Architecture d'un programme interactif graphique

Partie 1: Bases et Visualisation

Objectifs:

- Organiser un projet selon Model-View-Controller
- Premier contact avec GTKmm
- Dessin 2D avec l'API Cairo de GTKmm

Plan:

- Les 3 types de dialogue utilisateur
- Approche Model-View-Controller
- Premier contact avec GTKmm
- Visualisation: de l'espace du Modèle à celui du dessin
- Elements principaux de l'API Cairo pour le dessin

Les 3 styles de dialogues humain-machine

- **ligne de commande** (commandes UNIX-LINUX)
  - ❑ passage de paramètres au programme avec **argc** et **argv**
- **Conversationnel** (programmes semestre d'automne)
  - ❑ **cout**, **cin**, etc disponible avec la bibliothèque standard C++
- **Interface Graphique Utilisateur** (*Graphic User Interface* / *GUI*)
  - ❑ widgets (boutons, etc...) disponible dans ***bibliothèques***

Bibliothèque (*Library*) : regroupe le code objet de plusieurs modules réalisant une tâche bien définie (exemple: interface graphique, dessin, etc...)

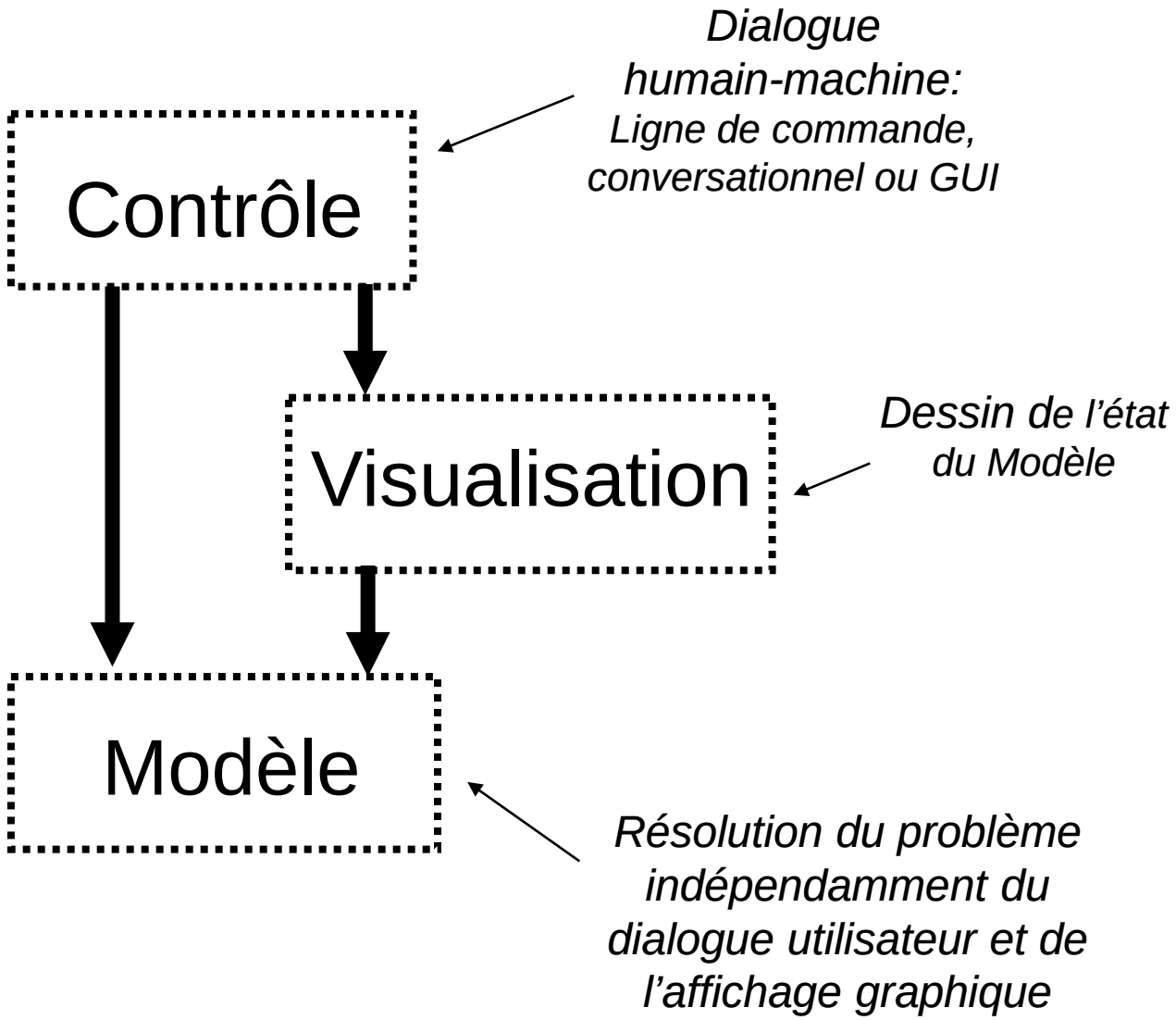
API (*Application Programming Interface*) : prototypes des fonctions exportées par une bibliothèque dans un fichier en-tête (exemple: **gtkmm.h**) = fichier d'interface de la bibliothèque

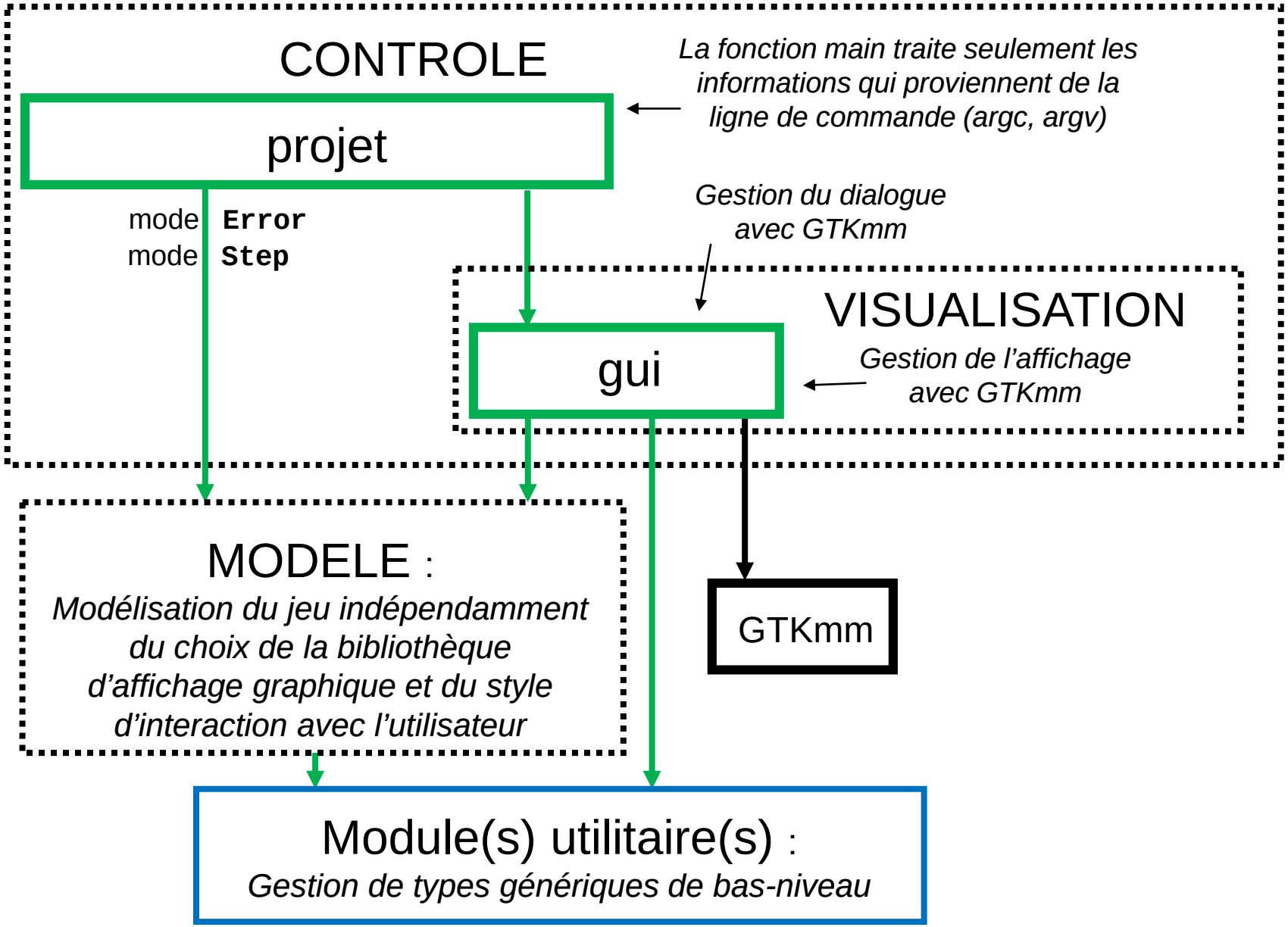
L'architecture MVC: Model-View-Controller (projet section 7.1)

Principe : Separation of concerns

Objectifs:

- Permettre plusieurs styles de Visualisations et de Contrôle/Dialogue sur un Modèle.
- S'adapter facilement à l'évolution rapide de la technologie des parties "Visualisation" et "Contrôle utilisateur" en gardant stable la partie "Modèle métier"
- Dans l'industrie, les compétences sont souvent focalisées sur un sous-système, rarement sur l'ensemble des trois.




CONTROLE
La fonction main traite seulement les informations qui proviennent de la ligne de commande (argc, argv)
projet
mode Error
mode Step
Gestion du dialogue avec GTKmm
VISUALISATION
Gestion de l'affichage avec GTKmm
gui
GTKmm
MODELE :
Modélisation du jeu indépendamment du choix de la bibliothèque d'affichage graphique et du style d'interaction avec l'utilisateur
Module(s) utilitaire(s) :
Gestion de types génériques de bas-niveau

Premier contact avec GTKmm

API de GTKmm

```
#include <gtkmm.h>

using namespace std;

int main(int argc, char ** argv)
{
        auto app = Gtk::Application::create(argc,argv,"org.gtkmm.examples.base");

        Gtk::Window window;
        window.set_default_size(200,200);

        return app->run(window);
}
```

Création d'une fenêtre

*Création d'un objet **Application** dont on récupère un pointeur dans **app***

*Lancement de l'**Application** à l'aide du pointeur **app***

Premier bouton avec GTKmm (1)

main.cc

```
#include "helloworld.h"
#include <gtkmm/application.h>

int main (int argc, char *argv[])
{
  auto app = Gtk::Application::create(argc, argv, "org.gtkmm.example");

  HelloWorld helloworld;

  //Shows the window and returns when it is closed.
  return app->run(helloworld);
}
```

+ module helloworld.cc

Premier bouton avec GTKmm (2)

helloworld.cc

helloworld.h

```
#include <gtkmm/button.h>
#include <gtkmm/window.h>

class HelloWorld : public Gtk::Window
{
public:
  HelloWorld();
  virtual ~HelloWorld();
protected:
  //Signal handlers:
  void on_button_clicked();

  //Member widget:
  Gtk::Button m_button;
};
```

```
#include "helloworld.h"
#include <iostream>

HelloWorld::HelloWorld()
: m_button("Hello World")
{
  set_border_width(10);

  m_button.signal_clicked()
          .connect(sigc::mem_fun(*this,
            &HelloWorld::on_button_clicked));

  add(m_button);  // ajoute à la fenètre
  m_button.show();// demande l'affichage
}

HelloWorld::~HelloWorld(){}

void HelloWorld::on_button_clicked()
{
  std::cout << "Hello World" << std::endl;
}
```

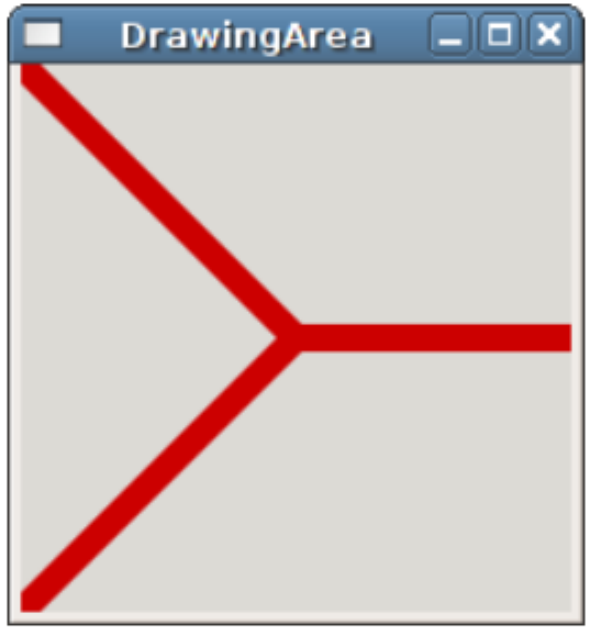
Connexion d'un signal sur m_button à une méthode définie par l'utilisateur

Elements principaux de l'API Cairomm pour le dessin

Le widget spécialé **DrawingArea** est celui destiné au dessin

Pour dessiner il faut surcharger la méthode **on_draw()**
Elle est aussi appelée automatiquement quand le système détecte que la fenêtre a besoin d'être redessinée.

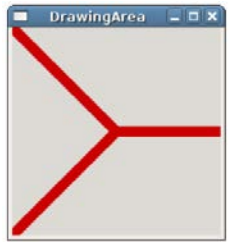
On dispose de méthodes pour dessiner des lignes droites, courbes ou des arcs de cercle,etc…



Un objet **Cairo::Context** permet de mémoriser tous les paramètres courants du dessin tels que l'épaisseur du trait, la couleur, etc…

Ainsi les fonctions de dessins n'ont pas à préciser ces paramètres à chaque appel

Exemple DrawingArea



mainDrawingArea.cc

```
#include "myarea.h"
#include <gtkmm/application.h>
#include <gtkmm/window.h>

int main(int argc, char** argv)
{
   auto app = Gtk::Application::create(argc, argv, "org.gtkmm.example");

   Gtk::Window win;
   win.set_title("DrawingArea");

   MyArea area; // widget que l'on ajoute à la fenêtre win
   win.add(area);
   area.show();

   return app->run(win);
}
```

+ module myarea.cc

Exemple DrawingArea(2)

myarea.cc

myarea.h

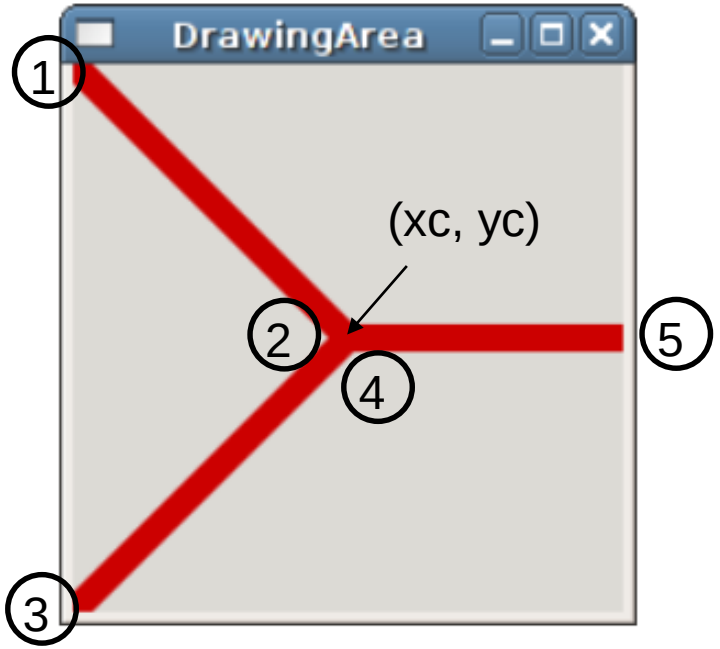
```
#ifndef GTKMM_EXAMPLE_MYAREA_H
#define GTKMM_EXAMPLE_MYAREA_H

#include <gtkmm/drawingarea.h>

class MyArea : public Gtk::DrawingArea
{
public:
  MyArea();
  virtual ~MyArea();

protected:
  //Override default signal handler:
  bool on_draw(const
          Cairo::RefPtr<Cairo::Context>& cr)
          override;
};
#endif // GTKMM_EXAMPLE_MYAREA_H
```

```
#include "myarea.h"
#include <cairomm/context.h>

MyArea::MyArea(){}
MyArea::~MyArea(){}

bool MyArea::on_draw(const Cairo::RefPtr<Cairo::Context>& cr)
{
  Gtk::Allocation allocation = get_allocation();
  const int width = allocation.get_width();
  const int height = allocation.get_height();

  // coordinates for the center of the GTKmm window
  int xc, yc;
  xc = width / 2;
  yc = height / 2;

  cr->set_line_width(10.0); // mémorisé à long terme dans cr

  // draw red lines out from the center of the window
  cr->set_source_rgb(0.8, 0.0, 0.0); // idem mémorisation cr
  cr->move_to(0, 0);
  cr->line_to(xc, yc);
  cr->line_to(0, height);
  cr->move_to(xc, yc);
  cr->line_to(width, yc);
  cr->stroke();

  return true;
}
```

Définition du tracé «path» mais sans faire le dessin

Commande effective de dessin

Les principales commandes d'un contexte



Définir un nouvel état permanent d'un paramètre

Variable d'état	Val par défaut	Méthode pour modifier l'état
Épaisseur du trait	1	**`set_line_width`**(val)
couleur du dessin	noir	**`set_source_rgb`**(R,G,B)

Dessiner un fond de couleur uniforme: **paint()**

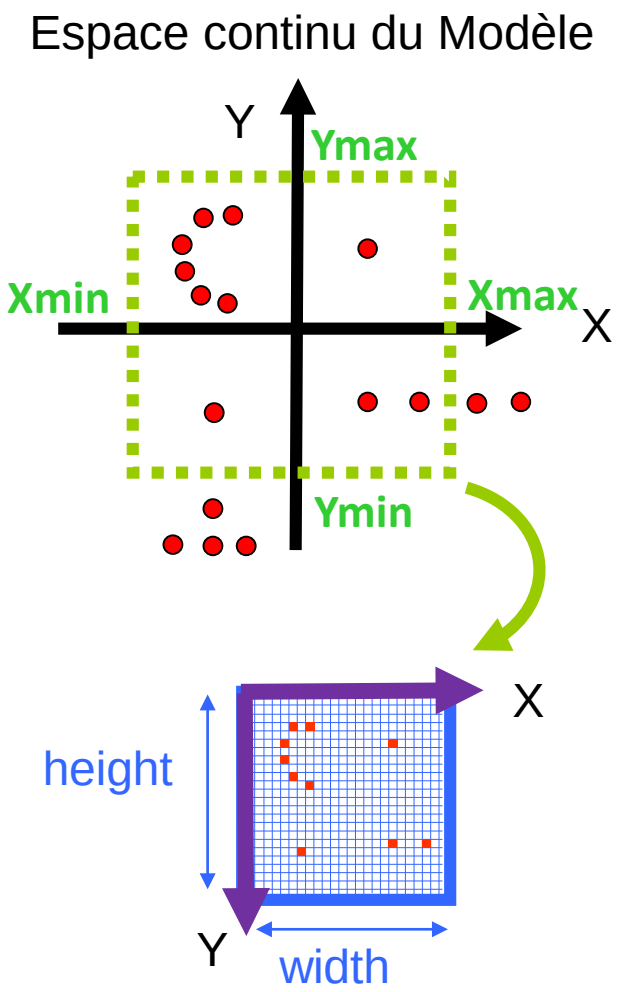
Créer une ligne ou polyline / concept de «**path**»:

Définir le début de ligne avec **`move_to(x,y)`**
Définir un point d'un path avec **`line_to(x,y)`**

Dessiner (puis supprimer) le «**path**» qui vient d'être créé: **`stroke()`**

```
cr->set_line_width(10.0);
cr->set_source_rgb(0.8, 0.0, 0.0);
1 cr->move_to(0, 0);
2 cr->line_to(xc, yc);
3 cr->line_to(0, height);
4 cr->move_to(xc, yc);
5 cr->line_to(width, yc);
cr->stroke();
```

Visualisation: de l'espace du Modèle à celui du dessin



Espace fini et discret de la fenêtre graphique exprimé en pixels: width x height

1) Le «Modèle" que l'on veut dessiner doit être *représentable* à l'aide de points, lignes, arcs, polygones (plein ou vide), etc... Il tient à jour les coordonnées (x,y) de ses éléments dans l'espace du problème qu'il résout ; on travaille en général en virgule flottante double précision.

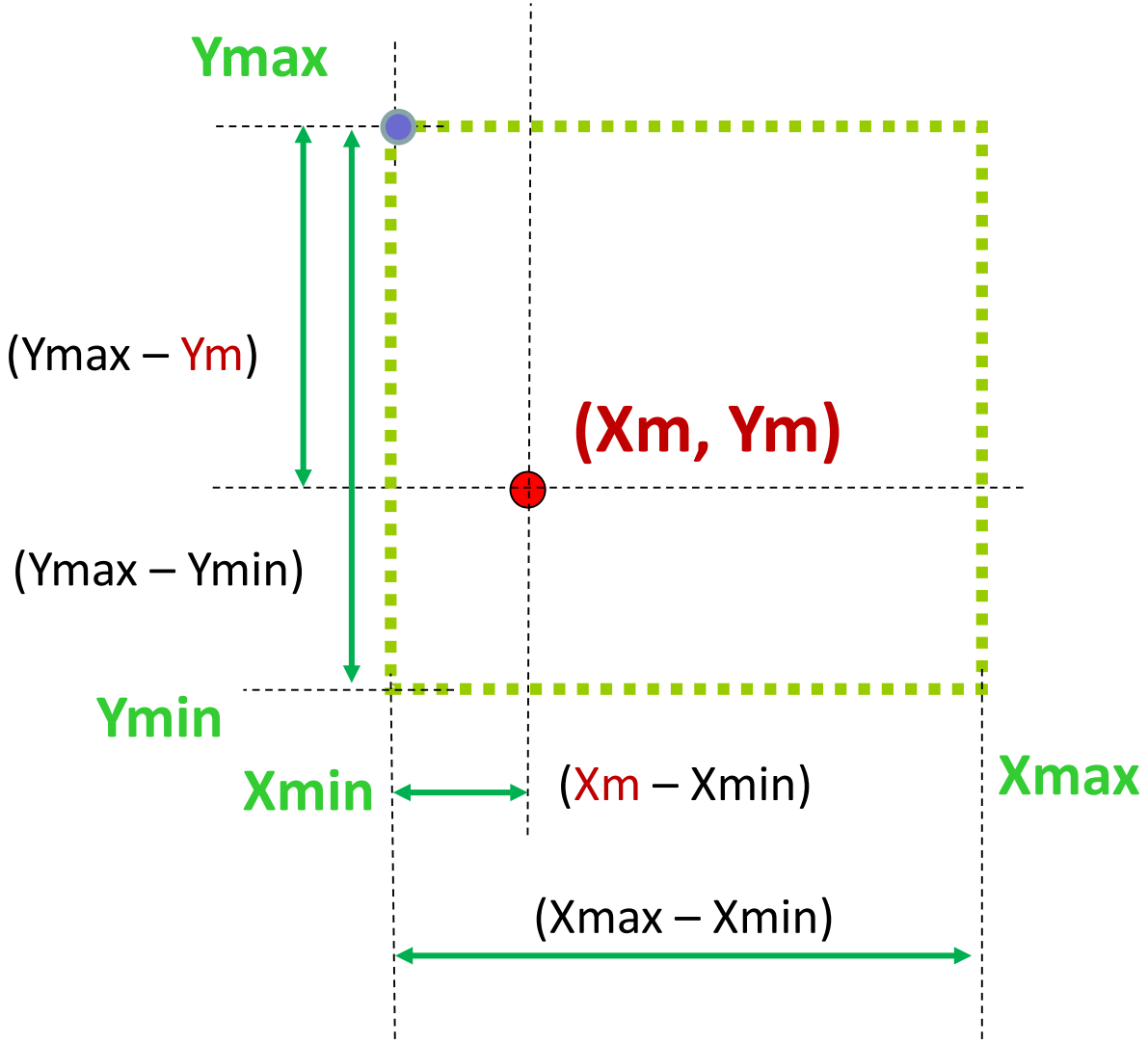
2) Cadrer: choisir la fenêtre **[Xmin, Xmax]** x **[Ymin, Ymax]** qui sera dessinée

3) Le système de visualisation GTKmm est responsable de convertir les coordonnées du Modèle dans son système de coordonnées :

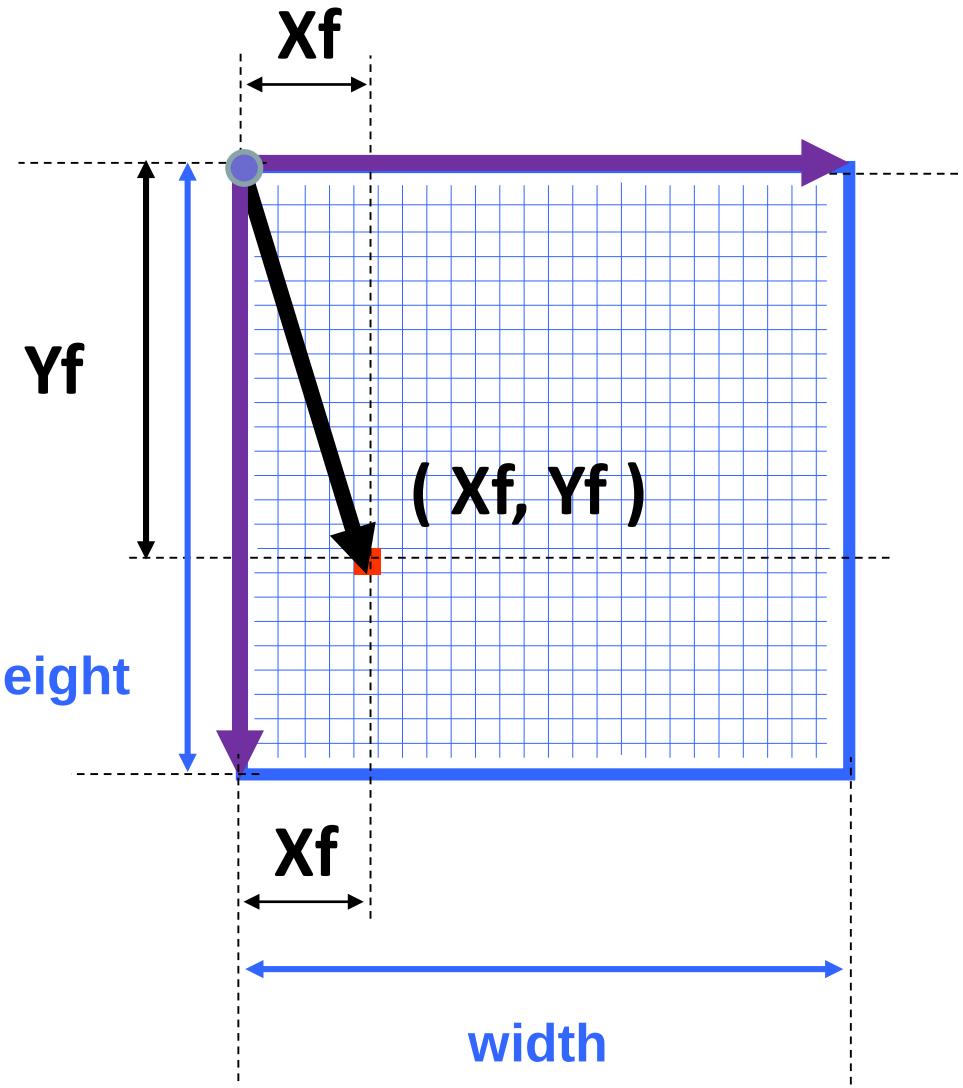
- **L'origine de la fenêtre GTKmm est dans le coin haut gauche**
- **L'axe X de GTKmm croît positivement vers la droite**
  - **X varie entre 0 et width (largeur de la fenêtre en pixels)**
- **L'axe Y de GTKmm croît positivement vers le bas**
  - **Y varie entre 0 et height (hauteur de la fenêtre en pixels)**

4) dessiner = appeler les fonction GTKmm de dessin en leur fournissant les coordonnées converties à l'étape précédente.

Espace continu du Modèle



Espace fini et discret de la fenêtre graphique exprimé en pixels: **width** x **height**



On pose l'égalité des proportions pour passer d'un espace à l'autre

$$\begin{cases} \dfrac{(Xm - Xmin)}{(Xmax - Xmin)} = \dfrac{Xf}{width} \\ \dfrac{(Ymax - Ym)}{(Ymax - Ymin)} = \dfrac{Yf}{height} \end{cases}$$

On pose l'égalité des proportions pour passer d'un espace à l'autre (2)

De l'espace du Modèle à la fenêtre du dessin

$$\begin{cases} Xf = width * (Xm - Xmin)/(Xmax - Xmin) \\ Yf = height * (Ymax - Ym)/(Ymax - Ymin) \end{cases}$$

De la fenêtre du dessin à l'espace du Modèle

$$\begin{cases} Xm = (Xf/width) * (Xmax - Xmin) + Xmin \\ Ym = Ymax - (Yf/height) * (Ymax - Ymin) \end{cases}$$

Pour éviter toute distortion à l'affichage
Il faut garantir le même facteur d'échelle selon X et selon Y

Xf = [width/(Xmax-Xmin)] * (Xm –Xmin)

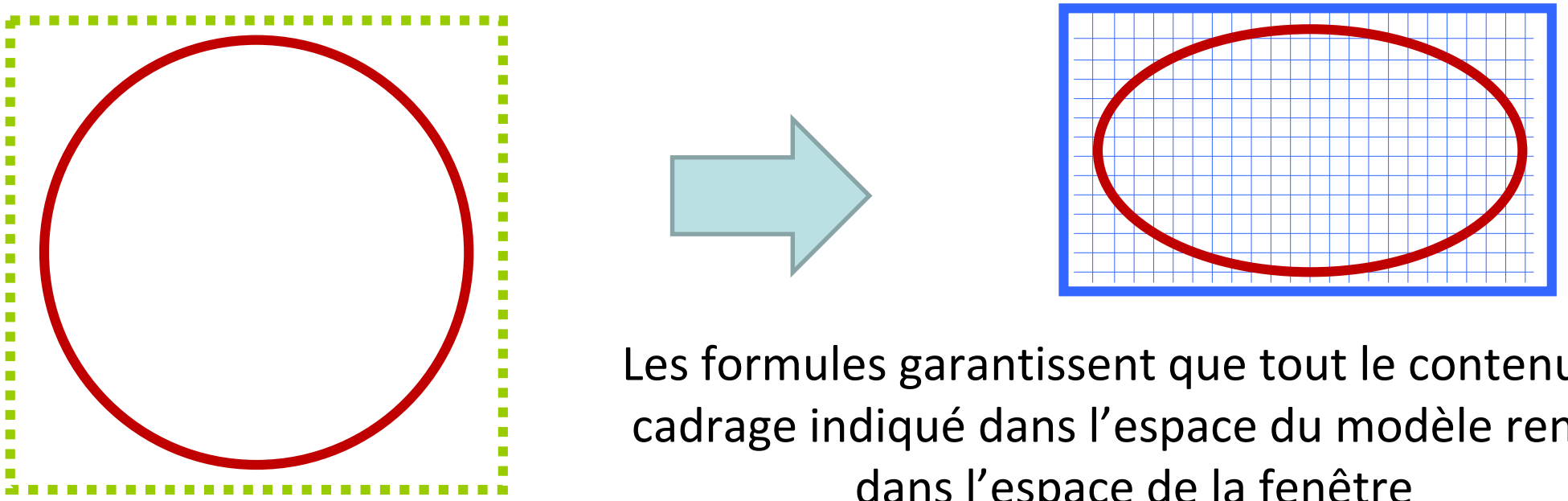
Yf = [height/(Ymax-Ymin)] * (Ymax – Ym)

width/(Xmax-Xmin)] = height/(Ymax-Ymin)

width/height = (Xmax-Xmin)/(Ymax-Ymin)

*Il suffit d'avoir **le même rapport largeur/hauteur** (aspect ratio)*

Exemple pour lequel l'aspect ratio est différent



Les formules garantissent que tout le contenu du cadrage indiqué dans l'espace du modèle rentre dans l'espace de la fenêtre

En supposant que la taille de la fenêtre est fixée, on peut ***ajuster le cadrage*** dans l'espace du modèle pour obtenir le même rapport largeur/hauteur.

Il existe une infinité de solutions : **(Xmax-Xmin)/(Ymax-Ymin) = width/height**
Il faut ajouter un critère supplémentaire, par exemple garder constant
(Xmax – Xmin) ou **(Ymax-Ymin)** et ajuster l'autre terme.

Résumé

- **En vertu du principe de séparation des fonctionalités nous adoptons l'approche Model-View-Controller pour l'architecture d'une application interactive.**
- **Le sous-système de Contrôle pilote celui de Visualisation et celui du Modèle du problème traité.**
- **GTKmm offre une hiérarchie de classes C++ pour construire une interface utilisateur**
- **Les widgets sont dérivés de la classe Window**
- **Pour dessiner on redéfinit (override) la méthode on_draw de la classe DrawingArea**
- **Les paramètres du dessin sont mémorisés dans un Context**
- **C'est le sous-système de Visualisation qui est responsable de la conversion de coordonnées du Modèle vers la fenêtre de dessin**