Concurrent Engineering in OpenGL® Product Development

Robert J. Casey

L. Leonard Lindstone

Time to market was reduced when tasks that had been traditionally serialized were completed in parallel.

Concurrent engineering is the convergence, in time and purpose, of interdependent engineering tasks. The benefits of concurrent engineering versus traditional serial dependency are shown in **Figure 1**. Careful planning and management of the concurrent engineering process result in:

- Faster time to market
- Lower engineering expenses
- Improved schedule predictability.

This article discusses the use of concurrent engineering for OpenGL product development at the HP Workstation Systems Division.

OpenGL Concurrent Engineering

We applied concurrent engineering concepts in the development of our OpenGL product in a number of ways, including:

- Closely coupled system design with partner laboratories
- Software architecture and design verification
- Real-use hardware verification
- Hardware simulation
- Milestones and communication
- Joint hardware and software design reviews
- Test programs written in parallel.

Robert J. Casey

A senior engineer in the graphics products laboratory at the HP Workstation Systems Division, Robert Casey was the chief software architect for the OpenGL product. Currently, he leads the efforts on Direct 3D® technology in the graphics products laboratory. He came to HP in 1987 after receiving a BS degree in computer engineering from Ohio State University. He was born in Columbus, Ohio, is married and has two children. His outside interests include skiing, soccer, and woodworking.

L. Leonard Lindstone

Leonard Lindstone is a project manager at the HP Workstation Systems Division. He is responsible for software drivers for new graphics hardware. He joined HP in 1976 at the Calculator Products Division after earning a BSEE degree from the University of Colorado. He also has an MS degree in computer science from Colorado State University. Leonard is married and has three children. He enjoys music of all kinds and historical fiction.

Cultural Enablers

In addition to these technical tactics, the OpenGL team enjoyed the benefits of several cultural enablers that have been nurtured over many years to encourage concurrent engineering. These include early concurrent staffing, an environment that invites, expects, and supports bottoms-up ideas to improve time to market, and the use of a focused program team to use expertise and gain acceptance from all functional areas and partners.

System Design with Partner Labs

We worked closely with the compiler and operating system laboratories to design new features to greatly improve our performance (see the "System Design Results" section in the article on page 9). Our early system design revealed that OpenGL inherently requires approximately ten times more procedure calls and graphics device accesses than our previous graphics libraries. This large increase in system use meant we had to minimize these costs we previously had been able to amortize over a complete primitive.

We worked closely with our partner laboratories to ensure success. Our management secured partner acceptance, funding, and staffing, and the engineers worked on the joint system design. Changes of this magnitude in the kernel and the compiler take time, and we could not afford to wait until we had graphics hardware and software running for problems to occur. Rather, we used careful system performance models and competitive performance projections to create processor state count budgets for procedure calls and device access. These performance goals guided our design. In fact, our first design to improve procedure call overhead missed by a few states per call, so we had to get more creative with our design to arrive at an industry-leading solution. We managed these dependencies throughout the project with frequent communication and interim milestones.

Software Architecture and Design Verification

We designed and followed a risk-driven life cycle. To support the concurrent engineering model, we needed a life cycle that avoided the big bang approach of integrating all

Figure 1

The benefits of concurrent engineering.

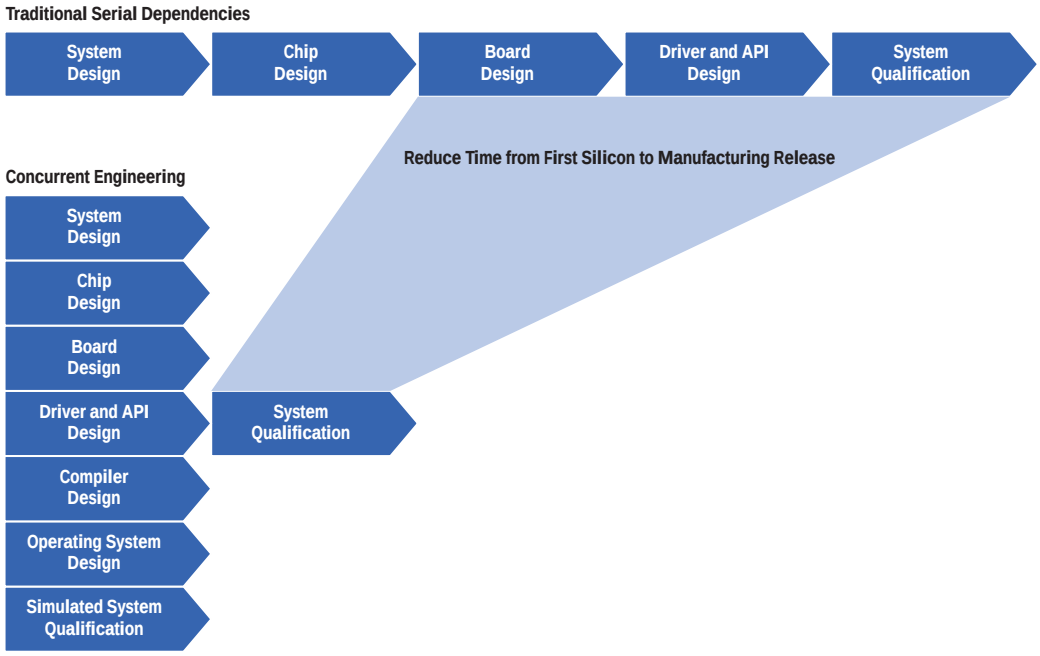
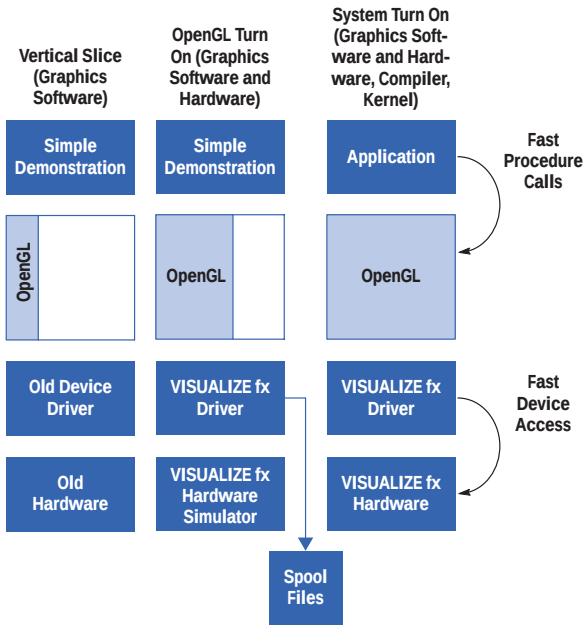


Figure 2

OpenGL concurrent engineering techniques.



the pieces at the end. This would result in a longer and less predictable time to market. Instead, we created a prototyping environment. This environment was initially created to test the software architecture and early design decisions. The life cycle included a number of checkpoints focused on interface specification, design, and prototyping.

One key prototyping checkpoint in this environment is what we called our "vertical slice," which represented a thin, tall slice through the early OpenGL architecture (see **Figure 2**). Thin because it supports a small subset of the full OpenGL functionality, and tall because it exercises all portions of the software architecture, from the API to the device driver-level interface. With this milestone, we had a simple OpenGL demonstration running on our software prototype.

The objectives of this vertical slice were to verify the OpenGL software architecture and design, create a prototyping design environment, and rally the team around this key deliverable.

Hardware Verification

Before we had completed verification of the software architecture, it became evident that this same environment needed to be quickly adapted and evolved to handle the demands of hardware verification. OpenGL features and performance represented the biggest challenge for the new VISUALIZE fx hardware. Although this hardware would also support our legacy APIs (Starbase, PHIGS, PEX), most of the newness and therefore risk was contained in our support of OpenGL. By evolving our prototyping environment for use as the hardware verification vehicle, we were able to exercise the hardware model in real-use scenarios (albeit considerably slower than full performance).

Evolving this environment for hardware verification required us to take the prototyping further than we would have for software verification alone. We had to add more functionality to more fully test the OpenGL features in hardware. We also had to do so quickly to avoid delaying the hardware tape release.

This led to our second key prototyping checkpoint, which we called "OpenGL turn on." This milestone included the same OpenGL demonstration running on the VISUALIZE fx hardware simulator. We also added functionality breadth to the vertical slice (see **Figure 2**). Doing all this for a new OpenGL API represented a new level of concurrent engineering, in that we were running OpenGL programs on a prototype OpenGL library and driver and displaying pictures on simulated VISUALIZE fx hardware, all more than a year before shipments.

The key objective of this milestone was to verify system design across the API, driver, operating system, and hardware. The system generated pictures and, more importantly, spool files (command and data streams that cross the hardware and software interface). These spool files are then run against the hardware models to verify hardware design under real OpenGL use scenarios.

This prototyping environment has the following advantages:

- Reduces risk for system design and component design
  - Resolve integration issues early

- □ Identify holes and design or architecture flaws
- □ Enable prototyping to evaluate design alternatives

- Enables key deliverables (hardware verification spool files)
- Creates exciting focal points for developers
- Fosters teamwork
- Enables joint development
- Provides a means to monitor progress
- Provides a jump start to our code development phase.

This environment also has potential downsides. We felt there was a risk that developers would feel that the need or desire to prototype (for system turn on and hardware verification) could overshadow the importance of product design. We did not want to leave engineers with the model: write some code, give it a try, and ship it if it works.

Thus, to keep the benefits of this environment and mitigate these potential downsides, we made a conscious decision to switch gears from system turn on and prototype mode to product code development mode. This point came after we had delivered the spool files required for hardware verification and before we had reached our design complete checkpoint. From that point on, we prototyped only for design purposes, not for enabling more system functionality. We also created explicit checkpoints for replacing previously prototyped code with designed product code. This was an important shift to avoid shipping prototype code. All product code had to be designed and reviewed.

Hardware Simulation

One key factor in our concurrent engineering process is hardware simulation. A detailed discussion of the hardware simulation techniques used in our project are beyond the scope of this article. Briefly, we use three levels of hardware simulation:

- A behavioral model (written in C)
- A register transfer level model (RTL)
- A gate model, which models the gate design and implementation.

The advantages of the behavioral model are that it can be done well before the RTL and gate model so we can use it with other components and prototypes. The behavioral model is also significantly faster than the other models (though still about 100 times slower than the real product), allowing us to run many simple real programs on it. The RTL model runs in Verilog and runs about one million times slower than the real product. This limits the number and size of test cases that can be run. The gate model is even slower. Even so, we kept over 30 workstations busy around the clock for months running these models. Often a simulation run will use C models for all but one of the new chips, with the one chip being simulated at the gate level.

Milestones and Communication

We set up a number of R&D milestones to guide and track our progress. The vertical slice and OpenGL turn on were two such key milestones. OpenGL developer meetings were held monthly to make sure that everyone had a clear understanding of where we were headed and how each of the developers' contributions helped us get there.

Software and Hardware Design Reviews

The hardware and software engineers also held joint design reviews. The value of design reviews is to minimize defects by enabling all the engineers to have the same model of the system and to catch design flaws early and correct them while defect finding and fixing is still inexpensive in terms of schedule and dollars.

On the software side, the review process focused heavily on up-front design reviews (where changes are cheaper) to get the design right. We maintained the importance of doing inspections but reduced the inspection coverage from 100 percent to a smaller representative subset of code, as determined by the review team. We also increased the number of reviewers at the design reviews and reduced the participation as we moved to code reviews. We maintained a consistent core set of reviewers who followed the component from design to code review.

Tests Written in Parallel

To bring more parallelism to the development process, we had an outside organization develop our OpenGL test programs. By doing so, we were able to begin nightly regression testing simultaneous with the code completion checkpoint because the test programs were immediately available. Historically, the developers have written the tests following design and coding. This translates into

a lull between the code completion checkpoint and the beginning of the testing phase.

Parallel development of the tests with the design and implementation of the system was a key success factor in our ability to ship a high-quality, software-only beta version of our OpenGL product. No severe defects were found in this beta product—our first OpenGL customer deliverable.

One thing we learned from using an outside organization to help with test writing was that writing test plans is more a part of design than of testing. The developers, with intimate knowledge of the API and the design, were able to write much more comprehensive test plans than the outside organization.

Conclusion

We achieved several positive results through the use of concurrent engineering on our OpenGL product. Ultimately, we reduced time to market by several months. Along the way, we made performance and reliability improvements in our software and hardware architectures and implementations, and we likely prevented a chip turn or two, which would have cost significant time to market.

▶ Go to Next Article

▶ Go to Journal Home Page