Colour factors of Feynman diagrams: the algorithms in the `colour` module

A pedagogical note for a master student

October 2026

For a student who knows the QCD Feynman rules but has never computed a colour factor by machine.

1 What problem are we solving?

Every QCD Feynman diagram factorises into a kinematic part (momenta, polarisations, propagators) and a **colour part**. The colour part is built from three objects only:

- the generators $(T^a)_{ij}$ of $SU(N)$ in the fundamental representation, one per quark–gluon vertex,
- the structure constants f^{abc}, one per triple-gluon or ghost–gluon vertex,
- Kronecker deltas δ_{ij} (quark propagator) and δ^{ab} (gluon propagator).

Indices that appear twice are summed. The result is a number, a polynomial in N, if every index is summed; or a tensor in the external colour indices if some are left open.

Example

The one-loop correction to the quark–gluon vertex has colour factor $T^bT^aT^b$. The answer is

$$T^bT^aT^b = \left(C_F - \frac{C_A}{2}\right)T^a, \qquad C_F = \frac{N^2-1}{2N}, \quad C_A = N.$$

We want a program that finds this, and the analogous result for a two-loop six-gluon diagram with a hundred terms, in a millisecond.

Conventions used throughout

$$[T^a, T^b] = if^{abc}T^c, \qquad \mathrm{Tr}(T^aT^b) = T_F\,\delta^{ab}, \qquad T_F = \tfrac{1}{2}.$$

The order of the three legs in f^{abc} matters because f is antisymmetric. **The colour module reads the legs in the order the diagram gives them and never reorders them.** The kinematic code must use the same order. This one rule removes an entire class of sign errors.

2 The two identities that do all the work

2.1 The Fierz identity

The generators form a basis of traceless Hermitian matrices, and completeness of that basis reads

$$\sum_a (T^a)_{ij}(T^a)_{kl} = T_F\left(\delta_{il}\delta_{kj} - \frac{1}{N}\,\delta_{ij}\delta_{kl}\right). \tag{1}$$

Picture it: a gluon connecting two points on quark lines is replaced by two terms. In the first the quark lines are reconnected crosswise; in the second they stay as they were, with a factor $-1/N$. Every $1/N$ in a colour factor comes from here.

Applied to the example:

$$T^b T^a T^b = T_F \left(\mathrm{Tr}(T^a)\not{\mathbb{1}} - \frac{1}{N} T^a \right) = -\frac{T_F}{N} T^a = \left(C_F - \frac{C_A}{2} \right) T^a .$$

2.2 Structure constants as traces

Multiply the commutator by T^c and take the trace:

$$f^{abc} = \frac{-i}{T_F} \Big[\mathrm{Tr}(T^a T^b T^c) - \mathrm{Tr}(T^a T^c T^b) \Big]. \tag{2}$$

So every f can be turned into quark-like traces, after which only Fierz is needed. The price is a factor two in the number of terms for every f.

2.3 Two trivial facts

$$\mathrm{Tr}(\not{\mathbb{1}}) = N, \qquad \mathrm{Tr}(T^a) = 0.$$

The first turns a closed quark loop with nothing on it into a factor N. The second kills any trace with a single generator.

2.4 Bookkeeping of the factor i

Each f brings a factor $-i$. We never carry complex numbers; we count the number n_f of structure constants and remember that the answer carries an overall $(-i)^{n_f}$. Internally we work with $\hat{F} \equiv if$, which is real in the trace representation (2).

3 The reference engine (`colour.hpp`): brute-force Fierz

This is the simplest correct algorithm and the one to understand first.

1. Replace every f (and d) by its two traces. For n_f structure constants this gives 2^{n_f} terms.
2. In each term, follow the quark indices to form **lines**: open strings $(T^{a_1} \cdots T^{a_k})_{ij}$ and closed traces $\mathrm{Tr}(T^{a_1} \cdots T^{a_k})$.
3. Pick a gluon label that appears twice on the lines and apply Fierz (1). Two new terms. Recurse.
4. Stop when no internal label is left. What remains is a product of traces and strings in the external indices only: a **residual structure**. Add the accumulated coefficient to that structure.

The output is a dictionary: residual structure $\mapsto$ polynomial in N and $1/N$ (a Laurent polynomial). This is the **trace basis** of colour space.

Two refinements already matter here. When a label appears twice on the *same* line, apply Fierz there first: it splits the line and creates a closed loop that often becomes N or 0 at once. And **memoisation**: the same intermediate state appears many times in the recursion tree; keyed on a textual form of the state, a cache turns repeated work into a lookup.

Arithmetic is exact: coefficients are rationals (a small 128-bit implementation; GMP can be swapped in). Nothing is ever a floating-point number.

4 The numerical oracle (`numeric()` and `colour_numeric.hpp`)

A symbolic engine must be tested against something that shares none of its code. Two independent checks are provided.

Brute force with explicit matrices. Build the $N^2 - 1$ generalised Gell-Mann matrices for $N = 2, 3, 4$, sum over every internal adjoint index explicitly, compare numbers. Exponential cost, fine for tests.

Tensor-network contraction with exact interpolation. A complete alternative algorithm, worth understanding on its own:

1. *Colour flow.* Replace every adjoint index a by a pair of fundamental indices (p,q): $T^a_{ij} \to \delta_{ip}\delta_{qj}$, and the gluon propagator becomes the Fierz tensor $T_F(\delta\delta - \delta\delta/N)$. Every object is now a sum of products of deltas with coefficients $\pm 1, \pm 1/N$. No square roots, so everything is exact at integer N.
2. *Contract* the network at $N = 2, 3, 4, \ldots$ with sparse tensors, choosing greedily which pair to contract next (smallest intermediate).
3. *Read off* the coefficient of each delta-permutation structure in the external indices by evaluating at $\text{out}_k = k$, $\text{in}_{\sigma(k)} = k$.
4. *Interpolate* in N: multiply by N^K to clear poles (K = number of gluon propagators), Newton interpolation, stop when two further points are predicted correctly.
5. *Translate* delta structures back to traces by following the chains.

This engine never uses Fierz as a rule, never expands f, and never canonicalises anything. It agrees with the symbolic engines on every test, which is the point.

5 Reducing the work: birdtrack rules before expansion (`colour_fast.hpp`)

Expanding every f is wasteful when the f's form small closed structures. Three identities remove them without expansion. In terms of $\hat{F} = if$:

$$\text{gluon bubble} \qquad \hat{F}^{acd}\hat{F}^{bcd} = -N\,\delta^{ab} \tag{3}$$

$$\text{gluon triangle} \qquad \hat{F}^{ade}\hat{F}^{bef}\hat{F}^{cfd} = -\tfrac{N}{2}\hat{F}^{abc} \tag{4}$$

$$\text{vertex correction} \qquad \hat{F}^{abc}T^bT^c = -\tfrac{N}{2}T^a \quad (b,c \text{ adjacent on a line}) \tag{5}$$

$$\text{tadpole} \qquad \hat{F}^{abb} = 0 \tag{6}$$

These are the simplest **birdtrack** rules (Cvitanović). Draw the f's as three-pronged vertices and the identities are literally "a bubble is a propagator times N", "a triangle is a vertex times $N/2$".

A sign detail worth doing once by hand

In the bubble, rotate each f so the unshared leg comes first; the sign is $+$ if the two shared legs appear in the same order in both, $-$ otherwise. The triangle needs the same care with three vertices; the code tries all orientations and uses antisymmetry to find the one that matches.

Only an f that no rule removes is expanded, and then one attached to a quark line, so the new trace merges with an existing line immediately.

5.1 Canonical forms: why the cache needs them

Two intermediate states that differ only in the names of internal labels are the same state. A cache keyed on raw labels misses them. A **canonical form** is a relabelling that is identical for all isomorphic states.

The algorithm: start from the external labels in name order; pop a label, emit the objects containing it (rotated so that label comes first), assign fresh numbers to unseen internal labels in order of appearance, push them on the queue. Whenever a step is ambiguous (two objects contain the same label, a trace contains it twice, a new disconnected component must be started) branch over the choices and keep the lexicographically smallest emission string, pruning branches already larger than the best found. For graphs of this size the branching is shallow. This is a small version of what NAUTY does for general graphs; here it is cheap because external labels anchor the traversal.

6 FORM's model of computation, and why it is fast

FORM processes an expression as a *list of terms*. A module applies one substitution to every term, then **sorts** the list and adds terms with identical structure. Three consequences:

1. Identical intermediate terms merge automatically at each sort: memoisation without a cache.
2. Dummy indices are renumbered into a fixed order after each step, so equal structures sort together. For products of deltas and strings of generators this is already a canonical form, obtained for free.
3. Terms are flat arrays of small integers and pattern matching is a tight C loop. No heap allocation per object.

Measured against this module: FORM's shipped $SU(N)$ routine (`SUn.prc`) needs about 14 ms for a two-loop six-gluon graph with open indices and 2 ms when fully contracted. Our recursive engines were comparable or slower. The lesson was to adopt FORM's processing model and add what FORM cannot do.

7 The hybrid engine (`colour_v2.hpp`)

7.1 Flat terms and breadth-first levels

A term is a vector of fixed-size POD objects: type (trace, string, f), length, two end indices, up to 24 labels. Copying is a `memcpy`. All terms of the current depth are processed together: each takes one reduction step, then the list is renumbered, sorted and merged. This is FORM's `.sort`.

7.2 One more rule: the commutator

When an f shares a single label with a quark line, use

$$\hat{F}^{abc}\,T^c = T^a T^b - T^b T^a. \tag{7}$$

It still gives two terms, but each stays a single line rather than a line plus a separate three-generator trace, so the terms merge much earlier with others. In pure-gluon graphs this rule is what defeats the 2^{n_f} growth: the first expansion creates a line, and every subsequent f is absorbed into it by the commutator rule.

7.3 Factorise into components, cache the components

When a term's colour graph is disconnected the factor splits: each connected component is reduced independently and the results are multiplied. Each component is canonicalised (Section 5.1) and cached. The cache lives in the engine object, so **all diagrams of an amplitude share it**. Sub-structures such as "a gluon bubble with two external legs attached in this way" recur in hundreds of diagrams and are computed once.

7.4 Measured

case	time per diagram
two-loop six-gluon ring, open indices	1.8 ms (FORM `SUn.prc`: 14 ms)
same, fully contracted	0.19 ms (FORM: 2 ms)
all 2280 two-loop five-gluon cubic graphs, one engine	0.10 ms each, 23 810 cache hits

All results agree with the reference engine, the numerical oracle, and (where applicable) FORM.

8 What a student should take away

1. Two identities, Fierz and f-as-trace, are sufficient. Everything else is about doing fewer operations.
2. Exact rational arithmetic is non-negotiable; floating point hides sign and $1/N$ errors.
3. Test against an algorithm that shares no code with the one under test. The tensor-network oracle caught two real bugs during development (an index-name clash in a FORM script, and a key-normalisation bug in the engines).
4. The expensive part is not the algebra but the combinatorics: how many intermediate terms exist and how early identical ones merge. Rules that keep terms connected (commutator, birdtracks) beat rules that split them (naive expansion).
5. Canonical forms are the bridge between "equal as mathematics" and "equal as bytes". Cheap ones (dummy renumbering) go a long way; exact ones are needed only where you cache.
6. Signs live in leg ordering. Make the colour code and the kinematic code read the same ordered list, and the problem disappears.

9 Where to read more

- P. Cvitanović, *Group Theory: Birdtracks, Lie's and Exceptional Groups* (free online) — the diagrammatic rules of Section 5.
- T. van Ritbergen, A. N. Schellekens, J. A. M. Vermaseren, *Group theory factors for Feynman diagrams*, Int. J. Mod. Phys. A14 (1999) 41 — the algorithm behind FORM's `color.h`, for arbitrary groups.
- J. A. M. Vermaseren, *FORM reference manual* — the term-and-sort model of Section 6.
- M. Sjödahl, *ColorMath*; S. Keppeler, M. Sjödahl, *ColorFull* — other implementations with different basis choices.
- V. Del Duca, L. Dixon, F. Maltoni, *New colour decompositions for gauge amplitudes at tree and loop level*, Nucl. Phys. B571 (2000) 51 — the DDM basis, the natural next step for multi-leg amplitudes.