Projet C
Interaction Graphique

Amphi de présentation

F. Bérard 26 mai 2016

Grenoble INP ensimag

Introduction

Objectifs

Apprentissage du langage C

Par la pratique

Apprentissage de la gestion de projet

Par la pratique

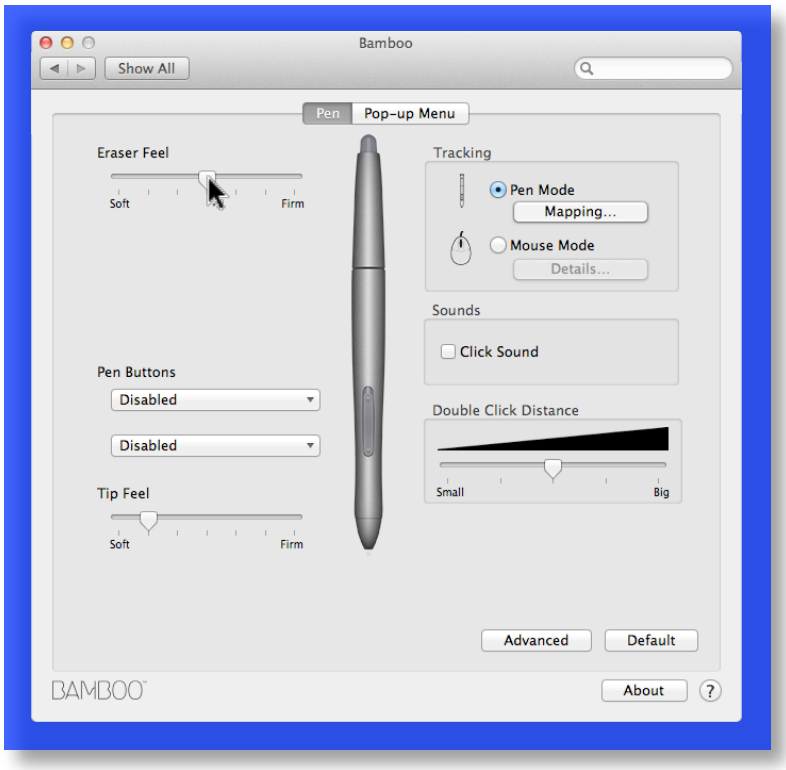
- Travail en groupe
- Travail d'envergure

Apprentissage de la programmation des interfaces graphiques

Par la réalisation d'une bibliothèque

Le sujet

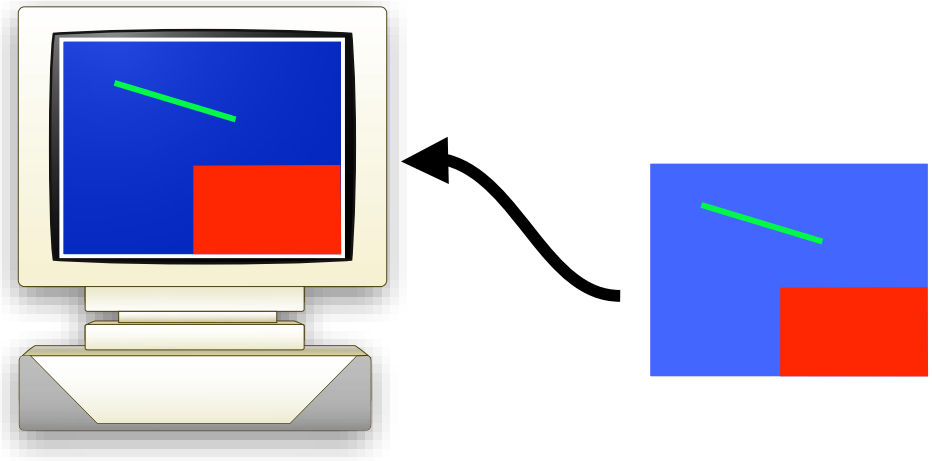
Réalisation d'un bibliothèque de programmation d'Interfaces Graphiques

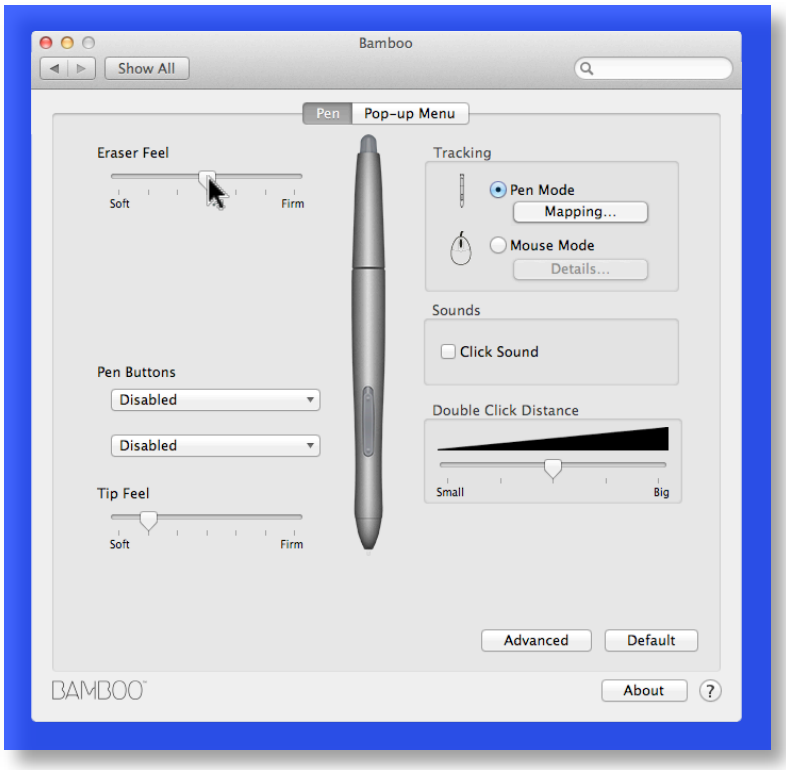


Le sujet

Réalisation d'un bibliothèque de programmation d'Interfaces Graphiques

En partant du *buffer graphique*



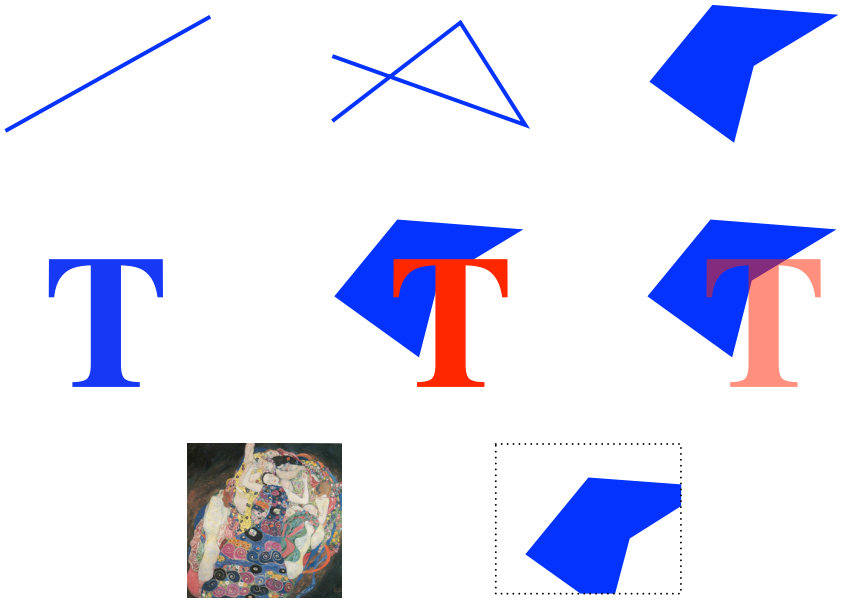


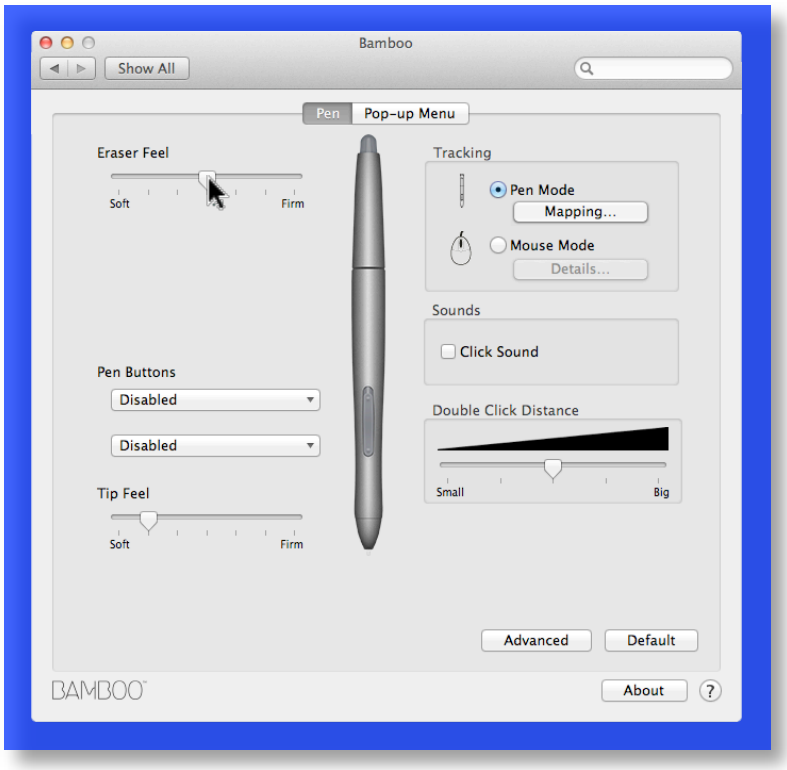
Le sujet

Réalisation d'un bibliothèque de programmation d'Interfaces Graphiques

En partant du *buffer graphique*,

des *primitives graphiques*





Le sujet

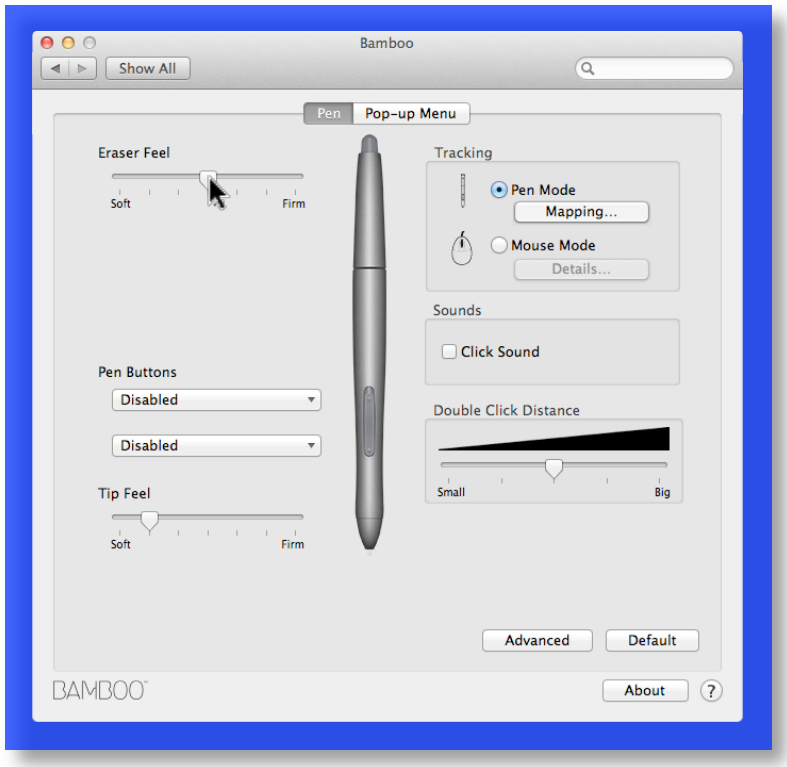
Réalisation d'un bibliothèque de programmation d'Interfaces Graphiques

En partant du *buffer graphique*,

des *primitives graphiques*,

des *événements utilisateur*

```
1876702   MouseMove          x=342 y=96
1876724   MouseMove          x=348 y=95
1877354   MouseButtonPress   n=1
1879265   MouseMove          x=328 y=90
1879287   MouseMove          x=302 y=86
1879302   MouseMove          x=299 y=80
1881076   MouseButtonRelease n=1
1892378   KeyPress           k="q"
```



Le sujet

Réalisation d'un bibliothèque de programmation d'Interfaces Graphiques

En partant du *buffer graphique,*

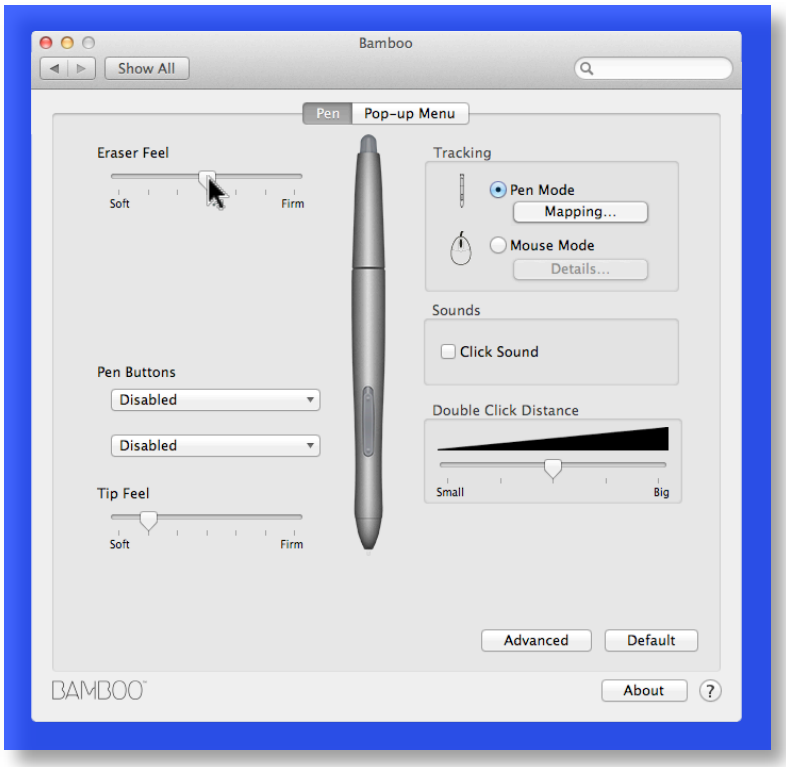
des *primitives graphiques,*

des *événements utilisateur,*

de la *structure* de la bibliothèque

(interface de programmation, ou API)
(fichiers .h fournis)

```
typedef void
     (*ei_widgetclass_drawfunc_t)
          (struct ei_widget_t*     widget,
           ei_surface_t            surface,
           ei_surface_t            pick_surface,
           ei_rect_t*              clipper);
```



Le sujet

Réalisation d'un bibliothèque de programmation d'Interfaces Graphiques

En partant du *buffer graphique,*

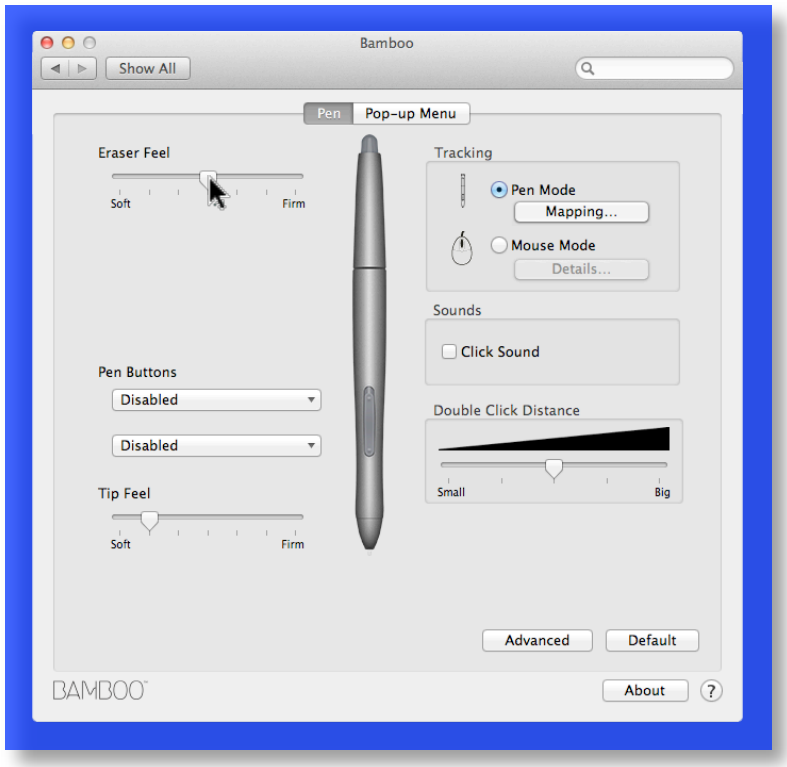
des *primitives graphiques,*

des *événements utilisateur,*

de la *structure* de la bibliothèque,

et d'*exemples d'applications*.

```
ei_widget_t*    button;
char            button_text[80] = "test";
int             but_x = 10, but_y = 12;

button = ei_widget_create("button",
                          ei_app_root_widget());
ei_button_configure(button, button_text);
ei_place(button, &but_x, &but_y);
```



Le sujet

Les acteurs

Les ***utilisateurs*** de d'applications

Les ***programmeurs d'applications***

Les programmeurs de la bibliothèque (***vous***)

Les ***concepteurs*** de la bibliothèque (nous, et vous en cas d'extensions)

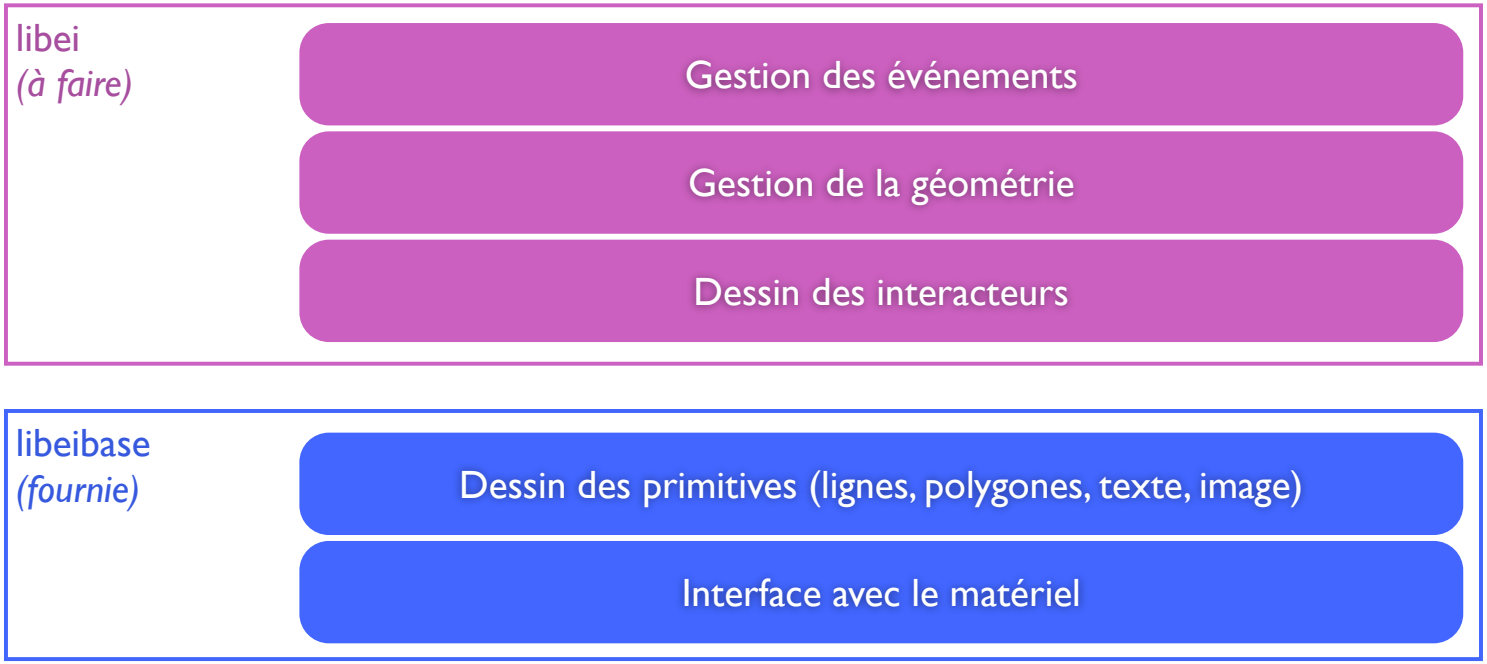
Services de la Bibliothèque

Survol

Création, configuration, et dessin des ***interacteurs*** (“***widgets***”)

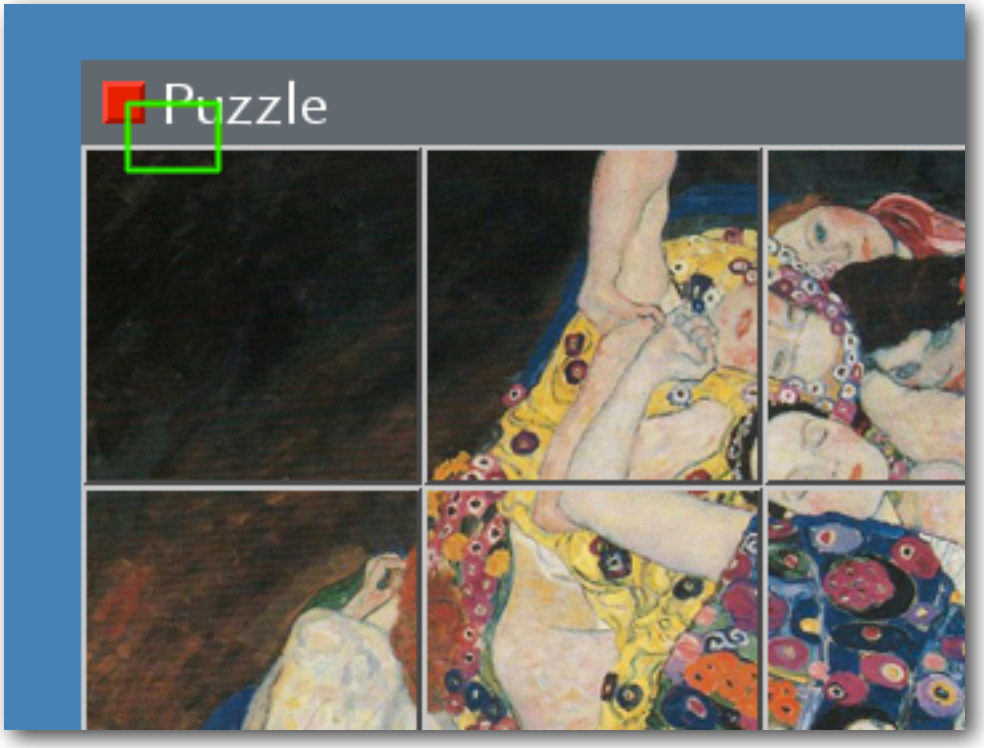
Placement à l’écran (position et taille) : gestion de la ***géométrie***

Prise en compte des actions utilisateur : gestion des ***événements***



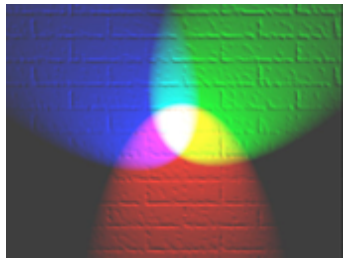
Génération d'Images Numériques

Représentation en mémoire



Représentation en mémoire

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB
1	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB
2	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB
3	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB
4	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB
5	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB
6	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB
7	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB
8	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB
9	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB
10	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB
11	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB
12	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB
13	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB
14	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB	RVB



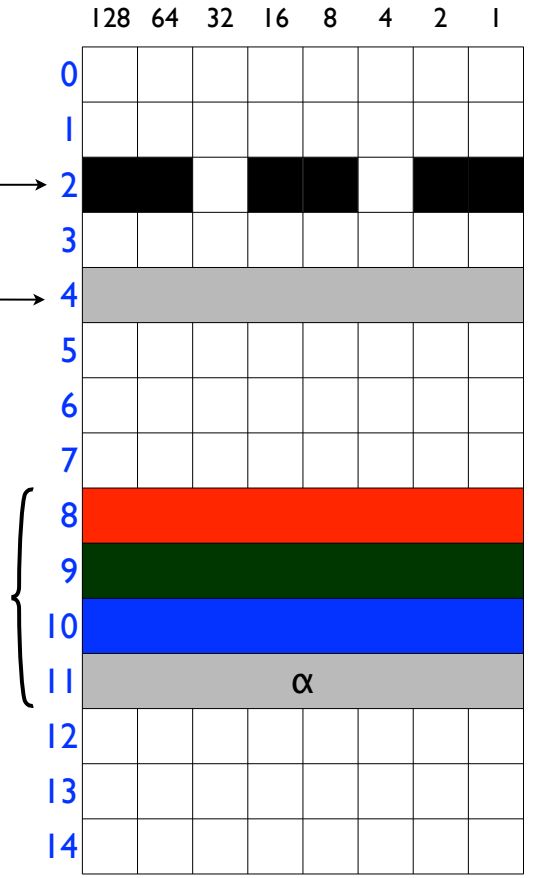
Représentation en mémoire

Différents formats de pixels

Image noir & blanc
1 pixel = 1 bit. 8 pixels dans un octet.

Image en niveaux de gris
256 niveaux de gris: 1 pixel = 1 octet

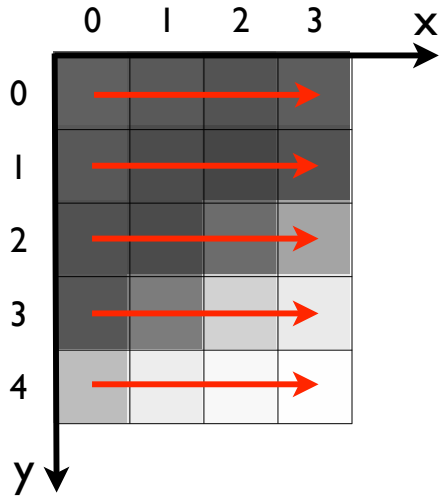
Image en couleur
+ transparence (alpha)
256 niveaux par composante R,V,B,A
(> 16 000 000 couleurs):
1 pixel = 4 octets (32 bits)
(l'ordre des canaux varie suivant les architectures)

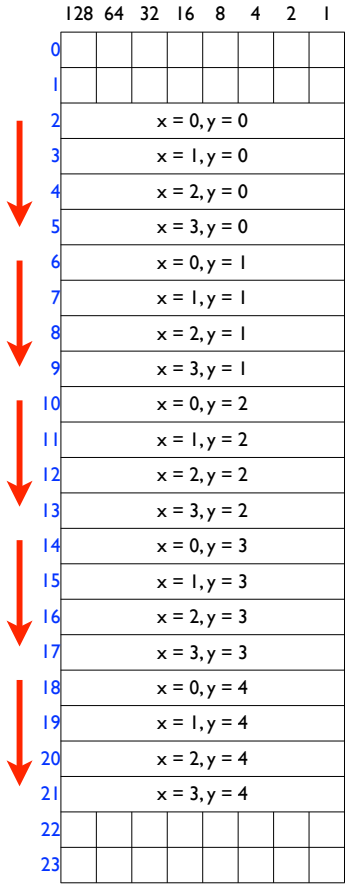


Représentation en mémoire

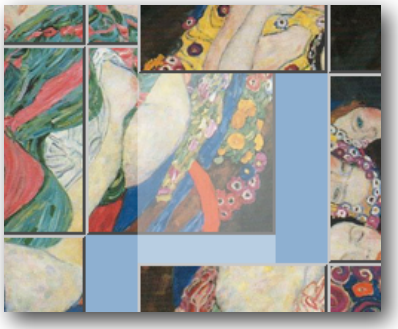
Ordre des pixels

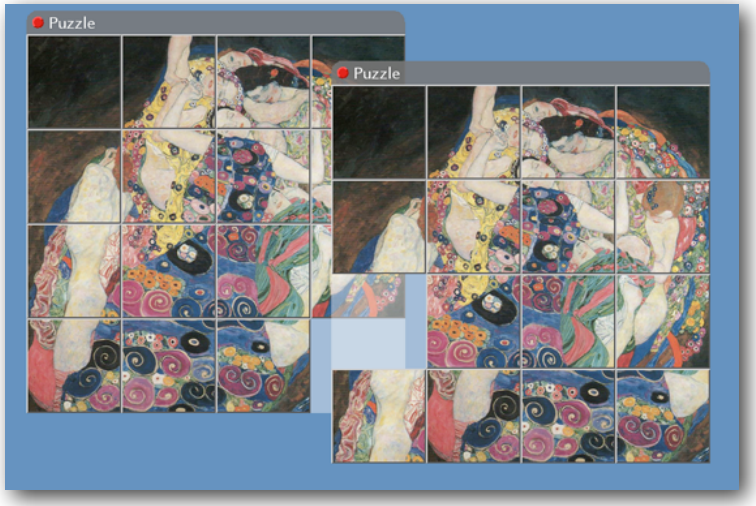
de haut en bas de l'image,
de gauche à droite d'une ligne.
Autorise un "parcours scanline"





Effet de transparence



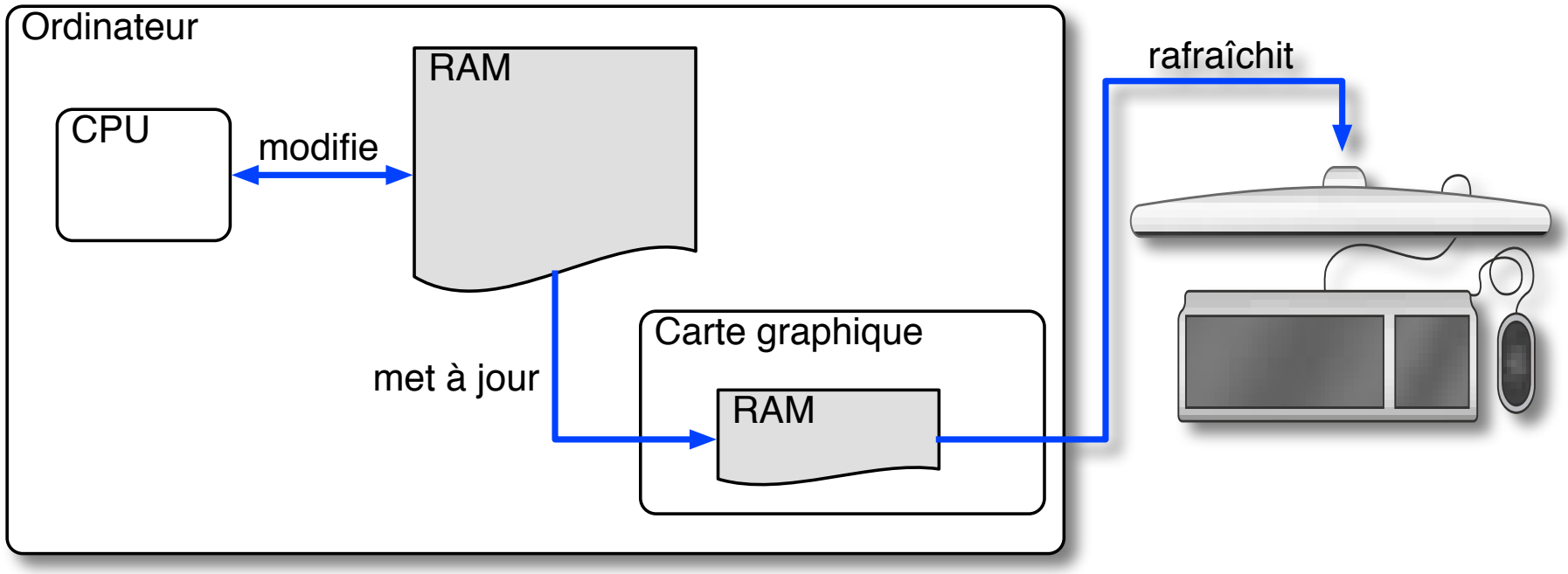


Dessin des "objets" du plus profond au plus proche

Le pixel résultat est une combinaison du pixel présent et du nouveau pixel à dessiner, en utilisant l'α du nouveau pixel :

$$D_R = (D_R.(255 - S_\alpha) + S_R.S_\alpha)/255$$
$$D_G = ...$$
$$D_B = ...$$

Architecture Matérielle



Le programme n'agit pas directement sur la RAM de la ***carte graphique*** (i.e. pas sur l'écran).

Les mises à jour de l'image doivent être copiées sur la carte.

Fonctions fournies dans "libeibase.a"

Fonctions déclarées dans "hw_interface.h" (1/2)

```
ei_surface_t hw_create_window    (ei_size_t* size, const ei_bool_t fullScreen);
ei_surface_t hw_surface_create   (const ei_surface_t root,
                                  const ei_size_t*   size,
                                  ei_bool_t          force_alpha);
void hw_surface_free(ei_surface_t surface);

void hw_surface_lock(ei_surface_t surface);
void hw_surface_unlock(ei_surface_t surface);

uint8_t* hw_surface_get_buffer(const ei_surface_t surface);
void hw_surface_get_channel_indices   (ei_surface_t surface,
                                       int* ir, int* ig, int* ib, int* ia);

void hw_surface_update_rects(ei_surface_t surface,
                             const ei_linked_rect_t* rects);
```

Fonctions fournies dans "libeibase.a"

Fonctions déclarées dans "hw_interface.h" (2/2)

```
ei_surface_t hw_text_create_surface   (const char*          text,
                                       const ei_font_t      font,
                                       const ei_color_t*    color);

ei_surface_t hw_image_load  (const char*            filename,
                             ei_surface_t           channels);

void hw_event_wait_next(struct ei_event_t* event);
double hw_now();
```

Fonctions à réaliser

Fonctions déclarées dans "ei_draw.h" (1/2)

```
typedef struct {
    unsigned char           red;
    unsigned char           green;
    unsigned char           blue;
    unsigned char           alpha;
} ei_color_t;

uint32_t ei_map_rgba    (ei_surface_t surface, const ei_color_t* color);

void ei_draw_polyline   (ei_surface_t                surface,
                         const ei_linked_point_t*    first_point,
                         const ei_color_t            color,
                         const ei_rect_t*            clipper);

void ei_draw_polygon    (...);
```

Fonctions à réaliser

Fonctions déclarées dans "ei_draw.h" (2/2)

```
void ei_fill            (ei_surface_t               surface,
                         const ei_color_t*          color,
                         const ei_rect_t*           clipper);

void ei_draw_text       (ei_surface_t               surface,
                         const ei_point_t*          where,
                         const char*                text,
                         const ei_font_t            font,
                         const ei_color_t*          color,
                         const ei_rect_t*           clipper);

int ei_copy_surface     (ei_surface_t               destination,
                         const ei_rect_t*           dst_rect,
                         const ei_surface_t         source,
                         const ei_rect_t*           src_rect,
                         const ei_bool_t            alpha);
```

Cycle de mise à jour de l'écran

Une "surface" est bloquée

```
hw_surface_lock(my_surface);
```

Le programme modifie la surface

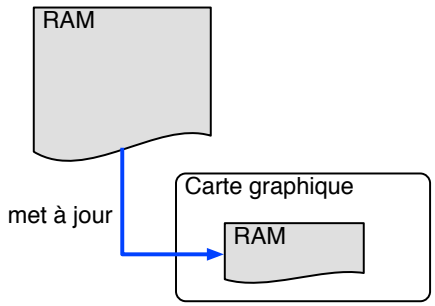
```
first_pixel     = hw_surface_get_buffer(my_surface);
*((uint32_t*)first_pixel)   = some_pixel_value;
ei_fill(my_surface, some_color, some_rectangle);
```

La surface est débloquée

```
hw_surface_unlock(my_surface);
```

Le programme demande la mise à jour de l'écran

```
hw_surface_update_rects(my_surface, the_rect_list);
```



Interface Utilisateur Graphique

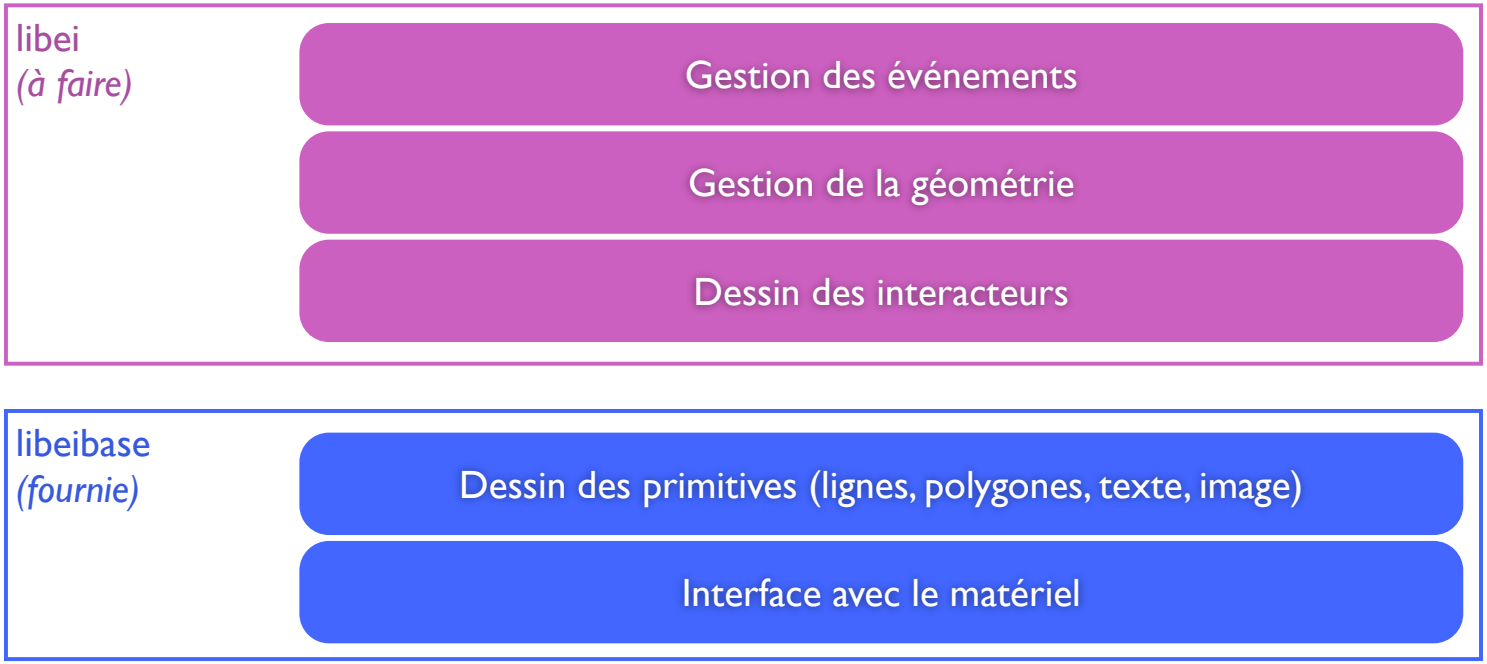
Services de la Bibliothèque

Survol

Création, configuration, et dessin des ***interacteurs*** (“***widgets***”)

Placement à l’écran (position et taille) : gestion de la ***géométrie***

Prise en compte des actions utilisateur : gestion des ***événements***

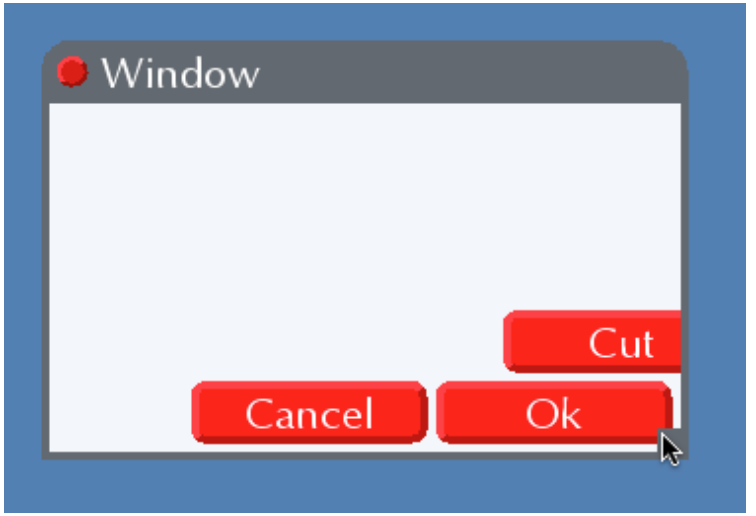


Gestion des interacteurs

Organisation Hiérarchique

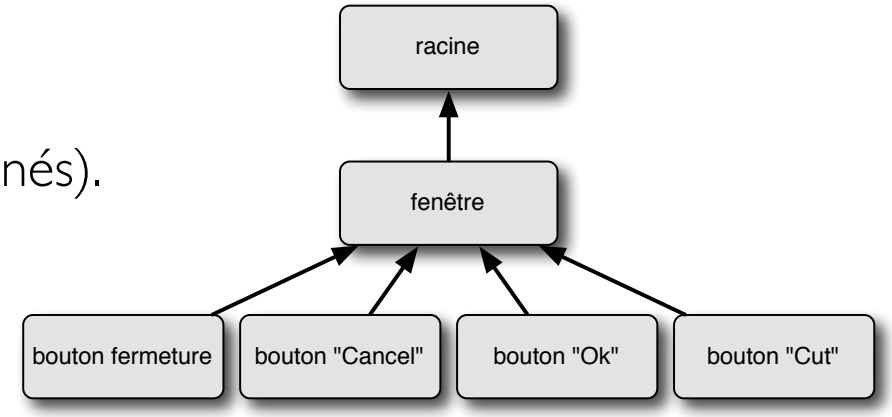
Tout interacteur :

- a un parent, hormis la ***racine***,
- est ***tronqué*** (“clipped”) dans les limites de son parent,
- est positionné par rapport à sont parent,
- est masqué avec son parent,
- est détruit avec son parent.



L’ordre de dessin est :

- en profondeur, puis,
- en largeur (les descendants sont ordonnés).



Classe d'interacteurs

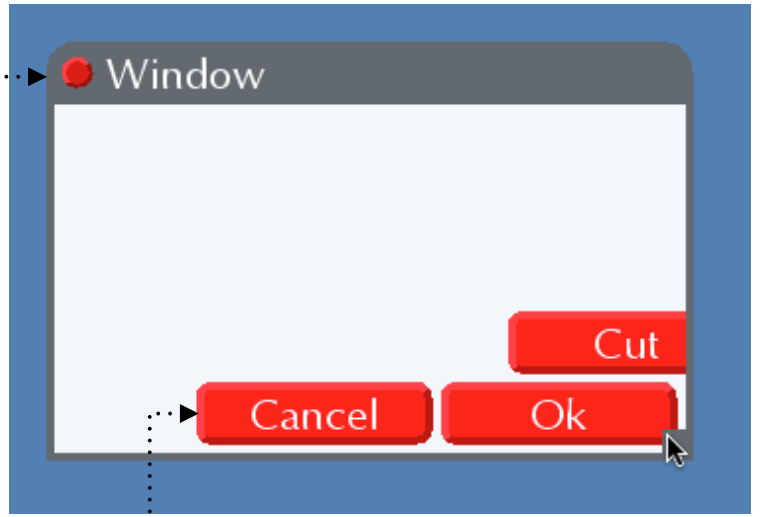
Principe

Tous les interacteurs partagent certaines caractéristiques *communes* (hiérarchie, géométrie, etc.)

Par contre, certaines caractéristiques sont *spécifiques* à une ***classe d'interacteur***.

Exemples :

- Fenêtre toplevel
- Boutons



Mais aussi : champ de saisie, barre de défilement, case à cocher, etc.

Classe d'interacteurs

Polymorphisme

Un ***bouton***, par exemple, doit pouvoir être vu :

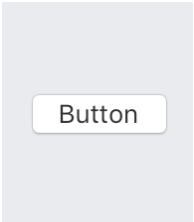
- comme un ***interacteur*** pour les traitements communs à tout interacteur (hiérarchie, etc.).
- ou comme un ***bouton*** pour les traitements spécifiques aux boutons (dessin, etc.),

➡ Nécessité d'un mécanisme de ***polymorphisme***.

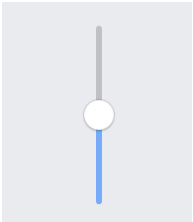
C'est le fondement de la ***programmation orienté objet***, qui fut inventée justement pour la programmation des interfaces graphiques.

Classe d'interacteurs

Polymorphisme




```
typedef struct ei_button_t {
     struct void*        desc_tete;

     struct void*        parent;
     struct void*        desc_suivant;

     const char*         label;
} ei_button_t;
```



```
typedef struct ei_scale_t {
     struct void*        desc_tete;

     struct void*        parent;
     struct void*        desc_suivant;

     ei_orient_t         orientation;
} ei_scale_t;
```

```
void ajout_descendant(void* parent, void* descendant)
{
     ajout_en_queue(parent->desc_tete, descendant)   /* deref. void* ???  */
     dessiner(descendant)                            /* dessin de bouton ? scale ? */
}
```

Programmation des classes d'interacteurs

Représentation des interacteurs en mémoire

Attributs communs à tout interacteur.

```
typedef struct ei_widget_t {
     ei_widgetclass_t*        wclass;

     struct ei_widget_t*      parent;
     struct ei_widget_t*      children_head;
     struct ei_widget_t*      children_tail;
     struct ei_widget_t*      next_sibling;

     ei_size_t                requested_size;
     ei_rect_t                screen_location;

} ei_widget_t;
```

Programmation des classes d'interacteurs

Représentation des interacteurs en mémoire

Ajout des attributs spécifiques à une classe donnée (ex: boutons).

```
typedef struct ei_widget_t {
     ei_widgetclass_t*        wclass;

     struct ei_widget_t*      parent;
     struct ei_widget_t*      children_head;
     struct ei_widget_t*      children_tail;
     struct ei_widget_t*      next_sibling;

     ei_size_t                requested_size;
     ei_rect_t                screen_location;

} ei_widget_t;
```

```
typedef struct {
     ei_widget_t         widget;

     int                 border_width;
     ei_relief_t         relief;
     char*               text;

} ei_button_widget_t;
```

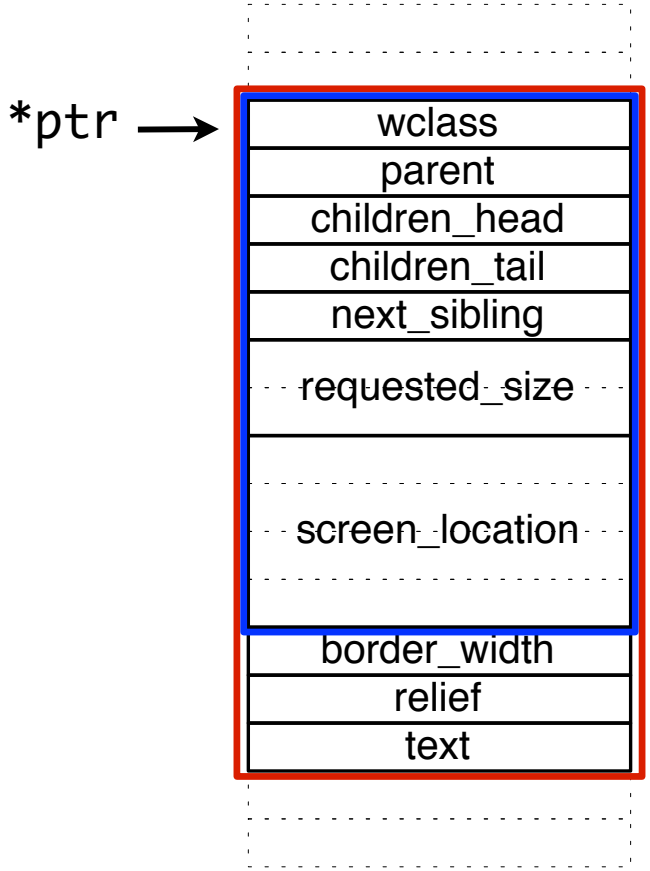
Programmation des classes d'interacteurs

Représentation des interacteurs en mémoire

Ajout des attributs spécifiques à une classe donnée (ex: boutons).

```
typedef struct ei_widget_t {
      ei_widgetclass_t*        wclass;

      struct ei_widget_t*      parent;
      struct ei_widget_t*      children_head;
      struct ei_widget_t*      children_tail;
      struct ei_widget_t*      next_sibling;

      ei_size_t                requested_size;
      ei_rect_t                screen_location;

} ei_widget_t;
```

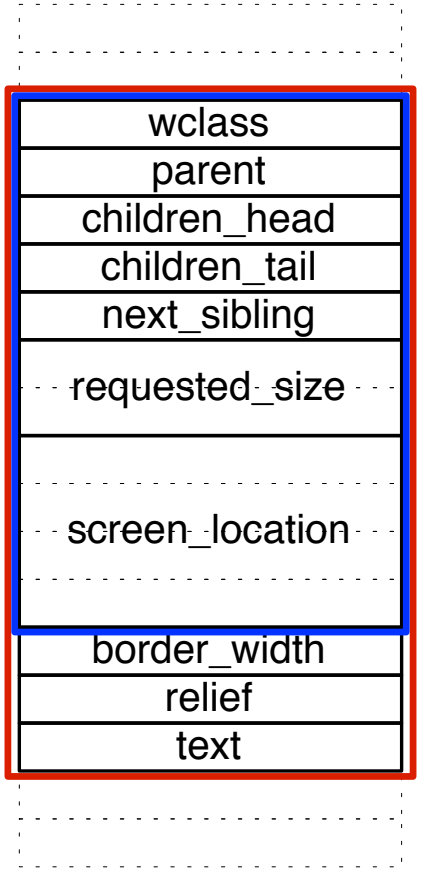
```
typedef struct {
      ei_widget_t              widget;

      int                      border_width;
      ei_relief_t              relief;
      char*                    text;

} ei_button_widget_t;
```



Programmation des classes d'interacteurs

Polymorphisme des données

```
ei_button_widget_t*     button;
button = malloc(sizeof(ei_button_widget_t));

void init_button(ei_widget_button_t* button, ei_widget_t* parent)
{
     init_widget((ei_widget_t*)button, g_button_class, parent);
     button->border_width    = 1;
     button->relief          = ei_relief_raised;
     button->text            = (char*)NULL;
}

void init_widget(ei_widget_t* widget, ei_widgetclass_t* wclass,
                                      ei_widget_t* parent)
{
     widget->wclass          = wclass;
     widget->parent          = parent;
     widget_add_child(parent, widget);
     widget->children_head   = (ei_widget_t*)NULL;
     ...
}
```

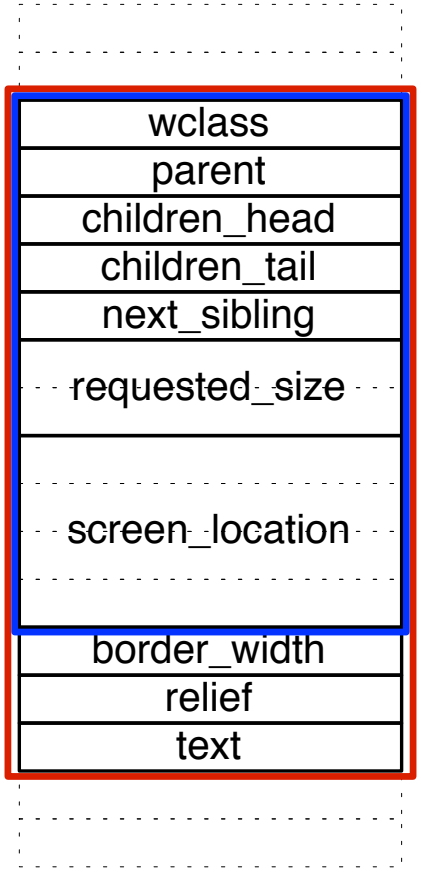


Programmation des classes d'interacteurs

Polymorphisme des traitements

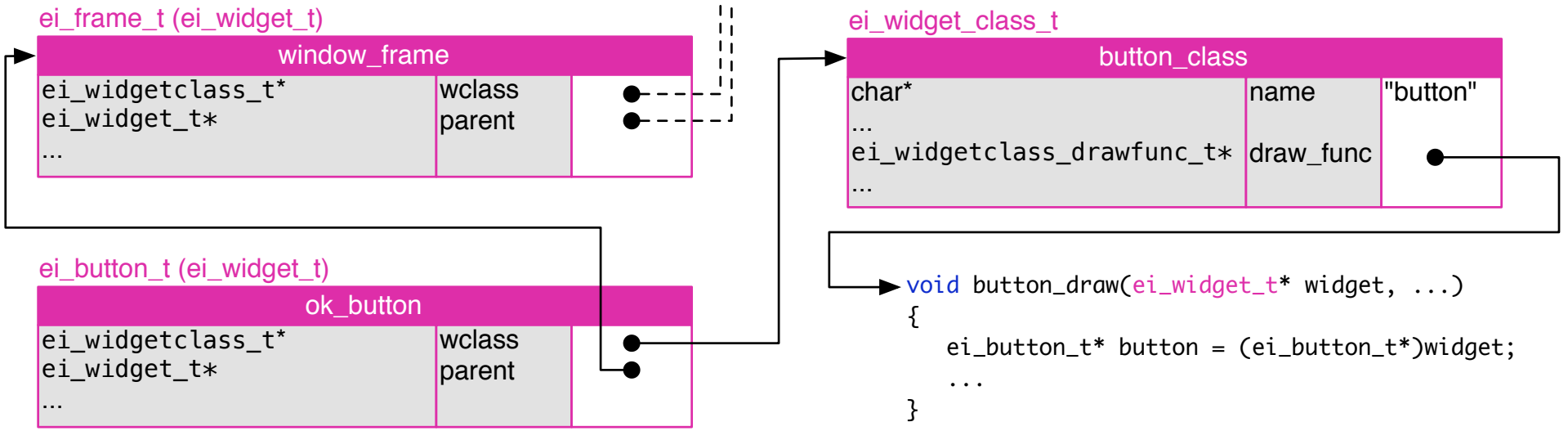
```
void widget_resize(ei_widget_t* widget, ei_size_t* new_size)
{
     widget->requested_size = new_size;
     ...
     widget_draw(widget);
}
```

Comment appeler la fonction de dessin qui correspond à la classe de l'interacteur ?



Programmation des classes d'interacteurs

Polymorphisme des traitements

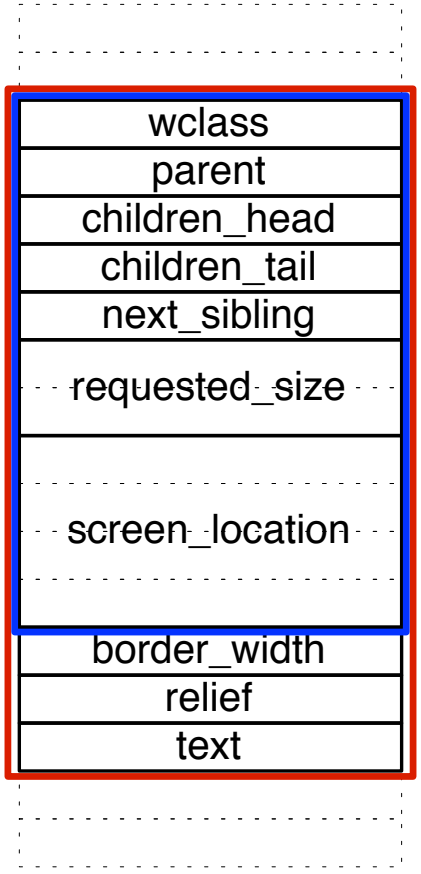



```
void widget_resize(ei_widget_t* widget, ei_size_t* new_size)
{
    ...
    widget->wclass->drawfunc(widget, ...);
}
```

Programmation des classes d'interacteurs

Polymorphisme des traitements

```
typedef void (*ei_widgetclass_drawfunc_t)
     (ei_widget_t* widget, ei_surface_t surface,
      ei_surface_t pick_surface, ei_rect_t* clipper);

typedef struct ei_widgetclass_t {
     ...
     ei_widgetclass_drawfunc_t       drawfunc;
     ...
} ei_widgetclass_t;

void button_draw (ei_widget_t* widget, ei_surface_t surface,
                  ei_surface_t pick_surface, ei_rect_t* clipper);

ei_widgetclass_t        button_class = { ..., button_draw, ...};
ei_button_widget_t      button       = { &button_class, ... };

void widget_resize(ei_widget_t* widget, ei_size_t* new_size)
{
     ...
     widget->wclass->drawfunc(widget);
}
```

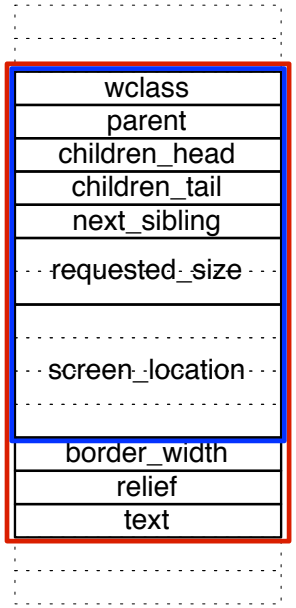


Programmation des classes d'interacteurs

Ajout d'une classe d'interacteur dans la bibliothèque

- Définition d'une structure qui étend `ei_widget_t` pour représenter les attributs spécifiques,
- définition de toutes les fonctions spécifiques,
- initialisation d'une instance de `ei_widgetclass_t`,
- enregistrement de la classe dans la bibliothèque par appel de `ei_widgetclass_register`.

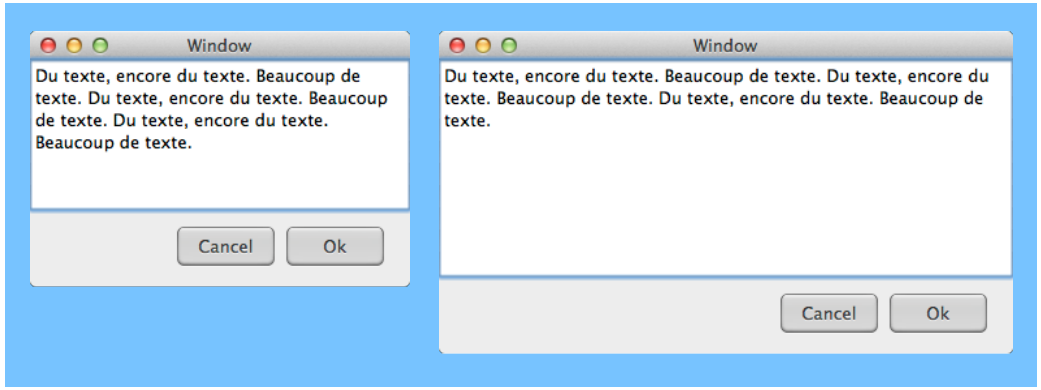
```
typedef struct ei_widgetclass_t {
    ei_widgetclass_name_t               name;
    ei_widgetclass_allocfunc_t          allocfunc;
    ei_widgetclass_releasefunc_t        releasefunc;
    ei_widgetclass_drawfunc_t           drawfunc;
    ei_widgetclass_setdefaultsfunc_t    setdefaultsfunc;
    ei_widgetclass_geomnotifyfunc_t     geomnotifyfunc;
    ei_widgetclass_handlefunc_t         handlefunc;
    struct ei_widgetclass_t*            next;
} ei_widgetclass_t;

void ei_widgetclass_register (ei_widgetclass_t* widgetclass);
```



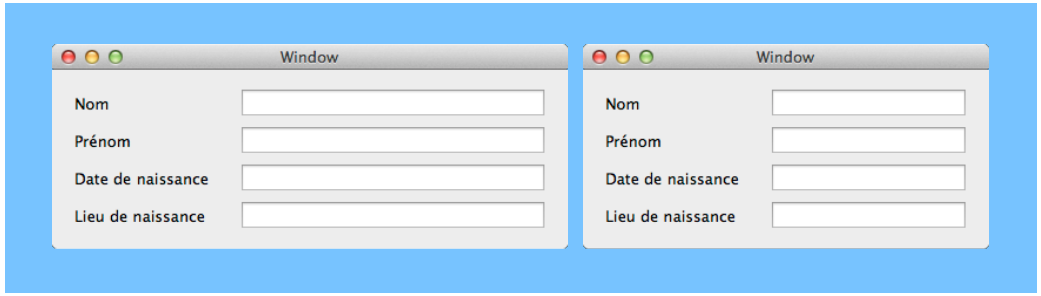
Gestion de la géométrie

Le problème

Position relative au parent



Position et taille relative au parent et aux autres descendants.



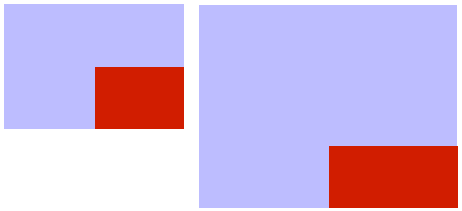
➡ Expression de contraintes pour la position et la taille des interacteurs, plutôt que des valeurs absolues.

Gestion de la géométrie

Différentes stratégies

Placeur

Contraintes par rapport au parent uniquement.
"Place l'interacteur dans l'angle en bas à droite, avec une hauteur de 100 pixels et la moitié de la largeur du parent".

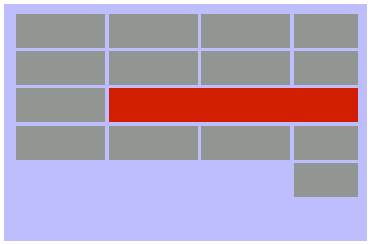


Packer

Contraintes par rapport au parent et aux descendants.
"Pack l'interacteur à droite des descendants déjà présents, en prenant toute la hauteur."

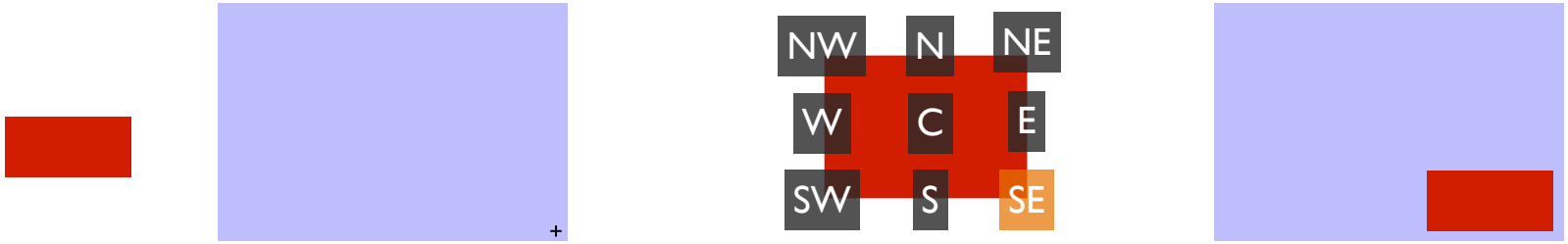
Gridder

Contraintes de grille.
"Grid l'interacteur en colonne 2, ligne 3, sur 3 colonnes."



Gestion de la géométrie

Interface de programmation du “placeur”




```
typedef enum {
    ei_anc_none,
    ei_anc_center, ei_anc_north, ei_anc_northeast, ei_anc_east,ei_anc_southeast,
    ei_anc_south, ei_anc_southwest, ei_anc_west, ei_anc_northwest
} ei_anchor_t;

void ei_place  (ei_widget_t*    widget,    ei_anchor_t*    anchor,
                int*            x,         int*            y,
                int*            width,     int*            height,
                float*          rel_x,     float*          rel_y,
                float*          rel_width, float*          rel_height);
```

Gestion de la géométrie

Interface de programmation du “placeur”

```
ei_anchor_t     anchor  = ei_anc_southeast;
int             x       = -10;
int             y       = -10;
float           rel_x   = 1.0;
float           rel_y   = 1.0;

ei_place(button, &anchor,   &x,       &y,     NULL, NULL,
                            &rel_x,   &rel_y, NULL, NULL);
```

Ok

Interface de programmation du "placeur"

Mise en oeuvre des valeurs par défaut

```
ei_place(button, &anchor,   &x,      &y,     NULL, NULL,
                            &rel_x,  &rel_y, NULL, NULL);
```

Un paramètre NULL signifie "valeur par défaut" :

- Lors du premier appel, la valeur par défaut est donnée dans les spécifications.
- Quand la valeur a déjà été définie lors d'un appel précédent, elle est conservée.

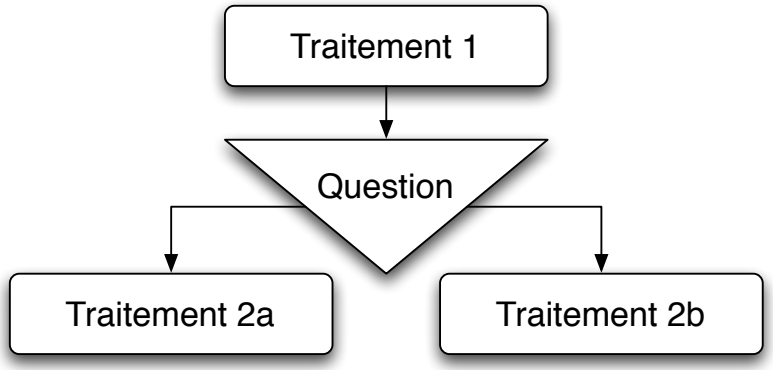
Le principe de valeur par défaut s'applique à la configuration des widgets

```
ei_color_t  background      = { 0x00, 0x00, 0xff, 0x88 };
void ei_button_configure(button, NULL, &background, NULL, ..., NULL);
```

Gestion des événements

Motivation

Programmation ***séquentielle***



Séquences
Répétitions
Branchements conditionnels

Le programme a le contrôle.
Le programme consulte les facteurs extérieurs à certains noeuds du graphe.

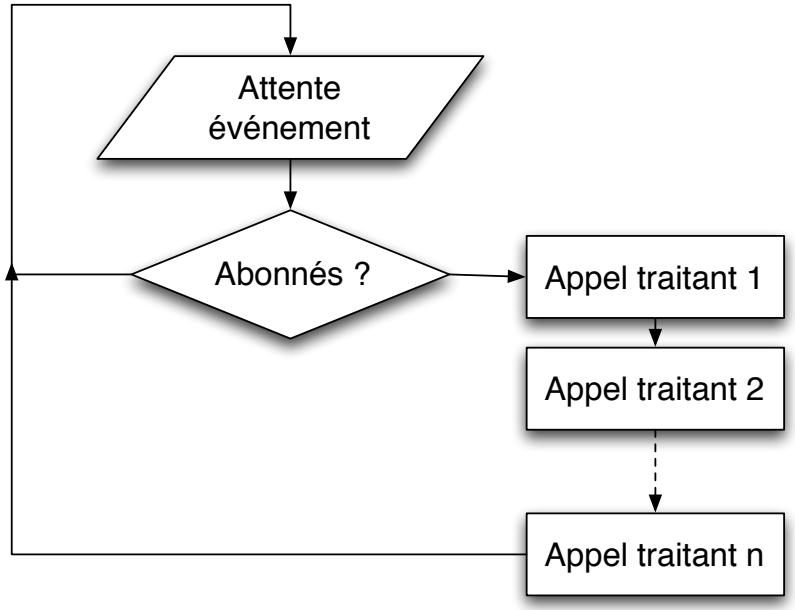
Cas des actions de l'utilisateur : à chaque étape, toute action est possible.

Gestion des événements

Motivation

Programmation ***événementielle***

L'utilisateur a le contrôle.

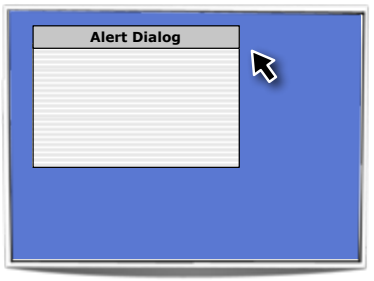


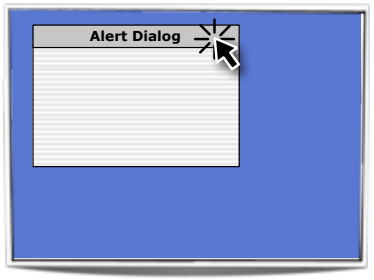
Le programmeur ***abonne*** des ***traitants*** à la réception d'événements.

Le programme principal se contente d'attendre un événement,
puis d'appeler les traitants abonnés.

Gestion des événements

Exemple : glisser-déposer





Initialisation

Abonnement d'une fonction "handle_button_press" à l'événement d'appui sur le bouton de la souris quand le pointeur est sur la barre de titre de la fenêtre.

Dans "handle_button_press"

Abonnement des fonctions "handle_motion" et "handle_button_release".

Dans "handle_motion"

Déplacement de la fenêtre.

Dans "handle_button_release"

Désabonnement de "handle_motion" et "handle_button_release".

Gestion des événements

Interface de programmation

```
typedef enum { ei_ev_none, ei_ev_app,
     ei_ev_keydown, ei_ev_keyup,
     ei_ev_mouse_buttondown, ei_ev_mouse_buttonup, ei_ev_mouse_move,
     ei_ev_last
} ei_eventtype_t;

typedef char* ei_tag_t;

typedef ei_bool_t(*ei_callback_t)   (ei_widget_t*         widget,
                                     struct ei_event_t*   event,
                                     void*                user_param);

void ei_bind (ei_eventtype_t    eventtype,
              ei_widget_t*      widget,
              ei_tag_t          tag,
              ei_callback_t     callback,
              void*             user_param);

void ei_unbind(ei_eventtype_t eventtype, ...);
```

Interface de programmation

Notion de tag

- Les tags (étiquettes) permettent des abonnements plus généraux que ceux liés aux widgets.
- Tout interacteur possède :
  - le tag "all" (permet de faire des abonnements sur tout widget),
  - le tag qui correspond à sa classe : "button", "frame", "toplevel", etc. (permet de faire des abonnements sur tout widget d'une classe).
- Un traitant lié à un tag connait l'interacteur qui a effectivement reçu l'événement : c'est son premier paramètre.
- En extension, vous pouvez proposer une API de gestion de tags d'un interacteur.

Interface de programmation

Paramètres d'événement

```
typedef struct {
     SDLKey                 key_sym;
     ei_modifier_mask_t     modifier_mask;
} ei_key_event_t;

typedef struct {
     ei_point_t             where;
     int                    button_number;
} ei_mouse_event_t;

typedef struct ei_event_t {
     ei_eventtype_t         type;
     union {
          ei_key_event_t    key;
          ei_mouse_event_t  mouse;
          ei_app_event_t    application;
     } param;
} ei_event_t;
```

Programmation de la gestion des événements

Programme principal

Le programmeur d'application :

- initialise l'interface graphique (création des widgets initiaux),
- enregistre ses traitants,
- lance la ***boucle principale*** (`ei_app_run()`).

Boucle principale

Le programmeur de la bibliothèque

- se met en attente d'un événement système (`hw_event_wait_next(&event)`),
- identifie le widget concerné,
- appelle les traitants concernés,
- met à jour l'écran,
- répète jusqu'à ce que le programmeur d'application appelle `ei_app_quit_request()`.

Programmation de la gestion des événements

Identification du widget concerné

Cas des événements clavier (`ei_ev_keydown`, `ei_ev_keyup`).

La bibliothèque doit gérer l'interacteur qui a le *focus clavier*.
Ce n'est pas demandé dans le projet, mais peut être réalisé en extension.

Cas des événement souris
(`ei_ev_mouse_buttondown`, `ei_ev_mouse_buttonup`, `ei_ev_mouse_move`)

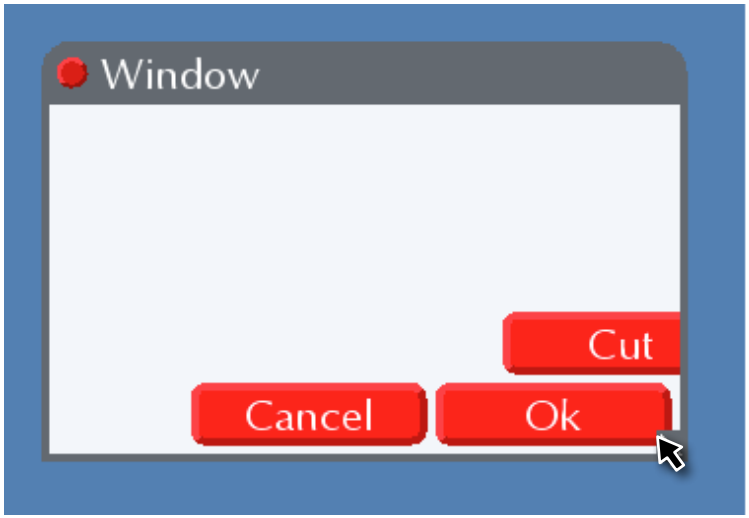
La bibliothèque doit pouvoir identifier l'interacteur *sous le pointeur* de la souris au moment de l'événement.
Ce service est appelé ***picking***.
Il y a différentes approches pour réaliser le picking.
Vous réaliserez un ***offscreen de picking***.

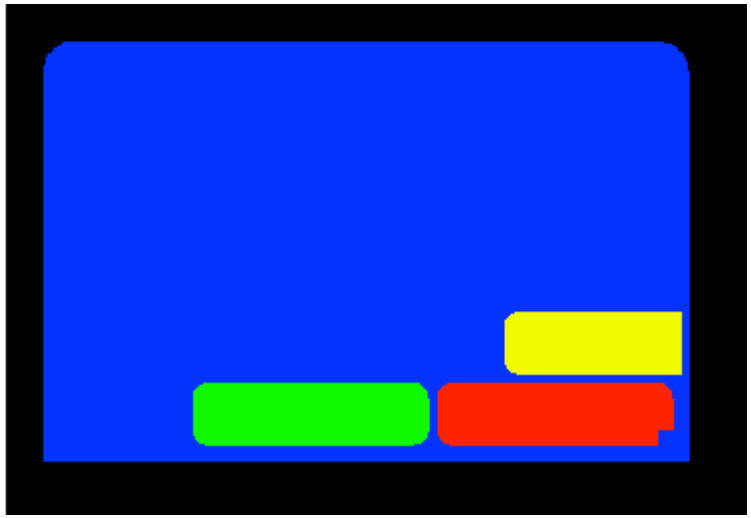
Réalisation d'un offscreen de picking

Offscreen : surface de dessin qui n'est pas affichée.

Principe

Pour toute mise à jour de l'écran, l'offscreen de picking est mis à jour à l'identique, si ce n'est que la "couleur" utilisée est l'identifiant de l'interacteur.





Le picking consiste simplement à lire l'identifiant dans l'offscreen de picking à la position du curseur.

Programmation de la gestion des événements

Réalisation d'un offscreen de picking

Attention à l'encodage des couleur, en particulier à la transparence.

```
typedef struct {
     unsigned char              red;
     unsigned char              green;
     unsigned char              blue;
     unsigned char              alpha;
} ei_color_t;

typedef struct ei_widget_t {
     ...
     uint32_t     pick_id;
     ei_color_t*  pick_color;
     ...
}

uint32_t ei_map_rgba  (ei_surface_t surface, const ei_color_t* color);

void ei_draw_polygone (ei_surface_t              surface,
                       const ei_linked_point_t*  first_point,
                       const ei_color_t          color,
                       const ei_rect_t*          clipper);
```

Mises à jour de l'écran

Principe

Les traitants ne font pas de mise à jour directement, ils *programment* la mise à jour.

```
void ei_app_invalidate_rect(ei_rect_t* rect);
```

La boucle principale, après avoir appelé les traitants, mets à jour l'écran sur tous les rectangles programmés.

Optimisation possible

- Pour ne pas dessiner deux fois les mêmes pixels,
- Pour minimiser le nombre de pixels à dessiner.

Mise à jour sur la carte graphique.

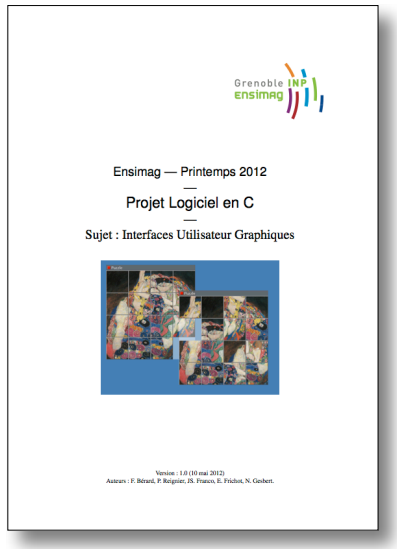
```
void hw_surface_update_rects(ei_surface_t surface,
                             const ei_linked_rect_t* rects);
```

Organisation du projet

Informations complémentaires

Documentation

Décrit tout ce qui vient d'être présenté, et plus encore, dans le détail.


Ensimag — Printemps 2012

—

Projet Logiciel en C

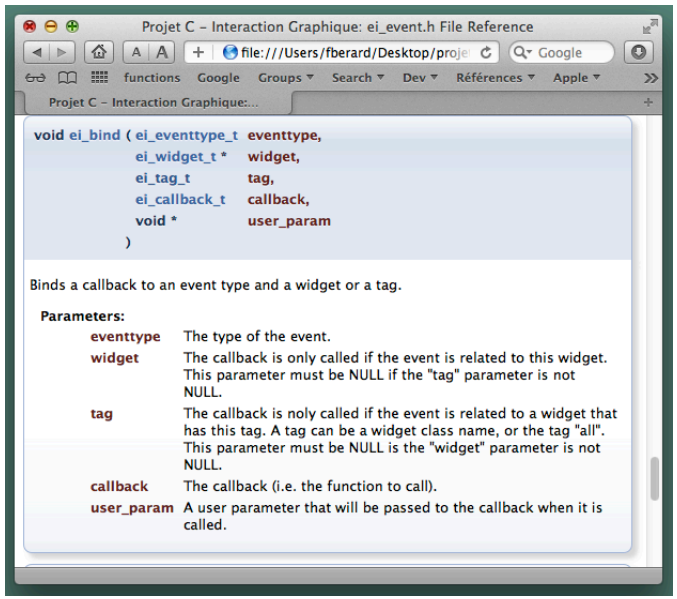
—

Sujet : Interfaces Utilisateur Graphiques

Commentaires du code

Utilisation de Doxygen : un système de génération de documentation à partir des commentaires.

```
make doc
open docs/html/index.html
```



Informations complémentaires

Les encadrants


Adrien Bullich (AB)


Denis Becker (DB)


Ignacio Merino (IM)


Charles Pelletier (CP)


Patrick Reignier (PR)


François Bérard (FB)

Les séances encadrées sont publiées sur ensiwiki.

Le site web du projet

Sur ensiwiki

http://ensiwiki.ensimag.fr/index.php/Nouvelles_du_projet_C_-_Interaction_Graphique

Informations complémentaires

Les créneaux encadrés

8h45 => 12h45

14h00 => 18h00

Salles ***E200*** & ***E201*** & ***E30x***

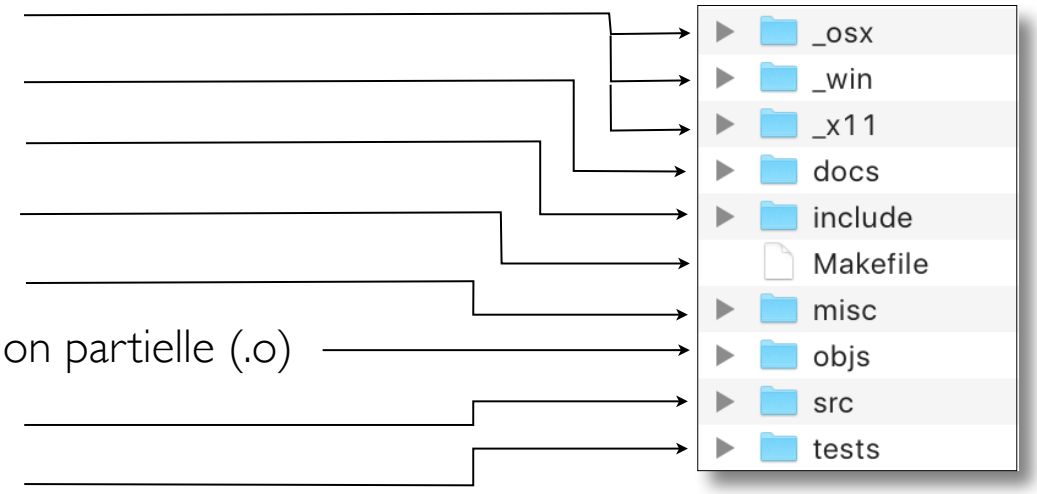
	Ven27		Lun30		Mar31		Mer1		Jeu2		Ven3		Lun6		Mar7		Mer8	
	am	pm	am	pm	am	pm	am	pm	am	pm	am	pm	am	pm	am	pm	am	pm
Francois																		
Patrick																		
Denis																		
Nacho																		
Adrien																		
Charles																		

Fichiers fournis

L'archive des fichiers

Téléchargeable depuis le site web du projet.

- Code compilé fourni (libeibase)
- Documentation doxygen
- Fichiers .h (API)
- Modèle de Makefile
- Fichiers annexes (fonte, image)
- Stockage des objets de compilation partielle (.o)
- Où mettre vos fichier source
- Exemples de code d'applications

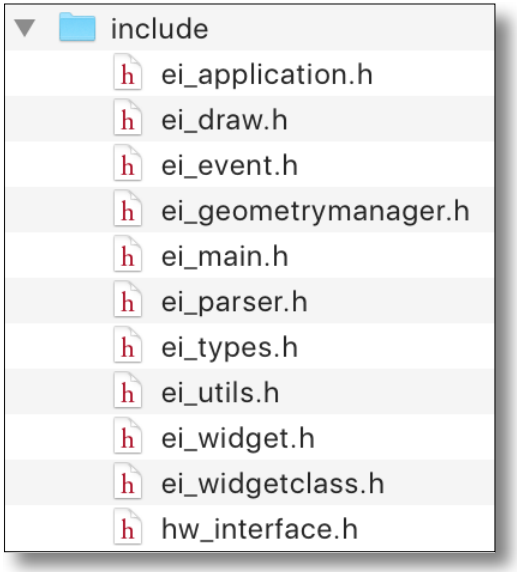


Fichiers fournis

L'archive des fichiers

Téléchargeable depuis le site web du projet.

Fichiers .h : interface de programmation de la bibliothèque



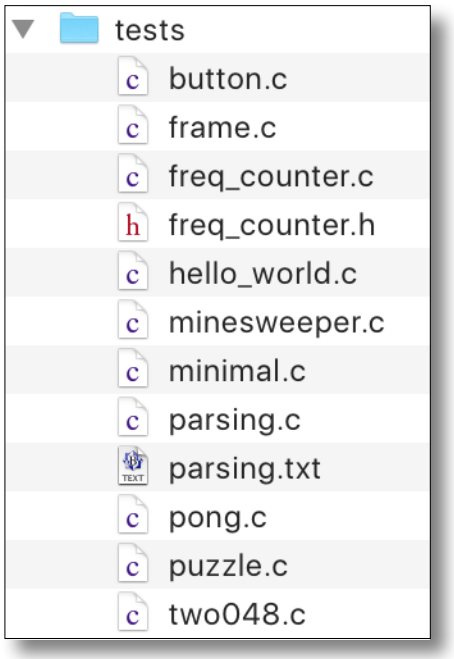
Interdiction absolue de modifier ces fichiers.
Placez vos déclarations dans d'autres fichiers.

Fichiers fournis

L'archive des fichiers

Téléchargeable depuis le site web du projet.

Exemples de code d'applications.



Travail sur les machines personnelles

Possible et encouragé (libeibase.a fournie pour Mac OS X, Linux, Windows), mais :

Les encadrants font du support uniquement pour les ***machines de l'Ensimag***.

L'évaluation se fera uniquement sur les ***machines de l'Ensimag***.

Si vous développez sur vos machines personnelles, testez ***très régulièrement*** que tout fonctionne à l'Ensimag.

Le projet nécessite la bibliothèque SDL et quelques dépendances.

http://ensiwiki.ensimag.fr/index.php/Projet_C_-_IG_-_Installation_de_SDL

Déroulement

Développement

Avant de vous lancer dans le code :

- lire la documentation,
- acquérir une compréhension globale du projet,
- la partager avec les membres du groupe,
- se répartir les tâches.

L'annexe A du document vous suggère les premières étapes de développement.

Déroulement

Extensions

Si vous avez complètement réalisé l'API spécifiée dans le répertoire "include", alors vous pouvez développer des extensions.

La section 4.2 du document propose un ensemble d'extensions (nouvelles classes d'interacteur, gestionnaire de géométrie en grille, gestion des tags).

Ce ne sont que des suggestions, vous pouvez proposer vos propres idées d'extensions : parlez-en aux encadrants.

Déroulement

Chronologie

Mercredi (8/6) au soir

vous rendez les fichiers de votre projet sur TEIDE

Jeudi (9/6) matin

vous préparez votre soutenance

Jeudi (9/6) après-midi et vendredi (10/6) matin

vous faites une soutenance devant un encadrant (1/2h, détail dans le document)

Évaluation

Critères d'évaluation

1. Exactitude : le projet fait ce qui est demandé.
2. Qualité de la structure de votre code (modules, fonctions).
3. Qualité de la forme du code (identificateurs, indentation, commentaires).
4. Extensions réalisées.