Aims of the project

Aims of the project

Aim 1

Construct **highly adaptive Cartesian non-uniform meshes** on which we can perform **numerical quadrature** and eventually implement **numerical solvers for PDEs**, mainly based on the Finite Volume method.

Aims of the project

Aim 1

Construct **highly adaptive Cartesian non-uniform meshes** on which we can perform **numerical quadrature** and eventually implement **numerical solvers for PDEs**, mainly based on the Finite Volume method.

Aim 2

Implement a simple method to **compress digital images**, which recognizes large areas of almost uniform color. Could be eventually used also for shape recognition.

Aims of the project

Aim 1

Construct **highly adaptive Cartesian non-uniform meshes** on which we can perform **numerical quadrature** and eventually implement **numerical solvers for PDEs**, mainly based on the Finite Volume method.

Aim 2

Implement a simple method to **compress digital images**, which recognizes large areas of almost uniform color. Could be eventually used also for shape recognition.

These two objectives may seems quite far one from the other but actually they can be linked by... **Quadtrees**.

What is a quadtree

What is a quadtree

Basically just an unbalances tree structure allowing **four children**. Generalized in 3D by the octrees.

What is a quadtree

Basically just an unbalances tree structure allowing **four children**. Generalized in 3D by the octrees.

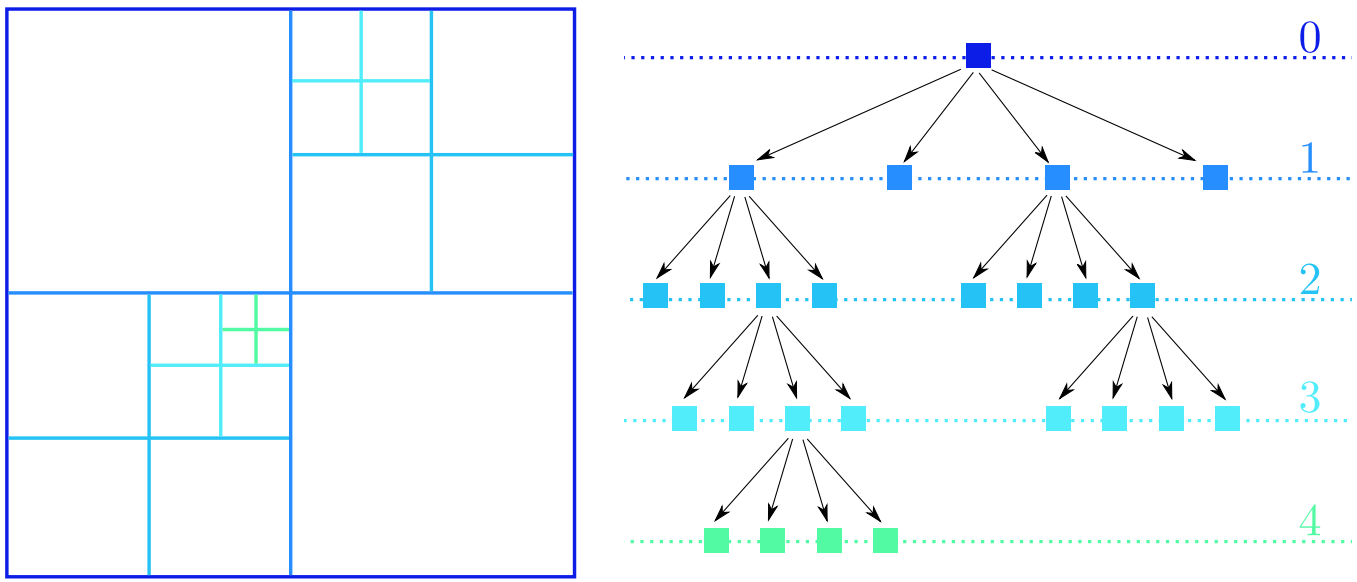


Image from: T. Bellotti, M. Theillard - A coupled level-set and reference map method for interface representation with applications to two-phase flows simulation - 2018 in press: J. Comput. Phys.

What is a quadtree

Basically just an unbalances tree structure allowing **four children**. Generalized in 3D by the octrees.

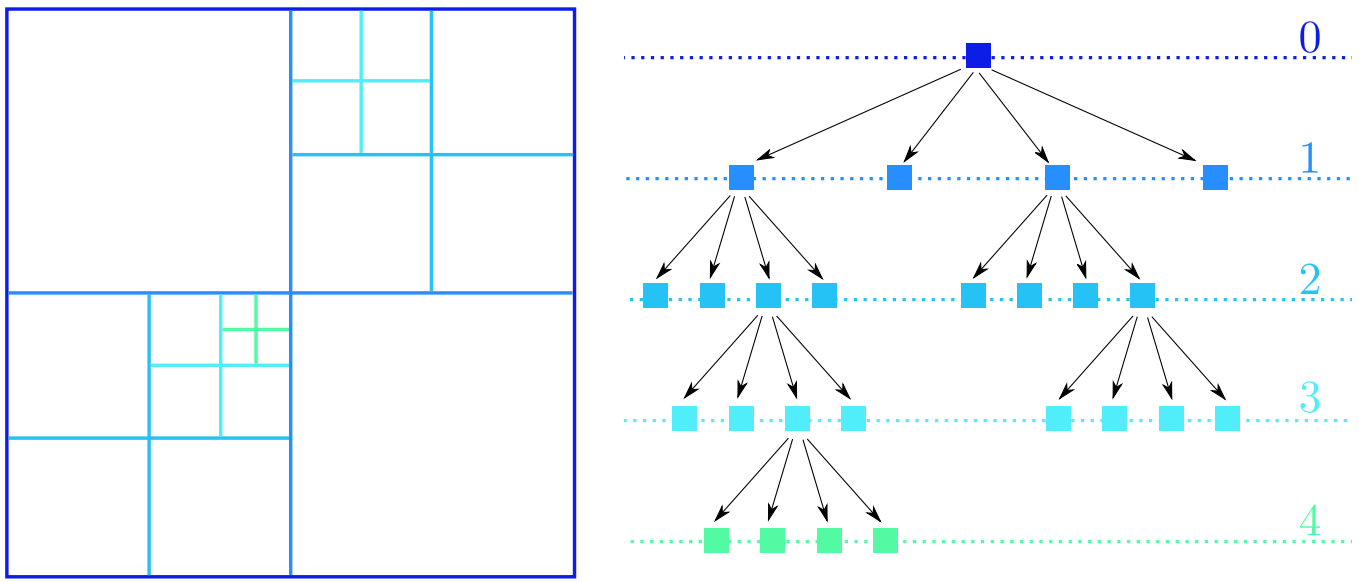


Image from: T. Bellotti, M. Theillard - A coupled level-set and reference map method for interface representation with applications to two-phase flows simulation - 2018 in press: J. Comput. Phys.

It has a clear geometrical counterpart.

Some terminology

- Leaf.
- Level
- Maximum level
- Minimum level.

Level-set theory

In this work, we only see it as a **strategy to build** beautiful and meaningful **adaptive meshes** (there is more than this).

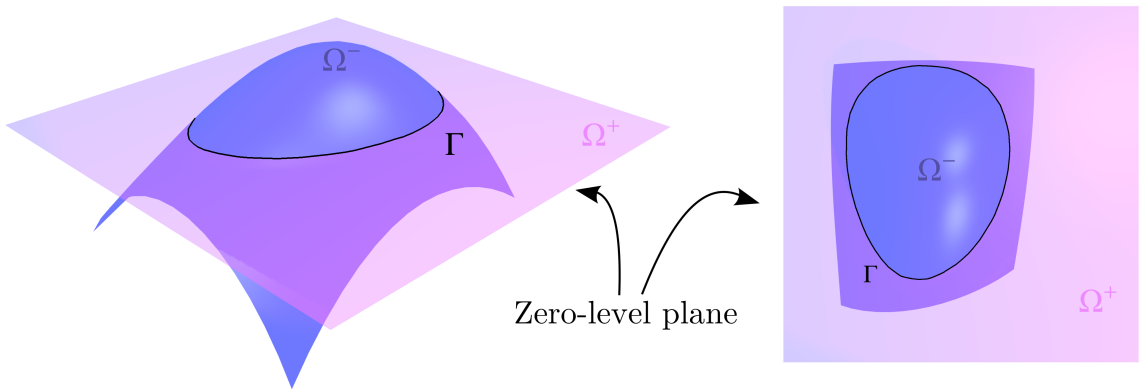


Image from: T. Bellotti, M. Theillard - A coupled level-set and reference map method for interface representation with applications to two-phase flows simulation - 2018 in press: J. Comput. Phys.

$$\Gamma = \{\mathbf{x} \in \Omega \ : \ \phi(\mathbf{x}) = 0\},$$
$$\Omega^- = \{\mathbf{x} \in \Omega \ : \ \phi(\mathbf{x}) < 0\},$$
$$\Omega^+ = \{\mathbf{x} \in \Omega \ : \ \phi(\mathbf{x}) > 0\}.$$

Level-set theory

It is probably the easiest way of **representing an interface** in a computationally efficient manner.

Level-set theory

It is probably the easiest way of **representing an interface** in a computationally efficient manner.

Example

In 2D, a circle centered in (x_0, y_0) with radius R can be represented by:

$$\phi(x, y) = \sqrt{(x - x_0)^2 + (y - y_0)^2} - R.$$

Level-set theory

It is probably the easiest way of **representing an interface** in a computationally efficient manner.

Example

In 2D, a circle centered in (x_0, y_0) with radius R can be represented by:

$$\phi(x, y) = \sqrt{(x - x_0)^2 + (y - y_0)^2} - R.$$

Notice that, given Γ, the level-set is not unique, but this is not our problem now. It is unique if we assume (and we will do so) that $\phi(x, y)$ is nothing but the **signed distance** of (x, y) from Γ.

$$\text{split } \mathcal{C} \text{ if } : \quad |\phi(\mathbf{x}_\mathcal{C})| \leq \text{Lip}(\phi) \cdot \text{diag}(\mathcal{C}) \qquad \text{and} \qquad \text{level}(\mathcal{C}) \leq \max_{\text{level}},$$

This naively means that we split a cell wheather its diagonal length exceeds the distance from the interface Γ represented by the level-set ϕ.

Gaussian quadrature

Considering a stardard quadrilateral domain $\Omega = [-1, 1]^2$ and given a possibly smooth function $f : \Omega \to \mathbb{R}$, we want to compute:

$$\int_\Omega f(x, y)\, dxdy.$$

This can be done by quadrature formulae, based on simple evaluation of the function f at certain points of the domain. For our purpose, we use two choices:

1. **"Naive" formula**, with only one function evaluation.

$$\int_\Omega f(x, y)\, dxdy \simeq |\Omega| f(0, 0) = 4f(0, 0)$$

2. **3rd order Gaussian formula**, with nine function evaluations (plus extra computations to adapt to a generic domain).

$$\begin{aligned}\int_\Omega f(x, y)\, dxdy \simeq \frac{5}{81}&\left[5f\left(-\sqrt{\frac{3}{5}}, -\sqrt{\frac{3}{5}}\right) + 8f\left(0, -\sqrt{\frac{3}{5}}\right) + 5f\left(\sqrt{\frac{3}{5}}, -\sqrt{\frac{3}{5}}\right)\right] \\ &+ \frac{8}{81}\left[5f\left(-\sqrt{\frac{3}{5}}, 0\right) + 8f\left(0, 0\right) + 5f\left(0, \sqrt{\frac{3}{5}}\right)\right] \\ &+ \frac{5}{81}\left[5f\left(-\sqrt{\frac{3}{5}}, \sqrt{\frac{3}{5}}\right) + 8f\left(0, \sqrt{\frac{3}{5}}\right) + 5f\left(\sqrt{\frac{3}{5}}, \sqrt{\frac{3}{5}}\right)\right].\end{aligned}$$

Rgb colors and their "distance"

It is probably the simplest way of representing colors, as a combination of three components, **red, green and blue**. Thus, a color C is nothing else than a triplet:

$$C = [C_{\text{red}}, C_{\text{green}}, C_{\text{blue}}],$$

where $C_{\text{red}}, C_{\text{green}}, C_{\text{blue}} \in \{0, \ldots, 255\}$. In this way, we can represent 16'777'216 different tones.
For our purpose, it is useful to define the notion of **distance** between two colors, which can be defined in many ways. In our case, we use the simple "corrected" formula:

$$d(C^1, C^2) = \sqrt{2\left(C^1_{\text{red}} - C^2_{\text{red}}\right)^2 + 4\left(C^1_{\text{green}} - C^2_{\text{green}}\right)^2 + 3\left(C^1_{\text{blue}} - C^2_{\text{blue}}\right)^2}.$$

The mean color between $\left\{C^1, \ldots, C^N\right\}$ is defined by:

$$\overline{C}\left(\left\{C^1, \ldots, C^N\right\}\right) = \left[\frac{1}{N}\sum_{n=1}^{N} C^n_{\text{red}}, \frac{1}{N}\sum_{n=1}^{N} C^n_{\text{green}}, \frac{1}{N}\sum_{n=1}^{N} C^n_{\text{blue}}\right],$$

and a sort of standard deviation as:

$$\sigma\left(\left\{C^1, \ldots, C^N\right\}\right) = \frac{1}{N}\sqrt{\sum_{n=1}^{N} d(C_n, \overline{C})^2} \in [0, 765].$$

How to compress an image?

Let $\mathcal{P}$ be a preleave whose children are $\mathcal{L}(\mathcal{P})$. We define a tolerance $0 < \epsilon \ll 1$. Then:

$$\text{merge} \quad \mathcal{P} \quad \text{if} \ : \ \sigma\left(\mathcal{L}(\mathcal{P})\right) \leq 765 \cdot \epsilon.$$

In the case of merging, we consider:

$$C_{\mathcal{P}} = \overline{C}\left(\mathcal{L}(\mathcal{P})\right).$$

The Cell class

	Cell<T>
#	const Point<T> base_point
#	const T dx
#	const T dx
#	const unsigned char level
#	std::shared_ptr<Cell> l_l
#	std::shared_ptr<Cell> l_r
#	std::shared_ptr<Cell> u_l
#	std::shared_ptr<Cell> u_r
+	Cell(Point<T>, T, T, unsigned char)
+	virtual ~Cell()
+	Point<T> getBasePoint() const
+	unsigned char getLevel() const
+	bool isLeaf() const
+	Point<T> getCenter() const
+	T getDx() const
+	T getDy() const
+	T getDiagonal() const
+	T cellSurface() const
+	virtual void splitCell()
+	virtual void mergeCell()
+	void refineCell(const RefinementCriterion<T> & , const unsigned char)
+	void simplifyCell(const RefinementCriterion<T> &, const unsigned char)
+	void updateCell(const RefinementCriterion<T> &, const unsigned char, const unsigned char)
+	std::vector<Point<T>> getVertices() const
+	void getLeaves(std::vector<std::shared_ptr<Cell<T>>> &)
+	void getPreLeaves(std::vector<std::shared_ptr<Cell<T>>> &) const
+	std::string tikzDot() const
+	std::string tikzSquare(const RGBColor color, const bool) const
+	T zeroOrderIntegration(const std::function<T(Point<T>)> &) const
+	T thirdOrderGaussianIntegration(const std::function<T(Point<T>)> &) const

The Pixel class inheriting from Cell

The “generalized” pixel is basically a Cell...

	Pixel<T> : public Cell<T>
#	RGBColor field
+	Pixel(Point<T>, T, T, unsigned char)
+	Pixel(Point<T>, T, T, unsigned char, const RGBColor &)
+	virtual ~Pixel()
+	void setField(const RGBColor &)
+	RGBColor getField() const
+	virtual void splitCell() override
+	RGBColor meanField()
+	double stdDevField()
+	virtual void mergeCell() override

plus a **color**. We have to be careful to **cast pointers** when we need to extract information from the Pixel.

The QuadTree class

This class is mostly a **wrapper** of the Cell class.

	QuadTree <T>
#	const T x_size
#	const T y_size
#	unsigned char min_level
#	unsigned char max_level
#	std::shared_ptr<Cell<T>> parent_cell
#	std::vector<std::shared_ptr<Cell<T>>> getLeaves() const
+	QuadTree(Point<T>, T, T, unsigned char, unsigned char)
+	virtual ~QuadTree()
+	T simpleIntegration(std::function<T(Point<T>)> &) const
+	unsigned getMinLevel() const
+	unsigned getMaxLevel() const
+	size_t numberOfLeaves() const
+	void buildUniform()
+	void buildUniform(unsigned)
+	void clear()
+	void updateWithLevelSet(const LipschitzFunction<T> &)
+	void updateQuadTree(const RefinementCriterion<T> &)
+	void updateQuadTree(const RefinementCriterion<T> &, const unsigned char, const unsigned char)
+	std::vector<Point<T>> getCenters()
+	void exportCentersTikz(const std::string &) const
+	void exportMeshTikz(const std::string &, bool) const
+	T simpleIntegration(const std::function<T(Point<T>)> &) const
+	T thirdOrderGaussianIntegration(const std::function<T(Point<T>)> &) const
+	T simpleIntegration(const std::function<T(std::shared_ptr<Cell<T>>)> &) const

The Image class

This class is mostly a **wrapper** of the Pixel class. We distinguished it from the QuadTree class (it does not inherit from it) because there are many features that they do not share.

	Image<T>
#	T x_size
#	T y_size
#	unsigned char min_level
#	unsigned char max_level
#	std::shared_ptr<Pixel<T>> parent_cell
+	Image()
+	Image(T, T, unsigned char, unsigned char)
+	~Image()
+	unsigned int getMinLevel() const
+	unsigned int getMaxLevel() const
+	size_t numberOfPixels() const
+	void clear()
+	void simplifyImage(double)
+	void buildUniform(unsigned char)
+	void createFromFile(std::string)
+	void saveImage(const std::string & filename) const

The way we update the mesh: the RefinementCriterion class

This is a very simple **abstract class** with an operator telling us if we have to split a Cell or not.

	RefinementCriterion <T>
+	RefinementCriterion()
+	virtual ~RefinementCriterion()
+	virtual bool operator()(std::shared_ptr<Cell<T>>) const = 0

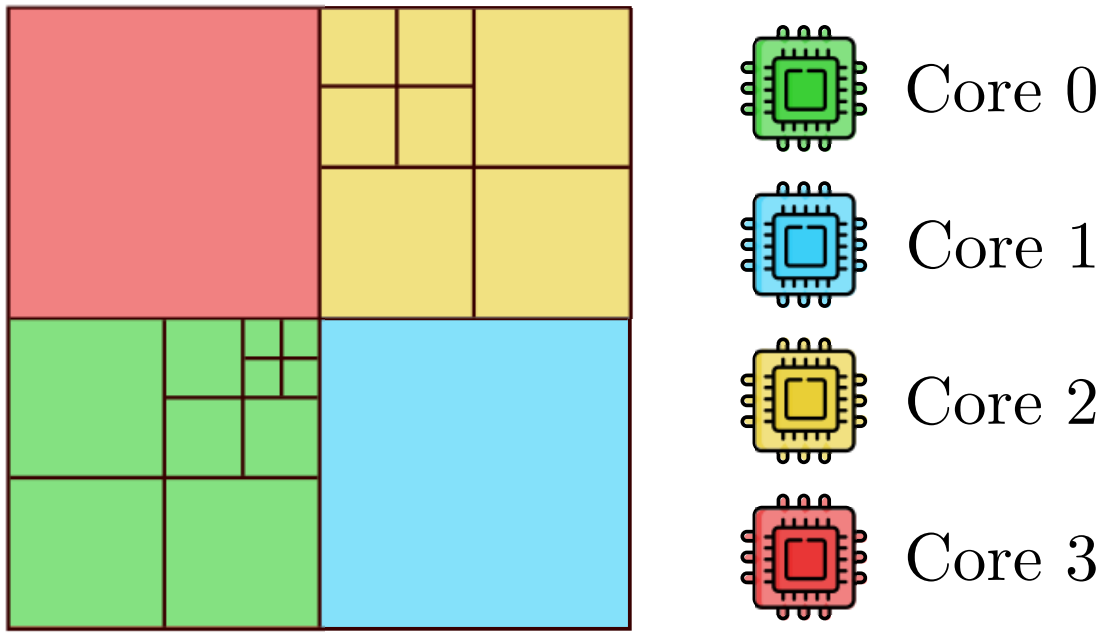
Many important criteria inherit from it:

	RefineAlwaysCriterion<T> : public RefinementCriterion<T>
+	RefineAlwaysCriterion()
+	virtual ~RefineAlwaysCriterion()
+	virtual bool operator()(std::shared_ptr<Cell<T>>) const override

	LevelSetCriterion<T> : public RefinementCriterion<T>
-	const LipschitzFunction<T> & level_set
+	LevelSetCriterion(const LipschitzFunction<T> &)
+	virtual ~LevelSetCriterion()
+	virtual bool operator()(std::shared_ptr<Cell<T>>) const override

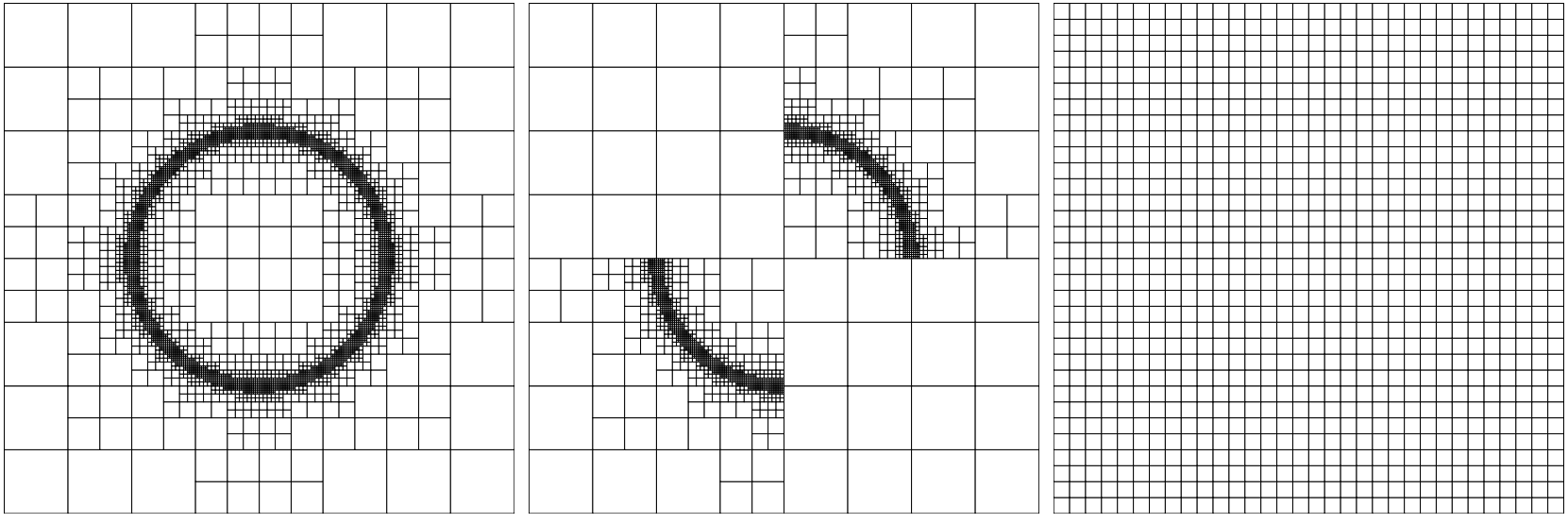
	CriterionVariance<T> : public RefinementCriterion<T>
-	double thr
+	CriterionVariance()
+	virtual ~CriterionVariance()
+	virtual bool operator()(std::shared_ptr<Cell<T>>) const override

The key idea is to have a number of core which is a power of 4 and have a minimum level large enough, so that we can avoid communications between cores and each of them has a local tree (no shared memory).



And each subtree (core) integrates independently. At the very end, the sub-integrals are summed with a **reduce** procedure.

$$\Omega = [-2, 2]^2 \quad \phi(x, y) = \sqrt{x^2 + y^2} - 1 \quad f(x, y) = \mathbb{I}_{\{\phi(x,y) \leq 0\}} \quad \int_\Omega f(x, y)\, dxdy = \pi.$$

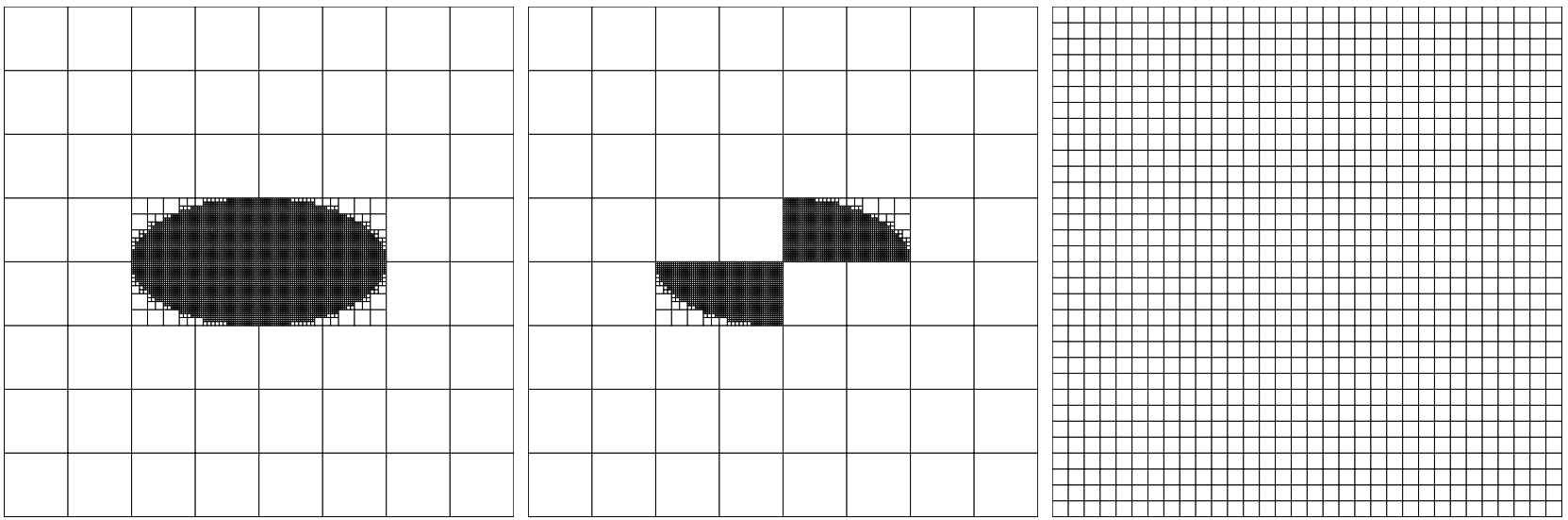


Naive	Mesh 1		Mesh 2		Mesh 3	
# of cores	Time [s]	Speedup	Time[s]	Speedup	Time [s]	Speedup
1	10.2	-	76.3	-	83.9	-
4	10.2	**54.4**	76.3	**45.2**	83.9	**54.5**
16	10.2	**87.0**	76.3	**43.2**	83.9	**0.34**

3rd Gaussian	Mesh 1		Mesh 2		Mesh 3	
# of cores	Time [s]	Speedup	Time[s]	Speedup	Time [s]	Speedup
1	10.2	-	76.3	-	83.9	-
4	10.2	**54.4**	76.3	**45.2**	83.9	**54.5**
16	10.2	**87.0**	76.3	**43.2**	83.9	**0.34**

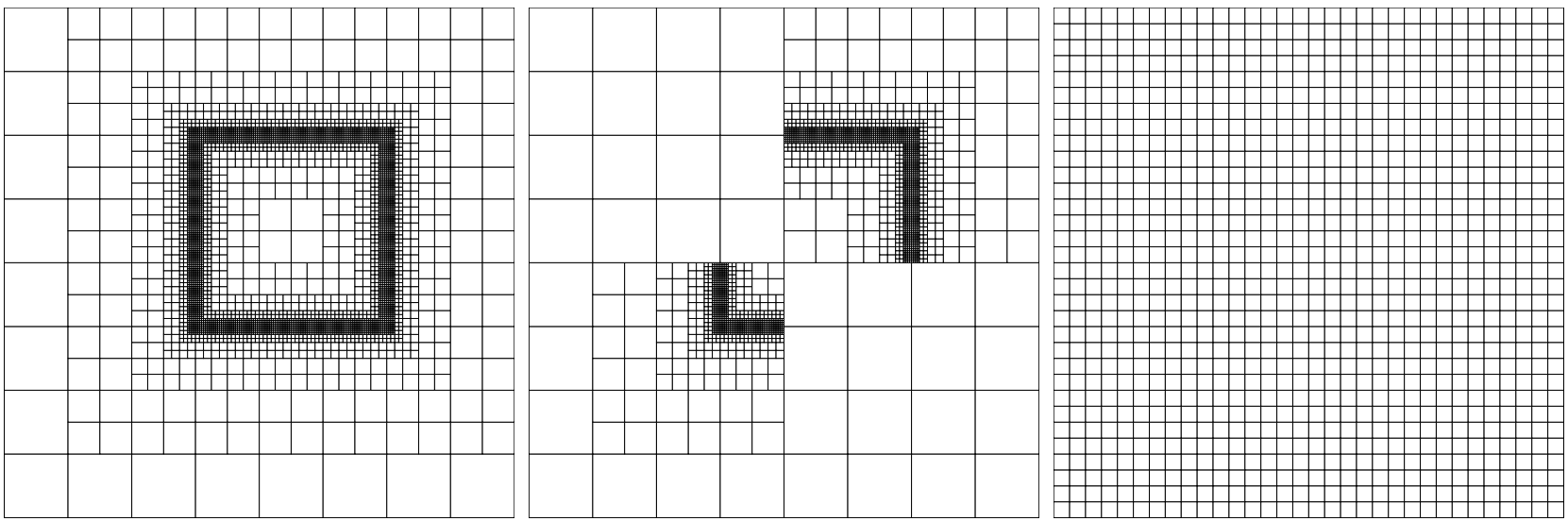
$$\Omega = [-2, 2] \quad f(x, y) = \frac{1}{2\pi\sigma_x\sigma_y} e^{-\frac{1}{2}\left(\frac{x^2}{\sigma_x^2} + \frac{y^2}{\sigma_y^2}\right)} \quad \sigma_x = 0.1 \quad \sigma_y = 0.05 \quad \int_\Omega f(x, y)\, dxdy \simeq 1.$$

We refine in the ellipse within 10 standard deviations.



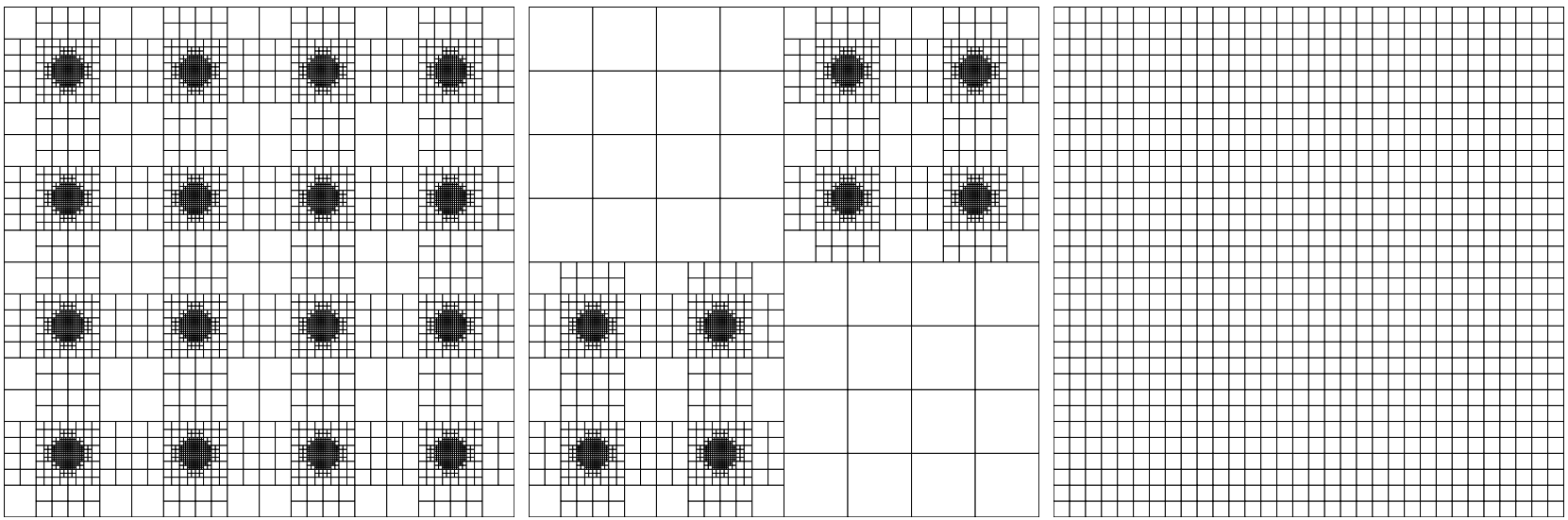
$$\Omega = [-2, 2]^2 \quad \phi(x, y) = \max\{|x - 0.25| - 0.75, |y - 0.25| - 0.75\}$$

$$f(x, y) = (x^2 + y^2)\left[\cos(\pi x) + \sin(\pi y)\right] \mathbb{I}_{\{\phi(x,y) \leq 0\}} \quad \int_\Omega f(x, y)\, dxdy = \frac{3(-16 - 12\pi + 7\pi^2)}{8\pi^3} \simeq 0.186109$$



$$\Omega = [0,8]^2 \quad \phi(x,y) = \min_{i,j=0,\dots,3} \left(\sqrt{(x-(2i+1))^2 + (y-(2j+1))^2} - 0.15 \right) \quad f(x,y) = \mathbb{I}_{\{\phi(x,y) \leq 0\}}$$

$$\int_\Omega \psi(x,y)\,dxdy = \frac{9\pi}{25}$$



Three different ϵ: $\epsilon = 0.012, 0.024, 0.048$.

Original figures size: 262144 px.



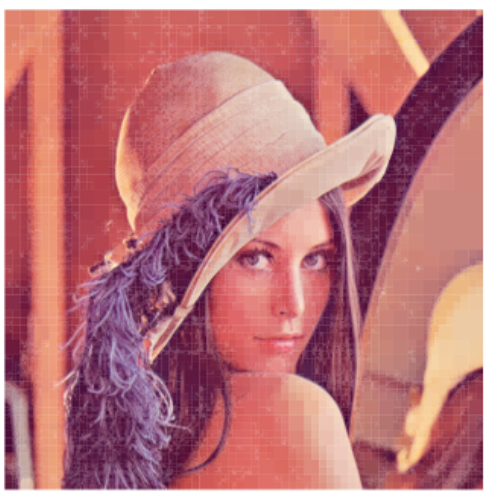
131509 px - comp. ratio = 1.99 84808 px - comp. ratio = 3.09 38152 px - comp. ratio = 6.87

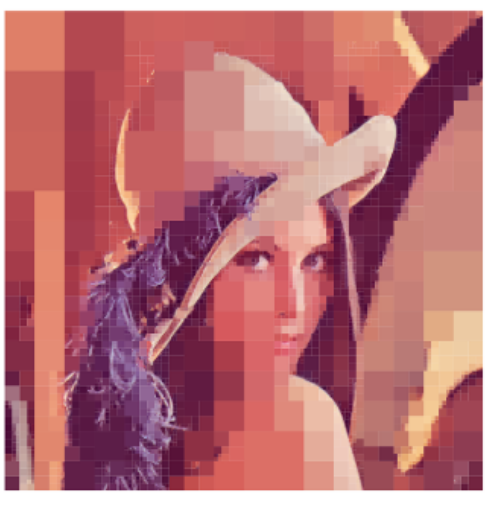


115438 px - comp. ratio = 2.27 69259 px - comp. ratio = 3.78 27394 px - comp. ratio = 9.57



32188 px - comp. ratio = 8.14 16405 px - comp. ratio = 15.98 6571 px - comp. ratio = 39.89





75172 px - comp. ratio = 3.49 29890 px - comp. ratio = 8.77 8902 px - comp. ratio = 29.45



171517 px - comp. ratio = 1.53 85258 px - comp. ratio = 3.07 24466 px - comp. ratio = 10.71