Real-time, 3-D Graphics in Linux[1]

Robert Geist, Dante Treglia Jr., James Westall
Department of Computer Science, Clemson University

Abstract

The present generation of accelerated PC graphics adapters are capable rendering complex 3-D scenes in real-time. However, software support for these adapters has been slow to evolve in Linux. In this paper we describe our experiences in developing a software system that provides accelerated graphics support to the Mesa graphics library. The performance of accelerated Mesa is compared with that of commerically available Windows NT software running on the same hardware and with mid-range and high-end SGI workstations.

1 Introduction

Until recently, real-time, 3-D graphics with sophisticated use of lighting and texture mapping has been the exclusive domain of special-purpose, high-cost, graphics systems. Most notable among such systems are those from Silicon Graphics, Inc. As SGI's OpenGL software interface to graphics hardware has emerged as an industry standard, designers of PC graphics adapters have begun to implement components of the OpenGL rendering pipeline in hardware. Such adapters now provide high-end graphics performance within the budgetary constraints of PC owners.

To obtain maximum performance from accelerated graphics adapters, a relatively large body of hardware-specific software must be developed and integrated with existing system software. Depending upon the availability and quality of the documentation for the target graphics adapter, developing this new software may be quite difficult, and system integration may prove to be more challenging still. Furthermore, product lifecycles of accelerated graphics adapters are now very short, and long term viability of specific product lines is often uncertain. For these and perhaps other reasons support for high-end graphics has been slow to evolve in Linux.

Nevertheless, the potential appeal of an inexpensive, high-end graphics enabled Linux workstation is undeniable, and we believe that such a machine would be capable of effectively competing with the Windows NT based solutions that are now widely available. Accordingly, we have undertaken a multi-year effort to investigate design alternatives and develop prototypes of such a system. This paper provides an account of that effort.

The remainder of the paper is organized as follows. In section 2 we describe the hardware and software platforms used in this work. In section 3 we discuss the functionality, design, and implementation of the kernel mode device driver. In section 4 we discuss the implementation of an accelerated graphics back-end for the Mesa graphics library. In section 5 we provide performance measurements and comparison. In section 6 we discuss alternative strategies for integration or interoperation of accelerated Mesa with the X Window system and identify ongoing related work in this area. Conclusions follow in section 7.

2 Platforms

Two PC platforms were used in this work. The system *glint* is a Micron Millennia Pro2 with dual 200 MHz Pentium Pro processors and 128MB of main memory. The graphics adapter is an Omnicomp Pro88 which uses the GLINT

[1]This work is supported by grant #NAG2-1107 from NASA Ames Research Center

500TX and Delta chips from 3Dlabs. The hardware rendering pipeline of the GLINT 500TX, and its successor the 500MX, are modeled after Silicon Graphics' OpenGL rendering pipeline [6]. They support many standard graphics operations in hardware, including z-buffering, texture-mapping, alpha-blending, fog, clipping, anti-aliasing, and vertex color and depth interpolation. The Pro88 has an 8MB frame buffer and an 8MB local buffer that is used for texture and depth. The complete system cost approximately $5,000 in 1997.

The other system, *glint2*, is a Gateway 2000 with a single 300 MHz Pentium II processor and 64MB of memory. It is equipped with a Leadtek WinFast L2300 graphics adapter. The L2300 is based upon the more tightly integrated and less expensive Permedia 2 from 3Dlabs. The Permedia 2 has an 8MB unified frame, texture, and depth buffer. This system cost approximately $2,000 in 1998. Our software development environment is Linux 2.0.34 in a Red Hat 5.1 distribution.

3 Device Driver Design

From the outset, our objective was to provide an OpenGL API at the application level. Mesa, the open source, OpenGL-work-alike, graphics library developed by Brian Paul [3] was a natural choice to build upon. In its default configuration, Mesa implements the entire rendering pipeline in software and relies only upon the availability of a simple DrawPixel() primitive to update the frame buffer. This approach is simple and portable but slow.

To take advantage of the hardware acceleration of the graphics board, it is necessary to develop code that intercepts the rendering as early as possible in the Mesa pipeline and passes high-order, device-dependent rendering primitives directly to the board. We felt that we would obtain a substantial benefit in dividing this new code into two distinct components: the Mesa back-end code which generates the board-dependent rendering primitives; and a kernel mode device driver whose responsibilities were initializing the graphics adapter and managing DMA data transfers. Reliance upon a standalone graphics device driver is historically a non-standard approach in the Unix world. For example, most X servers and the *svgalib* graphics library are completely self- contained.[2] Nevertheless, we felt that several specific benefits would be derived from this approach:

- Security concerns associated with graphics servers or applications having to run with root privileges are eliminated.
- Performance of double buffered real-time graphics can be improved through the use of interrupt driven DMA transfers.
- Integration with other software systems can be expedited. For example, by building upon our driver, we created an X server for the GLINT TX500 in only three days using fewer than 300 lines of new code.

Our GLINT and Permedia 2 device drivers share a common framework and are implemented as installable kernel modules. They operate as character devices, providing an *ioctl()* based API, and perform two basic functions. The first of these is effecting transitions between VGA and high resolution graphics video modes. This part of the drivers is small (fewer than 200 lines of code) but extremely hardware dependent and difficult to write. There is almost no common code here between the drivers for the related GLINT and Permedia 2 chip sets. The second function, managing the DMA transfer of graphics primitives, is much more general in nature and common code is used in both drivers.

[2]However, reports indicate that this approach may now also being used in the development of XFree86 4.0 [7]

3.1 DMA

DMA enhances performance by enabling overlapped operation of the graphics adapter and the CPU. In typical operation, the graphics adapter downloads a DMA buffer and executes the primitives therein while the CPU is busy filling another DMA buffer.

Various DMA buffer management strategies are reasonable depending upon one's objectives. Since we sought to achieve the maximum possible performance, we used a very simple strategy that minimizes buffer management overhead, avoids all user to kernel space copying, but retains flexibility in configuring buffer sizes and number of buffers. DMA buffers are allocated by the device driver at the time the driver module is installed. The maximum amount of physically contiguous memory that can be obtained with *kmalloc()* in a default kernel configuration is 128KB. We use buffers of size 124KB, since a page in each is lost to header information upon allocation and memory mapping requirements.

The number of buffers allocated can also affect performance. If the CPU and the graphics adapter were to perform at constant rates during an entire application, then by the standard Knuth argument [2] two buffers would suffice regardless of which device were faster. We find that real-time graphics benchmarks impose time varying loads, and so additional DMA buffers can be beneficial. We use eight. Adding buffers beyond eight does not improve the performance of any of our benchmarks, and any number of buffers in the range of four to eight has been found to provide good performance.

When an application wishes to initiate graphics operations, it must first open the graphics device and then issue the BIND_DMA *ioctl()* request. If the DMA buffer pool is not already in use, the device driver uses the kernel's *do_mmap()* function to map the buffers into the application's address space. It then returns a table of the virtual addresses of the buffers to the application.

3.2 Details of DMA Buffer Management

The DMA buffers are organized as a circular queue with two indices, *filling* and *draining*, identifying the buffers undergoing the named operations. When an application wishes to initiate the transfer of the next buffer, it makes a START_DMA *ioctl()* call passing the number of bytes in the buffer as a parameter. The driver code that manages START_DMA requests is shown here:

```
save_flags(flags);
cli();              /* disable interrupts */
if (filling == draining)
{
/* queue was empty until just now; we can re-enable */
   restore_flags(flags);
   filling = (filling + 1) % NUM_DMA_BUFS;
/* Send buffer addr to adapter ... and then start  */
/* the transfer by loading adapter count register. */
   LOAD_REGISTER(DMA_address,
        glint_board.dmabuf[draining].phys_base);
   LOAD_REGISTER(DMA_linecount,count);
   return;
}
```

```
    glint_board.dmabuf[filling].count = count;
    filling = (filling + 1) % NUM_DMA_BUFS;
    if (filling == draining)
    {
    /* queue is full; we can't return to the user */
       sleep_on(&dma_snooze);
    }
    restore_flags(flags);
    return;
}
```

The application may start filling the next buffer in the circular queue when control returns from the device driver. The call to *sleep_on()* prevents destructive over-writing of buffers not yet consumed by the adapter.

The interrupt service routine, shown below, is correspondingly simple. Thus, we believe our objective of minimizing overhead in buffer management has been realized.

```
    if (filling == draining)
    {
    /* queue was full until just now, wake-up   */
    /* any waiting process.                      */

       wake_up(&dma_snooze);
    }
    draining = (draining+1) % NUM_DMA_BUFS;
    if (filling == draining)
    {
    /* queue just emptied; there's nothing to do unless the user
       is sitting in the close routine waiting to wind it all up
    */
       wake_up(&close_snooze);
       return;
    }
    /* start the next one if there is one */
    LOAD_REGISTER(DMA_address,
          glint_board.dmabuf[draining].phys_base);
    LOAD_REGISTER(DMA_linecount,
          glint_board.dmabuf[draining].count);
    return;
}
```

4 The Mesa-Glint back-end

The function of a Mesa back-end is to intercept the rendering process as early as possible in the Mesa pipeline, construct buffers of high-order, device-dependent rendering primitives, and initiate their transfer to the graphics adapter. Mesa provides a clearly specified, nicely encapsulated collection of function pointers that can be redirected to back-end accelerated rendering routines. Thus, new back-ends can be installed without changes to the distributed Mesa source.

The GLINT hardware supports triangle-at-a-time, rather than pixel-at-a-time, primitives. Thus, most of the rendering code in our back-end constructs triangle drawing primitives. As an example, here is the accelerated routine to draw a smooth-shaded, z-buffered triangle:

```
void base_triangle_function(
GLcontext *ctx,
int v0, int v1, int v2)
{
   struct vertex_buffer *VB = ctx->VB;
   struct glint_mesa_context
          *gmc = (struct glint_mesa_context *) ctx->DriverCtx;
   unsigned long *dma_buf_ptr;

   dma_buf_ptr = get_dma_lines(gmc, 23);
   *(dma_buf_ptr++)=V_RGBXYZ_MASK|INDEX_MODE|V0_FLOAT_GROUP;
   *(float*)(dma_buf_ptr++)= ((float)(VB->Color[v0][0]))/255.0; /* R */
   *(float*)(dma_buf_ptr++)= ((float)(VB->Color[v0][1]))/255.0; /* G */
   *(float*)(dma_buf_ptr++)= ((float)(VB->Color[v0][2]))/255.0; /* B */
   *(float*)(dma_buf_ptr++)= VB->Win[v0][0];                    /* X */
   *(float*)(dma_buf_ptr++)= VB->Win[v0][1];                    /* Y */
   *(float*)(dma_buf_ptr++)= (VB->Win[v0][2])/
                                      ((float)(DEPTH_SCALE)); /* Z */

   *(dma_buf_ptr++)=V_RGBXYZ_MASK|INDEX_MODE|V1_FLOAT_GROUP;
   *(float*)(dma_buf_ptr++)= ((float)(VB->Color[v1][0]))/255.0;
   *(float*)(dma_buf_ptr++)= ((float)(VB->Color[v1][1]))/255.0;
   *(float*)(dma_buf_ptr++)= ((float)(VB->Color[v1][2]))/255.0;
   *(float*)(dma_buf_ptr++)= VB->Win[v1][0];
   *(float*)(dma_buf_ptr++)= VB->Win[v1][1];
   *(float*)(dma_buf_ptr++)= (VB->Win[v1][2])/((float)(DEPTH_SCALE));

   *(dma_buf_ptr++)=V_RGBXYZ_MASK|INDEX_MODE|V2_FLOAT_GROUP;
   *(float*)(dma_buf_ptr++)= ((float)(VB->Color[v2][0]))/255.0;
   *(float*)(dma_buf_ptr++)= ((float)(VB->Color[v2][1]))/255.0;
   *(float*)(dma_buf_ptr++)= ((float)(VB->Color[v2][2]))/255.0;
   *(float*)(dma_buf_ptr++)= VB->Win[v2][0];
   *(float*)(dma_buf_ptr++)= VB->Win[v2][1];
   *(float*)(dma_buf_ptr++)= (VB->Win[v2][2])/((float)(DEPTH_SCALE));

   *(dma_buf_ptr++)= DMA_Draw_Triangle;
   *(dma_buf_ptr++)= RENDER;
}
```

Three vertex registers on the board are each loaded with R,G,B color information and X,Y,Z coordinate information; the GLINT hardware handles the color interpolation. The call to *get_dma_lines()* will block in the device driver if the requested number of words is not available in the current buffer and all buffers are currently queued to the board. Buffer flushes are initiated when a buffer fills and for user level commands such as *glClear()* and *glFlush()*.

There is no provision for texture or blending data in the function shown above. We initially used a single generic triangle draw routine and disabled the texture and blend units on the adapter when they were not needed. This approach requires 37 DMA words for each triangle and creates a measurable and unacceptable performance impact. The GLINT TX500 and Permedia 2 are capable of rendering triangle strips with only one new vertex per triangle as shown below:

```
/* Draw remainder of triangle strip... One new vtx / triangle */
for(v2=2;v2<VB->Count;v2++)
```

```
{
   v2m3 = v2%3;
   dma_buf_ptr = get_dma_lines(gmc,9);
   *(dma_buf_ptr++)=inh_rgbxyz[v2m3];
   *(float*)(dma_buf_ptr++)= ((float)(VB->Color[v2][0]))/255.0;
   *(float*)(dma_buf_ptr++)= ((float)(VB->Color[v2][1]))/255.0;
   *(float*)(dma_buf_ptr++)= ((float)(VB->Color[v2][2]))/255.0;
   *(float*)(dma_buf_ptr++)= VB->Win[v2][0];
   *(float*)(dma_buf_ptr++)= VB->Win[v2][1];
   *(float*)(dma_buf_ptr++)= VB -> Win[v2][2]/((float)(DEPTH_SCALE));
   *(dma_buf_ptr++)= DMA_Draw_Triangle_Tag;
   *(dma_buf_ptr++)= RENDER_TRAPEZOID_SPC;
}
```

Thus, for long strips, the DMA cost approaches 9 words per vertex as compared to the 23 required to draw a complete triangle. Our Mesa back-end now contains the eight distinct but similar triangle drawing routines needed to support standalone triangles and triangle strips with all four possible combinations of texture and blending.

The Mesa back-end also performs a variety of initialization and state management activities. These include: initializing adapter rendering units; establishing windows and clipping regions; clearing depth and frame buffers; swapping frame buffers for double buffered contexts; and loading texture maps and transferring them to the adapter's local buffer. As with rendering, all of these functions are performed by filling DMA buffers with adapter-specific primitives and initiating their DMA transfer. The complete Mesa back- end is now over twice as large as the device driver. It contains 1103 lines as measured by semicolon count.

5 Performance

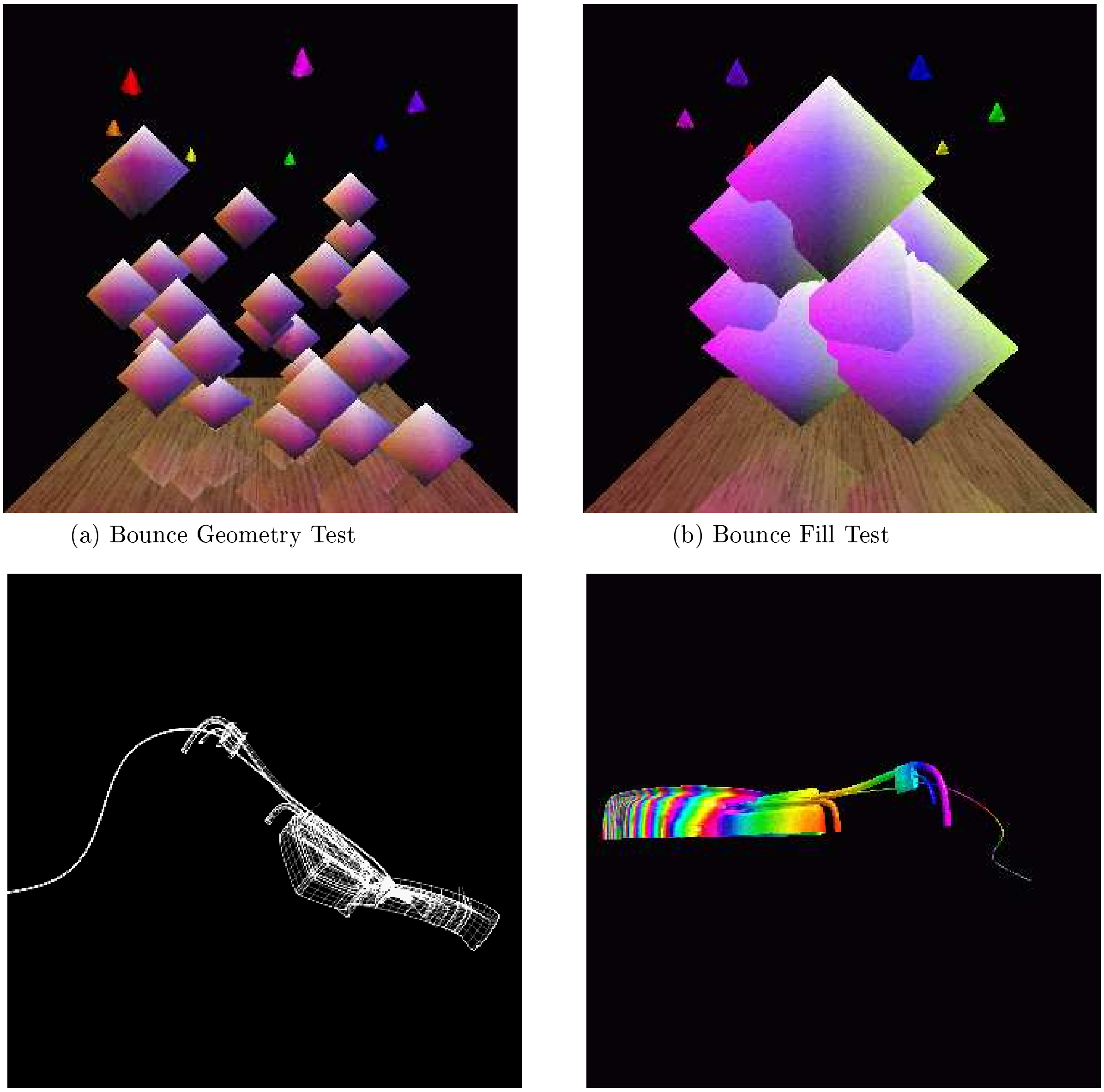
In this section we compare the performance of the GLINT TX500 based PC using both Linux and Windows NT software with Silicon Graphics systems in rendering 3-D objects in motion. The characteristics of the test systems are summarized in table 1.

Name	Model	Cost	Date	CPU	Clock	Graphics
slinky	Octane	28K	2/98	R10000	250MHz	EMXI
buzz	O2	18K	5/97	R10000	174Mhz	CRM
glint	Millenia	5K	5/97	P-Pro	200Mhz	GLINT 500TXD

Table 1: Test system characteristics

5.1 The Viewperf Benchmark

There are, at present, few well-accepted standard graphics benchmarks. Of those commonly cited, e.g., Viewperf (OPC project group), Glaze (Evans & Sutherland), and Rollercoaster (3DLabs) [5], the most well- known is probably the Viewperf collection. Viewperf is a portable OpenGL performance benchmark originally developed by IBM. It is endorsed by the OpenGL Performance Characterization (OPC) project group. Viewperf accepts numerous command-line options specifying a variety of rendering attributes. The object to be rendered is specified in an external file and a specific Viewperf benchmark is termed a *viewset*.



(a) Bounce Geometry Test

(b) Bounce Fill Test

(c) CDRS Test 1: Wireframe

(d) CDRS Test 6: Colors & Lighting

Figure 1: Benchmark Snap Shots

Lights	Linux	NT - Omni	NT - 3DL	02	Octane
1	13.17	10.60	24.47	11.36	34.95
2	12.14	9.79	19.42	11.35	34.25
3	11.34	9.42	18.61	11.23	34.18
4	10.92	8.96	17.89	10.79	34.07

Table 2: Viewperf Performance Results in Frames per Second

Of the standard OPC viewsets, the most commonly cited is Parametric Technology's CDRS, which includes seven different Viewperf animations. Each of the seven runs rotates a cut-away view of an electric lawn mower. Wire-frame, smooth gray-shaded, texture-mapped, colored, and partially transparent (blended) renditions are included in the tests. A weighted geometric mean frames per second (fps) is reported. Example frames from CDRS tests are shown in figures 1.c and 1.d.

We executed the CDRS viewset on each of the platforms in table 1. On the machine, *glint*, we tested our Linux software and two commercial Windows NT products. The results of the Viewperf tests on all the platforms are shown in table 2. It should be noted that the second processor on the system *glint* was not used in the benchmark runs. In fact, we had to install a UP version of HAL.DLL to get either of the Windows NT drivers to work at all!

We see that the performance of the PC compares very favorably with that of the SGI O2 machine which was acquired at the same time as the PC but was over three times as expensive. In contrast, the SGI Octane, the newest and most costly test platform, clearly outperforms the other systems by a wide margin. The most surprising aspect of these results was the outstanding performance of 3Dlabs driver for Windows NT. We discuss this in some detail below.

5.1.1 Analysis of Linux and Windows NT performance

Our Omnicomp Pro88 adapter was distributed with Omnicomp's Windows NT driver software, and that software was used in our initial testing. Our first Mesa back-end (with the single generic-triangle drawing routine) was slower than Omnicomp's. The values shown in table 2 were obtained after we added support for fast drawing of triangle strips and then gained an additional 20% in frame rate when we identified and eliminated three performance impacting float to integer conversions in Mesa [1].[3]

The 3Dlabs software is not simply a device driver. It contains (what appears to be) an OpenGL DLL that is over 1MB in size, a GLINT specific back-end DLL of about 630KB, and a true device driver of about 128KB. Even if one suspects that some Viewperf oriented tuning may be in play, the performance is clearly impressive.

In analyzing the performance gap we found that our CPU utilization was 100% throughout the Viewperf benchmark. This led us to suspect a Mesa related CPU bottleneck. To verify this, we modified our Mesa back-end so that it logged DMA buffers to disk as they were created. Next we constructed a simple application that read the entire log file into memory and then "replayed" the DMA buffers. This application obtained a significantly higher frame rate than the 3Dlabs software, demonstrating the Mesa was unable to fill DMA buffers at the rate they could be transferred and consumed.

Since the CDRS viewset contains over 30,000 vertices per frame, we felt that performance might be improved by more efficient vertex buffer management in Mesa. A fairly extensive rework of this part of Mesa was performed [4]

[3]This enhancement was subsequently incorporated by Brian Paul into Mesa 3.0.

System	Frames per second
Linux - First version	7.07
NT - Omnicomp driver	12.50
Linux - Final version	19.50
Linux - Final version + modified Mesa	25.00
Linux - Final version + unmodified Mesa + glint2	27.01
NT - 3DLabs driver	29.00
Linux - "Replay" DMA buffers	36.07

Table 3: CDRS Wireframe Test Results.

and a measureable performance improvement was obtained. Our frame rate increased from 13.17 to 15.45 for the complete CDRS-03 viewset, but still remains signficantly behind 3Dlabs software for Windows NT. The frame rates achieved on *glint* for the CDRS-03 wireframe test (which accounts for 50% of the overall score) are provided in table 3.

5.2 The Bounce Benchmark

We developed the *Bounce* benchmark to compare performance under a workload in which the graphics adapter rather than the host CPU is the bottleneck device. Boxes, varying in size and number, rendered with lighting, shading, and z-buffering, appear to bounce off a shiny textured wooden floor. Example frames from the Bounce tests are shown in figures 1.a and b. The reflected image in the floor is obtained by rendering inverted copies of the boxes under the floor and then writing the texture-mapped floor with partial transparency.

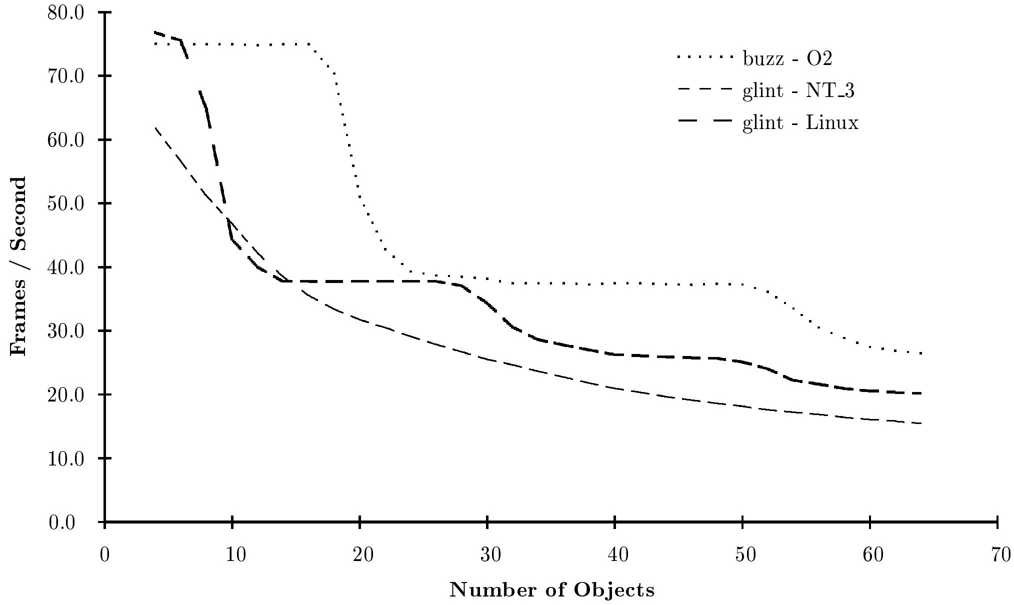


Figure 2: Bounce Geometry Test with 8 Lights

Figure 2 shows frame rates obtained as the number of bouncing boxes was varied from 4 to 64.[4] The SGI O2 system achieved consistently higher frame rates than the PC in our tests with Bounce benchmarks. The frame rate of our

[4]The SGI Octane system ran at its full frame rate throughout and is not shown on the graph.

Linux software in these adapter-bound benchmarks also generally exceeded that of the 3Dlabs software by a modest amount. This result indicates that while the 3Dlabs software is more CPU efficient than Mesa in rendering frames with very large numbers of triangles, the advantage disappears in simpler images.

6 X Windows Integration Issues

We are now investigating approaches for using accelerated Mesa in an X Windows environment. The direction and extent of this work have been complicated to some degree by the recent announcements of the XFree86 Project. In addition to the reorganization of the device dependent part of the X server in XFree86 R4.0 [7], Red Hat and Precision Insight [8] are developing the "Multipipe Direct Rendering Infrastructure for use with (sic) MESA." This infrastructure is said to "fully support direct hardware rendering into multiple X Windows." Both of these announcements are understandably vague with regard to release dates for specifications and beta version.[5] We wish neither to duplicate the work of others nor to build a large body of software that becomes incompatible with future standards. Thus, we plan to track these efforts as closely as possible in the hope of maintaining as much compatibility as we can.

At present, we have pursued two approaches to interoperating with X Windows. In the first of these efforts, we implemented an XFree86 3.3.2 based X server for the GLINT TX500. This server provides very reasonable performance for non-graphics applications although it presently uses hardware acceleration only for the PolyFillRect and CopyWindow BITBLTs. Accelerated graphics, using our Mesa back-end and driver, is supported, but only in a single X window at a time.[6] Double buffered visuals are supported via the standard approach of copying the frame buffer to the back buffer at the time the graphics application starts. This approach, also used in the 3Dlabs software, can introduce some "interesting" visual effects if another application updates the front buffer while the graphics application is running!

Our other approach to interoperating with X was developed in our work with Permedia 2. This "black-box" approach uses the accelerated X server for Permedia from SuSE. We provide one-window-at-time accelerated graphics in this environment in the following way. Each time the device driver receives a START_DMA request, it sends a SIGSTOP to the X server. It then saves a copy of the Permedia 2 registers comprising the graphics context of the server and copies the frame buffer to the back buffer. Shortly after the application completes a DMA transfer, the graphics context of the application is saved, the server's is restored, and SIGCONT is sent to the server.

Both of these approaches can be extended to support multiple preemptively scheduled graphics applications in straightforward ways. But support for interactive graphics and seamless, concurrent execution of non-graphic applications remains problematic at best. Thus, we strongly endorse all efforts to standardize both service and interface definitions for graphics libraries, device-dependent back-ends, X servers, and low level device drivers.

[5] Metrolink also markets an official, conformance certified OpenGL for Linux which is claimed to provide some degree of 2-D acceleration. However, it is not open source, requires the use of their X server, and provides no 3-D hardware acceleration at present.

[6] The 3Dlabs software for Windows NT also has the single accelerated window limitation, but they do support multiple slow graphics applications which we do not.

7 Conclusions

Rapid evolution of accelerated graphics adapters now permits generic PC based workstations to compete effectively with mid-range products from specialized vendors such as SGI. However, software support for high- performance graphics is an area which has been somewhat slow to evolve in Linux and one in which Windows NT now has a clear lead.

We have described here the design, implementation, and performance evaluation of a software system for providing high performance graphics in Linux via an accelerated back-end for the Mesa graphics library. While our Linux system competes effectively with mid-range SGI workstations, it clearly trails the 3Dlabs software when tens of thousands of triangles must be rendered per frame. The greatest strength of Mesa has been its ubiquitousness. It was designed for portability and has now been ported to over 80 different combinations of hardware and operating system. Nevertheless, certain design decisions that were absolutely necessary to permit it to work at all in a 640K DOS platform stand in the way of optimal performance on a 64M Linux platform. Thus, we now believe that some major reworking of Mesa is necessary if it is to compete effectively with well-done platform specific software on vertex intensive workloads.

Both our Linux software and that of 3Dlabs for Windows NT presently also have shortcomings in their ability to seamlessly support multiple, interactive, accelerated graphics applications under preemptive scheduling. Much work remains to be done before SGI is matched in this area.

Acknowledgments

GLINT, Permedia, and 3Dlabs are registered trademarks of 3Dlabs, Inc., Ltd. Windows and Windows NT are registered trademarks of Microsoft Corp. XFree86 is a registered trademark of the XFree Project, Inc. OpenGL and SGI are registered trademarks of Silicon Graphics, Inc.

References

[1] Geist, R., Smotherman, M., Treglia, D., and Westall, J., Real Time 3-D Graphics for the Linux PC, to appear in: "CMG98 Proceedings", Anaheim, Ca., Dec 1998.

[2] Knuth, D., "The Art of Computer Programming,", Vol. 1, second ed., Addison-Wesley, 1973, p 220.

[3] Paul, B., "The Mesa 3-D Graphics Library," http://www.ssec.wisc.edu/ brianp/Mesa.html.

[4] Treglia, D., "High Performance Graphics in Linux", M.S. Research Paper, Dept. of Computer Science, Clemson University, Clemson SC, Nov. 1998.

[5] Tome, C., "Accelerated 3D," *3D Design*, July, 1997, pp. 61 - 68.

[6] Woo, M., Neider, J., and Davis, T., *OpenGL Programming Guide, Second Edition*, Addison-Wesley, 1997.

[7] The XFree86 Project, Inc., "XFree86 Release Plans", Available on- line at http://www.xfree86.org/releaseplans.html, Oct. 1998.

[8] The XFree86 Project, Inc., "Major Contributors to XFree86 Development", Available on-line at http://www.xfree86.org/majorcontrib.html#pi, Oct. 1998.