlusion.

Figure 4-8. Performance vs. Percent Z Occlusion



Very few games will have just one triangle size per scene, but it is useful to analyze just one size at a time because it supplies many of the building blocks required to approximate triangle rate, fill rate, and duty cycle for more complex scenes. This example uses a game scene of 7,000 triangles of 75 pixels and 3,000 triangles of 175 pixels, has a 50% Z, uses a full feature set of GFgA20S30ZT, has a Tover_head of about 1 ms, and requires 30 frames per second. The average DC for the scene comes to 2.34, the pixels per scene is 1.2M, and it requires a fill rate of 37.1M pixels per second.

$$\text{Avg_DC_Goal} = (75 * 7,000 + 175 * 3,000) * .5 / (640 * 480) = 1.71$$

$$\text{Required_Fill_Rate} = 1.05\text{M} / (1/30 - .001) = 32.5\text{M}$$

The fill rate for this type of scene is not explicitly quoted in the graphs included in this document, but a weighted average based on numbers of pixels can be used to extrapolate the Intel740 resultant fill rate. For the previous example, the extrapolated fill rate of the Intel740 is 35.2M pixels/s.

$$\text{Pixels_per_Second (estimate)} = (525\text{k} * 30.4\text{M} + 525\text{k} * 40.0\text{M}) / 1.2\text{M} = 35.2\text{M Pix/s}$$

RasM can be used to test scenes with non-constant triangle sizes. When the hardware was tested for this case, the actual fill rate was reported to be 34.2M pixels/s. Most of the discrepancy can be attributed to the scene depth complexity in this example being below that of the quoted tests. For more information on how DC (or total packet size) can affect performance, see Table 4.1.4.2 “Triangle Packet Size” on page 4-12.

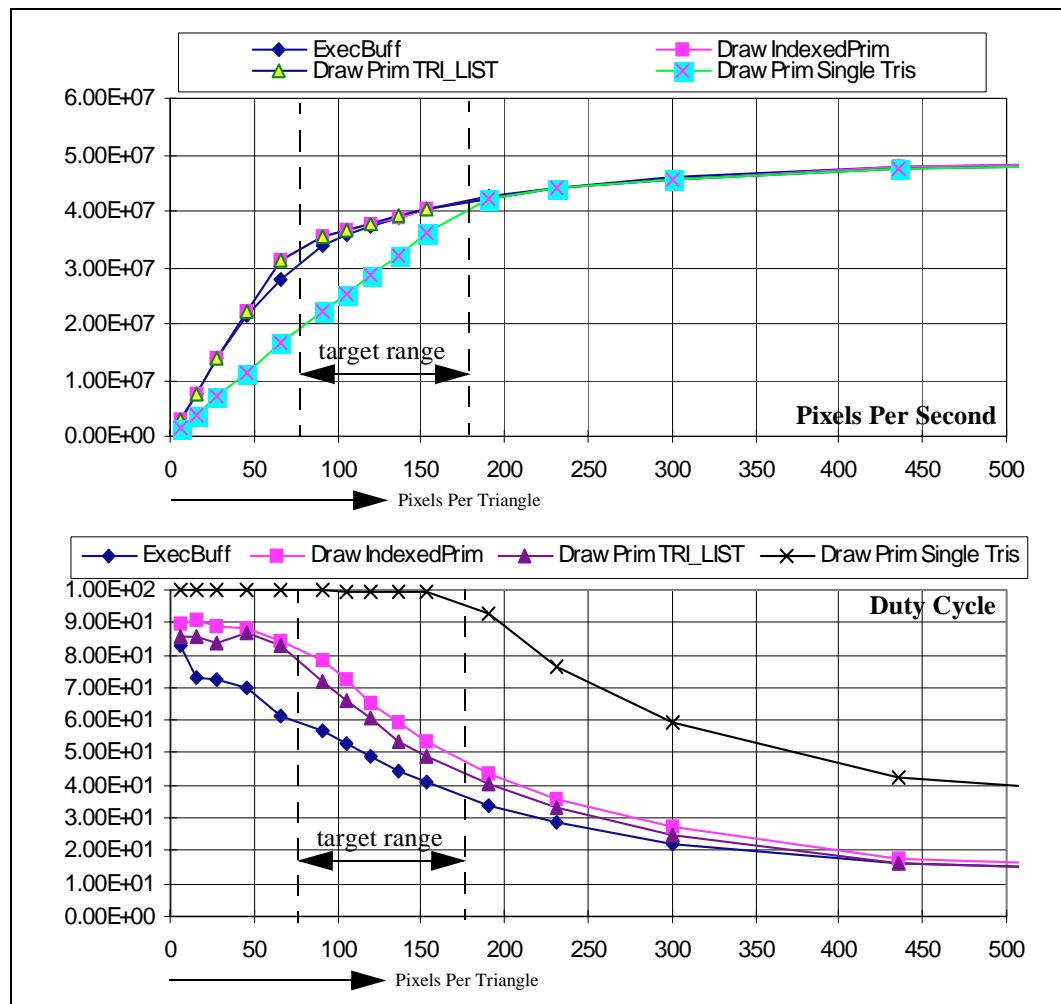
4.1.4 Special Performance Considerations

This section contains descriptions of subtle application design choices which can have considerable effects on performance.

4.1.4.1 Direct3D DrawPrimitive vs. Execute Buffers

Direct3D immediate mode allows programmers to choose between execute buffers and draw primitive methods of sending commands to the graphics hardware. The Intel740 performance and CPU driver duty cycle are both nearly identical for either sets of methods. This is the case as long as other considerations such as concurrency and packet size are not ignored. The following full feature sweeps (Fog, 20% Alpha, 30% Specular, MipMap Textures, 2.5 DC, 50% Z) use execute buffers, draw indexed primitive, draw primitive with triangle lists, and draw primitive with discrete triangles. Each of the instructions sending groups of triangles (includes all but draw primitive with discrete triangles) issues 500 triangles per instruction.

Figure 4-9. Performance of DrawPrimitive vs. Execute Buffer



Each method has an associated CPU overhead. Execute buffers have the lowest, followed by draw primitive with triangle lists. Sending a single triangle with each draw primitive command has a very high overhead; below about 200 pixels per triangle the CPU is unable to send enough triangles down per second to keep the Intel740 busy.

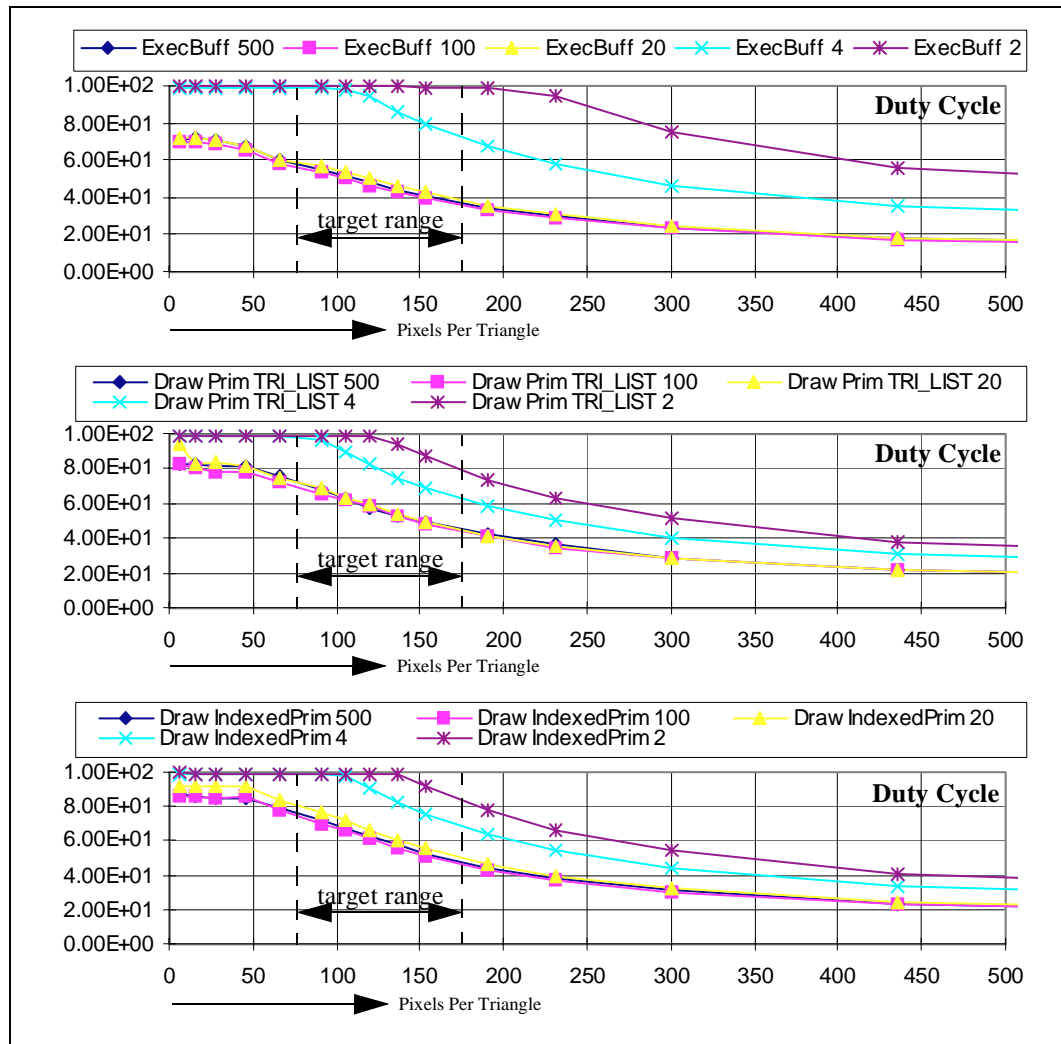
It is important to note that execute buffers tend to force applications to group triangle execution commands, which is advantageous for the Intel740 and its driver. For more information on Performance vs. triangle packet size see Section 4.1.4.2, “Triangle Packet Size” on page 4-12.

4.1.4.2 Triangle Packet Size

Software designers should try to bunch triangle packets sent to the Intel740 driver. Because of the overhead associated with starting the flow of command packets, sending a small number of triangles in a packet decreases performance. By sending out large triangle packets, the overhead is amortized over the rasterization time of all triangles. As a result, higher triangle and fill rates are achieved. Grouping rastered triangles is also critical to maintaining a high level of CPU/Intel740 concurrency. For more information on concurrency, see Section 4.1.2, “Using CPU/Intel740 Concurrency” on page 4-2.

This section addresses both performance vs. execute buffer/draw primitive buffer size, and performance vs. total packet size. The total packet size is the total number of triangles sent between breaks caused by game controls, lighting, or other CPU tasks. It consists of all the execute buffer/draw primitive buffers sent down one right after the other.

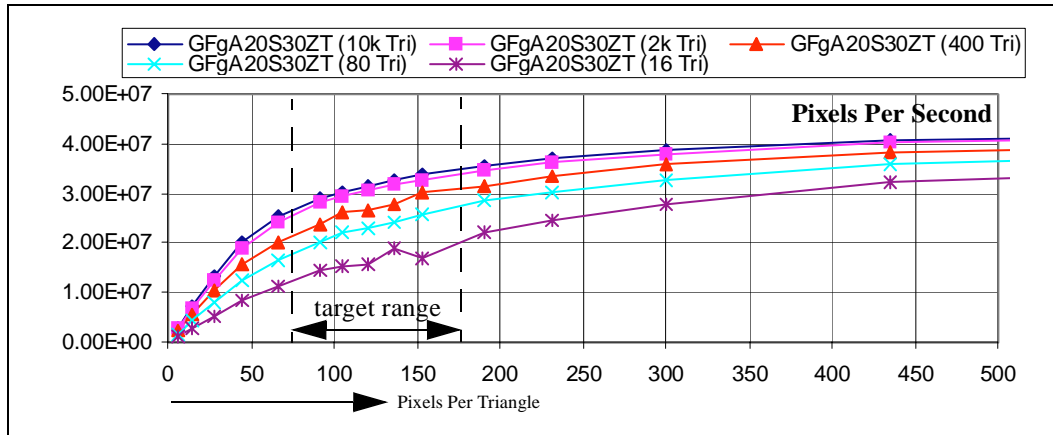
The following graphs illustrate the performance vs. execute buffer size, draw indexed primitive triangle list size, and draw primitive list size. All of these sweeps are full feature sweeps (Fog, 20% Alpha, 30% Specular, MipMap Textures) and have a constant 10,000 triangle total packet size. The Intel740 fill rate is not affected; the following graphs show duty cycle (CPU overhead).

Figure 4-10. Performance vs. Buffer Size (Duty Cycle)


Optimal D3D execute-buffer size on a Pentium® II processor system with an Intel740 has been determined to be 512 triangles. Keeping a buffer size above about 50 triangles may be considerably easier to implement and will only cost a few percent performance degradation.

The second and equally important concern is performance vs. total packet size. Applications need to have a minimum of about 2,000 triangles per packet (which if organized efficiently is equal to triangles per scene) to achieve near maximum system performance. The following graph illustrates how sending small numbers of triangles in a packet can drastically reduce performance. An example of how this can happen is an application with a render loop which sends many small triangle packets divided up by game controls. Note that the following curves have 100% Z writes in order to keep the %Z constant with changing triangle packet size.

Figure 4-11. Performance vs. Total Packet Size

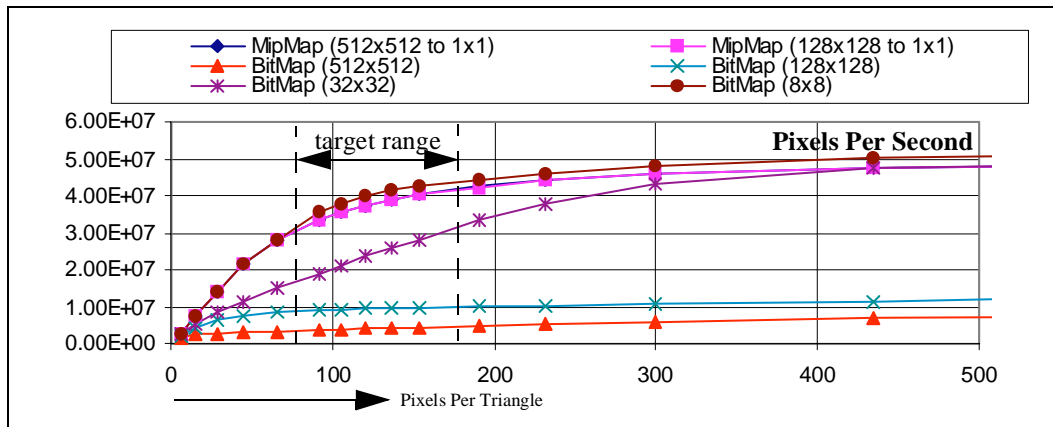


4.1.4.3 Texture Sizes

The ratio of texture-mapped area to triangle area can have a very significant performance impact. Mapping large non-mipmapped textures onto small triangles forces the Intel740 to scan through much of the texture for just a few texels. When a textured triangle can be viewed up close as well as far away, mipmapping is an excellent choice. Using mipmapped textures, in addition to looking better, alleviates this problem by selecting a texture map size which is close to the textured triangle size.

The following graph demonstrates how performance can be degraded by texture to triangle size mismatches.

Figure 4-12. Performance vs. Texture Size

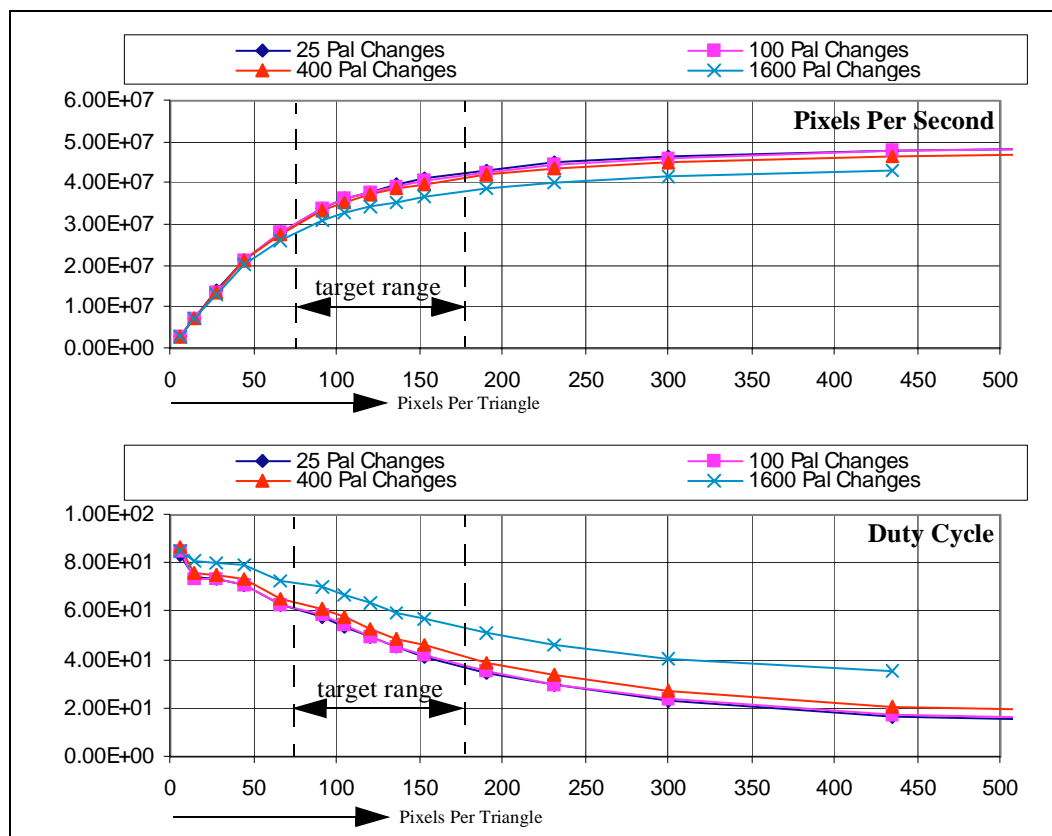


A 32x32 bitmap maps directly onto a 512 pixel triangle. Notice that this size bitmap considerably degrades performance of triangles smaller than about 300 pixels (about half of the direct mapped triangle size). In general, the bitmap area being mapped onto a triangle should be no larger that twice the triangle area in order to maintain high performance. The mipmapped textures (512x512 and 128x128) achieve high performance by allowing the Intel740 to select the texture size.

4.1.4.4 Palette Changes

The Intel740 is optimized for 16-bit textures. It is recommended that applications use 16-bit textures over 8-bit palettized textures. Palettized textures are supported with a relatively low overhead. The following graph reports the performance of a full-featured scene (Fog, 20% Alpha, 30% Specular, 2.5 DC, 50% Z) with a varied number of palette changes per scene.

Figure 4-13. Performance vs. Palette Changes



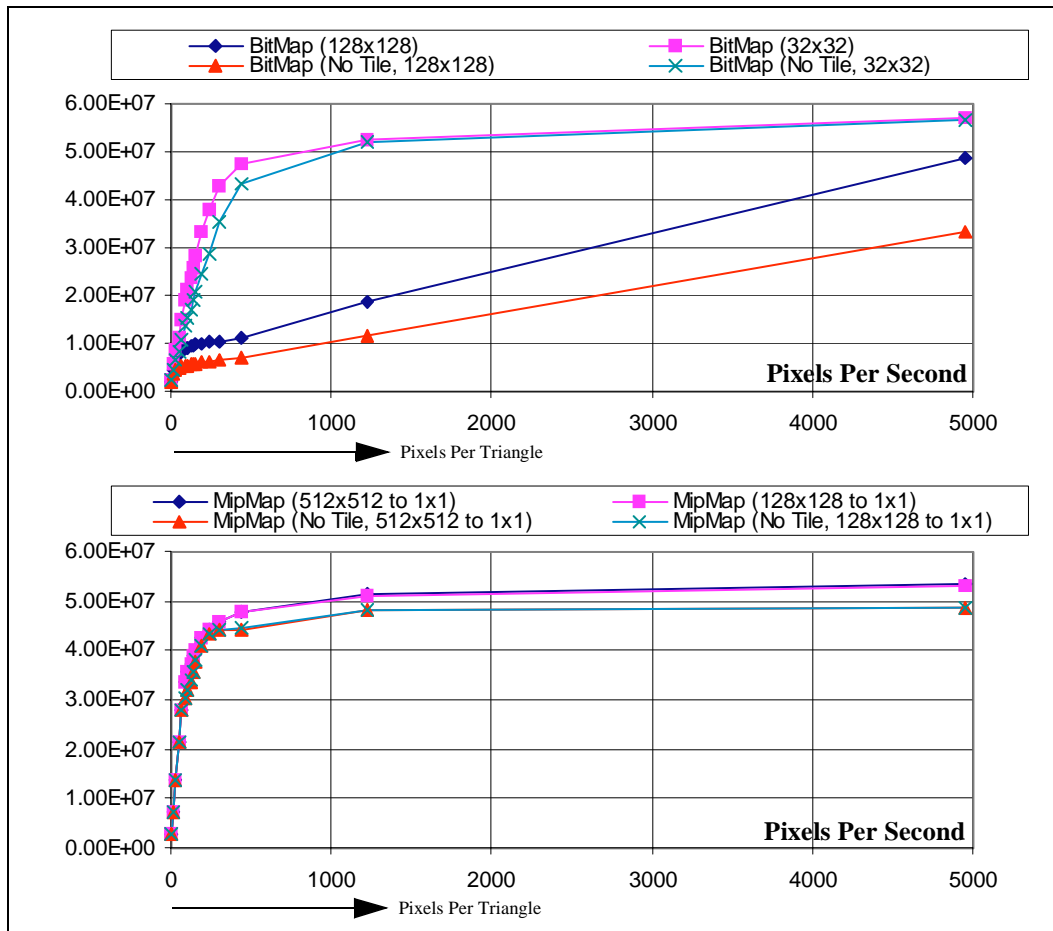
Application developers can use these graphs as an indicator of when to sort palettized textured triangles by texture handle. If an application is CPU limited, sorting by texture handle will degrade performance in most cases.

4.1.4.5 Untiled Textures for Procedural Texture Animation

Directly modifying texture surfaces in AGP memory can be used as a powerful method for creating many types of stunning effects. This section describes the performance implication of using untiled textures. For more information on how to create effects with procedural animation and on Intel740 tiling, see Section 4.2.1.4, "Animated Texture Effects" on page 4-21.

Triangles which use untiled textures will be processed with some performance degradation. Figure 4-14 illustrates the performance difference between triangles using tiled and untiled textures.

Figure 4-14. Performance with Untiled Textures

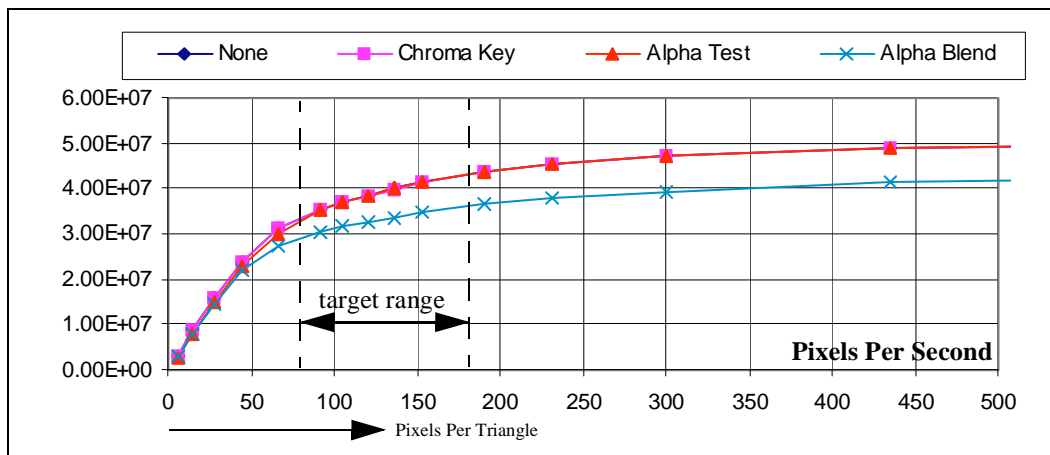


Untiled textures can degrade performance when large texture maps are used or when large triangle to texture map size mismatches are present. Note that in the case of mipmaps, only a small performance degradation is seen for both 512x512 and 128x128. This is because the triangle to texture size mismatch is minimized, so only the degradation with large texture maps is seen. For more information on performance of triangle to texture size mismatch, see Section 4.1.4.3, “Texture Sizes” on page 4-14.

4.1.4.6 High Performance Transparency

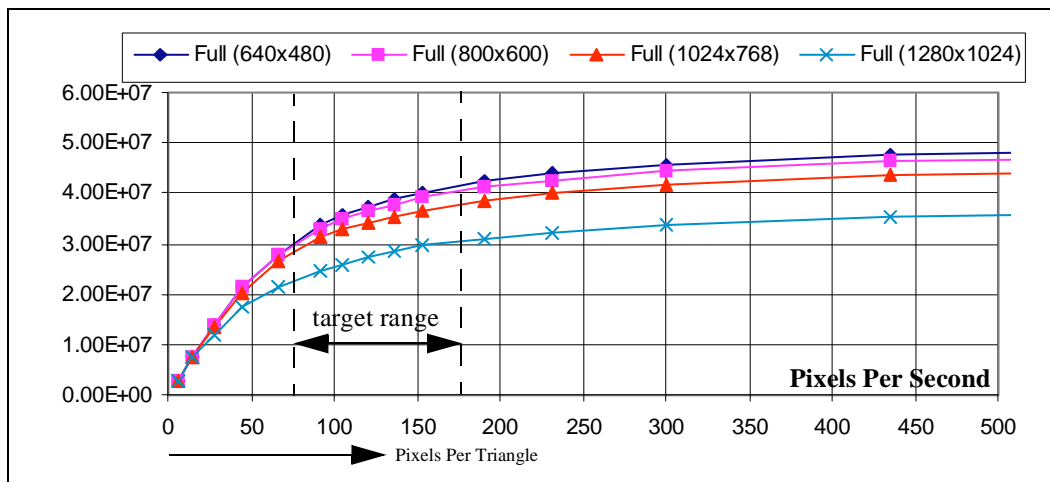
Methods of implementing transparency include: chroma keying, alpha testing, and alpha blending. If performance is the primary concern, chroma keying or alpha testing should be used over alpha blending. The Intel740 implements both without any performance degradation. When translucency is desired, alpha blending is supported with only a minor performance decrease.

The following graph illustrates the performance of chroma keying, alpha testing, and alpha blending. The sweeps use a feature set of Gouraud, Mipmapped Textures, 2.5 DC, and 50% Z.

Figure 4-15. Performance vs. Transparency


4.1.4.7 Screen Resolutions

The Intel740 3D performance is optimized for 640x480 and it is recommended that applications target 3D graphic intensive applications for this resolution. The following graph illustrates performance scaling for greater resolutions. The tests are full-feature sweeps (Fog, 20% Alpha, 30% Specular, Mipmapped Textures, 2.5 DC, 50% Z). This test is run with 8 Mbytes of video memory to enable 1280x1024 to fit both the render and Z-Buffer in local video.

Figure 4-16. Performance vs. Screen Resolution


Note that 1280x1024 mode is actually faster than 1024x768 mode because it is interlaced, which does not require as much local memory bandwidth.

4.1.5 Budgeting CPU Clock Cycles

For an application to achieve a sustainable high frame rate, the CPU must calculate lighting, geometry, and game controls, and send the triangle information off to the Intel740 — all within each frame period. Budgeting CPU clock cycles to fit within the Intel740 duty cycle is imperative to this task.

For the Intel740, it is suggested that developers of 3D applications target 10,000 triangles per frame at 30 frames per second. The numbers listed in Figure 4-3 show a conservative analysis of the needed CPU clock cycles and assumptions. The user can anticipate good overall performance when implementing full features of the Intel740 and using these targets.

Assumptions:

- Intel740 state and operand(s) change overhead not considered
- No Page-Miss on Execute Buffer Reads
- No FP to MMX™ instruction alignment cycles considered
- Theoretical full bandwidth of memory bus available
- Definition of 24 DWords/triangle (96 bytes)

Table 4-3. CPU Cycle Targets

Function	Description	Notes
Frames per Sec	30	
CPU Speed	233 MHz	
CPU Clocks/Triangle	200	
Triangles/Sec	300,000	Triangles/Frame * Frames/Second
Triangles/Frame	10,000	
CPU Clocks/Frame	2,000,000	Triangles/Second * CPU Clocks/Triangle

4.1.6 Video Performance

Figure 4-17. Available Memory Bandwidth on a Pentium® II Processor System

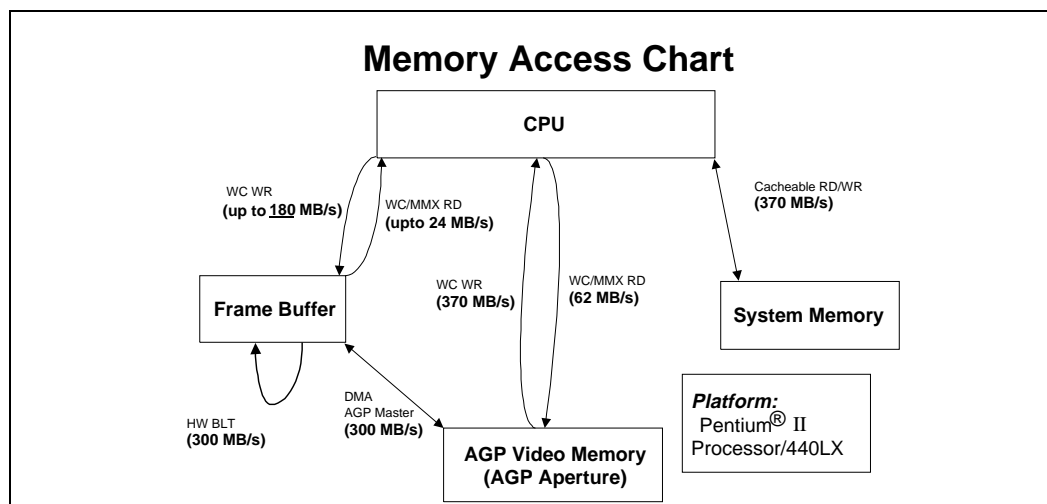


Table 4-4 shows the video/data rates for some typical applications. The highest data rate for video capture is in application of video conferencing on a 200 kbps ISDN line. The highest video display data rate is 20 Mbyte/s in DVD/MPEG-2 playback applications.

Table 4-4. Typical Video/Data Capture Applications

Application	Format	Frame Rate (fps)	Resolution hor*vert*pixdep	Frame Size (bytes)	Bandwidth (Mbytes/s)
Intercast (VBI)	Raw Data	30	800 x 22 x 2	35,200	1.0
POTS Video Conf	Sub-QCIF	15	128 x 96 x 2	24,576	0.37
POTS VC	QCIF	12	176 x 144 x 2	50,688	0.6
ISDN VC (128kbps)	CIF	12	352 x 288 x 2	202,752	2.4
ISDN VC (200kbps)	CIF	15	352 x 288 x 2	202,752	3.0
DVD/MPEG-2	DCIF	30	720 x 480 x 2	691,200	20

Table 4-5 shows the CPU usage for those applications, which can be calculated based on the memory bandwidth. Note that most applications will benefit from the higher read bandwidth of AGP aperture, if the video or VBI data can be routed through the AGP aperture. In this case, the CPU usage for data capturing will be under 5%, making the capture I/O a less degradation factor for the applications. Similarly, the high CPU write bandwidth of AGP aperture can also be useful for DVD/MPEG-2 playback applications.

Table 4-5. CPU Usage for Some Typical Applications

Video/Data Stream			CPU Usage (%)			
Format	Frame Rate	BW (Mbytes/s)	FB Read (BW= 24Mbytes/s)	AGP Read (BW= 62Mbytes/s)	FB Write (BW= 180Mbytes/s)	AGP Write (BW= 360Mbytes/s)
VBI	30fps	1.0	4.2%	1.6%		
SQCIF	15	.37	1.5%	0.6%		
QCIF	12	0.6	2.5%	1.0%		
CIF	12	2.4	10%	3.9%		
CIF	15	3.0	13%	4.8%		
DCIF	30	20			11%	5.6%

4.2 Other Programming Tips

4.2.1 Texture and Surface Effects

Several aspects of texture usage are discussed in this section including:

- “Texture Formats” on page 4-20
- “Texture Sizes” on page 4-21
- “Texture Storage” on page 4-21
- “Animated Texture Effects” on page 4-21
- “Multi-pass Texture Effects” on page 4-22

4.2.1.1 Texture Formats

Because AGP allows high bandwidth for texture execution, and virtually unlimited storage potential (based on the amount of system RAM available) the application developer is no longer limited to small 8-bit palettized textures. There is a whole new world of texture formats which can be experimented with to increase the look and feel of the application. These texture formats are discussed below:

16-bit RGB(A)	Using 16-bit RGB(A) textures is highly recommended because it frees the application from dependence upon the single hardware palette and it allows for a wider span of colors in each texture. Frequent changes of the hardware palette can put a slight strain on the overall performance.
8-bit YUV & 16-bit AYUV	Using YUV textures may provide the user with a new look. When using 8-bit YUV 4:2:2 texture format, storage space is minimized. Also the textures can be input as 16-bit YUV(A) with more colors and more intensities of color as well as alpha. An advantage of YUV is that 8 bits can be represented without the overhead of a palette. YUV is a format that favors the human eye’s sensitivity to color because it compresses the chrominance and luminance of a color rather than a degradation between colors. Usually the outcome is a picture which has kept its detail but which is slightly different in color values than the original. Like RGB(A), YUV(A) allows for a much larger range of colors than does a palette.
1, 2, 4, & 8 bit palettized	Sometimes it is good to use a palette for a texture because only a small amount of colors are employed. For instance, if the sky is mainly shades of blue mixed with white, a 4-bit palette could work very well. Because the palette is kept in the hardware, it is not as easy to animate palettes as it is in software. Every time the palette is changed, there is a change in state which causes a performance decrease. This performance decrease is estimated at about 1% if there are 30 palette changes in a frame with 10,000 triangles with full features on.
Live Video Capture	Live Video Capture can be used for a surface texture when combined with 2D Chroma Keying. It might make an astonishing effect if a game incorporated live input of the game player as they are playing the game!

4.2.1.2 Texture Sizes

Intel740 supports texture sizes ranging from 1024 x 1024 all the way down to 1 x 1 and any power of two-sided rectangle in between. It is recommended that a few large surface areas take advantage of the large map sizes to show-off this ability where it counts, such as when a background landscape is shown, or to get high resolution detail of a painting. It is best to balance the usage of large and small texture maps to object surfaces that can best utilize them without taking up memory resources when it is not necessary to have that large of a texture.

4.2.1.3 Texture Storage

Intel740 is optimized for texture storage in AGP memory because it allows simultaneous throughput of up to 533 Mbyte/s of textures with the 800 Mbyte/s of local video memory which may contain the display, render and Z-buffer. This equates to 1.3 Gbyte/s total throughput. This is a great advantage over non-AGP graphics accelerator hardware which must keep all the textures in local video memory equating to significantly fewer textures and less local video memory bus bandwidth because it has to share with display, frame and Z-buffers. It is not possible to store textures in local video memory on the Intel740. With DirectX it is easy to allocate a texture in AGP memory (also known as non-local memory) as it is in local video memory.

4.2.1.4 Animated Texture Effects

There are many strategies which can be used in animating textures. Each is described below:

UV Coordinates	One way to animate a texture is to change the texture U, V coordinates as they map on to the vertices for each frame. This method of animation is extremely fast since it does not cause any change of state for the hardware and therefore does not cause any performance degradation.
Texture Frames	Several frames can be loaded into AGP for one object and then the object's texture pointer can be changed to cycle through the different textures and give the effect of the textures changing. The drawback is that extra storage space is needed although with more space available for texture storage due to AGP, storing more textures is not a problem.
Specular Lighting	By changing the specular highlighting values along each vertex over time, a change in lighting patterns over an object to simulate water or flickering lights can be produced. Intel740 also allows the Specular Color value to be any R.G.B. color, which means that the colors could be animated to get different effects.
Fogging	As with Specular Highlight animation, Fogging values can be varied over time to produce new and unusual effects such as a whale jumping out of the water and the fog comes off of its body as it hits air and could be replaced with more shininess (specular highlights).
Alpha Blending	By changing the blending factors over time for each frame, the texture can appear more opaque or more translucent over time. This could allow for an effect such as a figure starting out as a ghost object and becoming more visible over time.
Procedural Textures	A procedural texture is one where texel values are changed between renders by a mathematical formula to produce such effects as ripples in water. When creating the texture surface in DirectX, the user needs to specify the DDSCAPS_3DDEVICE so that the surface will not be tiled if using AGP non local video memory. The texture can be written on by locking the texture surface and getting a pointer to it. Then the user can

traverse the texture memory space and apply their changes. This is a great way to represent fire or water in a texture and utilizes the extra CPU cycles while scene rendering is being done by the hardware. Textures should be stored in AGP memory to take advantage of the Direct Memory Execution (DME) abilities of the Intel740.

4.2.1.5 Multi-pass Texture Effects

There are a few more texture effects worth mentioning that can be obtained with multi-pass algorithms. Multi-pass means that the scene is rendered twice for each frame, hence causes a 2x slower performance. The different effects are listed below:

- | | |
|------------------------|---|
| Z-Buffer Shadowing | First the camera must view the scene from the point of view of a light source. The scene is rendered using Z-buffering. On the second pass, the scene is rendered from the real point of view of the camera, and then the old Z-buffer values are read with a color altering algorithm which is applied to the pixel values being rendered at the same x, y location, thus creating a shadow. |
| Dual Texture Rendering | The first scene is rendered with textures in their correct places, then the second scene is rendered with a common pattern (possibly using one of the animated texture techniques) such as a translucent lighting effect. In this way underwater rocks, plants, and animals could all appear to be affected by the same light patterns. |

4.2.2 Software Strategies

This section describes how to optimize applications which take advantage of the many features of the Intel740. Topics of discussion include:

- Z-Buffering
- Antialiasing
- Minimizing State Transitions
- Dynamic AGP Buffer Placement
- Texture Palettes
- Using Mipmapping
- Designing for Sub Pixel Accuracy
- Using Color/Chroma keying Over Alpha Blended Textures
- Avoiding Stippling Errors
- Avoiding Flipping Errors
- Texture Sorting is Not Required

4.2.2.1 Z-Buffering

The Intel740 performs all of an application's 3D depth compare in the hardware. This means that the hardware will correctly write all of the triangles in the scene as they overlap, without the need for breaking them up into smaller triangles or expensive sorting algorithms. What the programmer must remember is that if the polygons (triangles) are sorted from back to front in the application and then sent to the hardware with the Z-buffering enabled, this will give worse case results because the hardware Z-buffer algorithm checks each pixel in an x, y position against the last, and if it is in front of the last according to its z value, it will write over it. It is best not to sort at all if the Z-buffer is enabled. However, enabling anti-aliasing or alpha blending requires that the triangles be sorted from back to front. In this case Z-buffering may cause a performance hit which becomes a trade-off for rendering any intersecting triangles properly.

The Intel740 supports a 16-bit Z-buffer. Sometimes an application's scene depth complexity will cause rounding of the z bits resulting in unwanted tears along some polygons. To alleviate this problem the user could disable Z-buffering for background items and render them first. Another solution is to make the scene's z coordinates fit within a 16-bit range.

4.2.2.2 Antialiasing

It is very easy to implement anti-aliasing. Simply enable it. Sort the polygons/triangles back to front, and render the scene. The user should be cautioned to use anti-aliasing sparingly as it causes a performance slow down. The user should also note that anti-aliasing and Z-buffering can both be enabled at the same time. One strategy for rendering a scene with anti-aliasing and Z-buffering acceleration would be to render the background separately without anti-aliasing or Z-buffering enabled, then sort back to front the forefront items, enable both anti-aliasing and Z-buffering and then render the rest of the scene.

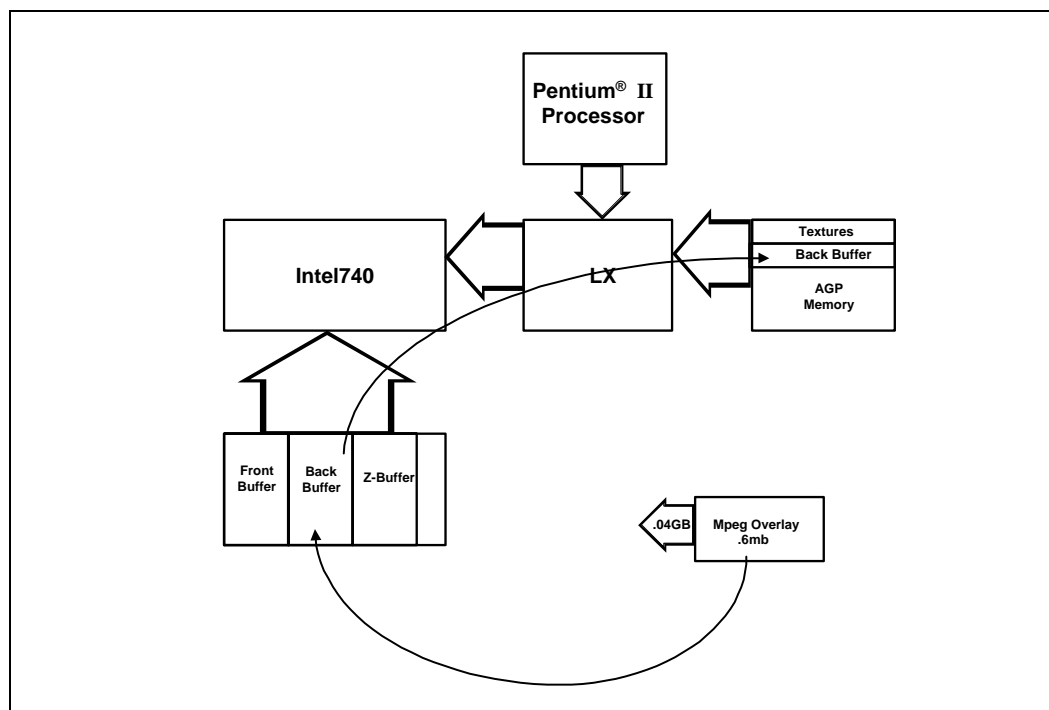
4.2.2.3 Minimizing State Transitions

It is encouraged that as much of the features of the Intel740 be utilized during the execution of a 3D program as is needed to achieve the maximum visual effect. There is little overhead for enabling all of the Intel740 3D features with the exception of alpha blending and anti-aliasing which should only be enabled as needed. Each time a feature is enabled or disabled, a state change must take place within the hardware. State changes cause a slight decrease in overall bandwidth and so causes a slight performance hit. Best performance will be ensured if triangles to be rendered are ordered according to their state or the set of features they have enabled. For the most part, state changes do not affect the Intel740. The only state changes which cause a pipeline flush are palette and stippling changes.

4.2.2.4 Dynamic AGP Buffer Placement

The Intel740 supports dynamic AGP Buffer Placement. Alternate buffers can be relocated from local video memory into AGP memory when necessary to allow full functionality. When there is two Mbytes of local video memory, at 640 x 480 x 16 the front buffer, back buffer and Z-buffer can all be placed in local video memory. When the resolution is changed to 800 x 600 x 16 or higher, then the back buffer can be relocated to AGP memory. The Intel740 supports rendering to the back buffer in AGP memory. Putting the back buffer into AGP memory can free up local video memory for MPEG Overlay as well.

Figure 4-18. Dynamic AGP Buffer Placement



4.2.2.5 Texture Palettes

It is best not to use palettized textures, because the Intel740 supports many formats of ARGB, YUV and AYUV, which allows more colors without the overhead of palette loads and changes. To use palettized textures, minimize changes in palettes. The hardware only supports one palette and to change it requires a state change and a pipeline flush, which slows overall performance. There are ways to combine many texture palettes into one with the use of a tool such as Debabelizer* which can find a common palette among many textures. It is best to use texture formats that require no palette at all.

4.2.2.6 Using Mipmapping

An application not only increases visual quality but can also increase performance of the application by using mipmapped textures. When an object becomes very small or distant and it has a large texture map associated with it, the ratio of texel look-up to texels used in rendering can be 8 to 1 because the Intel740 drivers are acquiring 16 bytes from a section of the texture map but only 2 bytes are actually being rendered. Mipmapping will give a 1 to 1 ratio of texture texels read from an image to those texels rendered in the scene. Mipmapping usually improves overall application performance by at least 10%.

Mipmapping provides better looking graphical representation of a scene by allowing the user to create various texture maps, which the hardware will choose to map onto the object based on how far the object is from the viewer. So if a scene has a patterned texture which is mapped onto an object, the user would want to create variations of that pattern which would get smaller and smaller to correspond with each mipmapping level. The user then sets a pointer to each level of mipmap so that the hardware will choose the correct texture based on the distance from the viewer. The Intel740 supports tri-linear mipmapping for added visual quality.

4.2.2.7 Designing For Sub Pixel Accuracy

Users of the Intel740 need to make sure that their polygon meshes are created so that no triangle sides are disjointed and to avoid triangle sides with “T”s because the Intel740 sub pixel accuracy for texture mapping will magnify this type of application error. This means that “floating” geometry pieces made to intersect other pieces as part of the scenery are highly discouraged. It is also important to make sure that clipping algorithms are working correctly and that all edges of adjoining polygons are free of gaps.

4.2.2.8 Using Color/chroma Keying Over Alpha Blended Textures

When using both alpha blending and chroma/colorkeying together in a rendered frame there are some renderstates which must be enabled to ensure that all textures are rendered properly. Use the following DirectX render states:

```
SetRenderState(D3DRENDERSTATE_ALPHATESTENABLE, TRUE);  
SetRenderState(D3DRENDERSTATE_ALPHAFUNC, D3DCMP_NOTEQUAL);  
SetRenderState(D3DRENDERSTATE_ALPHAREF, 0);
```

At the same time, chroma/color keying should also be enabled using the DirectX function, SetColorKey() and setting the dwColorSpaceLowValue and dwColorSpaceHighValue properly. Remember that for color keying, both values should be the same color palette index value, and for chroma keying, the both values should be the same value for high and low as to how the RGB has been defined.

4.2.2.9 Avoiding Stippling Errors

Some developers have set D3DRENDERSTATE_STIPPLEENABLE to TRUE which sets the default value of 0 to be set for all stippled patterns from D3DRENDERSTATE_STIPPLEPATTERN00 to D3DRENDERSTATE_STIPPLEPATTERN31. The result of enabling stippling without setting any values will be a black screen since all of the values are by default set to zero. If developers are not going to be using stippling, they should not use this render state at all. If they are going to use stippling, they should be sure to set the stippling values for all the D3DRENDERSTATE_STIPPLEPATTERNXX. When stippling is not to be in use, developers should make sure to set D3DRENDERSTATE_STIPPLEENABLE to FALSE.

4.2.2.10 Avoiding Flipping Errors

When using the DirectX API, it is important to always use the BeginScene and EndScene calls at the beginning and end of each frame to be written. These calls ensure that flipping errors such as blanking screens do not occur.

4.2.2.11 Texture Sorting Is Not Required

With the Intel740, the user does not have to sort textures because even though changing the texture pointer is a state change, it does not cause a pipeline flush and will not noticeably slow down the rendering. The application would be much slower at sorting textures than the Intel740 would be at swapping handles. If texture sorting for static geometry can be done once to affect many frames, it might be useful to do so. If palettized textures are used, a performance hit may result because each pixel written could change palettes many times when relying on hardware Z-buffering for sorting. Because hardware Z-buffering will always be faster than software sorting algorithms, it is recommended that the user move toward RGBA or YUV textures, which will not have a performance impact.

4.3 OpenGL Performance Guidelines

OpenGL implementations must be complete. They must support all commands and parameters as specified in *The OpenGL Graphics System: A Specification*. Typically this requires the OpenGL implementation to be divided between the CPU and the graphics subsystem, in varying degrees according to the operations involved and the functionality and performance of those system components. This characteristic of OpenGL implementations is desirable because the application is not required to understand the division of labor (and its resultant performance).

In many instances, the performance of a software implementation cannot be tolerated because minimum frame rates cannot be attained. This document specifies which functions/features of OpenGL V1.1 will be hardware-accelerated (vs. performed in software or require software rasterization) by the Intel740 OpenGL implementation. By using accelerated features and avoiding software rasterization, a developer can gain some assurance that the application will run at a high level of performance. An application still needs to be tuned to ensure the highest level of performance. That the Intel740 OpenGL implementation is “complete” and contains all the required functionality.

4.3.1 OpenGL Feature Classification

For the Intel740 OpenGL implementation, OpenGL “features” fall into three categories:

1. Features supported directly by graphics hardware (such as setup and most per-fragment operations). These features are rated “G” for “graphics accelerated.”
2. Features not supported by graphics hardware which would require software rasterization (such as stencil operations). These features are rated “X” for “excluded from the performance set.” Obviously these features should be avoided or, at best, used sparingly.
3. “CPU-supported” features (geometry, lighting, display lists, etc.) which, although not particularly accelerated by graphics hardware, are likely to provide a level of performance equal to (or greater) than similar functions performed in the application. These features are considered “PG” for “Programmer’s Discretion is Advised” since their use is not necessarily detrimental to performance. This rating is also used for “hybrid” (software + hardware) support.

Note: The programmer must consider all the pertinent state variables in order to understand what will be hardware accelerated — a single mode might preclude acceleration of all primitive rasterization.

Note: In some situations the definition of “hardware acceleration” can become ambiguous, since some features may require a hybrid solution (a mixture of software and hardware algorithms).

4.3.2 Feature Overview

The following table lists (at a high level) the rating of the OpenGL features.

Table 4-6. Rating OpenGL Features (Sheet 1 of 2)

Function	Rating†	Comments
Pixel Formats		
RGBA	G	
Color Index	X	
Vertex Specification		
Begin/End	PG	
Vertex Array	PG	
Evaluator	PG	
Model-view Transform	PG	
Lighting	PG	
Texture Generation	PG	
Texture Transform	PG	
User Clip Planes	PG	
Projection Transform	PG	
View Volume Clipping	PG	
Perspective Divide	PG	
Viewport Transform	PG	
Current Raster Position	PG	
Pixel Operations	X	
Point Rasterization		
Width	PG	
Anti-aliasing	X	
Line Rasterization		
Width	G/PG	G: Width = 1.0
Smoothing	G	
Stippling	G/X	G: for trivial patterns
Polygon Rasterization		
Culling	G	
Stippling	G	
Smoothing	G	
Fill Mode	G	
Point Mode	PG	
Line Mode	G	
Depth Offset	PG	
Pixel Rectangles / Bitmaps	G/X	G: simple copy operations

† “G” indicates “graphics accelerated”, “X” indicates “excluded from the performance set”, “PG” indicates “Programmer’s Discretion is Advised”. See also Section 4.3.1.

Table 4-6. Rating OpenGL Features (Sheet 2 of 2)

Function	Rating†	Comments
Texturing		
TexImage*	PG	
CopyTex	PG	
TexSubImage	PG	
CopyTexSubImage	PG	
Wrap	G	
Bilinear Filtering	G	
Trilinear Filtering	X	
Border	X	
Texture Objects	PG	
Replace, Modulate, Decal Modes	G	
Blend Mode	X	
Fog		
Per-Vertex	G	
Per-Pixel	X	
Per-Fragment Operations		
Pixel Ownership	X	when drawing to occluded front buffer
Scissor	X	
Alpha Test	G	
Stencil	X	
Depth Buffer Test	G	
Blending	G	Note: No destination alpha buffer with depth buffer
Dithering	G	
Logical Op	X	except for trivial operations
Whole Frame Buffer Operations		
FRONT_AND_BACK	PG	driver must draw twice
Stereo Buffers	n/a	Not supported
Auxiliary Buffers	n/a	Not supported
Buffer Masks	G/X	X: different R,G,B,A masks
Clear	G	
Accumulation Buffer	PG	accumulation performed in software
Read Pixels	PG	
Copy Pixels	G/X	G: simple copies
Selection	PG	
Feedback	PG	
State Requests	PG	

† “G” indicates “graphics accelerated”, “X” indicates “excluded from the performance set”, “PG” indicates “Programmer’s Discretion is Advised”. See also Section 4.3.1.

Note: The remainder of this chapter is structured as an “annotation” of the OpenGL V1.1 specification and specific extensions. Only performance notes will be discussed and included here, so one probably needs to read this document alongside the OpenGL specification.

4.3.3 OpenGL Operation

The following sections describe the classification of OpenGL features.

4.3.3.1 Begin/End Paradigm

There are no primitive (object) types excluded from hardware acceleration. Points, line segments, line segment loops, separated line segments, polygons, triangle strips, triangle fans, separated triangles, quadrilateral strips, and separated triangles are all candidates for hardware acceleration. This includes the specification of polygon edge flags.

4.3.3.2 Vertex Specification

All vertex and associated auxiliary data specifications are included in the performance set, with the following exceptions:

Since color index mode is not supported. Index specification is not of particular interest

4.3.3.3 Vertex Arrays

Vertex array specification is included in the performance set, and is the preferred means to describe objects with a large number of vertices.

4.3.3.4 Rectangles

Rectangles are included in the performance set.

4.3.3.5 Coordinate Transformation

The Intel740 does not provide hardware acceleration for transformations, although vertex, normal, and texture coordinate transformations are supported and optimized for the target platform. These operations are therefore rated PG.

Application designers wishing to perform these operations internally are referred to the “OpenGL Correctness Tips” provided in the *OpenGL Programming Guide*; directions are given to allow 2D rasterization specification. Note that the viewport transformation is always enabled and thus must be set to properly generate the proper window coordinates.

4.3.3.6 Clipping

The Intel740 OpenGL implementation does not provide hardware acceleration for view-volume or client clip plane clipping. These operations will require a software clipping stage prior to rasterization.

4.3.3.7 Current Raster Position

Not all operations which rely on the current raster position are hardware accelerated.

4.3.3.8 Colors and Coloring

The Intel740 does not provide hardware accelerated lighting operations. Although lighting is supported, applications wishing to perform these operations internally should ensure that lighting is disabled in OpenGL.

Both flat shading modes (SMOOTH and FLAT) are supported by the Intel740 hardware.

4.3.4 Rasterization

4.3.4.1 Antialiasing

Line and polygon smoothing is supported by the Intel740 hardware.

4.3.4.2 Points

Aliased points are rendered by the Intel740 hardware using short lines or triangles. Antialiased points will require software rasterization.

4.3.4.3 Line Segments

Only unit-width aliased or anti-aliased lines are supported by the Intel740 hardware. Stippled and/or wide lines are not supported by the hardware and will require a software or hybrid rasterization phase.

4.3.4.4 Polygons

Polygon culling is supported by the Intel740 hardware, as are stippled and/or anti-aliased polygons.

FILL and LINE polygon modes are supported by the Intel740 hardware. Depth offset is not directly supported by the hardware, but does not require software rasterization.

4.3.4.5 Pixel Rectangles

Pixel rectangles are not supported by the Intel740 hardware and will require software rasterization.

4.3.4.6 Bitmaps

Bitmaps are not supported by the Intel740 hardware and will require software rasterization.

4.3.4.7 Texturing

All texture mapping functions are supported by the Intel740 hardware, with the following exceptions:

- Border colors are ignored (textures are clamped to the edges)
- BLEND texture function requires software rasterization

4.3.4.8 Fog

The Intel740 hardware supports linear interpolation of the fog factor. Setting the FOG_HINT to NICEST when EXP or EXP2 modes are selected will require software rasterization.

4.3.4.9 Antialiasing Application

Line and polygon smoothing is supported by the Intel740 hardware.

4.3.5 Fragments And The Frame Buffer

4.3.5.1 Per-Fragment Operations

The following table defines which pre-fragment operations are included or excluded from the performance set.

Table 4-7. Included and Excluded Pre-Fragment Operations

Operation	Rating [†]
Pixel Ownership	X (when drawing to an occluded front buffer)
Scissor	X
Alpha Test	G
Stencil	X
Depth Buffer Test	G
Blending	G, though destination alpha buffer is not supported
Dithering	G
Logical Operation	X

[†] “G” indicates “graphics accelerated”, “X” indicates “excluded from the performance set”, “PG” indicates “Programmer’s Discretion is Advised”. See also Section 4.3.1.

4.3.5.2 Whole Framebuffer Operations

Drawing to the FRONT_AND_BACK will require two rasterization passes (internal to the OpenGL implementation). Stereo and auxiliary buffers are not supported.

Use of ColorMask should be limited to enabling or disabling all the color components concurrently. Software rasterization will be required if only some of the color components masked.

Those “whole frame buffer” operations related to stencil or accumulation buffers will require software rasterization.

4.3.5.3 Drawing, Reading, and Copying Pixels

Only “pure” copy pixel operations are hardware accelerated. Pixel reads will be performed in software.

4.3.6 Special Functions

The special functions (listed below) are all performed by the CPU and are therefore rated “PG.”

- Display lists
- Flush and Finish
- Evaluators
- Selection
- Feedback

4.3.7 State And State Requests

All of the state request commands are performed in software are therefore rated “PG.”

4.3.8 GL Command Summary

The following table provides “performance ratings” on a per-command basis, with notes on parameter settings.

Table 4-8. Command Performance Ratings (Sheet 1 of 5)

Command/Feature	Rating†	Comment/Exception
glAccum	PG	
glAlphaFunc	G	
glAreTexturesResident	PG	
glArrayElement	PG	
glBegin	PG	
glBindTexture	PG	
glBitmap	X	
glBlendFunc	G	
glCallList	PG	
glCallLists	PG	
glClear	G	
glClearAccum	X	
glClearColor	G	
glClearDepth	G	
glClearIndex	X	color index not supported
glClearStencil	X	
glClipPlane	PG	requires software clipping
glColor	PG	
glColorMask	G/X	G: only when all channels are masked or enabled together

† “G” indicates “graphics accelerated”, “X” indicates “excluded from the performance set”, “PG” indicates “Programmer’s Discretion is Advised”. See also Section 4.3.1.

Table 4-8. Command Performance Ratings (Sheet 2 of 5)

Command/Feature	Rating†	Comment/Exception
glColorMaterial	PG	
glColorPointer	PG	
glCopyPixels	G/X	G: for simple copies
glCopyTex*	G/X	G: for simple copies
glCullFace	G	
glDeleteLists	PG	
glDeleteTextures	PG	
glDepthFunc	G	
glDepthMask	G	
glDepthRange	PG	
glDisable	-	see glEnable
glDisableClientState	PG	
glDrawArrays	PG	
glDrawBuffer	G/PG	G: only NONE, FRONT or BACK
glDrawElements	PG	
glDrawPixels	G/X	G: for simple copies
glEdgeFlag	G	
glEdgeFlagPointer	PG	
glEnable		
*_ARRAY	PG	
ALPHA_TEST	G	
AUTO_NORMAL	PG	
BLEND	G/X	X: destination alpha buffer not supported
CLIP_PLANEi	PG	
COLOR_MATERIAL	PG	
CULL_FACE	G	
DEPTH_TEST	G	
DITHER	G	
FOG	G/X	X: when FOG_HINT == NICEST and not LINEAR fog
LIGHTi	PG	
LIGHTING	PG	
LINE_SMOOTH	G	
LINE_STIPPLE	G/X	G: trivial patterns
*_LOGIC_OP	G/X	G: trivial operations
MAP*	PG	
NORMALIZE	PG	

† “G” indicates “graphics accelerated”, “X” indicates “excluded from the performance set”, “PG” indicates “Programmer’s Discretion is Advised”. See also Section 4.3.1.

Table 4-8. Command Performance Ratings (Sheet 3 of 5)

Command/Feature	Rating†	Comment/Exception
POINT_SMOOTH	X	
POLYGON_OFFSET*	PG	
POLYGON_SMOOTH	G	
POLYGON_STIPPLE	G	
SCISSOR_TEST	X	
STENCIL_TEST	X	
TEXTURE_*D	PG	
TEXTURE_GEN*	PG	
glEnd	-	
glEndList	PG	
glEval*	PG	
glFeedbackBuffer	PG	
glFinish	PG	
glFlush	PG	
glFog	G/X	X: when FOG_HINT == NICEST and not LINEAR
glFrontFace	G	
glFrustum	PG	
glGenLists	PG	
glGenTextures	PG	
glGet*	PG	
glHint	-	depends on hint
glIndex*	X	color index not supported
glInitNames	PG	
glInterleavedArrays	PG	
glIs*	PG	
glLight	PG	
glLightModel	PG	
glLineStipple	G/X	G: when solid
glLineWidth	G/X	G: when 1.0
glListBase	PG	
glLoadIdentity	PG	
glLoadMatrix	PG	
glLoadName	PG	
glLogicOp	G/X	G: when CLEAR, COPY or SET
glMap*	PG	
glMaterial	PG	

† “G” indicates “graphics accelerated”, “X” indicates “excluded from the performance set”, “PG” indicates “Programmer’s Discretion is Advised”. See also Section 4.3.1.

Table 4-8. Command Performance Ratings (Sheet 4 of 5)

Command/Feature	Rating [†]	Comment/Exception
glMatrixMode	PG	
glMultMatrix	PG	
glNewList	PG	
glNormal	PG	
glNormalPointer	PG	
glOrtho	PG	
glPassThrough	PG	
glPixelMap	G/X	G: trivial operations
glPixelStore	G/X	G: trivial operations
glPixelTransfer	G/X	G: trivial operations
glPixelZoom	X	
glPointSize	G/PG	G: only for unit width
glPolygonMode	G/PG	G: when FILL or LINE
glPolygonOffset	PG	
glPolygonStipple	G	
glPopAttrib	PG	
glPopMatrix	PG	
glPopName	PG	
glPrioritizeTextures	PG	
glPushAttrib	PG	
glPushMatrix	PG	
glPushName	PG	
glRasterPos	PG	
glReadBuffer	X	
glReadPixels	X	
glRect	G	
glRenderMode	G/PG	G: RENDER; PG: SELECT or FEEDBACK
glRotate	PG	
glScale	PG	
glScissor	X	
glSelectBuffer	PG	
glShadeModel	G	
glStencil*	X	
glTexCoord	G	
glTexEnv	G/X	X: BLEND
glTexGen	PG	
glTexImage1d	PG	border colors ignored

[†] "G" indicates "graphics accelerated", "X" indicates "excluded from the performance set", "PG" indicates "Programmer's Discretion is Advised". See also Section 4.3.1.

Table 4-8. Command Performance Ratings (Sheet 5 of 5)

Command/Feature	Rating†	Comment/Exception
glTexImage2d	PG	border colors ignored
glTexParameter	G/X	X: *_MIPMAP_LINEAR and TEXTURE_BORDER_COLOR
glTextureSubImage*	PG	
glTranslate	PG	
glVertex	PG	
glVertexPointer	PG	
glViewport	PG	

† “G” indicates “graphics accelerated”, “X” indicates “excluded from the performance set”, “PG” indicates “Programmer’s Discretion is Advised”. See also Section 4.3.1.

Glossary

Accelerated Graphics Port (AGP)	A scalable architecture that increases the bandwidth available to a graphics controller and provides the performance necessary for a graphics controller to do texturing directly from system memory.
Alpha Blending	Uses a fourth color component which is not displayed but which corresponds to the opacity of a surface to control the amount of color of a pixel in the source surface to be blended with a pixel in the destination surface.
Antialiasing	An algorithm designed to reduce the stair-stepping artifacts (sometimes called jaggies) that result from drawing graphic primitives on a raster grid. The solution usually relies on the multi-bit raster's ability to display a number of pixel intensities. If the intensities of the neighboring pixels lie between the background and line intensities, the line becomes slightly blurred, and the jagged appearance is thereby diffused.
Bitmap	A representation, consisting of rows and columns of dots, of a graphics image in computer memory. The value of each dot (whether it is filled in or not) is stored in one or more bits of data. For simple monochrome images, one bit is sufficient to represent each dot, but for colors and shades of gray, each dot requires more than one bit of data. The more bits used to represent a dot, the more colors and shades of gray that can be represented.
BitPlane	A rectangular array of bits mapped one-to-one with pixels. The framebuffer is a stack of bitplanes.
Buffer	A group of bitplanes that store a single component (such as depth or red) or a single index (such as the color index).
Clipping	A three dimensional operation that reduces the number of drawing calculations the CPU makes by eliminating any objects, or portions of objects, outside the viewing area.
Depth Cueing	Reducing an object's color and intensity as a function of its distance from the observer. For instance, a bright, shiny red ball may look duller and darker the farther away it is from the observer.
Direct3D (D3D)	An Application Programming Interface (API) for manipulating and displaying 3-dimensional objects. Developed by Microsoft, Direct3D provides programmers with a way to develop 3-D programs abstracted from the hardware layer, but which can utilize 3-D capabilities of the underlying graphics accelerated hardware.
DirectDraw	Microsoft's new 2D library of graphics API's, enabling access to hardware's Blitting, clipping and flipping capabilities. DirectDraw provides low-level access to the frame buffer and advanced features of the display adapter.

DirectDraw Video Port Extension (VPE)	Microsoft's extension of DirectDraw to control the flow of data from a hardware video port device to a DirectDraw surface in video memory. As the VPE specification is finalized, it will be merged with the rest of the DirectDraw documentation.
Direct Memory Execution (D.M.E.)	Utilization of the entire AGP bandwidth through deep buffering and 2x side band signaling with write combining which provides the highest sustained data transfer rates across AGP.
Double Buffering	The process of using two frame buffers for smooth animation. Graphical contents of one frame buffer are displayed while updates occur on the other buffer. When the updates are complete, the buffers are switched. Only complete images are displayed, and the process of drawing is not shown. The result is the appearance of smooth animation.
Fogging	The alteration of the visibility of what is seen, depending on how far one is from the object.
Frame Buffer	A block of graphics memory that represents the display screen.
GDI	The Windows Graphics Device Interface, a library of video display and printer functions for 2D graphics.
Gouraud shading	Smooth interpolation of colors across a polygon or line segment. Colors are assigned at vertices and linearly interpolated across the primitive to produce a relatively smooth variation in color.
H.324	New communications standard for sharing video, voice and data over a single analog telephone line.
HAL	Hardware Abstraction Layer. A specification of a graphics hardware's functionality. Generally implemented into a device driver software program.
Hyper Pipelined Architecture	An architecture designed so that many operations are executed in parallel to improve performance.
I2C*	I2C* (Inter-Integrated Circuit) is a two-wire serial bus/protocol. A serial clock line (IICCLK) and serial data line (IICDAT) are used to transfer data between a bus master and a slave device. The maximum data rate is 100 Kbits/s. A slave may slow down the bus by inserting wait states. In the Intel740 a single bus master can be implemented by using two of Intel740's GPIO pins; one for IICCLK and one for IICDAT. Multiple slaves can be connected to this system (e.g., a TV tuner, video decoder, and digital TV encoder). However, only one I2C* master is allowed (Intel740). The timing for the I2C* is derived from Intel740's PCI clock.
Lighting	A mathematical formula for approximating the physical effect of light from various sources striking objects. Typical lighting models use light sources, an object's position & orientation and surface type.

Mipmapping	When viewing a distant texture-mapped object in a 3D world, many texels make up each pixel seen on the screen, causing the textures to often appear aliased or distorted, if point sampling, the most common texture-mapping technique, is used. Mipmapping solves that problem by precomputing (that is, prefiltering) different levels of detail of your texture image, and accessing the appropriate level according to the object's distance from the camera. For example, a texture image which is 16x16 texels, will have four more mipmaps at lower resolutions, 8x8, 4x4, 2x2 and 1x1. Bilinear mipmapping chooses the closest mipmap image to your pixel's level of detail, then performs a bilinear interpolation upon that texture image to get the color value for the pixel. Trilinear mipmapping requires over twice the computational cost, as it chooses the two closest mipmaps, performs a bilinear interpolation on each, then averages the two results to arrive at the final screen pixel value.
MMX™ Technology	A set of 57 multimedia instructions built into Intel's newest microprocessors. MMX™ Technology-enabled microprocessors can handle many common multimedia operations, such as digital signal processing (DSP), that are normally handled by a separate sound or video card. However, only software especially written to call MMX™ Instructions can take advantage of the MMX Instruction set.
Pixel	Short for picture element. The bits at location (x, y) of all the bitplanes in the framebuffer constitute the single pixel (x, y). It is the smallest discrete unit of a computer or TV tube that can be assigned a specific color, the "dots" that make up TV and computer screen pictures.
POTS Video	Low cost video conferencing over Plain Old Telephone Service (POTS).
Raster	A rectangular grid of picture elements, or pixels. The graphical data to be displayed on the raster is stored by the frame buffer. Raster operations can be performed on some portion or all of the raster. Such operations aid in the efficient handling of blocks of pixel data.
Rendering	The process of computing a graphical model's surface qualities, such as color, shading, smoothness, and texture, and creating a raster image.
Setup	Stage responsible for the precalculation of various derivatives used by inner loops of rendering algorithms.
Shading	The process of interpolating color within the interior of a polygon, or between the vertices of a line, during rasterization.
Texel	A texture element. A texel is obtained from texture memory and represents the color of the texture to be applied to a corresponding fragment.

Texture antialiasing	Bilinear or trilinear filtering. Also known as sub-textel positioning. If a pixel is between texels, the program chooses the color of the pixel by averaging the adjacent texels' colors instead of assigning it the exact color of one single texel. Without bilinear or trilinear filtering, the texture gets very blocky up close as multiple pixels get the exact same texel coloring, while the texture shimmers at a distance because small position changes keep producing large texel changes.
Texture mapping	The process of superimposing a 2-D texture or pattern over the surface of a 3-D graphical object. This is an efficient method of producing the appearance of texture, such as that of wood or stone, on a large surface area.
Three Dimensional Graphics	The display of objects and scenes with height, width, and depth information. The information is calculated in a coordinate system that represents three dimensions via x, y, and z axes.
VxD	Virtual Device Driver.
WDM	Win32 Driver Model (WDM) provides a common set of I/O services and binary-compatible device drivers for both Windows NT and future Windows operating systems. WDM will maximize system responsiveness and throughput by providing extremely low services and fewer ring transitions that interactive applications demand. All WDM drivers execute in Ring 0 and have access to low latency services. For backward compatibility, a Windows virtualization driver can be implemented to interface a hardware-specific legacy application to WDM.
Z-buffer	The depth buffer in 3-D graphics. The z-buffer memory locations, like those in the frame buffer, correspond to the pixels on the screen. The z-buffer, however, contains information relating only to the z-axis (or depth axis). The z-buffer is used in hidden surface removal algorithms, so that for each pixel written, the depth of the pixel is stored in the z-buffer. When subsequent objects attempt to draw that pixel, that object's z value is compared with the number in the z-buffer, and the write is omitted if the object is farther away from the eye.