FERmula 1
Tehnička dokumentacija
Verzija <2.0>

Studentski tim: Adnan Abdagić
Toni Kork
Nikola Martinec
Petar Mrazović
Robert Mrkonjić
Željko Mijočević
Ana Nekić
Mario Volarević

Nastavnik: Prof.dr.sc. Željka Mihajlović

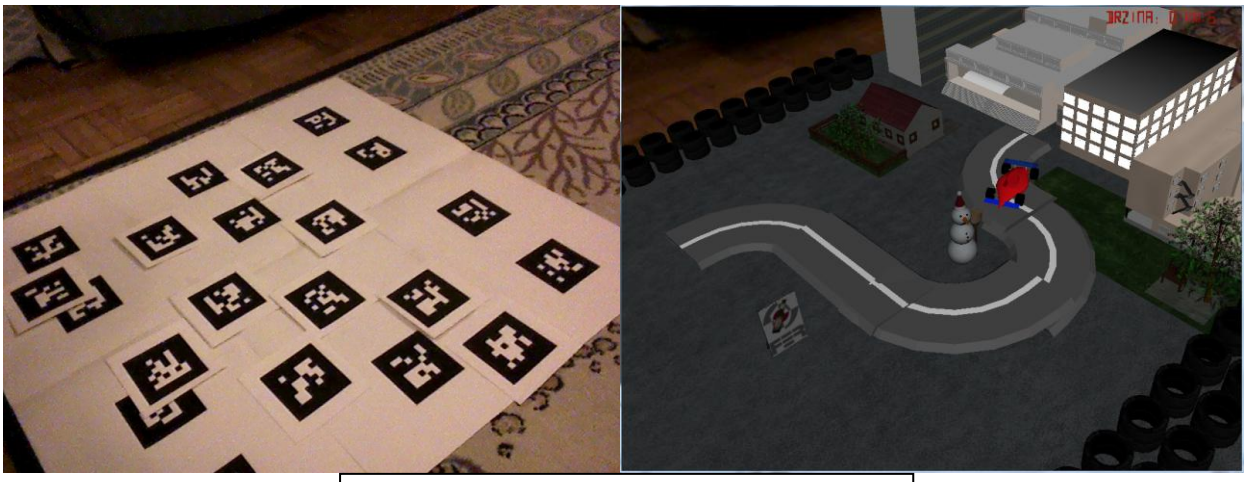
Sadržaj

1. Opis razvijenog proizvoda

Proizvod koji je nastao kao rezultat ovog projekta je program i nazvali smo ga: FERmula 1, tehnička demonstracija „proširene stvarnosti“. Iako je ovaj program izgledom i funkcionalnošću sličan igri ne može se tako nazvati jer mu nedostaju neki elementi poput početnog izbornika ili nekog specifičnog cilja koji je potrebno ostvariti, pa se zato naziva tehnička demostracija, a naslov FERmula 1 je nastao igrom riječi jer vozila podsjećaju na bolide formule 1 i okruženje je tematikom vezano za FER

Ovim programom smo htjeli na vizualno atraktivan i jednostavan način pokazati što je „proširena stvarnost“ i za što se može koristiti. Proširena stvarnost *(eng. Augmented Reality, AR)* dodaje virtualne elemente u stvarni svijet na taj način da izgledaju kao dio stvarnog svijeta. U ovom slučaju „proširenі“ dio stvarnosti proizlazi iz postavljanja virtualne scene FER-a u stvarni svijet i „drugačije“ interakcije sa virtualnim objektima (dijelovima ceste, plakatima, drvećem, snjegovićem itd.) pomoću objekata u stvarnom svijetu.

Pokretanjem programa i pravilnim postavljanjem kamere scena se pojavljuje na monitoru računala sa pozadinom stvarnog svijeta. Scena se sastoji od zgrade FER-a, parkirališta ispred FER-a, zida od guma koji se koristi kao granica i bolida FERmule 1 kojeg je moguće pomicati unutar granica scene. Dodavanjem markera[1] moguće je lako dodavanje novih modela na scenu i njihovo pomicanje, jer položaj markera odgovara položaju modela s kojim je povezan. Kako su na nekim markerima dijelovi ceste, a na drugima rekviziti (plakati, snjegović, klupa i drvo itd.), moguće je na jednostavan, intuitivniji način (korištenjem markera kao predstavnika virtualnih objekata, a ne pomoću tipkovnice i miša kako je inače slučaj) mijenjati konfiguraciju, tj. izgled scene i to u stvarnom vremenu, za vrijeme izvršavanja programa (Slika 1.).



Slika 1. Scena prije i poslije pokretanja programa

Za zanimljivije iskustvo je dodano nekoliko vozila koja je moguće voziti, no nažalost, sva imaju iste karakteristike ubrzavanja, brzine i mase. Ubrzavanje i inercija te zakretanje kotača su dodani zbog realističnijeg kretanja vozila. Ograničenje kretanja vozila je implementirano da vozilo ne bi izišlo izvan vidokruga kamere. U tu svrhu je korišten sustav kolizije između vozila i zidova od guma te vozila i zgrade FER-a. Također je dodana funkcija resetiranja pozicije FERmule zbog povremenog zaglavljivanja vozila nakon sudara s nekom od granica, najčešće kod bočnog sudara.

Osim video pozadine s kamere, u programu je odmah prilikom pokretanja vidljivo i jednostavno sučelje na kojem se prikazuje brzina čiji je prikaz moguće isključiti po želji. Uz to je moguće i pozvati dodatna sučelja pritiskom na odgovarajuće tipke čime se mogu dobiti dodatne statistike prilikom izvršavanja programa ili pomoć. Jedna od opcija je i prikaz FPS-a, ali je on unutar programa ograničen na 60 da bi se na svim računalima koja su dovoljno jaka vozilo kretalo jednako brzinom.

[1] marker – uzorak u stvarnom svijetu koji program koristi za prepoznavanje i orijentaciju virtualnih objekta

2. Korišteni alati

Nakon što smo se odlučili za temu i područje rada kojim ćemo se baviti da bi krenuli s radom potrebno je prvo odlučiti koje alate koristiti jer o njima ovisi koliko ćemo lako ili teško obaviti određeni posao.

U svakom projektu vezanom za grafiku i animaciju postoje modeli pa smo prvo odlučili koji program za modeliranje koristiti. Kako su neki od nas već imali iskustva s 3D modeliranjem i koristili su 3ds Max onda smo se odlučili za program *3ds Max 2010* koji je ujedno i poprilično jednostavniji od konkurentskih rješenja te ga početnici najlakše i najbrže nauče, a postoji i velika baza *„tutoriala"* koja olakšava učenje. Korištena je besplatna studentska verzija koju *Autodesk* nudi za download na svojim stranicama pod uvjetom da se ne koristi u komercijalne svrhe. Polovica tima je osim ostalih poslova radila i modele te smo ih tako dobili mnoštvo kojima smo mogli „ukrasiti" scenu. Teksture koje smo koristili su bile uglavnom ili proceduralno generirane unutar 3ds Maxa ili smo ih skidali s interneta iz nekog od mnoštva besplatnih repozitorija tekstura na internetu.

Odabir alata kojim bi ostvarili „proširenu stavarnost" je bio teži jer nitko nije imao iskustva u tom području. Tih alata je bilo nekoliko i svi su imali slabo napisanu dokumentaciju što bi veoma otežalo rad s njima. Na kraju je odlučeno da će biti korišten *osgART* (verzija 2.0 RC3, tada najnovija verzija) jer se čitajući dokumentaciju došlo do zaključka da pruža svu funkcionalnost koja je nama bila potrebna. *OsgART* je set alata i biblioteka otvorenog koda *(eng. open source)* pisanog u *C++* koji koristi *OpenGL* i sastoji se dva dijela: *ARToolkit-a* (v2.72) i *OpenSceneGraph-a* (v2.8.2).

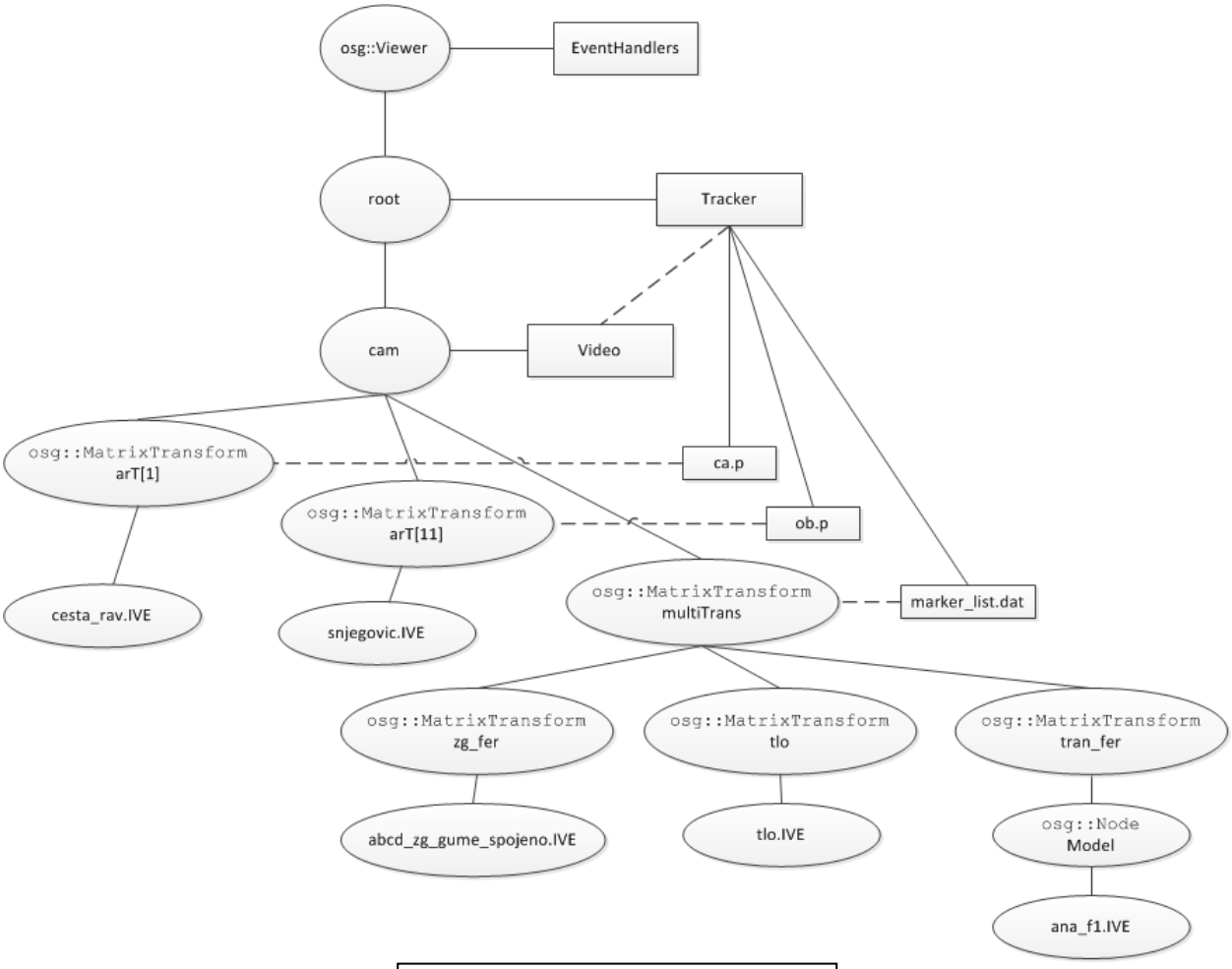
ARToolkit je dio koji obavlja poslove vezane za „proširenu stvarnost". U sebi sadrži funkcije za prepoznavanje i praćenje markera, podešavanje parametara kamere i dohvaćanje videa. Orginalno ga je razvio Dr. Hirokazu Kato, a kasnije su projekt preuzeli HIT Lab *(eng. Human Interface Technology Laboratory)* sveučilišta u Washingtonu, HIT Lab NZ sveučilišta u Canterburyu, Novi Zeland i ARToolworks, Inc, Seattle. Oni su razvili i *ARToolkit Pro* koji pruža naprednije mogućnosti poput boljeg praćenja markera, ali se plaća i nije otvorenog koda što nama ne odgovara.

Za sve ostalo se brinuo *OpenSceneGraph.* On je pružio grafički sustav, funkcije za učitavanje modela, transformacije matrica koje su bile potrebne za animaciju, sustav kolizije i još mnogo drugih stvari. Njegove značajke su velika brzina i učinkovitost, višeplatformska podrška i otvoreni kod koji se još uvijek nadograđuje (za vrijeme rada na ovom projektu već je izišla verzija 2.8.3, ali nju nismo mogli koristiti zbog kompatibilnosti). OSG je vrlo poznat alat (korišten je za izradu najboljeg open source simulatora letenja *Flightgear*) što sa sobom donosi veliku zajednicu korisnika od kojih se može tražiti pomoć na službenom forumu.

Za razvojnu okolinu smo trebali koristiti Visual Studio 2008 jer su to zahtijevale određene izvršne datoteke koje su dolazile uz *OpenSceneGraph*., a izabrali smo express verziju VS 2008 jer je besplatna, a ima sve što je potrebno.

3. Tehničke značajke

Korištenje *OpenSceneGrapha* unutar *osgART-a* je jako olakšalo rad s grafikom i programiranje u usporedbi s „čistim“ *OpenGL-om*, što je učinilo kod sažetim, lako čitljivim i preglednim. U *osgART-u* se sve radi s tzv. čvorovima *(eng. Node). Node* je najopćenitija vrsta čvora jer oni mogu predstavljati bilo kakav objekt u programu, npr. *MatrixTransform* je isto čvor, ali je to specijalna verzija *Node-a*. Povezivanjem svih čvorova (razni objekti koji se koriste u programu, modeli, matrice tranformacije, markeri...) se gradi stablo (slika 2.). Prilikom osvježavanja stanja programa u posebnoj klasi se putuje kroz to stablo i tada se prihvaćaju izmijenjeni podaci te se sve odvija na vrlo učinkovit način jer se točno zna gdje se što mora provjeravati.



Slika 2. Mali dio stabla iz programa

Od onog što osgART pruža mi smo koristili funkcije za:

- hvatanje događaja *(eng. event)* s tipkovnice
- kalibraciju kamere
- učitavanje slike s kamere
- učitavanje markera, modela, fontova...
- povezivanje markera s modelima i kamerom
- manipulaciju matrica za tranformacije koje su povezane s modelima i markerima

- određivanje granica modela i detekciju kolizije
- iscrtavanje sučelja
- osvježavanje podataka
- prikaz slike

U nastavku će biti dani kratki opisi kako su te funkcije bile implementirane u raznim dijelovima programa da se dobije željeni učinak što je na kraju dovelo do gotovog proizvoda.

3.1 Kamera i markeri

U svim programima koji prikazuju neku računalno generiranu sliku potrebno je ručno namještati poziciju kamere (treba je „okrenuti“ software-ski prema objektu koji želimo gledati), ali ne i kod „proširene stvarnosti“ jer se umjesto virtualne za scenu koristi stvarna kamera čiji položaj i orijentacija u stvarnom svijetu odgovaraju položaju i orijentaciji u virtualnom svijetu što određuje kakva će se slika prikazivati.

Da bi program prilikom okretanja kamere program znao gdje što treba prikazati jedna od mogućih tehnika je korištenje markera (ostale tehnike su pomoću GPS-a, akcelerometara, lasera itd.). Markeri su jedinstveni uzorci koji se nalaze na nekoj podlozi te služe kao referenca za virtualni položaj objekta, ali prije nego što se markeri mogu koristiti trebaju se prvo izraditi i registrati. Mi uzorke nismo izrađivali jer je moguće skinuti s interneta skupove već gotovih BCH markera sa stranica koje rade s AR, ali ih je zato trebalo registrirati. Postupak registracije markera se provodi s programom koji je došao uz *ARToolkit* gdje se uzorak na papiru „uslika“ kamerom te se takva slika sprema u specijalni format (slika 3.). Zatim se te datoteke učitavaju u glavni program pomoću sljedećih funkcija:

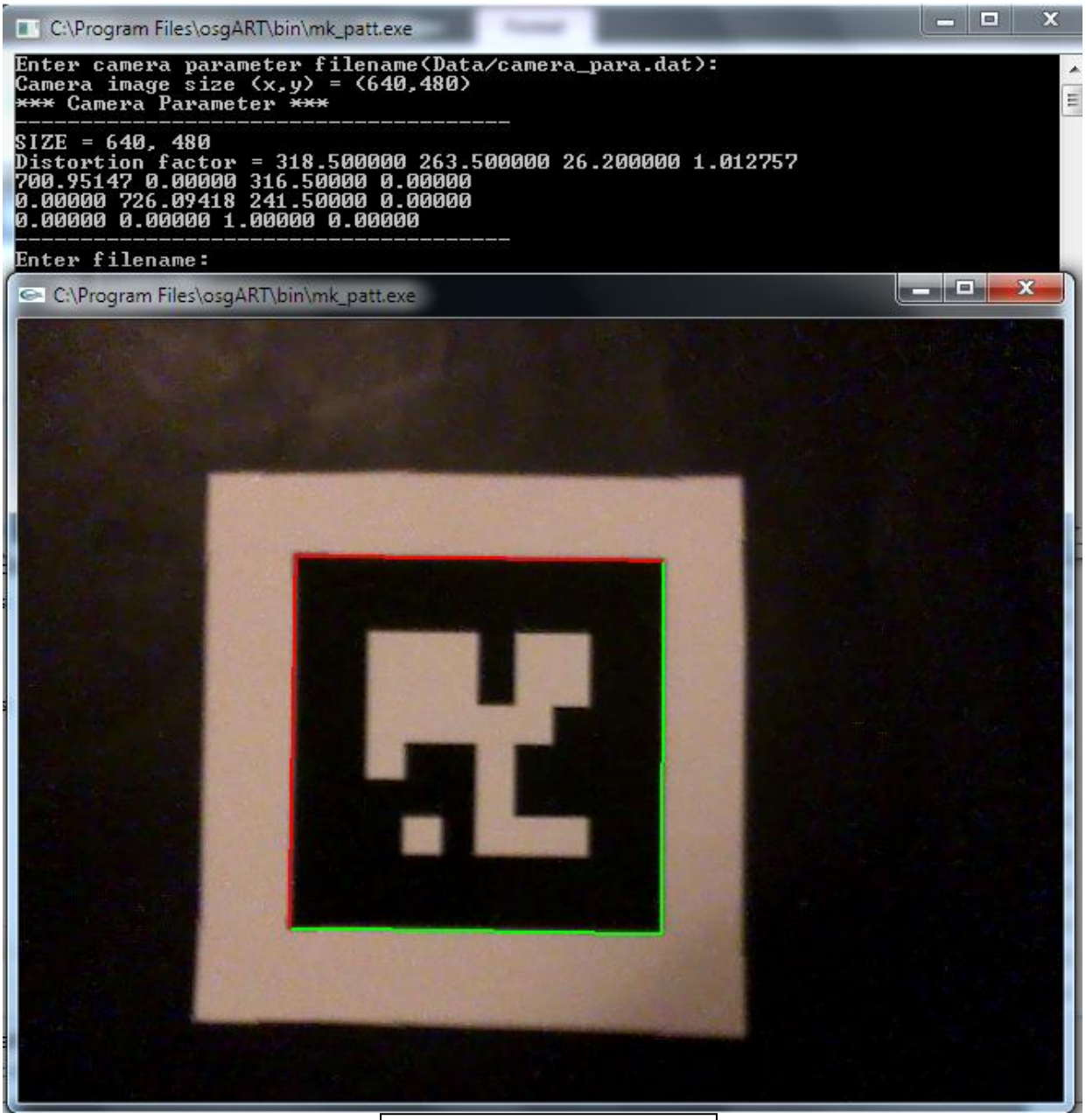
```
osg::ref_ptr<osgART::Marker> marker[br_markera];
marker[0] = tracker->addMarker("single;data/ca.p;60;0;0");
marker[1] = tracker->addMarker("single;data/cb.p;60;0;0");
marker[2] = tracker->addMarker("single;data/cc.p;60;0;0");
//... ostali markeri
for (int i=0;i<br_markera; i++)
      marker[i]->setActive(true);
```

Poslije aktiviranja markera, ali prije nego što možemo dodati modele moramo stvoriti čvorove s matricama tranformacije koje će sadržavati informaciju o položaju modela, a zatim treba povezati te čvorove s markerom tako da pomicanje markera utječe na vrijednosti u matrici:

```
osg::ref_ptr<osg::MatrixTransform> arT[br_markera];
for (int i = 0; i < br_markera; i++)
{
      arT[i] = new osg::MatrixTransform();
      osgART::attachDefaultEventCallbacks(arT[i],marker[i]);
      arT[i]->getOrCreateStateSet()->setRenderBinDetails(300, "RenderBin");
}
```

Nakon što su markeri i matrice tranformacije povezane možemo dodati modele:

```
for (int i = 0; i<5; i++){
      arT[i]->addChild(osgDB::readNodeFile("../../Modeli/cesta_rav.IVE"));
      arT[i+5]->addChild(osgDB::readNodeFile("../../Modeli/cesta_skr.IVE"));}
arT[10]->addChild(osgDB::readNodeFile("../../Modeli/reklama.IVE"));
arT[11]->addChild(osgDB::readNodeFile("../../Modeli/snjegovic.IVE"));
//...dodavanje ostalih modela
```



Slika 3. Registracija markera

Kao što se vidi u kodu, modeli se dodaju kao djeca čvora matrice transformacije koji je povezan s markerom što znači da će se, kao što je ranije rečeno, pomicanjem markera mijenjati matrica te će se te promjene prenositi na dijete, koje je zapravo model, a sve se to događa bez ikakave intervencije programera jer su najčešće korištene postavke za praćenje, koje odgovaraju i u ovom slučaju, prethodno kodirane u funkciji za povezivanje markera i transformacije.

U programu se koristi i specijalni marker koji se naziva multimarker i na njega je vezana osnovna scena (zgrada FER-a, tlo, gume, FERmula). Multimarker se sastoji od više pojedinačnih markera, ali se tretira kao jedan marker, koristi se u slučajevima kada ne možemo osigrati stalnu vidljivost svih markera jer ima svojstvo dok god je vidljiv bilo koji od markera od kojih se sastoji, program zna gdje treba postaviti koji model. Dodaje se na isti način kao što je prethodno opisano za obični marker, ali se umjesto imena markera u kod dodaje posebna datoteka u kojoj su zapisana imena i relativne udaljenosti markera od kojih se sastoji multimarker do proizvoljno odabrane točke.

3.2 Animacija kretanja

Vrlo važan dio tehničke demonstracije proširene stvarnosti zahtjeva implementaciju interakcije u stvarnom vremenu. U tu svrhu ostvareno je pokretanje vozila FERmula zadavanjem naredbi preko tipkovnice.

U nastavku je opisan način na koji je implementirano pokretanje FERmule: ubrzavanje, usporavanje, skretanje FERmule, te zakretanje kotača.

Kretanje modela se zasniva na upotrebi događaja(„events“). Pritiskom na tipku inicira se događaj koji pokreće niz akcija pomoću kojih se vrši kretanje modela naprijed, nazad, lijevo ili desno, ovisno koja je tipka pritisnuta. Otpuštanjem tipke, koja je prethodno bila pritisnuta, također se inicira niz akcija koje zaustavljaju prethodno pokrenutu funkciju.

U kodu se koriste tri klase za pokretanje modela. Klasa, imenom „MyKeyboardEventHandler“, služi za rukovanje različitih događaja koji će biti inicirani. Klasa, imenom „VoziloInputDeviceStateType“, sadrži skup varijabli koje se postave kada se dogodi događaj, te se pomoću njih određuje implementacija kretanja. Implementacija kretanja se nalazi u klasi imenom „UpdateVoziloPosCallback“. Sama implementacija kretanja je napravljena kao animacija koja se pokrene i izvrši bez mogućnosti da ju prekinemo.

3.2.1 Funkcije kretanja

Slijedi popis funkcija koje pokreću model. Uz svaku funkciju je dano objašnjenje i popis metoda neophodne za implementaciju funkcije.

RESET FUNKCIJA:

- resetira varijable i postavlja formulu na početnu poziciju
- koristi se *osg::Matrix::translate* metoda za micanje formule po x, y, z osi

```
if (voziloInputDeviceState->resetReq)
{
	vmt->setMatrix(osg::Matrix::identity());
	vmt->preMult(osg::Matrix::translate(osg::Vec3(150,-250,0)));
	v->brzina=0;
	v->vrijeme=0;
}
```

POKRET NAPRIJED:

- formula se pokreće naprijed ubrzavajući dok je tipka za naprijed stisnuta i usporava kad je tipka otpuštena
- formula ubrzava i usporava po formuli $\frac{x}{x+1}$, gdje je $x = v \rightarrow vrijeme$

```
if ((voziloInputDeviceState->moveFwdRequest) &&
	(!voziloInputDeviceState->moveBcwRequest))
{
	if (v->brzina<=(maxBrzina-1)&&(v->brzina>=0))
	{
		v->brzina=(maxBrzina*(v->vrijeme)/(v->vrijeme+1));
		v->vrijeme+=0.03f;
	}
	if (v->brzina<0)
	{
		v->brzina+=0.6;
	}
	vmt->preMult(osg::Matrix::translate(0,-(v->brzina),0));
}
```

POKRET NAZAD:

- formula se kreće unazad(„rikverc“) na isti način kao i naprijed, ali uz manju brzinu
- formula u ovom slučaju ubrzava i usporava po formuli $\frac{x}{1-x}$, gdje je $x = v \rightarrow brzina$

```
if ((voziloInputDeviceState->moveBcwRequest) &&
       (!voziloInputDeviceState->moveFwdRequest))
{
       if(v->brzina>=-maxBrzinaR)
       {
              if (v->brzina>0){
                     v->brzina-=0.7f;
                     v->vrijeme=v->brzina/(maxBrzina-v->brzina);
              }
              else v->brzina-=0.2;
       }
       vmt->preMult(osg::Matrix::translate(0,-(v -> brzina),0));
}
```

SKRETANJE LIJEVO:

- formula skreće lijevo za stalni kut u odnosu na z-os
- koristi se osg::Matrix::rotate metoda za okretanje za određeni kut oko x, y, ili z osi

```
if ((voziloInputDeviceState->rotLReq)&&(v->brzina>skretanjeLimit))
{
       vmt->preMult(osg::Matrix::rotate(osg::inDegrees(kutZakretanja),osg::Z_AXIS));
}
```

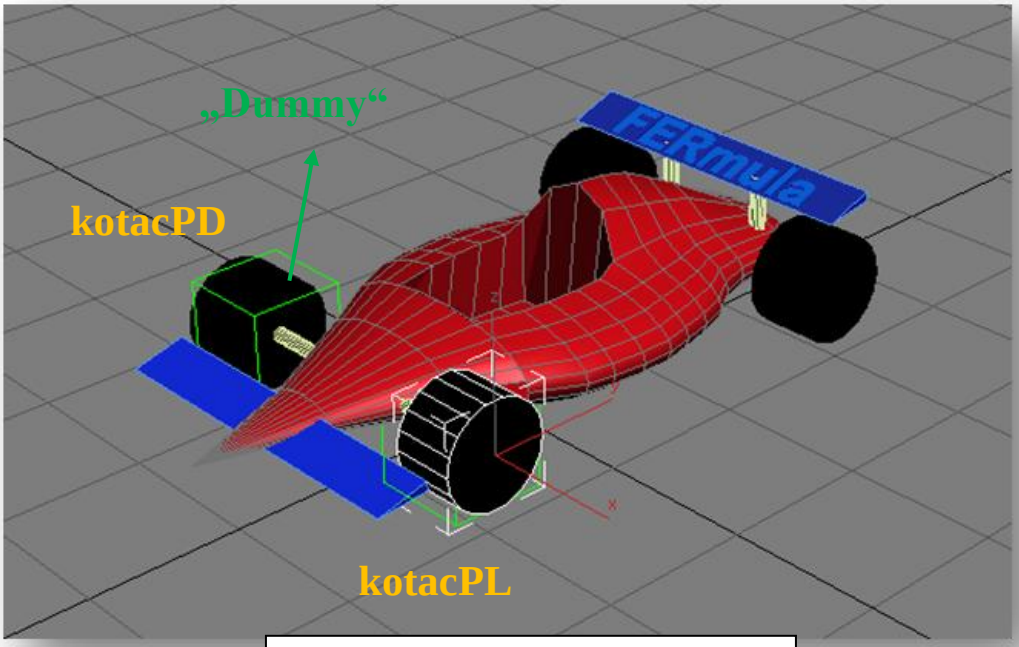
SKRETANJE DESNO:

- formula skreće desno za stalni kut u odnosu na z-os, identična načinu rada funkciji skretanje lijevo

```
if ((voziloInputDeviceState->rotRReq)&&(v->brzina>skretanjeLimit))
{
       vmt->preMult(osg::Matrix::rotate(osg::inDegrees(-kutZakretanja),osg::Z_AXIS));
}
```

3.2.2 Zakretanje kotača

Prilikom skretanja FERmule njezini prednji kotači pomiču se u odgovarajuću stranu, tj. rotiraju se za kut od 30° . Prilikom ostvarenja ove funkcionalnosti koristimo neke gotove funkcije. Funkcija *FindNodeByName* izdvaja nam dio modela čije ime predajemo funkciji kao argument. Nad pronađenim dijelom modela dalje je moguće vršiti transformacije. Da bi uspješno izdvojili dijelove modela, tj. kotače, potrebno ih je nazvati fiksnim imenima. U našem slučaju prednji kotači su nazvani *kotacPL* i *kotacPD*. Dodatno, funkcija *FindNodeByName* ograničava broj roditelja dijela modela koji se izdvaja. Iz tog razloga kotačima je pridodan točno jedan roditelj, nevidljivi objekt (tzv. *Dummy*) (slika 4.).



Slika 4. Model FERmule i imena kotača

Pritiskom na tipke za skretanje (KEY DOWN: left, right) poziva se odgovarajuća funkcija za zakretanje kotača (*okreniLijevo*, *okreniDesno*). Ukoliko su kotači već zakrenuti funkcije *okreniLijevo* i *okreniDesno* ne zakreću dodatno kotače, čime se sprečava beskonačna rotacija. Prilikom otpuštanja odgovarajućih tipki za skretanje (KEY UP: left, right) kotači se vraćaju u početni položaj, tj. rotiraju se za kut -30°.

```
case osgGA::GUIEventAdapter::KEY_Left:
{
voziloInputDeviceState->rotLReq = true;
if (v->okrenutL == 0)
v->okreniLijevo(true);
return false;
}
```

```
void okreniLijevo(bool t)
{
	if (t) okrenutL = 1;
	else okrenutD = 0;
	osg::Node* lijeviKotac = FindNodeByName( Model, "kotacPL" );
	osg::Node* desniKotac = FindNodeByName( Model, "kotacPD" );
	osg::MatrixTransform* ltr = AddMatrixTransform(lijeviKotac);
	osg::MatrixTransform* dtr = AddMatrixTransform(desniKotac);
	ltr->setMatrix(osg::Matrix::rotate(osg::inDegrees(30.0f),osg::Z_AXIS));
	dtr->setMatrix(osg::Matrix::rotate(osg::inDegrees(30.0f),osg::Z_AXIS));
}
```

```
osg::Node* FindNodeByName( osg::Node* pNode, const std::string& sName )
{
	if ( pNode->getName()==sName )
	{
		return pNode;
	}
	osg::Group* pGroup = pNode->asGroup();
	if ( pGroup )
	{
		for ( unsigned int i=0; i<pGroup->getNumChildren(); i++ )
		{
			osg::Node* pFound = FindNodeByName( pGroup->getChild(i), sName );
			if ( pFound )
			{
				return pFound;
			}
		}
	}
	return 0;
}
```

3.3 Detekcija kolizije

Detekcija kolizije je postupak određivanja sraza između dva ili više objekata na sceni. Zbog velike kompleksnosti detekcije kolizije između stvarnih i virtualnih objekata, odlučili smo implementirati samo detekciju kolizije između virtualnih objekata.

Budući da je detekcija kolizije procesno vrlo zahtjevna operacija ako se izvodi nad detaljnim modelima, vrlo često se ti modeli aproksimiraju jednostavnim geometrijskim tijelima čiji gabariti odgovaraju gabaritima modela. Te aproksimacije nazivaju se kolizijskim volumenom *(engl. collision volume)*.

Open Scene Graph podržava dva tipa kolizijskih volumena: kuglu i kvadar. Također posjeduje funkcije za određivanje kolizijskog volumena za zadani model te ispitivanje presjeka dva kolizijska volumena istog tipa.

Implementacija detekcije kolizije

Detekcija kolizije implementirana je u istoj funkciji koja i pomiče vozilo po virtualnoj sceni. Nakon svakog pomaka ispituje sa da li je došlo do kolizije između vozila i osnovne scene (zgrada FER-a i gume oko parkirališta). Ukoliko je taj uvjet ispunjen vozilo sa vraća u posljednji položaj u kojemu nije bilo kolizije te se brzina vozila postavlja na nulu. Ispitivanje kolizije sastoji se od određivanja pozicije kolizijskog volumena modela vozila, te tražanjem presjeka tog volumena sa kolizijskim volumenima osnovne scene. U svrhu određivanja kolizijskog volumena mogu mogu se koristiti funkcije `osg::MatrixTransform -> getBound()` koja vraća kuglu ili `osg::ComputeBoundsVisitor -> getBoundingBox()` koja vraća kvadar. Samo ispitivanje presjeka dvaju volumena određuje se funkcijom `osg::BoundingBox -> intersects (osg::BoundingBox)` ukoliko je riječ o kvadrima ili `osg::BoundingSphere->intersects(osg::BoundingSphere)` ukoliko je riječ o kuglama. Ispitivanje između kugli i kvadara nije podržano.

Ukoliko je došlo do kolizije vozilo translatiramo i rotiramo u suprotnim smjerovima od trenutnih te postavljamo brzinu i rotaciju na nulu. Na taj način izbjegavamo da se vozilo zaglavi u sceni.

3.4 HUD sučelje

HUD je skraćenica koja dolazi od engleskog izraza „heads – up display“. To je transparentni prozor koji prikazuje određene informacije. Osnovna osobina mu je da se prikazane informacije vide, a da korisnik ne izgubi cjelokupnu scenu iz vida. HUD je u početku razvijen za vojne avijacije te odatle i potiče ime. Naime, uz pomoć HUD-a piloti su bili u mogućnosti vidjeti informacije sa glavom gore (eng. heads up) tj. gledajući ravno, umjesto gledajući dolje na instrumente.

HUD se danas, osim u vojnom zrakoplovstvu, koristi i u komercijalnom zrakoplovstvu, automobilima i drugim aplikacijama (npr. igre).

U ovom projektu HUD sučelje sačinjava prikaz opcija i prikaz brzine vozila. Što će se prikazati određuje se pomoću tipkovnice. Veza između tipkovnice i akcija je ostvarena u metodi *handle* klase *KeyboardHandlerForHUD*.

3.4.1 Izgradnja HUD sučelja

HUD sučelje je samo još jedno podstablo cjelokupnog stabla koje čini cijelu scenu. Za HUD podstablo je potrebno u početku definirati korijen kojeg sačinjavaju odgovarajuća *projection* i *model view* matrica. Prva nam služi za prikaz HUD sučelja u ukupnoj sceni, dok nam druga omogućava njegovo pozicioniranje. Sljedeći kod prikazuje definiranje korijenâ HUD podstabla:

```
osg::Projection* HUDProjectionMatrix = new osg::Projection;
HUDProjectionMatrix->setMatrix(osg::Matrix::ortho2D(0,1024,0,768));

osg::MatrixTransform* HUDModelViewMatrix = new osg::MatrixTransform;
HUDModelViewMatrix->setMatrix(osg::Matrix::identity());
HUDModelViewMatrix->setReferenceFrame(osg::Transform::ABSOLUTE_RF);

root->addChild(HUDProjectionMatrix);
HUDProjectionMatrix->addChild(HUDModelViewMatrix);
```

Nakon definiranih korijenâ, moguće je dodavati proizvoljni broj djece, ovisno koliko elemenata sadrži željeno sučelje. Tako je u ovom sučelju bilo potrebno dodati jedno za prikaz opcija i drugo za prikaz brzine (oba su tipa *osg::Geode*).

3.4.2 Prikaz opcija

Prikaz opcija sadrži jedan geometrijski objekt (*HUDGeometry*) i jedan objekt za tekst (*HUDText*). Uz pomoć *HUDGeometry* se iscrtava pravokutnik, a preko HUDText se mijenja i ispisuje željeni tekst.

```
	osg::Geode* HUDGeode = new osg::Geode();

	osgText::Text* HUDText = new osgText::Text();
	osg::Geometry* HUDGeometry = new osg::Geometry();

	HUDGeode->addDrawable(HUDGeometry);
	HUDGeode->addDrawable(HUDText);

	HUDGeometry->setNormalArray(HUDnormals);
	HUDGeometry->setNormalBinding(osg::Geometry::BIND_OVERALL);
	HUDGeometry->addPrimitiveSet(HUDIndices);
	HUDGeometry->setVertexArray(HUDVertices);
	HUDGeometry->setColorArray(HUDColors);
	HUDGeometry->setColorBinding(osg::Geometry::BIND_OVERALL);

	HUDText->setAxisAlignment(osgText::Text::SCREEN);
	HUDText->setPosition(osg::Vec3(450,740,-1));
	HUDText->setColor(osg::Vec4(1,0,0,1));
```

3.4.3 Prikaz brzine

Prikaz brzine sadrži samo jedan objekt za tekst. Uzeta je vrijednost brzine vozila i ubačena u željenu formu ispisa te na kraju jednostavno ispisana na fiksnoj lokaciji u sceni (gornji desni kut).

```
osg::Geode* HUDBrzina=new osg::Geode();
osgText::Text* brzina=new osgText::Text();
HUDBrzina->addDrawable(brzina);

brzina->setPosition(osg::Vec3(800,740,-1));

void ispisBrzina(Vozilo* fermula){
	std::string brz;
	char b[5];
	itoa(fermula->brzina*2,b,10);
	brz.append(b);
	brz.append(" km/s");
	brzina->setText("Brzina: "+brz);
	return;
}
```

Na kraju, kad je izgrađeno podstablo za HUD sučelje, potrebno je osigurati da kad se prikazuje, da se iscrtava iznad svega. To se osigurava pomoću opcije *setRenderBin*.

```
	HUDStateSet->setRenderBinDetails(500,"RenderBin");
```

3.4.4 Metoda handle

Preko metode *handle* određuje se koji dio HUD sučelja se želi prikazati. Za prikaz opcija koristi se tipka 'o' dok se za prikaz brzine koristi tipka 'b'. Uz njih postoje i druge mogućnosti koje se pojave nakon otvaranja opcija.

```
bool KeyboardHandlerForHUD::handle(const osgGA::GUIEventAdapter& ea,
       osgGA::GUIActionAdapter&){
       switch(ea.getEventType()){
              case(osgGA::GUIEventAdapter::KEYDOWN):{
                     switch(ea.getKey()){
              case 'o':
                     _geode->setNodeMask(0xffffffff - _geode->getNodeMask());
                     _text->setText("Options: for more help press <p>");
                     _text->setPosition(osg::Vec3(10,640,-1));
                     return false;
                     //...
```

4. Izrada 3D modela

Za izradu 3D modela odabran je alat Autodesk 3ds Max 2010. Iako Open Scene Graph podržava mnoge formate 3D modela korišten je .IVE format da bi bila osigurana maksimalna kompatibilnost. Stoga su .max formati modela eksportirani koristeći dodatak za 3ds Max – OsgExporter. OsgExporter prebacuje modele u Open Scene Graph formate.

4.1 Autodesk 3ds Max

Autodesk 3ds Max je alat za integrirano 3D modeliranje, animaciju, renderiranje i kompoziciju složenih vizualnih scenografija, a osmišljen je prvenstveno za potporu umjetnicima i grafičkim dizajnerima. Jezgra alata je objedinjavanje tehnologije i značajki, kao i širokog spektra specijaliziranih alata namijenjenih podršci razvijateljima računalnih igara i vizualnih efekata umjetnika, kao i arhitektima i inženjerima.

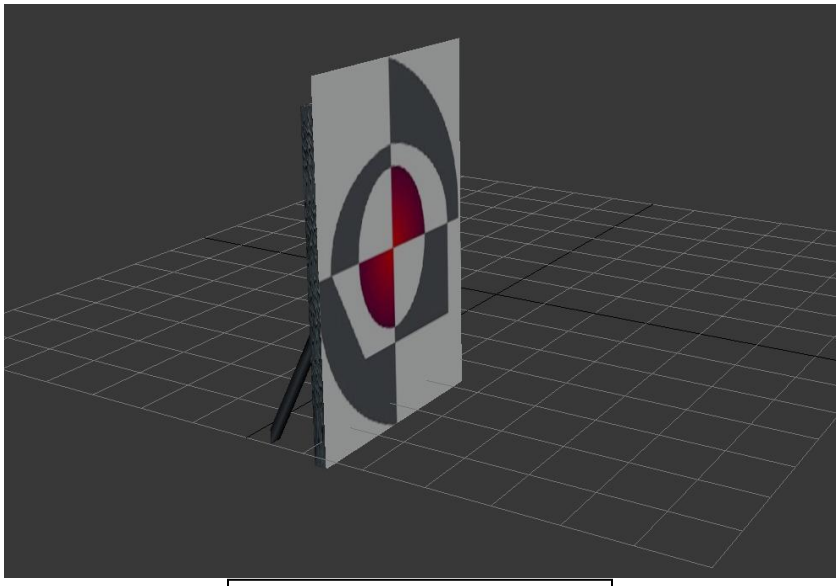
4.2 Modeliranje objekata

Postupku modeliranja prethodilo je osmišljavanje plana izgleda scenografije. Osnovna ideja igre jest vožnja vozila Fermula po improviziranoj stazi. Fermula je napravljena prema uzoru na klasični bolid Formule 1. Staza je smještena u prostor okolice Fakulteta elektrotehnike i računarstva. Prilikom modeliranja objekata nastojalo se što vjernije prikazati stvarni prostor. U svrhu poboljšanja vizualnog dojma staze, dodani su pojedini imaginarni elementi, primjerice model snjegovića, zaštitni zid od guma i reklamni pano.

Izrada modela započinje ubacivanjem standardnog objekta, najčešće pravokutnika ili cilindra, koji se potom editira ovisno o željenom ishodu. Nakon pretvorbe u editable poly, pristupa se modifikaciji njegovih diijelova površine, rubova ili pojedinih linija, odnosno točaka. Najčešće korišteni alati su izvlačenje (extrude), izvlačenje duž linije (extrude along spline), ukošavanje (bevel), izrezivanje (cut) i zaglađivanje (smooth). Nakon oblikovanja, modeli su nadopunjeni teksturama (najčešće u .tga ili .png formatu jer podržavaju transparentnost i jer se lako skaliraju bez većeg gubitka informacije, ali nedostatak naspram .jpg-a je veličina datoteke) i gotovim dijelovima ugrađenim u alat Autodesk 3ds Max 2010.

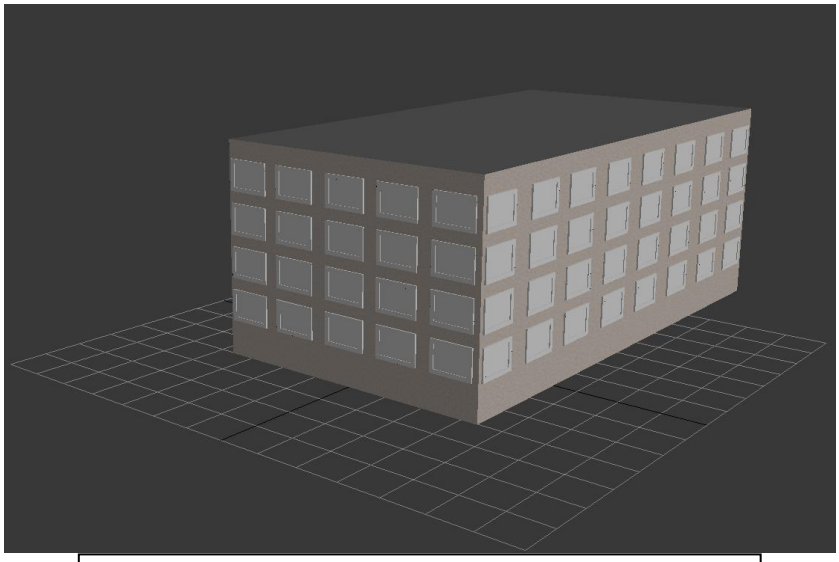
Modeli:

Reklamni pano sastoji se od ravnine i dva držača. Držači su kreirani od pravokutnika, koji su modificirani uz pomoć alata izvlačenje duž linije (extrude along spline). Model je nadopunjen teksturom metala i logotipom Fakulteta elektotehnike i računarstva.



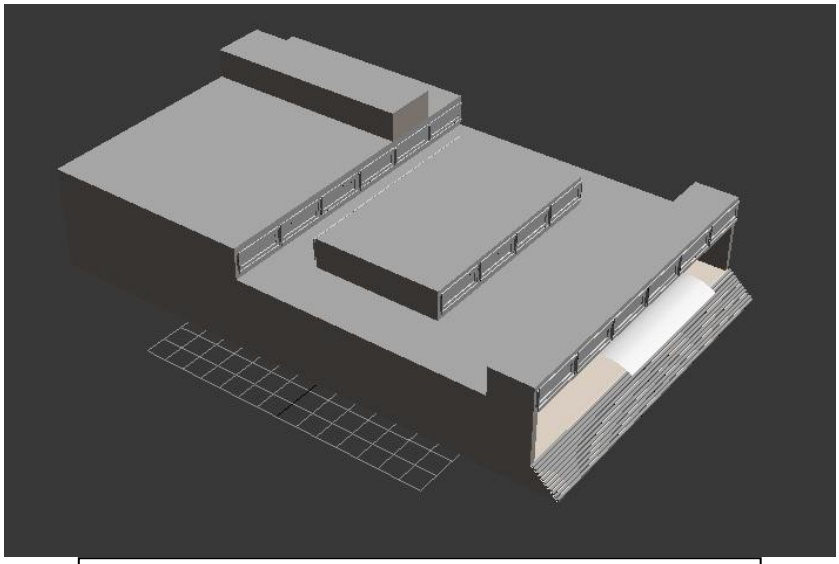
Slika 5. Reklamni pano

A zgrada Fakulteta elektrotehnike i računarstva modelirana je iz standardnog pravokutnika, kojem su dodani prozori ugrađeni u alat Autodesk 3ds Max 2010. Model je nadopunjem odgovarajućom teksturom.



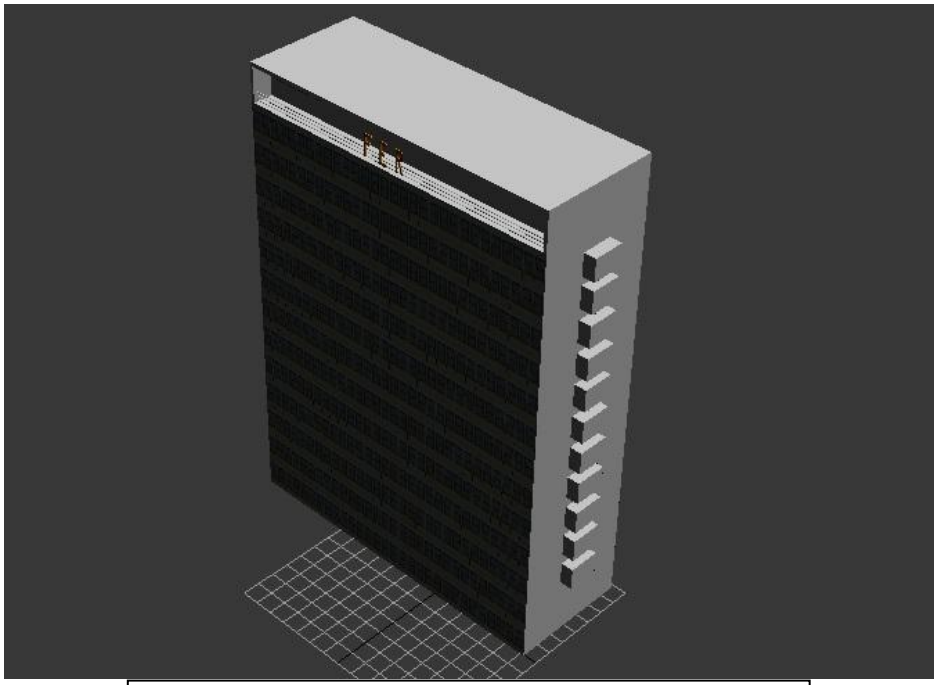
Slika 6. A zgrada Fakulteta elektrotehnike i računarstva

B zgrada ima kao bazu također standardni pravokutnik koji je modificiran da poprimi oblik dotičnog objekta. Stepenice, vrata i prozori su uzeti iz standardne biblioteke alata Autodesk 3ds max.



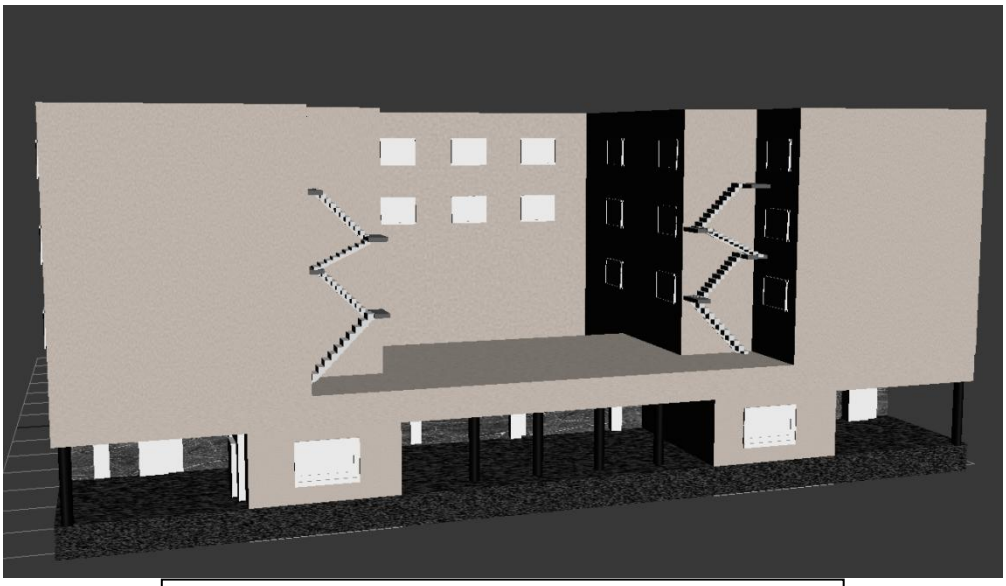
Slika 7. B zgrada Fakulteta elektrotehnike i računarstva

C zgrada FER-a se sastoji od nekoliko primitivnih objekata. Bazu čini kvadar, a cilindri su korišteni za prikaz ograde na zadnjem katu. Za modeliranje protupožarnih balkona/stepenica su korišteni obični kvadri ukrašeni cilindrom. Za prozore je uzeta tekstura jednog prozora C zgrade i skladno raspoređena na obje strane tako da zgrada ima 13x30 prozora. Na zadnji kat, konretno balkon C zgrade je dodan i natipis „FER“. Za njega je napravljena žuto-narančasta tekstura kako bi dala dojam svijetljenja.



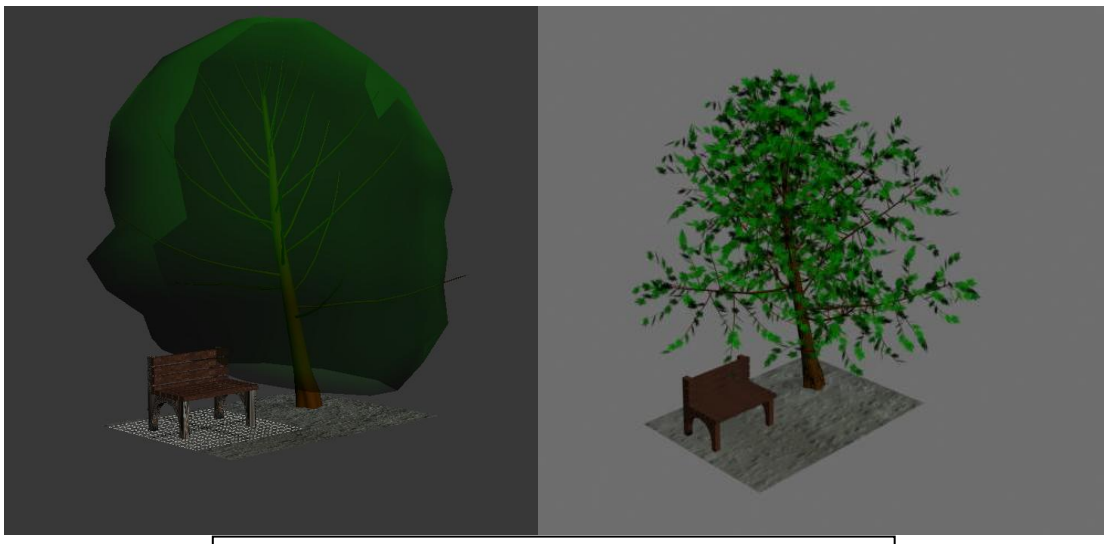
Slika 8. C zgrada Fakulteta elektrotehnike i računarstva

D zgrada Fakulteta elektrotehnike i računarstva modelirana je iz nekoliko standardnih pravokutnika, koji su modificirani alatima izvlačenja (extrude) i izrezivanja (cut). Prilikom izrade modela korišteni su cilindri (za modeliranje stupova koji se nalaze s prednje strane zgrade) i ugrađeni prozori. Evakuacijske stepenice složene su koristeći ugrađene objekte alata Autodesk 3ds max 2010. Tekstura D zgrade identična je prethodnoj teksturi A zgrade.



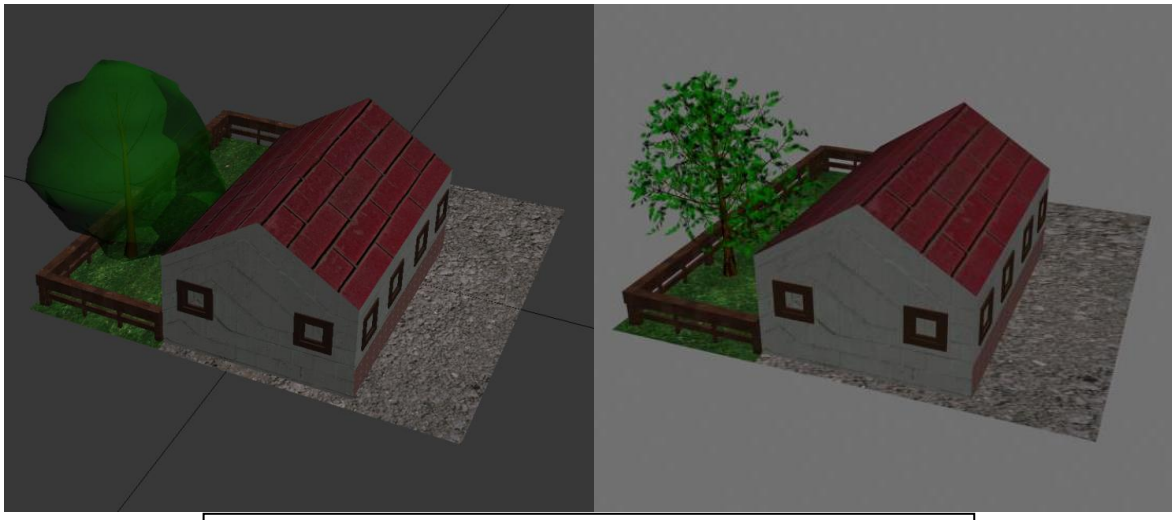
Slika 9. D zgrada Fakulteta elektrotehnike i računarstva

Klupa je modelirana iz pravokutnika i cilindra. Prvo je kreiran standardni cilindar, čiji je plašt podijeljen u 12 segmenata. Potom je alatom izrezivanja (cut) odstranjena polovica cilindra, a nakon toga su istim alatom odstranjene površine preostalih polukrugova. Ostatak plašta modificiran je alatima izvlačenja (extrude) i povezivanja (bridge), kako bi se dobio odgovarajući oblik nogu. Dijelovi su zaobljeni alatom chamfer. Daske su modelirane iz pravokutnika. Model je nadopunjen teksturama drveta i granita, kao i ugrađenim objektom drveta.



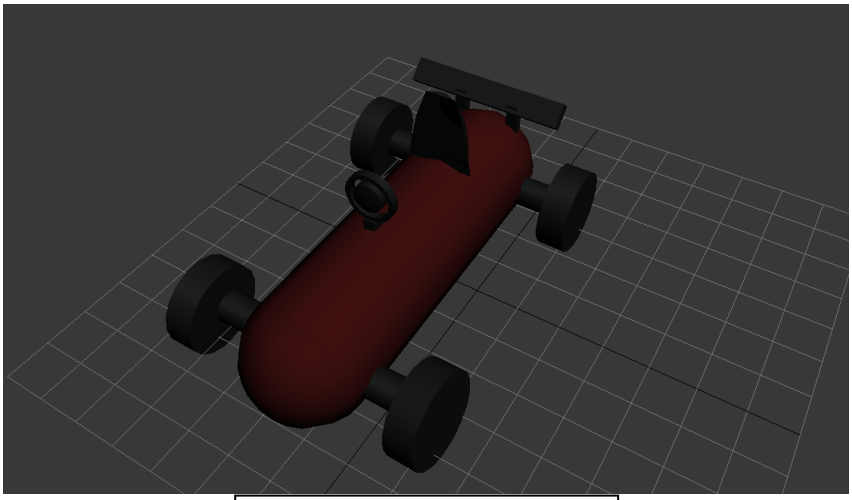
Slika 10. Klupa s drvetom prije i poslije „renderiranja“

Kuća je modelirana iz jednog standardnog pravokutnika. Krov kuće oblikovan je povezivanjem krajnjig bridova gornje površine pravokutnika, te izvlačenjem novonastalog brida alatom extrude. Model je nadopunjen ugrađenim prozorima, drvetom i ogradom. U model su unešene teksture opeke, kamena, trave, drveta i granita.



Slika 11. Susjedna kuća s drvetom prije i poslije „renderiranja“

Prva verzija fermule modelirana je iz nekoliko pravokutnika i cilindara, te jedne cijevi. Prilikom modeliranja korišteni su alati izrezivanja (cut), topljenja (melt) i zaglađivanja (smooth).

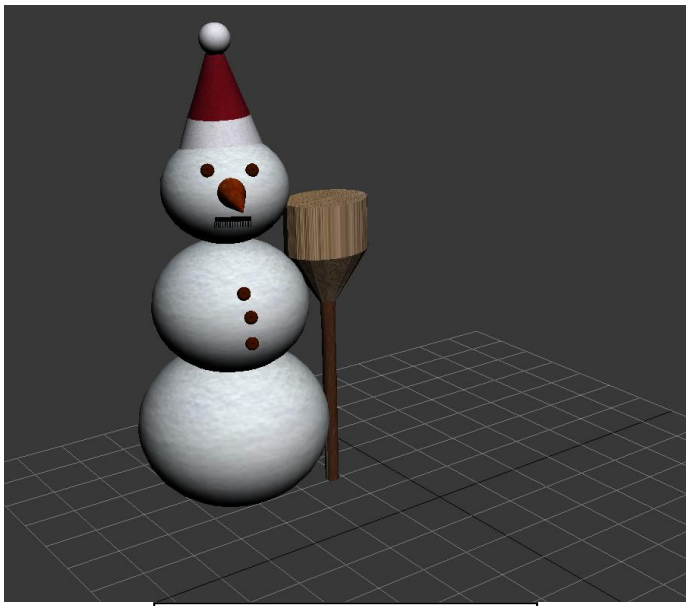


Slika 12. Fermula verzija 1

Guma je modelirana iz cilindra, kojemu su odstranjeni dijelovi kružnica s prednje i stražnje strane. Bridovi su zaobljeni odgovarajućim alatom (chamfer edge). Model je nadopunjem teksturom gume. Zaštitni zid nastao je višestrukim kloniranjem modelirane gume.

Slika 13. Zaštitni zid od guma

Snjegović je modeliran iz tri standardne sfere, koje su spljoštene skaliranjem. Oči i gumbi snjegovića modelirani su standardnim oblikom torus. Kapa snjegovića oblikovana je uporabom konusa i sfere. Nos snjegovića također je nastao je modeliranjem konusa. Metla je modelirana iz standardnog cilindra, uz pomoć alata za ukošavanje (bevel) i izvlačenje (extrude). Model je nadopunjen odgovarajućim teksturama crvene i bijele tkanine, snijega, mrkve, pruća i drveta.



Slika 14. Snjegović

5. Upute za korištenje

- Prvo je potrebno nabaviti svu opremu: računalo, web kameru, printer i datoteku za instalaciju programa.
- Zatim se pokrene instalacija te se izabere direktorij gdje želimo instalirati program.
- Nakon toga se isprintaju uzorci za markere iz direktorija *Uzorci*, datoteka *Markeri.pdf* (prvih 6 stranica s markerima tvori tzv. multimarker na kojem se nalazi glavna scena i oni se moraju poslagati kao što je naznačeno u *Multimarker.pdf* gdje prva stranica u *Markeri.pdf* odgovara A1, druga A2, treća B1 itd.)
- Ostale 3 stranice sadrže markere za ravne dijelove ceste, za zavoje i za „rekvizite“
- U opcijama printanja treba obavezno namjestiti *Page scaling* na **None** da bi veličina markera bila ispravna
- Kad smo sve pripremili možemo pokrenuti program, nalazi se u *\Bin\FERmula_1_bin\FERmula_1.exe* i kad se pokrene prvo pita za podešavanje parametara kamere
- Sada na pod postavimo *multimarker* te prema njemu uperimo kameru (ovisno o kameri potrebna je različita razina osvijetljenja, ali što je svjetlije, to bolje)
- Kameru okrećemo dok se ne pojavi scena
- Preko multimarkera možemo postavljati druge markere čime mijenjamo izgled scene
- Također je moguće okretati kameru kako želimo dok god su markeri dobro vidljivi i program ih prepoznaje
- Za upravljanje FERmulom se koriste tipke na tipkovnici:
    - strelica gore za naprijed
    - strelica dolje za nazad
    - strelice lijevo i desno za skretanje
    - F1 za reset pozicije
- Tipka *o* otvara izbornik s nekim opcijama programa
- Tipka *h* otvara *help* izbornik koji nudi kratice programskog alata, npr. *s* za *statistike*, *f* za *fullscreen toggle* (odabir hoće li slika biti preko cijelog ekrana ili u prozoru) itd.
- Pritiskom na tipku *Esc* se izlazi iz programa
- Ako želimo deinstalirati program dovoljno je dvoklikom pokrenuti skriptu *uninstall.bat*

6. Zaključak i budući razvoj

„Proširena stvarnost“ je zanimljiva tehnologija, ali u ovakvom obliku s markerima još nije dovoljno razvijena da uđe u širu primjenu jer nije zgodno printati ili nositi markere svaki put kad želimo nešto prikazati, a ako želimo gledati „prošireno“ potrebno je nositi nezgrapne naočale s kamerom i ekranima. No postoje i drugi problemi, poput nepreciznog praćenja i gubljenja slike, ali zato se razvijaju i koriste druge tehnike, kao npr. određivanje lokacije pomoću GPS-a i orijentacije pomoću akcelerometra ili žiroskopa. Te tehnike su nekoć bile jako skupe i stoga neiskoristive u komercijalnoj primjeni, ali danas, kako je tehnologija napredovala i pojeftinila, svaki „pametni telefon“ ima sve što mu je potrebno da bi koristio AR, čak i u tom naprednijem obliku.

Iako je za naš projekt korištenje markera dovoljno dobro, svejedno postoji prostor za unaprijeđenje. Između ostalog, mogla bi se dodati realističnija fizika i kolizija s drugim objektima na sceni, a ne samo s granicama. Također bi se mogle napisati nove funkcije za praćenje koje bi preciznije pratile markere te se modeli na njima ne bi toliko često gubili, a pozicija modela na sceni bi više odgovarala markeru u stvarnom svijetu. Mogle bi se optimizirati performanse programa čime bi se omogućile kompleksnije scene i više sadržaja te podrška za više korisnika. Svime navedenim i još mnogim drugim poboljšanjima ovaj projekt bi se iz tehničke demonstracije mogao pretvoriti u pravu igru, ali to će morati pričekati neke bolje dane.

7. Literatura