École Nationale Supérieure d'Informatique pour l'Industrie et l'Entreprise

Rapport du projet PAP

Équation de la chaleur

Étudiants : RA Veasna
DIN Sokheng

Enseignant : M. TORRI Vincent

04 janvier 2026

Contents

1 Introduction

L'équation de la chaleur est une équation aux dérivées partielles qui modélise la distribution de la chaleur (ou les changements de température) dans un domaine spécifique au fil du temps.

Supposons que l'on ait une fonction u qui représente la température en un lieu spécifique. Cette fonction évoluera dans le temps à mesure que la chaleur se diffuse dans l'espace. On utilise l'équation de la chaleur pour calculer comment cette fonction u évolue avec le temps. Le taux de variation de u est proportionnel à la "courbure" de u.

Pour une fonction $u(t, x)$ de la variable x et de la variable temps t, l'équation de la chaleur est:

$$\frac{\partial u}{\partial t} = \frac{\lambda}{\rho c}\Delta u + \frac{F}{\rho c} \tag{1}$$

où

$$u : \mathbb{R}^+ \times [0, L]^d \to \mathbb{R}^+ \tag{2}$$

$$(t, x) \mapsto u(t, x) \tag{3}$$

- $u(t, x)$: la température en °K du matériau étudié au temps t (s) et en point x (m)
- F : la source de chaleur appliquée (une flamme ou une source de froid)
- λ : la conductivité thermique
- ρ : la masse volumique
- c : la chaleur massique du matériau

Le tableau suivant contient les valeurs dans les unités internationales pour les matériaux considérés dans ce projet:

Matériau	λ (W/(m·K))	ρ (kg/m^3)	c (J/(kg·K))
Cuivre	389	8940	380
Fer	80.2	7874	440
Verre	1.2	2530	840
Polystyrène	0.1	1040	1200

Table 1: Propriétés thermiques des matériaux

1.1 Objectif

Le but de ce projet est de mettre en place cette simulation en C++ en utilisant la méthode des différences finies pour deux scénarios distincts, à savoir une barre mince et une plaque mince, chacune ayant des conditions aux limites spécifiques dans le cas implicite, et créer une animation de la solution de température au fil du temps avec la bibliothèque SDL.

2 Premier cas : la barre

Dans ce cas, nous travaillons avec une barre infiniment mince ($d = 1$) de longueur L. Pour la barre: condition de Neumann en $x = 0$ et de Dirichlet en $x = L$.

2.1 Discrétisation

Discrétisation temporelle

$t_0 = 0, t_1, \ldots, t_{M-2}, t_{\max} = t_{M-1}$.

Dans notre implémentation, nous utilisons un nombre fixe de $M = 1000$ pas de temps pour diviser l'intervalle $[0, t_{\max}]$. Mathématiquement, on définit:

$$\Delta t = \frac{t_{\max}}{1000} \quad \text{et} \quad t_n = n \times \Delta t \quad \text{pour } n \in \{0, \ldots, 999\}$$

Discrétisation de l'espace

$x_0 = 0, x_1, \ldots, x_{N-2}, x_{N-1} = L$.

Pour $N \in \mathbb{N}^*$, on divise l'intervalle $[0, L]$ par N points répartis uniformément dans cet intervalle. Mathématiquement, on définit:

$$\Delta x = \frac{L}{N-1} \quad \text{et} \quad x_j = j \times \Delta x \quad \text{pour } j \in \{0, \ldots, N-1\}$$

2.2 Préliminaires mathématiques

Nous avons l'équation suivante:

$$\begin{cases} \dfrac{\partial u}{\partial t} = \dfrac{\lambda}{\rho c}\dfrac{\partial^2 u}{\partial x^2} + \dfrac{F}{\rho c} & \\ u(0,x) = u_0, & \forall x \in [0, L] \\ \dfrac{\partial u}{\partial x}(t,0) = 0, & \forall t \in [0, t_{\max}] \\ u(t,L) = u_0, & \forall t \in [0, t_{\max}] \end{cases} \tag{4}$$

Pour appliquer la méthode des différences finies [1], nous subdivisons le temps et l'espace en intervalles infinitésimaux, nous permettant d'approximer les dérivées par des différences. Typiquement, nous avons $t = n \cdot \Delta t$ et $x = i \cdot \Delta x$ où n et i sont des entiers.

Nous commençons avec l'équation de la chaleur unidimensionnelle avec un terme source:

$$\frac{\partial u}{\partial t} = \frac{\lambda}{\rho c}\frac{\partial^2 u}{\partial x^2} + \frac{F}{\rho c} \tag{5}$$

En utilisant un développement de Taylor au premier ordre, nous pouvons discrétiser la dérivée temporelle:

$$\frac{\partial u}{\partial t} \approx \frac{u_i^{n+1} - u_i^n}{\Delta t} \tag{6}$$

De même, au second ordre, nous pouvons discrétiser la seconde dérivée spatiale:

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{\Delta x^2} \tag{7}$$

En substituant ces éléments dans l'équation de la chaleur et en réarrangeant les termes, nous obtenons le schéma implicite:

$$-ru_{i-1}^{n+1} + (1+2r)u_i^{n+1} - ru_{i+1}^{n+1} = u_i^n + \Delta t \frac{F_i}{\rho c} \tag{8}$$

où $r = \frac{\alpha \Delta t}{\Delta x^2}$ avec $\alpha = \frac{\lambda}{\rho c}$.

Cela conduit à un système d'équations linéaires qui peut être écrit sous forme matricielle comme $AU^{n+1} = BU^n + F^n$, où A est la matrice des coefficients, B est une matrice diagonale représentant l'étape temporelle précédente, et F^n est le vecteur du terme source à l'étape de temps actuelle.

La source de chaleur

La source de chaleur F varie le long de la barre et est définie comme:

$$F(x) = \begin{cases} t_{\max} f^2, & \text{si } x \in \left[\frac{L}{10}, \frac{2L}{10}\right], \\ \frac{3t_{\max} f^2}{4}, & \text{si } x \in \left[\frac{5L}{10}, \frac{6L}{10}\right], \\ 0, & \text{sinon.} \end{cases} \tag{9}$$

Matrices du système

Dans la méthode des différences finies implicite, la matrice des coefficients A pour un nœud intérieur i est tridiagonale:

$$A = \begin{bmatrix} 1+2r & -r & 0 & \cdots & 0 \\ -r & 1+2r & -r & \cdots & 0 \\ 0 & -r & 1+2r & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1+2r \end{bmatrix} \tag{10}$$

où $r = \frac{\lambda \Delta t}{\rho c \Delta x^2}$.

La matrice B est une matrice identité mise à l'échelle par $\frac{1}{\Delta t}$:

$$B = \begin{bmatrix} \frac{1}{\Delta t} & 0 & 0 & \cdots & 0 \\ 0 & \frac{1}{\Delta t} & 0 & \cdots & 0 \\ 0 & 0 & \frac{1}{\Delta t} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \frac{1}{\Delta t} \end{bmatrix} \tag{11}$$

Le vecteur de température U^n à l'étape de temps actuelle n est:

$$U^n = \begin{bmatrix} u_0^n \\ u_1^n \\ u_2^n \\ \vdots \\ u_{N-1}^n \end{bmatrix} \tag{12}$$

Le vecteur du membre de droite D^n est défini par:

$$D_i^n = u_i^n + \Delta t \frac{F_i}{\rho c}, \quad \text{pour } i \in \{0, \ldots, N-1\} \tag{13}$$

2.3 Algorithme et Solution

Nous utilisons l'algorithme de la matrice tridiagonale, également connu sous le nom d'**algorithme de Thomas** [3] (TDMA - TriDiagonal Matrix Algorithm), qui est une méthode directe et efficace pour résoudre des systèmes d'équations linéaires tridiagonaux avec une complexité algorithmique de $O(N)$.

Algorithme de Thomas (TDMA)

Pour un système tridiagonal de la forme:

$$a_i u_{i-1} + b_i u_i + c_i u_{i+1} = d_i \tag{14}$$

L'algorithme de Thomas fonctionne en deux phases:

1. **Élimination directe (Forward Sweep)**:

On transforme le système en un système triangulaire supérieur en calculant des coefficients modifiés:

$$c'_i = \begin{cases} \dfrac{c_0}{b_0}, & i = 0 \\ \dfrac{c_i}{b_i - a_i c'_{i-1}}, & i = 1, \ldots, N-2 \end{cases} \tag{15}$$

$$d'_i = \begin{cases} \dfrac{d_0}{b_0}, & i = 0 \\ \dfrac{d_i - a_i d'_{i-1}}{b_i - a_i c'_{i-1}}, & i = 1, \ldots, N-1 \end{cases} \tag{16}$$

2. **Substitution rétrograde (Back Substitution)**:
   À partir de la dernière équation, les valeurs de u_i sont calculées par substitution arrière:

$$u_{N-1} = d'_{N-1} \tag{17}$$
$$u_i = d'_i - c'_i u_{i+1}, \quad i = N-2, \ldots, 0 \tag{18}$$

Application aux conditions aux limites

Pour implémenter les conditions aux limites dans notre système:

- **Condition de Neumann en** $x = 0$: $\frac{\partial u}{\partial x}(t, 0) = 0$
  On modifie la première ligne du système:

$$b_0 = 1 + r \tag{19}$$
$$c_0 = -r \tag{20}$$
$$a_0 = 0 \tag{21}$$

- **Condition de Dirichlet en** $x = L$: $u(t, L) = u_0$
  On impose directement la valeur à la dernière ligne:

$$b_{N-1} = 1 \tag{22}$$
$$c_{N-1} = 0 \tag{23}$$
$$a_{N-1} = 0 \tag{24}$$
$$d_{N-1} = u_0 \tag{25}$$

2.4 Résultats de simulation 1D

Les figures suivantes montrent l'évolution de la température pour les quatre matériaux étudiés dans le cas unidimensionnel. Les cartes de chaleur illustrent la distribution spatiale de la température à différents instants de la simulation.

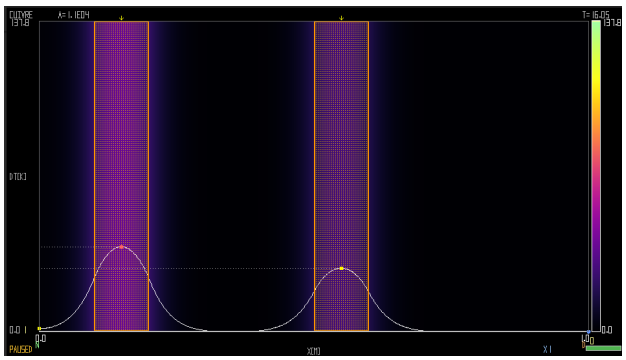

Figure 1: Cuivre 1D

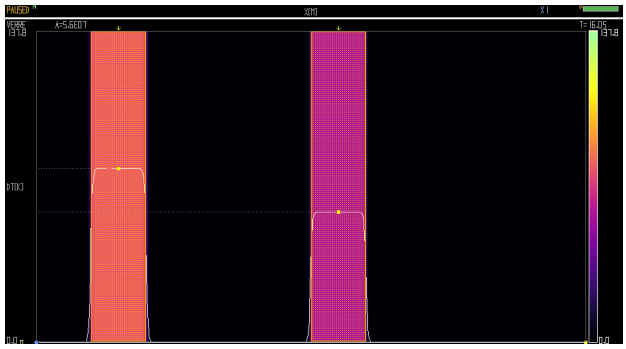

Figure 2: Verre 1D

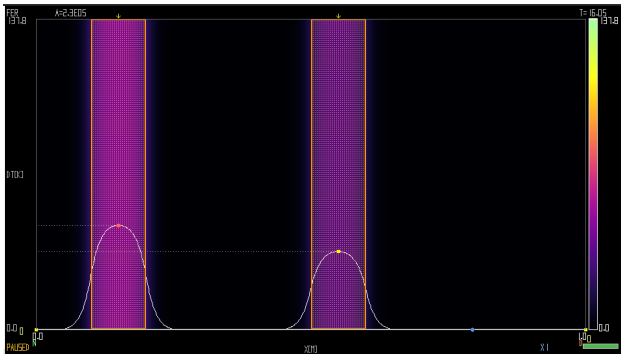

Figure 3: Fer 1D

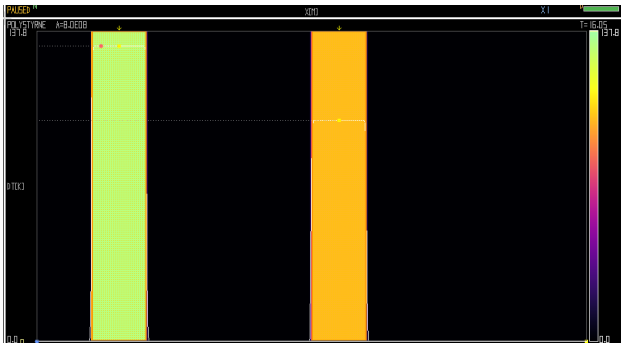

Figure 4: Polystyrène 1D

3 Deuxième cas : la plaque

Dans ce cas, nous travaillons dans l'espace bidimensionnel et notre matériau est une plaque carrée de côté L. Pour la plaque: condition de Neumann en $x = 0$ et $y = 0$, condition de Dirichlet en $x = L$ et $y = L$. L'équation de chaleur devient:

$$\frac{\partial u}{\partial t} = \frac{\lambda}{\rho c}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + \frac{F}{\rho c} \tag{26}$$

3.1 Discrétisation

Discrétisation temporelle

Nous utilisons la même discrétisation temporelle que dans le cas unidimensionnel.

Pour un nombre fixe de pas de temps $M = 1000$, on définit:

$$\Delta t = \frac{t_{\max}}{1000} \quad \text{et} \quad t_n = n \times \Delta t, \quad \forall n \in \{0, 1, \ldots, 999\}$$

Discrétisation spatiale

Nous divisons notre domaine carré $[0, L] \times [0, L]$ par une grille régulière de $N \times N$ points.

Pour $N \in \mathbb{N}^*$, soit:

$$\Delta x = \Delta y = \frac{L}{N-1}$$

Il y a $N \times N$ points (x_i, y_j) distribués uniformément dans $[0, L]^2$:

$$(x_i, y_j) = (i\Delta x, j\Delta y), \quad \forall (i,j) \in \{0, 1, \ldots, N-1\}^2$$

Si u est la solution de l'équation de la chaleur 2D (26), alors on note son approximation:

$$u(t_n, x_i, y_j) \approx u_{i,j}^n, \quad \forall (n,i,j) \in \{0, \ldots, 999\} \times \{0, \ldots, N-1\}^2 \tag{27}$$

3.2 Préliminaires mathématiques

Nous avons l'équation suivante avec ses conditions aux limites et initiale:

$$\begin{cases} \dfrac{\partial u}{\partial t} = \dfrac{\lambda}{\rho c}\left(\dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2}\right) + \dfrac{F}{\rho c}, & \forall (t,x,y) \in [0, t_{\max}] \times [0, L]^2 \\ u(0,x,y) = u_0, & \forall (x,y) \in [0, L]^2 \\ \dfrac{\partial u}{\partial x}(t,0,y) = 0, & \forall (t,y) \in [0, t_{\max}] \times [0, L] \\ \dfrac{\partial u}{\partial y}(t,x,0) = 0, & \forall (t,x) \in [0, t_{\max}] \times [0, L] \\ u(t,L,y) = u_0, & \forall (t,y) \in [0, t_{\max}] \times [0, L] \\ u(t,x,L) = u_0, & \forall (t,x) \in [0, t_{\max}] \times [0, L] \end{cases} \tag{28}$$

Nous commençons par considérer le temps $t = n\Delta t$ et les positions spatiales $x = i\Delta x$ et $y = j\Delta y$, où n, i, et j sont des entiers.

Étape 1 : Discrétisation temporelle

Utilisant un développement de Taylor au premier ordre (schéma d'Euler implicite), nous approximons la dérivée temporelle par:

$$\frac{\partial u}{\partial t} \approx \frac{u_{i,j}^{n+1} - u_{i,j}^n}{\Delta t} \tag{29}$$

Étape 2 : Discrétisation spatiale

De même, avec un développement de Taylor au second ordre, nous discrétisons les dérivées secondes spatiales:

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u_{i+1,j}^{n+1} - 2u_{i,j}^{n+1} + u_{i-1,j}^{n+1}}{\Delta x^2} \tag{30}$$

$$\frac{\partial^2 u}{\partial y^2} \approx \frac{u_{i,j+1}^{n+1} - 2u_{i,j}^{n+1} + u_{i,j-1}^{n+1}}{\Delta y^2} \tag{31}$$

Étape 3 : Formation du schéma implicite

En substituant ces approximations dans l'équation de la chaleur et en réarrangeant les termes, nous obtenons le schéma à 5 points (5-point stencil):

$$-ru_{i-1,j}^{n+1} - ru_{i,j-1}^{n+1} + (1+4r)u_{i,j}^{n+1} - ru_{i+1,j}^{n+1} - ru_{i,j+1}^{n+1} = u_{i,j}^n + \Delta t \frac{F_{i,j}}{\rho c} \tag{32}$$

où $r = \frac{\alpha \Delta t}{\Delta x^2}$ avec $\alpha = \frac{\lambda}{\rho c}$ la diffusivité thermique.

La source de chaleur

La source de chaleur F est répartie symétriquement sur quatre régions rectangulaires situées aux coins de la plaque:

$$F(x,y) = \begin{cases} t_{\max} f^2 \times s, & \text{si } (x,y) \in \mathcal{R}_1 \cup \mathcal{R}_2 \cup \mathcal{R}_3 \cup \mathcal{R}_4 \\ 0, & \text{sinon} \end{cases} \tag{33}$$

où les quatre régions sont définies par:

$$\mathcal{R}_1 = \left[\frac{L}{6}, \frac{2L}{6}\right] \times \left[\frac{L}{6}, \frac{2L}{6}\right] \quad \text{(coin inférieur gauche)} \tag{34}$$

$$\mathcal{R}_2 = \left[\frac{4L}{6}, \frac{5L}{6}\right] \times \left[\frac{L}{6}, \frac{2L}{6}\right] \quad \text{(coin inférieur droit)} \tag{35}$$

$$\mathcal{R}_3 = \left[\frac{L}{6}, \frac{2L}{6}\right] \times \left[\frac{4L}{6}, \frac{5L}{6}\right] \quad \text{(coin supérieur gauche)} \tag{36}$$

$$\mathcal{R}_4 = \left[\frac{4L}{6}, \frac{5L}{6}\right] \times \left[\frac{4L}{6}, \frac{5L}{6}\right] \quad \text{(coin supérieur droit)} \tag{37}$$

et $s = 100$ est le facteur d'amplification pour améliorer la visualisation.

3.3 Algorithme et Solution

Contrairement au cas unidimensionnel, le système d'équations résultant pour le problème 2D n'est plus tridiagonal mais devient un système matriciel de très grande taille ($N^2 \times N^2$). La résolution directe par factorisation serait trop coûteuse en mémoire et en temps de calcul.

Méthode itérative de Gauss-Seidel

Nous utilisons la **méthode de Gauss-Seidel** [2], qui est une méthode itérative efficace pour résoudre des systèmes linéaires creux. Pour chaque point (i, j) de la grille, nous résolvons l'équation:

$$u_{i,j}^{n+1} = \frac{1}{1+4r}\left[u_{i,j}^{n} + \Delta t \frac{F_{i,j}}{\rho c} + r(u_{i-1,j}^{n+1} + u_{i+1,j}^{n+1} + u_{i,j-1}^{n+1} + u_{i,j+1}^{n+1})\right] \tag{38}$$

Algorithme itératif

L'algorithme de Gauss-Seidel fonctionne comme suit:

1. **Initialisation**: $u_{i,j}^{n+1,(0)} = u_{i,j}^{n}$ pour tous les points intérieurs
2. **Itération**: Pour $k = 0, 1, 2, \ldots$ jusqu'à convergence:
   Pour chaque point (i, j) de la grille (sauf les bords):

$$u_{i,j}^{n+1,(k+1)} = \frac{r(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1}) + u_{i,j}^{n} + \Delta t \frac{F_{i,j}}{\rho c}}{1+4r} \tag{39}$$

3. **Critère de convergence**: L'itération s'arrête lorsque:

$$\max_{i,j} |u_{i,j}^{n+1,(k+1)} - u_{i,j}^{n+1,(k)}| < \epsilon \tag{40}$$

   où $\epsilon = 10^{-6}$ est la tolérance et le nombre maximal d'itérations est fixé à 100.

Implémentation des conditions aux limites

- **Conditions de Neumann en $x = 0$ et $y = 0$**: $\frac{\partial u}{\partial x}(t, 0, y) = 0$ et $\frac{\partial u}{\partial y}(t, x, 0) = 0$
  On utilise la technique du miroir (réflexion):

$$u_{-1,j}^{n+1} = u_{1,j}^{n+1} \quad \text{(miroir en } x = 0) \tag{41}$$

$$u_{i,-1}^{n+1} = u_{i,1}^{n+1} \quad \text{(miroir en } y = 0) \tag{42}$$

  Dans l'implémentation, lorsque $i = 0$ ou $j = 0$, on remplace le voisin gauche/bas par le voisin droit/haut correspondant.

- **Conditions de Dirichlet en $x = L$ et $y = L$**: $u(t, L, y) = u_0$ et $u(t, x, L) = u_0$
  On impose directement:

$$u_{N-1,j}^{n+1} = u_0, \quad \forall j \tag{43}$$

$$u_{i,N-1}^{n+1} = u_0, \quad \forall i \tag{44}$$

Avantages de la méthode de Gauss-Seidel

- **Économie de mémoire**: Pas besoin de stocker la matrice complète $N^2 \times N^2$
- **Simplicité d'implémentation**: Mise à jour locale de chaque point
- **Efficacité**: Convergence rapide pour les systèmes issus de la discrétisation de l'équation de la chaleur
- **Flexibilité**: Facile d'adapter aux différentes conditions aux limites

3.4 Résultats de simulation 2D

Les figures suivantes montrent l'évolution de la température pour les quatre matériaux étudiés dans le cas bidimensionnel. Les cartes de chaleur illustrent la distribution spatiale de la température à différents instants de la simulation.

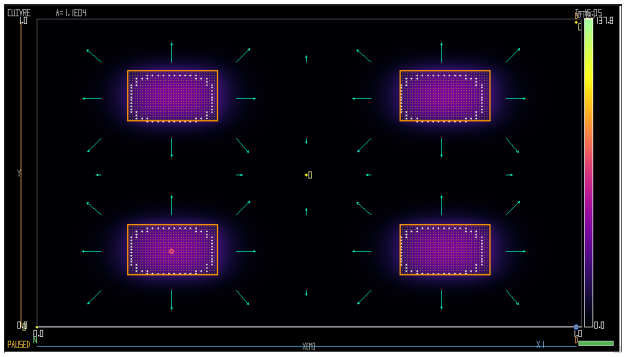

Figure 5: Cuivre 2D

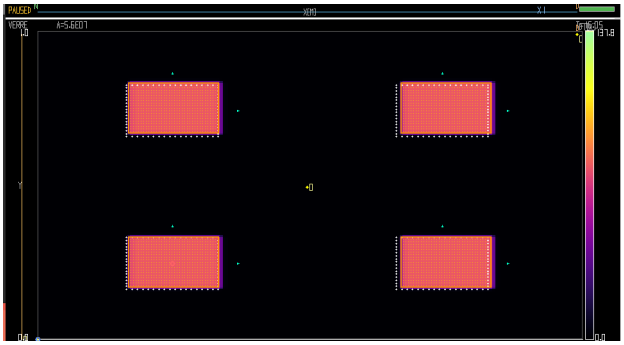

Figure 6: Verre 2D

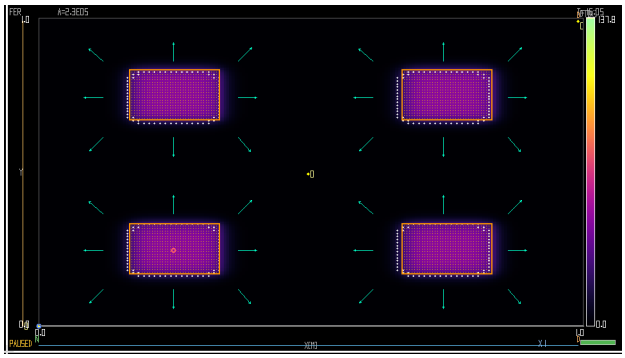

Figure 7: Fer 2D

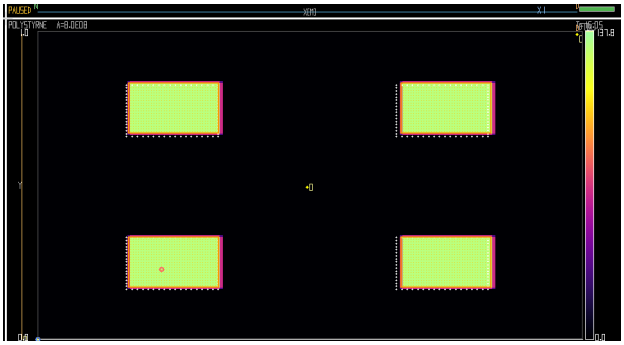

Figure 8: Polystyrène 2D

4 Conception et Diagramme de classes UML

En raison du nombre important d'attributs et de méthodes pour chaque classe et du contenu du projet, les relations entre les classes sont quelque peu complexes. J'ai donc décidé de dessiner le diagramme UML dans une version optimisée qui garantit la représentation complète du pipeline de mon projet.

La Figure 9 présente le diagramme de classes complet de l'application. Ce diagramme illustre les relations entre les différents composants du projet.

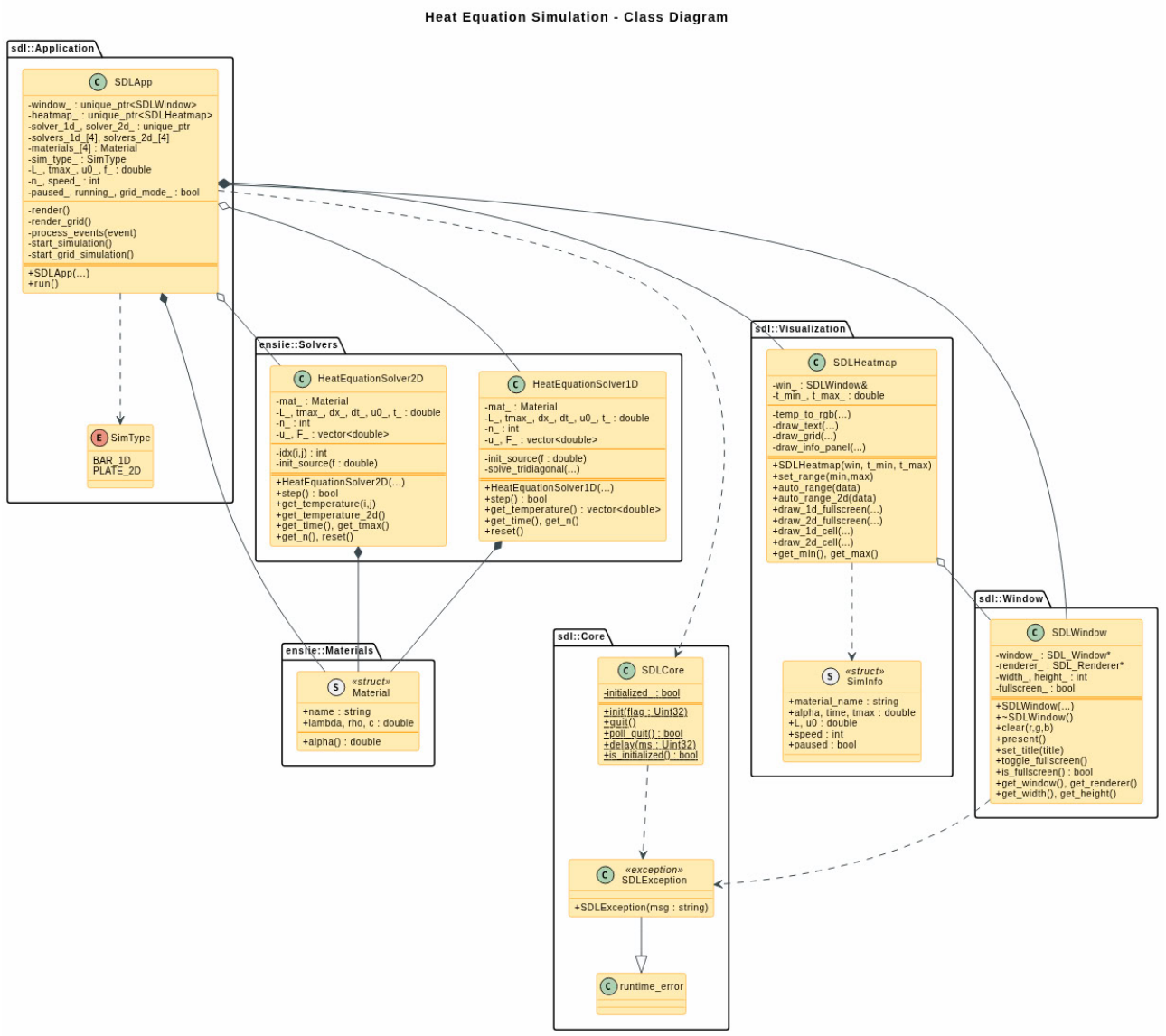

Figure 9: Diagramme de classes UML complet du projet

5 Compilation et Manuel d'utilisation du Projet

5.1 Installation et Lancement du programme

Le projet est développé en `C++` et utilise la bibliothèque graphique `SDL2` pour la visualisation des résultats de la simulation de l'équation de la chaleur. La compilation s'effectue à l'aide du compilateur `g++`, en activant les options de débogage et les avertissements afin de garantir la robustesse du code.

- Compiler le projet avec la commande suivante :

  ```
  g++ -g -Wall -Wextra -o prog *.cpp $(pkg-config --cflags --libs sdl2)
  ```

- Exécuter le programme compilé: `./prog`

La commande pour générer la documentation du projet est :

- Générer la documentation(Doxyfile) avec la commande suivante : `doxygen Doxyfile`
- Après la génération, vous pouvez consulter la documentation HTML: `xdg-open doc/html/index.html`

5.2 Interface Console et Choix de Simulation

Au lancement, le programme affiche un menu textuel permettant de configurer la simulation avant l'ouverture de la fenêtre SDL :

- **Simulation 1D** : barre thermique unidimensionnelle.
- **Simulation 2D** : plaque thermique bidimensionnelle.
- Chaque simulation est exécutée simultanément sur une grille **2×2** représentant quatre matériaux différents : *Cuivre, Fer, Verre et Polystyrène.*

5.3 Paramétrage de la Simulation

Avant le démarrage de la simulation, l'utilisateur peut définir les paramètres physiques principaux :

- Longueur du domaine spatial L (en mètres) ;
- Temps maximal de simulation t_{max} (en secondes) ;
- Température initiale u_0 (en degrés Celsius) ;
- Amplitude de la source thermique f.
- Taper `b` ou `B` pour revenir au menu précédent

Confirmation et démarrage

Avant de lancer la simulation, un écran de confirmation affiche la configuration choisie:

Listing 1: Écran de test de la simulation dans le terminal

```
========================================
   HEAT EQUATION SIMULATOR
   ENSIIE - Master 1
========================================

SELECT SIMULATION TYPE
----------------------
  1. 1D Bar  (All 4 Materials - 2x2 Grid)
  2. 2D Plate (All 4 Materials - 2x2 Grid)
  0. Quit
Choice: 1

PARAMETERS (Enter for default, 'b' to go back)
-----------------------------------------------
Domain length L [1.0] m:
Max time tmax [16.0] s:
Initial temp u0 [13.0] C:
Source amplitude f [80.0] C:

CONFIGURATION (2x2 Grid - All Materials)
----------------------------------------
  Type:      1D Bar
  Materials: Copper, Iron, Glass, Polystyrene
  L=1 m, tmax=16 s
  u0=13 C, f=80 C

Controls: SPACE=pause, R=reset, UP/DOWN=speed, ESC=quit

[S]tart  [B]ack  [Q]uit:
```

Options:

- `S`: Démarrer la simulation
- `B`: Retourner à la configuration des paramètres
- `Q`: Quitter vers le menu principal

5.4 Contrôles pendant la simulation

Une fois la fenêtre SDL ouverte, les contrôles clavier suivants sont disponibles:

Touche	Action
ESPACE	Mettre en pause / Reprendre la simulation
R	Réinitialiser la simulation à l'état initial
FLÈCHE HAUT (↑)	Augmenter la vitesse de simulation
FLÈCHE BAS (↓)	Diminuer la vitesse de simulation
ESC	Quitter la simulation et retourner au menu

Table 2: Contrôles clavier pendant la simulation

5.5 Résultat de la simulation

Simulations 1D

- **Cuivre:** Diffusion rapide, profils larges et lisses
- **Fer:** Diffusion modérée, gradients plus prononcés
- **Verre:** Diffusion limitée, pics localisés aigus
- **Polystyrène:** Diffusion négligeable, chaleur confinée aux sources

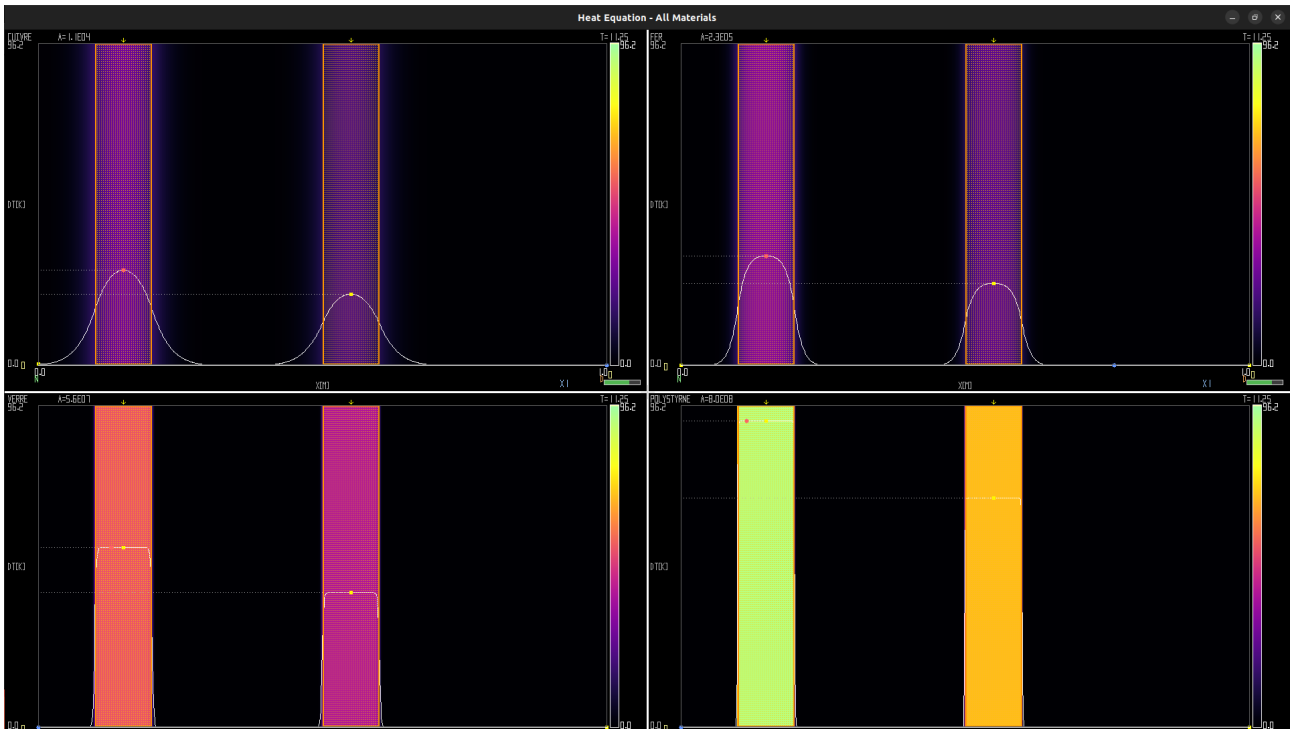

Figure 10: Simulation de 1D avec 4 matériaux différentsts

Simulations 2D

- **Cuivre:** Halos thermiques larges, propagation radiale efficace
- **Fer:** Étalement modéré, isothermes arrondies
- **Verre:** Hautement localisé, frontières nettes
- **Polystyrène:** Isolation thermique maximale, propagation minimale

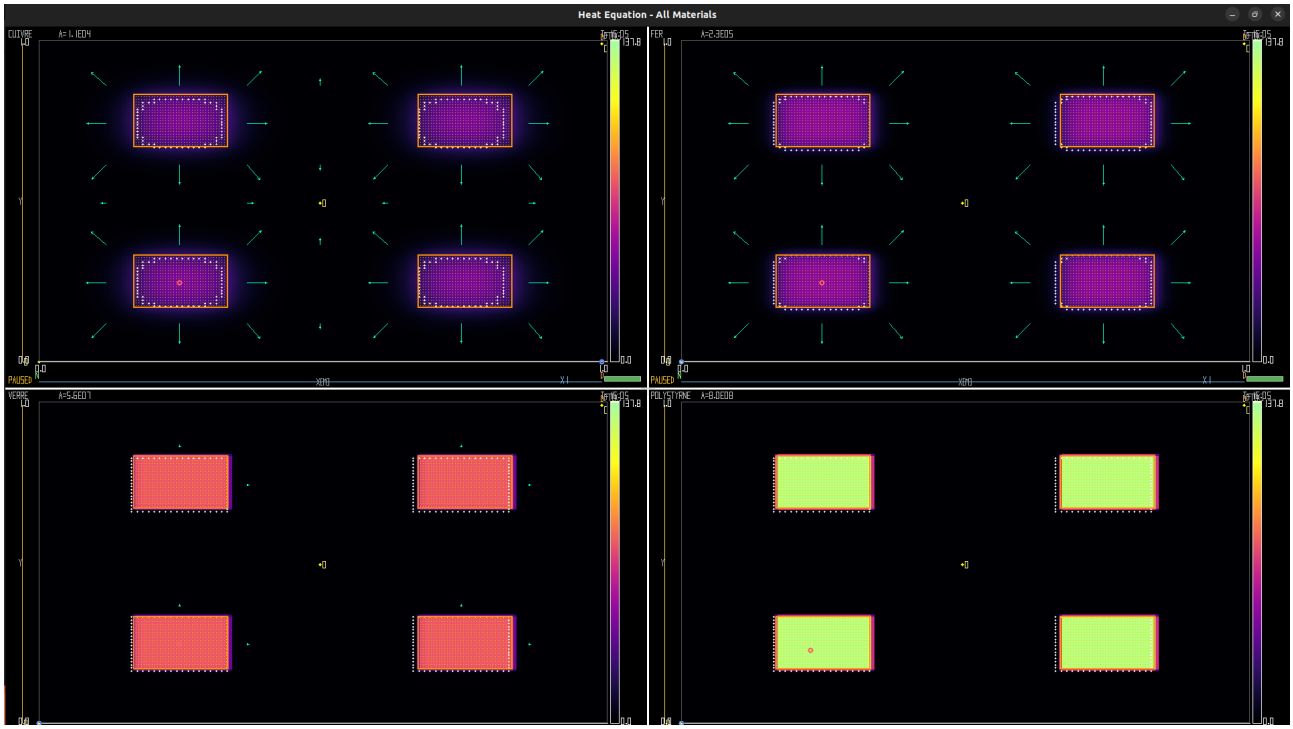

Figure 11: Simulation de 2D avec 4 matériaux différentsts

Le mode grille 2×2 confirme les différences de trois ordres de grandeur en diffusivité thermique à travers l'étendue spatiale observable, la douceur des profils et les motifs d'écoulement de chaleur.

6 Problèmes techniques

Au cours de ce projet, nous avons rencontré plusieurs défis techniques importants, tant au niveau théorique que pratique. Cette section détaille les problèmes majeurs et les solutions adoptées.

6.1 Problème 1 : Stabilité numérique du schéma

Description du problème

L'équation de la chaleur est donnée par:

$$\frac{\partial u}{\partial t} = \frac{\lambda}{\rho c}\Delta u + \frac{F}{\rho c} \tag{45}$$

où $\alpha = \frac{\lambda}{\rho c}$ est la diffusivité thermique.
Les schémas **explicites** (Euler avant) discrétisent cette équation comme suit:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \alpha \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} + \frac{F_i}{\rho c} \tag{46}$$

Ce qui donne:

$$u_i^{n+1} = u_i^n + r(u_{i+1}^n - 2u_i^n + u_{i-1}^n) + \Delta t \frac{F_i}{\rho c} \tag{47}$$

où $r = \frac{\alpha \Delta t}{\Delta x^2}$.

Problème critique: Les schémas explicites ne sont stables que lorsque:

$$r = \frac{\alpha \Delta t}{\Delta x^2} \leq \frac{1}{2} \tag{48}$$

Cela impose: $\Delta t \leq \frac{\Delta x^2}{2\alpha}$
Pour le cuivre avec $\alpha = 1.14 \times 10^{-4}$ m²/s et $\Delta x = 0.01$ m:

$$\Delta t \leq \frac{(0.01)^2}{2 \times 1.14 \times 10^{-4}} = 0.439 \text{ secondes} \tag{49}$$

Pour simuler 16 secondes, il faudrait au minimum 36 pas de temps. Mais pour obtenir une bonne précision, il faudrait des milliers de pas de temps, rendant la visualisation en temps réel impossible.

Solution : Schéma d'Euler implicite (Backward Euler)

Nous utilisons le **schéma d'Euler implicite** (également appelé schéma backward):

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \alpha \frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{\Delta x^2} + \frac{F_i}{\rho c} \tag{50}$$

En réarrangeant:

$$-r \cdot u_{i-1}^{n+1} + (1 + 2r) \cdot u_i^{n+1} - r \cdot u_{i+1}^{n+1} = u_i^n + \Delta t \frac{F_i}{\rho c} \tag{51}$$

Ce schéma est **inconditionnellement stable** pour tout $\Delta t > 0$, permettant d'utiliser des pas de temps beaucoup plus grands tout en maintenant la précision.
Compromis: Nous devons résoudre un système linéaire à chaque pas de temps au lieu d'un calcul direct.

Implémentation pour le cas 1D : Algorithme de Thomas

Le schéma implicite produit un **système tridiagonal**:

$$\begin{bmatrix} b_0 & c_0 & 0 & \cdots & 0 \\ a_1 & b_1 & c_1 & \cdots & 0 \\ 0 & a_2 & b_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & a_{n-1} & b_{n-1} \end{bmatrix} \begin{bmatrix} u_0^{n+1} \\ u_1^{n+1} \\ u_2^{n+1} \\ \vdots \\ u_{n-1}^{n+1} \end{bmatrix} = \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ \vdots \\ d_{n-1} \end{bmatrix} \tag{52}$$

Nous résolvons ce système avec l'**algorithme de Thomas** (TDMA - TriDiagonal Matrix Algorithm) avec une complexité de **O(n)**:

Listing 2: Implémentation de l'algorithme de Thomas

```
void HeatEquationSolver1D::solve_tridiagonal(
    const std::vector<double>& a,  // Lower diagonal
    const std::vector<double>& b,  // Main diagonal
    const std::vector<double>& c,  // Upper diagonal
    std::vector<double>& d,        // Right-hand side
    std::vector<double>& x         // Solution
) {
    int n = b.size();
    std::vector<double> c_star(n);
    std::vector<double> d_star(n);

    // Forward sweep
    c_star[0] = c[0] / b[0];
    d_star[0] = d[0] / b[0];

    for (int i = 1; i < n; i++) {
        double m = 1.0 / (b[i] - a[i] * c_star[i-1]);
        c_star[i] = c[i] * m;
        d_star[i] = (d[i] - a[i] * d_star[i-1]) * m;
    }

    // Back substitution
```

```
    x[n-1] = d_star[n-1];
    for (int i = n-2; i >= 0; i--) {
        x[i] = d_star[i] - c_star[i] * x[i+1];
    }
}
```

Fichier: ***heat_equation_solver.cpp****, lignes 108-145*

Implémentation pour le cas 2D : Méthode de Gauss-Seidel

Pour le cas 2D, la discrétisation devient:

$$\frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t} = \alpha \left(\frac{u_{i+1,j}^{n+1} - 2u_{i,j}^{n+1} + u_{i-1,j}^{n+1}}{\Delta x^2} + \frac{u_{i,j+1}^{n+1} - 2u_{i,j}^{n+1} + u_{i,j-1}^{n+1}}{\Delta y^2} \right) + \frac{F_{i,j}}{\rho c} \tag{53}$$

Avec $\Delta x = \Delta y$, on obtient:

$$(1 + 4r) \cdot u_{i,j}^{n+1} - r \cdot (u_{i+1,j}^{n+1} + u_{i-1,j}^{n+1} + u_{i,j+1}^{n+1} + u_{i,j-1}^{n+1}) = u_{i,j}^{n} + \Delta t \frac{F_{i,j}}{\rho c} \tag{54}$$

Cela crée un **système linéaire creux de grande taille**: une matrice $(n^2 \times n^2)$ avec seulement 5 entrées non nulles par ligne (stencil à 5 points).

Pourquoi pas une solution directe?

- Les méthodes directes denses (comme l'élimination de Gauss sans exploiter la structure creuse) nécessiteraient $O(n^6)$ opérations - totalement infaisable
- Même les solveurs directs creux (comme la factorisation LU creuse) nécessitent beaucoup de mémoire et de complexité
- Pour de grandes grilles (ex: $n = 100 \Rightarrow$ système $10,000 \times 10,000$), les méthodes itératives sont plus pratiques

Notre choix: Nous utilisons l'**itération de Gauss-Seidel**, une méthode itérative qui exploite la structure creuse et met à jour chaque point en utilisant:

$$u_{i,j}^{nouveau} = \frac{1}{1 + 4r} \left[u_{i,j}^{n} + \Delta t \frac{F_{i,j}}{\rho c} + r(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1}) \right] \tag{55}$$

Listing 3: Implémentation de Gauss-Seidel pour le cas 2D

```
bool HeatEquationSolver2D::step() {
    const int max_iter = 100;
    const double tol = 1e-6;

    std::vector<double> u_new = u_;
    double coeff = 1.0 / (1.0 + 4.0 * r);

    for (int iter = 0; iter < max_iter; iter++) {
        double max_diff = 0.0;

        for (int j = 1; j < n_ - 1; j++) {
            for (int i = 1; i < n_ - 1; i++) {
                double rhs = u_[idx(i,j)] + dt_ * F_[idx(i,j)] /
                             (mat_.rho * mat_.c);
                rhs += r * (u_new[idx(i+1,j)] + u_new[idx(i-1,j)] +
                           u_new[idx(i,j+1)] + u_new[idx(i,j-1)]);

                double u_old = u_new[idx(i,j)];
                u_new[idx(i,j)] = coeff * rhs;
                max_diff = std::max(max_diff,
                                    std::abs(u_new[idx(i,j)] - u_old));
            }
        }

        if (max_diff < tol) break;  // Convergence atteinte
```

```
26      }
27
28      u_ = u_new;
29      t_ += dt_;
30      return t_ < tmax_;
31  }
```

Fichier: `heat_equation_solver.cpp`, lignes 247-278

6.2 Problème 2 : Implémentation des conditions aux limites mixtes

Description du problème

Le problème nécessite deux types de conditions aux limites différents:

1. **Condition de Neumann** (flux nul): $\frac{\partial u}{\partial n} = 0$ en $x = 0$ (et $y = 0$ pour le 2D)
2. **Condition de Dirichlet** (température fixe): $u = u_0$ en $x = L$ (et $y = L$ pour le 2D)

Signification physique:

- Condition de Neumann: Frontière isolée - aucun flux de chaleur ne traverse
- Condition de Dirichlet: Frontière maintenue à température constante (puits de chaleur)

Défi: La dérivée dans la condition de Neumann ne s'intègre pas directement dans notre discrétisation.

Solution : Différence finie décentrée avant (Neumann)

Pour la condition de Neumann en $x = 0$, nous utilisons une **approximation par différence finie avant du premier ordre**:

$$\frac{\partial u}{\partial x}\Big|_{x=0} \approx \frac{u_1^{n+1} - u_0^{n+1}}{\Delta x} = 0 \tag{56}$$

Cela donne: $u_0^{n+1} = u_1^{n+1}$ (condition d'égalité à la frontière)
Pour implémenter cela dans le schéma implicite, nous écrivons l'équation de la chaleur en $i = 0$:

$$\frac{u_0^{n+1} - u_0^n}{\Delta t} = \alpha \frac{u_1^{n+1} - u_0^{n+1}}{\Delta x^2} + \frac{F_0}{\rho c} \tag{57}$$

En utilisant $r = \frac{\alpha \Delta t}{\Delta x^2}$:

$$u_0^{n+1} - u_0^n = r(u_1^{n+1} - u_0^{n+1}) + \Delta t \frac{F_0}{\rho c} \tag{58}$$

En réarrangeant:

$$(1 + r) \cdot u_0^{n+1} - r \cdot u_1^{n+1} = u_0^n + \Delta t \frac{F_0}{\rho c} \tag{59}$$

Note: C'est une approximation au premier ordre pour la condition de Neumann. Pour une précision supérieure, un schéma au second ordre utilisant des points fantômes pourrait être utilisé, mais le premier ordre est suffisant pour cette application et plus simple à implémenter.

Implémentation (1D)

Listing 4: Implémentation des conditions aux limites en 1D

```
1   bool HeatEquationSolver1D::step() {
2       // ... setup vectors a, b, c, d ...
3
4       // Neumann BC at i=0: du/dx = 0
5       a[0] = 0.0;
6       b[0] = 1.0 + r;
7       c[0] = -r;
8       d[0] = u_[0] + dt_ * F_[0] / (mat_.rho * mat_.c);
9
10      // Interior points
```

```
    for (int i = 1; i < n_ - 1; i++) {
        a[i] = -r;
        b[i] = 1.0 + 2.0 * r;
        c[i] = -r;
        d[i] = u_[i] + dt_ * F_[i] / (mat_.rho * mat_.c);
    }

    // Dirichlet BC at i=n-1: u = u0_kelvin_
    a[n_-1] = 0.0;
    b[n_-1] = 1.0;
    c[n_-1] = 0.0;
    d[n_-1] = u0_kelvin_;

    solve_tridiagonal(a, b, c, d, u_);
    t_ += dt_;
    return t_ < tmax_;
}
```

Fichier: ***heat_equation_solver.cpp****, lignes 147-176*

Implémentation (2D)

Pour le cas 2D, les conditions de Neumann en $x = 0$ et $y = 0$ sont implémentées par la technique du miroir:

Listing 5: Implémentation des conditions aux limites en 2D

```
// Apply boundary conditions
for (int j = 0; j < n_; j++) {
    // Neumann at x=0: u[-1,j] = u[1,j]
    u_new[idx(0,j)] = u_new[idx(1,j)];

    // Dirichlet at x=L: u[n-1,j] = u0
    u_new[idx(n_-1,j)] = u0_kelvin_;
}

for (int i = 0; i < n_; i++) {
    // Neumann at y=0: u[i,-1] = u[i,1]
    u_new[idx(i,0)] = u_new[idx(i,1)];

    // Dirichlet at y=L: u[i,n-1] = u0
    u_new[idx(i,n_-1)] = u0_kelvin_;
}
```

Fichier: ***heat_equation_solver.cpp****, lignes 258-276*

6.3 Problème 3 : Visibilité de la source de chaleur

Description du problème

La spécification définit la source de chaleur comme:

$$F = t_{max} \cdot f^2 \tag{60}$$

où $f = 80C = 80K$ et $t_{max} = 16s$.
Donc: $F = 16 \times 80^2 = 102\,400$ W/m^3
Cependant, le taux de chauffage réel est:

$$\frac{\Delta u}{\Delta t} = \frac{F}{\rho c} \tag{61}$$

Pour le fer: $\rho c = 7874 \times 440 = 3\,464\,560$ J/(m^3·K)
Taux de chauffage: $\frac{102\,400}{3\,464\,560} \approx 0.0296$ K/s
Sur $\Delta t = 0.016$ s: $\Delta u \approx 0.0005$ K par pas de temps!

Problème: Avec une température initiale $u_0 = 286.15$ K (13°C), un changement de 0.0005 K est **invisible** - moins de 0.0002% de variation. Les utilisateurs ne voient aucune propagation de chaleur!

Solution : Amplification non physique pour la visualisation

Note importante: Le modèle physique spécifié dans le projet utilise $F = t_{max} \cdot f^2$, qui produit un chauffage extrêmement lent et à peine visible. Pour permettre une **visualisation interactive en temps réel** tout en préservant la précision du solveur, nous avons fait un **choix pragmatique**:

Nous avons ajouté un **facteur d'échelle de 100×** pour amplifier la source de chaleur **uniquement à des fins de démonstration**. Cette modification **ne reflète pas le modèle physique exact** spécifié, mais crée plutôt un scénario plus dramatique qui montre clairement la propagation de la chaleur.

Listing 6: Amplification de la source pour la visualisation

```
void HeatEquationSolver1D::init_source(double f) {
    double f1 = tmax_ * f * f;         // Physical value = 102,400 W/m
    double f2 = 0.75 * tmax_ * f * f; // Physical value = 76,800 W/m
    double scale = 100.0;               // Non-physical amplification for visibility

    for (int i = 0; i < n_; i++) {
        double x = i * dx_;
        if (x >= L_ / 10.0 && x <= 2.0 * L_ / 10.0) {
            F_[i] = f1 * scale;  // Visualization: 10,240,000 W/m
        } else if (x >= 5.0 * L_ / 10.0 && x <= 6.0 * L_ / 10.0) {
            F_[i] = f2 * scale;  // Visualization: 7,680,000 W/m
        } else {
            F_[i] = 0.0;
        }
    }
}
```

Fichiers: `heat_equation_solver.cpp`, lignes 73-91 (1D), lignes 197-216 (2D)

Résultat: Maintenant le taux de chauffage est de 2.96 K/s pour le fer, donnant **0.047 K par pas de temps** - facilement visible!

Approche alternative: Au lieu de mettre à l'échelle la source, on pourrait visualiser la différence de température $\Delta u = u - u_0$ avec un ajustement automatique de l'échelle. Cela préserverait la précision physique tout en obtenant la visibilité. Cependant, pour un outil de démonstration/enseignement, la source amplifiée crée des visualisations plus dramatiques et pédagogiquement utiles.

6.4 Problème 4 : Plage de diffusivité thermique multi-matériaux

Description du problème

Les quatre matériaux ont des diffusivités thermiques très différentes:

Matériau	α (m²/s)	Ratio par rapport au Cuivre
Cuivre	1.14×10^{-4}	1×
Fer	2.31×10^{-5}	0.2×
Verre	5.64×10^{-7}	0.005×
Polystyrène	8.01×10^{-8}	0.0007×

Table 3: Diffusivités thermiques des matériaux

Différence de facteur: Le cuivre diffuse la chaleur **1,423× plus rapidement** que le polystyrène!

Défi: Dans la même période de temps:

- Le cuivre montre une propagation rapide de la chaleur sur tout le domaine
- Le polystyrène ne montre presque aucune propagation (la chaleur reste localisée)

Cela rend la comparaison difficile - ils nécessitent des échelles de visualisation et des échelles de temps différentes.

Solution : Mode grille 2×2 avec mise à l'échelle indépendante

Nous avons implémenté une **architecture multi-solveur simultanée**:

Listing 7: Architecture multi-solveur avec mise à l'échelle indépendante

```
class SDLApp {
private:
    // Grid mode: 4 independent solvers
    std::unique_ptr<ensiie::HeatEquationSolver1D> solvers_1d_[4];
    std::unique_ptr<ensiie::HeatEquationSolver2D> solvers_2d_[4];
    ensiie::Material materials_[4];

    void render_grid() {
        for (int idx = 0; idx < 4; idx++) {
            int row = idx / 2;
            int col = idx % 2;
            int cell_x = col * (win_w / 2);
            int cell_y = row * (win_h / 2);

            // Each cell has independent temperature scaling
            if (sim_type_ == SimType::BAR_1D) {
                auto temps = solvers_1d_[idx]->get_temperature();
                heatmap_->auto_range(temps);  // Auto-scale per material
                heatmap_->draw_1d_cell(temps, info, cell_x, cell_y, win_w/2, win_h/2)
                    ;
            } else {
                auto temps = solvers_2d_[idx]->get_temperature_2d();
                heatmap_->auto_range_2d(temps);  // Auto-scale per material
                heatmap_->draw_2d_cell(temps, info, cell_x, cell_y, win_w/2, win_h/2)
                    ;
            }
        }
    }
};
```

Fichiers: ***sdl_app.cpp****, lignes 135-157;* ***sdl_app.hpp****, lignes 46-51*

Cette approche permet de visualiser simultanément les quatre matériaux avec des échelles de couleur adaptées individuellement, rendant les phénomènes de diffusion thermique visibles pour tous les matériaux malgré leurs propriétés très différentes.

7 Conclusion

Après avoir travaillé sur ce projet, j'ai découvert la profondeur et la complexité du C++. La maîtrise ne se limite pas à la syntaxe, il est essentiel de comprendre les principes et les mécanismes sous-jacents. Notre compréhension du C++ et de ses capacités a considérablement augmenté à la suite de cette expérience, ce qui a également augmenté notre appréciation du langage. Nous avons amélioré nos compétences en codage en travaillant sur des applications réelles et développé une approche stratégique de la résolution de problèmes qui sera inestimable dans nos futures entreprises. Ce projet a été crucial pour montrer la valeur pratique de la théorie appliquée au développement logiciel.

References

[1] Wikipedia contributors. *Finite difference method*. Accessed: 2025-01-03. 2024. URL: https://en.wikipedia.org/wiki/Finite_difference_method.

[2] Wikipedia contributors. *Gauss–Seidel method*. Accessed: 2025-01-03. 2024. URL: https://en.wikipedia.org/wiki/Gauss%5C%E2%5C%80%5C%93Seidel_method.

[3] Wikipedia contributors. *Tridiagonal matrix algorithm*. Accessed: 2025-01-03. 2024. URL: https://en.wikipedia.org/wiki/Tridiagonal_matrix_algorithm.