Efficient Automated Computation of Tree-Level Amplitudes in QCD and QED

Diploma Thesis by **Daniel Götz**
May 2011

Institut für Physik
Staudingerweg 7, 55128 Mainz, Germany
Johannes Gutenberg-Universität Mainz

Contents

Chapter 1

Introduction

In March last year, the Large Hadron Collider (LHC) reached a center of mass energy of 7 TeV for the collision of two proton beams. At these previously unmatched energies, physicists hope to find signs of the last missing particle of the Standard Model of particle physics which has not yet been discovered: The higgs particle. In addition, searches for physics beyond the Standard Model, such as supersymmetry, can be approached properly for the first time. Next to these desired signals, however, there is always a large amount of background events. In order to identify the signals properly, the background has to be known as well as possible. Since the LHC collides protons consisting of quarks and gluons, this background is strongly dominated by QCD events. It is common that such events have a high number of particles in the final channel, so-called jets, which may include electroweak bosons.

Figure 1.1 presents a rough outline of what happens during a collision of two hadrons and how this is described theoretically. The two hadrons do not participate in the collision as single entities, instead their elementary constituents, quarks and gluons, enter what is called the *hard scattering* process. So-called *parton distribution functions* (PDFs) give the probability of finding a specific parton in the hadrons. These are universal quantities which can be measured once and reused for all experiments. The hard scattering describes the transition from the two initial partons into a (theoretically) arbitrary number of final particles; this is described by the S-matrix theory of quantum field theory. This process is followed by *parton showers* where the emitted partons radiate additional partons which are collinear, so-called soft partons. In the final stage, these partons hadronize, i.e. they form new hadrons which can possibly decay.

The goal of this thesis is to develop algorithms and methods which allow for an efficient automated computation of parts of the hard scattering process. We say "parts" here, since the application of perturbation theory yields an infinite series of terms, an expansion in the coupling constant of the interaction. Of course, such a computation is impossible, hence, we restrict ourselves to leading order (or tree-level) calculations. The program which we develop builds on the existing library *particle scattering* by S. Weinzierl and is intended as a part of a more elaborate package which also includes parton showers, etc.

Nowadays, the computation of tree-level amplitudes is well established. There exist many different techniques to perform such calculations analytically and over the

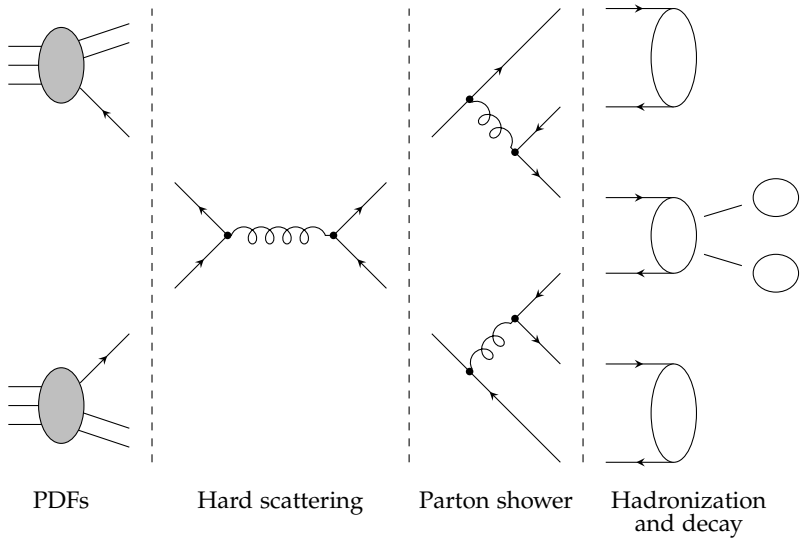


Figure 1.1: A rough outline of a hadron-hadron collision and its theoretical description. The picture is based on a figure in [7].

years these methods have been refined to simplify computations more. However, with an increasing number of final state particles, the number of tree-level terms grows significantly, quickly reaching numbers which make analytical calculations a hopeless business. This is where automated computation enters the game: A computer does not worry about the number of terms. At present, there are many programs available which provide for the computation of tree-level amplitudes, most notably the well-established *MadGraph* [8–11], which was joined by many others in recent years, for example *Comix* [12] or *CAMORRA* [13]. So, we have to ask ourselves whether yet another program is really neccessary?

Of course, the answer to this question is yes. To justify the answer, however, we have to look a bit further. As stated before, the program developed in this thesis is only a part of a bigger package. Other parts of the package are occupied with the computation of next-to-leading order (NLO) contributions to the hard-scattering process. It is very desirable to include such contributions since leading-order calculations suffer from a strong dependence on the renormalization scale; this scale determines the strength of the coupling. Thus, leading order calculations have larger uncertainties concerning the used physical parameters. An inclusion of NLO contributions reduces this uncertainty. Furthermore, leading-order computations approximate jets by one single parton, a crude approximation; NLO models jets as one to three partons. But why do we discuss NLO computations in a thesis about leading order? The answer is that NLO contributions can be decomposed into three parts, a real emission part, a virtual contribution and a collinear subtraction term, where both the real emission and the virtual contribution depend on leading-order matrix elements with color- and spin-correlations (see e.g. [7, 14, 15]). Hence, NLO computations require the knowledge of leading order matrix elements. In this sense, the algorithms we present here form the basis for the additional computation of NLO contributions.

What do we require from the program we present here? We will not discuss color- and spin-correlations, but we aim for computations with many external legs and we

require that they are reasonably fast, i.e. we try to optimize the computations as well as possible. In leading order, the external particles of a process often restrict the possible interactions, for example a process with only gluons as external particles can only contain purely gluonic interactions; this provides for an optimized and simpler implementation than the inclusion of quarks. Hence, we will implement separate routines for such cases, which will be called *(interaction) models*.

The structure of this thesis is as follows: First, we establish the physical background for the computation of amplitudes, including decompositions and formalisms which make the task more systematic. In chapter 3, we shortly present the existing parts of *particle scattering* and explain the general structure of the present program and how it builds on *particle scattering*. Based on the physical foundation from chapter 2, we will identify three basic ingredients for the computation of amplitudes. These can be implemented separately for the different models we want to employ. Before going into details of the specific models, however, we first direct our interest to a few general optimizations. Then, we explain the ideas and algorithms of the different models in chapters 4 to 7; these are ordered in such a way that each model builds on the previous model and extends it by another aspect. Along this way, we illustrate several optimization techniques for the specific models. Finally, chapter 8 presents explicit results of the program: First, we check the quality of the proposed optimizations before we compare the results of the program to reference results obtained by *MadGraph*, among others. This is followed by a short conclusion and an outlook. An appendix presents the Feynman rules which are used in the program.

Chapter 2

Physical Basics

Before entering the details of the present program, the physical basis has to be established. Therefore, this chapter presents an introduction to all tools neccessary for the understanding of the program. In the course of this, it will be clarified *what* an experimentalist is interested in at a collider such as the LHC. Once this has been established, the neccessary theory of quantum field theory will be unfolded using the formalisms which are applied to the program.

2.1 Computation of Squared Amplitudes

The most important method to probe the world of elementary particles is scattering experiments. At the LHC, two highly relativistic proton beams (currently with a center of mass energy $E_{\text{cm}} = 7\,\text{TeV}$) are collided so that the partons composing the protons scatter off each other in several detectors, such as ATLAS or CMS, along the beam line. In such collision experiments one measures the *cross section*, a process-intrinsic value which determines the probability of a certain particle configuration. Here, particle configuration denotes the number and types of outgoing particles.

Such cross sections σ are directly related to a squared quantity of the corresponding quantum field theory, the so-called *matrix element* or *(scattering) amplitude* $\mathcal{A}$. The following formula presents the general dependence of a differential cross section for an arbitrary $2 \to n$ process on $|\mathcal{A}|^2$, meaning that two particles collide and n arbitrary particles result from the collision:

$$\begin{aligned} \mathrm{d}\sigma = \frac{1}{2E_A 2E_B \left|v_A - v_B\right|} \left(\prod_f \frac{\mathrm{d}^3 p_f}{(2\pi)^3} \frac{1}{2E_f}\right) \times \\ \times \left|\mathcal{A}(p_A, p_B \to \{p_f\})\right|^2 (2\pi)^4 \delta^{(4)}(p_A + p_B - \sum p_f) \end{aligned} \tag{2.1}$$

Therein, indices A and B denote the particles in the incoming channels, f denotes all final state particles; v is the velocity of the particle as viewed from the laboratory frame and E its energy. We can also see a fourdimensional Minkowski delta distribution, ensuring momentum conservation. Moreover, we find a product of Lorentz-invariant phase space measures for the outgoing particles.

The only non-trivial variable appearing in equation (2.1) is the squared amplitude. Hence, the question arises of how this can be calculated. This is not a simple question;

to answer it in full detail, one has to dig rather deep into the details of quantum field theory. We will skip this here since there exist many excellent and detailed treatments of this topic.[1]

However, the answer to that question is most important since the goal of the program presented in this thesis is to provide squared amplitudes. So, what is a matrix element? A matrix element is a measure for the probability of the process $A, B \to \{f\}$ with the corresponding momenta. From classical mechanics we know that probabilities can be expressed in the form

$$P = \left|\langle\{\phi_f\}|\phi_A\phi_B\rangle\right|^2 ,$$

where ϕ loosely denotes a particle state (some sort of wavefunction). Applying the rules of quantum field theory and perturbation theory, this expression finally leads to long and complicated expressions. However, these expressions are of a regular and not so complicated structure, a feature discovered by Feynman in 1949 [18]. He found a way to exhibit this structure in a graphical way, the so-called *Feynman diagrams*, which appear ubiquitously in today's physical world. A Feynman diagram consists of a few basic building blocks emanating from the Lagrangian density of the theory:

Vertices: These are terms describing particle interactions which are represented as dots where the interacting particles meet. In the Standard Model of particle physics, there are only vertices with three or four particles.

Virtual particles: These are terms describing the probability for propagation of a particle, hence the name *propagators*. For each type of particle, there exists one such propagator. Virtual particles need not be *on-shell*, i.e. they do not have to satisfy the relation $p^2 = m^2$ for fourmomentum p and mass m. They are *not* the external particles described above. In a diagram, virtual particles are displayed as lines connecting two vertices to each other.

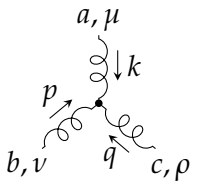
External particles: In contrast to virtual particles, external particles are the ingoing and outgoing particles A, B and $\{f\}$. They are described by their spin structure, i.e. spinors for spin $1/2$ particles and polarizations for vector bosons. Diagrammatically, they appear as lines connected only to one vertex at one end.

In this thesis, we will distinguish different particles by giving their particle lines different shapes: Fermions are represented as straight lines with an arrow indicating the particle flow (), photons as wavy lines () and gluons as curly lines ().

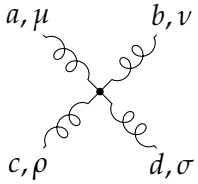
The number and structure of these ingredients are given by the specifics of the theory or the interaction model which one applies. Once a diagram has been drawn, it can be easily translated into a mathematical equation by means of the so-called *Feynman rules*. These rules give instructions on how to replace the above mentioned building blocks of a diagram by formulas. The following list presents the Feynman rules which

[1] See e.g. Peskin and Schroeder's textbook [16] or Zee's textbook [17] for a different approach using Functional Integration.

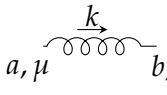
only include gluons as an example:



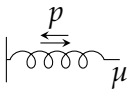
$$= g_3 f^{abc} \left[g^{\mu\nu}(k-p)^\rho + g^{\nu\rho}(p-q)^\mu + g^{\rho\mu}(q-k)^\nu \right] \tag{2.2a}$$



$$\begin{aligned} = \; & -ig_3^2 \big[f^{abe} f^{cde} (g^{\mu\rho} g^{\nu\sigma} - g^{\mu\sigma} g^{\nu\rho}) + \\ & f^{ace} f^{bde} (g^{\mu\nu} g^{\rho\sigma} - g^{\mu\sigma} g^{\nu\rho}) + \\ & f^{ade} f^{bce} (g^{\mu\nu} g^{\rho\sigma} - g^{\mu\rho} g^{\nu\sigma}) \big] \end{aligned} \tag{2.2b}$$



$$= \frac{-ig_{\mu\nu}}{k^2 + i\epsilon} \delta^{ab} \tag{2.2c}$$



$$= \epsilon_\mu(p) = \epsilon_\mu^*(p) \tag{2.2d}$$

Here, g_3 denotes the QCD coupling constant and f^{abc} denote the $SU(3)$ structure constants which describe the color part of QCD, they arise from the local invariance of QCD under the gauge group $SU(3)$. The arrows in diagram (2.2a) indicate the direction of the momentum. Diagrams (2.2a) and (2.2b) describe the vertices of the theory, (2.2c) is the propagator and (2.2d) is the polarization for external gluons; note that in the above notation, the polarization has to be chosen as a *real*-valued fourvector so that initial and final channels are equal. In addition to these graphical rules, one also has to ensure momentum conservation at each vertex and divide by the symmetry factor[2].[3]

To compute a process, one simply has to draw all Feynman diagrams with the given external particles that can possibly be drawn using the given Feynman rules. "Simply" is a little exaggerated in this case, since there is always an infinite number of diagrams — by increasing the number of vertices and propagators the diagrams can get arbitrarily complicated. This is where perturbation theory enters the game: In this thesis, we want to compute processes at tree-level only, i.e. diagrams to leading order in the coupling constant(s)[4] of the theory. When looking at the Feynman rules in (2.2), it becomes obvious that only the vertices carry coupling constants (which is natural, since they describe couplings). So, the order of a diagram is given by the number of vertices. More specifically, a tree-level diagram can have maximally $n-2$ vertices where n is the number of external particles; we say "maximally" here since the four

[2]The symmetry factor takes care of internal symmetries of a diagram. E.g. if we have a diagram which is symmetric under the exchange of two virtual particles, we get a symmetry factor of 2. A good explanation can be found in [16], chapter 4.4.

[3]In principle, one also has to integrate over each undetermined momentum which might appear for propagators. However, this only occurs when there are loops in the diagram. Since this does not happen at tree-level, we will not have to concern ourselves with these problems.

[4]When mixing QCD with photons, for example, there is both the strong and the QED coupling constant.

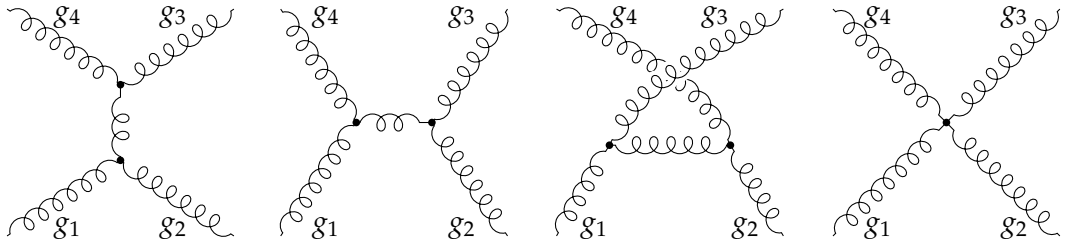


Figure 2.1: Leading order contributions to $gg \to gg$ scattering.

no. of gluons:	4	5	6	7	8	9	10
no. of diagrams:	4	25	220	2485	34 300	559 405	10 525 900

Table 2.1: Number of Feynman diagrams for a process with the given number of gluons. Table taken from [19, 20].

gluon vertex comes with a power of two in the coupling. Independent of the number of vertices, however, each tree-level process always has $n-2$ coupling constants which can be accepted easily after looking at a few examples.

To make this theoretic analyzation more practical, we will now take a look at the process $gg \to gg$. There are four tree-level diagrams, shown in figure 2.1: One diagram each for s-, t- and u-channels (diagrams (a) to (c)) and one diagram with the four gluon vertex (d). According to the above rules, it is not difficult to turn these diagrams into algebraic formulas, however, it becomes quickly obvious that it is a tedious exercise to do this — especially when thinking about reducing the resulting expressions to sensible results. As the number of external particles grows, the number of Feynman diagrams blows up, see table 2.1, making an evaluation of this kind almost impossible.

This is also of special importance when considering an implementation in terms of a numeric computer program: Once the diagrams get replaced by formulas, there is little left of the systematics of Feynman diagrams. Moreover, we have to note that within these formulas, different mathematical structures appear in an unsystematic way (at least not obviously systematic): There are color structures f^{abc} and δ^{ab} with color indices in the fundamental representation mixed with space-time structures recognizable by Lorentz indices.

2.2 Color Decomposition

To solve this problem, we will not ask how to implement these structures efficiently, but we will ask another question: How can we reformulate the technique of Feynman diagrams and Feynman rules so that it becomes more simple and systematic?

As an answer to this, the present section introduces a technique called *color decomposition*, which was first found by Berends and Giele in 1987 [21] and further developed later. Note that we deal with gluon amplitudes unless otherwise stated.

2.2.1 Separating Color Indices from Kinematics

The main goal of this technique is to organize the color factors in a more systematic way than the way which they appear in the Feynman rules. In fact, all color structures carrying color indices can not only be organized systematically, but they also get separated from the kinematical parts of the amplitude. The following form of color decomposition is not the original Berends and Giele equation (since it is no longer widely used), but the color decomposition found in 1988 by Mangano, Parke and Xu [22]:

$$\mathcal{A}(g_1, \ldots, g_n) = \sum_{S_{n-1}} \mathrm{Tr}\left\{T^{a_1} \ldots T^{a_n}\right\} A(1, \ldots, n). \tag{2.3}$$

Therein, the T symbols denote the generators of the symmetry group in the fundamental representation[5]; the trace describes the full color structure. The sum runs over all non-cyclic permutations $P \in S_n/Z_n = S_{n-1}$, where the permutation acts on all particle indices, i.e. the indices of the $SU(3)$ generators and the parameters of the newly introduced symbol A. A is called a *partial* or *color-ordered amplitude*. As mentioned above, it does not contain any color indices.[6] Note that equation (2.3) introduced a new notation, where an index i generally represents all relevant information on the respective particle, such as momentum and helicity: $i \mathrel{\widehat{=}} (p_i, \lambda_i)$. Hence, partial amplitudes depend on momenta and helicities. They satisfy several properties, among them are:

- Gauge invariance,
- Invariance under cyclic permutations, e.g. $A(1, 2, \ldots, n) = A(2, 3, \ldots, n, 1)$,
- Inversion rule: $A(1, 2, \ldots, n) = (-1)^n A(n, n-1, \ldots, 1)$,
- Dual Ward Identity:
$$\sum_{S_{n-1}} A(1, 2, \ldots, n) = 0.$$

We will illustrate that equation (2.3) indeed yields the same results as the Feynman diagram method, however, we will use another kind of color decomposition. Since the first idea of Berends and Giele, many papers concerning color decompositions have arisen. The above decomposition in the fundamental representation has the drawback that the number of partial amplitudes grows as $(n-1)!$ with the number of particles, whereas there are relations, the Kleiss-Kuijf relations [20], which reduce the number of independent partial amplitudes to $(n-2)!$.[7] This fact is exploited by a decomposition employing the adjoint representation of $SU(N)$ [24, 25], which makes it more attractive. However, this decomposition is restricted to gluonic amplitudes. Since we want to develop a program which ultimately computes Standard Model amplitudes, this is not very useful.

[5] Note that we deal with QCD, an $SU(3)$ gauge theory, in this paper. However, the presented decompositions are valid for all $SU(N)$ theories; to remain as general as possible in our description, we will leave this parameter open as often as possible. Furthermore, the boson of the general $SU(N)$ theories will be referred to as gluons.

[6] However, it can still depend on color information if quarks enter the game. This will be discussed in detail later.

[7] In fact, there are relations which reduce this to $(n-3)!$, see [23].

Apart from this, both color decompositions in the fundamental and in the adjoint representation have another serious drawback for an automated computation: In both cases, the color structures are composed of matrices, thus any computation of color structures requires the multiplication of several matrices, which is expensive in terms of computation time. So, we are looking for a color decomposition without any matrices.

Let us look at yet another complication: If we chose equation (2.3) as our color decomposition and wanted to include a pair of quarks, we would obtain a slightly different decomposition [7]:

$$\mathcal{A}(q,1,2,\ldots,n,\bar{q}) = \sum_{S_n} (T^{a_1}T^{a_2}\cdots T^{a_n})^{i_q}_{j_{\bar{q}}}\, A(q,1,2,\ldots,n,\bar{q}). \tag{2.4}$$

Here, the quark pair position is fixed while the sum runs over all gluons. In contrast to equation (2.3), the color structure is not a scalar variable (i.e. a trace), but it remains matrix-valued because of the quarks. With more quark pairs, there are more such "open strings" of matrices and things get more complicated.

2.2.2 Color-Flow Decomposition

To solve all the problems and complifications, the present section presents another color decomposition, first discovered by Maltoni, Paul, Stelzer and Willenbrock in 2002 [26]: The *color-flow* decomposition.

In this decomposition, all $SU(N)$ generators get rewritten into Kronecker deltas with color indices i and j. This yields the following formula for gluonic amplitudes:

$$\mathcal{A}(1,2,\ldots,n) = \sum_{S_{n-1}} \delta^{i_1}_{j_2}\,\delta^{i_2}_{j_3}\cdots\delta^{i_n}_{j_1}\, A(1,2,\ldots,n). \tag{2.5}$$

While still containing $(n-1)!$ partial amplitudes, the color structure no longer consists of matrices, but Kronecker deltas. Fortunately, Kronecker deltas can be treated simply and efficiently in a computational implementation, as will be shown in later chapters. The partial amplitudes are the same as those from the other decompositions, as proven in [26].

This decomposition introduces the color indices which describe, as the name of the decomposition suggests, the flow of color; in this sense, this decomposition has a very natural and intuitive application. Color flows from the lower indices (j) to the upper indices (i); they transform according to the antifundamental and fundamental representation of $SU(N)$, respectively. Each gluon is represented by two color indices, while each (anti-)quark is represented by only one index (j) i. This makes an incorporation of quarks easy: If we choose to include one quark pair as in equation (2.4) the amplitude takes the form

$$\mathcal{A}(q,1,2,\ldots,n,\bar{q}) = \sum_{S_n} \delta^{i_q}_{j_1}\,\delta^{i_1}_{j_2}\,\delta^{i_2}_{j_3}\cdots\delta^{i_n}_{j_{\bar{q}}}\, A(q,1,2,\ldots,n,\bar{q}), \tag{2.6}$$

hence a quark/antiquark pair behaves exactly like a gluon.

But before going into details with quarks, we first want to make sure that color decompositions — particularly the color-flow decomposition — actually work.

The Three Gluon Vertex

Let us illuminate this with the example of the three gluon vertex, equation (2.2a), with a more suggestive notation where all indices are named according to the particles they correspond to:

$$f^{a_1a_2a_3}\left[g^{\mu_1\mu_2}(p_1-p_2)^{\mu_3}+g^{\mu_2\mu_3}(p_2-p_3)^{\mu_1}+g^{\mu_3\mu_1}(p_3-p_1)^{\mu_2}\right]\equiv -\mathrm{i}f^{a_1a_2a_3}\,V_{(ggg)}^{\mu_1\mu_2\mu_3}, \tag{2.7}$$

the symbol $V_{(ggg)}^{\mu_1\mu_2\mu_3}$ abbreviates the kinematical part including an additional factor i; note that it is antisymmetric in all indices. Also note that the coupling constant has been left out, since it is not relevant for the following considerations. In order to arrive at the fundamental color decomposition, one has to replace the structure constants using the following relation:

$$f^{a_1a_2a_3}=-2\mathrm{i}\,\mathrm{Tr}\left\{T^{a_1}T^{a_2}T^{a_3}-T^{a_1}T^{a_3}T^{a_2}\right\}, \tag{2.8}$$

note that we work in the (arbitrary) normalization

$$\mathrm{Tr}\left\{T^aT^b\right\}=\frac{1}{2}\delta^{ab}.$$

It is already obvious from equation (2.8) that both terms in the trace correspond to the two non-cyclic permutations of S_3, i.e. to S_{3-1}. If one inserts this into the three vertex, one obtains

$$\begin{aligned}-\mathrm{i}f^{a_1a_2a_3}\,V_{(ggg)}^{\mu_1\mu_2\mu_3}&=-2\mathrm{Tr}\left\{T^{a_1}T^{a_2}T^{a_3}-T^{a_1}T^{a_3}T^{a_2}\right\}\,V_{(ggg)}^{\mu_1\mu_2\mu_3}\\&=-2\big(\mathrm{Tr}\left\{T^{a_1}T^{a_2}T^{a_3}\right\}-\mathrm{Tr}\left\{T^{a_1}T^{a_3}T^{a_2}\right\}\big)V_{(ggg)}^{\mu_1\mu_2\mu_3}.\end{aligned}$$

At this stage, one can already see that the fundamental color decomposition is fulfilled after employing antisymmetry in μ_2 and μ_3 in $V_{(ggg)}^{\mu_1\mu_2\mu_3}$: The sign between the two terms becomes positive and the expression can be written as a sum over the two non-cyclic permutations of S_3.

But before rewriting the term in this way, we want to go to color-flow decomposition. This is done in two steps. First, we employ the *double line notation* discovered by 't Hooft in 1973 [27], where we can write each color coupling between vertices V and edges E in the following form:

$$V^aE^a=V^a\delta^{ab}E^b=V^a\left(2(T^a)^i_j\,(T^b)^j_i\right)E^b=\left(\sqrt{2}(T^a)^i_j\,V^a\right)\left(\sqrt{2}(T^b)^j_i\,E^b\right). \tag{2.9}$$

Since we are dealing with a three gluon vertex, we have to apply the first term in brackets. Note that there are three such color couplings from the vertex to the particle edges, one for each $T^{a_k}, k=1,2,3$. The three vertex then becomes

$$\begin{aligned}&-2(\sqrt{2})^3(T^{a_1})^{i_1}_{j_1}\,(T^{a_2})^{i_2}_{j_2}\,(T^{a_3})^{i_3}_{j_3}\left[\mathrm{Tr}\left\{T^{a_1}T^{a_2}T^{a_3}\right\}-\mathrm{Tr}\left\{T^{a_1}T^{a_3}T^{a_2}\right\}\right]V_{(ggg)}^{\mu_1\mu_2\mu_3}\\&=-2(\sqrt{2})^3(T^{a_1})^{i_1}_{j_1}\,(T^{a_2})^{i_2}_{j_2}\,(T^{a_3})^{i_3}_{j_3}\left[(T^{a_1})^k_\ell(T^{a_2})^\ell_m(T^{a_3})^m_k-(T^{a_1})^k_\ell(T^{a_3})^\ell_m(T^{a_2})^m_k\right]V_{(ggg)}^{\mu_1\mu_2\mu_3}.\end{aligned}$$

Note that rewriting the traces into indices anticipates the next step: The application of

the Fierz identity

$$(T^a)^i_j\,(T^a)^k_\ell = \frac{1}{2}\left(\delta^i_\ell\,\delta^k_j - \frac{1}{N_c}\delta^i_j\,\delta^k_\ell\right). \tag{2.10}$$

This identity can be used three times, once for each new generator from the double line notation. We then arrive at

$$-\frac{1}{\sqrt{2}}\left[\delta^{i_1}_{j_2}\,\delta^{i_2}_{j_3}\,\delta^{i_3}_{j_1} - \delta^{i_1}_{j_3}\,\delta^{i_3}_{j_2}\,\delta^{i_2}_{j_1}\right] V^{\mu_1\mu_2\mu_3}_{(ggg)}.$$

The Fierz identity contains also terms of order $\mathcal{O}(1/N_c)$, however, these terms are symmetric in the indices 2 and 3. Since the two traces only differ by the transposition of particles 2 and 3 and a sign, the $\mathcal{O}(1/N_c)$ terms resulting from the two traces cancel each other.

Now, the antisymmetry of $V^{\mu_1\mu_2\mu_3}_{(ggg)}$ can be applied and we arrive at the final structure (with the coupling constant re-attached)

$$-\frac{g_3}{\sqrt{2}}\sum_{S_{3-1}}\delta^{i_1}_{j_2}\,\delta^{i_2}_{j_3}\,\delta^{i_3}_{j_1}\,V^{\mu_1\mu_2\mu_3}_{(ggg)}. \tag{2.11}$$

The Four Gluon Vertex

It is equally as simple to prove that the four gluon vertex can be rewritten in a similar way. However, in this case one has to look at the contraction of two structure constants which gives rise to more lengthy expressions. The first step is to rewrite each structure constant using equation (2.8),

$$\begin{aligned}
f^{abe}f^{cde} &= -4\,\mathrm{Tr}\left\{T^aT^bT^e - T^aT^eT^b\right\}\times\mathrm{Tr}\left\{T^cT^dT^e - T^cT^eT^d\right\}\\
&= -4\left(\mathrm{Tr}\left\{T^aT^bT^e\right\}\mathrm{Tr}\left\{T^cT^dT^e\right\} - \mathrm{Tr}\left\{T^aT^bT^e\right\}\mathrm{Tr}\left\{T^cT^eT^d\right\}\right.\\
&\qquad\left.-\mathrm{Tr}\left\{T^aT^eT^b\right\}\mathrm{Tr}\left\{T^cT^dT^e\right\} + \mathrm{Tr}\left\{T^aT^eT^b\right\}\mathrm{Tr}\left\{T^cT^eT^d\right\}\right).
\end{aligned}$$

In each term, two generators with index e appear. These can be replaced using the Fierz identity (2.10), which leads to the following expression:

$$\begin{aligned}
f^{abe}f^{cde} = -2\Bigg(&\mathrm{Tr}\left\{T^aT^bT^cT^d\right\} - \frac{1}{N_c}\mathrm{Tr}\left\{T^aT^b\right\}\mathrm{Tr}\left\{T^cT^d\right\}\\
&-\mathrm{Tr}\left\{T^aT^bT^dT^c\right\} + \frac{1}{N_c}\mathrm{Tr}\left\{T^aT^b\right\}\mathrm{Tr}\left\{T^cT^d\right\}\\
&-\mathrm{Tr}\left\{T^aT^cT^dT^b\right\} + \frac{1}{N_c}\mathrm{Tr}\left\{T^aT^b\right\}\mathrm{Tr}\left\{T^cT^d\right\}\\
&+\mathrm{Tr}\left\{T^aT^dT^cT^b\right\} - \frac{1}{N_c}\mathrm{Tr}\left\{T^aT^b\right\}\mathrm{Tr}\left\{T^cT^d\right\}\Bigg)\\
= -2\Big(&\mathrm{Tr}\left\{T^aT^bT^cT^d\right\} - \mathrm{Tr}\left\{T^aT^bT^dT^c\right\} - \mathrm{Tr}\left\{T^aT^cT^dT^b\right\} + \mathrm{Tr}\left\{T^aT^dT^cT^b\right\}\Big)
\end{aligned}$$

Obviously, all terms proportional to $1/N_c$ cancel each other. Now, one has to go through the tedious, but not difficult task of inserting this sum of traces into each term of the vertex, see equation (2.2b) and employing the double line notation as in equation (2.9). After calculating through this enormous sum of terms, one is left with six independent

terms which make up the non-cyclic permutations $P \in S_{n-1} = S_{4-1} = S_3$. All terms of order $\mathcal{O}(1/N_c)$ cancel each other in a similar fashion as with the three vertex. Using the symmetries of the metric tensors, one can rewrite the whole expression analogously to the three vertex to arrive at

$$\begin{aligned} &\frac{\mathrm{i}g_3^2}{2} \sum_{S_{4-1}} \delta^{i_1}_{j_2}\, \delta^{i_2}_{j_3}\, \delta^{i_3}_{j_4}\, \delta^{i_4}_{j_1} \left(2g^{\mu_1\mu_3} g^{\mu_2\mu_4} - g^{\mu_1\mu_2} g^{\mu_3\mu_4} - g^{\mu_1\mu_4} g^{\mu_2\mu_3}\right) \\ =&\frac{g_3^2}{2} \sum_{S_{4-1}} \delta^{i_1}_{j_2}\, \delta^{i_2}_{j_3}\, \delta^{i_3}_{j_4}\, \delta^{i_4}_{j_1}\, V^{\mu_1\mu_2\mu_3\mu_4}_{(gggg)}. \end{aligned} \tag{2.12}$$

Note that the additional factors $(-\mathrm{i}g_3^2)$ from the Feynman rule in (2.2b) have been re-applied and the factor i has again been included in the definition of $V_{(gggg)}$. Hence, the four vertex can also be written in color-flow decomposition.

The Four Gluon Amplitude

Until now, only the Feynman rules have been translated to color-flow decomposition. To apply the procedure to a full amplitude, we will now take a look at the process $gg \to gg$ for which the diagrams were already shown in figure 2.1. The first step is to write down the amplitudes for diagrams (a) to (c). For this, we first have to take a look at the propagator from equation (2.2c). Let us denote the kinematical part by

$$K_{\mu\nu} \equiv \frac{-\mathrm{i}g_{\mu\nu}}{k^2 + \mathrm{i}\epsilon}.$$

What about the color factor δ^{ab}? Here, the double line notation has to be applied again, but this time the right bracket of equation (2.9) since a propagator describes an edge in a diagram. This prescription has to be applied to *both* ends of the propagator, thus for both indices a and b. This works as follows:

$$2\,(T^a)^{j_a}_{i_a}\,\delta^{ab}\,(T^b)^{j_b}_{i_b} = 2(T^a)^{j_a}_{i_a}\,(T^b)^{j_b}_{i_b} = \delta^{j_a}_{i_b}\,\delta^{j_b}_{i_a} - \frac{1}{N_c}\delta^{j_a}_{i_a}\,\delta^{j_b}_{i_b}.$$

Note that in the last step, the Fierz identity (2.10) was used once again. Thus, the full propagator has the structure

$$K_{\mu\nu}\left(\delta^{j_a}_{i_b}\,\delta^{j_b}_{i_a} - \frac{1}{N_c}\delta^{j_a}_{i_a}\,\delta^{j_b}_{i_b}\right).$$

After assembling the components of the diagrams, one arrives at the following expressions:

$$\mathcal{A}_{(a)} = \frac{g_3^2}{2} \sum_{P(1,2)} \left(\delta^{i_1}_{j_2}\,\delta^{i_2}_{j_a}\,\delta^{i_a}_{j_1}\,V^{\mu_1\mu_2\nu}_{(ggg)}\right) K_{\nu\rho} \left(\delta^{j_a}_{i_b}\delta^{j_b}_{i_a} - \frac{1}{N_c}\delta^{j_a}_{i_a}\delta^{j_b}_{i_b}\right) \sum_{P(3,4)} \left(\delta^{i_3}_{j_4}\,\delta^{i_4}_{j_b}\,\delta^{i_b}_{j_3}\,V^{\mu_3\mu_4\rho}_{(ggg)}\right) \epsilon_1\epsilon_2\epsilon_3\epsilon_4,$$

$$\mathcal{A}_{(b)} = \frac{g_3^2}{2} \sum_{P(1,4)} \left(\delta^{i_1}_{j_4}\,\delta^{i_4}_{j_a}\,\delta^{i_a}_{j_1}\,V^{\mu_1\mu_4\nu}_{(ggg)}\right) K_{\nu\rho} \left(\delta^{j_a}_{i_b}\delta^{j_b}_{i_a} - \frac{1}{N_c}\delta^{j_a}_{i_a}\delta^{j_b}_{i_b}\right) \sum_{P(2,3)} \left(\delta^{i_2}_{j_3}\,\delta^{i_3}_{j_b}\,\delta^{i_b}_{j_2}\,V^{\mu_2\mu_3\rho}_{(ggg)}\right) \epsilon_1\epsilon_2\epsilon_3\epsilon_4,$$

$$\mathcal{A}_{(c)} = \frac{g_3^2}{2} \sum_{P(1,3)} \left(\delta^{i_1}_{j_3}\,\delta^{i_3}_{j_a}\,\delta^{i_a}_{j_1}\,V^{\mu_1\mu_3\nu}_{(ggg)}\right) K_{\nu\rho} \left(\delta^{j_a}_{i_b}\delta^{j_b}_{i_a} - \frac{1}{N_c}\delta^{j_a}_{i_a}\delta^{j_b}_{i_b}\right) \sum_{P(2,4)} \left(\delta^{i_2}_{j_4}\,\delta^{i_4}_{j_b}\,\delta^{i_b}_{j_2}\,V^{\mu_2\mu_4\rho}_{(ggg)}\right) \epsilon_1\epsilon_2\epsilon_3\epsilon_4.$$

Since the polarization vectors were, without loss of generality, chosen to be real, one does not have to distinguish between initial and final channel. This means they are invariant under permutations and can be included in the permutations of the three gluon vertices. Note that the non-cyclic permutations have been realized such that the external particles get permuted while internal indices are not permuted, which is indicated by the notation $P(1,2)$, etc.

For each diagram, the Kronecker deltas of the propagator have to be contracted with the Kronecker deltas of the three gluon vertices. Since the three diagrams are equal up to particle interchanges, we will look at diagram (a) as an example:

$$\mathcal{A}_{(a)} = \frac{g_3^2}{2} \sum_{\substack{P(1,2)\\P(3,4)}} \left(\delta_{j_2}^{i_1} \delta_{j_3}^{i_2} \delta_{j_4}^{i_3} \delta_{j_1}^{i_4} - \frac{1}{N_c} \left[\delta_{j_2}^{i_1} \delta_{j_1}^{i_2} \right] \left[\delta_{j_4}^{i_3} \delta_{j_3}^{i_4} \right] \right) V_{(ggg)}^{\mu_1\mu_2\nu} K_{\nu\rho} V_{(ggg)}^{\mu_3\mu_4\rho} \epsilon_1\epsilon_2\epsilon_3\epsilon_4$$

Here, the $1/N_c$ term vanishes again: Each three gluon vertex is antisymmetric in all indices while each of the delta terms in square brackets is symmetric in its respective indices. If one exchanges either of the index pairs $(1,2)$ or $(3,4)$, it is obvious that this term has to vanish.

The next step is to write out all permutations for all diagrams (this equals four terms per diagram) and add them up. One ends up with 12 terms, which can be reduced to six different permutations. These permutations occur equally in the color part and in the kinematical part, hence one can write them as one term $\mathcal{A}_{(abc)}$ with a sum over all non-cyclic permutations:

$$\mathcal{A}_{(abc)} = \frac{g_3^2}{2} \sum_{S_{4-1}} \delta_{j_2}^{i_1} \delta_{j_3}^{i_2} \delta_{j_4}^{i_3} \delta_{j_1}^{i_4} \left(V_{(ggg)}^{\mu_1\mu_2\nu} K_{\nu\rho} V_{(ggg)}^{\mu_3\mu_4\rho} + V_{(ggg)}^{\mu_1\mu_4\nu} K_{\nu\rho} V_{(ggg)}^{\mu_2\mu_3\rho} \right) \epsilon_1\epsilon_2\epsilon_3\epsilon_4.$$

This expression already has the form of the color-flow decomposition, equation (2.5). However, it still lacks the fourth diagram which consists of the four gluon vertex. This contribution is given by the expression for the four vertex, equation (2.12), together with the four polarization vectors. These are again invariant under the permutations and can therefore be included in them. Adding this contribution is now trivial, since the only change occurs in the kinematical part, where the four gluon vertex $V_{(gggg)}$ has to be added. This also requires a rearrangement of prefactors. The end result is

$$\mathcal{A}_{(abcd)} = \left(\frac{g_3}{\sqrt{2}} \right)^{n-2} \sum_{P \in S_{4-1}} \delta_{j_2}^{i_1} \delta_{j_3}^{i_2} \delta_{j_4}^{i_3} \delta_{j_1}^{i_4} A(1,2,3,4),$$

where A is the partial amplitude given by

$$A(1,2,3,4) = \epsilon_1\epsilon_2\epsilon_3\epsilon_4 \left(V_{(ggg)}^{\mu_1\mu_2\nu} K_{\nu\rho} V_{(ggg)}^{\mu_3\mu_4\rho} + V_{(ggg)}^{\mu_1\mu_4\nu} K_{\nu\rho} V_{(ggg)}^{\mu_2\mu_3\rho} + V_{(gggg)}^{\mu_1\mu_2\mu_3\mu_4} \right).$$

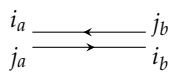
The above example can easily be extended to diagrams with an arbitrary number of gluons. Note that in the last step the prefactor was rewritten as $(g_3/\sqrt{2})^{n-2}$. This can be done for all gluon and QCD amplitudes, so that one can always extract this factor from the partial amplitudes.

2.2.3 The $U(1)$ Gluon and Color Clusters

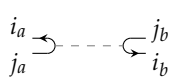
In the previous section, we came across the color structure for the gluon propagator,

$$\delta^{j_a}_{i_b}\delta^{j_b}_{i_a} - \frac{1}{N_c}\delta^{j_a}_{i_a}\delta^{j_b}_{i_b}.$$

While the gluon propagator in the fundamental representation consists of a simple Kronecker delta, this structure is more complicated with its two terms: The first term describes the color structure of a $U(N)$ gluon. To arrive at the desired $SU(N)$ theory, the second term has to be subtracted. It describes a so-called $U(1)$ *gluon*, a gluon which is colorless and can hence also be identified as a photon which couples with the strength of the strong coupling constant. This can be visualized in terms of color flows:



$$= \quad \delta^{j_a}_{i_b}\delta^{j_b}_{i_a} \qquad U(N)$$



$$= \quad -\frac{1}{N_c}\delta^{j_a}_{i_a}\delta^{j_b}_{i_b} \qquad U(1)$$

Note that the kinematical part is identical for both the $U(N)$ and the $U(1)$ part. As the above notation already suggests, we can and will, for the remaining parts of this thesis, deal with these two color structures as if they were separate particles. The $U(N)$ gluon will be simply called *gluon*, whereas the colorless $U(1)$ part will be designated by $U(1)$ *gluon*.

As the computation of the four gluon process made obvious, the $U(1)$ part does not couple to the three gluon vertex due to its antisymmetry in the kinematical part. A similar argument is valid for the four gluon vertex. Hence, gluon amplitudes can be treated as if they were described by a $U(3)$ gauge group, a fact which will be exploited later in this thesis.

When quarks enter the game, however, this has larger effects. The $U(1)$ gluon can couple via the quark-gluon vertex in the same fashion as a (colorless) photon couples to a quark pair: Color flows then from the antiquark leg to the quark leg. This can lead to diagrams where some parts are color-disconnected from the remaining parts, i.e. the diagram separates into parts which are only coupled by a (colorless) $U(1)$ gluon. We will call each such color-disconnected part a *color cluster*. Since external particles only appear by their polarizations and without any color information (this will be done when squaring the amplitude), it is not difficult to assure oneself that the appearance of a $U(1)$ gluon requires that the diagram contains at least two quark/antiquark pairs, i.e. two quark-gluon vertices. Thinking further, one can generally say that the maximal number of color clusters n_{cl} is always given by the number of quark/antiquark pairs n_q, $n_{\text{cl}} = n_q$. The maximum number of $U(1)$ gluons is given by the number of clusters minus one, $n_{\text{cl}} - 1 = n_q - 1$.

This is best illustrated by an example: Figure 2.2 illuminates the above discussion by showing the four different color structures which emanate from a diagram contributing to the process $\bar{q}q \to \bar{q}q\bar{q}q$. As discussed above, the maximum number of color clusters is $n_{\text{cl}} = n_q = 3$ and the maximum number of $U(1)$ gluons is given by $n_{\text{cl}} - 1 = 2$, as can be seen in diagram (d), where each of the quark pairs forms a separate cluster,

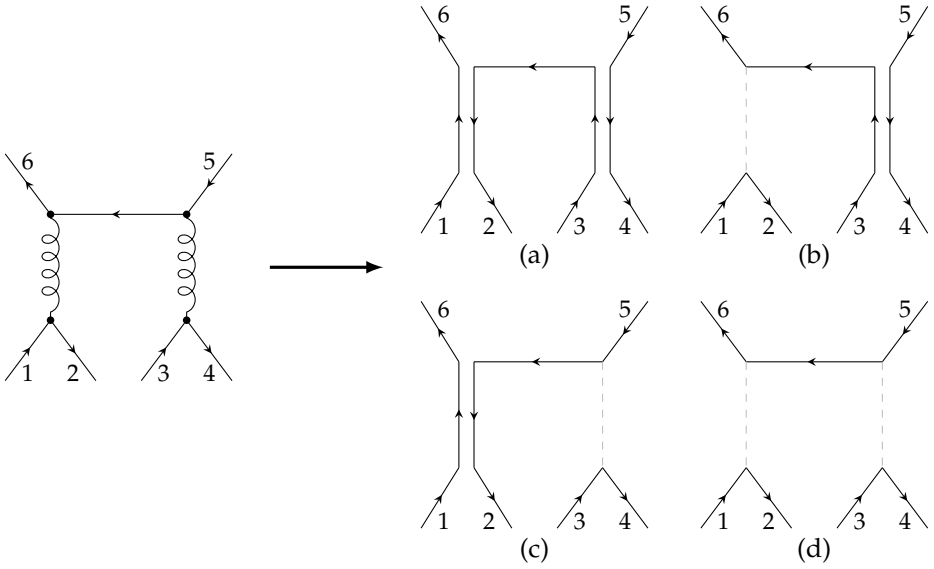


Figure 2.2: Color flows for one Feynman diagram contributing to the scattering of a quark/antiquark pair into two quark/antiquark pairs of arbitrary flavor. Dashed lines denote the $U(1)$. This figure is based on an example given in [14].

color-disconnected from the other parts of the diagram. Diagrams (b) and (c) show the two intermediate cases, where only one quark/antiquark pair is disconnected from the other two pairs. The first diagram is the full colored diagram which consists of only one cluster.

The appearance of color clusters naturally affects the color factor in the color-flow decomposition, equation (2.5) and equation (2.6): The disconnection has to show up in the Kronecker deltas for the individual clusters. Hence, each cluster obtains its own Kronecker delta structure; the resulting structures get multiplied. Furthermore, we have to add a factor $-1/N_c$ for each $U(1)$ gluon. If we denote the full color factor by c and the subfactors for the single clusters by $c^{(k)}$, we obtain the following formula:

$$c = \left(-\frac{1}{N_c}\right)^{n_{\mathrm{cl}}-1} \times \prod_{k=1}^{n_{\mathrm{cl}}} c^{(k)}. \tag{2.13}$$

As always, an example illustrates this best, so here are the color factors for the four diagrams of figure 2.2, which should be self-explanatory:

$$\begin{aligned}
c_{(a)} &= \delta^{i_2}_{j_3}\delta^{i_4}_{j_5}\delta^{i_6}_{j_1} \\
c_{(b)} &= -\frac{1}{N_c}\left(\delta^{i_2}_{j_1}\right)\left(\delta^{i_4}_{j_5}\delta^{i_6}_{j_3}\right) \\
c_{(c)} &= -\frac{1}{N_c}\left(\delta^{i_4}_{j_3}\right)\left(\delta^{i_2}_{j_5}\delta^{i_6}_{j_1}\right) \\
c_{(d)} &= \left(-\frac{1}{N_c}\right)^2\left(\delta^{i_2}_{j_1}\right)\left(\delta^{i_4}_{j_3}\right)\left(\delta^{i_6}_{j_5}\right).
\end{aligned}$$

2.2.4 Cyclic Ordering and Pseudo Particles

The last section dealt with the inclusion of more than one quark/antiquark pair, which was proven to be not too difficult for the color structure. However, we have not yet discussed how several quark/antiquark pairs affect the color decompositions.

Before tackling this problem we will first make another definition: In almost all following parts of this thesis we will treat processes such that all particles are in the final channel. Particles in the ingoing channel can be transferred to the outgoing channel by means of crossing symmetry. Crossing symmetry states that momenta experience an overall sign change and helicities/spins get flipped when a particle is promoted to the other channel. Furthermore, each particle becomes its antiparticle and vice versa. Consider the following example:

$$q(p_1^{\lambda_1})\, q(p_2^{\lambda_2}) \to q(p_3^{\lambda_3})\, q(p_4^{\lambda_4}) \qquad \Longrightarrow \qquad \bar{q}(-p_1^{-\lambda_1})\, \bar{q}(-p_2^{-\lambda_2})\, q(p_3^{\lambda_3})\, q(p_4^{\lambda_4})$$

The crossed process can be reordered such that it contains two quark/antiquark pairs, $(\bar{q}q;\bar{q}q)$. The advantage of this procedure is that whenever quarks (or fermions in general) are part of the process, they can be grouped into fermion/antifermion pairs.

Now it is clear why we spoke of quark/antiquark pairs before. Let us go back to the problem of how several quark/antiquark pairs affect the color decomposition. Equation (2.6) leaves the single quark pair out of the permutations and permutes the gluons fully, i.e. including cyclic permutations. As already mentioned during the discussion of the color factors, a quark/antiquark pair in principle acts as a single gluon, color-wise. We will now extend this further by generally defining a *pseudo particle* as a quark/antiquark pair or a gluon — which means we count quark/antiquark pairs equally as a gluon. Moreover, this enables us to define the so-called *cyclic ordering* which states that the particles should be ordered according to pseudo-particles, i.e. quark/antiquark pairs always appear as pairs and may never be severed from each other by gluons or other quark/antiquark pairs. An example:[8]

$(\bar{q}, q, g, g, \bar{q}, q, g)$ is in cyclic order,
$(\bar{q}, g, q, g, \bar{q}, q, g)$ is *not* in cyclic order.

We can now demand cyclic ordering of the particles and renew the definition of the color-flow representation by having the non-cyclic permutations act on all pseudo particles. There is, however, one further aspect which requires attention: Now, the quark pairs are included in the permutations, but the we must also account for permutations among the quark pairs. Thus, we have to introduce an additional permutation which permutes either all antiquarks or all quarks among each other. Since quarks obey Fermi statistics, we also need to include a permutation sign. Let $\tilde{i}$ denote a pseudo particle, then the full decomposition for an amplitude with n_p pseudo-particles reads

$$\mathcal{A}(\tilde{1},\tilde{2},\ldots,\tilde{n}_p) = \sum_{S_{n_p-1}} \sum_{P_q \in S_{n_q}} \sum_{\substack{\text{cluster} \\ \text{possibilities}}} \operatorname{sign}(P_q)\; c\, A(\tilde{1},\tilde{2},\ldots,\tilde{n}_p).$$

[8] Whether one defines $\bar{q}q$ as a pseudo particle or $q\bar{q}$ is not relevant; but one has to be consistent with the ordering. In the present implementation we use the ordering $\bar{q}q$.

Note that we have to take all possible cluster configurations into account; this affects the color factor c as given in equation (2.13).

In the above formula the partial amplitude is written with pseudo particles as arguments, however, it is not sensitive to them but treats each quark and antiquark as single particles. It remains to justify that this works. With cyclic ordering, quarks always appear "left" of gluons, antiquarks "right" of gluons (or the other way around, depending on whether one defines a pseudo particle as $\bar{q}q$ or $q\bar{q}$), this can be seen in the above example for cyclic ordering. One could think that this would result in a lack of contributions to the total amplitude. In fact, cyclic ordering gives all the contributions neccessary: Take, for example, a quark-gluon vertex with the legs $(q, \bar{q}, g)$. If one swaps the gluon and the antiquark, $(q, g, \bar{q})$, the contribution is the same. Hence, adding all particle configurations which are omitted by cyclic ordering would actually yield double contributions.

Note that the above considerations only apply to tree-level and cannot be adopted one-to-one for NLO calculations. Further information on the NLO case can be found in [15].

2.2.5 Obtaining Squared Amplitudes

So far, only the complex amplitudes have been discussed, but nothing was said about squaring those amplitudes. For this, let us clarify the notation once again: In the previous section, the symbol c was introduced for the color factors in the color decomposition. Thus, an amplitude can be written as

$$\mathcal{A} = \sum_{S_{n-1}} c\, A = \sum_{i \in S_{n-1}} c_i\, A_i,$$

where we have indexed the different permutations with i. As this notation suggests, it is convenient to put all partial amplitudes into a vector-like structure $\vec{A}$ such that each entry A_i corresponds to one partial amplitude. In the same way it is possible to organize the color factors c_i. Squaring then becomes rather intuitive except for one detail concerning the color factors, which shows when multiplying the color vector with its adjoint. To form the adjoint, a new set of indices has to be used which we will indicate by a bar over the color indices (not to be confused with the tilde for the pseudo particles!). Then we are faced with four different kinds of indices: i, j, $\bar{i}$ and $\bar{j}$, which have to be contracted somehow. This is done by a *projector term* P, which contains the color structures of the external particles. For quarks and antiquarks, the projectors are trivial,

$$P_q = \delta_{i\bar{i}}, \quad P_{\bar{q}} = \delta^{j\bar{j}}, \tag{2.14}$$

however, the gluon projector has the $U(N) - U(1)$ form:

$$P_g = \delta_{i\bar{i}}\, \delta^{j\bar{j}} - \frac{1}{N_c} \delta_i^j\, \delta_{\bar{i}}^{\bar{j}}. \tag{2.15}$$

The full projector term is then a combination of the projectors for all external particles:

$$P = \prod_{k=1}^{n_g+2n_q} P_k.$$

With this projector term attached, the squared amplitude takes the form

$$|\mathcal{A}|^2 = \sum_{i,j} \left(A_i^* \bar{c}_i\right) P \left(c_j A_j\right) = \sum_{i,j} A_i^* \left(\bar{c}_i P c_j\right) A_j = \sum_{i,j} A_i^* M_{ij} A_j.$$

Therein, the color factors and the projector have been gathered to form the *color matrix* M.[9] The matrix depends on both particle configurations which make up the respective color-orderings of the partial amplitudes, but it is still completely independent of kinematical indices. This enables one to compute the color matrix independently of the partial amplitudes. This is of special importance when the squared matrix element shall be evaluated several times with different kinematical settings. In such a case, the partial amplitudes have to be computed for each such setting, but the color matrix is identical since it is independent of kinematics. Examples where this will be of use are:

- Unpolarized cross sections. At colliders such as the LHC one measures unpolarized cross sections, meaning the helicity and spin configurations of the particles are not known. Thus, one performs a summation over all possible helicity configurations to obtain an average for the unpolarized cross section.
- For cross sections one has to integrate over phase space which influences the momenta of the particles in the final channel.[10] This integration is usually performed by a Monte Carlo integration, i.e. basically a summation over many squared amplitudes with randomly chosen momenta.

Let us shortly recap what we have discussed in this section: First, systematic treatment of amplitudes using color decompositions was introduced. This was followed by a short review of the different decompositions. We then presented the color-flow decomposition which is used for the present program since it abandons matrices in favor of Kronecker deltas which are easy to compute. This decomposition was then examined in detail, with special focus on the $U(1)$ gluon which is an artifact of this representation. Related to this, the inclusion of quarks was discussed before we finally clarified the computation of squared matrix elements in color decompositions.

2.3 Weyl-van der Waerden Formalism

From a theoretical viewpoint, color has been treated exhaustively concerning the computational realization of the computation of amplitudes. However, we have not spend too many thoughts about partial amplitudes, which also poses some difficulties for numerical evaluation. When mixing bosons and fermions, objects from two mathematically separate spaces appear: Lorentz structures and spin or Dirac structures. These mix via the Dirac gamma matrices whose elements live in spinor space while the four different gamma matrices form a Lorentz vector living in Minkowski space. This is awkward to handle. Thus, the following sections introduce techniques which reduce the representation of the Lorentz group for spin 1 bosons to the two-dimensional representations $(1/2, 0)$ and $(0, 1/2)$ of spin $1/2$ fermions, which provides for a consistent treatment of bosons and fermions on the same mathematical level.

[9] The reason that this is a matrix instead of one scalar accounts for interferences among the partial amplitudes.

[10] In this case, the final channel *before* applying crossing symmetry is meant.

2.3.1 Spinor Helicity Method

This method is based on the idea of using two-component Weyl spinors to define both fermion states and polarization states of vector bosons such as the photon and the gluon. In the case of massless spin $1/2$ particles, the Dirac equation decouples into two separate equations, the *Weyl equations*. The Weyl equations correspond to the two-dimensional representations $(1/2, 0)$ and $(0, 1/2)$ and thus explicitly show that there are two non-equivalent spin $1/2$ representations of the Lorentz group. The corresponding Weyl spinors are given by

$$|p\pm\rangle \equiv \mathcal{P}_{\pm} u(p) = \frac{1}{2}(1 \pm \gamma^5) u(p) \,, \tag{2.16}$$

where $\mathcal{P}_{\pm}$ is the chirality projector and $\pm$ denotes the helicity eigenstate. Note that $u(p)$ is a four component spinor, however, the relation is meant such that we take only the two non-vanishing components of the spinor. In the 1980s, it was first discovered that polarization vectors for massless spin 1 particles can be written as ([28, 29], see also [30, 31])

$$\epsilon_{\mu}^{+}(p,q) = \frac{\langle q-| \gamma_{\mu} |p-\rangle}{\sqrt{2}\langle q-|p+\rangle} \,, \qquad \epsilon_{\mu}^{-}(p,q) = \frac{\langle q+| \gamma_{\mu} |p+\rangle}{\sqrt{2}\langle p+|q-\rangle} \,. \tag{2.17}$$

Here, p is the fourmomentum of the boson and q denotes a light-like so-called *reference momentum* which is arbitrary up to the fact that it may not be parallel to the momentum p. This can be justified as follows:

i) Spinor products such as those appearing in the denominator of the polarizations obey the following rules:

$$\langle p-|k+\rangle \equiv \langle pk\rangle = -\langle kp\rangle \,, \qquad \langle p+|k-\rangle \equiv [kp] = -[pk] \,, \tag{2.18}$$

$$\Longrightarrow \quad \langle pp\rangle = 0 = [pp] \,.$$

Clearly, if q and p are parallel, $q = p$, the denominators in equation (2.17) vanish and the polarization vectors diverge.

ii) Changing the reference momentum is equivalent to an on-shell gauge transformation. With the help of equation (2.18) and the identities

$$|p\pm\rangle\langle p\pm| = \mathcal{P}_{\pm}\not{p} \,, \qquad \langle q-|\gamma_{\mu}|p-\rangle = \langle p+|\gamma_{\mu}|q+\rangle$$

it is not difficult to prove that under the change $q \to \tilde{q}$, the polarization vector remains the same up to a term proportional to p:

$$\epsilon_{\mu}^{+}(\tilde{q}) - \epsilon_{\mu}^{+}(q) = \frac{\sqrt{2}\langle q\tilde{q}\rangle}{\langle \tilde{q}p\rangle\langle qp\rangle} p_{\mu}.$$

An analogous calculation holds for polarization vectors for negative helicity. With the above result, it is clear that the reference momentum can be chosen arbitrarily.

It remains to show that the polarizations in (2.17) fulfill all neccessary criteria:

- The polarization vector has to be transverse to the momentum:

$$p^\mu \epsilon^+_\mu = \frac{\langle q- | \not{p} | p- \rangle}{\sqrt{2}\langle q- | p+ \rangle} = 0 \qquad \text{since } \not{p} | p\pm \rangle = 0$$

- Complex conjugation should flip the polarization's helicity:

$$(\epsilon^+_\mu)^* = \left(\frac{\langle q- | \gamma_\mu | p- \rangle}{\sqrt{2}\langle q- | p+ \rangle} \right)^* = \frac{\langle p- | \gamma_\mu | q- \rangle}{\sqrt{2}\langle p+ | q- \rangle} = \frac{\langle q+ | \gamma_\mu | p+ \rangle}{\sqrt{2}\langle p+ | q- \rangle} = \epsilon^-_\mu$$

- The polarization vectors should be normalized correctly:

$$(\epsilon^+)^\mu (\epsilon^+)^*_\mu = \epsilon^+ \cdot \epsilon^- = \frac{\langle q- | \gamma_\mu | p- \rangle \langle q+ | \gamma^\mu | p+ \rangle}{2\langle qp \rangle [pq]} = -1$$
$$(\epsilon^+)^\mu (\epsilon^-)^*_\mu = \epsilon^+ \cdot \epsilon^+ = \frac{\langle q- | \gamma_\mu | p- \rangle \langle q- | \gamma^\mu | p- \rangle}{2\langle qp \rangle^2} = 0$$

Hence, polarizations of the form (2.17) can be used conveniently to represent external massless vector bosons, while massless fermions are directly given by Weyl spinors. However, we also want to include massive fermions; the masses of most quarks may be negligible for center of mass energies of several TeV, such as they appear at the LHC, however, neglecting the top quark with a mass of roughly 172.0 GeV [32] can result in strongly deviating results. Fortunately, massive fermions can also be treated with this formalism [31, 33–35]. We will use massive spinors as given in [36]. The basic idea is the following: Suppose we have a spinor $|q-\rangle$ for a massless particle with a lightlike momentum q which obeys the Weyl equations and hence also the Dirac equation. One can then prove[11] that the spinor

$$u(p, q, +) = \frac{1}{\sqrt{2p \cdot q}} (\not{p} + m) |q-\rangle$$

is also an eigenspinor of the Dirac equation which describes a massive particle with momentum p, lightlike reference momentum q and spin $+1/2$, indicated by the third argument "+". In the same manner one can derive spinors for spin $-1/2$ and the corresponding antiparticle spinors v, the following shows a full list:

$$u(p, q, +) = \frac{1}{\sqrt{2p \cdot q}} (\not{p} + m) |q-\rangle , \qquad u(p, q, -) = \frac{1}{\sqrt{2p \cdot q}} (\not{p} + m) |q+\rangle ,$$
$$v(p, q, +) = \frac{1}{\sqrt{2p \cdot q}} (\not{p} - m) |q-\rangle , \qquad v(p, q, -) = \frac{1}{\sqrt{2p \cdot q}} (\not{p} - m) |q+\rangle .$$

Conjugating the spinors to obtain $\bar{u}$ and $\bar{v}$ is trivial and will not be done here. Note that these states are no longer helicity eigenstates since they mix both massless helicity eigenstates via the gamma matrix that appears in $\not{p}$.

Let us recap what he have achieved so far: We managed to express spin 1 polarization vectors in terms of Weyl spinors so, basically, we can describe both spin 1/2

[11] A similar proof can be found in [31].

fermions on the same mathematical footing as spin 1 particles. However, these polarizations still contain Lorentz indices in terms of the gamma matrix, which we already deemed as being rather complicated to deal with.

To get rid of this problem, let us first introduce a new notation for spinors, which is known from supersymmetry:

$$|p+\rangle = p_B \,, \qquad |p-\rangle = p^{\dot{B}} \,, \qquad \langle p+| = p_{\dot{A}} \,, \qquad \langle p-| = p^A \,. \tag{2.19}$$

Here, dotted and undotted indices explicitly display the two non-equivalent representations $(1/2, 0)$ and $(0, 1/2)$ of the Lorentz group $L_+^\uparrow$, exhibiting the fact that the Dirac equation describes both fermions and antifermions. The reason for this notation will become clear in the next section.

2.3.2 Double Line Notation Revisited

In fourdimensional Minkowski notation, one can define two sets of Pauli matrices by $\sigma^\mu = (\mathbb{1}, \vec{\sigma})$ and $\bar{\sigma}^\mu = (\mathbb{1}, -\vec{\sigma})$. Dirac matrices can then be written as

$$\gamma^\mu = \begin{pmatrix} 0 & \sigma^\mu \\ \bar{\sigma}^\mu & 0 \end{pmatrix} \,.$$

This is the so-called Weyl representation which makes the different representations of the Lorentz group even more explicit, since the Weyl equations for Weyl spinors then read

$$\mathrm{i}\bar{\sigma}^{\dot{B}A} \cdot \partial \psi_A = 0 \,, \qquad \mathrm{i}\sigma_{A\dot{B}} \cdot \partial \psi^{\dot{B}} = 0$$

where we have employed the notation from equation (2.19); ψ_A and $\psi^{\dot{B}}$ are the position space spinors corresponding to p_A and $p^{\dot{B}}$.

One can now perform a neat trick by using the identity

$$g^{\mu\nu} = \frac{1}{2}(\sigma^\mu)_{A\dot{B}}(\bar{\sigma}^\nu)^{\dot{B}A} = \frac{1}{\sqrt{2}}(\sigma^\mu)_{A\dot{B}}\frac{1}{\sqrt{2}}(\bar{\sigma}^\nu)^{\dot{B}A} \,.$$

Vector-like couplings always couple vertices V and edges E (i.e. propagators and polarizations) in terms of Minkowski products which, using the above formula, can be re-written into a trace over 2×2 matrices:

$$\begin{aligned} V_\mu E^\mu = V_\mu g^{\mu\nu} E_\nu &= \left(\frac{1}{\sqrt{2}} V_\mu (\sigma^\mu)_{A\dot{B}}\right)\left(\frac{1}{\sqrt{2}} E_\nu (\bar{\sigma}^\nu)^{\dot{B}A}\right) \\ &= \left(\frac{1}{\sqrt{2}} V_\mu \sigma^\mu\right)_{A\dot{B}}\left(\frac{1}{\sqrt{2}} E_\nu \bar{\sigma}^\nu\right)^{\dot{B}A} \,. \end{aligned} \tag{2.20}$$

Hence, by simple multiplication of σ^μ or $\bar{\sigma}^\mu$, respectively, it is possible to rewrite Lorentz vectors into matrices and thus get rid of Lorentz indices so that calculations are performed only with spinorial objects. Note the similarities to 't Hooft's double line notation for color indices, equation (2.9).

Such matrix objects are dubbed with the name *bispinor*; depending on the indices they belong to one representation of the Lorentz group, thus they can be further distinguished in terms of positive and negative bispinors. These designations can be

memorized easily by keeping in mind that such bispinors can also be constructed with two Weyl spinors:

$$\begin{aligned} \text{positive bispinor:} \qquad & p_A q_{\dot{B}} = |p+\rangle \langle q+| \\ \text{negative bispinor:} \qquad & p^{\dot{B}} q^A = |p-\rangle \langle q-| \ . \end{aligned}$$

It is not difficult to see that in this notation, the polarization vectors introduced in equation (2.17) can be written in terms of negative bispinors

$$\epsilon_+^{\dot{A}B}(p,q) = \frac{1}{\langle qp \rangle} p^{\dot{A}} q^B \ , \qquad \epsilon_-^{\dot{A}B}(p,q) = \frac{1}{[pq]} q^{\dot{A}} p^B \ . \tag{2.21}$$

In this equation, all Lorentz indices are gone, instead we only have to take spinor indices into account.

The formalism we are now working in, after employing the spinor helicity formalism and going into double line notation, is sometimes called *Weyl-van der Waerden formalism*, according to the founders Weyl and van der Waerden [37, 38]. It has been used frequently, see for example [21], since it is convenient especially for automated computations, for the reasons we already stated. A detailed overview over the full formalism can be found in [35].

2.3.3 The Feynman Rules

The only thing left to do is to rewrite the Feynman rules into double line notation. We will only look at the example of the three gluon vertex, since it encompasses everything one needs to know to derive the remaining Feynman rules easily.

We have already derived the color structure in color-flow representation, equation (2.11), so we will only look at the kinematical part defined in equation (2.7) as

$$V_{(ggg)}^{\mu_1\mu_2\mu_3} = \mathrm{i}\left[g^{\mu_1\mu_2}(p_1-p_2)^{\mu_3} + g^{\mu_2\mu_3}(p_2-p_3)^{\mu_1} + g^{\mu_3\mu_1}(p_3-p_1)^{\mu_2}\right].$$

As with the double line notation for color, we have to apply the vertex part of equation (2.20) to each external leg:

$$\frac{\mathrm{i}}{(\sqrt{2})^3}(\sigma_{\mu_1})_{A\dot{B}}(\sigma_{\mu_2})_{C\dot{D}}(\sigma_{\mu_3})_{E\dot{F}}\left[g^{\mu_1\mu_2}(p_1-p_2)^{\mu_3} + g^{\mu_2\mu_3}(p_2-p_3)^{\mu_1} + g^{\mu_3\mu_1}(p_3-p_1)^{\mu_2}\right].$$

The Lorentz indices can be contraced to form

$$\begin{aligned} \frac{\mathrm{i}}{(\sqrt{2})^3}\Big[& (\sigma_{\mu_1})_{A\dot{B}}\,(\sigma^{\mu_1})_{C\dot{D}}\,(p_1-p_2)_{E\dot{F}} + \\ & (\sigma_{\mu_2})_{C\dot{D}}\,(\sigma^{\mu_2})_{E\dot{F}}\,(p_2-p_3)_{A\dot{B}} + \\ & (\sigma_{\mu_3})_{E\dot{F}}\,(\sigma^{\mu_3})_{A\dot{B}}\,(p_3-p_1)_{C\dot{D}}\Big], \end{aligned}$$

which can be rewritten using the formula

$$(\sigma_\mu)_{A\dot{B}}\,(\sigma^\mu)_{C\dot{D}} = 2\epsilon_{AC}\,\epsilon_{\dot{B}\dot{D}},$$

where the ϵ matrix,

$$\epsilon = \mathrm{i}\sigma^2, \quad \epsilon^{AB} = \epsilon^{\dot{A}\dot{B}} = \epsilon_{AB} = \epsilon_{\dot{A}\dot{B}} = \begin{pmatrix} 0 & +1 \\ -1 & 0 \end{pmatrix},$$

takes the place of the metric $g^{\mu\nu}$ in spinor space. The vertex in its final Weyl-van der Waerden form is then given by

$$V^{(ggg)}_{A\dot{B}C\dot{D}E\dot{F}} = \frac{\mathrm{i}}{\sqrt{2}}\Big[\epsilon_{AC}\,\epsilon_{\dot{B}\dot{D}}\,(p_1 - p_2)_{E\dot{F}} + \epsilon_{CE}\,\epsilon_{\dot{D}\dot{F}}\,(p_2 - p_3)_{A\dot{B}} + \epsilon_{EA}\,\epsilon_{\dot{F}\dot{B}}\,(p_3 - p_1)_{C\dot{D}}\Big].$$

This from of the vertex is used in the program presented in this thesis. The remaining Feynman rules are listed in the appendix, page 101, they are just as simple to translate from the common versions.

2.4 Berends-Giele Recursion Relations

Up to this point, both the formalism for the color structure and the kinematical realization have been discussed, but the calculation of the four gluon process, see section 2.2.2, revealed an expression for the partial amplitude which cannot be easily generalized to arbitrary processes. Therefore, this section presents an elegant recursive approach to obtain partial amplitudes which was found by Berends and Giele in 1987 [39].

Their approach is to subdivide an n gluon amplitude into smaller components which they call *currents*. These currents, which we will denote by G to elucidate that we deal with gluonic currents, describe building blocks of smaller size with $m < n$ external legs and one off-shell leg, which themselves depend recursively on sub-currents as follows:

$$\begin{aligned} G^{\dot{A}B}_m(1,\ldots,m) = &\sum_{j=1}^{m-1} G^{\dot{C}D}_j(1,\ldots,j) G^{\dot{E}F}_{m-j}(j+1,\ldots,m)\, V^{(ggg)}_{D\dot{C}F\dot{E}H\dot{G}}(1,j+1)\, P^{\dot{G}H\dot{A}B}(p_{1,m}) \\ &+ \sum_{j=1}^{m-2}\sum_{k=j+1}^{m-1} G^{\dot{C}D}_j(1,\ldots,j) G^{\dot{E}F}_{k-j}(j+1,\ldots,k) G^{\dot{G}H}_{m-k}(k+1,\ldots,m)\times \\ &\qquad \times V^{(gggg)}_{D\dot{C}F\dot{E}H\dot{G}J\dot{I}}\, P^{\dot{I}J\dot{A}B}(p_{1,m}) \end{aligned} \tag{2.22}$$

Therein, $V^{(ggg)}$ and $V^{(gggg)}$ denote the three gluon and four gluon vertices, respectively, and $P(p_{i,j})$ denotes a gluon propagator where $p_{i,j} = \sum_{k=i}^{j} p_k$ is the associated momentum. This recursive relation is also shown pictorially in figure 2.3.

The recursion start is given by the currents consisting of only one particle — these are just the gluon polarization vectors from equation (2.21):

$$G^{\dot{A}B}_1(i) = \epsilon^{\dot{A}B}_{\pm}(p_i, q_i)\ .$$

The partial amplitude can be obtained by computing G_{n-1}, taking the off-shell leg on-shell (i.e. removing the propagator of the off-shell leg) and attaching the polarization vector of the n-th particle.

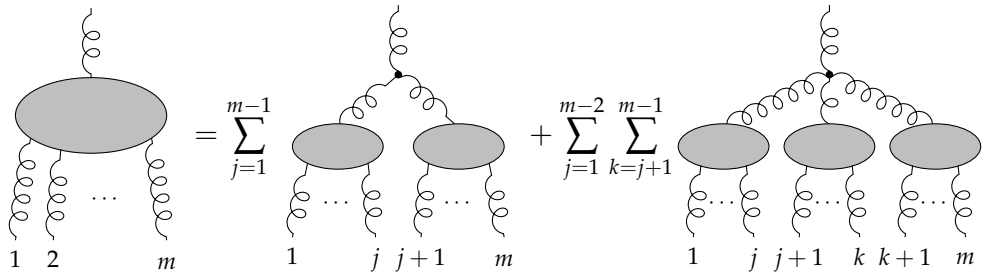


Figure 2.3: Graphical representation of the Berends-Giele recursion formula (2.22). All upper gluons are off-shell gluons.

In the course of this thesis, these relations will be extended to include quarks and photons. As will be shown, this extension is not difficult if one respects certain conditions for the subcurrents.

2.5 Basics of Monte Carlo Integration

In all various fields of computational physics, integrals have to be evaluated. High energy physics is no exception. Take equation (2.1) as an example: To obtain a cross section from a squared matrix element, one has to integrate over phase space — an integration which cannot be performed analytically, especially since the matrix element is not known in analytical form but only numerically. The topic of phase space integration will not be adressed in this thesis, however, it will neccessarily be part of the final program and we will use the forthcoming phase space integration to perform another integration, namely over helicity configurations. Hence, this section shall give a brief introduction on Monte Carlo integration techniques based on [40].

Since the above description of the problem is rather vague, let us first specify the starting point more precisely. Monte Carlo techniques generally tackle the problem of integrating a function $f(x_1, x_2, \ldots, x_n)$ which depends on n variables $z = (x_1, x_2, \ldots, x_n)$ over some interval. We will take this interval to be given by the unit hypercube $[0,1]^n$.[12] To put this into a formula, the integration we wish to solve is

$$I = \int \mathrm{d}z\, f(z) = \int \mathrm{d}^n x\, f(x_1, x_2, \ldots, x_n)$$

The estimate E for the value of I which one obtains by Monte Carlo integration is given by

$$E = \frac{1}{N} \sum_{i=1}^{N} f(z_i),$$

where each z_i denotes a set of n randomly chosen numbers within the unit hypercube. The law of large numbers guarantees that for large N, the estimate converges to the value of the integrand:

$$\lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} f(z_i) = I.$$

[12] For most integrations, it is possible to map the integration interval to the unit hypercube by means of substitution.

If one introduces the variance $\sigma^2(f)$ by[13]

$$\sigma^2(f) = \int \mathrm{d}z \left(f(z) - I\right)^2, \tag{2.23}$$

it is not difficult to show that

$$\int \mathrm{d}z_1 \int \mathrm{d}z_2 \cdots \int \mathrm{d}z_N \left(\frac{1}{N}\sum_{i=1}^{N} f(z_i) - I\right)^2 = \frac{\sigma^2(f)}{N}.$$

This equation tells us that the error of the Monte Carlo estimate E is on average given by $\sigma(f)/\sqrt{N}$, where $\sigma(f)$ is the standard deviation of f. It follows that the error of Monte Carlo integration only depends on the number of iterations N, but not on the dimension n of the integration. This is the great advantage that Monte Carlo techniques provide over classical numerical integration procedures (numerical quadrature rules): These approaches might work far better for one dimensional integrations compared to Monte Carlo, but for larger dimensions this advantage vanishes quickly since the error usually scales as $1/N^{\alpha/n}$, where α depends on the particular method used.

Since Monte Carlo integration is only neccessary if the integration result I is unknown, one cannot obtain the variance $\sigma^2(f)$ as given in equation (2.23). Hence, one uses an estimate for the error of a Monte Carlo integration given by the square root of the Monte Carlo variance

$$S^2 = \frac{1}{N}\sum_{i=1}^{N}\big(f(z_i)\big)^2 - E^2.$$

Note, however, that this is only an error *estimate* which only gives a probability for the error bounds.

Another remark on the dependence of the error on the number of iterations: The error scales as $1/\sqrt{N}$, which provides a rather slow convergence. Since the computation time should be minimized as much as possible, it is reasonable to find a way of improving this behaviour. Indeed, there exist two techniques:

Stratified sampling: This techniques divides the integration interval into smaller subsets. If one chooses these subspaces and the number of iterations in each subspace carefully, one can obtain better results than with an ordinary Monte Carlo integration. On the contrary, this might also lead to worse results if the choices are not good enough.

Importance sampling: Roughly speaking, this method introduces a probability density $p(z)$ for the integration interval. The variance depends on the choice of the probability density: The closer this density approximates $p(z) = f(z)/I$ the smaller the variance. However, since I is not known the variance can only be as good as one is able to guess $p(z)$.

Both methods can be combined to form an adaptive Monte Carlo algorithm called *VEGAS* [41] which acquires better knowledge of the integrand with each iteration.

[13] This definition requires that f is square integrable. Note that integrands f which are *not* square-integrable also yield estimates E which converge to I, but then the expectation value of the error has no significance.

We will not go into further details of these algorithms, since the basic Monte Carlo algorithm suffices for the tasks to be accomplished in this thesis, however they will be of relevance when the integration over phase space enters the game during the further development of the program.

Chapter 3

General Considerations

As already mentioned in the introductory chapter, the present implementation for the automated computation of tree-level diagrams forms a part of a larger library for the computation of scattering amplitudes. The basic foundation of this package already exists and also constitutes the basis for this present implementation. This library is named *particle scattering*.

Particle scattering was developed by Stefan Weinzierl and is written in C++, which is thus also the language the present implementation is written in. From the beginning, the whole package was designed to make use of the Weyl-van der Waerden formalism. Hence, those parts that constitute *particle scattering* version 0.0.1 — the version which the present program is built upon — provide basic tools for computations in this formalism. Among those tools are, most notably, classes for fourvectors (i.e. four-momenta), two-component Weyl spinors for bra and ket states, each with separate versions for both helicities, and also bispinors which are needed for bosonic building blocks such as polarizations and gluon currents according to the Berends-Giele relation (2.22). In addition, all color-ordered Feynman rules for QCD are already implemented in terms of functions for polarizations, vertices and propagators. The rules for fermions include both the massless and the massive case. Apart from these physical basics, *particle scattering* provides means to define different types of particles and distinguish them, as well as a class called `basic_event` which gathers all information on a particle process, including momenta, masses, helicities, etc. This information is structured such that each particle in the process obtains an integer index which is used to access all information on the particle. This is in accordance to the notation we already employed in the previous chapter. Furthermore, several classes of a more mathematical origin are present which provide means to deal with vectors, matrices, permutations of indices, etc. Usage of such classes will be explained in more detail when they are used in this thesis. It should be noted that, in contrast to the many tools for kinematical computations, *particle scattering* does not provide any means to compute the color matrix or parts thereof. However, the present program extends the package by this functionality and is fully capable of computing full color information, the details on this will be discussed in later chapters.

The following sections provide a general overview over the thoughts behind the present implementation. The first section outlines the basic structure of the present implementation. It will become clear that models for different interactions or gauge

groups are implemented in terms of subclasses which all inherit from one base class that provides several general methods and concepts. The non-trivial and physically interesting of these concepts are then presented in the remaining sections of this chapter.

3.1 General Program Structure

From the discussions in chapter 2, it is clear that the computation of a squared tree-level amplitude has two major ingredients: The *color matrix* and the *partial amplitudes*. Each of these ingredients, however, requires additional information which forms a third ingredient: all valid cyclic orderings which we will call *particle configurations*. Hence, there are three different types of information the program should provide, each depending on the interaction one wants to describe. This suggests the following program structure which is realized in the present implementation.

- An abstract base class called `particle_configuration` provides an interface for the management of all possible configurations. For Standard Model computations, information on the particle configuration is given by the particle permutation and, in the case of QCD, information on the color cluster which each particle belongs to. Both are internally realized by STL `vectors`[1] which store particle or cluster indices. The class `particle_configuration` provides methods to obtain both vectors, as well as a method to advance to the next particle configuration. There are currently three realizations for particle configurations in terms of subclasses, an inheritance diagram is shown in figure 3.1.

  particle_configuration_colorless This realization can be used for all theories with both fermions and bosons which do not depend on color information, i.e. for theories where the color cluster information is irrelevant. Examples where this comes to use are QED (see chapter 5) or the *broken* version of the $SU(2) \times U(1)$ theory of electroweak interactions, which is not yet implemented. It can also be used for QCD with leading color approximation, however, this is also not implemented.

  particle_configuration_colored This realization is meant for theories with color, in particular QCD with full color information. In this case, information on the color clusters is essential.

  particle_configuration_multi The final subclass is made for models which contain multiple gauge groups. In the present implementation this is used for computing QCD amplitudes with external photons, but it is written such that it relies on the above mentioned particle configuration classes. So, basically, this class can deal with a variety of mixed gauge groups. In particular, it is designed in view of a future implementation of the Standard Model.

  The particular realizations are discussed in detail later on.

- In contrast to the particle configurations, the color matrices are not realized using classes, but simple functions which compute the matrix and store them in a

[1] STL `vectors` are containers which resemble C-style arrays but are dynamic in size.

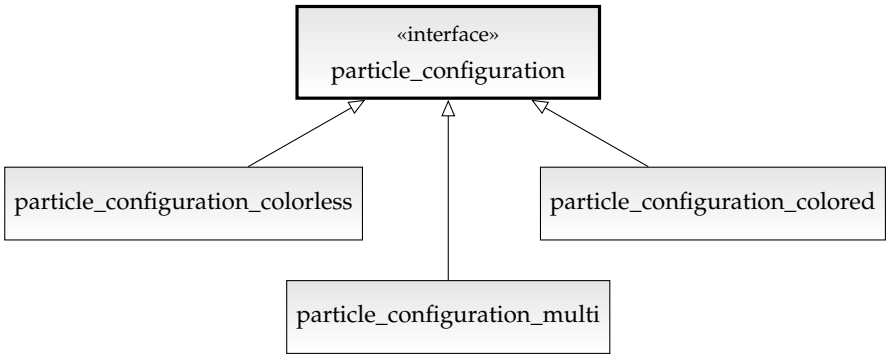


Figure 3.1: Inheritance diagram for all classes concerned with particle configurations.

matrix object given via reference.[2] There are three such functions at present:

compute_leading_color_matrix This function computes the color matrix in leading order, i.e. all terms of order $\mathcal{O}\left(1/N_c\right)$ are omitted. Since gluons do not couple to $U(1)$ gluons, the leading color matrix yields all the color information that is needed to compute purely gluonic amplitudes.

compute_full_color_matrix The matrix computed by this function contains the full color information, up to all orders in $1/N_c$.

compute_mixed_color_matrix For theories merging particles from a colored and a colorless gauge group, this function provides the correct color matrix. This function relies on the the full color matrix; the particulars are discussed in chapter 7.

- Finally, the partial amplitudes are realized using an abstract base class `general_amplitude` and several derived classes, one for each available interaction model. Each of these derived classes has to provide a function `partial_amplitudes` which computes and returns a vector containing all partial amplitudes for the current set of kinematical information. An overview of the inheritance relations between those classes can be found in figure 3.2. However, these classes do not solely compute the partial amplitudes. In fact, the interface provided by the base class `general_amplitude` is the main user interface and provides much more functionality. This will be illuminated in the next section.

3.2 Using the Program: A Short Overview

To explain the meaning of the class `general_amplitude` and illustrate some of the internal mechanisms, it is convenient to look at a short example on how to use the program. The following code excerpt shows how to set up an object for the computation of squared amplitudes. Note that all parameters are given as dummy variables, their meaning will be explained below:

```
1 amplitude_ptr amp;
```

[2] It would yield no advantage to put the color matrix into a class structure: The color matrix is a single entity which has to be computed exactly once per process.

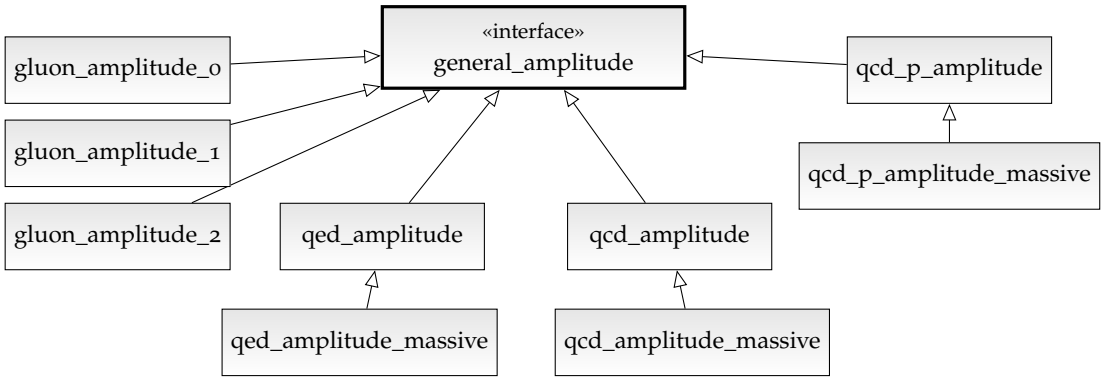


Figure 3.2: Inheritance diagram for all amplitude classes. The three gluon classes contain three different levels of optimization which are explained in chapter 4; `qcd_p` stands for QCD with external photons; all amplitudes except the gluon amplitudes exist as massive and as massless amplitudes.

```
2  make_amplitude(
3    amp,
4    process_string,
5    p,
6    M_QCD,
7    coupling_constants_default,
8    computation_information_default,
9    monte_carlo
10 );
```

The basic object a user has to deal with is an `amplitude_ptr`, a pointer[3] to an object of any derived class of `general_amplitude`. In the following, any (object of a) subclass of `general_amplitude` will be called *amplitude class* (*object*) for simplicity. Such a pointer to an amplitude class is created in the first line of code. The program provides a comfortable function `make_amplitude` to create an object for this pointer; the user does not need to know anything about the internal details of any amplitude class. This function takes several parameters which shall be discussed briefly in terms of line numbers:

3) The previously constructed `amplitude_ptr` which later points to the constructed object.

4) A `std::string` which contains information on the particles in the process. Note that all particles have to be given as if they were in the outgoing channel; all particles in the ingoing channel have to be converted by means of crossing symmetry. Recall that this turns particles into antiparticles and vice versa. As an example, one has to replace `process_string` by `"g g g g"` for a process with four gluons, or by `"b~ b g u~ u a"` for a process with a $\bar{b}b$ pair, a gluon, a $\bar{u}u$ pair and a photon.

[3]Technically, it is a `shared_ptr` from the *BOOST* libraries [6]. It provides the advantage that the user handles a pointer, but does not have to worry about allocation or deallocation of memory — this is done by the program itself.

5) The momenta for all particles have to be given in terms of a STL `vector` with fourvectors as elements, above denoted by `p`. Note that crossing symmetry applies here as well: Momenta of ingoing particles obtain an overall minus sign when being crossed into the outgoing channel (i.e. crossed momenta obtain negative energies). Hence, the overall sign of a momentum decides whether a particle is in the initial or final state. Furthermore, the norm of each momentum determines the mass of the particle via the on shell relation $p^2 = m^2$. The ordering of momenta in the vector corresponds to the ordering of particles in the `process_string`.

6) The fifth line describes the interaction model used for the computation. Possible values are `M_gluon`, `QED`, `M_QED_massless`, `QCD`, `M_QCD_massless`, `QCD_with_photons` and `M_QCD_with_photons_massless`. Another possible value is `M_self_determine` which tries to find a suitable model by looking at the process string given in line 2. Depending on the model which one chooses, the program automatically creates an object of the derived class of `general_amplitude` which describes this model. Note that this happens internally, so the user does not have to worry about the different subclasses.

7) This parameter is an object of a `struct` which contains values for coupling constants to be used. The predefined object `coupling_constants_default` can be used, which contains standard values for center of mass energies which are about the mass of electroweak bosons (approx. 90 GeV).

8) This parameter is an object of a `struct` which contains information on the computation such as possible levels of optimization or whether the squared amplitude should include symmetry factors or averaging over initial spins and colors.

9) The final parameter is a boolean value which decides whether a Monte Carlo integration should be carried out. Where and how Monte Carlo integration enters the game will be elaborated upon in section 3.6.2; as of now, this is of no concern.

Note that the CD which is appended to this thesis contains further documentation and a more elaborate example program on the usage of the program.

During the construction of any amplitude object, a lot of things happen in the background to ensure that all internal data is set up properly so that squared amplitudes can be computed. This includes the computation and storage of the color matrix and the computation of a prefactor which shall be multiplied to the squared amplitude; this prefactor is influenced by the parameter given by code line no. 8 and can also contain coupling constants, depending on the specific implementation of the chosen model.

The following three sections will illuminate three more important steps that take place during the creation of an amplitude object, before the final sections of this chapter will shed some light on how the squaring of amplitudes is realized.

3.3 Creating Cyclic Ordering

The user of the program is not required to enter the particles and their momenta in a certain ordering. However, section 2.2.4 made it clear that any color decomposition

requires the particles (and with them all data such as momenta and helicities) to be in cyclic ordering.

During the creation of an amplitude object, a resorting of all particles occurs so that a proper cyclic ordering is guranteed. The ordering obeys the following specific pattern:

$$g, \cdots, g; \bar{q}, q, \cdots, \bar{q}, q; \bar{\ell}, \ell, \cdots \bar{\ell}, \ell; \gamma, \cdots, \gamma.$$

Therein, g denotes a gluon, q a quark, ℓ a lepton and γ a photon. The bars indicate antiparticles; note that for charged leptons, the leptons with negative charge are taken to be particles, $\ell = \{e^-, \mu^-, \tau^-\}$ and the leptons with positive charge are taken to be antiparticles, $\bar{\ell} = \{e^+, \mu^+, \tau^+\}$. Fermions are always sorted in terms of particle/antiparticle pairs with equal flavor. Note that the sorting is also sensitive to fermion flavors, these are sorted according to the patterns u, d, s, c, t, b in the case of quarks and e, μ, τ in the case of leptons. Cyclic ordering in general does not require separating the different particle types from each other as is done above; however, sorting them in this fashion becomes neccessary when coupling two gauge groups, as will be discussed later on. Furthermore, this sequence creates a well-defined state for all possible combinations of particles and enables easier debugging.

3.4 Choosing Proper Reference Momenta

Section 2.3 made it obvious that the computation of polarizations for vector bosons and the computation of massive fermion spinors in the spinor helicity formalism requires reference momenta q which are arbitrary up to the restriction that they may not be collinear to the real momentum. These reference momenta can be chosen by the following prescription

$$\begin{aligned} q_0 &= p_1 \\ q_1 &= p_2 \\ &\vdots \\ q_{n-2} &= p_{n-1} \\ q_{n-1} &= p_0, \end{aligned} \tag{3.1}$$

which works fine unless two consecutive momenta are (nearly) collinear. Then the reference momentum of the former particle will be (nearly) collinear to the real momentum and the spinor product in the numerator of equation (2.21) tends to zero, which causes numerical problems.

While such a case can sometimes be avoided by carefully sorting the particles before assigning reference momenta, there is another consideration which strongly motivates a more sophisticated method for choosing the reference momenta: *Particle scattering* is a program for numerical computation from the very beginning, which means the basic building blocks of the calculation are matrices of complex numbers describing the polarization vectors (recursion start). Hence, if one can guarantee numerically stable polarization vectors, the end result is more likely to be free of numerical problems. Therefore, we will derive a methode which avoids possible poles and always computes well-defined reference momenta.

Equation (2.21) reveals the composition of the spinors: The momenta p and q form a bispinor matrix which gets divided by the spinor product $\langle qp \rangle$. If the spinor product is very small (and/or the numerator very large) or the other way around, the entries of the spinor matrix will have very large (or small) entries. Since the only non-fixed variables in this calculation are the reference momenta, they can be adjusted so that the entries in the spinor matrix will be numerically stable. Thus, the goal is to choose the reference momentum such that the spinor product $\langle qp \rangle$ is roughly of the same magnitude as the matrix entries resulting from $p^{\dot{A}} q^{B}$. In the end there will be two freely adjustable parameters $\mathcal{R}$ and $\mathcal{N}$ to control the generation of reference momenta.

Before going into the details, it is convenient to introduce a few additional and often occuring symbols to describe momenta in the double line notation. Consider any fourmomentum $p^{\mu} = (p_t, p_x, p_y, p_z)$ and define the following quantities:

$$\begin{aligned} p_{\pm} &= p_t \pm p_y \\ p_{\perp} &= p_z + \mathrm{i} p_x \\ p_n &= |p_t + p_y|^{-\frac{1}{2}}. \end{aligned}$$

The spinor product can be expressed as

$$\langle qp \rangle = \frac{1}{\sqrt{|p_+ q_+|}} \left(q_z p_+ - p_z q_+ + \mathrm{i} \left[p_x q_+ - q_x p_+ \right] \right). \tag{3.2}$$

One can choose the reference momenta such that the spinor product is real, which results in the constraint

$$p_x q_+ = q_x p_+. \tag{3.3}$$

The equation then reduces to

$$\langle qp \rangle = \frac{1}{\sqrt{|p_+ q_+|}} \left(q_z p_+ - p_z q_+ \right) = \mathcal{R}. \tag{3.4}$$

$\mathcal{R}$, the real value of the spinor product, is one of the two free parameters which should be roughly of magnitude of the spinor matrix entries $p^{\dot{A}} q^{B}$, as stated above. If one now starts with equation (3.4) and uses constraint (3.3) one can determine the spacial components of the reference momenta. Since this calculation yields no interesting information, we will not perform it here but only display the results:

$$\begin{aligned} q_x &= \mathcal{N} \cdot p_x \\ q_y &= \mathcal{N} \cdot p_y - \frac{\mathcal{R}}{p_+} \left(\frac{\mathcal{R}}{2} \operatorname{sign}(\mathcal{N}) + p_z \sqrt{|\mathcal{N}|} \operatorname{sign}(p_+) \right) \\ q_z &= \mathcal{N} \cdot p_z + \mathcal{R} \sqrt{|\mathcal{N}|} \operatorname{sign}(p_+) \end{aligned} \tag{3.5a}$$

Therein, $\mathcal{N} = \frac{q_x}{p_x}$ is the second free parameter for choosing the reference momenta. The time component q_t should not be determined in terms of the free parameters, but using the lightlike property of the reference momenta:

$$q_t = \operatorname{sign}(p_t) \sqrt{q_x^2 + q_y^2 + q_z^2}. \tag{3.5b}$$

The reason for this is that spinors for *massive* fermions require lightlike reference momenta. In this case, computing q_t in terms of the free parameters would yield a reference momentum which is not lightlike, resulting in wrong spinors.

Note that this method is also well suited for testing gauge invariance: A different choice for $(\mathcal{R}, \mathcal{N})$ yields a linearly independent reference momentum vector. So, by observing the value of the squared amplitude with varying parameters, one can test whether the computation is gauge invariant.

In the present implementation, both parameters are chosen randomly from the interval $[1, 1000]$ for each reference momentum. In some sense this automatically provides a check for gauge invariance: If one computes the same amplitude twice, the parameters are different and thus also the reference momenta. However, the squared amplitude should be gauge invariant and thus yield the same result.

For testing purposes, these parameters can also be chosen manually. For gluon amplitudes, we have checked that varying either of the parameters from 1 to 100 000 yields squared amplitudes which are identical up to at least nine leading digits. This means that a broad range of values yields numerically stable, gauge invariant amplitudes.

3.5 Momentum Sums

During the calculation of Berends-Giele currents, all possible sums of momenta of the external gluons are needed. This becomes obvious when looking at the propagator terms in the Berends-Giele relation (2.22) along with the sum over all particle permutations in the color-flow representation (2.5). Hence, it appears reasonable to calculate all sums at the very beginning of the computation, save them and draw back on these results during the actual computation of the partial amplitudes in order to save computation time.

This is easy to achieve: For n external momenta, there are 2^n different momentum sums (including single momenta), thus one can index each of these sums with a natural number $j \in [0, 2^n)$. The assignment from a set of particle momenta to this index is simple if one writes the index j in binary format. Let each binary digit stand for one momentum, then one can interpret the binary representation as a list of all momenta, where 0 means that the momentum is not part of the sum and 1 indicates that the momentum takes part.

As an example consider $n = 4$ external particles with momenta $p_0, \ldots, p_3$ and look at the momentum sum $p_{13} = p_1 + p_3$. The corresponding index is $j = 10$, this can be seen as follows:

$$j \mathrel{\widehat{=}} \begin{pmatrix} p_0 & p_1 & p_2 & p_3 \\ 0 & 1 & 0 & 1 \end{pmatrix} \mathrel{\widehat{=}} 0 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 = 10. \tag{3.6}$$

The matrix-like notation displays the binary representation of the index along with the momentum which each digit represents.

Such a sum can easily be computed and stored in an array or an STL `vector` which provides fast access to indexed elements. This has been implemented in all the different amplitude computation classes.

Note that we will not provide any tables on performance here, this will be done collectively for all methods concerned with optimization in chapter 8.

3.6 Methods for Squaring Amplitudes

The previous sections described the preliminaries for the computation of partial amplitudes. Once the described procedures have been performed, the color matrix and the partial amplitudes can be computed. The computation of these depends on the specific interaction model and will be presented in separate chapters (chapters 4 to 7). However, the consolidation of partial amplitudes and color matrix to form the squared amplitude is independent from the particular models and happens centrally in the class `general_amplitude`.

The `general_amplitude` class in the present program provides a function for the computatio of a single helicity amplitude, `helicity_amplitude`, which expects one parameter, a STL `vector` with helicities for all particles. Upon calling this function, the given particle helicities are set and the corresponding partial amplitudes are computed. To store the color matrix and the partial amplitudes, we use optimized data structures from the *BOOST* library uBLAS; hence the computation of the helicity amplitude,

$$|\mathcal{A}|^2 = \vec{A}^\dagger M \vec{A}, \tag{3.7}$$

is also done using optimized routines from *BOOST* (we will come back to this later). Proof that this yields fast results will be presented in section 8.1.3.

Yet, this is the squared amplitude for merely *one* specific helicity configuration. In the case of the LHC and similar colliders, particle collisions happen with *unpolarized* beams, meaning that the helicities of the particles are statistically distributed and not fixed. The present program provides different methods for obtaining such amplitudes, which are described in the following sections.

3.6.1 Helicity Summation

The simplest method is sometimes also called the *helicity method*. It is based on the observation, that over the course of a longer measurement, all possible helicity configurations appear with equal probability. Provided that this assumption holds, one can obtain the unpolarized squared amplitude by summing over the matrix elements for all possible helicity configurations and afterwards possibly average spins of the initial channel particles (in the case of two ingoing particles with spin $1/2$ or two massless spin 1 particles this corresponds to dividing the summed matrix element by a factor of 4). Denote the (external) particle helicities for a given process by $\lambda_0, \lambda_1, \ldots, \lambda_{n-1}$, then the helicity method works as follows:

$$\overline{|\mathcal{A}|}^2 = \sum_{\lambda_0} \sum_{\lambda_1} \cdots \sum_{\lambda_{n-1}} \left|\mathcal{A}_{\lambda_0,\lambda_1,\ldots,\lambda_{n-1}}\right|^2, \tag{3.8}$$

note that averaging over spins is *not* included in the above formula.

This method is exact, but it is very expensive in terms of computation time: If only particles with two possible spin settings are taken into account[4], then there are a priori 2^n possible helicity configurations which have to be considered in order to obtain the value of the unpolarized matrix element. This computation overhead can be partly

[4]This includes all fermions as well as photons and gluons. Massive vector bosons such as W^+, W^-, Z or scalar particles as the Higgs boson are neglected in this consideration.

reduced if one makes use of a fact mentioned in [42] and [20]: Each partial amplitude A_i for a given set of helicities is related to the partial amplitude with inverse helicities by parity:

$$\vec{A}_{\lambda_0,\lambda_1,\ldots,\lambda_{n-1}} = -\vec{A}^*_{-\lambda_0,-\lambda_1,\ldots,-\lambda_{n-1}}.$$

With this trick in mind, only half of the helicity configurations need to be computed.

Obtaining all possible helicity configurations works again like the indexing of the momentum sums as described in section 3.5: With the above trick, one increments an integer number from 0 to $2^n/2$ and decomposes each number into its binary representation. Again, each bit corresponds to one particle; the identification between value of the bit (0 or 1) and particle helicity should be clear.

3.6.2 A Monte Carlo Approach

The clear disadvantage of the helicity method is the excess in computation time it produces. This section presents a method to avoid the need of summing up helicity amplitudes at all.

As shown in section 2.1, equation (2.1), the computation of a cross section involves an integration over phase space, which is an integration over $4n$ dimensions for a process with n particles in the final state with four momentum components each. This integration will be performed in terms of a Monte Carlo algorithm.[5] The error of Monte Carlo integrations scales like $1/\sqrt{N}$ where N is the number of Monte Carlo iterations, a value which is unaffected by the dimensionality of the integration (as discussed in section 2.5; for further information see [40]). Thus, the integration converges equally as good if another few dimensions are added to it. This fact can be exploited in order to turn the helicity summation into an integration over helicity angles.

The method presented in the following was first developed by Draggiotis, Kleiss and Papadopoulos[6]. Their idea is to use a new polarization vector for the gluons which is given by a linear combination of the polarization vectors for positive and negative helicity from equation (2.17), where the coefficients are phases with an angle ϕ:

$$\epsilon_\mu(p,q,\phi) \equiv \mathrm{e}^{\mathrm{i}\phi}\,\epsilon^+_\mu(p,q) + \mathrm{e}^{-\mathrm{i}\phi}\,\epsilon^-_\mu(p,q). \tag{3.9}$$

An integration over ϕ then yields the sum over the usual polarizations, note that any explicit momentum dependency is left out for better readability:

$$\begin{aligned}
\frac{1}{2\pi}\int_0^{2\pi} \mathrm{d}\phi\,\left[\epsilon_\mu(\phi)\epsilon^*_\nu(\phi)\right] &= \frac{1}{2\pi}\int_0^{2\pi} \mathrm{d}\phi\,\left[\mathrm{e}^{\mathrm{i}\phi}\,\epsilon^+_\mu + \mathrm{e}^{-\mathrm{i}\phi}\,\epsilon^-_\mu\right]\left[\mathrm{e}^{-\mathrm{i}\phi}\,(\epsilon^+_\nu)^* + \mathrm{e}^{\mathrm{i}\phi}\,(\epsilon^-_\nu)^*\right] \\
&= \frac{1}{2\pi}\int_0^{2\pi} \mathrm{d}\phi\,\left[\epsilon^+_\mu(\epsilon^+_\nu)^* + \epsilon^-_\mu(\epsilon^-_\nu)^*\right] + \frac{1}{2\pi}\int_0^{2\pi} \mathrm{d}\phi\,\left[\mathrm{e}^{2\mathrm{i}\phi}\epsilon^+_\mu(\epsilon^-_\nu)^* + \mathrm{e}^{-2\mathrm{i}\phi}\epsilon^-_\mu(\epsilon^+_\nu)^*\right] \\
&= \sum_{\lambda=\pm} \epsilon^\lambda_\mu(\epsilon^\lambda_\nu)^*.
\end{aligned} \tag{3.10}$$

[5] Integration over phase space is not subject of this thesis, hence the future tense of this sentence.
[6] See [43]; further information in [44, 45].

The second line follows from the first line by expanding the product; in two of the resulting terms, the exponentials cancel each other, in the other two terms they add up. In the first term of the second line, the integrand is independent of ϕ, thus the integration is trivial and cancels the prefactor. This term can then be rewritten as the desired sum over helicities in the third line. The second term in the second line, however, vanishes upon integration.

The above described method to choose new polarizations is also valid for fermions. In the massless case, it is easy to see that choosing new spinors

$$\begin{aligned} u(p,\phi) &\equiv \mathrm{e}^{\mathrm{i}\phi}\, u_+(p) + \mathrm{e}^{-\mathrm{i}\phi}\, u_-(p) \quad \text{and} \\ \bar{u}(p,\phi) &\equiv \mathrm{e}^{\mathrm{i}\phi}\, \bar{u}_+(p) + \mathrm{e}^{-\mathrm{i}\phi}\, \bar{u}_-(p) \end{aligned} \tag{3.11}$$

(analogously for the spinors v and $\bar{v}$) yields a similar relation as equation (3.10):

$$\frac{1}{2\pi} \int_0^{2\pi} \mathrm{d}\phi \, [u(\phi)\, \bar{u}(\phi)] = \sum_\lambda u_\lambda(p)\, \bar{u}_\lambda(p), \tag{3.12}$$

again also for v and $\bar{v}$. For massive spinors, the helicity states are no longer eigenstates, but they mix to form a spin eigenstate. Hence, the spin quantum number serves to distinguish the two particle states. If one replaces the helicity state "$+$" by spin $1/2$ and "$-$" by spin $-1/2$, the above formulas (3.11) and (3.12) still hold.

The above sums over all helicities or spin states is exactly what appears during the summation of all helicity amplitudes, although it appears not only for one polarization vector or spinor, but once for each external particle in the amplitude. Thus it is possible to rewrite each such sum into an integration over corresponding helicity angles:

$$\sum_{\lambda_0}\sum_{\lambda_1}\cdots\sum_{\lambda_{n-1}} \left|\mathcal{A}_{\lambda_0,\lambda_1,\ldots,\lambda_{n-1}}\right|^2 = \frac{1}{(2\pi)^n} \int_0^{2\pi} \mathrm{d}\phi_0 \int_0^{2\pi} \mathrm{d}\phi_1 \cdots \int_0^{2\pi} \mathrm{d}\phi_{n-1} \left|\mathcal{A}_{\phi_0,\phi_1,\ldots,\phi_{n-1}}\right|^2. \tag{3.13}$$

Of course, the above consideration is also valid in the double line notation. It is then possible to replace the summation over all 2^n helicity configurations by the computation of one single amplitude per phase space point, where the linear combinations from equations (3.9) and (3.11) are used instead of the discrete versions. The helicity angles ϕ are chosen to be random numbers $\phi \in [0, 2\pi)$ and the integration is performed in terms of a Monte Carlo integration. The integration itself involves computing helicity angle amplitudes many times in order to arrive at a result with reasonable precision. As mentioned before, this integration has to be performed for the phase space regime anyway and creates no extra cost, in contrast to the manual helicity summation.

To prove that this method really works and behaves as expected, let us look at three different processes for the different models. Figure 3.3 shows a convergence plot for the process $gg \to gg$, including only bosonic Monte Carlo polarizations. Figure 3.4 shows the process $u\bar{u} \to u\bar{u}d\bar{d}g$, which additionally includes spinors for quarks. Finally, the process $b\bar{b} \to t\bar{t}g\gamma\gamma$ is shown in figure 3.5, including photons in addition and massive spinors for the b and t quarks. As should be obvious from the diagrams, the Monte Carlo technique shows the desired and expected convergence behaviour of $1/\sqrt{N}$ for multidimensional integration.

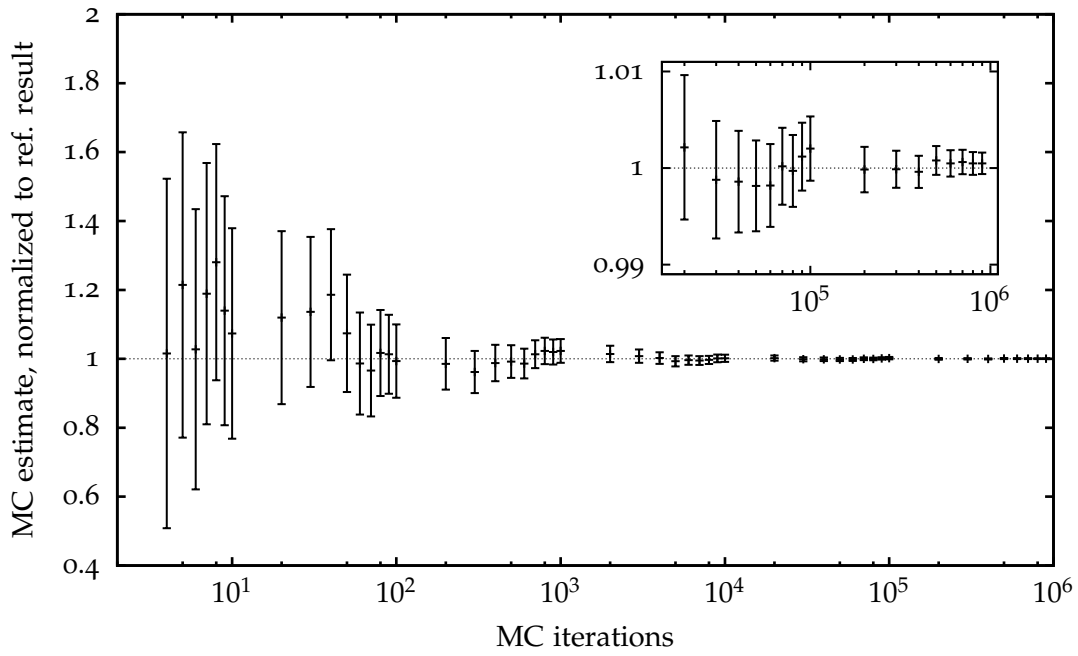


Figure 3.3: Convergence plot for Monte Carlo helicity integration of the process $gg \to gg$.

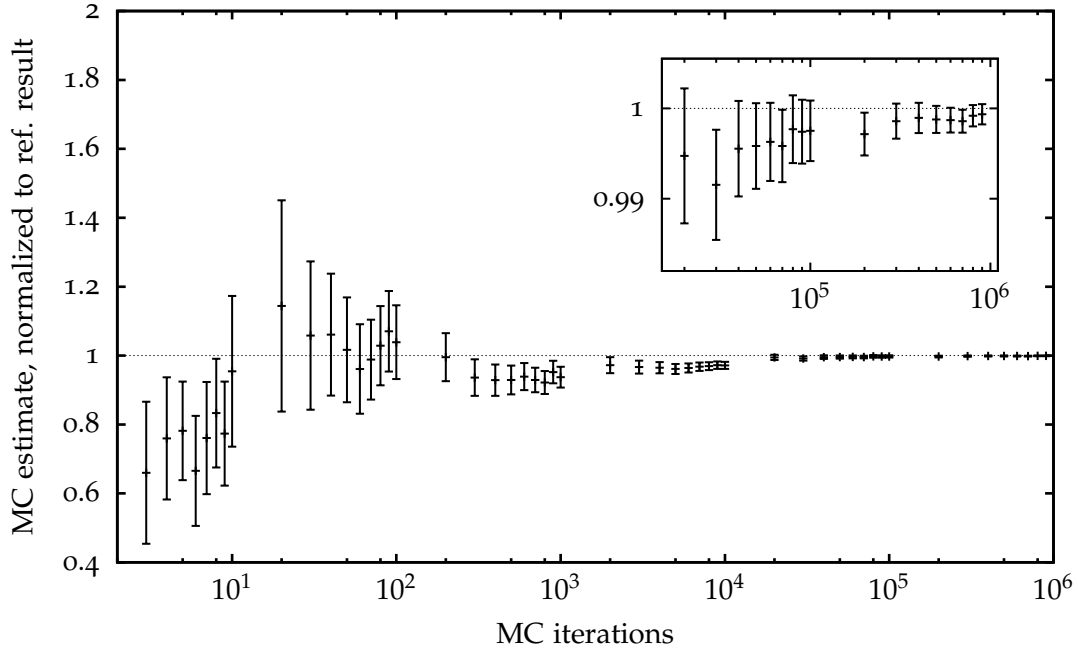


Figure 3.4: Convergence plot for Monte Carlo helicity integration of the process $u\bar{u} \to u\bar{u}d\bar{d}g$.

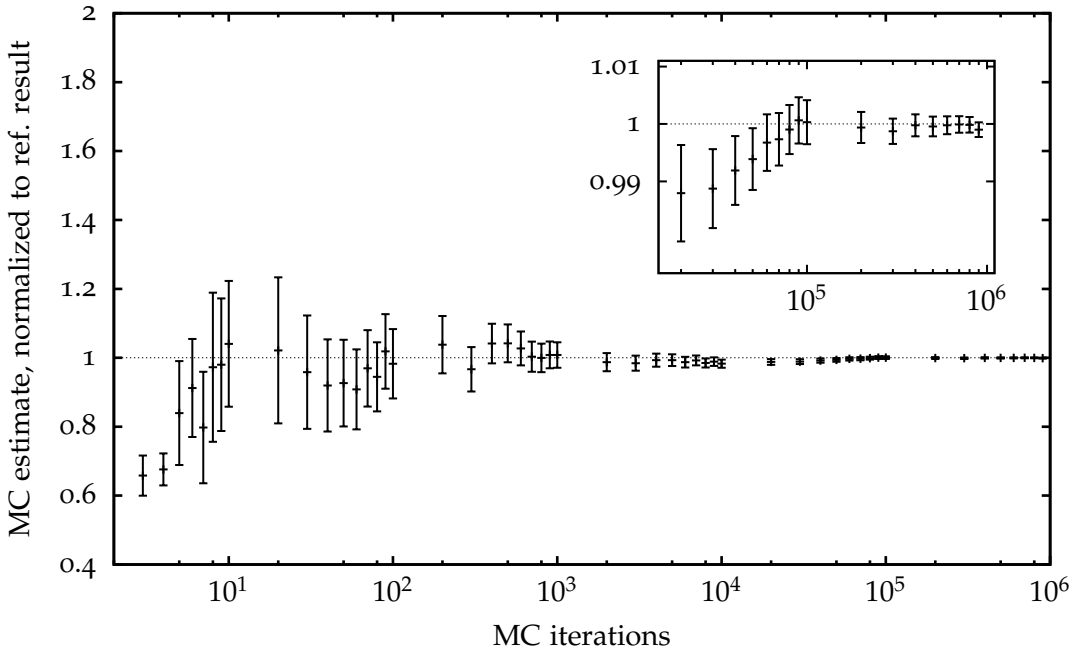


Figure 3.5: Convergence plot for Monte Carlo helicity integration of the process $b\bar{b} \to t\bar{t}g\gamma\gamma$.

3.6.3 Multi-Threading for Monte Carlo Integration

The Monte Carlo integration technique illustrated in the previous section has been implemented in terms of an additional class `monte_carlo` which provides a simple and crude Monte Carlo integration as explained in section 2.5. However, it also provides an interface for the possibility of derived classes with more sophisticated Monte Carlo algorithms. Such a `monte_carlo` object requires an `amplitude_ptr` object; analogously to the function `helicity_amplitude` described above, an amplitude object provides a function `monte_carlo_amplitude` for randomly chosen helicity angles.

A second approach, independent of the above class, was implemented in another class `monte_carlo_omp`. It provides the same crude Monte Carlo algorithm but it supports parallel computation on several CPU cores with the help of the OpenMP specification, delivered by GOMP, the GCC OpenMP support library. In C++, OpenMP provides multi-threading ability via preprocessor directives; as an example, the following code fragment between the braces is automatically executed in several threads:

```
#pragma omp parallel
{
  // Code to be executed via multi-threading
}
```

The number of threads is equal to the number of CPU cores by default; the present program provides the ability to change this number via a constructor parameter of the `monte_carlo_omp` class.

This multi-threaded integration is implemented such that the constructor of `monte_carlo_omp` accepts the same parameters as an `amplitude_ptr` up to the last parameter which is replace by said parameter for the number of threads to execute. The reason

is that `monte_carlo_omp` creates its own amplitude objects internally, one such object in each thread. When performing the actual integration, each thread computes only $1/\text{no. of threads}$ of the desired number of Monte Carlo steps. The work flow of the integration routine is shown in figure 3.6.

This method has the disadvantage that creating an amplitude object in each thread also causes each thread to compute the color matrix on its own. Since each thread should take about the same time for the computation of the color matrix, there is theoretically no loss in computation time in comparison to an approach where the color matrix is computed in advance and then used by each thread. However, the cost shows in terms of memory consumption, which is higher by a factor equal to the number of used threads. An eight gluon amplitude, for example, has $x = 7! = 5040$ partial amplitudes, hence the color matrix has $(x^2 + x)/2 \approx 12.7$ billion independent entries.[7] Each element is a double precision floating point value which usually consumes 8 bytes. Thus, one color matrix amounts to roughly 100 MB, which has to be multiplied by the number of threads. This is of course only an approximation which does not seem to be much on a modern computer. However, it depends on the machine and should be considered when running a multi-threaded computation.

So, what do we expect in terms of performance acceleration? Ideally, we would expect a reduction by $1/n_{\text{thr}}$ where n_{thr} is the number of threads; that is, if the number of threads is not greater than the number of processor cores. In practice, we will probably experience a lower speed-up since multi-threading always involves synchronization among the threads, which costs additional CPU time.

The issue with the color matrix raises the question whether there is a better solution than the method discussed above. An optimal solution would be to employ parallel computation for the color matrix elements as well, so that the color matrix can be computed once in advance and very fast. This color matrix could then be accessed by all Monte Carlo threads. However, this is currently not implemented and merely an outlook.

[7] This will be explained in section 4.2.

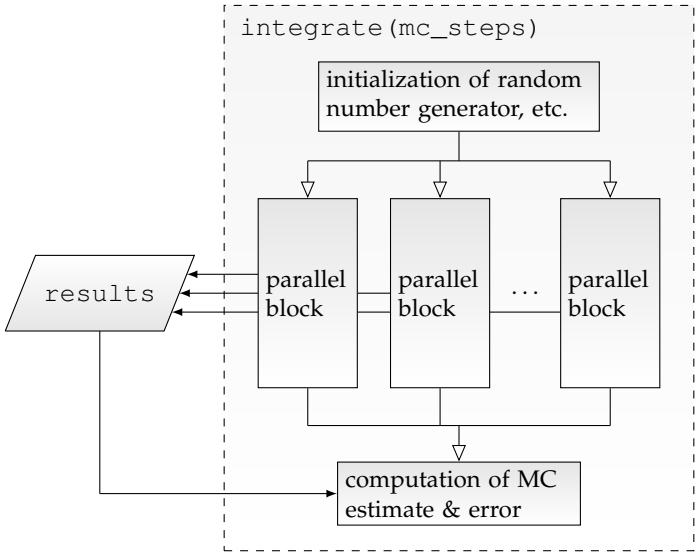


Figure 3.6: Flow chart for the integration routine of the `monte_carlo_omp` class. The large dashed box shows the member function `integrate` which takes an arbitrary number of Monte Carlo iterations as parameter `mc_steps`. Before the split into several threads occurs, a common random number generator is initialized. Each parallel block initializes its own amplitude object and computes $^{\text{mc\_steps}}/_{\text{no. of threads}}$ Monte Carlo iterations. Every single result is stored in a common container called `results`, a member of the `monte_carlo_omp` class. After all threads have finished computation, the single main thread takes all values from `results` to compute the Monte Carlo estimate and error.

Chapter 4

Gluon Amplitudes

After the general design principles of the program have been elucidated, we will now take a closer look at the computation of the amplitudes themselves. We will discuss each implemented model separately, ordered such that the respective model gradually builds on the previous model and the algorithms extend the previously described algorithms.

Hence, we will first discuss gluon amplitudes. While gluon amplitudes are not the easiest amplitudes to calculate from a physical point of view, they are simple to implement in terms of a computer program: The basic theory is given exhaustively by the Berends-Giele recursion relations discussed in section 2.4, so this chapter will contain less physical content but more ideas concerning computational evaluation.

The chapter is structured in a way which reflects the three basic ingredients for the computation of amplitudes, which were explained in section 3.1, beginning with the particle configuration.

4.1 The Particle Configuration — A Special Case for Gluons

Section 3.1 stated that there are in principle only two different particle configurations: One for the colored case and one for the case of a colorless theory. Both of these classes include all particle types, i.e. not only bosons but also fermions. In the case of gluon amplitudes we will not use any of these particle configurations, but employ a special case: The chapter on physical basics hopefully made it obvious that purely gluonic amplitudes imply a summation over all non-cyclic permutations, $S_n/Z_n = S_{n-1}$. To write an extra class only for permutations would be superfluous, since there are two simple ways of obtaining all permutations at hand.

On the one hand, the STL provides a method `next_permutation` which permutes elements of a container (for instance a `vector` with particle indices) given in terms of two iterators, which are basically pointers to container elements. Suppose the particle indices were stored in a container `particles`, then the function call looks like

```
std::next_permutation(particles.begin()+ 1, particles.end());
```

where `begin()` and `end()` yield the first and last element of the container which shall be permuted. `particles.begin()+ 1` tells the function to exclude the first particle from the permutation, resulting in all non-cyclic permutations only. This method per-

mutes according to the so-called lexicographical ordering, the specifics of which are not important at present. However, this ordering allows for a comparison of different permutations. The unit permutation $P_0 = (0, 1, \ldots, n-1)$ has lowest order and thus provides a good starting point. From there, each call of `next_permutation` advances to the next lexicographical ordering. Once the last permutation in this ordering has been reached, the sequence automatically starts again with the unit permutation. This is indicated by a boolean return value which yields true, unless the next permutation is the unit permutation. A great advantage of this STL function is that the container which is to be permuted does not need to contain integer indices ranging from 0 to any value; the datatypes and values can be completely arbitrary as long as they are lexicographically comparable.

The other method to generate permutations is by using a class `multi_index_permutation` from *particle scattering*. This class automatically creates a string of k elements where each element can take values in the interval $[0, N)$; N and k are constructor parameters. To obtain permutations, N and k have to be equal. The class provides methods to initialize the multi index, advance to the next configuration (i.e. next permutation) and check whether an overflow has occured. Overflow means that there exists no further permutation. In contrast to `next_permutation`, `multi_index_permutation` always permutes all indices, i.e. one cannot restrict the permutations to non-cyclic ones; however, this is not a problem in the case of gluon amplitudes since one can choose to leave out the last particle and feed only the first $n-1$ particles into the multi index. Furthermore, the values to be permuted have to be integer indices starting from 0, which restricts the use of this class compared to the STL function; still, this is of no concern for the present case of non-cyclic permutations of indices. As an advantage over `next_permutation`, multi index also provides a function for the sign of the permutation, $\mathrm{sign}(P)$. Moreover, if one chooses $k < N$, the multi index yields all *variations* instead of permutations; this will be used in section 4.4.

It turns out that `multi_index_permutation` — although implemented completely independent of the STL algorithm — also advances according to lexicographical ordering. The present implementation for the gluon amplitudes uses `multi_index_permutation` in order to be conform with *particle scattering*. However, the function `next_permutation` will be important later for reasons which will become obvious in the next chapters.

4.2 The Leading Order Color Matrix — A Simplified Approach

The fact that gluons do not couple to any terms of order $\mathcal{O}\left(1/N_c\right)$ enables us to treat the $SU(3)$ gauge group as a full $U(3)$ theory. In this section we will exploit this fact to provide a fast computation of the leading color matrix.

For better understanding, let us shortly recap the most important formulas for the composition of the color matrix. The color matrix can be decomposed as

$$M_{ab} = \bar{c}_a^{\{\bar{i},\bar{j}\}} P^{\{\bar{i},\bar{j},i,j\}} c_b^{\{i,j\}} \,, \tag{4.1}$$

where the upper indices in curly braces indicate the sets of color indices which the respective factors depend on. P is the projector term which connects adjoint color

indices with bars[1] to indices without bars. The projector term contains a projector for each external gluon, up to leading order they are given by:[2]

$$P_g = \delta_{\bar{i}i}\,\delta^{\bar{j}j}\,, \tag{4.2}$$

where each color index obtains an additional index for the particle number which it describes. c_a and c_b are vectors containing the color factors corresponding to the color ordered partial amplitude A_a and A_b (here, the particle configurations are denoted by the indices a and b). According to the previous discussion about the color-flow decomposition, these are given by simple strings of Kronecker deltas which connect the color indices properly:

$$c_a^{\{i,j\}} = \delta^{i_{P_a(1)}}_{j_{P_a(2)}}\delta^{i_{P_a(2)}}_{j_{P_a(3)}}\cdots\delta^{i_{P_a(n-1)}}_{j_{P_a(1)}} = \prod_{k=0}^{n-1}\delta^{i_{P_a(k)}}_{j_{P_a[(k+1)\%n]}},$$

note that the indices have to be permuted according to the particle permutation P_a which composes the particle configuration a.

4.2.1 Analytical Simplification

The full projector term including all external gluons reads

$$P^{\{\bar{i},\bar{j},i,j\}} = \prod_{k=0}^{n-1}\delta_{\bar{i}_k i_k}\,\delta^{\bar{j}_k j_k}.$$

Plugging this and the color factors with the respective permutations into equation (4.1), one obtains a product of $4n$ Kronecker deltas. However, this can easily be reduced to $2n$ deltas since the projector deltas are trivial; note that one has to distinguish between the two permutations P_a and P_b from the different color vectors:

$$\begin{aligned} M_{ab} &= \left(\prod_{k=0}^{n-1}\delta^{\bar{i}_{P_a(k)}}_{\bar{j}_{P_a[(k+1)\%n]}}\right)\left[\prod_{\ell=0}^{n-1}\delta_{\bar{i}_\ell i_\ell}\,\delta^{\bar{j}_\ell j_\ell}\right]\left(\prod_{m=0}^{n-1}\delta^{i_{P_b(m)}}_{j_{P_b[(m+1)\%n]}}\right) \\ &= \left(\prod_{k=0}^{n-1}\delta^{j_{P_a[(k+1)\%n]}}_{i_{P_a(k)}}\right)\left(\prod_{m=0}^{n-1}\delta^{i_{P_b(m)}}_{j_{P_b[(m+1)\%n]}}\right) \end{aligned} \tag{4.3}$$

After contracting the projectors with c_a, all adjoint color indices are gone. With a little thought, this formula can be simplified further: Two Kronecker deltas get contracted when their i index is identical, i.e. when

$$P_a(k) = P_b(m) \iff m = P_b^{-1}P_a(k)\,.$$

Since the products in equation (4.3) run over all indices, there is exactly one delta in each of the terms for which this condition is true. Hence, it is possible to rewrite the

[1] Note that in this case the term *adjoint* is not to be confused with the adjoint representation of a group; But let us refer to these indices as adjoint indices since they always appear with the hermitian conjugate of the color vector c_a.

[2] Note that this projector is equal to the product of a quark and antiquark projector, $P_g = P_q P_{\bar{q}}$, so if one would wish to perform a computation with additional quarks in leading order approximation, this is not difficult. However, we will treat full QCD only with full color in this thesis.

expression for the color matrix as follows:

$$M_{ab} = \prod_{k=0}^{n-1} \delta^{j_{P_a[(k+1)\%n]}}_{j_{P_b[(P_b^{-1}P_a(k)+1)\%n]}} \,. \tag{4.4}$$

This formula reduces the original $4n$ Kronecker deltas to merely n Kronecker deltas which have to be contracted, at the cost of having to obtain the inverse P_b^{-1} which is however negligible compared to the overhead that the contraction of $4n$ deltas brings along.

4.2.2 Computational Implementation

Now the above formula has to be translated into a computational form which enables automated computation. Although the concept of contracting Kronecker deltas is easy to comprehend with pencil and paper, it does not seem so natural in a programming language. However, there is a simple and fast method to display Kronecker deltas as numbers which resembles the method of storing momentum sums, see section 3.5.

Of the original four different types of color indices, $i, j, \bar{i}$ and $\bar{j}$, only j remains in the final formula. There is one such index for each external particle: $j_0, j_1, \ldots, j_{n-1}$. Each Kronecker delta can be represented by an unsigned numerical value[3] where the single bits correspond to the different indices j:

$$\begin{array}{r|cccccc} \text{Bit} & 0 & 1 & 2 & 3 & \ldots & n-1 \\ \text{Color index} & j_0 & j_1 & j_2 & j_3 & \ldots & j_{n-1} \end{array} \,, \tag{4.5}$$

This string of bits should be interpreted as follows: The bits corresponding to the two color indices of the delta should be set to 1, all other bits should be set to 0. To illuminate this further, look at the following examples for $n = 8$ particles. A condensed notation is used, where round brackets denote a binary representation of the number:

$$\begin{aligned} \delta_{j_0 j_2} &\mathrel{\widehat{=}} (10100000) \\ \delta_{j_2 j_6} &\mathrel{\widehat{=}} (00100010). \end{aligned} \tag{4.6}$$

Note that the actual decimal number corresponding to the bits is not important since the algorithm works exclusively in binary representation.

The benefit of this representation is that contracting deltas becomes very simple and efficient. We will look at the method with the help of the deltas from above; they contract as follows:

$$\delta_{j_0 j_2}\, \delta_{j_2 j_6} = \delta_{j_0 j_6}.$$

The contraction happens in two steps: First, one has to check whether two deltas can be contracted at all, i.e. whether they share at least one equal index. This can be accomplished with a binary `AND` operation on both deltas. If the result is zero the deltas do not have equal indices and cannot be contracted. In the above example,

$$\delta_{j_0 j_2} \wedge \delta_{j_0 j_6} \;\mathrel{\widehat{=}}\; (10100000) \wedge (00100010) = (00100000) \neq 0 \,,$$

[3] In this representation one has to make sure that the used data type is large enough to store all indices. On modern desktop systems, a convenient type is `size_t` from the STL which can safely be supposed to be at least 16 bits large which is sufficient for $n \leq 16$ particles.

it turns out that the deltas can be contracted, as it should be. The second step is to perform the contraction itself, which can be done as easily using a `XOR` operation:

$$\delta_{j_0 j_2} \otimes \delta_{j_2 j_6} \;\widehat{=}\; (10100000) \otimes (00100010) = (10000010) \;\widehat{=}\; \delta_{j_0 j_6} \,.$$

During such a contraction the case that the deltas contract to form a trace over a single delta will occur. In such a case, the `XOR` operation will yield 0 instead of a new Kronecker delta. As an example, take $\delta_{j_0 j_1}\delta_{j_0 j_1} = \mathrm{Tr}\,\{\delta\} = N_c$:

$$\delta_{j_0 j_1} \otimes \delta_{j_0 j_1} \;\widehat{=}\; (11000000) \otimes (11000000) = (00000000) \;\widehat{=}\; 0 \,.$$

Hence, the `AND` and `XOR` operations are all that is needed for performing contractions.

Of course, the n Kronecker deltas which appear have to be computed and stored first. This is best accomplished using a `list` from the STL; it provides fast removal or insertion of elements but slow index access.[4] However, in the present case, index access is not required whereas removal of single elements is frequently needed.

During the computation of the deltas according to equation (4.4), it may occur that a Kronecker delta has two identical indices from the start, meaning that one has to take the trace over it. In such a case, the delta is not stored in the `list` but a factor of N_c for the trace is directly multiplied to the color matrix element, since storing such deltas would only yield more computational effort.

Once all Kronecker deltas are computed, this list has to be fully contracted. This is done using two iterators to deltas with the following initial setting: One iterator I_1 points to the first delta in the `list`, the second iterator I_2 points to the second delta. Now perform the following steps iteratively until all deltas are contracted:

(i) If a contraction is impossible (e.g. $\delta_{j_0 j_1} \wedge \delta_{j_2 j_3} = 0$), move the second iterator I_2 to the next delta and retry the contraction.

(ii) If the deltas can be contrated, replace the delta at I_1 by the delta resulting from the contraction and remove the delta at I_2 from the `list`. If the new delta at I_1 is a trace (i.e. no bit in the delta is set), move I_1 to the next element in the `vector`. Let I_2 point to the element following I_1 and perform the next contraction.

This procedure has to be repeated until any incrementation of I_1 in step (ii) would move I_1 out of the bounds of the `list` because then no more un-contracted deltas are in the list. All elements in the vector which preceed I_1 are 0 and represent traces, thus the `list` will only consist of traces in the end and the number of elements in the `list` is equal to the number of traces. Hence, the final element of the color matrix is given by N_c to the power of the number of elements in the fully contracted `list`.

Figure 4.1 displays an example for five Kronecker deltas which get contracted according to the algorithm. The resultant color matrix element for this example is, in a loose but suggestive notation,

$$N_c^{\texttt{sizeof}(\text{list of deltas})} = N_c^2 .$$

[4] Instead, a `vector` has opposite features: Fast index accessing but slow removal or insertion of elements since these require moving all of the following elements.

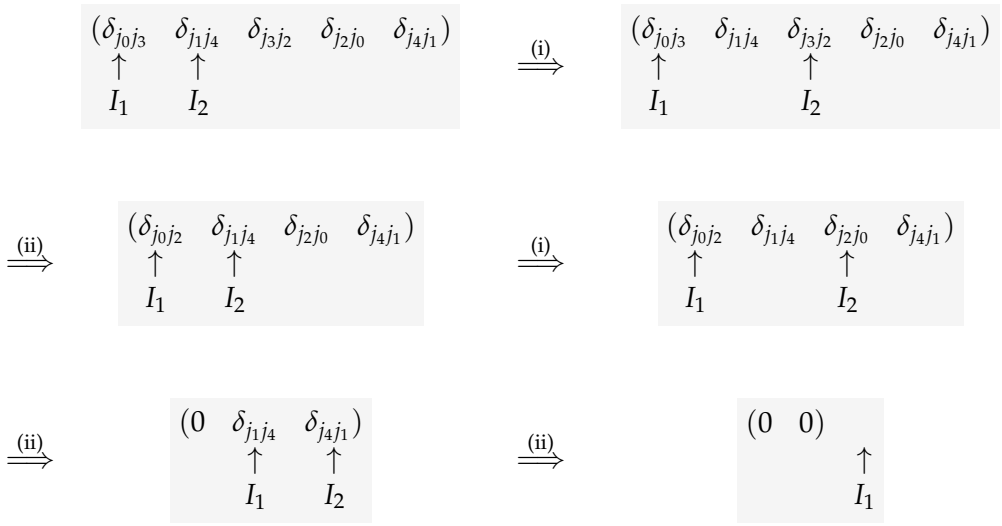


Figure 4.1: Example for the contraction algorithm (process $gg \to ggg$)

4.2.3 Further Improvements

With the above considerations, it is simple to compute each element of the color matrix. However, there is no need to compute all $|P|^2$ matrix elements, where $|P|$ denotes the number (the *cardinality*) of all non-cyclic permutations $P \in S_n/Z_n$. One can use two tricks to reduce the amount of matrix elements that have to be computed:

- Every matrix element on the main diagonal has the value N_c^n. This can be easily seen from equation (4.3): On the main diagonal, both permutations P_a and P_b are identical, so every delta from $\bar{c}_a$ forms a trace with one delta from Pc_b. Since there are n deltas in each term, this results in $\left[\mathrm{Tr}\left\{\delta\right\}\right]^n = N_c^n$.
- From equation (4.2), it is also apparent that the color matrix is symmetric: Since each projector term is symmetric under the exchange of adjoint indices $\bar{i}, \bar{j}$ and normal indices i, j, the matrix given in equation (4.1) is also symmetric under the exchange $a \leftrightarrow b$.

After applying these two tricks, there are only $\frac{|P|^2-|P|}{2}$ matrix elements left which require the computation of Kronecker deltas and their contractions, whereas the full matrix of course contains $|P|^2$ elements.

The matrix elements are stored in a data structure provided by the *BOOST* library *uBLAS*, a `symmetric_matrix` with double-valued entries. As the name suggests, this matrix is specialized for symmetric matrices and it only stores one triangle of the matrix, which saves much memory for processes with many gluons since the number of partial amplitudes is $(n-1)!$ in the purely gluonic case.

4.3 Computation of the Partial Amplitudes

The partial amplitudes are not difficult to implement: One has to turn the Berends-Giele recursion from equation (2.22) into a recursive function `gluon_current` and compute this current for $n-1$ particles. To illustrate how this works, pseudo-code

is shown below. Note that some calls and variable names have been altered to improve readability and make the code self-explanatory; furthermore, a few parts have been cut out since they are of no relevance at present.

```
bispinor_neg gluon_current(
  index start, // First particle in current
  index end, // Last particle in current
  bool on_shell) // On-shell parameter; see below
{
  // Recursion start:
  if (end == start)
  {
    // permutation is a multi_index_permutation object,
    // which is a member of the gluon_amplitude class:
    index perm_start = permutation(start);
    return gluon_polarization(
      momentum[perm_start],
      reference_momentum[perm_start],
      helicity[perm_start]
    );
  }

  bispinor_pos sub_current;

  for (index i = start ; i < end; ++i)
  {
    // Three gluon vertex:
    sub_current.sum_up(three_gluon_vertex(
       -get_momentum_sum(start, end),
       get_momentum_sum(start, i),
       get_momentum_sum(i + 1, end),
       gluon_current(start, i, false),
       gluon_current(i + 1, end, false)
    ));

    for (index j = i + 1; j < end; ++j)
    {
      // Four gluon vertex:
      sub_current.sum_up(four_gluon_vertex(
        gluon_current(start, i, false),
        gluon_current(i + 1, j, false),
        gluon_current(j + 1, end, false)
      ));
    }
  }

  // If the current shall be on-shell, attach no propagator:
  if (on_shell)
    return convert_to_bispinor_neg(sub_current);

  // Else, attach propagator:
  return gluon_propagator(get_momentum_sum(start,end), sub_current);
}
```

The `on_shell` parameter is used only for the "largest" current, i.e. the current with $n-1$ particles. If it is set to true, the attachment of a propagator is avoided so that the current becomes an on-shell current. This is needed since the final particle which "turns the current into an amplitude", in a manner of speaking, gets attached to the current with $n-1$ particles in terms of its polarization vector. This happens outside of this current function, in the function `partial_amplitudes`, which was shortly mentioned in section 3.1. This function is also responsible for looping over all non-cyclic permutations and storing the values in an appropriate data structure. As mentioned before, this structure is provided by the *BOOST* uBLAS library to provide efficient multiplication with the symmetric matrix from *BOOST*.

4.4 Optimization: Remembering Subcurrents

The above description applies for the class `gluon_amplitude_0`, however, section 3.1 mentioned three classes for gluon amplitudes. The above method works well, so why should there be the need for another version? The answer is simple: With the recursive method, the same building blocks get computed over and over again. Imagine two partial amplitudes with different particle permutations; suppose that one partial amplitudes requires the computation of the current $G(0,1,2,3)$ and the other $G(0,1,3,2)$. Both of these currents again depend on smaller subcurrents, which get computed recursively. For both partial amplitudes, this includes the computation of $G(0,1)$ — this subcurrent gets computed twice! Or, even worse: Each of the two currents will also involve computing the polarization vectors as recursion start — all four polarizations are computed again, separately for each partial amplitude. Let us substantiate this with a numerical example: During the computation of one partial amplitude for an eight gluon process, 11 612 161 polarization vectors are computed. If the polarizations did not have to be computed over again, this number could be reduced to merely 8 — note that this does not even include currents of higher order. Undoubtedly, the naive recursion is an extraordinary waste of computation power.

As a general rule, it is easy to see that the fewer legs a current has, the more often it has to be computed, since each computation of a current requires the knowledge of its subcurrents.

There are two possible ways of dealing with this problem:

1) Extending the recursion to allow for storing and retrieving currents on the fly, i.e. the recursion basically stays the same but each time a current shall be computed, the program first investigates whether this current has already been computed and can be reused. If not, the current is computed and stored for later use.

2) Turning recursion into iteration by first computing the smallest currents (i.e. polarizations) and working the way up to larger currents.

The first idea is implemented in `gluon_amplitude_1`, the second can be found in the class `gluon_amplitude_2`. Both methods require a way to store the currents.

So, what is the most suitable data structure for saving currents? One possibility is to make use of a hash table. With hash tables, there are two disadvantages: First, the particle numbers which identify the current have to be converted using a hash

function which costs time. Second, searching for a certain value in a hash table is of logarithmic complexity on average (e.g. using binary search). The first problem cannot be circumvented, because one always has to turn the particle interval into some sort of single value which identifies the current. The second problem can be avoided if one does not have to search for the value, but if one *knows* directly where to find it — this property is provided by a STL `vector`, which grants access to elements in constant time as long as the index of the element is known.

Hence, the most suitable container for currents is a `vector<bispinor_neg>`. It remains to find a convenient mapping from a set of particle indices to a natural number. First of all, the currents can be distinguished by the number of legs, i.e. we can store all currents with a fixed number of currents in a separate `vector`. Hence, the mapping does not include all possible particle indices, but it has to work for an arbitrary, but fixed number of legs. This set of particle indices obeys the following rules:

- Each particle index may only appear once; currents like $G(1,1,2)$ do not exist.
- For a current with k legs, only k of the total n_g indices are needed. However, all possible combinations of k indices have to be taken into account.

It turns out that all possible indices for a sub-current with k particles are given by all variations without repetition. A variation is classified by the set

$$V_n^k := \{(x_1, x_2, \ldots, x_k) | x_i \in \{1, 2, \ldots, n\} \setminus \{x_1, x_2, \ldots, x_{i-1}\}\} \; .$$

The required mapping has to be injective, for obvious reasons, but for a full optimization of the program it is reasonable to aim a little higher: With a bijective mapping

$$\varphi : V_n^k \longrightarrow \mathbb{N} \; , \; \varphi \text{ bijective}$$

one also uses as little memory as possible since the bijective property ensures that there is no unused index where an irrelevant current (or no current at all) would be saved.

An algorithm which realizes φ can be written as follows: Let n_g denote the total number of gluons and let a_k be the sequence of particle numbers that describes the current, i.e. $G(a_1, a_2, \ldots, a_k)$.

i) Calculate a new sequence $\{b_m, m \in [1,k]\}$ according to the following rule:

$$b_m = a_m - d \qquad \text{where} \qquad d = \sum_{j=1}^{m-1} \begin{cases} 0 & \text{for } a_j < a_m \\ +1 & \text{for } a_j \geq a_m \end{cases} \; .$$

ii) Use b_m to define another sequence c_m recursively:

$$c_m = (n_g - m) \cdot (b_m + c_{m-1}) \qquad \text{where } c_1 = (n_g - 1) \cdot b_1 \; .$$

iii) Finally, the desired index number $\mathcal{I} \in \mathbb{N}$ is given by

$$\begin{aligned} \mathcal{I} &= b_k + c_{k-1} \\ &= b_k + (n_g - k + 1) \cdot \Big(b_{k-1} + (n_g - k + 2) \times \\ &\qquad \big(b_{k-3} + \cdots + (n_g - 2) \cdot (b_2 + (n-1) \cdot b_1) \big) \Big) . \end{aligned}$$

Although the algorithm looks rather involved, the implementation is very short; note that, again, this is partly pseudo-code:

```
index get_current_index(
  const multi_index_permutation& array,
  index start,
  index k)
{
  index result = 0;

  for (index a = start; a < start + k; ++a)
  {
    index *= n_g - (a - start);

    index tmp = array(a);
    for (index b = start; b < a; ++b)
    {
      if (array(b) < array(a))
        tmp--;
    }

    result += tmp;
  }

  return result;
}
```

For the above code, it is important to know that the `multi_index_permutation array` is always of size $n_g - 1$. If one wants to get an index for, say, a current with only two particles, one has to set `start` to the first index describing the current and k to the number of particles in the current.

With this method, it is easy to implement the adapted recursion mentioned in item 1) above. However, the iterative method from item 2) requires a bit more work. To reverse the recursion, one has to make sure that all possible currents with k legs are computed before one starts computing the currents of with $k+1$ legs. How all possible currents can be generated has already been mentioned above: By looping over all variations. Fortunately, all variations can easily be obtained with the help of the `multi_index_permutation` class, by choosing the constructor parameter k equal to the number of legs of the current and N equal to the number of particles in the variation, i.e. $N = n_g - 1$. In this way, one can start with $k = 1$ and compute all polarizations up to $k = n - 2$. It is not neccessary to also save the currents with $n - 1$ legs because they make up the partial amplitudes along with the final polarization vector — and since partial amplitudes are only required once, they need not be saved.

This completes the discussion about optimization techniques for gluon amplitudes. Results for the performance of these optimizations are given in chapter 8.

Chapter 5

QED Amplitudes — Adding Fermions

After the last chapter introduced the basic concepts for the computation of amplitudes in terms of gluons, we will now turn to the first complication: The incorporation of fermions. In order to avoid further difficulties originating in the $SU(3)$ color structure, we will first look at the colorless theory of QED, a $U(1)$ gauge theory.

The bosonic part of QED is considerably simpler than the gluon counterparts of $SU(3)$. Instead of eight different bosons in QCD (distinguished by the color degree of freedom), the simpler group structure of $U(1)$ produces only a single boson, namely the photon which does not interact with itself; for that reason, there are also no processes such as $\gamma\gamma \to \gamma\gamma$ at tree-level. Hence, there is only one vertex which couples two leptons to a photon. As generally known, the fermion photon vertex of QED only differs from the quark gluon vertex by its lack of the color generator T^a. According to this, the kinematical structure is identical and we can use the color-ordered QCD Feynman rule in double line notation also for QED (see the appendix).

Furthermore, there is no color matrix to compute; all partial amplitudes weigh equally. To obtain all interferences among the partial amplitudes, we could either use a matrix where every entry is set to one, or we could get rid of the vector and matrix structure altogether and simply sum the partial amplitudes. The second approach is certainly more efficient since it avoids the expensive vector-matrix-vector product needed for the computation of the squared amplitude, see equation (3.7) on page 37. Still, the squaring in terms of this product is done centrally by the `general_amplitude` class, so it would be unreasonable to break out of this concept. As a solution to this problem, we will retain this general concept, but with a 1×1 color matrix and a partial amplitude vector with only one element. This element then contains the sum of all partial amplitudes.

Anyway, this does not change the basic approach which was explained in chapter 3 and already applied to the gluon amplitudes. Hence, we first need a particle configuration.

5.1 A Particle Configuration with Fermions

The basic idea of treating an arbitrary number of fermions within color decomposition was introduced in section 2.2.4: Fermion pairs can be counted as one pseudo-particle so that the non-cyclic permutations run over all pseudo-particles. Additionally, one has to take care of permuting either all fermions or all antifermions among each other. However, in theories where there are no flavor changing currents, this permutation only has to involve (anti-)fermions of equal flavor.

This particle configuration for a colorless theory is implemented in the aforementioned class `particle_configuration_colorless`. How is this realized? Since we need separate permutations for the fermions and for the pseudo particles, two particle index `vector`s are created, one for each of these permutations. Both are then computed separately and joined together to form the full particle configuration internally. These tasks are assumed by the member function `bool next()`. Initially, all data is set to unity permutations. A call of `next()` then performs the steps visualized in figure 5.1: It first tries to obtain the next fermion permutation by means of the STL function `next_permutation` without affecting the permutation of pseudo particles. If there exists such a permutation, the full permutation for all physical particles (let us call these *real* particles and, correspondingly, the *real* particle permutation) is computed. If the new fermion permutation is a permutation which has already been used, go to the next pseudo particle permutation. If this is also not available, all configurations have been used and the unity configuration is restored. In case the next pseudo particle permutation is available, the real permutation is computed. In principle, this finishes the computation. However, since QED is a theory which does not couple leptons of different flavors, another switch has been inserted which can veto fermion permutations where fermion pairs have different flavors. If this switch has been set, the fermion pairs gets checked for equal flavors at the end; if this check fails, the algorithm starts all over.

Merging the pseudo particle and the fermion permutations is done in another subroutine, however, we will not discuss this here since it is trivial to combine the two.

In the present implementation, the "unity state" which was mentioned in the above discussion signifies the state which the default cyclic ordering creates (see section 3.3). The non-cyclic permutations always run over the pseudo particles $(2, \ldots, n_p)$ for n_p pseudo particles. Since there are no pure photon amplitudes, the cyclic ordering always makes the first pseudo particle an lepton/antilepton pair. In the present program, the leptons get permuted (instead of the antileptons); this is pure convention, but it enhances "bookkeeping" for debugging purposes, since it ensures that the very first *real* particle is always the same antifermion, for each particle configuration.

5.2 The Partial Amplitudes

When fermions enter the game, one cannot write down recursion relations as easily as for gluons: When a photon current splits into a lepton/antilepton pair, one has to be careful how to split the current. In the standard Berends-Giele recursion for gluons, the sums run over all possible indices; this is correct since each subcurrent is neccessarily a gluon current for there are only gluons in the process. With fermions, however,

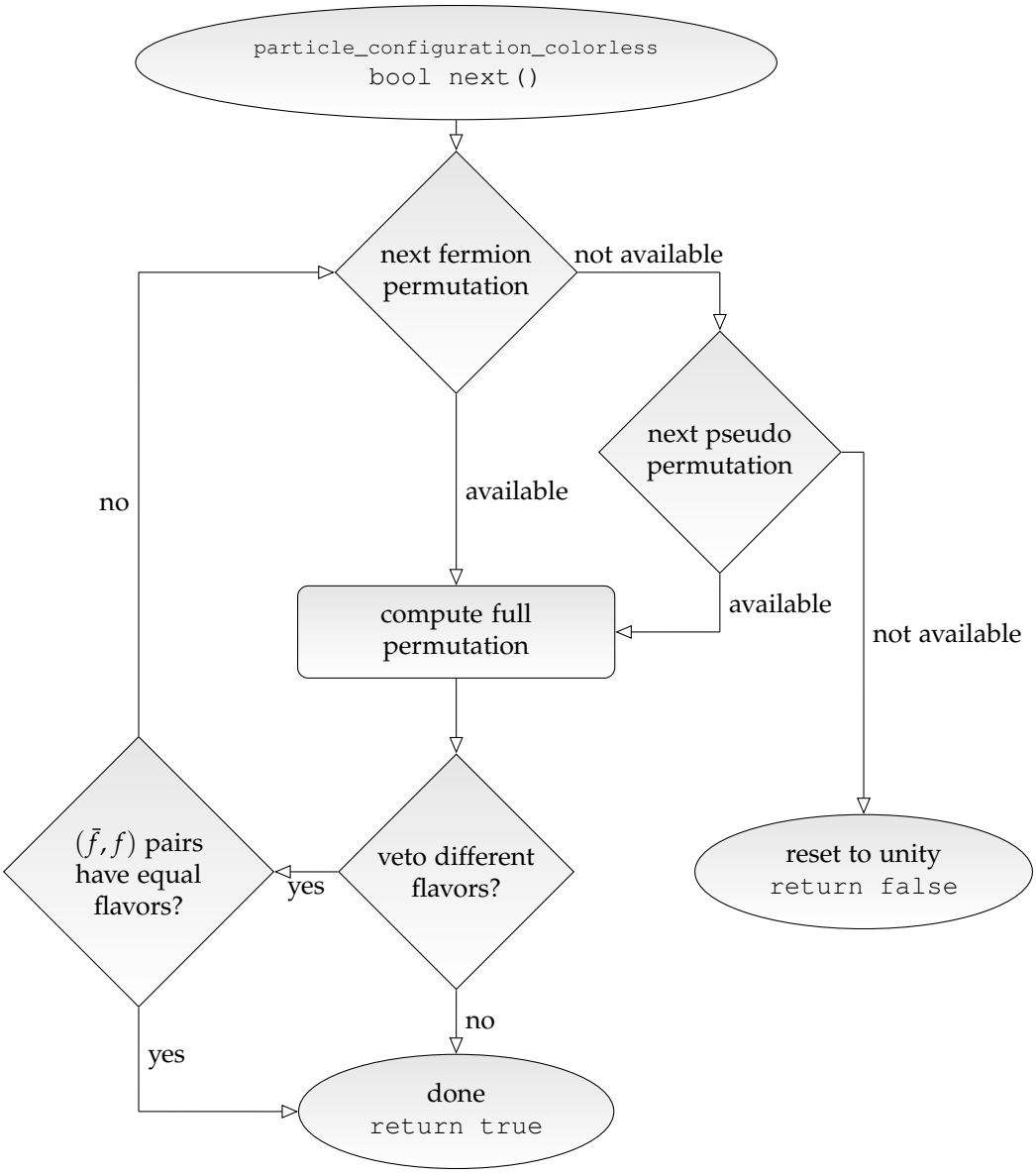


Figure 5.1: Flow chart for the `next` function of the class `particle_configuration_colorless`, which advances to the next particle configuration for a colorless theory with fermions.

the subcurrents emerging from splitting a photon current into lepton and antilepton subcurrents are not automatically currents of the correct type. This depends on the particle content of the respective currents; thanks to cyclic ordering one can be sure that if the first particle of the lepton current is a lepton, both resulting subcurrents are indeed currents of the right type.

As an example, let us take a look at the current $(\gamma, \gamma, \bar{\ell}, \ell, \gamma)$ and consider the splittings $(\gamma|\gamma, \bar{\ell}, \ell, \gamma)$, $(\gamma, \gamma|\bar{\ell}, \ell, \gamma)$ and $(\gamma, \gamma, \bar{\ell}, \ell|\gamma)$. They are not allowed since both subcurrents would be photon currents according to their particle content. The only valid splitting is $(\gamma, \gamma, \bar{\ell}|\ell, \gamma)$, where the first particle of the leptonic current, which is always the right current, is indeed a lepton.

Similar considerations apply to the splitting of lepton and antilepton currents. To transfer this into a formula, we will introduce a new symbol ε, which bears some similarity to a Kronecker delta. We define it as follows:

$$\varepsilon_i^p \equiv \begin{cases} 1 & \text{if particle } i \text{ is of type } p, \\ 0 & \text{else.} \end{cases} \tag{5.1a}$$

and its counterpart $\bar{\varepsilon}$:

$$\bar{\varepsilon}_i^p \equiv \begin{cases} 1 & \text{if particle } i \text{ is } \textit{not} \text{ of type } p, \\ 0 & \text{else.} \end{cases} \tag{5.1b}$$

The particle type will be given by the same suggestive notation which we employed before, i.e. γ for photon, ℓ for lepton, $\bar{\ell}$ for antilepton, etc.

Since general QED amplitudes contain three different types of particles, photons, leptons and antileptons, there are also three coupled recursion relations, one current for each particle type. We label these with the capital versions of the letters which denote their particle type, i.e. Γ for the photon current and $(\overline{L})$ L for the (anti-)lepton current. If one does not veto lepton/antilepton pairs with different flavors by the particle configuration, one has to perform this veto in the recursion relations. To achieve this, we add a flavor index for the lepton and antilepton currents. This veto can then be realized by a Kronecker delta with flavor indices f. However, it is presented here only for completeness; as stated in the previous section, non-matching flavor permutations are vetoed by the particle configuration in our case.

The recursion relations are then given by the following formulas:

$$\Gamma^{\dot{A}B}(1,\dots,m) = \sum_{j=1}^{m-1} \left(\delta_{f_1 f_2}\, \varepsilon_{j+1}^{\ell}\right) \overline{L}_{f_1}^{\dot{C}}(1,\dots,j)\, L_{f_2}^{D}(j+1,\dots m)\, V_{D\dot{C}F\dot{E}}^{(\gamma\bar{\ell}\ell)}\, P_{(\gamma)}^{\dot{E}F\dot{A}B}(p_{1,m})$$

$$L_f^{B}(1,\dots,m) = \sum_{j=1}^{m-1} \left(\delta_{f f_1}\, \bar{\varepsilon}_{j+1}^{\ell}\right) L_{f_1}^{D}(1,\dots,j)\, \Gamma^{\dot{E}F}(j+1,\dots,m)\, V_{F\dot{E}D\dot{C}}^{(\gamma\bar{\ell}\ell)}\, P_{(\ell)}^{\dot{C}B}(p_{1,m})$$

$$\overline{L}_f^{\dot{A}}(1,\dots,m) = \sum_{j=1}^{m-1} \left(\delta_{f f_2}\, \bar{\varepsilon}_{j+1}^{\ell}\right) \Gamma^{\dot{C}D}(1,\dots,j)\, \overline{L}_{f_2}^{\dot{E}}(j+1,\dots,m)\, V_{D\dot{C}F\dot{E}}^{(\gamma\bar{\ell}\ell)}\, P_{(\ell)}^{\dot{A}F}(p_{1,m})$$

5.3 Dirac Spinors

These relations can be implemented in the same straight forward manner as the gluon current, see section 4.3. Since we work in the Weyl-van der Waerden formalism, the leptonic currents have to be implemented twice: Once for each helicity of the current. This leads to a doubling of code, since the currents for the two different helicities only differ by the spinor types which they return, basically. Any such doubling of code is not only more difficult to read, but also more error-prone since any change has to be made at two places in the code. Thus, it is reasonable to combine the two helicities in some way.

There are two other arguments to support this statement: First, for the inclusion of massive fermions, one has to choose massive spinors which are mixtures of the Weyl-van der Waerden helicity eigenstates. Second, the replacement of particle polarizations and spinors by the continuous versions for the purpose of Monte Carlo integration (see section 3.6.2) mixes both helicity eigenstates as well.

Hence, it is more convenient to use Dirac spinors for helicity integration. In extension to *particle scattering*, classes for Dirac spinors, `dirac_bra` and `dirac_ket`, were added, which gather positive and negative helicity eigenstates as follows:

$$\langle p| = \left(\langle p-|\,, \langle p+|\,\right) \qquad , \qquad |p\rangle = \begin{pmatrix} |p+\rangle \\ |p-\rangle \end{pmatrix} \tag{5.2}$$

Since these classes internally use the Weyl spinor classes, the implementation for any operations such as multiplication, spinor products, etc. is straight forward, as shown in the following example which shows a simple Dirac spinor product:

```
1 complex_d operator* (const dirac_bra& bra, const dirac_ket& ket)
2 {
3   return
4     bra.get_pos() * ket.get_neg() + bra.get_neg() * ket.get_pos();
5 }
```

(Note that `get_pos()` and `get_neg()` are getters for the two-component helicity Weyl spinors). In a similar fashion, all Feynman rules and spinor operations have been translated from Weyl spinors to Dirac spinors.

5.4 Optimization Techniques

The presently implemented version of QED amplitudes is not yet optimized in such a sophisticated way as the gluon amplitudes are. However, some effort has been made in light of the fact that the particle polarizations are the currents which get computed most often in a purely recursive approach.

A problem which generally arises when fermions are included is that the spinor types of the currents, meaning bispinor or Dirac spinor, differ from each other. Hence, storing all currents in one container is not possible since a STL container only accepts one fixed datatype; one could also say STL containers follow the "multi-value, single type" motto. However, there exists a solution to this problem in terms of the *BOOST* library *Variant*, which in contrast provides a "multi-type, single value" container. If we define a general current type as

```
typedef boost::variant<bispinor_neg, dirac_bra, dirac_ket>
      current_type;
```

we can take a `vector` of this current type to store both bosonic and fermionic currents.

As of now, the implementation only pre-computes and stores the polarization vectors, which are the currents that need to be computed most often in a recursive approach. In principle, this method can be extended to store currents with more legs as well, analogously to the ideas presented for the gluon amplitudes. However, all attempts in this direction have been unsuccessful to produce faster computations than with the recursive approach; in fact, the computation took much longer. This might be a performance penalty caused by the use of the variant container; but it may also have other, presently unkown, reasons.

These problems also led to another idea: Instead of using a container for different data types, one could create a new unified spinor type which encompasses all other spinor types. This is possible since both bispinors and Dirac spinors have four components. The corresponding class has to consist of five elements: The four spinor components plus an additional tag which identifies the type of spinor. One has to take special care, though, since the definition of operators (i.e. multiplication, contraction, conversion operators, etc.) is a delicate subject which demands a case by case analysis for the different spinor types. For example, a spinor product between a Dirac bra and ket vector has to be treated completely different from a product between two bispinors, although one would normally denote both by a "$*$" operator. However, these are complications which affect the readability and useability of the program, but not the performance, if cleverly implemented.

This unified spinor has partly been implemented and tested for a few QCD amplitudes with positive results. However, this is still in the stage of current research and not included in the current version of the program.

To sum this section up, some effort has been put into the optimization of the purely recursive approach of computing amplitudes with fermions, however, the only working results as of now include the pre-computation of particle polarization which, nevertheless, speeds up the computation.

Chapter 6

QCD Amplitudes — The Return Of Color

This chapter deals with the computation of QCD amplitudes. QCD amplitudes can be expanded into a series in the color dimension factor N_c. Sometimes, amplitudes are only calculated in leading order, i.e. all terms of order $\mathcal{O}\left(1/N_c\right)$ get neglected. In doing so, one makes an error of roughly 33 % in the QCD case of $N_c = 3$. Since we do not want to make such an error, but obtain a prediction as precise as possible, we will tackle the task of computing up to all orders.

So, how do we do that in relation to what we have already accomplished? To the pure gluon couplings we dealt with in chapter 4, we now add the coupling to quarks. The basic principle behind this is the same as for QED, but the $SU(3)$ color structure additionally causes far-reaching complications: Not only does this lead to a more involved particle configuration — which has to include information on the color cluster of each particle, but the partial amplitudes also have to include the $U(1)$ gluon, which results from the color-flow representation treatment of $SU(3)$ as $U(3) - U(1)$. In addition, we also have to update the algorithm for the color matrix.

6.1 A Particle Configuration With Color Clusters

Let us first deal with the particle configuration. In full QCD, we have gluons, quarks and antiquarks — these can be treated in exactly the same way as we did for the QED particle configuration. But, since the color structure of the gluon propagator now includes a $U(3)$ and a $U(1)$ term, we must distinguish between the two gluons. Recall from section 2.2.3 that we only have to do this for *virtual* gluons, not for external gluons.[1] For this reason, we have to introduce the concept of color clusters. As explained in chapter 2, different clusters only couple to each other via the $U(1)$ gluon. Hence, we somehow have to assign a color cluster to each external particle and add this information to the particle configuration. For each particle permutation, there are several possible configurations of color clusters, depending on the particles in the process.

Before presenting an algorithm which generates all cluster configurations for a fixed cyclic ordering, we will first look at the structure of color clusters more closely and state

[1] This will be included in the color matrix.

some rules which color clusters have to obey.[2] Examples are given for the following cyclic ordering of particles:

$$g, \bar{q}, q, g, \bar{q}, q, \bar{q}, q, g.$$

- The maximum number of color clusters for a process (and therefore each of its particle permutations) is equal to the number of quark/antiquark pairs. Thus, we have to take into account all cluster configurations with $n_{\mathrm{cl}} = 1, 2, \ldots, n_q$ clusters.

- We assign color clusters by natural numbers, e.g.

$$\underbrace{g, \bar{q}, q}_{\text{cluster 1}}\,,\ \underbrace{g, \bar{q}, q}_{\text{cluster 2}}\,,\ \underbrace{\bar{q}, q, g}_{\text{cluster 3}} \mathrel{\widehat{=}} (111222333).$$

  These numbers are merely labels, so the above cluster configuration (111222333) is the same as configuration (111333222). Without loss of generality, we can thus assume the first particle to always belong to cluster 1. Additionally, we can demand that new clusters always get introduced in ascending order; for the above example this discards configuration (111333222).

- Each cluster has to contain at least one quark/antiquark pair. Furthermore, both particles that form a quark/antiquark pair belong to the same cluster, hence we can deal with pseudo particles instead of real particles; note, however, that the final particle configuration has to be available in terms of real particles for the computation of the partial amplitudes.

- Color clusters can contain embedded color clusters. For the above example, a possible configuration is (122111331), where both clusters 2 and 3 are separately embedded in cluster 1. This example is displayed graphically in figure 6.1.

  Embedded clustering can be nested, i.e. a cluster can contain an embedded cluster, which again contains an embedded cluster. As one can easily verify, such nesting can only occur if there are at least four clusters, i.e. four antiquark/quark pairs.

  This embedding generally succumbs to the following rule: If a cluster j is interrupted by another cluster k, cluster k has to be fully assigned until cluster j can be continued. For the above example, consider the cluster configuration (122122111) which violates this rule: No diagram can be drawn which realizes this configuration.

Now, it remains to realize all possible cluster configurations in a computational form. Steps 1) to 3) from the algorithm which we present in the following have already been established in [14] and are implemented in the present form. This algorithm ends with determining the first particle for each cluster and the number of particles per cluster. We extend it here by step 4) which shortly describes a simple method to generate the color cluster number for each particle.

1) Sum over all possible numbers of clusters $n_{\mathrm{cl}} = 1, 2, \ldots, n_q$, where n_q is the number of quark/antiquark pairs.

[2] This analysis draws on [14], where color clusters are treated in detail.

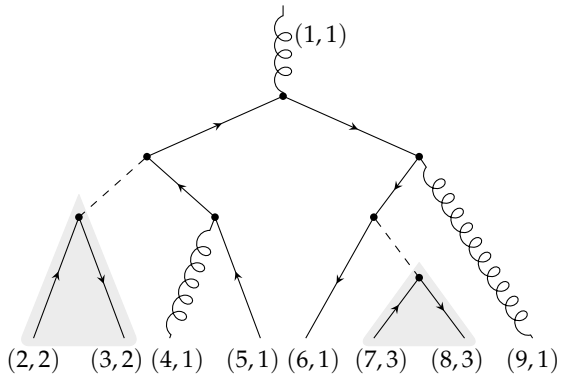


Figure 6.1: Diagrammatic example for embedded color clusters. The given pairs of numbers denote (particle number, cluster). The diagram is drawn in a Berends-Giele fashion, where the first particle has to be applied to the on-shell current; this first particle is shown as the upper leg. The light gray background shows the particles which are in separate clusters.

2) For each n_{cl}, distribute the number of pseudo particles among the clusters by summing over all partitions of $n_p = n_g + n_q$ pseudo particles into n_{cl} parts where part j contains n_j^{cl} parts; this is the number of particles in cluster j. Thereby, the condition

$$\sum_{j=1}^{n_{\mathrm{cl}}} n_j^{\mathrm{cl}} = n_p$$

has to be fulfilled so that this mapping is surjective (each pseudo particle has to belong to exactly one cluster). Also note that the ordering of clusters has to be taken into account, e.g. the partitions $(1,2,2)$, $(2,1,2)$ and $(2,2,1)$ have to be distinguished.

3) For each such partition, the starting point (i.e. the first particle) of each cluster can be chosen in various ways due to different possibilities of embedding clusters, consider for example the configurations (111222333) and (122211333). Here, the number of clusters, the number of particles per cluster and the starting points for clusters 1 and 3 are identical, however, cluster 2 starts at particle 4 in the first case and particle 2 in the second case. These possible starting points can be obtained by the following algorithm:

Define a set $m = (m_2, m_3, \ldots, m_{n_{\mathrm{cl}}})$ of $n_{\mathrm{cl}} - 1$ values which obeys the conditions

$$m_j \leq m_{j+1} \quad \text{and} \quad 1 \leq m_j \leq 2 - j + \sum_{k=1}^{j-1} n_k^{\mathrm{cl}}. \tag{6.1}$$

The starting points $n^0 = (n_1^0, n_2^0, \ldots, n_{n_{\mathrm{cl}}}^0)$ for each cluster then follow by

$$n_j^0 = \begin{cases} m_j + j - 1 & \text{for } j \geq 2 \\ 1 & \text{for } j = 1. \end{cases}$$

4) Obtaining the color cluster for each particle can be done by various methods.

The only difficulty is caused by the embedding of clusters, which is not directly obvious from the data gathered in steps 1) to 3). To avoid problems of this kind, one can exploit the fact that cluster j cannot be embedded in cluster k if $j > k$, a fact which follows directly from the above discussion. The following idea respects this:

(i) Start with the last cluster, number n_{cl}, which cannot be interrupted by any other cluster, hence all particles of this cluster appear in sequence and can be assigned easily. The first particle of the cluster is $n^0_{n_{\mathrm{cl}}}$.

(ii) Go to the next lower cluster number. If this cluster contains any embedded clusters, these clusters have a higher number. Hence, they have been dealt with before, so one knows exactly which particles belong to an embedded cluster. This makes it very easy to assign the current cluster number to the particles.

(iii) Repeat step (ii) until all particles are assigned to clusters.

In order to realize this algorithm, one has to implement the partitions and also all possibilities for choosing the set m. *Particle scattering* provides both in terms of `multi_index` classes, which were already used for permutations and variations in gluon amplitudes. The special realizations needed here are `multi_index_partition` for the partitions and `multi_index_ordered_eq_indv` for the set m. The functionality of the first should be clear; the second multi index realizes all conditions which m has to obey. The upper limits for the elements of m have to be computed separately and forwarded to the multi index in terms of a constructor parameter.

The only remaining task is to plug this algorithm into the particle configuration for a colored theory. A flow chart is shown in figure 6.2. Basically, the particle configuration is the same as for the QED particle configuration but complemented by the cluster algorithm. The reason that the parts from the cluster algorithm are inserted in between the fermion permutations and the pseudo particle permutations are due to implementation details which are not important to discuss.

6.2 The Full Color Matrix

In contrast to the leading order color matrix, it is not possible to simplify the algebraic structure of the full color matrix before the computational evaluation. This is due to the fact that the color projectors are no longer trivial "connectors" between the two color factors, but they contain the $U(1)$ gluon color structure,

$$P_g = \delta_{i\bar{i}}\,\delta^{j\bar{j}} - \frac{1}{N_c}\delta^j_i\,\delta^{\bar{j}}_{\bar{i}},$$

which cannot be treated as generally as before.

In comparison to the leading order algorithm, there are several complications:

1) Depending on the number of (external) gluons n_g, the projector term is no longer a single string of Kronecker deltas but a *sum* of strings of Kronecker deltas. Mul-

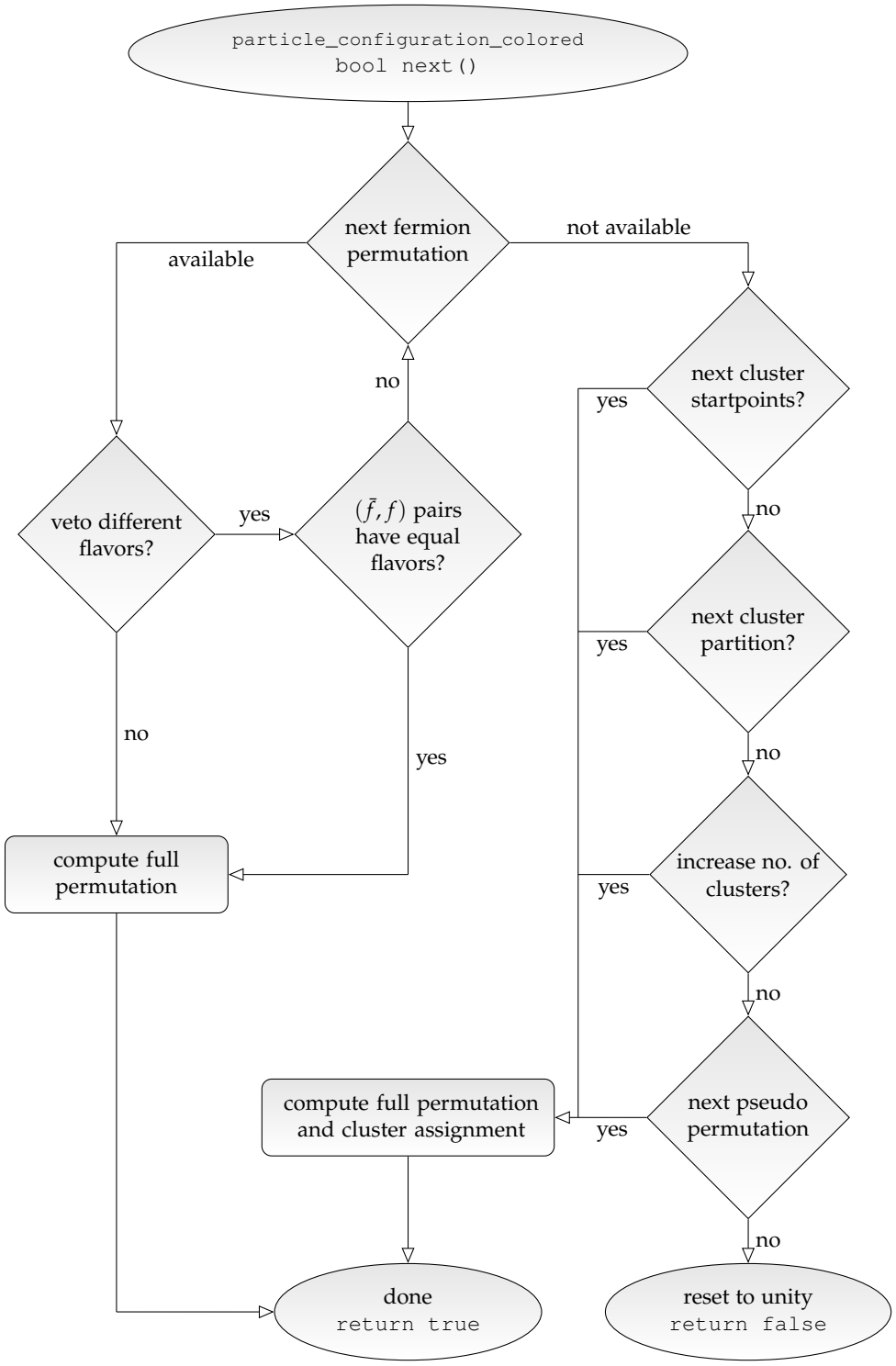


Figure 6.2: Flow chart for the `next` function of the class `particle_configuration_colored`, which advances to the next particle configuration for a theory with fermions and color.

tiplied out, this sum contains 2^{n_g} terms,

$$P = P_1 + P_2 + \cdots + P_{2^{n_g}},$$

so that a full color matrix element can be rewritten as

$$M_{ab} = \bar{c}_a P c_b = \bar{c}_a P_1 c_b + \bar{c}_a P_2 c_b + \cdots + \bar{c}_a P_{2^{n_g}} c_b. \tag{6.2}$$

The leading order algorithm was not designed to handle sums of terms.

2) Since the color structure of the $U(1)$ gluon comes with a factor of $-1/N_c$, the algorithm has to take those factors into account. The different terms described in item 1) have varying powers of the $U(1)$ color factor, so one does not only deal with an overall constant which can be applied after all contractions have been performed, but each sub-term is influenced individually by the $-1/N_c$ term.

3) Without any algebraic simplification, the four different types of color indices (i, j, $\bar{i}$ and $\bar{j}$) do *not* disappear as they do in leading order, where they reduced to one type of color index due to the contractions in advance.

The problem which is easiest to overcome is number 3): The original idea to represent Kronecker deltas with a color index, see equation (4.5), can be extended to take more than just one type of color indices in a straight forward way which can be explained best in terms of an assignment table similar to equation (4.5):

$$\begin{array}{r||ccc|ccc|ccc|ccc} \text{Bit} & 0 & \dots & n-1 & n & \dots & 2n-1 & 2n & \dots & 3n-1 & 3n & \dots & 4n-1 \\ \text{Index} & i_0 & \dots & i_{n-1} & j_0 & \dots & j_{n-1} & \bar{i}_0 & \dots & \bar{i}_{n-1} & \bar{j}_0 & \dots & \bar{j}_{n-1} \end{array} . \tag{6.3}$$

With this representation, the contraction algorithm can be adopted from the leading color matrix without any changes. However, the number of bits required to store an arbitrary Kronecker delta might pose a problem: As an example, a process with eight particles would require using a data type with 64 bits = 8 byte. To guarantee save operation on any computer, it is therefore reasonable not to trust in built-in data types, but to evade to the STL structure `bitset`. This container is specialized for storing an arbitrary number of single bits and provides the desired functionality on any machine, regardless of processor architechture, etc.

Problem number 2) is solved by exploiting the fact that each building block of the color matrix computation can be written in terms of a basic structure of the form

$$\pm (N_c)^X \prod \delta^{\bullet}_{\bullet}.$$

The above notation symbolizes

a) a sign,

b) some exponent of the color factor N_c and

c) a product (i.e. a list) of an arbitrary number of Kronecker deltas, with arbitrary types of color indices (indicated by bullets).

This basic structure has been implemented into a class called `color_structure`. It provides operators for multiplying such structures (i.e. multiplying signs, adding exponents and adding Kronecker deltas to the list), as well as performing a contraction of all deltas in the list. The contraction method automatically corrects the exponent of N_c such that after successful contraction, the exponent (along with the sign) carries the full information on the structure and no more counting of elements in the list of deltas has to be performed in order to obtain the contraction result.

The class `color_structure` has one more feature. For each color vector element c_a, an adjoint element which carries color indices with bars has to be computed as well. In the present representation of Kronecker deltas as `bitset`s, this is accomplished by shifting bits about $2n$ digits. The class provides a method called `conjugate`, which promotes all indices without bars to indices with bars.[3]

Usage of the `color_structure` class also provides for a simple solution of problem 1). It is possible to store all elements that contribute to the color matrix according to equation (6.2) in separate `color_structure` objects. The color factors $\bar{c}_a$ and c_b can be stored in a STL `vector` where each entry corresponds to one particle configuration. It is important to note that the projector term is independent of the particle configurations, however, it consists of 2^{n_g} sub-terms as described in item 1). The problem with the sub-terms can be solved by creating a vector of 2^{n_g} `color_structure` objects where each entry corresponds to one sub-term:

$$P = (P_1, P_2, \ldots, P_{2^{n_g}}).$$

The summation of the terms as shown in equation (6.2) then occurs as follows: Multiply the two color factors $\bar{c}_a$ and c_b once with each projector element P_i and fully contract the delta lists. The full matrix element is then given by the sum of the results from all contractions.

It remains to derive a method to fill this vector such that all possible contributions are present. For this, one needs an assignment from vector indices to contributing gluon projector terms. This is done similarly to the method described for the momentum sums (section 3.5): Rewrite the vector index into binary representation. Then each gluon gets assigned to one bit of the index; if a bit is zero it means the $U(N)$ term of the gluon projector contributes, if set to one the $U(1)$ term contributes. This ensures that all possible projector terms are computed. An example is shown in figure 6.3.

Now that we know how to compute the single color matrix entries, it is no problem to compute the full matrix. For the same reason that the leading color matrix is symmetric, the full color matrix is also symmetric: The projector terms are symmetric under the exchange $(i, j) \leftrightarrow (\bar{i}, \bar{j})$. Hence, it suffices to compute only one triangle of the matrix. For the leading color matrix, we could predict the matrix elements on the main diagonal analytically; in case of the full color matrix, this is not possible: The different cluster configurations which belong to the different partial amplitudes result in different matrix elements because of the more complicated structure of the color factors, see

[3] One has to be careful, however, that the delta list only contains Kronecker deltas where the indices have no bars. For performance reasons, `conjugate` only shifts all bits but does not check beforehand whether this shift is possible. Conjugating deltas with "barred" indices results in a crash of the program.

`vector` index			
decimal	binary		P_{Index}
	g_2	g_1	
0	0	0	$\left(\delta^{i_2\bar{i}_2}\,\delta_{j_2\bar{j}_2}\right)\left(\delta^{i_1\bar{i}_1}\,\delta_{j_1\bar{j}_1}\right)$
1	0	1	$\left(\delta^{i_2\bar{i}_2}\,\delta_{j_2\bar{j}_2}\right)\left(-\frac{1}{N_c}\delta^{\bar{i}_1}_{\bar{j}_1}\,\delta^{i_1}_{j_1}\right)$
2	1	0	$\left(-\frac{1}{N_c}\delta^{\bar{i}_2}_{\bar{j}_2}\,\delta^{i_2}_{j_2}\right)\left(\delta^{i_1\bar{i}_1}\,\delta_{j_1\bar{j}_1}\right)$
3	1	1	$\left(-\frac{1}{N_c}\delta^{\bar{i}_2}_{\bar{j}_2}\,\delta^{i_2}_{j_2}\right)\left(-\frac{1}{N_c}\delta^{\bar{i}_1}_{\bar{j}_1}\,\delta^{i_1}_{j_1}\right)$

Figure 6.3: Example for the representation of the projector terms as a STL `vector`. This example illustrates what happens for two gluons. There are $2^{n_g} = 2^2 = 4$ terms, their indices within the vector are given in the first column. The next two columns show the binary representation of the index (read from right to left) and the last column shows the corresponding projector term. Quarks can be added as desired; the corresponding Kronecker deltas can be simply multiplied to the projectors.

equation (2.13) on page 16. On the whole, we have to compute $\frac{|P|^2+|P|}{2}$ elements of the matrix, if $|P|$ denotes the number of particle configurations.

6.3 The Partial Amplitudes

If we take the recursion formulas from QED and change the particle types from photons to gluons and from leptons to quarks we already have the foundation for QCD recursion formulas. However, there are two complications arising from the color structure:

Since we distinguish between virtual gluons and $U(1)$ gluons, we need an additional current for the colorless gluon. In chapter 2, we argued that the kinematical part of the two gluon propagators is identical. However, the $U(1)$ gluon neither couples to the $U(3)$ gluon nor to itself. This has to be reflected in the corresponding current, which is why the $U(1)$ current differs from the gluon current insofar as it only contains the coupling to the quark gluon vertex. There is also no need for including a condition for the recursion start, i.e. a $U(1)$ polarization, since the $U(1)$ gluon only appears as a virtual particle.

Furthermore, the appearance of color clusters requires new conditions for the splitting of currents into smaller subcurrents. If we take another look at figure 6.1, it becomes obvious that not only the external particles can be assigned to a color cluster, but also the internal lines: The $U(1)$ lines divide the diagram into separate "trees" where each tree corresponds to one color cluster — this of course includes the internal lines and thereby all off-shell sub-currents (except for the colorless $U(1)$ current). When splitting a current into subcurrents, some splits are rejected due to non-matching color clusters. What "non-matching" means cannot be expressed easily in terms of formulas for the currents (as we did for QED). Determining whether a splitting is possible requires that we know the color cluster for each current. Since the color clusters of the external particles are known, the clusters of the currents can be assigned easily.

The following discussion of the splittings is based on [14]. All conditions are shown diagramatically in figure 6.4.

Gluon current: For the gluon current, we have to distinguish between the purely gluonic vertices and the quark gluon vertex. In the case of the gluonic vertices, we must ensure that each of the subcurrents contains at least one particle of the same cluster as the gluon current itself. Otherwise, we would connect a color-disconnected cluster to the remaining diagram by a gluonic vertex, which is impossible since clusters can only be connected via a $U(1)$ gluon.

In the case of the quark gluon vertex, both subcurrents have to belong to the same color cluster as the gluon current, otherwise we would connect different clusters via a $U(3)$ gluon. We can identify the color clusters of the subcurrents easily by looking at the two particles between which the split occurs (see figure 6.4(a)).

(Anti-)quark current: Each (anti-)quark current splits into one (anti-)quark and one gluonic subcurrent. The gluonic subcurrent can either be a $U(3)$ current or a $U(1)$ current. For the $U(3)$ subcurrent, at least one particle in this subcurrent has to belong to the same cluster as the off-shell current itself.

In case of a $U(1)$ subcurrent,

(i) all particles of the subcurrent must belong to different clusters than the cluster of both (anti-)quark currents,

(ii) the particles which appear in the $U(1)$ subcurrent must form complete color clusters.[4]

These conditions ensure that the $U(1)$ gluon only couples separate cluster trees to another; the second condition deals specially with embedded clusters. As an example, take the process $\bar{q}_1\, q_2\, \bar{q}_3\, q_4\, \bar{q}_5\, q_6\, \bar{q}_7\, q_8$ (the indices merely number the particles) with the corresponding color clusters (11223322). Suppose we wanted to compute the quark current $Q(q_2 \,|\, \bar{q}_3\, q_4)$, where the vertical line indicates the split point; q_2 is the only particle in the quark subcurrent while $\bar{q}_3$ and q_4 form the $U(1)$ subcurrent. With the given color cluster assignment, condition (i) is fulfilled. However, condition (ii) requires all particles of cluster 2 to belong to the $U(1)$ current; this is not the case since particles 7 and 8 also belong to cluster 2. Hence, this split is not possible with a $U(1)$ current. The only valid $U(1)$ subcurrents for this process are $(\bar{q}_3\, q_4\, \bar{q}_5\, q_6\, \bar{q}_7\, q_8)$ and $(\bar{q}_5\, q_6)$.

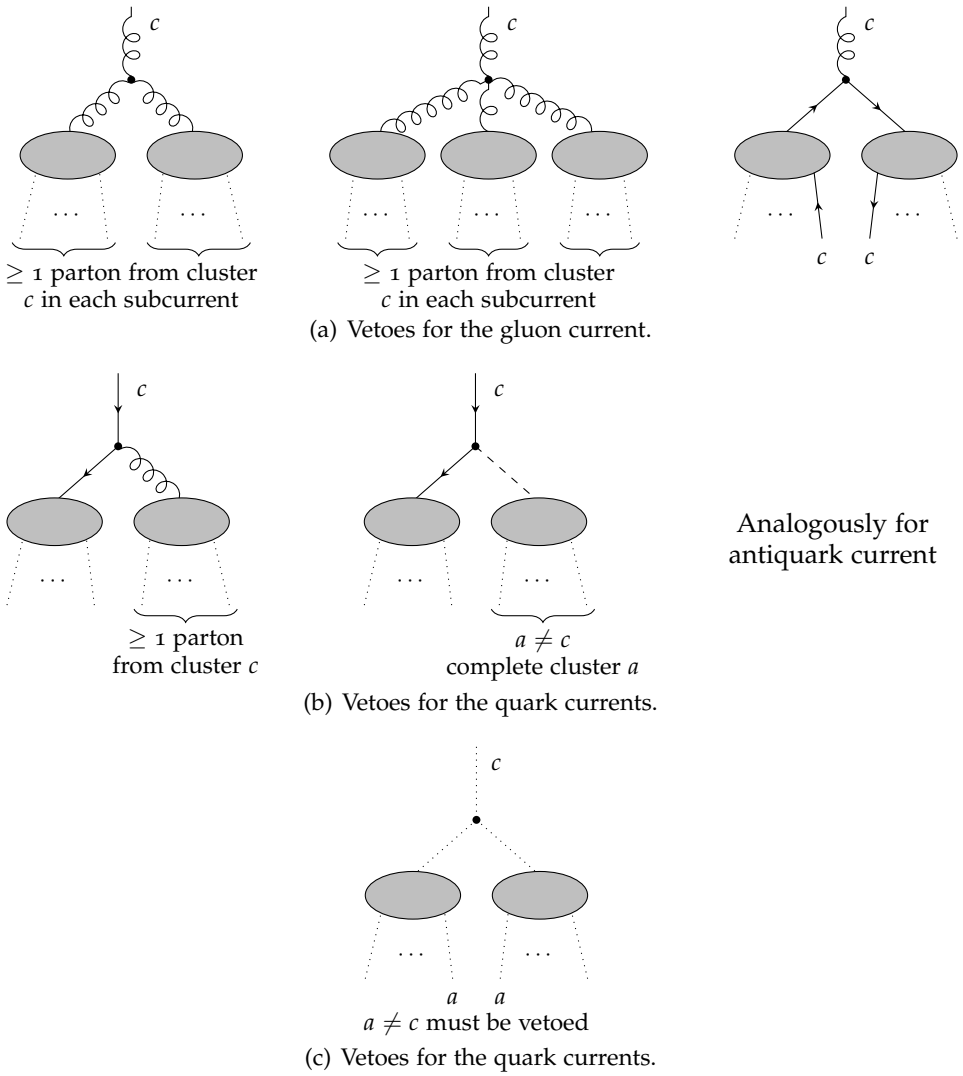
$U(1)$ current: No extra conditions have to be applied.

Generally, we have to add the rule that a current belonging to cluster a cannot split into two subcurrents belonging to another current b.

The implementation of these conditions can be realized with three functions:

- `has_cluster` checks whether the given subcurrent contains a particle of the given color cluster.

[4] This condition is an addition to those discussed in [14]; it is only neccessary for amplitudes with eight and more particles.



(a) Vetoes for the gluon current.

(b) Vetoes for the quark currents.

(c) Vetoes for the quark currents.

Figure 6.4: Graphical representation of the splitting conditions arising from the color clusters. Dotted lines indicate either gluons, quarks or antiquarks. Braces indicate that the given condition applies to all particles from the corresponding subcurrent. All indices c and a denote the cluster of the respective leg.

- `possible_split` allows a split if the last particle of the left subcurrent and the first particle of the right subcurrent belong to different clusters or if the first particle of the right subcurrent belongs the same cluster as the parent current.
- `contains_complete_cluster` checks whether a given sub-cluster contains all particles of one cluster; this is used for determining (anti-)quark current splittings into a $U(1)$ subcurrent (among others, this implements the general condition, figure 6.4(c)).

As stated before, all other ingredients for the implementation are the same as for QED. Naturally, we also employ Dirac spinors and precompute the particle polarizations as an optimization. This concludes the present algorithm for QCD amplitudes.

Chapter 7

QCD Amplitudes With Photons — Merging Two Gauge Groups

After QCD amplitudes with full color treatment have been established in the previous chapter, we will now direct our interest to adding external photons. Photons are, in a way, the particles which are most simple to deal with since they only couple to fermion pairs. However, we will first concern ourselves with the more general problem of combining two gauge groups to form one single amplitude.

The combination of multiple gauge groups has already been described by Mangano and Parke [19] for the general case of $SU(M) \times SU(N)$, where both gauge groups have their individual color degrees of freedom. Let us denote the corresponding color factors by c_M and c_N, where color indices are omitted. Let us furthermore denote all bosons subject to $SU(M)$ by b_M, all bosons subject to $SU(N)$ by b_N and fermions which couple to *both* gauge groups, by f. For simplicity, let us restrict the number of fermion pairs to one at present. Then an amplitude can be written in the form

$$\mathcal{A}(b_M; b_N; f) = \sum_{\substack{P \in S_M \\ \widetilde{P} \in S_N}} c_M \, c_N \, A_{M,N}(f, b_M; f, b_N).$$

Note that both gauge groups are treated separately, since we sum over the permutations of each gauge group individually. The reason for summing over all permutations (and not just non-cyclic ones) is that we keep the fermion pair out of the permutation, which results in the overall amplitude being summed over all non-cyclic permutations. The above formula also introduces new partial amplitudes $A_{M,N}$, where the argument indicates that we treat both gauge groups separately. These are related to the previously used partial amplitudes by the following relation:

$$A_{M,N}(f, b_M; f, b_N) = \sum_{\mathcal{S}} A(f, b_M, b_N).$$

The sum over $\mathcal{S}$ indicates a *shuffling* of all bosons among each other, which means that the bosons of the different gauge groups get interspersed without changing the order of bosons in each gauge group. This will be described in detail later, when an algorithm for such shuffles is developed.

This is the case which Mangano and Parke discuss in their paper. In the present

case, the situation is slightly different:

- Only one gauge group, $SU(3)$ of QCD, depends on color; the photon and its coupling to quarks is colorless.
- Fermion numbers shall not be restricted to one fermion/antifermion pair.

While the first item does not alter the above considerations much[1], the second point complicates the method a bit: There, the one fermion pair could be separated from any permutations, so that the overall amplitude got summed over all non-cyclic permutations. However, with more than one fermion pair, this is no longer possible, since the additional fermions have to be permuted in a similar way as was discussed for QED and QCD amplitudes.

The solution to this problem is, in fact, rather simple: We count the quarks as part of the QCD fraction of the amplitude and thus leave them out of the photon part. If we then change the permutations such that QCD is only summed over *non-cyclic* pseudo particle permutations and quark permutations, while photons are still summed over *all* permutations, we effectively arrive at the same result as the Mangano/Parke approach. To put this into a formula, let us go back to the pseudo particle approach and denote all QCD pseudo particles, i.e. gluons and quark pairs, by normal arabic numbers $1, 2, \ldots, n_p$ and photons by $\gamma_1, \gamma_2, \ldots, \gamma_{n_\gamma}$. Let c denote the QCD color factor with omitted color indices. Of course, the QCD part also has to be summed over all cluster configurations, as described in section 6.1, which affects the color factor. If we put all the ingredients together, we arrive at

$$\begin{aligned}\mathcal{A}(1,2,\ldots,n_p;\gamma_1,\gamma_2,\ldots,\gamma_{n_\gamma}) \\ = \underbrace{\sum_{S_{n_p-1}} \sum_{S_{n_f}} \sum_{\text{cluster}}}_{\text{QCD}} \underbrace{\sum_{S_{n_\gamma}}}_{\text{photons}} c\, A_{\text{QCD}\times\gamma}(1,2,\ldots,n_p;\gamma_1,\gamma_2,\ldots,\gamma_{n_\gamma}),\end{aligned} \tag{7.1a}$$

where

$$A_{\text{QCD}\times\gamma}(1,2,\ldots,n_p;\gamma_1,\gamma_2,\ldots,\gamma_{n_\gamma}) = \sum_{\mathcal{S}} A(1,2,\ldots,n_p,\gamma_1,\gamma_2,\ldots,\gamma_{n_\gamma}). \tag{7.1b}$$

Note that the partial amplitudes are written in terms of QCD pseudo particles, but depend on the real particles, in a similar fashion as the QCD amplitudes. However, the shuffling only occurs for pseudo particles, not for real particles.

7.1 A Mixed Particle Configuration

When one wants to implement equations (7.1a) and (7.1b) in the class `particle_configuration_multi`, one first has to think about a reasonable way to organize mutiple gauge groups. Each partial amplitude $A_{\text{QCD}\times\gamma}$ from equation (7.1a) is a gauge invariant quantity which belongs to one color factor. This fact makes it convenient *not* to count the shuffled amplitudes from equation (7.1b) as separate particle configurations, but to include the sum over all such shuffle amplitudes in each partial

[1] It saves some computation time, since we only have to compute color factors for one theory, but the way partial amplitudes are computed remains the same.

amplitude. To achieve this, we add another function to the particle configuration class, `next_shuffle`, which deals with the shuffles separately from the remaining particle configuration.

7.1.1 The next-Function

So, how do we realize the particle configuration without the shuffles? The standard cyclic ordering which is the default configuration for each process, see section 3.3, makes this easy: It separates all particle types from each other in such a way that all QCD particles come first and the photons (and leptons, for future purposes) are last. This enables us to use the existing particle configurations for QCD and QED to obtain the separate configurations and plug the results together. Therefore, the `particle_configuration_multi` class draws back on one colored and one colorless particle configuration and uses their respective `next` methods to obtain the next particle configuration for each gauge group.

It is then straight forward to combine the two permutations into one permutation. For the color clusters, we do the same but we have to assign a special value to the photons to signalize that they are colorless. Both permutations and color clusters are joined in such a way that the QCD particles always come first and photons last; not only is this easy to do, but it is also important for the shuffles, which will be apparent in a minute. Each call to the `next` function of the mixed particle configuration class then performs the following steps:

1) Go to the next photon permutation. If successful, create the full permutation and color cluster information.

2) If the next photon permutation is not available (as before, this means all permutations have already been generated), go to the next QCD particle configuration and reset the photon permutation.

This is visualized in a flow chart, see figure 7.1. An example for joining to separate particle configurations is given in table 7.1.

7.1.2 A Function for Shuffles

Now, were are left with the task of deriving a shuffle algorithm for the function `next_shuffle`. Above, we only vaguely defined what "shuffling" actually means. To clear this up, let us look at an example: In table 7.1, each mixed particle configuration has to be shuffled, hence we will pick the first example (1234abc). "Interspersing the particles", as we defined shuffling earlier, means to exchange the particles in all possible ways which keep the orderings (1234) and (abc) within the separate gauge groups. Of course, this obeys the general rule that cyclic permutations are forbidden, hence particle 1 is taken out of the shuffling. The shuffles are given by

$$(1234abc),\ (123a4bc),\ (123ab4c),\ (123abc4),\ (12a34bc),\ (12a3b4c),\ \ldots,\ (1abc234).$$

The shuffles for all other elements from table 7.1 work the same, however, the particles of each gauge group are permuted as given in the table.

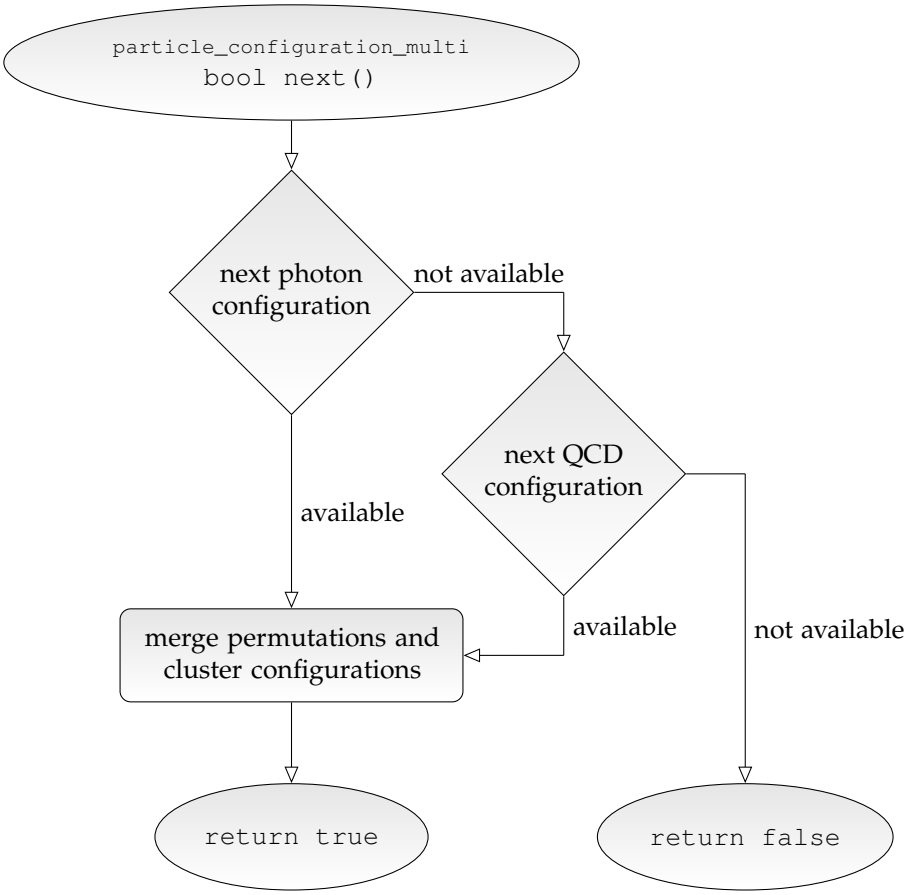


Figure 7.1: Flow chart for the `next` function of the class `particle_configuration_multi`.

QCD \ photons	abc	acb	bac	bca	cab	cba
1234	1234abc	1234acb	1234bac	1234bca	1234cab	1234cba
1243	1243abc	1243acb	1243bac	1243bca	1243cab	1243cba
1324	1324abc	1324acb	1324bac	1324bca	1324cab	1324cba
1342	1342abc	1342acb	1342bac	1342bca	1342cab	1342cba
1423	1423abc	1423acb	1423bac	1423bca	1423cab	1423cba

Table 7.1: Example for merging two particle configurations into one; For readability, the QCD part is denoted in terms of arabic numbers, the photon part in terms of lower case letters. For simplicity, the QCD part is represented only in terms of simple non-cyclic permutations; for an arbitrary process, one has to add quark permutations and cluster configurations. The first line shows the photon permutations; the first column shows the QCD configurations. All remaining entries show how `particle_configuration_multi` combines the two. The order in which the combined configurations appear is given by the usual manner of reading a text, i.e. start from the upper left element and go to the right until all photon permutations have been performed; then do the same for the next line, etc.

To implement this, we need a way to characterize these shuffles mathematically: The shuffles are given by all non-cyclic permutations of the particles indices, with the additional condition that we have to take particles from the same gauge group as indistinguishable. For the above example, this means that (1243abc) is *not* a shuffle permutation since particles 3 and 4 are transposed, which is a permutation of indistinguishable elements. To make this more visible, we can assign a number to each gauge group and replace the particle indices by the number of their respective gauge group. If we assign 1 to QCD and 2 to photons, we obtain the following examples:

$$\begin{aligned}(1234\text{abc}) &\longrightarrow [1111222]\\ (1243\text{abc}) &\longrightarrow [1111222]\\ (123\text{a}4\text{bc}) &\longrightarrow [1112122].\end{aligned}$$

It is clear that the second example is *not* a shuffle permutation of the first line (in contrast, it belongs to another particle configuration!). However, the third line represents a shuffle permutation of the first line.

This method of replacing the particle indices by the gauge group numbers can actually be used to realize an algorithm. The earlier used function `next_permutation` from the STL treats equal entries as indistinguishable, i.e. if we permute [1111222] with `next_permutation`, we automatically obtain all shuffle permutations. Hence, it is not difficult to formulate an algorithm for advancing to the next shuffle permutation:

1) Convert the particle indices to gauge group indices as explained above.

2) Apply `next_permutation` to obtain the next shuffle configuration.

3) Re-convert the gauge group indices to particle indices. This is easy to do since the separate particles configurations for QCD and photons are known: If the gauge group index is 1 insert the next particle from the QCD configuration; if the gauge group index is 2 insert the next photon from the photon configuration.

The color clusters can be treated in the same way, along with the particle indices.

7.2 A Shortcut for the Color Matrix

Now that the particle configuration is available, we have to deal with the color matrix. To compute the color matrix, we could in principle use the same algorithm as for the QCD color matrix; the algorithm only needs to know that it should ignore any photons that appear.

However, there is a more efficient method: If we look at the flow chart 7.1, it becomes obvious that for each QCD configuration, the algorithm runs through each photon configuration. Since the photon configurations do not affect the color factors, these color factors are the same. Hence, $n_\gamma!$ consecutive partial amplitudes carry the same color factor. This fact carries over to the color matrix in such a way that whole blocks of size $n_\gamma! \times n_\gamma!$ contain only one entry. We can use this fact to compute the color matrix in a more efficient way:

1) Compute the QCD matrix using the `compute_full_color_matrix` function.

2) Blow this matrix up by expanding each element into a block of $n_\gamma! \times n_\gamma!$ elements.

The result is the same we would get if we computed the matrix naively, but we avoid computing the same elements over and over again, which can save a lot of computation time if many photons are present.

As an example, take a process where the QCD matrix is a symmetric 3×3 matrix. Let us now assume that there are two photons in the process, amounting to $n_\gamma! = 2$ photon configurations. For this example, the method can be visualized as follows:

$$\underbrace{\begin{pmatrix} a & b & c \\ b & d & e \\ c & e & f \end{pmatrix}}_{\text{QCD}} \longrightarrow \underbrace{\begin{pmatrix} a & a & b & b & c & c \\ a & a & b & b & c & c \\ b & b & d & d & e & e \\ b & b & d & d & e & e \\ c & c & e & e & f & f \\ c & c & e & e & f & f \end{pmatrix}}_{\text{mixed}}$$

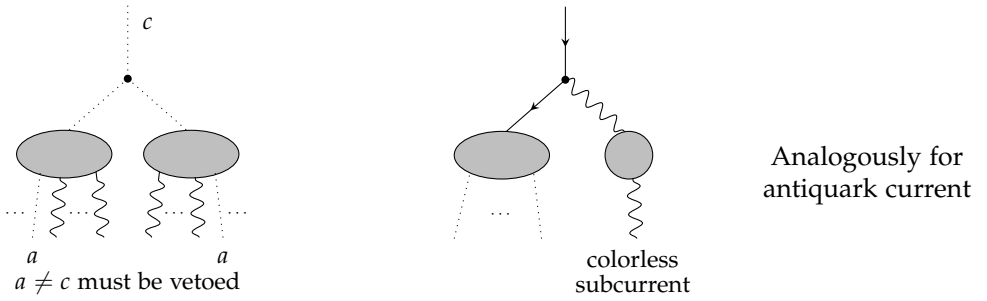
The letters denote the corresponding entries.

7.3 Partial Amplitudes

The implementation of the partial amplitudes is very similar to the QCD amplitudes: All extra conditions which arise from the color structure (see figure 6.4 on page 70) are still valid, with one exception which affects the general splitting rule. There, we said that the two particles between which the split occurs may not belong to a different cluster than the parent current (confer figure 6.4(c)). With photons, it is generally possible that these two particles are photons and thus colorless. In this case, the condition only works with a slight modification: If either of the particles around the splitting point is a photon, move one particle further away from the splitting point. This step is repeated until a colored particle, i.e. a gluon or quark, is found. This condition is represented in figure 7.2(a). If no colored particle is found (i.e. the subcurrent consists only of photons), the split is also valid in the case that the subcurrent contains only one particle, a photon. This case applies to quark and antiquark currents, which can emit a single external photon (see figure 7.2(b)).

This also leads to the next modification: We need a photon current which deals with external photons. This current can only be called from the quark and antiquark currents, since gluons do not couple to the photon. The question is, how do we find out whether the (anti-)quark current should split into a gluon or a photon current? We already know it! For QCD amplitudes, we had to distinguish between the $U(3)$ gluon and the colorless $U(1)$ gluon. But as we stated earlier, the $U(1)$ gluon is identical to a photon except that the $U(1)$ gluon couples with the strong coupling constant. If we exploit this fact, we can gather the $U(1)$ gluon and the photon within one current, i.e. we can incorporate the photon current into the $U(1)$ current. So, the only change is to add the possibility for a photon polarization in the $U(1)$ current; recall from section 6.3 that we left this out deliberately in QCD since there was no external $U(1)$ gluon.

When adding the polarization we have to pay attention to two more things which we merely hinted at above: The photon couples with the electromagnetic coupling



(a) General condition.

(b) Valid configurations for colorless subcurrents.

Figure 7.2: Graphical representation of the general splitting condition. The wavy lines denote photons, the dotted lines denote gluons and quarks.

constant, plus it is affected by the electric charge of the quark to which it couples: For up-type quarks (i.e. u, c, t) the coupling has to be multiplied by $2/3$, for down-type quarks (i.e. d, s, b) by $-1/3$. Hence, we have to know the flavor of the (anti-)quark to which the external photon is coupled. We have to obtain this beforehand, more precisely from the (anti-)quark current which emits the photon: This current splits into the photon and an (anti-)quark subcurrent. Thus, the flavor can be obtained from the (anti-)quark subcurrent. A function `get_flavor_charge` takes care of this in the program.

With these few modifications, QCD can be equipped with external photons.

Chapter 8

Tests and Performance

All previous chapters explained the ideas, algorithms and the functionality of the program. But we have yet to prove that the discussed methods actually work and deliver reasonable results. By reasonable, we refer to two aspects: The performance side and the correctness of the methods. The following two sections provide explicit results for both.

8.1 Performance

Many of the ideas presented in this thesis are concerned with a *fast* computation of squared amplitudes. This section gathers these performance-enhancing methods and investigates their use in terms of computation time.

All tests were performed on an *Intel Core i5* machine with four CPU cores at 2.67 GHz each and 4 GB RAM. Naturally, the program is inteded for production use, so all tests were compiled with GCC's `-O2` optimization level and with `#define NDEBUG`, which omits a lot of debugging assertions and accellerates the program greatly. This also influences the *BOOST* routines, most importantly those concerned with the vector-matrix-vector multiplication of the partial amplitudes and the color matrix. In addition, the special switches `#define BOOST_NDEBUG` and `#define BOOST_UBLAS_NDEBUG` were set, which avoid additional debugging assertions for *BOOST* and the uBLAS library.

For time measurements, wall clock time was measured using the the `timeval` structure and the function `gettimeofday` from the UNIX system header `sys/time.h`. This structure provides two variables for time values, one which counts seconds and one for microseconds, allowing both very short and precise measurements as well as long measurements which might take several hours. CPU time is not measured since it is not valid for the end-user. However, all time values given are average values of many runs, which was done in order to minimize possible distortions by background processes. More details on the particular measurements are given in the following sections.

The following program versions were used for the tests:

- GCC 4.5.2
- *BOOST* 1.42.0

GCC automatically contains GOMP, the GCC OpenMp support library; Starting with version 4.4, it implements version 3.0 of the OpenMP standard.[1]

8.1.1 Precomputation of Momentum Sums

In section 3.5, we suggested precomputing all possible sums of momenta which are needed for internal propagators, so that we can draw back on them during the computation of partial amplitudes.

As stated before, all amplitude classes are equipped with this precomputation. In order to test the improvement in performance, both the purely recursive and the iterative versions of the gluon amplitude class have additionally been implemented *without* precomputation of momentum sums. Results for the comparison of precomputed versus "on-the-fly" computed momentum sums are shown in table 8.1.

The table shows that pure recursion only yields minor performance gains (less than 2 %), while the iterative approach gains a little more. The speed-up is lower than we expected, hence we performed the same test with different versions of GCC and we also performed the test on two other machines (both *Intel Core 2 Duo* machines, one at 2.4 GHz with 2 GB RAM the other at 2.6 GHz with 4 GB RAM). This showed surprising results: Some configurations (especially on the 2.4 GHz machine) yield performance boosts of 5 to 6 %, others show no speed-up at all when using precomputed momentum sums.

At the moment, we can only guess at the reasons: When computing momentum sums on-the-fly, this happens in the processor cache which allows for very fast access. When precomputing momentum sums, on the other hand, they get stored in a `vector` by dynamical memory allocation. The exact location is not known in advance and might provide much slower access than processor cache. Assuming that this is the case, it is clear why different processor architectures yield different results.

Moreover, the compiler may optimize the on-the-fly computation of momentum sums in ways which it cannot do for the precomputed momentum sums. Since the different compiler versions of GCC also differ in their methods of optimization, this might explain why different compiler versions yield different results.

But, as stated, this is only a guess at the reasons. At the current state, we have to assess that the gain of performance due to a precomputation of momentum sums depends on the machine and the used compiler version.

8.1.2 Color Matrices

Section 4.2 presented a method to compute the gluon color matrix in an efficient way. In contrast to the algorithm which performs the full color computation, this algorithm reduces the number of Kronecker deltas greatly by analytical optimization beforehand.

One might wonder whether this optimization is really neccessary as an additional option to the full color matrix, since the color matrix is not time-critical in the sense that it has to be computed only once at the start of each computation. As it turns out, the optimized algorithm can save computation time of almost two hours[2] in the case of an eight gluon amplitude.

[1] see http://gcc.gnu.org/wiki/openmp

[2] This applies to the tested machine and differs on slower or faster processors, of course.

(a) Recursive algorithm

g	Precomputed		On-the-fly		Ratio %
4	56.6	µs	57.2	µs	99.0
5	899	µs	910	µs	98.8
6	18.1	ms	18.3	ms	98.9
7	436	ms	441	ms	98.9
8	1.24	s	1.25	s	99.2

(b) Iterative algorithm

g	Precomputed		On-the-fly		Ratio %
4	15.7	µs	16.1	µs	97.5
5	117	µs	121	µs	96.7
6	1.08	ms	1.12	ms	96.7
7	12.3	ms	12.6	ms	97.6
8	301	ms	305	ms	98.7

Table 8.1: Computation time in seconds for squared amplitudes, both without helicity summation and color matrix; *Precomputed* means that the momentum sums are calculated once and stored as explained in section 3.5, *On-the-fly* means the sums are computed on demand and are never stored. The given times are average values of 1000 runs in the case of 8 gluons up to 10 000 000 runs in the case of 4 gluons. *Ratio* equals the *Precomputed* time in relation to the *On-the-fly* time.

Process	Computation time Leading color		Full color		Ratio
$4g$	8.0	µs	330	µs	2.4 %
$5g$	0.13	ms	11	ms	1.2 %
$6g$	3.9	ms	640	ms	0.61 %
$7g$	0.16	s	55	s	0.29 %
$8g$	9.1	s	106	min	0.14 %

Table 8.2: Comparison of duration for computing the color matrix with leading order optimization and the slower full color algorithm. Ratio shows the leading color time in relation to the full color time.

Table 8.2 shows a comparison of computation times of the two algorithms for amplitudes with four to eight gluons. These measurements were performed such that each color matrix was computed many times (for roughly half an hour each) and this total time was averaged over the number of computations. In case of the full color matrix for eight gluons, the matrix was computed only once without any averaging because of the long duration.

From the table, it is obvious that the optimized algorithm is vastly superior to the full color algorithm. The four gluon amplitude, where only $(n^2+n)/2 = (16+4)/2 = 10$ matrix elements have to be computed, already shows a speed-up of almost 98 %. We can read off the table that with each additional gluon, the ratio between leading color and full color computation roughly halves. This can be attributed to the fact that each gluon brings along two terms in the projector in the case of the full color matrix, effectively doubling the amount of terms to compute. In the leading order case, the gluon projector enters the game as a single term — and can actually be omitted, as explained in section 4.2, thus there is no such doubling.

8.1.3 Vector-Matrix-Vector Multiplication

The vector-matrix-vector multiplication is of greater importance than the color matrix, in the sense that it has to be performed for each squared amplitude. Section 3.6 stated that the internal methods provided by the *BOOST* uBLAS library are used since they are fast. To verify that *BOOST* indeed provides fast routines, three different methods were tested.

Naive method: We will use this name to refer to a method which directly arises out of the mathematical formula written in indices:

$$\mathcal{A} = \vec{A}^\dagger M \vec{A} = \sum_{i,j} A_i^* M_{ij} A_j.$$

Note that this method does not make it obvious that $\mathcal{A}$ is a real number.

Optimized method: If one uses the fact that the color matrix is symmetric, $M_{ij} = M_{ji}$ and real-valued, $M_{ij} = M_{ij}^*$, one can refine the naive method by relating the two triangles of the matrix to each other. For this, one first separates the sum over the main diagonal elements (where $i = j$) from the triangles, which can then be reformulated:

$$\begin{aligned}
\mathcal{A} &= \underbrace{\sum_{i=0}^{n-1} A_i^* M_{ii} A_i}_{\text{main diagonal, } \equiv D} + \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} A_i^* M_{ij} A_j + \underbrace{\sum_{j=0}^{n-2} \sum_{i=j+1}^{n-1} A_i^* M_{ij} A_j}_{\text{rename indices } i \leftrightarrow j} \\
&= D + \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} \left(A_i^* M_{ij} A_j + A_j^* \underbrace{M_{ji}}_{=M_{ij}} A_i \right) \\
&= D + \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} \left(A_i^* M_{ij} A_j + (A_i^* M_{ij} A_j)^* \right) \\
&= D + \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 2 M_{ij} \, \mathfrak{Re}\left(A_i^* A_j \right).
\end{aligned}$$

This optimized version looks more complicated, however, it reduces the computation effort since there are only n elements to be computed on the main diagonal plus $(n^2 - n)/2$ elements for the off-diagonal elements, resulting in $(n^2 + n)/2$ elements in total. One might think of another speed up caused by the fact that each term of the sum is a real number instead of a complex number. However, the computation of each term in the sum involves a temporary complex variable $A_i^* A_j$ so that this is no real advantage.

***BOOST* method:** This version uses the internal realizations of the uBLAS library for computing the product in two steps, indicated by braces:

$$\mathcal{A} = \vec{A}^\dagger \left(M \vec{A} \right).$$

These three methods were tested with gluon amplitudes (iterative version), the results are shown in table 8.3. Times were obtained by performing many Monte Carlo

Process	No. of partial amplitudes	Time in ms naive	optimized	*BOOST*
$4g$	6	0.017	0.020	0.016
$5g$	24	0.12	0.15	0.12
$6g$	120	1.2	1.4	1.1
$7g$	720	17	14	12
$8g$	5040	606	266	302

Table 8.3: Comparison of different version for vector-matrix-vector multiplication of partial amplitudes and color matrix. The times shown are for the computation of a squared amplitude with one helicity configuration, averaged from the total duration of many Monte Carlo iterations.

iterations[3] and averaging the measured times over the number of iterations. Note that the measured time does not include the computation of the color matrix, but only the time required to perform the computation of the partial amplitudes and the vector-matrix-vector multiplication for squaring the amplitude. The single-threaded Monte Carlo version was used in order to avoid possible distortion created by synchronization issues among several threads.

The table shows that generally, the internal *BOOST* methods are fastest. The optimized version shows its advantages only at very high numbers of partial amplitudes; however, most processes, especially those including many quarks and photons, consist of a very moderate amount of partial amplitudes. The naive version is surprisingly fast for smaller numbers of partial amplitudes, but it is still generally the slowest and should not be used.

Since the *BOOST* method is generally fast, we use this method for all remaining tests, however, the user can choose the version by changing a parameter in the `computation_information` structure (see section 3.2).

8.1.4 Optimizations for Gluon Amplitudes

In section 4.4, we presented two possibilities for optimizing the pure Berends-Giele recursion approach to gluon amplitudes: An optimized recursion, which saves currents on-the-fly and draws back on them when they are needed again, and an iterative approach which avoids the recursion by building all currents starting from the polarizations. These three methods were tested for gluon amplitudes with four to eight gluons, the results are shown in table 8.4. Again, the given times are average values of many Monte Carlo iterations excluding the duration of the color matrix (each amplitude was computed for roughly five minutes; the given result is the average).

The table shows that both optimization methods are clearly superior to the pure recursion, as we expected. There is also a difference between the two optimizations, which is due to the fact that the optimized recursion still has to build the full recursion tree several times until most of the currents are computed while the iterative approach never builds a tree but always draws back on already computed currents.

[3]The number of iterations varied depending on the number of gluons; it ranged from 100 iterations in the case of eight gluons to 100 000 iterations in the case of four gluons.

Process	Full recursion	Optimized recursion		Iteration	
	Time	Time	Ratio	Time	Ratio
$4g$	57 µs	17 µs	30 %	16 µs	28 %
$5g$	900 µs	128 µs	14 %	118 µs	13 %
$6g$	18.1 ms	1.2 ms	6.6%	1.1 ms	6.1%
$7g$	437 ms	13 ms	3.0%	12 ms	2.7%
$8g$	12.44 s	0.31 s	2.5%	0.30 s	2.4%

Table 8.4: Performance of the different optimization techniques for gluon amplitudes. *Full recursion* is the non-optimized Berends-Giele recursion; *optimized recursion* stores computed currents and draws back on them; *iteration* inverts the recursion and builds the current tree starting from the polarizations. The given ratios give the relation between the respective time and the full recursion time.

Process	Time and ratios							
	1 thread	2 threads		3 threads		4 threads		
$4g$	173 ms	80 ms	46%	62 ms	36%	47 ms	27%	
$5g$	1170 ms	587 ms	50%	446 ms	38%	334 ms	29%	
$6g$	11.3 s	5.4 s	48%	4.1 s	36%	3.1 ms	27%	
$7g$	123 s	62 s	50%	47 s	38%	35 s	28%	
$8g$	3009 s	1533 s	51%	1134 s	38%	863 s	29%	

Table 8.5: Performance of the multi-threaded Monte Carlo routine with one to four threads. The given percent values give the relation between the respective multi-threaded computation time vs. the single-threaded computation time.

8.1.5 Monte Carlo: Single-Threaded vs. Multi-Threaded

The currently implemented multi-threaded Monte Carlo integration routine performs the computation in n_{thr} parallel threads, where each of the threads computes the color matrix separately. We argued in section 3.6.3 that this should still result in a reduction of computation time by roughly $1/n_{\text{thr}}$ if n_{thr} is not greater than the number of available physical processor cores. A test has been performed to check this behaviour, the results are presented in table 8.5. It shows that the reduction in computation time comes close to the prediction of $1/n_{\text{thr}}$, the deviations are probably due to synchronization issues, as explained in section 3.6.3.

The possible negative effects due to higher memory consumption (see section 3.6.3) do not show up; however, this is not unexpected since the test computer has 4 GB of memory and the color matrices should occupy less than 500 MB of memory in the case of the eight gluon amplitude with four threads.

8.2 Results and Comparison with MadGraph

Each of the different models presented in this thesis have been successfully compared to reference results for various processes from different sources. These sources include analytical results for matrix elements from [46, 47] and numerical results from [14] for gluon and QCD amplitudes. The variety of these reference results is, however, rather

limited in the numbers and types of particles, especially in the case of analytical results. Hence, we present reference results from other sources.

Table 8.6 shows the results for gluon amplitudes.[4] The reference results (along with the momenta) shown in this table stem from a program by S. Weinzierl. The table shows the process, the reference results, the results obtained with the present program and a rough duration for the computation of a single squared amplitude, i.e. a squared amplitude for one helicity configuration (one Monte Carlo step) without the computation of the color matrix. We give two results from the present program: First, the exact result obtained by helicity summation, second the result obtained by Monte Carlo integration (abbreviated by MC). Note that all results were computed using random parameters $(\mathcal{R}, \mathcal{N})$ for the computation of reference momenta (see section 3.4), hence the results also indicate gauge invariance.

All other reference results, i.e. results for the models QED, QCD and QCD with photons, were computed numerically using the well-known program *MadGraph* [8–10].[5] The corresponding data is shown in tables 8.7 to 8.12b with a similar structure as the gluon table except for the fact that these tables contain an additional column denoted *exponent* which contains an overall exponent that has to be multiplied to all given reference results. For each of these models, there exist two tables: One for the massless and one for the massive case. Note that in the massive cases, only $\tau^{\pm}$, the bottom quarks $\bar{b}$, b and the top quarks $\bar{t}$, t are massive since MadGraph only offers mass parameters for these particles. However, the corresponding tables also mix these massive particles with (in this case) massless particles such as up and down quarks or electrons. The reason is that this automatically checks whether the Feynman rules for massive fermions also work when the mass is set to zero.

For all tests, we use the standard settings from *MadGraph* to make the results reproducible in a simple way. This includes the coupling constants

$$\alpha_{\text{QED}} = \frac{1}{132.507}, \qquad \alpha_{\text{QCD}} = 0.118,$$

the masses

$$m_\tau = 1.777\,\text{GeV}, \qquad m_t = 174.3\,\text{GeV}, \qquad m_b = 4.7\,\text{GeV}$$

and the momenta. The momenta are created internally using the *RAMBO* algorithm [48] at a default center of mass energy of 1 TeV. We will not list them here since almost each process with massive particles has its own momentum configuration. For reference, they can be found in the source files `tests/momenta_1tev.h` for massless particles and `tests/momenta_qed_massive.h` and `tests/momenta_qcd_massive` for massive particles in the package which is attached to this thesis.

The processes we list in the tables were chosen such that (almost) all possible processes for four to eight particles appear. This includes distinguishing between massive and massless particles and particles of same / different flavor. Additionally, we present a few results for processes with nine and ten particles. Note that the processes are given in the "usual" way where the initial channel is *not* crossed into the final channel — for

[4] We show the results for the iterative approach; since the recursive and the optimized recursive approach are based on the same currents, they yield the same results.

[5] We use version v0.6.2.1 from the release *MadGraph 5*, see [11]. Since the present results were generated, version v1.1.0 of the program has been released.

the present program, crossing has to be applied.

In all tables, the reference results and the helicity summed results from the present program are given with a precision of 10 digits. For the majority of processes, the results match up to six or more digits. The deviation between reference results and helicity summed results generally grows as the number of particle increases. This is not surprising since each additional particle introduces four new parameters, the momenta, which are floating point values with limited precision. However, the results agree well in all cases.

The Monte Carlo results were obtained by performing 10 000 iterations. They are given along with the error estimate with a precision of two significant digits in the error (higher precision would not yield additional information). If we look at the Monte Carlo results for the total of 256 tested processes from a statistical point of view, we find that

$$\begin{aligned} 180 &\mathrel{\widehat{=}} 70\,\% \text{ of the results are within } 1\sigma \text{ range,} \\ 243 &\mathrel{\widehat{=}} 95\,\% \text{ of the results are within } 2\sigma \text{ range and} \\ 254 &\mathrel{\widehat{=}} 99\,\% \text{ of the results are within } 3\sigma \text{ range} \end{aligned}$$

in relation to the respective helicity summed results. This corresponds very well to the assumption[6] that Monte Carlo results are normally distributed for large numbers of Monte Carlo iterations and shows again that the method of integrating over helicity angles works as desired.

If we look at the computation durations for one squared amplitude we can see that the computation time generally increases with an increasing number of particles — which is clear, since additional particles increase the number of particle configurations and hence the number of currents to compute. We can also see that adding bosons increases computation time moreso than adding fermions. This is also clear since fermion pairs count as one pseudo particle while bosons count as single particles, hence there are more possible permutations with additional bosons than with fermions. In the case of gluons, computation time grows even more since gluons also couple via the three and four gluon vertices in contrast to photons. If we compare the times for processes with massive fermions to processes with massless fermions, it appears that the massive processes take a little longer in general. This is due to the more elaborate computation of the fermion propagators; note that particle polarizations are also more elaborate, but they get precomputed in all cases and do not influence the computation time.

[6] This assumption is based on the central limit theorem.

Process	Reference result	This work Summation	This work MC	Time
$gg \to gg$	440.658 235 4	440.658 235 4	444.1± 4.7	18 µs
$gg \to ggg$	1020.345 413	1020.345 413	1004 ± 12	130 µs
$gg \to gggg$	365.922 122 4	365.922 122 4	367.6± 5.3	1.1 ms
$gg \to ggggg$	193.590 964 3	193.590 964 3	188.9± 3.1	12 ms
$gg \to gggggg$	136.454 770 7	136.454 770 7	135.2± 2.0	300 ms

Table 8.6: Results for gluon amplitudes.

Process	MadGraph	This work Summation	This work MC	Exp.	Time
$e^-e^+ \to \gamma\gamma$	2.480 355 525	2.480 355 525	2.489± 0.023	10^{-2}	13 µs
$e^-e^+ \to e^-e^+$	4.585 837 685	4.585 837 685	4.553± 0.046	10^{-2}	9 µs
$e^-e^+ \to \mu^-\mu^+$	1.042 635 443	1.042 635 443	1.023± 0.010	10^{-2}	7 µs
$e^-e^+ \to \gamma\gamma\gamma$	1.391 897 173	1.391 897 173	1.399± 0.018	10^{-7}	51 µs
$e^-e^+ \to e^-e^+\gamma$	1.555 265 513	1.555 265 514	1.599± 0.021	10^{-6}	25 µs
$e^-e^+ \to \mu^-\mu^+\gamma$	3.331 185 052	3.331 185 052	3.309± 0.043	10^{-7}	8 µs
$e^-e^+ \to \gamma\gamma\gamma\gamma$	7.015 764 906	7.015 765 006	7.24 ± 0.13	10^{-12}	150 µs
$e^-e^+ \to e^-e^+\gamma\gamma$	3.608 040 122	3.608 040 121	3.600± 0.050	10^{-11}	81 µs
$e^-e^+ \to \mu^-\mu^+\gamma\gamma$	9.934 848 105	9.934 848 089	9.948± 0.014	10^{-12}	45 µs
$e^-e^+ \to e^-e^+e^-e^+$	1.536 037 248	1.536 037 248	1.547± 0.021	10^{-12}	61 µs
$e^-e^+ \to e^-e^+\mu^-\mu^+$	1.615 494 261	1.615 494 261	1.620± 0.018	10^{-13}	26 µs
$e^-e^+ \to \tau^-\tau^+\mu^-\mu^+$	1.453 149 049	1.453 149 049	1.443± 0.015	10^{-13}	16 µs
$e^-e^+ \to \gamma\gamma\gamma\gamma\gamma$	3.776 484 882	3.776 484 886	3.770± 0.085	10^{-16}	1.4 ms
$e^-e^+ \to e^-e^+\gamma\gamma\gamma$	2.140 236 907	2.140 236 906	2.181± 0.039	10^{-15}	600 µs
$e^-e^+ \to \mu^-\mu^+\gamma\gamma\gamma$	5.751 953 465	5.751 953 451	5.69 ± 0.11	10^{-16}	320 µs
$e^-e^+ \to e^-e^+e^-e^+\gamma$	7.043 369 596	7.043 369 597	7.11 ± 0.13	10^{-16}	390 µs
$e^-e^+ \to e^-e^+\mu^-\mu^+\gamma$	8.595 280 589	8.595 280 590	8.44 ± 0.13	10^{-17}	150 µs
$e^-e^+ \to \tau^-\tau^+\mu^-\mu^+\gamma$	7.564 372 579	7.564 372 579	7.45 ± 0.11	10^{-17}	88 µs
$e^-e^+ \to \gamma\gamma\gamma\gamma\gamma\gamma$	2.436 483 260	2.436 481 711	2.518± 0.060	10^{-20}	16 ms
$e^-e^+ \to e^-e^+\gamma\gamma\gamma$	7.263 289 314	7.263 289 779	7.52 ± 0.16	10^{-20}	6.2 ms
$e^-e^+ \to \mu^-\mu^+\gamma\gamma\gamma$	2.650 741 328	2.650 741 333	2.633± 0.052	10^{-20}	3.2 ms
$e^-e^+ \to e^-e^+e^-e^+\gamma\gamma$	1.407 837 891	1.407 837 896	1.434± 0.030	10^{-19}	3.5 ms
$e^-e^+ \to e^-e^+\mu^-\mu^+\gamma\gamma$	1.994 643 514	1.994 643 514	2.016± 0.038	10^{-20}	1.2 ms
$e^-e^+ \to \tau^-\tau^+\mu^-\mu^+\gamma\gamma$	1.615 245 092	1.615 245 093	1.611± 0.029	10^{-20}	650 µs
$e^-e^+ \to e^-e^+e^-e^+e^-e^+$	1.287 925 754	1.287 925 754	1.300± 0.022	10^{-21}	2.7 ms
$e^-e^+ \to e^-e^+e^-e^+\mu^-\mu^+$	1.478 832 458	1.478 832 458	1.491± 0.025	10^{-20}	720 µs
$e^-e^+ \to e^-e^+\tau^-\tau^+\mu^-\mu^+$	1.817 689 668	1.817 689 669	1.839± 0.028	10^{-21}	290 µs
$e^-e^+ \to \gamma\gamma\gamma\gamma\gamma\gamma\gamma$	3.017 082 072	3.017 087 972	2.754± 0.083	10^{-25}	230 ms
$e^-e^+ \to e^-e^+\tau^-\tau^+\mu^-\mu^+\gamma$	4.030 699 489	4.030 699 489	3.926± 0.072	10^{-25}	2.4 ms
$e^-e^+ \to e^-e^+e^-e^+\tau^-\tau^+\mu^-\mu^+$	4.837 687 543	4.837 687 543	4.95 ± 0.11	10^{-29}	13 ms

Table 8.7: Results for massless QED amplitudes.

Process	MadGraph	This work		Exp.	Time
		Summation	MC		
$\tau^-\tau^+ \to \gamma\gamma$	2.480 399 333	2.480 399 333	2.498± 0.023	10^{-2}	18 µs
$\tau^-\tau^+ \to \tau^-\tau^+$	4.585 851 516	4.585 851 516	4.513± 0.040	10^{-2}	7 µs
$\tau^-\tau^+ \to e^-e^+$	1.042 644 994	1.042 644 994	1.036± 0.010	10^{-2}	6 µs
$\tau^-\tau^+ \to \gamma\gamma\gamma$	1.391 912 355	1.391 912 355	1.406± 0.017	10^{-7}	25 µs
$\tau^-\tau^+ \to \tau^-\tau^+\gamma$	1.555 259 638	1.555 259 638	1.552± 0.019	10^{-6}	18 µs
$\tau^-\tau^+ \to e^-e^+\gamma$	3.331 185 143	3.331 185 069	3.299± 0.044	10^{-7}	13 µs
$\tau^-\tau^+ \to \gamma\gamma\gamma\gamma$	7.015 362 527	7.015 362 627	7.00 ± 0.12	10^{-12}	160 µs
$\tau^-\tau^+ \to \tau^-\tau^+\gamma\gamma$	3.606 810 100	3.606 810 100	3.732± 0.050	10^{-11}	90 µs
$\tau^-\tau^+ \to e^-e^+\gamma\gamma$	9.934 915 956	9.934 915 944	9.99 ± 0.14	10^{-12}	51 µs
$\tau^-\tau^+ \to \tau^-\tau^+\tau^-\tau^+$	1.536 172 397	1.536 172 397	1.551± 0.020	10^{-12}	80 µs
$\tau^-\tau^+ \to \tau^-\tau^+e^-e^+$	1.615 517 128	1.615 517 131	1.602± 0.017	10^{-13}	35 µs
$\tau^-\tau^+ \to \mu^-\mu^+e^-e^+$	1.453 165 977	1.453 165 977	1.449± 0.015	10^{-13}	24 µs
$\tau^-\tau^+ \to \gamma\gamma\gamma\gamma\gamma$	3.775 948 867	3.775 948 871	3.806± 0.081	10^{-16}	1.5 ms
$\tau^-\tau^+ \to \tau^-\tau^+\gamma\gamma\gamma$	2.139 350 732	2.139 350 732	2.172± 0.037	10^{-15}	680 µs
$\tau^-\tau^+ \to e^-e^+\gamma\gamma\gamma$	5.751 570 194	5.751 572 203	5.99 ± 0.11	10^{-16}	350 µs
$\tau^-\tau^+ \to \tau^-\tau^+\tau^-\tau^+\gamma$	7.042 021 837	7.042 021 838	7.24 ± 0.12	10^{-16}	500 µs
$\tau^-\tau^+ \to \tau^-\tau^+e^-e^+\gamma$	8.595 011 246	8.595 011 252	8.57 ± 0.13	10^{-17}	190 µs
$\tau^-\tau^+ \to \mu^-\mu^+e^-e^+\gamma$	7.564 083 861	7.564 083 855	7.50 ± 0.11	10^{-17}	110 µs
$\tau^-\tau^+ \to \gamma\gamma\gamma\gamma\gamma\gamma$	2.435 932 882	2.435 931 307	2.519± 0.058	10^{-20}	18 ms
$\tau^-\tau^+ \to \tau^-\tau^+\gamma\gamma\gamma\gamma$	7.258 868 052	7.258 868 804	7.50 ± 0.16	10^{-20}	6.8 ms
$\tau^-\tau^+ \to e^-e^+\gamma\gamma\gamma\gamma$	2.650 312 876	2.650 312 865	2.735± 0.052	10^{-20}	3.4 ms
$\tau^-\tau^+ \to \tau^-\tau^+\tau^-\tau^+\gamma\gamma$	1.407 002 705	1.407 002 711	1.378± 0.026	10^{-19}	4.4 ms
$\tau^-\tau^+ \to \tau^-\tau^+e^-e^+\gamma\gamma$	1.994 251 122	1.994 251 121	2.005± 0.037	10^{-20}	1.5 ms
$\tau^-\tau^+ \to \mu^-\mu^+e^-e^+\gamma\gamma$	1.615 064 955	1.615 064 956	1.623± 0.029	10^{-20}	810 µs
$\tau^-\tau^+ \to \tau^-\tau^+\tau^-\tau^+\tau^-\tau^+$	1.280 694 913	1.280 694 913	1.319± 0.021	10^{-21}	4.4 ms
$\tau^-\tau^+ \to \tau^-\tau^+\tau^-\tau^+e^-e^+$	1.478 398 138	1.478 399 676	1.490± 0.022	10^{-20}	1.2 ms
$\tau^-\tau^+ \to \tau^-\tau^+\mu^-\mu^+e^-e^+$	1.817 421 018	1.817 420 998	1.866± 0.028	10^{-21}	440 µs
$\tau^-\tau^+ \to \tau^-\tau^+\mu^-\mu^+e^-e^+\gamma$	4.029 695 445	4.029 695 443	4.058± 0.073	10^{-25}	3.8 ms
$\tau^-\tau^+ \to \tau^-\tau^+\mu^-\mu^+e^-e^+\gamma\gamma$	6.294 580 402	6.294 580 391	6.18 ± 0.14	10^{-28}	43 ms
$\tau^-\tau^+ \to \tau^-\tau^+\mu^-\mu^+e^-e^+e^-e^+$	1.535 142 635	1.535 142 634	1.543± 0.029	10^{-30}	20 ms

Table 8.8: Results for QED amplitudes; τ^+ and τ^- are massive ($m_\tau = 1.777\,\text{GeV}$).

Process	MadGraph	This work		Exponent	Time
		Summation	MC		
$gg \to gg$	5.517 925 063	5.517 925 306	5.560± 0.055	10^{1}	25 µs
$u\bar{u} \to gg$	1.894 101 514	1.894 101 514	1.907± 0.018		8 µs
$u\bar{u} \to u\bar{u}$	2.827 692 864	2.827 692 859	2.784± 0.023		8 µs
$u\bar{u} \to d\bar{d}$	5.664 500 612	5.664 500 612	5.676± 0.053	10^{-1}	6 µs
$gg \to ggg$	1.874 071 116	1.874 071 084	1.855± 0.015	10^{-2}	290 µs
$u\bar{u} \to ggg$	4.160 575 209	4.160 575 212	4.114± 0.042	10^{-4}	53 µs
$u\bar{u} \to u\bar{u}g$	1.503 648 786	1.503 648 789	1.473± 0.012	10^{-3}	42 µs
$u\bar{u} \to d\bar{d}g$	2.006 117 283	2.006 117 304	2.009± 0.020	10^{-4}	26 µs
$gg \to gggg$	1.592 992 585	1.592 992 582	1.587± 0.020	10^{-4}	5.8 ms
$u\bar{u} \to gggg$	2.971 190 596	2.971 190 596	2.958± 0.043	10^{-6}	640 µs
$u\bar{u} \to u\bar{u}gg$	4.010 720 370	4.010 720 390	3.987± 0.035	10^{-7}	360 µs
$u\bar{u} \to d\bar{d}gg$	9.500 669 440	9.500 669 510	9.50 ± 0.12	10^{-8}	210 µs
$u\bar{u} \to u\bar{u}u\bar{u}$	1.615 594 434	1.615 594 429	1.609± 0.013	10^{-8}	290 µs
$u\bar{u} \to u\bar{u}d\bar{d}$	1.260 887 951	1.260 887 947	1.264± 0.009	10^{-9}	120 µs
$u\bar{u} \to s\bar{s}d\bar{d}$	5.893 608 395	5.893 608 404	5.901± 0.056	10^{-10}	780 µs
$u\bar{u} \to ggggg$	8.221 390 281	8.221 390 296	8.23 ± 0.14	10^{-9}	12 ms
$u\bar{u} \to u\bar{u}ggg$	1.066 307 259	1.066 307 262	1.078± 0.011	10^{-9}	5.1 ms
$u\bar{u} \to d\bar{d}ggg$	2.401 573 559	2.401 573 569	2.383± 0.030	10^{-10}	2.6 ms
$u\bar{u} \to u\bar{u}u\bar{u}g$	1.546 913 633	1.546 913 631	1.533± 0.018	10^{-10}	3.1 ms
$u\bar{u} \to u\bar{u}d\bar{d}g$	1.063 656 616	1.063 656 614	1.053± 0.011	10^{-11}	1.1 ms
$u\bar{u} \to s\bar{s}d\bar{d}g$	4.689 219 755	4.689 219 764	4.686± 0.059	10^{-12}	630 µs
$u\bar{u} \to gggggg$	MG crashes	1.996 697 894	1.996± 0.028	10^{-11}	290 ms
$u\bar{u} \to u\bar{u}gggg$	3.163 329 136	3.163 329 162	3.178± 0.028	10^{-12}	110 ms
$u\bar{u} \to d\bar{d}gggg$	7.647 779 017	7.647 779 081	7.604± 0.077	10^{-13}	49 ms
$u\bar{u} \to u\bar{u}u\bar{u}gg$	1.169 132 377	1.169 132 373	1.186± 0.012	10^{-12}	73 ms
$u\bar{u} \to u\bar{u}d\bar{d}gg$	5.306 732 681	5.306 732 681	5.381± 0.051	10^{-14}	17 ms
$u\bar{u} \to s\bar{s}d\bar{d}gg$	2.782 964 637	2.782 964 643	2.834± 0.032	10^{-14}	8.7 ms
$u\bar{u} \to u\bar{u}u\bar{u}u\bar{u}$	5.187 078 128	5.187 078 099	5.143± 0.053	10^{-15}	56 ms
$u\bar{u} \to u\bar{u}u\bar{u}d\bar{d}$	3.434 168 171	3.434 168 160	3.401± 0.039	10^{-14}	9.4 ms
$u\bar{u} \to u\bar{u}s\bar{s}d\bar{d}$	1.667 728 261	1.667 728 260	1.651± 0.018	10^{-15}	3.5 ms
$u\bar{u} \to c\bar{c}s\bar{s}d\bar{d}$	9.108 404 943	9.108 404 965	9.06 ± 0.12	10^{-16}	2.1 ms
$u\bar{u} \to c\bar{c}s\bar{s}d\bar{d}g$	2.481 788 680	2.481 788 681	2.519± 0.031	10^{-18}	27 ms
$u\bar{u} \to c\bar{c}s\bar{s}d\bar{d}gg$	5.652 243 643	5.652 243 646	5.668± 0.080	10^{-20}	770 ms
$u\bar{u} \to t\bar{t}c\bar{c}s\bar{s}d\bar{d}$	2.659 899 850	2.659 943 280	2.683± 0.036	10^{-22}	110 ms

Table 8.9: Results for massless QCD amplitudes.

Process	MadGraph	This work		Exponent	Time
		Summation	MC		
$b\bar{b}\to gg$	1.894 307 057	1.894 307 057	1.902± 0.017		24 µs
$b\bar{b}\to b\bar{b}$	2.827 913 527	2.827 913 522	2.794± 0.019		11 µs
$b\bar{b}\to t\bar{t}$	6.164 071 899	6.164 071 899	6.153± 0.054	10^{-1}	9 µs
$b\bar{b}\to ggg$	4.161 080 888	4.161 080 888	4.189± 0.042	10^{-4}	57 µs
$b\bar{b}\to b\bar{b}g$	1.503 689 872	1.503 689 874	1.506± 0.011	10^{-3}	47 µs
$b\bar{b}\to t\bar{t}g$	2.307 788 087	2.307 794 561	2.287± 0.021	10^{-4}	30 µs
$b\bar{b}\to u\bar{u}g$	2.006 105 588	2.006 105 592	2.000± 0.018	10^{-4}	30 µs
$b\bar{b}\to gggg$	2.970 647 369	2.970 647 371	2.991± 0.041	10^{-6}	670 µs
$b\bar{b}\to b\bar{b}gg$	4.003 163 884	4.003 163 904	3.978± 0.032	10^{-7}	380 µs
$b\bar{b}\to t\bar{t}gg$	1.986 626 008	1.986 773 712	1.979± 0.016	10^{-8}	220 µs
$b\bar{b}\to u\bar{u}gg$	9.501 287 305	9.501 287 379	9.56 ± 0.11	10^{-8}	220 µs
$b\bar{b}\to b\bar{b}b\bar{b}$	1.612 310 599	1.612 310 594	1.615± 0.011	10^{-8}	310 µs
$b\bar{b}\to b\bar{b}t\bar{t}$	1.476 276 143	1.476 275 123	1.454± 0.009	10^{-9}	130 µs
$b\bar{b}\to b\bar{b}u\bar{u}$	1.261 022 794	1.261 022 790	1.251± 0.008	10^{-9}	130 µs
$b\bar{b}\to t\bar{t}u\bar{u}$	4.995 374 822	4.995 374 511	5.061± 0.039	10^{-10}	84 µs
$b\bar{b}\to u\bar{u}d\bar{d}$	5.894 074 019	5.894 074 050	5.935± 0.052	10^{-10}	87 µs
$b\bar{b}\to ggggg$	8.217 301 646	8.217 301 651	8.27 ± 0.15	10^{-9}	12 ms
$b\bar{b}\to b\bar{b}ggg$	1.064 065 558	1.064 065 560	1.049± 0.010	10^{-9}	5.3 ms
$b\bar{b}\to t\bar{t}ggg$	9.123 811 793	9.124 279 522	9.10 ± 0.11	10^{-11}	2.7 ms
$b\bar{b}\to u\bar{u}ggg$	2.400 956 398	2.400 956 414	2.428± 0.029	10^{-10}	2.7 ms
$b\bar{b}\to b\bar{b}b\bar{b}g$	1.539 839 995	1.539 839 994	1.544± 0.017	10^{-10}	3.3 ms
$b\bar{b}\to b\bar{b}t\bar{t}g$	1.191 784 746	1.191 794 804	1.167± 0.011	10^{-11}	1.4 ms
$b\bar{b}\to b\bar{b}u\bar{u}g$	1.063 747 694	1.063 747 694	1.062± 0.010	10^{-11}	1.3 ms
$b\bar{b}\to t\bar{t}u\bar{u}g$	5.066 004 846	5.066 016 584	5.119± 0.056	10^{-12}	810 µs
$b\bar{b}\to u\bar{u}d\bar{d}g$	4.687 903 567	4.687 903 574	4.623± 0.057	10^{-12}	670 µs
$b\bar{b}\to gggggg$	MG crashes	1.995 512 405	1.969± 0.026	10^{-11}	290 ms
$b\bar{b}\to b\bar{b}gggg$	3.151 060 659	3.151 060 679	3.168± 0.026	10^{-12}	110 ms
$b\bar{b}\to t\bar{t}gggg$	3.032 975 546	3.033 141 415	3.051± 0.034	10^{-13}	51 ms
$b\bar{b}\to u\bar{u}gggg$	7.644 702 624	7.644 702 687	7.738± 0.076	10^{-13}	51 ms
$b\bar{b}\to b\bar{b}b\bar{b}gg$	1.156 892 157	1.156 892 154	1.148± 0.010	10^{-12}	76 ms
$b\bar{b}\to b\bar{b}t\bar{t}gg$	7.589 343 534	7.589 356 491	7.635± 0.073	10^{-14}	21 ms
$b\bar{b}\to b\bar{b}u\bar{u}gg$	5.306 019 637	5.306 019 885	5.384± 0.047	10^{-14}	18 ms
$b\bar{b}\to t\bar{t}u\bar{u}gg$	2.881 753 736	2.881 747 256	2.948± 0.030	10^{-14}	9.0 ms
$b\bar{b}\to u\bar{u}d\bar{d}gg$	2.781 658 246	2.781 658 251	2.802± 0.031	10^{-14}	9.1 ms
$b\bar{b}\to b\bar{b}b\bar{b}b\bar{b}$	4.976 786 168	4.976 786 140	4.961± 0.046	10^{-15}	58 ms
$b\bar{b}\to b\bar{b}b\bar{b}t\bar{t}$	6.075 705 590	6.075 629 485	6.156± 0.047	10^{-16}	10 ms
$b\bar{b}\to b\bar{b}b\bar{b}u\bar{u}$	3.400 713 929	3.400 713 919	3.387± 0.034	10^{-14}	9.8 ms
$b\bar{b}\to b\bar{b}t\bar{t}t\bar{t}$	2.310 318 512	2.310 317 649	2.328± 0.024	10^{-17}	6.6 ms
$b\bar{b}\to b\bar{b}t\bar{t}u\bar{u}$	2.845 538 948	2.845 530 568	2.818± 0.030	10^{-15}	3.6 ms
$b\bar{b}\to t\bar{t}u\bar{u}d\bar{d}$	9.963 104 890	9.963 077 643	10.05 ± 0.11	10^{-16}	2.2 ms
$b\bar{b}\to t\bar{t}u\bar{u}d\bar{d}g$	3.115 771 908	3.115 784 889	3.140± 0.040	10^{-18}	29 ms
$b\bar{b}\to t\bar{t}u\bar{u}d\bar{d}gg$	2.756 291 325	2.756 324 475	2.750± 0.032	10^{-20}	790 ms
$b\bar{b}\to t\bar{t}u\bar{u}d\bar{d}s\bar{s}$	6.085 860 969	6.085 757 155	6.065± 0.093	10^{-22}	120 ms

Table 8.10: Results for QCD amplitudes; t and b quarks are massive ($m_t = 174.3\,\text{GeV}$, $m_b = 4.7\,\text{GeV}$).

Process	MadGraph	This work		Exponent	Time
		Summation	MC		
$u\bar{u} \to \gamma\gamma$	1.633 155 901	1.633 155 901	1.622± 0.015	10^{-3}	20 µs
$u\bar{u} \to g\gamma$	1.532 144 490	1.532 144 490	1.513± 0.014	10^{-1}	7 µs
$u\bar{u} \to \gamma\gamma\gamma$	4.073 224 465	4.073 224 464	4.116± 0.052	10^{-9}	35 µs
$u\bar{u} \to g\gamma\gamma$	5.731 940 609	5.731 940 608	5.740± 0.073	10^{-7}	23 µs
$u\bar{u} \to gg\gamma$	1.422 023 299	1.422 023 299	1.398± 0.017	10^{-5}	39 µs
$u\bar{u} \to u\bar{u}\gamma$	4.294 580 742	4.294 580 734	4.324± 0.050	10^{-5}	39 µs
$u\bar{u} \to d\bar{d}\gamma$	4.412 646 475	4.412 646 476	4.390± 0.057	10^{-6}	22 µs
$u\bar{u} \to \gamma\gamma\gamma\gamma$	9.124 807 275	9.124 807 343	9.14 ± 0.16	10^{-14}	230 µs
$u\bar{u} \to g\gamma\gamma\gamma$	1.712 086 786	1.712 086 785	1.738± 0.031	10^{-11}	130 µs
$u\bar{u} \to gg\gamma\gamma$	9.253 871 209	9.253 871 209	9.52 ± 0.17	10^{-10}	220 µs
$u\bar{u} \to ggg\gamma$	2.656 030 228	2.656 030 228	2.628± 0.044	10^{-7}	410 µs
$u\bar{u} \to u\bar{u}\gamma\gamma$	4.625 038 055	4.625 038 044	4.684± 0.056	10^{-10}	220 µs
$u\bar{u} \to u\bar{u}g\gamma$	4.292 899 101	4.292 899 093	4.215± 0.042	10^{-8}	240 µs
$u\bar{u} \to d\bar{d}\gamma\gamma$	9.888 856 074	9.888 856 076	9.97 ± 0.14	10^{-12}	120 µs
$u\bar{u} \to d\bar{d}g\gamma$	1.778 734 781	1.778 734 782	1.781± 0.020	10^{-9}	130 µs
$u\bar{u} \to \gamma\gamma\gamma\gamma\gamma$	2.183 000 852	2.183 000 856	2.169± 0.048	10^{-18}	2.2 ms
$u\bar{u} \to g\gamma\gamma\gamma\gamma$	5.119 953 222	5.119 953 224	5.06 ± 0.11	10^{-16}	920 µs
$u\bar{u} \to gg\gamma\gamma\gamma$	3.612 888 005	3.612 888 007	3.630± 0.078	10^{-14}	1.7 ms
$u\bar{u} \to ggg\gamma\gamma$	1.461 503 009	1.461 502 989	1.457± 0.031	10^{-11}	3.4 ms
$u\bar{u} \to gggg\gamma$	3.412 442 730	3.412 442 730	3.345± 0.065	10^{-10}	8.2 ms
$u\bar{u} \to u\bar{u}\gamma\gamma\gamma$	1.200 723 898	1.200 723 895	1.240± 0.020	10^{-14}	1.9 ms
$u\bar{u} \to u\bar{u}g\gamma\gamma$	1.607 153 738	1.607 153 735	1.590± 0.021	10^{-12}	1.6 ms
$u\bar{u} \to u\bar{u}gg\gamma$	3.060 595 514	3.060 595 526	3.066± 0.037	10^{-11}	3.1 ms
$u\bar{u} \to d\bar{d}\gamma\gamma\gamma$	4.294 867 963	4.294 867 963	4.323± 0.078	10^{-16}	910 µs
$u\bar{u} \to d\bar{d}g\gamma\gamma$	1.152 774 507	1.152 774 508	1.140± 0.017	10^{-13}	820 µs
$u\bar{u} \to d\bar{d}gg\gamma$	1.380 675 327	1.380 675 337	1.362± 0.022	10^{-11}	1.6 ms
$u\bar{u} \to u\bar{u}u\bar{u}\gamma$	3.577 945 551	3.577 945 540	3.572± 0.043	10^{-12}	2.5 ms
$u\bar{u} \to u\bar{u}d\bar{d}\gamma$	7.417 498 076	7.417 498 060	7.404± 0.083	10^{-14}	850 µs
$u\bar{u} \to d\bar{d}s\bar{s}\gamma$	3.790 751 071	3.790 751 080	3.739± 0.051	10^{-14}	450 µs
$u\bar{u} \to \gamma\gamma\gamma\gamma\gamma\gamma$	6.259 607 014	6.259 606 915	6.509± 0.16	10^{-23}	27 ms
$u\bar{u} \to g\gamma\gamma\gamma\gamma\gamma$	1.761 734 270	1.761 734 271	1.787± 0.043	10^{-20}	9.1 ms
$u\bar{u} \to gg\gamma\gamma\gamma\gamma$	1.453 613 746	1.453 613 746	1.533± 0.036	10^{-18}	16 ms
$u\bar{u} \to ggg\gamma\gamma\gamma$	5.542 273 143	5.542 273 141	5.64 ± 0.12	10^{-16}	34 ms
$u\bar{u} \to gggg\gamma\gamma$	2.504 377 604	2.504 377 606	2.473± 0.051	10^{-14}	73 ms
$u\bar{u} \to ggggg\gamma$	9.943 865 518	9.943 865 520	9.79 ± 0.19	10^{-13}	150 ms
$u\bar{u} \to u\bar{u}\gamma\gamma\gamma\gamma$	1.734 344 143	1.734 344 142	1.749± 0.030	10^{-19}	19 ms
$u\bar{u} \to u\bar{u}g\gamma\gamma\gamma$	3.123 807 920	3.123 807 918	3.140± 0.048	10^{-17}	14 ms
$u\bar{u} \to u\bar{u}gg\gamma\gamma$	9.132 402 665	9.132 402 710	9.07 ± 0.12	10^{-16}	27 ms
$u\bar{u} \to u\bar{u}ggg\gamma$	4.937 353 433	4.937 353 481	4.963± 0.059	10^{-14}	54 ms

Table 8.11a: Results for massless QCD amplitudes with photons (4 to 8 particles).

Process	MadGraph	This work		Exponent	Time
		Summation	MC		
$u\bar{u} \to d\bar{d}\gamma\gamma\gamma\gamma$	2.232 170 080	2.232 170 081	2.180± 0.043	10^{-20}	9.6 ms
$u\bar{u} \to d\bar{d}g\gamma\gamma\gamma$	6.926 133 862	6.926 133 868	6.93 ± 0.11	10^{-18}	7.4 ms
$u\bar{u} \to d\bar{d}gg\gamma\gamma$	1.285 677 300	1.285 677 300	1.305± 0.023	10^{-15}	14 ms
$u\bar{u} \to d\bar{d}ggg\gamma$	4.521 343 531	4.521 343 557	4.549± 0.064	10^{-14}	27 ms
$u\bar{u} \to u\bar{u}u\bar{u}\gamma\gamma$	4.120 105 878	4.120 105 864	4.135± 0.062	10^{-16}	23 ms
$u\bar{u} \to u\bar{u}u\bar{u}g\gamma$	3.108 451 965	3.108 451 959	3.097± 0.041	10^{-14}	28 ms
$u\bar{u} \to u\bar{u}d\bar{d}\gamma\gamma$	2.327 988 021	2.327 988 025	2.363± 0.030	10^{-18}	7.7 ms
$u\bar{u} \to u\bar{u}d\bar{d}g\gamma$	5.157 945 726	5.157 945 744	5.173± 0.067	10^{-16}	9.3 ms
$u\bar{u} \to d\bar{d}s\bar{s}\gamma\gamma$	3.013 569 291	3.013 569 301	3.034± 0.047	10^{-18}	3.9 ms
$u\bar{u} \to d\bar{d}s\bar{s}g\gamma$	6.119 618 814	6.119 618 835	6.118± 0.091	10^{-16}	4.8 ms
$u\bar{u} \to d\bar{d}s\bar{s}c\bar{c}\gamma$	4.365 881 729	4.365 881 736	4.379± 0.076	10^{-20}	18 ms
$u\bar{u} \to d\bar{d}s\bar{s}c\bar{c}\gamma\gamma$	1.710 082 897	1.710 082 903	1.777± 0.038	10^{-24}	200 ms
$u\bar{u} \to d\bar{d}s\bar{s}c\bar{c}g\gamma$	1.996 821 078	1.996 821 083	1.929± 0.030	10^{-22}	250 ms

Table 8.11b: Results for massless QCD amplitudes with photons (8 to 10 particles).

Process	MadGraph	This work		Exponent	Time
		Summation	MC		
$b\bar{b} \to \gamma\gamma$	1.020 848 537	1.020 848 537	1.018± 0.009	10^{-4}	11 µs
$b\bar{b} \to g\gamma$	3.830 834 424	3.830 834 424	3.807± 0.034	10^{-2}	10 µs
$b\bar{b} \to \gamma\gamma\gamma$	6.364 898 558	6.364 898 557	6.346± 0.079	10^{-11}	39 µs
$b\bar{b} \to g\gamma\gamma$	3.582 736 069	3.582 736 068	3.619± 0.044	10^{-8}	27 µs
$b\bar{b} \to gg\gamma$	3.555 291 657	3.555 291 657	3.549± 0.043	10^{-6}	40 µs
$b\bar{b} \to b\bar{b}\gamma$	1.073 674 789	1.073 674 787	1.076± 0.012	10^{-5}	44 µs
$b\bar{b} \to t\bar{t}\gamma$	3.513 272 039	3.513 290 324	3.478± 0.042	10^{-6}	27 µs
$b\bar{b} \to u\bar{u}\gamma$	3.418 372 365	3.418 372 367	3.380± 0.042	10^{-6}	27 µs
$b\bar{b} \to \gamma\gamma\gamma\gamma$	3.562 947 911	3.562 952 606	3.654± 0.064	10^{-16}	250 µs
$b\bar{b} \to g\gamma\gamma\gamma$	2.674 062 412	2.674 062 412	2.693± 0.045	10^{-13}	140 µs
$b\bar{b} \to gg\gamma\gamma$	5.781 033 993	5.781 033 993	5.820± 0.097	10^{-11}	220 µs
$b\bar{b} \to ggg\gamma$	6.637 363 570	6.637 363 572	6.575± 0.010	10^{-8}	410 µs
$b\bar{b} \to b\bar{b}\gamma\gamma$	2.882 522 414	2.882 522 407	2.899± 0.033	10^{-11}	240 µs
$b\bar{b} \to b\bar{b}g\gamma$	1.070 150 993	1.070 150 991	1.067± 0.009	10^{-8}	250 µs
$b\bar{b} \to t\bar{t}\gamma\gamma$	2.452 000 385	2.452 269 896	2.439± 0.030	10^{-12}	130 µs
$b\bar{b} \to t\bar{t}g\gamma$	2.874 440 976	2.874 633 879	2.839± 0.022	10^{-10}	140 µs
$b\bar{b} \to u\bar{u}\gamma\gamma$	5.934 390 043	5.934 390 107	5.788± 0.081	10^{-12}	130 µs
$b\bar{b} \to u\bar{u}g\gamma$	5.733 577 253	5.733 577 265	5.747± 0.066	10^{-10}	104 µs

Table 8.12a: Results for QCD amplitudes with photons (4 to 6 particles); t and b quarks are massive ($m_t = 174.3\,\text{GeV}$, $m_b = 4.7\,\text{GeV}$).

Process	MadGraph	This work		Exponent	Time
		Summation	MC		
$b\bar{b}\to \gamma\gamma\gamma\gamma\gamma$	2.129 720 980	2.129 720 981	2.137± 0.044	10^{-21}	2.4 ms
$b\bar{b}\to g\gamma\gamma\gamma\gamma$	1.997 996 800	1.997 996 799	1.969± 0.042	10^{-18}	990 µs
$b\bar{b}\to gg\gamma\gamma\gamma$	5.639 091 259	5.639 091 264	5.53 ± 0.12	10^{-16}	1.7 ms
$b\bar{b}\to ggg\gamma\gamma$	9.125 000 018	9.125 000 022	9.00 ± 0.17	10^{-13}	3.4 ms
$b\bar{b}\to gggg\gamma$	8.524 855 621	8.524 855 623	8.34 ± 0.15	10^{-11}	7.0 ms
$b\bar{b}\to b\bar{b}\gamma\gamma\gamma$	1.869 757 416	1.869 757 412	1.860± 0.027	10^{-16}	1.9 ms
$b\bar{b}\to b\bar{b}g\gamma\gamma$	1.000 932 860	1.000 932 859	0.966± 0.012	10^{-13}	1.7 ms
$b\bar{b}\to b\bar{b}gg\gamma$	7.632 678 852	7.632 678 891	7.639± 0.087	10^{-12}	3.1 ms
$b\bar{b}\to t\bar{t}\gamma\gamma\gamma$	6.697 670 480	6.698 700 134	6.58 ± 0.11	10^{-17}	980 µs
$b\bar{b}\to t\bar{t}g\gamma\gamma$	1.098 821 071	1.098 947 264	1.083± 0.014	10^{-14}	860 µs
$b\bar{b}\to t\bar{t}gg\gamma$	1.410 029 517	1.410 200 326	1.408± 0.017	10^{-12}	1.6 ms
$b\bar{b}\to u\bar{u}\gamma\gamma\gamma$	1.991 328 819	1.991 328 844	1.959± 0.033	10^{-16}	990 µs
$b\bar{b}\to u\bar{u}g\gamma\gamma$	2.803 208 989	2.803 208 982	2.811± 0.045	10^{-14}	860 µs
$b\bar{b}\to u\bar{u}gg\gamma$	8.129 405 141	8.129 405 213	8.07 ± 0.13	10^{-12}	1.6 ms
$b\bar{b}\to b\bar{b}b\bar{b}\gamma$	8.906 232 056	8.906 232 029	8.871± 0.097	10^{-13}	2.6 ms
$b\bar{b}\to b\bar{b}t\bar{t}\gamma$	2.568 618 161	2.568 470 617	2.576± 0.026	10^{-14}	910 µs
$b\bar{b}\to t\bar{t}u\bar{u}\gamma$	2.165 411 446	2.164 991 004	2.159± 0.017	10^{-15}	480 µs
$b\bar{b}\to \gamma\gamma\gamma\gamma\gamma\gamma$	1.525 811 122	1.525 811 031	1.565± 0.035	10^{-26}	29 ms
$b\bar{b}\to g\gamma\gamma\gamma\gamma\gamma$	1.717 726 839	1.717 726 841	1.770± 0.042	10^{-23}	9.7 ms
$b\bar{b}\to gg\gamma\gamma\gamma\gamma$	5.668 598 806	5.668 598 807	6.07 ± 0.14	10^{-21}	17 ms
$b\bar{b}\to ggg\gamma\gamma\gamma$	8.645 562 896	8.645 562 896	8.85 ± 0.19	10^{-18}	35 ms
$b\bar{b}\to gggg\gamma\gamma$	1.563 377 847	1.563 377 850	1.542± 0.031	10^{-15}	75 ms
$b\bar{b}\to ggggg\gamma$	2.483 471 988	2.483 472 017	2.456± 0.046	10^{-13}	160 ms
$b\bar{b}\to b\bar{b}\gamma\gamma\gamma\gamma$	6.736 970 738	6.736 970 703	6.72 ± 0.11	10^{-22}	20 ms
$b\bar{b}\to b\bar{b}g\gamma\gamma\gamma$	4.851 557 748	4.851 557 744	4.848± 0.069	10^{-19}	15 ms
$b\bar{b}\to b\bar{b}gg\gamma\gamma$	5.682 274 033	5.682 274 055	5.693± 0.072	10^{-17}	28 ms
$b\bar{b}\to b\bar{b}ggg\gamma$	1.229 158 078	1.229 158 079	1.225± 0.014	10^{-14}	55 ms
$b\bar{b}\to t\bar{t}\gamma\gamma\gamma$	1.196 966 857	1.197 267 612	1.148± 0.026	10^{-21}	10 ms
$b\bar{b}\to t\bar{t}g\gamma\gamma$	2.648 688 125	2.649 135 642	2.673± 0.047	10^{-19}	7.6 ms
$b\bar{b}\to t\bar{t}gg\gamma\gamma$	4.263 218 357	4.263 989 963	4.190± 0.068	10^{-17}	14 ms
$b\bar{b}\to t\bar{t}ggg\gamma$	6.286 654 330	6.287 148 695	6.283± 0.092	10^{-15}	28 ms
$b\bar{b}\to u\bar{u}\gamma\gamma\gamma\gamma$	5.896 306 652	5.896 306 657	5.82 ± 0.12	10^{-21}	10 ms
$b\bar{b}\to u\bar{u}g\gamma\gamma\gamma$	1.114 241 181	1.114 241 247	1.110± 0.020	10^{-18}	7.7 ms
$b\bar{b}\to u\bar{u}gg\gamma\gamma$	3.523 111 698	3.523 111 734	3.448± 0.064	10^{-16}	14 ms
$b\bar{b}\to u\bar{u}ggg\gamma$	2.339 919 389	2.339 919 392	2.300± 0.030	10^{-14}	28 ms
$b\bar{b}\to b\bar{b}b\bar{b}\gamma\gamma$	2.550 142 767	2.550 142 758	2.574± 0.036	10^{-17}	25 ms
$b\bar{b}\to b\bar{b}b\bar{b}g\gamma$	7.689 025 854	7.689 025 836	7.662± 0.094	10^{-15}	28 ms
$b\bar{b}\to b\bar{b}t\bar{t}\gamma\gamma$	2.883 160 738	2.882 833 944	2.862± 0.039	10^{-19}	8.2 ms
$b\bar{b}\to b\bar{b}t\bar{t}g\gamma$	1.634 520 501	1.634 537 154	1.628± 0.022	10^{-16}	9.6 ms
$b\bar{b}\to t\bar{t}u\bar{u}\gamma\gamma$	7.459 971 806	7.458 685 641	7.519± 0.090	10^{-20}	4.1 ms
$b\bar{b}\to t\bar{t}u\bar{u}g\gamma$	1.914 902 780	1.914 932 528	1.942± 0.027	10^{-16}	4.9 ms
$b\bar{b}\to t\bar{t}u\bar{u}d\bar{d}\gamma$	2.764 245 038	2.764 031 935	2.752± 0.036	10^{-20}	18 ms
$b\bar{b}\to t\bar{t}u\bar{u}d\bar{d}\gamma\gamma$	4.830 902 487	4.830 462 551	4.880± 0.085	10^{-24}	210 ms
$b\bar{b}\to t\bar{t}u\bar{u}d\bar{d}g\gamma$	8.045 013 536	8.044 752 171	8.22 ± 0.12	10^{-22}	260 ms

Table 8.12b: Results for QCD amplitudes with photons (7 to 10 particles); t and b quarks are massive ($m_t = 174.3\,\text{GeV}$, $m_b = 4.7\,\text{GeV}$).

Chapter 9

Conclusion and Outlook

What have we achieved in this thesis? On top of the existing library *particle scattering*, we developed and implemented algorithms to compute squared amplitudes for different parts of the Standard Model. The results for all implemented models, i.e. gluon amplitudes, QED, QCD and QCD with external photons, agree well with results obtained with the program *MadGraph* for a total of 256 tested processes — including massless and massive fermions. Various methods for optimizing the computation were developed and have proven to speed up computations.

This chapter shortly summarizes the approach to the implementation, recapitulates the results and, based on this, gives an outlook on what can be done to extend the present program and further optimize the methods.

9.1 Summary

To implement sophisticated computations such as the computation of tree-level amplitudes for arbitrary processes within the context of the available interaction models, one has to begin with a well-defined physical basis formulated in appropriate formalisms. To this end, the color-flow decomposition for QCD amplitudes was introduced, along with the Weyl-van der Waerden formalism for helicity spinors.

Based on this, we then identified three main ingredients for the computation of matrix elements:

- The *particle configuration* which yields all possible color-orderings,
- the *color matrix* which contains all structures resulting from the $SU(N)$ character of the gauge theory and
- the computation of *partial amplitudes* which was realized with Berends-Giele-like recursion formulas.

The program was designed such that these general ingredients can be implemented separately for different interaction models, but the user accesses them via one common interface. This interface generally performs the squaring of amplitudes, for which we employed the method of turning helicities and spins into continuous variables. This enables one to perform a Monte Carlo integration instead of summation over all helicity configurations. When combining this with the phase space integration which will be

added in the future, this effectively reduces this sum over all helicities down to one squared amplitude with continuous helicities.

After a few other general considerations in chapter 3, chapters 4 to 7 explained the concepts that go into the different models. We found that gluon amplitudes, see chapter 4, are not difficult to implement since their coupling among each other can be described by a more simple $U(3)$ gauge group. Still, we saw that the computational effort grows factorially with the number of gluons, hence the Berends-Giele recursion was optimized by turning it into an iterative approach which avoids computing the same currents over and over again. QED amplitudes, chapter 5, introduced fermions which lead to the introduction of Dirac spinors instead of the previously used Weyl spinors. In chapter 6 we turned to the full $SU(3)$ color gauge group by looking at QCD amplitudes, where the concept of color clusters and the $U(1)$ yielded several complications. Finally, we added external photons to QCD amplitudes as an example for mixing different gauge groups. Therein, we developed rather general methods which can be extended to couple other gauge groups as well, which ultimately aims at the computation of the full Standard Model.

9.2 Discussion and Outlook

In the final chapter, we put the theoretical discussion to practical use by actually testing the program. This was done in two separate parts.

On the one hand, we verified that the developed algorithms and methods actually work by computing squared amplitudes for many processes and comparing them successfully to reference results obtained by *MadGraph*. Not only did this show that the exact method of summing over all helicity configurations works well, but it also proved that the proposed integration over helicity angles yields the expected results.

On the other hand, we tested the program for performance. For this, we checked the proposed methods of optimization against non-optimized versions. In doing so we found that by far the most efficient methods to speed up the computation are those methods specifically developed for the gluon amplitudes: The algorithm for the leading color matrix (speed-up about more than 97 %) and the iterative precomputation of all currents (speed-up about more than 70 %).

In light of this observation, one primary concern for the future is to extend this kind of optimization to all other models. The color matrix optimization is restricted to the leading color approach and hence to gluon amplitudes, but the iterative precomputation of currents can be adapted. Although a straightforward extension is not possible due to several technical details which we discussed in the respective chapters, we presented some promising ideas which have to be developed further.

Apart from optimizing the existing methods, another important aspect is to extend the program by new models: The existing set of models covers only part of the Standard Model of particle physics. The model which adds external photons to QCD provides the basic ideas to include even more interactions. This can be extended in several ways:

- Leptons can be added. Since they do not couple to gluons or quarks, this requires only minor changes to the algorithm for the partial amplitudes.

- Currently, photons are only considered as *external* particles; however, when mixing different gauge groups, the particles can also appear as virtual particles. Since the $U(1)$ current already provides for a colorless virtual particle, other colorless virtual particles can be included easily. In this sense, the appearance of color clusters and the $U(1)$ gluon which first look like an inconvenient disadvantage of the color-flow representation, seems to turn out as a feature for the inclusion of colorless particles.

- The available models lack another important part of the Standard Model: Weak interactions. If electroweak bosons get included in terms of the broken electroweak symmetry where we have W^+, W^- and Z^0 bosons in addition to photons (plus the currently missing neutrinos), we add them as colorless particles, hence the basic ideas for the coupling of external photons to QCD apply here as well.

In conclusion, we have achieved the efficient automated computation of tree-level amplitudes for the implemented models which forms a solid foundation for further optimizations and extension to other models such as the full Standard Model of particle physics.

Appendix A

Color-Flow Feynman Rules

This chapter lists the color-flow Feynman rules used for the program. They are not difficult to derive, in parts this is done in section 2.3.3. We leave out particle polarizations and spinors since they were exhaustively discussed in section 2.3.

A.1 Propagators

Gluon:

$$\dot{B}A \xrightarrow{k} \dot{D}C \quad = \frac{\mathrm{i}}{k^2}\left(-\epsilon^{\dot{B}\dot{D}}\,\epsilon^{AC}\right)$$

$$\begin{matrix} i_a \\ j_a \end{matrix} \begin{matrix} \longleftarrow \\ \longrightarrow \end{matrix} \begin{matrix} j_b \\ i_b \end{matrix} \quad = \delta^{j_b}_{i_a}\,\delta^{j_a}_{i_b}$$

$$\begin{matrix} i_a \\ j_a \end{matrix} \circlearrowleft \circlearrowright \begin{matrix} j_b \\ i_b \end{matrix} \quad = -\frac{1}{N_c}\delta^{j_a}_{i_a}\,\delta^{j_b}_{i_b}$$

Quark / antiquark:

$$B/\dot{B} \xleftarrow{p} \dot{A}/A \quad = \frac{\mathrm{i}}{p^2 - m^2}\begin{pmatrix} m\,\delta^B_A & p_\mu \sigma^\mu_{A\dot{B}} \\ p_\mu \bar{\sigma}^{\mu,\dot{A}B} & m\,\delta^{\dot{A}}_{\dot{B}} \end{pmatrix}$$

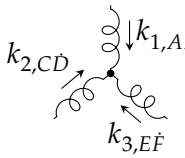
$$i \longleftarrow j \quad = \delta^i_j$$

Photon:

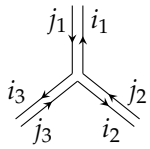
$$\dot{B}A \xrightarrow{k} \dot{D}C \quad = \frac{\mathrm{i}}{k^2}\left(-\epsilon^{\dot{B}\dot{D}}\,\epsilon^{AC}\right)$$

A.2 Vertices

Three gluon vertex:

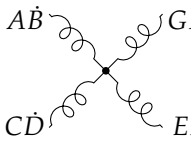


$$= \frac{ig_3}{\sqrt{2}}\Big(\epsilon_{AC}\,\epsilon_{\dot{B}\dot{D}}\,(k_2-k_1)_{E\dot{F}} + \epsilon_{CE}\,\epsilon_{\dot{D}\dot{F}}\,(k_3-k_2)_{A\dot{B}} + \epsilon_{EA}\,\epsilon_{\dot{F}\dot{B}}\,(k_1-k_3)_{C\dot{D}}\Big)$$

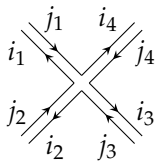


$$= \frac{1}{\sqrt{2}}\,\delta^{i_1}_{j_2}\,\delta^{i_2}_{j_3}\,\delta^{i_3}_{j_1}$$

Four gluon vertex:

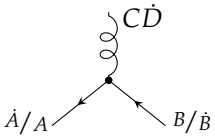


$$= ig_3^2\Big(2\,\epsilon_{AE}\,\epsilon_{\dot{B}\dot{F}}\,\epsilon_{CG}\,\epsilon_{\dot{D}\dot{H}} - \epsilon_{AC}\,\epsilon_{\dot{B}\dot{D}}\,\epsilon_{EG}\,\epsilon_{\dot{F}\dot{H}} - \epsilon_{AG}\,\epsilon_{\dot{B}\dot{H}}\,\epsilon_{CE}\,\epsilon_{\dot{D}\dot{F}}\Big)$$

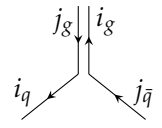


$$= \frac{1}{\sqrt{2}}\,\delta^{i_1}_{j_2}\,\delta^{i_2}_{j_3}\,\delta^{i_3}_{j_4}\,\delta^{i_4}_{j_1}$$

Quark gluon vertex:

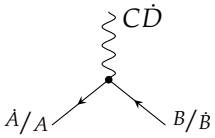


$$= \begin{cases} -ig_3\sqrt{2}\,\epsilon_{CA}\,\epsilon_{\dot{D}\dot{B}} & \text{for indices } A,\dot{B} \\ -ig_3\sqrt{2}\,\delta_C^{\ B}\,\delta_{\dot{D}}^{\ \dot{A}} & \text{for indices } \dot{A},B \end{cases}$$

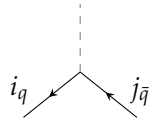


$$= \frac{1}{\sqrt{2}}\,\delta^{i_g}_{j_{\bar{q}}}\,\delta^{i_q}_{j_g}$$

Quark photon vertex:

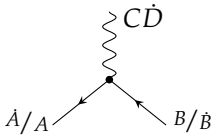


$$= \begin{cases} -ie\sqrt{2}\,Q_f\,\epsilon_{CA}\,\epsilon_{\dot{D}\dot{B}} & \text{for indices } A,\dot{B} \\ -ie\sqrt{2}\,Q_f\,\delta_C^{\ B}\,\delta_{\dot{D}}^{\ \dot{A}} & \text{for indices } \dot{A},B \end{cases}$$



$$= \delta^{i_q}_{j_{\bar{q}}}$$

Lepton photon vertex:



$$= \begin{cases} -\mathrm{ie}\sqrt{2}\,\epsilon_{CA}\,\epsilon_{\dot{D}\dot{B}} & \text{for indices } A, \dot{B} \\ -\mathrm{ie}\sqrt{2}\,\delta_C^{\ B}\,\delta_{\dot{D}}^{\ \dot{A}} & \text{for indices } \dot{A}, B \end{cases}$$

References

Entries [1–5] provide literature not cited in this thesis, but used for the implementation of the program.

[1] B. Stroustrup. *The C++ programming language*. Addison-Wesley, 1997.

[2] S. Meyers. *Effective C++: 55 specific ways to improve your programs and designs*. Addison-Wesley Professional, 2005.

[3] S. Meyers. *More Effective C++: 35 new ways to improve your programs and designs*. Addison-Wesley Professional, 1995.

[4] S. Meyers. *Effective STL*. Addison-Wesley Professional, 2008.

[5] S. Hoffmann and R. Lienhart. *OpenMP (Informatik im Fokus)*. Berlin: Springer Verlag, 2008.

[6] *BOOST C++ Libraries*. URL: http://www.boost.org.

[7] S. Weinzierl. *Automated calculations for multi-leg processes*. PoS **ACAT** (2007), p. 005.

[8] T. Stelzer and W. F. Long. *Automatic generation of tree level helicity amplitudes*. Comput. Phys. Commun. **81** (1994), pp. 357–371.

[9] J. Alwall et al. *MadGraph/MadEvent v4: The New Web Generation*. JHEP **09** (2007), p. 028.

[10] J. Alwall et al. *New Developments in MadGraph/MadEvent*. AIP Conf. Proc. **1078** (2009), pp. 84–89.

[11] *The MadGraph Matrix Element Generator version 5*. URL: http://launchpad.net/madgraph5.

[12] T Gleisberg and S Hoeche. *Comix, a new matrix element generator*. JHEP **12** (2008), p. 039.

[13] R. Kleiss and G. van den Oord. *CAMORRA: a C++ library for recursive computation of particle scattering amplitudes*. Comput. Phys. Commun. **182** (2011), pp. 435–447.

[14] S. Weinzierl. *Automated computation of spin- and colour-correlated Born matrix elements*. Eur. Phys. J. **C45** (2006), pp. 745–757.

[15] S. Becker, C. Reuschle, and S. Weinzierl. *Numerical NLO QCD calculations*. JHEP **12** (2010), p. 013.

[16] M. E. Peskin and D. V. Schroeder. *An Introduction to Quantum Field Theory*. Westview Press, 1995.

[17] A. Zee. *Quantum Field Theory in a Nutshell*. Princeton University Press, 2010.

[18] R. P. Feynman. *Space-Time Approach to Quantum Electrodynamics*. Phys. Rev. **76**.6 (Sept. 1949), pp. 769–789.

[19] M. L. Mangano and S. J. Parke. *Multi-Parton Amplitudes in Gauge Theories*. Phys. Rept. **200** (1991), pp. 301–367.

[20] R. Kleiss and H. Kuijf. *Multigluon cross sections and 5-jet production at hadron colliders*. Nucl. Phys. **B312**.3 (1989), pp. 616–644.

[21] F. A. Berends and W. T. Giele. *The Six Gluon Process as an Example of Weyl-Van Der Waerden Spinor Calculus*. Nucl. Phys. **B294** (1987), p. 700.

[22] M. L. Mangano, S. J. Parke, and Z. Xu. *Duality and Multi - Gluon Scattering*. Nucl. Phys. **B298** (1988), p. 653.

[23] Z. Bern, J. J. M. Carrasco, and H. Johansson. *New Relations for Gauge-Theory Amplitudes*. Phys.Rev.D **78** (2008), p. 085011.

[24] V. Del Duca, A. Frizzo, and F. Maltoni. *Factorization of tree QCD amplitudes in the high-energy limit and in the collinear limit*. Nucl. Phys. **B568**.1-2 (2000), pp. 211–262.

[25] V. Del Duca, L. Dixon, and F. Maltoni. *New color decompositions for gauge amplitudes at tree and loop level*. Nucl. Phys. **B571**.1-2 (2000), pp. 51–70.

[26] F. Maltoni et al. *Color-flow decomposition of QCD amplitudes*. Phys. Rev. **D67** (2003), p. 014026.

[27] G. 't Hooft. *A Planar Diagram Theory For Strong Interactions*. Nucl. Phys. **B72** (1974), p. 461.

[28] P. De Causmaecker et al. *Multiple bremsstrahlung in gauge theories at high energies (I). General formalism for quantum electrodynamics*. Nucl. Phys. **B206**.1 (1982), pp. 53–60.

[29] J. F. Gunion and Z. Kunszt. *Improved analytic techniques for tree graph calculations and the subprocess*. Phys. Lett. **B161**.4-6 (1985), pp. 333–340.

[30] L. Dixon. *Calculating Scattering Amplitudes Efficiently*. SLAC-PUB-96-7106 (Dec. 1996).

[31] R. Kleiss and W. J. Stirling. *Spinor Techniques for Calculating $p\bar{p} \to W^{\pm}/Z^0$ + Jets*. Nucl. Phys. **B262** (1985), pp. 235–262.

[32] K. Nakamura et al. *Particle data group*. J. Phys. **G37** (2010), p. 075021.

[33] F. A. Berends, P. H. Daverveldt, and R. Kleiss. *Complete Lowest Order Calculations for Four Lepton Final States in electron-Positron Collisions*. Nucl. Phys. **B253** (1985), p. 441.

[34] A. Ballestrero and E. Maina. *A New Method for Helicity Calculations*. Phys. Lett. **B350** (1995), pp. 225–233.

[35] S. Dittmaier. *Weyl-van-der-Waerden formalism for helicity amplitudes of massive particles*. Phys. Rev. **D59** (1999), p. 016007.

[36] J. van der Heide et al. *Helicity amplitudes for single top production*. Phys. Rev. **D62** (2000), p. 074025.

[37] B. L. van der Waerden. *Spinoranalyse*. Göttinger Nachrichten (1929), pp. 100–109.

[38] O. Laporte and G. E. Uhlenbeck. *Application of Spinor Analysis to the Maxwell and Dirac Equations*. Phys. Rev. **37**.11 (June 1931), pp. 1380–1397.

[39] F. A. Berends and W. T. Giele. *Recursive Calculations for Processes with n Gluons*. Nucl. Phys. **B306** (1988), p. 759.

[40] S. Weinzierl. *Introduction to Monte Carlo methods*. NIKHEF-00-012 (June 2000).

[41] G. P. Lepage. *A New Algorithm for Adaptive Multidimensional Integration*. J. Comput. Phys. **27** (1978), p. 192.

[42] M. Dinsdale, M. Ternick, and S. Weinzierl. *A comparison of efficient methods for the computation of Born gluon amplitudes*. JHEP **03** (2006), p. 056.

[43] P. D. Draggiotis, R. Kleiss, and C. G. Papadopoulos. *On the computation of multigluon amplitudes*. Phys. Lett. **B439** (1998), pp. 157–164.

[44] P. D. Draggiotis, R. Kleiss, and C. G. Papadopoulos. *Multi-jet production in hadron collisions*. Eur. Phys. J. **C24** (2002), pp. 447–458.

[45] M. Czakon, C. G. Papadopoulos, and M. Worek. *Polarizing the Dipoles*. JHEP **08** (2009), p. 085.

[46] V. I. Borodulin, R. N. Rogalyov, and S. R. Slabospitsky. *CORE 2.1 (COmpendium of RElations, Version 2.1)*. IHEP-95-90 (July 1995).

[47] R. Gastmans and T. T. Wu. *The ubiquitous photon*. New York, NY (United States); Oxford University Press, 1990.

[48] R. Kleiss, W. J. Stirling, and S. D. Ellis. *A new Monte Carlo treatment of multiparticle phase space at high energies*. Comput. Phys. Commun. **40**.2-3 (1986), pp. 359–373.

Acknowledgements

First and foremost, I would like to thank my advisor Prof. Dr. Stefan Weinzierl for introducing me to this intersting part of particle physics and for always being helpful and supportive throughout the last year.

Furthermore, I would like to thank my fellow student Christopher Schwan for the good times and awesome cooperation during the last years, for many helpful physical discussions and especially for his tireless and neverending help with the Linux operating system, programming languages, libraries, tools, etc. Many thanks also go to my fellow workgroup members Sebastian Becker and Christian Reuschle for many interesting discussions.

Last but not least, I would like to thank my parents for their continuing love and support and Antje for always being there for me, especially during the last stressful weeks — Thank you so much!