FOAM
Multi-Purpose
Celular Monte Carlo Generator

S. Jadach

Institute of Nuclear Physics, Kraków, Poland

Outline:

- **General introduction.**
- **Foam algorithm and new MCell (Mega-Cell MC).**
- **Numerical tests.**
- **Conclusions.**

Slides, program sources: http://home.cern.ch/jadach

Just to give an idea...

Modern PC desktop can create within 15minCPU n_C =5k-50k cells and subsequently generate N =10M events for relatively strongly peaked n-dimesional integrand, $n < 12$.

With $\frac{\sigma}{<W>} = 0.3$ that provides 3-digit value of the integral, $\frac{\delta I}{I} = \frac{\sigma}{\sqrt{N}} \simeq 10^{-3}$,
or alternatively 3M events with $W = 1$,
assuming $\frac{W_{\max}}{<W>} = 0.3$.

Not bad!! What are the key limitations?

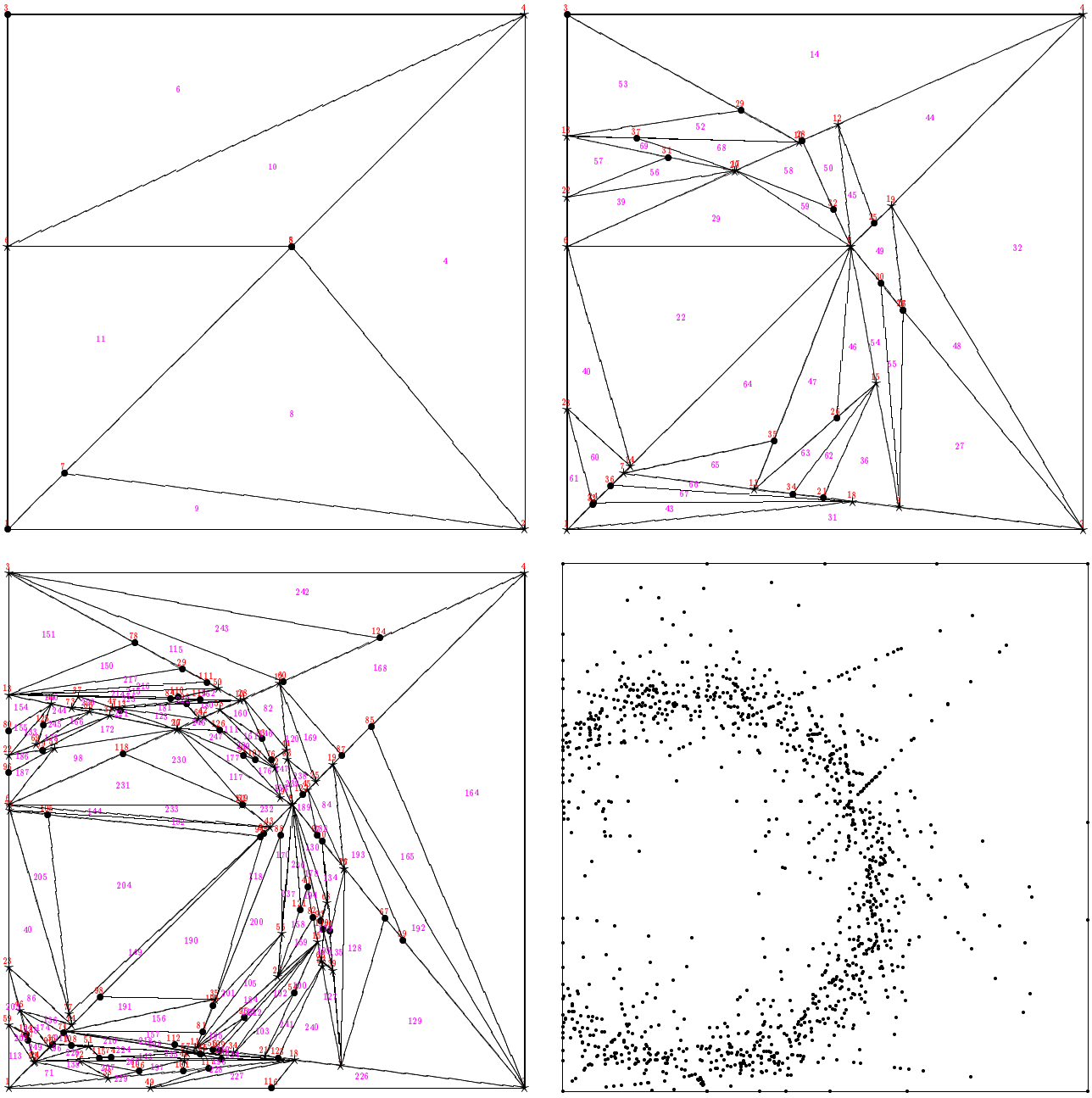
Memory: $\sim 16 \times n \times n_C$ Bytes. <50MB.

CPU time: $\sim n_C$/1k Minutes CPU. <1h.

Efficient algorithm of build-up of cells:
maximizing $\frac{<W>}{W_{\max}}$ and/or minimizing $\frac{\sigma}{<W>}$.

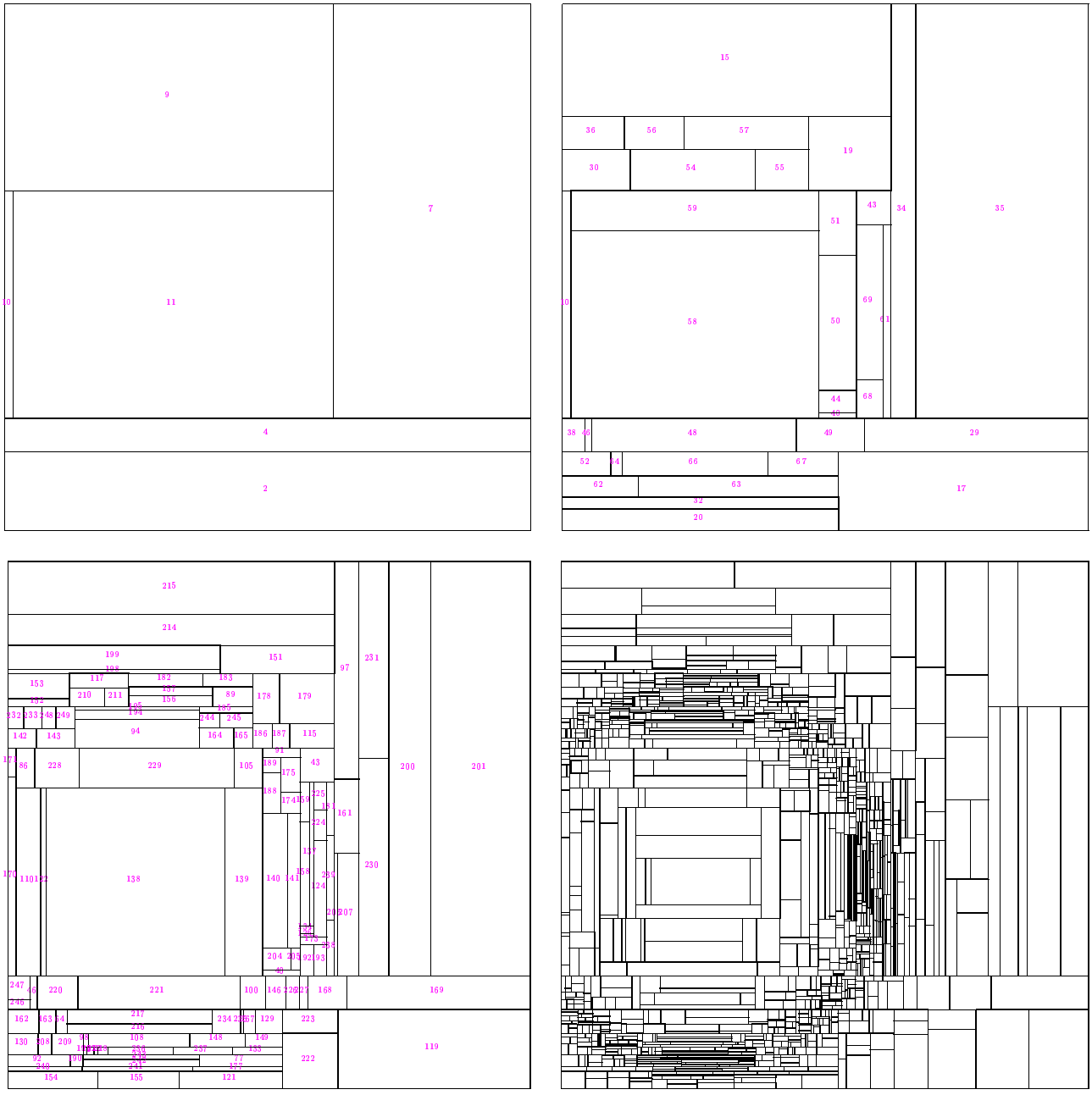
Algorithms based on BINARY SPLIT of cells seems to fulfill the constraints. Parent cells kept in record $\longrightarrow$
$\longrightarrow$ hierarchical data organization (tree-like linked list).

Evolution of simplical foam



Number of cells= 10, 70, 250, 2500.

Evolution of hyper-cubic. foam

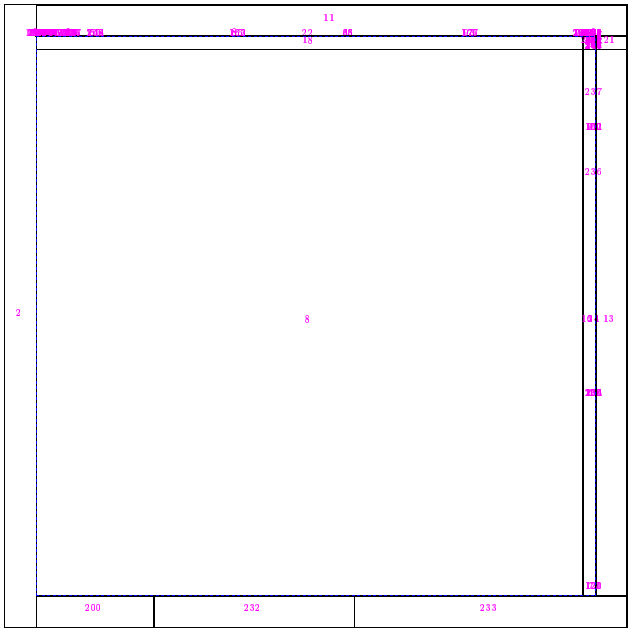


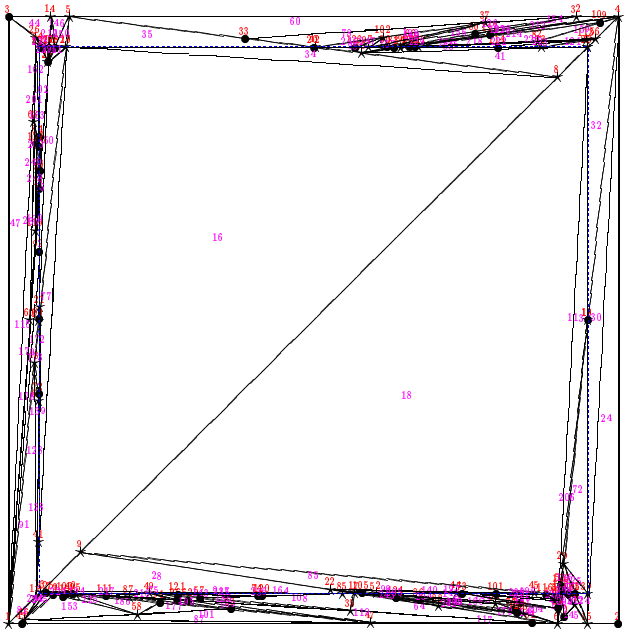
Number of cells= 10, 70, 250, 2500.

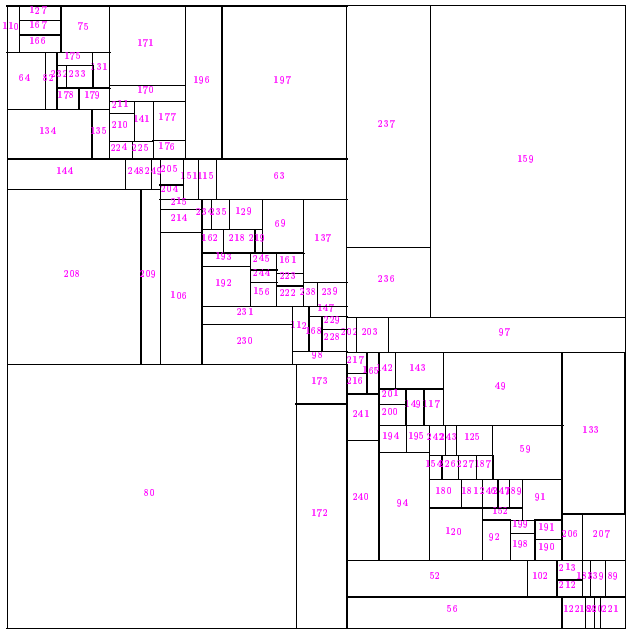
S. Jadach

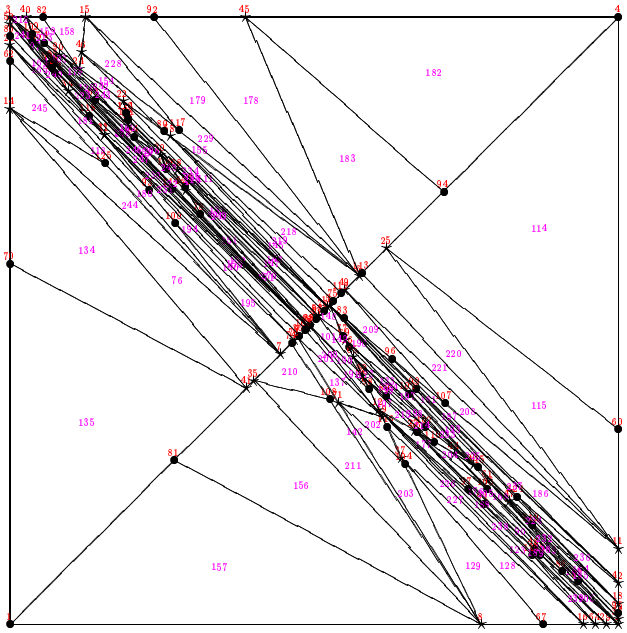
Kraków, September, 2001

Void and Diagonal









Number of cells= 250.

S. Jadach

Kraków, September, 2001

Terminology&Notation

Integration domain $\Omega = [0,1]^n$ is split into Cells $\omega_1 ... \omega_k$. Two AUXILIARY distributions $\rho'(x)$ and $\rho_{loss}(x)$ related to integrand $\rho(x)$, which are CONSTANT over each cell are constructed. How? See next slides.

TWO STEP SCENARIO:

(1) Foam of cells BUILD-UP:

$\rho_{loss}(x)$ is evolving with the foam, it DRIVES the division of cells, $I_{loss} = \int \rho_{loss} d^n x$ is minimized in the process.

(2) MC generation:

Events are generated according to $\rho'(x)$.

$I' = \int \rho' d^n x$ is known exactly.

$I = I' < W >'$ where $W = \rho/\rho'$,

The average $< .. >'$ means over events generated according to ρ'

OUR AIMS:

Minimize rejection (loss) rate $L = 1 - \frac{<W>}{W_{\max}}$, OR

Minimize variance $\sigma = \sqrt{< W^2 > - < W >^2}$.

binary split of a cell

How do we minimize $L = 1 - \frac{<W>}{W_{\max}}$?

DEF: $\rho'(x) \equiv \max_{y \in Cell_i} \rho(y)$, for $x \in Cell_i$.

is a maximum of $\rho(x)$ within each cell:

Maximum over cell found experimentally using $\sim$1000 MC events.

We MINIMIZE during the foam construction:

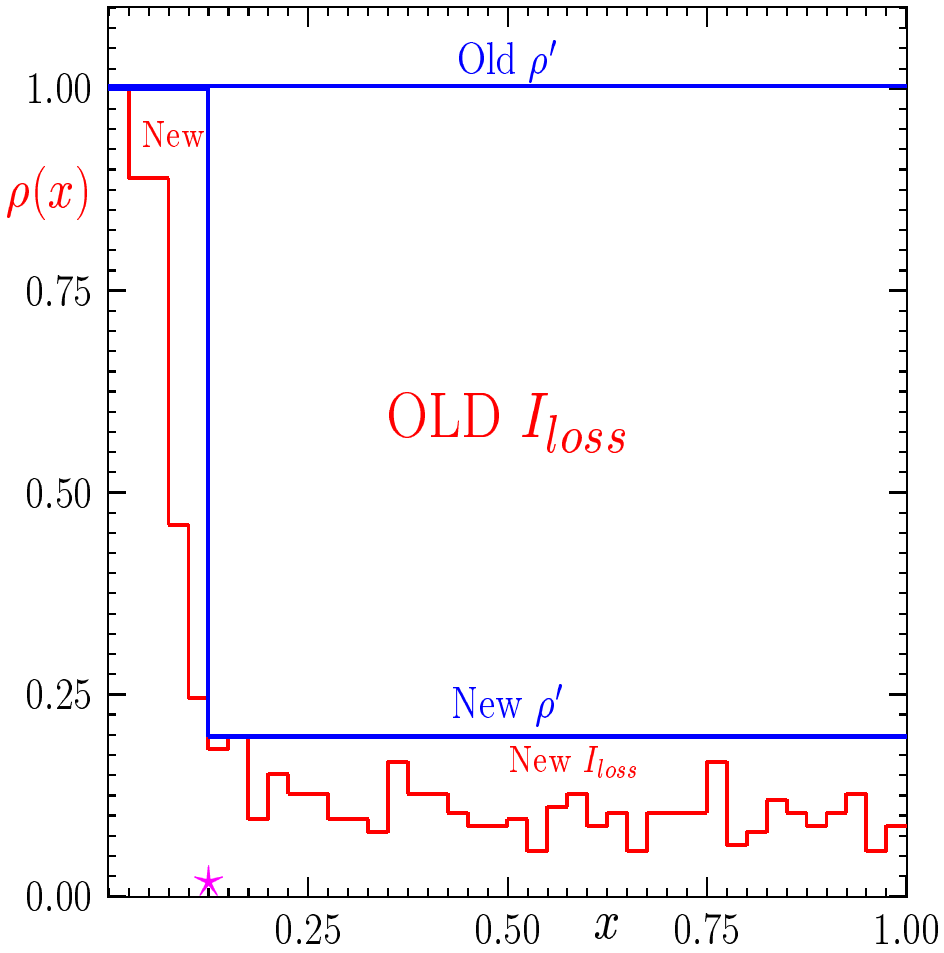
$$I_{loss} = \int d^n x \, [\rho'(x) - \rho(x)] \equiv \int d^n x \, \rho_{loss}(x)$$

Each SPLIT of a Cell: $\omega \longrightarrow \omega' + \omega''$
should obey: $I^{\omega'}_{loss} + I^{\omega''}_{loss} << I^{\omega}_{loss}$.

RULES to follow:

- For next split we choose a CELL with the biggest I_{loss}.
- Position/direction of a plane dividing a parent CELL into two daughter CELLs is chosen to minimize total I_{loss}.

Binary split 1-dim. example



$\rho(x)$ is integrand

Old ρ' for parent cell (majorizing $\rho(x)$)

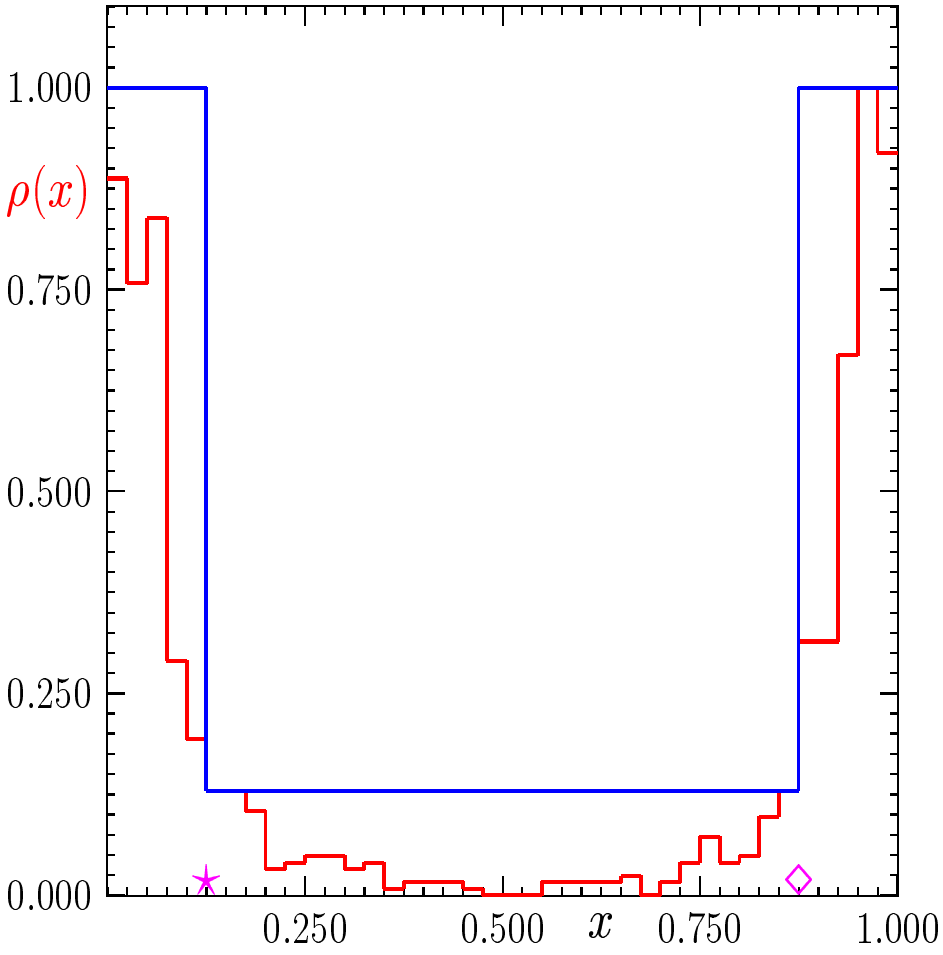
New ρ' for two daughter cells

OLD I_{loss} all area above red line, for the parent cell

New I_{loss} between red line and New ρ', for 2 daughters.

Obviously $I'_{New} < I'_{Old}$, the division point $\star$ is chosen to MINIMIZE THE LOSS functional/integral I_{loss}!

Binary split 1-dim. example



The two-maximum case is a little bit more complicated.

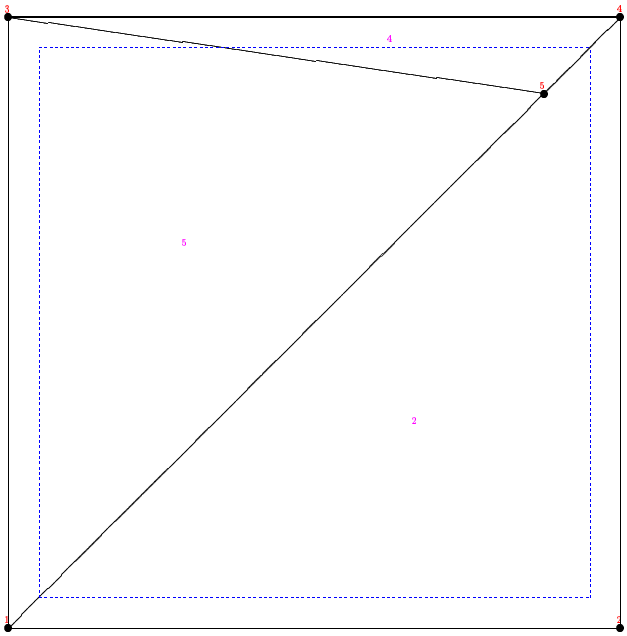
We follow simple prescription:

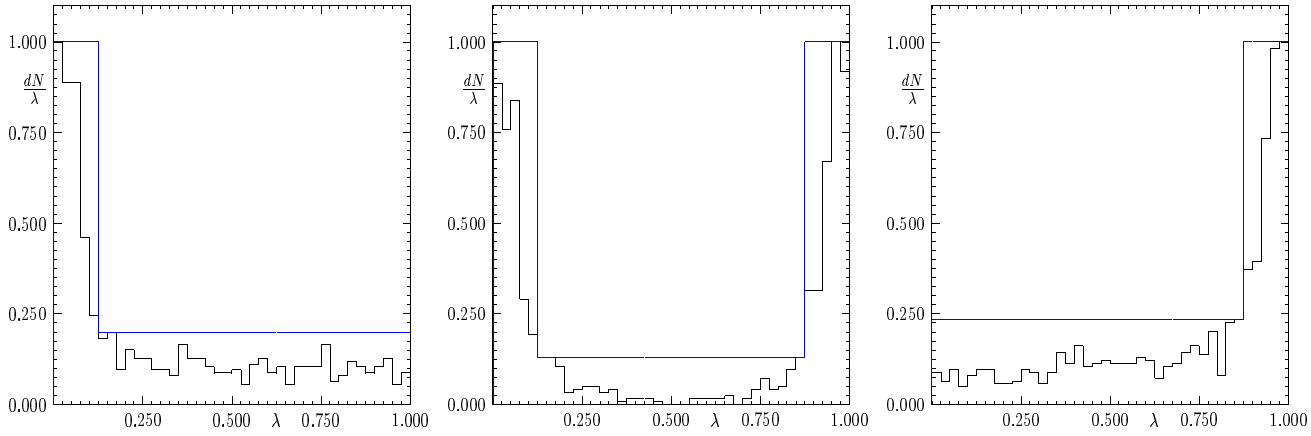
Determine the blue line MAXIMIZING I_{loss}, see picture,

and apply the division procedure twice (obtaining 3 cells).

Division points are $\star$ and $\diamond$

Binary split 2-dim. example





Integrand covers narrow strip along edges. **Intend to split upper triangle.**

Generate 1000 w-ted events and project them on 3 sides of the parent triangle.

3 Projections are analyzed.

We choose the one with the smallest LOSS functional I_{loss} (middle plot).

Two resulting daughter triangles are shown.

The DIVISION line EXPELLED from the "empty" area in the middle!!!

Minimizing variance

How to Minimize $\frac{\sigma^2}{<W>^2} = \frac{<W^2>}{<W>^2} - 1$?

DEF: $\rho'(x) \equiv V_i \times \sqrt{< \rho^2 >_i}$, for $x \in Cell_i$.

The $< .. >_i$ is for uniformly distributed points $\in Cell_i$.

Cells with bigger contribution to total variance, "promoted" in MC generation.

We MINIMIZE during the foam build-up:

$I_{loss} = \int d^n x \ \rho_{loss}(x)$ where

$\rho_{loss}(x) \equiv V_i \times \sqrt{< \rho^2 >_i - < \rho >_i^2}$, for $x \in Cell_i$.

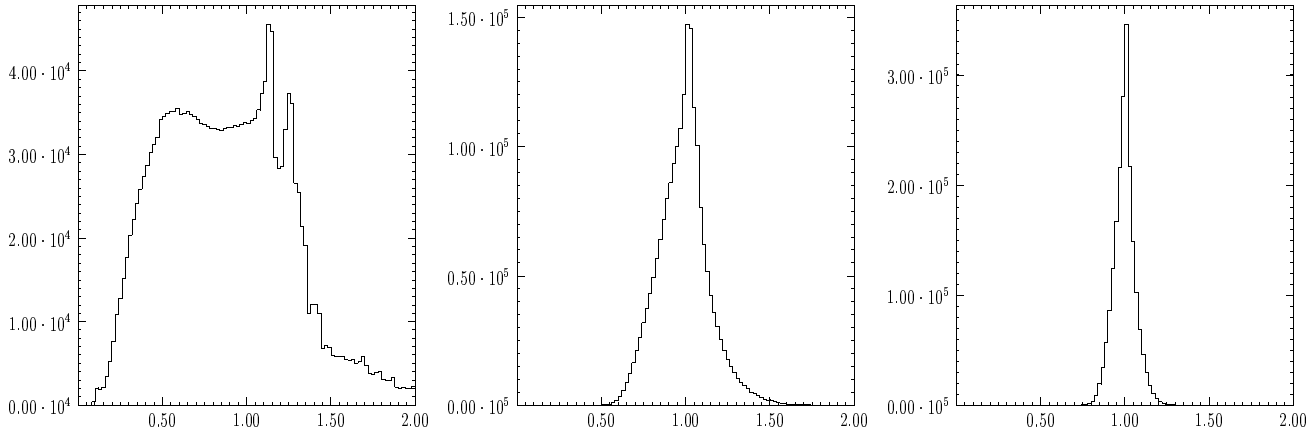
The algorithm avoids dividing cells in the areas where the integrand is flat,

the dispersion of the function over the cell is small.

All the algorithm of cell division remain the same.

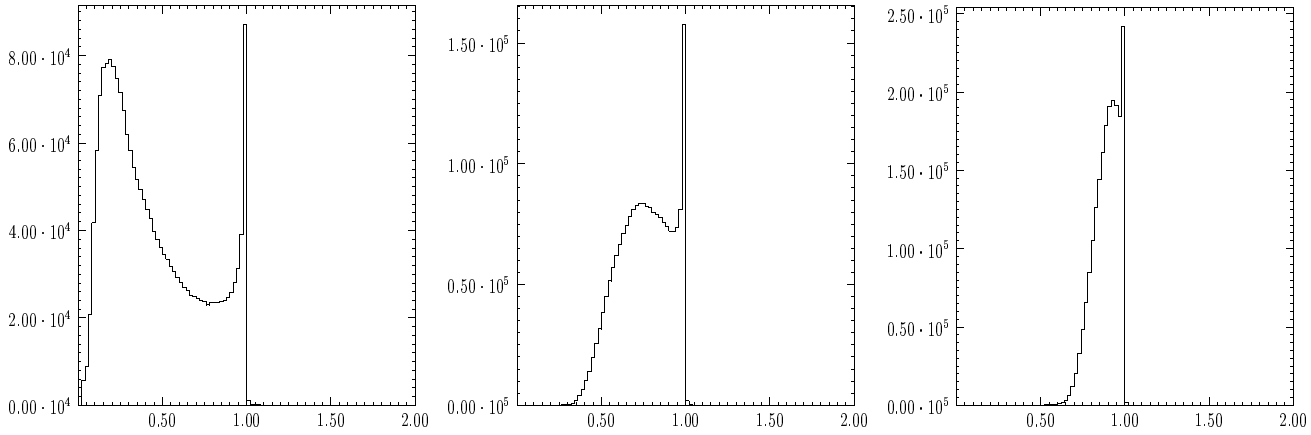
We did less effort to optimize this mode of the program.

Weight distribution: minimization of variance



Number of cells: 200, 2k, 20k

Weight distribution: minimization of rejection rate



Number of cells: 200, 2k, 20k

Why hypercubes as cells? Why not?

Consideration in favor of simplexes:

MEMORY: In n-dimensions, simplical foam (binary division) requires only one vertex vector $\vec{v}$ for each new cell!

That means: $8n$ Bytes/Cell.

Every new hypercubic at n=15 has 32k vertices :-(

However, it can be parameterized using only 2 vectors: $\vec{v}$ for "lower left corner" vertex position, and $\vec{d}$ defining n lengths along each direction :-) Hence only $16n$ Bytes/Cell.

(The interior of the hypercube is $x^i = v^i + \lambda_i d^i, 0 < \lambda_i < 1$.)

Consideration in favor of hypercubes:

Simplexes limited to $n < 6$ and N_{Cell} <5k because:

(1) Already for n=10 initial h-cube $\rightarrow n! \sim 10^6$ simplexes,

(2) geometry become CPU consuming (determinants).

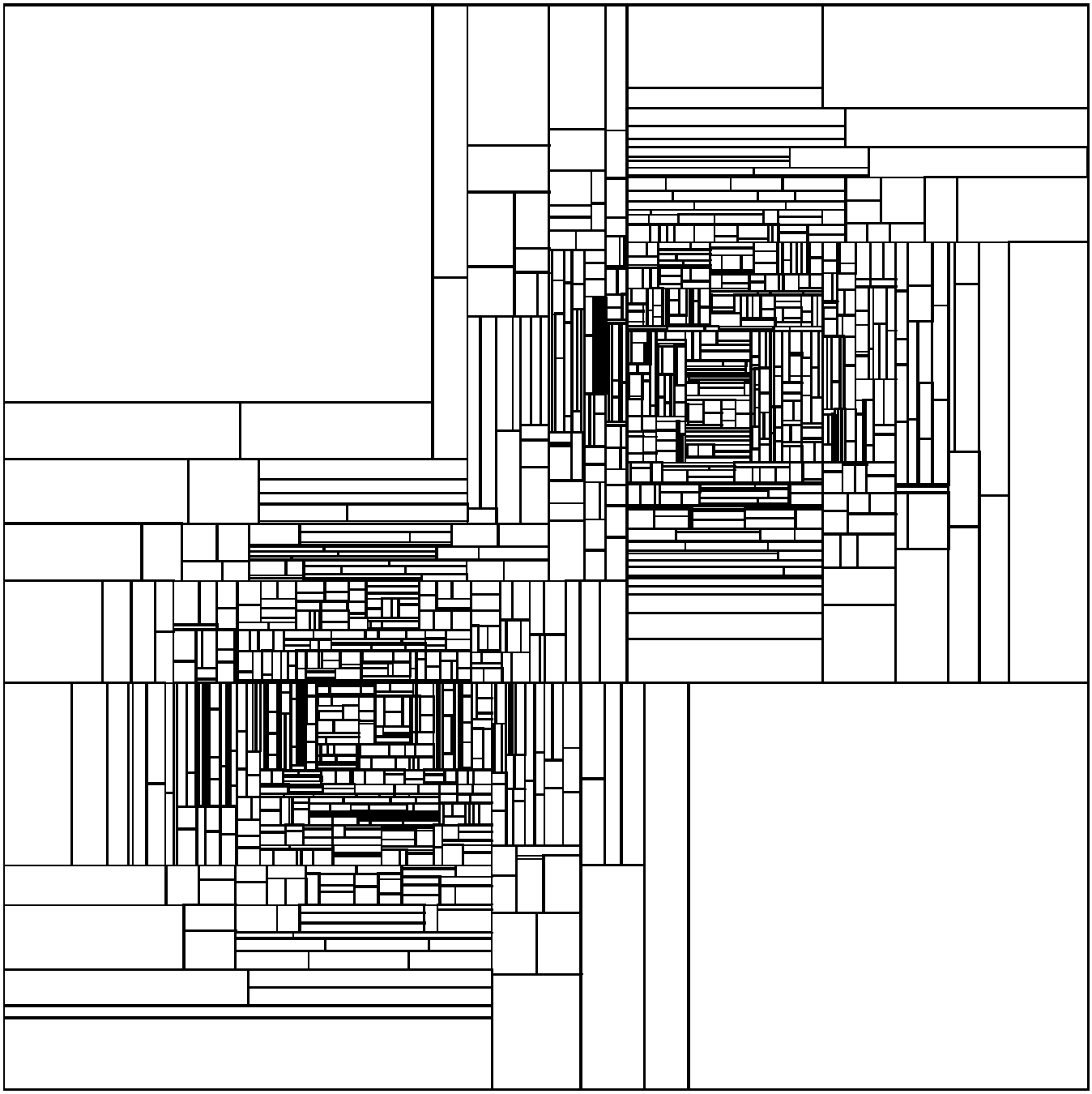
(2) memory limitation,

However, memory consumption for h-cubes can be limited to below 50Bytes/Cell indep. of n, albeit with some tolerable CPU overhead, and no determinants :-))

BOTTOM LINE:

With hypercubes we can reach 1Mega Cells at $n \sim 15$.

Hyper-cubics = rectangles in 2-dimen.



The "Camel" Double-Gauss function of Peter Lepage.

Miracle of $\sim$50Bytes/Cell

Two essential ingredients:

- **$\vec{v}$ and $\vec{x}$ are not stored but (re)calculated using the division parameter λ and the index i of the edge along which the division occurred, all over $\sim \ln_2 N_{Cell}$ steps, up to first Cell.**
- **Furthermore parameter $\lambda = j/n_{bin}$ can be stored as a short inter j (2Bytes sufficient).**

Each cell has several other attributes like I', I_{loss} (16B), pointers to parent and daughter cells (12B), status (8B) etc., altogether ~ 50B/Cell.

Can be easily reduced to 32B/Cell if necessary.

BOTTOM LINE:

The way to 1Mega Cells is open.

We hit CPU barrier instead. See next slide.

CPU barrier: one quick fix is found

Playing with hypercubical code MCell for N_c ∼100k Cells at $n \sim 10$ dimensions shows that the obtained efficiency $\frac{<W>}{W_{\max}}$ and variance σ is improved ESSENTIALLY by the INCREASING of N_c.

However, another important factor is the number of MC events used to explore each newly created Cell N_{samp}.

A necessary value of N_{samp} increases for higher n.
$\longrightarrow$ Dramatic increase of total CPU time for the foam build-up, which is $T \sim n \times N_c \times N_{samp}$.

The trick limiting the above effect:

During MC exploration of a new cell continuously monitor the no. of accumulated effective $W = 1$ events: $N_{eff} = \frac{(\sum w_i)^2}{\sum w_i^2}$ and stop when $N_{eff}/n_{bin} > 25$. n_{bin} is the number of bins in the histogram used to estimate the best division direction/edge and parameter.

The increase of N_{samp} is not wasted for cells in which integrand is already not varying too much. We are now able to push input N_{samp} to high values, with little increase of CPU time.

Program usage very easy (f77)

```
*-------------------------------------------------------
DOUBLE PRECISION   Density
EXTERNAL           Density
CALL MCellA_SetkDim(       3)   ! number of dimensions
CALL MCellA_SetnBuf(    2000)   ! no. of cells
CALL MCellA_SetnSampl(   500)   ! no. MC ev./cell (init.)
CALL MCellA_Initialize(Density) ! initialization
DO loop = 1, 200000
  CALL MCellA_MakeEvent(Density)    ! generate MC event
  CALL MCellA_GetMCvector(MCvector) ! get MC event vector
  CALL MCellA_GetMCwt(MCwt)         ! get MC event weight
  CALL GLK_Fil1(1000, MCwt,1d0)     ! histogramming
ENDDO
CALL MCellA_Finalize(MCresult,MCerror) !get integr.+err.
*-------------------------------------------------------
```

User has to set only the number of dimensions **kDim**.

The other input variables are already preset for the user,

thus calling setters for them is optional.

User needs to provide his own integrand function,

which is here **Density**.

Program usage in C++ also easy

```
//-----------------------------------------------------
  Foam FoamX;                       // Create Simulator
  FunTest Density1(FunType);        // Create integrand fun.
  FoamX.SetnDim(        nDim);      // Dimension simplical
  FoamX.SetkDim(        kDim);      // Dimension hyp-cubical
  FoamX.SetnCells(      nCells);    // No. of cells
  FoamX.Initialize( &Density1 );    // Initialize Foam cells
  double *MCvect =new double[nDim+kDim];
  for(long loop=0; loop<NevTot; loop++){
    FoamX.MakeEvent();              // Generate MC event
    FoamX.GetMCvect( MCvect);       // Get MC event
    MCwt=FoamX.GetMCwt();           // Get MC weight
  }
  double MCresult, MCerror;
  FoamX.Finalize( &MCresult, &MCerror); // get integral
//-----------------------------------------------------
```

User has to set only the number of dimensions **kDim, nDim**.

User needs to provide his own integrand function,

which is here **Density**. Its type **FunTest** has to inherit from

abstract class **FIntegrand**.

Programs

MCell v2.02, f77:

with up to 1Mega cells is specialized for higher dimensions, $n \leq 20$, hypercubical cells only.

Foam v2.02, f77 :

with $N_c \leq 10000$ cells, is aimed for up to six-dimensions.
It is upgraded and improved:
both simplices ($n \leq 5$) and hypercubes ($n \leq 10$) available.

Foam v2.02, c++ :

unlimited number of cells N_c and dimension nDim+kDim.
Both simplices and hypercubes available.
Includes hypercubical algorithm MCell with low memory consumtion as default option.

Foam v2.02, c++: bbased on ROOT package (R. Brun et.el.)

Other improvements:

- Cells can be simultaneously simplices in n-dimensions and hypercubics in k-dimensions.
- It is possibile to start FOAM from SINGLE simplex instead of unit hyper-cubic.

Tests of MCell/Foam at low dimensions

Three 2-dimensional testing functions:

$$f_a = 1 - \Theta(0.5 - |x_1 - 0.5| - \gamma)\ \Theta(0.5 - |x_1 - 0.5 - \gamma|),\ \gamma = 0.05,$$

$$f_b = \frac{1}{4\pi R^2}\ \frac{\gamma}{\pi[(R - \sqrt{(x_1 - 0.25)^2 + (x_2 - 0.40)^2})^2 + \gamma^2)]},$$

$$\gamma = 0.02,\ R = 0.35$$

$$f_c = \frac{\gamma}{\pi[(x_1 + x_2)^2 + \gamma^2)]},\ \gamma = 0.02.$$

Functions, 2-dimens.	Foam	MCell	VEGAS
$f_a(x_1, x_2)$ (diagonal ridge)	0.94	0.74	0.03
$f_b(x_1, x_2)$ (circular ridge)	0.80	0.80	0.16
$f_c(x_1, x_2)$ (edge of square)	1.00	1.00	0.53

Results from Foam/MCell are for 5000 cells (2500 active cells) and cell exploration based on 200 MC events/cell.

Efficiencies are $< W > /W^{\varepsilon}_{\max}$ with ε=0.0005.

Functions, 3-dimens.	Foam	MCell	VEGAS
f_a (thin diagonal)	0.56	0.62	0.002
f_b (thin sphere)	0.27	0.50	0.11
f_c (surface of cube)	0.66	1.00	0.30

The main point is that efficiency of MCell can be trivially increased to 100% by increasing no of cells, while for VEGAS we can NEVER do better.

Tests of MCell at higher dimensions

Double-hump CAMEL test-function of P. Lepage.

n	N_{Cell}	$\frac{N_{MC}}{Cell}$	Effic.	$\frac{\sigma}{<W>}$	I	δI
6	10k	0.3k	0.0387	1.18	0.99789	0.00083
6	10k	1k	0.1275	1.22	1.00007	0.00086
6	10k	10k	0.1231	1.28	1.00070	0.00090
6	10k	33k	0.1325	1.21	1.00040	0.00086
6	100k	0.3k	0.2574	0.75	0.99939	0.00053
6	100k	1k	0.2626	0.77	1.00030	0.00054
6	100k	3k	0.2750	0.76	1.00016	0.00053
6	100k	10k	0.2681	0.78	1.00022	0.00055
9	100k	1k	0.0336	1.74	0.99285	0.00123
9	100k	3k	0.0435	1.84	0.99765	0.00130
9	100k	10k	0.0407	1.89	0.99871	0.00133
12	100k	1k	0.0037	3.13	0.84019	0.00221
12	100k	3k	0.0038	3.33	0.48950	0.00235
12	100k	10k	0.0036	4.48	0.99512	0.00317
12	100k	100k	0.0032	5.20	1.00003	0.00368
12	1000k	10k	0.0055	3.89	0.99779	0.00275

Efficiency depends mainly on number of cells M_{Cell}.

Number of MC trials per Cell cannot be too small.

Conclusions

Summary

- **New f77 program MCell for 1 Mega cells and n$<$20 is now available, in addition to upgraded f77 Foam.**
- **C++ version of the Foam with dynamic memory allocation now also available! Also version with ROOT.**
- MCell/Foam are are general purpose Monte Carlo simulators/integrators based on a self-adapting FOAM OF CELLS.
- Cells are simplices and/or hyper-cubics in Foam (f77 and c++), only hyper-cubics in MCell (f77).
- f77 MCell (and c++ Foam) is able to cope with CPU and MEMORY barriers at higher dimensions, as numerical tests demonstrate.
- **Looking forward for a KillerApp!**

Postscript of these slides: **http://home.cern.ch/jadach/public/MCell-talk.ps.gz**