xdvmpGenerator

An Monte Carlo Generator for Exclusive Diffractive Vector Meson Production

Status of the implementation of the b-Sat/b-CGC Model for ep and eA

Thomas Ullrich

January 21, 2010

BROOKHAVEN NATIONAL LABORATORY

Motivation

Exclusive diffractive vector meson production is one of the most promising ways to study saturation in ep/eA

- Naive: $\sigma \sim G(x,Q^2)^2$

Issues:

- Experimentally difficult
  - rapidity gap, breakup, $\int L dt$ needed ?
  - reconstruction of t
  - detector requirements (resolution, acceptance)
  - sensitivity to physics (saturation)?
  - need to study in ep and eA

What's on the market?

RAPGAP

- only ep
- buggy ($t = (p-p')^2 \neq (p_{\gamma^*} - p_V)^2$, $p_z' > E'$, etc.)
- cannot run $J/\psi \Rightarrow$ need to add extra program
- hard to manipulate (see the code)
- in FORTRAN (cumbersome integration with ROOT and other tools)

Other ?

- None with the features we want

Requirements for a new generator

- Simple, i.e. easy to use, manipulate and modify
  - single purpose: e p → e' p' V
  - write only the necessary core
  - reuse what is available (and accessible)
- Based on a model that is known to describe data well
  - Dipole model (works well at Hera)
- Extendable to eA
  - Dipole model does that
- Modern
  - C++, integrates with ROOT and other tools
- Output should follow standards as much as possible
- Useful for detector/acceptance studies as well as physics studies (e.g., sensitivity to $G(x,Q^2)$ etc.)

Dipole Model (I)

Cross-section for production of final state VM:

$$\frac{\mathrm{d}\sigma^{\gamma^* p\to Ep}_{T,L}}{\mathrm{d}t} = \frac{1}{16\pi}\left|\mathcal{A}^{\gamma^* p\to Ep}_{T,L}\right|^2 = \frac{1}{16\pi}\left|\int \mathrm{d}^2\boldsymbol{r}\int_0^1 \frac{\mathrm{d}z}{4\pi}\int \mathrm{d}^2\boldsymbol{b}\,(\Psi_E^*\Psi)_{T,L}\,\mathrm{e}^{-\mathrm{i}[\boldsymbol{b}-(1-z)\boldsymbol{r}]\cdot\boldsymbol{\Delta}}\frac{\mathrm{d}\sigma_{q\bar{q}}}{\mathrm{d}^2\boldsymbol{b}}\right|^2$$

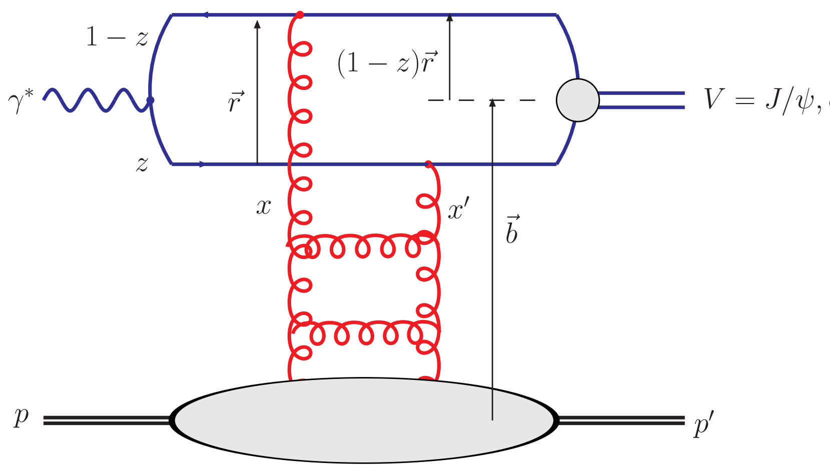
Amplitude

Overlap between photon and VM wave function

Dipole Cross-Section

Many dipole models on the market:

- Use : H. Kowalski, L. Motyka, G. Watt, Phys. Rev. D74, 074016
- Describes Hera data well
- has b-dependence
- Michael & TU have experience with it
- Henri is around to ask
- can be “easily” modified to do eA (via b-dependence)



Using it implies the generator has to be amplitude based (which is a bit problematic)

Dipole Model (II)

Cross-section for production of final state VM:

$$\frac{\mathrm{d}\sigma_{T,L}^{\gamma^* p \to Ep}}{\mathrm{d}t} = \frac{1}{16\pi}\left|\mathcal{A}_{T,L}^{\gamma^* p\to Ep}\right|^2 = \frac{1}{16\pi}\left|\int \mathrm{d}^2\boldsymbol{r}\int_0^1 \frac{\mathrm{d}z}{4\pi}\int \mathrm{d}^2\boldsymbol{b}\,(\Psi_E^*\Psi)_{T,L}\,\mathrm{e}^{-\mathrm{i}[\boldsymbol{b}-(1-z)\boldsymbol{r}]\cdot\boldsymbol{\Delta}}\frac{\mathrm{d}\sigma_{q\bar{q}}}{\mathrm{d}^2\boldsymbol{b}}\right|^2$$

Overlap between photon and VM wave function

Dipole Cross-Section

Wave function:

- Boosted Gaussian
  - Forshaw, Sandapen, Shaw
- GausLC
  - Dosch, Gousset, Kulzinger, Pirner, Teaney, Kowalski
- Parameters tuned for HERA are available
- any improved wave function can be easily plugged in

Dipole Cross-Section:

- b-Sat
  - uses DGLAP evolution from initial G(x,Q)
  - can be adapted for A (b-dependence)
- b-CGC
- Parameters tuned for HERA are available

Photon Flux

Dipole models provide $\sigma_{L,T}$ (γ^* p $\rightarrow$ p' V)

For generator we need to consider σ (e p $\rightarrow$ e' p' V)

Need Photon Flux Γ_T , Γ_L

$\sigma^{e\,p \rightarrow e'\,p'\,V} = \Gamma_L\, \sigma_L^{\gamma^*\,p \rightarrow p'\,V} + \Gamma_T\, \sigma_T^{\gamma^*\,p \rightarrow p'\,V}$

The full formula is rather complex

What is used is a simplification (not always justified):

For $Q^2/(4E^2) = 0$ and $Q^2/\nu^2 = 0$, $m_e = 0$

Pick 2 independent variables best for MC: x, Q^2

$$\frac{d^2\sigma}{dx\,dQ^2} = \frac{\alpha}{2\pi}\frac{1}{xQ^2}\left(\left[1+(1-y)^2 - 2(1-y)\frac{Q^2_{min}}{Q^2}\right]\sigma_T + 2(1-y)\sigma_L\right)$$

where $Q^2_{min} = \dfrac{m_e^2 y^2}{1-y}$

Jacobian!

Full Shebang ...

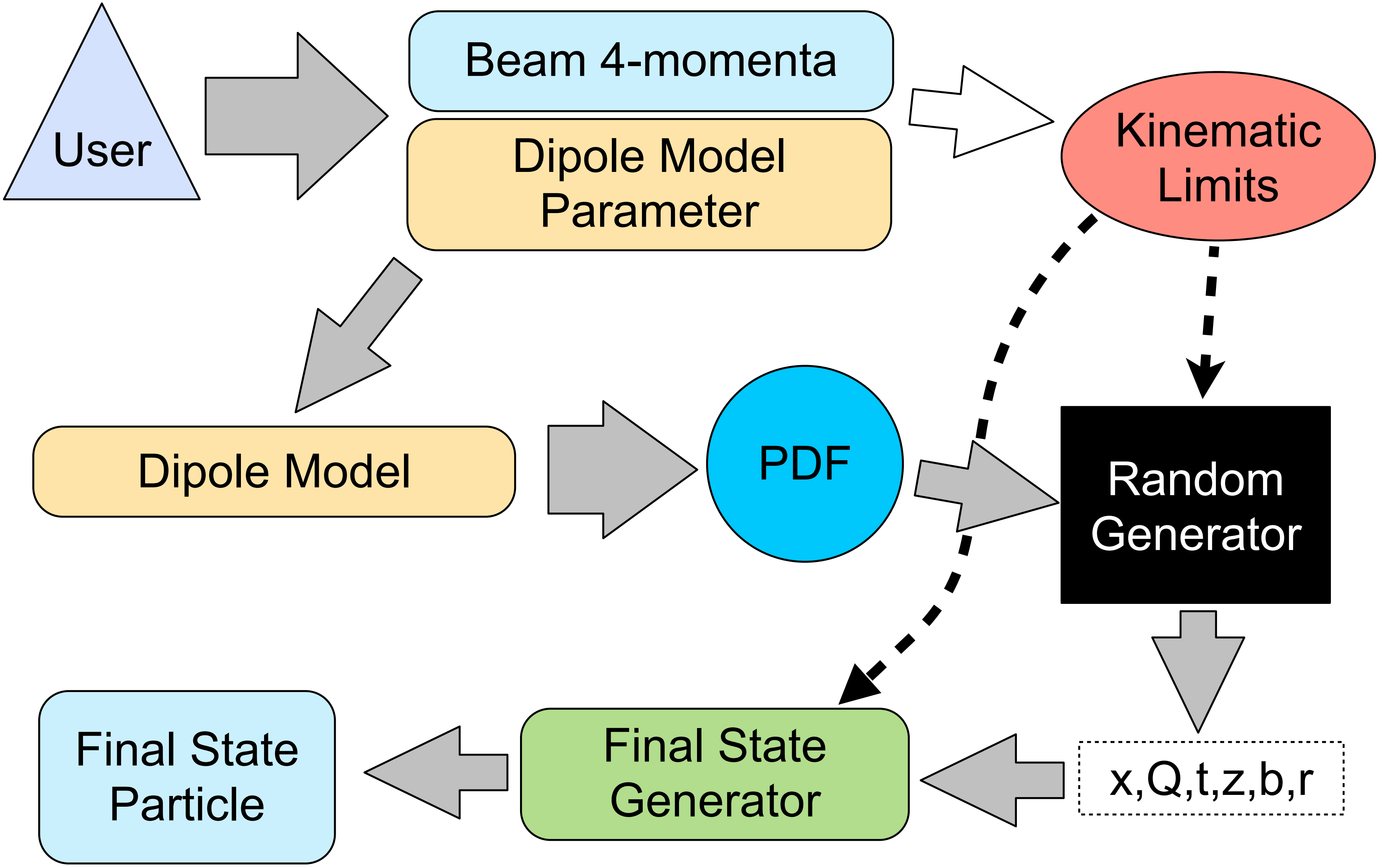
Dipole model calculations + flux give:

$$\frac{d^6\sigma}{dx\ dQ^2\ dt\ db\ dz\ dr}$$

- 6-dim Probability Distribution Function (PDF)
- all variables independent

- Given (input): beam energies **p**e, **p**p

Basic scheme behind *xdvmpGenerator*



Random Generator

Big Problem: generate random numbers according to a given distribution (here 6D PDF)

Techniques (good overview in Pythia6 manual chapter 4):

- Inverse transform method (invert cumulative PDF)
  - must integrate pdf and invert (note we have a DGLAP evolution in the PDF)
- Acceptance-rejection method (Von Neumann)
  - good if pdf is too complex
  - rather easy in 1-D, nightmare in N-D
- and many more
- General recommendation in all text books for N-dim: factorize
  - Problem is we cannot do that since the 6 parameters are heavily intertwined
- Largest fraction of code in most simulators is spent on this topic

UNURAN to the rescue (http://statmath.wu.ac.at/unuran/)

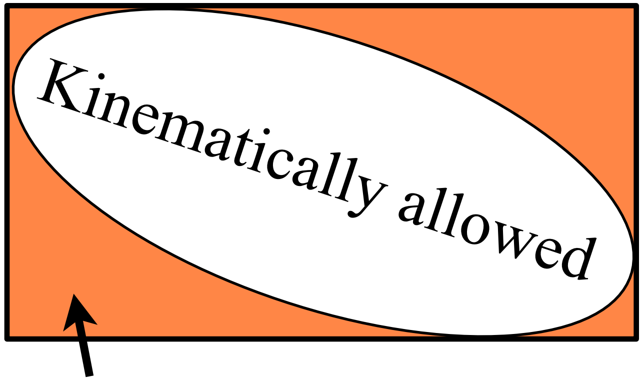
UNU.RAN Package

Universal Non-Uniform RANdom number generators
(Math Department University Vienna)

- provides tools to generate pretty much everything
- xdvmpGenerator:
  - Markov chain samplers for continuous multivariate distributions
  - HITRO: Hit-and-Run Sampler
- Bare minimum is implemented in Root/MathCore

Issues:

Requires uniform limits (domains)



Kinematically not allowed but generated
Need to discard after generation (tries > events)

Requires to pass mode of pdf to UNURAN

- pdf is max at $|t| = |t|_{min}$, $x=x_{min}$, $Q=Q_{min}$
- less obvious for b, z, r

Use MINUIT (TMinuit2)

Final State Particles

Given: $\mathbf{p}_e$, $\mathbf{p}_p$, s, t, x, Q^2, y

Need: $\mathbf{p}_{e'}$, $\mathbf{p}_{p'}$, $\mathbf{p}_{\gamma^*}$, $\mathbf{p}_{VM}$

Hannes Jung (DESY) gave me analytic solutions for all. After many checks: $\mathbf{p}_{e'}$, $\mathbf{p}_{\gamma^*}$ formulas are correct!

$\mathbf{p}_{p'}$ is not correct (possible source of problems in RAPGAP?)

New Ansatz:

- $t = (\mathbf{p}-\mathbf{p}')^2$, $m_{VM}^2 = (\mathbf{p}_{\gamma^*} + \mathbf{p}_p - \mathbf{p}_{p'})^2$, $|\mathbf{p}_{p'}| = m_p$
- allows to derive $\mathbf{p}_p$ numerically (root finder)
- use Hanne's analytic formula as first guess
  - fails at times since first guess is off by several GeV
- $\mathbf{p}_{VM}$ trough $\mathbf{p}_e + \mathbf{p}_p = \mathbf{p}_{e'} + \mathbf{p}_{p'} + \mathbf{p}_{VM}$
- solution obtained this way is fully consistent
  - $\mathbf{p}_{e'}$, $\mathbf{p}_{p'}$, $\mathbf{p}_{\gamma^*}$, $\mathbf{p}_{VM}$ $\Rightarrow$ s, t, x, Q^2, y

Kinematic Boundaries

Tricky since some formulas neglect masses others not (something to still work on)

$$s = \frac{Q^2}{xy} + m_p^2 + m_e^2$$

not just $Q^2 = s\,x\,y$

Currently implemented (but not sufficient):

$$0 < x < 1$$

$$0 < y < 1$$

$$\frac{m_e^2 y^2}{1-y} < Q^2 < s - m_e^2 - m_p^2$$

$$x_{IP} = \frac{m_V^2 + Q^2}{ys}$$

$$t < -\frac{x_{IP}^2 m_p^2}{1 - x_{IP}}$$

Implementation

Follow Pythia8 philosophy

- main program to be provided by user
- *xdvmpGenerator* is class with simple methods
  - init(), generateEvent(), printEventRecord(), ...
  - event record in plain structure (*xdvmpEvent*)
  - setup through runcard (txt file) or programmatically
- *xdvmpGenerator* uses many other classes and functions
  - class *xdvmpDipoleModel* (dipole model implementation)
    - alphaStrong.cpp (fcts to calculate α_s - adapted from MRST, rewritten in C++)
    - laguerre.c, dglap.c (for DGLAP from F. Gelis)
  - class *xdvmpFinalStateGenerator* (generate final state particles from *x*, Q^2, *s*, *t*)
  - class *xdvmpSettings* (handle parameter & runcard parsing)
- Total ~ 4200 lines of code only (requires only GSL, ROOT libs)

Example Main Program

```
#include "xdvmpGenerator.h"

int main(int argc, char *argv[])
{
    xdvmpGenerator generator;
    bool ok = generator.init(“xdvmpRuncard.txt”);
    xdvmpSettings settings = generator.runSettings(); // for convinience
    TFile *hfile = new TFile(settings.rootfile().c_str(),"RECREATE");
    TH1D *histo_r = new TH1D("histo_r", "r distribution", 200, 0., 2.);

    int nPrint = settings.numberOfEvents()/settings.timesToShow();
    unsigned long maxEvents = settings.numberOfEvents();

    generator.printEventHeader(cout);

    for (unsigned long iEvent = 0; iEvent < maxEvents; iEvent++) {
        xdvmpEvent event = generator.generateEvent();
        if (iEvent%nPrint == 0) {
            cout << "processed " << iEvent << " events" << endl;
        }
        histo_r->Fill(event.r);
        generator.printEventRecord(cout);
    }
    hfile->Write();
    generator.printStatistics();
    return 0;
}
```

Example Runcard

```
#==========================================================================
#   Comments start with a #
#   Name and value are separated by a "=":  name = value
#
#   The following settings are currently implemented:
#   eBeamEnergy:      electron beam energy (GeV)   (default = 10)
#   pBeamEnergy:      proton beam energy in (GeV)  (default = 250)
#   numberOfEvents:   number of events to generate (default = 10000)
#   vectorMeson:      rho | phi | jpsi  (default = rho)
#   waveFunction:     GausLC | BoostedGaussian (default = BoostedGaussian)
#   dipoleModel:      bSat | bCGC (default = bCGC)
#   timesToShow:      # of print-outs to tell how far we are (default=0)
#   rootfile:         name of root file for histos etc. (default ="")
#   xmin:             min x value (default = 1e-3)
#   Q2min:            min Q2 value (GeV^2)  (default = 1.)
#==========================================================================
eBeamEnergy = 10
pBeamEnergy = 250
vectorMeson = rho
dipoleModel = bSat
waveFunction = BoostedGaussian
numberOfEvents = 10000
timesToShow = 10;
rootfile = bla.root
Q2min = 1;
xmin = 1e-3;
```

Example Output (1)

```
#==================================================================
#
#   xdvmGenerator
#
#   An event generator for exclusive diffractive vector meson
#   production using the dipole model.
#
#   Code compiled on Jan 20 2010  16:50:46
#==================================================================
Run started at Wed Jan 20 23:22:34 2010
Runcard is 'xdvmpRuncard.txt'
mXmin = 0.001
Electron beam is: 0     0       -10     10      (0.000510999)
Proton beam is:   0     0       249.998 250     (0.93827)
sqrt(s) = 100.004
Initializing the xdvmp dipole model:
Vector meson to generate: rho
Dipole model used: bCGC
Wave function used: BoostedGaussian
```

Example Output (2)

```
Range of kinematic variables (domain) used in generator:
t = [-4, 0]
Q = [1, 100.004]
x = [0.001, 0.99]
b = [0, 2]
z = [1e-12, 1]
r = [0.001, 2]
Finding mode of pdf:
mode = (t=0, Q=1, x=0.001, b=0.453883, z=0.5, r=0.526119; value of
pdf = 107769)
Initializing the random generator:
Dimensions used: 6
pdf in log: no
Number of events to process: 10000
xdvmpGenerator is initialized.
```

For bCGC this takes < 1 s

For bSat ~ 1-2 min (due to DGLAP setup)

Example Event Record

```
xdvmpGen event file
============================================
iEvent, t, Q2, x, y, b, z, r, s
============================================
i, id, px, py, px, E, m, vx, vy, vz
============================================
processed 0 events
1       -0.171395       2.03611         0.00254752      0.0799258       0.525637        0.380722        0.344587        10000.8
============================================
1            11              0              0            -10             10    0.000510999             0              0              0
2          2212              0              0        249.998            250        0.93827              0              0              0
3            11    -0.00222092        1.36871       -9.14977        9.25157    0.000510999             0              0              0
4          2212       0.214882       0.352692        248.818         248.82        0.93827              0              0              0
5           113      -0.212661        -1.7214        0.33036        1.92867          0.776              0              0              0
============= End Event Record =============
2       -0.171395       2.03611         0.00254752      0.0799258       0.525637        0.380722        0.554715        10000.8
============================================
1            11              0              0            -10             10    0.000510999             0              0              0
2          2212              0              0        249.998            250        0.93827              0              0              0
3            11        1.34006      -0.278549       -9.14977        9.25157    0.000510999             0              0              0
4          2212       0.390437      -0.134496        248.769        248.771        0.93827              0              0              0
5           113        -1.7305       0.413045       0.379414        1.97772          0.776              0              0              0
============= End Event Record =============
```

- Note the VM does not decay (Geant can do this if needed)
- VM have zero width (should probably change that)
- The event record can be directly written into a ROOT Tree or any other format, the print-out shown here is optional
- Time to generate 1M events ~ 4 min on my 3y old MacBook Pro

To-Do List

- Improve kinematic limit checks
  - See still NaN in event record
- Calculate total cross-section within given limits
  - needed to normalize output and get “barns”
  - comes at a high price (6D integration takes time)
- Print-out format to follow that of other generators
- Test, test, test
- Add eA
  - how do kinematic limits change here

Could need volunteers that help to check, test, and improve ... anyone?