Showers MC's: where to

Revival of theoretical efforts on improving SMC's

- Matrix element corrections (Seymour+Corcella; CKKW, etc.)
- NLO accuracy: first implementation (Frixione, Webber);
  ideas for alternatives (Soper, M. Krämer, Nagy, P.N.);
- NLL showers (Collins)

and new codes

- c++ rewriting of Herwig and Pythia
- Sherpa

Why

Emphasis on QCD tests diminished (after LEP, Tevatron);

- A test phase needs simplicity: unambiguous fixed order results
- Many ingredients in SMC's difficult to control (biases?).

More focus on applications;

- For applications: exclusive final states more natural
- Fixed order + SMC easier to understand.

Until now the numerics of fixed order QCD work where happily carried out
with FORTRAN programs, using of CERNLIB, VEGAS etc.
QUESTION:
in what framework will the SMC improved NLO or even NNLO and NLL be carried out?

My experience: New method for SMC+NLO

from NLO side needs

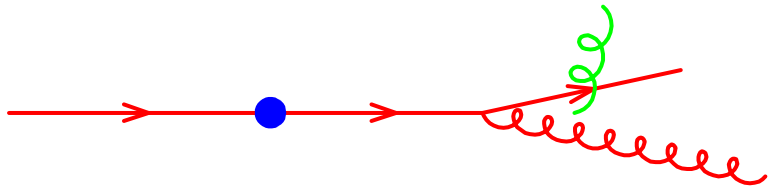
$$\left(B + V + \int_{\text{rad}} [R - C]\right) \exp\left\{-\int_{\text{rad}} R/B\right\}$$

for the generation of the NLO emission (Positive!).

From shower side needs:

- Transverse momentum vetoed angular ordered showers: $p_T <$ NLO p_T.
- Truncated Showers from pairs of partons.

  Soft gluons at angles $>$ NLO angle coherently emitted from pair of partons: shower truncated to NLO angle.



Ready to give up Truncated Showers, just give me vetoed Showers...

Attempts to implement shower requirements in HERWIG failed

- Discouraged to try to use FORTRAN code (too messy, bound to halt developments)
- Unable to understand c++ code (`ThePEG` in `Herwig++`)
- Unable to get suitable help from developers (too busy with more severe problems)

So! roll your own ...

- Learn some c++: doable with modest effort;
- Find an environment (i.e. libraries to use, work-style, etc.)
- Code a shower program

What I learned

- c++ not worse than FORTRAN; can link to older FORTRAN codes,
  offers interesting features: used ONLY what I needed.
- Found useful items in CLHEP: geometry and relativity, fair documentation.
- Useful items in gnu scientific library: random numbers, histograms, integration,
  wonderful documentation. Even VEGAS implemented there. Not c++ but OO style
  (straightforward to interface to c++).
- Great stuff in the c++ standard library (used to be STL).
  Handles all sort of containers, lists, etc..
- Tried to study ThePEG: no success.
- Borrowed code from friendly people (S. Gieseke for shape variable computation)

Not difficult to produce a working (final state for now) shower algorithm,
with veto options and truncated showers (1 month full time). But:

- Is this the way to go?
- Are these the tools (CLHEP, gsl, etc.) people will use?

```
void Parton::reconstruct () {
  if(FirstChild == NULL) { // No children;
    double m = phpars.partonMass(flavour);
    double p = sqrt(EnMom*EnMom-m*m);
    k.set(EnMom,0,0,p);    // on-shell vector along x
    // syntax is k.set(x,y,z,t)
  }
  else { // If we have children, reconstruct them first
    FirstChild->reconstruct();
    SecondChild->reconstruct();
    // assume that t=(1-cos theta)*E^2
    double CosTheta=1-OrderingVariable/pow(EnMom,2);
    if(CosTheta>1) CosTheta=1;
    if(CosTheta<-1) CosTheta=-1;
    // Length of total
    double v1 = FirstChild->k.getX();
    double v2 = SecondChild->k.getX();
    double v = sqrt(v1*v1 + v2*v2 + 2*CosTheta*v1*v2);
    double CosTheta1 = -(v2*v2-v1*v1-v*v)/(2*v1*v);
    double CosTheta2 = -(v1*v1-v2*v2-v*v)/(2*v2*v);
    double theta1=acos(CosTheta1);
    double theta2=acos(CosTheta2);
    Hep3Vector u(0,cos(Azimuth),sin(Azimuth));
    FirstChild->rotate(theta1,u);
    SecondChild->rotate(-theta2,u);
    k = FirstChild->k + SecondChild->k;
  }
}
```

```
Parton* FollowQuarkColour  (Parton& father,
                            list< pair<Parton*,Parton*> > & clusters) {
  if(father.FirstChild == NULL) return &father;
  else {
    Parton* gluon;
    Parton* quark;
    if(father.FirstChild->type == QUARK) {
      gluon = father.SecondChild;
      quark = father.FirstChild;
    } else {
      gluon = father.FirstChild;
      quark = father.SecondChild;
    }
    pair<Parton*,Parton*> gl=FollowGluonColour( *gluon, clusters);
    Parton* qrk=FollowQuarkColour(*quark,clusters);
    if(father.flavour>0) {
      clusters.push_front(make_pair(qrk, gl.second));
      return gl.first;
    } else {
      clusters.push_front(make_pair(gl.first, qrk));
      return gl.second;
    }
  }
}
```

```
pair<Parton*, Parton*> FollowGluonColour(Parton& father,
                                         list< pair<Parton*,Parton*> > & clusters) {
  if(father.FirstChild == NULL) {
    cout << "Gluon should have showered!" ;
    exit(1);
  }
  if(father.FirstChild->type == QUARK) {
    if(father.FirstChild->flavour>0) // first colour, second anticolour
      return make_pair(FollowQuarkColour( *(father.FirstChild), clusters),
                       FollowQuarkColour( *(father.SecondChild), clusters));
    else
      return make_pair(FollowQuarkColour( *(father.SecondChild), clusters),
                       FollowQuarkColour( *(father.FirstChild), clusters));
  }
  else { // Children can only be gluons
    pair<Parton*,Parton*> gl1=FollowGluonColour( *(father.FirstChild), clusters );
    pair<Parton*,Parton*> gl2=FollowGluonColour( *(father.SecondChild), clusters );
    clusters.push_front(make_pair(gl2.first,gl1.second));
    return make_pair(gl1.first,gl2.second);
  }
}
```

Readable code (well.. if you know c++, the Standard Library, the Vectors in CLHEP).
Borrowed pieces of code using the same stuff are easily reusable.

Large Class Libraries for large SMC programs, using their own class library may become quickly unaccessible (there is never time to write documentation).
Also, more than one framework at the end! (`ThePEG`, `PYTHIA++`, SHERPA...)

So:

- TH developments in SMC physics would benefit from easy access to SMC codes
- New SMC code developments could make this more difficult: overhead for learning new language plus need of expert assistance for learning new code
- c++ code does not necessarily lead to easily modifiable programs
- Easy to use a c++ framework for proof of concepts in SMC physics
- Very difficult instead to perform same task using existing SMC codes without expert's assistance. (unlikely that this will improve...)

Wishes

- Agreement on minimal environment in c++ for TH-Pheno work;
  could ease code exchange among theorists (much better than FORTRAN).
- Documented hooks to jump into SMC's (there were some in old FORTRAN programs), using minimal environment. Where to jump: collision, minimum bias, hard event, shower, hadronization, decays.
- Standards for some kind of objects (Partons? Showers? Hadrons?) to help newcomers to "read" SMC codes.