Osnove virtualnih okruženja

Laboratorijske vježbe

Vježba 6:
Programsko sučelje više razine:
OpenSceneGraph

FER – ZTE – Igor S. Pandžić
Suradnja na pripremi vježbe:
Zlatko Đukić

1. Uvod

OpenSceneGraph [u nastavku teksta OSG], je grafičko programsko sučelje (*eng*. API – application programming interface) visoke razine. Izvorni kôd je besplatan i javno dostupan na stranici www.openscenegraph.org. Paket se temelji na OpenGL grafičkoj biblioteci, pa se zbog toga preporučuju NVidia-ini grafički čipovi koji u većoj mjeri podržavaju OpenGL od ATI-evih. Podržana je višedretvena izvedba, prikaz jednog sadržaja scene na više prozora i slično.

Zadatak svih programskih sučelja više razine, pa tako i OSG-a, jest olakšavanje posla kod stvaranja grafičkih aplikacija što uključuje: definiranje scene, učitavanje modela u scenu, podešavanje osvjetljenja, transformacije nad bilo kojim objektima i naposljetku prikaz scene.

Radi jednostavnosti paket je podijeljen u tri cjeline.

- **OpenSceneGraph** - osnova paketa za rad s grafom scene
- **OpenProducer** - dodatak za prikaz na ekranu
- **OpenThreads** - implemetacija višedretvenih funkcija koje nisu dostupne u Windows operacijskim sustavima.

2. Alati potrebni za izvođenje vježbe

Za izvođenje vježbe potreban je **MS Visual Studio 2003.net** i spomenuti **OSG API**.

Paket sa sadržajem OSG API-ja ponuđen je u dvije inačice: kao izvorni kôd i kao tzv. binaries. Drugi paket sadrži već preveden cijeli izvorni kôd i dolazi s instalacijskom procedurom koja u velikoj mjeri olakšava integriranje svih dijelova paketa sa MS Visual Studiom. Očito je da ćemo zato na ovim vježbama koristiti binarnu inačicu paketa koja je dostupna na web stranicama OSG-a, a isto tako i na web stranicama predmeta. Ime joj glasi **osg1.0_setup_2005-12-09.exe** [~20MB]. U instalacijskom paketu dolaze sve tri potrebne komponente ovog paketa, a tekuća i stabilna verzija je 1.0 kao što je vidljivo iz imena datoteke.

2.1 Instalacija paketa

Instalaciju pokrećemo dvoklikom na danu .exe datoteku i slijedimo upute. Predlažem da se za odredišni folder ostavi ponuđena vrijednost: *C:\Program Files\OpenSceneGraph*. U nastavku dokumenta smatrat ćemo da je odabrana navedena mapa i u skladu s tim opisivati dodatne korake podešavanja. U nastavku instalacijske procedure potrebno je još samo uključiti opciju *Set OSG environment variables in Registry*, a ostale po želji. Instalirani paket zauzima nešto više od 80MB na disku.

Da budemo sigurni u ispravnost instalacije uvjerimo se da su *environment* varijable ispravno podešene. Nekoliko je načina provjere i postavki tih varijabli, a najlakše je preko Control Panela i pripadnog System panela [najlakše do njega doći pritiskom na tipke Windows na tipkovnici i Pause/Break], zatim Advanced odjeljak pa dugme Environment variables. Pod System variables treba pronaći zapise OSG_FILE_PATH i OSG_ROOT te se uvjeriti da su oni postavljeni na `C:\Program Files\OpenSceneGraph\data;C:\Program Files\OpenSceneGraph\data\Images;C:\Program Files\OpenSceneGraph\data\fonts`

odnosno `C:\Program Files\OpenSceneGraph.`

Varijabla PATH treba sadržavati uz prethodno postavljene unose još i put do .dll biblioteka koji glasi `C:\Program Files\OpenSceneGraph\bin;C:\Program Files\OpenSceneGraph\data.`

2.2 Opis instaliranih datoteka

Osim izvornog kôda cijele biblioteke instalacijom se stvaraju i .dll i pripadne .h i .lib datoteke. Radi cjelovitosti opisat ćemo te tipove datoteka.

- DLL – (eng. Dynamic Link Library), datoteke s izvršnim kôdom
- H – header datoteke koje uključujemo u projekt putem include direktive
- LIB – library datoteke koje služe kao sučelje prema dll

Razlika u odnosu na uobičajena imenovanja jest samo u tome da nemaju sve header datoteke nastavak .h . LIB datoteke koje imaju u imenu posljednje slovo '**d**' označavaju da se radi o debug verziji izvršne datoteke, a ako nemaju onda je riječ o release verziji. Razlika je u tome da se kod prvih debugger može *šetati* i unutar funkcija iz biblioteka. Mi ćemo koristiti release verzije tih datoteka jer su jedino one uključene u instalaciju. Zaglavlja [.h i bez ekstenzije] su unutar **`include`** foldera, biblioteke [.lib] unutar **`lib`** foldera i izvršne bliblioteke [.dll] unutar **`bin`** foldera.

2.3 Izvođenje vježbe na vlastitom računalu

Za izvođenje vježbe potreban je Microsoft Visual Studio .net (odgovarajuća verzija) u kojem se programira i OpenSG instalacijski paket.

3. Teorijska podloga

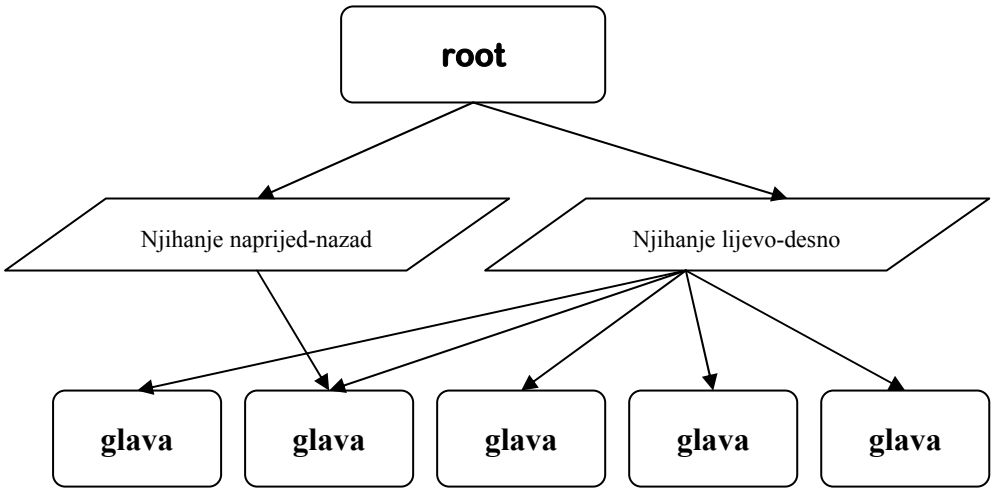
Jasno je da su osnovni elementi iscrtavanja točke. Očito je da bi bilo vrlo komplicirano, a vjerojatno i nemoguće današnje grafičke zadatke izvršavati kada bi korisnik [programer] morao definirati neki prikaz preko točaka koje ga tvore.

Zato se *penjemo* na jedan nivo više i dolazimo do 2D grafike, koja omogućuje grupiranje točaka u linije, trokute i općenito poligone. Također omogućuje manipulaciju tim objektima, definira njihove atribute [duljina, debljina, boja itd.]. Jasno je da za realističan prikaz npr. čovjekova lica nije dovoljno samo nekoliko poligona nego ih je poželjno imati barem nekoliko stotina. Tolikim brojem poligona prilično je nezahvalno upravljati [programeru]. Osim toga, svaki objekt koji želimo prikazati definiran je u 3D prostoru, a njegovom definicijom u 2D sustavu nemamo mogućnost pregleda objekta sa svih strana. Ovo nam nameće ponovno još jedan novi nivo, 3D grafiku.

Sada su poligoni grupirani u skupine, dodjeljena su im neka zajednička svojstva. Npr. svi poligoni koji tvore čovjekovo lice imaju sličnu boju, ali ne moraju biti samo vidljiva svojstva zajednička. Takve skupine nazivamo objekti [ne pomiješati ovaj izraz s istim izrazom iz programiranja]. U slučaju potrebe za pomakom lica na npr. drugi kraj ekrana, sada ne moramo reći za svaki poligon da želimo translaciju, nego samo za jedan objekt – čovjekovo lice [glavu]. Ali, što ako imamo više glava u sceni koju modeliramo, npr. pjevački zbor koji

se njiše u ritmu glazbe lijevo-desno? Imali bismo definirano mnogo objekata tipa glava, ali s obizrom da se oni sinkronizirano njišu, bismo li ih mogli podrediti pod neki novi objekt? Odgovor je: da i ne.

Time dolazimo do našeg nivoa, grafa scene. Skupinu glava iz prethodnog pitanja mogli bismo definirati kao novi objekt, ali time bismo izgubili opet njihovu *pojedinačnost*; moguće je da jedan od njih na nekoliko trenutaka počne i s njihanjem naprijed-nazad, a mi nemamo načina da mu pristupimo pojedinačno i dodamo i ovo njihanje. Sada dolazi rješenje - **graf scene kao apstraktna hijerarhijska struktura kojom definiramo objekte i njihove odnose te transformacije koje želimo vršiti nad njima.** Ova struktura je zapravo jedno zamišljeno stablo, na koje smo već navikli u računarstvu. Za početak ćemo pojednostavniti njegov prikaz; sastojat će se od čvorova koji mogu biti: glava i transformacija. Iako ih zamišljamo ravnopravno, moramo primijetiti razliku: **čvor transformacije utječe na svu svoju djecu**, pa ćemo ga malo drugačije označavati u stablu. U grafu scene mora postojati izvorni (korijenski) čvor, obično se naziva **root** čvorom.



Slika 3-1: Graf scene za navedeni primjer

Ovakvom hijerarhijom načinili smo odnose transformacija gibanja koji su vrlo jednostavni za izmjenu, nadopunu i sl.

U sceni se mogu nalaziti i čvorovi različitih tipova poput osvjetljenja i emitera čestica i još mnoštvo raznih, a značajnije ćemo upoznati u nastavku. Čvoru se mogu dodijeliti **stanja** što je od velikog značaja. Sva djeca čvora komu je dodijeljeno stanje nasljeđuju to isto stanje, a ona se koriste za definiranje tekstura objekata, materijala itd.

3.1 Prva OSG aplikacija

Opisat ćemo način stvaranja aplikacija temeljenih na OSG grafičkoj biblioteci koristeći razvojno okruženje MS VS.net2003. Najjednostavnija aplikacija, a ujedno i temelj svake grafičke aplikacije je *viewer* ili preglednik. Ona se sastoji samo od inicijalizacije grafa scene i inicijalizacije objekta koji sadrži metode za iscrtavanje na ekranu (pogledati upute za rad 2. zadatka u 5. poglavlju).

Obično se kôd za prikaz [iscrtavanje] nalazi u **`main()`** funkciji i uvijek je nužan ukoliko nešto želimo prikazati na ekranu.

Možda će biti jasnije ako se princip iskaže na ovaj način: pisanje aplikacija se svodi na proširenje viewera, odnosno kôda za prikaz na ekranu.

3.1.1 Priprema MS VS.net-a za razvoj aplikacije

Potrebno je pokrenuti novi projekt kao praznu *Console* aplikaciju, prijedlog imena projekta je testOSG, a izvorne datoteke OSGtest.cpp. Unutar *Solution Explorera* desni klik na ime projekta [testOSG] i odabrati *Properties*. Na lijevoj strani Poperty prozora odabiremo grupu C/C++ i njezin skup opcija pod nazivom Code Generation. Na desnoj odabiremo pod opcijom Runtime Library izraz **Multi-threaded DLL (/MD)**. Unutar iste grupe C/C++ odabiremo skup opcija Language i vrijednost Enable Run-Time Type Info postavljamo na **YES(/GR)**.

Za grupu odabrati Linker, zatim skup opcija Input i kod Additional Dependencies treba upisati `.lib` datoteke potrebne za naš projekt. U ovom slučaju biti će dovoljne sljedeće [odvojene razmakom]:

```
osg.lib osgDB.lib osgProducer.lib
```

Ispod je popis svih biblioteka koje su potrebne za ovu vježbu pa je jednostavnije odmah ih uključit u projekt kako se kasnije ne bi zaboravilo na njih.

```
OpenThreadsWin32.lib Producer.lib osgProducer.lib osg.lib libjpeg.lib
libpng.lib libtiff.lib libungif.lib osgDB.lib osgFX.lib osgGA.lib
osgParticle.lib osgSim.lib osgText.lib osgUtil.lib zlib.lib
```

Svaki novi projekt zathjeva ova podešavanja.

Potrebno je podesiti putove unutar MS Visual Studia do datoteka zaglavlja, biblioteka i njihovih sučelja. To je moguće učiniti zasebno za tekući projekt ili općenito za sve projekte, a mi ćemo se odlučiti za drugu opciju jer prilikom stvaranja svakog projekta bismo morali ponovno podešavati iste parametre.

Odabrati izbornik *Tools* pa *Options*, zatim *Projects* na lijevoj strani prozora te konačno *VC++ directories*. U gornjem desnom dijelu prozora nalazi se padajući izbornik pod nazivom *Show directories for* gdje najprije odabiremo *Executable Files*. U popis foldera dodajemo put do ***bin*** foldera iz OSG direktorija, primjerice: `C:\Program Files\OpenSceneGraph\bin`.

Zatim iz padajućeg izbornika odabiremo *Include Files* te dodajemo ***include*** folder, primjerice `C:\Program Files\OpenSceneGraph\include`.

Napokon, odabiremo *Library files* i dodajemo put do library foldera ***lib***, primjerice `C:\Program Files\OpenSceneGraph\include`.

3.2 Korištenje specifičnih 3D komponenata

Kod modeliranja 3D aplikacija koriste se unaprijed definirani programski modeli i potpuno podržani od strane API-ja kojeg koristimo, a u ovom slučaju to je OSG.

Najprije ćemo teoretski obrazložiti koje se najčešće metode koriste prilikom stvaranja ovakvih aplikacija. Unutar ovih uputa za izradu laboratorijske vježbe nije moguće nabrojati sve mogućnosti ovog API-ja, ali ćemo se zato osvrnuti na najvažnije tj. one koji se najčešće koriste. Tako ćemo upoznati pojmove:

- Texture
- Billboard
- Particle

Prije nego naučimo što su spomenuti pojmovi i pojasnimo način njihove implementacije, prvo ćemo se upoznati malo deatljnije s čvorovima koji čine graf scene.

Već smo spomenuli tip **osg::Group** koji se koristi kao čvor kojeg koristimo kada želimo grupirati više djece čvorova radi lakših manipulacija njima.

Tip **osg::Node** je čvor koji služi kao baza svim drugim čvorovima.

Podatkovni tip **osg::Geode** je geometrijski čvor na mjestu lista u grafu, a može imati postavljene opcije za specifično renderiranje, kao što je to Billboard (opisano u nastavku).

osg::Drawable je virtualna bazna klasa za geometrijske podatke.

osg::Geometry nasljeđuje osg::Drawable klasu i omogućuje definicije geometrijskih podataka poput primitiva.

osg::Matrix bazna klasa za implementaciju matrica, **osg::Matrixd** i **osg::Matrixf** su najvažnije, a implementiraju matrice s elementima tipa double i float.

osg::Vec*xy* definira vektor, ali pri tome se na mjestu *x* nalazi broj 2, 3 ili 4, a *y* je slovo koje oznacava tip podatka, npr. *d* ili *f*. Primjerice, osg::Vec3f ili osg::Vec2d.

osg::MatrixTransform je tip čvora koji definira transformaciju nad svom svojom djecom, a mora mu se postavit matrica trannsformacije npr. osg::Matrixd.

Billboard je posebno definiran geometrijski čvor tipa **osg::Billboard** koji uvijek orijentira svoju djecu tipa Drawable prema poziciji očiju odnosno kameri. Primjerice, drvo u prirodi može biti definirano samo jednom teksturom radi uštede na resursima; ako bi ona bila nepomična i postavljena u smjeru sjever-jug tako da ju pravilno vidimo sa zapada ili istoka, tada ne bismo dobro vidjeli to drvo sa sjevera ili juga. Ako se ono uvijek okreće prema kameri odnosno oku promatrača, tada uvijek drvo vidimo na ispravan način, a štedimo na resursima jer smo definirali samo jedan jednostavan resursno nezahtjevan objekt oblika ploče.

Particle system je mehanizam koji se sastoji od nekoliko vrsta čvorova, koristit ćemo ih u zadacima, ali nećemo detaljno pojašnjavati jer bismo prešli opseg ovog dokumenta. Ideja mehanizma čestica jest generiranje odnosno simulacija čestica poput dima i sl. koji mogu biti pod utjecajem vjetra; ili fluida, eksplozija i slično.

Texture nije objekt niti čvor u sceni, nego je parametar iz tzv. ***state set***-a odnosno skupa stanja. Inicijalizira se tip podatka **osg::StateSet** i njemu postavi atribut na teksturu i uključi. I

svjetlost je jedan od atributa koji se mogu uključiti i isključiti, ali ovdje ju nećemo koristiti. Nakon inicijalizacije skupa stanja, potrebno je taj skup dodijeliti određenom čvoru, a važno je napomenuti da tada i sva njegova djeca nasljeđuju taj isti skup stanja. Sve spomenuto ćemo vidjeti u izvornom kodu zadatka.

Ovime smo samo dotaknuli mehanizme koji se koriste kod stvaranja 3D aplikacija, a zajednički su (barem idejno) svim sustavima koji implementiraju programske funkcije višeg nivoa za stvaranje 3D aplikacija.

4. Zadatak

Zadatak 1.

Nakon pravilne konfiguracije radne okoline potrebno je prikazati nekoliko (barem 2) modela iz foldera *data*, mijenjati položaj kamere mišem i priložiti sliku ekrana u izvješće.

Napomena: defaultni prikaz je preko cijelog ekrana, tipkom 'f' se scena smanjuje na nivo prozora.

Zadatak 2.

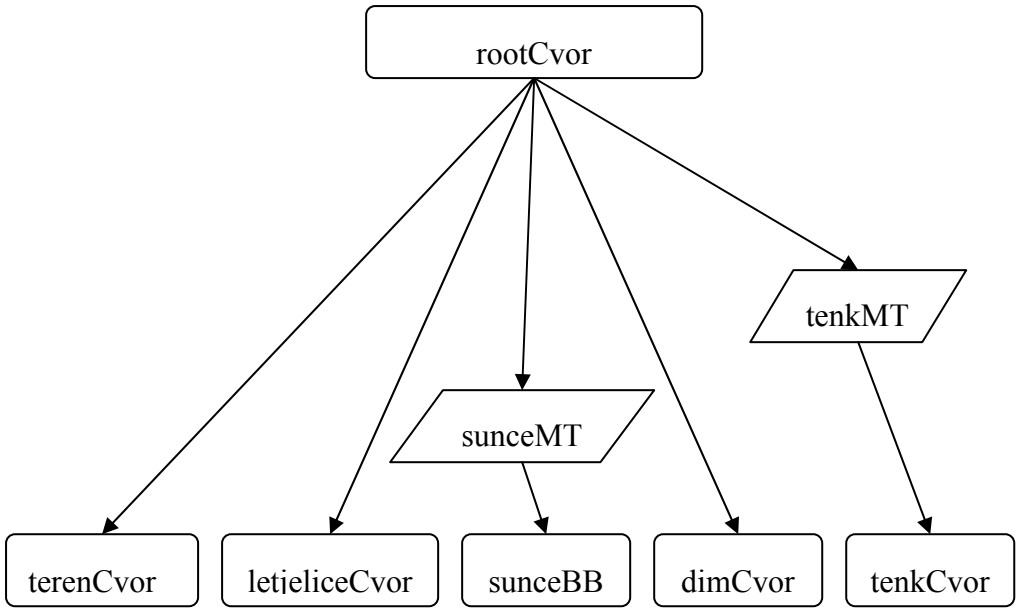
Nakon podešavanja MS Visual Studia potrebno je napisati aplikaciju viewer za pregled modela koristeći zadani kôd. Priložiti kôd i sliku ekrana u izvješću.

Napredni zadatak (neobavezno): izgradnjom još jednog čvora učitati dodatan model i postaviti u scenu te priložiti u izvješće kôd i sliku ekrana.

Napomena: koristiti potpuno istu tehniku za učitavanje u čvor i dodati još jednu naredbu koja sadrži metodu *addChild(...)* kako bi se dodao novi čvor korijenskom čvoru, odnosno novi model sceni.

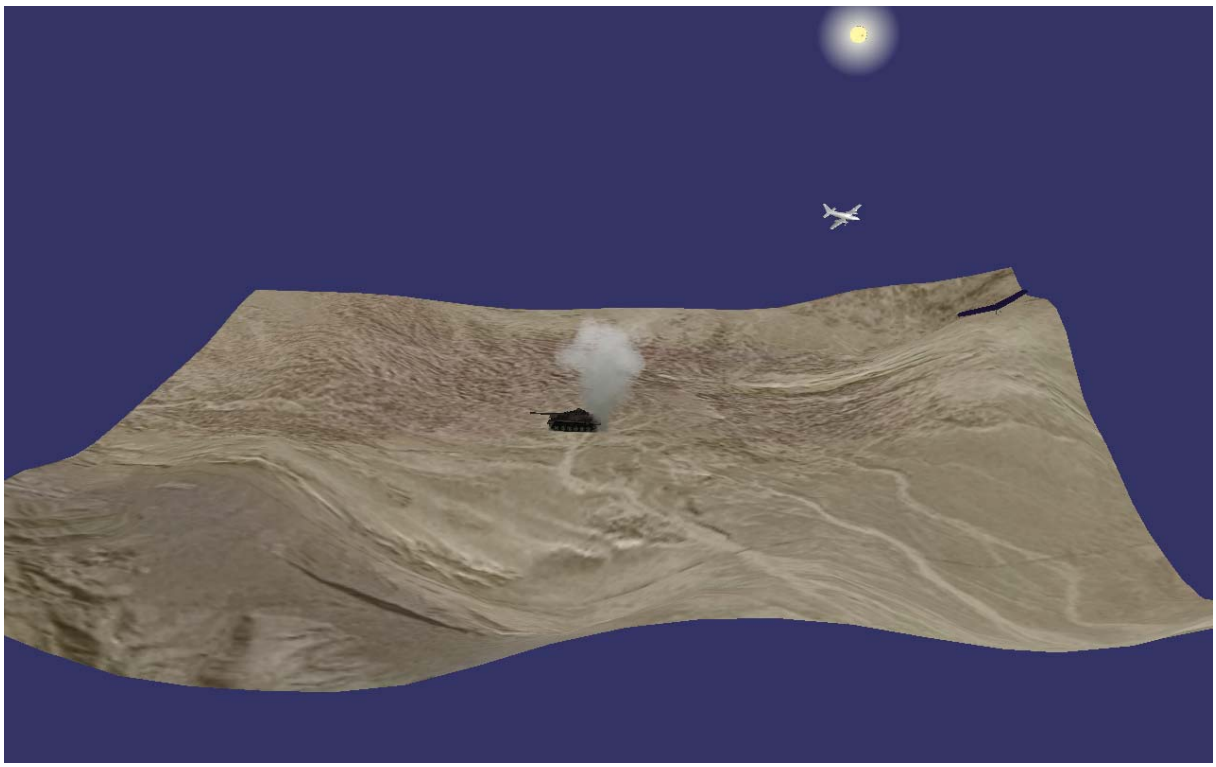
Zadatak 3.

Na temelju zadanih klasa koje implementiraju dim, sunce, letjelice, tenk i teren stvoriti graf scene te prikazati scenu na ekranu. Projekt u kojemu je samo potrebno dodati kôd za implementaciju grafa scene dan je na stranicama predmeta pod nazivom **`zadatak3.zip`** . Grafički prikaz stabla (grafa scene) kojeg je potebno realizirati dan je na blok shemi ispod. Implementirani kôd i sliku ekrana priložiti u izvješću. Pomicati kameru i uočiti kako je sunce uvijek okrenuto prema njoj.



Slika 4-1: Scena koju je potrebno implementirati

Ovako stvorena scena morala bi odgovarati sceni na slici 4-2.



Slika 4-2: Scena (u nekom trenutku) nakon pravilnog povezivanja čvorova u grafu

Napredni zadatak (neobavezno): Modificirati postavke matrice za postavljanje tenka u scenu, izmjeniti veličinu, orijentaciju (unutar klase koja implementira tenk). Mijenjati postavke obrasca za čestice (unutar klase *ovoDim*), udvostručiti duljinu života čestica što rezultira većom visinom dima, mijenjati boju i sl. Priložiti sliku scene.

5. Upute za rad

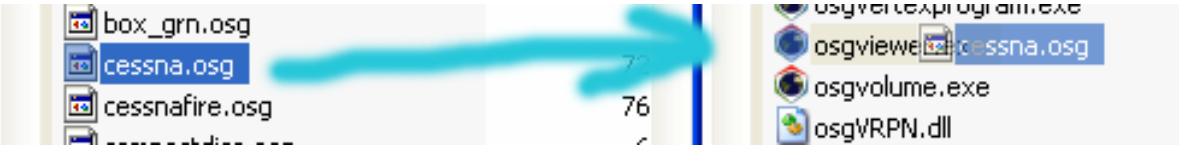
Modele tenka, terena, tekstura i fontova (pogledati poglavlje 6.) potrebno je ekstrahirati iz datoteke **resursi.zip** u *data* folder unutar *OpenSceneGraph* foldera tako da put do njih bude:

```
C:\Program Files\OpenSceneGraph\data\Models\JoeDirt
C:\Program Files\OpenSceneGraph\data\Models\t72-tank
C:\Program Files\OpenSceneGraph\data\fonts\
C:\Program Files\OpenSceneGraph\data\Textures
```

5.1 Upute za 1. zadatak

Kako bismo probali radi li uspješno instalirani paket pokušajmo prikazati neki model uz pomoć programa `osgviewer.exe` koji se nalazi u `bin` folderu instalacije. Prijedlozi za testiranje ispravnosti rada su: pokrenuti iz komandnog prompta kao što je navedeno ili iz `data` foldera odvući pojedini od navedenih modela [*.osg] na `osgviewer.exe` unutar `bin` foldera.

```
C:\> osgviewer cow.osg
C:\> osgviewer fountain.osg
```



Slika 5-1: Jednostavan način pregleda modela

Primijetiti da se može pritisnuti slovo **‚h'** za vrijeme prikaza scene što daje pomoć na ekranu, a tiče se pomicanja kamere, iscrtavanja nekih podataka vezanih uz opterećenje sustava tokom animacija i sl. Kod izgradnje aplikacija koristit ćemo jednostavniju verziju viewera kod koje neće biti moguće dobiti pomoć pritiskom na spomeunuto slovo.

Probati kamere 1 i 3 [pritiskom na te brojeve na tipkovnici] i kombinirati sa sve tri mišje tipke i pomakom miša za vrijeme držanja jedne od tipaka. Space pomiče kameru unutar globalnog koordinatnog sustava na poziciju ishodišta.

5.2 Upute za 2. zadatak

Prvo ćemo opisati osnove OSG viewer-a. Kostur je prikazan u sljedećem ispisu.

```
#include <osgProducer/Viewer>
. . .
int main() {
      . . .
      osgProducer::Viewer viewer;
      . . .
      . . .
      viewer.setUpViewer(osgProducer::Viewer::STANDARD_SETTINGS);
      viewer.setSceneData( root );
      viewer.realize();

      while( !viewer.done() )
      {
          viewer.sync();
          viewer.update();
          viewer.frame();
      }
}
```

Potrebno je uključiti odgovarajuće zaglavlje, deklarirati objekt tipa Viewer, postaviti opcije, postaviti **izvorni čvor** [**root**] scene grafa i stvoriti **`while`** petlju unutar koje se moraju nalaziti pozivi funkcija koje se neprekidno izvršavaju kako bi se dobilo kontinuirano iscrtavanje. Detalje nećemo ovdje navoditi. Dani kôd nije potpun jer nema definiranog grafa pa se nema što iscrtati, ali ćemo ga proširiti već u sljedećem primjeru.

Svi se grafički objekti mogu programski stvoriti od primitiva (ploče, kocke, sfere, piramide…), ali to bi bilo vrlo nepraktično kada bi bio jedini način. Stoga je podržan rad s objektima definiranim u poznatim grafičkim formatima poput 3D Studio Maxa [.3ds], OpenFlight [.flt] i brojnim drugima. Nadalje, omogućen je uvoz (import) i izvoz (export) objekata [scena] stvorenih programski u OSG aplikacijama.

Sada smo u stanju napisati cjelovit program temeljen na OSGu koji će prikazati scenu učitanu iz datoteke.

Iz direktorija `C:\Program Files\OpenSceneGraph\data` potrebno je učitati proizvoljan model u čvor i prikazati ga. Kao pomoć dan je kôd deklaracije čvora koji učitava model krave. Tu deklaraciju postaviti odmah ispod deklaracije viewera. Primjetiti dvostruke *backslash* znakove: prvi je escape sekvenca, a drugi označava odjeljivanje direktorija. Umjesto dvostrukih backslah znakova može se koristiti i frontslash '/'.

```
osg::Node* cowN = osgDB::readNodeFile("C:\\Virt real\\zadaci\\cow.osg");
```

Za korištenje funkcije **`readNodeFile`** potrebno je uključiti zaglavlje `ReadFile`:

```
#include <osgDB/ReadFile>
```

`Za korištenje tipa podataka` **`Node`** `potrebno je uključiti Node zaglavlje:`

```
#include <osg/Node>
```

Zatim je potrebno deklarirati **root** čvor naredbom:

```
osg::Group *root = new osg::Group;
```

Za tip podatka **`Group`** potrebno je zaglavlje:

```
#include <osg/Group>
```

Na poslijetku treba stvoriti graf scene tako da se doda čvor u kojeg je učitan teren kao dijete vršnom čvoru **root** [obratiti pažnju na mjesto pozicioniranja ove naredbe u kôdu, mora doći nakon deklaracija!].

```
root->addChild(cowN);
```

5.3 Upute za 3. zadatak

S obzirom na kompleksnost implementacije svega navedenog, naš zadatak bit će stvaranje strukture grafa scene koristeći module odnosno klase unutar kojih je već pripremljen u potpunosti pojedini dio zadataka.

Zadatak je implementirati scenu kod koje ćemo koristiti svaki od navedenih mehanizama.

Scena će se sastojati od terena, tenka u gibanju, glider-a i aviona i koji se uzdižu i spuštaju dok kružno lete (demonstracija callback-ova), sunca (demonstracija billboard-a i teksture), te dima (demonstracija mehanizma čestica).

Izvorni kôd se sastoji od klasa (po jedna *.cpp* i *.h* datoteka za svaku klasu) koje implementiraju:

- Letjelice (cessna i glider)
- Dim
- Sunce
- Tenk
- Teren

Datoteka *Glavni.cpp* implementira funkciju main() i inicijalizira scenu. Potrebno ju je modificirati kako bismo stvorili graf scene dan na slici u 3. poglavlju.

Radi jednostavnosti korištenja kôd je koncipiran na spomenuti način iako bi mogao biti kompaktnije napisan, ali to bi unijelo nepotrebnu nejasnoću.

Dakle, potrebno je pogledati primjer kojim je dodan tenk u scenu, a sastoji se od:

- Inicijalizacije klase koja implementira tenk i sve potrebno za tenk
- Inicijalizacije čvora kojemu se odmah odgovarajućom funkcijom iz instance klase tenka dodjeljuje vrijednost

- Inicijalizacija čvorat matrice koja se isto tako postavlja pozivom odgovarajuće funkcije iz instance klase tenka
- Postavljanjem matrice kao dijete korijenskom čvoru
- Postavljanjem čvora u kojemu je tenk dijete čvoru matrice

Kôd koji implementira spomenuto izgleda ovako:

```
// ne zaboraviti incijalizirati glavni cvor tipa Group


// inicijalizacija tenka i njegove matrice za pozicioniranje

ovoTenk* oTenk = new ovoTenk;
osg::MatrixTransform* tenkMT = oTenk->tenkMatrixTransform();
osg::Node* tenkCvor = oTenk->stvoriTenk();



// dodaj tenk u scenu tako da najprije dodas matricu korijenu
// a zatim matrici dodaj cvor u komu je tenk

rootCvor->addChild(tenkMT);
tenkMT->addChild(tenkCvor);
```

Savjet za ubrzanje prevođenja kôda (nije nužno)

Prije realizacije može se odabrati Release verziju projekta. Iz padajućeg izbornika *Build* odabrati *Configuration Manager*; *Active Solution Manager* postaviti na *Release* i provjeriti da u tablici pod imenom projekta u stupcu *Configuration* piše *Release*. Nakon toga slijediti uputstva iz poglavlja 4.1 vezana uz postavljanje parametara projekta, ali sada za *Release* verziju (padajući izbornik pri vrhu *Properties* prozora). Napomena: ubrzanje se neće uvijek dogoditi.

6. Pomoćni materijali za izradu vježbe

Zbog toga što OSG nije komercijalan proizvod ne postoje formalni dokumenti koji opisuju postupak instalacije niti postupak izgradnje. Pomoć je jedino dostupna na listi elektroničke pošte (mailing listi) od strane korisnikâ OSG-a i u obliku web tutoriala. Na webu se mogu naći i opisi programskog koda generirani alatom Doxygen.

Na adresi http://www.nps.navy.mil/cs/sullivan/osgtutorials/index.htm nalazi se pregledan tutorial. Mi ćemo iskoristiti modele dane na toj stranici, **`JoeDirt`** teren, **`T72`** tenk i još nekoliko tekstura. Arhiva je dostupna na stranicama predmeta pod nazivom **resursi.zip**.

Projekt koji treba doraditi kako bi se riješio 3. zadatak također je dostupan na web stranicama predmeta u arhivi **zadatak3.zip**.

7. Predavanje rezultata vježbe

Rezultati vježbe se predaju zapakirani u arhivu ***OVO-V6- Rezultati-<ImePrezime>.zip*** koja treba sadržavati:

- Izvještaj o izvođenju vježbe s opisom postupka rješavanja zadatka, te sa slikama kako je naznačeno u 1., 2. i 3. zadatku.
- Napisani kod koji predstavlja rješenje 2. zadatka.
- Dobro komentiran source kod implementacije grafa scene iz 3. zadatka.
- Rješenja naprednih zadataka (opcionalno).

Navedena arhiva treba biti predana korištenjem *Web aplikacije za predaju vježbi* dostupne preko web stranica predmeta.

Napomena: Rezultati se šalju isključivo preko gore navedene aplikacije. U slučaju problema, javiti se email-om na adresu ovo@tel.fer.hr. Sačuvajte kopiju poslanih rezultata.

8. Napredni zadaci (ovaj dio nije obavezan)

Opis naprednih zadataka nalazi se u sklopu osnovnih zadataka u poglavlju 4, budući da se nadovezuju na zadane zadatke.