SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Diplomski rad br. 1909

IZRADA PROGRAMSKE BIBLIOTEKE ZA PRIKAZ REZULTATA POSTUPAKA RAČUNALNOG VIDA

Dražen Kačar

Zagreb, svibanj 2012.

Sadržaj

Uvod

Obrada slika u realnom vremenu traje onoliko koliko mi imamo strpljenja čekati da obrada završi, što je obično nekoliko stotina milisekundi. Ako upotrijebimo jača računala, obrada slike ponovo traje jednako, samo što s takvim sklopovljem obrađujemo veće rezolucije ili koristimo zahtjevnije algoritme. Ili oboje.

U novijim radovima[54] o dizajnu korisničkog sučelja (ili dizajnu interakcije s korisnikom) možemo pročitati da se trebamo odmaknuti od načina programiranja koje pokušava minimizirati upotrebu procesorskog vremena, jer tog vremena navodno ima na bacanje. To je točno za određenu klasu programa, ali obrada slika još nije ni blizu tom stanju. A kad će biti, ne zna se.

Banalno svojstvo programa za obradu slika je da je njihov ulaz (a često i izlaz) grafički, pa im je potreban način za grafički prikaz izlaznih rezultata i jednostavnu interakciju s korisnikom. Međutim, većina grafičkih sustava (i biblioteka za izradu korisničkih sučelja) pretpostavlja da program vremena ima na bacanje jer uglavnom čeka da korisnik pritisne tipku. Za programe koji ne spadaju u tu kategoriju korištenje popularnih grafičkih biblioteka nije najbolja opcija.

Postoji potreba za jednostavnom grafičkom bibliotekom koja će zadovoljiti sljedeća svojstva:

- Vrlo jednostavna za upotrebu, ne zahtijeva da korisnik poznaje detalje prozorskog sustava.
- Čuva sadržaj prozora, što znači da se sama brine o eventualnom ponovnom iscrtavanju kad to zatraži prozorski sustav, a prema korisniku održavati iluziju perzistentne površine za iscrtavanje.
- Ne prekida glavni program zahtjevima za obradu. Umjesto toga sprema događaje (npr. pritisnute tipke) u međuspremnik odakle ih glavni program može pročitati kad je njemu zgodno.

Kod izrade programa s grafičkim sučeljem najraširenija paradigma je programiranje upravljano događajima (eng. event-driven programming). Takav status ima zato što je većina popularnih biblioteka za razvoj korisničkih sučelja prihvatila tu paradigmu[28].

U takvom načinu programiranja tok programa određuju događaji, kao što su pritisci na tipke miša ili prekidi uzrokovani mjeračem vremena. Obično takvi programi imaju glavnu petlju koja obrađuje sve moguće izvore događaja i poziva odgovarajuće funkcije za obradu. Funkcije za obradu svoj

posao mogu raditi sinkrono ili asinkrono. Asinkrono programiranje je dosta teže od sinkronog, ali ne blokira glavnu petlju.

Ako prihvatimo ovakvu paradigmu, obično uz nju moramo prihvatiti i arhitekturu koji će uvjetovati način programiranja, odn. dizajn programa. Konkretne implementacije takvih biblioteka, naime, dolaze s konkretnim implementacijama glavne petlje i čvrsto postavljenim temeljima u koje se aplikacija mora uklopiti.

Programiranje upravljano događajima na prvi pogled izgleda jednostavno zbog jednostavnog reaktivnog uzorka ponašanja. Program uglavnom miruje u glavnoj petlji, a budi se kad dođe neki podražaj. Podražaj uzrokuje izvršavanje funkcije za obradu, a kad ona završi svoj posao, program se ponovo vraća mirovanju u glavnoj petlji.

Ovakav uzorak ponašanja uistinu jest jednostavan sve dok se u obradu podražaja ne uplete ovisnost o kontekstu, tj. mogućnost da se isti podražaj obradi na različite načine, ovisno o stanju programa. Kako programiranje upravljano događajima nema dobar način za rad s kontekstom, veliki broj programa koji koriste ovu pradigmu većinu svog koda troše na upravljanje kontekstom, odn. na čuvanje stanja po lokalnim varijablama (bilo globalnim, bilo enkapsuliranim u objekte ili složenim na neki drugi način, npr. kao stanja u konačnom automatu)[29].

Održavanje ovakvog koda zahtijeva nemali napor koji obično raste s brojem linija u programu, bez obzira na pokušaje modularizacije i enkapsulacije. Moguće je da je taj napor vrijedno uložiti u programe koji trebaju biti gotovi, ispolirani proizvodi. Međutim, vrlo je upitna isplativost investiranja vremena i energije u programiranje pod ovom paradigmom za demo programe kod kojih je grafički prikaz nužnost zbog grafičkog ulaza i izlaza programa, a pravi se posao zapravo odvija u dijelu programa koji obrađuje slike.

Osim povećeg napora za održavanje, ova paradigma nije jako pogodna ni za rad s višestrukim tokovima izvršavanja. Jednostavni prethodno opisana model je predviđen za jedan tok u kojem se izvršava glavna petlja i funkcije za obradu, dok ostali procesorski resursi ostaju neiskorišteni. Postoje, doduše, klase problema koji se uspješno mogu rješavati multipliciranjem glavne petlje. Uglavnom se radi o klijent-poslužitelj modelu u kojem se na strani poslužitelja odvoji jedan tok za svakog klijenta, pa moduli unutar poslužitelja mogu biti pisani jednostavno, bez obraćanja pažnje na višestruke tokove i sinkronizaciju. Međutim, upotreba ovakvog modela ovisi o postojanju više klijenata koji nezavisno jedan od drugoga traže neku obradu od poslužitelja.

Programi koji imaju grafički izlaz (a možda i ulaz) često imaju algoritme kojima nikad nije dovoljno procesorske snage, pa je iskorištavanje pune mogućnosti sklopovlja vrlo poželjna karakteristika.

Na kraju, programi koji imaju grafički izlaz se moraju moći izboriti s prozorskim sustavom i korisnikom koji sjedi ispred tog sustava, te njihovim kombiniranim zahtjevima. Prozorski sustavi obično ulazne podražaje koji dolaze direktno od korisnika (npr. pritisak tipke ili pomicanje miša) grafičkim programima dostavlja kao događaj i očekuje da oni na njega odmah reagiraju. Neke događaje generira sam prozorski sustav, npr. zahtjev za iscrtavanjem dijela prozora koji je prethodno bio nevidljiv, a upravo je postao vidljiv. Neki događaji dolaze od drugih aplikacija, npr. zahtjev za sadržajem međuspremnika prozorskog sustava (eng. clipboarda).

Na sve ove događaje treba reagirati u vrlo kratkom vremenskom intervalu, a za programiranje odgovarajuće reakcije je potrebno prilično dobro poznavanje prozorskog sustava. Biblioteke koje implementiraju korisnička sučelja obično apstrahiraju najniže detalje rada prozorskog sustava i programeru predstavljaju pojednostavljeni model kojeg je lakše svladati, ali i dalje zahtijevaju učenje rada s tom bibliotekom.

Posljednji problem na koji ćemo se osvrnuti je portabilnost. Prozorskih sustava ima više i ne postoji standardizirano sučelje za programiranje. Postoje biblioteke koje nude portabilni nivo funkcionalnosti, a zatim one implementiraju tu funkcionalnost na prozorskim sustavima koje podržavaju. Korištenje takvih biblioteka je obično opravdano za programe koji koriste veliki dio funkcionalnosti tih biblioteka, ali je prilično upitno za programe kojima je potrebna samo minimalna funkcionalnost.

Naime, moguće je da će portiranje biblioteke na novu platformu potrajati neko dulje vrijeme (do nekoliko godina). S druge strane, implementiranje minimalne funkcionalnosti direktno na novom prozorskom sustavu će vjerojatno uzeti svega nekoliko dana. Međutim, program koji je izgrađen oko glavne petlje biblioteke koja implementira procesiranje događaja više nije jednostavno prenosiv na platformu na kojoj ta biblioteka nije dostupna.

Zbog gore navedenih razloga postoji klasa programa kojima dominantna paradigma programiranja na grafičkim sustavima ne odgovara. Tome možemo dodati i želju da se programera potpuno oslobodi rada s prozorskim sustavom kad njegov program nije proizvod namijenjen tržištu, nego ima drugu svrhu, na primjer edukaciju ili istraživanje.

U ovom radu ćemo predstaviti PRPlib, grafički modul koji na X Window sustavu implementira jednostavnu biblioteku za prikaz rezultata postupaka računalnog vida. U prvom poglavlju opisujemo neke od postojećih sustava koji već imaju biblioteku sličnu PRPlibu. U drugom poglavlju opisujemo X Window sustav, te biblioteku GTK koja olakšava izradu programa s grafičkim sučeljem. GTK pokazuje sve karakteristike dominantne arhitekture zbog koje se dizajn glavnog programa mora njoj podčiniti, a PRPlibova zadaća je da programera potpuno izolira od takvih zahtjeva. Arhitektura PRPliba je opisana u trećem poglavlju, gdje su navedeni i rezultati mjerenja performansi različitih verzija PRPliba. U četvrtom poglavlju donosimo prijedloge za budući razvoj za koji je potrebna kooperacija PRPliba i glavne aplikacije, pa ih PRPlib ne može nezavisno implementirati.

1. Postojeći sustavi

Ovdje ćemo ukratko prikazati srodna rješenja u postojećim sustavima drugih autora. Zanimaju nas portabilnost s obzirom na OS, iz kojih se programskih jezika sustav može koristiti, te pod kojim se uvjetima sustav distribuira.

Arhitekturalno nam je zanimljiva podrška za višestruke tokove izvršavanja, podrška za ispisivanje lokaliziranog teksta, te jednostavnost korištenja u smislu izrade naivnih, nesofisticiranih grafičkih programa. Često je potrebno demonstrirati neku grafičku funkcionalnost (npr. algoritma), a za to je obično potrebno prikazati nekoliko slika na ekranu i baratati jednostavnim ulazima s tipkovnice i miša.

1.1 OpenCV

OpenCV[45] (Open Source Computer Vision) je kolekcija algoritama i programa za rad na polju računalnog vida. Distribuira se pod BSD licencom. Postoje sučelja za C, C++ i Python na Microsoft Windowsima, Unix sustavima, Androidu i Macu. Biblioteka je kompatibilna s IPL (Intelova Image Processing Library) i može koristiti IPP (Intelova Integrated Performance Primitives biblioteka) za poboljšavanje performansi.

OpenCV pruža portabilne tipove podataka i operatore na niskom nivou, te skup funkcionalnosti višeg nivoa za hvatanje video zapisa, procesiranje i analizu slika, strukturnu analizu, analizu pokreta i praćenje objekata, raspoznavanje objekata, kalibraciju kamere i 3D rekonstrukciju.

OpenCV je dizajniran za primjenu u normalnim aplikacijama, pa se očekuje da će se za takve svrhe upotrebljavati grafička sučelja u kojima se inače pišu aplikacije (npr. Qt, WinForms ili Cocoa). Međutim, ponekad je potrebno ili poželjno imati jednostavno grafičko sučelje za brzu izradu prototipa. Za tu namjenu OpenCV sadrži HighGUI modul kojeg ćemo ovdje ukratko opisati.

U verziji 2.3.1 HighGUI modul sadrži funkcije koje se grubo mogu podijeliti u dvije grupe: korisničko sučelje i funkcije za čitanje i pisanje različitih slikovnih i video formata. Funkcionalnost korisničkog sučelja je implementirana pomoću sljedećih grafičkih biblioteka: Qt, GTK, Win32, Carbon i Cocoa, ali samo jedna od njih može biti izabrana prilikom konfiguracije prevođenja

OpenCV-a.

Korisničko sučelje omogućava kreiranje prozora koji pokazuju sliku i sami se brinu za iscrtavanje kad od operativnog sustava dobiju zahtjev za ponovnim iscrtavanjem sadržaja prozora. Svaki prozor može imati jedan ili više trackbara s pridruženom callback funkcijom koja će biti pozvana kad korisnik pomakne traku za pomicanje. Prozore je moguće pomicati i mijenjati im veličinu.

HighGUI omogućava postavljanje funkcije koja će dobijati informacije o mišu: promjene koordinata i pritiske na tipke. Također postoji funkcija kojom se dobijaju pritisnute tipke, ali je specifična zato što je to jedina HighGUI funkcija koja obrađuje poruke od operativnog sustava, pa o njoj ovisi i iscrtavanje prozora. HighGUI grafički sustav reagira na podražaje samo kad aplikacija čeka na pritisak tipke, pa će u slučaju malo dulje obrade podataka grafičko sučelje biti zamrznuto.

Takvo smrzavanje grafičkog sučelja nije poželjno, pa API zbog toga ima mogućnost pokretanja grafičkog sustava u zasebnom toku izvršavanja, ali je ta funkcionalnost trenutno implementirana samo za GTK modul. Službena dokumentacija za ovu funkcionalnost ne postoji, pa je nezahvalno pogađati kakvi će se zahtjevi postavljati pred programe s višestrukim tokovima izvršavanja jednom kad bude dovršena. GTK modul pomalo nespretno koristi jedan veliki mutex za međusobno isključivanje različitih tokova, pa se može pretpostaviti da HighGUI ne namjerava postavljati nikakve dodatne zahtjeve aplikativnom programu.

Postoji i dodatna funkcionalnost koja je implementirana samo u Qt modulu: statusbar, toolbar, upravljački panel i mogućnost ispisivanja teksta preko slike.

1.2 ViSP

ViSP[46] (Visual Servoing Platform) je C++ platforma za razvoj programa za upravljanje robotima pomoću računalnog vida (eng. visual based robot control). Distribuira se pod GPLv2 i ViSP Professional Edition licencama. Potonja omogućava distribuiranje aplikacija bez izvornog koda. Postoji samo C++ API, a ViSP radi pod Unix sustavima, OS X-om i Microsoft Windowsima.

ViSP biblioteka uključuje i portabilno grafičko sučelje za prikaz slika i razvoj rudimentarnih GUI programa. To sučelje je implementirano korištenjem vanjskih grafičkih biblioteka: Xlib, GTK, Windows Graphics Device Interface (GDI), Windows Direct3D, te pomoću OpenCV biblioteke opisane u prethodnom odjeljku.

API[47] je pomalo rogobatan jer korisnički program mora instancirati neki od specifičnih razreda za prikaz (npr. `vpDisplayX` ili `vpDisplayGTK`), pa programer mora u niz `#ifdef` direktiva staviti instanciranje svih podržanih biblioteka na sljedeći način:

```
#include <visp/vpConfig.h>
#include <visp/vpDisplayX.h>
#include <visp/vpDisplayGTK.h>
#include <visp/vpDisplayGDI.h>
#include <visp/vpDisplayD3D.h>
#include <visp/vpDisplayOpenCV.h>

int main()
{
    vpDisplay *d;

#if defined(VISP_HAVE_X11)
    d = new vpDisplayX;
#elif defined(VISP_HAVE_GTK)
    d = new vpDisplayGTK;
#elif defined(VISP_HAVE_GDI)
    d = new vpDisplayGDI;
#elif defined(VISP_HAVE_D3D9)
    d = new vpDisplayD3D;
#elif defined(VISP_HAVE_OPENCV)
    d = new vpDisplayOpenCV;
#else
#error Cannot find suitable graphics system.
#endif
    ...
}
```

Ovakav način programiranja je mogući izvor problema ako se ViSP koristi kao zajednička biblioteka. Može nam se dogoditi da prevedemo program na jednom sustavu gdje će se koristiti npr. razred `vpDisplayGTK`. Ako nakon toga izvršni kod prenesemo na neki drugi sustav gdje je ViSP instaliran bez modula GTK, dobit ćemo grešku prilikom pokretanja našeg programa. Također je moguće da se u budućnosti s novijom verzijom ViSP-a pojavi neki novi modul za prikaz, te da umjesto stare verzije instaliramo novu koja ima samo taj novi modul. U tom slučaju će naši postojeći ViSP programi prestati funkcionirati na istom računalu na kojem su prvi put prevedeni.

Možemo pokušati izbjeći ove pojave tako što ćemo koristiti razred za prikaz za koji možemo razumno očekivati da će uvijek biti prisutan, ali dosadašnje iskustvo pokazuje da je jedino što možemo razumno očekivati to da ne možemo ni izbliza predvidjeti kako će okolina izgledati za 10 godina. Trenutno, na Unix sustavima, `vpDisplayX` izgleda kao prilično siguran izbor jer je GTK

implementiran korištenjem Xlib-a, pa nam može izgledati da Xlib svakako mora biti prisutan na sustavu. Isto se može reći i za sve grafičke module koje podržava OpenCV.

Međutim, postoje dva problema s tom logikom. Prvi je da GTK podržava više izlaznih modula, a X Window sustav je samo jedan od njih. GTK na Linuxu podržava i direktno korištenje ekranskog spremnika pomoću DirectFB biblioteke, što bi moglo biti i od praktične koristi kod robotskih uređaja. Drugi problem je da je Xlib samo referentna biblioteka za korištenje X11 protokola iz programskog jezika C, ali nije i jedina takva biblioteka. Već danas bismo mogli pomoću biblioteke XCB (X protocol C language Binding) implementirati novi grafički modul koji bi radio s X11 sustavom bez Xlib-a.

Ukratko, bilo bi poželjno da ViSP implementira mogućnost korištenja grafičkog sustava bez direktnog instanciranja konkretnih grafičkih modula.

1.3 Octave

Octave[48] je programski jezik visokog nivoa, primarno namijenjen numeričkim izračunima. Ljuska za rad pruža komandnu liniju za numeričko rješavanje linearnih i nelinearnih problema, kao i za izvođenje drugih numeričkih eksperimenata. Također pruža opsežne grafičke mogućnosti za vizualizaciju i manipulaciju podataka. Octave je uglavnom kompatibilan s programskim paketom Matlab.

U verziji 3.4 postoje dva grafička sustava koji se mogu koristiti. Stariji koristi program gnuplot, a noviji je implementiran pomoću FLTK[49] i OpenGL biblioteka. S obzirom da je Octave specijalizirani jezik visokog nivoa, u njemu se uglavnom crtaju grafovi, pa su i oba grafička sustava prilagođena toj namjeni.

Postoji sedam tipova grafičkih objekata, složenih u hijerarhiju:

1. Globalni objekt `root figure` od kojeg svi drugi objekti nasljeđuju svojstva.
2. Ispod objekta `root` su objekti tipa `figure`. Ovaj objekt predstavlja prozor na ekranu.
3. Ispod `figure` su objekti tipa `axes`.
4. Ispod `axes` su objekti tipa `line`, `text`, `patch`, `surface` i `image`.

Objekt tipa `line` predstavlja liniju u dvije ili tri dimenzije, tip `patch` predstavlja ispunjene poligone (trenutno limitirane na dvije dimenzije), tip `surface` predstavlja trodimenzionalnu površinu, a objekti tipa `text` i `image` predstavljaju tekstualne anotacije i slike. Svaki objekt roditelj može sadržavati više objekata djece, pa tako jedan prozor može sadržavati nekoliko grafova (svaki će biti tipa `axes`) od kojih svaki opet može sadržavati više objekata najnižeg nivoa.

Svaki objekt ima niz svojstava (npr. boja, širina linije, itd.) pomoću kojih se upravlja svim atributima prikaza slike na ekranu. Ukoliko se na već nacrtanom objektu promijeni neko svojstvo, slika će se ponovo iscrtati s novom vrijednošću. S obzirom da je jezik namijenjen vizualizaciji podataka, ne postoje funkcije za crtanje koje su uobičajene kod grafičkih sustava niske razine, kao što su funkcije za crtanje linija, kružnica ili poligona. Umjesto toga postoje funkcije za crtanje matematičkih funkcija, različitih vrsta histograma, kontura, tortnih dijagrama, pa čak i jednostavnih animacija u stilu komete.

Na svaki graf se mogu dodati naslov, oznake za osi i proizvoljni tekst. Tekst koji se ispisuje na grafovima će često biti matematička formula, pa Octave omogućava relativno jednostavno slaganje matematičkih izraza. Objekti tipa `text` imaju atribut `interpreter` kojim se bira jedan od postojećih mehanizama za ispisivanje teksta. Trenutno je implementirana samo opcija `tex` kojom se dobija podskup funkcionalnosti TeX-a kod prikazivanja teksta, što omogućava umetanje grčkih i drugih matematičkih simbola. Također postoje naredbe za upotrebu masnih slova i kurziva, te korištenje određenog pisma i postavljanje njegove veličine.

Zanimljivo je napomenuti da nismo pronašli način za ispisivanje hrvatskih dijakritičkih znakova. U modalitetu `tex` nisu podržane ni TeX ni LaTeX naredbe za kombiniranje znakova i naglasaka, pa se tako ne mogu dobiti ta slova. Ukoliko se kao grafičko sučelje koristi Gnuplot, na ekranu se ispisuje tekst u kodnoj stranici Latin 1, neovisno o lokalu u kojem je Octave pokrenut. Gnuplot (barem novije verzije) može koristiti i druge kodne stranice za ispis teksta, ali mu treba specificirati kodnu stranicu. Octave to ne čini, pa se koristi Latin 1 kao podrazumijevana vrijednost. U ovakvoj okolini je moguće za Gnuplotov X11 terminal specificirati pismo koje koristi kodnu stranicu Latin 2, pa bi se ispravni znakovi pojavljivali na ekranu, ali ne i u drugim izlaznim formatima koji su namijenjeni za ispis ili uključivanje u druge dokumente.

Kada se koristi FLTK/OpenGL grafičko sučelje na ekranu se jednostavno ne ispisuju znakovi koji nisu u ASCII rasponu, obično uz ispisivanje dijagnostičkih poruka s greškama za pojedine znakove.

FLTK 2.0 interno koristi UTF-8, tako da bi rad u UTF-8 lokalu trebao biti moguć bez ikakve dodatne konfiguracije. Pretpostavljamo da do šuma u radu dolazi zato što Octave interno memorira stringove kao matrice (polja) znakova, a za svaki od tih znakova se pretpostavlja da je u kodnoj stranici Latin 1. Službena dokumentacija uopće ne spominje kodne stranice, pa možemo pretpostaviti da korektno ispisivanje teksta ne spada u prioritete ovog programskog paketa.

Octave je dugo vremena bio terminalski program koji je koristio vanjski program (Gnuplot) za crtanje, pa nije imao nikakvo grafičko sučelje za interakciju s korisnikom. Gnuplot je vrlo kompetentan program za crtanje 2D grafova i sasvim pristojan za crtanje 3D grafova, ali je korisničko sučelje za interaktivni rad minimalističko. Moguće je mišem pomicati i povećavati (zoom) 2D graf, a 3D graf se može i rotirati. Prozor s grafom reagira i na neke tipke (npr. "u" za unzoom, "g" za toggle grid itd.), ali je vrlo teško doći do liste svih naredbi jer nisu navedene ni u Gnuplotovoj dokumentaciji. Jedini način je pritisnuti "h" u prozoru s grafom, što će ispisati listu svih naredbi koje se mogu zadati mišem i tipkovnicom. Gnuplot to ispisuje na svoj standardni izlaz, što znači da će tekst ispisati usred komandne linije Octave ljuske. To će poremetiti poziciju značke na ekranu, pa će ljuska i njen korisnik vjerojatno ostati zbunjeni. Jedini izlaz iz ove situacije je Ctrl-C, što prekida uređivanje trenutne Octave naredbe i daje novi prompt.

Gnuplot ima više od 70 različitih izlaznih formata, koji se u njegovoj terminologiji nazivaju terminali. Većinom se radi o slikovnim formatima (npr. GIF, PNG, JPEG, SVG, itd.) ili formatima za dokumente (PostScript, PDF, LaTeX, itd.), ali postoje i različiti interaktivni terminali za prikaz na ekranu (npr. Macintosh, Windows i X11). Za interaktivni rad pod X11 grafičkim sustavom postoje dva Gnuplot terminala: X11 i Xwt. X11 je stariji i nešto lošiji po izgledu i mogućnostima od novijeg terminala Xwt. Xwt iznad prostora za graf ima alatnu traku s ikonama za neke akcije: kopiranje slike u međuspremnik, aktiviranje i deaktiviranje koordinatne mreže, rad s povećanjem slike, te aktiviranje konfiguracijskog dijaloga s dodatnim opcijama od kojih je najvažnija antialiasing, što rezultira puno kvalitetnijim grafovima.

Iako je Xwt Gnuplotov podrazumijevani terminal, Octave koristi X11, pa većina korisnika ni ne zna da bi korisničko sučelje moglo izgledati malo bolje. Moguće je koristiti terminal Xwt postavljanjem varijable okoline GNUTERM na vrijednost `xwt`, ali Octave dokumentacija takvu mogućnost ne spominje. Octave također ne dokumentira koje mehanizme koristi za snimanje slika u različitim formatima. Eksperimentalno se može utvrditi da se za tu svrhu koriste različiti programi, barem Gnuplot i Ghostscript, ali i jedan i drugi imaju mogućnost snimanja slike u najvažnijim formatima

(PS, PDF, JPEG, PNG), pa je nejasno koji će se program kad koristiti. Izgleda da Octave preferira Ghostscript, a ako on ne podržava traženu konverziju koristi se Gnuplot.

Zbog takvog izbora se slika spremljena u datoteku dosta razlikuje od slike koja se vidi na ekranu. Ghostscript snima jako veliku sliku i pri tome koristi pismo konstantne širine koje vrlo ružno izgleda. Octave ima opcije kojim se mogu podesiti ove karakteristike snimljenog dokumenta, ali jedina metoda koja nam stoji na raspolaganju je metoda pokušaja i pogreške. S druge strane, formati koje snima Gnuplot (npr. SVG) izgledaju onako kako izgleda i slika na ekranu, pa njima uglavnom nije potrebno dodatno naštimavanje. Međutim, Octave ne nudi mogućnost izbora programa s kojim će se snimiti datoteka, tako da je u ovom području korisnik prepušten vlastitoj snalažljivosti.

Posljednje nama zanimljivo područje je interaktivno grafičko sučelje. Ono nije ostvarivo kad se za vizualni prikaz koristi Gnuplot zato što taj program nema takvih mogućnosti. Zbog toga je Octave dobio i drugo grafičko sučelje, bazirano na grafičkim bibliotekama FLTK i OpenGL. FLTK se koristi za korisničko sučelje, a pomoću OpenGL-a se crtaju grafovi.

FLTK/OpenGL sučelje na vrhu ima izbornu vrpcu s izbornicima za manipulaciju slikom (snimanje na disk, uključivanje i isključivanje koordinatne mreže, te izbor funkcionalnosti koja se dobija mišem: pomicanje ili rotiranje slike). Postoji API s kojim je moguće dodavati proizvoljne izbornike u izbornu vrpcu i pomoću njih aktivirati funkcije u programu. Taj API je kopija Matlabovog API-ja koji uz to ima i razne druge mogućnosti, ali Octave ih još uvijek nema implementirane.

Objekti tipa `figure` (prozor na ekranu) imaju svojstva pomoću kojih se može specificirati funkcija koju će Octave pozvati kad se u prozoru pritisne tipka ili pomakne miš, pa se i na taj način mogu pisati interaktivni programi. Ova funkcionalnost je podržana samo u FLTK/OpenGL grafičkom sučelju.

2 Grafički sustav na Unixu

2.1 X Window System

Operativni sustavi Unix danas velikom većinom koriste X Window System[2,3], grafički sustav koji je rođen 1984-e, a verzija X11, koja je stabilna osnova i za današnje inačice, izašla je 1987. godine.

Glavne osobine su mu:

- razdvojenost od operativnog sustava
- mrežna transparentnost,
- klijent-poslužitelj arhitektura,
- namjerno izostavljanje grafičkog korisničkog sučelja (GUI-ja).

X sustav je nastajao u doba kad su se grafičke radne stanice počele pojavljivati, a svaki je proizvođač sklopovlja imao i svoje grafičko sučelje koje nije bilo kompatibilno ni s jednim drugim. Takva situacija nije odgovarala proizvođačima programa jer su morali podržavati razne grafičke sustave, pa se vrlo brzo ukazala potreba za jedinstvenim grafičkim sustavom. Ni jedna tvrtka nije imala povjerenja u svoje direktne konkurente, pa je rješenje pronađeno u otvorenom kodu razvijanom na sveučilištu MIT, u zajedničkom projektu s DEC-om i IBM-om. Sustav je morao biti (razumno) portabilan i neovisan o operativnom sustavu da bi ga različiti proizvođači jednostavno mogli uključiti u svoje operativne sustave. Zbog toga X sustav nije ograničen samo na Unix operativne sustave, ali je danas on na njima de facto standard[22].

Mrežna transparentnost je posljedica klijent-poslužitelj arhitekture. X sustav je zapravo mrežni protokol[32] kojim aplikacija (klijent) zahtijeva određene operacije od poslužitelja. X poslužitelj je program koji se izvršava na računalu za kojim sjedi korisnik i koji upravlja ulaznim i izlaznim uređajima, tipično grafičkom karticom, tipkovnicom i mišem. Osim toga on i izvršava grafičke operacije koje traže klijenti, obično crtanje po ekranu.

X klijent je bilo koji aplikacijski program. Može se izvršavati na istom računalu kao i X poslužitelj, ali ne mora. Računalo na kojem se izvršava program klijent ne mora imati ni isti operativni sustav ni isto sklopovlje kao računalo na kojem se izvršava X poslužitelj. Jezgra X11 protokola nije nikad mijenjana, a nove mogućnosti se dodaju u proširenja oko čije upotrebe se klijent i poslužitelj mogu lako dogovoriti. Ovakva arhitektura je rezultirala izuzetnom stabilnošću, pa bi klijenti nastali krajem 80-ih bez ikakvih preinaka uredno funkcionirali s najnovijim X poslužiteljima.

Između normalnih X klijenata i X poslužitelja se za neke operacije nalazi ravnatelj prozorima (eng. window manager), također program klijent čija je zadaća implementacija načina rada s prozorima, manipuliranja fokusom i sličnim operacijama. Ovaj program implementira iscrtavanje vanjskih okvira prozora, promjenu veličine prozora, ikonificiranje, virtualne radne površine i sl.

X Window sustav je primarno protokol i specifikacija primitivnih grafičkih operacija, te namjerno ne sadrži specifikaciju elemenata grafičkog korisničkog sučelja. Umjesto njega aplikacijski programi definiraju taj nivo funkcionalnosti. U praksi obično grafičke biblioteke implementiraju pojedinačne elemente grafičkog sučelja, a aplikacije se implementiraju korišteljem jedne od takvih biblioteka. Zbog toga je tipična radna površina na Unix računalima mješavina različitih grafičkih sučelja. Manjak konzistencije i povremeni problemi s interoperabilnošću među aplikacijama su godinama bili jedan od najžešćih izvora kritike i nezadovoljstva, ali ovakva arhitektura je imala i jednu veliku prednost — bilo je moguće eksperimentirati i zamijeniti bilo koji od nivoa u grafičkom sučelju.

Nakon 20 godina upotrebe pokazalo se da je X11 upotrebljiv prozorski sustav, ali i da ima nekoliko fundamentalno loših karakteristika[9]:

- neadekvatna arhitektura za pisma (fontove) zbog koje je izuzetno teško implementirati dobre WYSIWYG sustave,
- neadekvatna 2D grafika, razvijena na monokromatskim sustavima,
- neadekvatna podrška za ispis,
- arhitekturalno loše izvedena internacionalizacija i upravljanje bojama (color management),
- nemogućnost korištenja naprednih sposobnosti modernog grafičkog sklopovlja, kao što je transparentnost.

Ovi nedostaci su dijelom i posljedica odustajanja korporativnih sponzora od X Window sustava na stolnim računalima, jer se smatralo da je Microsoft zauzeo to tržište, pa je sustav ostao bez sredstava za razvoj. U drugoj polovici devedesetih je razvoj praktično stao, pa su čak i stručne publikacije prestale izlaziti. Jedini vidljivi projekt je bio XFree86, koji se bavio razvojem X poslužitelja na PC sustavima, prvenstveno Linuxu i BSD porodici.

Neočekivana popularnost Linuxa i (s korporativne točke gledišta) uskrsnuće Unixa na stolnim računalima je dovela do preokreta početkom 21. stoljeća, pa se na većini navedenih problema počelo raditi[8], a implementirana rješenja su zahtijevala i preinake u grafičkim bibliotekama i aplikacijama koje su htjele koristiti nove mogućnosti.

2.2 Arhitektura X11 grafičkog okruženja

X Window sustav se oduvijek ponosio time da specificira mehanizam, ali ne i način na koji će taj mehanizam biti implementiran za krajnjeg korisnika (eng. mantra: mechanism, not policy). To znači da uz njega dolaze programi i biblioteke koje implementiraju neke nivoe funkcionalnosti, ali upotreba tih programa i biblioteka nije obavezna.

Zbog toga pod grafičkim okruženjem podrazumijevamo osnovne dijelove X sustava zajedno s popularnijim (ili povijesno važnim) bibliotekama čija upotreba nikad nije bila obavezna.

Osnovna biblioteka koja implementira X protokol se zove Xlib[31] i nju koristi svaki poznati X program — ne zato što mora, nego zato što nije postojala druga biblioteka s istom funkcionalnošću. Xlib sadrži C sučelje za korištenje osnovne funkcionalnosti X protokola, implementira različite vrste optimizacija (npr. više `DrawPoint` zahtjeva kombinira u jedan `PolyPoint` zahtjev), te funkcionalnost za internacionalizaciju ulaza i izlaza. Xlib je jako velika biblioteka jer pokušava biti prikladna za svako korištenje: od malih komponenti do velikih aplikacija[12].

Trenutno je u razvoju biblioteka XCB[50] (X C Language Bindings) koja ima puno uži fokus od Xlib-a. XCB omogućava direktno korištenje protokola, s minimalnom nadgradnjom u samoj biblioteci. Glavne prednosti pred Xlib-om su joj dobra podrška za višestruke tokove izvršavanja i skrivanje latencije pomoću asinkronog rada.

Mada je X protokol asinkron, Xlib sučelje ga čini sinkronim u slučajevima kad klijent treba dobiti odgovor od poslužitelja. XCB nema tako dizajnirano sučelje, pa može poslati više zahtjeva poslužitelju bez čekanja odgovora.

Biblioteka XCB je rađena za relativno uzak skup korisnika — očekuje se da će ju koristiti druge biblioteke za izradu grafičkih elemenata, ali ne nužno i aplikacije. Mada XCB donosi opipljive prednosti, nije realno očekivati da će veliki broj programa biti portiran zato što će u većini slučajeva trošak portiranja biti veći od beneficija koje donosi korištenje XCB-a. Zbog toga je napravljena biblioteka XCL[51] koja implementira Xlib sučelje, ali koristi XCB za implementaciju X protokola.

Postoje programi koji su izgrađeni samo na Xlibu, ali ih nema mnogo. Većina programa želi korisniku prikazati standardizirano korisničko sučelje s njemu poznatim grafičkim elementima za upravljanje. Za tu funkcionalnost su zadužene posebne biblioteke s elementima korisničkog sučelja (eng. widget toolkit). X sustav se oduvijek isporučivao s demo bibliotekom te vrste, po imenu Athena. Athena je potpuno opcionalna biblioteka, za koju se očekivalo da će u ozbiljnoj primjeni

biti zamijenjena boljom bibliotekom istovrsne namjene.

Međutim, između Athene i Xliba sjedi biblioteka zvana Xt (X Toolkit Intrinsics), još jedna opcionalna biblioteka čije je korištenje bilo skoro pa obavezno. Naime, u Xt biblioteku je izdvojen sav negrafički kod koji bi trebao biti zajednički bibliotekama za izradu korisničkih sučelja. Primjerice, Xt sadrži funkcije za postavljanje i dohvaćanje vrijednosti parametara koji opisuju grafičke elemente (npr. boja, debljina linije, širina ispunjenja, izbor pisma, itd.), izradu razreda i nasljeđivanja u C-u, rad s bazom X resursa itd.

Kako funkcionalnost biblioteke Xt zadovoljava mantru mehanizma, ali ne i pravila upotrebe, moglo ju se proglasiti obaveznim dijelom ozbiljne X grafičke okoline, mada ju ni jedna aplikacija ili biblioteka nije morala koristiti.

Prve biblioteke za izradu korisničkih sučelja su koristile (ili pokušavale koristiti) Xt. Danas popularne biblioteke te vrste niti koriste Xt, niti same implementiraju protokole i mehanizme koje je Xt implementirao, pa je Xt i čitav koncept konfiguracije aplikacija pomoću te biblioteke pao u zaborav.

X Window sustav je bio jedan od prvih prozorskih sustava, pa je prirodno očekivati da su se neka njegova arhitekturalna rješenja pokazala dobrima, a neka lošima, te da bi mlađi prozorski sustavi općenito trebali biti bolji jer su mogli učiti na pogreškama prethodnika.

X sustav se razvijao na MIT-u u sklopu akademskog projekta Athena (čiji je krajnji proizvod bila gore spomenuta biblioteka za koju se očekivalo da će u ozbiljnoj primjeni biti zamijenjena kvalitetnijom). Otprilike u isto doba su se razvijali drugi prozorski sustavi na drugim mjestima, pa možemo X sustav usporediti s kasnijom direktnom konkurencijom.

U Sunu je James Gosling sa suradnicima napravio grafički sustav SunDew, kasnije distribuiran pod nazivom NeWS[24] (Network extensible Window System), o kojem kaže[1]:

> SunDew je distribuiran i proširiv prozorski sustav koji se trenutno razvija u Sunu. Proizašao je iz ideje da napravimo korak natrag i promotrimo razna pitanja koja se pojavljuju kod prozorskih sustava bez uobičajenih ograničenja prilikom razvoja proizvoda. Treba ga gledati kao spekulativno istraživanje u smjeru nalaženja pravog načina za izgradnju prozorskih sustava. Počeli smo razgledavanjem postojećih prozorskih sustava i njihovih klijenata i došli do skupa ciljeva. Iz tog skupa ciljeva, i nešto inspiracije, došli smo do dizajna sustava.

Kako je SunDew pandan X Window sustavu, navest ćemo ovdje ciljeve projekta SunDew i usporediti ih s onim što X sustav implementira. Oba su sustava razvijana 1980-ih nezavisno jedan od drugoga. Za svoj idealni sustav Gosling želi ova svojstva:

- **Sačuvani prozori**: čisto sučelje treba u potpunosti sakriti oštećenja prozora od programera. Njegov model prozora treba biti površina po kojoj se može pisati, a ono što se napiše će biti perzistentno.
- **Fleksibilnost**: korisnici trebaju moći priključivati uređaje, mijenjati ponašanje izbornika i općenito moći modificirati bilo koju komponentu sustava. Primjerice, modul koji implementira izbornike trebao bi biti neovisan o pojedinom formatu ili sadržaju izbornika, tako da omogući korisniku razvijanje njegovih vlastitih idioma bez potrebe za ponovnim implementiranjem čitavog sustava. Može izgledati da je ova ekstremna fleksibilnost u suprotnosti s ciljem izrade čistog, jednostavnog i dobro dizajniranog sučelja za programiranje, ali mi vjerujemo da nije tako.
- **Udaljeni pristup prozorima**: u distribuiranoj mrežnoj okolini je prirodno moći pristupiti prozoru s drugog računala.
- **Moćne grafičke primitive**: primitive koje nalazimo na Macintoshu (1985. - op. a.) trebaju biti donja granica. Posljedica naglaska na moć i fleksibilnost je mogućnost emuliranja drugih prozorskih sustava. Primjerice, bilo bi vrlo korisno moći emulirati Macintoshev skup alata.
- **Iskoristiti sklopovlje**: ni jedan od ranije spomenutih sustava nema dobru podršku za boje. U budućnosti će boje igrati još veću ulogu u dizajnu ekrana. Crno-bijelu tehnologiju možemo promatrati kao privremeni nadomjestak za bolju tehnologiju, baš kao što se dogodilo s televizijom. Iz toga proizilazi da datotečni formati za pisma moraju skrivati detalje reprezentacije znakova, s obzirom da bismo jednom mogli željeti podržavati antialiasirani tekst, pa čak i iluminirana samostanska pisma.
- **Dobre performanse**: performanse trenutnog prozorskog sustava su minimum prihvatljivosti. Performanse najčešćih operacija su naročito kritične. Novi sustav bi trebao biti brži od trenutnog kod izvođenja čestih operacija kao što su ponovno iscrtavanje (eng. repainting) ili pomicanje (eng. scrolling) teksta.

Sustav NeWS je bio zasnovan na PostScriptu, koji se do tad koristio isključivo na pisačima. Kod svih drugih onodobnih grafičkih sustava (a i većine današnjih) klijent prozorskom sustavu šalje neku vrstu poruke koja (obično) rezultira iscrtavanjem grafike na ekranu. U sustavu NeWS je ta poruka zapravo PostScript program kojeg aplikacija šalje, a izvršava se unutar prozorskog sustava. Na toj osnovi su izgrađene napredne mogućnosti sustava:

- "Pametni" elementi grafičkog sučelja. S obzirom da se aplikativni program izvršava unutar prozorskog sustava moguće je da npr. tipka sama iscrta svoj novi oblik nakon što ju korisnik pritisne, bez slanja poruke aplikaciji.
- Prema Goslingu, jednostavna zamjena modula. On kaže da se njegov primjer sa zamjenom modula koji implementira izbornike jednostavno rješava tako da korisnik modificira PostScript proceduru, a pravila dometa (eng. scope rules) u jeziku će definirati je li promjena lokalna ili globalna. Međutim, ova mogućnost je varljiva i puna zamki jer ne osigurava kompatibilnost promijenjenog modula s budućim verzijama originalnog modula, pa bi se lako moglo dogoditi da bude kontraproduktivna. S druge strane, s obzirom da je PostScript interpretirani jezik, ovakav dizajn omogućava korisnicima popravljanje grešaka bez čekanja na vlasnika autorskih prava. Međutim, čak i liberalni zakoni koji dopuštaju takve promjene ne dopuštaju distribuciju promijenjenog koda bez dozvole vlasnika autorskih prava, pa čak i ta mogućnost ima diskutabilnu vrijednost.
- Sinkrona obrada događaja bez nužnog kontaktiranja aplikacije. PostScript programi koje je aplikacija poslala prozorskom sustavu mogu obrađivati događaje i odlučivati hoće li kontaktirati aplikaciju ili ne. Ovo također omogućava jednostavnu i pouzdanu sikronizaciju događaja.
- Izvanredne grafičke primitive, daleko iznad mogućnosti bilo kojeg drugog sustava iz tog vremena. Osim za samo crtanje, PostScript se jednostavno mogao koristiti i za definiranje nepravokutnih oblika prozora, bez većeg utjecaja na brzinu prikaza.

Glavne mane sustava su bile loše performanse (tadašnje radne stanice su imale jedva dovoljno procesorske snage za takav sustav) i potreba da programeri ranije ili kasnije nauče PostScript. Drugi zahtjev je predstavljao veći problem za šire prihvaćanje. PostScript je jezik baziran na stogu, pa je za njega jednostavno generirati programe, ali je ljudima takav način programiranja teži. Uz to su Gosling i suradnici svom PostScriptu dodali i objektno-orijentirane mogućnosti, što je u to vrijeme

bila nepoznata paradigma van akademskih krugova, pa je i to djelovalo kao barijera prihvaćanju.

Sustav NeWS se nije uspio nametnuti kao glavna grafička okolina na Unix sustavima, pa je sljedeća inkarnacija bila u vidu dualnog sustava koji je podržavao i NeWS i X, ali je u takvoj kohabitaciji grafički sustav imao dosta lošije performanse od bilo koje komponente zasebno. Na kraju se X sustav uspio nametnuti kao standard za Unix, a NeWS projekt je prekinut i povučen s tržišta. Nakon toga se James Gosling prebacio na izradu Jave i tu je jako pazio da platforma bude prihvatljiva prosječnom programeru.

Ovdje nam nije cilj direktno uspoređivati mogućnosti sustava X i NeWS, nego promotriti arhitekturalne odlike X sustava iz druge perspektive. To će nam dati zanimljiviju i korisniju sliku od kritičkog osvrta na tekstove samih autora tog sustava.

2.2.1 Sačuvani prozori

X sustav implementira suprotnu paradigmu u kojoj prozorski sustav ne čuva sadržaje prozora koje je prekrio neki drugi prozor. Kad prozor (ili njegov dio) ponovno postane vidljiv sustav obavještava aplikaciju o tome i očekuje da ona ponovno iscrta "oštećeni" dio.

Da bi se u izbjegli neugodni vizualni artefakti koje ovakav pristup donosi, X sustav ima mogućnost specificiranja čuvanja sadržaja prozora ili sadržaja ekrana koji se nalazi ispod prozora (backing store i save under), ali s preporukom da se te opcije štedljivo koriste zbog čuvanja resursa na strani X poslužžitelja, pa su se aplikacije i grafičke biblioteke te preporuke i držale.

Unatrag zadnjih nekoliko godina se ta politika drastično izmijenila zbog dva razloga. Prvi je povećanje dostupnih resursa: očekivana količina memorije na raspolaganju X poslužitelju je drastično veća nego prije 20 godina. Drugi razlog je povećanje očekivanja od grafičkog sustava, uglavnom zbog zahtjeva za mogućnošću korištenja alfa kanala za transparenciju. Takva sposobnost zahtijeva iscrtavanje svakog prozora u nezavisni memorijske spremnik, te naknadno komponiranje tih spremnika, uzimajući u obzir alfa kanal na svakom od njih. Međutim, trenutne implementacije ne čuvaju te međuspremnike, pa prozorski sustav i dalje redovito zahtijeva ponovna iscrtavanja.

2.2.2 Fleksibilnost

Ovdje moramo napomenuti da je Goslingov "korisnik" zapravo programer, tj. osoba koja je sposobna implementirati vlastitu varijantu izbornika, kao što je tortni izbornik[20] (eng. *pie menu)*,

koji je za NeWS implementirao Don Hopkins, tada dio Goslingovog tima u Sunu. Takav model korisnika ne odgovara modelu korisnika na kojeg možemo naići u projektima s "uobičajenim ograničenjima prilikom razvoja proizvoda". Današnje shvaćanje konfigurabilnosti i fleksibilnosti ne zahtijeva od korisnika da najprije nauči kako sustav koji upotrebljava funkcionira na tehničkom nivou, da bi ga tek oboružan tim znanjem mogao (smisleno) konfigurirati, ali i NeWS i X su pisani za upravo takve korisnike.

Nivo konfigurabilnosti i fleksibilnosti se dosta razlikuje kod različitih komponenti sustava. Konfigurabilnost rada X poslužitelja se svodi na zadavanje opcija na komandnoj liniji, čime se može uključiti ili isključiti određena funkcionalnost za koju je predviđeno da bude uključena ili isključena, ali ništa više od toga. Kao primjer možemo navesti opcije za forsiranje čuvanje sadržaja svakog prozora i za forsiranje ignoriranja takvih zahtjeva od aplikacija.

Uključivanje prve opcije ne daje sustav koji zadovoljava prvi Goslingov cilj jer programer mora napisati kod i za slučaj da ova opcija nije uključena, ali poboljšava responsivnost s točke gledišta korisnika. U tom slučaju aplikacije neće dobijati zahtjeve za ponovnim iscrtavanjem (dijela) svojih prozora, pa neće biti ni treperenja koje često prati takvo iscrtavanje. Druga opcija je predviđena za računala s malom količinom memorije, s ciljem sprječavanja prozorskog sustava da potroši previše. Međutim, to je vrlo grub prekidač kojim se potrošnja memorije za čuvanje prozora limitira na nulu. Puno bi korisniji bio prekidač kojim bi se moglo specificirati koliko memorije X poslužitelj smije potrošiti u tu svrhu, ali takav prekidač nitko nije implementirao, mada nema tehničkih prepreka za njega.

Na X sustavu ne postoji jedna specifikacija korisničkog sučelja, niti samo jedna osnovna biblioteka koja služi za iscrtavanje primitivnih dijelova sučelja (tipki, izbornika, itd.), ali se to teško može nazvati fleksibilnošću, jer je najčešća zamjerka korištenju različitih biblioteka to što ih se ni na koji način ne može natjerati na isti izgled i ponašanje, pa samim time ne iskazuju dovoljnu količinu fleksibilnosti.

Popularnije biblioteke obično imaju mehanizme kojima se u detalje mogu specificirati boje, debljine linija ili pisma, ali te se mogućnosti uglavnom svode na to da se jednoj, već postojećoj varijabli promijeni vrijednost. Međutim, originalno zamišljeni mehanizam za tu svrhu (Xt resursi) je mijenjao ponašanje u raznim verzijama (iz X11R2 na X11R3, pa na X11R4), ali čak i bez toga je toliko kompliciran i zamršen da ga je u praksi rijetko tko uspješno koristio[6].

Moderne biblioteke za grafičko sučelje (npr. GTK+ i Qt) imaju mehanizam za mijenjanje tema, koji korisnicima omogućava jednostavnu promjenu izgleda, a ne zahtijeva nikakvo tehničko znanje. Na taj način je moguće izabrati temu koju je netko drugi napravio, a za sam razvoj teme je potrebno veće ili manje tehničko znanje, ovisno o tome što sve autor želi promijeniti. Međutim, i mehanizmi za izmjenu tema imaju svoje limite, pa obično ne posjeduju beskonačnu fleksibilnost.

Zbog modularnog dizajna je obično moguće dodavati nove grafičke elemente (kao što su tortni izbornici), ali je upitna mogućnost zamjene već postojećih, bez ikakvih izmjena u aplikacijama. Mogućnost bi teoretski mogla biti implementirana preko mehanizama za izmjenu teme, ali obično ovakve mogućnosti nisu među prioritetima.

Kod programa za ravnanje prozorima nailazimo na različite nivoe fleksibilnosti. Najviši teoretski mogući s točke gledišta korisnika-programera je ugradnja programskog jezika u takav program, što omogućava beskonačni nivo fleksibilnosti ako se tim programskim jezikom može upravljati svim funkcijama programa. Međutim, s točke gledišta korisnika koji nije programer, takav program nema nikakvu fleksibilnost jer za bilo kakvu promjenu zahtijeva njemu nedostupnu vještinu.

Najstariji takav program je GWM[21] (Generic Window Manager), baziran na Lisp interpreteru pomoću kojeg se moglo upravljati svim funkcijama upravljanja prozorima. Prva verzija je izašla 1989, a posljednja 1995. godine. GWM je bio i ostao program za hackere, i samim tim prilično marginalan produkt. Međutim, ista ideja integriranog jezika je našla put i u Sawfish[53], koji je namijenjen prosječnim korisnicima, a jedno vrijeme je bio i oficijelni ravnatelj prozora u okolini GNOME. Sawfish ima integriran svoj dijalekt Lispa u kojem se mogu programirati akcije na podražaje iz prozorskog sustava, ali ima i uobičajene konfiguracijske dijaloge, pa ne zahtijeva od korisnika poznavanje programiranja.

Za X sustav je napisano više od sto ravnatelja prozora[52] s različitim filozofijama, mogućnostima i konfiguracijskim opcijama. Od tog broja otprilike dvadesetak ravnatelja spada u stabilni i održavani software za svakodnevnu upotrebu.

Općenito gledajući, ovaj Goslingov cilj u svom pojašnjenju implicitno sadrži pretpostavku da je korisnik prozorskog sustava ujedno i programer zainteresiran za taj sustav, pa je ideja fleksibilnosti zapravo prilagođena potrebama i željama autora sustava. Ista ideja se nalazi i u osnovi X sustava, pa je i taj sustav rađen za korisnike koji najprije moraju naučiti kako on tehnički funkcionira da bi ga mogli uspješno koristiti. U novije vrijeme grafičke biblioteke za korisnička sučelja i pripadajuće

radne okoline (prvenstveno GTK/GNOME i Qt/KDE) pokušavaju pobjeći od te paradigme i učiniti sustav pristupačnijim običnim korisnicima, ali se njihovi napori većinom svode na kopiranje sučelja Microsoft Windowsa ili Mac OS X-a, a upitno je koliko se takvim metodama može postići.

2.2.3 Udaljeni pristup

X sustav je od samog začetka bio dizajniran kao distribuirani sustav gdje su udaljena računala bila potpuno ravnopravna i koristila isti protokol kao i lokalno računalo. Međutim, u praksi se pojavio problem s performansama aplikacija s udaljenih strojeva, pogotovo onih koji nisu na lokalnoj mreži. Kontinuirano poboljšavanje mrežne opreme je dovelo do velikog porasta mrežne propusnosti (bandwidtha), ali latencija se nije značajno smanjivala.

X je asinkroni protokol, što znači da aplikacija ne mora čekati na odgovor od X poslužitelja da bi nastavila slati zahtjeve, ali su API i implementacija Xlib-a (najniže biblioteke koju koriste sve ostale) takvi da komunikacija postaje sinkrona za zahtjeve koji moraju dobiti odgovor od poslužitelja. Često i biblioteke višeg nivoa doprinose ovom problemu, pogotovo ako su dizajnirane kao školski primjer objektno orijentiranog koda. Primjerice Java grafičko sučelje prije verzije 1.5 je bilo toliko sporo prilikom udaljenog rada da se često golim okom moglo vidjeti kako se elementi grafičkog sučelja iscrtavaju.

U praksi se pokazalo da tehnologije koje slijepo kopiraju sadržaj ekrana (npr. VNC[25]) ponekad korisniku daju bolju responsivnost od direktnog korištenja aplikacije. U ovim slučajevima velika mrežna propusnost uspije dati bolje rezultate od izrazito loše latencije. Postoji još jedan važan razlog za korištenje ovakvih tehnologija, a to je preusmjeravanje sadržaja prozora na drugi ekran (ili nastavljanje rada na istom, nakon što pukne mrežna konekcija).

Naime, X sustav nema mogućnost prebacivanja prikaza aplikacije s jednog X poslužitelja na drugi[10] (ili nastavljanja rada nakon pucanja i oporavka mrežne konekcije), pa korisnici s takvim potrebama moraju koristiti dodatne tehnologije. Pozitivna strana X sustava u ovom slučaju je što je na njegovoj arhitekturi izuzetno jednostavno implementirati takve tehnologije. Umjesto da se aplikacija spaja na X poslužitelj koji upravlja grafičkom karticom, aplikaciju se spoji na virtualni X poslužitelj koji iscrtava po memorijskom spremniku, pa onda taj sadržaj prosljeđuje dalje. Zbog toga na jednom X sustavu možemo imati neograničeni broj VNC poslužitelja, a na nekim drugim sustavima (npr. Microsoft Windowsima) samo jedan.

2.2.4 Grafičke primitive

X sustav je na svom početku (1987.) potpuno podbacio na ovom planu. Njegove grafičke primitive se svode na crtanje linija, pravokutnika, kružnica i poligona. Operacije sa slikama su bile izuzetno limitirane i zahtjevne za aplikacije, a biblioteke koje su dolazile s X sustavom nisu nudile nikakvu pomoć. Arhitekturalno je čitav sustav izvorno napravljen za monokromatske zaslone, a operacije s bojama su na brzinu nakalemljene naknadno. Za ilustraciju možemo navesti da X sustav garantira dostupnost samo monokromatskih površina za slike, a za bilo kakav rad s bojama se aplikacija mora prilagoditi mogućnostima koje oglasi X poslužitelj. Prve biblioteke s elementima za izradu korisničkih sučelja su čitav problem zaobišle i tipično nisu imale mogućnost prikaza kolor slike na ekranu, pa je čitav problem ostavljen aplikacijama.

Autori su predvidjeli da se skup operacija nad slikama doda kasnije u jednom od sljedećih izdanja sustava, ali je to donijela tek verzija X11R6.7 iz 2004. godine s ekstenzijom Render[7].

U međuvremenu je bilo nekoliko pokušaja da se popravi takvo stanje i razvoj aplikacija učini lakšim, ali većina dodataka (ekstenzija za X poslužitelj i grafičkih biblioteka) nije ušla u široku upotrebu jer su stvarali dodatni posao programerima aplikacija. Aplikacije, naime, moraju moći funkcionirati i u okolini u kojoj svaki pojedini dodatak ne postoji, pa dodaci koji su zahtijevali održavanje dvaju verzija koda nisu bili popularni.

Današnje moderne biblioteke za izradu korisničkih sučelja imaju mogućnosti rada s kolor slikama i boljim grafičkim primitivima od onih koje se nalaze u jezgri X protokola, te ne prebacuju taj posao na aplikacije.

2.2.5 Iskorištavanje sklopovlja

X sustav je dizajniran u vrijeme crno-bijele tehnologije, a naknadni pokušaji uvođenja kvalitetnijeg rada s bojama nisu najsretnije prošli. CMS (Color Management System) mogućnosti su dodane u Xlib u verziji X11R5 (1991.), ali taj dio biblioteke nikad nije bio u širokoj upotrebi i ne zadovoljava današnje potrebe. Službena X arhitektura još uvijek nema zamjenu ili poboljšanja, a u međuvremenu su u upotrebu ušla različita rješenja. Komercijalni Unix sustavi su obično imali razumno kvalitetna, ali međusobno nekompatibilna rješenja. Otvoreni kod trenutno ima kolekciju aplikacija i biblioteka koje uglavnom zadovoljavaju tehničke zahtjeve, ali još uvijek se radi na međusobnoj integraciji i jednostavnosti korištenja.

Možemo smatrati da je Gosling u ovoj točki naveo upravljanje bojama kao primjer mogućnosti sklopovlja koju prozorski sustav treba podržati. Isti zahtjev možemo primijeniti i na sve ostale mogućnosti sklopovlja. Na ovom području se X protokol pokazao dobro dizajniranim jer je omogućio jednostavno dodavanje novih mogućnosti, pa čak i onih koje su napravljene s namjerom da zamijene već postojeću funkcionalnost u jezgri sustava.

Međutim, sama podrška za različite nove vrste sklopovlja koje su se pojavile u dvadeset godina postojanja sustava je bila ambivalentna. S jedne strane su komercijalni Unix sustavi imali izvanrednu podršku za sklopovlje proizvođača zato što su se i sklopovlje i programska podrška dizajnirali i proizvodili na istom mjestu, pa nije bilo problema s integracijom. S druge strane je to rezultiralo proliferacijom nekompatibilnih biblioteka sa sličnom funkcionalnošću, što je izuzetno kompliciralo stvari za aplikacije koje su željele biti portabilne, pa takve biblioteke nikad nisu postale jako popularne.

Službene verzije X sustava godinama nisu dobijale nikakvu značajnu podršku za nove mogućnosti sklopovlja. Jedini značajni dodatak je Xinerama u verziji X11R6.4 (1998.) koja omogućava kombiniranje više fizičkih ekrana u jedan logički.

Situacija se počela mijenjati početkom stoljeća kad su dodane ekstenzije s podrškom za mogućnosti modernog sklopovlja[11] (Render - kompozicija slika i slova; RandR - promjena veličine i rotacije ekrana; XInputExtension 2.0 - podrška za ulazne uređaje, uključujući i hot-plug; Composite - podrška za iscrtavanje u spremnik koji se ne prikazuje na ekranu, što omogućuje kasniju manipulaciju, npr. dodavanje podrške za transparenciju).

2.2.6 Performanse

Službeni kod X sustava nikad nije bio brzinski šampion. Verzije koje je izdavao konzorcij su uvijek bile označene kao ogledne implementacije i vrlo su često bile implementirane kao dokaz koncepta, pri čemu su performanse uglavnom bile zapostavljene. Obično je to značilo da se isporučivala funkcionalnost bez akceleracije u sklopovlju. Istu situaciju imamo i danas. Nekoliko godina nakon dodavanja Render ekstenzije koja je postala glavni mehanizam za iscrtavanje, X.org poslužitelj je imao implementiranu akceleraciju za sklopovlje za svega nekoliko grafičkih kartica.

Poboljšanja performansi su se mogla naći u komercijalnim sustavima baziranim na slobodnoj distribuciji koda. Međutim, obično je postojao samo jedan proizvođač za jednu platformu, pa je

usporedba samih programskih rješenja bila gotovo neizvodiva. Takvo stanje otežava analizu performansi sustava jer je komercijalni kod uvijek bio zatvoren i distribuiran bez opisa promjena u odnosu na polazišnu verziju.

Standardni test za mjerenje performansi X sustava je `x11perf`, koji mjeri različite operacije jezgre protokola, ali te operacije više nisu relevantne. Danas je vrlo vjerojatno da će aplikacija većinu vremena provesti u neakceleriranom kodu ekstenzije Render. Osim toga, brzina rada na sustavu puno više ovisi o implementaciji grafičkih biblioteka na kojima su aplikacije bazirane, nego na performansama najnižeg nivoa sustava (X poslužitelja).

Asinkroni dizajn sustava poboljšava performanse u nekim slučajevima (jer se može poslati više grafičkih naredbi bez čekanja na izvršenje), ali u nekim drugim pogoršava iskustvo korištenja. Obrada događaja je asinkrona i uključuje obavještavanje aplikacije koja zatim mora poslati reakciju natrag. Na slabijem sklopovlju ili u vrijeme dok je procesor opterećen događa se da svi ti programi ne stignu dovoljno brzo reagirati, pa se korisniku dogodi neka vrsta neugodnosti, npr. ulaz s tipkovnice ode u krivi prozor. Današnja računala su dovoljno brza, pa se takvi efekti rijetko događaju, ali u 80-im i 90-im godinama prošlog stoljeća je to bila uobičajena pojava. Danas je još uvijek uobičajena prilikom rada s udaljenim aplikacijama.

X sustav je sustav za 2D grafiku, a 3D grafika je dodana naknadno, pomoću ekstenzija. Međutim, arhitektura X sustava je prespora za zahtjeve koji se uobičajeno postavljaju pred 3D grafičke aplikacije, pa su svi operativni sustavi nezavisno implementirali mogućnost da aplikacija direktno pristupa sklopovlju grafičke kartice i tako potpuno zaobiđe X poslužitelj. U praksi sklopovlju obično pristupaju grafičke biblioteke (npr. OpenGL), a ne sam aplikativni kod.

Takav pristup je podigao performanse 3D grafike po cijenu dodatne kompleksnosti unutar operativnog sustava. U takvoj arhitekturi OS mora arbitrirati između X poslužitelja i aplikacija koje direktno pristupaju grafičkom sklopovlju.

Ukupno gledajući, performanse X sustava nikad nisu bile izuzetno dobre, i zbog arhitekture i zbog implementacije te arhitekture. Pozitivni aspekt je da su se popravke i ispravke uspijevale implementirati ne narušavajući kompatibilnost sa starijim verzijama sustava.

2.3 GTK

Ranije smo objasnili da su za elemente grafičkog sučelja (tipke, izbornike, itd.) zadužene posebne

grafičke biblioteke, tzv. *widget toolkiti*[26]. Kroz povijest X sustava je napravljeno više biblioteka takve vrste, a najvažnije su:

- Athena, biblioteka koja se kao demo biblioteka distribuira uz X sustav. Prednost joj je velika raširenost - prisutna je na gotovo svakom X sustavu. Mane su joj limitiranost grafičkih elemenata i izuzetna ružnoća prikazanog na ekranu. Zbog toga je bilo puno pokušaja poljepšavanja grafičkih elemenata biblioteke Athena, ali ni jedan od tih pokušaja se nije uspio etablirati kao važna biblioteka za razvoj aplikacija.
- Motif[4], biblioteka koja je izašla kao pobjednik u GUI ratovima komercijalnih Unixa u 80-im godinama prošlog stoljeća (glavna konkurencija joj je bila Open Look biblioteka). Motif je bio de facto standard za grafičke aplikacije na komercijalnim sustavima Unix, Na njemu je izgrađen CDE (Common Desktop Environment), prva standardizirana radna okolina za Unix.
- GTK[23] (GIMP ToolKit) je originalno napravljen kao dio GIMP-a (GNU Image Manipulation Program), a zatim je izdvojen u posebnu biblioteku kako bi ju i drugi projekti mogli koristiti. Današnje verzije (2.24 i 3.4) su višeplatformske (podržavaju X11, Microsoft Windowse i Mac OS X), imaju vrlo dobro riješenu internacionalizaciju i pristupačnost za hendikepirane osobe. Postoji i mogućnost korištenja iz svih popularnijih programskih jezika.
- Qt, portabilna biblioteka za razvoj aplikacija koja sadrži i grafičko korisničko sučelje. Za razliku od prethodne tri biblioteke, ova biblioteka je implemetirana u C++-u uz dodatni generator koda, zvan MOC (Meta Object Compiler). Od samog početka je bila dvostruko licencirana: verzija za Windowse je bila dostupna pod komercijalnom licencom, a verzija za Unix i pod komercijalnom i pod GPL-om. Qt je osnova za radnu okolinu KDE.

Biblioteka GTK je nastala skoro slučajno, kao rezultat frustracije Motifom prilikom razvoja GIMP-a. Peter Mattis, izvorni autor većine njenog koda, ovako je objasnio motivaciju za razvoj te grafičke biblioteke[30]:

> Trebate razumjeti da GIMP i GTk nisu nastali zato da bismo popunili rupe na listi programa dostupnih pod GPL-om i LGPL-om. GIMP je pokrenut zato što sam htio napraviti web stranicu. GTk je pokrenuta zato što sam bio nezadovoljan s Motifom, a i zanimalo me kako se piše biblioteka za grafičko korisničko sučelje. To su potpuno sebični razlozi. Vjerojatno su

> zbog toga projekti toliko daleko dogurali i na kraju i uspjeli. Meni je jako teško dulje vrijeme raditi na bilo čemu iz potpuno nesebičnih razloga.

Gornji citat bi nas mogao usmjeriti prema zaključku da je GTK neozbiljan projekt zato što su njegovi autori u projekt ušli s izrazitom namjerom eksperimentiranja i učenja pisanja grafičkih biblioteka. Međutim, s mogućim izuzetkom Motifa, i autori svih ostalih grafičkih biblioteka ove vrste su bili u istoj situaciji i prilazili svom projektu s istom motivacijom. Jedina važna razlika je što će autori otvorenog koda to i reći, a autori zatvorenog koda tipično nemaju pravo reći bilo što za javnost, jer ih ugovori s njihovim korporacijama (a nerijetko i zakoni) u tome sprječavaju.

GTK verzija 1.0 se sastojala od tri biblioteke:

- GLib: biblioteka s korisnim C rutinama za rad sa stablima, hash tablicama, listama, stringovima i sl. Također sadrži sustav tipova koji implementira objektno-orijentiranu paradigmu u C-u, glavnu petlju za događaje itd.
- GDK (GIMP Drawing Kit): grafička biblioteka koja djeluje kao omotač oko grafičkih funkcija niskog nivoa. Sučelje je vrlo slično sučelju Xlib-a, a namjena ove biblioteke je stvaranje nivoa koji će omogućiti relativno jednostavno portiranje na druge platforme.
- GTK (GIMP ToolKit): grafička biblioteka koja implementira elemente korisničkog sučelja.

Od tada do danas se stanje nešto izmijenilo, ali gornje tri biblioteke su i dalje najvažniji dijelovi sustava. Glib biblioteka je izdvojena i distribuira se kao zaseban proizvod zato što je koriste i projekti koji nemaju nikakve veze s grafikom. Objektni sustav je izdvojen u biblioteku gobject, a dodana je i biblioteka gio koja implementira virtualni datotečni sustav.

Dodane su još i biblioteke ATK, Pango i GdkPixbuf. ATK (Accessibility Toolkit) donosi podršku za hendikepirane osobe, a Pango donosi podršku za slaganje i prikazivanje visokokvalitetnog multilingvalnog teksta. GdkPixbuf omogućava učitavanje i snimanje datoteka sa slikovnim formatima (PNG, JPEG, itd.), neke jednostavnije manipulacije sa slikama u memoriji, te iscrtavanje slika na ekran, uključujući i animaciju. GdkPixbuf je najprije bio nezavisni dodatak, a kad se API stabilizirao, biblioteka je uključena u GTK.

Prva verzija biblioteke GTK je radila samo na X sustavu, a kasnije je portirana i na druge grafičke sustave (Microsoft Windows i Mac OS X) i to onako kako je predviđeno originalnom arhitekturom: portiranjem GDK nivoa. Povijesno su za nas najznačajnije ove verzije:

- **1.0**, travanj 1998. — prva stabilna verzija.
- **2.0**, ožujak 2002. — biblioteka GObject izdvojena, puna podrška za Unicode i UTF-8.
- **2.8**, kolovoz 2005. — integracija s Cairom.
- **2.10**, srpanj 2006. — Podrška za ispis.
- **2.22**, rujan 2010. — GdkPixbuf postaje posebni modul, većina GDK iscrtavanja bazirana na Cairu.
- **3.0**, veljača 2011. — iscrtavanje isključivo preko Caira, novi mehanizam za teme baziran na CSS-u (Cascading Style Sheets).
- **3.2**, rujan 2011. — eksperimentalna podrška za Wayland i HTML5.
- **3.4**, ožujak 2012. — bolja podrška za Microsoft Windowse i OS X, poboljšani Wayland i CSS moduli.

GTK je grafička biblioteka na kojoj se temelji GNOME radna okolina. Različite GNOME biblioteke implementiraju funkcionalnost koja je potrebna aplikacijama, a povremeno se ta funkcionalnost prebaci u GTK, kad se pokaže da je implementacija stabilna i uspješna. Neuspješne biblioteke obično budu zamijenjene novim pokušajem, pa se ciklus ponavlja.

GTK inačice 1.x su bile stabilne verzije biblioteke za kreiranje grafičkih sučelja, ali su imale uobičajene nedostatke grafičkih aplikacija na X sustavu: lošu lokalizaciju, nemogućnost ispisivanja grafike na pisač, te grafički model koji je pratio (slabašne) grafičke mogućnosti X sustava.

Verzija 2.0 je donijela punu podršku za Unicode, a zatim i integraciju s bibliotekom Cairo, koja postaje glavni grafički mehanizam za iscrtavanje. Još uvijek je moguće koristiti stari GDK API za crtanje da bi se zadržala kompatibilnost s prijašnjim verzijama. S prelazom na verziju 3.0 koja više ne mora čuvati kompatibilnost sa serijom 2.x, stari API je izbačen i GTK iscrtava isključivo pomoću Caira.

Cairo[14, 15] je moderna biblioteka za iscrtavanje dizajnirana za pružanje visokokvalitetnog ispisa na pisaču pomoću istog programskog sučelja koje se koristi za iscrtavanje na ekranu. Sučelje biblioteke je vektorsko i ne ovisi o pikselima na ekranu. Postoji više izlaznih modula: Xlib i Xcb za X Window sustav, OpenGL, Microsoft Windows GDI, Mac OS X Quartz, Postscript, PDF, SVG, memorijski spremnik koji se može spremiti u datoteku (npr. u PNG formatu) itd.

Grafički model je bogat i moćan. Grafički model i osnovne operacije su preuzeti iz PostScripta. Dodane su primitive za komponiranje slika bazirane na Porter-Duffovoj algebri[16], gradijenti, maskiranje i rezanje (eng. clipping). Za grafičke primitive su birane one za koje postoji akceleracija u sklopovlju na modernim grafičkim karticama, što rezultira značajnim ubrzanjem (na nekim testovima i preko 100 puta[13]) u odnosu na neakceleriranu verziju.

GTK je bila prva značajnija biblioteka koja je uzela Cairo za grafičke operacije niskog nivoa[17]. Prije integracije su u GNOME okolini postojala tri sučelja za iscrtavanje: GDK (iscrtavanje na ekran), gnome-print (generiranje PostScript i PDF datoteka) i GnomeCanvas (platno za struktuiranu grafiku). Posljednja dva logički pripadaju biblioteci GTK, ali nije postojala dovoljno dobra implementacija, pa je funkcionalnost bila izdvojena u posebne biblioteke. U sljedećoj verziji nakon integracije s Cairom GTK je dobila podršku za ispis.

Za razliku od prijašnje implementacije u kojoj je GDK bio omotač oko funkcionalnosti Xlib-a, integracija s Cairom izlaže izvorno sučelje Caira programeru, uz manje od deset dodanih funkcija za povezivanje funkcionalnosti. U starom modelu je GDK sučelje za iscrtavanje imalo dvije funkcije: omotati Xlib sučelje da bi ga se učinilo ugodnijim za rad, te stvoriti nivo apstrakcije koji omogućava jednostavno portiranje na druge platforme.

Cairo sam za sebe podržava različite grafičke sustave (štoviše, značajno više njih nego GDK), pa se dodavanjem još jednog nivoa iznad Caira u tom pogledu ne bi ništa dobilo. Drugi razlog, udobnost korištenja sučelja, uvijek je jednim dijelom u oko promatrača, ali je očigledno da su GTK/GDK i Cairo programska sučelja izrađena u istom stilu, pa se ni tu ništa značajno ne bi dobilo omatanjem u sučelje GDK.

Prednosti izlaganja Caira su u lakšem odravanju GDK biblioteke (ne treba pisati ni dokumentirati omotač), bržem dovođenju nove funkcionalnosti dodane u Cairo do programera koji koristi GTK (jer ne treba čekati da se najprije napravi omotač na nivou GDK), lakša integracija s drugim bibliotekama koje koriste Cairo (jer GDK nema svoje apstrakcije i podatke, pa ne treba programirati mostove, nego se može direktno komunicirati preko Cairovih objekata).

Gore nabrojene prednosti su uglavnom prednosti za programere koji rade na biblioteci GTK. S točke gledišta programera koji samo upotrebljavaju GTK za pisanje svoje aplikacije, stvari su malo drukčije. Većina aplikacija tranziciju neće ni osjetiti zato što koriste gotove grafičke elemente i same ne iscrtavaju ništa. Aplikacije koje same iscrtavaju (a naša aplikacija je upravo takva) se

moraju portirati prilikom prijelaza s GTK 2.x na GTK 3.x.

3. PRPlib

API design is hard stuff

- *You won't get it all right*
- *You get to live with the mistakes for a long time*
- *Few tasks are harder than inventing names*

Carl Worth: Designing a Library that's Easy to Use[18]

PRPlib je programska biblioteka za **P**rikaz **R**ezultata **P**ostupaka računalnog vida, tema ovog rada, nazvana tako zato što je PBPRPRV bilo teško izgovarati.

PRPlib je modul za program Cvsh (Computer Vision SHell) kojem omogućava prikaz na X Window sustavu. Cvsh je ljuska koja omogućava jednostavno eksperimentiranje s algoritmima računalnog vida. Ljuska implementira funkcionalnost koja je potrebna svim modulima koji implementiraju algoritme: pribavljanje slike, iscrtavanje slike, te korisničko sučelje.

3.1 Programsko sučelje

Tamo gdje je zbog različitosti platformi potrebno imati poseban kod za svaku platformu cvsh ima implementirane generičke razrede koji pozivaju konkretne implementacije u pojedinim modulima. Glavni program definira sučelje koje moduli moraju implementirati. Cvsh je napisan u C++-u, pa i njegovi moduli moraju koristiti isti programski jezik. Za module koji iscrtavaju sliku sučelje izgleda ovako:

```
public:
    static char const* s_name();
    virtual char const* name();

public:
    // window properties (get)
    virtual int width();
    virtual int height();

    // window properties (set)
    virtual void position(int, int);
    virtual void size(int, int);
    virtual void caption(char const *);
```

```
public:
    // window output
    virtual void flash(); // beep sound

    // window input
    virtual bool event(int& iterator, win_event_abstract&) const;

private:
    // window output
    // img should be width() x height()
    void setImg(const img_wrap&);
    virtual void putImg(const img_wrap&);
    virtual void putImgAnn(const img_wrap&, const win_ann_abstract&);
    virtual void putAnn(const win_ann_abstract&);
```

Konstruktor (nije prikazan gore) otvara prozor u kojem se nakon toga može iscrtavati korištenjem metoda navedenih u sučelju.

Metode `name` i `s_name` vraćaju ime modula, što je u našem slučaju `gtk`. Metode `width` i `height` vraćaju širinu i visinu prozora u pikselima. Metoda `position` treba postaviti prozor na zadano mjesto na ekranu, metoda `size` zadaje veličinu prozora, a metoda `caption` postavlja ime prozora.

Metoda `flash` treba skrenuti pažnju korisnika, što se tipično postiže kratkim zvukom ili bljeskanjem prozora. Događaji iz prozorskog sustava (pritisnute tipke na tipkovnici i mišu) se spremaju u ciklički spremnik kojeg implementira ljuska, a metodom `event` se iz tog spremnika uzima zadnji događaj.

Metode za iscrtavanje (`putImg`, `putAnn` i `putImgAnn`) kao ulazne parametre dobijaju sliku i anotacije koje treba prikazati na ekranu. Slika može biti u jednom od (nedokumentiranih) formata koje podržava cvsh, a i očekuje se da će modul znati ispravno raditi s formatom koji će se tek dodati u budućnosti. To je moguće postići korištenjem konverzijske funkcije koju implementira cvsh, pa tako modul može dobiti sliku u formatu s kojim zna raditi.

Anotacije su lista primitivnih operacija koje treba iscrtati preko slike, a razred koji ih iscrtava treba implementirati sljedeće sučelje:

```
    // point drawing
public:
    virtual void drawPtDotImp(int x, int y);
    virtual void drawPtBoldImp(int x, int y, int sz);

    // string drawing
```

```
public:
    virtual void drawStrImp(int x, int y, char const*, int sz);

    // line drawing
public:
    virtual void drawLineImp(int xs, int ys, int xe, int ye, int width);
    virtual void drawLineDashedImp(int xs, int ys, int xe, int ye, int
width);

    // circle drawing
public:
    virtual void drawCircleImp(int x, int y, int radius, int width);
    virtual void drawCircleDashedImp(int x, int y, int radius, int
width);
    virtual void drawCircleFilledImp(int x, int y, int radius);

    // colours
public:
    virtual void colour(int i);
    virtual void grey(int i);
    virtual void bw(bool white);

    //music (optional)
public:
    virtual void playSound(int t, int f);
```

Postoje metode za iscrtavanje točke, pune i crtkane linije, pune i crtkane kružnice, te ispunjenog kruga. Postoje tri metode kojima se bira boja kojom se iscrtava iz jedne od preddefiniranih paleta. Metoda `colour` prima indeks za paletu boja, a metoda `grey` prima indeks za paletu sivih nijansi. Metodom `bw` se bira bijela boja. Metodom `drawStrImp` se ispisuje UTF-8 tekst na zadanoj poziciji.

S PRPlibove točke gledišta ova sučelja su zadana i nepromjenjiva jer postoje brojni moduli koji ih koriste. Za dodavanje nove funkcionalnosti je moguće dodati nove metode koje bi onda trebao implementirati glavni program i ostali moduli za iscrtavanje (trenutno postoji samo još jedan takav modul namijenjen Microsoft Windowsima).

3.2 Izbor grafičke biblioteke

Kad je cvsh originalno pisan na X Window sustavu nije postojala zadovoljavajuća biblioteka s korisničkim sučeljem. Kao što je u uvodu rečeno, biblioteke za izradu grafičkih sučelja su izbjegavale implementirati funkcionalnost za prikaz kolor slike na ekranu, što je ovoj aplikaciji bilo neophodno. Osim tog nedostatka, tada postojeće biblioteke su imale i druge probleme koji su priječili upotrebu: Athena ima neuobičajeno korisničko sučelje koje je korisnicima predstavljalo

problem, Motif je bio dostupan samo na komercijalnim Unix sustavima, biblioteka XForms je bila besplatna, ali ne i slobodna i nije se moglo predvidjeti hoće li i u budućnosti biti dostupna, a biblioteka GTK je još bila previše nestabilna.

Zbog toga je prvo sučelje za X Window sustav bilo napravljeno korištenjem samo biblioteka Xlib i Imlib. Biblioteka Imlib je rješavala prikaz slike na ekranu, a direktnim korištenjem Xlib-a je implementirano minimlno sučelje prema korisniku. Rezultat nije imao velike mogućnosti na području korisničkog sučelja, ali je bio funkcionalan i nije zahtijevao održavanje otprilike 10 godina.

U međuvremenu je autor Imliba uvidio da mu programsku sučelje biblioteke ne dozvoljava implementaciju nove funkcionalnosti, pa je napravio biblioteku Imlib2, a verziju 1.x proglasio zastarjelom i prestao razvijati. U nekoliko sljedećih godina su održavane aplikacije prešle s verzije 1 na verziju 2 i novije verzije operativnih sustava su polako počele izbacivati Imlib 1 iz programske podrške koju su distribuirale. Na nekim operativnim sustavima Imlib 1 više ne funkcionira bez preinaka, što je vrlo nesretna situacija za korisnike cvsh koji ujedno nisu i osposobljeni za programiranje na X Window sustavu.

Postojale su dvije opcije za dovođenje ovog modula u radno stanje:

1. Zadržavanje Xlib koda i zamjena biblioteke za prikazivanje slika,
2. Pisanje novog modula koji koristi neku od modernih biblioteka s grafičkim sučeljem.

Prva opcija je jednostavnija za izvedbu, ali je problem nalaženje adekvatne biblioteke. Kao prvi kandidat se nameće Imlib 2, ali nije baš jasno koji je status te biblioteke. Postoje aplikacije koje ju koriste i trenutno se isporučuje u operativnim sustavima koji su prestali isporučivati Imlib 1, ali izgleda da su autori napustili razvoj i umjesto nje razvijaju Evas, novu i značajno ambiciozniju biblioteku. Službene informacije o životnom vijeku biblioteke Imlib 2 ne postoje, pa se može dogoditi da i nju zadesi ista sudbina kao i Imlib 1.

Postoje i druge biblioteke koje sadrže potrebnu funkcionalnost, ali sve one su postale marginalne od kad su moderne biblioteke za kreiranje korisničkih sučelja (prvenstveno GTK i Qt) dobile podršku za iscrtavanje kolor slike na ekran, pa je i za njih upitno održavanje u budućnosti.

Za pisanje novog modula jedini su realni kandidati GTK i Qt. Cvsh ima jako male potrebe za svoju osnovnu funkcionalnost, pa je korištenje bilo koje grafičke biblioteke za korisničko sučelje moguće,

ali druge biblioteke imaju puno manje mogućnosti, pa s njima ne bismo besplatno dobili neke dodatne mogućnosti. Primjerice, GTK/Cairo kombinacija omogućava snimanje slike u PNG ili SVG formatu bez dodatnog programiranja jer sav kod za izradu tih formata sadrži Cairo.

Izbor između Qt i GTK biblioteka nije lagan i u praksi se obično svodi na osobne preference autora: oni kojima se sviđa C++ biraju Qt, a oni kojima se sviđa C biraju GTK. Od objektivnijih pokazatelja nam je najvažnija kompatibilnost novih verzija sa starim kodom, a GTK je tu bolja. Između verzije 1.0 i 2.0 je prošlo samo četiri godine jer je bilo nužno brzo popraviti goruće nedostatke prve verzije, ali je zato serija 2.x s punom kompatibilnošću unatrag trajala 10 godina. Serija 3.x bi također trebala potrajati desetak godina prije sljedeće nekompatibilne promjene.

Qt, s druge strane, svakih nekoliko godina izdaje novu seriju koja je nekompatibilna sa starom, što nije najsretnije rješenje za cvsh.

Zbog gore navedenih razloga je za implementaciju odabrana biblioteka GTK. PRPlib je implementiran u dvije verzije, jednom koja koristi stari GDK API i drugom koja koristi Cairo API za iscrtavanje. Prva verzija se može koristiti s čitavom 2.x serijom biblioteke GTK, a druga od verzije 2.8 nadalje i s čitavom serijom 3.x.

3.3 Arhitektura i izvedba PRPliba

3.3.1 Višestruki tokovi izvođenja

Osim gore opisanog sučelja PRPlib mora podržavati i rad s višestrukim tokovima izvođenja, s ogradom da glavni program specificira da će jednom prozoru pristupati samo jedan tok, pa ne moramo zaključavati pristup podacima u pojedinom razredu.

S obzirom da koristimo GTK, nemamo preveliku arhitekturalnu slobodu jer GTK podrazumijeva da će se koristiti petlja za obradu događaja iz te biblioteke. PRPlib prilikom prvog pozivanja konstruktora prozora pokreće tok koji inicijalizira GTK za rad s višestrukim tokovima i tijekom čitavog života programa izvršava glavnu petlju.

Glavnu GTK petlju izvršava GLib, biblioteka koja je potpuno sigurna za rad s višestrukim tokovima. Ona automatski štiti pristup svojim podacima, osim pristupa elementima struktura podataka koji se ne štite automatski zbog poboljšavanja performansi.

Biblioteka GTK nije sigurna za rad s višestrukim tokovima, ali ih je svjesna u smislu da postoje

pravila kojih se programer mora pridržavati da bi program ispravno funkcionirao. Biblioteka GTK ima jedan veliki globalni mutex kojim se upravlja pomoću funkcija `gdk_threads_enter` i `gdk_threads_leave`. Tim mutexom se međusobno isključuju različiti tokovi programa.

Svaki poziv GTK ili GDK funkcija mora biti unutar bloka koji počinje pozivom funkcije `gdk_threads_enter` i završava pozivom funkcije `gdk_threads_leave`, osim ako se radi o povratnoj (eng. callback) funkciji koju poziva GTK da bi obradila događaj. U tom slučaju će mutex već biti zaključan i aplikativni kod ga ne mora dirati. U slučaju da registriramo neku od povratnih funkcija koje poziva GLib (a ne GTK ili GDK), moramo sami implementirati zaključavanje i otključavanje u povratnu funkciju.

Iz gornjeg opisa je očigledno da GTK kod može izvršavati samo jedan tok, pa se više procesora ne može iskoristiti za paralelni rad. Međutim, sve to vrijedi samo za rad na sustavu X11. Na Microsoft Windowsima GTK uopće ne funkcionira u programima s višestrukim tokovima, pa PRPlib tamo nećemo moći koristiti. GTK verzija 3.4 ima podršku za Mac OS X, ali dokumentacija taj OS ne spominje, pa je nepoznato hoće li PRPlib na njemu funkcionirati.

3.3.2 Glavno grafičko sučelje

PRPlib ima jako jednostavan zadatak: otvoriti prozor, u njemu prikazati sliku, te preko te slike iscrtati jednostavne grafičkih primitive i ispisati tekst.

Za to od GTK elemenata koristimo razrede `GtkWindow` i `GtkDrawingArea`. `GtkWindow` implementira glavni prozor aplikacije, a `GtkDrawingArea` nam daje prostor na kojem ćemo iscrtati sliku i anotacije. PRPlib reagira na četiri događaja iz prozorskog sustava: *realize*, *expose*, *keypress* i *buttonpress*.

Prozorski sustav šalje događaj *realize* kad se stvori prozor, a prije nego što se prikaže na ekranu. Funkcija koja obrađuje taj događaj tipično alocira resurse koji ovise o karakteristikama prozorskog sustava. PRPlib u toj funkciji alocira boje, a verzija koja za iscrtavanje koristi sučelje GDK tu alocira i pisma.

Događaj *expose* dolazi prilikom prvog prikazivanja prozora na ekran, a kasnije svaki put kad se otkrije dio prozora koji je bio prekriven drugim prozorom, pa aplikacija mora ponovo iscrtati sadržaj jer ga prozorsku sustav nije zapamtio.

Događaji *keypress* i *buttonpress* dolaze kad korisnik pritisne tipku na tipkovnici ili mišu. PRPlib u

ovim slučajevima taj događaj sprema u ciklički spremnik u skladu s konvencijama koje zahtijeva glavna aplikacija. Postoji manje razmimoilaženje između GDK implementacije i originalne cvsh implementacije rada s mišem. Originalna cvsh implementacija prepoznaje samo jednostruki i dvostruki pritisak na tipku miša, dok GDK prepoznaje i trostruki. Kako ne postoji mogućnost signaliziranja trostrukog pritiska, PRPlib u tom slučaju signalizira dvostruki pritisak praćen jednostrukim.

Postoje još neki događaji koji se mogu pojaviti (npr. kad korisnik zatvara prozor), ali je obrada koju GTK implementira sasvim zadovoljavajuća, pa PRPlib ne mora implementirati dodatnu obradu.

Svi ovi događaji pozivaju funkcije u toku programa koji izvršava glavnu petlju. Prozori i sve ostale GTK funkcije za kreiranje sadržaja u prozoru se pozivaju iz jednog od radnih tokova aplikacije, a funkcije koje obrađuju ove događaje se pozivaju iz toka u kojem se izvršava glavna petlja, pa zbog toga treba sinkronizirati pristup podacima.

Svi podaci koji opisuju jedan prozor i njegov sadržaj se nalaze u varijablama razreda `win_gtk`. Kako je pristup tim varijablama (osim u trivijalnim slučajevima postavljanja i čitanja) uvijek povezan s pozivima GTK funkcija koje zahtijevaju korištenje velikog globalnog mutexa, nema smisla koristiti mutex za svaku instancu razreda. pa za sinkroniziranje pristupa podacima razreda koristimo veliki GTK-ov mutex.

3.3.3 Sučelja za iscrtavanje

Biblioteke GTK verzija 2.x (za x >= 8) imaju dva sučelja za iscrtavanje: GDK funkcije koje su uglavnom tanki omotač za funkcionalnost Xlib-a i sučelje Cairo. GDK funkcije crtaju direktno po objektu tipa `GdkDrawable`, koristeći grafički kontekst tipa `GdkGC`. Objekti tipa `GdkDrawable` mogu biti prozori, te monokromatske ili kolor slike. Grafički kontekst sadrži informacije o bojama pera i pozadine, tipu linija (puna ili iscrtkana), i sl. Oba ova objekta postoje u X poslužitelju i klijent strana (aplikacija) ima samo reference na njih.

U Cairo svijetu se crta po površini tipa `cairo_surface_t`, koja može biti prozor u prozorskom sustavu, PostScript ili PDF dokument, memorijski spremnik iz kojeg će biti generirana datoteka u nekom od slikovnih formata (npr. JPEG, PNG ili SVG) itd. Iscrtava se korištenjem objekta tipa `cairo_t` koji sadrži pokazivač na površinu, te opcije za iscrtavanje koje sadrže iste ili slične podatke kao i grafički kontekst na X sustavu.

Cairo ima puno bogatiji repertoar grafičkih primitiva od onoga što nam je na raspolaganju u GDK-ovom skupu, ali ovaj modul treba samo nekoliko jednostavnih operacija (točke, linije i kružnice) za koje u obje biblioteke postoje vrlo slične funkcije. Za usporedbu možemo pokazati metode koje crtaju liniju u obje implementacije.

GDK implementacija:

```
GdkDrawable *drawable;
GdkGC *gc;

void drawLineImp(int xs, int ys, int xe, int ye, int width)
{
    gdk_gc_set_line_attributes(gc, width, GDK_LINE_SOLID,
                               GDK_CAP_BUTT, GDK_JOIN_ROUND);
    gdk_draw_line(drawable, gc, xs, ys, xe, ye);
}
```

Cairo implementacija:

```
cairo_t *cr;
double on_off_dash[2] = { 4.0, 4.0 };

void drawLineImp(int xs, int ys, int xe, int ye, int  width)
{
    cairo_set_line_cap(cr, CAIRO_LINE_CAP_BUTT);
    cairo_set_line_join(cr, CAIRO_LINE_JOIN_ROUND);
    cairo_set_dash(cr, on_off_dash, 0, 0);
    cairo_set_line_width(cr, (double)width);
    cairo_move_to(cr, (double)xs, (double)ys);
    cairo_line_to(cr, (double)xe, (double)ye);
    cairo_stroke(cr);
}
```

Cairo djeluje malo verboznije zato što se svaki atribut mora postaviti pozivanjem posebne funkcije, dok GDK tipično ima funkciju kojom može postaviti više atributa za jedan objekt, u ovom slučaju liniju. Postoje dvije važne razlike. Cairo koristi racionalne brojeve za specifikaciju koordinata i debljine linije (u gornjem primjeru je pored njih napisan inače nepotreban cast da bi se to naglasilo), a GDK koristi cjelobrojne. GDK, kao i X protokol, za crtanje koristi model piksela na ekranu, gdje je svaki piksel mali pravokutnik koji ima svoje cjelobrojne koordinate.

Cairo, s druge strane, koristi PostScript model[5] u kojem su koordinate uređeni parovi realnih brojeva koji lociraju točku unutar Kartezijevog koordinatnog sustava na trenutnoj stranici.

Koordinate se specificiraju u korisničkom prostoru i uvijek imaju isti odnos prema trenutnoj stranici. S druge strane, svaka izlazna površina (u Cairo terminologiji) ili uređaj (u PostScript terminologiji) ima svoj koordinatni sustav (eng. device space) koji odgovara karakteristikama uređaja. Cairo automatski prebacuje koordinate iz korisničkog prostora u prostor uređaja i tako omogućuje da isti program producira (otprilike) isti izlaz na različitim medijima.

Druga razlika je što GDK funkcije imaju trenutni efekt, tj. odmah iscrtatavaju sliku, a u Cairo svijetu se gradi putanja (eng. path) koja se iscrtava upotrebom jedog od finalizacijskih operatora: `cairo_stroke` (iscrtava konturu) ili `cairo_fill` (ispunjava konturu). U Cairovom sučelju postoji još operatora, ali nam za implementaciju PRPliba nisu bili potrebni.

Da bismo crtali po prozoru potreban nam je odgovarajući `cairo_t` objekt kojeg dobijemo funkcijom `gdk_cairo_create`, a koja prima `GdkDrawable` kao argument. GTK dokumentacija ne specificira je li moguće istovremeno ili sekvencijalno koristiti oba sučelja za crtanje po istom prozoru.

3.3.4 Iscrtavanje

Postoje dva izvora iz kojih PRPlib može dobiti nalog za iscrtavanjem slike na ekranu. Jedan je direktni zahtjev aplikacije kad poziva jednu od metoda za iscrtavanje slike ili anotacija, a drugi je obavijest prozorskog sustava da je određeni dio prozora postao vidljiv, što znači da tamo treba iscrtati sliku. Stari cvsh grafički modul, kao i PRPlib implementacija koja koristi GDK sučelje jednostavno crta s dva mjesta u programu, pozivajući iste funkcije. Međutim, pokušaj implementacije iste logike i s Cairovim sučeljem je naišao na grešku u biblioteci GTK. Ta se greška može zaobići, ali njeno postojanje zorno pokazuje da se ne može računati da će nepreporučeni načini korištenja biblioteke GTK raditi bez problema.

Kod iscrtavanja različitih grafičkih elemenata dolazi do treperenja zato što se povremeno primitivni elementi iscrtavaju jedni preko drugih. Da bi se izbjeglo treperenje GTK na GDK nivou koristi dvostruke međuspremnike. Predviđeno je da aplikacija iscrtava u spremnik koji se trenutno ne prikazuje na ekranu, a kad završi s iscrtavanjem GDK nalaže prozorskom sustavu da prikaže novi međuspremnik i tako se izbjegava treperenje.

Jedino mjesto na kojem GDK može znati kad je iscrtavanje gotovo je funkcija koja obrađuje *expose* događaj, pa GTK dokumentacija zbog toga preporučuje iscrtavanje samo unutar te funkcije. Zbog

veće udobnosti i jednostavnosti programiranja GTK prilikom pozivanja *expose* povratne funkcije automatski zamijeni međuspremnike, a elementi koji to ne žele mogu za sebe isključiti takvo ponašanje. To moraju napraviti elementi koji crtaju izvan *expose* funkcije, jer bi u suprotnom dijelovi slike završavali u različitim međuspremnicima.

Prva implementacija PRPliba koja je koristila Cairo je isključila dvostruke međuspremnike i pokušavala crtati s dva mjesta u programu (kao i implementacija koja je koristila GDK), ali se ništa nije pojavljivalo na ekranu. Pokazalo se da je problem u tome što se objekt `cairo_t` kreirao unutar povratne funkcije *realize*. To je normalno mjesto za kreiranje objekata čiji je životni vijek vrijeme trajanja prozora, ali u ovom slučaju program nije funkcionirao na očekivani način.

Zaobilazno rješenje problema je implementirano tako da su napravljene dvije funkcije za obradu događaja *expose*. Prva funkcija kreira objekt `cairo_t` i spremi ga za kasnije korištenje. Zatim registrira drugu funkciju, a deregistrira sebe, te na kraju pozove drugu funkciju i vrati njen rezultat. Druga funkcija radi normalno iscrtavanje ekrana. Iscrtavanje po objektu `cairo_t` dobivenom ovom metodom je rezultiralo prikazom slike na ekranu, ali očigledno se radi o grešci u GTK biblioteci jer ovo nije i ne može biti normalni način stvaranja objekata koji će se koristiti tijekom cijelog života prozora.

3.3.4.1 Crtanje u expose funkciji

Sljedeća verzija PRPliba je crtala samo u *ekspose* funkciji, ali se pojavio problem sinkronizacije između dretvi u programu. Glavna aplikacija bi u svom toku izvršavanja pozvala metodu PRPliba i naložila iscrtavanje slike. Pozvana metoda bi spremila pokazivače na sliku i anotacije u svoje lokalne varijable, te zatim generirala GTK zahtjev za ponovnim iscrtavanjem prozora. Glavna GTK petlja, koja se izvršava u svojom posebnom toku bi zapremila ove zahtjeve i iscrtavala prozor.

Međutim, kako nije bilo nikakve eksplicitne sinkronizacije, događalo se da aplikacija pošalje nekoliko slika za redom, a da ih GTK propusti prikazati na ekranu, vjerojatno zato što tok u kojem se izvršava iscrtavanje nije dobio procesorsko vrijeme.

Postoje dvije metode sinkronizacije. Prva je korištenje GDK funkcije koja sinkrono poziva funkciju za obradu događaja *expose*. Druga metoda je eksplicitno implementiranje sinkronizacije u aplikativnom kodu. Implementirane su obje metode i vremena izvršavanja testnog programa su vrlo slična.

Prva metoda se sastoji od dodavanja poziva funkcije `gdk_window_process_updates` u metode `putImg`, `putAnn` i `putImgAnn`. Ta funkcija će odmah pozvati odgovarajuću funkciju za iscrtavanje.

Druga metoda je kompleksnija. Razred `win_gtk` je proširen cjelobrojnom varijablom `imgdrawn` koja pamti je li *expose* funkcija nacrtala sliku na ekran i uvjetnom varijablom `cnd` koja služi za signalizaciju među tokovima programa. Za signalizaciju u POSIX okruženju su nam potrebni mutex i uvjetna varijabla, pa bismo mogli upotrijebiti veliki GDK mutex za ovu svrhu, ali nam biblioteka GDK to ne dozvoljava jer je njoj taj mutex implementacijski detalj.

Naime, GDK sučelje za rad s višestrukim tokovima programa sadrži samo funkcije i nema podržani način da se dođe do adrese mutexa koji koristi. Srećom, postoji mogućnost da aplikativni kod dostavi svoje funkcije za međusobno isključivanje GDK tokova, pa PRPlib to i radi. Kod izgleda ovako:

```
#ifdef CREATE_GDK_MUTEX
pthread_mutex_t big_gdk_mutex = PTHREAD_MUTEX_INITIALIZER;

static void
impl_gdk_lock(void)
{
    pthread_mutex_lock(&big_gdk_mutex);
}

static void
impl_gdk_unlock(void)
{
    pthread_mutex_unlock(&big_gdk_mutex);
}
#endif

static void *
gtk_thread_start(void *arg)
{
    ...
#ifdef CREATE_GDK_MUTEX
    gdk_threads_set_lock_functions(impl_gdk_lock,impl_gdk_unlock);
#endif
    g_thread_init(NULL);
    gdk_threads_init();
    gdk_threads_enter();
    ...
```

Nakon ove egzibicije možemo bez problema koristiti `big_gdk_mutex` zajedno s uvjetnim varijablama iz `win_gtk` razreda. U toj varijanti koda *expose* funkcija iscrta sadržaj prozora, postavi

varijablu `imgdrawn` na 1 i nakon toga signalizira preko varijable `cnd`. Metode `putImg`, `putAnn` i `putImgAnn` na samom početku zaključaju veliki GDK mutex, a zatim, ukoliko je `imgdrawn` nula, čekaju na uvjetnoj varijabli. Nakon što obave svoj posao postavljanja varijabli u internim strukturama postavljaju `imgdrawn` natrag na nulu.

To će osigurati da se dvije metode za iscrtavanje ne izvrše ako se između njih nije izvršila funkcija *expose* koja sliku stvarno i iscrtava na ekran.

Obje varijante sinkronizacijskog koda su ostale i u konačnoj verziji PRPliba. Ukoliko se kod prevede s opcijom `-DCREATE_GDK_MUTEX`, PRPlib će kreirati svoj mutex i koristiti sinkronizaciju pomoću uvjetne varijable. Prevođenje bez te opcije će koristiti sinkrono pozivanje funkcije za iscrtavanje.

3.3.4.2 Dvostruki međuspremnici

Nakon što smo osigurali da se grafika iscrtava samo iz *expose* funkcije, možemo dozvoliti GDK okolini korištenje dvostrukih međuspremnika. Time nestaje treperenje na ekranu koje je bilo vidljivo u prethodnim verzijama, uz nešto slabije performanse.

Ova verzija se iz prethodne dobija izbacivanjem koda koji se brinuo da postoji isti objekt tipa `cairo_t` tijekom čitavog života prozora, te malim rearanžiranjem preostalog koda koji ga koristi.

3.3.4.3 GTK 3.x

Za prelazak na GTK 3 je bilo potrebno još nešto portiranja. Izvorni kod je poprskan određenom količinom `#ifdef` direktiva, što PRPlibu omogućuje korištenje obje glavne verzije biblioteke GTK, ali smanjuje čitljivost programa.

GTK 3 ima više nekompatibilnih promjena u odnosu na GTK 2[19], a za PRPlib su bile relevantne ove:

- Zamjena strukture GdkColor sa strukturom GdkRGBA. Ovim se upravljanje bojama prebacuje s X poslužitelja na X klijenta.
- Zamjena modula za rad s temama. Nova biblioteka je dobila novi i puno moćniji modul za rad s temama. PRPlib od te funkcionalnosti treba podrazumijevano pismo za ispis tekstualnih anotacija i funkciju za iscrtavanje fokusa.
- Promjena imena i argumenata događaja za iscrtavanje slike. Novi se događaj zove *draw* i

prima objekt tipa `cairo_t` koji je već napravljen i iskonfiguriran. U verziji 2.x su programi morali sami napraviti taj objekt.

Nakon promjena na ova tri područja dobili smo izvorni kod koji se može prevesti i s jednom i s drugom verzijom biblioteke.

3.3.5 Performanse

Gore opisane različite verzije PRPliba imaju i različite performanse. Naš testni program otvara dva prozora i u svakom od njih sto puta iscrtava sliku i jednostavne anotacije (dva piksela, dvije linije i tri kratka termina), te mjeri koliko mu je vremena trebalo za to. Mjera koju program ispisuje je broj prikazanih slika u sekundi.

Gore smo opisali sedam verzija PRPliba, od kojih šest radi ispravno. Neispravna verzija ne iscrtava sve slike zbog nepostojanja sinkronizacije, pa ćemo ju izostaviti iz mjerenja. U donjoj tablici se navodi zbog potpunosti.

Rbr.	Naziv	Opis
1.	GDKDraw	Iscrtava korištenjem GDK API-ja koji se 1:1 prevodi u Xlib pozive. Može koristiti samo GTK 2.
2.	CairoDraw	Koristi Cairo. Iscrtava iz glavnog programa i expose funkcije.
3.	CairoExposeNoSync	**Neispravan.** Koristi Cairo, iscrtava samo iz expose funkcije, ali bez sinkronizacije, pa "guta" slike.
4.	CairoExposeForceSyncDraw	Koristi Cairo, iscrtava samo iz expose funkcije, upravljačke metode za iscrtavanje sinkrono pozivaju expose funkciju.
5.	CairoExposeSyncPthread	Koristi Cairo, iscrtava samo iz expose funkcije koja se s upravljačkim metodama sinkronizira pomoću lokota.
6.	CairoExposeDBufForceSyncDraw	Koristi Cairo i dvostruke međuspremnike, iscrtava samo iz exppose funkcije, upravljačke metode sinkrono pozivaju expose funkciju
7.	CairoExposeDBufSyncPthread	Koristi Cairo i dvostruke međuspremnike, iscrtava samo iz expose funkcije koja se s upravljačkim metodama sinkronizira pomoću lokota.

Tablica 1: Testirane varijante PRPliba

Sva mjerenja su provedena na notebooku HP 8710p pod operativnim sustavom Ubuntu 11.10. Procesor je Intel Core2 Duo koji sadrži dvije jezgre i omogućava minimalnu količinu paralelizma.

Grafička kartica je Nvidia Quadro NVS 320M za koju proizvođačevi driveri omogućavaju ubrzavanje u sklopovlju. X poslužitelj je X.Org 1.10.4.

Svaku smo verziju programa pokrenuli sedam puta. Prvo pokretanje je služilo da bi se sav potreban kod učitao u memoriju, pa smo njegov rezultat odbacivali. Ovdje navodimo prosjek preostalih pet mjerenja.

Svaku verziju smo, ukoliko je to bilo moguće, testirali s bibliotekama GTK 2.24.6 i GTK 3.2.0. U tablici 2 su rezultati izvršavanja neoptimiziranog koda. Primarna namjena ljuske cvsh je razvoj postupaka računalnog vida, pa možemo očekivati da će se kod prevoditi bez optimizacije dok traje razvoj.

Br.	**Naziv**	**FPS s GTK 2.24.6**	**FPS s GTK 3.2.0**	**Treperenje**	**Napomena**
1.	GDKDraw	75.4	—	Da	Može koristiti samo GTK 2.x.
2.	CairoDraw	56.4	56.8	Da	
3.	CairoExposeNoSync	—	—	—	Neispravan.
4.	CairoExposeForceSyncDraw	54.2	50.6	Da	
5.	CairoExposeSyncPthread	52	52.2	Da	
6.	CairoExposeDBufForceSyncDraw	48.8	45.8	Ne	
7.	CairoExposeDBufSyncPthread	49.8	48.4	Ne	

Tablica 2: Brzina prikaza neoptimiziranog koda (više je bolje)

Vidimo da velika razlika postoji u izboru tehnologije za iscrtavanje (stari GDK API nasuprot Cairu). Verzije s najljepšim prikazom na ekranu bez treperenja su očekivano sporije od prethodnih verzija koje koriste Cairo, ali to usporenje nije toliko da bismo se morali odreći dvostrukih međuspremnika zato što to previše usporava programe.

Zanimljivo je primijetiti da je kod sinkronizacijskih algoritama s GTK 2 bibliotekom bolje upotrijebiti njeno sinkronizirano iscrtavanje, a s bibliotekom GTK 3 je bolje sinkronizaciju implementirati u aplikaciji. Ovaj rezultat je vjerojatno posljedica mladosti biblioteke GTK 3 koja još uvijek nije bila podvrgnuta ozbiljnom optimizacijskom postupku.

U tablici 3 donosimo rezultate izvršavanja optimiziranih programa.

Br.	Naziv	FPS s GTK 2.24.6	FPS s GTK 3.2.0	Treperenje	Napomena
1.	GDKDraw	133.6	—	Da	Može koristiti samo GTK 2.x.
2.	CairoDraw	167	162.4	Da	
3.	CairoExposeNoSync	—	—	—	Neispravan.
4.	CairoExposeForceSyncDraw	131.3	108.7	Da	
5.	CairoExposeSyncPthread	131.3	130.3	Da	
6.	CairoExposeDBufForceSyncDraw	113/155	100/152	Ne	
7.	CairoExposeDBufSyncPthread	114/155	100/154	Ne	

Tablica 3: Brzina prikaza optimiziranog koda (više je bolje)

Posljednja dva retka imaju po dva broja zato što ove varijante programa konzistentno iskazuju dva različita ponašanja. Manji broj dobijamo kad proces uzme oko 50% procesorske snage (dakle jednu od dvije postojeće jezgre). Veći broj dobijamo kad proces uzme oko 75% procesorske snage, što znači da koristi obje jezgre odjednom. Računalo na kojem se testiranje provodi je pri tome potpuno neopterećeno, pa možemo zaključiti da razliku uzrokuje operativni sustav i njegovi algoritmi za dodjelu procesora.

Općenito primjećujemo da su optimizirane verzije više nego dvostruko brže (čak i kad gledamo manji od brojeva u posljednja dva retka). Staro GDK sučelje za iscrtavanje više nema značajnu prednost, čak ni kad gledamo manje brojeve u zadnja dva retka. Verzija 3.2 biblioteke GTK je i dalje nešto sporija od verzije 2.24.

3.3.6 Komunikacija s ostalim dijelovima programa cvsh

Prije upotrebe GTK se mora inicijalizirati pozivom funkcije `gtk_init` čiji bi ulazni argumenti trebali biti pokazivači na `argc` i `argv` argumente funkcije `main`, tj. polje argumenata s komandne linije. GTK inicijalizacijska funkcija parsira polje argumenata tražeći standardne GTK argumente, inicijalizira se u skladu s njima i uklanja ih iz polja, tako da glavna aplikacija ništa o svemu tome ne mora znati. Ove opcije su navedene u GTK dokumentaciji, a uglavnom se radi o opcijama predviđenima za ispravljanje grešaka u programima, pa bi ih bilo zgodno podržati.

Međutim, cvsh nema sučelje kojim bi mogao prenijeti opcije s komandne linije do svog modula, pa PRPlib ovu funkcionalnost trenutno ne može iskoristiti. Bilo bi poželjno da glavni program dobije

takvu mogućnost, što bi grafičkom modulu omogućilo da implementira i neke svoje opcije. Trenutno se za implementaciju korisničkih opcija mogu koristiti varijable okoline, ali to je daleko od prijateljski nastrojene okoline za korisnika. Idealno bi bilo da se PRPlibove opcije vide kad se glavni program pozove s opcijom `--help`.

3.4 Korisničko sučelje

Do sada opisani PRPlib moduli nemaju nikakvo vidljivo korisničko sučelje. Stari modul za prikaz je omogućavao snimanje slike u BMP ili EPS formatu pritiskom na tipku "d" (dump), što nije baš očigledno i intuitivno sučelje. Međutim, taj je modul implementiran na golom Xlib-u i bilo kakvo grafičko sučelje bi iziskivalo preveliki napor.

S obzirom da je novi modul implementiran na biblioteci za izradu grafičkih sučelja, napor potreban za izradu grafičkog sučelja je značajno manji, pa smo implementirali izbornik s dvije funkcije: snimanjem slike u datoteku u jednom od podržanih formata (PNG, PS, EPS, PDF i SVG), te isporučivanje slike drugim aplikacijama putem ekranskog međuspremnika (eng. clipboard).

Zanimljivo je napomenuti da je prilikom izrade ovog dijela funkcionalnosti program počeo ispisivati greške na nivou X protokola, kakve se često događaju kod direktnog korištenja Xlib-a, ali su izuzetno rijetke kod korištenja viših biblioteka kao što je GTK. Ispostavilo se da su greške kombinacija dvaju faktora. Prvi je da razvojni sustav (Ubuntu 11.10) isporučuje XCL/XCB kombinaciju umjesto Xlib-a, mada za XCB njegovi autori tvrde da još nije dovoljno istestiran da bi bio spreman za produkcijsku upotrebu.

Drugi faktor, i izvor grešaka, je da biblioteka GDK prilikom inicijalizacije rada za višestruke tokove programa ne koristi funkciju `XInitThreads`, što Xlib za taj slučaj zahtijeva, nego inicijalizira svoj mutex i uopće ne koristi međusobno isključivanje tokova programa na nivou Xlib-a. Pretpostavljamo da ovaj pristup funkcionira ako se stvarno i koristi Xlib (mada je izuzetno prljav i neispravan), ali ne funkcionira kad se koristi XCL/XCB kombinacija.

Nakon što smo u kod PRPliba dodali poziv funkcije `XInitThreads`, svi problemi su nestali.

3.5 Budući rad

Poželjno je implementirati neke od ideja koje smo susreli u opisu srodnih programa. Ovdje ćemo nabrojiti one koje nam se čine najvažnijima.

3.5.1 Grafičko sučelje u glavnom prozoru

OpenCV ima mogućnost dodavanja jednog ili više klizača (eng. slider) u svaki prozor kojim upravlja. Klizač prima adresu varijable i adresu funkcije. Ukoliko te adrese nisu NULL, prilikom promjene položaja klizača će se nova vrijednost upisati u varijablu, a zatim će se pozvati i funkcija.

Ovim se elementom omogućuje korisniku da u realnom vremenu upravlja varijablom koja bi trebala upravljati radom programa.

Spremanje nove vrijednosti varijable nam ne predstavlja problem. Ovo bi omogućilo glavnom programu da pogleda vrijednost varijable kad je njemu zgodno (tipično prije obrade sljedeće slike). S obzirom da različiti tokovi programa pišu i čitaju istu varijablu, potrebno je sinkronizirati pristup, što se može napraviti jednostavnim sučeljem koje će sakriti (obično neportabilne) implementacijske detalje.

Međutim, poziv povratne funkcije nije jednostavan zato što bi iz jednog toka programa (eng. threada) trebalo pozvati povratnu funkciju u drugom toku. Na Unix sustavima to možemo implementirati na dva načina. Prvi je eksplicitna sinkronizacija među tokovima programa, što je kompleksno, podložno greškama koje je izuzetno teško otklanjati i uglavnom nepotrebno troši procesorsko vrijeme na provjeravanje signalizacije.

Druga metoda je slanje signala iz jednog toka izvršavanja drugome, što će (razumno brzo) pozvati proceduru za obradu u drugom toku, ali ta procedura smije koristiti samo funkcije koje su sigurne za asinkrono korištenje (eng. async-signal-safe), što izuzetno limitira mogućnosti. Obično se u ovakvim rutinama samo upiše vrijednost u varijablu čiju vrijednost glavni tok periodično provjerava. Međutim, takvim načinom upotrebe ne dobijamo ništa u odnosu na implementaciju u kojoj PRPlib sprema vrijednost varijable na zadanu memorijsku lokaciju.

Ovaj bismo problem mogli riješiti uz malu nadogradnju mogućnosti jezgre operativnog sustava. Naime, POSIX sučelje za rad s višestrukim tokovima programa definira točke u kojima se tok može ukinuti (eng. cancellation point). U tim točkama bi se mogle pozvati rutine za obradu signala koje bi smjele pozivati bilo kakve funkcije (a ne samo async-signal-safe) zato što je stanje toka dobro definirano. Međutim, ovakva mogućnost ni na jednom operativnom sustavu iz Unix porodice nije na raspolaganju.

To znači da se u eventualnim budućim implementacijama moramo ograničiti na postavljanje varijable i odustati od pozivanja povratne funkcije. Octave, Matlab i OpenCV imaju takve

mogućnosti zato što ne koriste višestruke tokove, pa se povratna funkcija poziva iz istog toka programa, za što nema tehničkih poteškoća. Pored toga, Octave i Matlab korisniku pružaju viši programski jezik čija implementacija može koristiti primitive za sinkronizaciju među tokovima programa, a da to bude potpuno transparentno za korisnika.

3.5.2 Ispisivanje matematičkih formula

Octave ima lijepu mogućnost slaganja matematičkih formula korištenjem LaTeX notacije. PRPlib koristi biblioteku Pango za ispis teksta, pa time dobija jako dobre mogućnosti rada s Unicodeom, ali ne i za slaganje matematičkih formula, jer Pango nema podršku za to.

Nadamo se da će jedna od budućih verzija PRPliba dobiti ovakvu mogućnost.

4. Zaključak

Pokazali smo da možemo upotrijebiti biblioteku upravljanu događajima da bismo izgradili biblioteku koja korisnika štiti od takve paradigme. Arhitekturalno je to izvedeno pokretanjem posebnog toka izvođenja programa (eng. threada), tako da taj tok ne utječe na ostale tokove koji pripadaju glavnoj aplikaciji.

Time smo dobili biblioteku koja zadovoljava osnovne zahtjeve navedene u uvodu (jednostavnost korištenja, perzistenciju sadržaja prozora i odgođenu obradu događaja), a izgubili mogućnost jednostavnog pozivanja povratnih funkcija. Moguće ih je pozivati ako glavna aplikacija i ova biblioteka koriste uobičajene primitive za sinkroniziranje (npr. mutexe i uvjetne varijable), ali takva implementacija ima previše negativnih strana (kompleksnost i vrlo teško otklanjanje grešaka), pa se ne isplati.

Upotreba gotove biblioteke za izradu korisničkih sučelja značajno skraćuje vrijeme razvoja programa, pa smo brzo i jednostavno implementirali rudimentarno korisničko sučelje za snimanje datoteka. Međutim, gotove biblioteke nemaju uvijek zadovoljavajuću fukcionalnost, pa sučelje nudi dvije mogućnosti za snimanje EPS formata. Prva poziva Cairovu implementaciju, a druga implementaciju koja je napravljena posebno za cvsh. Razlika je u tome što cvsh implementacija generira PostScript koji čovjek kasnije može razumjeti i obraživati u tekstualnom editoru, a Cairova implementacija generira PostScript koji nije *user friendly*.

Recentni dodaci u Window sustav (prvenstveno ekstenzije Render, Composite i XInputMethod 2.0,

te biblioteka Cairo) su donijeli nove mogućnosti na niskom nivou. Korisnici tog nivoa su obično biblioteke za izradu korisničkih sučelja, a korisnici tih biblioteka su aplikacije namijenjene živim korisnicima. Nove verzije biblioteka za izradu korisničkih sučelja implementiraju nove mogućnosti, ali ih u tome limitiraju njihova stara sučelja i potreba za održavanjem kompatibilnosti.

Zbog toga takve biblioteke obično započinju novu seriju inačica, nekompatibilnu sa starom, pri čemu se razvoj na staroj seriji prekida. Stare verzije biblioteka i aplikacija i dalje jednako dobro funkcioniraju, ali su aplikacije koje žele koristiti novu funkcionalnost (koja ne mora biti povezana s novim mogućnostima X sustava) prisiljene na portiranje da bi mogle koristiti nove verzije biblioteka.

PRPlib, biblioteka opisana u ovom radu, za izradu korisničkog sučelja koristi biblioteku GTK. Kod je napisan tako da omogući korištenje GTK serija 2.x i 3.x. Nadamo se da će serija 3.x potrajati sljedećih deset godina, te da u tom periodu neće biti potrebe za portiranjem.

Na jednostavnom primjeru smo pokazali da je biblioteka Cairo uz dobru optimizaciju nije nužno sporija od GDK sučelja (koje je tanki omotač za Xlib). Međutim, brzina će uvijek ovisiti o operacijama koje se izvode, a naš testni program nije relevantna mjera brzine za svaki program koji će se izvršavati unutar ljuske cvsh.

5. Literatura

1. James Gosling, 1985, SunDew - A Distributed and Extensible Window System, *Methodology of Window Management, Proceedings of the Alvey Workshop at Cosener's House, Abingdon*

   http://www.chilton-computing.org.uk/inf/literature/books/wm/p005.htm

2. Valerie Quercia, 1990, X Window System user's guide, *O'Reilly & Associates*

   http://archive.org/details/xwindowsystem03quermiss

3. Robert W. Scheifler, James Gettys, 1992, X Window System, *Digital Press*

4. Antony Fountain, Jeremy Huxtable, 2002, Motif Programming manual (vol 6A) 3rd ed., *O'Reilly & Associates*

   http://www.ist.co.uk/motif/books/vol6A/

5. Adobe Systems Incorporated, 1990, PostScript Language Reference Manual, 2nd ed.,

Addison-Wesley Publishing Company

6. Don Hopkins, 1994, The X-Windows Disaster, *UNIX-Haters Handbook, IDG books*

   http://www.art.net/studios/Hackers/Hopkins/Don/unix-haters/x-windows/disaster.html

7. Keith Packard, 2001, Design and Implementation of the X Rendering Extension, *Usenix Conference 2001*

   http://keithp.com/~keithp/talks/usenix2001/

8. Jim Gettys, 2003, Open Source Desktop Technology Road Map

   http://web.archive.org/web/20080413140042/http://people.freedesktop.org/~jg/roadmap.html

9. Jim Gettys, Keith Packard,2004, The (Re)Architecture of the X Window System, *Ottawa Linux Symposium 2004*

   http://keithp.com/~keithp/talks/xarch_ols2004/

10. Jim Gettys, 2002, The Future is Coming: Where the X Window System Should Go, *Usenix Conference 2002*

    http://static.usenix.org/events/usenix02/tech/freenix/gettys.html

11. Keith Packard, 2004, Getting X Off The Hardware, *Ottawa Linux Symposium 2004*

    http://keithp.com/~keithp/talks/xserver_ols2004/

12. Jamey Sharp, 2004, How Xlib is Implemented (And What We're Doing About It), *Usenix Conference 2004*

    http://static.usenix.org/events/usenix04/tech/freenix/full_papers/sharp/sharp_html/index.html

13. Peter Nilsson, David Reveman, 2004, Glitz: Hardware Accelerated Image Compositing Using OpenGL, *USENIX Conference 2004*

    http://static.usenix.org/events/usenix04/tech/freenix/nilsson.html

14. Carl D. Worth: Cairo, 2005, Making Graphics Easy to Print, *linux.conf.au 2005*

    http://cworth.org/~cworth/papers/cairo_lca2005/

15. Carl Worth, Keith Packard, 2003, Cairo: Cross-device Rendering for Vector Graphics, *Ottawa Linux Symposium 2003*

http://cworth.org/~cworth/papers/xr_ols2003/

16. Thomas Porter, Tom Duff, 1984, Compositing Digital Images, *Computer Graphics Volume 18, Number 3, pp 253-259*

http://keithp.com/~keithp/porterduff/

17. Owen Taylor, 2004, The future of rendering in GNOME, *GUADEC 5*

http://people.redhat.com/otaylor/guadec5/

18. Carl Worth, 2006, Designing a Library that's Easy to Use, *GUADEC 2006*

http://cworth.org/~cworth/papers/guadec_2006/

19. Benjamin Otte, 2010, Rendering Cleanup, *stranica bloga*

http://blogs.gnome.org/otte/2010/07/27/rendering-cleanup/

20. Don Hopkins, 1997, Pie Menus, *web stranica*

http://www.art.net/studios/Hackers/Hopkins/Don/piemenus/index.html

21. Colas Nahaboo, 1996, GWM: The X11 Generic Window Manager WWW page, *web stranica*

http://web.archive.org/web/20070213041643/http://koala.ilog.fr/gwm/

22. X Window System, *članak na Wikipediji*

http://en.wikipedia.org/wiki/X11

23. GTK+, *članak na Wikipediji*

http://en.wikipedia.org/wiki/Gtk

24. NeWS, *članak na Wikipediji*

http://en.wikipedia.org/wiki/NeWS

25. Virtual Network Computing, *članak na Wikipediji*

http://en.wikipedia.org/wiki/Vnc

26. Widget toolkit, *članak na Wikipediji*

http://en.wikipedia.org/wiki/Widget_toolkit

27. History of the graphical user interface, *članak na Wikipediji*

http://en.wikipedia.org/wiki/History_of_the_graphical_user_interface

28. Event-driven programming, *članak na Wikipediji*

http://en.wikipedia.org/wiki/Event-driven_programming

29. Miro Samek, 2003, Who Moved My State?, *Dr. Dobb's Journal*

http://www.drdobbs.com/cpp/184401643

30. Stig Hackvän, 1999, Where did Spencer Kimball and Peter Mattis go?, *LinuxWorld Magazine*

http://web.archive.org/web/19990417052141/http://www.linuxworld.com/linuxworld/lw-1999-01/lw-01-gimp.html

31. X.org Foundation, 2002, Xlib - C Language Interface (X11R7 edition)

http://www.x.org/releases/X11R7.6/doc/libX11/specs/libX11/libX11.html

32. X.org Foundation, 2004, X Window System Protocol (X11R6.8 edition)

http://www.x.org/releases/X11R7.6/doc/xproto/x11protocol.html

33. GNOME Library, GTK+ 3 Reference Manual, *web izdanje*

http://developer.gnome.org/gtk3/

34. GNOME Library, GTK+ 2 Reference Manual, *web izdanje*

http://developer.gnome.org/gtk/

35. GNOME Library, GDK 3 Reference Manual, *web izdanje*

http://developer.gnome.org/gdk3/

36. GNOME Library, GDK 2 Reference Manual, *web izdanje*

http://developer.gnome.org/gdk/

37. GNOME Library, GDK-Pixbuf Reference Manual, *web izdanje*

http://developer.gnome.org/gdk-pixbuf/

38. GNOME Library, GLib Reference Manual, *web izdanje*

    http://developer.gnome.org/glib/

39. GNOME Library, Pango Reference Manual, *web izdanje*

    http://developer.gnome.org/pango/

40. ZetCode, Cairo Graphics Tutorial, *web izdanje*

    http://zetcode.com/tutorials/cairographicstutorial/

41. Davyd Madeley, 2005, Writing a Widget Using Cairo and GTK+2.8, *The GNOME Journal, web izdanje*

    http://www.gnomejournal.org/article/34/writing-a-widget-using-cairo-and-gtk28

42. cairographics.org, Cairo Reference Manual, *web izdanje*

    http://www.cairographics.org/manual/

43. cairographics.org, Frequently Asked Questions, *web izdanje*

    http://www.cairographics.org/FAQ/

44. David R. Butenhof, 1997, Programming with POSIX Threads, *Addison-Wesley*

45. opencv.org, OpenCV 2.4.0 Documentation, *web izdanje*

    http://docs.opencv.org/

46. INRIA Lagadic, ViSP General overview, *web stranica*

    http://www.irisa.fr/lagadic/visp/general-overview.html

47. INRIA Lagadic, ViSP API documentation v2.6.1, *web izdanje*

    http://www.irisa.fr/lagadic/visp/documentation/visp-2.6.1/classes.html

48. John W. Eaton, 2011, GNU Octave, *web izdanje*

    http://www.gnu.org/software/octave/doc/interpreter/

49. Bill Spitzak, 2011, FLTK 2.0 Documentation, *web izdanje*

    http://www.fltk.org/doc-2.0/html/index.html

50. Bart Massey, Jamey Sharp, 2001, XCB: An X Protocol C Binding, *Proceedings of the XFree86 Technical Conference 2001*

http://www.linuxshowcase.org/2001/full_papers/massey/massey.pdf

51. Jamey Sharp, Bart Massey, 2002, XCL: An Xlib Compatibility Layer for XCB, *Usenix Conference 2002*

http://www.usenix.org/events/usenix02/tech/freenix/sharp.html

52. Matt Chapman, 2012, Window Managers for X, *web stranica*

http://xwinman.org/

53. Sawfish - Window Manager, *Wiki*

http://sawfish.wikia.com/wiki/Main_Page

54. Alan Cooper, 2004, The Inmates Are Running the Asylum, *Sams Publishing*