Cours d'Interfaces Graphiques n°3

Edouard THIEL

Faculté des Sciences
Université d'Aix-Marseille (AMU)

Janvier 2016

Les transparents de ce cours sont téléchargeables ici :
http://pageperso.lif.univ-mrs.fr/~edouard.thiel/ens/igra/

Lien court : http://j.mp/optigra

Plan du cours n°3

1. Widget `GtkDrawingArea`
2. Événements
3. Librairie Cairo - premiers pas

1 - Widget GtkDrawingArea

Zone de dessin GDK ou Cairo

```
GObject
    |-- GtkWidget
        |-- GtkDrawingArea
```

```
GtkWidget *area = gtk_drawing_area_new ();
```

Permet de gérer événements souris et clavier, via `GtkWidget`

N'est pas un conteneur

Événement traité

Nombreux signaux possibles pour un `GtkWidget`

Souvent plusieurs niveaux de callbacks

```
gboolean on_area_something (GtkWidget *area, ...)
{
  ...
  return TRUE;  // événement traité
}
```

Renvoie `TRUE` : pas d'autre callback appelée

Signal `draw`

Émis lorsque le widget doit redessiner son contenu

→ C'est uniquement dans cette fonction que l'on dessine

```
g_signal_connect (area, "draw",
                  G_CALLBACK (on_area_draw), data);

gboolean on_area_draw (GtkWidget *area,
                       cairo_t *cr, gpointer data)
{
  // on dessine avec cr

  return TRUE;  // événement traité
}
```

Vidé à la couleur du fond avant l'appel

Les dessins sont rognés

Signal size-allocate

Émis lorsque le widget change de taille

```
g_signal_connect (area, "size-allocate",
                  G_CALLBACK (on_area_size_allocate), data);

void on_area_size_allocate (GtkWidget *area,
                            GdkRectangle *rect, gpointer data)
{
  printf ("New size: %d * %d\n",
          rect->width, rect->height);
}
```

Demander taille minimale : important si area est dans un GtkScrolledWindow

```
gtk_widget_set_size_request (area, width, height);
```

Récupérer la taille de area

- On stocke la taille dans `data` $\rightarrow$ peut être utilisé n'importe quand

```
void on_area_size_allocate (GtkWidget *area, .., data)
{
  data->area_width  = rect->width;
  data->area_height = rect->height;
}
```

- On récupère la taille lors d'un signal $\rightarrow$ utilisable uniquement dans la callback

```
gboolean on_area_draw (GtkWidget *area, ..)
{
  int width  = gtk_widget_get_allocated_width  (area),
      height = gtk_widget_get_allocated_height (area);

}
```

Provoquer le ré-affichage

```
void refresh_area (GtkWidget *area)
{
  GdkWindow *win = gtk_widget_get_window (area);

  if (win == NULL) return;  // widget pas encore réalisé

  gdk_window_invalidate_rect (
        win,
        NULL,   // tout le window
        FALSE); // pas de sous-window
}
```

→ Provoque envoi signal `draw`

Compression des signaux `draw` ;
double ou triple buffer pour animations

2 - Événements

Événements utilisateurs $\rightarrow$ signaux de `GtkWidget`

- clavier : touche frappée/relachée, focus
- souris : clic, déplacement, franchissement fenêtre
- double clic, touchpad, drag and drop, ...

Callbacks signaux : reçoient `GdkEvent *event`

Type union GdkEvent

Union de struct :

```
typedef union {
    GdkEventType      type;
    GdkEventAny       any;
    GdkEventKey       key;
    GdkEventFocus     focus_change;
    GdkEventButton    button;
    GdkEventMotion    motion;
    GdkEventCrossing  crossing;
    ...
} GdkEvent;
```

Champs communs

```
typedef union {
    GdkEventType     type;
    GdkEventAny      any;
    ...
} GdkEvent;
```

Champ `type` : en premier dans tous les struct

Enum : `GDK_BUTTON_PRESS`, `GDK_BUTTON_RELEASE`, ...

Champ `any` : trois premiers champs communs

```
typedef struct {
    GdkEventType type;
    GdkWindow *window;  // window qui a reçu l'événement
    gint8 send_event;   // TRUE si envoi par le programme
} GdkEventAny;
```

Usage de GdkEvent

Événement → appel callback, qui reçoit `GdkEvent *event`

Une callback / signal → `event->type` peut être ignoré

```
on_area_key_press (..., GdkEvent *event, ...)
{
  GdkEventKey *evk = &event->key;
  ...
}
on_area_button_press (..., GdkEvent *event, ...)
{
  GdkEventButton *evb = &event->button;
  ...
}
on_area_motion_notify (..., GdkEvent *event, ...)
{
  GdkEventMotion *evm = &event->motion;
  ...
}
```

Touche clavier

Signaux `key-press-event`, `key-release-event`

```
g_signal_connect (area, "key-press-event",
                  G_CALLBACK (on_area_key_press), data);
g_signal_connect (area, "key-release-event",
                  G_CALLBACK (on_area_key_release), data);

gboolean on_area_key_press (GtkWidget *area,
                            GdkEvent *event, gpointer data)
{ ... }

gboolean on_area_key_release (GtkWidget *area,
                              GdkEvent *event, gpointer data)
{ ... }
```

Répétition automatique

Exemple typique

```
gboolean on_area_key_press (GtkWidget *area,
                            GdkEvent *event, gpointer data)
{
  GdkEventKey *evk = &event->key;

  printf ("%s: GDK_KEY_%s\n",
          __func__, gdk_keyval_name(evk->keyval));

  switch (evk->keyval) {
    case GDK_KEY_q : quitter_appli(); break;
    ...
  }

  return TRUE;  // événement traité
}
```

Focus clavier

Autoriser focus : `gtk_widget_set_can_focus (area, TRUE);`

Prendre focus : `gtk_widget_grab_focus (area);`
(le widget doit être réalisé)

Détecter changement de focus : signaux `focus-in-event`, `focus-out-event`

Accepter des événements

Masque d'événements utilisateurs

```
gtk_widget_add_events (area,
  GDK_KEY_PRESS_MASK | GDK_KEY_RELEASE_MASK |
  GDK_FOCUS_CHANGE_MASK |
  GDK_BUTTON_PRESS_MASK | GDK_BUTTON_RELEASE_MASK |
  GDK_POINTER_MOTION_MASK |
  GDK_ENTER_NOTIFY_MASK | GDK_LEAVE_NOTIFY_MASK |
  ...
);
```

Clic souris

Signaux `button-press-event`, `button-release-event`

```
g_signal_connect (area, "button-press-event",
                  G_CALLBACK (on_area_button_press), data);
g_signal_connect (area, "button-release-event",
                  G_CALLBACK (on_area_button_release), data);

gboolean on_area_button_press (GtkWidget *area,
                               GdkEvent *event, gpointer data)
{ ... }

gboolean on_area_button_release (GtkWidget *area,
                                 GdkEvent *event, gpointer data)
{ ... }
```

Exemple d'usage

```
gboolean on_area_button_press (GtkWidget *area,
                               GdkEvent *event, gpointer data)
{
  GdkEventButton *evb = &event->button;

  printf ("%s: %d %.1f %.1f\n",
          __func__, evb->button, evb->x, evb->y);

  return TRUE;   // événement traité
}
```

Déplacement souris

Signal `motion-notify-event`

```
g_signal_connect (area, "motion-notify-event",
                  G_CALLBACK (on_area_motion_notify), data);

gboolean on_area_motion_notify (GtkWidget *area,
                                GdkEvent *event, gpointer data)
{
  GdkEventMotion *evm = &event->motion;

  printf ("%s: %.1f %.1f\n", __func__, evm->x, evm->y);

  return TRUE;   // événement traité
}
```

`evm` ne contient pas numéro bouton $\rightarrow$ sauver dans `data`
Drag hors fenêtre

Franchissement de la fenêtre

Signaux `enter-notify-event`, `leave-notify-event`

```
g_signal_connect (area, "enter-notify-event",
                  G_CALLBACK (on_area_enter_notify, data);
g_signal_connect (area, "leave-notify-event",
                  G_CALLBACK (on_area_leave_notify), data);

gboolean on_area_enter_notify (GtkWidget *area,
                               GdkEvent *event, gpointer data)
{ ... }

gboolean on_area_leave_notify (GtkWidget *area,
                               GdkEvent *event, gpointer data)
{ ... }
```

Fin drag $\rightarrow$ `leave-notify-event`

Usage conseillé

```
gboolean on_area_enter_notify (GtkWidget *area,
                               GdkEvent *event, gpointer data)
{
  GdkEventCrossing *evc = &event->crossing;

  printf ("%s: %.1f %.1f\n", __func__, evc->x, evc->y);

  // On avait peut-être perdu le focus
  gtk_widget_grab_focus (area);

  return TRUE;   // événement traité
}
```

3 - Librairie Cairo - premiers pas

Librairie graphique vectorielle 2D

Utilise accélération matérielle

Depuis GTK 2.8, tous les widgets sont rendus avec Cairo

Principe :

- on crée un chemin invisible
- on rend ensuite visible
  - le chemin : opération `stroke`
  - son intérieur : opération `fill`

Contexte graphique `cairo_t`

Contient tous les paramètres graphiques :

- épaisseur de ligne
- pointillé
- couleur
- la surface de dessin (window, pdf, svg, ...)
- ...

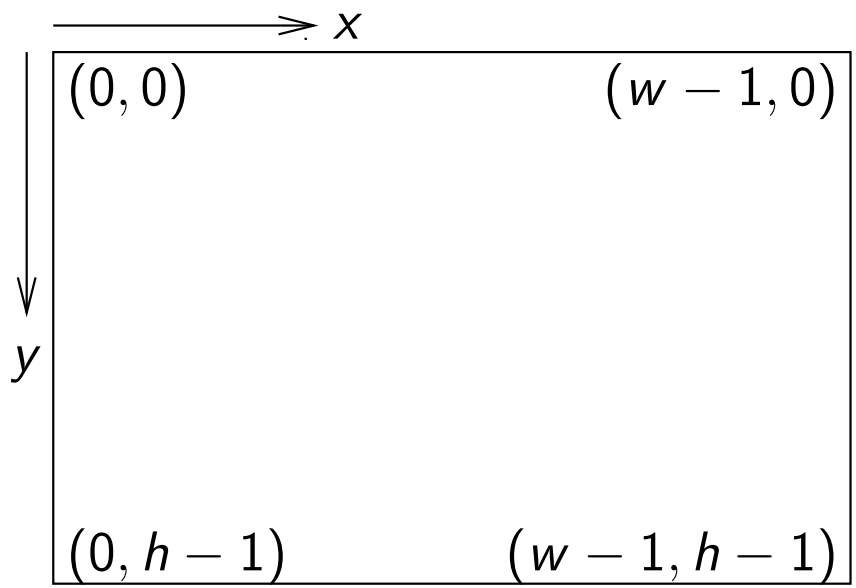
Intérêt : moins de paramètres aux fonctions Cairo

```
gboolean on_area_draw (GtkWidget *area,
                       cairo_t *cr, gpointer data)
```

`cr` directement utilisable
Réinitialisé à chaque appel

Coordonnées

Coordonnées relatives au widget
Coordonnées dessin = coordonnées souris

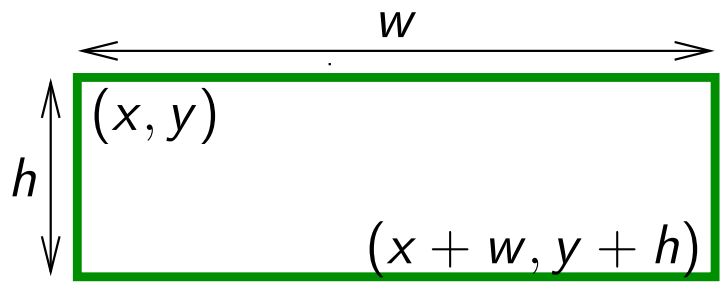


Coordonnées réelles
Dessins rognés automatiquement

Dessiner un rectangle

```
void cairo_set_source_rgb (cairo_t *cr,
    double red, double green, double blue); // entre 0 et 1

void cairo_rectangle (cairo_t *cr,
    double x, double y, double width, double height);
```



Exemple :

```
cairo_set_line_width (cr, 1.5);
cairo_set_source_rgb (cr, 0, 1.0, 0);
cairo_rectangle (cr, 10.0, 20.0, 50.0, 40.0);
cairo_stroke (cr);

cairo_set_source_rgb (cr, 0, 0, 1.0);
cairo_rectangle (cr, 10.0, 70.0, 50.0, 40.0);
cairo_fill (cr);
```

Fill et stroke

`cairo_stroke (cr)` : "faire des traits"

- dessine le chemin
- puis supprime le chemin

`cairo_fill (cr)` : "remplir"

- remplis le chemin
- puis supprime le chemin

→ Après un fill ou un stroke, on commence un nouveau chemin

Le contexte est conservé : couleur, épaisseur, etc

On peut conserver le chemin :

```
cairo_stroke_preserve (cr)
cairo_fill_preserve (cr)
```

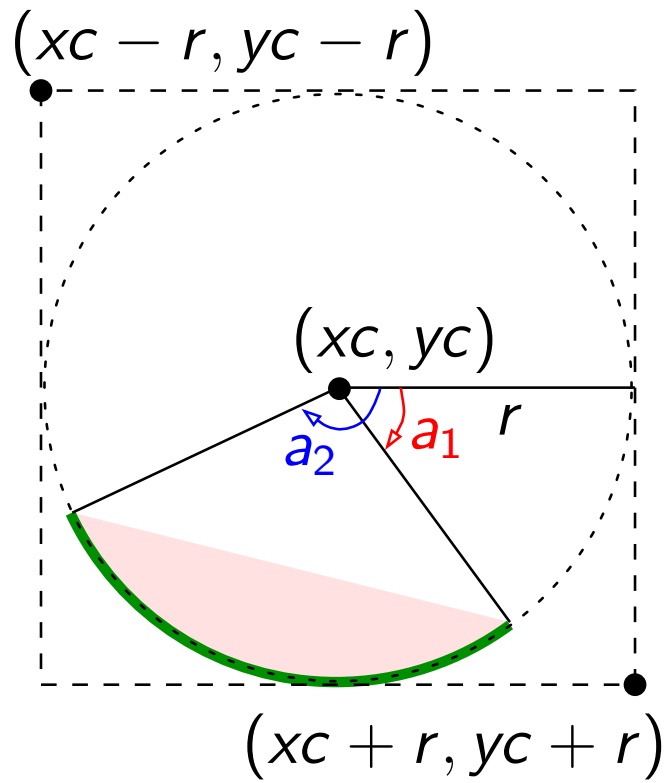
Dessiner un cercle

```
void cairo_arc (cairo_t *cr,
    double xc, double yc,              // centre
    double radius,                     // rayon
    double angle1, double angle2);     // en radians
```

Consertion en degrés :

```
degree * G_PI / 180
```

Cercle : `0, G_PI*2`



Dessiner une ligne polygonale

```
void cairo_move_to (cairo_t *cr,
    double x, double y);                // nouveau sous-chemin

void cairo_line_to (cairo_t *cr,
    double x, double y);                // ajoute un segment

void cairo_close_path (cairo_t *cr);  // ferme le sous-chemin
```

Le point courant devient (x, y)

Afficher un morceau d'image

On donne comme source un pixbuf :

```
void gdk_cairo_set_source_pixbuf (cairo_t *cr,
     const GdkPixbuf *pixbuf,
     double px, double py);     // origine dans pixbuf
```

On crée ensuite un chemin :

```
cairo_rectangle
cairo_arc
cairo_move_to cairo_line_to ... cairo_close_path
```

Enfin on découpe le pixbuf avec le chemin :

```
cairo_fill (cr);
```