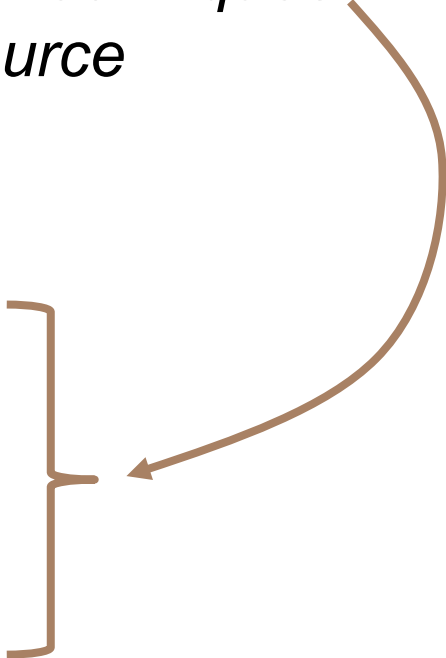


# GTKmm4 : Bases et Visualisation

*La série4 niveau0 développe les détails techniques  
et fournit l'ensemble du code source*

## Plan:

- Première application GTKmm4
  - Un premier bouton
  - l'API Cairomm pour le dessin
  - Le premier dessin avec MyDrawingArea
  - Conversions entre espaces du Modèle et du dessin
- 

# Première Application GTKmm4

API de GTKmm

```
#include <gtkmm.h>
```

```
class MyWindow : public Gtk::Window
```

```
{  
public:  
    MyWindow();  
};
```

définition d'une classe *MyWindow*  
dérivée de *Gtk::Window* que l'on  
peut adapter à nos besoins

```
MyWindow::MyWindow()
```

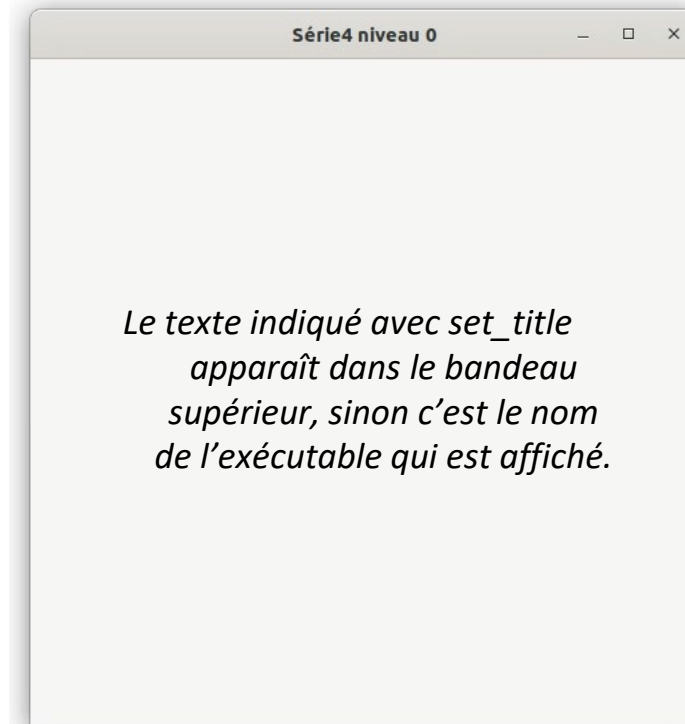
```
{  
    set_title("Série4 niveau 0");  
    set_default_size(500, 500);  
}
```

```
int main(int argc, char ** argv)
```

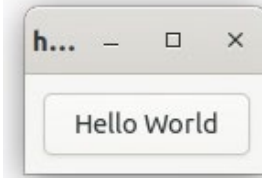
```
{  
    auto app = Gtk::Application::create();
```

```
    return app->make_window_and_run<MyWindow>(argc, argv);  
}
```

appel d'une méthode **template** C++ dont le code est adapté au type `<MyWindow>` :  
création et lancement d'une fenêtre *MyWindow*



# Premier bouton avec GTKmm4 (1)



main.cc

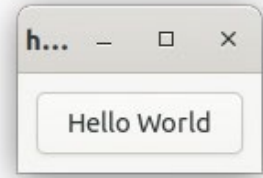
```
#include "helloworld.h"
#include <gtkmm/application.h>

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

    //Shows the window and returns when it is closed.
    return app->make_window_and_run<HelloWorld>(argc,argv);
}
```

+ module helloworld.cc

# Premier bouton avec GTKmm4 (2)



helloworld.cc  
helloworld.h

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

class HelloWorld : public Gtk::Window
{
public:
    HelloWorld();
    ~HelloWorld() override;

protected:
    //Signal handlers:
    void on_button_clicked();

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

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

HelloWorld::HelloWorld()
    : m_button("Hello World")
{
    m_button.set_margin(10);

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

    set_child(m_button); // ajoute à la fenêtre
}

HelloWorld::~HelloWorld() {}

void HelloWorld::on_button_clicked()
{
    static unsigned count(0);
    std::cout << ++count
        << " Hello World \a" << 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 définir la méthode **on\_draw()**

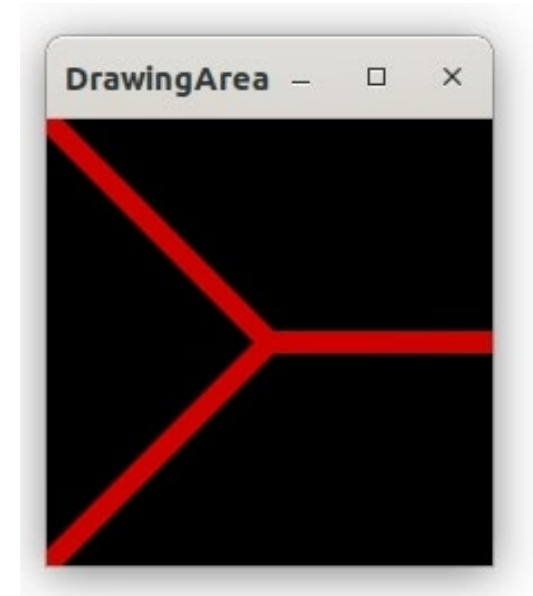
Elle est aussi appelée automatiquement quand le système détecte que la fenêtre a besoin d'être redessinée.

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 avec `paint()`

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

```
cr->paint();
```

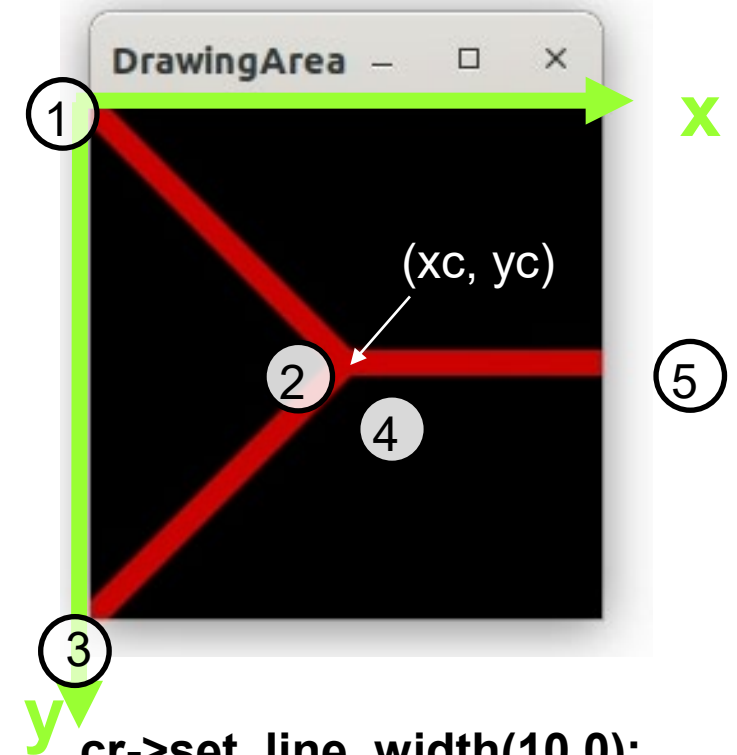


Créer une ligne ou polyline / concept de «**path**»:

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

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

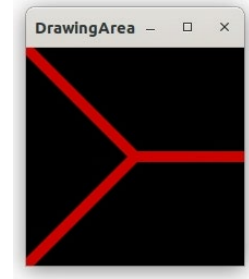
```
class ExampleWindow : public Gtk::Window  
{  
public:  
    ExampleWindow();  
protected:  
    MyArea m_area;  
};
```

```
ExampleWindow::ExampleWindow()  
{  
    set_title("DrawingArea");  
    set_child(m_area);  
}
```

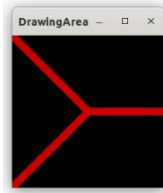
```
int main(int argc, char** argv)  
{  
    auto app = Gtk::Application::create();  
    return app->make_window_and_run<ExampleWindow>(argc, argv);  
}
```

main.cc

+ 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:
    void on_draw(const
        Cairo::RefPtr<Cairo::Context>& cr,
        int width, int height);
};

#endif // GTKMM_EXAMPLE_MYAREA_H
```

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

MyArea::MyArea()
{
    set_draw_func(sigc::mem_fun(*this, &MyArea::on_draw));
}

MyArea::~MyArea() {}

void MyArea::on_draw(const Cairo::RefPtr<Cairo::Context>& cr
                    int width, int 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();
}

}

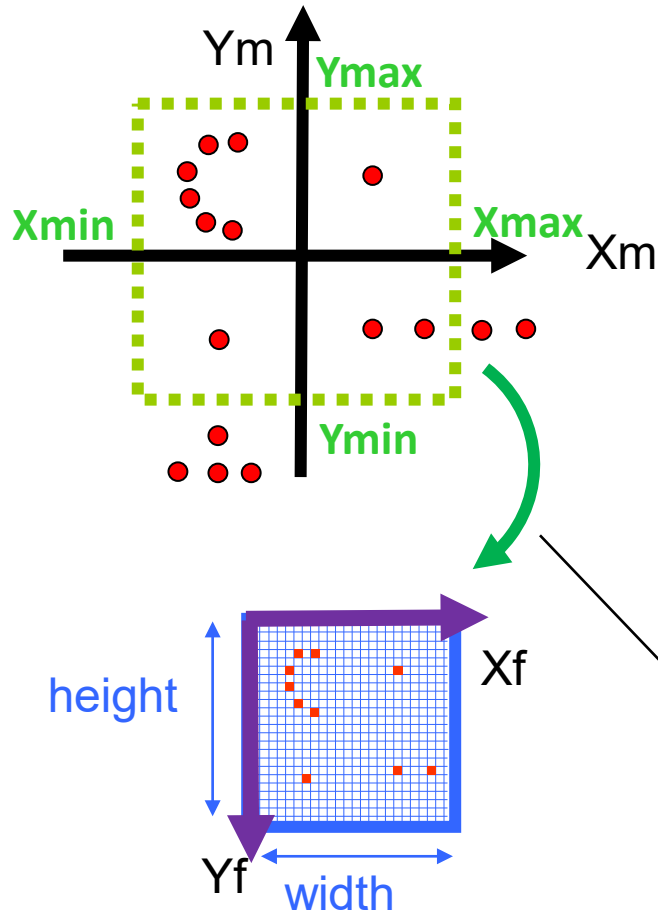
Définition du tracé  
«path» mais sans  
faire le dessin

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) ***transformations translate et scale de GTKmm Cairo***  
Dessiner avec les fonctions GTKmm de dessin avec les coordonnées **( $X_m, Y_m$ )**

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

## 4) Préparation avec des transformations translate et scale de GTKmm Cairo

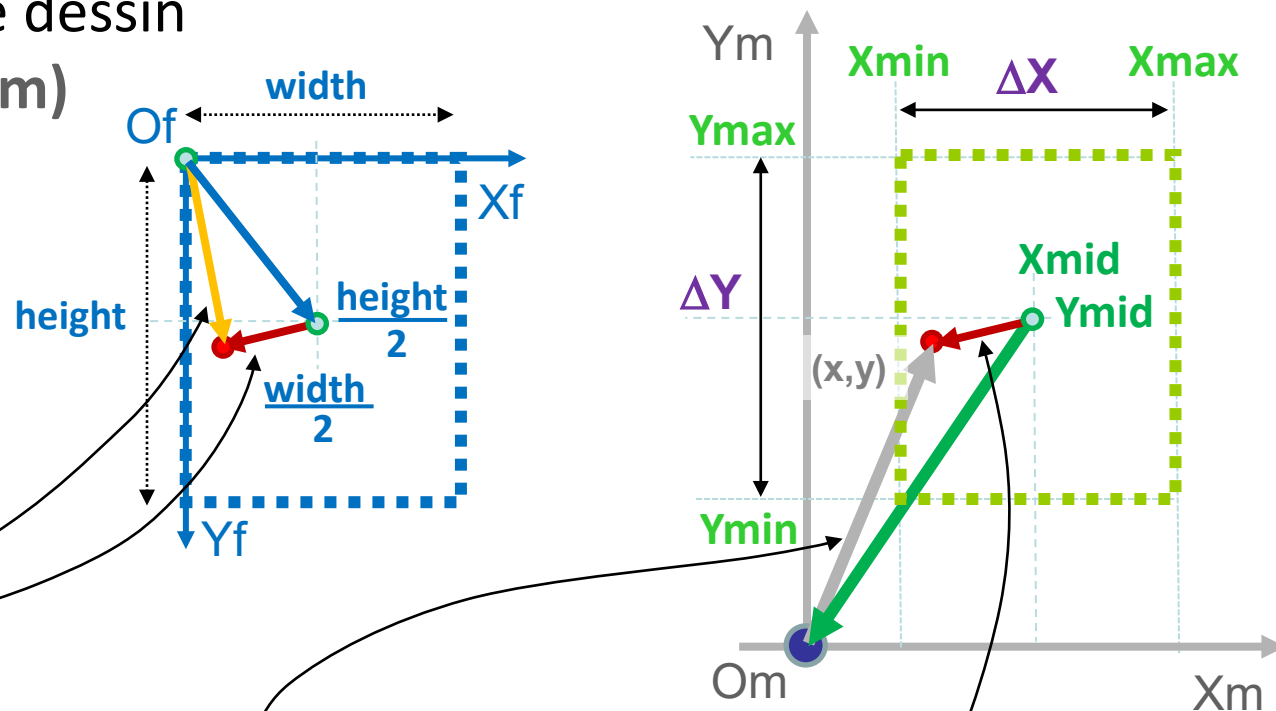
Puis dessiner avec les fonctions GTKmm de dessin avec les coordonnées  $(x, y)$  dans  $(O_m, X_m, Y_m)$

Effectuer ces 3 transformations dans `on_draw()` dans cet ordre a-b-c avant de faire des appels de dessin avec des coordonnées dans l'espace du MODELE ( $O_m, X_m, Y_m$ ) :

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  $(O_f, X_f, Y_f)$

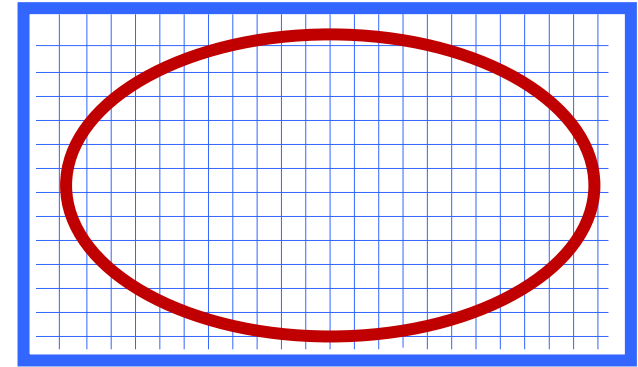
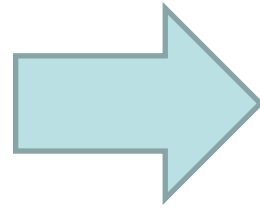
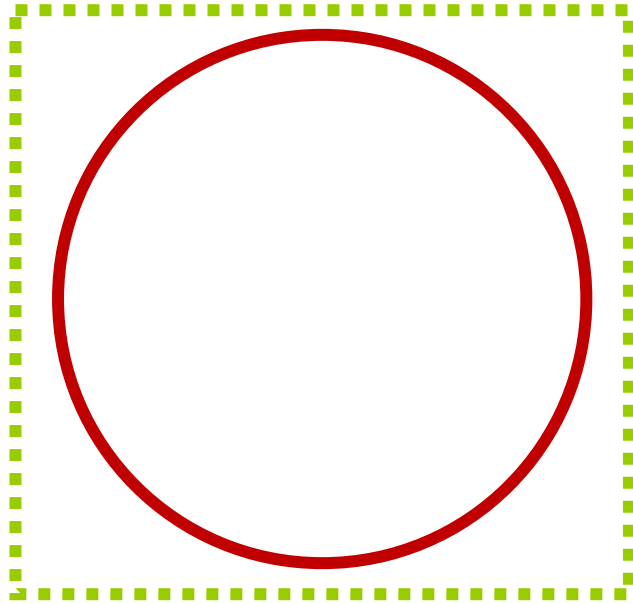
Pour éviter toute distortion à l'affichage  
Il faut garantir le même *facteur d'échelle* selon X et selon Y  
dans la transformation `cr->scale(width/ΔX, -height/ΔY);`

$$\text{width} / \Delta X = \text{height} / \Delta Y$$

$$\text{width/height} = \Delta X / \Delta Y$$

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

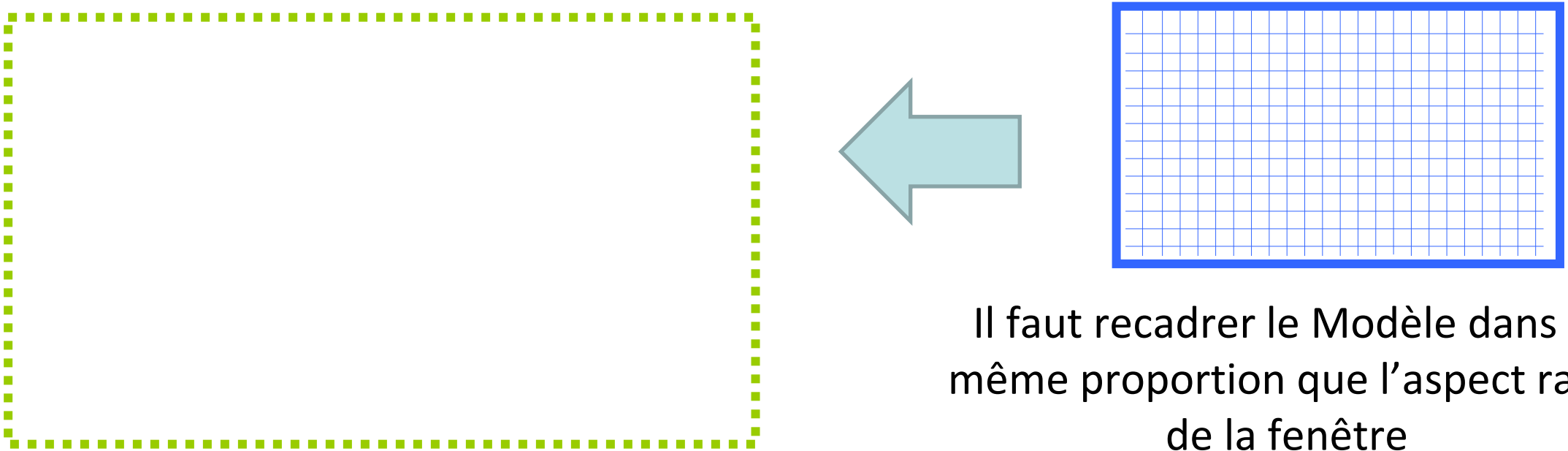
## Exemple pour lequel l'aspect ratio est différent



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

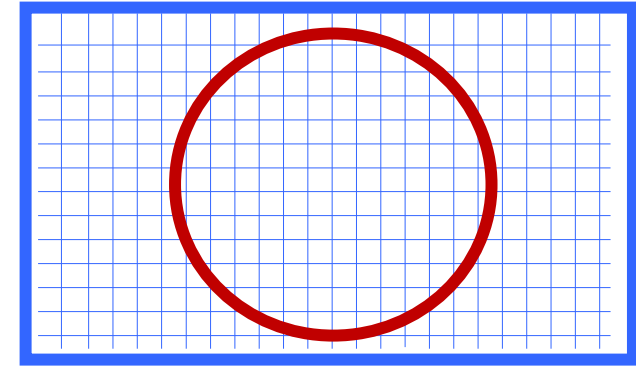
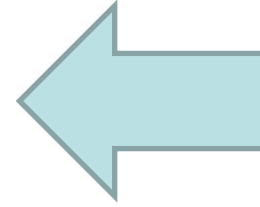
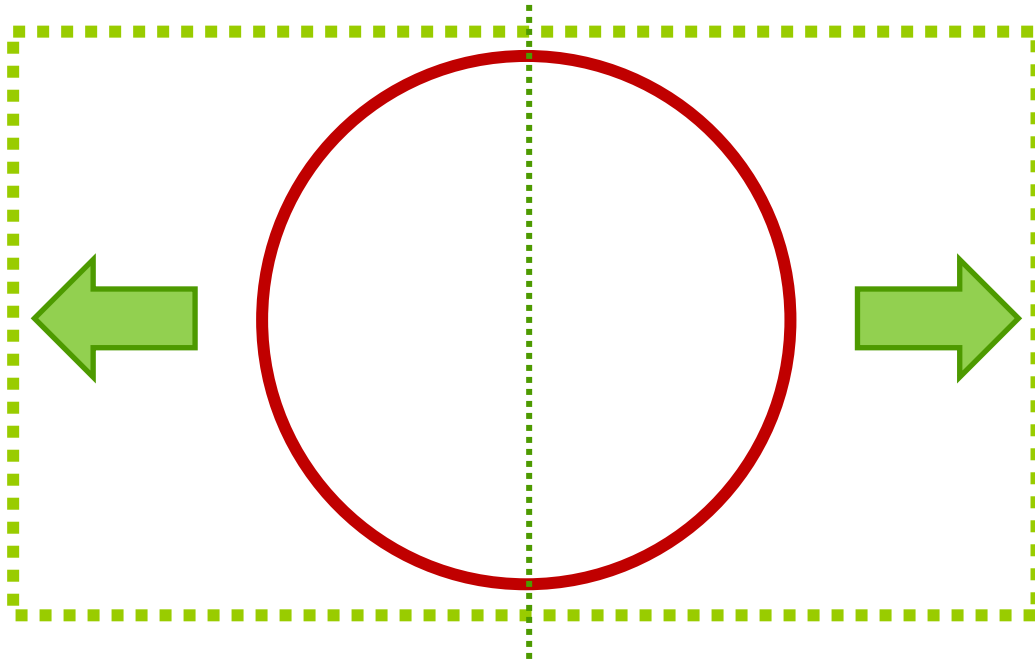


Il faut recadrer le Modèle dans la même proportion que l'aspect ratio de la fenêtre

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.

## Exemple: conserver ( $Y_{\max} - Y_{\min}$ ) et ajuster ( $X_{\max} - X_{\min}$ )



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  $X_{\text{mid}} = (X_{\text{max}} + X_{\text{min}}) / 2$

Et d'élargir symétriquement l'intervalle selon X

Les nouvelles valeurs  $\Delta X'$ ,  $X_{\text{min}}'$  et  $X_{\text{max}}'$  sont alors:

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

$$X_{\text{min}}' = X_{\text{mid}} - \Delta X' / 2$$

$$X_{\text{max}}' = X_{\text{mid}} + \Delta X' / 2$$

# 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.