AMRHT2D – Mesh generation with adaptive refinement using hash tables

Priscila Cardoso Calegari and Álvaro Junio Pereira Franco

September 8, 2021

In this work we have presented a tool to generate meshes with adaptive and dynamic refinement. A hash table is used to manage the mesh elements and for searching, insertion, deletion, refine, and coarsening. The numerical examples was performed in two-dimensional meshes with many different refinement levels.

Contents

Chapter 1

Introduction

1.1 Installation

In order to access the AMRHT2D repository, first go to *https://github.com/pccalegari/AMRHT2D* and download the code. After this, install some auxiliary libraries, see *doc/configure_szip_hdf5_silo.txt*.

Chapter 2

Adaptive mesh refinement generation

Chapter 3

Hash Table data structure

Chapter 4

Numerical Examples

4.1 First example

In the first numerical example, we have generated a static refined mesh. Here, the mesh is two-dimensional domain $[a_1, b_1] \times [a_2, b_2]$, with $a_1 = a_2 = 0$ and $b_1 = b_2 = 1$. The initial condition is given by,

$$\phi(x, y) = \tanh[\alpha(r - d)], \tag{4.1}$$

where $\alpha = 75$, $r = 0.25$ the circumference radius, $d = \sqrt{(x - x_c)^2 + (y - y_c)^2}$ the distance between each point (x, y) of domain, and the circumference center $(x_c, y_c) = (0.5, 0.5)$. Figure 4.1 presents the function ϕ and the adaptive mesh with five refinement levels.

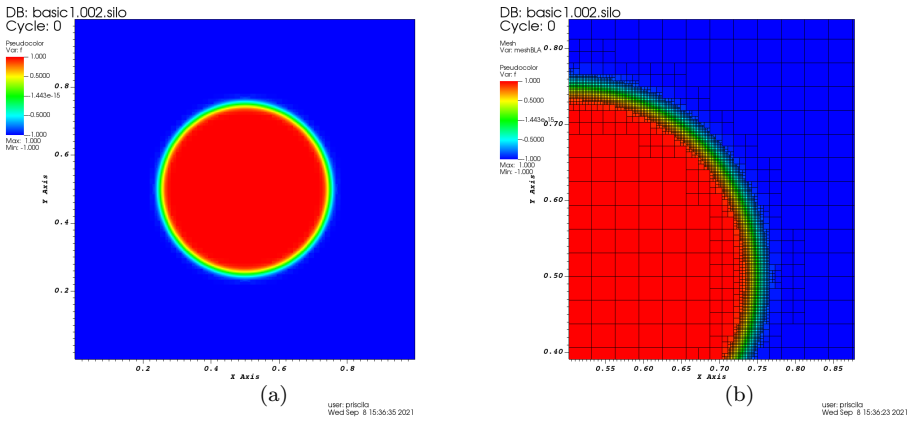

Figure 4.1: Function $\phi(x, y)$ evaluated in adaptive mesh with five refinement levels and base level with 32×32 cells.

We have generated meshes with base level 32×32 cells and some different refinement levels. The refinement criterium was the same for all cases. The refined cells were those who satisfied $-0.999 + 0.05 \cdot l < \phi(x, y) < 0.999 - 0.05 \cdot l$, with $l = 0, \ldots L - 1$ and L the number of levels.

To run this test, use Makefile: *make test1* and *make runtest1*. This example can be accessed in the file *test/test.cpp*.

Table 4.1 presents information of the hash table, L level number of mesh, N cell number of mesh, P number of positions in hash table, $\alpha = N/P$ appoximated load factor, C (MD) number of collisions with Division's method, C (MF) number of collisions with Fibonacci's method, $CMAX$ maximum number of cells per table position for each method, and lower limit for the height of any binary tree that stores N cells ($\lfloor \log_2(N) \rfloor$).

Table 4.1: Hash table information.

L	N	P	α	C (MD)	$CMAX$	C (MF)	$CMAX$	$\lfloor \log_2(N) \rfloor$
3	2.488	1.638	1,52	1.691	7	1.081	5	11
4	5.500	6.553	0,84	1.717	4	1.499	4	12
5	15.772	26.214	0,60	6.882	5	3.763	5	13
6	51.808	104.857	0,49	16.928	4	13.022	4	15
7	179.512	419.430	0,43	63.238	5	33.751	5	17
8	635.632	1.677.721	0.38	132.876	4	84.765	4	19

The number of positions in the hash table is nearly 10% of the cell number of a uniform mesh equivalent to finest level. For example, in the experiment with three refinement levels ($L3$) and base level with 32×32 cells, the cell number of a mesh equivalent to finest level is $128 \times 128 = 16.384$ cells (see first column and column P in table 4.1).

4.2 Second example

The purpose of this experiment is to show the dynamics of the adaptive mesh. For this, we have included the transport of lagrangian particles, modeled by the ODE system,

$$\begin{aligned} \frac{d}{dt} x_p(t) &= u(t, x_p, y_p), \\ \frac{d}{dt} y_p(t) &= v(t, x_p, y_p), \end{aligned} \tag{4.2}$$

where (x_p, y_p) is the particle position in the computational domain. The velocity field is analytically defined by,

$$\begin{aligned} u(t,x,y) &= \frac{(y-y_c)vt(d,t)}{d}, \\ v(t,x,y) &= -\frac{(x-x_c)vt(d,t)}{d}, \end{aligned} \tag{4.3}$$

where $vt(d,t) = (A/2\pi d)(1-e^{-d^2/4\mu t})$, with $A = 500$, $\mu = 10$, $(x_c, y_c) = (0,0)$, and $d = \sqrt{(x-x_c)^2 + (y-y_c)^2}$. The computational domain is $[a_1, b_1] \times [a_2, b_2]$ with $a_1 = a_2 = -0.5$ and $b_1 = b_2 = 0.5$, and initially 1.000 particles are distributed along the axis x (with a slight perturbation, see Figure 4.2). The Runge-Kutta's method of fourth order was used to obtain the numerical solution of the ODE system (4.2). The refinement criterium to generate the dynamic mesh was the particle position. The base level has 16×16 cells and four refinement levels.

To run this test, use Makefile: *make test2* and *make runtest2*. This example can be accessed in the file *test/test_particles.cpp*.

Figure 4.2 presents the numerical solution in different time steps.

We have used VisIt to visualize and analyze the results of simulations, see *https://visit-dav.github.io/visit-website/*.

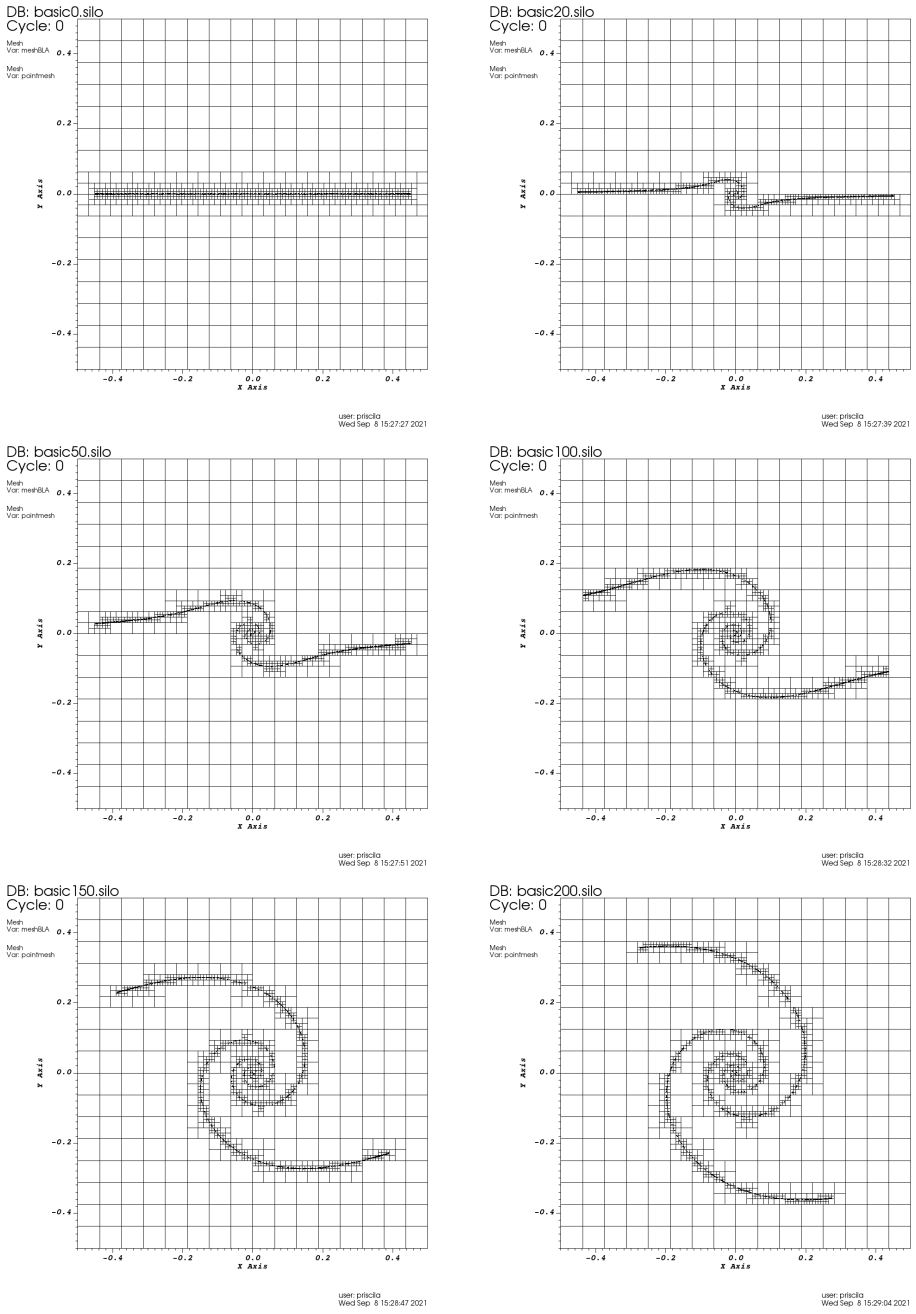

Figure 4.2: Numerical simulation in some time steps.