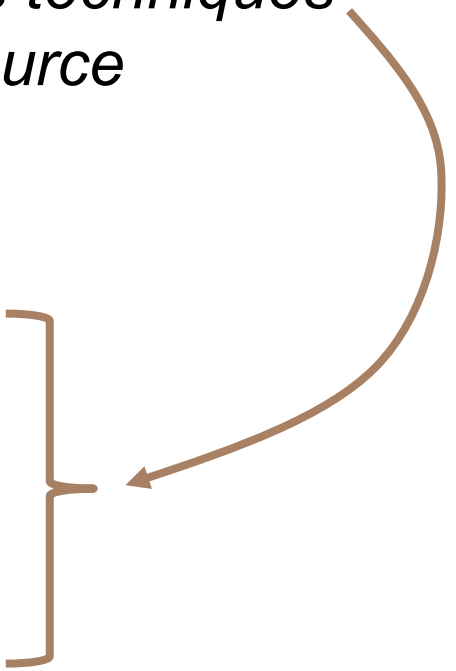
GTKmm : Bases et Visualisation

La série5 niveau0 développe les détails techniques et fournit l'ensemble du code source



Plan:

- Première application GTKmm
- Un premier bouton
- l'API Cairomm pour le dessin
- Le premier dessin avec MyDrawingArea

- Conversions entre espaces du Modèle et du dessin

Première **Application** GTKmm

API de GTKmm

```
#include <gtkmm.h>

using namespace std;

int main(int argc, char ** argv)
{
        auto app = Gtk::Application::create();

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

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

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

Création d'une fenêtre

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

Le nom de l'exécutable apparaît dans l'en-tête de la fenêtre

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();

  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;
}
```

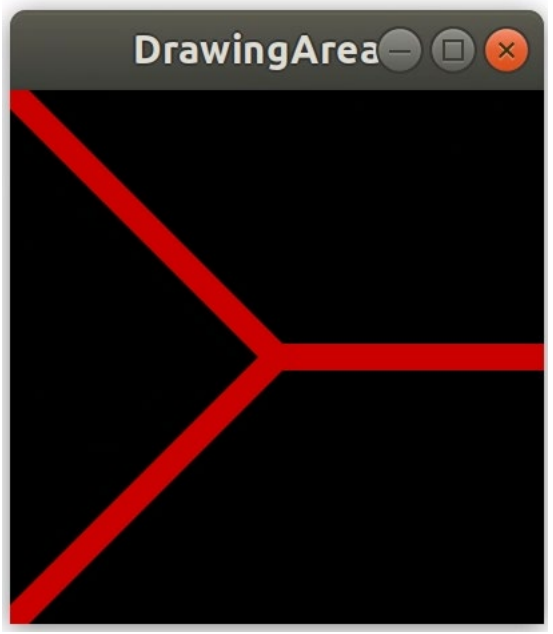
Connexion d'un signal sur m_button à une méthode définie par l'utilisateur

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.

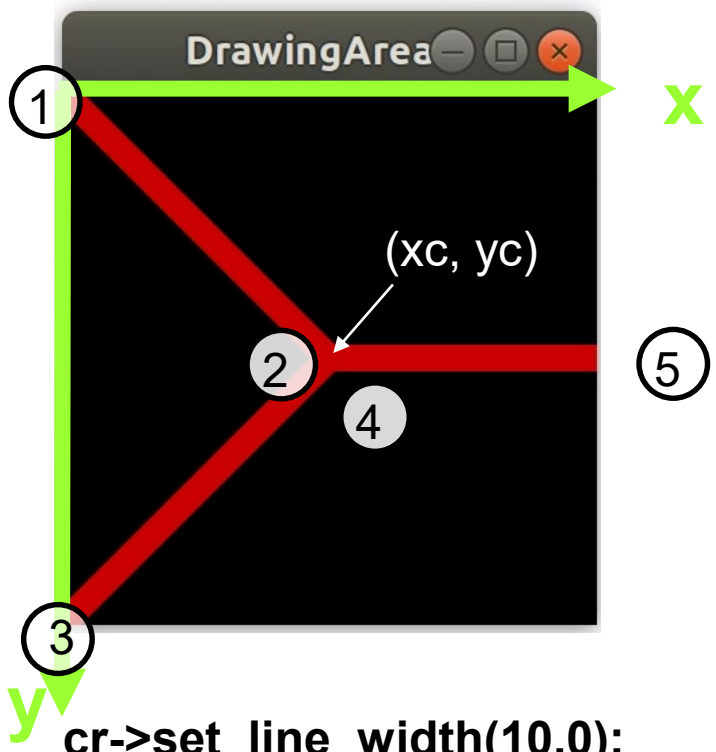
On dispose de méthodes pour dessiner des lignes droites, courbes ou des arcs de cercle,etc…



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

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 avec **paint()**

```
cr->set_source_rgb(0.0, 0.0, 0.0);
cr->paint();
```

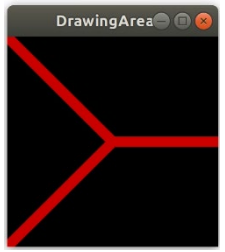
Créer une ligne ou polyline / concept de «**path**»:

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();
```

Exemple DrawingArea

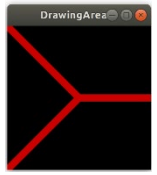


mainDrawingArea.cc

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

int main(int argc, char** argv)
{
   auto app = Gtk::Application::create();

   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

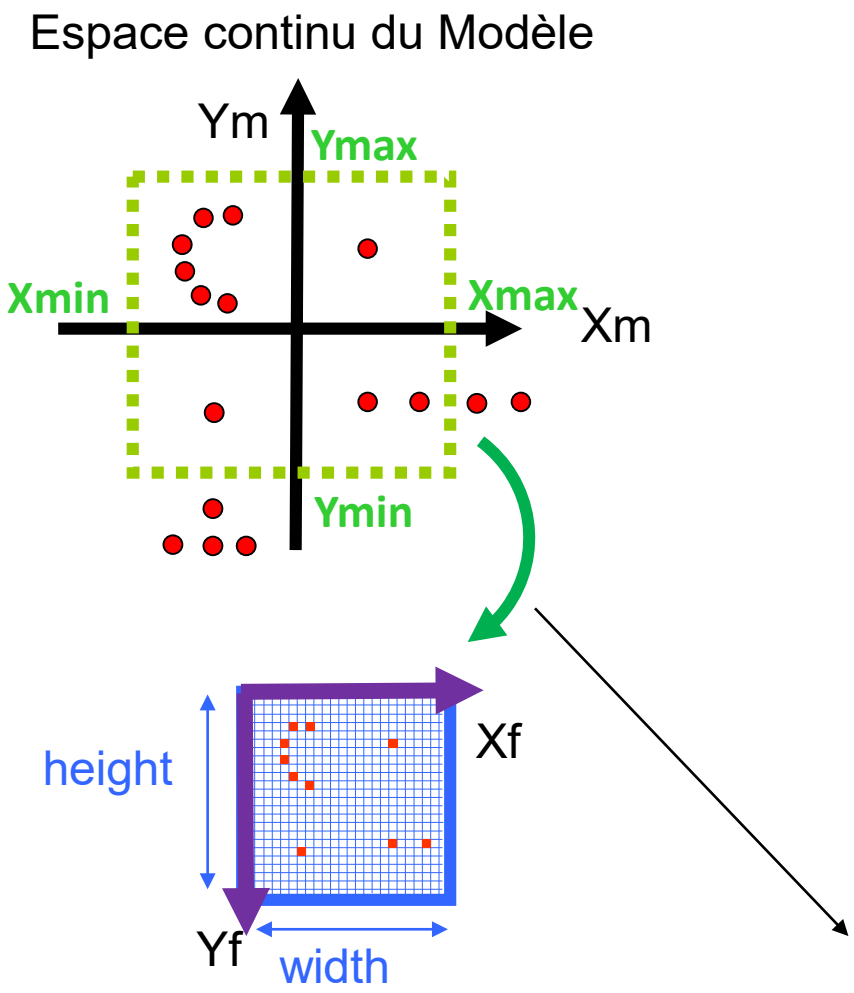
```
#ifndef GTKMM_EXAMPLE_MYAREA_H
#define GTKMM_EXAMPLE_MYAREA_H

#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();

  // changing the default background color to black
  cr->set_source_rgb(0.0, 0.0, 0.0); //mémorisé dans cr
  cr->paint();

  // center of the GTKmm window
  int xc(width/2), yc(height/2);

  cr->set_line_width(10.0); // idem mémorisé 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

Visualisation: de l'espace du Modèle (Xm,Ym) à celui de la fenêtre (Xf, Yf)



Espace fini et discret de la fenêtre graphique GTKmm 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 **(Xm,Ym)** 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 ne connaît que le système de coordonnées **(Xf, Yf)** de la fenêtre graphique:

- **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) transformations translate et scale de GTKmm Cairo

Dessiner avec les fonctions GTKmm de dessin avec les coordonnées ***(Xm,Ym)***

Visualisation: de l'espace du Modèle (Xm,Ym) à celui de la fenêtre (Xf, Yf)

4.2) Alternative avec transformations translate et scale de GTKmm Cairo

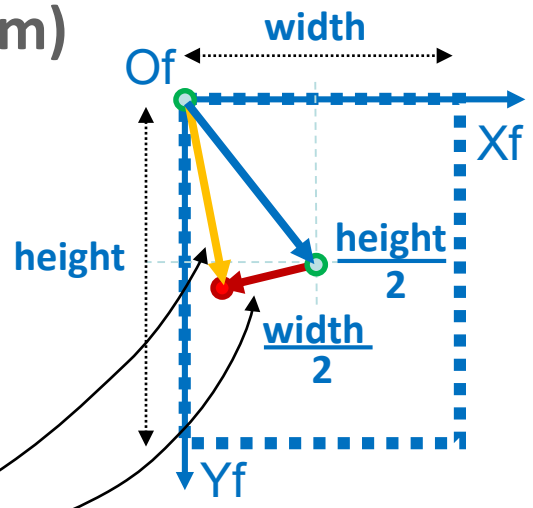
Dessiner avec les fonctions GTKmm de dessin
avec les coordonnées (x,y) dans (Om,Xm,Ym)

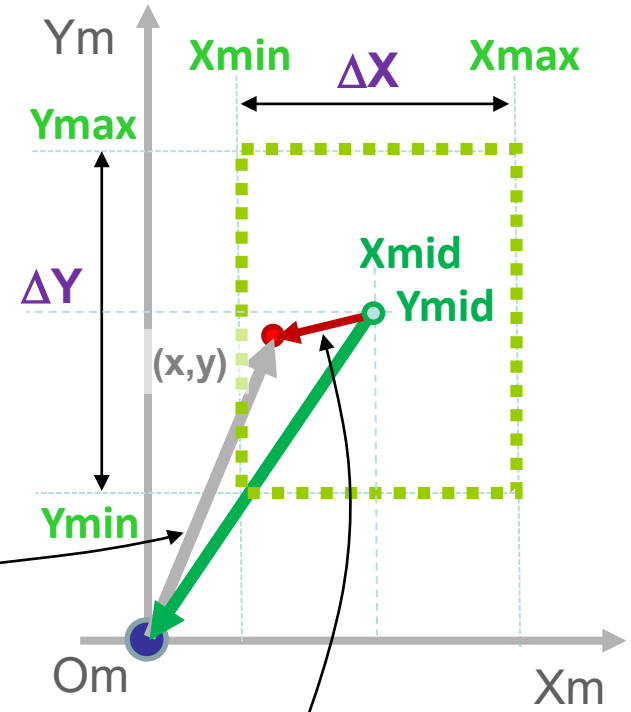
Effectuer ces 3 transformations dans cet ordre
Après avoir mis à jour **width** et **height** et avant de faire des appels avec des coordonnées dans l'espace (Om,Xm, Ym) :

a **cr->translate(width/2, height/2)**

b **cr->scale(width/ΔX, -height/ΔY);**

c **cr->translate(-Xmid, -Ymid);**





Examinons l'effet des trois transformations
en partant du vecteur (x,y) dans (Om,Xm,Ym)

c L'addition du vecteur **(-Xmid, -Ymid)** produit le vecteur **(x,y)**

b Mise à l'échelle de la fenêtre graphique et inversion de l'axe **Y**

a L'addition du vecteur **(width/2, height/2)** produit le vecteur **(x,y)** dans **(Of, Xf,Yf)**

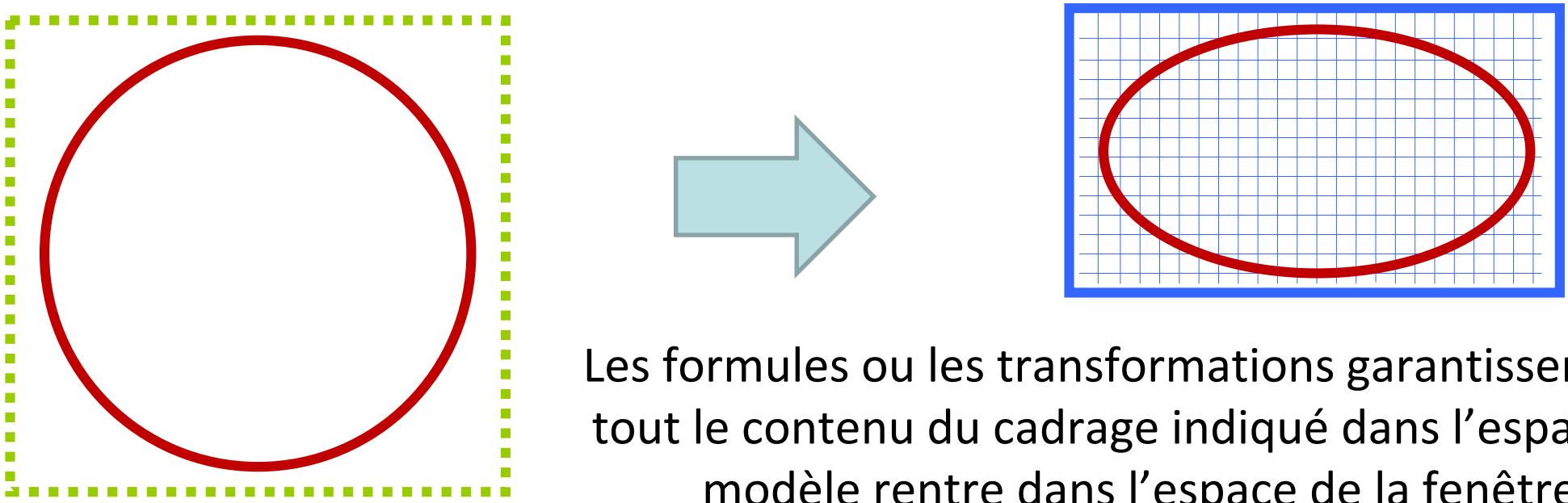
Pour éviter toute distortion à l'affichage
Il faut garantir le même *facteur d'échelle* selon X et selon Y
dans la transformation **cr->scale(width/ΔX, -height/ΔY);**

width/ ΔX = height/ ΔY

width/height = ΔX / ΔY

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

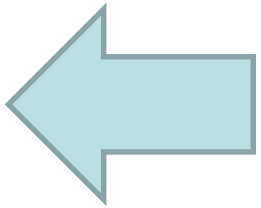
Exemple pour lequel l’aspect ratio est différent

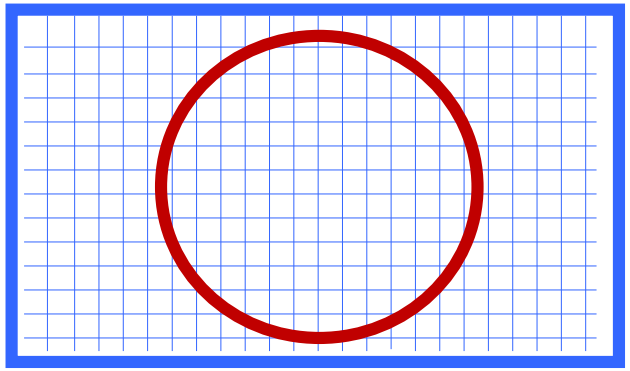


Les formules ou les transformations garantissent que tout le contenu du cadrage indiqué dans l’espace du modèle rentre dans l’espace de la fenêtre

Cela introduit une distorsion indésirable.

Ajustement de l’aspect ratio du cadrage du Modèle pour supprimer la distorsion



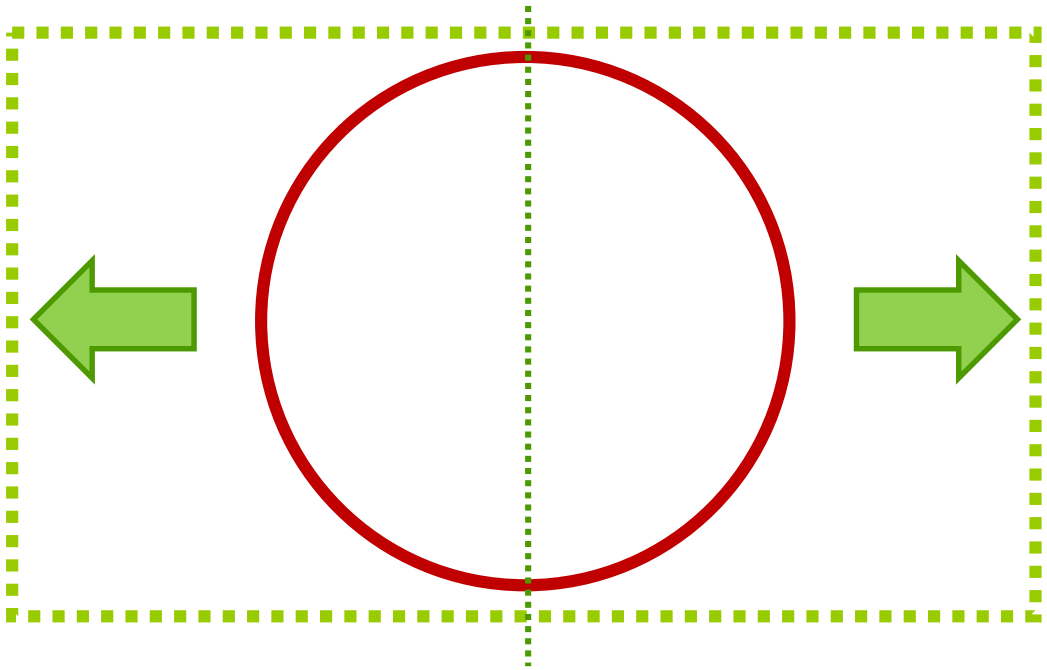


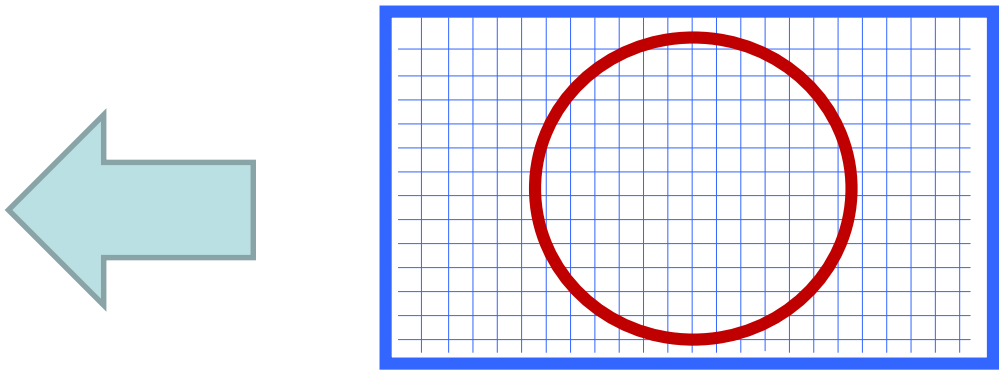
Il faut recadrer le Modèle dans la même proportion que l’aspect ratio de la fenêtre

Il existe une infinité de solutions qui respectent l’égalité des aspects ratios:
(Xmax-Xmin)/(Ymax-Ymin) = width/height

Il faut choisir un critère supplémentaire, par exemple garder constant **(Xmax – Xmin)** ou **(Ymax-Ymin)** et ajuster l’autre terme.

Exemple: conserver (Ymax –Ymin) et ajuster (Xmax-Xmin)





Il faut recadrer le Modèle dans la même proportion que l'aspect ratio de la fenêtre

Ici on choisit de conserver le même centre dans l'espace du Modèle,
donné par **Xmid= (Xmax+Xmin)/2**
Et d'élargir symétriquement l'intervalle selon X
Les nouvelles valeurs $\Delta X'$, Xmin' et Xmax' sont alors:

$$\Delta X' = \Delta Y.(\text{width/height})$$

$$Xmin' = Xmid - \Delta X'/2$$
$$Xmax' = Xmid + \Delta X'/2$$

Résumé

- **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 (méthode on_draw()) qui est responsable des conversions de coordonnées entre le Modèle et la fenêtre de dessin**
- **Il suffit de 2 appels à translate() et un seul de scale() au début de on_draw() pour cette conversion ; le Modèle travaille ainsi toujours dans son espace sans se soucier du système de coordonnées de l'affichage final.**