Evaluation of GPU-based Conductive Heat Transfer Algorithms

Daniele Döhle[1], Kristian Börger[1] and Lukas Arnold[1,2]

[1] Computational Civil Engineering, University of Wuppertal, Germany
[2] Institut for Advanced Simulation, Forschungszentrum Jülich, Germany

E-mail: daniele.doehle@gmx.de, boerger@uni-wuppertal.de, l.arnold@fz-juelich.de / arnold@uni-wuppertal.de

Abstract. The one-dimensional heat transfer algorithm of the Fire Dynamics Simulator (FDS) is currently implemented to run on a CPU (Central Processing Unit). This study explores the potential advantages of adapting the algorithm for Graphics Processing Units (GPUs), which could offer significant computational benefits.

The motivation behind this work stems from the intention to speed up numerical fire simulations. Up to now, simplifications with regard to grid resolution and level of detail have been made, compromising accuracy for quicker results. Simulations, especially for heat transfer in solid objects such as walls, require computationally intensive resources. By leveraging the GPUs' superior parallel processing capabilities, it is possible to conduct faster and more accurate simulations, avoiding these compromises.

Both a CPU and a GPU algorithm for computing the 1D heat transfer are developed, and the computation time is compared against each other. Both implementations are validated against a simple FDS simulation with identical boundary conditions. The investigations show that the GPU algorithm is promising above a certain number of wall elements, depending on the employed hardware. The results show that this is generally the case from 2048 elements.

1. Introduction

Conductive heat transfer in solids is part of many phenomena relevant for fire safety science. This includes for example thermal stress on building elements or the prediction of thermal decomposition of solids (pyrolysis). The interaction of the solid and gas phase occurs mainly via convection and thermal radiation at the interface.

With the increased interest in enhancing the level of detail in fire simulations, especially in the pyrolysis modelling, the computing time for numerically solving the conductive heat transfer is rising. Relevant CFD (computational fluid dynamics) software, like FDS (Fire Dynamics Simulator), run exclusively on CPUs (Central Processing Unit). However, modern computers are generally equipped with GPUs (Graphics Processing Unit), which can be utilised to speed up the computations. Within this contribution, we introduce an implementation of a conductive heat transfer solver for a GPU, based on the current numerical implementation in FDS (see the Technical Reference Guide [1], version 6.8.0). We investigate three different memory handling approaches, using exactly the same numerical scheme as FDS to allow for a rigid runtime comparison. The implementation is written in the script language Rust, while the GPU code uses WGSL and Vulcan as a backend, see reference implementation [2].

The goal here is to evaluate the speed-up and efficiency of the GPU implementation and provide insights for implementations in full CFD software. The implementation used in this demonstrative work is computing only the solid phase and uses the gas phase information provided by an accompanying FDS simulation. From this, values of the heat transfer coefficient h_f, the radiative heat flux to the wall $\dot{q}''_{r,in}$ and the temperature of the adjacent gas cell T_g^{n+1} are extracted for the solid phase boundary conditions. Furthermore, the FDS simulation is used to verify the solution of the implementation.

2. Principle of Heat Transfer

The underlying conductive heat transfer through a solid is given by equation (1), which describes the solid temperature T_s at a point x in the Cartesian coordinate system at a given time t. The heat transfer depends on the density ρ_s, the specific heat capacity c_s and the thermal conductivity k_s. The boundary conditions for this equation are given by the heat flux from radiation and convection at the solid surface. Additional volume heat sources $\dot{q}'''_\mathrm{s}$ can be considered as well. Following the FDS implementation, the Crank-Nicolson scheme is used to discretise the equation, see equation (2).

$$\rho_\mathrm{s} \cdot c_\mathrm{s} \cdot \frac{\partial T_\mathrm{s}}{\partial t} = \frac{\partial}{\partial x}\left(k_\mathrm{s} \cdot \frac{\partial T_\mathrm{s}}{\partial x}\right) + \dot{q}'''_\mathrm{s} \tag{1}$$

$$\begin{aligned}(\rho_\mathrm{s} \cdot c_\mathrm{s})_i \cdot \frac{T^{n+1}_{\mathrm{s},i} - T^n_{\mathrm{s},i}}{\Delta t_\mathrm{s}} &= \frac{1}{2 \cdot \Delta x_i} \cdot \left(k_{\mathrm{s},i+\frac{1}{2}} \cdot \frac{T^n_{\mathrm{s},i+1} - T^n_{\mathrm{s},i}}{\Delta x_{i+\frac{1}{2}}} - k_{\mathrm{s},i-\frac{1}{2}} \cdot \frac{T^n_{\mathrm{s},i} - T^n_{\mathrm{s},i-1}}{\Delta x_{i-\frac{1}{2}}}\right) \\ &+ \frac{1}{2 \cdot \Delta x_i} \cdot \left(k_{\mathrm{s},i+\frac{1}{2}} \cdot \frac{T^{n+1}_{\mathrm{s},i+1} - T^{n+1}_{\mathrm{s},i}}{\Delta x_{i+\frac{1}{2}}} - k_{\mathrm{s},i-\frac{1}{2}} \cdot \frac{T^{n+1}_{\mathrm{s},i} - T^{n+1}_{\mathrm{s},i-1}}{\Delta x_{i-\frac{1}{2}}}\right) + \dot{q}'''_\mathrm{s}\end{aligned} \tag{2}$$

The 2D surface of a wall is divided into a grid. Each grid element is assigned a 1D wall element with thickness w and material, which is divided into cells according to Figure 1. The number of cells depends on the material properties ρ_s, c_s and k_s. The cells are denser at the edges than in the centre to better resolve steep temperature gradients [3]. The heat transfer can be calculated on the discretised wall element. Values from the solid cells are indexed with $_s$ and for gas cells with $_g$, $_f$ stands for the front side.

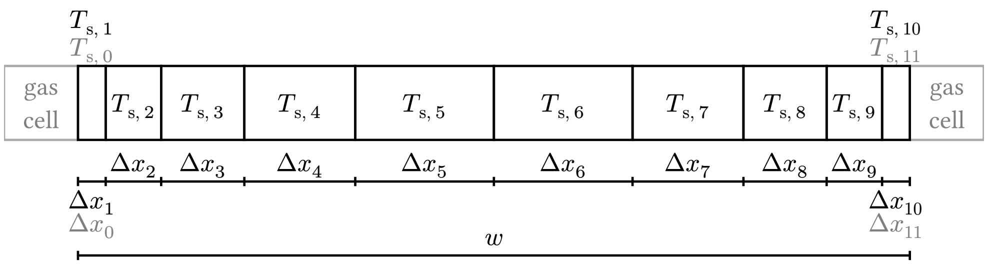

Figure 1. Structure of the 1D wall element with variable cell size

If the temperature difference between adjacent cells is too large, the numerical scheme becomes unstable due to the Courant-Friedrichs-Lewy condition. To counteract this, the time interval Δt_s is reduced. To do this, the maximum temperature difference $\Delta T^n_{s,max}$ between the cells is calculated using the equation (3). Finally, $\Delta t_\mathrm{s,new}$ is calculated using the equation (4) and the algorithm is repeated S times.

$$\Delta T^n_\mathrm{s,max} = \max_{i=1,N}\left[\frac{\Delta t_\mathrm{s}}{(\rho_\mathrm{s} \cdot c_\mathrm{s})_i \cdot \Delta x_i}\left|k_{i+\frac{1}{2}} \cdot \frac{T^n_{\mathrm{s},i+1} - T^n_{\mathrm{s},i}}{\Delta x_{i+\frac{1}{2}}} - k_{i-\frac{1}{2}} \cdot \frac{T^n_{\mathrm{s},i} - T^n_{\mathrm{s},i-1}}{\Delta x_{i-\frac{1}{2}}}\right|\right] \tag{3}$$

$$\Delta t_{\mathrm{s,new}} = \frac{\Delta t_{\mathrm{s}}}{S} \qquad \text{with} \qquad \eta = \frac{\Delta T^n_{\mathrm{s,max}}}{10\ \mathrm{K}} \qquad S = \left\lceil \frac{\log(\eta)}{\log(2)} \right\rceil \leq 4 \tag{4}$$

The heat transfer can be solved using a tridiagonal matrix according to equation (5), where the required values are determined according to equation (6). The Thomas algorithm is used for efficient calculation. The resulting surface temperature is calculated according to equation (7).

$$\begin{pmatrix} d_1 & a_1 & 0 & \cdots & 0 \\ & \ddots & \ddots & & \\ & b_i & d_i & a_i & \\ & & \ddots & \ddots & \\ 0 & \cdots & 0 & b_{N-1} & d_{N-1} \end{pmatrix} \cdot \begin{pmatrix} T^{n+1}_{\mathrm{s},1} \\ \vdots \\ T^{n+1}_{\mathrm{s},i} \\ \vdots \\ T^{n+1}_{\mathrm{s},N-1} \end{pmatrix} = \begin{pmatrix} c_1 \\ \vdots \\ c_i \\ \vdots \\ c_{N-1} \end{pmatrix} \tag{5}$$

$$a_i = \frac{k_{i+\frac{1}{2}} \cdot \Delta t_{\mathrm{s,new}}}{2\left(\rho_{\mathrm{s}} c_{\mathrm{s}}\right)_i \cdot \Delta x_i \cdot \Delta x_{i+\frac{1}{2}}} \qquad b_i = \frac{k_{i-\frac{1}{2}} \cdot \Delta t_{\mathrm{s,new}}}{2\left(\rho_{\mathrm{s}} c_{\mathrm{s}}\right)_i \cdot \Delta x_i \cdot \Delta x_{i-\frac{1}{2}}}$$
$$c_i = T^n_{\mathrm{s},i} - a_i\left(T^n_{\mathrm{s},,i+1} - T^n_{\mathrm{s},,i}\right) + b_i\left(T^n_{\mathrm{s},,i} - T^n_{\mathrm{s},,i-1}\right) \qquad d_i = 1 - a_i - b_i \tag{6}$$

$$T^{n+1}_{\mathrm{s,f}} = \underbrace{\frac{\frac{k_{\mathrm{f},1}}{\Delta x_{\frac{1}{2}}} - \frac{1}{2}h^{n+1}_{\mathrm{g,f}} - 2\epsilon\sigma \cdot \left(T^n_{\mathrm{s},\frac{1}{2}}\right)^3}{\frac{k_{\mathrm{f},1}}{\Delta x_{\frac{1}{2}}} + \frac{1}{2}h^{n+1}_{\mathrm{g,f}} - 2\epsilon\sigma \cdot \left(T^n_{\mathrm{s},\frac{1}{2}}\right)^3}}_{\mathrm{rfac2_f}} \cdot T^{n+1}_{\mathrm{s},1} + \underbrace{\frac{\overbrace{h^{n+1}_{\mathrm{g,f}} \cdot T^{n+1}_{\mathrm{g,f}} + \dot{q}''_{\mathrm{r,in,f}}}^{\dot{q}''_{\mathrm{in,f}}} + 3\epsilon\sigma \cdot \left(T^n_{\mathrm{s},\frac{1}{2}}\right)^4}{\frac{k_{\mathrm{f},1}}{\Delta x_{\frac{1}{2}}} + \frac{1}{2}h^{n+1}_{\mathrm{g,f}} - 2\epsilon\sigma \cdot \left(T^n_{\mathrm{s},\frac{1}{2}}\right)^3}}_{\mathrm{qdxk_f}} \tag{7}$$

3. Setup of the Simulations

Figure 2 shows a schematic visualisation of the simulation setup. A volume of 0.30 m × 0.25 m × 0.25 m is created with an isotropic mesh spacing of 0.05 m. The boundary conditions of the box are set to adiabatic. The box is separated in the centre by a 0.10 m thick wall. Two simulation of identical geometry are conducted for the materials concrete and steel on the dividing wall. The values of these material properties are taken from the respective building standards [4] [5]. The material properties affect not only the heat flux but also the number of cells in a wall element, which can be taken from the table 1. For validation purposes, 6000 s are simulated while 200 s are sufficient for the benchmarks.

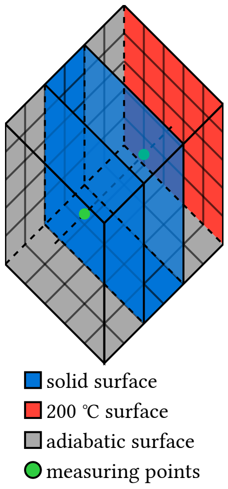

Figure 2. Model of the diabatic FDS simulation.

Table 1. Number of solid cells needed for different thicknesses of materials.

Thickness [m]	0.05	0.10	0.20	0.50	1.00	2.00	5.00
Concrete [cells]	12	14	16	19	21	23	26
Steel [cells]	6	8	10	13	15	16	19

4. Implementation

The implementation of the algorithm is illustrated in Figure 3, while the actual calculation is implemented identically on CPU and GPU. In the context of the GPU, the algorithm is executed as a shader, which is a program that has been explicitly compiled and optimised for the GPU [6]. There are various shader types that can access different GPU functions. However, only compute shaders are relevant for calculating the heat transport.

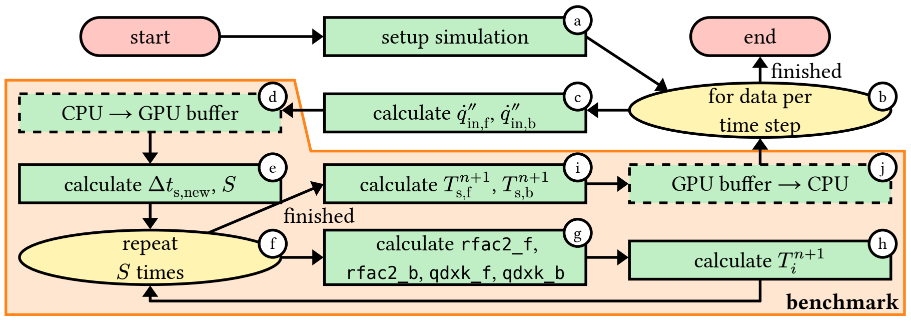
start
setup simulation
a
end
finished
for data per time step
b
CPU → GPU buffer
d
calculate $\dot{q}''_{\mathrm{in,f}}$, $\dot{q}''_{\mathrm{in,b}}$
c
calculate $\Delta t_{\mathrm{s,new}}$, S
e
calculate $T^{n+1}_{\mathrm{s,f}}$, $T^{n+1}_{\mathrm{s,b}}$
i
GPU buffer → CPU
j
finished
repeat S times
f
calculate rfac2_f, rfac2_b, qdxk_f, qdxk_b
g
calculate T^{n+1}_i
h
benchmark

Figure 3. Flowchart of the implemented heat transfer algorithm.

(a) The FDS setup file is read and used to set up the simulation.
(b) The "*_devc.csv" file is read and the heat transfer is calculated for each time step, the values read out are $h_{\mathrm{g,f}}$, $h_{\mathrm{g,b}}$, $T^{n+1}_{\mathrm{g,f}}$, $T^{n+1}_{\mathrm{g,b}}$, $\dot{q}_{\mathrm{r,in,f}}$ and $\dot{q}_{\mathrm{r,in,b}}$.
(c) Calculation of the values $\dot{q}''_{\mathrm{in,f}}$ and $\dot{q}''_{\mathrm{in,b}}$ on the CPU.
(d) If executed for the GPU, the required data is written to a GPU buffer.
(e) Calculation of the reduced time interval $\Delta t_{\mathrm{s,new}}$ and the number of repetitions S.
(f) Repeat the heat transfer algorithm S times.
(g) Calculation of the gas interaction variables rfac2$_{\mathrm{f}}$, rfac2$_{\mathrm{b}}$, qdxk$_{\mathrm{f}}$ and qdxk$_{\mathrm{b}}$.
(h) Populate the solution matrix according to equation (6) and solve it.
(i) Calculation of the wall surface temperature according to equation (7).
(j) If executed for the GPU, the edge temperature is copied from the GPU buffer.

Depending on the size and structure of the simulation, the maximum number of wall elements that can be calculated with one dispatch command is quickly reached. Figure 4 shows how step (d) is implemented in Figure 3 to avoid this problem by first splitting and then overlapping computation and data transfer. The read data is first divided into n chunks. These are each written to a buffer associated with the chunk. Once this has been written, the start command is given. Immediately afterwards, the same procedure is performed with the next chunk. In the meantime, calculations can be performed on the GPU. This procedure is repeated for all n chunks. When the GPU part of a chunk is finished and all chunks have given the start command, the chunks are read from the buffer. This implementation is asynchronous.

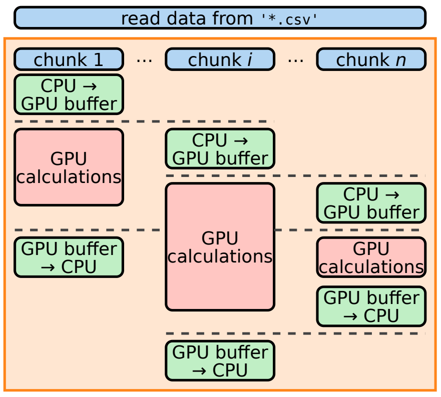

Figure 4. Schematic representation of the division of data into chunks.

In order for the algorithm to run on the GPU, it must be written as a shader. Part of the shader is created dynamically after the simulation file is read. For example, the materials used, and their properties are inserted.

In a shader, it is not easy to create a 2D array whose subarrays have different lengths. For this reason, 3 methods were investigated to work around this problem. Figure 5 shows 2 materials (M_1, M_2) with walls of different thicknesses (D_1, D_2). This results in 3 different surfaces with different numbers of cells. The advantages and disadvantages of these methods are as follows:

Method 1: The individual cells are stored in a buffer. An array containing information about the start and end indices is transferred. These can then be used to extract the wall element. The advantages of this method are that the same shader can be used for each surface

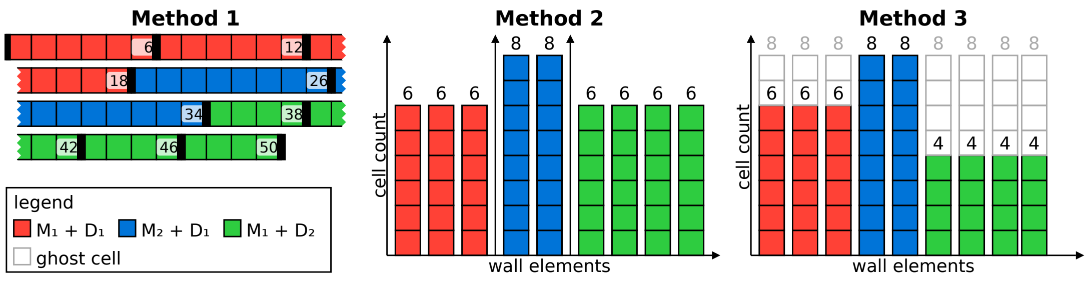

Figure 5. Schematic representation of the methods analysed.

and no ghost cells are created. The disadvantages are that additional buffers have to be created for the indices and the solution matrix, as it is not possible to use a workgroup variable.

Method 2: A separate shader is created for each surface type when the model is initialised, and the cell sizes Δx_i and material indices are inserted directly into a private array at initialisation. The advantages of this method are that there is no need to pass material and cell size through buffers. Additionally, the solution matrix lives in the workgroup address space. The main disadvantage of this method is that running two shaders with $x/2$ values takes longer than running one shader for x values.

Method 3: First the wall element with the most cells $z_{\max}$ is determined. Then a 2D buffer can be created to hold the data for all wall elements, assuming that all wall elements have $z_{\max}$ cells. A second buffer is also allocated for the actual number of cells. This means that all cells that extend beyond the actual length are ghost cells whose values are never checked. When dynamically generating the shader, only the maximum number of cells $z_{\max}$ needs to be specified. The advantage of this method is that the same shader can be used for each surface. The solution matrix also lives in the workgroup address space. The disadvantage of this method is that there are ghost cells which occupy unused memory space.

5. Validation

To validate the created programme, the same simulation is run as in FDS and the resulting wall temperatures are compared. The results are largely consistent as shown in Figure 6. The heat transfer itself was implemented identically for CPU and GPU, so that the results of the simulations match.

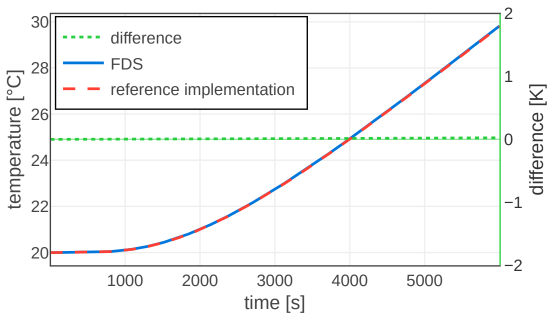

Figure 6. Difference in temperature on the back side between FDS and the reference implementation for a concrete sample.

6. Benchmarks

The required computation time depends heavily on the hardware on which the programs are running. The operating system also has an effect. The benchmarks were run on computers with the specified components shown in Table 2.

Table 2. Technical details of the used computer hardware.

	CPU	**GPU**	**operating system**
Desktop	AMD Ryzen 5 3600XT	AMD Radeon 5600 XT	Linux & Windows
Laptop	AMD Ryzen 7 5700U	AMD Radeon RX 640	Linux
Workstation	Intel i9-10900X	NVIDIA RTX A4500	Linux & Windows

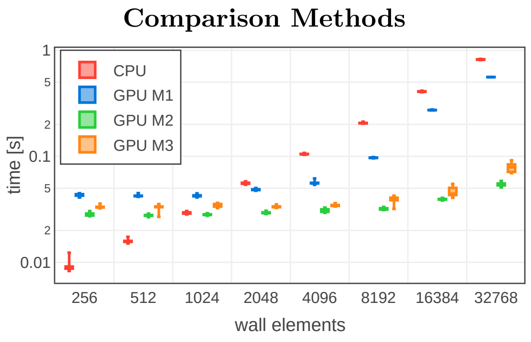

Figure 7. Benchmark of the adiabat simulation with concrete surface comparing methods, using Linux running on the Desktop hardware.

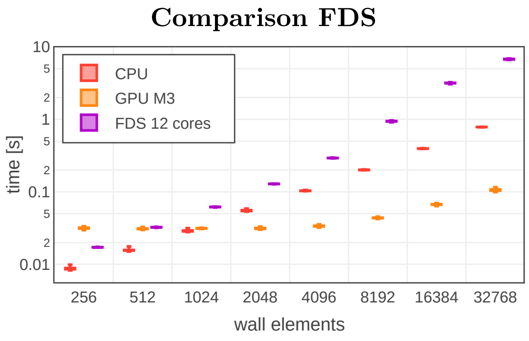

Figure 8. Benchmark of the diabate simulation with concrete surface compared with FDS, using Linux running on the Desktop hardware.

Figure 7 shows the influence of the different methods. With a small number of wall elements, the CPU is significantly faster than all analysed options on the GPU. From 2048 wall elements, all GPU algorithms are faster. Method 2 is the fastest, followed by method 3 and finally method 1. However, when multiple materials are used, method 2 falls behind, as described in the disadvantages. For this reason, the following graphs only compare method 3.

Figure 8 shows that the FDS heat transfer simulation requires more computation time than the created program. The opposite would imply that the implementations are not optimised and there may be errors. This is not the case. Although an attempt has been made to eliminate all unnecessary calculations, the longer computation time of FDS may be due to the fact that more than just heat transfer is involved.

7. Conclusion

This study demonstrates how switching from using traditional computer processors (CPUs) to graphics processors (GPUs) can speed up numerical fire simulations involving heat transfer in solids. Benchmarks on various computer configurations have shown that the developed programme generally has a positive effect on the computation time. The GPU algorithm in method 3 is particularly promising for simulation cases involving a wall element count ≥ 2048. A room simulation, $8\,\text{m} \times 8\,\text{m} \times 4\,\text{m}$ in size, achieves this quantity with a mesh spacing of $0.125\,\text{m}$. Adding obstructions increases the number of wall elements and thus the effectiveness. Therefore implementing this algorithm shortens the computation time and makes it possible to run simulations faster or perform more accurate simulations in the same time.

Acknowledgements

Some of the computations were using hardware funded by the BMBF project CoBra, with the funding number 13N15497.

References

[1] Kevin McGrattan et al. *Fire Dynamics Simulator Technical Reference Guide*. 6th ed. Revision: FDS-6.8.0-0-g886e009 April 18, 2023. NIST. DOI: `10.6028/NIST.SP.1018`.

[2] Daniele Döhle. *heat_transfer*. Version v1.0.0. URL: `https://github.com/XanthronWriter/heat_transfer`.

[3] Kevin McGrattan et al. *Fire Dynamics Simulator User's Guide*. 6th ed. Revision: FDS-6.8.0-0-g886e009 April 18, 2023. NIST. DOI: `10.6028/NIST.SP.1018`.

[4] EN 1992-1-2. *Eurocode 2: Design of concrete structures - Part 1-2: General rules - Structural fire design.* Tech. rep. Dec. 2004.

[5] EN 1993-1-2. *Eurocode 3: Design of steel structures - Part 1-2: General rules - Structural fire design.* Tech. rep. Dec. 2005.

[6] *WebGPU Shading Language.* URL: `https://www.w3.org/TR/2023/WD-WGSL-20231124/`. (W3C Working Draft, 24.11.2023).