Wireframe Project Proposal*

Stephen A. Edwards (se2007)

Spring 2024

The core of this project is a hardware accelerator for animating numerous straight line segments on a VGA display (Figure 1). The project as a whole is meant to be a greatly simplified model of modern 3D graphics accelerators for displaying wireframe models instead of filled, textured triangles. It will read and display objects represented in some 3D file format (such as Wavefront OBJ) and allow the user to adjust the viewpoint.

The hardware accelerator will be responsible for taking a list of line segments expressed in 2D coordinates and displaying them on the screen; software will be responsible for supplying this line segment information, which it will generate from rotated, scaled, and projected 3D data loaded from a 3D file format. Each line segment will be drawn one pixel wide with white pixels against a black background; the hardware will not attempt to handle any 3D considerations such as hidden line removal.

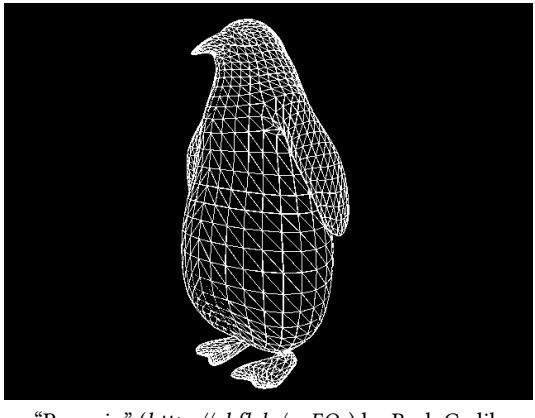

Figure 1: A wireframe image. This is drawn as many straight line segments, but this 3D model started as a list of 3.4k triangles that references a list of 1.7k 3D vertices.

*This is meant as an exemplary project proposal for the author's CSEE 4840 Embedded System Design

Framerate will be 60 frames per second. The display will be flicker-free by synchronizing the delivery of new lists of line segments from the software with the screen refresh.

Resolution will be a VGA standard with a minimum of 640×480. The ultimate choice of resolution will be based on the amount of memory available on the FPGA.

Frame Buffer will be one bit-per-pixel and likely be double-buffered so that one frame will be displayed and erased (I will try to take advantage of write-after-read behavior) while the other is being rendered, but this will ultimately depend on memory constraints. Frame buffer memory will be on the FPGA, not in an external DRAM.

Line Drawing will be done using Bresenham's line algorithm with no attempt at antialiasing. Coordinates supplied will be in units of integer pixels from the upper left corner, likely 10 bits per dimension.

Hardware-Software Interface will accommodate two main functions: transferring the display list (of line segments) from software to hardware once per frame and synchronization of the software so that it supplies data at exactly 60 frames per second.

Because I assume the software will recalculate the display list for each frame, there seem to be two main choices for the communication style. A critical design decision is which of these approaches to adopt.

In a command-based style, the software sends a sequence of drawing commands to the hardware a word at a time through a single command port (the commands themselves may be a variable number of words). Internally, the hardware may buffer these commands. This style encourages the use of a DMA controller to perform the actual transfers, freeing the software to focus purely on preparing the display list to transfer. Sony's Playstation takes this approach and includes a DMA controller that can stream command sequences from in-memory linked lists[1]. This approach may reduce the amount of command buffering needed on the accelerator but may preclude elaborate commands (e.g., looping) and certain parallelism.

In a memory-based style, the software stores the display list in a shared memory, which the hardware then scans to perform rendering operations. Such an approach is more flexible (e.g., it allows more complicated data structures to be transferred such as an array of triangles that references an array of vertices) but requires more buffer memory and a more elaborate system in the hardware for reading commands. Modern GPUs, such as those supplied by Intel in recent Core series processors, take this approach.

File Format Ultimately, the software will be able to read one of the many 3D object file formats. Wavefront OBJ is fairly simple and textual, but will still require some processing to convert to a simple set of line segments. The STL format, commonly used for 3D printing, is another alternative.

[1] Joshua Walker, Everything You Have Always Wanted to Know about the Playstation But Were Afraid to Ask, *https://archive.org/details/psx-everything-you-have-always-wanted-to-know-about-the-playstation*

3D to 2D Projection The software will use the standard algorithm that starts with a position of a camera to project vertices onto the plane.

User Interface The user will be able to move a virtual camera around and through a 3D model. Options include using a mouse to implement standard controls (e.g., dragging a virtual point around a sphere, translating the camera horizontally and vertically, and moving closer to or farther from the center of the image), using an analog USB game controller with one or more joysticks, or using a multiple-knob little keyboard with a USB interface. One of these interfaces will be chosen as part of the design phase.

Major Tasks

- Design decisions for all major points (e.g., resolution, memory size, hardware/software interface style and details, any use of DMA, and supported 3D file format). These will be in the design document
- Desktop software prototype running under SDL able to read and display 3D object data as a wireframe
- Verilator-based testbench able to run the hardware accelerator to verify line drawing, double-buffering (if applicable), and delivery of the display list. The testbench will write out each generated video frame as a PBM file.
- Hardware accelerator source code, consisting of VGA timing generator, frame buffer display, line drawing, and software synchronization mechanism
- Linux device driver for the hardware accelerator including a "high-level" interface meant to be called from C
- User interface for 3D software that uses *libusb* to read from the chosen user interface.