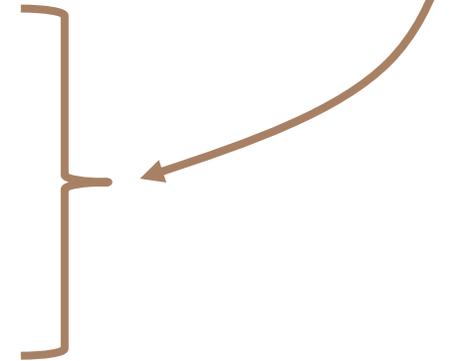


GTKmm : Bases et Visualisation

La série 5 niveau 0 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(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>

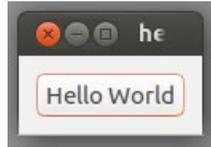
int main (int argc, char *argv[])
{
    auto app = Gtk::Application::create(argc, argv, "org.gtkmm.example");

    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 fenetre
    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

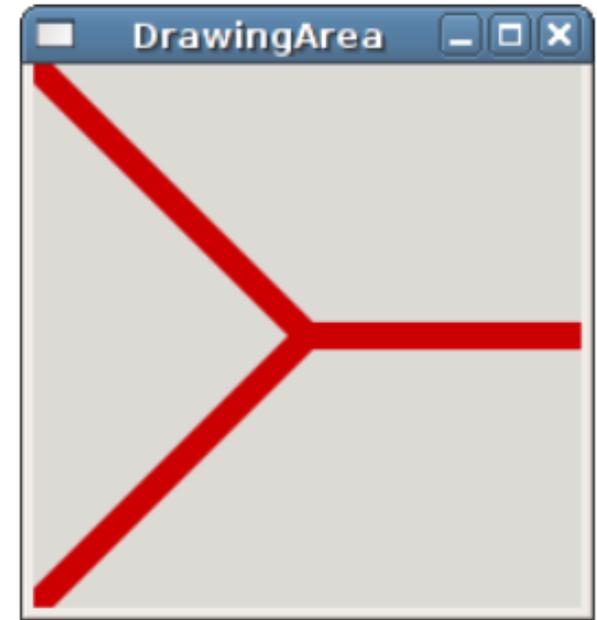
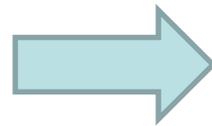
Elements principaux de l'API Cairo pour le dessin

Le widget spécial **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...

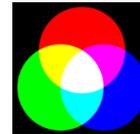


Ainsi les fonctions de dessins n'ont pas à préciser ces paramètres à chaque appel

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	<code>set_line_width(val)</code>
couleur du dessin	noir	<code>set_source_rgb(R,G,B)</code>



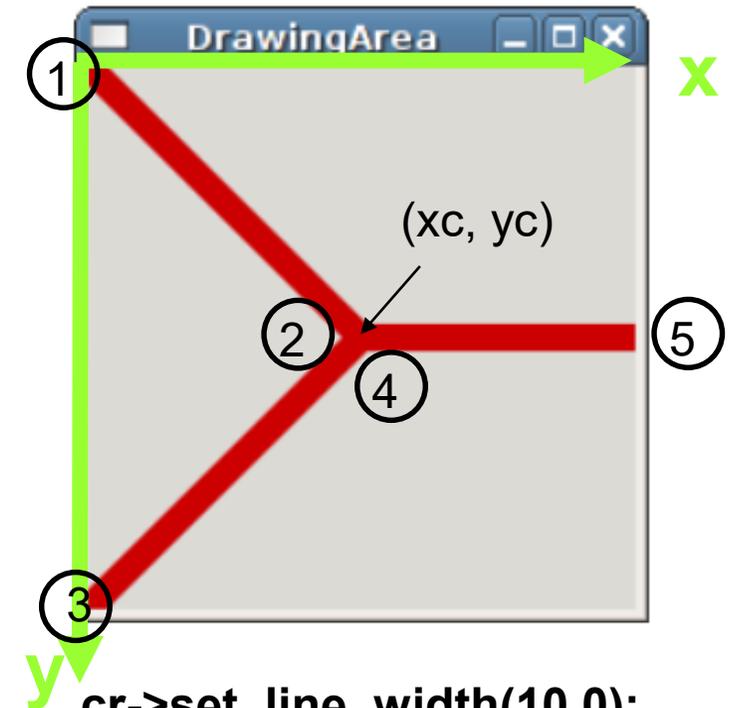
Dessiner un fond de couleur uniforme: `paint()`

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->stroke();`



```
cr->set_line_width(10.0);  
cr->set_source_rgb(0.8, 0.0, 0.0);
```

```
① cr->move_to(0, 0);
```

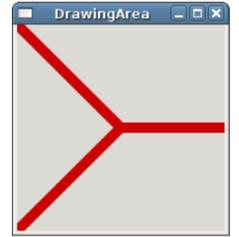
```
② cr->line_to(xc, yc);
```

```
③ cr->line_to(0, height);
```

```
④ cr->move_to(xc, yc);
```

```
⑤ cr->line_to(width, yc);
```

Exemple DrawingArea



mainDrawingArea.cc

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

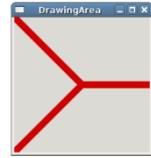
int main(int argc, char** argv)
{
    auto app = Gtk::Application::create(argc, argv, "org.gtkmm.example");

    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);
}
```

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();

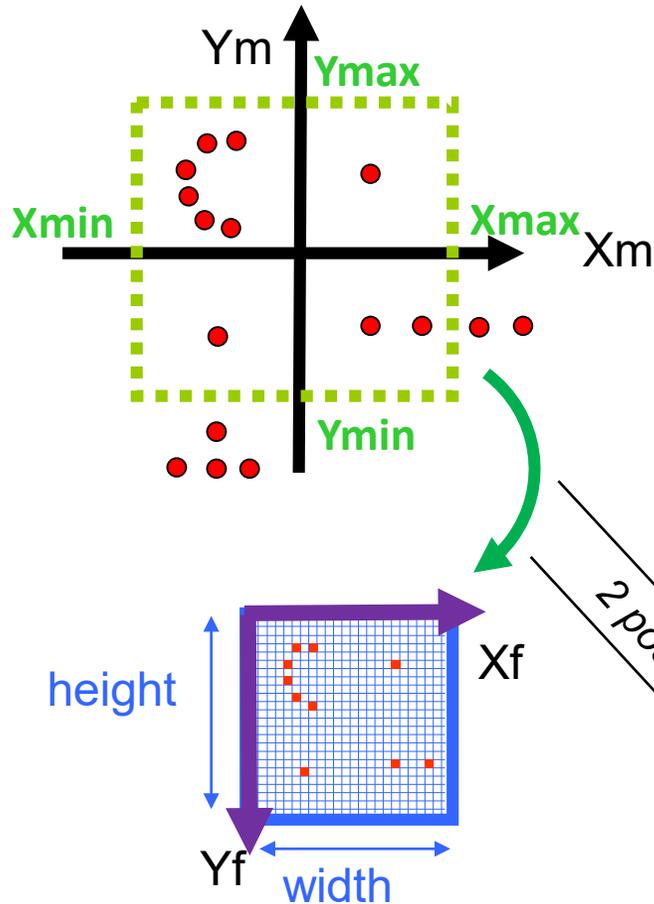
    return true;
}
```

*Définition du tracé
«path» mais sans
faire le dessin*

Commande effective de dessin

Visualisation: de l'espace du Modèle (X_m, Y_m) à celui de la fenêtre (X_f, Y_f)

Espace continu du Modèle



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 (X_m, Y_m) 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 [X_{min}, X_{max}] x [Y_{min}, Y_{max}] qui sera dessinée

3) Le système de visualisation GTKmm ne connaît que le système de coordonnées (X_f, Y_f) 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)

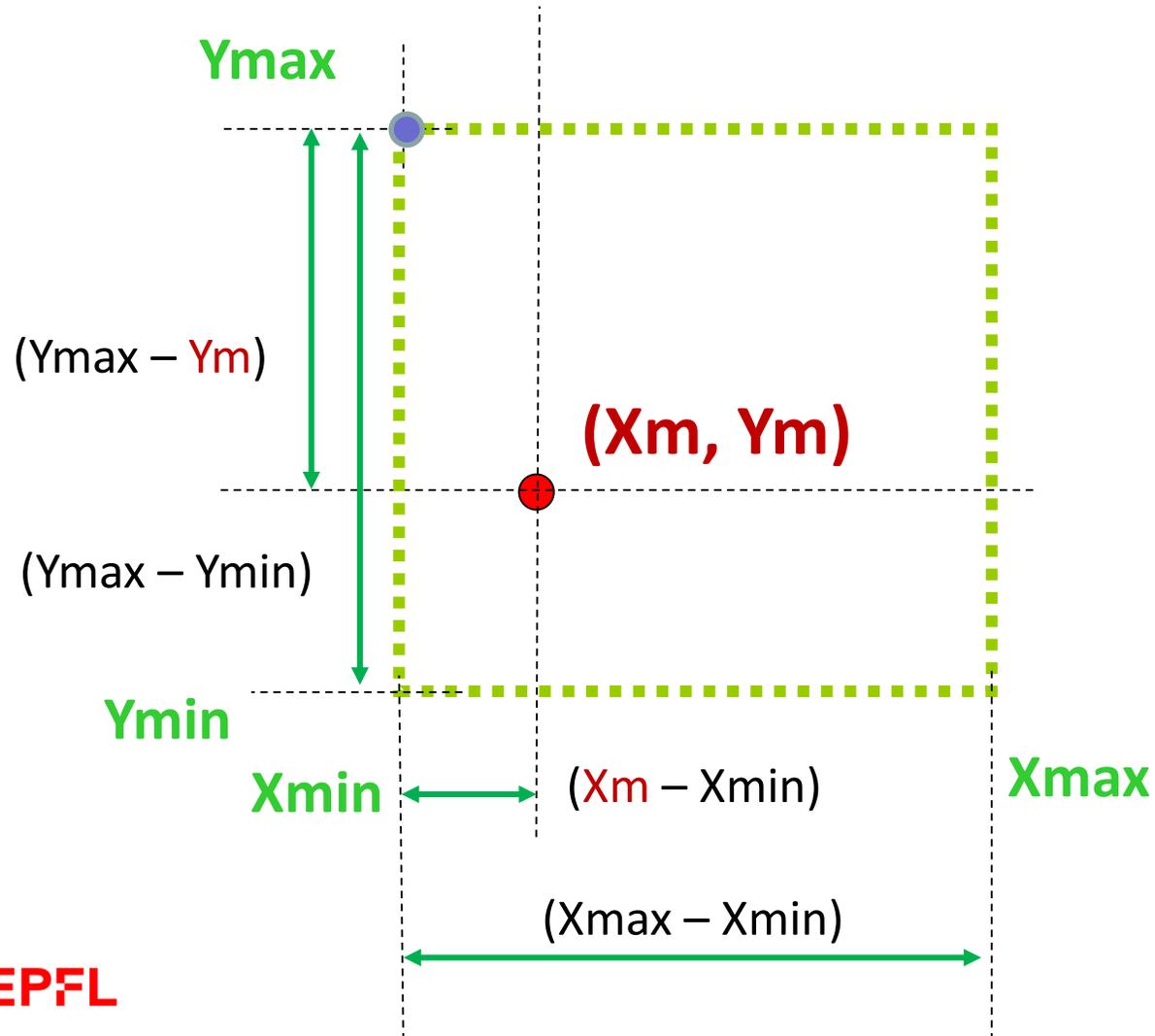
4.1) Conversion des coord du Modèle (X_m, Y_m) vers GTKmm (X_f, Y_f).

Dessiner avec les fonctions GTKmm de dessin avec les coordonnées (X_f, Y_f)

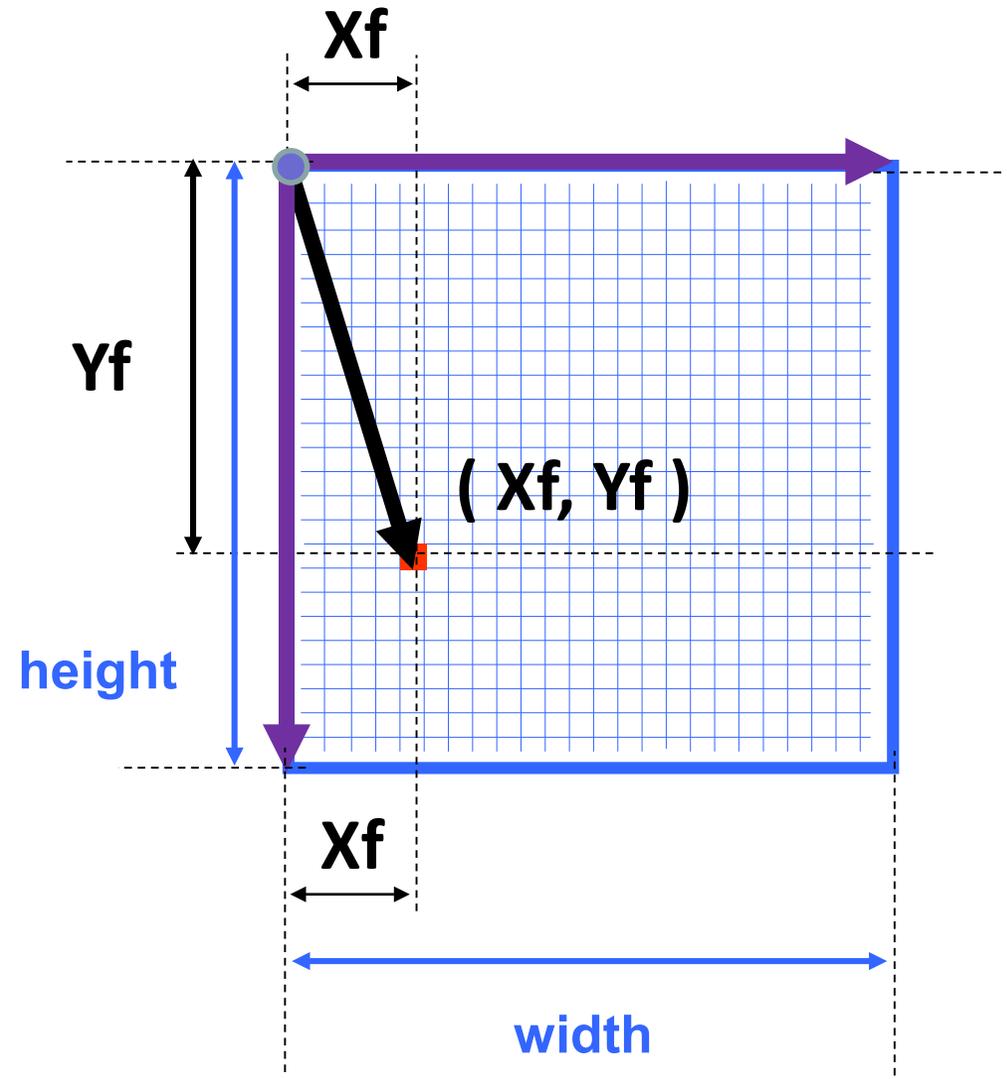
4.2) Alternative avec transformations translate et scale de GTKmm Cairo

Dessiner avec les fonctions GTKmm de dessin avec les coordonnées (X_m, Y_m)

Espace continu (X_m, Y_m) du Modèle



Espace fini et discret (X_f, Y_f)
de la fenêtre graphique
exprimé en pixels: **width** x **height**



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

$$\left\{ \begin{array}{l} \frac{(X_m - X_{\min})}{(X_{\max} - X_{\min})} = \frac{X_f}{\text{width}} \\ \frac{(Y_{\max} - Y_m)}{(Y_{\max} - Y_{\min})} = \frac{Y_f}{\text{height}} \end{array} \right.$$

Visualisation: de l'espace du Modèle (X_m, Y_m) à celui de la fenêtre (X_f, Y_f)

4.1) Conversion des coord du Modèle (X_m, Y_m) vers GTKmm (X_f, Y_f).

Dessiner avec les fonctions GTKmm de dessin avec les coordonnées (X_f, Y_f)

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

$$X_f = \text{width} * (X_m - X_{\min}) / (X_{\max} - X_{\min})$$

$$Y_f = \text{height} * (Y_{\max} - Y_m) / (Y_{\max} - Y_{\min})$$

De la fenêtre du dessin
à l'espace du Modèle

$$X_m = (X_f / \text{width}) * (X_{\max} - X_{\min}) + X_{\min}$$

$$Y_m = Y_{\max} - (Y_f / \text{height}) * (Y_{\max} - Y_{\min})$$

Visualisation: de l'espace du Modèle (X_m, Y_m) à celui de la fenêtre (X_f, Y_f)

4.2) Alternative avec transformations translate et scale de GTKmm Cairo

Dessiner avec les fonctions GTKmm de dessin avec les coordonnées (x, y) dans (O_m, X_m, Y_m)

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 (O_m, X_m, Y_m) :

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

b `cr->scale(width/ΔX, -height/ΔY);`

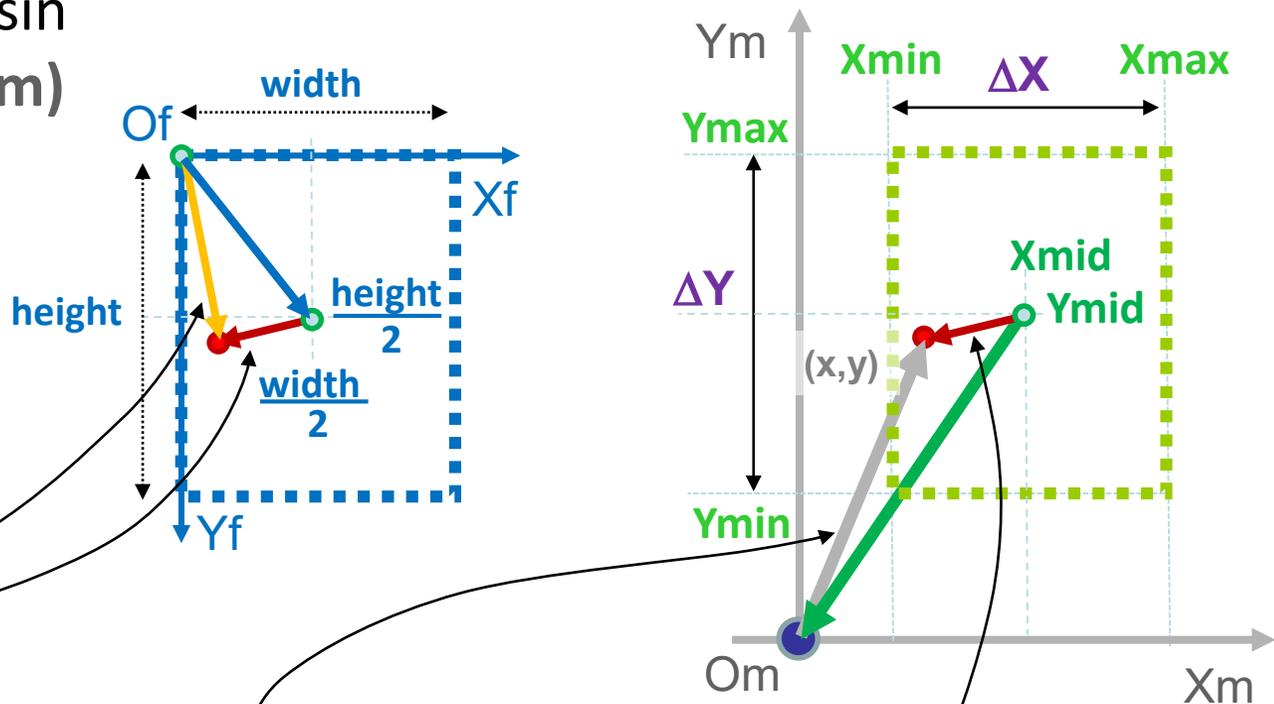
c `cr->translate(-Xmid, -Ymid);`

Examinons l'effet des trois transformations en partant du vecteur (x, y) dans (O_m, X_m, Y_m)

c L'addition du vecteur $(-X_{mid}, -Y_{mid})$ 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 (O_f, X_f, Y_f)



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

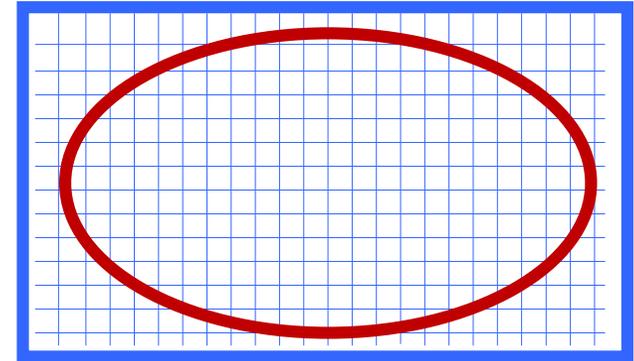
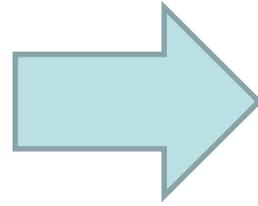
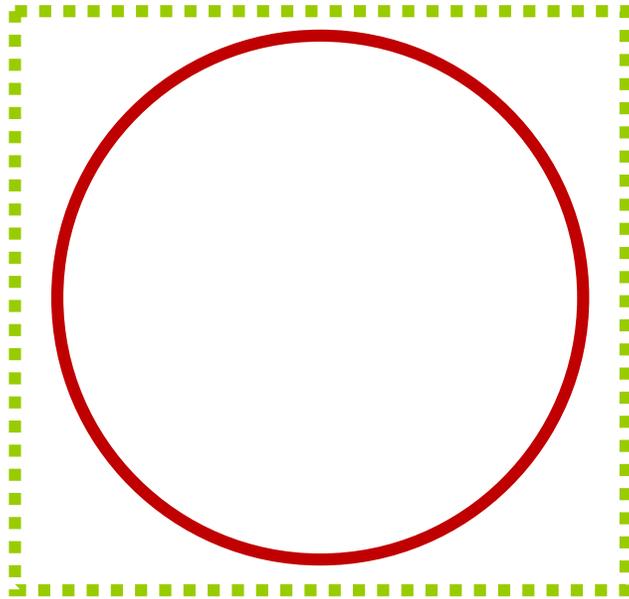
$$X_f = \left[\frac{\text{width}}{X_{\max} - X_{\min}} \right] * (X_m - X_{\min})$$
$$Y_f = \left[\frac{\text{height}}{Y_{\max} - Y_{\min}} \right] * (Y_{\max} - Y_m)$$

$$\left[\frac{\text{width}}{X_{\max} - X_{\min}} \right] = \left[\frac{\text{height}}{Y_{\max} - Y_{\min}} \right]$$

$$\frac{\text{width}}{\text{height}} = \frac{X_{\max} - X_{\min}}{Y_{\max} - Y_{\min}}$$

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

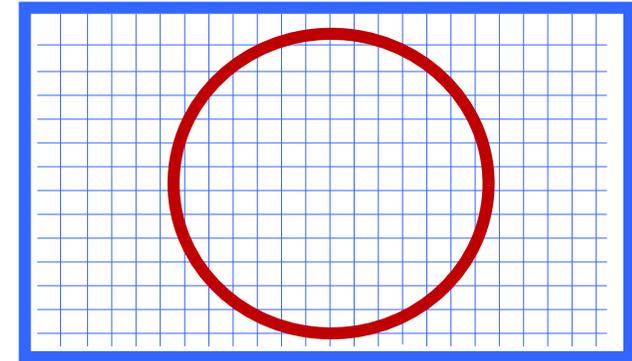
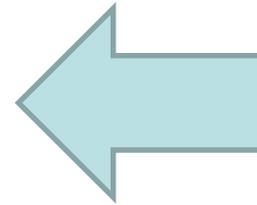
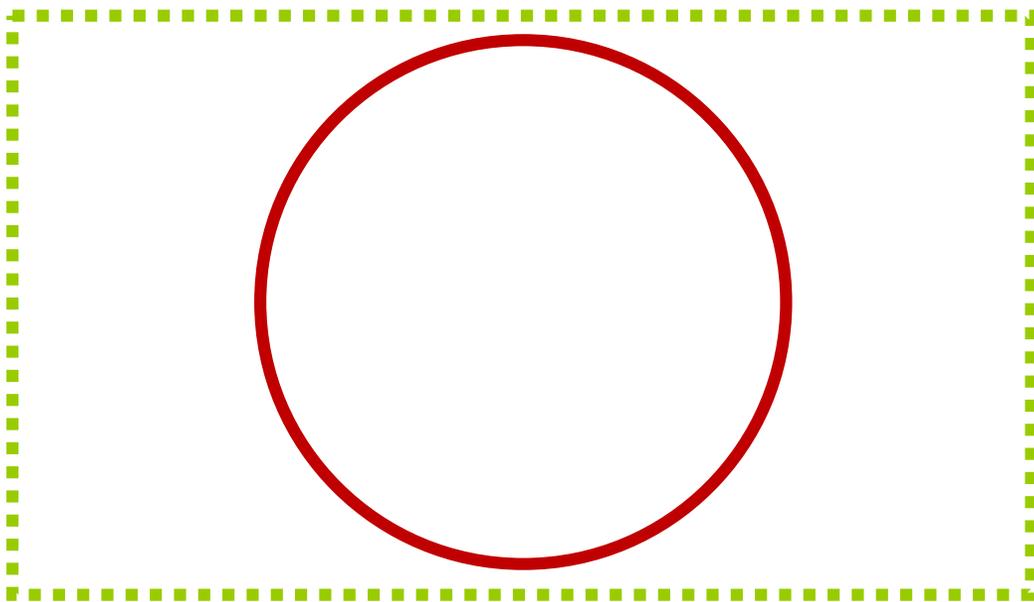
Exemple pour lequel l'aspect ratio est différent



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:
 $(X_{\max} - X_{\min}) / (Y_{\max} - Y_{\min}) = \text{width/height}$

Il faut choisir un critère supplémentaire, par exemple garder constant **$(X_{\max} - X_{\min})$** ou **$(Y_{\max} - Y_{\min})$** et ajuster l'autre terme.

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** qui est responsable des conversions de coordonnées entre le **Modèle** et la fenêtre de dessin
- On doit écrire soi-même les fonctions de conversion au niveau du système de **Visualisation** pour convertir les coordonnées fournies par le **Modèle** en celles de la fenêtre graphique GTKmm