Priručnik za korišćenje grafičke biblioteke KBGI

dr Kalman Babković

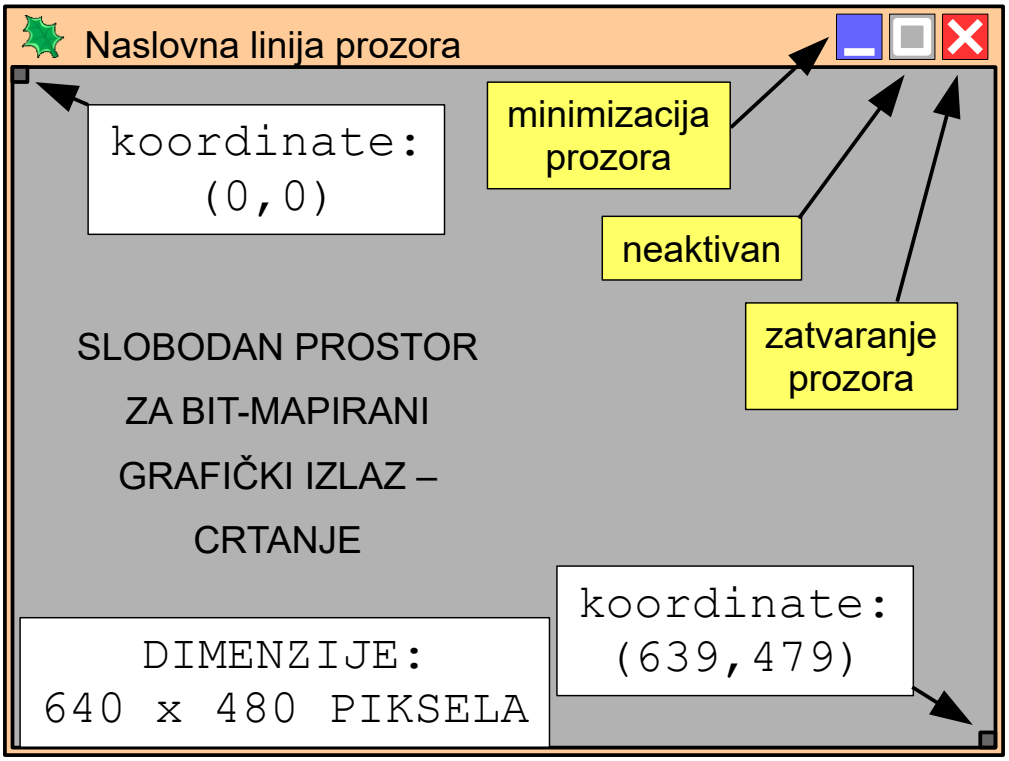
Uvod

Grafička biblioteka KBGI (**K**lasični **B**it-map orijentisani **G**rafički **I**nterfejs) namenjena je pisanju programa u jeziku C koji koriste grafiku. Prilagođena je operativnom sistemu Windows i kompajlerskom sistemu MinGW. Drugačiji sistemi trenutno nisu podržani.

Opšte informacije o radu sa KBGI bibliotekom

Koncept rada sa KBGI bibliotekom

Iako je namena KBGI biblioteke da programima učini dostupnim grafički izlaz visoke rezolucije, sam program koji biblioteku koristi i dalje može da koristi konzolu kao ulazno/izlazni uređaj. Grafički izlaz se ostvaruje preko nezavisnih prozora koji se ni po čemu ne razlikuju od ostalih standardnih prozora Windows okruženja. Jedan takav prozor šematski je prikazan na sledećoj slici:



Iako operativni sistem Windows omogućuje da prozori budu proizvoljnih dimenzija (čak i veći od samog ekrana) prozori u čijoj klijentskoj zoni (sivo polje na slici) se pojavljuje grafički izlaz KBGI biblioteke su fiksnih dimenzija. Zapravo veličina klijentske zone je ono što je fiksnih dimenzija koje iznose 640 piksela po horizontalnoj dimenziji i 480 po vertikalnoj. Prozor je moguće pomerati po ekranu, minimizovati ga, pa čak i zatvoriti kao i svaki drugi prozor Windows okruženja. Jedino nije moguće promeniti mu dimenzije, pa je kao posledica toga i ikolica za proširenje prozora na ceo

ekran neaktivna.

Crtanje unutar klijentske zone je **bit-map** orijentisano (nasuprot vektorski orijentisanom). To znači da se svaki novi nacrtani element crta preko već postojećih elemenata. Ako se preko nekih elementa nacrta novi, taj novi element prekriva stare, a prekriveni elementi se trajno gube. Drugim rečima, moguće je direktno kontrolisati boju svakog od 640 x 480 elemenata slike (tačke) koji se uobičajeno nazivaju **pikselima** (*engl. picture element*).

Svaki piksel u klijentskoj zoni prozora određen je svojim koordinatama. Koordinata po horizontalnoj osi (x) raste s leva na desno pa su pikseli sa x koordinatom 0 uz levu ivicu prozora, dok su pikseli sa x koordinatom 639 uz desnu ivicu prozora. Nasuprot uobičajenoj praksi u matematici, vertikalna osa (y) je orijentisana prema dole pa ova koordinata raste od gore prema dole – pikseli sa y koordinatom 0 su uz gornju ivicu prozora, a sa y koordinatom 479 uz donju.

Istovremeno je moguće otvoriti do `WINDOW_MAX` nezavisnih prozora na ekranu. Ova konstanta u aktuelnoj verziji biblioteke ima vrednost 10.

Pre bilo kakve operacije crtanja neophodno je **otvoriti** prozor. Svaki prozor ima svoj `prozor_id` – ceo broj između 0 i (`WINDOW_MAX-1`) kojim se prozor jednoznačno identifikuje. Sve operacije za crtanje imaju `prozor_id` kao svoj parametar i na osnovu toga se zna koji prozor je cilj neke grafičke operacije.

Korisnik može u bilo kom trenutku da zatvori bilo koji od prozora klikom na odgovarajuću ikonicu bez obzira šta se u programu koji je prozor otvorio dešava. Operacije crtanja usmerene ka prozoru koji je zatvoren će biti ignorisane. **Važno:** zatvaranje prozora neće automatski značiti da se i izvršavanje programa završilo, utvrđivanje da li je prozor još uvek na raspolaganju je potpunosti u nadležnosti autora programa koji koristi KBGI biblioteku.

Uključivanje heder fajla

U sve programske module koji koriste KBGI biblioteku potrebno je uključiti heder fajl `kbgi_lib.h`. To znači da pre poziva ijedne od funkcija biblioteke u programskom modulu treba da se nalazi sledeća linija:

```
#include "kbgi_files/kbgi_lib.h"
```

Linija programa u primeru pretpostavlja da se heder fajl zajedno sa ostalim relevantnim fajlovima biblioteke nalazi u poddirektorijumu `kbgi_files` u odnosu na fajl sa izvornim kodom modula u kojem se ova linija nalazi. Naravno, fajlovi mogu i drugačije da se organizuju, ali to neće biti razmatrano u ovom uputstvu.

Proces linkovanja sa bibliotekom

U poslednjoj fazi kompajliranja programa – linkovanju – programski kod svih modula programa se spaja sa programskim kodom biblioteka dajući jedan izvršni program. U ovoj fazi je neophodno da kompajler bude podešen da pronađe arhivske fajlove koji te biblioteke sadrže. U slučaju KBGI biblioteke, ti fajlovi su sledeći:

- `libkbgi_lib.a` – standardna verzija biblioteke,
- `libkbgi_libD.a` – debug verzija biblioteke.

Konačna verzija programa treba da koristi standardnu verziju, a verzija sa kojom će se možda obavljati debagovanje – izvršavanje korak po korak – treba da koristi **debug** verziju. Debug verzija može da ima nešto lošije performanse u radu tako da je nikako ne treba koristiti za konačnu verziju

programa.

Budući da KBGI biblioteka koristi grafičku biblioteku Windows operativnog sistema (**GDI** – Graphics Device Interface), potrebno je za linkovanje navesti i ovu biblioteku. U slučaju MinGW kompajlerskog sistema ova se biblioteka nalazi u arhivskom fajlu `libgdi32.a`.

KBGI biblioteka ima i nekoliko grafičkih elemenata sadržanih u resursnom fajlu, pa je potrebno prilikom linkovanja priključiti i tzv. resursni fajl koji nosi ime `resource.o`. Izostavljanje resursnog fajla neće biti katastrofalno po rad programa, ali će konačnom programu nedostajati neki estetski elementi poput ikonice i specijalnog kursora.

Komunikacija sa KBGI bibliotekom

Komunikacija sa bibliotekom se obavlja pozivanjem njenih funkcija. Njihov opis sledi u narednim odeljcima.

Vrednosti koje vraćaju funkcije KBGI biblioteke

Skoro svaka funkcija biblioteke vraća `int` vrednost. Ta vrednost je jednaka konstanti `KBGI_SUCCESS` ako je komanda prosleđena pozivom funkcije uspešno izvršena. U suprotnom vrednost je jednaka konstanti `KBGI_ERROR`.

Inicijalizacija biblioteke

Pre poziva ijedne funkcije iz biblioteke KBGI neophodno je izvršiti inicijalizaciju biblioteke. To se postiže pozivom funkcije **`K_kbgi`**. Ovo je dovoljno uraditi samo jednom u okviru nekog programa. Ako program ima više modula, dovoljno je da samo jedan pozove ovu funkciju. Ukoliko se ova inicijalizacija ne obavi, nijedna funkcija biblioteke neće uspešno obaviti svoj zadatak.

Utvrđivanje verzije biblioteke

Ako je iz nekog razloga neophodno da program utvrdi sa kojom verzijom KBGI biblioteke radi, na raspolaganju je funkcija **`K_Ver`**. Funkcija ima dva parametra, oba su pokazivači na promenljive tipa `unsigned int`. Nakon poziva ove funkcije, ove dve promenljive će sadržati redom veliki i mali broj verzije (u tekućoj verziji, to su brojevi 1 i 2).

```
unsigned int major_v, minor_v;
K_Ver(&major, &minor);
```

Operacije s prozorima

Neposredno nakon inicijalizacije nijedan prozor ne postoji, a samim tim nijedna grafička operacija neće imati efekta. Kreiranje novog prozora naziva se *otvaranjem prozora*, a postiže se pozivom funkcije **`K_OpenWindow`**. Funkciji se prosleđuje `prozor_id` koji je ceo broj u opsegu od 0 do 9. Drugi parametar je običan C string čiji sadržaj će biti prosleđen kao *naslovna linija prozora*.

```
K_kbgi();   // globalna, jednokratna inicijalizacija

// otvaranje prozora sa prozor_id-om 3
uspeh = K_OpenWindow(3, "Grafik funkcije");
if(uspeh == KBGI_ERROR)
   printf("Otvaranje prozora neuspesno!\n");
```

Funkcija vraće `KBGI_SUCCESS` ako je prozor uspešno otvoren, a `KBGI_ERROR` u slučaju greške.

Pri otvaranju prozora ne postoji mogućnost kontrole gde će se prozor nalaziti na ekranu, odnosno radnoj površini. Pozivom funkcije **K_MoveWindow** moguće je pomeriti prozor na željeno mesto. Funkciji se prosleđuju nove koordinate gornjeg levog ugla prozora u odnosu na gornji levi ugao desktopa izražene u pikselima.

Kada više nema potrebe za prozorom on se može zatvoriti pozivom funkcije **K_EndWindow**. Zatvaranje prozora najčešće nije neophodno eksplicitno uraditi jer je uobičajeno program koncipirati tako da se on završava onda kada korisnik zatvori prozor. Takođe, ukoliko se program završi (povratak iz funkcije `main`) bez zatvaranja prozora, operativni sistem će automatski zatvoriti sve prozore koje tom programu pripadaju. Sledeći fragment programa pomera prozor sa brojem 4, a potom ga zatvara.

```
K_MoveWindow(4, 100, 100);
K_EndWindow(4);
```

Status prozora

Pošto je svaki od prozora nezavisan od glavnog toka programa, korisnik može da zatvori prozor kad god želi. Iako pokušaj crtanja u prozor koji je zatvoren (ili još nije otvoren) neće imati nikakve negativne posledice osim što operacija neće biti izvršena, potreban je neki mehanizam da program utvrdi postoji li još uvek određeni prozor. Funkcija **K_WindowExists** vraća vrednost različitu od nule ako prozor još uvek postoji. Sledeći fragment programa u beskonačnoj petlji crta neki sadržaj unutar prozora sve dok on postoji (pozivom hipotetičke funkcije `Crtnje_i_Pauza`). Nakon što korisnik zatvori prozor, iz beskonačne petlje se izlazi (a logično je da se i ceo program završi).

```
while(1)
{
   if(K_WindowExists(3))
        Crtanje_i_Pauza();
   else
        break;
}
```

Operacije čekanja

U interakciji sa korisnikom brzina računara je prevelika pa su čekanja od strane računara, tj. programske pauze uobičajena praksa. KBGI biblioteka nudi rešenja za to.

Čekanje da korisnik zatvori prozor postiže se funkcijom **K_WaitForClose** ili **K_WaitAllClose**. Prva funkcija čeka da se određeni prozor (`prozor_id` se prenosi kao parametar) zatvori, dok druga čeka da se zatvore svi prozori koje je program otvorio. Funkcija nakon poziva obustavlja rad programa sve dok se odgovarajući prozor ili prozori ne zatvore, a program se nastavlja potom.

```
K_WaitForClose(4);// nastavlja se nakon zatvaranja proz. 4
```

Moguće je i čekanje određeno vreme – funkcija **K_Wait** kao parametar ima broj koji određuje dužinu čekanja u milisekundama. Nakon isteka tog vremena program nastavlja sa radom.

```
K_Wait(500); // programska pauza u trajanju od 0.5 s
```

Operacije crtanja

Glavna svrha biblioteke je da se upravlja sadržajem klijentske površine otvorenih prozora, tj. da se

unutar njih prikaže željeni grafički sadržaj. To se postiže pomoću operacija crtanja.

Upravljanje pojedinačnim pikselima

Svakoj pojedinačnoj tački na površini za prikaz, tj. pikselu može da se zada boja pozivanjem funkcija **`K_Dot`**. Funkciji se prosleđuje `prozor_id` prozora u kom se piksel od interesa nalazi, koordinate tačke i na kraju i izabrana boja. Boja je određena odgovarajućim brojem, tj. konstantom koje su date u tabeli boja u dodatku B.

```
K_Dot(0, x, y, KBGI_YELLOW);
```

U primeru se tačka na radnoj površini prozora 0 čije su koordinate određene vrednostima promenljivih `x` i `y` se dovodi na žutu boju.

Zadavanje boje grafičkih elemenata

Postoje 3 odvojena podešavanja za boju i ona su lokalna za svaki jedan otvoreni prozor. Neposredno nakon otvaranja prozora, sva tri podešavanja su postavljena na belu boju. Svaka boja se može izabrati iz palete od 16 unapred definisanih boja (date su u dodatku B).

- Boja za crtanje linija i okvira pravougaonika i elipsi – **Draw Color** – postavlja se funkcijom **`K_SetDrawColor`**.
- Boja za popunjavanje pravougaonika i elipsi – **Fill Color** – postavlja se funkcijom **`K_SetFillColor`**.
- Boja za ispis teksta – **Text Color** – postavlja se funkcijom **`K_SetTextColor`**.

Sve funkcije kao parametar imaju `prozor_id` i novu boju koja se bira.

```
// sve postavke boja u primeru se odnose na prozor br. 2
K_SetDrawColor(2, KBGI_YELLOW); // bira zutu boju za crtanje
K_SetFillColor(2, KBGI_BLUE); // bira plavu boju za popunjavanje
K_SetTextColor(2, KBGI_WHITE); // bira belu boju za tekst
```

Crtanje složenijih grafičkih elemenata

Biblioteka KBGI omogućuje ispis tri vrste geometrijskih elemenata:

- linije
- pravougaonici
- elipse

Pravougaonici i elipse mogu biti i popunjeni i prazni. Ako su prazni onda unutrašnji deo elementa ostaje neizmenjen na ekranu, tj. u tom prostoru ostaje netaknut sadržaj koji se već nalazio na tom mestu. Prazan element crta se bojom izabranom za **Draw Color**. Unutrašnji deo popunjenih elemenata se crta bojom izabranom za **Fill Color**.

Popunjeni pravougaonici se crtaju pozivom funkcije **`K_Rectangle`**, a prazni pozivom funkcije **`K_Box`**. Prvi parametar je `prozor_id` prozora u kojem se element iscrtava, a zatim slede četiti celobrojne vrednosti: x i y koordinata gornjeg levog ugla, a potom x i y koordinata donjeg desnog ugla.

Popunjene elipse se crtaju pozivom funkcije **`K_FilledEllipse`**, a prazne pozivom funkcije **`K_Ellipse`**. Prvi parametar je `prozor_id` prozora u kojem se element iscrtava, a zatim slede x

i y koordinata centra elipse, a potom poluprečnik u pravcu x-ose i poluprečnik u pravcu y-ose.

Linije se crtaju zadavanjem koordinata dveju tačaka između kojih se linija crta. Sa stanovišta samog crtanja linije nije važno koja se tačka zadaje prva, a koja druga. Međutim, druga tačka se u svakom prozoru individualno zapamti kao **aktivna tačka**. Liniju je moguće crtati i zadavanjem samo jedne tačke i tada se za drugu tačku linije uzima aktivna tačka, a tačka koja je u toj operaciji zadata postaje nova aktivna tačka. Ovaj sistem crtanja je naročito pogodan za crtanje kontinualnih izlomljnenih linija – ovaj slučaj često imamo prilikom crtanja grafika.

Funkcija **`K_Line`** crta liniju između dve zadate tačke. Prvi parametar je `prozor_id` prozora u kojem se linija iscrtava, a zatim slede četiti celobrojne vrednosti: x i y koordinata prve tačke, a potom x i y koordinata druge tačke.

Drugi način crtanja linije omogućen je parom funkcija **`K_MoveTo`** i **`K_LineTo`**. Prva ne daje nikakav grafički izlaz, samo postavlja položaj aktivne tačke u skladu sa zadatim koordinatama. Druga crta liniju između aktivne tačke i tačke koja je zadata preko parametara funkcije. Obe funkcije imaju isti raspored parametara: prvi je `prozor_id` prozora u koji se crta, zatim sledi x-koordinata, a na kraju i y-koordinata tačke.

```
// 1. poziv ima isti efekat kao 2. i 3. zajedno
// crta liniju od tacke (0,0) do tacke (100,100) u prozor 1
K_Line(1, 0, 0, 100, 100);
K_MoveTo(1 , 0, 0);
K_LineTo(1, 100, 100);
// 1. poziv crta popunjen pravougaonik, 2. prazan u prozor 1
K_Rectangle(1, 120, 150, 220, 250);
K_Box(1, 230, 150, 330, 250);
// 1. poziv crta praznu elipsu, 2. popunjenu u prozor 1
K_Ellipse(1, 100, 400, 50, 50);
K_FilledEllipse(1, 400, 400, 50, 50);
```

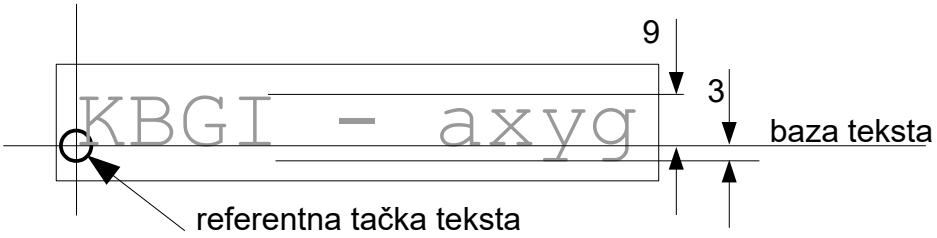
Nakon poziva funkcije **`K_LineTo`**, nova aktivna tačka postaje ona čije su koordinate prosleđene ovoj funkciji. Dakle, crtanje izlomljene linije započinje pozivom funkcije **`K_MoveTo`**, a nakon toga je dovoljno pozivanje funkcije **`K_LineTo`**.

Ispis teksta

I tekst može biti dodat na grafički ekran poput ostalih grafičkih elemenata. Veličina teksta je fiksna i ne može se menjati, a dimenzije u pikselima su date na slici niže. Boja teksta se određuje pozivom funkcije **`K_SetTextColor`** pre poziva funkcije za ispis teksta.

Sam ispis teksta se obavlja pozivom funkcije **`K_Text`** koja ima 5 parametara. Prvi je `prozor_id` prozora u kom će se tekst pojaviti, a zatim slede dve celobrojne vrednosti koje predstavljaju koordinate referentne tačke teksta (vidi sliku niže). Četvrti parametar je pokazivač na niz karaktera – string – koji sadrži tekst koji će biti ispisan. Poslednji parametar je ceo broj koji se interpretira kao dužina teksta koji će biti ispisan. Taj poslednji parametar je neophodan, nulti karakter kao oznaka kraja stringa nije dovoljan.

```
K_Text(1, 10, 30, "KBGI - axyg", 11);
```



Brisanje površine za crtanje

Često je neophodno obrisati sve nacrtane grafičke elemente. Ovo se može postići pozivom funkcije **`K_Clear`**. Prvi parametar je `prozor_id` prozora koji će se brisati, a drugi je konstanta koja određuje boju kojom će se brisanje obaviti. Drugim rečima celokupna površina za crtanje biće uniformno prekrivena izabranom bojom koja se zato često naziva **bojom pozadine**.

Neposredno nakon otvaranja (i prikazivanja) prozora njegova površina će biti obrisana i prekrivena crnom bojom. Ako je potrebno da boja pozadine bude neka druga to se postiže pozivom navedene funkcije.

Podrška dvostrukom baferisanju

Postupak u kom se kompletna slika iscrtava u pozadinskom baferu (sadržaj se ne vidi na ekranu) čiji se sadržaj u pogodnom trenutku u celosti prebacuje na ekran naziva se **dvostruko baferisanje** (*engl. double buffering*). Prednost ovog postupka je što korisnik na ekranu vidi samo smenu kompletno iscrtanih slika, a postupak iscrtavanja ostaje skriven od njega. Na taj način se drastično popravlja kvalitet pokretnog sadržaja (animacije) na ekranu za šta se ovaj postupak prvenstveno i koristi.

Neposredno nakon otvaranja prozora sistem funkcioniše tako da svaki poziv funkcije za crtanje svoj efekat prikazuje čim pre je to moguće – ovaj mod rada zove se **autorefresh**. U ovom modu je moguće da se svi grafički elementi neće pojaviti na ekranu u isto vreme jer između pojedinačnih operacija crtanja postoji vremenska zadrška koju nije moguće predvideti. Pri crtanju uzastopnih različitih slika ovo će sigurno izazvati vizuelno treperenje. Ipak, ako program crta samo jednu statičnu sliku ili grafičke elemente dodaje tokom dužeg vremenskog intervala ovo ponašanje je poželjno i prihvatljivo. Ako programer proceni da za njegov program autorefresh mod (default) nije primeren, treba da podesi sistem na dvostruko baferisanje (odnosno isključi autorefresh mod).

Pozivom funkcije **`K_SetAutorefresh`** bira se hoće li sistem raditi u autorefresh modu ili u modu dvostrukog baferisanja. Funkcija osim uobičajenog prvog parametra koji određuje `prozor_id` prozora na koji će se podešavanje odnositi zahteva i drugi parametar. Ako je drugi parametar različit od nule bira se autorefresh mod rada, a ako ima vrednost nula bira se mod dvostrukog baferovanja. Efektivno, funcija uključuje ili isključuje autorefresh mod rada, a ako je autorefresh isključen, podrazumeva se da je uključen mod dvostrukog baferisanja.

U modu dvostrukog baferisanja sve funkcije za crtanje rade isto kao i inače, samo što se njihovo dejstvo neće primetiti na ekranu sve dok se ne pozove funkcija **`K_Refresh.`** Funkcija ima jedan parametar, a to je `prozor_id` prozora na koji se crtanje odnosi. Uobičajeni sled operacija je brisanje prostora za crtanje, iscrtavanje svih grafičkih elemenata i na kraju, poziv funkcije **`K_Refresh`**. Pošto je najčešće cilj prikaz animacije, potrebno je i relativno tačno kontrolisanje vremena smenjivanja slika na ekranu. Ako se pretpostavi da su sami proračuni i crtanje grafičkih elemenata kraćeg trajanja nego potrebno vreme koliko jedna slika treba da provede na ekranu, onda

može da se primeni sledeći fragment programa:

```
K_SetAutorefresh(1, 0);      // ukljucuje dvost. baferisanje
while(K_WindowExists(1))     // dok korisnik ne zatvori prozor
{
   K_Clear(1, KBGI_BLACK);  // brisanje ekrana (u crno)
   crtaj_scenu();       // crtanje svih elem. u pozadinski bafer
   K_Refresh(1);        // pozadinski bafer se prenosi na ekran
   K_Wait(500);         // cekanje do sledece scene
}
```

Pretpostavka o kratkom trajanju proračuna i iscrtavanja je tačna kod računara novije generacije ako je broj grafičkih elemenata reda nekoliko stotina. Međutim, treba imati u vidu da to u mnogome zavisi od složenosti same scene koja se prikazuje i broja grafičkih elemenata. Savremeni računari raspolažu specijalizovanim hardverom koji izvodi operacije crtanja. Njihova brzina seže i do nekoliko stotina miliona poligona u sekundi, ali je ipak konačna. KBGI biblioteka za svoj rad ne koristi ovakav specijalizovani hardver i njene performanse su daleko skromnije.

Komunikacija sa korisnikom

Korisnik obavlja interakciju sa računarom preko tastature i miša. Savremeni računari imaju i druge mogućnosti za interakciju, ali one nisu neposredno podržane KBGI bibliotekom.

Očitavanje tastature

Pod operativnim sistemom Windows ulaz preko tastature usmeren je prema prozoru koji je u tom trenutku *aktivan*. Aktivan prozor se prepoznaje po tome da je na to prozor koji se nalazi ispred svih ostalih prozora i obično ima drugačije obojenu naslovnu liniju od ostalih prozora. Pozivanjem funkcije **`K_ReadKey`** dobija se ASCII kod karaktera koji odgovara tasteru pritisnutom na tastaturi. Ovo je jedna od retkih funkcija koja umesto dijagnostičkog koda vraća stvarni rezultat – u ovom slučaju ASCII kod. Ukoliko na tastaturi nije pritisnut nijedan taster vraćena vrednost jednaka je konstanti `KBGI_KBUF_EMPTY`. Jedini parametar funkcije je `prozor_id` prozora u kojem se želi pratiti unos karaktera.

Između dva uzastopna poziva funkcije moguće je da bude pritisnuto nekoliko tastera. Da bi se izbegla potreba za čestim pozivanjem funkcije **`K_ReadKey`**, a da se informacija o pritisnutim tasterima ne bi izgubila, svaki pritisnuti taster se privremeno beleži u tzv. **bafer** koji može da zabeleži poslednjih `KBD_BUF_LEN` pritisnutih tastera (u aktuelnoj verziji biblioteke vrednost ove konstante je 10). Svaki poziv funkcije **`K_ReadKey`** vraća kod koji odgovara tasteru koji je najranije smešten u bafer (nakon što je pritisnut, naravno). Ukoliko korisnik pritisne više od `KBD_BUF_LEN` tastera između dva očitavanja (pozivom funkcije **`K_ReadKey`**) u baferu se pamti poslednjih `KBD_BUF_LEN` kodova (oni stariji se gube).

Ukoliko korisnik aktivira neki drugi prozor u međuvremenu, svi kodovi smešteni u bafer se gube. Na taj način se eliminiše mogućnost da se aktivacijom prozora nenadano pojave neki stari (fantomski) pritisci na tastere koji više nisu aktuelni i korisnik ih ne očekuje.

Da bi se izbeglo da kodovi predugo borave u baferu i time rizikuje da ne budu obrađeni (jer će biti prepisani novounetim kodovima ili poništeni aktivacijom drugog prozora) preporučuje se da se svaki put kada program analizira pritiske tastera na tastaturi bafer isprazni (ponavljenim pozivanjem funkcije **`K_ReadKey`**) na način dat u sledećem primeru:

```
while((ch = K_ReadKey(1)) != KBGI_KBUF_EMPTY)
{
   obrada_koda(ch);     // hipoteticka funkcija koja obradjuje
                        // pojedinacne kodove unete preko tast.
}
```

Ovaj način **nije pogodan** za očitavanje specijalnih tasteria koji nemaju uobičajeni odziv u vidu nekog vidljivog karaktera (npr. kursorski tasteri, funkcijski tasteri, shift/alt/ctrl i sl.). Neki specijalni tasteri pak daju odziv prilikom pozivanja ove funkcije iako njihova funkcija nije ispis vidljivog karaktera. Njih obavezno treba analizirati preko numeričkog koda koji je rezultat njihovog pritiskanja. Najvažniji tasteri i kodovi koje daju sumirani su u sledećoj tabeli:

NAZIV	**ASCII KOD (decimalno)**
CARRIAGE RETURN (ENTER)	13
BACKSPACE	8
ESCAPE	27
DEL	127
TAB	9

Funkcija **`K_ClearKeys`** omogućava da se bafer unetih kodova isprazni. Kodovi svih pritisnutih tastera koji su bili u baferu se trajno gube. Jedini parametar funkcije je `prozor_id` prozora čiji će bafer biti ispražnjen, na druge prozore funkcija ne utiče.

Očitavanje kursorskih tastera na tastaturi kao upravljačkog ulaza

Kursorski tasteri se uobičajeno koriste za pomeranje tekstualnog kursora unutar programa koji omogućuju uređivanje teksta. Međutim, pošto je namena KBGI biblioteke crtanje grafičkih elemenata (a ne tekstualnih, karakter orijentisanih) ovi tasteri su slobodni za upotrebu kao upravljački ulazi za interakciju sa grafičkim okruženjem – poput kontrolera za igre (džojstika). KBGI omogućuje veoma jednostavan rad sa kursorskim tasterima na ovaj način.

Pozivom funkcije **`K_KbdDir`** sa `prozor_id`-om prozora (kao jedinim parametrom) preko kog se obezbeđuje ulaz, kao povratna vrednost dobija se celobrojna vrednost koja u sebi sadrži informacije o tome koji je od kursorskih tastera pritisnut. Svaki od najniža četiri bita odgovara tasteru sa jednom od strelica:

bit:	3	2	1	0
taster:	LEVO	DOLE	DESNO	GORE
vrednost bita:	8	4	2	1
makro konstanta:	DIR_LEFT	DIR_DOWN	DIR_RIGHT	DIR_UP

Preporučuje se upotreba makro konstanti (definisanih u hederu `kbgi_lib.h`) za maskiranje pojedinih bita u reči i utvrđivanje njihovih stanja. U programu to može da izgleda ovako:

```
joy = K_KbdDir(3);
if(joy & DIR_UP)
   radi_gore();
```

```
else if(joy & DIR_DOWN)
   radi_dole();
if(joy & DIR_LEFT)
   radi_levo();
else if(joy & DIR_RIGHT)
   radi_desno();
```

U ovom fragmentu programa uvažava se nemogućnost da istovremeno budu aktivne suprotnosti, tj. i gore i dole, odnosno i levo i desno. Kose kombinacije (npr. i gore i levo) su moguće i veoma uobičajene.

Očitavanje akcija korisnika izvedenih pomoću miša

Interakcija pomoću miša koju biblioteka KBGI podržava je skromna – moguće je samo očitavanje položaja na kom je korisnik unutar prozora kliknuo levim tasterom miša. Registruje se samo prvi klik unutar prozora, sledeći klik se neće registrovati ukoliko se prethodni ne očita pozivanjem funkcije **`K_ReadMouse`** ili eksplicitno obriše pomoću funkcije **`K_ClearMouse`**.

Prilikom klika na prozor, veoma je bitna informacija na kom mestu se unutar tog prozora nalazio kursor. Zbog toga, funkcija **`K_ReadMouse`** treba da obezbedi 3 podatka:

- da li se desio klik na površinu prozora,
- ako se desio klik, koja je bila x-koordinata kursora,
- ako se desio klik, koja je bila y-koordinata kursora.

Prva informacija je sadržana u vrednosti koju funkcija vraća. Vraćena vrednost je jednaka nuli (može se iskoristiti i makro `KBGI_NO_CLICK`) ako se nije desio klik od poslednjeg poziva funkcije, ili jednaka 1 (`KBGI_CLICK`) ukoliko jeste.

Informacija o koordinatama se prenosi preko pokazivača na strukturu sledećeg oblika:

```
struct mouse_position {
    short mouse_x;
    short mouse_y;
};
```

Ova struktura definisana je u hederu `kbgi_lib.h`, funkciji se prosleđuje pokazivač na nju. Ukoliko je funkcija vratila informaciju da se klik desio, struktura će biti popunjena podacima o koordinati, u suprotnom njen sadržaj će biti neizmenjen. Sledeći primer ilustruje primenu ovog sistema:

```
struct mouse_position mi;
...
if( KBGI_CLICK == K_ReadMouse(1, &mi) )
{
  // obrada klika, crtanje bele tacke na toj poziciji
  K_Dot(1, mi.x, mi.y, KBGI_WHITE);
}
```

Ako se desilo više od jednog klika na prozor od poslednjeg poziva funkcije **`K_ReadMouse`**, vraćena informacija će se odnositi na prvi od njih. Ostali klikovi su ignorisani. Pozivom funkcije **`K_ClearMouse`** (jedini parametar je `prozor_id`) briše se zapamćena informacija o ranijem kliku. Na ovaj način se obezbeđuje svež podatak o interakciji sa korisnikom posredstvom miša.

Posebnosti DEBUG verzije biblioteke

U interesu optimalnog iskorišćenja sistemskih resursa pojavljivanje nacrtanih geometrijskih oblika na ekranu se često ne poklapa sa povratkom iz funkcija za crtanje KBGI_biblioteke. Drugim rečima, moguće je da će se novi geometrijski oblik pojaviti zajedno sa desetinama drugih geometrijskih oblika posle određene vremenske zadrške. Ovakvo ponašanje naziva se **asinhronim**, tj. pojavljivanje grafičkih elemenata **nije sinhronizovano** sa pozivom odgovarajućih funkcija za crtanje. U sistemima gde je postizanje visokih performansi cilj ovakvo ponašanje je sasvim uobičajeno, a zbog velike brzine crtanja korisnik koji posmatra ekran nije ni svestan asinhronizma.

Debagovanje je pak okolnost u kojoj ovakvo ponašanje može da bude nepoželjno jer se javlja situacija u kojoj dibager pozove funkciju za crtanje i vrati se iz nje, međutim grafički element se ne pojavljuje na ekranu zbog asinhronizma. U ovakvim okolnostima forsiranje sinhronizma funkcija za crtanje postaje poželjno i na neki način obavezno. Debug verzija KBGI biblioteke (sadržana u fajlu `libkbgi_libD.a`) se razlikuje od obične verzije po nametanju ovakvog sinhronizma. Ne preporučuje se njeno korišćenje za konačne verzije programa jer može imati primetno lošije performanse od obične verzije.

DODATAK A – Funkcije abecednim redom

```
int K_Box(unsigned int WinN, int x0, int y0, int x1, int y1);
int K_Clear(unsigned int WinN, unsigned int color);
int K_ClearKeys(unsigned int WinN);
int K_ClearMouse(unsigned int WinN);
int K_Dot(unsigned int WinN, int x, int y, unsigned int color);
int K_Ellipse(unsigned int WinN, int x, int y, int R1, int R2);
int K_EndWindow(unsigned int WinN);
int K_FilledEllipse(unsigned int WinN, int x, int y,
    int R1, int R2);
int K_KbdDir(unsigned int WinN);
int K_kbgi(void);
int K_Line(unsigned int WinN, int x0, int y0, int x1, int y1);
int K_LineTo(unsigned int WinN, int x, int y);
int K_MoveTo(unsigned int WinN, int x, int y);
int K_MoveWindow(unsigned int WinN, int x, int y);
int K_OpenWindow(unsigned int WinN, char *WindowTitle);
int K_ReadKey(unsigned int WinN);
int K_ReadMouse(unsigned int WinN, struct mouse_position *mp);
int K_Rectangle(unsigned int WinN, int x0, int y0,
    int x1, int y1);
int K_Refresh(unsigned int WinN);
int K_SetAutorefresh(unsigned int WinN, unsigned char x);
int K_SetDrawColor(unsigned int WinN, unsigned int color);
int K_SetFillColor(unsigned int WinN, unsigned int color);
int K_SetTextColor(unsigned int WinN, unsigned int color);
int K_Text(unsigned int WinN, int x, int y, char *text, int len);
void K_Ver(unsigned int *major, unsigned int *minor);
void K_Wait(unsigned int msec);
int K_WaitAllClose(void);
int K_WaitForClose(unsigned int WinN);
int K_WindowExists(unsigned int WinN);
```

DODATAK B – Spisak boja

BOJA	VREDNOST	UZORAK BOJE
KBGI_BLACK	0	
KBGI_RED	1	
KBGI_YELLOW	2	
KBGI_GREEN	3	
KBGI_CYAN	4	
KBGI_MAGENTA	5	
KBGI_BLUE	6	
KBGI_WHITE	7	
KBGI_GREY	8	
KBGI_DARK_GREY	9	
KBGI_LIGHT_GREY	10	
KBGI_DARK_BLUE	11	
KBGI_DARK_GREEN	12	
KBGI_DARK_RED	13	
KBGI_DARK_YELLOW	14	
KBGI_ORANGE	15	