# SpcTools: A simple visualization library for C++ programs

SPC group, CUI, University of Geneva

January 2003

## 1 Introduction

This library, called SpcTools, contains some C++ classes that are useful to visualize images and to draw simple graphs by opening a drawing window on the computer screen and writing pixels or lines in it. It is a subset of the PELABS library developed by Alexandre Dupuis.

The classes of the library use the X11 library and can be used from Unix and Linux systems. The functionality is basic but should be sufficient for simple applications. As a limitation, note that the drawing window is not automatically refreshed if another window is dragged over it. Also, at the current stage, the axis in plot are not labeled.

## 2 Images and plots

### 2.1 The drawing window

In order to visualize a image or to draw a graph, the first step is to open a window on the computer screen.

Using the SpcTool library, such a drawing window (called here `myWindow`) is declared in a C++ program, as

```
GraphicsInterface myWindow;
```

It requires the header file

```
#include "GraphicsInterface.hh"
```

defined in the library.

To make the drawing window actually appear on the screen, use the instruction

```
myWindow.open(nz,nx);
```

where `nz` and `nx` are integers variables representing, in pixels, the vertical and horizontal size of the drawing window. For instance

```
myWindow.open(100,200);
```

opens a 200-pixel large window and 100-pixel high.

Many windows can be open simultaneously, with different names (e.g. `GraphicInterface window1, window2;` ). When a drawing window is no longer needed, it can be closed with

```
myWindow.close();
```

## 2.2 How to display an image

An image can be drawn on a drawing window. To this end, the image must be stored as a `Matrix` object declared as follows:

```
Matrix<unsigned char> image;
```

Its size must match that of the drawing window:

```
image.setDimension(nz,nx);
```

Each pixel of the image is coded as a `char` type. The content of pixel `(i,j)` of the image is set to a chosen value (for instance the value of 2) in the following way:

```
image.set(j,i,(unsigned char) (2)); // 2 produces a red pixel
```

There are 256 possible colors, ranging from 0 to 255. The value 0 is white, 1 is black, 2 is red and, then, there is continuous palette. Run the file `TestGI` in the `SpcTool/GI` directory to see the color palette.

In order to display `image` on the drawing window, use the `DrawMatrix` method:

```
mywindow.drawMatrix(image);
```

An example of application is given in section 5. The program `parity.cc` implements a cellular automata and displays its evolution after each time step.

## 2.3 Saving an image as a file

It is possible to save on the disk the contents of a drawing window as a ppm file. The format ppm is a simple format to save pixel images. It can then be converted to other formats using programs such as `convert file.ppm file.ps`, or `display`, etc.

For instance, if `image` contains the image and `myWindow` is a drawing window, the instruction `myWindow.printInto(``myImage.ppm'',image);` will create the file `myImage.ppm` and print the `image` matrix in it.

## 2.4 Simple plot

The SpcTools library offers some possibilities to draw lines or graphs on a plotting window. The following header files are then needed:

```
#include "GraphicsInterface.hh"
#include "Point.hh"
#include "PlotGI.hh"
```

For instance, a plot window (called here `plot1`) is declared and opened with

```
PlotGI plot1(256,256,0.,1.,0.,1.);
```

where the first two values `nx`, `ny` are the window size in pixels and the last four values `xmin`, `xmax`, `ymin` and `ymax` are `float` that indicate the desired user coordinates `(xmin,ymin)` of the lower left corner and `(xmax,ymax)` of the upper right corner. These coordinates can be changed at any time using the method `setXY`. For instance,

```
plot1.setXY(0.,64.,-1.,1.);}
```

defines a plot region with `x` ranging between 0 and 64 and `y` between -1 and 1.

To draw a straight line from `(x0,y0)` to `(x1,y1)` on the window `plot1`, proceed as follows:

```
plot1.moveto(x0,y0);
plot1.lineto(x1,y1,mycolor1);
```

where `mycolor1` specifies the color of the line. Colors must be declared as

```
Color mycolor1=GI_green, mycolor2=GI_blue, mycolor3=GI_red;
```

In order to add an extra segment joining `(x2,y2)` to the current line `(x0,y0)` to `(x1,y1)` simply use again the `lineto` method:

```
plot1.lineto(x2,y2,mycolor1);
```

The above `x` and `y` variables must be `float` and in the range specified by `setXY` method. Otherwise, an error will occur.

In many cases, it is desired to plot a graph defined through its `x[i]` and `y[i]` coordinates, where `i` ranges from 0 to `size-1` and `x` and `y` are arrays. Such a graph can be drawn directly using the `plotArray` method. For instance

```
int size=64
float* x=new float[size];
float* y=new float[size];
for(int i=0;i<size;i++){x[i]=i; y[i]=cos(.2*(float) i); }
plot1.setXY(0.,64.,-1.,1.);
plot1.plotArray(x,y,size,mycolor2);
```

In some application, it is useful to visualize a graph which changes during the execution of the program. This can be done using the `startPlotArray` and `updatePlotArray` methods. An example is given below, where, at each iteration, the amplitude of the `y[i]` decreases:

```
plot1.startPlotArray(x,y,size,mycolor3);

float scale=1.;
for(int iter=0;iter<200;iter++){
  scale*=.995;
  for(int i=0;i<size;i++){x[i]=i; y[i]=scale*cos(.2*(float) i);}
  plot1.updatePlotArray(x,y);
}
```

To close a plot window, use the method `close`. In our example:

```
plot1.close();
```

A C++ example program illustrating the above concepts is available in the SpcTools directory `SpcTools/Example/plotExple.cc`.

# 3 Compilation and execution

If you are connected to a linux PC in the CUI network, the library is already installed. If not, first refer to section 4 before reading the present section.

The SpcTools library is installed in the `/user/connex/parallel/SpcTools/Lib` directory. It is made of two distinct components, the `Misc` and `GI` libraries.

The correct path must be included in the compilation line in order to instruct `g++` where to look for these libraries and the corresponding header files. For instance, to compile the program given in section 5, type, in on the same line,

```
g++ -O3  -o parity  -I/user/connex/parallel/SpcTools/Include
   parity.cc   -L/user/connex/parallel/SpcTools/Lib -L/usr/X11R6/lib
                  -lMisc -lGI -lX11
```

then execute with

```
parity 200 200 100
```

where the arguments 200 200 define the array size and 100 the number of iterations. Note that it may be necessary to initialize the environment variable `DISPLAY`. If this is needed, type

```
setenv DISPLAY :0
```

In order to avoid typing a long line as shown above to compile the program, it is possible to use a `Makefile`. A `Makefile` is a file containing instructions on how to build an application. It is a very useful tool is Linux/Unix environment.

As an example to see how use it, copy the file

```
/user/connex/parallel/SpcTools/Example/Makefile
```

in your directory, as well as the `parity.cc` file, which is in the same directory.

In our example, the `Makefile` allows us to compile the `parity.cc` program simply by typing:

```
make parity
```

As a second example, also copy the file `plotExple.cc` from the same directory, and type

```
make plotExple
```

You will get an example for drawing a graph. To execute it, type `plotExple`.

The `Makefile` file can be edited so that new programs can be compiled (there can be only one `Makefile` per directory). Follow the structure of the example `Makefile` to add your own compilation lines. Refer to standard Linux documentation to know more about `Makefile`.

# 4 Installing the library

If you want to use the SpcTools library on a Linux PC or Unix station on which it is not installed, download it from

`http://cui.unige.ch/~chopard/FTP/SpcTools/spcTools.tar.gz`

to your home directory.

Uncompress it with `gunzip spcTools.tar.gz` and "untar" it with `tar -xvf scpTools.tar`. This will produce a directory `SpcTools` containing all the source files. In order to build the library, change directory to `SpcTools` and type `make`.

If you are lucky, this should be enough. The success of the installation can be tested by running the test programs `SpcTools/GI/TestGI` and `SpcTools/Misc/TestMisc`. To remove the library, go the your home directory and type `rm -r SpcTools`

# 5 Example

The following program, `parity.cc`, implements a cellular automata with the so-called parity rule and displays its evolution after each time step.

In this example, each cell in the 2D array can be either 0 or 1. The parity rule amounts to replacing each cell state by the sum modulo 2 of its four neighbors.

The program reads the array size on the command line, as well as the desired number of iterations. The header files "GraphicsInterface.hh" and "Random.hh" are defined in the SpcTools library. See section 3 to see how to compile and execute the following program.

```
#include <fstream.h>
#include <stdlib.h>
#include "GraphicsInterface.hh"
#include "Random.hh"

// for simplicity, we use global variables. No need to
// pass the parameters to the functions

Random rng;
int nx, nz, nbIter;
int** ac;
```

```
int** nac;

GraphicsInterface window;
Matrix<unsigned char> image;

void drawImage(int** a)  // this is a function to display a 2D array a
{
  int i,j;
  unsigned char pix;

  for (i=0;i<nx;i++) {
    for (j=0;j<nz;j++){
      image.set(j,i,(unsigned char) (a[i][j]+1));
         // the +1 is to have red/black colors
    }
  }
  window.drawMatrix(image);
}

int main(int argc, char ** argv)
{
  if (argc != 4) {
    cout << "Usage: parity <sizex> <sizey>  <nbIter>" << endl;
    return 1;
  }

  nx=atoi(argv[1]);
  nz=atoi(argv[2]);
  nbIter=atoi(argv[3]);

  rng.init();
  window.open(nz,nx);
  image.setDimension(nz,nx);

  // allocation memory for the ac
  ac=new int*[nx];
  for(int i=0;i<nx;i++)ac[i]=new int[nz];
  nac=new int*[nx];
  for(int i=0;i<nx;i++)nac[i]=new int[nz];

  // initial condition: zero everywhere except in the center----------
```

```
  for(int i=0;i<nx;i++)
    for(int j=0;j<nz;j++)ac[i][j]=0;

  // put some random sites with values 1
  for(int i=nx/2-3;i<nx/2+3;i++)
    for(int j=nz/2-3;j<nz/2+3;j++)ac[i][j]=(int) (rng.get()+.5);
  //-----------------------------------------

  cout << "size = " << nx <<"  " << nz << endl;
  cout << "nbIter = " << nbIter << endl;

  //-------Iterations of the Cellular Automata --------------
  for (int iter=0;iter<nbIter;iter++) {

    // implement the evolution rule (Parity rule, here)
    for(int i=0;i<nx;i++){
      for (int j=0;j<nz;j++){
nac[i][j]=(ac[(i+nx-1)%nx][j] + ac[(i+nx+1)%nx][j] +
        ac[i][(j+nz-1)%nz] + ac[i][(j+nz+1)%nz]
                  )%2;
      }
    }

    // copy new values in old ones
    for(int i=0;i<nx;i++)
      for(int j=0;j<nz;j++)ac[i][j]=nac[i][j];

    // display the new configuration
    drawImage(ac);
  }


  window.close();

  return 0;
}
```