Heat diffusion simulation

Sadat Sofiane

February 15, 2024

Abstract

The objective of this project is to create a real-time simulation of heat diffusion across a 2D grid, employing a constant boundary condition. The simulation is designed to be interactive, allowing users to set initial conditions through direct interaction with the graphical interface, which is powered by OpenGL At the core of the simulation lies the explicit scheme for solving the heat diffusion equation, a choice that facilitates straightforward implementation and immediate visual feedback. However, this method's reliance on the grid's resolution for computational accuracy introduces a significant challenge: as the grid's resolution increases to capture finer details, the computational latency correspondingly escalates. This increase in latency can severely impact the real-time performance of the simulation, making the use of GPU acceleration not just beneficial but essential for maintaining fluid interaction at higher resolutions. The parallel processing capabilities of GPUs can dramatically reduce computation times, which is evidenced by the observed performance improvement of over 9000% when comparing GPU-accelerated applications to their CPU-only counterparts.

Despite the satisfying results achieved with the explicit scheme in terms of real-time performance and visual output, it's important to consider the limitations inherent to this approach, particularly concerning stability. The explicit method's conditional stability can pose restrictions on the time step and grid resolution, potentially leading to numerical instabilities in certain scenarios. To circumvent these limitations and achieve unconditional stability, it might be worthwhile to explore implicit numerical methods for heat diffusion simulations.

1 Design Methodology

In the simulation of real-time heat diffusion, The pipeline of the application as shown in 1 1 begins on the host side, where the initial setup takes place. This setup involves initializing the application's parameters, which dictate the behavior and characteristics of the simulation, such as the size of the simulation grid and the simulation's time step. Alongside these parameters, the OpenGL settings are configured to render the simulation's visual output effectively. This rendering setup is crucial for displaying the heat diffusion simulation results in real-time and with high visual fidelity.

User interaction plays a pivotal role in this initial phase. Through the application's interface, users can set initial conditions for the simulation, such as the initial temperature distribution across the 2D grid. This interactive feature allows users to directly influence the simulation's starting state, making the experience more engaging and educational.

Once the initial setup is complete and the user has defined the initial conditions, the application enters a continuous loop, marking the core of the simulation's runtime. In this loop, the host prepares the data grid, which represents the current state of the simulation, for processing by the GPU. This preparation might include updating the grid based on user interactions or the simulation's progression rules.

After the host has prepared the data grid, it is transferred to the GPU, where the heavy lifting of the simulation occurs. The GPU processes the data using parallel computing techniques, applying the heat diffusion equation to simulate the temperature changes across the grid. This step leverages the GPU's ability to handle vast amounts of data simultaneously, significantly speeding up the simulation compared to CPU-only processing.

Following the GPU's processing, the updated data grid is sent back to the host. The host then uses OpenGL to render the updated simulation state, visually representing the heat diffusion across the 2D grid. This rendering step is crucial for providing real-time feedback to the user, allowing them to observe the simulation's evolution visually.

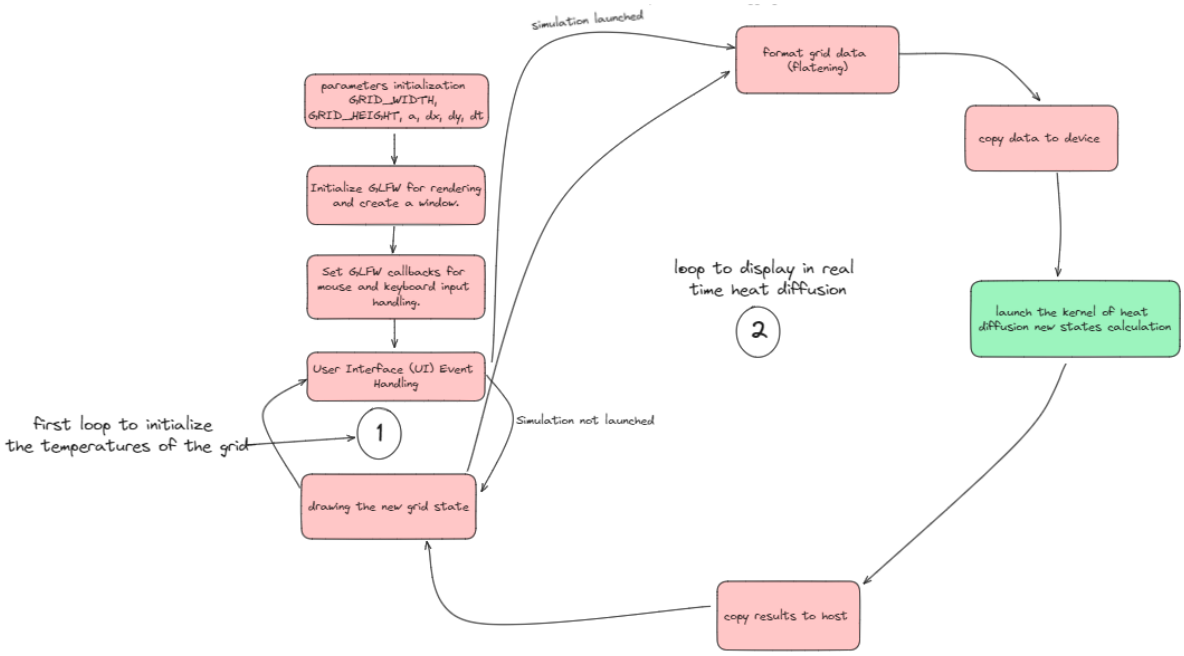


Figure 1: Application pipeline

This process, from data preparation on the host to processing on the GPU and back to rendering on the host, forms a continuous loop that runs indefinitely. The loop ensures that the heat diffusion simulation keeps running in real-time, allowing for ongoing user interaction and real-time visualization of the simulation's results. This interactive and dynamic approach enables users to explore the principles of heat diffusion in a visually intuitive and engaging manner.

In the following section, we will delve deeper into the calculations occurring at the device level.

Heat diffusion calculation

heat diffusion over a 2d surface with constant limit condition is ruled by the differential equation as follow

$$\frac{\partial u}{\partial t} = \alpha \nabla^2 u \tag{1}$$

where $u = u(x, y, t)$ represents the temperature at position (x, y) and time t, α is the thermal diffusivity of the material, and ∇^2 is the Laplacian operator. In a two-dimensional Cartesian coordinate system, the Laplacian operator ∇^2 for a function $u(x, y)$ is given by:

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \tag{2}$$

The analytical solution to the two-dimensional heat diffusion equation with constant boundary conditions can be expressed, under certain conditions, using separation of variables and Fourier series. For a rectangular domain with sides of lengths L_x and L_y, and assuming Dirichlet boundary conditions (temperature held at zero at the boundaries), the solution is given by:

$$u(x, y, t) = \sum_{n=1}^{\infty} \sum_{m=1}^{\infty} B_{nm} \sin\left(\frac{n\pi x}{L_x}\right) \sin\left(\frac{m\pi y}{L_y}\right) e^{-\alpha\pi^2(n^2/L_x^2 + m^2/L_y^2)t} \tag{3}$$

where α is the thermal diffusivity of the material, and B_{nm} are coefficients determined by the initial temperature distribution $f(x, y)$, calculated as:

$$B_{nm} = \frac{4}{L_x L_y} \int_0^{L_x} \int_0^{L_y} f(x, y) \sin\left(\frac{n\pi x}{L_x}\right) \sin\left(\frac{m\pi y}{L_y}\right) dy\, dx \tag{4}$$

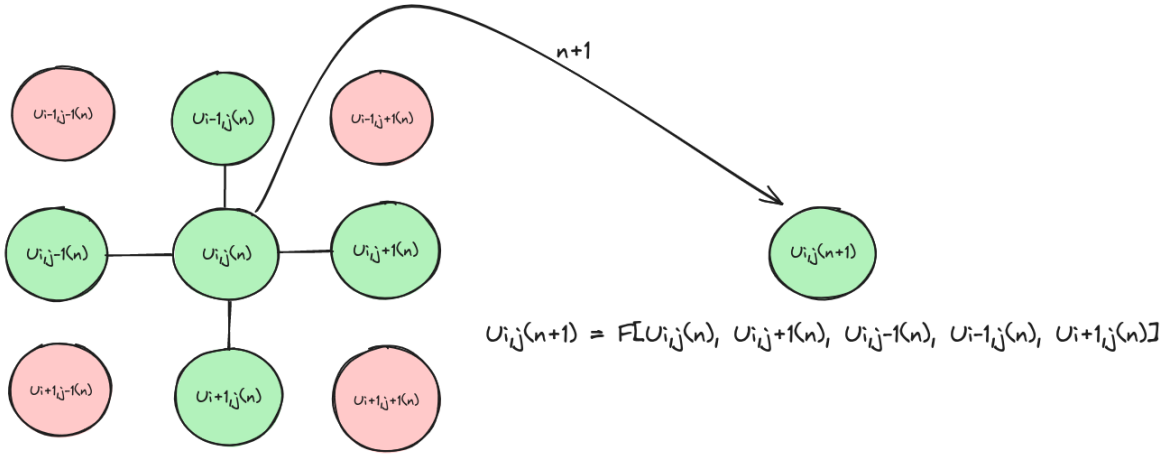


Figure 2: Representation of the calculation of the new state based on the previous state and the neighboring states.

It is important to note that obtaining such an analytical solution is not straightforward and is highly dependent on the simplicity of the domain geometry, boundary conditions, and initial conditions. For more complex scenarios, numerical methods are often employed to approximate the solution.

That why the use of numerical methods are important and can propose a satisfying results approaching the real physic phenomenon with a high accuracy. Many methods to resolve this partial differential equation (PDE) are possible each one has its own set of equations and implementation details, involving discretization of the continuous domain into a grid and approximation of derivatives at grid points. and each one has its trade-off between computational efficiency and accuracy, for our application we will choose to use Explicit forward time-centred Space (FTCS) method.

to resolve the equation numerically we use finite difference method and we approximate the first derivative with forward difference $f'(x) \approx \frac{f(x+h)-f(x)}{h}$ for more efficiency it involves using the current state to predict the next state. For the second derivative using the central finite difference is better for more accuracy this $f''(x) \approx \frac{f(x+h)-2f(x)+f(x-h)}{h^2}$ so the finite difference of 1 is :

$$\frac{U_{i,j}^{n+1} - U_{i,j}^{n}}{dt} = \alpha \left(\frac{U_{i-1,j}^{n} - 2U_{i,j}^{n} + U_{i+1,j}^{n}}{dx} + \frac{U_{i,j-1}^{n} - 2U_{i,j}^{n} + U_{i,j+1}^{n}}{dy} \right) \tag{5}$$

rearranging the formula to make the new state apparent get us the forumla that necetitate to be parralalized :

$$U_{i,j}^{n+1} = U_{i,j}^{n} + dt \cdot a \left(\frac{U_{i-1,j}^{n} - 2U_{i,j}^{n} + U_{i+1,j}^{n}}{dx} + \frac{U_{i,j-1}^{n} - 2U_{i,j}^{n} + U_{i,j+1}^{n}}{dy} \right) \tag{6}$$

So, for updating a point to its new state, we will need not only the current state of that point but also its four neighbors, as shown in Figure 2. The algorithm we aim to parallelize is depicted below.

Algorithm 1 2D Heat Equation Solver using Explicit Scheme

$nx \leftarrow$ width of the grid
$ny \leftarrow$ height of the grid
$numSteps \leftarrow$ number of time steps
$a \leftarrow$ diffusion constant
$dt \leftarrow$ time step size
$dx \leftarrow$ space step in x-direction
$dy \leftarrow$ space step in y-direction
Un $\leftarrow$ array of current temperature values
Unp1 $\leftarrow$ array of next time step temperature values
function GETINDEX(i, j, ny)
  return $i \times ny + j$
end function
for $n = 0$ to $numSteps$ **do**
  for $i = 1$ to $nx - 1$ **do**
    for $j = 1$ to $ny - 1$ **do**
      $index \leftarrow$ GETINDEX(i, j, ny)
      $uij \leftarrow$ Un$[index]$
      $uim1j \leftarrow$ Un[GETINDEX$(i - 1, j, ny)$]
      $uijm1 \leftarrow$ Un[GETINDEX$(i, j - 1, ny)$]
      $uip1j \leftarrow$ Un[GETINDEX$(i + 1, j, ny)$]
      $uijp1 \leftarrow$ Un[GETINDEX$(i, j + 1, ny)$]
      Unp1$[index] \leftarrow uij + a \cdot dt \cdot \left(\frac{uim1j - 2 \cdot uij + uip1j}{dx^2} + \frac{uijm1 - 2 \cdot uij + uijp1}{dy^2} \right)$
    end for
  end for
end for=0

to parrallelize this calculation we have tested 3 different approaches :

1. **Approach 1:** At each iteration, the grid is copied from host to device, calculations are performed, and results are sent back to the host. This ensures up-to-date synchronization but incurs significant data transfer overhead.

2. **Approach 2:** Similar to the first approach, but the grid is copied back from device to host only when results need to be displayed. This reduces data transfer frequency, improving performance while potentially sacrificing real-time result observation.

3. **Approach 3:** Extends the second approach by utilizing shared memory on the device for calculations, reducing global memory access and enhancing performance by leveraging faster memory access within CUDA blocks.

Results of those approaches are illustrated in the next section

Limits of this approach

1. Stability Condition : The FTCS method is conditionally stable, and its stability is governed by the following criterion:

$$\Delta t \leq \frac{1}{2\alpha} \left(\frac{1}{(\Delta x)^2} + \frac{1}{(\Delta y)^2} \right)^{-1}$$

   where Δt is the time step, Δx is the spatial step, and α is the thermal diffusivity of the material. so we must make sure that dt satisfy the condition

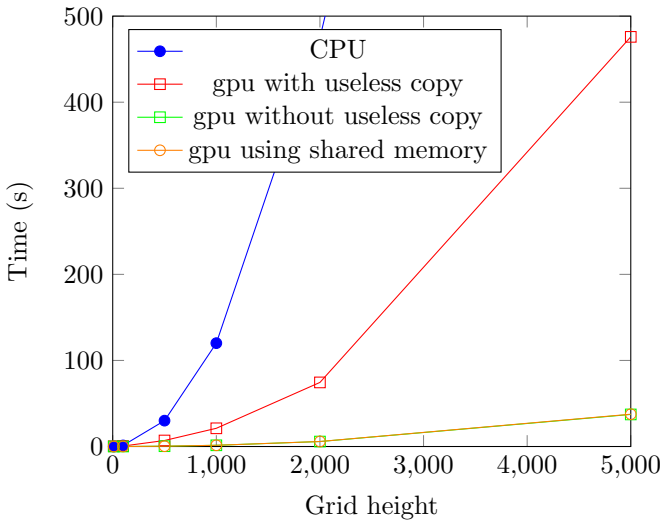
2. Accuracy : The FTCS method is first-order accurate in time and second-order accurate in space, which may not be sufficient for simulations requiring high precision:

$$O(\Delta t, (\Delta x)^2)$$

2 Results/Data Analysis

Analyzing the latency of each algorithm, these results were obtained with a distant connection to the Ensimag server, which increases the latency. The results are presented as follows :

Line graph representing the latency based on the grid size



Grid Height	CPU	GPU with useless copy	GPU without useless copy	GPU using shared memory
10	0.02	0.31	0.03	0.03
100	1.26	0.55	0.03	0.03
500	30.04	6.98	0.36	0.36
1000	120.07	21.17	1.43	1.43
2000	474.82	74.46	5.76	5.83
5000	2000	475.91	37.26	37.54

Table representing the latency of each approach testing with different grid size

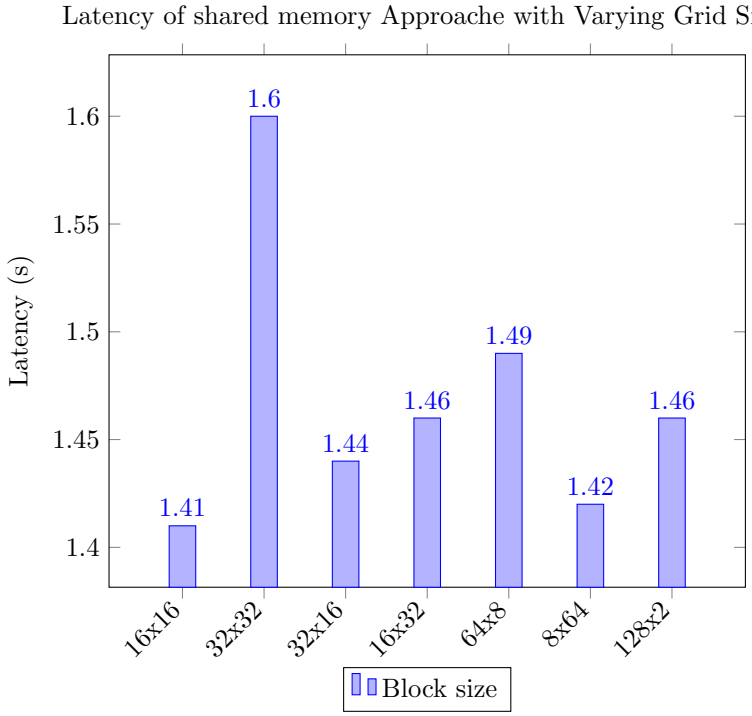
The comparison between CPU and GPU performance in this study clearly shows that GPUs outperform CPUs in all tested grid sizes, underscoring the superior parallel processing capabilities of GPUs. As the grid height increases, we observe a corresponding rise in latency across all computational methods. This latency increase is almost linear for CPUs, but for GPUs, the scenario changes significantly. GPUs that engage in unnecessary data copying exhibit a pronounced increase in latency, highlighting the detrimental impact of redundant memory operations. Conversely, GPUs that avoid such needless copying and utilize shared memory experience a much lower rate of latency growth.

A notable observation from the study is the significant latency disparity between GPU configurations with and without unnecessary copying. This finding emphasizes the performance bottleneck caused by inefficient memory operations. By eliminating these unnecessary data transfers, a substantial boost in performance is achieved. Interestingly, when comparing the performance of GPUs with and without needless copying to those employing shared memory, the differences in latency are minimal. This indicates that for the specific case of heat diffusion simulations, the advantage of shared memory is marginal compared to the benefits gained by simply avoiding unnecessary memory operations.

In terms of scalability, GPU methods prove to be superior to CPU methods. The CPU's latency increases sharply with grid size, from 0.02ms at a grid height of 10 to 2000ms at 5000. In stark contrast, the GPU maintains a more modest increase in latency, peaking at 37.54ms for the largest grid height tested, provided unnecessary data copies are omitted.

These observations highlight the importance of optimizing GPU memory operations in latency-sensitive applications like heat diffusion simulations. The data suggests that developers should focus on reducing unnecessary memory copies and strategically manage memory usage to maintain high levels of performance. In conclusion, the inherent parallel processing strength of GPUs makes them highly suitable for large-scale grid computations seen in heat diffusion simulations. Furthermore, performance

optimization, particularly in reducing memory operations, plays a crucial role in enhancing the overall efficiency of the computational process.



The analysis of GPU block configurations and their impact on latency reveals that smaller block sizes, particularly the 16x16 configuration, achieve the lowest latency at 1.41 ms. This suggests that smaller blocks are more efficient for these types of operations. In contrast, larger block sizes, such as the 32x32 configuration, exhibit higher latencies, reaching up to 1.60 ms. This increase in latency could be attributed to factors like thread management overhead or resource contention, which tend to reduce computational efficiency. Rectangular block configurations, such as 32x16 and 16x32, show relatively similar latencys of 1.44 ms and 1.46 ms, respectively, indicating that the shape of the block has a less significant impact on performance than its overall size. However, the performance of asymmetric configurations, such as 8x64 at 1.42 ms and 128x2 at 1.46 ms, challenges this observation, suggesting that it's possible to maintain consistent performance across a wide range of block dimensions.

Having identified the optimal algorithm and block size, we can proceed to incorporate this configuration into our real-time simulation application. We will adopt the shared memory approach with a block size of 16x16 for a simulation grid of 1000x1000. Thus, in our final application, we will utilize the shared memory strategy with a 16x16 block size, optimizing performance for our real-time simulation needs.

3 Results/Data Analysis

4 Conclusion

The application has demonstrated exceptional performance in simulating heat diffusion across a 2D plane in real-time. The transition from a CPU-based to a GPU-accelerated approach has notably enhanced the speed of computations and reduced the program's latency. This improvement is particularly evident in the dynamic representation of heat distribution, where the GPU's parallel processing capabilities significantly outperform the CPU's sequential execution.

However, challenges arise when scaling the resolution of the simulation grid, particularly at dimensions as large as 10,000 by 10,000. At this resolution, even the GPU-accelerated program encounters noticeable latency. This bottleneck highlights the need for further optimization. Investigating deeper, it becomes apparent that optimizing memory access patterns could yield substantial performance gains.

Avoiding bank conflicts and restructuring variables to enable coalesced access to global memory can enhance the efficiency of memory utilization on the GPU.

Furthermore, leveraging the interoperability features between CUDA and OpenGL presents an opportunity to streamline the visualization of simulation results. By directly rendering the results from the GPU memory using OpenGL shaders and buffers, the need to transfer processed data back to the host for display is eliminated. This approach not only reduces data transfer overhead but also exploits the GPU's graphics rendering capabilities, potentially leading to a significant leap in performance.

When it comes to numerical methods for simulating heat diffusion, it is crucial to acknowledge the trade-offs between simplicity, speed, and accuracy. Explicit numerical methods, while straightforward and conditionally stable, may not always provide the desired precision for complex simulations. In such cases, turning to implicit numerical methods can offer a more accurate solution. Examples of such methods include the Crank-Nicolson method, which offers a good balance between stability and accuracy, and the Backward Euler method, known for its unconditional stability in time-dependent problems. These methods, while more computationally intensive, can provide more precise results in simulations where fine detail and accuracy are paramount.

Figure 3: This frog was uploaded via the file-tree menu.

Item	Quantity
Widgets	42
Gadgets	13

Table 1: An example table.

Your introduction goes here! Simply start writing your document and use the Recompile button to view the updated PDF preview. Examples of commonly used commands and features are listed below, to help you get started.

Once you're familiar with the editor, you can find various project settings in the Overleaf menu, accessed via the button in the very top left of the editor. To view tutorials, user guides, and further documentation, please visit our help library, or head to our plans page to choose your plan.

5 Design Methodology

5.1 How to create Sections and Subsections

Simply use the section and subsection commands, as in this example document! With Overleaf, all the formatting and numbering is handled automatically according to the template you've chosen. If you're using the Visual Editor, you can also create new section and subsections via the buttons in the editor toolbar.

5.2 How to include Figures

First you have to upload the image file from your computer using the upload link in the file-tree menu. Then use the includegraphics command to include it in your document. Use the figure environment and the caption command to add a number and a caption to your figure. See the code for Figure 3 in this section for an example.

Note that your figure will automatically be placed in the most appropriate place for it, given the surrounding text and taking into account other figures or tables that may be close by. You can find out more about adding images to your documents in this help article on including images on Overleaf.

5.3 How to add Tables

Use the table and tabular environments for basic tables — see Table 1, for example. For more information, please see this help article on tables.

5.4 How to add Comments and Track Changes

Comments can be added to your project by highlighting some text and clicking "Add comment" in the top right of the editor pane. To view existing comments, click on the Review menu in the toolbar above. To reply to a comment, click on the Reply button in the lower right corner of the comment. You can close the Review pane by clicking its name on the toolbar when you're done reviewing for the time being.

Track changes are available on all our premium plans, and can be toggled on or off using the option at the top of the Review pane. Track changes allow you to keep track of every change made to the document, along with the person making the change.

5.5 How to add Lists

You can make lists with automatic numbering ...

1. Like this,
2. and like this.

... or bullet points ...

- Like this,
- and like this.

5.6 How to write Mathematics

LaTeX is great at typesetting mathematics. Let $X_1, X_2, \ldots, X_n$ be a sequence of independent and identically distributed random variables with $\mathrm{E}[X_i] = \mu$ and $\mathrm{Var}[X_i] = \sigma^2 < \infty$, and let

$$S_n = \frac{X_1 + X_2 + \cdots + X_n}{n} = \frac{1}{n}\sum_{i}^{n} X_i$$

denote their mean. Then as n approaches infinity, the random variables $\sqrt{n}(S_n - \mu)$ converge in distribution to a normal $\mathcal{N}(0, \sigma^2)$.

5.7 How to change the margins and paper size

Usually the template you're using will have the page margins and paper size set correctly for that use-case. For example, if you're using a journal article template provided by the journal publisher, that template will be formatted according to their requirements. In these cases, it's best not to alter the margins directly.

If however you're using a more general template, such as this one, and would like to alter the margins, a common way to do so is via the geometry package. You can find the geometry package loaded in the preamble at the top of this example file, and if you'd like to learn more about how to adjust the settings, please visit this help article on page size and margins.

5.8 How to change the document language and spell check settings

Overleaf supports many different languages, including multiple different languages within one document.

To configure the document language, simply edit the option provided to the babel package in the preamble at the top of this example project. To learn more about the different options, please visit this help article on international language support.

To change the spell check language, simply open the Overleaf menu at the top left of the editor window, scroll down to the spell check setting, and adjust accordingly.

5.9 How to add Citations and a References List

You can simply upload a `.bib` file containing your BibTeX entries, created with a tool such as JabRef. You can then cite entries from it, like this: [Gre93]. Just remember to specify a bibliography style, as well as the filename of the `.bib`. You can find a video tutorial here to learn more about BibTeX.

If you have an upgraded account, you can also import your Mendeley or Zotero library directly as a `.bib` file, via the upload menu in the file-tree.

5.10 Good luck!

We hope you find Overleaf useful, and do take a look at our help library for more tutorials and user guides! Please also let us know if you have any feedback using the Contact Us link at the bottom of the Overleaf menu — or use the contact form at `https://www.overleaf.com/contact`.

References

[Gre93] George D. Greenwade. The Comprehensive Tex Archive Network (CTAN). *TUGBoat*, 14(3):342–351, 1993.