gEvas: a GTK+2 és az Evas összekapcsolása

Ismerjük meg az Enlightenment néhány alapkönyvtárát és azok alkalmazását gEvas-szel, nagy teljesítményű GTK+2 alkalmazások készítéséhez.

Az ***Evas*** olyan függvénykönyvtár, amellyel a rasztergrafikák gyors megjelenítése megvalósítható, az áttetszőség támogatásával. Az ***Evas*** az ***Enlightenment*** alapkönyvtárainak (***Enlightenment Foundation Libraries – EFL***) egyike, valójában azonban függvénykönyvtárak gyűjteménye, melyeket eredetileg az ***Enlightenment DR 17*** támogatására készítettek.
Az ***Edje*** és az ***Embryo*** kombinációja, az ***Emotion*** további olyan ***EFL***-könyvtárak, amelyek az ***Evas***-t kiegészítik. Az ***Edje***-vel betűkészletek, grafikák és funkciók csomagolhatók témafájlokba. Az ***Embryo*** egy egyszerű, ugyanakkor ***Turing***-teljes szkriptnyelv, amivel egyszerű parancsokat ágyazhatunk az ***Edje***-fájlokba. Az ***Emotionnel*** egyidejűleg több videófolyamot használhatunk elsőosztályú keretobjektumként, ami annyit jelent, hogy mozgathatjuk, átméretezhetjük, rétegezhetjük vagy akár áttetszővé tehetjük ezeket.
A ***gEvas*** könyvtár lehetővé teszi, hogy az ***Evast*** felhasználhassuk ***GTK+2.x*** alkalmazásokban. Kedvcsinálónak elmesélem, hogy engem leginkább az egyszerű ***API*** és az áttetszőséggel biztosított gyors képalkotás vezetett az ***Evas*** használatáig, és végül a ***gEvas*** megírásáig.
Ahhoz, hogy a képalkotás sebességéről ilyen kijelentéseket tehessünk, egy kis kirándulást kell tennünk a benchmarking világába. Az ***Evas*** disztribúcióban található ***evas_bench*** alkalmazás sok kép- és szövegelemet tartalmaz, azok átméretezésének és elegyítésének a lehetőségével. A képátméretezés eredménye pl. annyira kiváló, hogy párja sem akad. Átírtam az ***evas_bench***-et, hogy ***GNOME Canvast*** használjon.

Az *1. ábrán* látható a ***GNOME Canvasra*** átírt ***evas_bench*** képernyőképe. Készítettem néhány egyszerűbb tesztet is a keretátméretezés algoritmusához, áttetsző és nem áttetsző keretekkel egyaránt. Utóbbi esetben egy levél képét méretezzük át ciklusban a keretnél nagyobb méretről 0x0-ra, majd vissza, a *2. ábrán* látható módon. Az áttetszőségnél egy, a levél képének méretével megegyező piros téglalapot használtam, ahol az alfa értéke 0 volt a bal felső sarokban és maximális az alsón. Ahogy már említettem, a ***gEvast*** jómagam írtam, és néhány olvasónak talán az is feltűnik, hogy az ***Enlightenment*** projekt fejlesztői csapatának tagja vagyok. Bár mindez igaz, mindent elkövettem annak érdekében, hogy a benchmark elfogulatlan eredményt adjon. A benchmark forráskódja letölthető (lásd az online anyagokat). Azoktól, akik belenéznek, előre is elnézést kérek a sebtében végzett kódolás közben gyakran alkalmazott, kevéssé optimális megoldások miatt.
Azok kedvéért, akik a ***GNOME Canvast*** kevéssé ismerik, elmondom, hogy két képalkotó back-endje létezik. A ***GNOME Canvas*** fejlesztői dokumentációja szerint (lásd a kapcsolódó anyagokat): A ***GNOME Canvas***-ben két képalkotó back-end közül lehet választani. Az egyik ***Xlib*** alapú és rendkívül gyors megjelenítést biztosít, a másik ***Libart*** alapú, kifinomult, élfinomítást és áttetszőséget támogató motorral. Az ***Evas*** megpróbálja a két világ előnyeit egyszerre nyújtani. Mindkettővel összehasonlítottam az ***Evast***.

1. ábra GNOME Canvas-ra átírt evas_bench alkalmazás. A keretben sok kép- és szövegelem mozog és átméreteződik.

Egyes esetekben természetesen az ***Evas*** is lehet rossz választás.
A ***GNOME Canvas***-ben találunk egy ***Bezier***-görbéket támogató elemet is, míg az ***Evas***-ban ilyen pillanatnyilag nem létezik. Ráadásul az is kevésbé valószínű, hogy az ***Evast*** és a ***gEvast*** előtelepítve találjuk, mint a ***GNOME Canvas*** esetében.
A ***Bezier***-görbék támogatása a jövőben megvalósulhat az ***Evasban*** is. Ezzel kapcsolatban ***Raster*** (becses nevén ***Carsten Haitzler***) a következő sommás

véleményt fogalmazta meg: *„Ha valaki vektorgrafikus szerkesztőt vagy valami hasonlót keres, használja a GNOME Canvast, mivel az abban jobb. Ha viszont bárhol, áttetszőséget is támogató, valósidejű megjelenítésre van szükség, az Evas a nyerő.”*
Az *Evas* önmagában is számos képalkotó back-endet támogat, köztük a *framebuffert*, az *XLib*-et és az *OpenGL*-t. A *gEvas* ezek közül egyelőre csak az *XLib*-et használja. Mivel a *GTK+2 framebufferen* is fut, a *gEvas*-nak is kellene, ezt azonban eddig még nem próbáltam.
A *GNOME Canvas* és az *Evas* adatmodellje hasonló. A *Qt QCanvas* adatmodellje azonban már eléggé különbözik ahhoz, hogy megnehezítse a benchmarkot. Az egyik legmarkánsabb különbség, ahogy a *QCanvas* a képeket kezeli. Úgy tehetünk képet a keretre, hogy létrehozunk egy *QCanvasSprite*-ot, egyetlen kerettel. A képet eztán a `QImage::scale()` és a `QImage::smoothScale()` metódusok egyikével méretezhetjük át, aminek eredményeképp egy képet kapunk, és ezzel frissíthetjük a `QCanvasSprite` tartalmát. A méretezés és a kapcsolódó gyorsítótár kezelése ezzel az ügyfélalkalmazásba kerül. Ezzel szemben mind a *GNOME Canvas*, mind az *Evas* közvetlenül támogatja a méretezést, és felelős az átméretezett képek gyorsítótárának kezeléséért is.
A másik fontos különbség, hogy a *Qt* lehetőséget ad a frissítendő terület méretének meghatározására. A *Qt* dokumentációja szerint: *„Ökölszabály szerint válasszunk az átlagos keretméretnél, mozgó objektumok esetén viszont csak azok átlagos méreténél valamivel kisebbet.”*

2. ábra Az evas_bench képátméretezési és -elrendezési benchmarktesztjének gEvas-os változata. A levelet fokozatosan kicsinyíti, amíg eléri a keret bal felső sarkát, majd nagyítja az eredeti méretre.

Az adatmodellek különbözősége miatt az *evas_bench*-nek még nem készítettem el a *Qt*-re írt változatát. Írtam viszont egy keretméretező és összhatást változtató programot, de néhány tervezési különbség miatt nem lehetséges a teljesen egyértelmű összehasonlítás.
Minthogy a *Qt* teljesen az ügyfélprogramba delegálja a gyorsítótár kezelését, úgy döntöttem, hogy az első iterációban az összes átméretezett képet a gyorsítótárba helyezem, és a benchmarkot alaphelyzetbe hozom a későbbi iterációkhoz, amelyek már csak a gyorsítótárban lévő képeket használják.
Ne felejtsük el tehát, hogy a *Qt* méretezési benchmarkja úgy készült, hogy az összes átméretezett kép előre a gyorsítótárba került, és a frissítendő területhez több méretet is használtunk. Emiatt a *QCanvas* eleve előnnyel indult a versenyben az *Evashoz* és a *GNOME Canvashoz* képest.

Induljon a verseny

A fajlagos eredmények a hardvertől függetlenül hasonlóak kellenek, hogy legyenek.
A teljesség kedvéért az általam használt gép paraméterei: *AMD XP-Mobile* 2.4 GHz, 200 MHz *FSB*, 1GB RAM 400 MHz kétcsatornás *cas222*-n és *NVIDIA 5900* videokártya, és a programok, amik a teljesítményt befolyásolhatták: *xorg-x11-6.8.2-1.FC3.13*, *GCC 4.0.0 20050308* (*Red Hat 4.0.0-0.32*) vagy *GCC 3.4.3.* Az *X11*-et *TwinView*-val használom, az egyik képernyő 1024x768, a másik 1600x1200 felbontású, mindkettő 85 Hz-es frissítéssel és 32 bites színmélységgel. A *TwinView* valószínűleg nem befolyásolja a mérést, mivel a keretek megjelenítése olyan szoftverágakon zajlik, amelyek sokkal érzékenyebbek a *CPU*/*RAM* sebességre.
A benchmarkok fordításához általában ezeket a `CFLAGS` és `CXXFLAGS` kapcsolókat használják:

```
-O3 -march=athlon-xp -fomit-
➥frame-pointer
```

Az *Evas* 2005. május 28-án *CVS*-ből letöltött verzióját ezért szintén ezekkel a `CFLAGS` kapcsolókkal fordítottam,

1. táblázat *A qt-canvas-resize program benchmarkja különféle területfrissítési értékekkel*

Alkalmazás	Területfrissítés mértéke	Levél képsebessége	Téglalap képsebessége
`qt-canvas-resize`	default	114	72
`qt-canvas-resize`	32	128	80
`qt-canvas-resize`	64	136	82
`qt-canvas-resize`	128	142	81

csakúgy mint a ***qt-3.3.4*** és ***libgnomecanvas-2.10.0*** könyvtárakat. A ***qt-canvas-resize*** program benchmarkját külön végeztem el az eltérő gyorsítótáras megoldás és a korábban említett területfrissítési optimalizáció miatt. Az eredmények az *1. táblázatban* láthatók. A fő ciklus ***Qt*** része lényegében a következő:

```
QCanvasSprite* leaf_sprite =
➥...;
QCanvasPixmapArray* leaf_tiles =
➥...;
while( running )
{
  while( app->has
  ➥PendingEvents() )
    app->processEvents();
  QImage im = ... from cache
  ➥...;
  QCanvasPixmap* qpix = new
  ➥QCanvasPixmap( im );
  leaf_tiles->setImage( 0,
  ➥qpix );
  leaf_sprite->setFrame(0);
  canvas->update();
}
```

A program működését számos parancssori kapcsoló befolyásolja: az `--alpha-blend-image` kapcsolóval adhatunk áttetszőséget a piros téglalapnak a levél helyett, a `--chunk-size` kapcsolóval bírálhatjuk felül a területfrissítés alapértékét. Az `--alpha-blend-image` kapcsolót a `qt-canvas-resize`, a `gnome-canvas-resize` és a `(g)evas-resize` programok mindegyike használhatja.
A ***Valgrind*** `callgrind`-jével néhány percig futtatva a `qt-canvas-resize` programot, az alapértelmezett területfrissítést használva, a levél esetében a `QCanvas::update()` a teljes futásidő 30 %-át emészti fel, míg 59 %-ért a `QCanvasPixmap::init()` metódus felel. Ezen feltétlenül lehet javítani, ha a gyorsítótárba kerülő képeket a `QCanvasSprite` számára `QCanvasPixmapArray`-ben tároljuk. Az előtárazásnak ezt a szintjét a `-Z` kapcsoló hozzáadásával teszteltem, aminek hatására a gyorsítótárban lévő összes kép egy `QCanvasPixmapArray`-be kerül a `QCanvasSprite` támogatására. Ezzel az optimalizációval 559-es képsebesség valósítható meg, miközben a futásidő 78 %-át viszi el a `QCanvas::update()` és 7 %-át a `QCanvasSprite::setFrame()`.

2. táblázat *A GNOME Canvas és a (g)Evas képátméretezési benchmarkja*

Alkalmazás	Levél képsebessége	Téglalap képsebessége
`gnome-canvas-resize`	21	21
`gnome-canvas-resize --aa`	149	127
`evas-resize`	190	184
`gevas-resize`	185	177

3. táblázat *Az evas_bench és a GNOME Canvas-ra átírt változatának viadala*

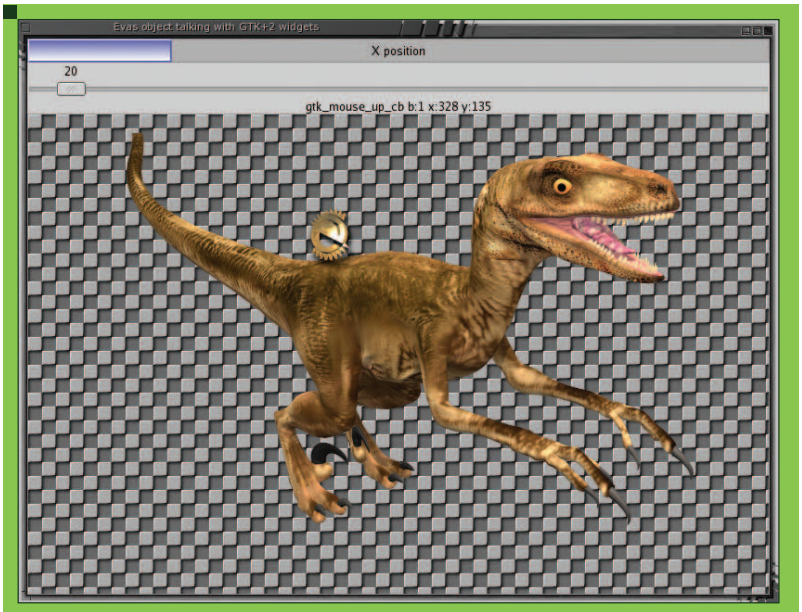
Alkalmazás	Képsebesség	EVAS_BENCH
`gnome-canvas-port-evas-bench --aa`	90	1.49
`evas_software_x11_main`	164	2.75
`evas_software_x11_main --smooth-off-for-some`	200	3.32
`evas_buffer_test`	290	4.83

Meg kell jegyezni, hogy az előtárazásnak ez a mértéke méltánytalan előnyhöz juttatja a ***QCanvast*** a képalkotás sebességében.
A képméretezés és összhatás ***GNOME Canvas*** programja a `gnome-canvas-resize`, melynek `--a` kapcsolójával választható ki a ***GNOME Canvas*** áttetszőséget megvalósító back-endje. Az `evas-resize` programnak nincs parancssori kapcsolója.
A `--a` kapcsoló hiányában a `gnome-canvas-resize` az idő 99 %-át a `gtk_widget_send_expose()` függvényben tölti, amelyet így vagy úgy a `g_main_context_iteration()` függvényből hívunk. Úgy tűnik, hogy a másik ***GTK+2*** motor számára nem előnyös a benchmark diktálta igénybevétel. Ha `callgrind`-del megnézzük a `--aa` ***GNOME Canvas*** back-endet, azt látjuk, hogy az idő 96 %-át a `gtk_widget_send_expose()`-ban tölti, ugyanakkor most az idő 66%-a telik el a `gdk_pixbuf_composite()`-ban, amelyet indirekt módon hívunk a `gtk_widget_send_expose()`-ból. Az `evas-resize` az idő 99 %-át az `evas_render_updates()`-ben tölti, és az ebből hívott függvények az idő 91 %-ában a méretezéssel foglalkoznak. Az `evas-resize` programot úgy is átírtam, hogy a ***gEvas API***-ját használja. Némi sebességcsökkenés észlelhető a ***GTK+2*** szignálok és a ***gEvas*** körítése miatt, de ez nem jelentős.

Az összehasonlítás szerint a közvetlen képátméretezésben a ***GNOME Canvas*** és a ***QCanvas*** hasonló eredményeket adnak, és mindkettő lassabb az ***Evas***-nál, amely ráadásul további előnyhöz jut, ha a képet áttetszővé kell tenni a háttér előtt.
Az `evas_bench` programból eltávolítottam azokat a funkciókat, amelyeket nehéz lett volna megvalósítani a ***GNOME Canvas***-ben. A többi funkció teljesítményre gyakorolt hatásának mérése opcionális.
A vágóterületek beállítása ***Evasból*** nehezen vihető át a ***GNOME Canvas***-be, így ezt az `evas_bench` programban nem engedélyeztem. Az egyenletes átméretezés nagy mértékben rontja a teljesítményt, így az ***Evas***-változathoz egy parancssori kapcsolót adtam, amellyel ez kikapcsolható.
Hadd jegyezzem meg, hogy az ***Evas*** pillanatnyilag nem rendelkezik gyorsítótárral az átméretezett képek tárolására, így az ***Evas*** benchmarkokban minden egyes keret végrehajtja az átméretezést és az áttetszőség biztosítását.
Az `evas_buffer_test` program is ezt teszi, de a képet egy, a memóriában lévő, 32 bites RGBA tárolóban állítja elő.

gEvas

A ***gEvas*** magja lényegében 5 dologból áll: megmondja az ***Evas***-nak, mikor rajzolja újra magát, segédkezik az ***Evas*** eseményeinek és a ***GTK+2***



3. ábra Az Evas és GTK+2 szignálok összekapcsolása. A dino közvetlenül vagy a csúszkák segítségével is mozgatható.

szignálok összerendelésében, kezeli az ***Edje*** időzítőit az animációk támogatásához, segít abban, hogy az ***Evas*** jól kezelje a ***GTK+2*** widgeteket, továbbá kódot nyújt az ***Evas*** helyes használatához. Mivel az ***Evas*** úgy készült, hogy beágyazott rendszerekhez is megfelelő legyen, az ***Evas*** magjának egy része kimaradt, hogy a kódméret kicsi maradjon. Mivel a ***gEvas*** desktop-orientált, néhány hasznos függvényt is tartalmaz a desktop-alkalmazásokhoz.
Az alábbi kód egy gördíthető területen hoz létre ***gEvas*** keretet, majd egy ***GTK+2*** ablakhoz csatolja. Minthogy nem minden gördíthető ***gEvas***, így például a ***GIMP*** sem, engedi a középső egérgombbal megváltoztatni a keret helyzetét, ezt a `gevas_new_gtkscrolledwindow()` függvényen kívül kell beállítani:

```
GtkWidget* window = 0;
GtkWidget* scw    = 0;
GtkWidget* gevas  = 0;
window = gtk_window_new
➥(GTK_WINDOW_TOPLEVEL);
gevas_new_gtkscrolledwindow(
   (GtkgEvas**)(&gevas), &scw );
gtk_container_add(GTK_CONTAINER
➥(window), scw);
gtk_scrolled_window_set_policy(
   GTK_SCROLLED_WINDOW(scw),
   GTK_POLICY_AUTOMATIC,
   ➥GTK_POLICY_AUTOMATIC);
gevas_set_middleb_scrolls(GTK_G
EVAS(gevas), 1,
  gtk_scrolled_window_get_
  ➥hadjustment(
     GTK_SCROLLED_WINDOW(scw)),
  gtk_scrolled_window_get_
  ➥vadjustment(
     GTK_SCROLLED_WINDOW
     ➥(scw)));
```

Az objektumok létrehozása ***gEvas***-ben és a ***GTK+***-ban egy tőről fakad, néhány metódusnak a `GtkgEvasObj` osztályhoz kapcsolásával. Ebből származik a többi speciális osztály, például a `GtkgEvasImage`. Sajnos, ez maga után vonja a típuskényszerítéssel terhelt ***ANSI C GTK+*** kódolást.
Az alábbi kód egy képet készít, amely eredeti méretben mutatja egy ***PNG***-fájl tartalmát. Ezután a képet mozgatjuk, és a keretben lévő rétegét is megváltoztatjuk:

```
GtkgEvasImage* gi;
GtkgEvasObj* go;
gi = gevasimage_new_from_
➥metadata(
        GTK_GEVAS(gevas), "/my/
        ➥path/foo.png" );
go = GTK_GEVASOBJ( gi );
int x = 100, y = 50;
gevasobj_move(      go, x, y );
gevasobj_set_layer( go, 1 );
```

Készítettem egy egyszerű programot, amely bemutatja, hogyan kapcsoljuk az ***Evas*** eseményeit egy ***GTK+2*** widgethez a ***gEvas*** widgeten kívül. A ***signalconnect.c*** a ***gEvas*** csomag ***demo*** könyvtárában található. Bonyolultabb példákkal, köztük. ***glib*** szignálokra irányított, egyszerű ***Evas*** és ***Evas***-vezérelt függvényvisszahívásokkal szolgál a testgevas program. A szignálkapcsolást a *3. ábra* mutatja be.
Az ***Evas*** eseményeit a `GtkgEvasEvHClass` alosztály segítségével kapcsoljuk. Az alábbi kódrészletben az evh az egér fel-le mozgásának ***Evas***-eseményeit ***glib***-szignálokra irányítja, azokat pedig eseménykezelő függvényekhez kapcsoljuk. Ha a felhasználó mozgatja a ***raptort***, a `raptor_moved()` függvény hívódik egy ***glib2***-szignál hatására, hogy különféle ***GTK+2*** widgeteket frissítsen a kép aktuális koordinátáival:

```
static gint raptor_moved(
    GtkgEvasObj* o,
    Evas_Coord* x, Evas_Coord*
    ➥y,
        gpointer user_data )
{
 gtk_progress_bar_set_fraction
 ➥( x_coord_tracker,
      (1.0 * (*x)) /
      ➥CANVAS_WIDTH );
 gtk_range_set_value(
      GTK_RANGE
      ➥(y_coord_tracker), *y );

 return GEVASOBJ_SIG_OK;
}
static gboolean
gtk_mouse_down_cb(GtkObject *
➥object,
 GtkObject * gevasobj, gint _b,
 ➥gint _x, gint _y,
 gpointer data)
{
 char buffer[1024];
 snprintf(buffer,1000,
 ➥"mouse_down b:%d x:%d y:%d",
          _b, _x, _y);
 gtk_label_set_text( e_logo
 ➥_label, buffer );
 return FALSE;
}
...
gi = gevasimage_new();
go = GTK_GEVASOBJ( gi );
gevasimage_set_image_name( gi,
➥"raptor.png" );
```

```
...
/** Let the user drag the
➥raptor around **/
GtkObject *evh = gevasevh_
➥drag_new();
gevasobj_add_evhandler( GTK_
➥GEVASOBJ( gi ), evh );
gtk_signal_connect( go,
➥"move_absolute",
   GTK_SIGNAL_FUNC( raptor_
   ➥moved ), go );
gi = gevasimage_new();
go = GTK_GEVASOBJ( gi );
gevasimage_set_image_name( gi,
➥"e_logo.png" );
...
evh = gevasevh_to_gtk_
➥signal_new();
gevasobj_add_evhandler( GTK_
➥GEVASOBJ( gi ), evh );
gtk_signal_connect(GTK_OBJECT
➥(evh), "mouse_down",
  GTK_SIGNAL_FUNC
  ➥(gtk_mouse_down_cb), NULL);
gtk_signal_connect(GTK_OBJECT
➥(evh), "mouse_up",
  GTK_SIGNAL_FUNC
➥(gtk_mouse_up_cb), NULL);
```

Íme néhány hasznosabbnak tűnő eseménykezelő, amelyeket szintén csatolhatunk:

```
/* Standard GTK+ popup menu
➥creation + handling */
static gboolean
gtk_popup_activate_cb(GtkObject
➥* object,
   GtkObject * gevasobj, gint
   ➥_b, gint _x, gint _y,
   gpointer data)
{
 static GtkMenu *menu = 0;
 ...
}
GtkgEvasObj* go  = ...;
GtkObject*   evh = 0;
/* Make the object throb when
➥mouse is over it */
GtkgEvasEvHThrob* evht =
➥gevasevh_throb_new( go );
/* Allow the user to drag the
➥object around  */
evh = gevasevh_drag_new();
gevasobj_add_evhandler( go,
➥evh );
/* Make a popup menu appear on
➥right mouse click */
evh = gevasevh_popup_new();
gevasobj_add_evhandler( go,
➥evh );
gtk_signal_connect(GTK_OBJECT
➥(evh),"popup_activate",
   GTK_SIGNAL_FUNC(gtk_popup_
   ➥activate_cb), NULL);
```

A keretben lévő kijelölés kezelése az előző eseménykezelőknél kicsit trükkösebb, mivel a kijelölésben több objektum vesz részt. Létre kell hoznunk a `GtkgEvasEvHGroupSelector` osztályból egy objektumot, amely a nem kiválasztható háttérobjektumhoz kapcsolódik. Úgy is gondolhatunk erre az objektumra, mint ahová a kijelölő téglalapot húzzuk, hogy megmutassuk, mely objektumokat szeretnénk kiválasztani. A kijelölő téglalapnak mindig a kiválasztandó objektumok feletti rétegben kell elhelyezkednie. Így a keretben mindegyik kiválasztható objektumhoz egy `GtkgEvasEvHSelectable` objektum csatlakozik, amely a `GtkgEvasEvHGroupSelector` objektummal kommunikál:

```
GtkWidget*     gevas = ...;
GtkObject*     evh_selector =
➥0;
GtkgEvasImage* gevas_image;
gevas_image = gevasimage_new();
gevasobj_set_gevas(gevas_image,
➥gevas);
gevasimage_set_image_name(gevas
➥_image,".../bg.png");
/* Make this a group_selector */
evh_selector = gevasevh_group_
➥selector_new();
gevasevh_group_selector_set_obj
➥ect(
  (GtkgEvasEvHGroup
  ➥Selector*)evh_selector,
  GTK_GEVASOBJ(gevas_image));
GtkgEvasObj* go  = ...;
make_selectable( gevas, go,
➥evh_selector );
...
/* lets make this object also
➥selectable */
void make_selectable( Gtkg
➥EvasObj* object,
            GtkObject* evhsel )
{
 GtkgEvasObj* ct  = 0;
 GtkObject* evh = gevasevh_
 ➥selectable_new( evhsel );
 gevasevh_selectable_
 ➥set_confine(
    GTK_GEVASEVH_SELECTABLE
    ➥(evh), 1 );
gevasobj_add_evhandler(object,
➥evh);
gevasevh_selectable_set_
➥normal_gevasobj(
   GTK_GEVASEVH_SELECTABLE
   ➥(evh),  object);
ct = (GtkgEvasObj*)gevasgrad_
➥new(
    gevasobj_get_gevas( GTK_
    ➥OBJECT(object)));
gevasobj_set_color( ct, 255,
➥200, 255, 200);
gevasgrad_add_color(ct, 120,
➥150, 170, 45, 8);
gevasgrad_add_color(ct, 200,
➥170, 90, 150, 16);
gevasgrad_set_angle(ct, 150);
gevasobj_resize( ct, 200,100);
gevasobj_set_layer(ct, 9999);
gevasevh_selectable_set_
➥selected_gevasobj(evh,ct);
}
```

Ezután már könnyen ellenőrizhető, hogy egy objektum kiválasztásra került-e, illetve létrehozhatunk egy gyűjtőobjektumot, amely műveleteket végez az összes kiválasztott objektumon:

```
GtkgEvasEvHGroupSelector* ev =
➥...;
GtkgEvasEvHSelectable*    o  =
➥...;
GtkgEvasObjCollection*    col =
➥0;
gboolean yn = gevasevh_group_
➥selector_isinsel(ev,o);
col = gevasevh_group_selector_
➥get_collection( ev );
gevas_obj_collection_move_
➥relative( col, 100, 200 );
```

Néhány objektumot, köztük a ***geTransAlphaWipe***-ot, továbbá azért hoztak létre, hogy képátmeneteket hajtsanak végre, még az ***Edje*** létezése előtt. Bár az ***Edje*** a jövő útja, az ***alphawipe*** kóddal egyszerű képátmenet valósítható meg az ***Edje*** bevonása nélkül. Ezt használja a ***gevasanim***, hogy foltszerű objektumot hozzon létre, amely keretről keretre változik az áttetszőség segítségével.
A `gEvas xxx_from_metadata()` függvényeivel egyetlen karakterfüzérrel beállítható egy új objektum helyzete, a képfájl neve, a láthatóság és más attribútumok. A `from_metadata()` és a képátmenettel kapcsolatos kód már megtalálhatók az ***Edje***-ben is:

```
sprite = gevas_sprite_new
➥( GTK_GEVAS(gevas) );
for( i=1; i<frame_count; ++i )
{
 gchar* md = g_strdup_printf(
   "cell%ld.png?x=120&y=
   ➥120&visible=0&fill_size=1"
   ,i);
 gi = gevasimage_new_from_
 ➥metadata( GTK_GEVAS(gevas),
 ➥md );
 g_free( md );
 gevas_sprite_add( sprite, GTK_
 ➥GEVASOBJ( gi ) );
}
gevas_sprite_set_default_frame_
➥delay( sprite,2000 );
gevas_sprite_play_forever(
➥sprite );
/* frame transitions */
geTransAlphaWipe* trans = 0;
trans = gevastrans_
➥alphawipe_new();
for( i=0; i<frame_count; ++i )
 gevas_sprite_set_transition_
 ➥function(
     sprite, i, trans );
```

Végezetül, a *gEvas* keretre egy *Edje* objektumot teszek. A cikkben is használt *gevasedje* bemutatóprogram a *gEvas* disztribúció *demo* könyvtárában található. A legérdekesebb részletek alább láthatók. Az *Edje*-nek egy forgó *Enlightenment* logója van, amely egérkattintásra lüktet:

```
/* init engines */
ecore_init();
edje_init();
gtk_init(&argc, &argv);
...
gevas = ...;
/* allow edje to update the
➥canvas as well */
gevas_setup_ecore( (GtkgEvas*)
➥gevas );
/* place an edje object */
GtkgEvasEdje* gedje
 = gevasedje_new_with_canvas
 ➥( gevas );
/* eet files can contain many
➥edje objects */
gevasedje_set_file( gedje,
➥"e_logo.eet", "test" );
go = GTK_GEVASOBJ(gedje);
gevasobj_move(      go, 300,
➥300 );
gevasobj_resize(    go, 370,
➥350 );
gevasobj_set_layer( go, 10 );
gevasobj_show(      go );
```

Zárszó

A memóriafogyasztással kapcsolatos eredményeket részben terjedelmi okokból mellőztem, részben pedig azért, mert a korszerű asztali számítógépekben ez a tényező kevésbé fontos. A *QTCanvas* és az *Edje* képátméretezésének igazságosabb összehasonlítását egy másik benchmarknak kell majd elvégeznie. Ha egy *GTK+2* alkalmazás készítéséhez fedőkönyvtárra lenne szükség, vegyük számításba a *GNOME Canvast* és a *gEvas-Edje* párost is, mielőtt nekiállnánk barkácsolni.

Linux Journal 2006., 141. szám

Ben Martin ideje javát virtuális fájlrendszerekkel és adatbányászattal tölti, bár úgy is jól ismert, mint aki szeret mások képeinek két- és háromdimenziós animációjával játszadozni.

KAPCSOLÓDÓ CÍMEK

- www.linuxjournal.com/article/8647