Project Interim Report

A light-weight CPU implementation of a 3D graphics pipeline for embedded systems

Charles Nicklas Christensen (GitHub: charlesnchr / CRSid: cnc39)

December 2nd, 2017

Since writing the project proposal I have worked an average of 2 hours per week on this project amounting to roughly 8 hours in total. Below I will address how those hours were spent, what insight this effort has given and how well the time schedule holds.

1. As a brief reminder the project is about using a MCU with a display to render 3D graphics. Hence, the only two components as such is a MCU and a display, both of which we have access to by default (note that the current sensor breakout might be used for evaluation). However, an appropriate MCU is required, i.e. one that has sufficient memory to perform all steps in a 3D graphics pipeline. The KL03 MCU that we have been provided with only has 2 kB SRAM, which may very well inhibit the capabilities of the implementation. There are some possible remedies that I have considered – i.e. reducing colour depth; using an upscaling technique such as checkerboard rendering to render at a sub-native resolution but still use all pixels in the display; or finally using tiled rendering, which will be less memory-consuming but not very efficient. Even using these techniques, it would be very helpful to have more memory and so using the KL05 MCU instead is desired. I have already been offered to use this MCU by Phillip, and I plan to accept this offer and pick it up from him in the near future, possibly next week.

2. The progress so far has mainly revolved around doing a prototype implementation in MATLAB. In roughly 200 lines of code, subroutines for performing model-view-projection transformation has been implemented along with a simplified rasteriser based on the scanline algorithm and no clipping. This small high-level implementation constitutes a minimal pipeline that renders declared 3D triangle meshes into a bitmap of a defined resolution taking into account the position, scale and rotation of objects and properties of a camera based on a perspective projection (position, direction, rotation, field of view and aspect ratio).

   Apart from this high-level implementation some effort has been put into handling the GDDRAM of the display. I am now able to copy an array of colour values directly from the MCU SRAM to the GDDRAM. By increasing the bandwidth of the SPI connection, a 96x64 bitmap could be drawn on the display in a fraction of a second. This kind of memory transfer is unsurprisingly much faster than drawing pixels one-by-one by rectangle draw commands.

   Finally, the coursework on applying the current sensor breakout board might become useful by the end of my project as I can now quantify the power usage of the display, when I am to test and benchmark the overall 3D rendering system.

3. With the recent prototyping in MATLAB, the conversion of the implementation into C should be relatively straight-forward. However, it has become more clear that memory might be an issue, and so I will try to port the minimal rasteriser sooner in order to discover how challenging the memory restriction will be, such that I can potentially begin focusing on one of the mentioned remedies in due time. After that I will go back over the initially planned features of the pipeline and implement those not part of the minimal implementation (such as clipping and fragment shader).

4. As stated above an approximate 8 hours has now been put into the project. As this has already resulted in a working minimal prototype as well as the knowledge of how to write a frame buffer to the GDDRAM, I think it likely that porting should not take more than a couple of hours after which additional pipeline features as well as memory challenges can be addressed for several hours. Assuming the porting of the current prototype will take 3 hours, the Gantt chart of my proposal holds true (since step 2, 3, 4.2 and 6 would then be done).