An Interactive, Virtual Wind Tunnel using Virtual Reality and Unreal Engine 4

Adrian R. G. Harwood*, Alistair J. Revell

School of Mechanical, Aerospace and Civil Engineering, The University of Manchester, Sackville Street, M1 3BB, United Kingdom

20th May 2019

Abstract

This article presents an integrated, interactive modelling and simulation tool for aerodynamic design and analysis. The development and coupling of a GPU-accelerated Computational Fluid Dynamics (CFD) library, a 3D scanning library and a virtual-reality-powered video game are presented and the resulting virtual wind tunnel described. Users of the tunnel may interact with both geometry and solver from within the virtual environment providing a truly unique tool for performing high-level aerodynamic analysis around imported and scanned objects. Here we introduce the relevant technologies and approaches used to build the components and discuss the technical challenges of integrating them. The flow solver is validated against experimental data for a representative turbulent flow and demonstrates excellent agreement with available data. We also present the peak performance of the solver on current hardware. Limitations and potential expansions for this proof of concept are discussed as well as applications in a range of other scientific fields.

Keywords: Virtual Reality, Interactive Physics, Lattice-Boltzmann Method, CUDA, Unreal Engine 4

1. Introduction

Modelling and simulation is an essential part of engineering. Physical testing of designs can require significant effort and expense; test components must be manufactured and suitable test facilities constructed, acquired or hired. Furthermore, based on the results of testing, designs and test procedures may need to be reworked incurring further costs in terms of both time and money. Modelling and simulation avoids this expense by allowing engineers to conduct a wide range of tests virtually. As designs can remain entirely digital, this activity generally occurs early in the design process. Upstream modifications of a design are almost always cheaper to implement compared with a downstream counterpart.

Modelling and simulation in engineering is a complex and sometimes costly process, with simulations taking time and resources to configure, run and post-process. Typical use cases of modelling and simulation in engineering require highly accurate modelling of potentially complex phenomena and thus, simulation run times can be of the order of days or weeks [1]. However, in the case of early-stage design, there is a recognised need to run a large number of fast simulations with reduced accuracy to allow early-stage development or communication of concepts and principles. Therefore, it is necessary for researchers to develop modelling and simulation approaches which can be used legitimately to provide lower accuracy estimates in a very short amount of time. This relaxation of requirements allows reduced-order modelling and increased approximation, creating opportunities for simulations to be run at 'interactive speeds'. In such simulations, results are simulated and visualised instantly, allowing users to also interact with, and reconfigure, their simulations at run time.

Computational Fluid Dynamics (CFD) is a modelling and simulation discipline which simulates the behaviour of fluids with the fluid represented by a grid of control volumes (cells) or a collection of particles. In recent decades, CFD simulations based on the Boltzmann equation have been increased in popularity and are capable of representing macroscopic hydrodynamics correctly [2]. The principal advantage of these formulations over more traditional CFD methods which solve the macroscopic Navier-Stokes Equations is the ability to accelerate the calculation on mass-parallel hardware such as

*Corresponding author
Email address: adrian.harwood@manchester.ac.uk (Adrian R. G. Harwood)

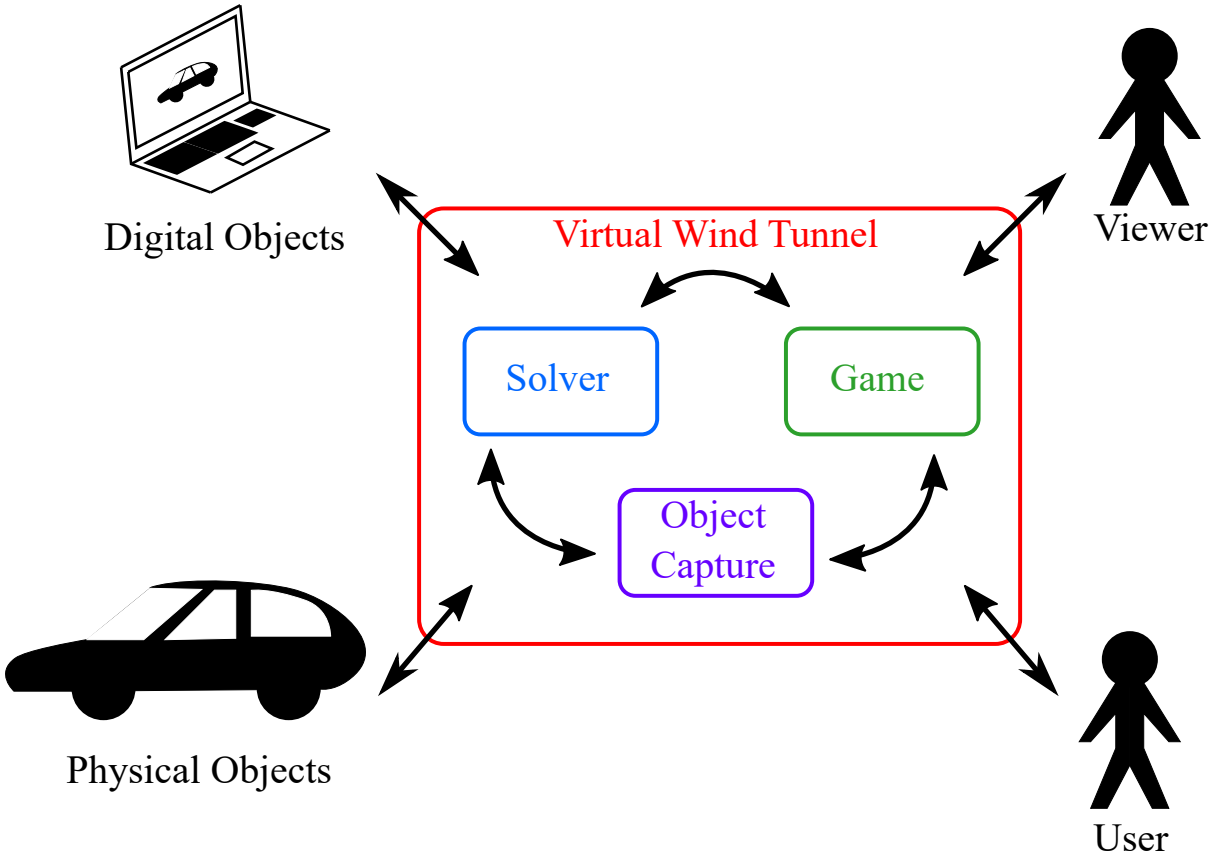


Figure 1: The virtual wind tunnel concept integrating three key areas of technology to realise a working prototype.

Graphics Processing Units (GPUs). This is enabled principally by to the spatial locality of the numerical schemes involved. In many cases, the computational throughput of such simulations is so high that predictions take place at speeds approaching real-time [3, 4], thus facilitating interactive applications.

An application which combines an interactive flow solver with an in-situ visualisation and user interface is termed a 'virtual wind tunnel' and is a tool often identified as part of the virtual engineering paradigm [5]. Virtual wind tunnels represent a shift in applied CFD methodologies and offer the potential to transform the way CFD is used within the industry. However, in order for them to be considered more than simply 'toys', work must be done to integrate useful physical modelling of a suitable degree of accuracy. In this paper, we present our implementation of a virtual wind tunnel incorporating a state of the art GPU-accelerated flow solver, virtual reality (VR) interaction and a 3D object capture facility and mesh processor. These capabilities are integrated to allow flow round physical or digital objects to be simulated, viewed and interactively investigated by a user. Figure 1 illustrates the concept realised by the implementation.

1.1. Previous Virtual Wind Tunnels

The concept of a virtual wind tunnel is not new, with the 20th century work of Bryson and Levitt [6] and the patent of Strumolo and Babu [7] recognising the potential for interactive investigation tools in engineering design. However, in general, older incarnations of the virtual wind tunnel concept were built for the inspection of pre-computed data. More recently, the potential for a real-time, interactive CFD simulation is starting to be realised. The work of Bormann *et al.* [8], Wenisch *et al.* [9], Linxweiler *et al.* [10], Delbosc [11], Glessmer and Janßen [12] as well as Harwood and Revell [13–15] introduce different realisations of interactive CFD solutions for the run-time steering and simulation modification.

However, interaction and visualisation elements of these realisations have been, thus far, limited to desk-based interfaces with 'flat' interaction through widgets, touch gestures or mouse pointers. Their user interfaces deliver scientific presentations of information Fig. 2 which, although very capable, do not present a truly immersive or intuitive simulation environment for the broader user base. Conversely, the present work includes VR and incorporates its inherent ability for head and controller tracking to provide a more immersive experience for a user.

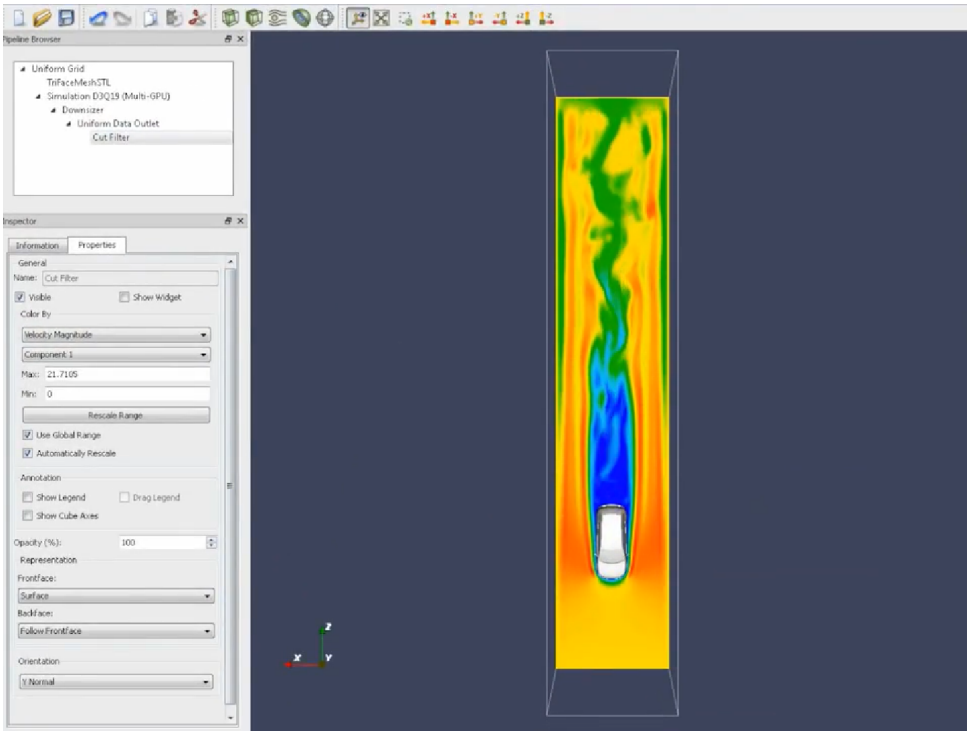


Figure 2: Scientific interface of the VirtualFluids interactive CFD application [10].

1.2. Virtual Reality, Virtual Environments and Game Engines

A virtual wind tunnel is, at least in part, an enhanced virtual environment, the technology for which has been steadily advancing. The application of 3D virtual environments in scientific fields was originally motivated by the need to collaboratively visualise spatially 3D data [16, 17]. Such environments are an effective means of communicating this type of information due to the added dimensionality perceived by humans with binocular vision [17]. Such visualisation environments are termed 'CAVEs' and have been used to communicate complex 3D architectural, medical, manufacturing and geographical data [18]. Virtual environments can be built digitally, acquired from CAD tools or scanned using depth-sensing cameras and displayed along with simulation results or other datasets.

CAVE-type solutions are usually projector-based, and hence are expensive to purchase and require a large amount of space to set up and use. More recently, virtual environments have made use of head-mounted displays (HMD) such as the HTC Vive and Oculus Rift headsets, a more affordable, compact solution. Manipulation of 3D environments can also be performed using associated controllers. The availability of software development kits (SDKs) like OpenVR [19] allow developers to easily harness the power of VR kits for a more immersive visualisation and navigation experience [20–23].

3D video games rely heavily on the construction of a realistic virtual environment to present a convincing interpretation of reality to a player. More recent video games support the integration of HMDs and VR hardware as the principal means of user interaction and researchers in a wide range of fields have been quick to exploit this potential for analysis and testing [24–29], disaster and evacuation simulation [25, 30], the simulation of physical impairment [31], computer vision [32, 33] and applications in education and training [34–37].

Video games are built and run using a collection of tools known as a game engine. Game engine packages include authoring tools for the creation of 3D worlds (a 3D editor application), the automation of game logic within a world (a scripting interface) and a comprehensive set of APIs for the management of interaction between world components (called *actors*). Game engines include powerful rendering capabilities responsible for scene transformation, shading and lighting and typically incorporate a physics engine which provides localised physical modelling. Figure 3 shows the typical components of a game engine package and also illustrates the plethora of managed capabilities. Programmers may also extend these capabilities through the definition of their own actor classes. Game engines, thus, provide a capable platform on which 3D interactive simulation tools may be built. However, many of the more mature game engines are often proprietary such as Frostbite [38], CryEngine3 [39] and Source [40] and must be licensed for commercial use. Unity3D [41] and Unreal Engine 4 [42] however, are relatively mature

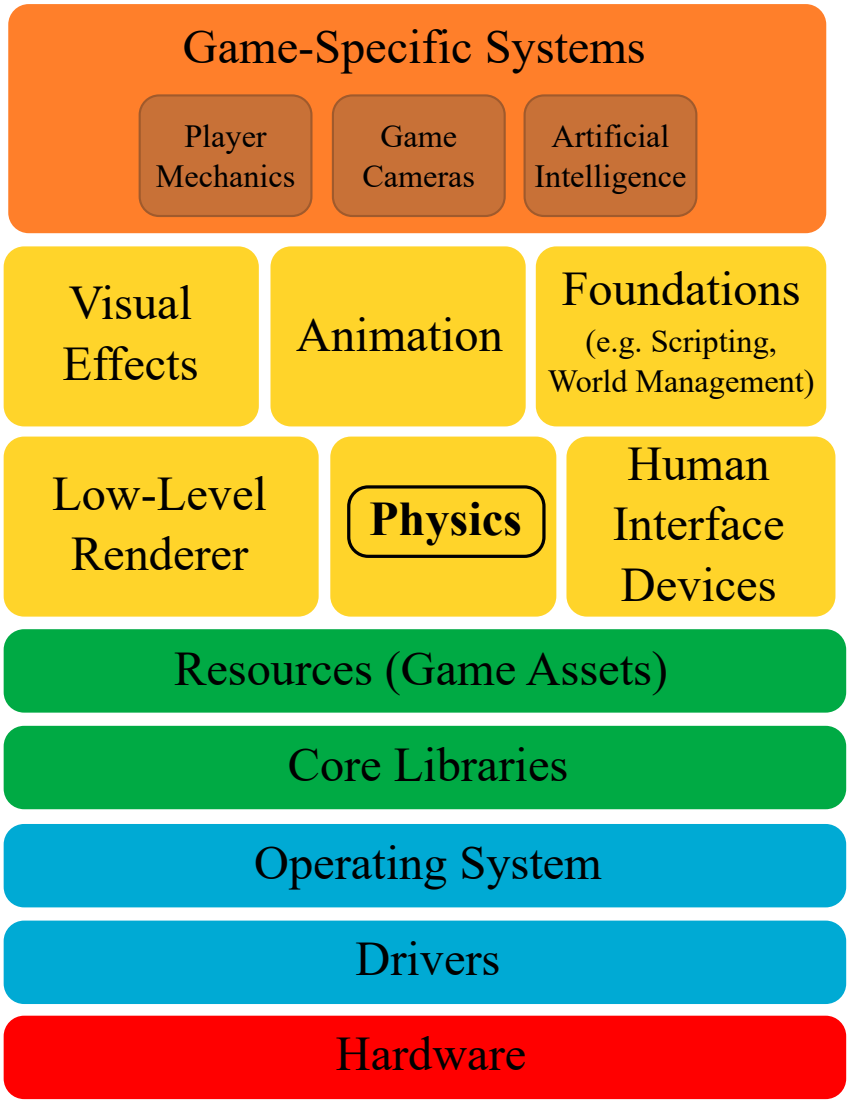


Figure 3: Illustration of a generic game engine with its key constituents.

engines that are free for educational or academic use.

1.3. Real-Time Game Engine Physics

If we propose using a game engine for visualisation and interaction purposes, it raises the question as to whether we should consider using the physics simulation capabilities available within those engines to simulate the fluid. The simulation of fluids and flexible structures in real-time is indeed offered by some game engines or by game-engine-compatible third party libraries such as Havok [43] or PhysX [44]. Existing models developed for these libraries started within the computer graphics community and offer impressive levels of performance. However, not all simulation in computer graphics, is physically-based; a simulation which looks convincing can be achieved without the need for a physically-based model. In some cases, where exaggerated behaviour is required (such as for explosions in games and films), physically-based simulation would produce an uninspiring result.

Physically-based models for fluid simulation in games and animation attempt to solve the Navier-Stokes equations on either a grid (Eulerian representation) or a set of moving particles (Lagrangian representation). In order to achieve real-time evolution, these solvers are typically implemented to run on commodity graphics hardware. Examples of Navier-Stokes solvers used for fluid simulation in games include Stable Fluids [45], FLIP [46], APEX [47] and Smoothed-Particle Hydrodynamics [48]. However, the purpose of a game physics simulation is not to provide a high-fidelity result but to provide visual plausibility. Therefore, it is often the case that simplifications are made to the equations through modelling or stringent domain size restrictions in order to maintain speed or stability, for example:

- Removal of non-linear terms in the equations.
- Restriction on the degrees of freedom.
- Use of low-order, local and/or explicit solvers with large time steps.
- Use of low-tolerance iterative solution processes.
- Approximation of fluid viscosity through numerical error.

The Flex simulation framework [49, 50], achieves very convincing, fast and stable results but does not solve Navier-Stokes at all. Instead it replaces the full governing equations by a set of particle-particle positional constraints representing physical heuristics. These constraints, although physically-based, collectively do not represent the full physics of the problem being simulated. However, the resulting numerics reduce to a linear system which can be solved rapidly with particle dynamics producing visually convincing and stable behaviour.

In summary, game physics solvers produce simulations that are fast, robust and integrate well with rendering pipelines. However, they often lack physical details due to numerical errors and artefacts produced by aggressive over-approximation. Despite their impressive speed and stability, their inflexibility and over-simplification means they lack the control over accuracy required for the standard of simulation demanded by engineers. Thus, in this work we opt to replace the flow physics engine shipped with our choice of game engine with our own physically-based LBM simulation which offers a suitable balance between accuracy and speed acceptable to engineers.

In the remainder of this article, we discuss our developments in the three key areas of technology necessary for realising the virtual wind tunnel: the solver, the object scanner and the game (c.f. Fig. 1). The flow solver is validated for a 3D turbulent channel flow in Section 2.2. We then conclude with some limitations of the current prototype in Section 6.1 and summarise in Section 7 with some remarks on the limitations as well as future trends for the underlying technology. We also identify a number of opportunities in other disciplines for which this technology could be adapted.

2. Solver Development

The flow solver component of Fig. 1 is a GPU-accelerated implementation of the Lattice-Boltzmann Method (LBM) [2]. Unlike traditional CFD methods which solve some form of the Navier-Stokes equations, the LBM represents fluid dynamics in terms of a discrete Boltzmann transport equation. Rather than the transport quantity being a macroscopic variable such as density or momentum, or a microscopic property such as the velocity of a fluid particle, the Boltzmann equation models the transport of a set of *mesoscopic* particle distribution functions f. A single value of f represents probability of finding a particle with a particular velocity $\vec{c}$ at a given location in space and time. The redistribution of these probabilities in time is enforced through a collision model Ω. The general Boltzmann equation is given in Eq. (1a).

A discrete, computational domain on which to solve this equation is represented as a uniform Cartesian grid of cells, with nodal locations at cell centres, inter-connected by a finite set of lattice links. The set of links $\{i\}$ dictates admissible directions along which individual f_i quantities may be convected. A set of vectors $\{\vec{c}_i\}$ are associated with the set of lattice links.

After discretisation, the Boltzmann equation may be expressed as in Eq. (1b). The left-hand side of Eq. (1b) represents the convection of f_i along lattice link i to an adjacent node – termed the *streaming* step. The values to be convected are computed by performing a collision operation Ω on the current set of f values at a given node as per the right-hand side of the equation. The collision operation is carried out by relaxing the current distribution functions towards a local equilibrium distribution f^{eq} which is a second-order expansion in the macroscopic velocity of the Maxwell-Boltzmann distribution [51]. This part of the evaluation is termed the *collision* step. An illustration of an LBM time step is given in Fig. 4.

$$\left(\frac{\partial}{\partial t} + \vec{c} \cdot \nabla\right) f = \Omega \tag{1a}$$

$$f_i(\vec{x}, t) - f_i(\vec{x} + \vec{c}_i \Delta t, t + \Delta t) = \Omega(f_i, f_i^{eq}) \tag{1b}$$

Macroscopic quantities are related to the mesoscopic distribution functions through their first two moments whose discrete forms are given in Eq. (2).

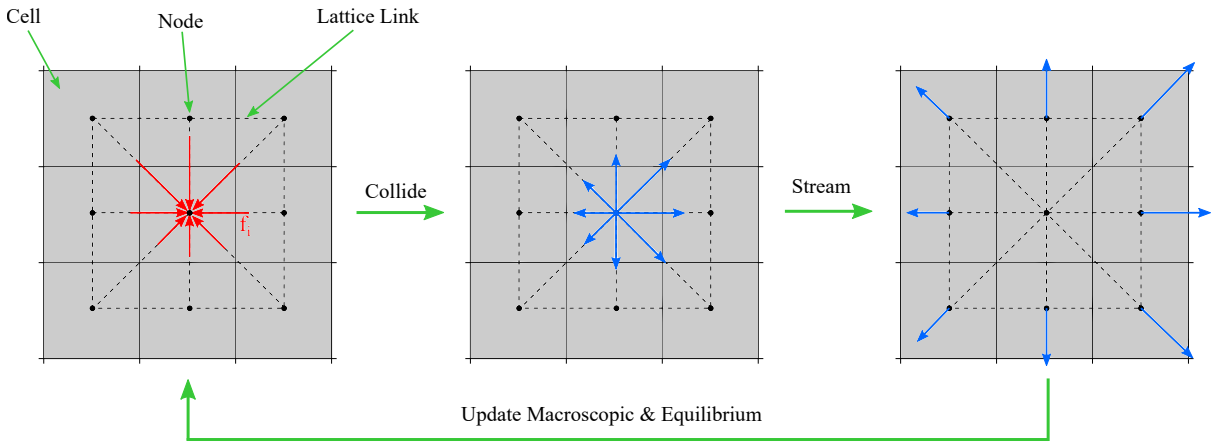


Figure 4: Illustration of the LBM with discrete cells shown as shaded blocks, lattice links as dashed lines the distribution functions f as coloured arrows. Populations before collision are red, populations after are blue.

$$\rho = \sum_i f_i \tag{2a}$$

$$\rho u_j = \sum_i c_{i,j} f_i \tag{2b}$$

The LBM recovers the quasi-incompressible Navier-Stokes equations at the macroscale so long as the lattice Mach number remains small. This is due to the fact that the equilibrium function used in the collision process is developed through a local, second order expansion in the velocity. Its application is, therefore, limited to incompressible flow at present with compressible extensions of the method an area of active research [52].

2.1. Implementation on the GPU

The main appeal of LBM for CFD researchers is the spatial locality and simplicity of Eq. (1b). This allows the method to be parallelised on graphics hardware and executed much more rapidly than traditional CFD methods which often require highly non-local interpolation and differencing stencils. This work uses LBM due to its ability to provide accurate modelling for the engineering community while offering the potential for acceleration up to real-time speeds for interactive applications.

The suitability of LBM for general purpose GPU (GPGPU) computing has seen numerous, highly-efficient implementations of LBM on GPU in recent years [4, 53–58]. Although initially developed using graphics APIs [59], higher-level APIs for GPGPU computing (such as CUDA from NVIDIA [60]) have, more recently, made this pursuit even more accessible to researchers from a range of disciplines.

In order to maximise the performance of the LBM algorithm on a GPU, the limitations of GPU hardware need to be understood. The single-instruction-multiple-thread (SIMT) execution model used by GPUs is most efficient when all threads (i.e. lattice nodes, assuming one-to-one matching between threads and nodes) perform the same set of instructions. Hence, branch divergence should be avoided. Furthermore, the LBM requires each thread to read and write each population to and from GPU main memory. This is a costly operation and hence the algorithm should be implemented to minimise global memory operations and use as much on-chip, cache memory as possible. These, along with other established guidelines for performance are summarised neatly in [11].

The performance of an LBM algorithm can be measured in terms of the number of lattice nodes which can be updated (complete stream and collide) in a physical second of wall clock time. This measure is given the units Million Lattice Updates Per Second (MLUPS) and is computed by using wall clock timers in the code as

$$\text{MLUPS} = \frac{N \times T}{t_w} \tag{3}$$

where N is the total number of lattice nodes and t_w is the wall clock time for executing T time steps. The LBM on GPU is memory-bound [56] and hence throughput, as measured in MLUPS, will only increase up to a point at which the memory bandwidth of the device becomes saturated and execution begins

Authors	Year	GPU	MLUPS
Tölke *et al.* [53]	2008	8800 Ultra	200-600
Mawson [3]	2013	Tesla K20	700-1000
Mawson [3]	2013	Quadro K5000m	300-400
Delbosc [11]	2016	Tesla K40	1480
Delbosc [11]	2016	GTX 780Ti	1785
Glessmer & Janßen [12]	2017	Quadro M6000	450-700
Tran *et al.* [58]	2017	Tesla K20	1000-1200
Present Work	2018	GTX 1080Ti	1000-1700

Table 1: Single-precision performance of recent 3D GPU-accelerated LBM implementations. Range of performance due to different modelling and launch parameter configurations.

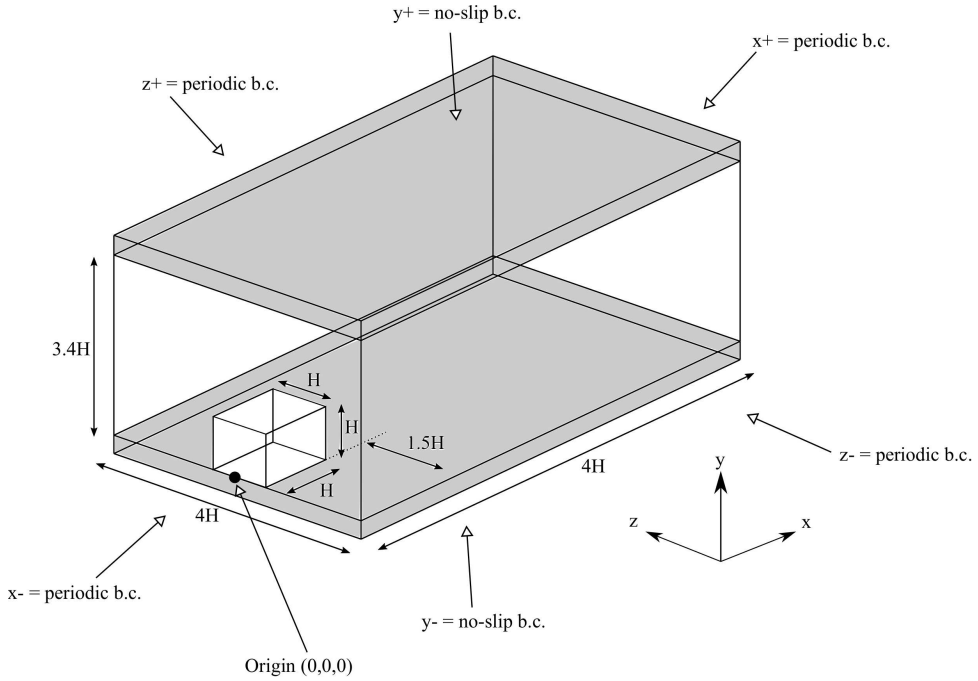


Figure 5: Case configuration for 3D turbulent channel flow with array of wall-mounted cubes.

to queue while memory transactions are carried out. The typical performance of a 3D, single-precision LBM implementation on a single GPU are tabulated in Table 1 from the literature. Peak performance can be anything up to 1800 MLUPS depending on the device and model complexity used. We include our own performance results to Table 1 for comparison and find them to be in line with the state of the art with improved performance over existing implementations largely due to use of the next generation of GPU hardware.

To maximise performance, the LBM configuration used in this work is deliberately basic. The BGK collision operator [61] is used, which is the least expensive from a computational perspective. Rigid wall boundaries, are implemented as simple bounce-back boundary conditions [62], which are second-order accurate in the cell size. A Smagorinsky turbulence model [63] is also implemented to provide additional stability and flow through the domain may be introduced using either the forcing scheme of Guo [64] or a forced-equilibrium inlet/outlet boundary. In the latter, the values of f are set equal to $f^{eq}(\rho, \vec{u})$ where ρ and $\vec{u}$ are desired free-stream conditions. These configurations give a stable and efficient flow simulation with a good degree of accuracy.

2.2. Validation and Performance

In order to validate the solver, we simulate a 3D turbulent channel containing an array of rigid, wall-mounted cubes. This case is representative of an external flow around an object in a wind tunnel. Experimental data is available from Meinders and Hanjalič [65] and Hellsten *et al.* [66] for the purposes of comparison. The simulation domain is shown in Fig. 5.

The number of lattice sites based on the reference length $H = 50$ This resolution gives a total number of cells of approximately 6.8M. Body forcing is used to accelerate the flow from rest which is iteratively adjusted to ensure the volume flow rate computed over the inlet face matches that of the reference case. When the mass flow rate through the domain is tuned to within $< 1\%$ of the target value, the simulation is run for 200 domain flows while velocity components and their 3D first order products are cumulatively averaged in time. The Smagorinsky model available in the solver is enable to account for the coarse resolution used. Table 2 documents the parameters used for the simulation. The simulation is executed on eight NVIDIA GTX 1080Ti GPUs.

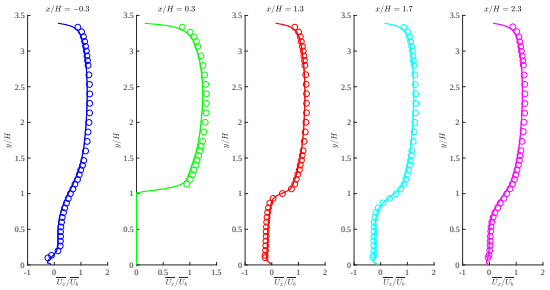
Parameter	Value
Cube Height H	$0.015m$
Reference Velocity U_{ref}	$3.86m/s$
Reference Time t_{ref}	$0.00389s$
Dimensionless Force	0.006229
Smag. Constant c_s	0.3
Lattice Spacing dx	0.02
Dimensionless Timestep dt	0.0011
Reynolds Number	3831
Lattice Viscosity ν	0.000718
Relaxation Time τ	0.502153

Table 2: Values used to simulate the 3D turbulent wall-mounted cube case of [65].

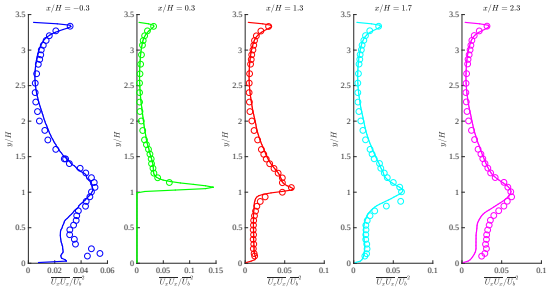
The peak performance of the solver on a single GPU has been presented against other implementations in Table 1. However, based on the observations of Harwood and Revell [14], it is essential to be able to extend the calculation over multiple GPUs to maintain the required evolution rate of the simulation as resolution increases. The domain is decomposed in 1D in the Z-direction. A halo region is used to pass data between devices and is chosen to be 16 cells thick based on the recommendation of [67].

2.2.1. Results

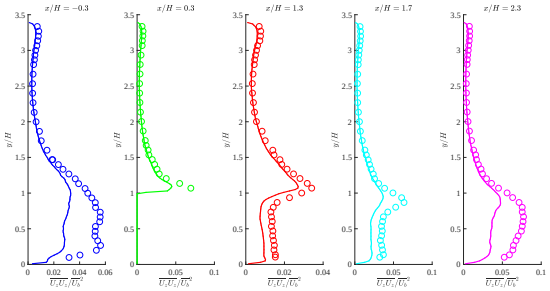
The stream-wise and span-wise velocity profiles are plotted in a selection of vertical and horizontal locations. All values of velocity are normalised with respect to the bulk velocity $\overline{U_b}$, computed by integrating the time-averaged stream-wise velocity $\overline{U_x}$ over the inlet face of the domain and dividing by the area. Vertical slices are shown in Fig. 6 and horizontal slices in Fig. 7.
The simulation results show excellent agreement with the experimental results despite not using any wall modelling and only a modest resolution.



(a) Averaged U_x velocity at different stream-wise locations compared with the experimental results.

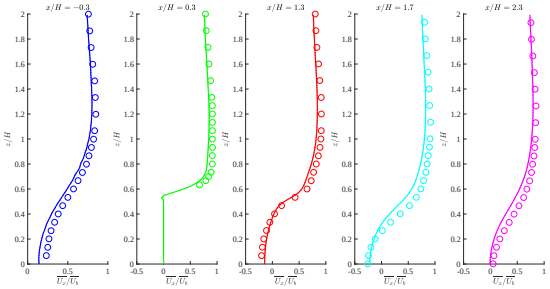


(b) Averaged $U_x'U_x'$ stress component at different stream-wise locations compared with the experimental results.

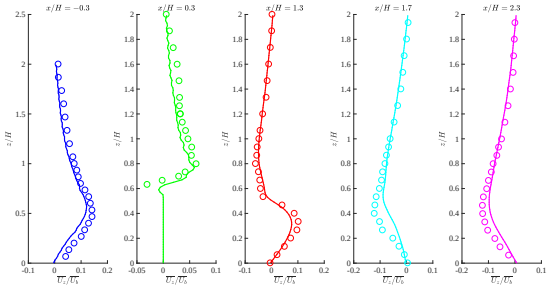


(c) Averaged $U_z'U_z'$ stress component at different stream-wise locations compared with the experimental results.

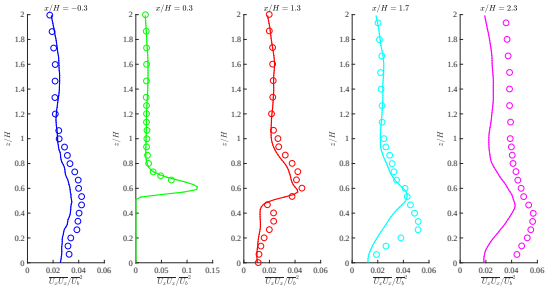
Figure 6: Simulated velocity profiles and Reynolds stresses (solid line) along vertical lines compared with experimental data of [65] and [66] (circles).



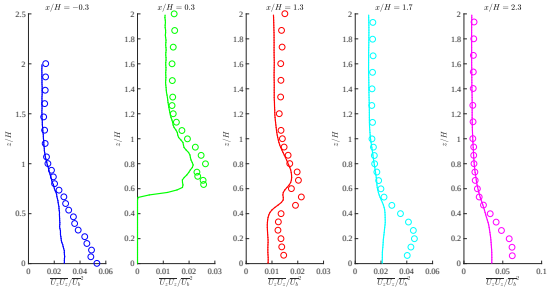
(a) Averaged U_x velocity at different stream-wise locations compared with the experimental results.



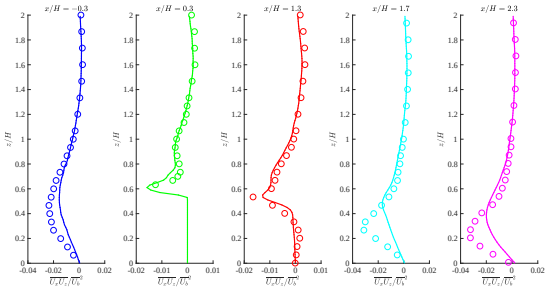
(b) Averaged U_z velocity at different stream-wise locations compared with the experimental results.



(c) Averaged $U'_x U'_x$ stress component at different stream-wise locations compared with the experimental results.



(d) Averaged $U'_z U'_z$ stress component at different stream-wise locations compared with the experimental results.



(e) Averaged $U'_x U'_z$ stress component at different horiz stream-wise compared with the experimental results.

Figure 7: Simulated velocity profiles and Reynolds stresses (solid line) along horizontal lines (half channel width as symmetric) compared with experimental data of [65] and [66] (circles)

3. Object Capture

In order to simulate the air flow around a wide variety of objects, our object capture software is able to construct compatible representations for objects supplied digitally or physically. The former is usual in engineering where products or components are routinely built digitally. However, the latter capability enables the simulation of flow around objects for which a digital counterpart does not exist. Examples could be clay-sculpted design prototypes in the automotive industry or pupil-built models in a classroom environment.

The aim of the object capture library component of the virtual wind tunnel is to capture surface information and to translate this information into two necessary representations:

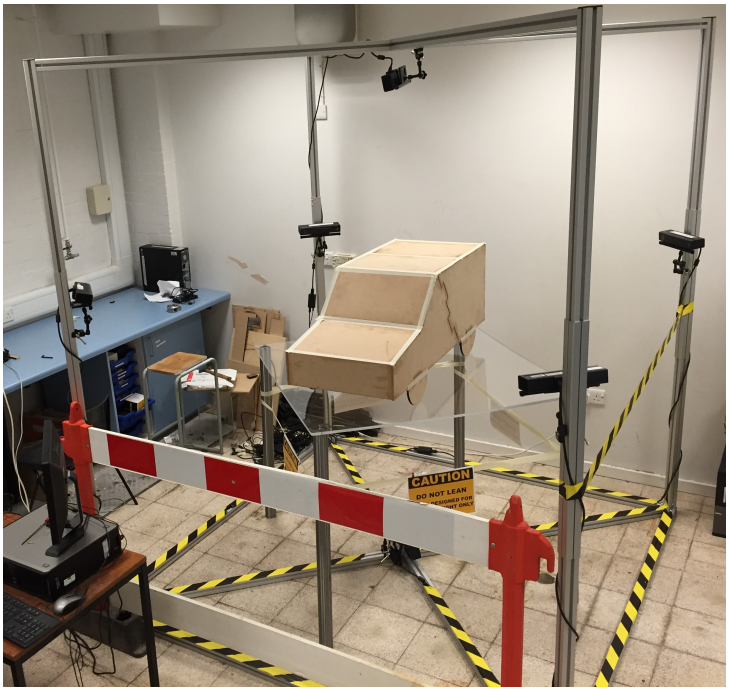
- A suitable set of boundary conditions for the solver
- A suitable visual mesh for the game

Justification for these two representations is detailed in Section 5.2 as it relates to the integration strategy employed. Surface information is readily available from CAD software in the form of an STL file export. However, capturing reliable surface data for physical 3D objects is challenging. Depth-sensing camera technology is a potential solution, with the Kinect camera by Microsoft shown to be a capable, low-cost sensor [68–72]. A free software development kit (SDK) is provided to enable interfacing with the hardware. It is a popular choice for indoor robotics, object recognition and 3D scene reconstruction [73].

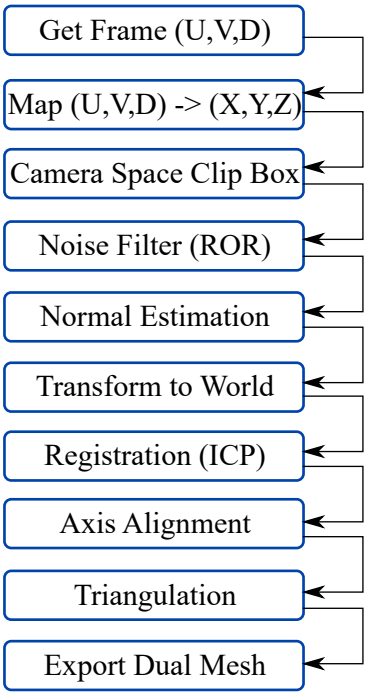
The Kinect camera uses triangulation of a structured light pattern reflected from the surface of a physical object to capture surface data. The field of view of the Kinect 2 camera is able to capture the 3D position of $512 \times 424 = 217,088$ points [74]. In order to obtain a complete picture of the object, we use an aluminium frame to position six Kinect cameras around a transparent table on which physical objects may be placed.This laboratory set up is shown in Fig. 8a. The transparent table allows both the capture of the bottom surface of objects and for objects to appear in the centre of the capture space.

Once capture has taken place, the point clouds from each camera are registered using the Point Cloud Library (PCL) [75]. As position and orientation of the cameras is fixed, the cloud may be automatically clipped to the capture space and the cloud aligned to the floor using calls to the PCL API. In order to enable floor alignment, the relative translation and orientation of one of the cameras (used as a reference camera) must be known in advance. Its point cloud is then the first to be transformed so it aligns with the desired global reference frame (level with the table-top and facing the 'forward' direction). This orientation is determined when setting up the lab by performing a calibration test; a cube is scanned and then an approximate transform applied to the global reference frame. This transformation is then successively refined until the point cloud faces are determined to be oriented correctly with the global reference frame. The approximate transformation is necessary to ensure convergence o the desired orientation. The transform is then stored and used to calibrate the registration process.

Data obtained from Kinect, like all optical depth-sensors, can be noisy, particularly when structure light falls on sharp edges or interfaces between materials of contrasting reflectivity. Automatic noise removal filters, such as a Radial Outlier Removal (ROR) filter, can be applied although tuning the algorithms involved for consistent performance is difficult. Alternatively, manual intervention can be employed to clean up the registered point cloud. The final step implemented by the object capture library is to employ the ball pivoting algorithm [76] to triangulate the point cloud to form a surface mesh which may be exported as an STL file. The complete processing pipeline is depicted in Fig. 8b.



(a) Photograph of the laboratory set up for scanning 3D physical objects.



(b) Breakdown of the steps involved when processing the data acquired from the Kinect cameras. All steps can be achieved through the PCL and Kinect APIs.

Figure 8: A photograph of the scanning laboratory and a summary of the processing pipeline used to transform the data from the cameras into point cloud and surface mesh assets.

4. Game Development

Unreal Engine 4.16 (UE4) is chosen as the game engine on which to base the application. This engine is considered most suitable due to its maturity and wide range of capabilities. As the intended application for the game is to simulate a wind tunnel, a suitable map (`UMap`) was created in the editor using custom mesh assets (`UAssets`) to reflect this virtual environment. These assets were created using the 3D modelling software Blender [77] and imported from an FBX format into the editor along with generated UV maps and collision meshes. Suitable materials were assigned and any dynamic behaviour (such as a rotating fan) programmed using the visual scripting interface within the UE4 editor. A single player was added to the map with controller configurations inherited from the built-in VR classes suitable for use with the HTC Vive VR hardware. Particle-based visual effects were placed upstream in the tunnel to provide smoke streaks, a common technique for experimental flow visualisation and often replicated in virtual simulation environments [78].

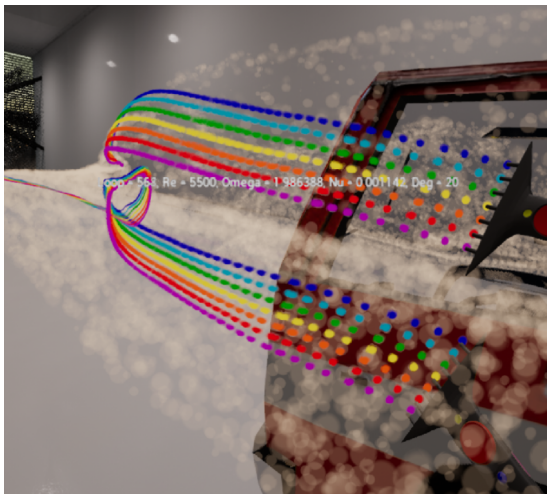
4.1. Player Capabilities

For the game to be useful as a design and analysis tool, users must have a degree of run-time interaction with the environment and the solver. An in-game menu is the main device for interacting with the simulation. However, some capabilities are mapped to controller buttons.

In addition to the ability to cycle through different objects at run-time, the user may rotate the object to investigate the affect on flow angle on flow behaviour. This rotation is implemented in just a single plane at present although its extension to other axes is trivial. The rotation of the object can be reset at any time from the menu.

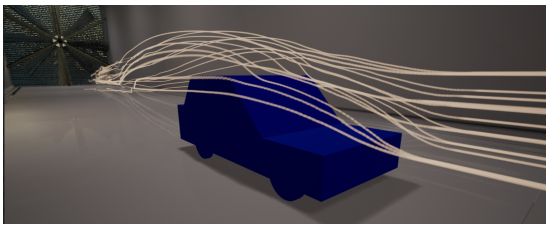
Some VR systems (like the HTC Vive) provide a tracking system to allow a user to physically move around the space through walking in the real world. This positional tracking is harnessed to allow the user to move around inside the wind tunnel in the same way. However, as the environment is larger than the capture region for the VR positional tracking, a teleportation system was implemented. Triggered by controller buttons, a user may glide forwards or backwards in the direction they are facing to allow them to traverse larger distances.

(a) Single-nozzle wand

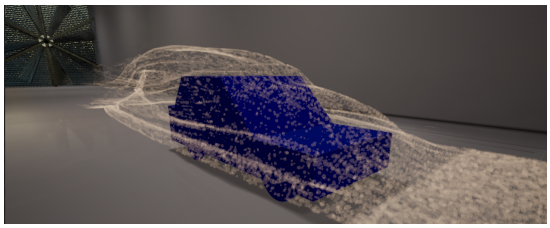


(b) Multi-nozzle wand

Figure 9: Screen capture of a user using the smoke wand capability to visualise the 3D flow field around the object.



(a) Streak Line configuration



(b) Smoke Sheet configuration

Figure 10: Demonstration of the two different tunnel smoke configurations. The smoke sheet may be moved in the vertical plane at run-time.

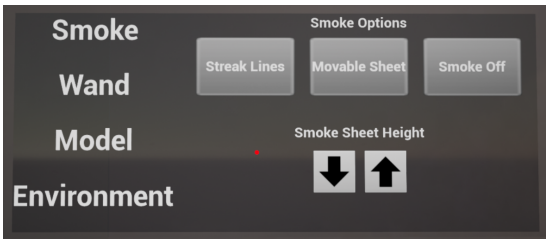
The visual representation of the controller in the game is changed for a mesh asset build to look like a smoke wand. The controller trigger may be used in the same way to activate a particle system attached to the controller tip. A screen capture of this capability in use is shown in Fig. 9. The user may also swap the single-nozzle wand for a multi-nozzle rake wand with seven, coloured particle streaks whose smoke provides an indication of flow rotation.

The general smoke visualisation in the tunnel may either be in the form of smoke streaks or a smoke sheet. The latter may be positioned at regularly-spaced, discrete locations above the tunnel floor. Both configurations of tunnel smoke are shown in Fig. 10.

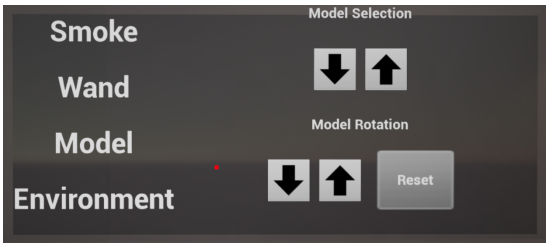
Finally, the user may adjust the Reynolds number (velocity) of the simulated fluid flow where the Reynolds number is defined based on the inlet velocity and the tunnel height.

4.2. User Interface Design

Mapping capabilities to controller buttons soon becomes impractical as the number of interactions increase. Hence, we proposed an in-game menu attached to the top of the left-hand controller. It may be shown or hidden using the a button on the controller. The options are sorted into categories that each have a corresponding pane of buttons which can be displayed on the right-hand side of the menu widget. The left-hand side thus offers category selection. The user may point at a button on either pane and press a controller button to select it. The four menu panes are shown in Fig. 11 containing all of the features of the tunnel.

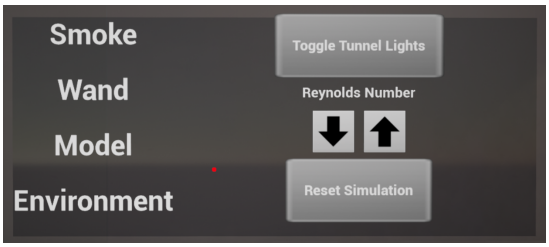


(a) Smoke menu



(b) Model menu

(c) Wand menu



(d) Environment menu

Figure 11: Screen captures of the different menu views currently implemented in the virtual wind tunnel user interface.

5. Component Integration

Having described the individual components required to build the virtual wind tunnel, we next describe our integration of these components and highlight the general challenges this poses. An overall schematic of how the components integrate is shown in Fig. 12

5.1. Solver Integration

To ensure modularity of the software, the LBM solver is compiled as a standalone library encapsulating all the dependencies required to run a flow calculation. A façade software design pattern is the used to provide an API for the library as shown in Fig. 13. This decision allows the LBM solver to be embedded in any number of external applications without the need to reference any third-party dependencies.

However, in order to allow the game to access the data provided by the library, a new *actor* is created and embedded inside the game world. This actor is added to the map through the editor and given a bounding box visual representation which correspond to the computational domain boundaries. Its behaviour is defined such that the game engine will *tick* the actor during run-time execution. This provides a mechanism for synchronised triggering of simulation iterations at the maximum update rate possible. We call this actor class `LbmPhysics` and game components may interact with it as with any other actor in the game. It is defined in Appendix A and is responsible for invoking LBM library methods. A view of the virtual wind tunnel map with the `LbmPhysics` actor highlighted is shown in Fig. 14.

Finally, in order to have the particle streaks in the game describe the simulated field, the `LbmPhysics` actor is assigned a `UVectorFieldComponent`. This is a UE4 actor, distributed with the engine, capable of storing a 3D vector field. Particle systems in the map interpret this vector data as a velocity field and will trace the vectors when passing through its region of influence. The contents of the vector field are updated through the library API where vector field cells pull data from the underlying LBM grid.

5.2. Object Integration

The object capture module, like the solver, is built as a library and referenced by the both the game and the solver to allow access to its capabilities at run-time. The module is capable of generating both a point cloud and an STL surface mesh via run-time calls to the API. Both the solver library and the game need to be made aware separately of the change of object event. This is because the game represents object as a `UStaticMeshComponent` attached to a static mesh `UAsset` and the solver represents the object as a set of solid wall boundary conditions. These two different representations therefore need handling in two different ways.

The assets representing the objects in the game (e.g. mesh, materials and textures) must be present in the game content directories before launch. The `LbmPhysics` actor is instructed to load these assets on request, thus changing the visual representation of the object the user sees. As most CAD software used

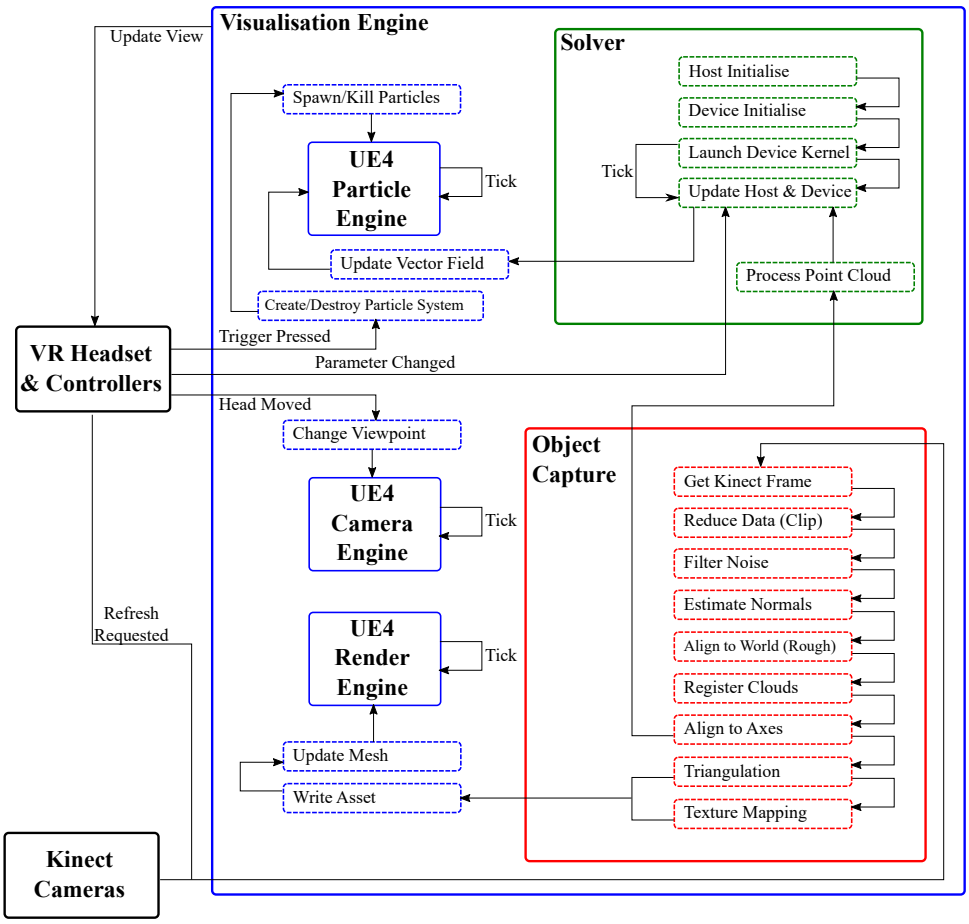


Figure 12: Schematic outlining how the solver and capture components integrate within the game.

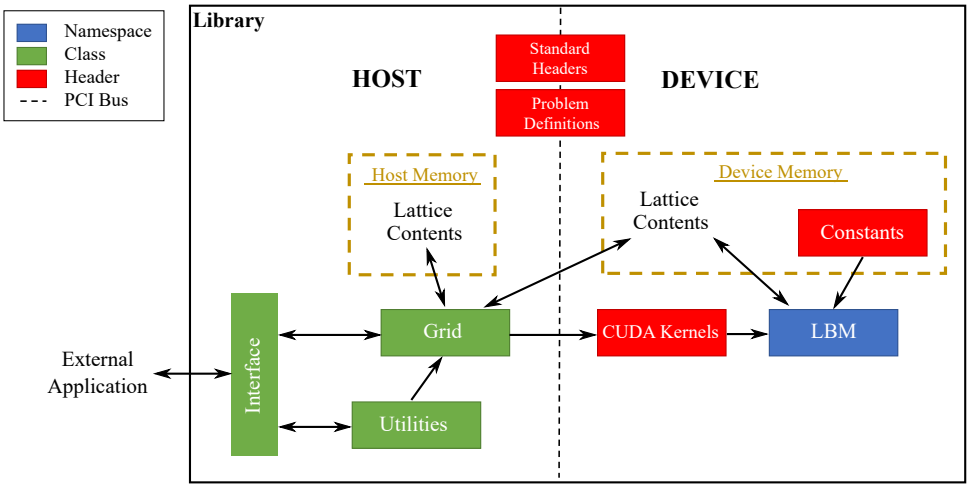


Figure 13: Illustration of the structure of the GPU-accelerated LBM library used as the physics solver in the virtual wind tunnel.

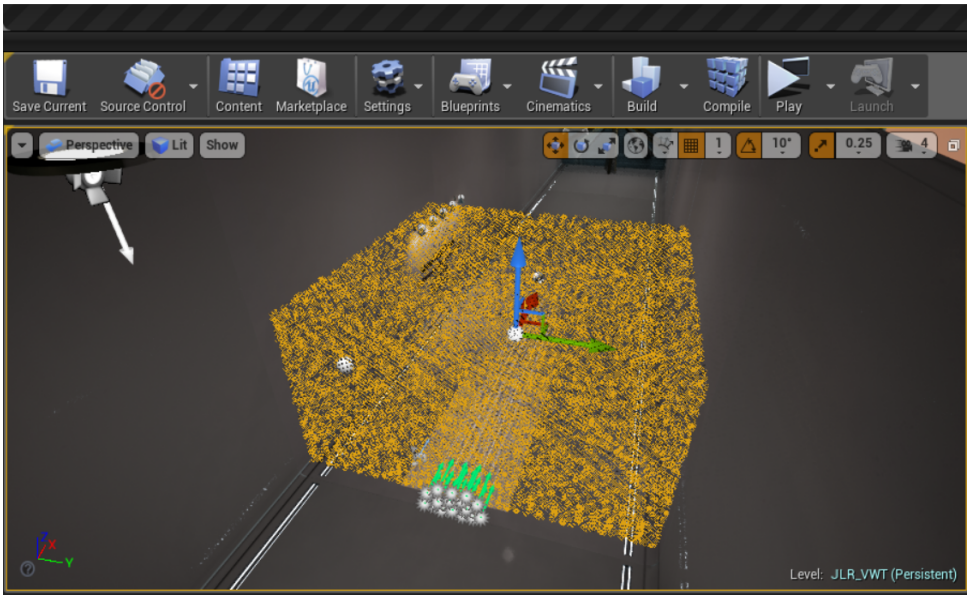
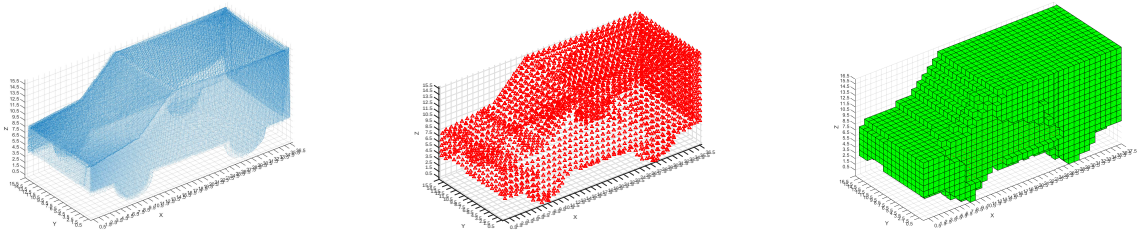


Figure 14: The virtual wind tunnel map as seen in the UE4 editor. The custom `LbmPhysics` actor for interfacing with the physics solver is highlighted in orange.



(a) Raw point cloud (b) Voxelised representation (c) Solid boundary condition

Figure 15: Illustration of the stages in mapping a point cloud, under the voxel grid filter, to an LBM, label-based boundary representation.

in engineering will most likely export meshes without a texture (UV) map, open-source tools such as Blender [77] must be used to prepare the mesh accordingly and must be done as a pre-processing step offline.

In order to generate the boundary conditions for the LBM solver, the point cloud produced by the object capture pipeline is passed to the library at run-time. The library then applies a voxel grid filter to the data, essentially sampling it to the LBM grid and labelling cells which contain at least one point as solid. A very course, illustrative example of this process is shown in Fig. 15.

Rotation of the object is therefore implemented as two sequential function calls by the game:

1. rotate the visual mesh asset in the game
2. reinitialise the boundary conditions in the LBM simulation

The consequence of this is that the visual representation of the object is a fixed mesh with a static representation of the boundary, whereas the representation of the object in the solver is resolution-dependent; if resolution of the simulation is very course, definition of the object seen by the simulation reduces despite the visual mesh remaining unaltered. Alternate representations such as BFL-type boundary conditions [79] can be used to improve the accuracy but at the expensive of slowing the solver down due to additional effort in computing the flow at the boundaries.

6. Measuring the Performance of Interactive Simulations

In Table 1, we quantified the performance of our CFD solver in terms of the metric MLUPS, an interpretation of computational throughput favoured by the LBM community. However, when measuring

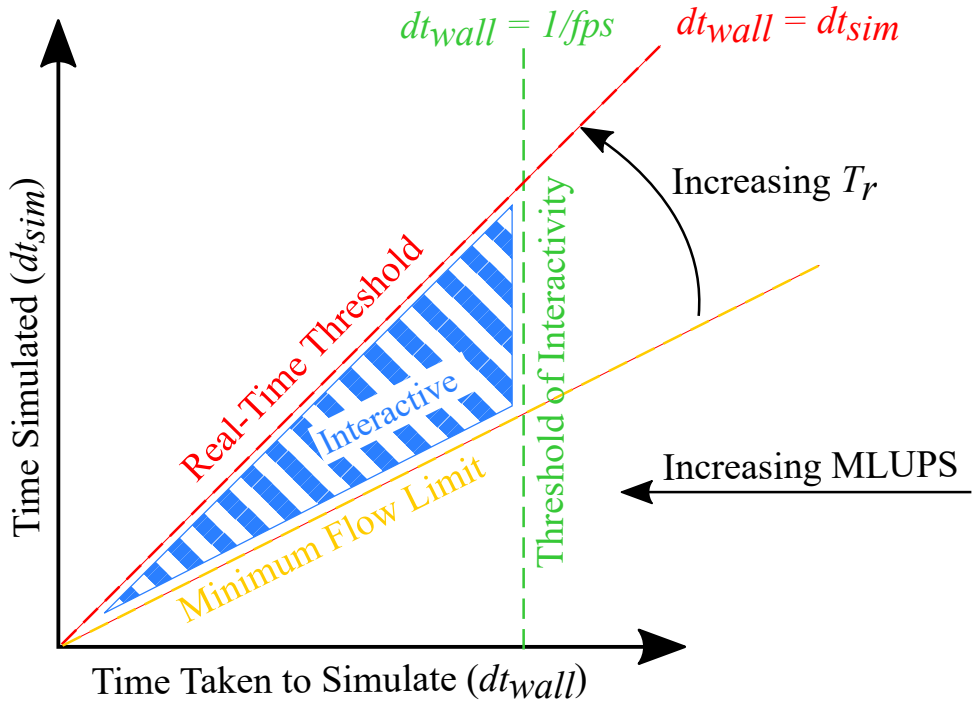


Figure 16: The relationship between wall clock time and simulated time and identification of the target range of real-time ratio T_r for interactive simulations. T_r represents the gradient of any line drawn on the axes. An increase in the traditional throughput measure of the LBM community (MLUPS) moves any point on the graph to the left. The Threshold of Interactivity represents the lowest tolerable frame rate for continuous viewing and the Minimum Flow Limit represents the lowest tolerable rate of convection of a structure of interest in the flow being studied.

the performance of interactive simulations, this is not sufficient. In [14], the ratio of the time step used in the simulation dt_{sim} and how long it takes our computing hardware to simulate that time step dt_{wall} is defined as the *Real-Time Ratio* T_r A $T_r \leq 1$ defines a simulation running at or slower than real-time, typically the region achievable for interactive simulations at present depending on the accuracy required.

Here we extend the concept of real-time ratio further by considering the use case of interactive simulations: a typical user wants to run a simulation on a given flow problem such that

- They can smoothly view a quantity of interest while the simulation is running;
- Structures of interest convect through the visualised domain at an appropriate speed for study;

The first requirement imposes a maximum tolerable wall clock time before the results are updates so infrequently that visualisation will appear 'choppy'. Typically, humans require at least 24 frames per second (fps) for smooth viewing. We define this limit as the *Threshold of Interactivity*. The second requirement imposes a limit on the T_r. For a given time scale in the simulation (represented by dt_{sim}), the throughput should be sufficiently high (or wall clock time sufficiently low) to update the results frequently enough that structures of interest move at an appropriate speed. We define this as the *Minimum Flow Limit*. Thus we have bounded our simulation configuration below the *Real-Time Threshold* $\equiv T_r = 1$ based on the user requirements as depicted in Fig. 16. It is important to note, that increasing the throughput of computing devices will reduce the dt_{wall}, driving points to the left of the graph. This will also increase T_r accordingly.

Using these principles, we can identify for a range of scenarios how effective our virtual wind tunnel implementation is for interactive simulation. The relevant tests are deferred for the production article.

6.1. Limitations

The implementation presented here is a prototype system originally designed for automotive external aerodynamic design. However, there is huge potential for extension and migration to other applications. In particular, this concept may be used with little modification to read in medical geometry and to simulate the flow of bodily fluids. Thus, the tool may be used as an interactive tool for medical analysis. However, there are a number of limitations associated with the prototype which need to be addressed before applications are be broadened.

Reliance on existing but inflexible UE4 visualisation components means that the loose coupling between the solver and visualisation degrades utility. Custom visualisation of the flow field may be implemented by modifying the engine code. Shaders could be written to take GPU field data and set pixel values accordingly [15, 80]. This tight integration of simulation and visualisation is a key enabler for the technology.

When using the `VectorField` actor in UE4 to control the particle motion, a new vector field must be constructed anew from GPU data using the classes provided by the engine and swapped with the current version at flow update intervals. In practice, this requires both the passing of information between the device (GPU) and the host as well as the repacking of data into the `VectorField` structure. This bottleneck can be treated in future in two stages: first, the LBM data on the GPU must be structured such that it maps directly onto the vector field source data without the need for reassembly on the host; second, by modifying the particle simulation classes in the game engine source to allow direct swapping of vector field source data resources.

Furthermore, a single vector field is restricted in size by the largest texture unit supported by the engine (128^3). This limits the overall size of the simulation if the `LbmPhysics` actor were to support just a single vector field. A restriction on resolution, would therefore restrict accuracy. This issue may be addressed by attaching multiple vector fields to a single `LbmPhysics` actor with vector field construction performed using asynchronous background tasks to maintain parallelism.

Finally, using a game engine as the main time marching mechanism means that solver iterations are directly coupled the frequency of the game. This frequency is generally set at between 60-90Hz. Even with sub-cycling the solver, the frequency with which the simulation advances may not be high enough for real-time simulation. Although it is not necessary to be real-time (but merely interactive) in some cases [81], the game and solver ought to be coupled asynchronously to remove this tight coupling in time and allow the solver to run as fast as possible.

Finally, UE4 games must have all their visual mesh assets at compile-time. This is a UE4 specification to enable efficient rendering of 3D objects, however, it does prevent the generation of new objects for the virtual wind tunnel at run-time in the present iteration.

7. Conclusions

This paper has presented an real-time interactive, virtual wind tunnel for engineering design and development. The design and implementation has been detailed and validation and performance of the underlying LBM solver given. Simulated results for the turbulent flow over a cube in a channel at a modest resolution with no wall modelling shows excellent agreement with experimental data. Execution of the 6.8M cell calculation was performed on commodity GPU hardware. A second simulation, with no modelling was performed on a turbulent channel of $Re_\tau = 180$ and showed near perfect agreement with DNS data.

The virtual wind tunnel integrates object scanning and CAD import work-flow into a 3D wind tunnel world, offering in-game user interaction with the real-time simulation kernel and visualisation customisation through an in-game menu. Interaction within the virtual wind tunnel game is routed to the solver through a single façade class which allows it to be used in different games with potentially different contexts. A range of smoke-based visualisation options are available and controllers may be used as smoke wands for unobstructed, flexible investigation of the flow behaviour.

Performance bottlenecks in the design of integrating into UE4, which inhibit scalability, have been identified and are the focus of current research. Possible avenues for application extension have also been presented. As the LBM component is built as a library, it may already be used in other contexts with little or no modification.

Finally, the use of virtual reality and game-engine integration provides a novel, immersive experience for visualisation and huge potential for intuitive investigation and analysis. Coupling this mechanism with a physically-accurate solver approaching real-time speeds in 3D, has created a novel and powerful tool with a high potential for impact and future extension.

Acknowledgements

This work was supported by Engineering and Physical Science Research Council Impact Accelerator Account (grant number: EP/K503782/1).

A. LbmPhysics Actor

The `LbmPhysics` actor used to couple the solver to the game is defined as follows:

```
// ************* LbmPhysics.h ************

// New structure to pass pointers to LBM data to a VectorField constructor
class DataPointers
{
        public :
        int32 DataX;
        int32 DataY;
        int32 DataZ;
        FBox Bounds;
        int32 numPoints;
        float const * const X;
        float const * const Y;
        float const * const Z;

        // Constructor with initialisers
        DataPointers(int32 Nx, int32 Ny, int32 Nz, const float *lbmBounds,
        const float *Ux, const float *Uy, const float *Uz)
        : DataX(Nx), DataY(Ny), DataZ(Nz), X(Ux), Y(Uy), Z(Uz)
        {

                // Assign bounds
                Bounds.Min.X = *lbmBounds++;
                Bounds.Min.Y = *lbmBounds++;
                Bounds.Min.Z = *lbmBounds++;
                Bounds.Max.X = *lbmBounds++;
                Bounds.Max.Y = *lbmBounds++;
                Bounds.Max.Z = *lbmBounds++;

                // Number of points
                numPoints = Nx * Ny * Nz;

        };

};

UCLASS()
class JLR_VWT_API ALbmPhysics : public AActor
{
        GENERATED_BODY()

        public:

        //////////////
        /* METHODS */
        //////////////

        // Constructor and destructor
        ALbmPhysics();
        ~ALbmPhysics();

        // Called when the game starts or when spawned -- reads in all the static
        meshes representing the object visuals; binds to the pointers from where
        the velocity data can be found to contruct the vector fields
        virtual void BeginPlay() override;
```

```
496
497         // Called every frame -- fires the LBM solver time step and recreates
498         vector field in the game
499         virtual void Tick(float DeltaSeconds) override;
500
501         // Methods to interact with the solver that must be accessed through
502         blueprints fired from menu selections -- these are essentially calls to the
503         solver API
504         UFUNCTION(BlueprintCallable, Category = "Interaction")
505         void setRe(float reynolds);
506
507         UFUNCTION(BlueprintCallable, Category = "Interaction")
508         void resetSim();
509
510         UFUNCTION(BlueprintCallable, Category = "Interaction")
511         void rotateObject(float degrees, bool replace);
512
513         UFUNCTION(BlueprintCallable, Category = "Interaction")
514         void swapObject(int32 objectidx);
515
516         UFUNCTION(BlueprintCallable, Category = "Solid Objects")
517         int32 getNumberOfObjectsAvailable();
518
519         // Method to create a VectorFieldStatic with the data provided as a
520         MyFFGAContents structure
521         UObject* CreateVectorFieldStatic(DataPointers *Data, UClass *InClass,
522         UObject *InParent, FName InName);
523
524         // Method to build a PC from a file
525         bool readPointCloudPoints(PCpts* &_PCpts, FString filename);
526
527
528         ////////////////////////
529         /* SOLVER PROPERTIES */
530         ////////////////////////
531
532         // LBM solver interface
533         Lumis* LumisInterface = nullptr;
534
535         // Reynolds number
536         UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "LBM")
537         int32 ReynoldsNumber = 5000;
538
539         // Update frequency
540         UPROPERTY(EditAnywhere, Category = "LBM")
541         int32 TicksPerRefresh = 10;
542
543         // Fudge factor
544         UPROPERTY(EditAnywhere, Category = "LBM")
545         float FudgeFactor = 20.0f;
546
547         // Current object selected
548         UPROPERTY(BlueprintReadWrite, Category = "Solid Objects")
549         int32 currentObject = 0;
550
551
552         ////////////////////////////
553         /* VECTOR FIELD PROPERTIES */
```

```
554         ////////////////////////////
555
556         // Structure with info for building a vector field object
557         DataPointers *FieldData;
558
559         // VectorField(s) that represent the LBM solver velocity field
560         TArray<UVectorFieldComponent*> VelocityFieldSegments;
561
562
563         //////////////////
564         /* General Data */
565         //////////////////
566
567         // Domain size (as returned by the Lumis Interface after creation)
568         int32 Nx;
569         int32 Ny;
570         int32 Nz;
571
572         // HUD text retrieval
573         UFUNCTION(BlueprintCallable, Category = "VR_HUD")
574         FString getStatText();
575
576         private:
577
578         // String containing status information
579         FString hud_string;
580
581         // Pointer to the HUD so we can change text etc. based on LBM solver
582         settings
583         ALbmHUD* hud;
584
585         // List of static meshes to be used for visual object swapping
586         TArray<UStaticMesh*> swappableMeshes;
587
588         // List of point clouds for each object
589         TArray<PCpts*> pointClouds;
590
591         // List of file names of the objects & point clouds to be loaded (from
592         object capture pipeline)
593         TArray<FString> objectNames;
594
595         // Array of object lengths and positions
596         TArray<float> objectPositions;
597         TArray<float> objectLengths;
598
599         // Domain extents specified in UE coordinates
600         UPROPERTY(VisibleAnywhere, Category = "LBM")
601         FVector DomainExtents = FVector(540.0f, 540.0f, 270.0f);
602
603         // Object bounding box in UE relative coordinates
604         FBox ObjectBox;
605
606         // Domain size information
607         FVector DomainBoxScaling;
608
609         // Pointer to the mesh components which represents the domain and the object
610         UStaticMeshComponent* BoxVisual;
611         UStaticMeshComponent* ObjectVisual;
```

```
612
613         // Method for reading in the meshes from the file in the Input directory
614         int32 readObjectDataFromInputFile();
615
616         // Private method for adding a pre-loaded object mesh to the tunnel
617         void setObjectMesh(int32 objectidx);
618
619         // Number of objects available
620         int32 numObjects;
621
622 };
```

References

[1] D. Gatti, Turbulent Skin-Friction Drag Reduction at High Reynolds Numbers, Springer International Publishing, Cham, 2016, pp. 389–398.

[2] S. Chen, G. D. Doolen, Lattice Boltzmann Method for Fluid Flows, Annual Review of Fluid Mechanics 30 (1) (1998) 329–364.

[3] M. Mawson, Interactive Fluid-Structure Interaction with Many-core Accelerators, Ph.D. thesis, School of Mechanical, Aerospace & Civil Engineering, The University of Manchester (2013).

[4] N. Delbosc, J. Summers, A. Khan, N. Kapur, C. Noakes, Optimized implementation of the Lattice Boltzmann Method on a graphics processing unit towards real-time fluid simulation, Computers & Mathematics with Applications 67 (2) (2014) 462 – 475, mesoscopic Methods for Engineering and Science (Proceedings of ICMMES-2012, Taipei, Taiwan, 23-27 July 2012).

[5] A. Bernard, Virtual engineering: Methods and tools, Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture 219 (5) (2005) 413–421. doi:10.1243/095440505X32238.

[6] S. Bryson, C. Levit, The Virtual Windtunnel: An Environment for the Exploration of Three-Dimensional Unsteady Flows, in: Visualization, 1991. Visualization '91, Proceedings., IEEE Conference on, 1991, pp. 17–24, 407. doi:10.1109/VISUAL.1991.175771.

[7] G. S. Strumolo, V. Babu, Method and system for providing a virtual wind tunnel, uS Patent 6,088,521 (2000).

[8] A. Borrmann, P. Wenisch, C. van Treeck, E. Rank, Collaborative computational steering: Principles and application in hvac layout, Integr. Comput.-Aided Eng. 13 (4) (2006) 361–376.

[9] P. Wenisch, C. van Treeck, A. Borrmann, E. Rank, O. Wenisch, Computational steering on distributed systems: Indoor comfort simulations as a case study of interactive CFD on supercomputers, International Journal of Parallel, Emergent and Distributed Systems 22 (4) (2007) 275–291.

[10] J. Linxweiler, M. Krafczyk, J. Tölke, Highly interactive computational steering for coupled 3D flow problems utilizing multiple GPUs, Computing and Visualization in Science 13 (7) (2010) 299–314.

[11] N. Delbosc, Real-time simulation of indoor air flow using the lattice Boltzmann method on Graphiscs Processing Units, Ph.D. thesis, School of Mechanical Engineering, The University of Leeds (2015).

[12] M. S. Glessmer, C. F. Janßen, Using an Interactive Lattice Boltzmann Solver in Fluid Mechanics Instruction, Computation 5 (3) (2017). doi:10.3390/computation5030035.

[13] A. R. G. Harwood, A. J. Revell, Parallelisation of an interactive lattice-Boltzmann method on an Android-powered mobile device, Advances in Engineering Software 104 (1) (2017) 38–50.

[14] A. R. G. Harwood, A. J. Revell, Interactive flow simulation using Tegra-powered mobile devices, Advances in Engineering Software 115 (Supplement C) (2018) 363 – 373.

[15] A. R. G. Harwood, GPU-powered, interactive flow simulation on a peer-to-peer group of mobile devices, Advances in Engineering Software 133 (2019) 39 – 51. doi:https://doi.org/10.1016/j.advengsoft.2019.04.003.

[16] C. Cruz-Neira, J. Leigh, M. Papka, C. Barnes, S. M. Cohen, S. Das, R. Engelmann, R. Hudson, T. Roy, L. Siegel, C. Vasilakis, T. A. DeFanti, D. J. Sandin, Scientists in wonderland: A report on visualization applications in the CAVE virtual reality environment, in: Proceedings of 1993 IEEE Research Properties in Virtual Reality Symposium, 1993, pp. 59–66. doi:10.1109/VRAIS.1993.378262.

[17] M. I. Billen, O. Kreylos, B. Hamann, M. A. Jadamec, L. H. Kellogg, O. Staadt, D. Y. Sumner, A geoscience perspective on immersive 3D gridded data visualization, Computers & Geosciences 34 (9) (2008) 1056 – 1072. doi:https://doi.org/10.1016/j.cageo.2007.11.009.

[18] T. W. Kuhlen, B. Hentschel, Quo Vadis CAVE: Does Immersive Visualization Still Matter?, IEEE Computer Graphics and Applications 34 (5) (2014) 14–21. doi:10.1109/MCG.2014.97.

[19] Open VR, `https://github.com/ValveSoftware/openvr`, accessed: 2017-12-12.

[20] J. Jacobson, M. Lewis, Game engine virtual reality with caveut, Computer 38 (4) (2005) 79–82. doi:10.1109/MC.2005.126.

[21] S. Wang, Z. Mao, C. Zeng, H. Gong, S. Li, B. Chen, A new method of virtual reality based on unity3d, in: 2010 18th International Conference on Geoinformatics, 2010, pp. 1–5. doi:10.1109/GEOINFORMATICS.2010.5567608.

[22] W. Yan, C. Culp, R. Graf, Integrating BIM and gaming for real-time interactive architectural visualization, Automation in Construction 20 (4) (2011) 446 – 458. doi:https://doi.org/10.1016/j.autcon.2010.11.013.

[23] C. Donalek, S. G. Djorgovski, A. Cioc, A. Wang, J. Zhang, E. Lawler, S. Yeh, A. Mahabal, M. Graham, A. Drake, S. Davidoff, J. S. Norris, G. Longo, Immersive and collaborative data visualization using virtual reality platforms, in: 2014 IEEE International Conference on Big Data (Big Data), 2014, pp. 609–614. doi:10.1109/BigData.2014.7004282.

[24] J. Wang, M. Lewis, J. Gennari, A game engine based simulation of the NIST urban search and rescue arenas, in: Proceedings of the 2003 Winter Simulation Conference, 2003., Vol. 1, 2003, pp. 1039–1045 Vol.1. doi:10.1109/WSC.2003.1261528.

[25] A. C. A. Mól, C. A. F. Jorge, P. M. Couto, Using a Game Engine for VR Simulations in Evacuation Planning, IEEE Computer Graphics and Applications 28 (3) (2008) 6–12. doi:10.1109/MCG.2008.61.

[26] J. R. Juang, W. H. Hung, S. C. Kang, Using game engines for physical-based simulations – a forklift, in: Special Issue: Use of Gaming Technology in Architecture, Engineering and Construction, Vol. 16, 2011, pp. 3–22, `http://www.itcon.org/2011/2`.

[27] K. Yang, J. Jie, S. Haihui, Study on the virtual natural landscape walkthrough by using Unity 3D, in: 2011 IEEE International Symposium on VR Innovation, 2011, pp. 235–238. doi:10.1109/ISVRI.2011.5759642.

[28] A. Falcone, A. Garro, F. Longo, F. Spadafora, Simulation Exploration Experience: A Communication System and a 3D Real Time Visualization for a Moon Base Simulated Scenario, in: 2014 IEEE/ACM 18th International Symposium on Distributed Simulation and Real Time Applications, 2014, pp. 113–120. doi:10.1109/DS-RT.2014.22.

[29] A. Li, X. Zheng, W. Wang, Motion Simulation of Hydraulic Support Based on Unity 3D, in: Proceedings of the First International Conference on Information Sciences, Machinery, Materials and Energy, Advances in Intelligent Systems Research, 2015. doi:10.2991/icismme-15.2015.128.

[30] U. Rüppel, K. Schatz, Designing a BIM-based serious game for fire safety evacuation simulations, Advanced Engineering Informatics 25 (4) (2011) 600 – 611, special Section: Advances and Challenges in Computing in Civil and Building Engineering. doi:https://doi.org/10.1016/j.aei.2011.08.001.

[31] J. Lewis, D. Brown, W. Cranton, R. Mason, Simulating visual impairments using the Unreal Engine 3 game engine, in: 2011 IEEE 1st International Conference on Serious Games and Applications for Health (SeGAH), 2011, pp. 1–8. doi:10.1109/SeGAH.2011.6165430.

[32] W. Meng, Y. Hu, J. Lin, F. Lin, R. Teo, ROS + Unity: An efficient high-fidelity 3D multi-UAV navigation and control simulator in GPS-denied environments, in: IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society, 2015, pp. 002562–002567. doi:10.1109/IECON.2015.7392488.

[33] W. Qiu, A. Yuille, UnrealCV: Connecting Computer Vision to Unreal Engine, Springer International Publishing, Cham, 2016, pp. 909–916.

[34] K. Yang, J. Jie, The Designing of Training Simulation System Based on Unity 3D, in: 2011 Fourth International Conference on Intelligent Computation Technology and Automation, Vol. 1, 2011, pp. 976–978. doi:10.1109/ICICTA.2011.245.

[35] E. Sudarmilah, R. Ferdiana, L. E. Nugroho, A. Susanto, N. Ramdhani, Tech review: Game platform for upgrading counting ability on preschool children, in: 2013 International Conference on Information Technology and Electrical Engineering (ICITEE), 2013, pp. 226–231. doi:10.1109/ICITEED.2013.6676243.

[36] F. King, J. Jayender, S. K. Bhagavatula, P. B. Shyn, S. Pieper, T. Kapur, A. Lasso, G. Fichtinger, An Immersive Virtual Reality Environment for Diagnostic Imaging, Journal of Medical Robotics Research 01 (01) (2016) 1640003. doi:10.1142/S2424905X16400031.

[37] P. Zhou, X. Wang, U. Morales, Integration of Virtual Reality and CFD Techniques for Thermal Fluid Education, in: Proceedings of the 2017 Heat Transfer Summer Conference, 2017. doi:10.1115/HT2017-4793.

[38] Frostbite Engine – the most adopted platform for game development – EA, `https://www.ea.com/frostbite`, accessed: 2017-12-12.

[39] CryENGINE3 – Crytek, `http://www.crytek.com/cryengine/cryengine3/overview`, accessed: 2017-12-12.

[40] Source SDK – Valve Development Community, `https://developer.valvesoftware.com/wiki/SDK`, accessed: 2017-12-12.

[41] Unity Game Engine, `https://unity3d.com/`, accessed: 2017-12-12.

[42] Game Engine Technology by Unreal, `https://www.unrealengine.com/en-US/what-is-unreal-engine-4`, accessed: 2017-12-12.

[43] Havok Physics, `https://www.havok.com/physics/`, accessed: 2017-12-12.

[44] GameWorks PhysX Overview, `https://developer.nvidia.com/gameworks-physx-overview`, accessed: 2017-12-12.

[45] J. Stam, Stable fluids, in: Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '99, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1999, pp. 121–128. doi:10.1145/311535.311548.
URL `http://dx.doi.org/10.1145/311535.311548`

[46] J. Brackbill, D. Kothe, H. Ruppel, Flip: A low-dissipation, particle-in-cell method for fluid flow, Computer Physics Communications 48 (1) (1988) 25 – 38. doi:https://doi.org/10.1016/0010-4655(88)90020-3.

[47] J. M. Cohen, S. Tariq, S. Green, Interactive Fluid-particle Simulation Using Translating Eulerian Grids, in: Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, I3D '10, ACM, New York, NY, USA, 2010, pp. 15–22. doi:10.1145/1730804.1730807.

[48] M. Müller, D. Charypar, M. Gross, Particle-based fluid simulation for interactive applications, in: Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '03, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2003, pp. 154–159.

[49] M. Macklin, M. Müller, N. Chentanez, T.-Y. Kim, Unified Particle Physics for Real-time Applications, ACM Trans. Graph. 33 (4) (2014) 153:1–153:12.

[50] J. Bender, M. Müller, M. A. Otaduy, M. Teschner, M. Macklin, A survey on position-based simulation methods in computer graphics, Comput. Graph. Forum 33 (6) (2014) 228–251. doi:10.1111/cgf.12346.

[51] K. Sharp, F. Matschinsky, Translation of Ludwig Boltzmann's Paper "On the Relationship between the Second Fundamental Theorem of the Mechanical Theory of Heat and Probability Calculations Regarding the Conditions for Thermal Equilibrium" sitzungberichte der kaiserlichen akademie der wissenschaften. mathematisch-naturwissen classe. abt. ii, lxxvi 1877, pp 373-435 (wien. ber. 1877, 76:373-435). reprinted in wiss. abhandlungen, vol. ii, reprint 42, p. 164-223, barth, leipzig, 1909, Entropy 17 (4) (2015) 1971–2009.

[52] The lattice Boltzmann method for compressible flows at high Mach number.

[53] J. Tölke, Implementation of a lattice boltzmann kernel using the compute unified device architecture developed by nvidia, Computing and Visualization in Science 13 (1) (2008) 29. doi:10.1007/s00791-008-0120-2.

[54] C. Obrecht, F. Kuznik, B. Tourancheau, J.-J. Roux, Scalable lattice Boltzmann solvers for CUDA GPU clusters, Parallel Computing 39 (6) (2013) 259 – 270. doi:https://doi.org/10.1016/j.parco.2013.04.001.

[55] X. Wang, Y. Shangguan, N. Onodera, H. Kobayashi, T. Aoki, Direct Numerical Simulation and Large Eddy Simulation on a Turbulent Wall-Bounded Flow Using Lattice Boltzmann Method and Multiple GPUs, Mathematical Problems in Engineering 2014 (2014). doi:http://www.doi.org/10.1155/2014/742432.

[56] M. J. Mawson, A. J. Revell, Memory transfer optimization for a lattice Boltzmann solver on Kepler architecture nVidia GPUs, Computer Physics Communications 185 (10) (2014) 2566–2574.

[57] Y. Koda, F.-S. Lien, The Lattice Boltzmann Method Implemented on the GPU to Simulate the Turbulent Flow Over a Square Cylinder Confined in a Channel, Flow, Turbulence and Combustion 94 (3) (2015) 495–512. doi:10.1007/s10494-014-9584-y.

[58] N.-P. Tran, M. Lee, S. Hong, Performance Optimization of 3D Lattice Boltzmann Flow Solver on a GPU, Scientific Programming 2017 (2017). doi:http://www.doi.org/10.1155/2017/1205892.

[59] W. Li, X. Wei, A. Kaufman, Implementing lattice Boltzmann computation on graphics hardware, The Visual Computer 19 (7) (2003) 444–456. doi:10.1007/s00371-003-0210-6.

[60] NVIDIA, CUDA Toolkit Documentation v9.1.85, `http://docs.nvidia.com/cuda/`, , Accessed: 2017-12-12.

[61] P. L. Bhatnagar, E. P. Gross, M. Krook, A Model for Collision Processes in Gases. I. Small Amplitude Processes in Charged and Neutral One-Component Systems, Phys. Rev. 94 (1954) 511–525.

[62] D. P. Ziegler, Boundary conditions for lattice Boltzmann simulations, Journal of Statistical Physics 71 (5) (1993) 1171–1177.

[63] H. Yu, S. S. Girimaji, L.-S. Luo, DNS and LES of decaying isotropic turbulence with and without frame rotation using lattice Boltzmann method, Journal of Computational Physics 209 (2) (2005) 599 – 616. doi:https://doi.org/10.1016/j.jcp.2005.03.022.

[64] Z. Guo, C. Zheng, B. Shi, Discrete lattice effects on the forcing term in the lattice Boltzmann method, Physical Review E 65 (2002) 046308.

[65] E. Meinders, K. Hanjalić, Vortex structure and heat transfer in turbulent flow over a wall-mounted matrix of cubes, International Journal of Heat and Fluid Flow 20 (3) (1999) 255 – 267.

[66] A. Hellsten, P. Rautaheimo, S. Laine, T. Siikonen, 8th ercoftac/iahr/cost workshop on refined turbulence modelling (Dec 1999).

[67] A. R. G. Harwood, P. Wenisch, A. J. Revell, A Real-Time Modelling and Simulation Platform for Virtual Engineering Design and Analysis, in: Proceedings of 6th European Conference on Computational Mechanics (ECCM 6) and 7th European Conference on Computational Fluid Dynamics (ECFD 7), 11-15 June 2018, Glasgow, UK, ECCOMAS, 2018.

[68] W. Boehler, A. Marbs, 3D Scanning Instruments, in: Proceedings of the CIPA WG 6 International Workshop on Scanning for Cultural Heritage Recording, Ziti, Thessaloniki, 2002, pp. 9–18.

[69] B. Curless, From Range Scans to 3D Models, ACM SIGGRAPH Computer Graphics 33 (4) (1999) 38–41.

[70] T. Butkiewicz, Low-cost coastal mapping using Kinect v2 time-of-flight cameras, in: 2014 Oceans - St. John's, 2014, pp. 1–9.

[71] P. Fankhauser, M. Bloesch, D. Rodriguez, R. Kaestner, M. Hutter, R. Siegwart, Kinect v2 for mobile robot navigation: Evaluation and modeling, in: 2015 International Conference on Advanced Robotics (ICAR), 2015, pp. 388–394.

[72] L. Yang, L. Zhang, H. Dong, A. Alelaiwi, A. E. Saddik, Evaluating and Improving the Depth Accuracy of Kinect for Windows v2, IEEE Sensors Journal 15 (8) (2015) 4275–4285.

[73] J. Smisek, M. Jancosek, T. Pajdla, 3D with Kinect, in: Consumer Depth Cameras for Computer Vision, Springer, 2013, pp. 3–25.

[74] E. Lachat, H. Macher, T. Landes, P. Grussenmeyer, Assessment and calibration of a rgb-d camera (kinect v2 sensor) towards a potential use for close-range 3d modeling, Remote Sensing 7 (10) (2015) 13070–13097. doi:10.3390/rs71013070.

[75] R. B. Rusu, S. Cousins, 3D is here: Point Cloud Library (PCL), in: Robotics and Automation (ICRA), 2011 IEEE International Conference on, 2011, pp. 1–4.

[76] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, G. Taubin, The ball-pivoting algorithm for surface reconstruction, IEEE Transactions on Visualization and Computer Graphics 5 (4) (1999) 349–359. doi:10.1109/2945.817351.

[77] Blender 3D – a 3D modelling and rendering package, `http://www.blender.org`, accessed: 2017-12-12.

[78] W. von Funck, T. Weinkauf, H. Theisel, H. P. Seidel, Smoke Surfaces: An Interactive Flow Visualization Technique Inspired by Real-World Flow Experiments, IEEE Transactions on Visualization and Computer Graphics 14 (6) (2008) 1396–1403. doi:10.1109/TVCG.2008.163.

[79] M. Bouzidi, M. Firdaouss, P. Lallemand, Momentum transfer of a Boltzmann-lattice fluid with boundaries, Physics of Fluids 13 (11) (2001) 3452–3459.

[80] N. Koliha, C. F. Janßen, T. Rung, Towards Online Visualization and Interactive Monitoring of Real-Time CFD Simulations on Commodity Hardware, Computation 3 (3) (2015) 444–478. doi:10.3390/computation3030444.

[81] A. R. G. Harwood, Interactive Modelling and Simulation for Engineering Design and Analysis, NAFEMS Benchmark Magazine Oct 2018 (2018) 20–24.