pyvale.zraster: A Performant Software Rasteriser for Digital Image Correlation Uncertainty Quantification

L. Fletcher[1a], J. Hirst[1], W. Bielajewa[1], M. Sampson[1], A. Marsh[1], and A. Tayeb[1]

[1]UK Atomic Energy Authority, Culham Campus, Abingdon, OX14 3DV, United Kingdom

[a]lloyd.fletcher@ukaea.uk

Abstract. Uncertainty quantification (UQ) is key to making quantitative use of digital image correlation (DIC) measurements for simulation validation. While characterising random errors for DIC is straightforward, synthetic image deformation is required to characterise the systematic errors in DIC. Computer graphics pipelines for image simulation are optimised for linear triangles whereas there is a need to accurately render quadrilateral and higher order elements for DIC UQ. To address this need, we have developed a performant software rasterisation renderer for DIC UQ as part of the `pyvale` python package called `zraster`.

Introduction

There has been considerable effort in the experimental mechanics community to develop methodologies for assessing uncertainties of digital image correlation (DIC) measurements [1,2]. This effort is key to allow DIC to be used for quantitative measurements, inverse identification, and model validation. There are two main types of measurement errors in DIC: random and systematic errors. While assessment of random errors is straightforward, using standard statistical analysis methods to extract noise floors makes the assessment of systematic errors more challenging. Systematic errors in DIC arise from several factors including: spatial/temporal resolution and digitisation error.

There are two main techniques in computer graphics for rendering images: object-order rendering (rasterisation) and image-order rendering (ray/path tracing). Here we focus on rasterisation due to its computational performance over ray/path tracing. All computer graphics rendering methods are highly optimised for linear triangular meshes and typically do not support the higher order elements that are common in finite element (FE) simulations. How this is accounted for in common visualisation platforms like the Visualisation Tool Kit (VTK) [3] is to sub-tessellate the higher order elements into a series of linear triangles. When using sub-tessellation of higher order elements an additional bias is introduced into the rendering pipeline which will appear as an unrealistic systematic error in the DIC results when these images are analysed. Therefore, it is desirable to have image rendering tools for DIC that can accurately render speckle patterns on the higher order elements typically used in FE simulations. Linear triangular rendering operations are often run on a specifically optimised graphics processing unit (GPU) which is referred to as hardware rasterisation, as opposed to software rasterisation which is implemented as code run on the central processing unit (CPU). Here we focus on software rasterisation as it is well suited to solving the non-linear equations required to accurately render higher order elements.

In this work we develop a software rasteriser as part of the `pyvale` python package [4] called `zraster` (short for Zig rasteriser) with support for linear and quadrilateral elements with linear or quadratic shape functions. The core of our software rasteriser is written in the Zig programming language as it is a compiled language with manual memory management and support for single instruction multiple data (SIMD) vector types. In the future we will connect our underlying Zig code to a python interface allowing for ease of use for the experimental mechanics community while maintaining the excellent performance of a compiled language under the hood in keeping with the `pyvale` design philosophy.

Software Architecture and Methodology

Our software rasteriser uses a tiling architecture in which the camera is divided into a series of small 32x32 pixels tiles. When projecting our elements onto the camera sensor we can then assign elements to the tiles and each tile can be processed independently allowing for ease of threading and cache locality of data. Our rasteriser also supports sub-pixel anti-aliasing with a default 2x2 sub-division of all pixels. An overview of the raster pipeline for `zraster` is shown in Fig. 1, this sequence is executed for each frame the user wants to render. The pipeline starts by preparing all meshes to be rendered including applying displacements to nodal coordinates and reshaping the data into an element-wise array. The second stage of the pipeline performs coordinate transformations, culls elements that are not visible or back-facing and then calculates the sensor space bounding boxes for the remaining elements. The third stage of the pipeline counts the number of elements overlapping each tile then allocates and assigns data structures for the active tiles and the overlap bounding boxes between the tile and the elements. The fourth stage of the pipeline is the raster loop, here we loop over the active tiles and scan the sub-pixels to determine if they are in or out of the projection of the element onto the sensor. If the sub-pixel is inside the projection, then we check its depth in the scene. If the element is in front, we shade that sub-pixel with the user define shader. The final part of the raster loop is to average the sub-pixel image buffer and write to our output image buffer. The fifth and final stage of the pipeline saves the images to disk and/or returns them to the user in memory.

The raster loop is the most computationally expensive part of the pipeline. To increase performance in the raster loop we have used Zig's compile time code execution to ensure that the inner loop is branch less by constructing a series of geometry kernels and shader kernels. The geometry kernels we support include tri3, tri6, quad4, quad8 and quad9 elements. The shader kernels we support include flat or texture shading with a greyscale or RGB textures with 8 or 16 bits. Texture shading enables the user to input a speckle pattern mapped to their FE mesh and render this. To ensure accuracy of texture interpolation we have implemented cubic and quintic texture interpolation schemes.

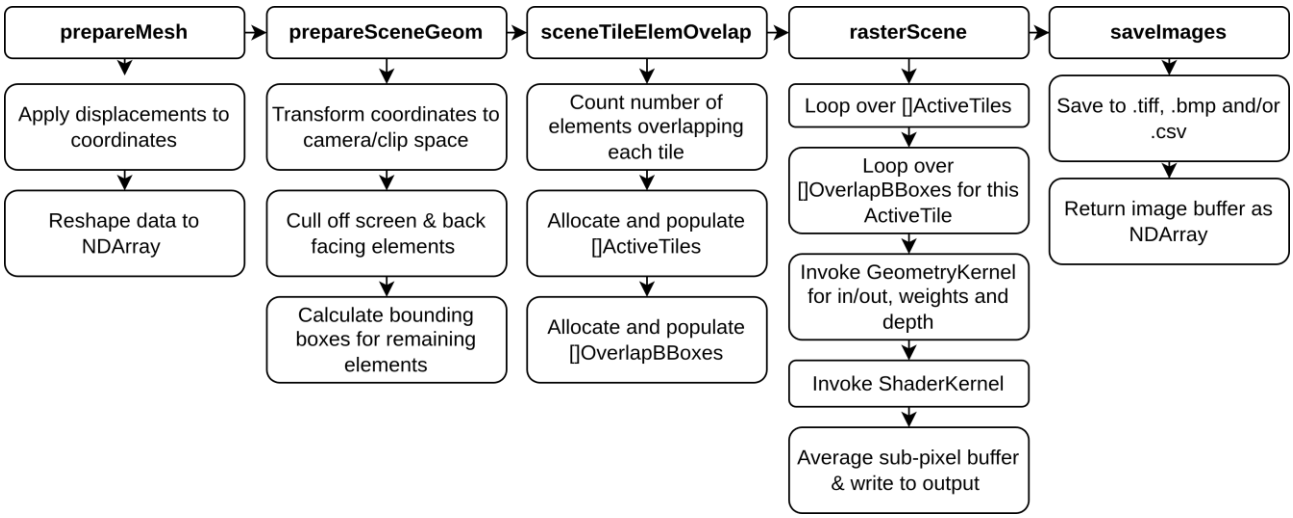

Figure 1: Overview of the `zraster` pipeline executed for each rendered frame.

An example test output from `zraster` is shown in Fig. 2 demonstrating the ability to render multiple elements of all supported types with flat or texture shading in the same scene. For the quadratic elements in this figure we can see that both concave and convex edges are accurately rendered.

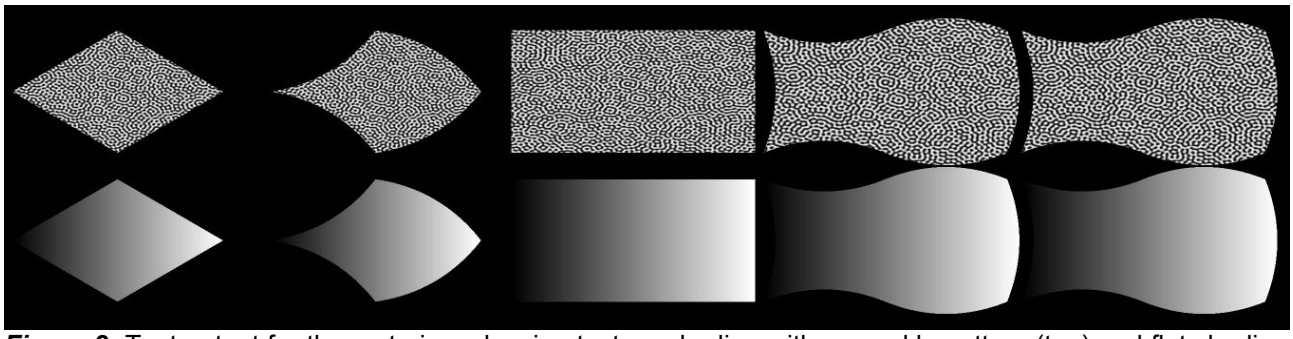

Figure 2: Test output for the rasteriser showing texture shading with a speckle pattern (top) and flat shading of a displacement field (bottom). Two elements are shown of each supported type, from left to right: tri3, tri6, quad4, quad8 and quad9.

Conclusion & Future Work

We have developed a performant software rasteriser (pyvale.zraster) specifically designed for DIC UQ. Our current implementation supports a tiling architecture, accurately renders linear and quadratic elements, as well as supporting flat and texture shading. In the future we will implement threading to increase performance. Once these are implemented, we will create a C compatible interface and connect this to a user-friendly python interface in `pyvale`.

Acknowledgments. We would like to acknowledge the support of UKRI through the Future Leaders Fellowship scheme, grant number MR/Y015916/1.

References

[1] Lava P, Jones EMC, Wittevrongel L, Pierron F (2020) Validation of finite-element models using full-field experimental data: Levelling finite-element analysis data through a digital image correlation engine. Strain 56:e12350. https://doi.org/10.1111/str.12350
[2] Rohe D, Jones E (2021) Generation of Synthetic Digital Image Correlation Images Using the Open-Source Blender Software. Experimental Techniques 46:. https://doi.org/10.1007/s40799-021-00491-z
[3] Schroeder, W. Martin, K., Lorensen, B., (2006), The Visualization Toolkit (4th ed.), Kitware, ISBN 978-1-930934-19-1
[4] Fletcher, L., et al., pyvale, https://github.com/Computer-Aided-Validation-Laboratory/pyvale, 2025