Report INP No. 1504/PH

SIMULATION OF THE SPACE-TIME EVOLUTION OF COLOR-FLUX TUBES

(guidelines to the TERMITE program)*

Andrzej DYREK[1], Wojciech FLORKOWSKI[2]

[1] Institute of Physics, Jagellonian University
Reymonta 4, 30-059 Kraków, Poland

[2] H. Niewodniczański Institute of Nuclear Physics
Radzikowskiego 152, 31-342 Kraków, Poland

Abstract:

We give the description of the computer program which simulates boost-invariant evolution of color flux tubes in high-energy processes. The program provides a graphic demonstration of space-time trajectories of created particles and can also be used as Monte-Carlo generator of events.

August 1990

* Supported in part by CPBP Grants 01.03 and 01.09

1. Introduction

In the present paper we are going to give the details of the TERMITE simulation program which describes the space-time evolution of color-flux tubes including their decays due to the Schwinger tunneling mechanism [1]. Such color-flux tubes are believed to be created in high-energy interactions: e^+e^- annihilations, heavy-ion collisions [2,3].

The program is a practical realization of the model formulated in collaboration with A. Białas and W. Czyż [4]. A distinguishing feature of this model is that all physical processes are explicitly boost-invariant, therefore we expect that it works well in the limit of very high energies.

In our approach quarks (antiquarks) are regarded as classical particles with well defined positions and momenta. They carry color charges which are the source of chromoelectric field. In the space between two color charges the field is uniform and confined to a tube. The value of the field is quantized: it is the straightforward result of application of the Gauss law to the system.

Quarks (antiquarks) move along the trajectories determined by the classical equations of motion. The chromoelectric field decays through the Schwinger mechanism generating partons (we will use this name for both quarks and antiquarks). It leads to the conversion of the energy and momentum of the field into the energy and momentum of particles.

Finally, we obtain a self-consistent system: on the one hand the field creates particles and causes their motion, on the other hand the color charges of partons determine the value of the field.

Recently we have used our program in order to investigate the connection between intermittency and the Schwinger tunneling mechanism. First results [5], exhibiting an intermittency pattern, encouraged us to continue the work with the program. So far we have taken into account only elementary color-flux tubes, *i.e.*

tubes spanned by quark-antiquark pairs. In the future we are going to develop the program and describe the evolution of stronger (non-elementary) color fields.

The program TERMITE is written in the Turbo Pascal language (we have used the version 5.0 but with small changes the program can be also compiled with the version 4.0). TERMITE will run on every IBM PC, optionally supported by an arithmetic coprocessor. It includes a graphic demonstration module which allows the User to watch on-line the evolution of the system. The results of the Monte-Carlo simulation can be stored on a disk and later processed by additional programs.

In the next Section we present the model. We discuss the dynamics of partons, the initial conditions for the evolution, formation and interactions of yo-yo's and tunneling of virtual particles. Section 3 gives the description of the main modules of TERMITE and its hardware and compiler requirements. The Summary and the Appendix containing the listing complete the paper.

2. General description of the model

a) Dynamics of partons

We consider a 1+1 dimensional model. The space-time positions of partons are given by the vectors $x^\mu = (t, z)$. The energy and the momentum of a parton, $p^\mu = (E, p_L)$, satisfy the condition $E^2 - p_L^2 = m_T^2$, where $m_T = \sqrt{m^2 + p_T^2} = \sqrt{m^2 + p_x^2 + p_y^2}$ is the effective transverse mass. One can notice that the transverse momentum gives only the contribution to the total inertia of a parton, it is not a dynamical variable. The quantity m is the rest mass of a quark.

In practice, we prefer to use the variables which have more convenient transformation properties under the Lorentz boosts along the z axis. Therefore we introduce the rapidity and the quasirapidity of a parton

$$y = \frac{1}{2} \ln \frac{E + p_L}{E - p_L}, \tag{2a.1}$$

$$\eta = \frac{1}{2} \ln \frac{t + z}{t - z}, \tag{2a.2}$$

and also the invariant time

$$\tau = \sqrt{t^2 - z^2}. \tag{2a.3}$$

The latter is the evolution parameter for our system. It appears in the equations of motion [4] as an independent variable

$$\frac{d\eta}{d\tau} = \frac{\tanh(y - \eta)}{\tau}, \tag{2a.4}$$

$$\frac{dy}{d\tau} = \frac{F}{m_T \cosh(y - \eta)}. \tag{2a.5}$$

In Eq. (2a.5) F is the classical force. Let us now discuss the method of its calculation. The crucial thing is to find the chromoelectric force acting on a parton. In order to do that, we apply the Gauss law to the system of quarks and antiquarks at the given time τ. We use the convention according to which the field is positive when the lines of the field go from the left (smaller values of quasirapidity) to the right (greater values of quasirapidity). Therefore a quark is the source of the positive field on its right side and the negative field on its left side. An antiquark produces the field of the opposite sign. The effective field is the superposition of the elementary contributions from all partons. This rule leads to the screening of the field. Between the groups of partons which are neutral as a whole the field is canceled (see Fig. 1).

If we want to calculate the force acting on a parton we first find the value of the field on both sides of the parton: $\mathcal{E}^{left}$ and $\mathcal{E}^{right}$. The force is given by the expression

$$F = \pm\frac{1}{2}Q(\mathcal{E}^{left} + \mathcal{E}^{right}). \tag{2a.6}$$

where $\pm Q$ is the parton's charge. For instance we can take into consideration an elementary quark-antiquark tube (see Fig. 2). The Gauss law gives:

$$A\mathcal{E} = Q, \tag{2a.7}$$

where A is the transverse cross section of the tube and $\mathcal{E}$ is the chromoelectric field between the quark and antiquark. The string tension is defined as usual

$$\sigma = \frac{1}{2}\mathcal{E}^2 A = \frac{Q^2}{2A}. \tag{2a.8}$$

Now we find (in accordance with the prescription given above):

$$F^+ = \frac{1}{2}\,(0 + \frac{Q}{A})\,Q = \sigma, \tag{2a.9}$$

$$F^- = -\frac{1}{2}\,(\frac{Q}{A} + 0)\,Q = -\sigma, \tag{2a.10}$$

where the superscript '+' ('−') denotes the quark's (antiquark's) variables.

b) Initial conditions

We start our simulation having a simple system of a quark and an antiquark which are connected by the tube of the chromoelectric field. Each member of the pair has the same energy: $E^+ = E^- = \frac{1}{2}\sqrt{s}$, and the same transverse mass $m_T^+ = m_T^- = m$. Here $\sqrt{s}$ is the total energy measured in the center-of-mass system. We assume that the initial partons have no transverse momentum. In [5] we have used the values: $m = 10\,\mathrm{MeV}$, $\sqrt{s} = 30\,\mathrm{GeV}$.

At the very beginning of the process the quark and the antiquark are placed at the point $(t = 0, z = 0)$ with the opposite rapidities

$$y_0^\pm = \mp\,\mathrm{Arcosh}\Big(\frac{\sqrt{s}}{2m}\Big). \tag{2b.1}$$

They are receding from each other along the z axis: the quark goes to the left and the antiquark goes to the right. They span the chromoelectric field loosing their kinetic energy. One can check, however, that in the quasirapidity space situation is quite different (see Fig. 3): partons start their motion from distant points $(\tau = 0, \eta^\pm = y_0^\pm)$ and approach each other. The difference comes from the singularity appearing in the definition (2a.2) for $t = z = 0$.

The equations of motion for the initial particles can be solved analytically, for the initial conditions discussed here we obtain:

$$\eta^{\pm}(\tau) = y_0^{\pm} \pm \mathrm{Arsinh}\Big(\frac{\sigma\tau}{2m}\Big). \qquad (2b.2)$$

$$y^{\pm}(\tau) = \mathrm{Arsinh}\Big[\sinh(y_0^{\pm}) \pm \frac{\sigma\tau}{m}\cosh(\eta^{\pm}(\tau))\Big]. \qquad (2b.3)$$

We observe that for $\tau = 0$ the quasirapidities of partons are equal to their rapidities.*

c) Formation of yo-yo's

After some time, the first tube spontaneously decays producing a $q-\bar{q}$ pair. In the region between the members of the new pair the chromoelectric field is canceled (because of screening) and we obtain two new tubes which move in the space-time like the first one, but with different initial conditions. At the later stages this pattern is repeated many times developing a cascade which leads to the very strong fluctuations of the field. The initial tube is broken into pieces.

When a certain tube does not decay for a sufficiently long time, the quark and the antiquark start oscillations forming a yo-yo. The energy is transfered from partons to the field and back but the total energy of a yo-yo E^{yo-yo} and the momentum p_L^{yo-yo} are conserved. The energy and momentum of a yo-yo as a whole are defined below:

$$E^{yo-yo} = E^{+} + E^{-} + E^{field}, \qquad (2c.1)$$

$$p_L^{yo-yo} = p_L^{+} + p_L^{-} + p^{field}, \qquad (2c.2)$$

* In the first version of our calculations [5] we avoided the problems related to that singularity starting simulation from a non-zero (but very small) value of τ.

where

$$E^{\pm} = m_T^{\pm} \cosh y^{\pm}, \tag{2c.3}$$

$$p_L^{\pm} = m_T^{\pm} \sinh y^{\pm}. \tag{2c.4}$$

$$E^{field} = \sigma\tau\Big(\sinh \eta^{right} - \sinh \eta^{left}\Big), \tag{2c.5}$$

$$p^{field} = \sigma\tau\Big(\cosh \eta^{right} - \cosh \eta^{left}\Big). \tag{2c.6}$$

Here η^{right} and η^{left} denote the positions of the ends of a tube:

$$\eta^{right} = \mathrm{Max}(\eta^{+}, \eta^{-}), \tag{2c.7}$$

$$\eta^{left} = \mathrm{Min}(\eta^{+}, \eta^{-}). \tag{2c.8}$$

Having defined the energy and the longitudinal momentum of a yo-yo we can calculate its rapidity:

$$y^{yo-yo} = \frac{1}{2} \ln \frac{E^{yo-yo} + p_L^{yo-yo}}{E^{yo-yo} - p_L^{yo-yo}}. \tag{2c.9}$$

Similarly, we can calculate the transverse momentum of a yo-yo, however we need an additional information about the correlations between the partons' transverse momenta

$$p_T^{yo-yo} = \sqrt{(p_T^{+})^2 + (p_T^{-})^2 + 2\, p_T^{+}\, p_T^{-} \cos(\phi^{+} - \phi^{-})}. \tag{2c.10}$$

The angles $\phi^{\pm}$ are defined through the relations: $p_x^{\pm} = p_T^{\pm}\cos\phi^{\pm}$ and $p_y^{\pm} = p_T^{\pm}\sin\phi^{\pm}$. We want to stress here once again that both p_T and ϕ are not dynamical variables -- they do not change in time. The values of p_T and ϕ are determined during the tunneling process and at later stages they remain constant. Using the above expressions we can also define the mass of a yo-yo:

$$M^{yo-yo} = \sqrt{(E^{yo-yo})^2 - (p_L^{yo-yo})^2 - (p_T^{yo-yo})^2}. \qquad (2c.11)$$

d) Interactions of yo-yo's

So far, we have taken into account only elementary color-flux tubes. Nevertheless our results are useful because our system can be always treated as a set of elementary quark-antiquark tubes (yo-yo's). A priori more complicated structures can be created when two tubes overlap. In this situation one can distinguish between the two different cases: i.e. at the ends of the overlapping tubes there are partons either with the same color charges or with the different ones (see Fig. 4).

Let us concentrate on the first possibility. A quark from the first yo-yo meets and passes by the quark from the second yo-yo. The chromoelectric field is immediately screened between the quarks and they are connected with new antiquarks: the quark from the first yo-yo is connected by the tube of the color field with the antiquark from the second yo-yo, the quark from the second yo-yo joins the antiquark from the first yo-yo. Of course the same process of *exchange of partons* can be started by two antiquarks coming from different yo-yo's. Here, we only want to stress that the new tubes are elementary.

The second possibility takes place when at the ends of the overlapping tubes there are different color charges — a quark and an antiquark. In such cases more complicated structures are created which consist of four quarks joined by three color tubes. The intensity of the chromoelectric field in the internal tube is two

times greater than the intensity of the field in an elementary tube. This may result in many interesting effects; *e.g.* the internal tube may decay creating a very complicated system of six quarks. Also one can observe that two internal partons strongly attracted by the internal field come back, pass by each other and the system of four quarks becomes again a system of two independent elementary tubes. The latter process treated as a whole looks like an *elastic collision of two yo-yo's*. Because in the present version of our program we are not ready to describe the behavior of complicated many-quark structures we assume that the second possibility of interaction always leads to the collision of the yo-yo's. In practice, whenever two tubes start to overlap and create the region with non-elementary field we assume that the partons are scattered and consequently the tubes reppel from each other.*

e) Tunneling

We assume [4] that the tunneling (virtual) particles go along the boost-invariant hyperbola $\tau = \text{const}$. Observing two pairs which are created at the different time moments τ_1 and τ_2 we can say that the first is earlier than the second if $\tau_1 < \tau_2$. This time subsequence is absolute in the sense that it does not depend on the choose of the Lorentz frame. Moreover, the above constraint on the tunneling process allows us to fulfill the requirements of causality for our system.

The probability of tunneling of one virtual pair is given by

$$P = \exp\left(-\frac{\pi m_T^2}{\sigma}\right), \qquad (2e.1)$$

* In the version 2.0 of TERMITE the tubes may overlap and produce stronger fields. This is more physical approach but it is more difficult to describe creation of pairs in such fields since field can vary along the trajectories of tunneling particles.

whereas the probability of the decay of a tube is obtained by the multiplication of the above formula by the number of virtual pairs in the tube.

One of the interesting consequences of our approach is that the tunneling particles, when they become real, have non-zero longitudinal momenta. For instance, the members of a virtual pair which starts from the point $\eta^{st} = 0$ gain the rapidities [4]

$$y^{tun} = \pm \mathrm{Arsinh}\Big(\frac{m_T^2}{2\sigma\tau}\Big), \qquad \eta^{tun} = 2\,y^{tun}. \tag{2e.2}$$

When a pair starts tunneling from an arbitrary point η^{st} lying on the hyperbola $\tau = \mathrm{const}$, then the final rapidity and quasirapidity are boosted by η^{st}:

$$y^{tun} = \eta^{st} \pm \frac{1}{4}\Delta\eta, \quad \eta^{tun} = \eta^{st} \pm \frac{1}{2}\Delta\eta. \tag{2e.3}$$

Here we have introduced a boost-invariant quantity $\Delta\eta = 4\,\mathrm{Arsinh}(m_T/2\sigma\tau)$ which has a simple physical interpretation, namely it gives the distance in the quasirapidity space between the members of the emerging pair.

Now one can observe that during the decay of a quark-antiquark tube the transverse momenta of tunneling partons cannot be arbitrarily large. When at the moment τ the ends of a tube have the quasirapidities η^{right} and η^{left} then the length of the tube $\Delta\eta^{tube} = \eta^{right} - \eta^{left}$ and the transverse mass of tunneling particles satisfy the condition

$$m_T^2 \le 4\,\sigma^2\,\tau^2 \sinh^2\Big(\frac{\Delta\eta^{tube}}{4}\Big). \tag{2e.4}$$

There are also arguments [6] that the transverse dimension of the tube determines the largest possible value of the tunneling transverse mass, hence we can write

$$m_T^2 \leq (m_T^{max})^2 \qquad (2.5)$$

The above conditions, put together, determine the region in the p_x-p_y space which is available for tunneling particles (see Fig. 5a):

$$m^2 + p_x^2 + p_y^2 \leq \mathrm{Min}\left\{ (m_T^{max})^2, \left(\sigma^{tube} \sin h\left(\frac{\Delta\eta^{tube}}{4}\right)\right)^2 \right\} \qquad (2.6)$$

For small values of $\Delta\eta^{tube}$ the tunneling processes are not allowed: the energy of the field is not sufficient for creation a quark-antiquark pair even when the latter has no transverse momentum.

We can also see that the starting point η^{st} cannot be situated too close to one of the ends of the tube (η^{left} or η^{right}). For a given value of m_T the values of η^{st} are restricted to the interval:

$$\eta^{left} + \frac{\Delta\eta}{2} < \eta^{st} < \eta^{right} - \frac{\Delta\eta}{2}. \qquad (2.7)$$

In the simulation we assume that $\vec{p}_T = (p_x, p_y)$ is a random vector uniformly distributed in the region (2e.6). For the chosen value of $\vec{p}_T$ we generate a value of η^{st}. Their distribution is assumed to be uniform in the interval (2e.7) (see Fig. 5b).

f) Remarks on hadronization

A hadronization scheme is absent in our program. Products of the decays of color-flux tubes (oscillating yo-yo's) do not correspond to direct observables in the real world. Their mass distribution is a continuous curve whose maximum depends on the time-length of the evolution: the longer is our simulation the greater is the number of light yo-yo's. Nevertheless the User can select and extract interesting yo-yo states (*e.g.* the yo-yo's whose masses are close to the mass of a meson).

3. Guidelines for Users of TERMITE

a) Hardware and Compiler Requirements

The program may be used on every IBM PC but it is recommended to equip the User's PC with a math coprocessor (which makes the program run much faster) and with a hard disk (which can collect large files of TERMITE's results). No special memory extension is required since the program does not occupy more than 64 kilobytes for reasonable values of MAX_PARTONS, MAX_YOYOS, MAX_AVER and MAX_RAP.

The source code of TERMITE can be compiled with versions 4.0 through 5.5 of Turbo Pascal compiler. However, the User should choose the set of compiler directives according to the language version and to the presence of a floating-point coprocessor. These are the prescribed lists of directives:

— for version 4.0 (with coprocessor):

$$R-, S-, I+, D-, T-, F-, V-, B-, N+, L+$$

— for version 5.0 or 5.5 (with coprocessor):

$$A+, B-, D-, E-, F-, I+, L+, N+, O-, R-, S-, V-$$

— for version 5.0 or 5.5 (without coprocessor):

$$A+, B-, D-, E+, F-, I+, L+, N+, O-, R-, S-, V-$$

The program uses three standard units of subroutines: Crt, Dos and Graph which have to be present during the compilation.

b) Program configuration

The User of TERMITE can choose different options of the configuration of program execution. The program source text begins with declarations of the five Boolean flags:

GraphOn — controls displaying the trajectories of partons and of yo-yo's on the screen,

SoundOn — toggles the sound buzzer,

SaveOn — if TRUE, the results of the simulation are saved in disk files,

VerifyOn — toggles the procedure VerifyYoyos which selects and afterwards removes the yo-yo's satisfying certain conditions.

WaitOn — if TRUE, the program waits for pressing a key after each event.

The above flags may be set on or off completely independently of each other.

c) Parameters of the program

The parameters of TERMITE are divided into some groups:

— physical parameters,

— numerical parameters,

— graphics parameters,

— names of disk files,

— auxiliary parameters.

The *physical parameters* determine the initial conditions for the simulation and affect the strength of interactions and the probability of tunneling:

REST_MASS — rest mass of a parton (m),

CMS_ENERGY — energy of collisions in the center-of-mass frame ($\sqrt{s}$),

DEG_FREEDOM — number of degrees of freedom related to spin, color and flavor ($N_s N_c N_f$),

STRING_TENSION — quark string tension (σ),

MESON_MASS — meson mass, used for the selection of yo-yo's in the VerifyYoyos procedure,

T_FINISH — time interval for one event,

TRANS_CUTOFF — maximal transverse mass related to the finite transverse dimension of the tube

The time steps used in the numerical scheme of solving the equations of motion can be treated as the *numerical parameters* which determine the accuracy of solutions. Their values depend on the quark rest mass and the string tension. The steps are denoted by:

DT_BEGIN — used in the early phase of an event when the considered system contains one single yo-yo and the equations of motion are solved analytically,

DT_STANDARD — used after creation of the first pair.

The basic *graphics parameter* (BGI_PATH) specifies the path to the disk directory containing the BGI driver for the current graphic device. (The drivers are supplied together with Turbo Pascal system.) If the driver is stored in the same directory as TERMITE — this parameter may be left simply as an empty string (' '). The remaining six graphics parameters are the colors for the objects appearing in the screen (if the option GraphOn is set to TRUE):

AXIS_COLOR — axes, scales, screen header,

QUARK_COLOR — quark trajectories in the (τ, η) space,

ANTIQUARK_COLOR — antiquark trajectories in the (τ, η) space,

TUNNEL_COLOR — the gap between the tunneling partons,

RAPIDITY_COLOR — trajectories of yo-yo's in the rapidity space,

REMOVED_COLOR — circles denoting the positions of removed yo-yo's.

The User can customize the colors according to his personal preference and to the available type of graphics adapter.

The results of the simulation are optionally written to *disk files* whose names are the following string constants:

MASS_FILE_NAME — evolution of the average rest mass of yo-yo's (in MeV).

PT_FILE_NAME — evolution of the average transverse momentum of yo-yo's (in MeV).

NUM_FILE_NAME — evolution of the number of yo-yo's,

RAP_FILE_NAME — list of yo-yo rapidities at the end of an event.

The last five parameters have *auxiliary* character:

MAX_PARTONS — maximal number of partons created in one event,

MAX_YOYOS — maximal number of yo-yo's created in one event,

MAX_EVENTS — number of events in one run of the program,

MAX_AVER — number of time points at which the average quantities are being saved,

MAX_RAP — number of time points at which the rapidity spectrum is written to the disk.

d) Representation of Physical Quantities

The state of each parton is described by giving its position in quasi-rapidity (*eta*), its *rapidity*, transverse *mass*, and the azimuthal *angle* of transverse momentum in

(x, y) plane. The four variables together with a logical flag denoting whether the parton is a quark or an antiquark form a five-component record type declared as PartonType. The chain of partons lying along the collision axis is represented as an array variable named parton. The color field between every two neighbors in the chain is assumed either to be equal to an elementary field ($\mathcal{E} = \pm Q/A$) or to vanish ($\mathcal{E} = 0$). Therefore, the chain of color tubes (field) can be represented as an array of short integers.

e) The Main Program Body

The session with the program starts with calling the procedure InitProgram which initializes the random number generator, activates the graphics module (if the GraphOn flag is set to TRUE) and initializes variables and files related to the process of saving the results (provided the SaveOn flag is set to TRUE). Then, the main loop begins. Each pass of the loop corresponds to one physical event, *i.e.* to the evolution from $\tau = 0$ till $\tau =$ T_FINISH. The loop is repeated until the event counter (num_events) reaches the value of MAX_EVENTS or the User breaks the execution (see *"Controlling TERMITE"*). The program ends with the procedure Quit which, if necessary, deactivates the graphic device and restores the text mode of the screen.

The initial conditions for each event are fixed by the procedure InitEvent. Time τ (tau) is zeroed, the number of color tubes (num_tubes) is set to 1 and the dynamical quantities for the initial $q - \bar{q}$ pair are assigned with the values according to the global settings of the session. The procedure OutYoyos is called — it computes (and eventually reports in the screen) the current number of yo-yo's, average mass and transverse momentum of yo-yo's. The process of saving the results of the event is also initialized (see details in *"Saving the Results"*).

After the event has been initialized, the time loop starts and at each step the current value of time tau is increased by dtau. The actual value of dtau is choosen

by the procedure SetTimeStep: before the very first decay the time step is equal to DT_BEGIN, afterwards we accept the value of DT_STANDARD. These constants should be small enough to give a good convergence of the discrete numerical scheme of integration of the trajectories but large enough to give a reasonable consumption of computer time per one event.*

f) Motion of Partons — Procedure Dynamics

The procedure Dynamics finds the change of positions and rapidities of partons within time interval $(\tau, \tau + d\tau)$. If the system contains only one $q - \bar{q}$ pair (namely the initial one), the trajectories are obtained from the exact formulae (2b.2)-(2b.3). Otherwise, the partons are subsequently moved to next positions in the phase space. The new (quasi)rapidities are calculated by the procedure MoveParton which uses the Runge-Kutta method for numerical integration of the equation of motion (2a.4)-(2a.5). When the seqnence of partons at time $\tau + d\tau$ is different from that at time τ (*i.e.* if some trajectories cross each other) the procedure Dynamics makes use of the algorithm described in Section 2d. Scattering or exchanging partons may affect some average quantities, therefore the procedure OutYoyos is called after such cases in order to verify and re-display the averages and the number of yo-yo's.

g) Production of Particles — Procedures: Decays and Tunneling

The procedure Decays scans the series of tubes and applies the procedure Tunneling to those which have non-zero color field.

The procedure Tunneling generates a transverse mass (function VirTrM) that can be produced inside a given tube (see formula (2e.6)). Afterwards the probability of the decay is computed. The expression for the tunneling probability (2e.1)

* In the new version of TERMITE the value of the time step varies and follows current dynamical conditions.

is multiplied by the number of virtual pairs in the considered space-time volume and also by the factor related to the spin, color and flavor degrees of freedom (DEG_FREEDOM).

The decay of the tube is generated according to the Poissonian distribution (function Poisson). If a pair is to be created then its physical parameters are established and the pair is inserted into the existing chain (procedure InsertPair). Each decay increases the number of color tubes by two.

h) Verification of Yo-yo States — Procedure VerifyYoyos

There is a possibility in the program that interesting yo-yo's can be selected and removed from the evolution of the whole system. For instance, one can assume that the yo-yo's having masses which are close to that of a meson become non-interacting final-state hadrons. The User finds enclosed the procedure which extracts the yo-yo's whose masses are between one and two meson masses. The latter is introduced in the program as a constant MESON_MASS (see *"Parameters of the Program"*).

i) Graphics

The graphics module is enabled if the GraphOn flag is set to TRUE. The header displayed in the screen contains information about the current number of yo-yo's and of the averages of rest masses and transverse momenta of yo-yo's. The trajectories of partons in the (τ, η) space are displayed in the lower half of the screen while the trajectories of yo-yo's in the rapidity space are displayed in the upper half of the screen. The vertical scales refer to the invariant time τ in Fermi's. The horizontal axis is scaled in the dimensionless units of (quasi)rapidity.

The graphic demonstration module will run on every graphic device that is supported by a proper BGI driver. For instance, if the User's PC is equipped

with the Hercules Graphics Card, the User has to copy the driver HERC.BGI to his own disk directory or specify the DOS path to the directory containing all drivers (constant BGI_PATH). In case of any doubts what type of adapter is in use, the better way is the latter one because the program will automatically choose the proper driver.

When TERMITE is used as a Monte-Carlo generator of data, the graphics should be turned off in order to make the program run faster.

j) Saving the results

The results of each event are stored in disk files if the SaveOn flag is set to TRUE. The saving is performed at the end of the event. As mentioned before, TERMITE reports the evolution of number of yo-yo's, of the average rest mass of yo-yo's and of the average transverse momentum of yo-yo's. The average quantities are being stored in arrays AverNum, AverMass and AverPt. Each such an array contains MAX_AVER+1 elements. At the end of each event the arrays are written to proper text disk files in a format which is acceptable by standard graphic programs for drawing two-dimensional diagrams. Here is the format of saved data:

$\tau_0 \quad f(\tau_0)$

$\tau_1 \quad f(\tau_1)$

$\vdots$

$\tau_N \quad f(\tau_N)$

(End-of-file)

where N = MAX_AVER, $\tau_0 = 0$, τ_N = T_FINISH and f refers to the considered average quantity.

Moreover, the program saves rapidities of yo-yo's at each of MAX_RAP "ticks" during one event — the last "tick" coincides with T_FINISH. The values of rapidities can be used to obtain a rapidity spectrum of created yo-yo's at different τ's.

The lists of rapidities corresponding to different "ticks" are written to separate text files of the same name (string constant RAP_FILE_NAME) but with different extensions (according to the number of a given tick). The format of such file is as follows:

N_1

$y_{1,1}$

$y_{1,2}$

$\vdots$

y_{1,N_1}

N_2

$y_{2,1}$

$y_{2,2}$

$\vdots$

y_{2,N_2}

$\vdots$

N_M

$y_{M,1}$

$y_{M,2}$

$\vdots$

y_{M,N_M}

(End–of–file)

where M is the number of events in this session, N_i is the number of yo-yo's in i-th event, and $y_{j,k}$ is the rapidity of k-th yo-yo in j-th event.

k) Controlling TERMITE

The procedure KeyControl (called after each time step) allows the User to control the execution of the program. Here is the complete list of "hot-keys":

Key	Action
⟨Esc⟩ or ⟨Ctrl-C⟩	Quits the program and returns to the operating system.
⟨P⟩ or ⟨spacebar⟩	Pause — Suspends the execution and waits for pressing a key. If the next key is ⟨Esc⟩ or ⟨Ctrl-C⟩, quits the program.
⟨R⟩	Restarts the program and clears all the data stored during the session.
⟨T⟩	Time — When not in graphic mode reports the actual value of τ in the current event.

Quitting the program (by pressing ⟨Esc⟩ or ⟨Ctrl-C⟩) when simulating the n-th event does not erase the results of previous $n-1$ events (if those are saved on disk). However, restarting the program will cancel all prevoius events and will start the session again from the first event.

Summary

The TERMITE program can be used as a demonstration program for presenting the evolution of the system: "quarks + antiquarks + color field". Moreover, it may work as a typical Monte Carlo program, giving some quantitative predictions for rapidity spectra of particles, average masses and transverse momenta, etc. One of its important features is modularity and possibility of development and of modifications — the User can add his own output procedures computing other physical quantities. The physical background of TERMITE is based on a boost-invariant model of pair production proposed in Ref. [4]. The advantage of the model is its simplicity, a small number of physical input parameters and a consistent semi-classical ultrarelativistic description of creation of $q-\bar{q}$ pairs including the feedback on the original field.

The presented version of TERMITE has been designed for users of IBM PC. We prepare, however, a new version written in Fortran 77 which may be acceptable by a big computer. The new version is going to be much richer — it will consider a possibility of creation of stronger color fields, different colors and flavors of quarks, and creation of gluons (in the way of tunneling and as a bremsstrahlung). Nevertheless, we think that even this simpler version of TERMITE may be useful for investigating various processes and phenomena in high-energy collisions.

Acknowledgments

We would like to thank Professor A. Bialas and Professor W. Czyż for their critical comments, discussions and encouragment. We express our gratitude to Professor R. Peschanski for his stimulating remarks and the hospitality at CEN-Saclay where a part of this work has been done.

References

[1] J. Schwinger, Phys. Rev. **82** (1951) 664

[2] X. Artru and G. Mennessier, Nucl. Phys. **B70** (1974) 93

[3] B. Anderson et al., Phys. Rep. **97** (1983) 31

[4] A. Białas, W. Czyż, A. Dyrek and W. Florkowski, Zeit. f. Physik C **46** (1990) 439, A. Dyrek, Ph. D. Thesis, Krakow Preprint TPJU 3/90

[5] A. Białas, W. Czyż, A. Dyrek, W. Florkowski and R. Peschanski, Phys. Lett. **B 229** (1989) 398

[6] A. Casher, H. Neuberger, S. Nussinov, Phys. Rev. **D 20** (1979) 179

Figure Captions

Fig. 1 The chromoelectric field generated by: a quark (a), an antiquark (b) and a quark-antiquark pair (c).

Fig. 2 A quark-antiquark system in the (τ, η) space.

Fig. 3 Two different representations of the trajectories of a quark and an antiquark which form the initial color-flux tube: the first one (a) in the (t, z) space, the second one (b) in the (τ, η) space.

Fig. 4 Interaction of two yo-yo's. In the space-time region where two yo-yo's might overlap (a) the color field is canceled or doubled. The first possibility leads to the exchange of partons possesing the same color charge (b), whereas the second one causes repulsion of the yo-yo's. In the latter case we neglect the creation of pairs in the region where the tubes overlap.

Fig. 5 The available phase space for tunneling partons.

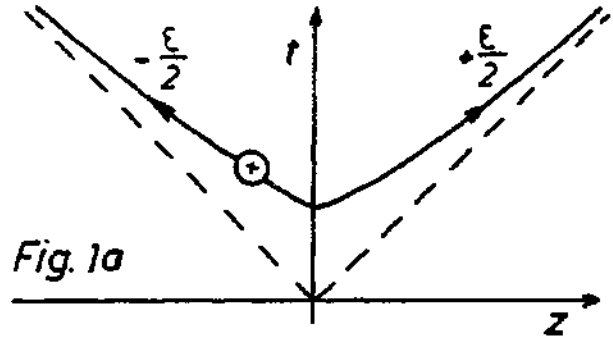


Fig. 1a

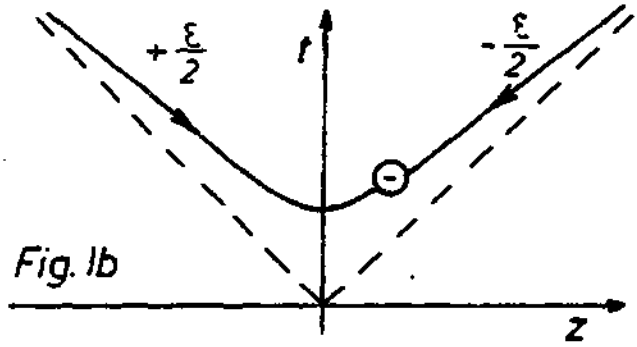


Fig. 1b

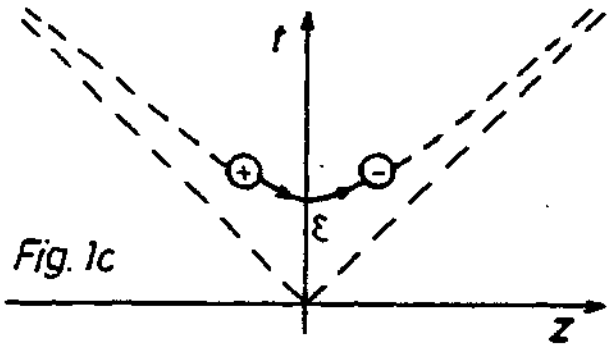


Fig. 1c

Fig. 2

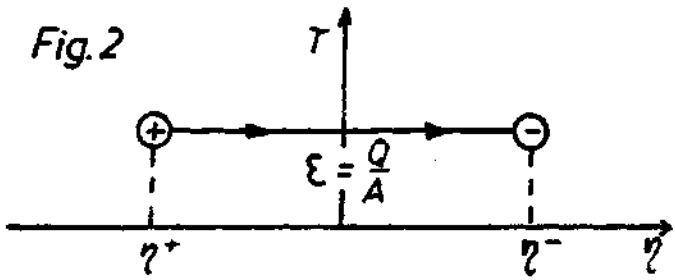


Fig. 3a

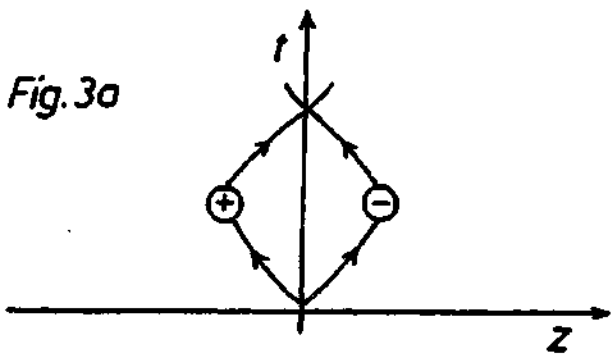


Fig. 3b

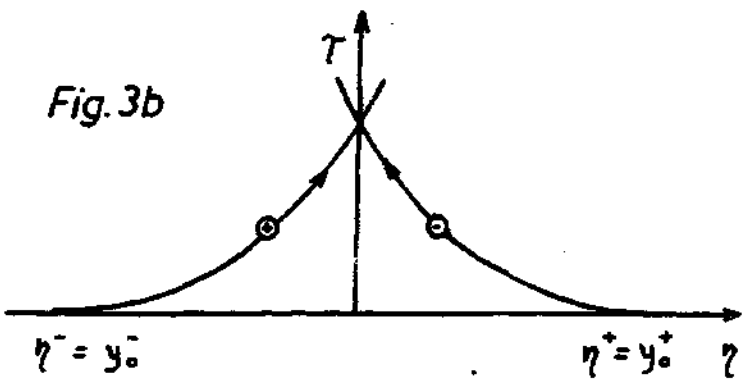


Fig. 4a

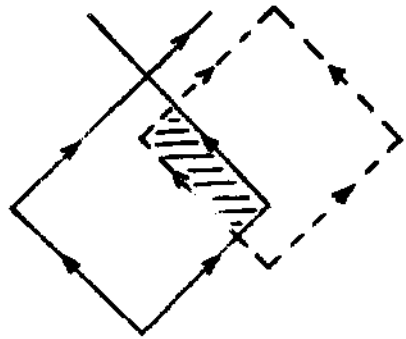


Fig. 4b

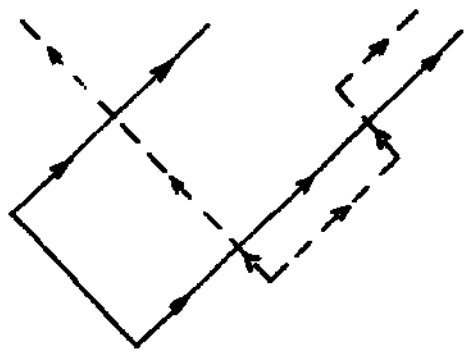


Fig. 4c

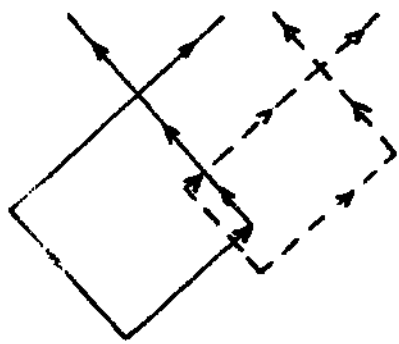


Fig. 5a

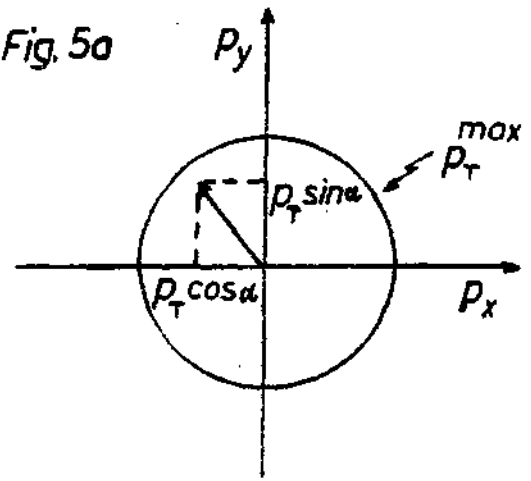
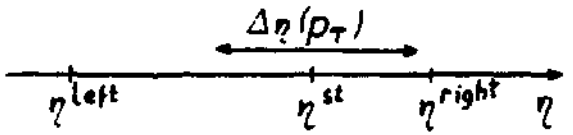

p_y
max
p_T
$p_T \sin\alpha$
$p_T \cos\alpha$
p_x

Fig. 5b


$\Delta\eta\,(p_T)$
η^{left}
η^{st}
η^{right}
η

Appendix

Listing of the Program

```
{ Compiler directives:

    - for version 4.0 (with coprocessor):

        A-,S-,I+,D-,T-,F-,V-,B-,N+,L-

    - for version 5.0 (with coprocessor):

        A+,B-,D-,E-,F-,I+,L+,N+,O-,R-,S-,V-

    - for version 5.0 (without coprocessor):

        A+,B-,D-,E+,F-,I+,L+,N+,O-,R-,S-,V- }

{================================================================}

{$A+,B-,D-,E-,F- I+,L+,N+,O-,R-,S-,V-}
{$M 16384,0,655360}

PROGRAM Termite;

USES
   Crt, Dos, Graph

CONST
   { Configuration flags }
   GraphOn   = TRUE;
   SoundOn   = TRUE;
   SaveOn    = FALSE;
   VerifyOn  = FALSE;
   WaitOn    = TRUE;

   { Physical parameters }
   REST_MASS      : Extended = 1.5;    { 1.5 1/fm = 300 MeV }
   CMS_ENERGY     : Extended = 150.0;  { sqrt(s) = 150 1/fm = 30 GeV }
   DEG_FREEDOM    : Integer  = 12;     { 2x3x2 (spin, color, flavor) }
   STRING_TENSION: Extended = 5.0;     { 5 1/fm^2 = 1 GeV/fm }
   MESON_MASS     : Extended = 5.0;    { 5 1/fm = 1 GeV }
   T_FINISH       : Extended = 10.0;   { fm }
   TRANS_CUTOFF   : Extended = 5.0;    { 5 1/fm = 1 GeV }

   { Numerical parameters }
   DT_BEGIN    : Extended = 0.01;  { fm }
   DT_STANDARD: Extended = 0.01;  { fm }
```

```
    { Graphics parameters }
    BGI_PATH         = '';    { path to graphic BGI driver }
    AXIS_COLOR       = White;
    QUARK_COLOR      = White;
    ANTIQUARK_COLOR  = White;
    TUNNEL_COLOR     = White;
    RAPIDITY_COLOR   = White;
    REMOVED_COLOR    = White;

    { Disk file names }
    MASS_FILE_NAME = 'mass';
    PT_FILE_NAME   = 'pt';
    NUM_FILE_NAME  = 'num';
    RAP_FILE_NAME  = 'rap';

    { Auxiliary parameters }
    MAX_PARTONS = 200;     { maximal number of partons }
    MAX_YOYOS   = 100;     { maximal number of yoyos }
    MAX_EVENTS  = 10;      { maximal number of events  }
    MAX_AVER    = 50;
    MAX_RAP     = 10;

TYPE
    PartonType = RECORD
                   eta,
                   rapidity,
                   mass,
                   angle: Extended;
                   quark: Boolean
                   END;
    FieldType = ShortInt;
    ListRapType = RECORD
                    num: Integer;
                    rap: ARRAY [1..MAX_YOYOS] OF Extended
                    END;

VAR
    { chain of partons and color-field tubes }
    parton: ARRAY [1..MAX_PARTONS] OF PartonType;
    field: ARRAY [0..MAX_PARTONS] OF FieldType;

    init_rapidity,          { initial rapidity }
    rest_mass2,             { quark rest mass squared }
    tau,                    { actual invariant time }
    dtau: Extended;         { actual time step }

    num_events,             { number of events }
    num_tubes: Integer;     { number of tubes }
```

```
    Restart,                { event restart flag }
    FirstPair: Boolean;     { first pair flag }

    MassFile,               { average yo-yo mass evolution }
    PtFile,                 { average yo-yo tr. momentum evolution }
    NumFile: Text;          { number of yo-yo's evolution }
    RapFile: Text;          { file for final yo-yo rapidities }
    ListRap: ARRAY [1..MAX_RAP] OF ListRapType;

    old_mass, cur_mass,
    old_pt, cur_pt, cur_rap, cur_tau: Extended;
    old_num, cur_num: Integer;

    AverPt,
    AverMass: ARRAY [0..MAX_AVER] OF Extended;
    AverNum: ARRAY [0..MAX_AVER] OF LongInt;

    rap_pass, aver_pass: Integer;
    rap_pass_ratio, aver_pass_ratio: Extended;

{==========================================================}

{  HEADERS OF FUNCTIONS AND PROCEDURES  }

{  GENERAL FUNCTIONS MODULE  }

FUNCTION  TimeStr: String; FORWARD;
FUNCTION  Int2Str(L: LongInt): String; FORWARD;
FUNCTION  sinh(x: Extended): Extended; FORWARD;
FUNCTION  cosh(x: Extended): Extended; FORWARD;
FUNCTION  tanh(x: Extended): Extended; FORWARD;
FUNCTION  arsinh(x: Extended): Extended; FORWARD;
FUNCTION  arcosh(x: Extended): Extended; FORWARD;
FUNCTION  artanh(x: Extended): Extended; FORWARD;

{  GRAPHICS MODULE  }

PROCEDURE  InitGraphics; FORWARD;
PROCEDURE  DrawBorder; FORWARD;
PROCEDURE  FullPort; FORWARD;
PROCEDURE  DrawTrajectories; FORWARD;
PROCEDURE  DrawAxes; FORWARD;
PROCEDURE  OutYoyos; FORWARD;

{  INITIAL MODULE  }

PROCEDURE  InitProgram; FORWARD;
PROCEDURE  InitEvent; FORWARD;
PROCEDURE  EndEvent; FORWARD;
```

```
{  DYNAMICS MODULE  }

PROCEDURE  SetTimeStep; FORWARD;
PROCEDURE  Dynamics; FORWARD;

{  DECAYS MODULE  }

PROCEDURE  Tunneling(VAR tube_to_decay: Integer); FORWARD;
PROCEDURE  Decays; FORWARD;

{  YO-YO MODULE  }

PROCEDURE  ClearSums; FORWARD;
PROCEDURE  SaveResults; FORWARD;
PROCEDURE  OutRapidities(RapFileNum: Integer); FORWARD;
FUNCTION  YoyoMass(i: Integer): Extended; FORWARD;
FUNCTION  YoyoPt(i: Integer): Extended; FORWARD;
FUNCTION  YoyoRapidity(i: Integer): Extended; FORWARD;
PROCEDURE  VerifyYoyos; FORWARD;
FUNCTION  AverageMeVMass: Extended; FORWARD;
FUNCTION  AverageMeVPt: Extended; FORWARD;
FUNCTION  SumMass: Extended; FORWARD;
FUNCTION  SumPt: Extended; FORWARD;

{  CONTROL MODULE  }

PROCEDURE  Quit; FORWARD;
PROCEDURE  KeyControl; FORWARD;

{=============================================================}

{  GENERAL FUNCTIONS MODULE  }

FUNCTION  Int2Str(L: LongInt): String;
{ Converts an integer to a string }
VAR
   S: string;
BEGIN
Str(L, S);
Int2Str := S
END; { Int2Str }

FUNCTION  TimeStr: String;
{ Returns the system time converted to a string }
VAR
   hour, min, sec, sec100: Word;
   hourstr, minstr, secstr: String;
BEGIN
GetTime(hour, min, sec, sec100);
hourstr := Int2Str(hour);
```

```
minstr := Int2Str(min,
IF  Length(minstr) = 1  THEN
   minstr := '0' + minstr;
secstr := Int2Str(sec);
IF  Length(secstr) = 1  THEN
   secstr := '0' + secstr;
TimeStr := hourstr + ':' + minstr + ':'   secstr
END; { TimeStr }

FUNCTION  sinh(x: Extended): Extended;
{ Hyperbolic sine }
BEGIN
     sinh := 0.5*(exp(x) - exp(-x))
END;

FUNCTION  cosh(x: Extended): Extended;
{ Hyperbolic cosine }
BEGIN
cosh := 0.5*(exp(x) + exp(-x))
END;

FUNCTION  tanh(x: Extended): Extended;
{ Hyperbolic tangent }
BEGIN
IF  x > 10.0  THEN
   tanh := 1.0
ELSE  IF  x < -10.0  THEN
   tanh := -1.0
ELSE
   tanh := (exp(x+x) - 1.0)/(exp(x+x) + 1.0)
END;

FUNCTION  arsinh(x: Extended): Extended;
{ Area hyperbolic sine }
BEGIN
arsinh := ln(x + sqrt(x*x + 1.0))
END;

FUNCTION  arcosh(x: Extended): Extended;
{ Area hyperbolic cosine }
BEGIN
arcosh := ln(x + sqrt(x*x - 1.0))
END;

FUNCTION  artanh(x: Extended): Extended;
{ Area hyperbolic tangent }
BEGIN
artanh := 0.5*ln((1.0 + x)/(1.0 - x))
END;
```

```
{  GRAPHICS MODULE  }

VAR
   GraphDriver,              { graphic (BGI) driver }
   GraphMode,                { graphic mode }
   ErrorCode   : Integer;    { graphics error code }
   MaxX, MaxY  : Word;       { screen resolution }
   HorizFactor,              { scaling factors }
   VertFactor: Extended;

PROCEDURE  InitGraphics;
{ Initialize graphics }
BEGIN
DirectVideo := False;
GraphDriver := Detect;
InitGraph(GraphDriver, GraphMode, BGI_PATH);
ErrorCode := GraphResult;
IF  ErrorCode <> grOk  THEN BEGIN
   Writeln(GraphErrorMsg(ErrorCode));
   Halt(1)
   END;
MaxX := GetMaxX;
MaxY := GetMaxY;
HorizFactor := 0.5*MaxX/init_rapidity;
VertFactor := (MaxY div 2 - 10)/T_FINISH;
END; { InitGraphics }

PROCEDURE  DrawBorder;
{ Draw a border around the current view port }
VAR
   ViewPort: ViewPortType;
BEGIN
SetLineStyle(SolidLn, 0, NormWidth);
GetViewSettings(ViewPort);
WITH  ViewPort  DO
    Rectangle(0, 0, x2-x1, y2-y1)
END; { DrawBorder }

PROCEDURE  FullPort;
{ Set the view port to the entire screen }
BEGIN
SetViewPort(0, 0, MaxX, MaxY, ClipOn);
END; { FullPort }

PROCEDURE  DrawTrajectories;
{ Draw positions of partons in quasi-rapidity and }
{ positions of yo-yo's in rapidity }
VAR
   tx, i: Integer;
BEGIN
```

```
tx := MaxY - Round(VertFactor*tau);
FOR  i := 1  TO  num_tubes+1  DO
   WITH  parton[i]  DO
      IF  quark  THEN
         PutPixel(Round(HorizFactor*eta) + MaxX div 2, tx,
                  QUARK_COLOR)
      ELSE
         PutPixel(Round(HorizFactor*eta) + MaxX div 2, tx,
                  ANTIQUARK_COLOR);
Dec(tx, MaxY div 2);
i := 1;
REPEAT
   PutPixel(Round(HorizFactor*ToyoRapidity(i)) + MaxX div 2, tx,
            RAPIDITY_COLOR);
   Inc(i, 2)
   UNTIL  i > num_tubes
END; { DrawTrajectories }

PROCEDURE  DrawAxes;
{ Draw the borders, axes and scales }
VAR
   i: ShortInt;
BEGIN
SetColor(AXIS_COLOR);
ClearDevice;
SetTextJustify(CenterText, TopText);
DrawBorder;
FOR  i := -Round(init_rapidity)  TO  Round(init_rapidity)  DO BEGIN
   Line(Round(i*HorizFactor) + MaxX div 2, MaxY div 2 - 3,
        Round(i*HorizFactor) + MaxX div 2, MaxY div 2 + 3);
   Line(Round(i*HorizFactor) + MaxX div 2, MaxY - 3,
        Round(i*HorizFactor) + MaxX div 2, MaxY);
   OutTextXY(Round(i*HorizFactor) + MaxX div 2,
             MaxY div 2 + 3, Int2Str(i))
   END;
SetTextJustify(LeftText, CenterText);
Line(1, MaxY div 2, MaxX, MaxY div 2);
FOR  i := 1  TO  Round(t_finish)  DO BEGIN
   Line(0, MaxY - Round(VertFactor*i),
        5, MaxY - Round(VertFactor*i));
   Line(0, MaxY div 2 - Round(VertFactor*i),
        5, MaxY div 2- Round(VertFactor*i));
   OutTextXY(8, MaxY - Round(VertFactor*i), Int2Str(i));
   OutTextXY(8, MaxY div 2- Round(VertFactor*i), Int2Str(i));
   Line(MaxX, MaxY - Round(VertFactor*i),
        MaxX - 5, MaxY - Round(VertFactor*i));
   Line(MaxX, MaxY div 2 - Round(VertFactor*i),
        MaxX - 5, MaxY div 2- Round(VertFactor*i));
   END;
SetTextJustify(LeftText, TopText);
```

```
END;  { DrawAxes }

PROCEDURE  OutYoyos;
{ Write the actual number of yo-yo's, average mass and Pt }
VAR
   Msg: String;
BEGIN
cur_tau  := tau;
old_mass := cur_mass;
cur_mass := AverageMeVMass;
old_pt   := cur_pt;
cur_pt   := AverageMeVPt;
old_num  := cur_num;
cur_num  := (num_tubes + 1) div 2;
IF  (old_num = cur_num) AND
    (old_mass = cur_mass) AND
    (old_pt = cur_pt)  THEN Exit;
IF  GraphOn  THEN BEGIN
   SetColor(ALIS_COLOR);
   Msg := 'Event: '+Int2Str(num_events) +
          '          Yo-yo #: '+Int2Str(cur_num) +
          '          <M>: '+Int2Str(Round(cur_mass)) + ' MeV' +
          '          <Pt>: '+Int2Str(Round(cur_pt)) + ' MeV';
   SetViewPort(30, 1, MaxX - 20, 10, ClipOff);
   ClearViewPort;
   OutTextXY(1, 1, Msg);
   FullPort
   END;
END; { OutYoyos }

{  INITIAL MODULE  }

PROCEDURE  InitRapFiles;
{ Initialize files for appending }
VAR
   i : Integer;
BEGIN
FOR  i := 1  TO  MAX_RAP  DO BEGIN
   Assign(RapFile, RAP_FILE_NAME+'.'+Int2Str(i));
   Rewrite(RapFile);
   Close(RapFile)
   END
END; { InitRapFiles }

PROCEDURE  InitProgram;
{ Initialize the main program body }
BEGIN
Randomize;
num_events := 1;
init_rapidity := arcosh(0.5*CMS_ENERGY/REST_MASS);
```

```
rest_mass2 := sqr(REST_MASS);
IF  GraphOn  THEN
   InitGraphics
ELSE
   ClrScr;
IF  SaveOn  THEN BEGIN
   ClearSums;
   InitMapFiles;
   map_pass_ratio := T_FINISH/MAX_MAP;
   aver_pass_ratio := T_FINISH/MAX_AVER
   END
END; { InitProgram }

PROCEDURE  InitEvent;
{ Setting parameters for one event }
BEGIN {InitEvent}
FillChar(field, SizeOf(field), 0);  { This cleans all the tubes }
FirstPair := TRUE;
Restart   := FALSE;
tau       := 0.0;
num_tubes := 1;
field[1]  := 1;
WITH  parton[1]  DO BEGIN
   eta       := -init_rapidity;
   rapidity  := -init_rapidity;
   mass      := REST_MASS;
   angle     := 0.0;
   quark     := TRUE
   END;
WITH  parton[2]  DO BEGIN
   eta       := init_rapidity;
   rapidity  := init_rapidity;
   mass      := REST_MASS;
   angle     := 0.0;
   quark     := FALSE
   END;
IF  GraphOn  THEN
   DrawAxes
ELSE
   Writeln('Event No.: ', num_events, ' started at '+TimeStr);
cur_num  := 0;
cur_mass := -1.0;
cur_pt   := -1.0;
OutYoyos;
```

```
IF  SaveOn  THEN BEGIN
   rap_pass     := 1;
   aver_pass    := 1;
   AverPt[0]    := AverPt[0] + SumPt;
   AverMass[0]  := AverMass[0] + SumMass;
   AverNum[0]   := AverNum[0] + 1;
   END;
END; {InitEvent}

PROCEDURE  EndEvent;
{ Close one event }
VAR
   i: Integer;
BEGIN
IF  NOT GraphOn  THEN
   Writeln(cur_num:8, ' yo-yo''s');
IF  NOT Restart  THEN BEGIN
   IF  SaveOn  THEN
      SaveResults;
   IF  WaitOn  THEN BEGIN
      IF  NOT GraphOn  THEN
         Writeln('End of event - Press any key...');
      IF  ReadKey IN [#27, ^C]  THEN  Quit;
      END;
   Inc(num_events)
   END
ELSE BEGIN
   num_events := 1;
   ClearSums
   END
END; { EndEvent }

{  DYNAMICS MODULE  }

PROCEDURE  SetTimeStep;
{ Set the time step value }
BEGIN
IF  FirstPair  THEN
   dtau := DT_BEGIN
ELSE
   dtau := DT_STANDARD
END; { SetTimeStep }

PROCEDURE  Dynamics;
{ Movement of partons and swapping in the case of overlap of tubes }
```

```
VAR
   temp: PartonType;      { temporary parton }
   temp_eta: Extended;    { temporary position in pseudorapidity }
   i: Integer;            { parton counter }
   m1, m2
   p, e, ycms,
   eta1, eta2,
   eta1_new,
   eta2_new
   rap1, rap2: Extended;
   field_new: ShortInt;

   PROCEDURE MoveParton(VAR aparton: PartonType);
   { Move one parton }
   VAR
      e1, e2, e3, e4,
      r1, r2, r3, r4: Extended;

      FUNCTION Force(n: Integer): ShortInt;
      { Color force in units of string tension }
      VAR
         i: Integer;
         nhelp: ShortInt;
      BEGIN
      nhelp := 0;
      IF n > 1 THEN
         FOR i := 1 TO n-1 DO
            IF parton[i].quark THEN
               Inc(nhelp)
            ELSE
               Dec(nhelp);
      IF n < num_tubes+1 THEN
         FOR i := n+1 TO num_tubes+1 DO
            IF parton[i].quark THEN
               Dec(nhelp)
            ELSE
               Inc(nhelp);
      IF parton[n].quark THEN
         Force := nhelp
      ELSE
         Force := -nhelp
      END; { Force }

      FUNCTION Feta(x0, x1, x2: Extended): Extended;
      { R.h.s. of the equation of motion for quasi-rapidity }
      BEGIN
      Feta := tanh(x1 - x2)/x0
      END;
```

```
      FUNCTION Frap(i: Integer; m, x1, x2: Extended): Extended;
      { R.h.s. of the equation of motion for rapidity }
      BEGIN
      Frap := STRING_TENSION*Force(i)/(m*cosh(x1 - x2))
      END;

   BEGIN { MoveParton }
   WITH  aparton  DO BEGIN
      { Runge-Kutta method is used here to integrate
        the equations of motion }
      r1 := dtau * Frap(i, mass, rapidity, eta);
      e1 := dtau * Feta(tau, rapidity, eta);
      r2 := dtau * Frap(i, mass, rapidity + 0.5*r1, eta + 0.5*e1);
      e2 := dtau * Feta(tau + 0.5*dtau, rapidity + 0.5*r1, eta + 0.5*e1);
      r3 := dtau * Frap(i, mass, rapidity + 0.5*r2, eta + 0.5*e2);
      e3 := dtau * Feta(tau + 0.5*dtau, rapidity + 0.5*r2, eta + 0.5*e2);
      r4 := dtau * Frap(i, mass, rapidity + r3, eta + e3);
      e4 := dtau * Feta(tau + dtau, rapidity + r3, eta + e3);

      eta := eta + (e1 + e2 + e2 + e3 + e3 + e4)/6.0;
      rapidity := rapidity + (r1 + r2 + r2 + r3 + r3 + r4)/6.0
      END
   END; { MoveParton }

BEGIN { Dynamics }
IF  GraphOn  THEN
   DrawTrajectories;
IF  FirstPair  THEN BEGIN
   { Exact trajectories }
   WITH  parton[2]  DO BEGIN
      eta      := init_rapidity - arsinh(0.5*STRING_TENSION*tau/mass);
      rapidity := arsinh(sinh(init_rapidity) -
                         STRING_TENSION*tau/mass*cosh(eta))
      END;
   parton[1].eta := - parton[2].eta;
   parton[1].rapidity := - parton[2].rapidity;
   EXIT
   END;
i := 1;
WHILE i <= num_tubes DO BEGIN
   { Prediction of the position of the new parton }
   eta1     := parton[i].eta;
   eta2     := parton[i+1].eta;
   rap1     := parton[i].rapidity;
   rap2     := parton[i+1].rapidity;
   eta1_new := eta1 + dtau*tanh(rap1-eta1)/tau;
   eta2_new := eta2 + dtau*tanh(rap1-eta2)/tau;
```

```
    IF  eta1_new < eta2_new  THEN BEGIN
       { Tubes do not overlap }
       temp := parton[i];
       MoveParton(temp);
       IF  (i > 1)
          AND (temp.eta < parton[i+1].eta)
          AND (parton[i-1].eta < temp.eta)
          OR (i = 1)
          AND (temp.eta < parton[i+1].eta)  THEN  parton[i] := temp;
       Inc(i)
       END
    ELSE BEGIN
       { Tubes overlap }
       field_new := field[i+1] + field[i-1] - field[i];
       IF  abs(field_new) = abs(field[i])  THEN BEGIN
          { Exchange of partons }
          temp_eta        := parton[i].eta;
          parton[i].eta   := parton[i+1].eta;
          parton[i+1].eta := temp_eta;
          field[i]        := field_new;
          temp            := parton[i];
          parton[i]       := parton[i+1];
          parton[i+1]     := temp;
          Inc(i, 2);
          OutYoyos
          END
       ELSE BEGIN
          { Parton scattering }
          m1   := parton[i].mass;
          m2   := parton[i+1].mass;
          p    := m1*sinh(rap1) + m2*sinh(rap2);
          e    := m1*cosh(rap1) + m2*cosh(rap2);
          ycms := artanh(p/e);
          WITH  parton[i]  DO
             rapidity := - rapidity + ycms + ycms;
          WITH  parton[i+1] DO
             rapidity := - rapidity + ycms + ycms;
          Inc(i, 2);
          OutYoyos
          END
       END
    END;
 IF  i = num_tubes + 1  THEN BEGIN
    temp := parton[i];
    MoveParton(temp);
    IF  temp.eta > parton[i-1].eta  THEN
       parton[i] := temp
    END
 END; { Dynamics }
```

```
{  DECAYS MODULE  }

PROCEDURE  Tunneling(VAR tube_to_decay: Integer);
{ Tunneling in one tube }
VAR
   left_parton,
   right_parton       {just created partons}
      : PartonType;
   central_point,     {here the virtual particles were born}
   expected_decays,   {expected number of decays}
   init_dist,         {initial distance of a pair that is to be created}
   left_end,          {eta of the left end of the decaying tube}
   left_new,          {eta of the left end of the new tube}
   length,            {length of the decaying tube}
   prob_tunnel,       {probability of tunneling}
   right_end,         {eta of the right end of the decaying tube}
   right_new,         {eta of the right end of the new tube}
   virtual_mass       {transverse mass of the pair that is to be created}
     : Extended;

  FUNCTION  Gauss(mass: Extended): Extended;
  { Gauss distribution }
  VAR
     expon: Extended;
  BEGIN
  expon := pi*mass*mass/STRING_TENSION;
  IF  expon > 30.0  THEN
     Gauss := 0.0
  ELSE
     Gauss := exp(-expon)
  END; { Gauss }

  FUNCTION  VirTrm(Len: Extended): Extended;
  { Transverse mass for a virtual pair }
  VAR
     Pmax: Extended;
  BEGIN
  Pmax := 2.0 * tau * STRING_TENSION * sinh(0.25 * Len);
  IF  Pmax > TRANS_CUTOFF  THEN
     Pmax := TRANS_CUTOFF;
  IF  Pmax <= REST_MASS  THEN BEGIN
     VirTrm := -1.0;
     EXIT
     END;
  VirTrm := sqrt(rest_mass2 + Random*(Pmax*Pmax - rest_mass2))
  END; {VirTrm}
```

```
    FUNCTION  Poisson(ne: Extended): ShortInt;
    { Gives the number of decays: 0 or 1;
       ne = expected value }
    VAR
       n: ShortInt;
       R, thresh: Extended;
    BEGIN
    { Algorithm taken from R. Zielinski book, p.74 }
    thresh := exp(-ne);
    n      := -1;
    R      := 1.0;
    REPEAT
       R := R * Random;
       Inc(n)
       UNTIL  R < thresh;
    Poisson := n
    END; { Poisson }

    PROCEDURE  InsertPair;
    { Inserting of a pair into the decaying tube }
    VAR
       it: Integer;
    BEGIN {InsertPair}
    Inc(num_tubes, 2);
    {shifting the tubes}
    FOR  it := num_tubes  DOWNTO  tube_to_decay+2  DO
       field[it] := field[it-2];
    FOR  it := num_tubes+1  DOWNTO  tube_to_decay+3  DO
       parton[it] := parton[it-2];
    {screening the field}
    field[tube_to_decay+1] := 0;
    {inserting partons}
    parton[tube_to_decay+1] := left_parton;
    parton[tube_to_decay+2] := right_parton;
    OutYoyos;
    IF  NOT GraphOn  THEN
       Write(' Creation at ', tau:6:3, ' ')
    END; {InsertPair}

BEGIN {Tunneling}
left_end     := parton[tube_to_decay].eta;
right_end    := parton[tube_to_decay+1].eta;
length       := right_end - left_end;
virtual_mass := VirTrm(length);
IF  virtual_mass < 0.0  THEN EXIT;
init_dist       := 4.0 * arsinh(virtual_mass/(2.0*STRING_TENSION*tau));
central_point   := left_end + 0.5*init_dist +
                   Random * (length - init_dist);
prob_tunnel     := DEG_FREEDOM * Gauss(virtual_mass);
```

```
expected_decays := prob_tunnel * 0.5 * STRING_TENSION *
                   (length-init_dist) * tau * dtau/pi;

IF  Poisson(expected_decays) = 0  THEN EXIT

left_new  := central_point - 0.5 * init_dist;
right_new := left_new + init_dist;
IF  FirstPair  THEN  FirstPair := FALSE;
IF  SoundOn  THEN BEGIN
   Sound(1000);
   Delay(30);
   NoSound
   END;
IF  GraphOn  THEN BEGIN
   SetColor(TUNNEL_COLOR);
   SetLineStyle(DottedLn, 0, NormWidth)
   Line(Round(left_new*HorizFactor) + MaxX div 2,
        MaxY-Round(tau*VertFactor),
        Round(right_new*HorizFactor) + MaxX div 2,
        MaxY-Round(tau*VertFactor));
   SetLineStyle(SolidLn, 0, NormWidth);
   END;
WITH  left_parton  DO BEGIN
   rapidity := central_point - 0.25 * init_dist;
   mass     := virtual_mass;
   angle    := random * 2.0 * pi;
   eta      := left_new
   END;
WITH  right_parton  DO BEGIN
   rapidity := central_point + 0.25 * init_dist;
   mass     := virtual_mass;
   angle    := pi + left_parton.angle;
   eta      := right_new
   END;
IF  field[tube_to_decay] > 0  THEN BEGIN
   left_parton.quark  := FALSE;
   right_parton.quark := TRUE
   END
ELSE BEGIN
   left_parton.quark  := TRUE;
   right_parton.quark := FALSE
   END;
InsertPair;
Inc(tube_to_decay, 2)
END; {Tunneling}

PROCEDURE  Decays;
VAR
   nt: Integer;
BEGIN
```

```
nt := 1;
REPEAT
   Tunneling(nt);
   Inc(nt, 2)
   UNTIL  nt > num_tubes
END; { Decays }

{  YD-YC MODULE  }

PROCEDURE  ClearSums;
VAR
   i: Integer;
BEGIN
FOR  i := 0  TO  MAX_AVER  DO BEGIN
   AverPt[i]   := 0.0;
   AverMass[i] := 0.0;
   AverNum[i]  := 0
   END
END; { ClearSums }

PROCEDURE SaveResults;
VAR
   i, j: Integer;
   t: Extended;
BEGIN
Assign(MassFile, MASS_FILE_NAME);
Assign(PtFile, PT_FILE_NAME);
Assign(NumFile, NUM_FILE_NAME);
Rewrite(MassFile);
Rewrite(PtFile);
Rewrite(NumFile);
FOR  i := 0  TO  MAX_AVER  DO BEGIN
   t := i * aver_pass_ratio;
   Writeln(MassFile, t:6:3, ' ', 200.0*AverMass[i]/AverNum[i]);
   Writeln(PtFile, t:6:3, ' ', 200.0*AverPt[i]/AverNum[i]);
   Writeln(NumFile, t:6:3, ' ', (1.0*AverNum[i])/num_events);
   END;
Close(MassFile);
Close(PtFile);
Close(NumFile);
FOR  i := 1  TO  MAX_RAP  DO BEGIN
   Assign(RapFile, RAP_FILE_NAME+'.'+Int2Str(i));
   Append(RapFile);
   WITH  ListRap[i]  DO BEGIN
      Writeln(RapFile, num);
      FOR  j := 1  TO  num  DO
         Writeln(RapFile, rap[j])
      END;
   Close(RapFile)
   END
```

```
END; { SaveResults }

PROCEDURE  OutRapidities(RapFileNum: Integer);
VAR
   i : Integer;
BEGIN
i := 1;
WITH  ListRap[RapFileNum]  DO BEGIN
   num := (num_tubes + 1) div 2;
   WHILE  i <= num_tubes  DO BEGIN
      rap[(i+1) div 2] := YoyoRapidity(i);
      Inc(i, 2)
      END
   END
END; { OutRapidities }

FUNCTION  YoyoMass(i: Integer): Extended;
{ Rest mass of a yo-yo = sqrt(E*E - Pz*Pz - Pt*Pt) }
VAR
   y1, y2,
   m1, m2,
   eta1, eta2,
   particle_energy,
   field_energy,
   total_energy,
   particle_momentum,
   field_momentum,
   total_momentum: Extended;
BEGIN
WITH  parton[i]  DO BEGIN
   eta1 := eta;
   y1   := rapidity;
   m1   := mass;
   END;
WITH  parton[i+1]  DO BEGIN
   eta2 := eta;
   y2   := rapidity;
   m2   := mass;
   END;
particle_energy := m1*cosh(y1) + m2*cosh(y2);
field_energy := tau*string_tension*(sinh(eta2) - sinh(eta1));
total_energy:= particle_energy + field_energy;
particle_momentum := m1*sinh(y1) + m2*sinh(y2);
field_momentum := tau*string_tension*(cosh(eta2) - cosh(eta1));
total_momentum := particle_momentum + field_momentum;
YoyoMass := sqrt(sqr(total_energy) -
            sqr(total_momentum) - sqr(YoyoPt(i)))
END; { YoyoMass }

FUNCTION  YoyoPt(i: Integer): Extended;
```

```
{ Transverse momentum of a yo-yo }
VAR
   m1, m2,
   pt1, pt2,
   phi1, phi2: Extended;
BEGIN
WITH  parton[i]  DO BEGIN
   m1   := mass;
   pt1  := sqrt(m1*m1 - rest_mass2);
   phi1 := angle
   END;
WITH  parton[i+1]  DO BEGIN
   m2   := mass;
   pt2  := sqrt(m2*m2 - rest_mass2);
   phi2 := angle
   END;
YoyoPt := sqrt(sqr(pt1) + sqr(pt2) + 2.0*pt1*pt2*cos(phi1-phi2))
END; { YoyoPt }

FUNCTION  YoyoRapidity(i: Integer): Extended;
{ Rapidity of a yoyo }
VAR
   y1, y2,
   m1, m2,
   eta1, eta2,
   particle_energy,
   field_energy,
   total_energy,
   particle_momentum,
   field_momentum,
   total_momentum: Extended;
BEGIN
y1 := parton[i].rapidity;
y2 := parton[i+1].rapidity;
m1 := parton[i].mass;
m2 := parton[i+1].mass;
eta1 := parton[i].eta;
eta2 := parton[i+1].eta;
particle_energy := m1*cosh(y1) + m2*cosh(y2);
field_energy := tau*string_tension*(sinh(eta2) - sinh(eta1));
total_energy:= particle_energy + field_energy;
particle_momentum := m1*sinh(y1) + m2*sinh(y2);
field_momentum := tau*string_tension*(cosh(eta2) - cosh(eta1));
total_momentum := particle_momentum + field_momentum;
YoyoRapidity := artanh(total_momentum/total_energy);
END; { YoyoRapidity }

PROCEDURE  VerifyYoyos;
{ Verification: if the rest mass of a yo-yo is between one and
  two meson masses }
```

```
VAR
   i, j: Integer;
   Pos: Extended;
BEGIN
i := 1;
REPEAT
   IF  (YoyoMass(i) < 2*MESON_MASS)
      AND (YoyoMass(i) > MESON_MASS)  THEN BEGIN
      IF  SoundOn  THEN BEGIN
         Sound(100);
         Delay(20);
         NoSound;
         END;
      Dec(num_tubes, 2);
      IF  GraphOn  THEN BEGIN
         SetColor(REMOVED_COLOR);
         Pos := 0.5*(parton[i].eta + parton[i+1].eta);
         Circle( Round(HorizFactor*Pos) + MaxX div 2,
                 MaxY  - Round(VertFactor*tau), 10)
         END
      ELSE
         Write('    Meson at ', tau:6:3, '  ');
      IF  num_tubes < 1  THEN BEGIN
         tau := 2.0*T_FINISH;
         OutYoyos;
         EXIT
         END;
      FOR  j := 1  TO  num_tubes+1  DO
         parton[j] := parton[j+2];
      FOR  j := i  TO  num_tubes  DO
         field[j] := field[j+2];
      OutYoyos
      END;
   Inc(i, 2)
   UNTIL  i > num_tubes
END; { VerifyYoyos }

FUNCTION  SumMass: Extended;
VAR
   sum: Extended;
   i: Integer;
BEGIN
IF  num_tubes < 1  THEN BEGIN
   SumMass := 0.0;
   EXIT
   END;
IF  num_tubes = 1  THEN
   sum := YoyoMass(1)
```

```
ELSE BEGIN
   sum := 0.0;
   i := 1;
   WHILE  i <= num_tubes  DO BEGIN
      sum := sum + YoyoMass(i);
      Inc(i, 2)
      END
   END;
SumMass := sum
END; { SumMass }

FUNCTION  AverageMeVMass: Extended;
BEGIN
IF  num_tubes = -1  THEN
   AverageMeVMass := 0.0
ELSE
   AverageMeVMass := 200.0*SumMass/((num_tubes+1) div 2) ;
END; { AverageMeVMass }

FUNCTION  SumPt: Extended;
VAR
   sum: Extended;
   i: Integer;
BEGIN
IF  num_tubes < 1  THEN BEGIN
   SumPt := 0.0;
   EXIT
   END;
IF  num_tubes = 1  THEN
   sum := YoyoPt(1)
ELSE BEGIN
   sum := 0 0;
   i := 1;
   WHILE  i <= num_tubes  DO BEGIN
      sum := sum + YoyoPt(i);
      Inc(i, 2)
      END
   END;
SumPt := sum
END; { SumPt }

FUNCTION  AverageMeVPt: Extended;
BEGIN
IF  num_tubes = -1  THEN
   AverageMeVPt := 0.0
ELSE
   AverageMeVPt := 200.0*SumPt/((num_tubes+1) div 2) ;
END; { AverageMeVPt }

{  CONTROL MODULE  }
```

```
PROCEDURE  Quit;
VAR
   i: Integer;
BEGIN
IF  GraphOn  THEN
   CloseGraph;
Halt
END; { Quit }

PROCEDURE  KeyControl;
VAR
   i: Integer;
BEGIN
IF  NOT KeyPressed  THEN EXIT;
CASE  ReadKey  OF
   ^C,
   #27: Quit;
   'p',
   'P',
   ' ': BEGIN
          IF  NOT GraphOn  THEN
             Writeln('Suspended - Press any key...');
          IF  ReadKey IN [#27, ^C]  THEN
             Quit
          END;
   'r',
   'R': BEGIN
          tau := 2.0*T_FINISH;
          Restart := TRUE;
          IF  SaveOn  THEN BEGIN
             ClearSums;
             InitRapFiles;
             END
          END;
   't',
   'T': IF  NOT GraphOn  THEN
          Write(' Actual time ', tau:6:3, ' ');
   END
END;  {KeyControl}

PROCEDURE  VerifyPass;
BEGIN
IF  tau > rap_pass_ratio*rap_pass  THEN BEGIN
   OutRapidities(rap_pass);
   Inc(rap_pass)
   END;
```

```
IF  tau > aver_pass_ratio*aver_pass  THEN BEGIN
   AverPt[aver_pass]    := AverPt[aver_pass] + SumPt;
   AverMass[aver_pass] := AverMass[aver_pass] + SumMass;
   AverNum[aver_pass]  := AverNum[aver_pass] + cur_num;
   Inc(aver_pass)
   END
END; { VerifyPass }

{=============================================================}

BEGIN  { PROGRAM BODY }
InitProgram;
REPEAT
   InitEvent;
   WHILE  tau <= T_FINISH  DO BEGIN
      SetTimeStep;
      Dynamics;
      Decays;
      IF  VerifyOn  THEN
         VerifyYoyos;
      KeyControl;
      tau := tau + dtau;
      IF  SaveOn  THEN
         VerifyPass;
      END;
   EndEvent
   UNTIL  num_events > MAX_EVENTS;
Quit
END.  { OF PROGRAM }
```

IFJ Kraków, zam. 121/90 130 egz.