# Terminology and Abbreviations

|  |  |
| --- | --- |
| ISA | Instruction Set Architecture |
| SWR | Software Rasteriser |
| HWR | Hardware Rasteriser |
| RISC | Reduced Instruction Set Computer |
| MVP | Minimum Viable Product |
| RV32 | 32-bit RISC-V ISA |
| ISE | Instruction Set Extension |
|  |  |
|  |  |

# Aim of the project

The aim of the project would be to design and implement an extension to the RISC-V ISA which would allow for the hardware acceleration of graphics rendering.

Before multicore, general-purpose GPUs became common, GPUs were fixed-function pipelines designed to accelerate rendering of computer graphics, taking a 3D scene made of vertices and triangles as an input and providing rendered frames as an output.

## Deliverables

### Minimum Viable Product (MVP)

 The minimum viable product for this project would be to add a single instruction to the RISC-V 32G ISA that would accelerate both loops of the triangle rasterization algorithm in hardware.

The system would be a combination of a pre-existing RV32G compliant CPU core with an added hardware accelerator to implement the rasterization instruction. I wish to implement the system on an FPGA.

This is because rasterization forms a fundamental part of the 3D graphics pipeline and one that can benefit greatly from a fixed function accelerator.

I would like the hardware rasteriser to be compliant with Chapter 14.6 of the OpenGL 4.6 Core Profile Specification. This provides a functional specification for a fixed function rasteriser with no anti-aliasing and with support for a depth buffer.

The hardware system will run a software implementation of the graphics pipeline in order to render a scene. However, the part of the pipeline corresponding to rasterization will be replaced by the custom RISC-V instruction.

The implementation does not need to be highly performant, it just needs to work.

### Extensions

The first obvious extension is to optimise the performance of the system. This can be at the architectural, or micro-architectural level.

A additional features from the OpenGL spec can also be added like anti-aliasing. This would be another place to look to extend the project.

Additional instructions to accelerate other parts of the graphics pipeline can be designed and implemented as well. For example, adding an instruction to do texture mapping, vertex transformations and image filtering or to do rasterization of other graphics primitives.

### Fallbacks

If time is limited, the features of the MVP can be reduced for example by not implementing a depth buffer or by using a more naïve rasterization algorithm.

Another option for simplifying the project would be to write the RTL for the project but only run it in a waves simulation instead of implementing it on an FPGA. This alone could save huge amounts of time and because this step is the last in the project it could very easily be skipped without making previous work redundant.

This second option is more likely to be the fallback I take.

# Risks

I am mostly on track with the plan given in my project proposal however a few things have changed. I’ve found is a SW implementation of the graphics pipeline with a rasteriser written in C++. I have also been able to create custom models in blender, save their vertices and attributes and use the SWR to render the objects from this data. However, the rasterization component of the benchmark is not OpenGL compliant and does not have some features I would like. Therefore, I will need to modify it to suit my needs.

The risk here is that this modification takes too long and that I spend too long in the modelling and architectural simulation phase.

I am taking an exam module this term and one next term, this will allow my time in both the Winter and Spring to work on this project. However, having two summer exams means my time in the Summer term will be more limited. This means I will need to make steady and meaningful progress in the first two terms

I take part in some Extracurricular activities that take my time, those being climbing and being team captain of the imperial college climbing team.

# Background

## Graphics and the Rendering pipeline

The graphics pipeline is a conceptual framework for describing the steps needed to render a frame from a geometric description of a scene, this includes lighting, textures, filtering and more. The steps outlined below can be implemented in hardware or in software.

The graphics pipeline can be split into 5 broad stages: (<https://materials.doc.ic.ac.uk/view/2021/60005/Course%20Material/22> ):

1. Vertex Data (input)
2. Vertex processing - Transformations, Lighting, Interpolation
3. Rasterization – mapping of 3D scene to 2D screen
4. Fragment processing - Lighting, Texture operations, Filtering
5. Frame Buffer (output)

## Inputs and Outputs

### Vertices

A Vertex in this context is a point in 3D space which can have multiple associated attributes such as colours, texture coordinates, material properties and normal. These will be processed by stage 2 and are then passed to stage 3 as input. This input will likely be in the form of an array of vertices that are interpreted as sets of triangles by the rasteriser.

### Triangles

### Fragments

The rasteriser then produces fragments as output. Fragments are pixel candidates, each pixel can have multiple fragments associated with it. They are produced by the rasteriser. A pixel will have multiple candidates if there are multiple triangles with points on them that map to the same pixel.

The final colour of the pixel will be determined by some combination of these fragments. This can be in the form of averaging the fragment colours for transparent objects or if the object at the front is opaque then only the closes fragments colour will be used (depth buffer).

## Rasterisation

The rasterization stage is the one I will be focusing on for the MVP of the project. It contains these stages as described by the OpenGL spec:

1. Face Culling
2. Project Triangles to screen space
3. Checking if a pixel lies in a projected triangle
4. Triangle Clipping in screen space
   1. Triangles may be turned into quads
5. Interpolation of vertex attributes including depth

A more detailed description of the MVP can be found in the Architecture Spec Document

## RISC-V

RISC-V is a new open source ISA that is designed the be highly extensible. The design of a new ISA is extremely expensive and time consuming, the open-source nature of RISC-V means that individuals and companies without the time or money to design their own can create new products and CPU based systems using RISC-V.

RISC-V is designed to be highly extensible, therefore there is plenty of space left in its opcode space to add custom instructions. The base integer 32-bit ISA is called RV32I. There are also several standard instructions set extensions such as F which adds floating point arithmetic support, M which adds multiply and divide instructions and many others.

There several standard ISE’s which are grouped together to make RV32G which intended to be a general-purpose ISA.

# Method

TODO: finish writing up the new method section

TODO: create new timetable based on new method

**NB: Additional method steps have been added since the initial proposal**

## What have been done

1. Find benchmarks that can be used for evaluation of the system
   1. Should use the whole graphics pipeline so I don’t have to find new benchmarks for each new feature I want to add
   2. This has been partially found in the form of a software rasteriser built in C++.

## What is left to do

1. Compile software pipeline benchmark to RV32G
2. Run above compiled program in an RV32 simulator
   1. Gem5 – can be cycle accurate but would have to choose implementation which isn’t super useful
   2. Qemu – functional emulation, not cycle accurate
3. Choose RV32G core to add my HWR to
4. Specification of instruction format and behaviour
   1. use OpenGL spec as a base
5. Modelling and Validation
   1. Model the behaviour of the instruction in a high-level language like MATLAB so architectural features like data types can be designed more easily
   2. Allows for precise definition of functional requirements of the block
   3. Allows for model checking during top level verification of the block
6. System Design and verification
   1. Modify RISC-V soft core to implement this new instruction in RTL
      1. What parts of the architecture will need changing
      2. Microarchitectural design
      3. Implementation in RTL
      4. RTL simulation
      5. Unit level testing and subsystem testing
      6. FPGA implementation
      7. Top level testing
         1. Implement benchmark with new instruction and run on system
   2. Verification
      1. Likely do directed testing, maybe do constrained random if time?
      2. Each level of testing should have a well thought out test plan
7. Choose systems I wish to compare my design to:
   1. Configurable RISC-V core (one which I am planning to extend with my instr.)
   2. RISC-V core with vector extension (same configurable core as above)
   3. Commercial CPU, other non-RISC soft CPUs
   4. Commercial GPU, soft GPU
8. Evaluation
   1. Take power and performance measurements on design while running benchmarks as in step 3
   2. Compare my design to currently available software implementations from step 8
   3. The project can be considered a success if the benchmarks are able to run on the design and produce correct results.
   4. If power and performance specifications are met this is a bonus.
   5. If not, a discussion about optimisation can be had

# Rough timetable and Intermediate milestones

|  |  |  |
| --- | --- | --- |
| Task | Time needed (weeks) | End date |
| Basic Background research | 2 | 22 Nov |
| Find benchmarks that can be used for evaluation of the system | 1 | 18 Nov |
| Inception Report |  | 22 Nov (6 weeks between this report and interim report) |
| More Background research | continuous until the next report | 2 Jan |
| Choose systems I wish to compare my design to | 2 | 2 Dec |
| Implement those benchmarks on the above systems | 2 | 16 Dec |
| Measure performance (cycles or real time) and power of benchmarks | 1 | 23 Dec |
| Interim Report |  | 2 Jan (20 weeks between this report and first draft final report) |
| Create performance and power specifications for system | 2 | 16 Jan |
| Specification of instruction format and behaviour | 2 | 30 Jan |
| Modelling and Validation | 4 | 27 Feb |
| System Design and verification | 10 | 15 May |
| Evaluation | 1 | 22 May |
| Abstract and Draft Report |  | 31 May |
| Final Report |  | 16 June |
| Presentations |  | 21 June |

Green rows represent project deliverables, others represent milestones based on the method section.

**NB:** The steps between the interim report and the first draft of the final report can be repeated for any additional feature I wish to add. I would not need to repeat any of the steps before January as long as the new instruction accelerates part of the benchmarks. This makes it critical to find benchmarks which use the entire graphics pipeline.