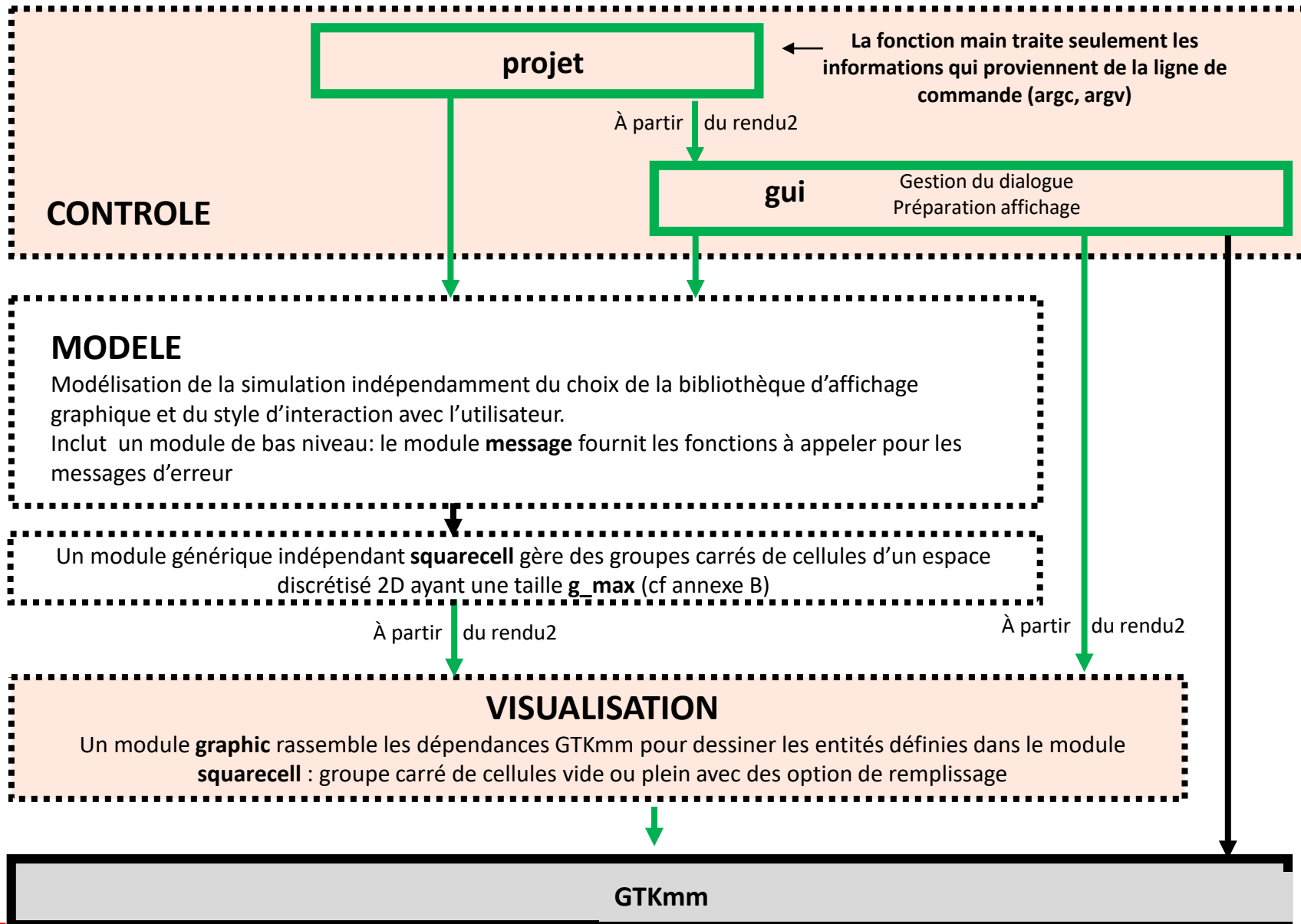


MVC et GTKmm



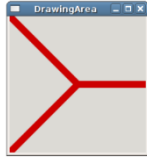
Comment garantir l'indépendance du Modèle vis-à-vis de GTKmm ?

Le module **graphic** regroupe les dépendances de dessin vis-à-vis d'une bibliothèque externe (GTKmm).

Il offre le moyen de dessiner des éléments simples en 2D.

On doit transmettre un pointeur
`Cairo::Context` au **Modèle**

⇒ Crée une dépendance
envers **GTKmm**



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 <caiomm/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

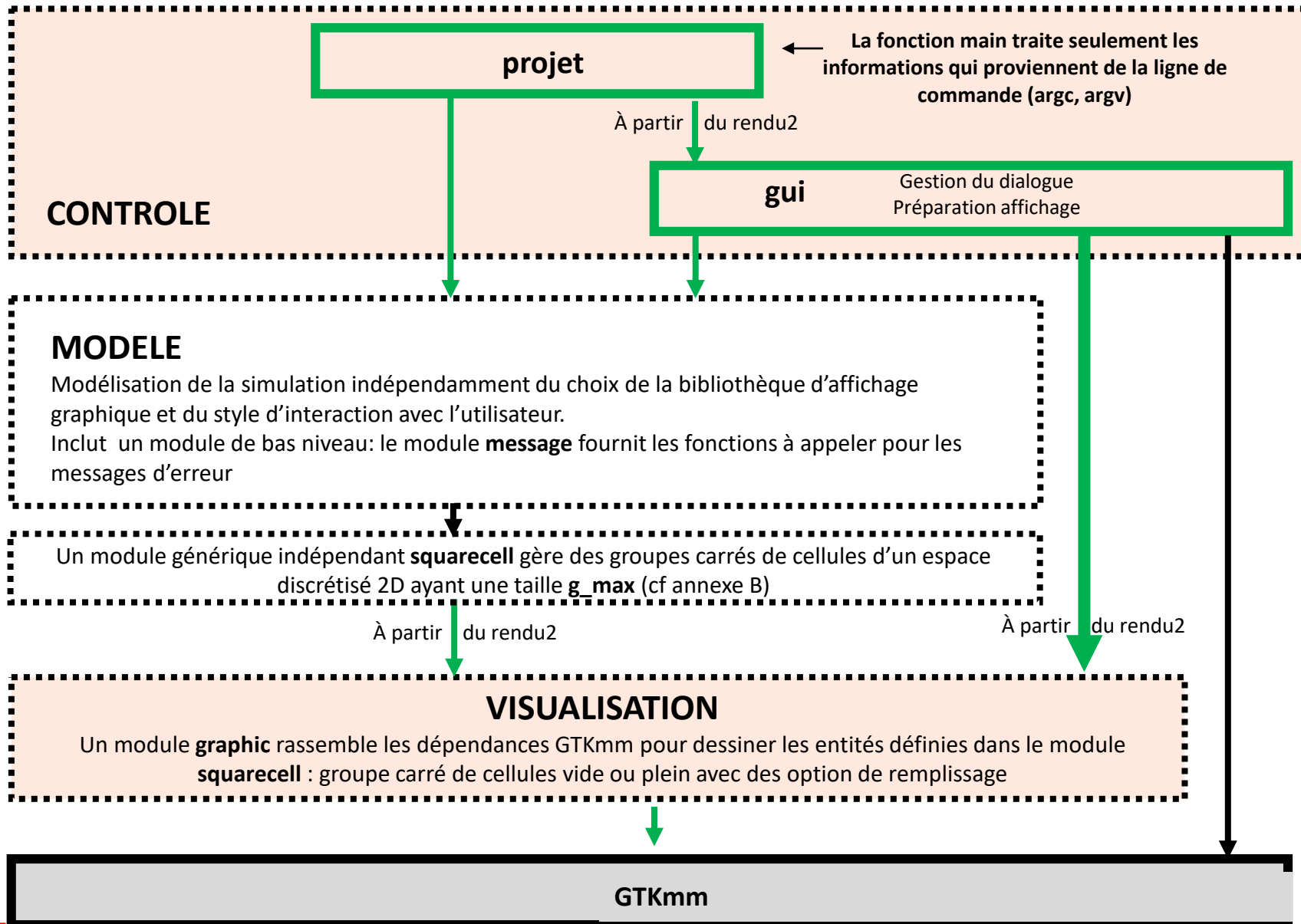
    // draw red lines out from the center of the window
    cr->set_source_rgb(0.8, 0.0, 0.0); // idem mémorisation cr
    cr->move_to(0, 0);
    cr->line_to(xc, yc);
    cr->line_to(0, height);
    cr->move_to(xc, yc);
    cr->line_to(width, yc);
    cr->stroke();

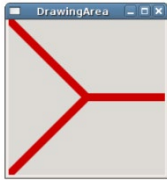
    return true;
}
```

Problème!

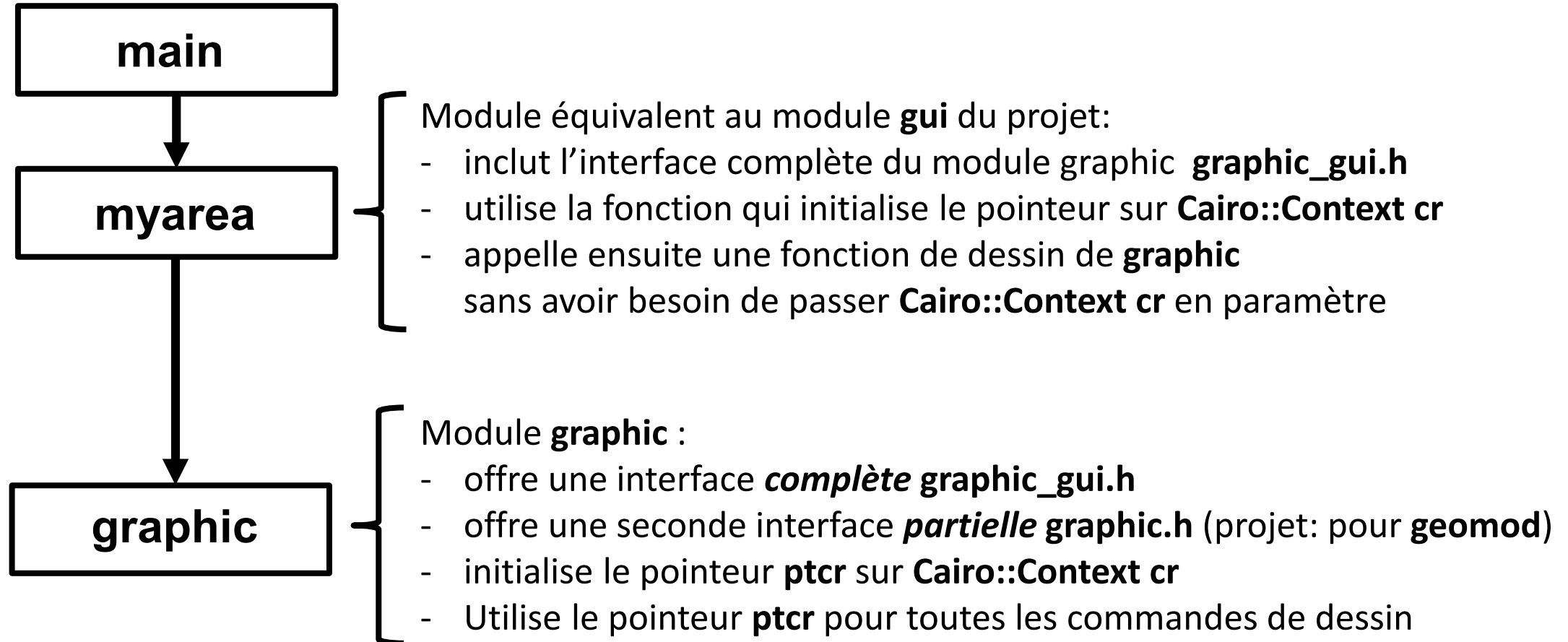
*Tous les appels définissant
les attribut du dessin et
effectuant le tracé ont
besoin du pointeur
`Cairo::Context`*

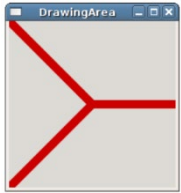
Solution : mémoriser un *pointeur* sur **Cairo::Context** dans **graphic.cc**





Exemple : GTKdrawingArea_avec_deux_modules (1)





GTKdrawingArea_avec_deux_modules (2)

main.cc

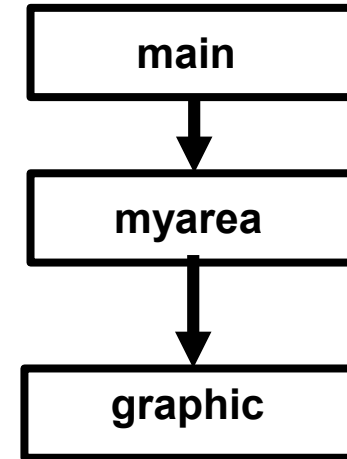
```
#include "myarea.h"
#include <gtkmm/application.h>
#include <gtkmm/window.h>

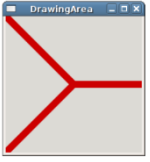
int main(int argc, char** argv)
{
    auto app = Gtk::Application::create();

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

    MyArea area;
    win.add(area);
    area.show();

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





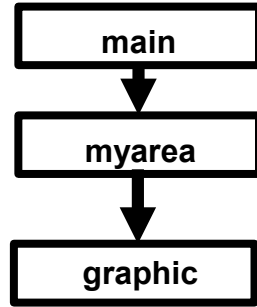
myarea.h

```
#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
```



GTKdrawingArea_avec_deux_modules (3)

myarea.cc

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

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

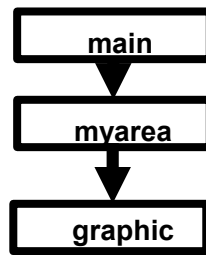
bool MyArea::on_draw(const Cairo::RefPtr<Cairo::Context>& cr)
{
    Gtk::Allocation allocation = get_allocation();
    const int width = allocation.get_width();
    const int height = allocation.get_height();

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

    graphic_set_context(cr);

    graphic_draw_shape(width, height, xc, yc);

    return true;
}
```



GTKdrawingArea_avec_deux_modules (4)

graphic.cc

graphic.h // interface *partielle*

```
#ifndef GTKMM_EXAMPLE_GRAPHIC_H
#define GTKMM_EXAMPLE_GRAPHIC_H

void graphic_draw_shape(const int width,
                        const int height, int xc, int yc);

#endif // GTKMM_EXAMPLE_GRAPHIC_H
```

graphic_gui.h // interface *complète*

```
#ifndef GTKMM_EXAMPLE_GRAPHIC_GUI_H
#define GTKMM_EXAMPLE_GRAPHIC_GUI_H

#include <gtkmm/drawingarea.h>
#include "graphic.h"

void graphic_set_context(const
                        Cairo::RefPtr<Cairo::Context>& cr);

#endif // GTKMM_EXAMPLE_GRAPHIC_GUI_H
```

```
#include "graphic_gui.h"

static const Cairo::RefPtr<Cairo::Context>* ptcr(nullptr);

void graphic_set_context(const Cairo::RefPtr<Cairo::Context>& cr)
{
    ptcr = &cr;
}

void graphic_draw_shape(const int width, const int height,
                        int xc, int yc)
{
    (*ptcr)->set_line_width(10.0);

    // draw red lines out from the center of the window
    (*ptcr)->set_source_rgb(0.8, 0.0, 0.0);
    (*ptcr)->move_to(0, 0);
    (*ptcr)->line_to(xc, yc);
    (*ptcr)->line_to(0, height);
    (*ptcr)->move_to(xc, yc);
    (*ptcr)->line_to(width, yc);
    (*ptcr)->stroke();
}
```

*initialise le pointeur ptcr
sur Cairo::Context cr*

*Utilise le pointeur ptcr
pour toutes les
commandes de dessin*