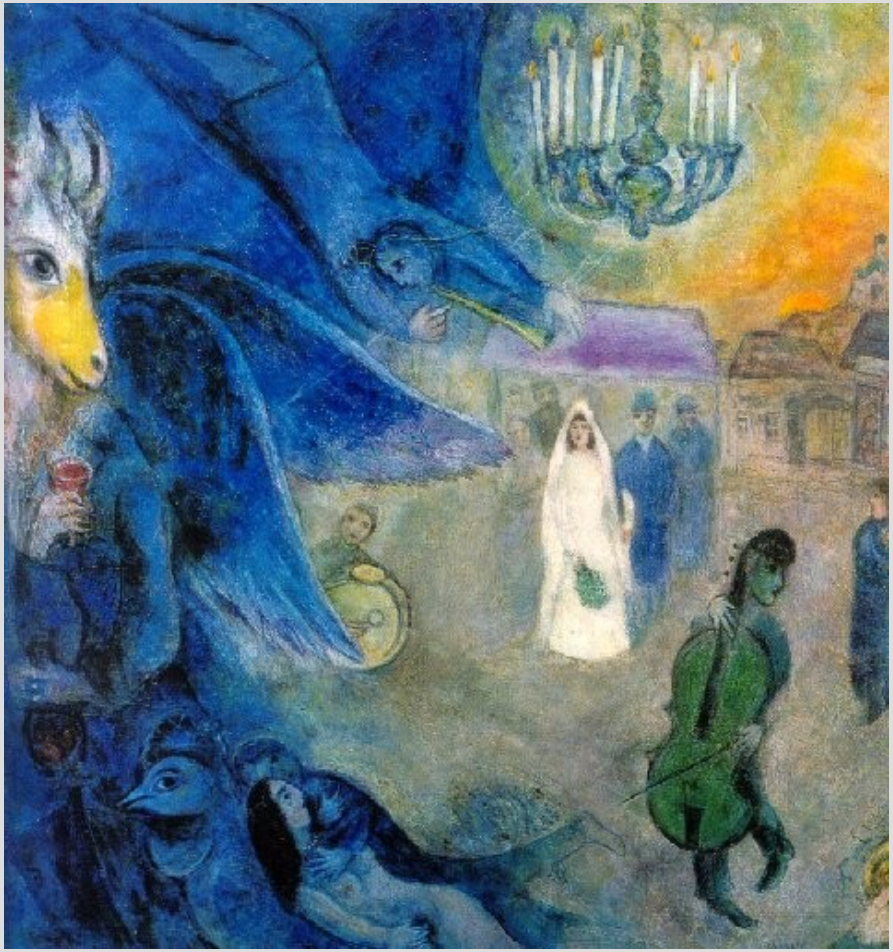
Parte 17

GUI – Parte terza



[M.Chagall – The Wedding Candles, 1945]

Immagini e figure geometriche

- La libreria **GTK+** utilizza a sua volta le funzioni di più basso livello delle librerie **cairo** e **GDK**
- Tra le altre cose, tali librerie forniscono anche funzioni per **disegnare figure geometriche, per caricare da file e manipolare immagini**
- A partire da GTK+ versione 2.8 (quindi compresa la GTK+ 3) è consigliato usare **cairo** invece di GDK

Cairo

- Libreria su cui si basa GTK+ 3 per renderizzare i suoi widget
- Fornisce API per grafica vettoriale, tramite primitive per il disegno in 2 dimensioni
- Scritta in C, grande portabilità: supporta output su molteplici “backend” (surface)
  - X Window System, Win32 GDI, Mac OS Quartz, BeOS API, PS/2, OpenGL
  - Local image buffer, file di tipo PNG, PDF, PS, SVG, etc.

Documentazione cairo e GDK

Manuale di riferimento GDK:

- Sul sito di GTK+ all'URL http://library.gnome.org/devel/gdk/stable/

Manuale di riferimento cairo:

- Sul sito di cairo all'URL http://cairographics.org/documentation/

Area di disegno

Nella libreria **GTK+** troviamo un **widget**, chiamato **GtkDrawingArea**, che non è altro che un'area vuota in cui è possibile:

- *disegnare figure geometriche*
- *disegnare immagini*

  per esempio utilizzando le funzioni di disegno della libreria **cairo**

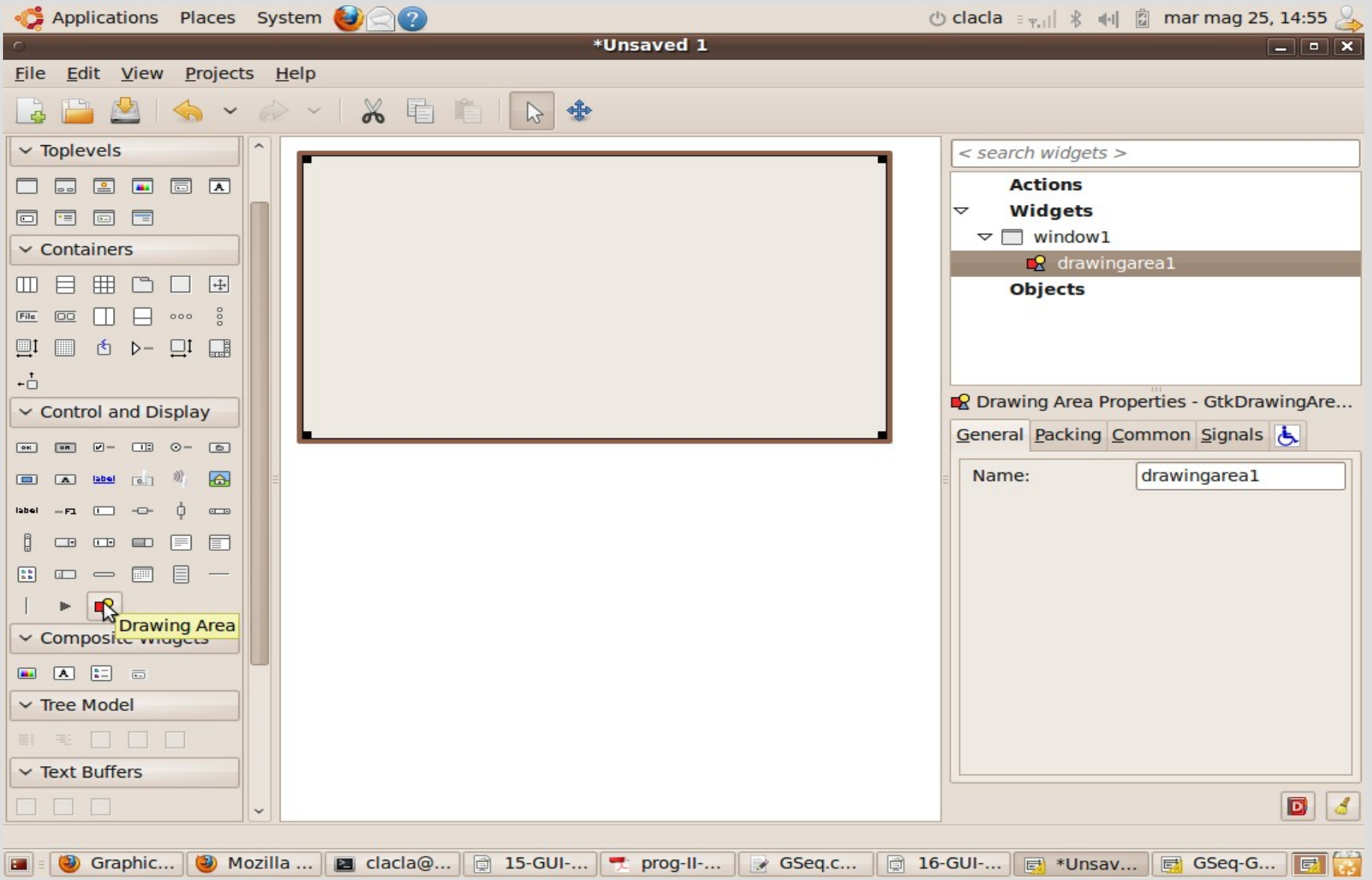
Cercate **GtkDrawingArea** sul manuale dei **Widget**:

https://developer.gnome.org/gtk3/3.8/GtkDrawingArea.html

Inserire una Drawing Area

- Per inserire una **Drawing Area** nel nostro progetto – per realizzare per esempio la finestra istogramma - occorre:
- Inserire prima un **widget GtkWindow**
  - Controllare che abbia proprietà *Visible=No*
- Riempirlo con un **widget Drawing Area,** come nella slide seguente

Inserire una Drawing Area



Gestione DrawingArea

- La **GtkDrawingArea** non è molto di più di un contenitore di una finestra del **server X**
- E' compito nostro
  - **disegnarne il contenuto** (a tal fine useremo le funzioni della libreria **cairo**)
  - **gestire ogni evento che la riguarda** (tramite funzioni di callback associate ai segnali emessi)

Disegno

- Il tipo **GtkDrawingArea** è derivato da **GtkWidget**, da cui eredita il *segnale* ***draw***
- ***draw*** è il segnale emesso tutte le volte che occorre (ri)disegnare il widget stesso
- Ad esempio:
  - la prima volta che appare
  - quando il **widget** (o parte di esso) torna ad essere visibile dopo essere stato coperto (in parte) da un altro **widget**

Segnale draw

- Handler di **draw**:

  *gboolean user_function (GtkWidget *widget,*
  *CairoContext *cr, gpointer user_data)*

- Oltre ai classici parametri, troviamo un **CairoContext *cr**
  - Vediamolo come un disegno associato al nostro widget (nella fattispecie, la GtkDrawingArea)

Cairo context

- Un cairo context **cr** è un oggetto della classe principale della libreria cairo
- Contiene le specifiche necessarie per disegnare un oggetto qualsiasi
  - Spessore delle linee, colore, superficie di sfondo, forme, etc.
- Attraverso l'handler di draw, è possibile modificare **cr** in modo da disegnare ciò che si vuole sul widget (DrawingArea) in oggetto

Tutorial

- Abbiamo tutti gli elementi concettuali per capire come si fa in pratica a **gestire eventi** per una **GtkDrawingArea**
- **Ora vediamo come disegnare figure geometriche tramite la libreria cairo**
- Teniamo come riferimento il **Cairo Tutorial**
  - http://cairographics.org/tutorial/

Cairo basics

- Il concetto di base di disegno in Cairo implica
  - la definizione di path invisibili (fase di preparazione del disegno)
  - la loro successiva instanziazione, tramite operazioni che li rendono visibili (fase di disegno vero e proprio)

Cairo's drawing model

- Il modello di disegno di Cairo è piuttosto semplice e si basa su **soggetti** e **azioni**
- I soggetti vengono manipolati al fine di preparare gli strumenti di disegno (il cairo context)
- Le azioni servono per creare il disegno vero e proprio attraverso tali strumenti

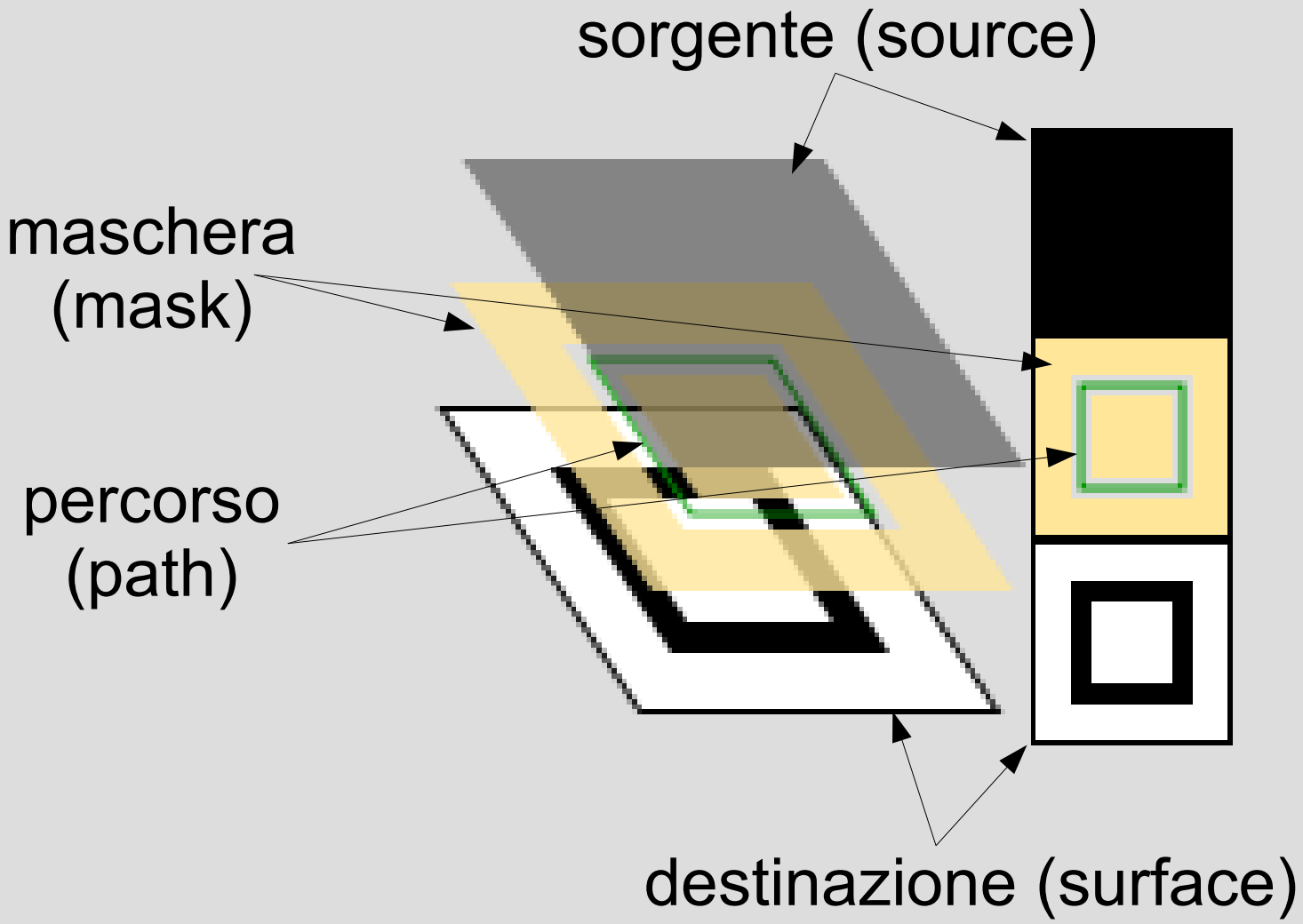
Soggetti e azioni

- Soggetti:
  - **sorgente (source)**
  - **destinazione (surface)**
  - **maschera (mask)**
  - **percorso (path)**

  } **contesto (context)**

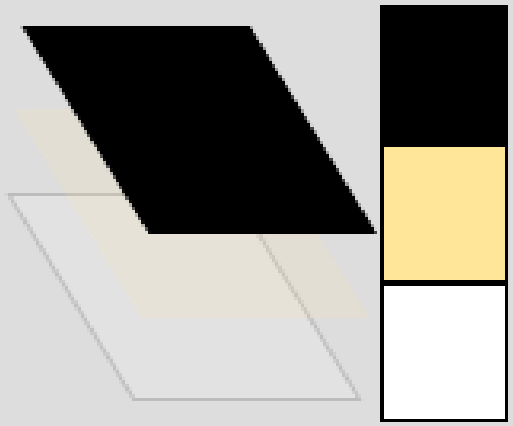
- Azioni:
  - **stroke, fill, text, paint, mask**

Contesto


sorgente (source)
maschera
(mask)
percorso
(path)
destinazione (surface)

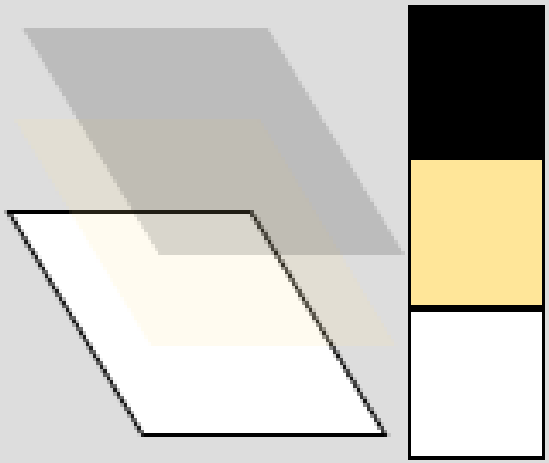
Sorgente (source)

- E' il “pennello” con cui lavoriamo
- Ma può essere di vari colori oppure un'immagine precedentemente creata



Destinazione (surface)

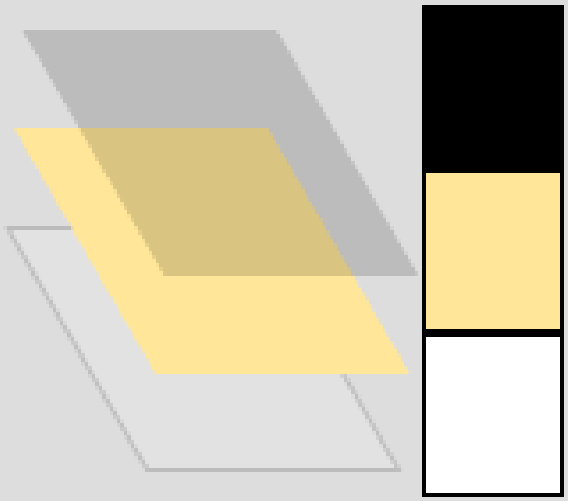
- E' il “foglio” su cui lavoriamo
  - Può corrispondere ad un array di pixel (ad esempio l'area associata alla nostra GtkDrawingArea) o ad un file PNG, SVG, etc.



  - Viene scritto/disegnato tramite le azioni

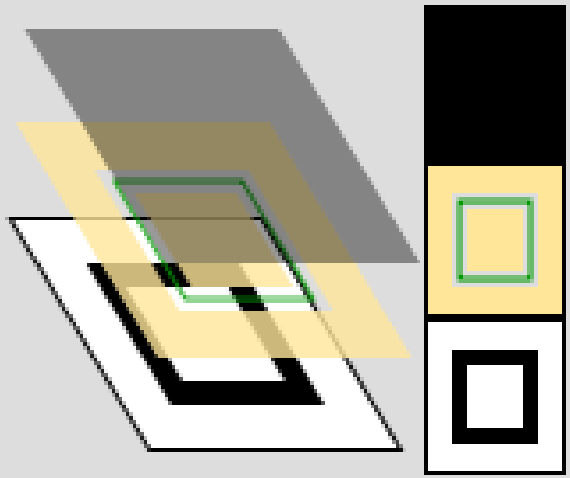
Maschera (mask)

- Controlla i punti in cui la source viene applicata sulla surface
- Viene incisa creando il path



Percorso (path)

- E' l'incisione sulla maschera
  - Specifica quali parti di source saranno impresse su surface
  - L'incisione avviene tramite “path verbs”



Contesto (context)

- Composto da
  - una source, una mask (e relativo path) e una surface
  - diverse variabili di stile (linea, font, etc.)
- Inizialmente occorre sempre creare un contesto
- Il contesto è sempre collegato ad una surface (già in fase di creazione)

Creazione di un contesto

- Il contesto è di tipo **cairo_t**
- La surface di tipo **cairo_surface_t**

```
cairo_surface_t *surface;
cairo_t *cr;

surface = cairo_image_surface_create
        (CAIRO_FORMAT_ARGB32, 120, 120);
cr = cairo_create (surface);
```

- Esistono vari tipi di surface, una per ogni formato supportato da cairo

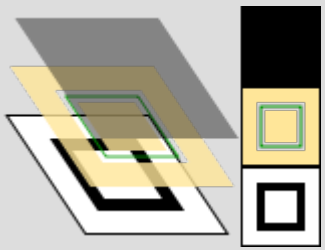
[nell'esempio: surface da 120 x 120 pixel, con 32 bits/pixel, e informazioni di colore RGB e Alpha]

Azioni (verbs)

- Provocano l'impressione della source sulla surface, dove permesso dalla mask
- Variano nella costruzione della mask

Stroke

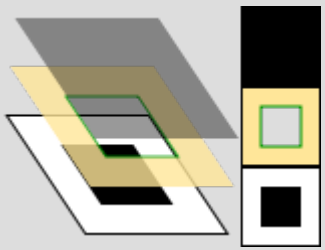
- Incide la maschera con una linea che segue il path
- Il tipo di linea (spessore, tratteggio, end point) dipende dal contesto associato
- Operazione **cairo_stroke(cr)**
- Es:



```
cairo_set_line_width (cr, 0.1);
cairo_set_source_rgb (cr, 0, 0, 0);
cairo_rectangle (cr, 0.25, 0.25, 0.5, 0.5);
cairo_stroke (cr);
```

Fill

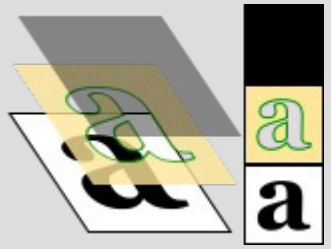
- Taglia l'area della maschera delimitata dal path (con apposite regole per path complessi)
- Consente di disegnare facilmente figure a riempimento pieno
- Operazione **cairo_fill(cr)**
- Es:



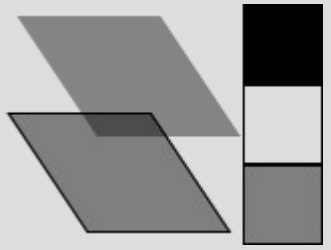
```
cairo_set_source_rgb (cr, 0, 0, 0);
cairo_rectangle (cr, 0.25, 0.25, 0.5, 0.5);
cairo_fill (cr);
```

Show text, paint, mask

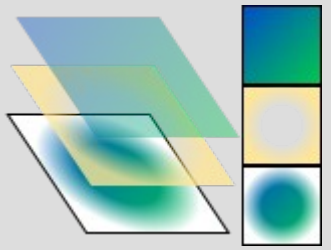
- **cairo_show_text(cr,”text”)** consente di tagliare la maschera formando lettere



- **cairo_paint(cr)** trasferisce tutta la source sulla surface (è possibile specificare la trasparenza con alpha)



- **cairo_mask(cr,pattern)** trasferisce la source sulla surface secondo trasparenza/opacità di un pattern



Selezione della sorgente

- Esistono tre tipi di sorgenti
  - **Colori**: la source assume un colore uniforme (eventualmente con grado di opacità alpha)
  - **Gradienti**: permette di definire una progressione di colore per la sorgente
  - **Immagini**: carica nell sorgente un'immagine esistente (es. da un file)

Color source

- `cairo_set_source_rgba (cr, r, g, b, alpha);`
  - Setta la sorgente del colore indicato da r,g,b con grado di opacità alpha
- `cairo_set_source_rgb (cr, r, g, b);`
  - Equivalente al precedente con `alpha=1.0`

Creazione del path: spostamenti

- Nella creazione di un path, le operazioni vengono sempre eseguite a partire dal punto in cui era arrivata la precedente
- Per saltare ad un altro punto si può usare

```
cairo_move_to(cr, x, y);
```

  - Passa al punto di coordinate (x,y)

```
cairo_rel_move_to(cr, dx, dy);
```

  - Passa al punto distante (dx,dy) da quello corrente

Creazione del path: linee

- Per tracciare una linea nel path si usa

```
cairo_line_to(cr, x, y);
```

  - Traccia una linea dal punto corrente al punto di coordinate (x,y)

```
cairo_rel_line_to(cr, dx, dy);
```

  - Traccia una linea dal punto corrente al punto distante (dx,dy) da quello corrente

Creazione del path: rettangoli

- Per tracciare il contorno di un rettangolo nel path si usa

```
cairo_rectangle(cr,x,y,largezza,altezza);
```

  - Traccia un rettangolo avente l'angolo in alto a sinistra di coordinate (x,y), e altezza/larghezza specificate

Creazione del path: archi

- Per tracciare una linea nel path si usa

```
cairo_arc(cr,xc,yc,raggio,a1,a2);
```

  - Traccia un arco di circonferenza avente centro (xc,yc), raggio specificato, compreso tra gli angoli a1 e a2 (senso orario)

```
cairo_arc_negative(cr,xc,yc,raggio,a1,a2);
```

  - Come sopra, ma in senso antiorario

- Se l'arco è fuori dal path, viene aggiunta una linea dall'ultimo punto all'inizio dell'arco

Creazione del path: curve

- Le curve sono modellate come “cubic Bézier splines”

```
cairo_curve_to(cr,x1,y1,x2,y2,x3,y3);
```

  - Traccia una curva dal punto corrente a (x3,y3), usando (x1,y1) e (x2,y2) come punti di controllo

```
cairo_rel_curve_to(cr,x1,y1,x2,y2,x3,y3);
```

  - Come sopra, ma in coordinate relative al punto corrente

Creazione del path: chiusura

- E' possibile chiudere il path, attraverso una linea che connette il punto corrente all'inizio del (sotto-)path

```
cairo_close_path(cr);
```

Creazione del path: testo

- Per inserire testo in un path, si può usare

  `cairo_text_path(cr,”testo a piacere”);`

  (eventualmente seguito da `cairo_fill(cr);`)
- Il testo viene aggiunto nella posizione corrente
- Per stringhe lunghe, meglio usare `cairo_show_text(cr,”testo a piacere”);`
- Esistono varie funzioni per settare il tipo di testo desiderato (vedere il manuale)

Trasformazioni

- Permettono di lavorare in coordinate più intuitive, scalare un immagine, deformarla, etc.
- Esempi:
  - `cairo_scale(cr, sx, sy);`

    permette di scalare tutte le operazioni di (sx,sy)
  - `cairo_translate(cr, tx, ty);`

    aggiunge un offset di (tx,ty) a tutte le operazioni

Salvataggio

- Per salvare una surface su un file, si può usare:

```
cairo_surface_write_to_png(surface,"file.png");
```

- E' possibile poi visualizzare il file con qualsiasi visualizzatore di immagini

Deallocazione

- Per deallocare un cairo context:

```
cairo_destroy(cr);
```

- Per deallocare una surface:

```
cairo_surface_destroy(surface);
```

Compilazione

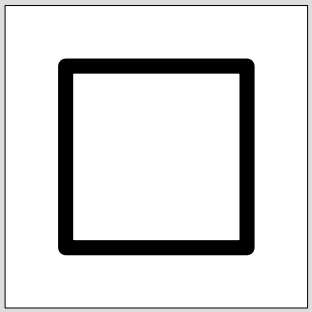
- Per compilare un programma che utilizza la libreria cairo:

```
g++ stroke.cc $(pkg-config --cflags --libs cairo)
```

- In seguito, nell'utilizzo di cairo con GTK+, tale operazione non sarà necessaria
  - generalizzata dall'inclusione/linking delle GTK+ (che a loro volta utilizzano cairo)

Esercizio 1

- Disegnare il contorno di un quadrato nero come in figura:



- Soluzione in *cairo/stroke.cc*

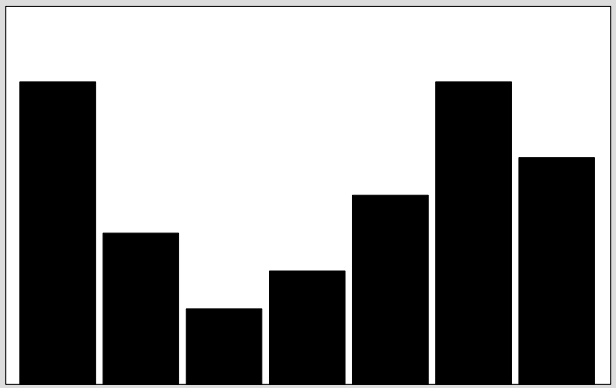
Esercizio 2

- Disegnare il contorno di una figura irregolare a piacere
- Es:

- Soluzione in *cairo/closepath.cc*

Istogramma

- Ora possediamo le conoscenze necessarie per realizzare un istogramma per il nostro gestore di sequenze utilizzando le funzioni di **cairo** e il widget **GtkDrawingArea**
- Creare un istogramma che visualizzi il valore delle chiavi degli elementi della sequenza



Specifiche istogramma

- Chiavi maggiori corrispondono a barre più alte
- L'istogramma deve occupare **tutta l'area disponibile** della finestra
  - La largezza delle barre deve scalare opportunamente, affinchè tutte le barre siano contenute nella finestra, e non avanzi spazio
  - L'altezza della barra più alta deve corrispondere all'altezza massima della finestra
  - Le altre barre devono scalare relativamente alla altezza massima
  - Prevedere opportuni spazi tra le barre

Dimensioni di un widget

- L'istogramma deve **ridimensionarsi correttamente** quando si ridimensiona la finestra che lo contiene
- Funzioni per accedere alle dimensioni di un widget:

```
int gtk_widget_get_allocated_width
                        (GtkWidget *widget);
int gtk_widget_get_allocated_height
                        (GtkWidget *widget);
```

- Soluzione in *progetto_GUI_istogramma*

Completamento GUI

- Ora possediamo tutti gli elementi necessari per completare l'interfaccia grafica del nostro programma di gestione delle sequenze