Università degli Studi di Pisa

Facoltà di Scienze Matematiche Fisiche e Naturali
Corso di Laurea in Scienze dell'Informazione

Tesi di Laurea

Computational Geometry techniques for approximation of electrostatic forces

Candidato
Paolo Bientinesi

Relatore
Marco Pellegrini

Controrelatore
Milvio Capovani

Anno Accademico 1996-1997

Ringrazio particolarmente il prof. Mauro Leoncini per i consigli e per le minuziose correzioni alle stesure preliminari.

Un ringraziamento sincero agli amici che non si sono tirati indietro di fronte ai miei quesiti : Batman, Buffalmacco, Cippa, Raffa. Uno 'special thanks' allo Zio che oltretutto ha rischiato la lettura di quel che segue.

Un grazie ai miei genitori per la fornitura hardware e software senza la quale mi sarei intrattenuto su queste pagine per 24 mesi (anziché 17). So che non capiranno ció per cui li sto ringraziando.

Nei primi anni di Universitá fui colto dalla malattia del numero. I responsabili furono il prof. Milvio Capovani, la prof.ssa Menchi ed il prof. Romani (che mi ha anche aiutato nel Capitolo 4): grazie 1000.

P.S.: il Galateo vieta di ringraziare il relatore.

Contents

Have you heard the news?
Sort out this confusion
for people must be jumping
to the wrong conclusion
that one and one makes five.

Neil Tennant

Chapter 1

Introduction

Mathematics permeates every scientific and technical discipline. Mathematical models approximate natural phenomena, helping us in simulating and foreseeing systems' developments without having to make physical experiments. Mathematical issues, though abstract, can be useful in describing and solving real life problems and conversely nature always stimulates the growth of new mathematical ideas. Nowadays mathematics takes part in every day life even if we may not be aware of this.

Following the blueprint of [BBCM92], mathematics' approach to real life problems could be expressed by this scheme: mathematical modelization, analysis of the model, determination of solving methods and implementation. The first phase consists in describing the considered phenomenon with a system of equations; in the second phase the mathematical model is analysed in order to get qualitative properties for the solution of the system such as existence, uniqueness, stability, regularity; methods for solving the system are derived and compared in the third phase and in the last phase these methods are implemented through suitable programming languages. This scheme must not be seen as a strict sequential rule to process problems, in fact it is possible that both results taken from the elaboration and each single phase suggest modifications to be applied to some of the earlier phases. The aim is either to obtain a more faithful description of the evolution of the phenomenon, or to avoid bad conditioned problems, whose drawbacks are amplified by the computer representation of numbers or to search for a more stable algorithm.

In this paradigm it's easy to associate physical issues with the first two steps, mathematical issues with the second and third step, and computer science issues with the last two steps. In particular, in computer science the two subareas affected are the algorithmic and the numerical analysis, since in general solutions to the problems are either not explicitly expressible through elementary functions, or difficult to handle because of their (algebraic) complexity. Numerical methods are then needed every time a qualitative analysis of solution is not sufficient for the intended application.

This thesis tackles the problem of computing the electrostatic force acting between two charged bodies in the 3-dimensional space, B_1 and B_2, endowed with a known charge density. Physics introduces Coulomb's law solving the problem for two punctiform charges, then extends the solution to two bodies thanks to the integral operation. Mathematics guarantees the existence of the integral, and since close form solutions for this kind of integrals are possible only in special cases, computer science together with mathematics provides methods to approximate them. This classical model for the problem leads to integrals whose kernel function presents a singularity when the distance between the bodies is zero (that is, the bodies touch each other). This fact, besides colliding with our intuition, makes numerical approximation techniques less effective, especially when the poles reside within the integration domain; i.e., the bodies have common boundary points.

Pellegrini [Pel96, Pel97], analysed the problem through a different mathematical model, exploiting an integral geometry approach. Instead of starting from the expression of the force acting between two points p_1 and p_2 and integrating then over the bodies, he expresses the force that B_1 and B_2 exert on each other along lines in 3-space, reducing the problem to an integral over a set of lines. This leads to different integrals whose kernel is free from singularities.

1.1 Overview of the thesis

In this thesis we develop, and implement several algorithms for computing electrostatic forces based on the integral geometric approach proposed in [Pel96, Pel97]. The accuracy of the results is validated against reference values ob-

tained independently through a mixed analytic and numeric approach. For the force between two polyhedra in 3-space (tetrahedra) the best performance in terms of accuracy vs. running time and reliability, among the algorithms considered, is attained by a Quasi Monte Carlo algorithm. For the force between two polygons in 3-space (triangles) a method based on Gaussian Adaptive integration scheme attains the best performance, among the algorithms considered, in terms of accuracy vs. running time and reliability.

Chapter 2 is an introduction to the problem from a mathematical point of view. In Section 2.1, we formulate the problem of computing the electrostatic force exerted by two charged convex bodies according to the classical electrostatic theory: we show Coulomb's law and how integrals are derived from it. In Section 2.2, referring to the articles [Pel96, Pel97], we define the electrostatic field generated by a charged body according to an integral geometry approach. Thanks to this formula we derive the integrals for the electrostatic force when the input bodies are two tetrahedra and two triangles in the 3-space. Since in general it is not possible to analytically find the primitive of such integrals, in Section 2.3 we describe three general methods for numerical integration: Monte Carlo, Quasi Monte Carlo and Gaussian quadrature. For this we rely on [DR84, HH64, Nie92] and [BBCM92]. Then in Subsection 2.3.4, we give an high level presentation of an adaptive algorithm (Gauss Adaptive) which takes advantage of the properties of the kernel functions. Section 2.4 contains a brief survey about other techniques used to solve the integrals derived from classical theory.

Chapter 3 is dedicated to the automatic computation of approximations of the electrostatic force through computer implementation. The integrals derived from the new interpretation have kernel functions exactly computable. In Sections 3.2 and 3.3, we show efficient methods based on computational geometry which allow the calculations to be done in constant time for a single kernel evaluation ([O'R94], [PS85]). Moreover we always try to avoid dangerous operation from the point of view of the error propagation. In Section 3.1.1 we present several algorithms to generate sampling nodes within the integration domains and a technique (Section 3.4) which make it possible to filter

out nodes without actual contribute to the approximations of the integrals. In Subsection 3.5 we describe in detail the implementation of the Gauss-adaptive method, specifically designed to deal with this kind of integrals. Finally, in Section 3.6, we treat the topic of the representation of numbers with a finite precision and the consequent problems.

In Chapter 4 we present a selection of experiments for the case of the tetrahedra. We consider three sets of experiments, relatively to tetrahedra sharing one vertex (Subsection 4.4.1), one edge (Subsection 4.4.2) and one face (Subsection 4.4.3). After explaining how we computed the reference values (Section 4.3), in Sections 4.4 and 4.5, we show the results as tables and graphics, reporting the relative error against number of sampling points and relative error against computation time.

Equally Chapter 5 treats the experiments for the case of triangles as input bodies. We consider triangles sharing one edge and one vertex. In Section 5.3 we illustrate in detail how the reference values have been computed. Sections 5.4 and 5.5 include a selection of tables and plots.

In Chapter 6 we consider two boundary volume problems whose solution involves integrals of the same kind as those studied in Chapter 5.

In Appendix A we report in detail some of the algorithms used throughout the thesis, showing references for each of them.

We can't agree about anything
You got a different point of view
But I know what you're likely to say
that I'm going about it the wrong way
Neil Tennant

Chapter 2

The Problem

We want to compute the electrostatic force acting between two convex bodies in 3-space, B_1 and B_2, assuming they both are polyhedra endowed with uniform volume charge densities (dielectrics) or they are flat polygons endowed with uniform surface charge densities. In this chapter we show the classic technique to derive the formula for the force based on Coulomb's law and we introduce a second technique that will make possible to obviate the drawbacks of the first one. Moreover we describe several methods to compute the kind of integrals derived from these two interpretations. Throughout the thesis we will consider the instance of the general problem where the polyhedra are tetrahedra and polygons are triangles. These assumptions maintains the complete generality of the problem, since any polygon can be partitioned in triangles and any polyhedron can be subdivided in tetrahedra.

2.1 Electrostatic Force

Here we recall the classic electrostatic theory, entirely based upon Coulomb's law. Let q_1 and q_2 be two particles of charge respectively in position p_1 and p_2. The force $\vec{F}_{12}$ that q_1 exerts over q_2, is determined by Coulomb's law:

$$\vec{F}_{12} = \frac{1}{4\pi\epsilon_0} \frac{q_1 q_2}{|p_1 - p_2|^2} \frac{p_1 - p_2}{|p_1 - p_2|}.$$

Considering two convex bodies B_1 and B_2, endowed with volume charge densities $\rho_1(p)$ and $\rho_2(p)$ instead of two particles of charge, the force that B_1

exerts over B_2 is obtained integrating Coulomb's law over the points of the two bodies:

$$\vec{F}_{12} = \frac{1}{4\pi\epsilon_0} \int_{p_1 \in B_1} \int_{p_2 \in B_2} \frac{\rho_1(p_1)\,\rho_2(p_2)}{|p_1 - p_2|^2} \frac{p_1 - p_2}{|p_1 - p_2|}\, dp_1\, dp_2. \tag{2.1.1}$$

We will refer to such integrals as volume-to-volume integrals. The force acting between two flat convex polygons C_1 and C_2 endowed with surface charge densities $\sigma_1(p)$ and $\sigma_2(p)$ is:

$$\vec{F}_{12} = \frac{1}{4\pi\epsilon_0} \int_{p_1 \in C_1} \int_{p_2 \in C_2} \frac{\sigma_1(p_1)\,\sigma_2(p_2)}{|p_1 - p_2|^2} \frac{p_1 - p_2}{|p_1 - p_2|}\, dp_1\, dp_2, \tag{2.1.2}$$

and we will refer to these kind of integrals as surface-to-surface integrals. From here on we consider the instance of the general problem where functions $\rho_1(p)$, $\rho_2(p)$, $\sigma_1(p)$ and $\sigma_2(p)$, are constants, calling them ρ_1, ρ_2, σ_1 and σ_2, respectively; this allows to always take out those factor out of the integral operators. The x-component of the force $\vec{F}_{12}$, integral (2.1.1) becomes:

$$\frac{\rho_1\rho_2}{4\pi\epsilon_0} \iiint_{B_1} \iiint_{B_2} \frac{(x_1 - x_2)dx_1\, dy_1\, dz_1\, dx_2\, dy_2\, dz_2}{((x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2)^{3/2}} \tag{2.1.3}$$

where $p_1 = (x_1, y_1, z_1) \in B_1$ and $p_2 = (x_2, y_2, z_2) \in B_2$. On the other side, integrals (2.1.2) are quadruple integrals of the same form, being the z-coordinates constrained by the fact that C_1 and C_2 are flat polygons. The primitive of these integrals can be explicitly expressed only for very particular shapes, positions and symmetries of the bodies and often it requires complex symbolic calculations. In general numerical methods are needed. Difficulties arise because the kernel function diverges as the distance between the points p_1 and p_2 decreases to zero. This is the major drawback in computing the force that two bodies exert one another when they share a vertex, an edge or even a face, as the singularities of the kernel lie within the integration domain.

2.2 Integral geometry approach

Now we illustrate a new geometric interpretation of the electrostatic field $\vec{E}$, taken from [Pel96] which is free from singularities; let us call this field the

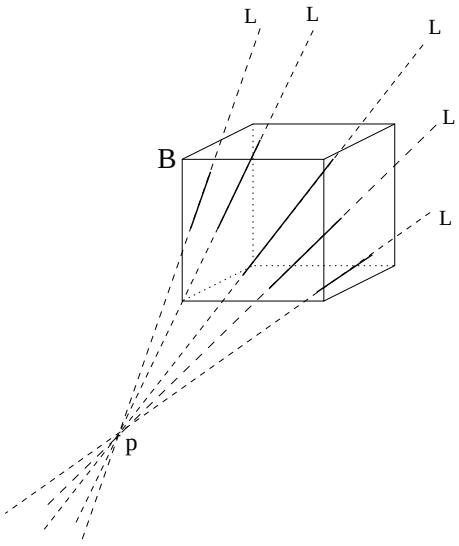

Figure 2.1: Electrostatic field in p is a vectorial integral over the lines L that meet body B. In case of uniform volume charge density of B, the contribution along each line is the measure of $L \cap B$.

Geometric Field $\vec{G}$. Then, exploiting $\vec{G}$, we will express the electrostatic force acting between two tetrahedra and between two triangles.

The basic idea behind the new definition is that electrostatic forces act along straight lines in a homogeneous medium; this allows $\vec{F}_{12}$ to be expressed as an integral over the set of directions in 3-space.

2.2.1 Geometric Field $\vec{G}$

Let L be an oriented line in $\mathbb{R}^3$, $\vec{L}$ the versor along L. Let B be a fully dimensional convex body with uniform charge density ρ. The *Geometric Field* $\vec{G}$ in a point p external to B is:

$$\vec{G}(p) = \int_{L \cap p \neq 0} l(L \cap B) \rho \vec{L} dL, \tag{2.2.1}$$

where $l(s)$ is the measure of the segment s and the integral is a vectorial one (figure 2.1). The component of $\vec{G}$ along a direction $\vec{u}$ is:

$$\vec{G}(p) \cdot \vec{u} = \int_{L \cap p \neq 0} l(L \cap B) \rho (\vec{L} \cdot \vec{u}) dL,$$

Note that we are working in the Gaussian unit system, formulas can then be translated to the SI system and vice-versa by setting the factor $1/(4\pi\epsilon_0)$ to 1.

In [Pel96] it is shown that definition (2.2.1) satisfies Gauss's Law of flux through a closed convex surface. Since the whole theory of electrostatic fields can be derived from Gauss's Law, that means the equivalence between the fields $\vec{E}$ and $\vec{G}$ which will not be distinguished anymore.

2.2.2 Volume-to-volume integrals

The electrostatic force acting between two tetrahedra T_1 and T_2 can be found integrating over the points of T_2 the field $\vec{E_1}$ generated by T_1 at each such point and multiplying by the charge:

$$\vec{F}_{12} = \int_{p \in T_2} \vec{E_1}(p) \rho_2 \, dp,$$

and expanding by formula (2.2.1):

$$\vec{F}_{12} = \int_{p \in T_2} [\int_{L \cap p \neq \emptyset} l(L \cap T_1) \rho_1 \vec{L} \, dL] \, \rho_2 \, dp.$$

Then transposing the integrals and taking ρ_1 and ρ_2 out of them (they are constants):

$$\vec{F}_{12} = \rho_1 \rho_2 \int_L [\int_{L \cap p \neq \emptyset, p \in T_2} l(L \cap T_1) dp] \vec{L} \, dL. \tag{2.2.2}$$

The kernel function $l(L \cap T_1)$ does not depend on p and can be taken out of the inner integral; what remains

$$\int_{L \cap p \neq \emptyset, p \in T_2} 1 dp$$

is the measure of the intersection of L and T_2, $l(L \cap T_2)$, so (2.2.2) is:

$$\vec{F}_{12} = \rho_1 \rho_2 \int_L l(L \cap T_1) l(L \cap T_2) \vec{L} \, dL. \tag{2.2.3}$$

A line L in the space can be expressed by means of two parameters: a direction $u \in \Omega$ and a point q that represents the intersection point between L and the plane orthogonal to u and passing through the origin [San67]. Instead of integrating over the set Ω of directed directions, it is convenient to refer to the

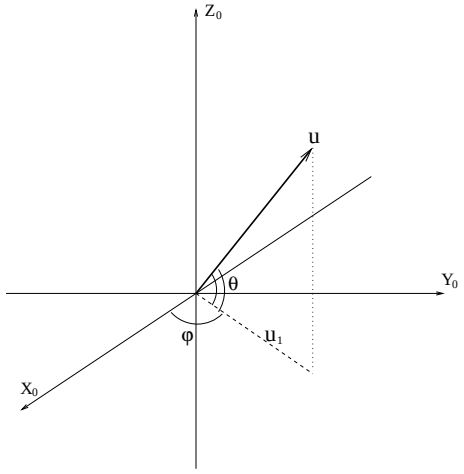

Figure 2.2: Polar coordinates

set U of unoriented directions, obtained by Ω identifying antipodal points of the unit sphere. Thus (2.2.3) becomes:

$$\vec{F}_{12} = \rho_1 \rho_2 \int_{u \in U} V_{12}(u) \vec{u}\, du, \tag{2.2.4}$$

with

$$V_{12}(u) = \int_q l(L(u,q) \cap T_1)\, l(L(u,q) \cap T_2) dq. \tag{2.2.5}$$

The inner scalar integral $V_{12}(u)$ is a planar integral over the polygon P obtained by the intersection of the projections of the tetrahedra over a plane orthogonal to the direction u, (fig. 2.3), and can be always computed exactly. Whenever a line does not meet one or both the bodies, $V_{12}(u)$ gives null contribution, as one of the expressions $l(L(u,q) \cap T_1)$, $l(L(u,q) \cap T_2)$ is zero. Expressing a direction u in polar coordinates as $u(\varphi, \theta)$, where θ is the angle formed by u and the x-y plane and φ is the angle between the projection of u on the x-y plane and the x axis (figure 2.2), formula (2.2.4) becomes:

$$\vec{F}_{12} = \rho_1 \rho_2 \int_\varphi \int_\theta \cos(\theta) [V_{12}(u(\varphi, \theta))] \vec{u}\, d\theta\, d\varphi, \tag{2.2.6}$$

where the coordinates (φ, θ) vary according to one of the following ranges: $[k\pi, (k+1)\pi] \times [j\pi, (j+1)\pi]$ or $[k\pi, (k+2)\pi] \times [j\pi, (j+1/2)\pi]$, with $k, j \in \mathbb{R}$. It must be stressed that function (2.2.5) does not diverge when the distance

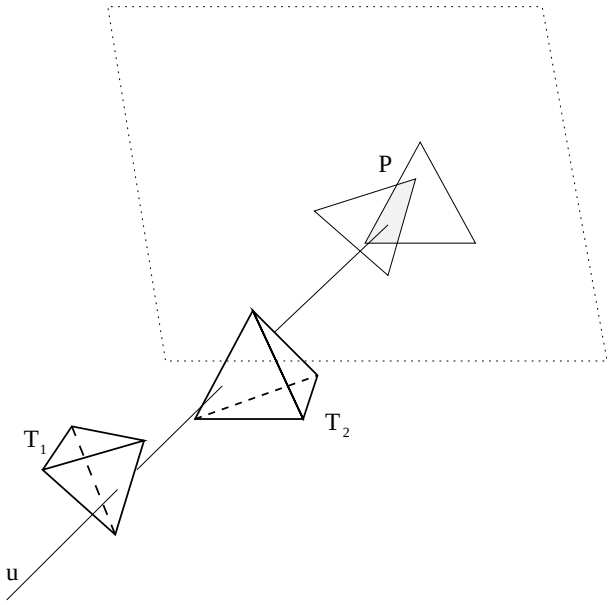

Figure 2.3: Intersection area of the projections of the tetrahedra

between tetrahedra goes to zero and it does not present any kind of singularity within the integration domain.

Since a direction is characterized by two parameters, with respect to the classic expression of $\vec{F}_{12}$ formula (2.2.4) has also a dimensional improvement, being a two-dimensional integral in place of a six-dimensional one.

2.2.3 Surface-to-surface integrals

The theory in the previous subsection applies to extended bodies in three-space (volume-to-volume integrals). In applications such as the Boundary Element Method it is important to face also the case of flat objects endowed with surface densities of charge (surface-to-surface integrals). In order to determine the electrostatic force that triangle T_1 exerts over triangle T_2, respectively having surface charge density σ_1 and σ_2, it is considered formula (2.2.4) applied to two prisms with basis T_1 and T_2 and by letting the height of the prisms go to zero while maintaining a consistency condition on the local charge. As shown in [Pel97], the component C_{12} of $\vec{F}_{12}$ along the normal to T_1 is:

$$C_{12} = \int_{u \in U} \left[\frac{1}{\cos \psi(u)} \int_q \delta(T_1, T_2, u, q)\, \sigma_1 \sigma_2 \, dq \right] du, \qquad (2.2.7)$$

where $\psi(u)$ is the angle formed by the line $L(u)$ and the normal to T_2 and δ is so defined:

$$\delta(T_1, T_2, u, q) = \begin{cases} 1 & \text{if } (L(u,q) \cap T_1 \neq \emptyset) \wedge (L(u,q) \cap T_2 \neq \emptyset) \\ 0 & \text{otherwise} \end{cases}$$

On the same reference it is proved that the kernel function

$$K(u) = \frac{1}{\cos\psi(u)} \int_q \delta(T_1, T_2, u, q)\, dq \tag{2.2.8}$$

does not diverge for $\cos\psi(u)$ going to zero and that the value of the limit can be derived by applying a limiting process to the theory of the electrostatic force of extended bodies.

Switching to polar coordinates (figure 2.2), we get:

$$C_{12} = \sigma_1\sigma_2 \int_\varphi \int_\theta \left[\frac{\cos\theta}{\cos\psi(u(\varphi,\theta))} \int_q \delta(T_1, T_2, u, q)\, dq \right] d\theta\, d\varphi. \tag{2.2.9}$$

Note that $\int_q \delta(T_1, T_2, u, q)\, dq$ is the area of the polygon $P(u)$ obtained by the intersection of the projections of T_1 and T_2 over a plane orthogonal to the direction u; P is a convex polygon with at most 6 edges.

2.3 Numerical integration

In the previous section we have derived the classes of integrals that arise in the computation of the electrostatic force through Coulomb's law (formulas 2.1.1, 2.1.2) and those resulting from the geometric interpretation of the electrostatic field (formulas 2.2.4, 2.2.7):

$$\vec{F}_{12} = \rho_1\rho_2 \int_{u\in U} V_{12}(u)\vec{u}\, du$$

$$V_{12}(u) = \int_q l(L(u,q) \cap T_1)\, l(L(u,q) \cap T_2) dq$$

and

$$C_{12} = \sigma_1\sigma_2 \int_{u\in U} K(u)\, du$$

$$K(u) = \frac{1}{\cos\psi(u)} \int_q \delta(T_1, T_2, u, q)\, dq.$$

The aim of the following chapters is to obtain accurate approximations of the latter class of integrals. For each direction u it is possible to give explicit expressions (as it will be shown in chapter (3)), for the kernels as functions of the bodies' (tetrahedra or triangles) vertices, but they are laborious to write, being V and K piecewise smooth functions, and in general they don't have elementary primitive. That is why we have to rely on approximations. Therefore here we introduce several general methods and one *ad-hoc* method built upon as much information about the kernel function as possible. All the algorithms belong to the "quadrature sums" class and they basically differ in the choice of sampling points within the integration domain U.

2.3.1 Gaussian Formulas

Quadrature formulas to approximate the integral $\int_a^b f(x)\,dx$ usually have this form:

$$S_n = \sum_{i=1}^{n} \omega_i f(x_i)$$

where n is the *order* of the formula, points $x_i,\ i = 1, 2, \ldots, n$ are the *nodes* and $\omega_i,\ i = 1, 2, \ldots, n$ are the *coefficients* or *weights*. Coefficients do not depend on f but depend on n and on the choice of the nodes. Calling the *remainder* of a quadrature formula S_n the absolute analytic error:

$$r_n = |\int_a^b f(x)\,dx - \sum_{i=1}^{n} \omega_i f(x_i)|,$$

we say that formula S_n has *precision order* k if:

$$r_n = 0 \qquad \text{when} \qquad f(x) = x^j \text{ with } j = 0, 1, \ldots, k.$$

By linearity of the integral operator, a formula with precision order k gives the exact value whenever it is applied to a function that is a polynomial of degree $\leq k$.

Often quadrature formulas are derived by substituting to the integrand function $f(x)$ an approximating polynomial $p(x)$ and integrating it:

$$\int_a^b f(x)\,dx \approx \int_a^b p(x)\,dx.$$

If p is an interpolatory polynomial with respect to the function f, the formula is called an *interpolatory formula*.

Gaussian formulas are interpolatory formulas in which both nodes and weights are chosen with the purpose of maximizing the precision order. In fact the n-th formula has precision order $2n-1$ that is the highest precision attainable from this kind of formulas (see [DR84] for this and the following statements).

The n nodes of the n-th formula are the n zeroes of the orthogonal polynomial of degree n defined over the integration interval $[a,b]$ with respect to the weight function 1. In order to avoid the computation of the nodes for each different interval, it's useful to use the following substitution

$$\int_a^b f(x)dx = \frac{b-a}{2}\int_{-1}^{1} f(\frac{(b-a)}{2}t + \frac{(a+b)}{2})dt, \tag{2.3.1}$$

that allows to always use the Legendre orthogonal polynomials (hence Gauss-Legendre formulas) which are defined over the range $[-1,1]$, at the cost of a slight increment of the complexity for the evaluations of f.

Fixed n, the n weights of Gaussian formulas can be expressed explicitly and they all are positive. This property guarantees the theoretical convergence of formulas as n increases. Historically the diffusion of Gaussian formulas was hindered by the fact that the nodes and the weights are irrational numbers. This was a drawback as the precision of the coefficients (computed by roots) decreased with the growth of the formula dimension n. Nowadays this is no more a problem, since coefficients can be efficiently computed by computers with many digits, keeping the round-off error low even for values of n relatively big. It must be noted that the nodes and the weights of the n-th formula can not be re-used in the computation of the $(n+1)$-th formula; this involve the evaluation of the kernel function that usually is the costly operation. Since we are interested in precision rather than execution time and since the kernel function is quite easy to calculate, we will not take in consideration variants to the Gaussian formulas that allow the re-utilization of former evaluation in place of a decrease of the attainable accuracy (Kronrod formulas [DR84, Eng80, SS66]).

2.3.2 Monte Carlo Method

Let us consider the generic integration problem, $I = \int_G f(x)\,dx$, where the integration domain G has finite Lebesgue measure. Monte Carlo method ([HH64] for a full explanation) applied to this kind of problem yields an approximation I' of I by the following formula:

$$I = \int_G f(x)\,dx \approx \frac{\lambda(G)}{N}\sum_{n=1}^{N} f(r_n) = I',$$

where λ denotes the Lebesgue measure function and $r_1, \ldots, r_n \in G$ are N independent random samples over G. Referring to our instances of the general problem, we have

$$I_1 = \vec{F}_{12} \approx \rho_1\rho_2\frac{2\pi}{N}\sum_{i=1}^{N} V_{12}(u_i)\,\vec{u_i} = I_1',$$

and

$$I_2 = C_{12} \approx \sigma_1\sigma_2\frac{2\pi}{N}\sum_{i=1}^{N}\int_q \delta(T_1, T_2, u, q)\,dp = I_2'$$

where u_i are directions chosen uniformly at random over the unit semi-sphere.

Thanks to this method we can get (ϵ, δ)-approximations of the integrals, approximations that are with probability $1-\delta$ within an absolute error ϵ from the correct value, choosing $N = O\left(\frac{1}{\epsilon^2\delta}\right)$ ([Pel96, par.4]; it is also determined the constant factor). Conversely, fixed the δ parameter, results with an absolute error ϵ the evaluations can be computed in $O\left(\frac{1}{\sqrt{N}}\right)$. Better asymptotic results can be achieved by using a trick described in [JVV86]: instead of computing the mean value out of $N = k\frac{1}{\epsilon^2\delta}$ directions, we calculate the median value of $O(\log(1/\delta))$ experiments made with $O(4/\epsilon^2)$ directions. The total number of evaluations of $V_{12}(u)$ is then $O\left(\frac{\log(1/\delta)}{\epsilon^2}\right)$ that is asymptotically better than $N = O\left(\frac{1}{\epsilon^2\delta}\right)$.

2.3.3 Quasi Monte Carlo Methods

The basic idea of Quasi Monte Carlo methods is to choose in a deterministic way the sampling points instead of trying to simulate randomness (see Niederraiter for a detailed treatment of the Quasi Monte Carlo methods, [Nie92]).

Therefore the Quasi Monte Carlo formula to approximate $I = \int_G f(x)\,dx$ is:

$$I = \int_G f(x)\,dx \approx \frac{\lambda(G)}{N} \sum_{n=1}^{N} f(x_n) = I'',$$

where $x_1, x_2, \ldots, x_N \in G$ are points chosen deterministically.

Error analysis in [Nie92] shows that $|I'' - I|_N$, that is the absolute error obtained approximating I with I'' using N sampling points, is directly proportional to the *discrepancy* of the set $\{x_1, x_2, \ldots, x_n\}$. Given a set of points Q, the discrepancy $D(Q)$ can be intended as a measure of the uniformity of distribution of the points in Q. If we fix the number N of elements of the set —obtaining the so called *point set*— so that all the points of the set can be pre-computed, then $D(Q) = O\left(\frac{1}{N}\right)$. This is an improvement compared to the $O\left(\frac{1}{\sqrt{N}}\right)$ achieved by the Monte Carlo method. Whenever it is needed to do several experiments over the same function with an increasing number of sampling points, it is helpful to save the former evaluations in order to include them in the In this case we talk about *sequence* as a completely determined infinite sequence of points S, whose first N terms are considered. In this case we have that $D_N(S) = O\left(\frac{\log N}{N}\right)$.

As regard our integrals, it is convenient to use the formulas involving the φ and θ variables (2.2.6, 2.2.9), because methods to produce point-sets are defined over a n-dimensional cube (n $\geq$ 2) rather than over the unit sphere.

The simplest point-sets are obtained by diving the edges of the rectangular integration domain in k parts and taking the Cartesian product of such points building a *grid* made up by $(k+1)^2$ points. Even though these grids are extremely easy to construct, they have a couple of drawbacks: their discrepancy is quite large, there are many other sets that better simulate uniformity, and the points can be reused only doubling the number of points on each side of the cube.

2.3.4 Gauss Adaptive Method

All the methods shown so far generate sampling directions without any knowledge of the kernel functions, in particular without exploiting the fact they are

continuous piecewise smooth functions. Here we present the idea underlying an adaptive algorithm that will be discussed in detail in section 3.5.
We put attention on the case of triangles as input bodies; the kernel function is:

$$K(u) = \frac{1}{\cos\psi(u)} \int_q \delta(T_1, T_2, u, q)\, dq,$$

we want to explicitly express it as function of the vertices of T_1 and T_2 by means of elementary functions. The factor $\psi(u)$ is the angle formed by the line $L(u)$ and the normal to T_2, then $\cos\psi(u)$ is derived from the following formula:

$$\cos\upsilon = \frac{v'_x v''_x + v'_y v''_y + v'_z v''_z}{\|v'\|\|v''\|}, \tag{2.3.2}$$

where v' and v'' are vectors and υ is the angle within them. The other factor $\int_q \delta(T_1, T_2, u, q)\, dq$ is merely the area of the polygon $P(u)$ obtained by the intersection of the projections of T_1 and T_2 over a plane orthogonal to the direction u. Let us take as example the triangles ABC and ABD sharing the edge AB. Supposing their projections intersect originating a proper polygon P. As shown in figure 2.4, there are only four possible configurations:

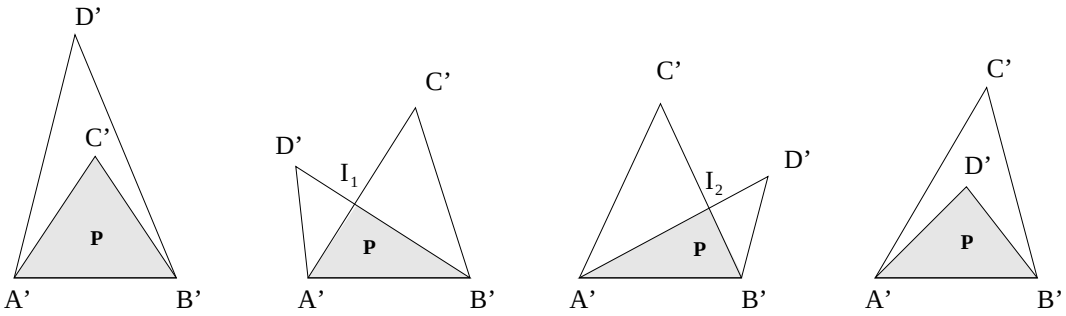

Figure 2.4: Triangles sharing one edge: there are four kinds of intersection polygon P

To each configuration corresponds a different formula for the area of P according to the names of the vertices of the intersection polygon P. The idea of the Gauss adaptive method is to locate the zones of the φ-θ plane in which the names of the vertices of polygon P are constant and generate sampling points within each zone by Gaussian formulas.

2.4 State of the Art

The approximation of integrals by a weighted sum of evaluations of the integrand function is a classic problem of numerical analysis. The most extensive bibliography about this topic can be found in [DR84]. Recent developments are reviewed in [EE92] and [KE92]. In [Duf82], [LM80], and [Sch94], it has been treated the case of integrand functions with singularity within the integration domain, particularly when domain is a simplex and the singularity is in a vertex. In the first article ([Duf82]), it is shown that for a class of singularities which includes singularities of the form $\frac{1}{p^\alpha}$ $(0 < \alpha < d)$, in $\mathbb{R}^d$ it is possible to reduce the order of the singularity so that standard product integration rules are applicable. Schwab [Sch94] proposes an adaptive quadrature rule that for a vast class of integrals achieves an exponential order of convergence. The case of integrand functions with a set of singularities with dimension greater than zero (e.g. polyhedra sharing one face or one edge), seems not to immediately fit in the theory of the mentioned works. With reference to our experiments, we did not find any treatment about integrals of the Coulomb's law over two tetrahedra and in general over polyhedra. Rough estimates of such integrals are obtained by identifying one of the polyhedra with a point and assuming the charge being concentrated in it. Integrals of the form (2.1.2) have been greatly studied for their importance for the Boundary Element Method, (see [Zho93], [BL73], [SW92a], [SW92b], [ES]). In [HS93], Hackbusch and Sauter consider a class of integrals which include integrals of form (2.1.2) and obtain an equivalent formulation where the domain of integration is tridimensional and the kernel function has no singularities within the integration domain. Then approximations are obtained by Gaussian formulas, but an error analysis is not given. Sauter and Schwab [SS96], consider the same class of integrals and obtain 4-dimensional integrals whose kernel function is free from singularities within the domain of integration. In order to approximate such integrals they propose tensor-product formula which provably converges as $O(e^{-c\sqrt[4]{n}})$, where n is the number of points used and c is a constant. Pellegrini [Pel96], [Pel97], thanks to the integral geometry approach obtains for integrals (2.1.1) and (2.1.2) 2-dimensional integrals and proves that a Gaussian integration

converges as $O(e^{-\sqrt{n}})$, where n is the number of Gaussian points used by the algorithm.

What's going on tonight?
And oh, my God, look, you have just discovered
The way one think can lead to another
But isn't it funny how one thing leads to another?

Neil Tennant

Chapter 3

Implementation

In the previous chapter we derived the integrals (2.2.4, 2.2.7) which are new formulas for the electrostatic force acting between two tetrahedra and two triangles. Moreover we showed numerical methods to approximate such integrals. In this chapter we describe the implementation of those methods and we dwell upon the evaluation of the kernel functions. In fact up to now we used "procedural" definitions for the kernel functions rather than explicit definitions by means of elementary functions.

3.1 General Algorithm

We have implemented all the methods discussed so far: Gauss, Monte Carlo, Quasi Monte Carlo and some variants to them. We also took in exam the Gauss Adaptive method when the input bodies B_1 and B_2 are triangles. All these methods can be grouped together under the class of the "quadrature sums" algorithms: the value of $\int_D f(x)\,dx$ is approximated by $\sum_{i=1}^{N} w_i\, f(n_i)$ with $n_i \in D$. The choice of w_i, the *weights*, and n_i, the *nodes*, vary according to the method. The implementation is naturally split in two phases: the computation of nodes and weights relatively to the used method and the evaluation of the kernel function in such nodes. In our instance, given a direction u, the kernels yield a non null-contribution if and only if the input bodies projected on a plane orthogonal to u shape a proper intersection polygon. Thus the second phase can be farther subdivided in two steps, the first being the detection of the intersection polygon and the second being the actual evaluation of the kernel.

The inputs of the algorithm —whatever method we use— are the coordinates of the vertices of the bodies (tetrahedra or triangles), expressed as triple of reals, and the integer N, the number of directions that will be used to get an estimation of the integrals (2.2.4, 2.2.7). In the case of tetrahedra the output is an approximated value of the x-projection of the electrostatic force vector; if the bodies are triangles the output is the approximation of the projection of the electrostatic force on a vector orthogonal to the triangle T_1.

Given two bodies B_1 and B_2 and an integer N, the blueprint of all the implemented algorithms is a loop like the one in Table 3.1.

$int = 0$;
for $(i = 0;\ i < N;\ i = i + 1)$
{
$u =$ `generate_direction(`i`);`
if `is_intersection(`B_1, B_2, u`)`
then $int = int\ +$ `contribution(`B_1, B_2, u`);`
};

Table 3.1: Scheme of the general algorithm

The outcome —an approximated value of integrals (2.2.4, 2.2.7)— is the content of the variable int.

To the function `generate_direction` correspond several implementations according to the chosen method; it does not depend on the dimension of the input bodies. In first instance all the methods but the Gauss adaptive are also independent from the positions of the bodies in the space. We will see why this is a drawback and how to avoid it.

The function `is_intersection` yields $True$ as answer if the projections of the bodies over a plane orthogonal to the direction u have a common area. The implementation for tetrahedra is easily derived from the one for triangles.

The function `contribution` computes the exact value of the kernels, $V_{12}(u)$ for tetrahedra (2.2.5) and $K(u)$ for triangles (2.2.8), and yields the value $w_i\, f(u_i)$. It has different implementations for tetrahedra and triangles.

3.1.1 Generate_direction

In the case of Monte Carlo method, we refer to the equations (2.2.4, 2.2.7); a direction is then meant to be a point on the unit sphere and an unoriented direction is a point on a semi-sphere. For all the other methods we applied the transformation to polar coordinates (fig.2.2) to the previous integrals (obtaining the equations 2.2.6, 2.2.9), hence a direction is a point (φ, θ) belonging to one of the following rectangles: $[k\pi, (k+1)\pi] \times [j\pi, (j+1)\pi]$ or $[k\pi, (k+2)\pi] \times [j\pi, (j+1/2)\pi]$, with $k, j \in \mathbb{R}$.

The Monte Carlo method (MC) provides the directions for being randomly generated; it means we have to produce sampling point over the unit semisphere. We obtain such points by the algorithm presented by Knuth in [Knu69], (see Appendix A.1). We also found other two techniques but all the algorithms give essentially the same results from a 'randomness' point of view. It must

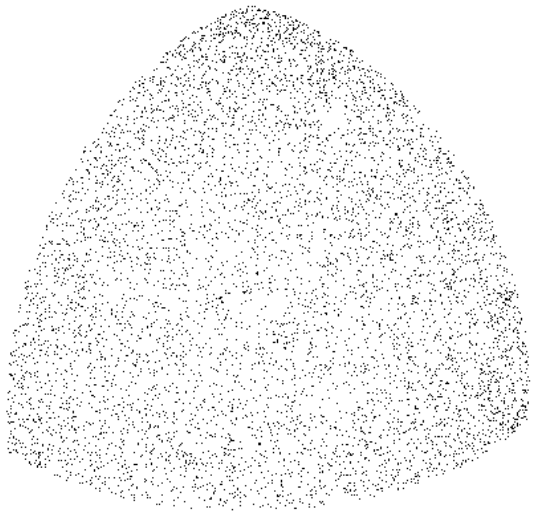

Figure 3.1: Random points in a spherical zone

be stressed that all these algorithms rely on random variables uniformly distributed over the interval $[0, 1]$, therefore the effectiveness of the set returned greatly depends on the underlying pseudo-random number generator. Looking at the pattern of figure (3.1), it is evident that sampling points are oddly distributed on the sphere, causing 'empty' and 'overcrowded' zones, against our

intuition of uniform distribution. Clearly several executions of the algorithms with the same inputs return different results.

One of the aims of the Quasi Monte Carlo method (QMC) is to overcome this problem of simulating the randomness by creating pre-determined points —they are always the same for all the executions of the algorithm— which well approximate the distribution of random points, that is, a set with small discrepancy. Since we know the precise dimension (N) of the set we have to generate, we can use a formula for point sets rather than for sequences, getting lower discrepancy values (subsection 2.3.3). In [Sch72] it is proven that the set of points (φ, θ), with the lowest discrepancy is the *N-element Hammersley point set in base three* (see also [Nie92]):

$$(x_n, y_n) = \left(\frac{n}{N}, \phi_3(n)\right) \text{ for } n = 0, 1, \ldots, N-1$$

where $\phi_b(n)$ is the *radical inverse function in base b* (see Appendix A.2). The points on the rectangle $[a, b] \times [c, d]$ will be: $(a + (b-a)\, x_n, c + (d-c)\, y_n)$. See figures 3.2, 3.3.

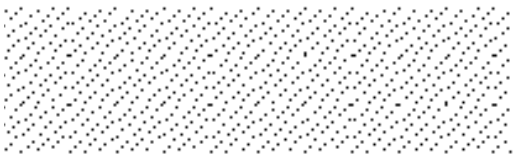

Figure 3.2: 1000 sampling points over the rectangle $[0, 2\pi] \times [0, \pi/2]$ generated by the Hammersley point set

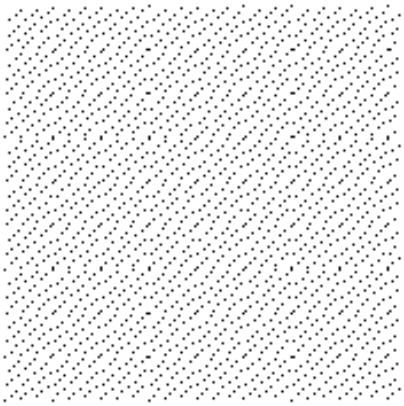

Figure 3.3: 2000 sampling points over the square $[0, \pi] \times [0, \pi]$ generated by the Hammersley point set

The points forming a grid of dimension N over the square $[0, \pi] \times [0, \pi]$ are those of the Cartesian product between the points that subdivide each axis in equal parts. Assuming $N = k^2$, or conversely computing $k = \lfloor \sqrt{N} \rfloor$, points

are:

$$p_{ij} = (\varphi_i, \theta_j) = \left(i\,\frac{\pi}{k}, j\,\frac{\pi}{k} \right) \text{ with } i, j = 0, \ldots, (k-1).$$

On the other side, having to build a grid over the rectangle $[0, 2\,\pi] \times [0, \pi/2]$ we forced the number of points on the φ axis to be four times the number of points on the θ axis (figure 3.4). With $N = 4k^2$ we have:

$$p_{ij} = (\varphi_i, \theta_j) = \left(i\,\frac{2\pi}{4k}, j\,\frac{\pi}{2k} \right) = \left(i\,\frac{\pi}{2k}, j\,\frac{\pi}{2k} \right)$$

$$\text{with } i = 0, \ldots, (4k) - 1) \text{ and } j = 0, \ldots, (k-1).$$

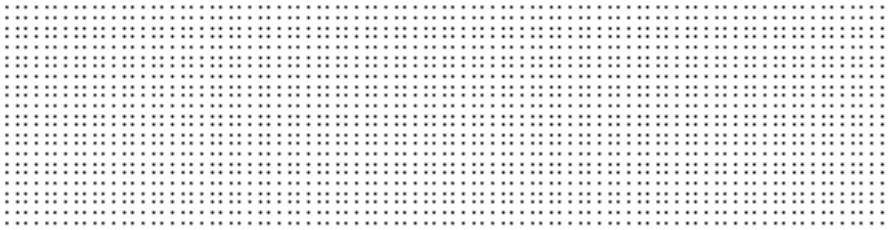

Figure 3.4: Rectangular grid: the number of points on axis φ is 4 times the number of points on θ axis

In the Gauss method, once we are given the nodes and the weights of the k-th Gaussian formula over the interval $[-1, 1]$, with $k^2 = N$, we get sampling directions on the rectangle $[a, b] \times [c, d]$ by the following Cartesian product:

$$p_{ij} = (\varphi_i, \theta_j) = \left(\frac{b-a}{2} x_i + \frac{a+b}{2}, \frac{d-c}{2} x_j + \frac{c+d}{2} \right) \text{ with } i, j = 0, \ldots, (k-1),$$

x_t and being the t-th Gaussian point of the k-th formula. See figure 3.5.
In section 3.5 we show how to generate points in the φ-θ plane according to an adaptive method.

3.2 Is_intersection

Once it is generated a direction u, the following step within the main algorithm (3.1) consists in deciding whether B'_1 and B'_2 —respectively the projections of

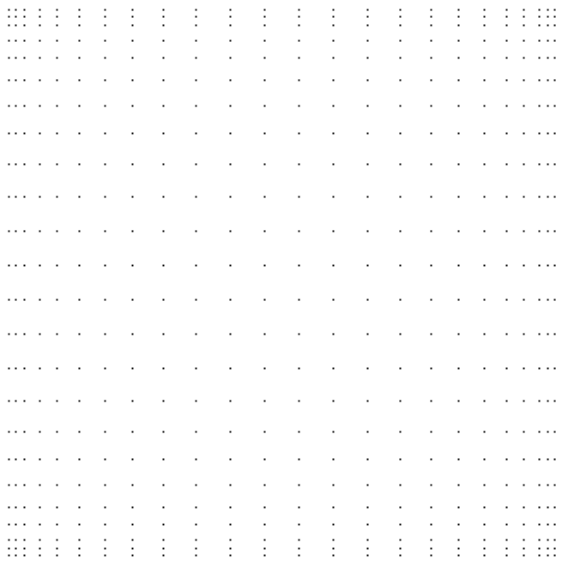

Figure 3.5: Gaussian point over the square $[0, \pi] \times [0, \pi]$

the input bodies B_1 and B_2 over a plane orthogonal to u— intersect or not. We start assuming B_i to be triangles and then we will show how the process can be extended to tetrahedra. It is straightforward to subdivide this function in two steps: projecting the input bodies on a plane orthogonal to direction u generating the polygons B_1' and B_2' and testing whether they intersect or not.

3.2.1 Projection

We have to project the triangles T_1 and T_2 over a plane ζ. This means we must fix a Cartesian planar reference system Π defined over ζ. Then we have to express by Π the coordinates of the points generated by the intersection of the plane ζ and the lines parallel to u and passing from the triangles' vertices. For this we use the Euler rotation matrices $\mathbf{Q}(\alpha), \mathbf{R}(\beta), \mathbf{S}(\gamma)$:

$$\mathbf{Q}(\alpha) = \begin{pmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \mathbf{R}(\beta) = \begin{pmatrix} \cos\beta & 0 & -\sin\beta \\ 0 & 1 & 0 \\ \sin\beta & 0 & \cos\beta \end{pmatrix}$$

$$\mathbf{S}(\gamma) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & \sin\gamma \\ 0 & -\sin\gamma & \cos\gamma \end{pmatrix}$$

Fixed the angle α, the matrix $\mathbf{Q}(\alpha)$ represents a rotation of the Cartesian reference system around the Z axis measured α radians. In the same way, $\mathbf{R}(\beta)$ is a rotation around the Y axis measured β radians and $\mathbf{S}(\gamma)$ represents a rotation around the X axis measured γ radians. So if the reference system $\Pi_0 = (X_0, Y_0, Z_0)$ experiences a rotation around the Z_0 axis of t_1 radians ($t_i \in \mathbb{R}$), a point p with (old) coordinates (p_x, p_y, p_z), acquires coordinates

$$\mathbf{Q}(t_1) \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix}$$

with respect to the new reference system $\Pi_1 = (X_1, Y_1, Z_1{=}Z_0)$. Then the coordinates of p with respect to a new reference system $\Pi_2 = (X_2, Y_2{=}Y_1, Z_2)$, generated from Π_1 by a rotation around the axis Y_1 are:

$$\mathbf{R}(t_2)\mathbf{Q}(t_1) \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix}$$

Finally, a rotation of t_3 radians around the axis X_2 causes the coordinates of p to be

$$\mathbf{S}(t_3)\mathbf{R}(t_2)\mathbf{Q}(t_1) \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix}$$

with respect to the reference system $\Pi_3 = (X_3 = X_2, Y_3, Z_3)$. Since every rotation of the reference system can be decomposed in three successive rotations like these, the general expression for the coordinates of a point p is: $\mathbf{M}(\alpha, \beta, \gamma)\,(p_x, p_y, p_z)^T$, where $\mathbf{M}(\alpha, \beta, \gamma) = \mathbf{S}(\gamma)\,\mathbf{R}(\beta)\,\mathbf{Q}(\alpha)$.

A direction u is univocally expressed by means of two angles, φ and θ, as shown in figure 2.2. In order to establish a planar reference system for the plane ζ, orthogonal to u, we first rotate the global reference system (where the triangles are defined) around the axis Z_0 for φ radians, so that the axis X_0 overlaps the segment u_0 (figure 3.6). Then a rotation around Y_1 following the θ angle makes the new axis X_2 be coincident with u (figure 3.7). For fixing a reference system over ζ we still have to specify the parameter γ, that indicates the rotations around the new X axis (X_2): we conventionally set

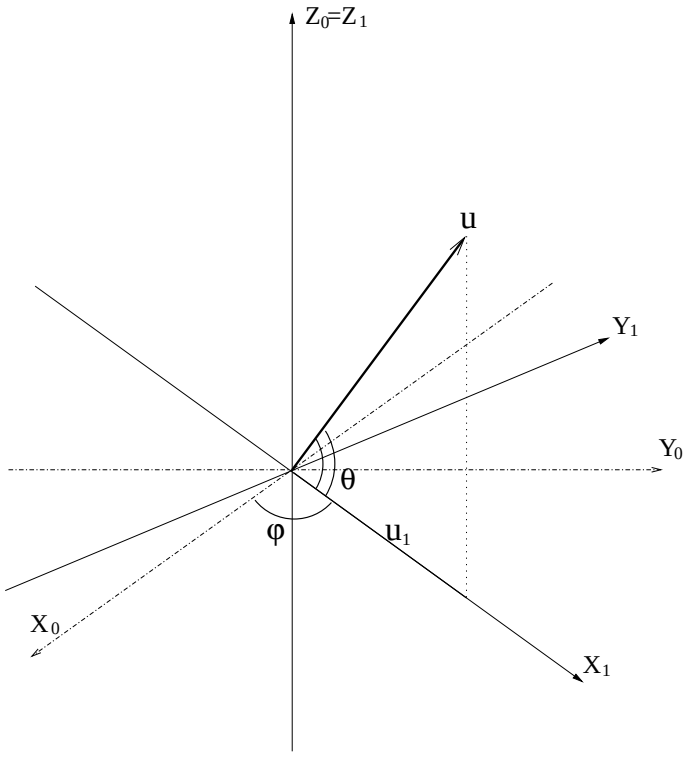

Figure 3.6: Rotation of φ radians around the axis Z_0

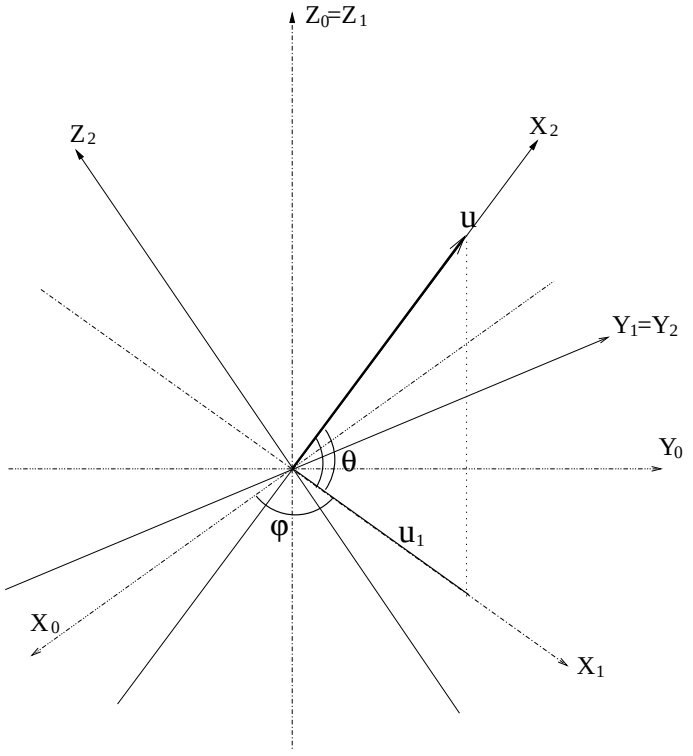

Figure 3.7: Rotation of θ radians around the axis Y_1

$\gamma = 0$ for every u. Thus, the axes Y_2 and Z_2 induce a planar reference system over the plane ζ. As regards the explicit computation of vertices' coordinates, $\mathbf{S}(\gamma) = \mathbf{S}(0)$ is the identity matrix of dimension 3 and $\mathbf{M}(\alpha, \beta, 0)$ with $\alpha = \varphi$ and $\beta = \theta$ becomes:

$$\mathbf{M}(\varphi, \theta, 0) = \begin{pmatrix} \cos\varphi\,\cos\theta & \sin\varphi\,\cos\theta & -\sin\theta \\ -\sin\varphi & \cos\varphi & 0 \\ \cos\varphi\,\sin\theta & \sin\varphi\,\sin\theta & \cos\theta \end{pmatrix}.$$

Thus right-multiplying $\mathbf{M}$ by the vector $(p_x, p_y, p_z)^T$ we get:

$$(p'_x, p'_y, p'_z) = \mathbf{M}(\alpha, \beta, 0)\,(p_x, p_y, p_z)^T =$$

$$= \begin{pmatrix} \cos(\varphi)\cos(\theta)\,p_x + \sin(\varphi)\cos(\theta)\,p_y - \sin(\theta)\,p_z \\ -\sin(\varphi)\,p_x + \sin(\varphi)\,p_y \\ \cos(\varphi)\sin(\theta)\,p_x + \sin(\varphi)\sin(\theta)\,p_y - \cos(\theta)\,p_z \end{pmatrix}^T$$

that are the coordinates of the point p referred to the new reference system. The planar coordinates of p are (p'_y, p'_z). We have then expressed the projection of the triangles vertices as functions of the vertices and α and β that are derived from the direction u.

3.2.2 Intersection test

Given two triangles T'_1 and T'_2 we are interested in deciding whether they intersect or not. It must be noted that at this stage of the algorithm we just want to know -if- they generate an intersection polygon, not to determine it. This allow us to use fast geometrical tools in place of the analytic computation of the intersection points. Next we introduce such tools ([O'R94, PS85]).

Area of a triangle

Here we show how to compute the area of a triangle T without using operations that somehow increase the round-off error; in fact the well known formula $Area(T) = 1/2 \times base \times height$, needs functions such as square roots, sines, that inevitably have inherent round-off error. It is however possible to express the area of a triangles only by means of sums, differences and multiplications.

Definition 3.2.1 *Given two planar vectors v_1 and v_2, respectively with components (v_{1x}, v_{1y}) and (v_{1x}, v_{1y}), the* Cross Product of v_1 and v_2 *is a vector orthogonal to v_1 and v_2 whose modulus is $v_{1x}v_{2y} - v_{1y}v_{2x}$.*

The modulus of the cross product vector of v_1 and v_2 can be interpreted as the signed area of the parallelogram formed by the points $(0,0), v_1, v_2$, and $v_1 + v_2 = (v_{1x} + v_{2x}, v_{1y} + v_{2y})$. Thus the area of the triangles $(0,0), v_1, v_2$, is found by halving the absolute value of the mentioned product. In general, given a triangle $T = \{V_0, V_1, V_2\}$, with $V_0 = \{x_0, y_0\}, V_1 = \{x_1, y_1\}, V_2 = \{x_2, y_2\}$, the area is given by:

$$\text{Area}(T) = \frac{1}{2} \left| ((x_1 - x_0)(y_2 - y_0) - (y_1 - y_0)(x_2 - x_0)) \right|,$$

expanding and simplifying

$$\text{Area}(T) = \frac{1}{2} \left| x_1 y_2 - x_1 y_0 - x_0 y_2 - x_2 y_1 + x_0 y_1 + x_2 y_0 \right|$$

and introducing the function $\kappa(V_i, V_j) = V_{ix}V_{jy} - V_{iy}V_{jx}$, it can be rewritten as

$$\text{Area}(T) = \frac{1}{2} \left| \kappa(V_0, V_1) + \kappa(V_1, V_2) + \kappa(V_2, V_0) \right|.$$

Left Turn

Definition 3.2.2 *Given three disjoint points p_0, p_1, p_2, in a plane, we say they form a* Left Turn *if they are counterclockwise; they form a* Right Turn *if they are clockwise.*

In order to know whether three points p_0, p_1, p_2, form a left turn or a right turn, it is enough to compute $k = \kappa(p_0, p_1) + \kappa(p_1, p_2) + \kappa(p_2, p_0)$, that is twice the signed area of the triangles whose vertices are p_0, p_1, p_2:

- if $k \geq 0$ then the points are in anti clockwise order, i.e., they form a left turn. In this case we set $\texttt{left\_turn}(p_0, p_1, p_2) = 1$
- if $k \leq 0$ then the points are in clockwise order, i.e., they form a right turn. In this case we set $\texttt{left\_turn}(p_0, p_1, p_2) = 0$
- if $k = 0$ then the points are collinear.
  In this case we set $\texttt{left\_turn}(p_0, p_1, p_2) = -1$

Intersection of segments

Let $\{a, b\}$ and $\{c, d\}$ be two segments. Let us suppose they are not collinear, then they intersect in a point if and only if

$$(\texttt{left\_turn}(a, b, c) \neq \texttt{left\_turn}(a, b, d)) \ \wedge$$
$$(\texttt{left\_turn}(c, d, a) \neq \texttt{left\_turn}(c, d, b))$$

For the general case, it must be defined the function `between`, that taken the coordinates of three points yields *True* if the third point lays on the segment formed by the first two points:

```
between(a, b, c : points):=
if left_turn(a, b, c) ≠ -1
   then return error
if a_x ≠ b_x
   then return (min(a_x, b_x) ≤ c_x ≤ max(a_x, b_x))
   else return (min(a_y, b_y) ≤ c_y ≤ max(a_y, b_y))
```

So if it holds $\texttt{left\_turn}(a, b, c) = \texttt{left\_turn}(a, b, d) = -1$, i.e., the segments are collinear, they intersect if and only if $\texttt{between}(a, b, c) \vee \texttt{between}(a, b, d)$. We have thus completely defined the function `intersection` that taken two segments returns *True* when they intersect:

```
intersection(a, b, c, d : points):=
if left_turn(a, b, c) = left_turn(a, b, d) = −1
   then return (between(a, b, c) ∨ between(a, b, d))
   else return ((left_turn(a, b, c) ≠ left_turn(a, b, d)) ∧
               (left_turn(c, d, a) ≠ left_turn(c, d, b)))
```

This is the tool we need to detect whether two triangles intersect or not. Given two triangles $T1 = \{t1_0, t1_1, t1_2\}, T2 = \{t2_0, t2_1, t2_2\}$, the function $\texttt{is\_intersection}(T_1, T_2)$ is so defined:

```
is_intersection(T_1, T_2 : triangles):=
found = False;
while (i < 2) ∧ (found = False)
```

while $(j < 2) \wedge (found = False)$
$found$ = `intersection(`$t1_i, t1_{(i+1)_3}, t2_j, t2_{(j+1)_3}$`);`
if $found = False$
then $found =$ `is_inside(`T_1, T_2`);`
if $found = False$
then $found =$ `is_inside(`T_2, T_1`);`
return($found$**);**

The function `is_inside` takes two polygons as input and just tests whether a the vertex of the second polygon is internal to the first one: that would be enough to establish that the second polygon lays inside the first one. The implementation for `is_inside` is an immediate derivation of the `left_turn` function.

3.2.3 Tetrahedra

The projection of a tetrahedron on a plane generates a triangle or a quadrilateral. In computer notation, a triangle is represented by a list of three points, no matter of the order, while a quadrilateral needs more care: depending on the order, four points can give birth (rise) to non-simple quadrilaterals, that is, self-intersecting polygons . For this, once we have projected the four tetrahedron's vertices on a plane, we compute their convex hull (see Appendix A.3): the results is a triangle if one of the vertices falls within the zone shaped by the other three, or a quadrilateral whose vertices are specified in counterclockwise order. The function `is_intersection` can be then extended to deal with any kind of polygons, provided they are well specified:

`is_intersection(`$P_1, P_2 : polygons$`):=`
$found = False$;
while $(i <$ `length(`P_1`)`$) \wedge (found = False)$
while $(j <$ `length(`P_2`)`$) \wedge (found = False)$
$found$ = `intersection(`$t1_i, t1_{(i+1)_{\texttt{length}(P_1)}}, t2_j, t2_{(j+1)_{\texttt{length}(P_2)}}$`);`
if $found = False$
then $found =$ `is_inside(`P_1, P_2`);`

if $found = False$
  then $found =$ `is_inside(`P_2, P_1`);`
return($found$**);**

The function `length` takes a polygon as input and return the number of its edges.

3.3 Contribution

Once we have established that given a direction u the projections B_1' and B_2' intersect, we have to exactly derive the polygon P as first step to compute the kernels, for it is the integration domain of the integrals involved in the formulas $V_{12}(u)$, (2.2.5), and $K(u)$, (2.2.8). We build a set calculating the intersection points detected with the `intersection` function, adding then the vertices of the polygons B_1' internal to B_2' and vice versa. If the convex hull (Appendix A.3) of such set returns a polygon formed by more than two vertices, it is the searched intersection polygon P; otherwise the contribution for that direction is null. In the actual implementation we do not split the detection and the computation of P, the steps are merged together in a function that yield either *False*, if P do not exist or it is made up by one or two vertices, or the list of the vertices of P. In the following two subsections we show how the kernel functions can be exactly expressed by means of direction u (specifically by the parameters φ and θ which identify u) and by means of the coordinates of the vertices of B_1, B_2 and P.

3.3.1 Triangles

We have to compute

$$K(u) = \frac{1}{\cos\psi(u)} \int_q \delta(T_1, T_2, u, q)\, dq.$$

The factor $\cos\psi(u)$ represents the cosine of the angle formed by the line $L(u)$ and the normal to T_2. It is computed thanks to formula (2.3.2), where $v' = \{\cos\varphi\cos\theta, \sin\varphi\cos\theta, \sin\theta\}$, and v'' is obtained from the coefficients of x, y

and z, resulting from the equation of the plane spanning $T_2 = \{t0, t1, t2\}$. This is given by the determinant of the matrix

$$\begin{pmatrix} x & y & z & 1 \\ t0_x & t0_y & t0_z & 1 \\ t1_x & t1_y & t1_z & 1 \\ t2_x & t2_y & t2_z & 1 \end{pmatrix}.$$

The former formula is not helpful when $\cos\psi(u)$ is zero or almost zero as we would incur effects as computation over-flow. But

The expression $\int_q \delta(T_1, T_2, u, q)\, dq$ is the area of the polygon P. It is computed using a generalization of the algorithm shown for triangles, without using square root, divisions and sines:

$$\text{Area}(P) = \frac{1}{2}\left| \sum_{i=0}^{\texttt{lenght}(P)-1} \kappa(p_i, p_{(i+1)_{\texttt{lenght}(P)}}) \right|,$$

p_i being the vertices of the polygon P specified anti clockwise.

3.3.2 Tetrahedra

We have to compute

$$V_{12}(u) = \int_q l(L(u,q) \cap T_1)\, l(L(u,q) \cap T_2) dq.$$

Instead of using numerical integration routines, we make a symbolical computation by means of the vertices of the input tetrahedra and by means of the coordinates of the vertices of the intersection polygon P. Let us denote with X, Y, Z, the Cartesian reference frame where X is the coordinate along the u direction and Y and Z are coordinates on the plane orthogonal to u. The integration domain is the polygon P, in fact every line $L(u,q)$, parallel to u but not intersecting P makes null at least one of the factors $l(L(u,q) \cap T_i)$. As shown in figure 3.8, P is divided in k sub-polygons P_k such that all the lines over $\mathbb{R}^3$ parallel to u and intersecting P_k always intercept the same faces of T_j, $j \in \{1, 2\}$ (henceforth called the *covering* faces of P_k). For each sub-polygon P_k, let $X_1^0(Y,Z)$, $X_1^1(Y,Z)$, the equations of the planes spanning the faces of T_1 and $X_2^0(Y,Z)$, $X_2^1(Y,Z)$, the equations of the planes spanning the faces of

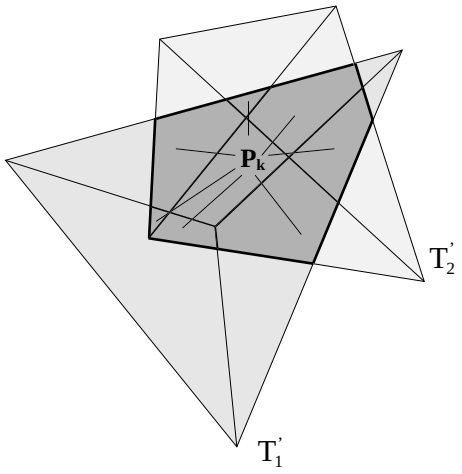

Figure 3.8: Subdivision of the intersection polygon P in sub-polygons P_k

T_2 which cover P_k. Note that $l(L \cap T_j)$ corresponds to $|Z_j^1 - Z_j^0|, \quad j \in \{1, 2\}$. Thus, equation (2.2.5) can be rewritten as:

$$V_{12}(u) = \sum_k \int_{P_k} \left|(X_1^0 - X_1^1)(X_2^0 - X_2^1)\right| \, dY \, dZ. \tag{3.3.1}$$

Polygons P_k have at most six edges, so it's immediate to triangulate them in s triangles T_s, allowing to write an integration routine specifically for triangles. Equation (3.3.1) is then

$$V_{12}(u) = \sum_k \sum_s \int_{T_s} \left|(X_1^0 - X_1^1)(X_2^0 - X_2^1)\right| \, dY \, dZ. \tag{3.3.2}$$

The general form of X_j^i is $aY + bZ + c$, where a, b, c are determined from the vertices of the faces intercepted by L. Assuming the vertices are $p_0 = \{x_0, y_0, z_0\}, p_1 = \{x_1, y_1, z_1\}, p_2 = \{x_2, y_2, z_2\}$, the terms a, b, c, result:

$$a = \frac{x2\ (z0 - z1) + x0\ (z1 - z2) + x1\ (-z0 + z2)}{-(y1\,z0) + y2\,z0 + y0\,z1 - y2\,z1 - y0\,z2 + y1\,z2},$$

$$b = \frac{x2\ (-y0 + y1) + x1\ (y0 - y2) + x0\ (-y1 + y2)}{-(y1\,z0) + y2\,z0 + y0\,z1 - y2\,z1 - y0\,z2 + y1\,z2},$$

$$c = -\frac{x2\,y1\,z0 - x1\,y2\,z0 - x2\,y0\,z1 + x0\,y2\,z1 + x1\,y0\,z2 - x0\,y1\,z2}{-(y1\,z0) + y2\,z0 + y0\,z1 - y2\,z1 - y0\,z2 + y1\,z2}.$$

Expanding and grouping formula (3.3.2) becomes:

$$V_{12}(u) = \sum_k \sum_s \int_{T_s} \Big|[(a_1^0 - a_1^1)Y + (b_1^0 - b_1^1)Z + (c_1^0 - c_1^1)] \\ [(a_2^0 - a_2^1)Y + (b_2^0 - b_2^1)Z + (c_2^0 - c_2^1)]\Big| \, dY \, dZ.$$

Grouping similar monomials and calling f, g, h, i, l, m the resulting coefficients, we obtain the following formula:

$$V_{12}(u) = \sum_k \sum_s \int_{T_s} \left| f\,Y^2 + g\,Y\,Z + h\,Y + i\,Z^2 + l\,Z + m \right| dY\,dZ.$$

Now we separate the computation of the inner integral for every edge of the triangles T_s, reducing to three integrals of the form $\int_{vi_Y}^{vj_Y} \int_0^{\texttt{line}(vi,vj,Z)} |f\,Y^2 + g\,Y\,Z + h\,Y + i\,Z^2 + l\,Z + m|\,dZ\,dY$, where vi and vj are the extremes of the edge and the function $\texttt{line}(p_1, p_2, W)$ yields the equation in variable W of the line passing through p_1 and p_2. Indicating $\texttt{line}(vi, vj, Z)$ with $rZ + t$, $r = \frac{vi_Z - vj_Z}{vi_Y - vj_Y}$, $t = \frac{vj_Z vi_Y - vi_Z vj_Y}{vi_Y - vj_Y}$, and assuming $vi_Y \neq vj_Y$ the primitive is:

$$\begin{aligned}
&\frac{-\left(b\,\left(2\,b^2\,i + 3\,b\,l + 6\,m\right)\,vi_Z\right)}{6} - \frac{\left(b^2\,g + 2\,b\,h + 2\,a\,b^2\,i + 2\,a\,b\,l + 2\,a\,m\right)\,{vi_Z}^2}{4} \\
&- \frac{\left(2\,b\,f + 2\,a\,b\,g + 2\,a\,h + 2\,a^2\,b\,i + a^2\,l\right)\,{vi_Z}^3}{6} - \frac{a\,\left(6\,f + 3\,a\,g + 2\,a^2\,i\right)\,{vi_Z}^4}{24} \\
&+ \frac{b\,\left(2\,b^2\,i + 3\,b\,l + 6\,m\right)\,vj_Z}{6} + \frac{\left(b^2\,g + 2\,b\,h + 2\,a\,b^2\,i + 2\,a\,b\,l + 2\,a\,m\right)\,{vj_Z}^2}{4} \\
&+ \frac{\left(2\,b\,f + 2\,a\,b\,g + 2\,a\,h + 2\,a^2\,b\,i + a^2\,l\right)\,{vj_Z}^3}{6} + \frac{a\,\left(6\,f + 3\,a\,g + 2\,a^2\,i\right)\,{vj_Z}^4}{24}
\end{aligned}$$

While if it holds that $vi_Y = vj_Y$, the primitive is 0.

3.3.3 Comments

We have found exact formulas for the kernel functions (2.2.5, 2.2.8) given a direction u. Moreover, the algorithms run in time $O(1)$, as the number of needed operations depends only on the number of edges of the polygons involved within the calculations and it is always possible to bound it with a constant. When the input bodies are triangles the intersection polygon P can in fact have at most 6 edges, as two triangles intersecting can generate only polygons with 3, 4, 5 or 6 edges. If the input bodies are tetrahedra, the number of operations depends on the subpolygons P_k; in order to analyse the worst case, we assume that both the tetrahedra projections give rise to quadrilaterals (made up by 4 triangles), therefore they generate at most 16 subpolygons each one with 6 edges. Every subpolygon P_k can then be triangulated in 4 triangles to which the integration formula can be applied.

Once the detection and the effective computation of the intersecting polygon P are joint in a single step, the only deficiency of the presented algorithm is that not every generated direction contributes to the summation. In the next section we show how it is possible to overcome this drawback.

3.4 Null-contribution direction

We can improve on the basic idea of algorithm 3.1 by filtering out efficiently the directions for which the contribution to the summation is zero. This also make it possible to take the detection step out of the algorithm.

In order to filter out null-contribution directions, we determine a spherical polygon SP over the unit sphere such that the points inside SP represent the only directions with effective contribution.

Definition 3.4.1 *An* extremal separating plane ϑ *for simplicies* S_1 *and* S_2 *is a plane which spans a vertex of* S_1 *and an edge of* S_2*, or vice versa, and such that* S_1 *and* S_2 *lie in opposite half-spaces generated by* ϑ*.*

Let us call *positive* the half-plane containing the simplex S_1. We compute the set of extremal separating planes and we translate them over the origin. The intersection of the shifted positive half-spaces gives birth to a cone C^+ with apex in the origin. The intersection between C^+ and the unit sphere produces the spherical polygon SP. It is then possible to create uniformly distributed sampling points in its interiors by triangulating it and using the algorithm suggested in [Arv95].

3.5 Gauss adaptive

In this section we discuss an adaptive method specifically designed to deal with the kernel function $K(u)$ which arises in the computation of the electrostatic force acting between two charged triangles in the tridimensional space.

Given two input triangles T_1, T_2, and a direction u, P is the polygon obtained by the intersection of the projections of T_1 and T_2 on a plane orthogonal

to u. Let us call $Vert(u) = \{V_0, \ldots, V_k\}$, the collection of the vertices of such polygon P in counterclockwise order. As we told in subsection 2.3.4, the basic idea of the Gauss adaptive method is to divide the plane φ-θ (it is equivalent to divide the unit semi-sphere U) into regions such that in each region $Vert(u)$ is combinatorially invariant. Changing u continuously, $Vert(u)$ changes from a combinatorial configuration to another when the projections of three triangles' vertices are collinear. The locus of points of the unit sphere Ω for which three specified vertices are collinear is a *transition curve*. A transition curve is a great circle on Ω. The set of great circles obtained by all the triples of vertices induces an arrangement on U and every zone of such an arrangement corresponds to a set of directions for which $Vert(U)$ is combinatorially invariant (see [Pel97] for details).

3.5.1 Great circles

In order to establish the arrangement of the φ-θ plane, we have to compute all the great circle induced by the triangles' vertices. A great circle can be univocally identified with a vector orthogonal to the plane spanned by the circle. Given three vertices V_1, V_2, V_3, they determine a plane $\mathcal{F}$ and its normal w. The directions u such that the projections of the vertices V_1, V_2, V_3 are collinear, are those for which holds:

$$w \cdot u = 0$$

that is the great circle identified by w. Writing u by means of its components φ and θ (fig. 2.2):

$$(\cos\theta)(w_x \cos\varphi + w_y \sin\varphi) + w_z \sin\theta = 0,$$

and solving with respect to θ we get:

$$\tan\theta = \left(-\frac{w_x}{w_z}\right)\cos\varphi - \left(-\frac{w_y}{w_z}\right)\sin\varphi. \tag{3.5.1}$$

If the plane determined by V_1, V_2 and V_3 is not vertical, the former equation becomes:

$$\theta = \arctan(-H\cos\varphi - K\sin\varphi), \quad \text{where} \quad \begin{cases} H = \frac{w_x}{w_z} \\ K = \frac{w_y}{w_z} \end{cases}$$

So the curve can be totally determined by the coefficients H and K. If $w_z = 0$, i.e. the plane $\mathcal{F}$ is vertical, the great circle is a vertical line on the φ-θ plane and it is identified by the angle formed by $\mathcal{F}$ and the plane x-z.

3.5.2 The algorithm

We have seen how it is possible to express all the great circles that form the arrangement of U and consequently of the plane φ-θ. In order to identify every zone, we still need to compute the intersection points among the great circles. The intersection between two curves is easily found by equating the right hand sides of formula 3.5.1 for the two curves and solving by making $\tan\varphi$ explicit. Thus the steps performed by the algorithm are:

- Computation of all the great circles. Since we want to integrate on the square $[-\frac{1}{2}\pi, \frac{1}{2}\pi] \times [-\frac{1}{2}\pi, \frac{1}{2}\pi]$, we also add the vertical lines $\varphi = -\frac{\pi}{2}$, $\varphi = \frac{\pi}{2}$ and the horizontal lines $\theta = -\frac{\pi}{2}$ and $\theta = \frac{\pi}{2}$.

- Computation of all the intersections among the lines defined in the former step.

- Taken two non-vertical lines $\mathcal{C}_1$ and $\mathcal{C}_2$ which vertically define a zone and two intersection points int_1 and int_2, which horizontally define a zone, we create n Gaussian nodes g_i^φ and weights h_i^φ, $(i = 0, \ldots, n-1)$, between int_1 and int_2 thanks to the formula 2.3.1.

- For each g_i^φ we determine the intersection points $vint_1$, $vint_2$ between the vertical line $\theta = g_i^\varphi$ and the curves $\mathcal{C}_1$, $\mathcal{C}_2$. Then n Gaussian points g_j^θ, and weights w_j^θ are generated in the range $[vint_1, vint_2]$.

- The approximation of integral 2.2.9 is obtained by

$$\sum_{i,j=0}^{N-1} w_i^\varphi \, w_j^\theta \, \cos(g_j^\theta) K(u(g_i^\varphi, g_j^\theta)).$$

Here we show the zones (fig. 3.9) and the sampling points (fig. 3.10) for the case of the triangles $T_1 = \{\{0, 0, -1\}, \{0, 0, 1\}, \{0, -1, 0\}\}$ and $T_2 =$

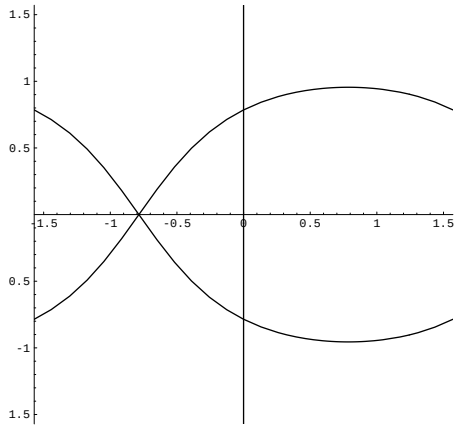

Figure 3.9: Division of the φ-θ plane in zones such that the names of vertices of the intersection polygon P are constant

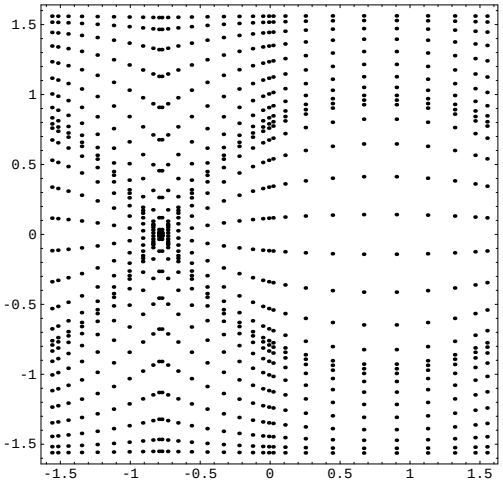

Figure 3.10: Sampling points for the Gauss adaptive algorithm

$\{\{0,0,-1\},\{0,0,1\},\{1,0,0\}\}$; i.e., they share the edge $\{\{0,0,-1\},\{0,0,1\}\}$.

The number of zones only depends on the six vertices and it is then a constant number. If the triangles share one edge the plane is divided in seven zones but only four give actual contribute to the approximation. In order to avoid the computation of useless direction it is enough to compute just one direction $\overline{u}$ for each zone: if the projections of the triangles intersect on a plane orthogonal to $\overline{u}$, then the all the directions within such zone bring actual contribute. Otherwise the zone is skipped. In figure 3.11 we illustrate the division in zones of the plane for two triangles sharing one vertex: $T_1 = \{\{0,0,0\},\{0,0,1\},\{1,0,0\}\}$, $T_2 = \{\{0,0,0\},\{0,1,0\},\{0,1,1\}\}$; in figure 3.11 we show the sampling points created only within zones which give actual contribution.

3.6 Numbers representation

Computers, being finite machines, represent real numbers with a finite sequence of digits. In consequence it is not possible to exactly store numbers

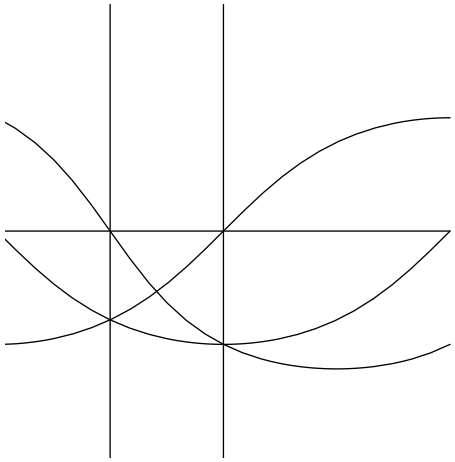

Figure 3.11: Triangles sharing one vertex: division of the φ-θ plane

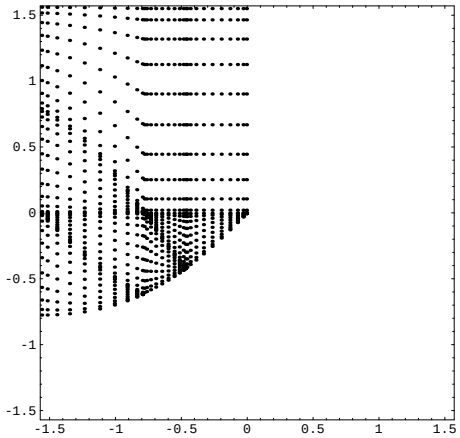

Figure 3.12: Sampling points for the Gauss adaptive algorithm

with an infinite extension such as π or $\sqrt{2}$. The same problem occurs with rational numbers whose extended representation is periodical ($1/3 = 0.3333\ldots$). Besides with a finite number of digits, only a finite set of numbers can be represented; if numbers too big or too small are involved in a computation, they can cause the calculation process to be stopped as a situation of *overflow* or *underflow* has been detected. In general, given a number d it is internally represented by a computer with a machine number $\tilde{d}$ which approximates d. The difference $|d - \tilde{d}|$ depends on how numbers are stored in that particular computer; internal representations differ on parameters such as the number of digits dedicated to a number and the base used to represent numbers.

Since a number can only have finite extension, even the arithmetic is approximated (*finite arithmetic*) and new errors are introduced by the execution of arithmetical operations. A possible remedy to the error brought by the finite arithmetic is the symbolic computation: instead of representing π by truncating its value $3.141592\ldots$, it is stored as the symbol π. Then the computation will take place algebraically rather numerically, maybe obtaining expressions in place of numbers. The final value comes from the evaluation of the outcoming expression. We used this technique whenever possible, but every time it is requested a value in order to decide the future steps of an algorithm, the

approximate arithmetic must be used.

It is now clear that expressions such as:

if $|val| = 0$
then ...

or

if $val_1 > val_2$
then ...

have different semantic with respect to the same expression used in contexts with exact arithmetic and exact number representation. It is then necessary to set a real number ϵ which will represent the range of acceptable values:

Mathematical semantic	**Finite arithmetic semantic**
if $val = 0$ **then** ...	**if** $\lvert val\rvert < \epsilon$ **then** ...
if $val = k$ **then** ...	**if** $(val < k + \epsilon) \wedge (val > k - \epsilon)$ **then** ...
if $val > k$ **then** ...	**if** $val > k - \epsilon$ **then** ...

The range ϵ is somehow a bound to the precision of the algorithm too. In our algorithm ϵ plays an important role in the `left_turn` function, in particular in testing whether three points are collinear or not; in our implementation we set $\epsilon = 10^{-12}$. Another crucial point is in the computation of the kernel function when the input bodies are triangles (2.2.8): mathematically the value for K can not be derived from formula (2.2.8) when the factor $\cos\psi(u)$ is zero; working with finite arithmetic we have that the formula can not be used if $\cos\psi(u)$ is zero or a value very near to zero. For the experiments we set the range for $\cos\psi(u)$ to be zero as $]-10^{-7}, 10^{7}[$ avoiding all the special cases.

If people say I'm crazy,
I tell 'em that it's true
Let them watch with amazement
I say it's most unusual
I wouldn't normally do this kind of thing.

Neil Tennant

Chapter 4

Experiments on volume to volume integrals

In this chapter we take in consideration the case of volume-to-volume integrals (2.2.5); the input bodies are tetrahedra placed in critical positions. We describe the experiments made comparing the methods introduced so far. We report the results as tables and graphics.

4.0.1 Computing Equipment

The experiments have been made on a Pentium 100 processor with 64Mb of Ram; the operating system was Linux running the X-Windows graphical user interface. All the algorithms have been implemented in C language. The reference values have been computed by Mathematica (version 2.2.2) run under the Windows '95 operating system.

4.1 Inputs

We made three sets of experiments, considering critical tetrahedra positions: T_1 will always be the 'unit' tetrahedron: $\{\{0,0,0\},\{1,0,0\},\{0,1,0\},\{0,0,1\}\}$; in the first set of experiments we take in consideration the case in which T_1 and T_2 share a face: T_2 is the symmetric of T_1 respect to the face $\{\{0,0,0\},\{0,1,0\},\{0,0,1\}\}$, (fig. 4.1). In the second experiments, T_2 is: $\{\{0,0,0\},\{-1,0,0\},\{0,1,0\},\{0,0,-1\}\}$, so the tetrahedra share the edge

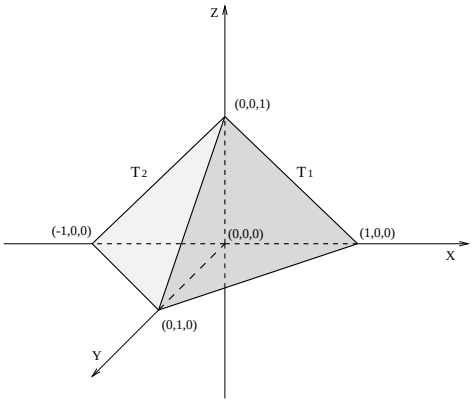

Figure 4.1: T_1 and T_2 share a face

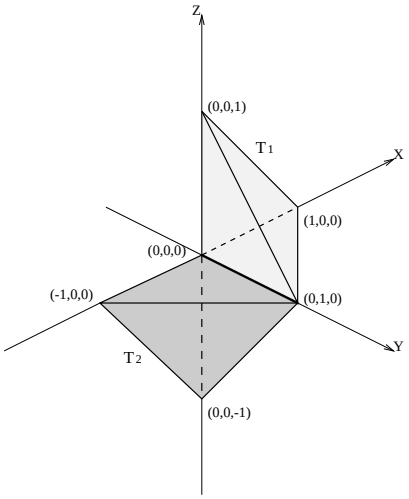

Figure 4.2: T_1 and T_2 share one edge

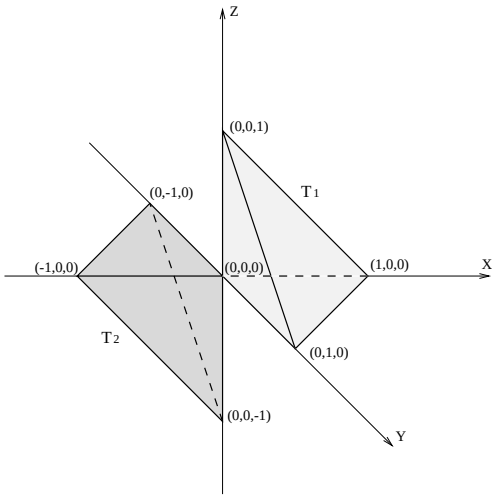

Figure 4.3: T_1 and T_2 share one vertex

$\{\{0,0,0\},\{0,1,0\}\}$, (fig. 4.2). In the third set the tetrahedron T_2 is symmetrical to T_1 with respect to the origin: $\{\{0,0,0\},\{-1,0,0\},\{0,-1,0\},\{0,0,-1\}\}$, (see fig. 4.3), thus tetrahedra T_1 and T_2 share the $\{0,0,0\}$ vertex.

In regard to N, the number of directions to be generated (table 3.1), we run the algorithms with $n_0 = 10, \ldots, n_i, \ldots, n_{49} = 10,000$ directions where n_i, $(i = 0, \ldots, 49)$, are such that they are equally spaced in the logarithmic scale. We have that:

$$n_{i+1} = \lceil n_i \, e^{\Delta} \rceil \, , \text{where } \Delta = \frac{\log 10,000 - \log 10}{50},$$

so $n_0 = 10, n_1 = 12, n_2 = 14, \ldots$. Because of the ceiling function the last step $n_{48} - n_{49}$ doesn't match the definition, since n_{49} would be greater than $10,000$.

4.2 Algorithms

Here follows the list of the algorithms involved in the experiments.

4.2.1 Monte Carlo (MC)

This is the basic algorithm, the outcome is just the mean value of the contributions of N directions; its quality, especially for big values of N, depends greatly on the random number generator. For this we used both the algorithms (Knuth and Trig) shown in Appendix A.1 and a third one; here we present only the results derived from the Knuth's algorithm since we did not notice considerable differences among them. For the sake of the uniformity of randomness, we did not keep trace of the former results, calculating n_i new directions for each i.

We also show results for the *Monte Carlo with median* (MC-med) algorithm: instead of a plain mean out of N directions, we take as result the median of 10 experiments run with $\lfloor \frac{N}{10} \rfloor$ directions. As explained in subsection 2.3.2, this method should asymptotically permit to reach the same precision ϵ achievable by the pure *MC* method but using less sampling points.

In the second and third set of experiments, not every direction brings an actual contribution. We used the technique described in section 3.4 in order to avoid the useless directions. Thereafter we refer to this method as the *MC-cone* method.

4.2.2 Quasi Monte Carlo (QMC)

Since the values of N are relatively small, and we are not interested in reusing the evaluations of the integrand functions, we used the Hammersley point set to produce sampling points; it is possible to gain something in terms of speed, and probably reducing a bit the likelihood of error, implementing sequences.

We also report the tests made with 'grid' sampling sets; as explained in subsection 3.1.1, the grids are built upon the constriction that the number of points on the φ edge of the grid must be four times the number of the points on θ edge. This affects the number of sampling points, often greater than the fixed N, so if N is small ($N < 250$), we can get the same set of grid points (and consequentially the same results) for different values of N. We do not show the results obtained with square grids (same number of points on both the axes) since the results are substantially the same of the 'rectangular' grids.

As for the Monte Carlo methods, we implemented the direction reduction (subsection 3.1.1), in order to filter out null-contribution direction. For the generation of sampling points over a spherical polygon with other methods than the uniform distribution, we used again the isomorphism from the unit sphere to the (φ, θ) plane, transforming a spherical polygon to a planar zone Z and producing sampling points inside Z. In our experiments it is immediate to apply the isomorphism: in the second set of experiments the cone if formed by all the points with x and z coordinate greater or equal to zero, thus Z is $\varphi \in [0, \pi], \theta \in [0, \pi/2]$. In the first set of experiments the cone corresponds to the spherical triangle whose vertex are $\{\{1, 0, 0\}, \{0, 1, 0\}, \{0, 0, 1\}\}$, and the resulting zone Z is $\varphi \in [0, \pi/2], \theta \in [0, \pi/2]$.

4.2.3 Gauss (non adaptive)

For this method we did not fix the integer N as explained in section 4.1, we just used Gaussian square grids over the rectangle $[0, 2\pi] \times [0, \pi/2]$ (see subsection 3.1.1). A Gaussian grid is the product of two Gaussian formulas; they are squared because we took the same number of nodes for both axes. The nodes and the weights are pre-computed, this means that the algorithm itself is faster but it needs a pre-processing time. The biggest grid is $45 * 45 = 2025$ directions because of problems of numerical instability that arise starting from the $43 * 43$ grid.

4.3 Computation of reference values

Given the problem of computing the electrostatic force acting between two polyhedra, it is extremely rare to find analytical solutions. We did not find analytic solutions for any of our experiments, neither for the integrals derived from the geometrical interpretation nor for those derived from the classic theory. Having to compare methods based on a new mathematical model, we decided to get the reference value through long simulations starting from the integrals coming from Coulomb's law:

$$\iiint_{T_1}\iiint_{T_2} \frac{dx_1\,dy_1\,dz_1\,dx_2\,dy_2\,dz_2}{(x_2-x_1)^2+(y_2-y_1)^2+(z_2-z_1)^2}\cos\varphi\,\cos\theta, \tag{4.3.1}$$

where T_1 and T_2 are the input tetrahedra, $p_1 = (x_1, y_1, z_1) \in T_1, p_2 = (x_2, y_2, z_2) \in T_2$, θ is the angle that the line $\overline{p_1p_2}$ creates with the plane x-y, and φ is the angle formed by the x axis and the line $\overline{p_1'p_2'}$, being $\overline{p_1'p_2'}$ the projection of $\overline{p_1p_2}$ on the x-y plane. Then expressing φ and θ in terms of p_1 and p_2, formula (4.3.1) becomes:

$$\iiint_{T_1}\iiint_{T_2} \frac{|x_2-x_1|}{[(x_2-x_1)^2+(y_2-y_1)^2+(z_2-z_1)^2]^{\frac{3}{2}}}dx_1\,dy_1\,dz_1\,dx_2\,dy_2\,dz_2.$$

Thinking to the dimension of the integral, the integration domain and the ease of implementation, we chose the Monte Carlo method. In order to produce random points we used the internal random number generator (`Random[Real, {0,1}]`, see [Wol93] for details) of the Mathematica software (version 2.2.2) run under the Microsoft Windows '95 operating system. We create a random point in $\mathbb{R}^6$, by a pair of 3-coordinates. Each triple of coordinates is tested for inclusion in the input tetrahedra. A 6-coordinates point is useful for the computation if both the triples represent points inside the tetrahedra. The final value is the mean of 60 computations of 150,000 useful 6-dimensional points (in total about 36 hours of computing time). Table (4.1) shows the reference values. Passing from tetrahedra sharing one vertex to tetrahedra sharing one face we noticed that the values of the 60 computations were increasingly discordant in the first digits. Our experience suggests the first value (tetrahedra sharing a face) is precise for about 4 digits, the second value for 5 digits and the third value (tetrahedra sharing one vertex) is precise for about 6-7 digits.

Experiment	Reference Value
One vertex shared	0.01817798402479134
One edge shared	0.03465072761418757
One face shared	0.0870330066795285

Table 4.1: Reference values for volume-to-volume experiments

4.3.1 Algorithms derived from Coulomb's law

In the plots of section 4.5, we also report results about two algorithms derived from Coulomb's law. The first one is based on the Monte Carlo method while the second one is based on the Quasi Monte Carlo method. The Monte Carlo Coulomb (MCC) algorithm is basically the translation in "C" of the algorithm used to obtain the reference values; the C-program is much faster than the Mathematica one but it is not suitable for huge computations such as those done for the reference values, because of numerical reasons. In the Quasi Monte Carlo Coulomb (QMCC) algorithm, the 3-dimensional points $p_1 = (x_1, y_1, z_1)$ and $p_2 = (x_2, y_2, z_2)$, are generated according to the Hammersley Point set of dimension 6 (see Appendix A.2):

$$(p_{1_n}, p_{2_n}) = \left(\left\{ \frac{n}{N}, \phi_2(n), \phi_3(n) \right\}, \{\phi_5(n), \phi_7(n), \phi_{11}(n)\} \right)$$

for $n = 0, 1, \ldots, N-1$. Following the rule shown in section 4.1, the algorithms are run with $n_0 = 10, \ldots, n_{49} = 10000$, 6-dimensional points (formed by two tridimensional points), provided they are internal to the 6-dimensional integration domain.

4.4 Tables

In this section we show in tabular form experimental results for tetrahedra sharing one face (subsection 4.4.1, fig. 4.1), one edge (subsection 4.4.2, fig. 4.2) and one vertex (subsection 4.4.3, fig. 4.3). The methods are compared on the basis of the relative error with respect to the values derived from equation 4.3.1 (table 4.1). In each subsection the first table gives the precision (relative error) attained after a certain elapsed time from the beginning of the computation. The second table gives the relative error as function of the number of direction

N used to get the approximation. In general given a number of direction N and a method, the corresponding value is not obtained with exactly n direction but with a n' near to n. Data of this section is a significant sample of the those used to produce plots shown in section 4.5.

4.4.1 Tetrahedra sharing one face

sec.	**MC**	**QMC**	**Gauss**
0.02	-	-	0.9095636884
0.05	0.2172960317	0.3464714943	0.7608771551
0.1	0.2190223237	0.1843529755	0.2206250663
0.3	0.1754300267	0.0496682069	0.0540405414
0.5	0.1524753308	0.0298159267	0.0341402263
1	0.1142680815	0.0128046662	0.0195223527
3	0.1555730073	0.0039306736	0.0062803643
5	0.0824792578	0.0023132319	0.0087351927
10	0.1334911976	0.0013137913	0.0033960990
30	0.0757315219	0.0006984604	-

Table 4.2: Tetrahedra sharing one face. Number of seconds of computation/Relative error

N	**MC**	**MC-med**	**Grid**	**QMC**	**Gauss**
10	0.2172960317	0.4797263377	1.1388556434	0.4350593236	0.7608771551
25	0.2972954121	0.1648044525	0.6625503084	0.1536621516	0.2536975336
50	0.1187496594	0.0386984693	-	0.0591742188	0.1397661223
75	0.0720437750	0.0507212273	0.4655621182	0.0352331554	0.0540405414
100	0.1524753308	0.1902002872	0.3580519776	0.0251656549	0.0341402263
250	0.0216468265	0.1683448218	0.2122399258	0.0095377367	0.0146212383
500	0.0112126087	0.0849674291	0.1509617357	0.0044767558	0.0076053915
750	0.1279990722	0.0310393884	0.1171930965	0.0029904976	0.0113857701
1000	0.1226292590	0.0230988684	0.1020022204	0.0023132319	0.0035086255
2500	0.1086360147	0.0158508316	0.0672515715	0.0010836465	-
5000	0.1239731383	0.0059034661	0.0445899522	0.0007431816	-
7500	0.1192700585	0.0093141262	0.0364466665	0.0006290607	-
10000	0.1126237021	0.0181955552	0.0320705985	0.0005778573	-

Table 4.3: Tetrahedra sharing one face. Comparison among all the methods: Number of directions/Relative error

4.4.2 Tetrahedra sharing one edge

sec.	**MC**	**QMC**	**QMC-cone**	**Gauss**
0.02	-	-	-	0.8487815315
0.03	-	-	-	0.0244223926
0.05	0.1870173865	0.4350259116	-	0.0751396871
0.1	0.1979278780	0.2394331843	0.0550902093	0.0563635003
0.3	0.3403607613	0.0722383277	0.0114031604	0.0159342981
0.5	0.0093819657	0.0406854481	0.0073581006	0.0007218235
1	0.0047283428	0.0141363719	0.0024682679	0.0036420661
3	0.1193026605	0.0035581338	0.0003300526	0.0009837747
5	0.0696042435	0.0017446695	0.0000338936	0.0001165337
10	0.0005430197	0.0007049249	0.0001455872	-
30	0.0288792734	0.0003036743	0.0002135612	-

Table 4.4: Tetrahedra sharing one edge. Number of seconds of computation/Relative error

N	**MC**	**MC-med**	**MC-cone**	**Grid**	**QMC**	**QMC-cone**	**Gauss**
10	0.1870173	0.5420063	0.2878292	0.2172313	0.5626566	0.1126332	0.0244223
25	0.2027878	0.3901404	0.2332866	0.0803293	0.2124278	0.0333664	0.0563635
50	0.1995851	0.1491213	0.1135433	-	0.0854972	0.0114031	0.0159342
75	0.1094424	0.0298665	0.1034774	0.0685767	0.0539134	0.0073581	0.0007218
100	0.0093819	0.0213865	0.0042243	0.0274327	0.0406854	0.0055140	0.0057290
250	0.1859330	0.1517546	0.0518835	0.0301330	0.0120186	0.0013957	0.0019957
500	0.0370110	0.0780963	0.0015015	0.0057558	0.0042811	0.0003300	0.0009837
750	0.0314281	0.0197617	0.0432198	0.0149387	0.0024936	0.0000862	0.0003512
1000	0.0696042	0.0221787	0.0068535	0.0143108	0.0017446	0.0000163	0.0004992
2500	0.0044063	0.0680709	0.0159748	0.0091680	0.0005452	0.0001833	-
5000	0.0431573	0.0315603	0.0106629	0.0063938	0.0003233	0.0002135	-
7500	0.0261014	0.0087121	0.0162534	0.0006228	0.0002766	0.0002198	-
10000	0.0121362	0.0108387	0.0011991	0.0042378	0.0002580	0.0002223	-

Table 4.5: Tetrahedra sharing one edge. Comparison among all the methods: Number of directions/Relative error

4.4.3 Tetrahedra sharing one vertex

sec.	**MC**	**QMC**	**QMC-cone**	**Gauss**
0.02	-	-	-	0.2799442018
0.03	-	-	-	0.7501879952
0.05	0.5313319381	0.3334888986	-	0.1369718856
0.1	0.4717037973	0.2260358654	0.1053087389	0.2214823012
0.3	0.1222689216	0.0806854301	0.0348917374	0.0772082578
0.5	0.1582587477	0.0475938392	0.0139023127	0.0262061191
1	0.0168448088	0.0163410169	0.0049841196	0.0174635800
3	0.1722930268	0.0040431211	0.0008787299	0.0000117925
5	0.1373738458	0.0019280281	0.0004419166	-
10	0.0083889438	0.0006127605	0.0001400457	-
30	0.0480796788	0.0000986040	0.0000165361	-

Table 4.6: Tetrahedra sharing one vertex. Number of seconds of computation/Relative error

N	**MC**	**MC-med**	**MC-cone**	**Grid**	**QMC**	**QMC-cone**	**Gauss**
10	0.5313319	1.0000000	0.1607906	0.5371114	0.3334888	0.1709842	0.7501879
25	0.1242159	0.3985034	0.0892234	0.2849726	0.2148558	0.0581635	0.2214823
50	0.0425790	0.6425200	0.0038246	-	0.0922418	0.0204937	0.0558888
75	0.1936358	0.2340174	0.0826289	0.1713175	0.0623974	0.0116481	0.0177179
100	0.1582587	0.0225219	0.0387520	0.1133529	0.0475938	0.0081398	0.0262061
250	0.1072913	0.1609050	0.0145239	0.0556409	0.0138959	0.0021143	0.0093056
500	0.0533968	0.1691089	0.0605032	0.0245930	0.0049142	0.0006827	0.0000117
750	0.1197336	0.0811849	0.0412531	0.0222233	0.0027854	0.0003571	-
1000	0.1373738	0.1080146	0.0194688	0.0193777	0.0019280	0.0002333	-
2500	0.0466104	0.0657747	0.0270633	0.0108110	0.0004131	0.0001400	-
5000	0.0082751	0.0015148	0.0006364	0.0069645	0.0001239	0.0000077	-
7500	0.0337709	0.0327048	0.0087242	0.0039133	0.0000628	0.0000008	-
10000	0.0394626	0.0220637	0.0011865	0.0041268	0.0000378	0.0000017	-

Table 4.7: Tetrahedra sharing one vertex. Comparison among all the methods: Number of directions/Relative error

4.5 Graphics

Here we present the results of the experiments as plots of the relative error against the number of directions and relative error against computation time expressed as number of seconds. All the plot are drawn in logarithmic scale for both the axis (log-log scale, base 10). In the next subsection we show comparisons between Monte Carlo and Monte Carlo with median methods (subsection 4.5.1), between Monte Carlo and Monte Carlo with filtering of useless directions (subsection 4.5.2), between Monte Carlo and Quasi Monte Carlo methods (subsection 4.5.3) and among all the algorithms based on the Quasi Monte Carlo method (subsection 4.5.4). In subsection 4.5.5 all the methods are grouped with respect to the relative error against directions while in subsection 4.5.7 they are compared with respect to the relative error against computation time. It must be noted that all the experiments related to randomness have been repeated several times and the figures shown represent the general behaviour.

4.5.1 Monte Carlo versus Monte Carlo with Median

As explained in subsection 2.3.2, the Monte Carlo method achieves a precision ϵ with probability $1-\delta$ choosing $O\left(\frac{1}{\epsilon^2\delta}\right)$ directions, while the MC-median method should attain the same results setting $N = O\left(\frac{\log(1/\delta)}{\epsilon^2}\right)$. Referring to our experiments, there is an improvement only for tetrahedra sharing a face (figure 4.4), while there is little difference between results obtained with a straight Monte Carlo and Monte Carlo with median algorithms in the other cases (figures 4.5, 4.6). This is probably due to two causes: the magnitude of N and the bound for the MC method. On a side the outcome of the median algorithm is derived from ten MC experiments made with $N = \lfloor \frac{N}{10} \rfloor$, so it relies on experiments made with $N = 1, \ldots, 1000$ directions. In that range the Monte Carlo method is still highly oscillatory; we presume better result can be achieved with bigger values for N. On the other side experimental results suggest that bound given for the MC method is too rough.

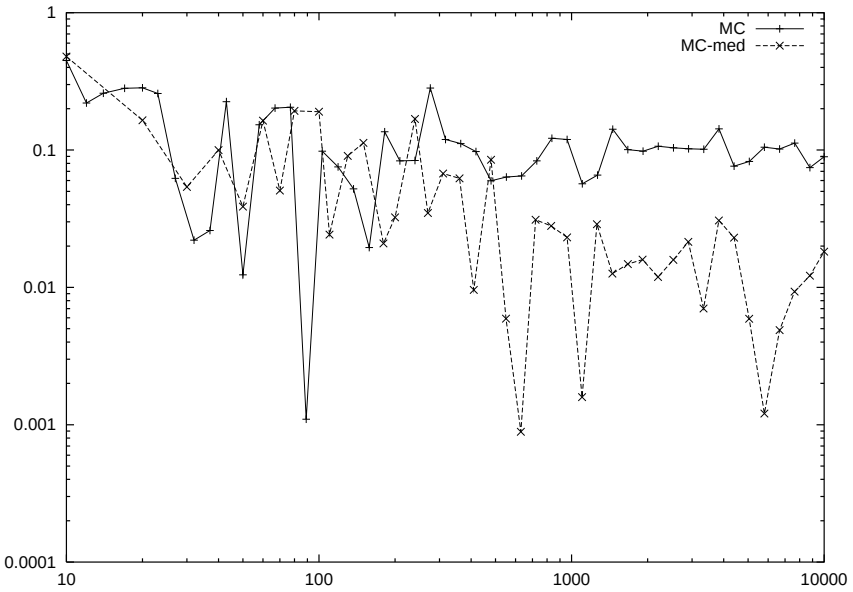

Figure 4.4: One face shared. Monte Carlo versus MC-median: Number of directions/relative error

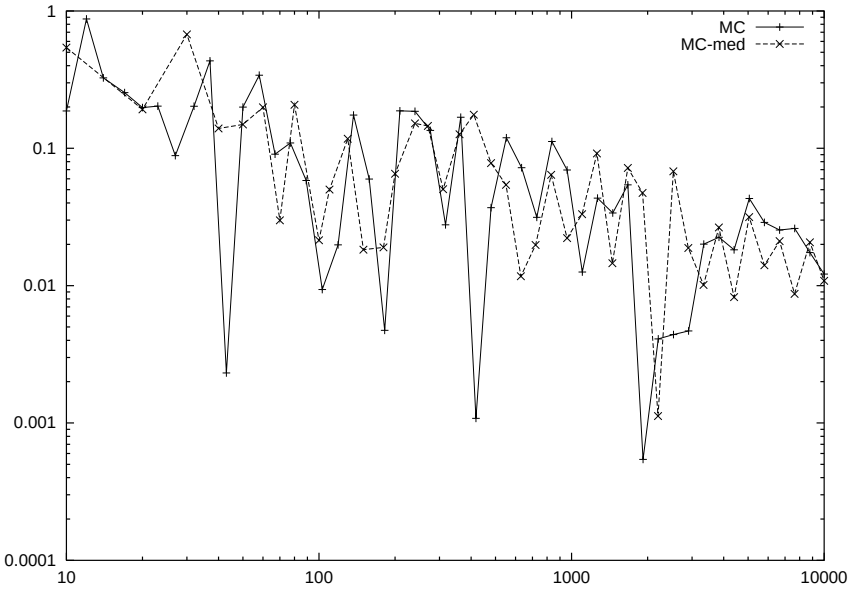

Figure 4.5: One edge shared. Monte Carlo versus MC-median: Number of directions/relative error

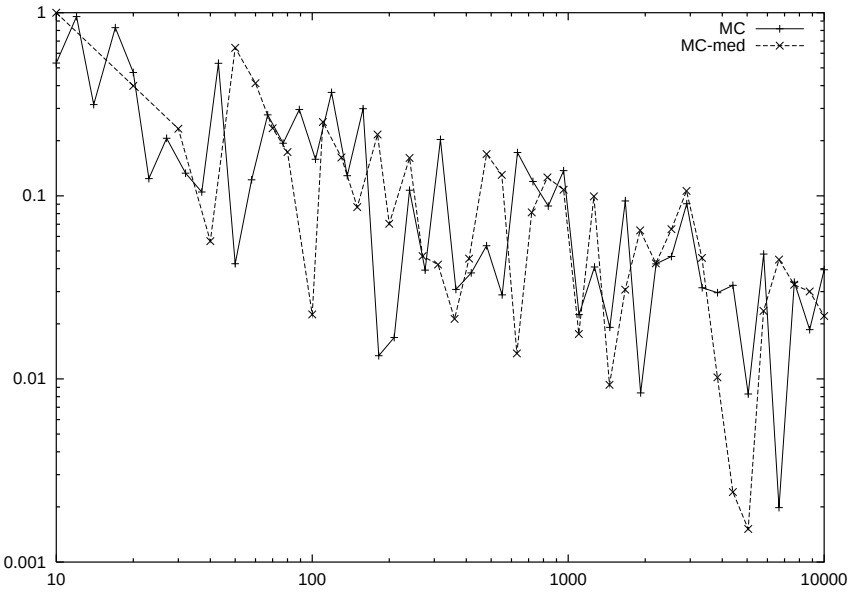

Figure 4.6: One vertex shared. Monte Carlo versus MC-median: Number of directions/relative error

4.5.2 Monte Carlo versus Monte Carlo cone

In the second and in the third set of experiments it is possible to implement the trick described in section 3.4, preventing the algorithm to generate useless sampling directions. Referring to figures 4.7 and 4.8, in spite of the high irregularities of the results (oscillations), it is clear that the MC-cone curve lays under (better approximation) the Monte Carlo one. This is utterly evident for tetrahedra sharing one vertex (figure 4.8), as without the computation of the cone only one direction out of four brings actual contribute.

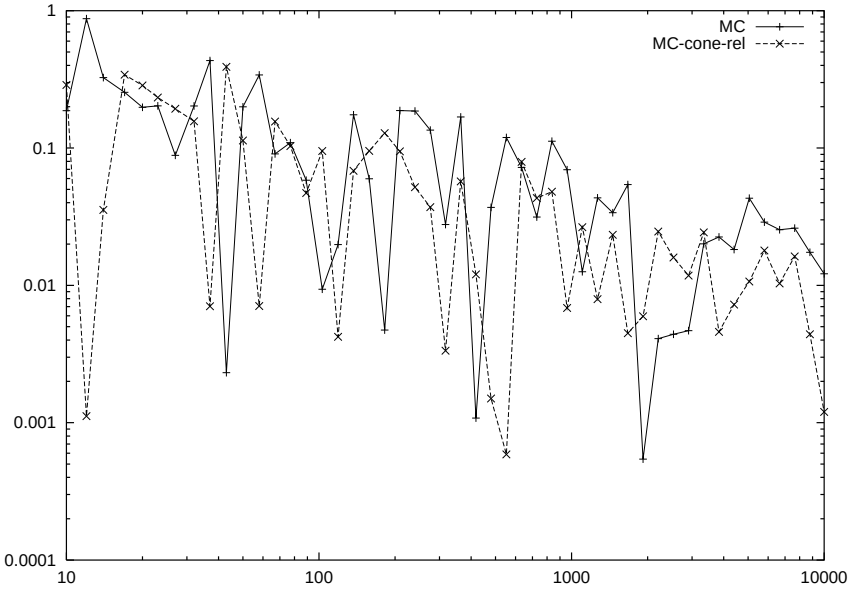

Figure 4.7: One edge shared. Monte Carlo versus MC-cone: Number of directions/relative error

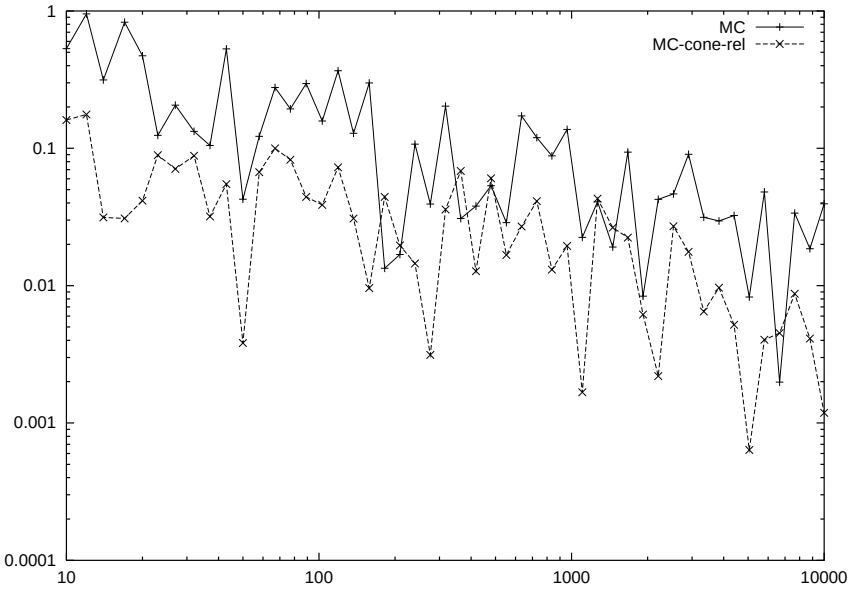

Figure 4.8: One vertex shared. Monte Carlo versus MC-cone: Number of directions/relative error

4.5.3 Monte Carlo versus Quasi Monte Carlo

The aim of the Quasi Monte Carlo method is to generate sampling points in a deterministic way in order to produce a set of points with properties similar to those of a random sets but preventing the irregularities of such sets (figures 3.2 and 3.3). Figures 4.9, 4.10 and 4.11, show that the curve obtained by a Quasi Monte Carlo approach are much smoother than those derived from the Monte Carlo approach and they are consistently lower. Besides they are non-increasing curves. The flattenings of the QMC curves starting from $N \approx 2000$ present in the cases of tetrahedra sharing one face and one edge, are probably due to the precision of the reference value; confirming the assumptions we made about the precisions of the reference values, the flattening appears at higher values of the relative error in figure 4.9 (one face shared) than in figure 4.10 (one edge shared), and it does not appear at all in figure 4.11 (one vertex shared).

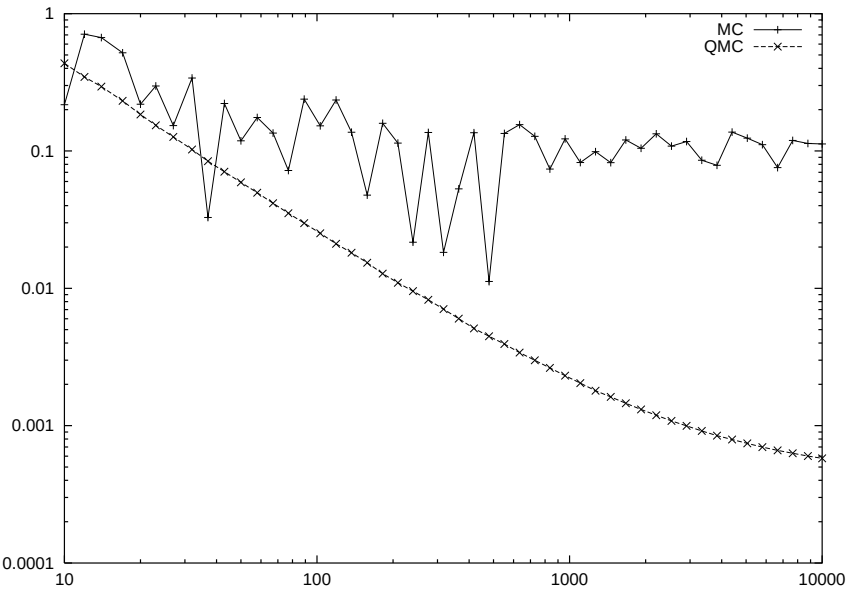

Figure 4.9: One face shared. Monte Carlo versus Quasi Monte Carlo: Number of directions/relative error

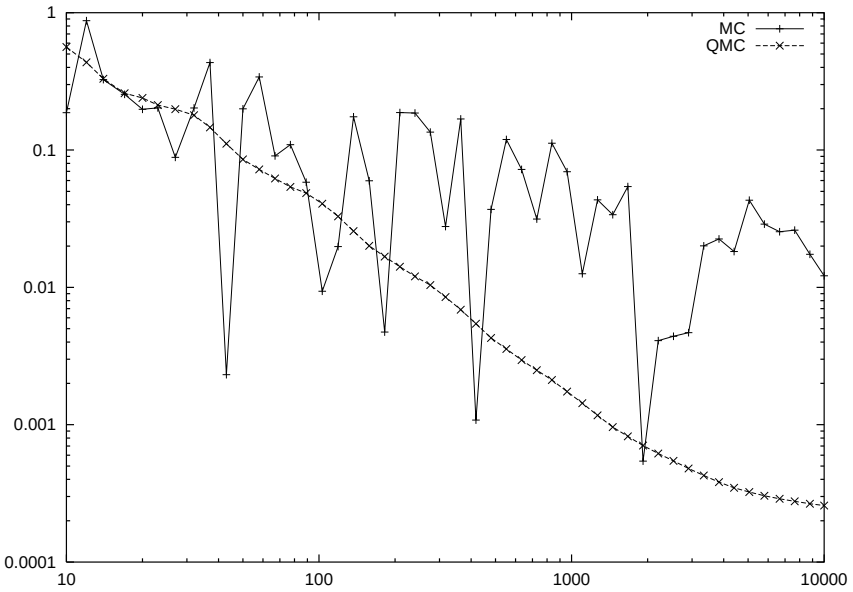

Figure 4.10: One edge shared. Monte Carlo versus Quasi Monte Carlo: Number of directions/relative error

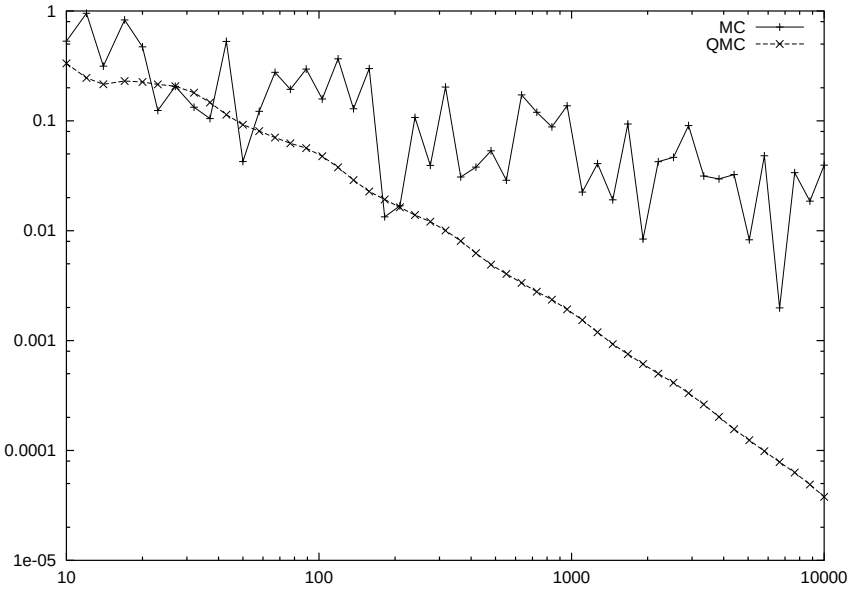

Figure 4.11: One vertex shared. Monte Carlo versus Quasi Monte Carlo: Number of directions/relative error

4.5.4 Quasi Monte Carlo methods

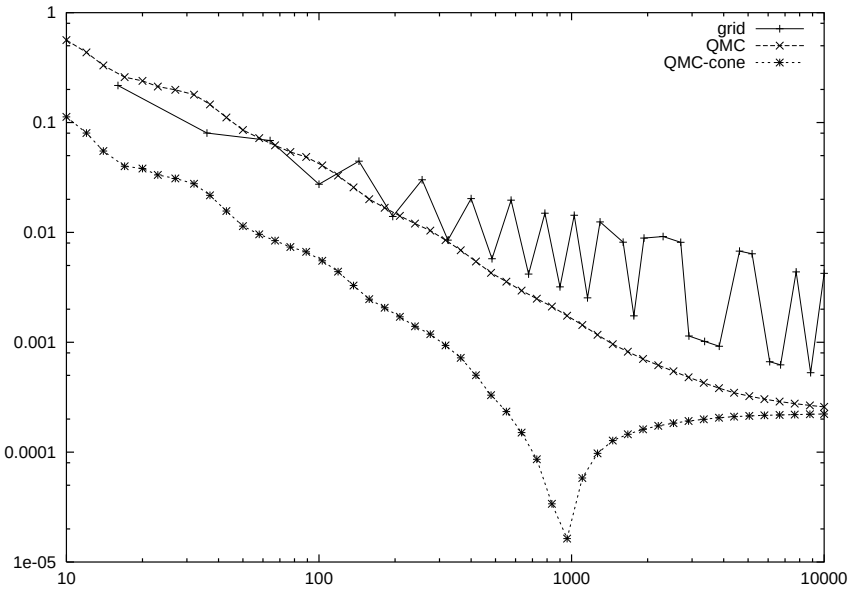

Figure 4.12: One edge shared. Algorithms based on the Quasi Monte Carlo method: Number of directions/relative error

The reduction of directions can be applied to the Quasi Monte Carlo method as well. In figures 4.12 and 4.13 we show the curves obtained by the Quasi Monte Carlo approach, by the same approach with the computation of the cone and by the grid approach. The latter is intended to show that even if the grids are the simplest Quasi Monte Carlo sets, they are not as good as Hammerley point sets (see appendix A.2) because of their high discrepancy (see subsection 2.3.3).

The curves obtained by the QMC-cone approach are manifestly lower than the QMC ones and in spite of the scale of the plots, the improvement of the QMC-cone algorithm with respect to the QMC algorithm is greater in the case of tetrahedra sharing one vertex. In both the experiments the QMC-cone curves presents a peak. We impute this behaviour to the reaching of the precision of the reference value. The peak in figure 4.12 appears about at 10^{-5} (relative error) and about at 10^{-6} in figure 4.13, confirming again our suppositions about the accuracy of the reference values. We interpret the

convergence of the QMC and QMC-cone curves to a unique value of the relative error in figure 4.12 as the convergence of these methods to the real value of the electrostatic force exerted by the tetrahedra. Note that even after the peak point, the QMC-cone method approaches the "convergence" point faster than the QMC one.

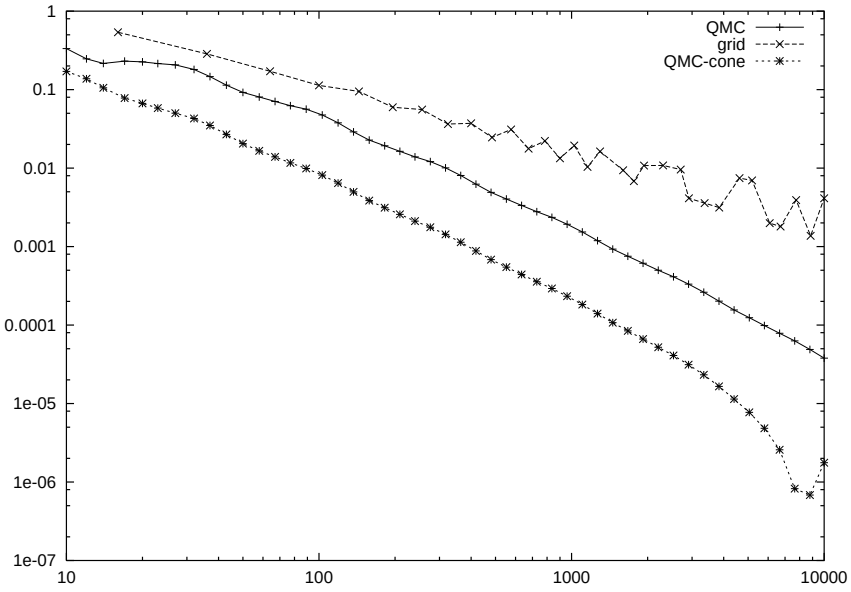

Figure 4.13: One vertex shared. Algorithms based on the Quasi Monte Carlo method: Number of directions/relative error

4.5.5 All methods

In figures 4.14, 4.15 and 4.16 we report the results for all the methods, including the curves obtained with the Gauss algorithm (in figures 4.15 and 4.16 we excluded the MC-med curves). In general the Quasi Monte Carlo (possibly with cone computation) approach attains the best results followed by and the Gauss method; Quasi Monte Carlo method is also the most reliable giving rise to monotone proceedings. All the other methods are evidently less effective and only occasionally reach results better than the QMC algorithm.

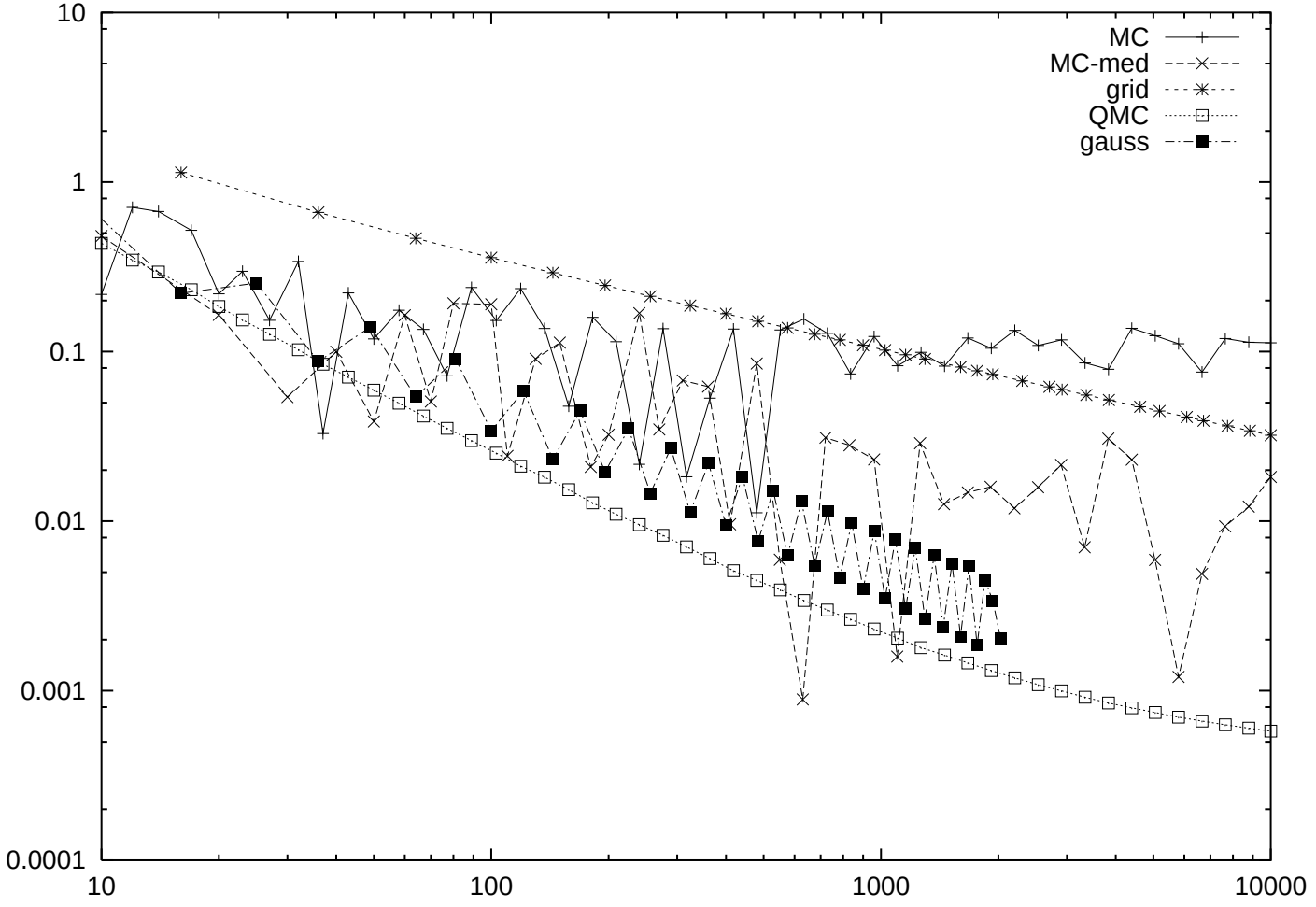

Figure 4.14: One face shared. Comparison among all the methods: Number of directions/relative error

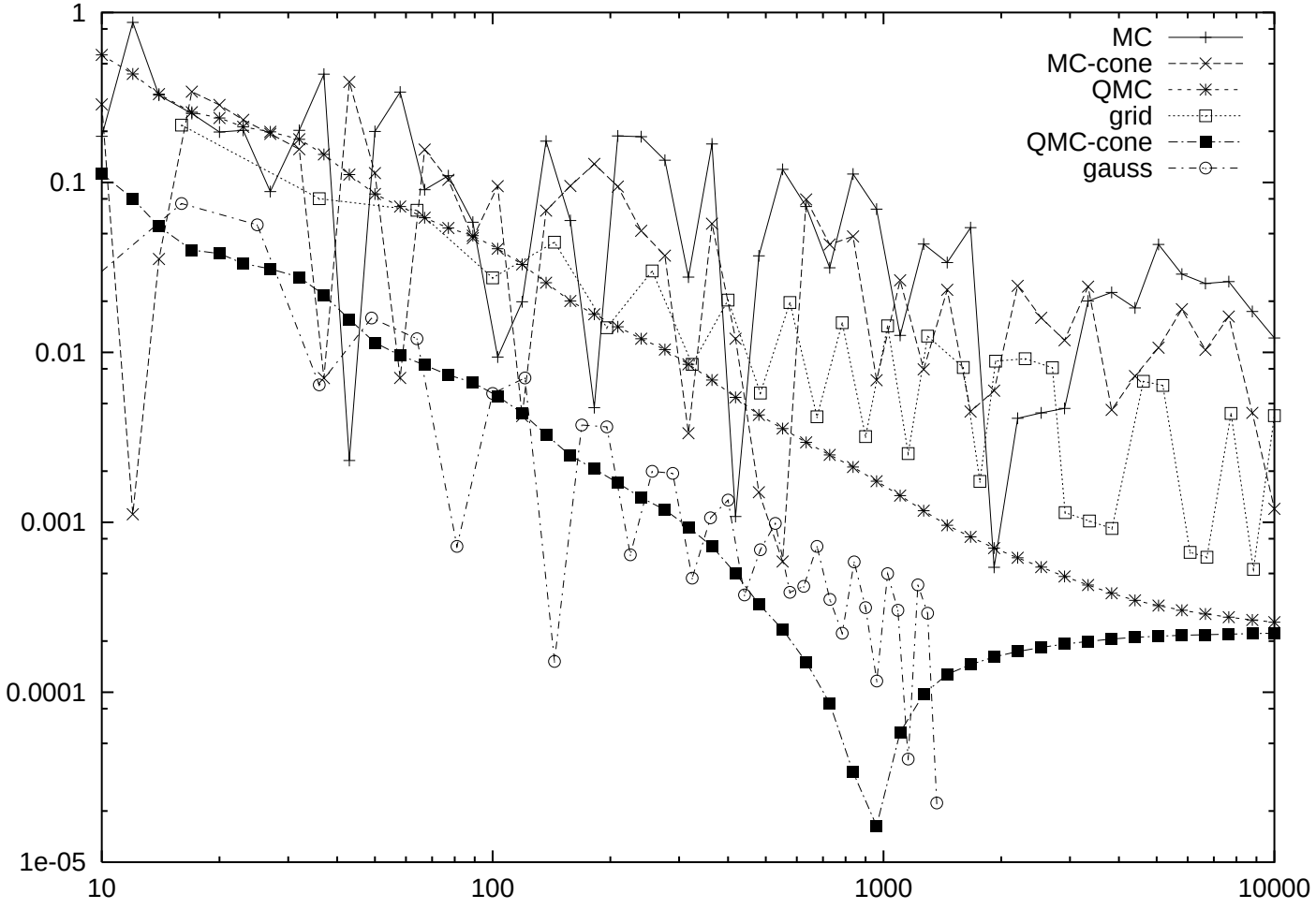

Figure 4.15: One edge shared. Comparison among all the methods: Number of directions/relative error

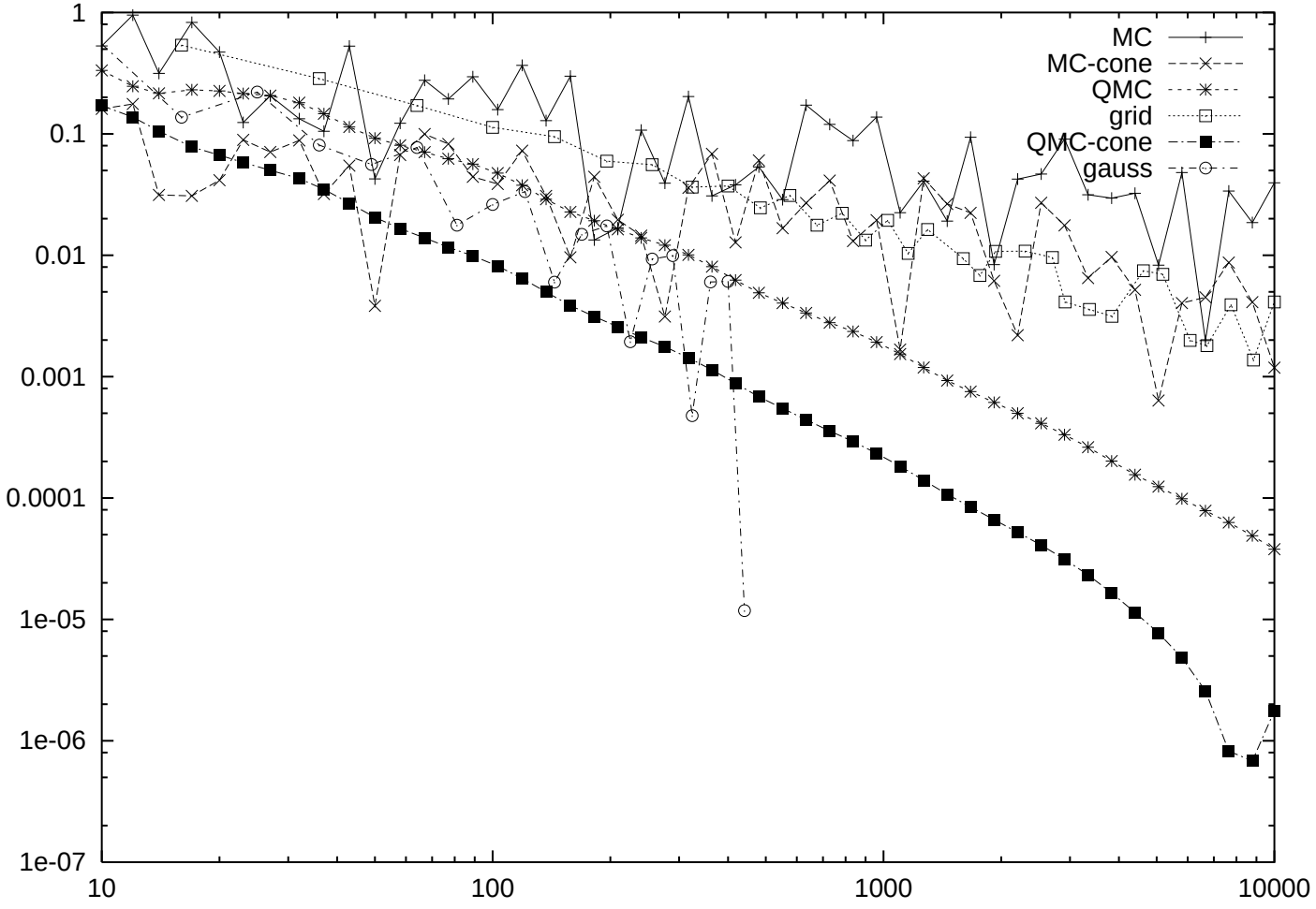

Figure 4.16: One vertex shared. Comparison among all the methods: Number of directions/relative error

4.5.6 Monte Carlo Coulomb versus Quasi Monte Carlo Coulomb

In the light of the improvements brought by the Quasi Monte Carlo approach with respect to the Monte Carlo method, we decided to verify whether it happens the same with the algorithms derived from Coulomb's law. The plots 4.17, 4.18 and 4.19 show the results of the Monte Carlo Coulomb and Quasi Monte Carlo Coulomb algorithms. Both the curves have non-smooth behaviour. It is also evident the raising of the relative error as the dimension of the singularity increases (point, line, face).

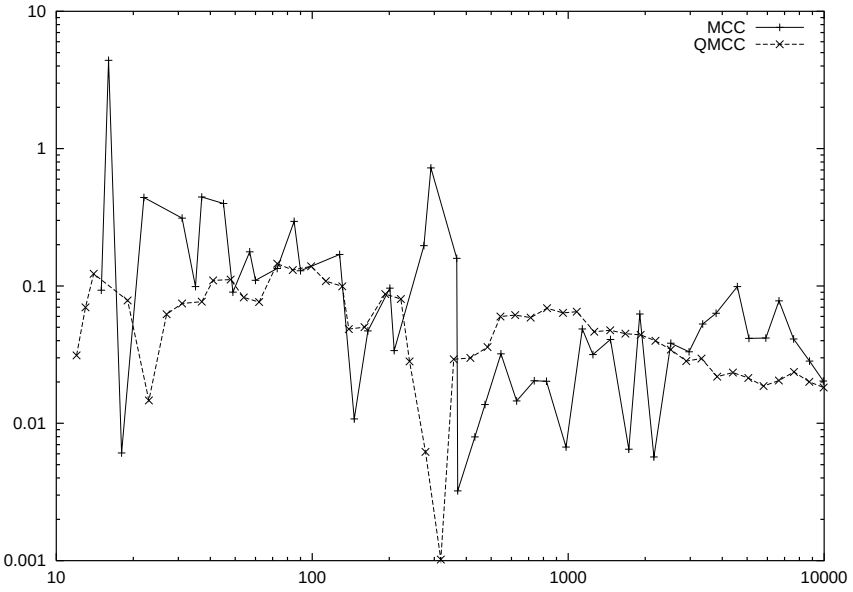

Figure 4.17: One face shared. Number of 6-points/relative error

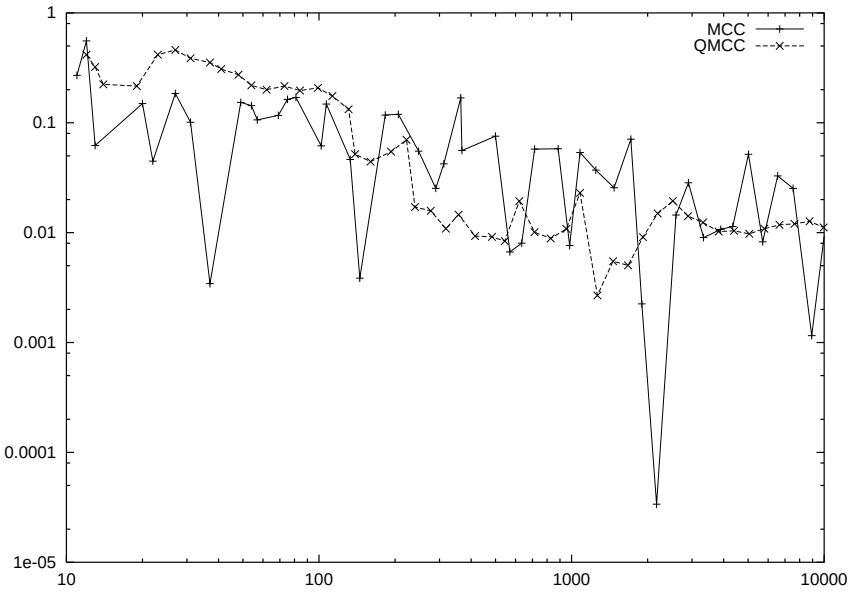

Figure 4.18: One face shared. Number of 6-points/relative error

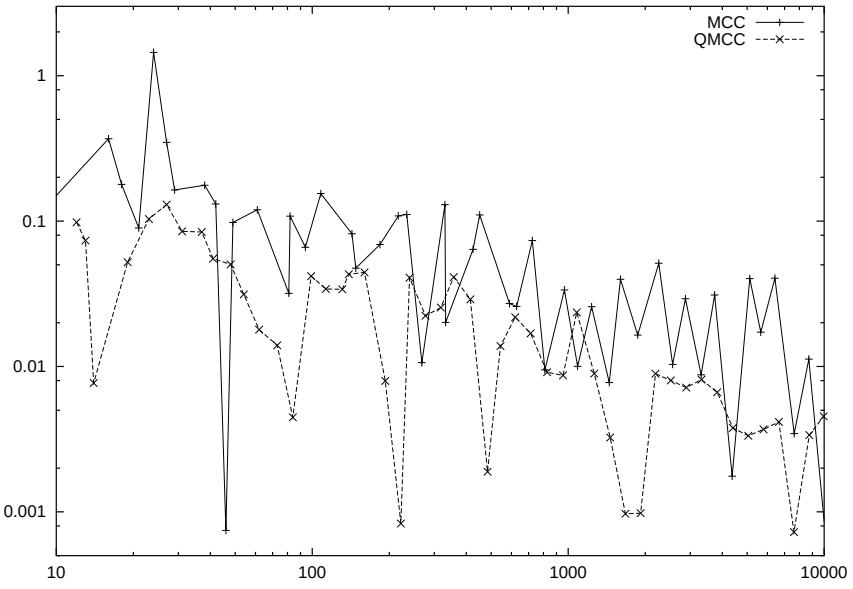

Figure 4.19: One face shared. Number of 6-points/relative error

4.5.7 Running time versus relative error

The figures 4.20, 4.21 and 4.22 report the relative error as function of the computation time for the Monte Carlo, Quasi Monte Carlo (with cone) and Gauss methods. The good performance of the QMC - QMC-cone method is confirmed always attaining the best results.

It must be noted that time for the Gauss method do not include the preprocessing time for the computation of nodes and weights. Actually this step can be done just once using the transformation formula showed in subsection 2.3.1. The algorithms implementing the Gauss method are the fastest to produce results, while the Quasi Monte Carlo cone is the slowest, as it needs the computation of the cone within which create sampling directions.

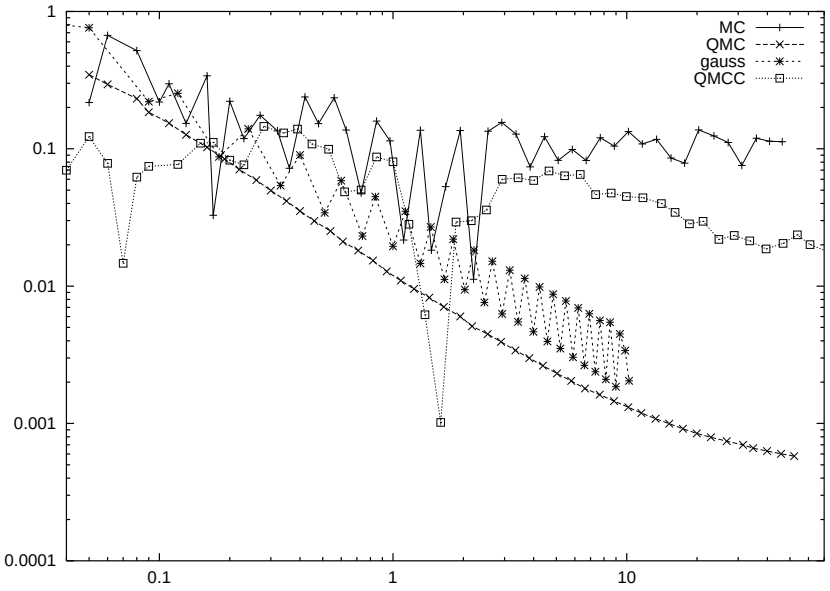

Figure 4.20: One face shared. Number of seconds of computation/relative error

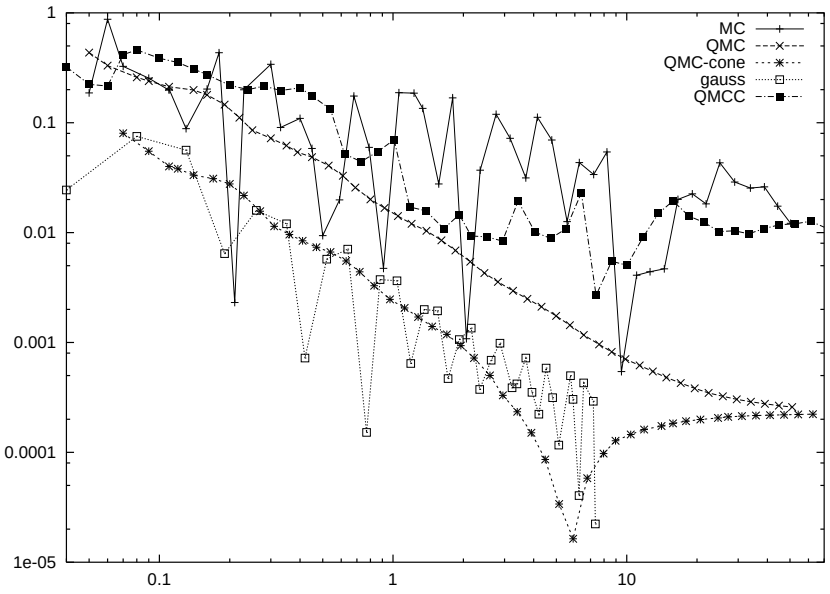

Figure 4.21: One edge shared. Number of seconds of computation/relative error

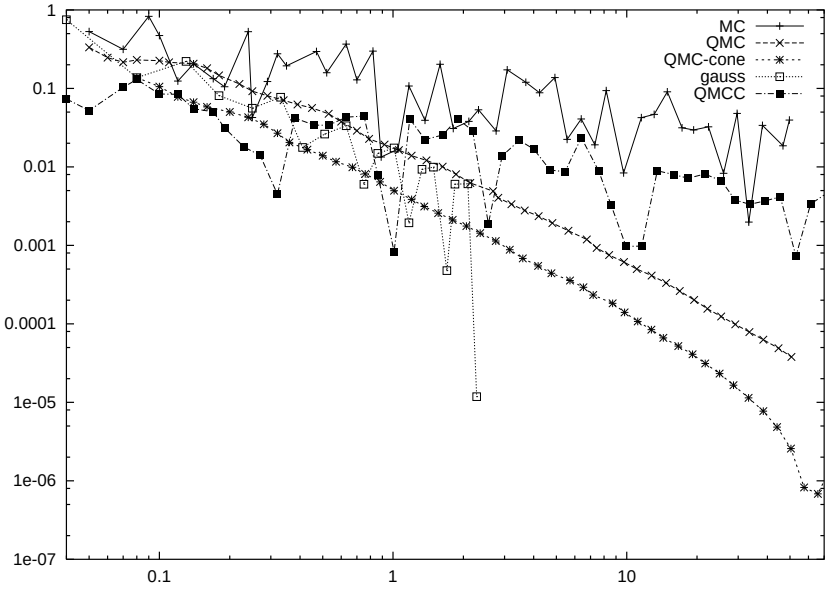

Figure 4.22: One vertex shared. Number of seconds of computation/relative error

You're the brightest hope by far that anyone can see
You're gaining fame and claiming credibility
How can you expect to be taken seriously?
Neil Tennant

Chapter 5

Experiments on surface to surface integrals

We examine the case of triangles as input bodies (2.2.7). We put attention particularly on the case of triangles sharing one edge, showing results for the Quasi Monte Carlo, the Gauss and Gauss-adaptive methods. We also report an experiment done with triangles sharing one vertex.

5.1 Inputs

We considered triangles sharing one edge and one vertex. In the first case we executed two series of experiments modifying the angle formed by the planes spanning the input triangles and modifying the aspect ratio of one of the triangles. In the first set of experiments we set $T_1 = \{\{0,0,-1\},\{0,0,1\},\{0,-1,0\}\}$, and $T_2 = \{\{0,0,-1\},\{0,0,1\},\{\sin(\Phi),-\cos(\Phi),0\}\}$ (fig. 5.1), the angle Φ assuming the values $\frac{31}{32}\pi, \frac{15}{16}\pi, \frac{3}{4}\pi, \frac{1}{2}\pi, \frac{1}{4}\pi, \frac{1}{8}\pi, \frac{1}{16}\pi, \frac{1}{32}\pi$. These experiments are intended to study the dependency of the algorithms on the dihedral angle between the input triangles. In the second set of experiments T_1 is as before, while $T_2 = \{\{0,0,-1\},\{0,0,1\},\{-\sin(\Theta),0,\cos(\Theta)\}\}$ (fig. 5.2), with $\Theta = \frac{1}{2^i}\pi$ and $i = 2,3,4,5$. This set of experiments is intended to study the dependency of the algorithms on the aspect ratio of one of the two triangles. The third experiment involves two triangles sharing one vertex as shown in figure 5.3: $T_1 = \{\{0,0,0\},\{0,0,1\},\{1,0,0\}\}$, $T_2 = \{\{0,0,0\},\{0,1,0\},\{0,1,1\}\}$. As for the experiments with tetrahedra, we executed the Quasi Monte Carlo algorithm

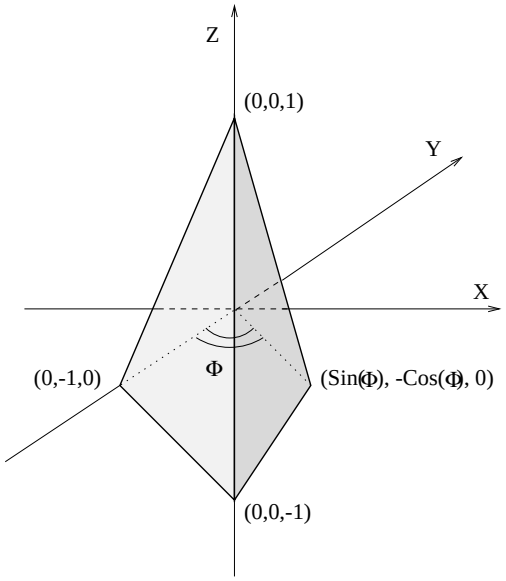

Figure 5.1: Triangles sharing one edge: first set of experiments. $\Phi = \frac{31}{32}\pi, \frac{15}{16}\pi, \frac{3}{4}\pi, \frac{1}{2}\pi, \frac{1}{4}\pi, \frac{1}{8}\pi, \frac{1}{16}\pi, \frac{1}{32}\pi$.

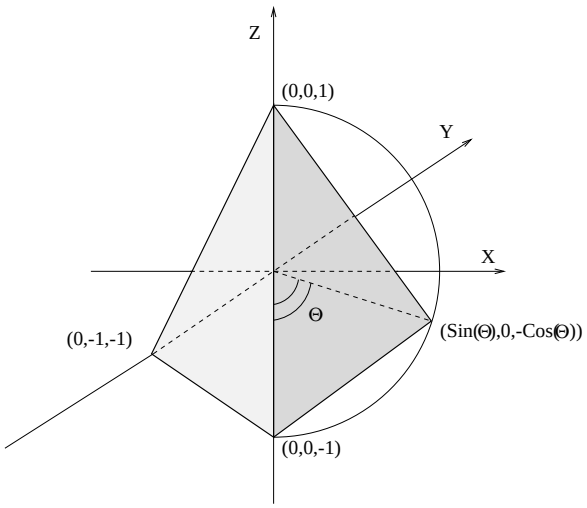

Figure 5.2: Triangles sharing one edge: second set of experiments. $\Theta = \frac{1}{2^i}\pi$ with $i = 2, 3, 4, 5$

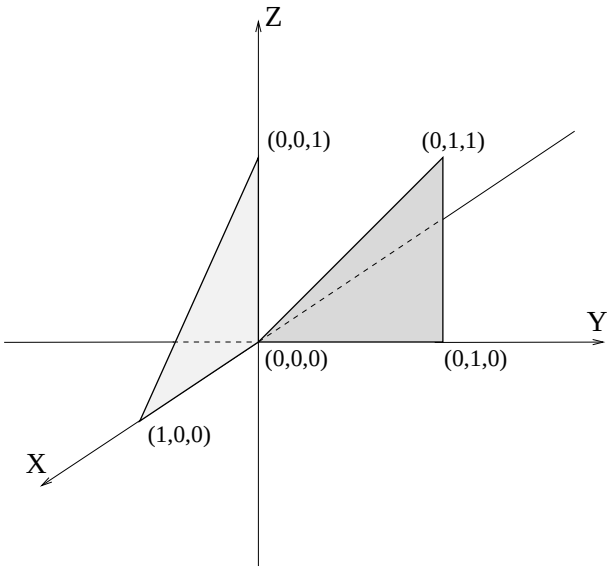

Figure 5.3: Triangles sharing one vertex

with $n_0 = 10, \ldots, n_i, \ldots, n_{49} = 10,000$ directions where n_i, $(i = 0, \ldots, 49)$, are such that they are equally spaced in the logarithmic scale. On the other side we get approximation from the Gauss methods increasing the dimension of product formulas, stopping computations when round-off error occur.

5.2 Algorithms

In the light of the results of the tetrahedron-tetrahedron experiments, we include here only the Quasi Monte Carlo and the Gauss methods plus the Gauss adaptive method which has been developed right for this kind of experiments.

5.2.1 Quasi Monte Carlo

The algorithm is the same as that treated for the case of tetrahedra; the sampling point are generated in the φ-θ plane within the square $[-\frac{1}{2}\pi, \frac{1}{2}\pi] \times [-\frac{1}{2}\pi, \frac{1}{2}\pi]$. In the first set of experiments it is possible to easily avoid all the directions that give null contribute to the computation of the electrostatic field. The fact that a direction has null contribute depends only on the value of φ. Precisely, if α is the dihedral angle, all directions with $-\frac{1}{2}\pi \leq \varphi \leq -\frac{1}{2}\pi + \alpha$ have null contribution to the computation of the electrostatic field, while all

directions with $-\frac{1}{2}\pi + \alpha \leq \varphi \leq \frac{1}{2}\pi$ give actual contribution. The bigger is the value of α, the more convenient is to produce sampling points over the rectangle $[-\frac{1}{2}\pi + \alpha, \frac{1}{2}\pi] \times [-\frac{1}{2}\pi, \frac{1}{2}\pi]$.

5.2.2 Gauss

The basic algorithm has no differences from the implementation for the volume-to-volume integrals except that the zone of the φ-θ plane over which are generated sampling points is $[-\frac{1}{2}\pi, \frac{1}{2}\pi] \times [-\frac{1}{2}\pi, \frac{1}{2}\pi]$. Moreover, as described in the former section, for the first set of experiments it is possible to easily prevent the production of useless direction generating Gaussian points over a reduced rectangular zone of the ϕ-θ plane.

5.2.3 Gauss Adaptive

In this method the parameter N is not specified. The number of directions used to obtain an approximation depends on the number of zones the φ-θ is divided according to the strategy explained in section 3.5. We run the algorithm producing $(k \times k)$-Gaussian grid for each zone, starting from $k = 2$ and increasing k till round-off error occurred.

5.3 Computation of reference values

At first we tried to obtain a reference value following the steps described in section 4.3, using the formula derived from the Coulomb's law and running a huge computation by the Monte Carlo method. Even after few days of computation the results were extremely discordant (starting from the third digit). We then decided to tackle the integral derived from the geometrical interpretation (formula 2.2.6).

In the case of triangles sharing one edge, once the φ-θ plane has been divided according to the technique explained in section 3.5, there are only four zones which give actual contribute to the computation. For each of them we explicitly wrote the formulas for the kernel as function of φ and θ. Here we report such formulas for the case $\Phi = \frac{1}{2}\pi$ of the first set of experiments:

Zone 1 $\rightarrow \cos(\theta)$,

Zone 2 $\rightarrow \cos(\theta)\ \cot(\phi)$,

$$\text{Zone 3} \rightarrow \frac{2\ \cos(\phi)\cos(\theta)^2}{\cos(\phi)\ (\cos(\theta)-\cos(\phi)\ \sin(\theta))-\sin(\phi)\ (-\cos(\theta)+\sin(\phi)\ \sin(\theta))},$$

$$\text{Zone 4} \rightarrow \frac{2\ \cos(\phi)\cos(\theta)^2}{-(\cos(\phi)\ (-\cos(\theta)-\cos(\phi)\ \sin(\theta)))+\sin(\phi)\ (\cos(\theta)+\sin(\phi)\ \sin(\theta))}.$$

Then we symbolically integrated (by Mathematica) these functions on the θ variable (this step requires the subdivision of zones 3 and 4 in two subzones). Finally we numerically integrated on the φ variable by the internal Mathematica routine `NIntegrate` (version 3.01, see [Wol96]). Here we show the results of the symbolical integration step:

- $SymbF1(\phi) = \dfrac{2\,(\cos(\phi)-\sin(\phi))}{\sqrt{2-\sin(2\,\phi)}}$
- $SymbF2(\phi) = \dfrac{-2\ \cot(\phi)\ (\cos(\phi)-\sin(\phi))}{\sqrt{2-\sin(2\,\phi)}}$
- $SymbF3\mathrm{a}(\phi) =$

$$\frac{-4\,i\ \arctan\left(\frac{-i\sqrt{2}+(-1)^{\frac{1}{4}}\ \cos(\phi)+(-1)^{\frac{3}{4}}\ \cos(\phi)+(-1)^{\frac{1}{4}}\ \sin(\phi)+(-1)^{\frac{3}{4}}\ \sin(\phi)}{\sqrt{2}\ \sqrt{-i+\cos(\phi)+\sin(\phi)}\ \sqrt{i+\cos(\phi)+\sin(\phi)}}\right)\ \cos(\phi)}{(-i+\cos(\phi)+\sin(\phi))^{\frac{3}{2}}\ (i+\cos(\phi)+\sin(\phi))^{\frac{3}{2}}}$$

$$+\frac{4\,i\ \arctan\left(\frac{\sec K1\,(i\ \cos(\phi-K1)-i\ \cos(\phi+K1)-2\,i\ \cos K1-i\ \sin(\phi-K1)+i\ \sin(\phi+K1))}{2\ \sqrt{-i+\cos(\phi)+\sin(\phi)}\ \sqrt{i+\cos(\phi)+\sin(\phi)}}\right)\ \cos(\phi)}{(-i+\cos(\phi)+\sin(\phi))^{\frac{3}{2}}\ (i+\cos(\phi)+\sin(\phi))^{\frac{3}{2}}}$$

$$+\frac{1+\cos(2\,\phi)+\sin(2\,\phi)}{2+\sin(2\,\phi)}$$

$$+\frac{-\cos(\phi-2\,K1)-0.5\ \cos(2\,\phi-2\,K1)-\cos(\phi+2\,K1)+0.5\ \cos(2\,\phi+2\,K1)}{2+\sin(2\,\phi)}$$

$$+\frac{-\frac{(\cos(\phi)-\sin(\phi))}{\sqrt{2-\sin(2\,\phi)}}+0.5\ \sin(2\,\phi-2\,K1)-0.5\ \sin(2\,\phi+2\,K1)}{2+\sin(2\,\phi)},$$

where $K1 = \left(\frac{\arctan(\cos(\phi)-\sin(\phi))}{2}\right)$.

- $SymbF3\mathrm{b}(\phi)=$

$$\frac{-4\,i\,\arctan\left(\frac{-i\sqrt{2}+(-1)^{\frac{1}{4}}\cos(\phi)+(-1)^{\frac{3}{4}}\cos(\phi)+(-1)^{\frac{1}{4}}\sin(\phi)+(-1)^{\frac{3}{4}}\sin(\phi)}{\sqrt{2}\sqrt{-i+\cos(\phi)+\sin(\phi)}\sqrt{i+\cos(\phi)+\sin(\phi)}}\right)\cos(\phi)}{(-i+\cos(\phi)+\sin(\phi))^{\frac{3}{2}}\,(i+\cos(\phi)+\sin(\phi))^{\frac{3}{2}}}$$

$$+\frac{4\,i\,\arctan(\frac{\sec K1\,(-i\cos(\phi-K1)+i\cos(\phi+K1)-2\,i\cos K1+i\sin(\phi-K1)-i\sin(\phi+K1))}{2\sqrt{-i+\cos(\phi)+\sin(\phi)}\sqrt{i+\cos(\phi)+\sin(\phi)}})\cos(\phi)}{(-i+\cos(\phi)+\sin(\phi))^{\frac{3}{2}}\,(i+\cos(\phi)+\sin(\phi))^{\frac{3}{2}}}$$

$$+\frac{1+\cos(2\,\phi)+\sin(2\,\phi)}{2+\sin(2\,\phi)}$$

$$+\frac{-\cos(\phi-2\,K1)+0.5\cos(2\,\phi-2\,K1)-\cos(\phi+2\,K1)-0.5\cos(2\,\phi+2\,K1)}{2+\sin(2\,\phi)}$$

$$+\frac{\frac{(\cos(\phi)-\sin(\phi))}{\sqrt{2-\sin(2\,\phi)}}-0.5\sin(2\,\phi-2\,K1)+0.5\sin(2\,\phi+2\,K1)}{2+\sin(2\,\phi)},$$

  where $K1=(\frac{\arctan(\cos(\phi)-\sin(\phi))}{2})$.

  Functions $SymbF4\mathrm{a}(\phi)$ and $SymbF4\mathrm{b}(\phi)$, deriving from the "Zone4" formula are similar to $SymbF3\mathrm{a}(\phi)$ and $SymbF3\mathrm{b}(\phi)$.

Formulas for the second set of experiments are of the same kind of those shown above. Note that only the primitive relative to Zone1 and Zone2 have a dimensional reduction, while the other ones present the imaginary unit $i=\sqrt{-1}$ which makes the functions $SymbF3\mathrm{a}(\phi)$, $SymbF3\mathrm{b}(\phi)$, $SymbF4\mathrm{a}(\phi)$ and $SymbF4\mathrm{a}(\phi)$ defined over the complex plane. The results are numbers with at most sixteen digits (see table 5.1). Even raising the number of digits involved in the computation (`$WorkingPrecision`), the final results had no more than sixteen digits and they were identical to the former ones.

For the experiment involving two triangles sharing one edge, the symbolical integration step has not been possible, so we proceeded this way:

- We computed all the zones of the φ-θ plane.
- We tested each zone for actual contribute; for this, given a direction u internal to a zone, it is sufficient to detect whether the intersection polygon P exists or not.

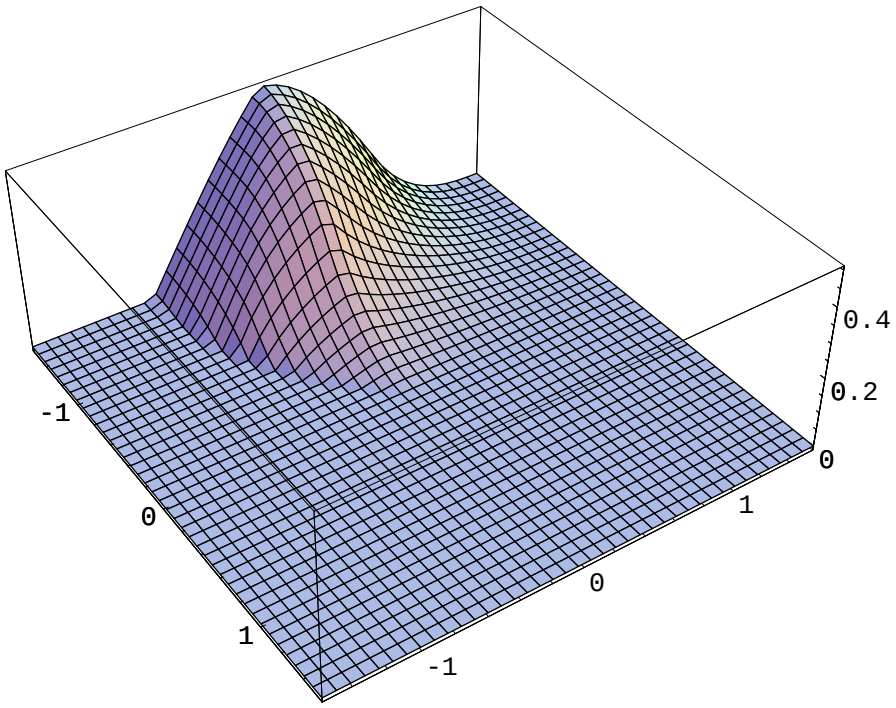

Figure 5.4: Kernel function for the case of triangles sharing one edge

- For each zone with actual contribution we wrote by hand the formula of the kernel of integral 2.2.6.
- We got an approximation of the integral by summing the values obtained for each zone by the `NIntegrate` procedure of Mathematica (version 3.0 for Linux).

It must be noted that Mathematica needed 4505 evaluations of the various explicit functions in order to return the approximation. In this case the reference value has 15 digits (see table 5.1). As for the former values, we are not able to establish the actual precision. In figure 5.4 we show the plot of the kernel function obtained by the explicit formulas written for each zone.

First set of experiments	
Φ	Reference Value
$\pi/32$	5.426712557037823
$\pi/16$	4.916274055459017
$\pi/8$	4.184555630328516
$\pi/4$	3.189605706244585
$\pi/2$	1.872097326276217
$3/4\pi$	0.881467434864744
$15/16\pi$	0.216634710095288
$31/32\pi$	0.1082268646099841
Second set of experiments	
Θ	Reference Value
$\pi/4$	1.45023916712522
$\pi/8$	0.895303787001583
$\pi/16$	0.5069602532526432
$\pi/32$	0.2738958947435712
Triangles sharing one vertex	
Reference value	
0.365071957561911	

Table 5.1: Reference values for surface-to-surface experiments

5.4 Tables

Tables 5.2, 5.3 and 5.4 show the experimental results (relative error versus number of directions) for the triangles sharing one edge in the case the dihedral angle Φ is $\frac{1}{32}\pi$, $\frac{1}{2}\pi$ and $\frac{31}{32}\pi$; the behaviour for the other values of Φ is shown by plots in section 5.5. In table 5.5 we report the relative error as function of the computation time for the case $\Phi = \frac{1}{2}\pi$; it is representative for all the family of experiments. Tables 5.6-5.9 show the results for the second set of experiments for several values of Θ. Last table (5.10) is the relative error versus computation time for the case $\Theta = \frac{1}{16}\pi$.

N	**QMC**	**QMC-cone**	**Gauss**	**Gauss-cone**	**Gauss adaptive**
10	0.0106040917	0.0067392711	0.0210938350	0.0099399448	-
25	0.0196996272	0.0010757823	0.0034273109	0.0043247568	0.012406136272012
50	0.0127838531	0.0037025071	0.0077730135	0.0013574675	0.004827410731152
75	0.0054989721	0.0030053660	0.0016186055	0.0002780876	-
100	0.0043332627	0.0029641034	0.0035424189	0.0004783674	0.001741267285380
250	0.0017554410	0.0014515091	0.0012447138	0.0001302307	0.000217525382338
500	0.0009584164	0.0008909433	0.0000789556	0.0000617531	0.000008450932804
750	0.0006362474	0.0005924587	0.0003438715	0.0000293214	0.000000785699765
1000	0.0005009333	0.0004762579	0.0004450780	0.0000232381	0.000000029977178
2500	0.0001448690	0.0001387213	-	-	0.000000000806948
5000	0.0000737030	0.0000696641	-	-	0.000000000000899
7500	0.0000690616	0.0000670131	-	-	-
10000	0.0000378462	0.0000352472	-	-	-

Table 5.2: Triangles sharing one edge. First set of experiments. Dihedral angle $\Phi = \frac{1}{32}\pi$. Number of directions/Relative error

N	**QMC**	**QMC-cone**	**Gauss**	**Gauss-cone**	**Gauss adaptive**
10	0.0972989862	0.0065320365	0.2165219575	0.0664795933	-
25	0.0361609524	0.0366486271	0.0408470774	0.0130667482	0.002283139023758
50	0.0377776609	0.0180217632	0.0387407106	0.0093797082	0.000208637303841
75	0.0122253210	0.0086970395	0.0272269108	0.0022322334	-
100	0.0173427760	0.0084036175	0.0103432812	0.0004240828	0.000015489387907
250	0.0058673260	0.0031552382	0.0041953194	0.0003179505	0.000000102017689
500	0.0032767973	0.0017211057	0.0022276241	0.0000037783	0.000000000060139
750	0.0021792458	0.0009804032	0.0028297653	0.0002155594	0.000000000000457
1000	0.0017298619	0.0008849364	0.0007542757	0.0000792836	0.000000000000014
2500	0.0003607547	0.0002074811	-	-	-
5000	0.0002124380	0.0001072733	-	-	-
7500	0.0002205657	0.0001076782	-	-	-
10000	0.0001120866	0.0000623931	-	-	-

Table 5.3: Triangles sharing one edge. First set of experiments. Dihedral angle $\Phi = \frac{1}{2}\pi$. Number of directions/Relative error

N	**QMC**	**QMC-cone**	**Gauss**	**Gauss-cone**	**Gauss adaptive**
10	1.0000000000	0.0018173148	1.0000000000	0.1077251995	-
25	1.0000000000	0.0403887926	1.0000000000	0.0214583371	0.001817906997040
50	1.0000000000	0.0199081808	0.2566510497	0.0138371379	0.000141763967485
75	0.7493109618	0.0086943770	0.4832349237	0.0045375277	-
100	0.3697200461	0.0087037266	0.3851940817	0.0007038178	0.000008916478328
250	0.1759238088	0.0035472341	0.1130647122	0.0001687729	0.000000036005836
500	0.0739634437	0.0023217931	0.0140931473	0.0002667381	0.000000000009550
750	0.0317971218	0.0010912243	0.0159135937	0.0000292218	0.000000000000039
1000	0.0280967410	0.0009485341	0.0215013098	0.0000191569	0.000000000000003
2500	0.0097841175	0.0002146158	-	-	-
5000	0.0039686447	0.0001108734	-	-	-
7500	0.0036731796	0.0001138743	-	-	-
10000	0.0020213066	0.0000546820	-	-	-

Table 5.4: First set of experiments. Dihedral angle $\Phi = \frac{31}{32}\pi$. Number of directions/Relative error

sec.	**QMC**	**QMC-cone**	**Gauss**	**Gauss-cone**	**Gauss adaptive**
0.01	0.0403382947	0.0121735460	0.0304167504	0.0130667482	-
0.03	0.0235569588	0.0109673498	0.0058906327	0.0016333301	0.000208637303841
0.05	0.0108154758	0.0084036175	0.0109412861	0.0007172048	0.000015489387907
0.07	0.0077506981	0.0058634408	0.0114208259	0.0010498308	0.000001210636453
0.1	0.0086394318	0.0043504276	0.0041953194	0.0003179505	0.000000102017689
0.3	0.0026226941	0.0011319942	0.0028297653	0.0000941544	0.000000000005165
0.5	0.0017298619	0.0008849364	0.0007648762	0.0000192332	0.000000000000014
1	0.0008798669	0.0004423672	-	-	-
3	0.0002442961	0.0001072733	-	-	-

Table 5.5: First set of experiments. Dihedral angle $\Phi = \frac{1}{2}\pi$. Number of seconds of computation/Relative error

N	**QMC**	**Gauss adaptive**
10	0.1073645758	-
25	0.0591233640	0.004739885970635
50	0.0300279571	0.000618152696938
75	0.0180348957	-
100	0.0130557080	0.000102204760620
250	0.0080276991	0.00000274927355
500	0.0035581103	0.000000012843913
750	0.0019745205	0.000000000368697
1000	0.0015377026	0.000000000010505
1350	0.0009032502	0.000000000000063
2500	0.0005327552	-
5000	0.0002530672	-
7500	0.0001764577	-
10000	0.0001166639	-

Table 5.6: Second set of experiments. $\Theta = \frac{1}{4}\pi$. Relative error/Number of directions

N	**QMC**	**Gauss adaptive**
10	0.1101518446	-
25	0.0561332615	0.005200010610612
50	0.0255623250	0.000840038445773
75	0.0178422220	-
100	0.0146219069	0.000263536647826
250	0.0071789678	0.000020640498742
500	0.0026361306	0.000000388285443
750	0.0017265220	0.000000027113299
1000	0.0012740468	0.000000001916440
2000	0.0007429855	0.000000000001558
2500	0.0004341022	-
5000	0.0002285288	-
7500	0.0001670301	-
10000	0.0001121254	-

Table 5.7: Second set of experiments. $\Theta = \frac{1}{8}\pi$. Relative error/Number of directions

N	**QMC**	**Gauss adaptive**
10	0.0396876388	-
25	0.0527426679	0.002186671960565
50	0.0244360887	0.000035331685995
75	0.0191125874	-
100	0.0143751340	0.000144115756655
250	0.0059936946	0.000040530977500
500	0.0032835610	0.000003180498856
750	0.0018460935	0.000000497563158
1000	0.0011932937	0.000000073637885
2500	0.0004320562	0.000000000082417
3000	0.0003835113	0.000000000005400
5000	0.0002065243	-
7500	0.0001481685	-
10000	0.0001071912	-

Table 5.8: Second set of experiments. $\Theta = \frac{1}{16}\pi$. Relative error/Number of directions

N	**QMC**	**Gauss adaptive**
10	0.0040277315	-
25	0.0311554559	0.001494021477299
50	0.0245128268	0.001660520320911
75	0.0142277825	-
100	0.0131725482	0.000500217487681
250	0.0064887377	0.000046556816721
500	0.0029823780	0.000002376090449
750	0.0016766577	0.000001629245172
1000	0.0015780418	0.000000600249247
2500	0.0003671131	0.000000006355308
5000	0.0002070420	0.000000000019445
5500	0.0001905450	0.000000000004773
7500	0.0001762077	-
10000	0.0001018173	-

Table 5.9: Second set of experiments. $\Theta = \frac{1}{32}\pi$. Relative error/Number of directions

sec.	**QMC**	**Gauss adaptive**
0.01	0.0545837077	-
0.03	0.0305356709	0.002186671960565
0.05	0.0143751340	0.000144115756655
0.07	0.0079158082	0.000083311692056
0.1	0.0065259176	0.000040530977500
0.3	0.0019772847	0.000001270744987
0.5	0.0011932937	0.000000073637885
1	0.0006413101	0.000000000225978
1.5	0.0003835113	0.000000000002860
3	0.0002117066	-

Table 5.10: Triangles sharing one edge. Second set of experiments. $\Theta = \frac{1}{16}\pi$. Number of seconds of computation/Relative error

5.5 Graphics

We have subsections for algorithms based on Quasi Monte Carlo method, Gauss method and Gauss adaptive method, in which we report the relative error against the number of sampling direction. In subsection 5.5.5 we also report the relative error as function of the computation time. All the plots are in log-log scale (logarithmic scale on both the axis).

5.5.1 Quasi Monte Carlo methods

Figures from 5.5 to 5.12 show the curves relative to the first set of experiments for the Quasi Monte Carlo method and for the Monte Carlo method with cone computation, that is avoiding useless direction. When the dihedral angle Φ assumes small values, such as $\frac{1}{32}\pi$, the triangles T_1 and T_2 are very close each other and almost every direction causes the projections of T_1 and T_2 to intersect. This is why the curves for the QMC and for the QMC-cone algorithms are almost coincident from a certain number of sampling direction (see figures 5.5 and 5.6). The convenience of an clever strategy of sampling becomes more clear as Φ increases: in figures 5.7 ($\Phi = \frac{1}{8}\pi$) and 5.8 ($\Phi = \frac{1}{4}\pi$) the lines are no more overlapped and for bigger values of Φ the QMC-cone algorithm obtain

relative errors ten or hundred times lower than the QMC algorithm (figures 5.11 and 5.12). In figure 5.13 we report all the curves obtained by the plain QMC method for different values of Φ: it is clear that the smaller is Φ the lower is the curve and it is again due to the approaching of the cone of useful direction to the full φ-θ plane when Φ decreases. A plot similar to 5.12 but drawn with curves for the QMC-cone algorithms would show all the curves very close each other. This means the dihedral angle greatly influences the QMC method but not the QMC-cone one. It is interesting to note that both the curves for $\Phi = \frac{15}{16}\pi$ and $\Phi = \frac{31}{32}\pi$ have a starting phase in which the precisely obtain a relative error equal to 1. This is because none of the sampling direction give contribution, causing the integral to be approximated by the value 0. From figure 5.14 we see that the QMC method is not affected by the "aspect ratio" angle Θ.

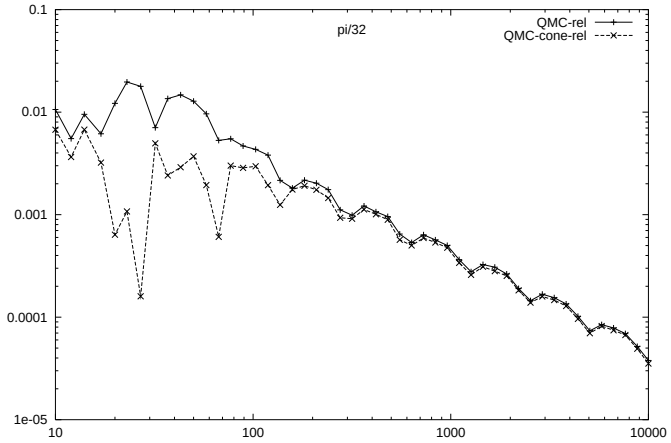

Figure 5.5: First experiment: $\Phi = \frac{1}{32}\pi$

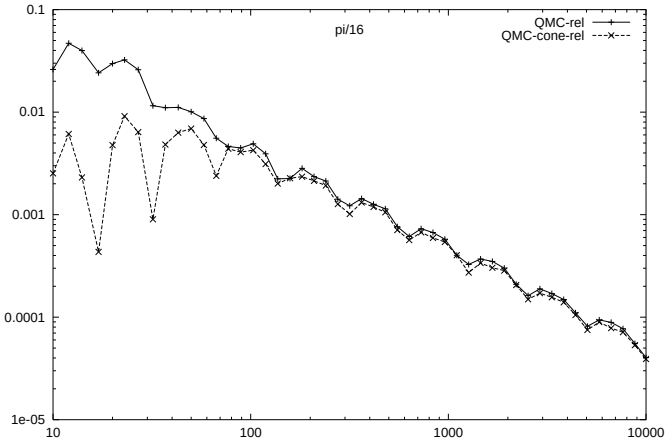

Figure 5.6: First experiment: $\Phi = \frac{1}{16}\pi$

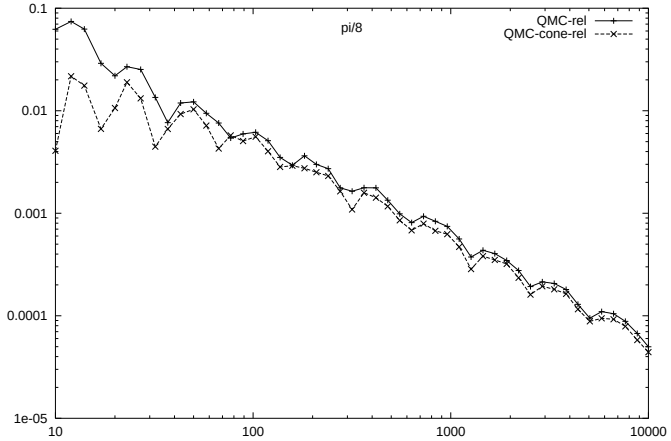

Figure 5.7: First experiment: $\Phi = \frac{1}{8}\pi$

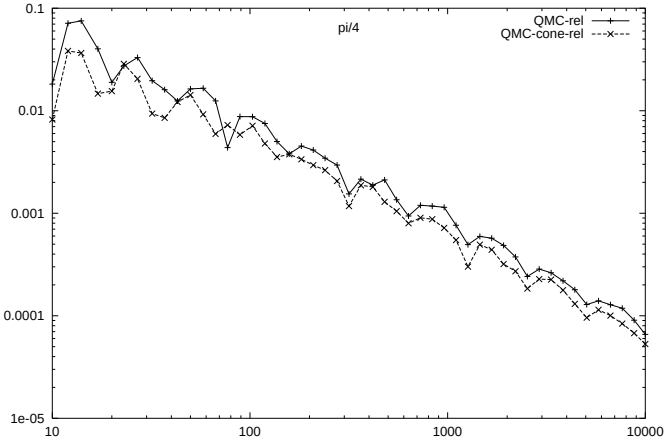

Figure 5.8: First experiment: $\Phi = \frac{1}{4}\pi$

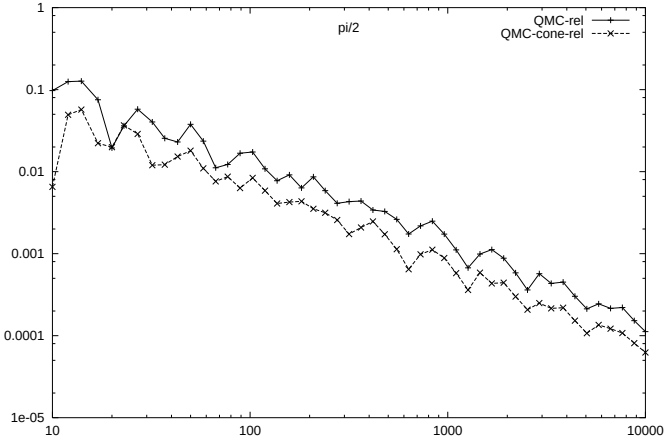

Figure 5.9: First experiment: $\Phi = \frac{1}{2}\pi$

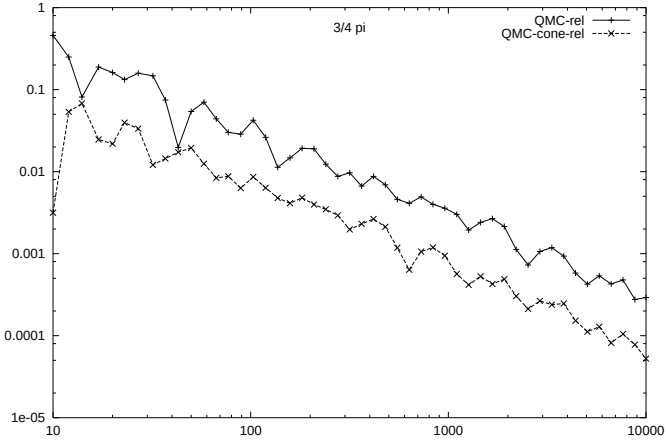

Figure 5.10: First experiment: $\Phi = \frac{3}{4}\pi$

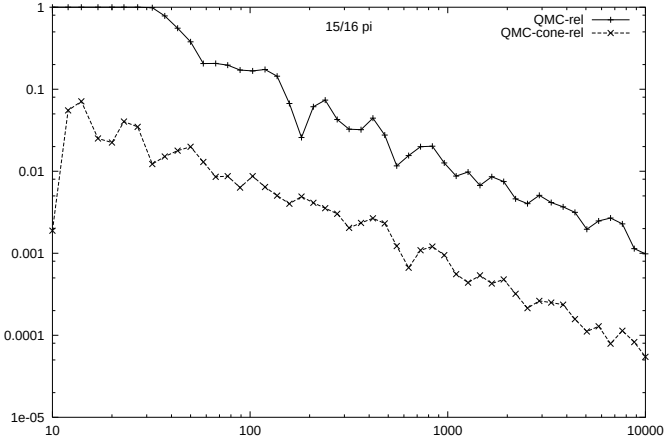

Figure 5.11: First experiment: $\Phi = \frac{15}{16}\pi$

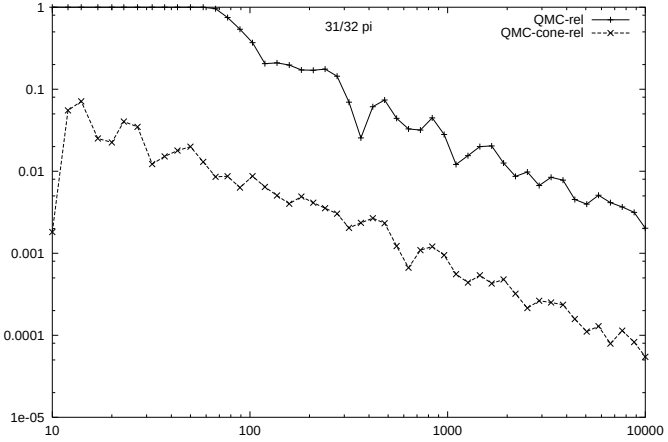

Figure 5.12: First experiment: $\Phi = \frac{31}{32}\pi$

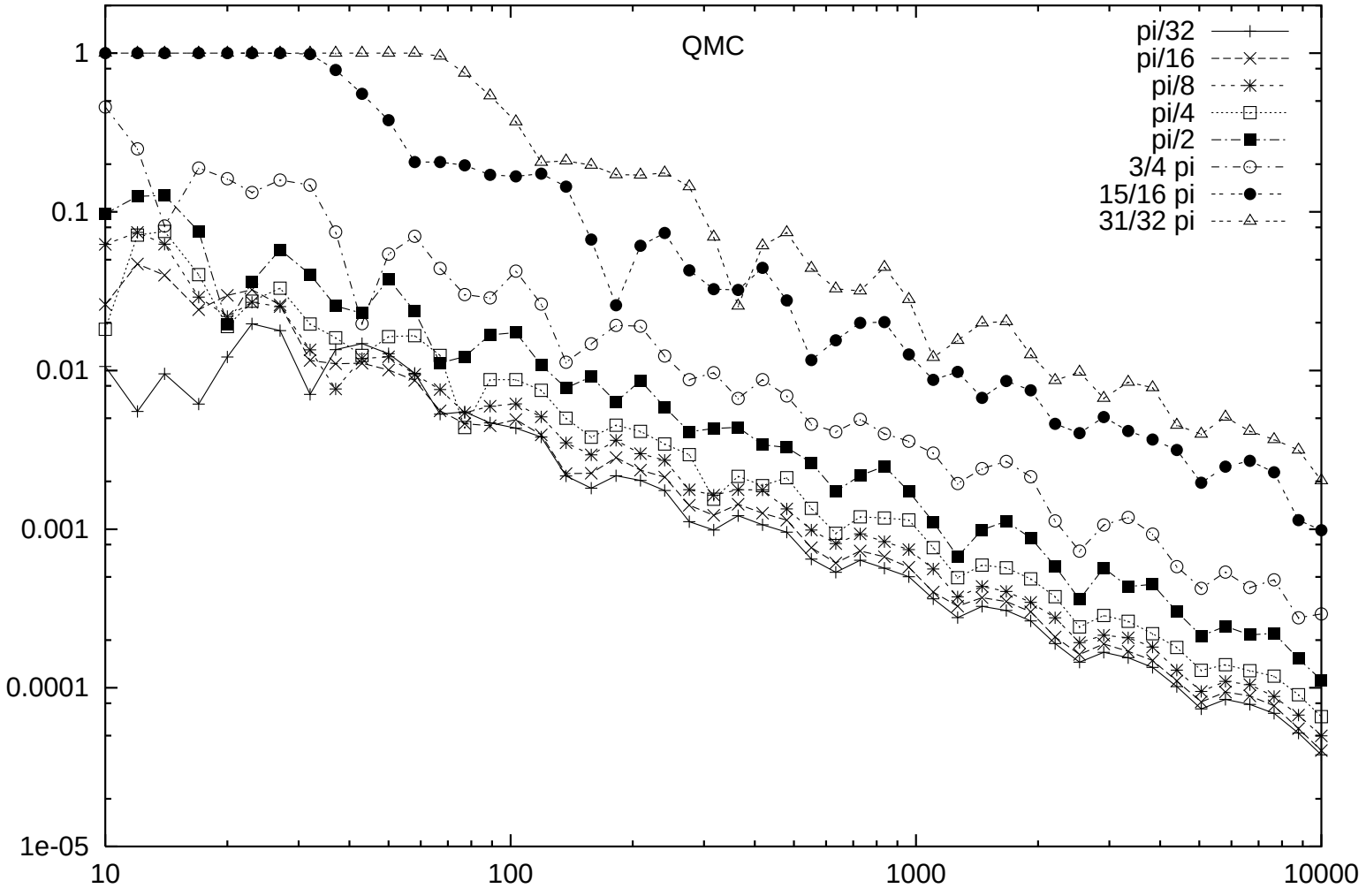

Figure 5.13: First set of experiments: comparison of the Quasi Monte Carlo method varying the angle Φ

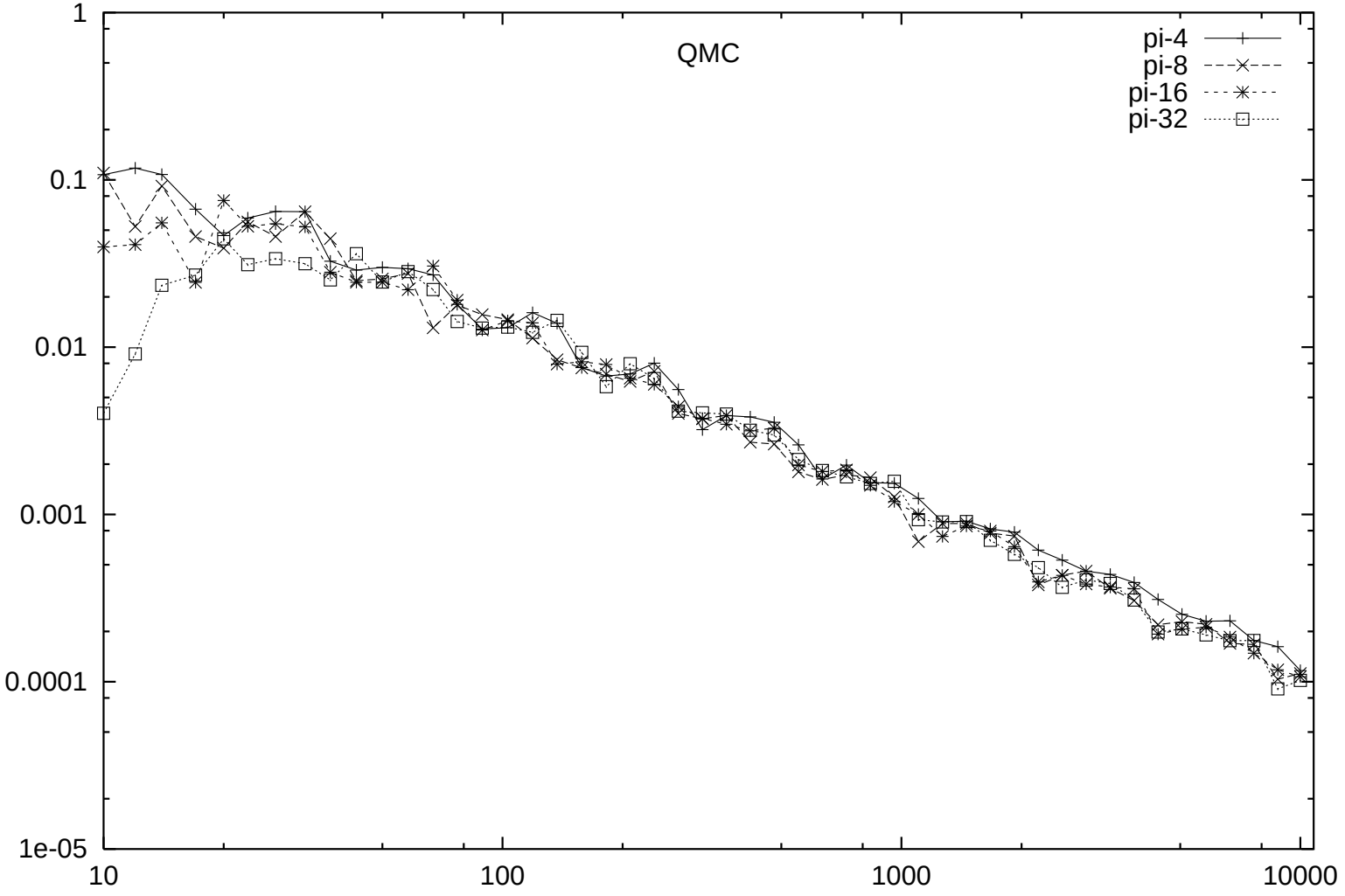

Figure 5.14: Second set of experiments: comparison of the QMC method varying Θ

5.5.2 Gauss method

The same comments made for the QMC-cone method against the QMC method could be repeated for the Gauss-cone and the plain Gauss method: if Φ is small, the cone of directions with actual contribution is almost the whole φ-θ plane and there is little convenience in implementing the cone computation. In reality Gauss method is more affected by the dihedral angle than QMC. Figures from 5.15 to 5.22 show the Gauss method against the Gauss-cone method. Even for $\Phi = \frac{1}{32}\pi$ there is a big difference between the lines derived from these algorithms. This is due to the fact that Gaussian nodes have the geometrical property of being gathered toward the boundary of the integration domain; given the angle Φ, the useless directions within the square $[-\frac{1}{2}\pi, \frac{1}{2}\pi]\times[-\frac{1}{2}\pi, \frac{1}{2}\pi]$ are those of the rectangle $[-\frac{1}{2}\pi, -\frac{\pi}{2} + \Phi] \times [-\frac{1}{2}\pi, \frac{1}{2}\pi]$ which is a strip on the external part of the integration domain. Thus many sampling points fall into a zone with null-contribute.

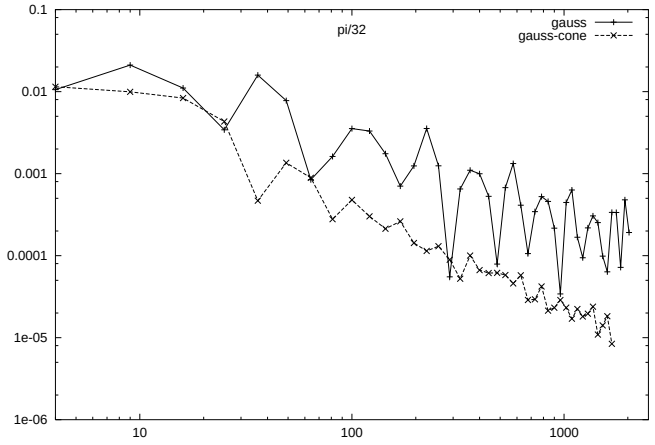

Figure 5.15: Gauss versus Gauss-cone. $\Phi = \frac{1}{32}\pi$

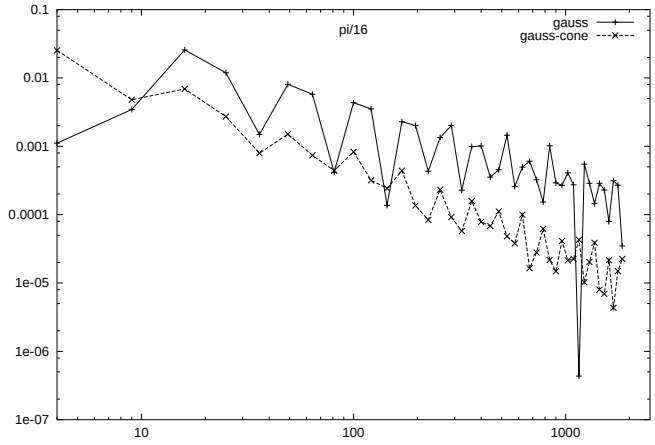

Figure 5.16: Gauss versus Gauss-cone. $\Phi = \frac{1}{16}\pi$

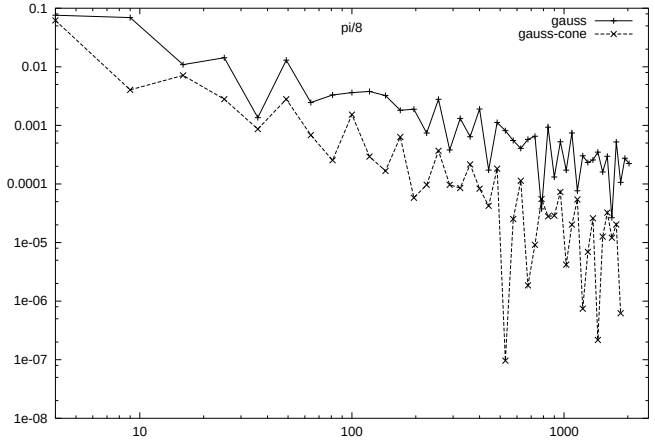

Figure 5.17: Gauss versus Gauss-cone. $\Phi = \frac{1}{8}\pi$

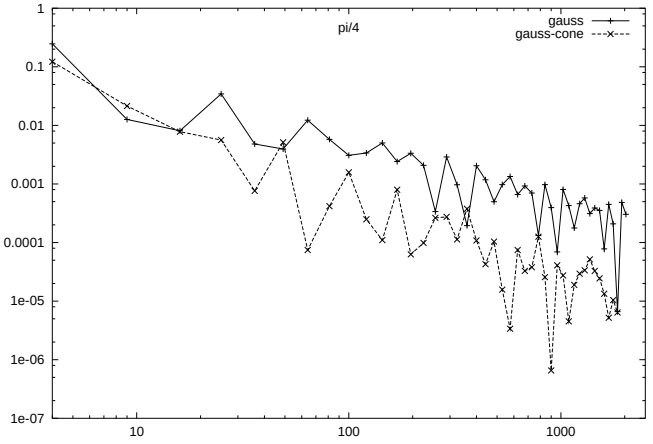

Figure 5.18: Gauss versus Gauss-cone. $\Phi = \frac{1}{4}\pi$

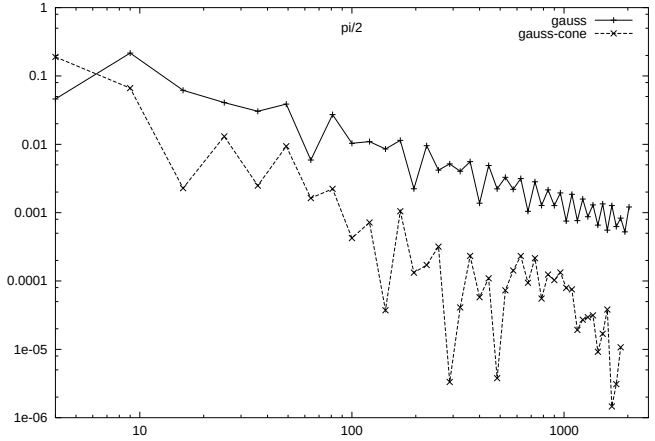

Figure 5.19: Gauss versus Gauss-cone. $\Phi = \frac{1}{2}\pi$

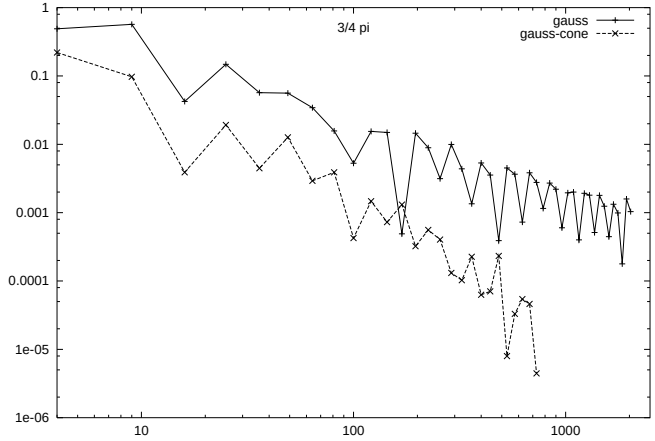

Figure 5.20: Gauss versus Gauss-cone. $\Phi = \frac{3}{4}\pi$

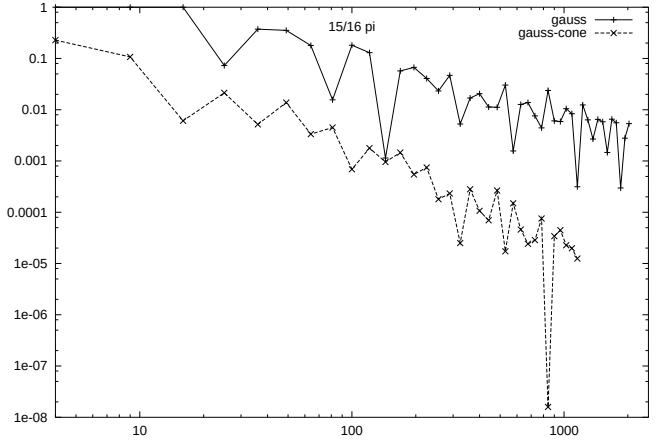

Figure 5.21: Gauss versus Gauss-cone. $\Phi = \frac{15}{16}\pi$

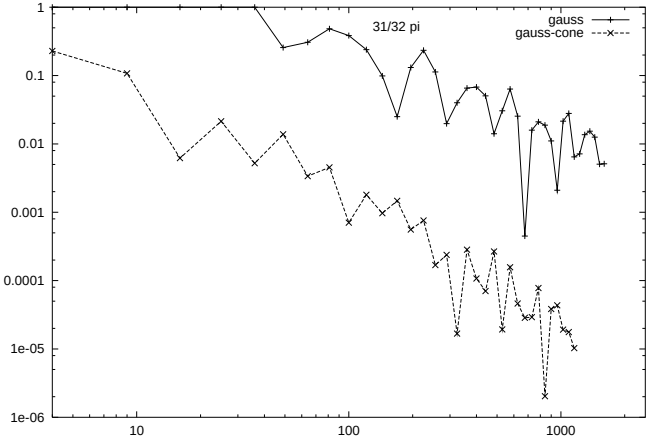

Figure 5.22: Gauss versus Gauss-cone. $\Phi = \frac{31}{32}\pi$

Gauss adaptive		
directions	relative error	result
36	0.002728203704016	0.3640759668950581
81	0.000162041348493	0.3650128008096105
144	0.000010275184097	0.3650682063803381
225	0.000000723755874	0.3650716933389369
324	0.000000047762480	0.3650719401251686
441	0.000000002515114	0.3650719566437131
576	0.000000000542752	0.3650719577600547
729	0.000000000747062	0.3650719578346426
900	0.000000000759855	0.3650719578393130
1089	0.000000000760583	0.3650719578395785
1296	0.000000000760615	0.3650719578395902
1521	0.000000000760615	0.3650719578395903
1764	0.000000000760615	0.3650719578395904

Table 5.11: Gauss adaptive results for triangles sharing one vertex

5.5.3 Gauss adaptive method

Figure 5.23 reports the curves for the Gauss adaptive method for all the values of Φ. In all the cases it attains a relative error less than 10^{-11} producing rather smooth curves. For values of the dihedral angle in the range $\frac{\pi}{4}$ to $\frac{31}{32}\pi$ the Gauss adaptive algorithm attains errors in the range 10^{-12} to 10^{-15} already for 1000 directions. When ϕ increases relative errors under the threshold (10^{-11}) are obtained with 2000-5000 directions.

In figure 5.24 we show the curves for the second set of experiments. Again a relative error of at least 10^{-11} is achieved: they are necessary 1000 directions when $\Theta = \frac{1}{4}\pi$ (with 1500 directions it is obtained 10^{-13}), up to 10000 directions for $\Theta = \frac{1}{32}\pi$.

The case of triangle sharing one vertex is illustrated in figure 5.25 The Gauss adaptive method achieve a precision of 10^{-9} already for 600 directions, settling to that values even for more directions. In the light of the results (see table 5.11), we suppose the reference values is not as accurate as the estimations obtained from the Gauss adaptive algorithm.

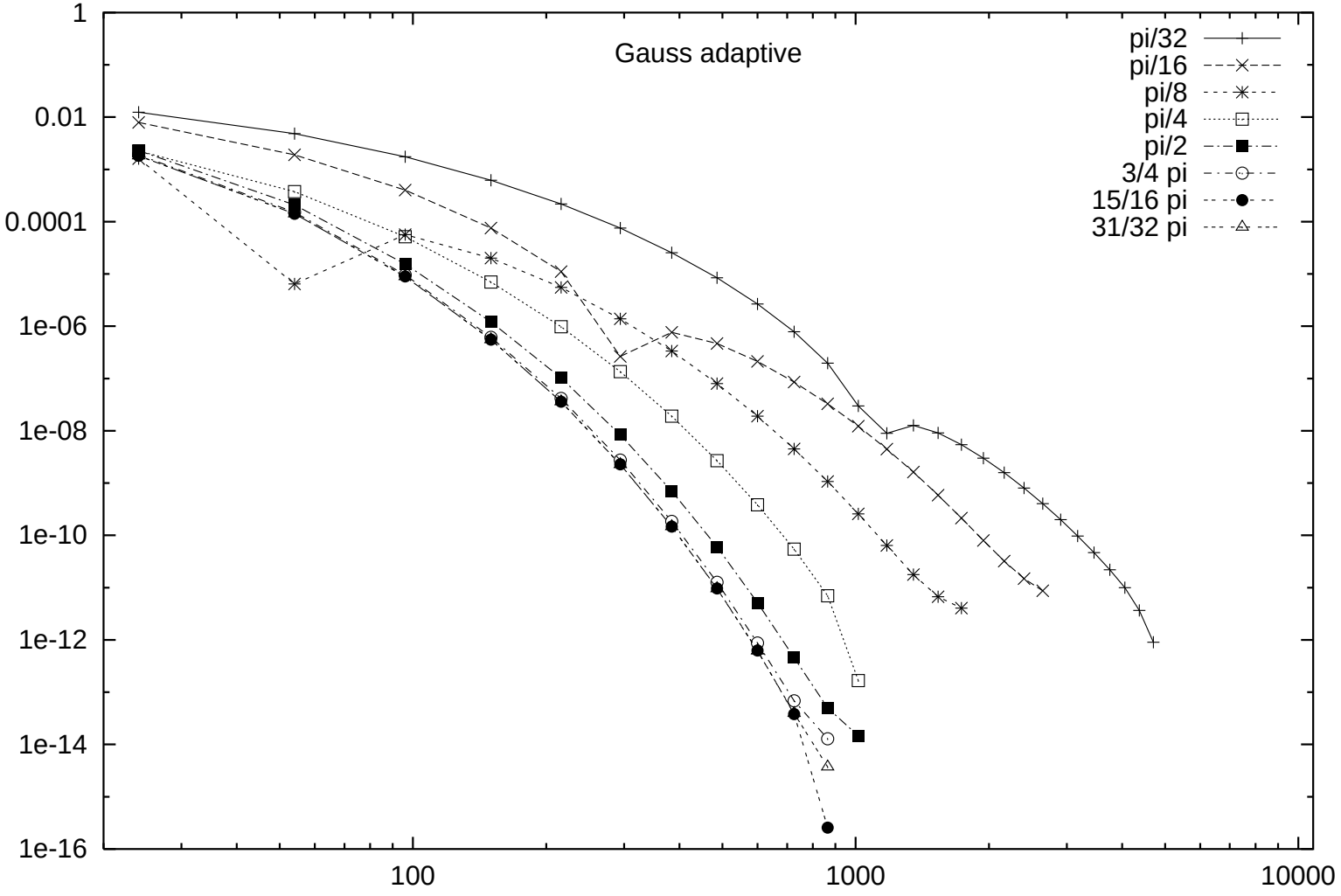

Figure 5.23: First set of experiments: comparison of the gauss adaptive method varying Φ

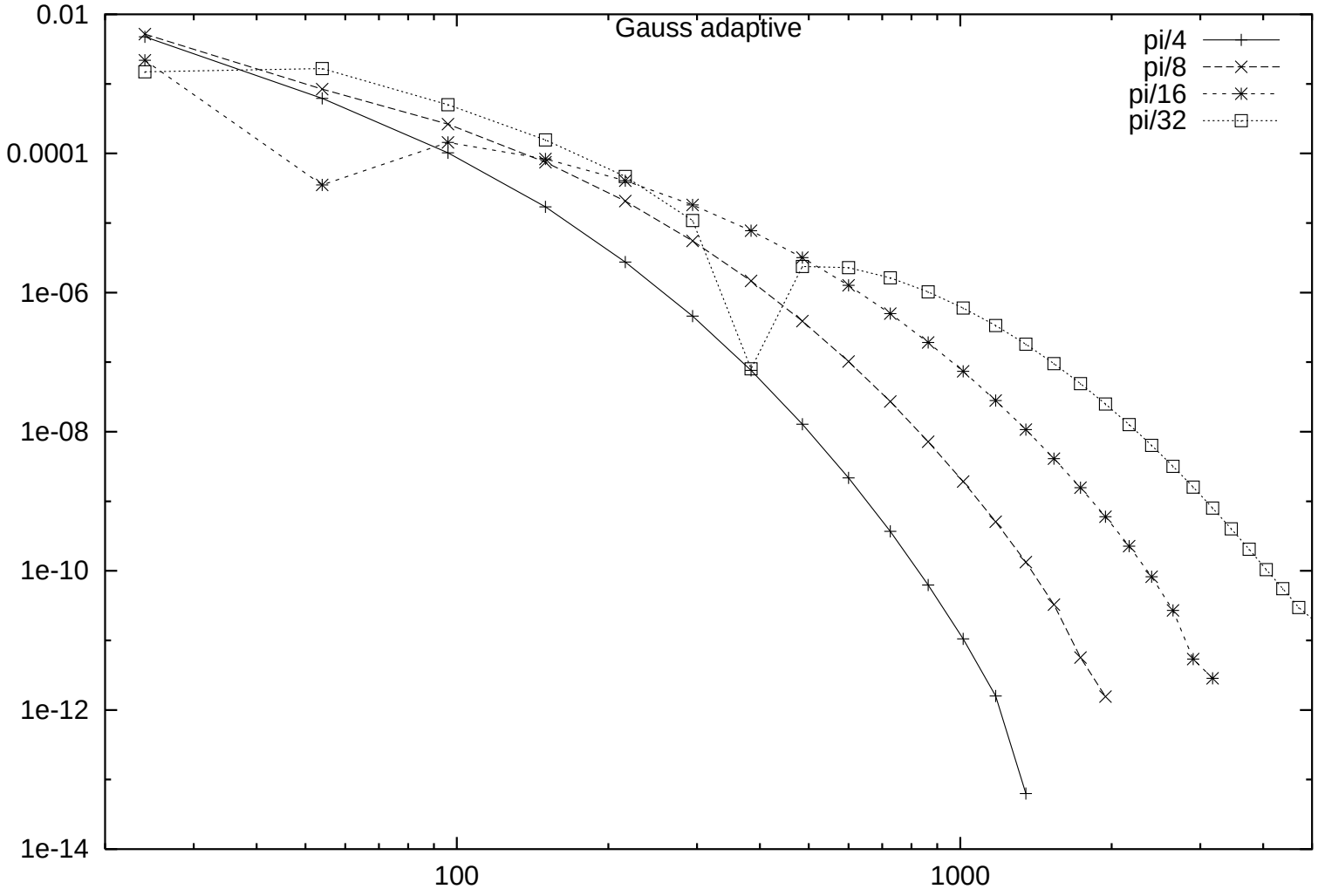

Figure 5.24: Second set of experiments: comparison of the gauss adaptive method varying Θ

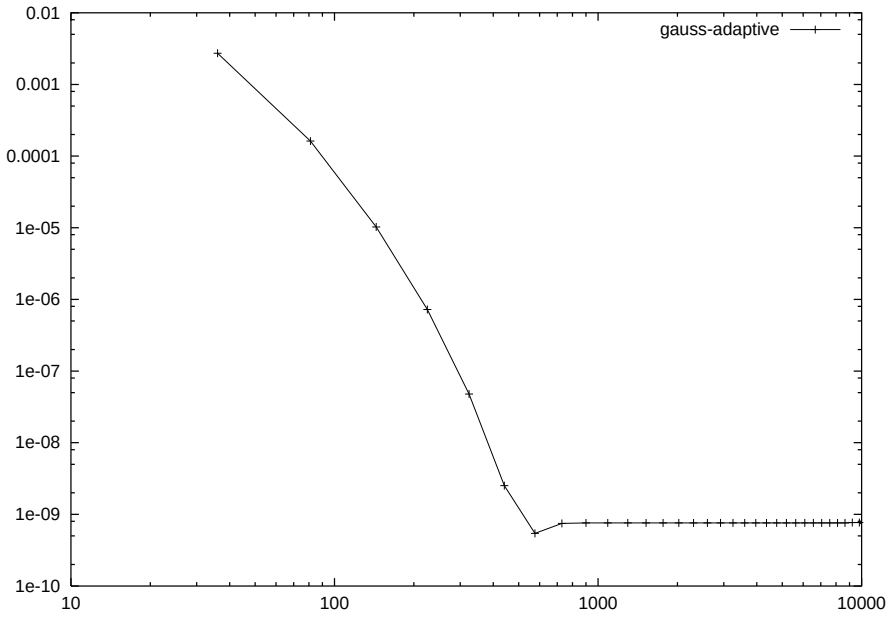

Figure 5.25: Triangles sharing one edge

5.5.4 All the methods

Figures from 5.26 to 5.33 are relative to the first set of experiments, while figures from 5.34 to 5.37 show the second experiment. All the methods tested, except the Gaussian adaptive, in the range of directions $[10, 10000]$ achieve at best error 10^{-5}. As we already said in the former subsection, the Gauss adaptive method already for 1000 directions attains errors between 10^{-8} and 10^{-14}. Dependency of the error on the angles Φ is weak for values of Φ less than $\frac{\pi}{2}$ with slightly worse performance for values greater than $\frac{\pi}{4}$. This behaviour is the opposite of the QMC and Gauss ones. The second experiments suggests that the Gauss adaptive methods is particularly effective for triangles somehow symmetrical, but good results are obtained even for highly skew triangles.

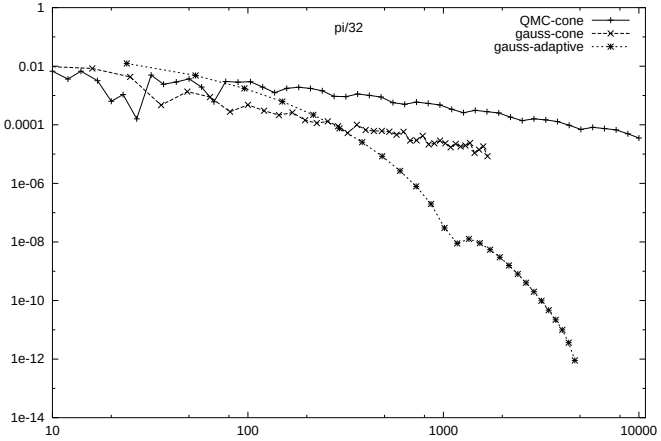

Figure 5.26: $\Phi = \frac{1}{32}\pi$

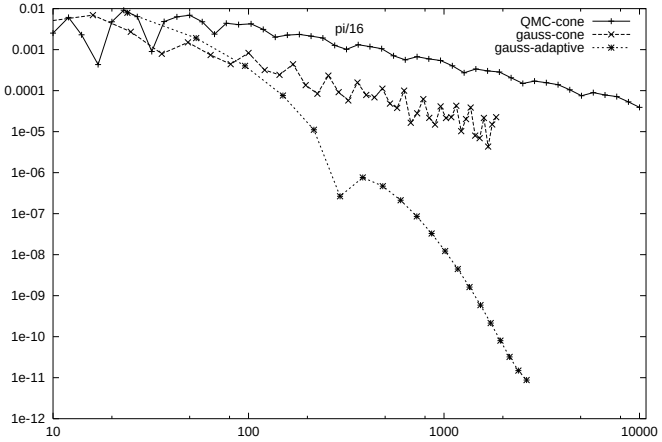

Figure 5.27: $\Phi = \frac{1}{16}\pi$

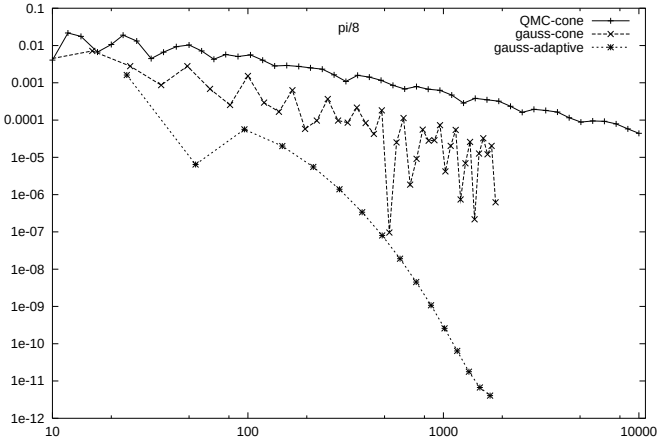

Figure 5.28: $\Phi = \frac{1}{8}\pi$

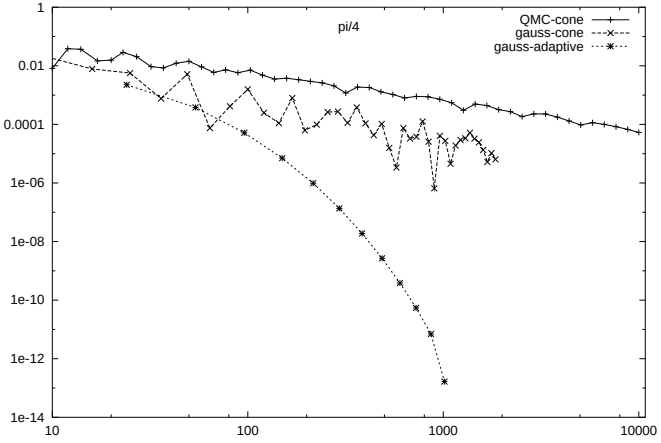

Figure 5.29: $\Phi = \frac{1}{4}\pi$

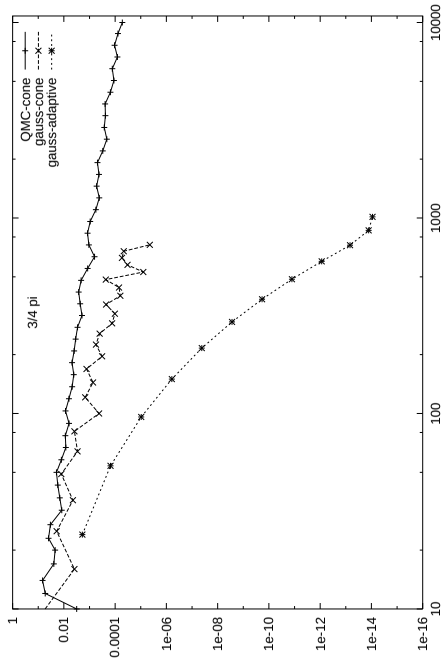

Figure 5.31: $\Phi = \frac{3}{4}\pi$

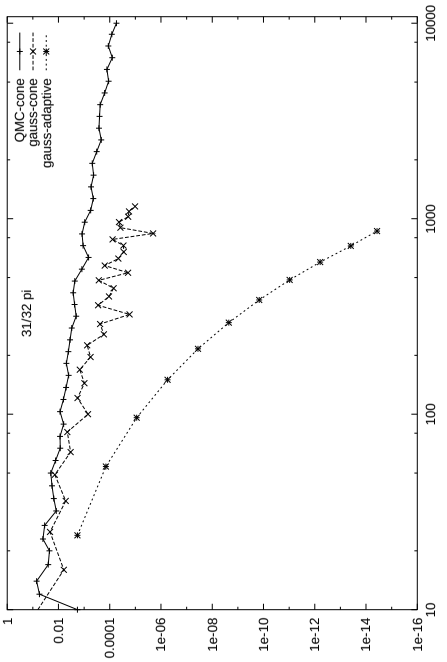

Figure 5.33: $\Phi = \frac{31}{32}\pi$

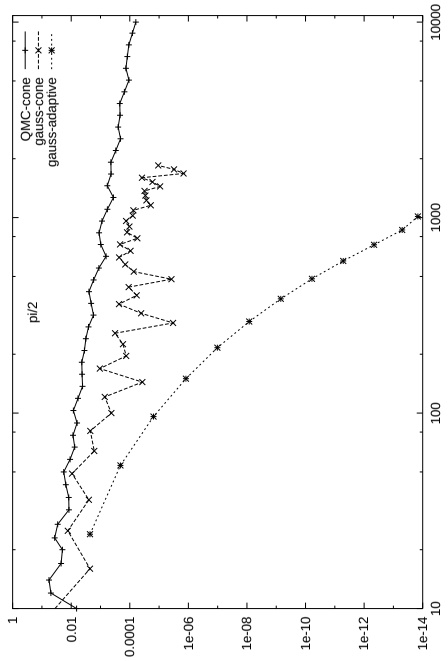

Figure 5.30: $\Phi = \frac{1}{2}\pi$

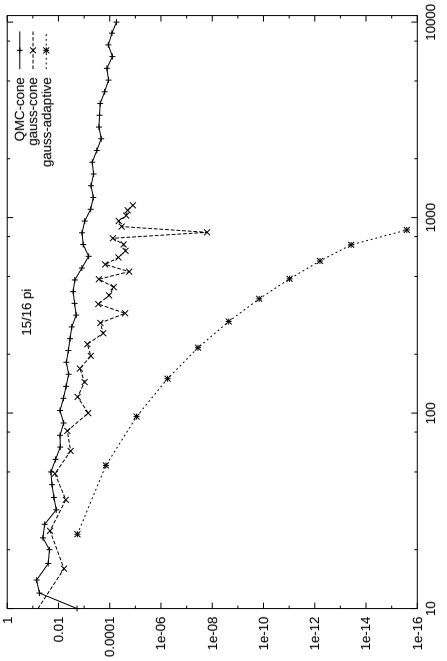

Figure 5.32: $\Phi = \frac{15}{16}\pi$

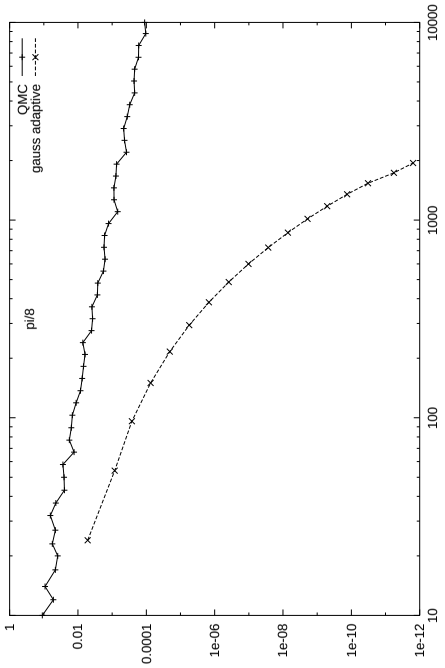

Figure 5.34: $\Theta = \frac{1}{4}\pi$

Figure 5.35: $\Theta = \frac{1}{8}\pi$

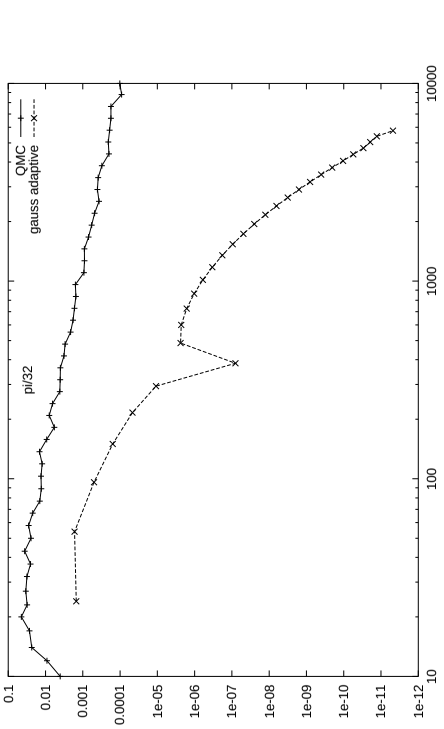

Figure 5.36: $\Theta = \frac{1}{16}\pi$

Figure 5.37: $\Theta = \frac{1}{32}\pi$

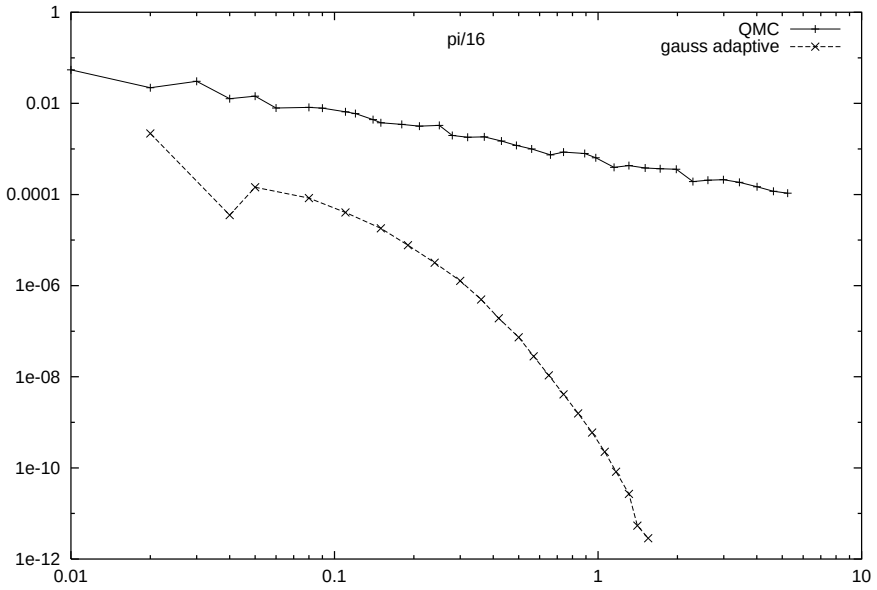

Figure 5.38: Second experiment: $\Theta = \frac{1}{16}\pi$ Computation time/Relative error

5.5.5 Relative error versus computation time

In this subsection we show the plots of the relative error against computation time. We present only two figures (5.38 and 5.39) as these are well representative for all the experiments. The Gauss adaptive algorithm is the slowest in term of the first result but it attains the best precision in each experiment in the range 0.5 to 2 seconds. These times should be considered considered only indicative since at the moment no effort has been put in optimizing the codes.

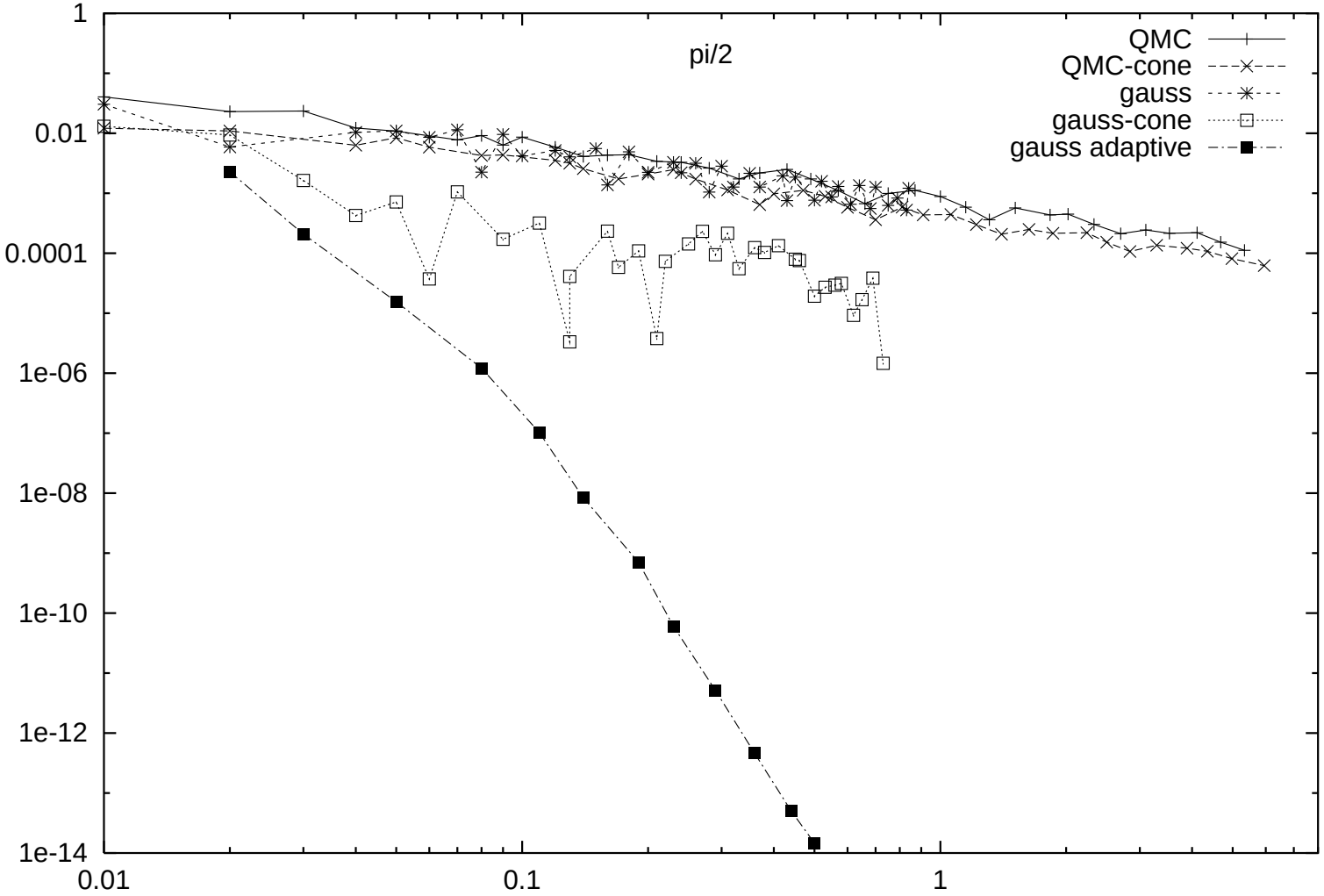

Figure 5.39: First set of experiments; dihedral angle $\Theta = \frac{1}{2}\pi$. Relative error/computation time

.

A lawyer with a case to prove
would sift the evidence, then move
It's the only thing to do
So why can't I face the facts?
It hurts too much to face the truth

Neil Tennant

Chapter 6

Boundary Element Method

Boundary value problems (BVP) are problems governed by a partial differential equation which is applicable over a domain Γ. A common boundary value problem is represented by the *potential* or *Poisson* equation defined in a domain $\Gamma \in \mathbb{R}^3$:

$$\left(\frac{\partial}{\partial x_1^2} + \frac{\partial}{\partial x_2^2} + \frac{\partial}{\partial x_3^2}\right) w(x) = h(x), \tag{6.0.1}$$

where w is the unknown function to be determined and $h(x)$ is a given function applicable over Γ. To be able to obtain a unique solution for a differential equation such as 6.0.1, we need to specify some 'boundary conditions'. If the boundary conditions take the form $w(x) = g(x)$ where g is a given function, then they are called the *Dirichlet type* and the problem is called the *Dirichlet* BVP. The potential problem with Dirichlet conditions is:

$$\begin{cases} \left(\frac{\partial}{\partial x_1^2} + \frac{\partial}{\partial x_2^2} + \frac{\partial}{\partial x_3^2}\right) w(x) = h(x) & \text{on } \Gamma, \\ w(x) = g(x) & \text{on } \partial\Gamma. \end{cases}$$

Using Robin's equation 6.0.2 (the fundamental solution) and introducing the density f as new unknown function, the potential problem with Dirichlet conditions can be reformulated as boundary integral equations, i.e., integral equations that relate only variables on the boundary of the solution domain.

$$2\pi f(p) = \int_{p' \in \partial\Gamma} \frac{\langle n(x), y - x \rangle}{||p' - p||^3} f(p')\, dp', \quad \forall p \in \partial\Gamma \tag{6.0.2}$$

In this chapter we consider two boundary value problems whose resolution through the Boundary Element Method brings to integral equations that after the discretization step involve surface-to-surface integrals of the kind we considered in section 5 (2.2.9).

6.1 Double layer ansatz for the Poisson Equation

6.1.1 Problem formulation

Let Γ be unit cube $\in \mathbb{R}^3$. Following [SS96], we consider the boundary element problem

$$\Delta u(x) = 0 \quad \text{in} \quad \Gamma, \qquad u(x) = \varphi(x) = x_1 \quad \text{in} \quad \partial\Gamma.$$

The exact solution is given by $u(x) = x_1$, indicating with x_j the j-th component of x. We employ a double layer charge density function $f(x)$, with reference to formula at page 276 of [CZ92] and to formula (85) of [SS96],

$$u(x) = -\frac{1}{4\pi} \int_{\partial\Gamma} \frac{\langle n(y), y - x \rangle}{\|x - y\|^3} f(y)\, dy \qquad x \in \Gamma. \tag{6.1.1}$$

Theorem 6.10.1 of [CZ92] states that the density function f is then the unique solution to the following boundary integral equation:

$$-2\varphi(x) - \frac{1}{2\pi} \int_{\partial\Gamma} \frac{\langle n(y), y - x \rangle}{\|x - y\|^3} f(y)\, dy = f(x) \qquad x \in \Gamma. \tag{6.1.2}$$

Here follow the steps to discretize equation (6.1.2) with a Galerkin error criterion in order to set a linear system:

$$\int_{\partial\Gamma} \frac{\langle n(y), y - x \rangle}{\|x - y\|^3} f(y)\, dy = -2\pi f(x) - 4\pi\varphi(x),$$

and integrating over $\partial\Gamma$:

$$\int_{\partial\Gamma} \int_{\partial\Gamma} \frac{\langle n(y), y - x \rangle}{\|x - y\|^3} f(y)\, dy\, dx = \int_{\partial\Gamma} -2\pi f(x)\, dx - \int_{\partial\Gamma} 4\pi\varphi(x)\, dx.$$

We now subdivide the domain $\partial\Gamma$ in patches; denoting with $I = (0, \ldots, n-1)$ the indices of the patches e_k such that $\bigcup_{k=0}^{n-1} e_k = \partial\Gamma$, we get

$$\sum_{i\in I}\sum_{j\in I}\int_{e_i}\int_{e_j}\frac{\langle n(y), y-x\rangle}{\|x-y\|^3}f(y)\,dy\,dx + 2\pi\int_{e_i} f(x)\,dx = -4\pi\int_{e_1}\varphi(x)\,dx,$$

and assuming the density functions $f(t)$ are constant on each patch, the former equation becomes:

$$\sum_{i\in I}\sum_{j\in I} f(e_j)\int_{e_i}\int_{e_j}\frac{\langle n(y), y-x\rangle}{\|x-y\|^3}\,dy\,dx + 2\pi f(e_i)\int_{e_i} 1\,dx = -4\pi\int_{e_1} x_1\,dx. \tag{6.1.3}$$

The expression $\dfrac{\langle n(y), y-x\rangle}{\|x-y\|^3}$, (with $y \in \partial\Gamma$) can be rewritten as $\dfrac{\cos_\Gamma(y, xy)}{d^2}$ where d is the distance between the points x and y, and $cos_\Gamma(y, xy)$ is the cosine of the angle formed by the vector normal to $\partial\Gamma$ in y and the vector from x to y. Let us call $K_{i,j} = \int_{e_i}\int_{e_j}\frac{\cos_\Gamma(y,xy)}{d^2}\,dx\,dy$. Formula $\int_{e_i} 1\,dx$ is the area A of the i-th patch and the right-hand terms $H_i = -4\pi\int_{e_1} x_1\,dx$ are exactly computed by symbolic integration. Thus from equation (6.1.3) follows the linear system:

$$\forall i \quad \sum_{j\in I, j\neq i} f(e_j)K_{i,j} + 2\pi f(e_i)A(e_i) = H_i, \tag{6.1.4}$$

explicitly:

$$\begin{pmatrix} 2\pi A(e_0) & K_{0,1} & K_{0,2} & \cdots \\ K_{1,0} & 2\pi A(e_1) & K_{1,2} & \cdots \\ K_{2,0} & K_{2,1} & 2\pi A(e_2) & \\ \vdots & \vdots & & \ddots \end{pmatrix}\begin{pmatrix} f(e_0) \\ f(e_1) \\ f(e_2) \\ \vdots \end{pmatrix} = \begin{pmatrix} H_1 \\ H_2 \\ H_3 \\ \vdots \end{pmatrix}$$

6.1.2 Numerical Experiments

In order to solve the linear system 6.1.4, coefficients $K_{i,j}$ (matrix M) must be computed. In [Pel97] it is shown that integrals

$$K_{i,j} = \int_{e_i}\int_{e_j}\frac{\cos_\Gamma(y, xy)}{d^2}\,dx\,dy$$

can be rewritten thanks to integral geometry transformations in integrals of the form 2.2.9 and therefore they can be approximated with high precision

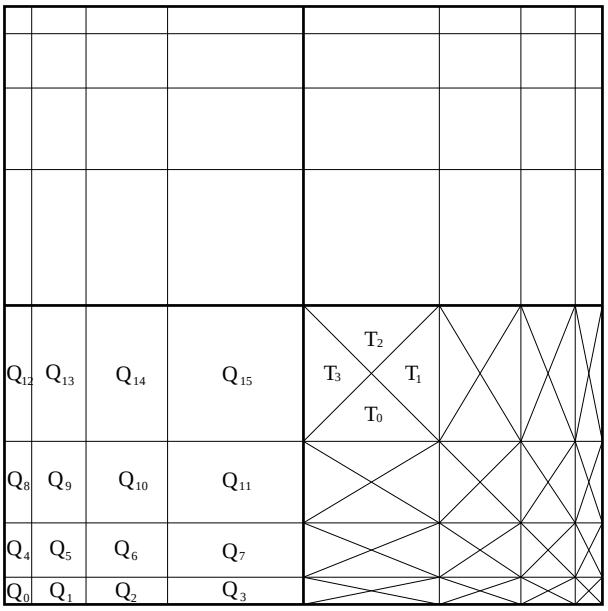

Figure 6.1: Division in patches of the faces of the cube

through the Gauss adaptive algorithm once the patches e are triangular. We produced the patches $e = t$, as shown in figure 6.1: first each face of the cube is divided into four equal squares; with reference to the low-left square of figure 6.1, each such square is then subdivided into rectangles Q_l, spacing the vertices according to the formula

$$v_s = \frac{1}{2}\left(1 - \left(\frac{s}{n}\right)^2\right) \quad \text{with } s = 0, \ldots, n,$$

where n^2 is the number of desired rectangles (for each of the 4 squares) as suggested in [Rea97]. Every rectangle is finally subdivided into 4 triangles as shown on the low-right square of figure 6.1. The first level of refinement ($n = 1$) for the cube is formed by $4 \times 4 \times 6 = 96$ patches, the second level, $n = 2$, by $4 \times 4 \times 4 \times 6 = 384$ patches and the third one has $4 \times 9 \times 4 \times 6 = 864$ triangles.

If the surface of the cube is covered by N patches, the computation of $K_{i,j}$ would involve $N^2 - N$ integrals ($i \neq j$), but several semplifications are applicable. Let us fix the patch t_k, all the coefficients $K_{k,j}$ such that t_j is in the same face as t_k and $j \neq k$, give zero as result. This is because $K_{i,j}$ represents the projection of the electrostatic force acting between the triangles t_i and t_j

on a vector orthogonal to t_j. When the triangles are coplanar such projection is null. The stiffness matrix M is then a block matrix of the form:

$$M = \begin{pmatrix} D^0 & K^{0,1} & K^{0,2} & K^{0,3} & K^{0,4} & K^{0,5} \\ K^{1,0} & D^1 & K^{1,2} & K^{1,3} & K^{1,4} & K^{1,5} \\ K^{2,0} & K^{2,1} & D^2 & K^{2,3} & K^{2,4} & K^{2,5} \\ K^{3,0} & K^{3,1} & K^{3,2} & D^3 & K^{3,4} & K^{3,5} \\ K^{4,0} & K^{4,1} & K^{4,2} & K^{4,3} & D^4 & K^{4,5} \\ K^{5,0} & K^{5,1} & K^{5,2} & K^{5,3} & K^{5,4} & D^5 \end{pmatrix} \tag{6.1.5}$$

Each block is a matrix of dimension $\frac{N}{6} \times \frac{N}{6}$; the diagonal block D^q, with $q = 0, \ldots, 5$, is a diagonal matrix with diagonal elements equal to $2\pi A(t_r)$, r being the row index; the matrix $K^{a,b}$ $(a, b = 0, \ldots, 5)$, is a fully populated matrix whose generic entry $v_{c,d}$ $(c, d = 0, \ldots, \frac{N}{6} - 1)$, is the electrostatic force acting between the c-th triangle of the a-th face of the cube and the d-th triangle of the b-th face. In table 6.1 we show our numeration for the faces. Exploiting the symmetries of the cube, it is possible to express all the matrix

Face Number	Face Vertices
0	$\{\{0,0,0\},\{1,1,0\}\}$
1	$\{\{0,0,0\},\{1,0,1\}\}$
2	$\{\{1,0,0\},\{1,1,1\}\}$
3	$\{\{1,1,0\},\{0,1,1\}\}$
4	$\{\{0,1,0\},\{0,0,1\}\}$
5	$\{\{0,0,1\},\{1,1,1\}\}$

Table 6.1: Numbering of the faces of the cube

M computing only two blocks: $K^{0,1}$ and $K^{0,5}$. In fact two faces of a cube can only be frontal or adjacent. Since our subdivision in patches of each face is totally coherent to the four symmetry axes of the square, all the adjacent faces are equivalent once the numeration of the patches has been rearranged. In particular the blocks $K^{1,3}, K^{2,4}, K^{3,1}, K^{4,2}$ and $K^{5,0}$ represent frontal faces and are totally derivable from the block $K^{0,5}$. All the other blocks $K^{a,b}$, with $a \neq b$ (and different from the former blocks), represent adjacent faces and are derivable from block $K^{0,1}$. As regards the computation of the "base" blocks $K^{0,1}$ and $K^{0,5}$, other simplifications are possible. In particular for the first it is possible to compute half the elements taking advantage of the vertical

simmetry of the faces, while in the second block, exploiting all the symmetries we compute only 1/8 of the elements. With these tricks we reduce the number of coefficients to be calculated to $\frac{5}{288}N^2 - N$ instead of $N^2 - N$.

We computed all the coefficients for the first three refinement levels of patches by the Gauss adaptive method, setting the dimension of the Gaussian grid to 15 (the algorithm produces 15^2 directions within each zone); this should guarantee the entries of the stiffness matrix to be precise with a relative error of at least 10^{-8}. The linear system 6.1.4 has then been solved by the Mathematica command `LinearSolve` (see [Wol96]), determining the densities $f(t_i)$. In order to check the error versus the known solution $u(x) = x_1$, we have to compute the value of u by equation 6.1.1 at predetermined points $X_0, \ldots, X_m$, inside the domain Γ. Equation 6.1.1 can be written as:

$$w(X_m) = -\frac{1}{4\pi} \int_{\partial\Gamma} \frac{\cos_\Gamma(y, X_m y)}{d^2} f(y)\, dy,$$

and using the discretization by patches

$$w(X_m) = -\frac{1}{4\pi} \sum_{i \in I} \int_{e_i} \frac{\cos_\Gamma(y, X_m y)}{d^2} f(y)\, dy,$$

now, since we considered the function f as constant over each patch e,

$$w(X_m) = -\frac{1}{4\pi} \sum_{i \in I} f(e_i) \int_{e_i} \frac{\cos_\Gamma(y, X_m y)}{d^2}\, dy = -\frac{1}{4\pi}\, \mathbf{f}.\mathbf{p}$$

where $\mathbf{f}$ is the vector solution of the linear system and $\mathbf{p}$ is the vector $\int_{e_i} \frac{\cos_\Gamma(y, X_m y)}{d^2}\, dy$, with $i = 0, \ldots, N-1$. We computed the vector $\mathbf{p}$ by a first step of symbolical integration followed by a numerical approximation step made by the `NIntegrate` command of the Mathematica software. As test-points we set $X_0 = \{0.5, 0.5, 0.5\}$, $X_1 = \{0.4, 0.5, 0.6\}$, $X_2 = \{0.7, 0.7, 0.7\}$ and $X_3 = \{0.2, 0.2, 0.2\}$.

In each of the test points the relative error $\left|\frac{w(X_m) - u(x)}{u(x)}\right| = \left|\frac{w(X_m) - x_1}{x_1}\right|$ is in the range 0.01 to 0.03 already for the first set of patches (16 per face) without improvements at the second and third refinement level.

6.2 Capacitance problem

Suppose we are given a conducting polyhedron C in 3-space with charge Q. We want to compute the surface density function that satisfies the equilibrium conditions for conducting bodies. We consider the istance of the problem where the polyhedron C is the unitary cube and the charge Q is 1. The discretization step is made by subdiving each face of the cube in $\frac{N}{6}$ triangular patches t_i as described in subsection 6.1.2. Moreover we suppose the surface charge density σ to be constant over each patch. Following steps similar to those applied in the previous section, we arrive to the linear system

$$\begin{cases} 2\pi\sigma_i A(t_i) = \displaystyle\sum_{j,j\neq i} (\vec{F_{ij}} \cdot \vec{n_j})\,\sigma_j & \text{for } i = 0, \ldots, N-2 \\ \displaystyle\sum_{j=0}^{N-1} \sigma_j A(t_j) = 1 & \text{for } i = N-1 \end{cases} \tag{6.2.1}$$

where σ_k indicates the constant surface charge density over the k-th patch of the cube and $A(t_i)$ is the area of the i-th triangular patch. The expression $\vec{F_{ij}} \cdot \vec{n_j}$ corresponds to the projection of the electrostatic force acting between the patches t_i and t_j on a vector orthogonal to t_j. The system 6.2.1 is then of the form

$$\begin{pmatrix} 2\pi A(t_0) & K_{0,1} & K_{0,2} & \cdots \\ K_{1,0} & 2\pi A(t_1) & K_{1,2} & \cdots \\ K_{2,0} & K_{2,1} & 2\pi A(t_2) & \\ \vdots & \vdots & & \ddots \\ A(t_0) & A(t_1) & A(t_2) & \cdots \end{pmatrix} \begin{pmatrix} \sigma_0 \\ \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_{N-1} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix}.$$

Therefore the matrix has a block structure much similar to the matrix M (6.1.5) displayed in subsection 6.1.2 so that two blocks are enough to specify all of it (excepting the elements of the diagonal and the last row). The entries can be efficiently computed by the Gauss adaptive algorithm (section 5).

Once the unknowns have been determined, we have an approximated representation of the distribution of charge on the surface of the cube C. It is then possible to express the electrostatic potential in a point x internal to C through the formula

$$V(x) = \int_{y\in\partial C} \frac{\sigma(y)}{|x-y|}\, dy.$$

Point	Level 1	Level 2	Level 3
$\{0.5, 0.5, 0.5\}$	0.0244953	0.00760677	0.0038885
$\{0.6, 0.5, 0.55\}$	0.0244991	0.00760802	0.00388898
$\{0.4, 0.4, 0.4\}$	0.024426	0.00758319	0.00387941
$\{0.3, 0.4, 0.3\}$	0.0240427	0.00745473	0.00382987
$\{0.2, 0.8, 0.8\}$	0.0204362	0.00562386	0.00309756
$\{0.1, 0.4, 0.9\}$	0.0160977	0.00651248	0.00340792
$\{0.1, 0.1, 0.1\}$	0.0012754	0.00002271	0.00068473

Table 6.2: Cube capacitance. Relative error/refinement level.

In our case, discretizing the surface in patches t_i and assuming σ constant on each patch, we get

$$V(x) = \sum_i \sigma(t_i) \int_{t_i} \frac{1}{|x-y|}\, dy.$$

If the equilibrium conditions are satisfied, in the interior of a conducting body we know the potential to be constant. The problem of analytically compute this constant has never been solved. In order to get a reference value we use the formula that relates the electrostatic potential V and the charge Q of a body with its capacitance C:

$$C = \frac{q}{V}.$$

The problem of finding the capacitance of the unitary square and cube seems to be one of the oldest problems in electrostatics since Maxwell got the first results [Max78]. The best approximation so far should be the one given by Goto *et al.* in [GSY92], which is considered reliable up to 10^-7.

We computed the coefficients of the stiffness matrix thanks to the Gauss adaptive method run with 15 direction as dimension of the Gaussian grids to be used. We subdivided the cube in 96, 384 and 864 patches. Given a point X we calculated the integrals $\int_{t_i} \frac{1}{|X-y|}\, dy$, numerically integrating with Mathematica the results obtained by a first step of symbolical integration. We set seven test-points in which we computed the electrostatic potential consequently obtaining estimates for the cube capacitance. In table 6.2 we show the relative error for out approximations of the capacitance with respect to the value taken from [GSY92].

6.3 Discussion of results

The results of the experiments described in this chapter are not as satisfactory as we expected. In a first time we discretized the cube subdividing each face in triangular patches in such a way it was not possible to drastically reduce the number of coefficients of the stiffness matrix to be computed. This led to big calculations even for low dimensional meshes. Besides we did not use the strategy of thickening the patches near the edges of the faces and particularly toward the vertices. In table 6.3 we report the results for the Capacitance experiment made with such patches. The relative error improved with the

Patches	Relative error
12	0.0488067
96	0.0256226
192	0.0187804
300	0.0146265

Table 6.3: Meshing strategy without symmetries and thickening: patches against relative error for the Capacitance problem

new meshing strategy but the convergence is still too slow. Analysing articles such as [HS93], it appears evident that the shortcoming step in our method is the assumption the function σ to be constant over each patch. As the number of meshes increase, it is necessary to use polynomials in place of constants especially in the patches near the edges and vertices.

Was it worth it?
Yes, it's worth living for
Was it worth it?
Yes, it's worth giving more

Neil Tennant

Chapter 7

Conclusions

We started analysing the following problem, the computation of the electrostatic force between two convex bodies in 3-space, whose classical solution is an integral with a singularity which can reside within the integration domain. Recently the problem has been tackled by a new geometrical approach, obtaining integrals free from singularities and defined by means of geometrical entities.

In this thesis we give explicit analytic formulas for the kernel function of this new kind of integrals when the input bodies are tetrahedra or triangles, assuming constant the densities of charge. We then present tools to exactly and efficiently compute such kernels. Thanks to these tools we have been able to develop and implement several algorithms, among which an adaptive one, for finding good approximations of the electrostatic force. With reference to the case of triangles, in all the experiments we achieve a relative error in the range 10^{-9} to 10^{-15} with computation time in the range 0.4 to 2.7 seconds.

Finally we consider the problem of computing the charge distribution on the surface of a conductor. Through the boundary element method the surface is subdivided in triangular patches; under the assumption the charge density is constant over each patch, the entries of the stiffness matrix are calculated by the Gauss adaptive method. It turns out the supposition of constant density is too strict in order to obtain solutions with high accuracy. It will be central in future developments to express the electrostatic force between bodies with non-constant charge densities.

The results for the case of fully dimensional bodies are open to improvements once an adaptive method similar to the one for triangles is used.

Appendix A

Algorithms

Here we show in detail some of the algorithms used throughout the thesis.

A.1 Random point on 3-dimensional sphere with radius one

Knuth

Algorithm taken from [Knu69, pag. 116-117 and 130-131]:

1. Let U_1 and U_2 two independent random variables uniformly distributed between zero and one.
2. Set $V_1 = (2\,U_1) - 1$ and $V_2 = (2\,U_2) - 1$.
3. Set $S = V_1^2 + V_2^2$.
4. If $S \geq 1$ then start again from step 1; else the desired point on the surface of a globe is $(\alpha V_1, \alpha V_2, 2S - 1)$, where $\alpha = 2\sqrt{1 - S}$.

The Trig method

Algorithm taken from the "Frequent Asked Questions" (FAQ) of the comp.graphics.algorithms newsgroup.

1. Let U_1 and U_2 two independent random variables uniformly distributed between zero and one.

2. Set $z = (2\,U_1) - 1$.

3. Set $t = 2\,\pi\,U_2$.

4. Let $r = \sqrt{(1 - z^2)}$.

5. Let $x = r\cos(t)$.

6. Let $y = r\sin(t)$.

The triple (x, y, z) is the requested point.

A.2 N-element Hammersley point set

For an integer $b \geq 2$, we put $Z_b = 0, 1, \ldots, b-1$. Every integer $n \geq 0$ has a unique digit expansion

$$n = \sum_{j=0}^{\infty} a_j(n)\, b^j \tag{A.2.1}$$

in base b, where $a_j(n) \in Z_b$ for all $j \geq 0$ and $a_j(n) = 0$ for all sufficiently large j; i.e., the sum in (A.2) is finite.

Definition A.2.1 (Radical-inverse function) *For an integer $b \geq 2$, the* radical-inverse function ϕ_b in base b *is defined by:*

$$\phi_b(n) = \sum_{j=0}^{\infty} a_j(n)\, b^{-j-1} \qquad \textit{for all integers } n \geq 0,$$

where n is given by its digit expansion (A.2) in base b.

Note that $\phi_b(n) \in [0, 1)$ for all $n \geq 0$.

Definition A.2.2 *For a dimension $s \geq 2$ and for integers $N \geq 1$ and $b_1, \ldots, b_{s-1} \geq 2$, the* N-element Hammersley point set in the bases $b_1, \ldots, b_{s_1}$ *is given by*

$$\left(\frac{n}{N}, \phi_{b_1}(n), \ldots, \phi_{b_{s-1}}(n)\right) \in [0, 1)^s \qquad \textit{for } n = 0, 1, \ldots, N-1$$

A.3 Convex Hull - Jarvis's march

Definition A.3.1 *The* convex hull *of a set Q of points is the smallest convex polygon P for which each point in Q is either on the boundary of P or in its interior.*

Jarvis's march ([O'R94]) computes the convex hull of a set Q, (CH(Q)), by a technique known as *package wrapping* (or *gift wrapping*). The algorithm runs in time $O(nh)$ where h is the number of vertices of CH(Q), and n is the number of points in Q. Starting from the point p_0 with the smallest y-coordinate (breaking ties with the biggest x-coordinate), Jarvis's march builds a sequence $\{p_0, p_1, \ldots, p_{h-1}\}$ of the vertices of CH(Q). At the k-th step, after having determined the vertex p_{k-1}, it is searched the point in Q with the least polar angle. The algorithm can be totally implemented by left-turn tests.

Bibliography

[Arv95] J. Arvo. Stratified sampling of spherical triangles. *Computer Graphics*, 29(Annual Conference Series):437–438, 1995.

[BBCM92] Roberto Bevilacqua, Dario Bini, Milvio Capovani, and Ornella Menchi. *Metodi Numerici*. Zanichelli, Bologna, 1992.

[BL73] K.J. Binnes and P.J. Lawrenson. *Analysis and computation of electric and magnetic field problems*. Pergamon Press, Oxford, 1973.

[CZ92] G. Chen and J. Zhou. *Boundary Element Methods*. Academic Press, London, 1992.

[DR84] Philip J. Davis and Philip Rabinowitz. *Methods of Numerical Integration*. Academic Press, London, second edition, 1984.

[Duf82] M.G. Duffy. Quadrature over a pyramid or cube of integrands with a singularity at a vertex. *SIAM J. Numerical Analysis*, 19(6):1260–1262, 1982.

[EE92] T.O. Espelid and A. Genz (Eds.). *Numerical integration: recent developments, software and applications*, volume 357. Kluwer Academic Publishers, London, 1992.

[Eng80] H. Engels. *Numerical Quadrature and Cubature*. Academic Press, New York, 1980.

[ES] Stefan Erichsen and Stefan A. Sauter. Efficient automatic quadrature in 3-d galerikin bem.

[GSY92] E. Goto, Y. Shi, and N. Yoshida. Extrapolated surface charge mathod for capacity calculation of polygons and polyhedra. *J. Computational Physics*, 100:105–115, 1992.

[HH64] J. M. Hammersley and D. C. Handscomb. *Monte Carlo Methods.* Methuen, London, 1964.

[HS93] W. Hackbusch and S.A. Sauter. On the efficient use of the galerkin-method to solve fredholm integral equations. *Applications of mathematics*, 38:301–322, 1993.

[JVV86] M.R. Jerrum, L.G. Valiant, and V.V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43:169–188, 1986.

[KE92] P. Keast and G. Fairweather (Eds.). *Numerical integration: recent developments, software and applications*, volume 203. Kluwer Academic Publishers, London, 1992.

[Knu69] D.E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming.* Addison-Wesley, Reading, MA, 1969.

[LM80] J.N. Lyness and G. Monegato. Quadrature error functional expansions for the simplex when the integrand function has singularities at vertices. *Mathematics of Computation*, 34(149):213–225, 1980.

[Max78] J.C. Maxwell. *Electrical Research of the honorable Henry Cavendish.* Cambridge University Press, Cambridge, 1878.

[Nie92] Harald Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods*, volume 63 of *CBMS-NSF regional conference series in Appl. Math.* SIAM, Philadelphia, 1992.

[O'R94] J. O'Rourke. *Computational Geometry in C.* Cambridge University Press, 1994.

[Pel96] Marco Pellegrini. Electrostatic fields without singularities: Theory and algorithms. In *Proc. 7th ACM-SIAM Sympos. Discrete Algorithms*, pages 184–191, 1996. Extended version in Tech. Report IMC B4-96-02, Istituto Matematica Computazionale, CNR, Pisa, Italy.

[Pel97] Marco Pellegrini. Electrostatic field without singularities: Theory, alghoritms and error analisys. Technical Report IMC B4-97-15, Istituto Matematica Computazionale, CNR, Pisa, Italy, November 1997.

[PS85] F.P. Preparata and M.I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, NY, 1985.

[Rea97] F.H. Read. Improved extrapolation technique in the boundary element method to find the capacitances of the unit square and cube. *J. Computational Physics*, 133:1–5, 1997.

[San67] L.A. Santalo. Integral geometry. In S.S. Chern, editor, *Studies in Global Geometry and Analysis*. The Mathematical Association of America, 1967.

[Sch72] W.M. Schmidt. Irregolaries of distributions. *Acta Arith.*, VII(21):45–50, 1972.

[Sch94] C. Schwab. Variable order composite quadrature of singular and nearly singular integrals. *Computing*, 53:173–194, 1994.

[SS66] A.H. Stroud and D.H. Secrest. *Gaussian Quadrature Formulas*. Prentice-Hall, Englewood Cliffs, 1966.

[SS96] C. Schwab S.A. Sauter. Quadrature for hp-galerkin bem in 3-d. Technical Report TR-96-02, Seminar for Applied mathematics, Department of Mathematics, ETH, Zurich, 1996.

[SW92a] C. Schwab and W.L. Wedland. Kernel properties and representation of boundary integral operators. *Math. Nachr.*, 156:187–216, 1992.

[SW92b] C. Schwab and W.L. Wedland. On numerical cubature of singular surface integrals in boundary element methods. *Num. Math.*, 62:343–369, 1992.

[Wol93] Stephen Wolfram. *Mathematica: a system for doing mathematics by computer*. Addison Wesley, 1993.

[Wol96] Stephen Wolfram. *The Mathematica book*. Addison Wesley, 1996.

[Zho93] P.B. Zhou. *Numerical Analysis of Electromagnetical fields*. Springer Verlag, 1993.