Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Aljaž Pintar

Izbira in uporaba odprtokodnega pogona za razvoj računalniške igre

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: doc. dr. Peter Peer
Asistent: as. Bojan Klemenc, univ. dipl. inž.

Ljubljana 2011

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00149/2011
Datum: 02.09.2011

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **ALJAŽ PINTAR**

Naslov: **IZBIRA IN UPORABA ODPRTOKODNEGA POGONA ZA RAZVOJ RAČUNALNIŠKE IGRE**

DEVELOPMENT OF A COMPUTER GAME USING AN OPENSOURCE GAME ENGINE

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Pri izdelavi računalniške igre si lahko prihranimo precej časa, če vzamemo že obstoječ pogon, na katerem zgradimo svojo igro. Preučite odprtokodne pogone iger. Zamislite si krajšo predstavitveno igro, ki bi jo razvili s pomočjo pogona. Pogone primerjajte med seboj, ovrednotite in izberite najustreznejši pogon, s katerim razvijete igro, ki bo predstavila lastnosti pogona. Pri primerjavi pogonov upoštevajte tako tehnične lastnosti (npr. podpora za programske vtičnike) kot netehnične lastnosti (npr. licence).

Mentor:
doc. dr. Peter Peer

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:
prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Aljaž Pintar, z vpisno številko **63060063**, sem avtor diplomskega dela z naslovom:

Izbira in uporaba odprtokodnega pogona za razvoj računalniške igre.

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Petra Peera in asistenco Bojana Klemenca,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 11. september 2011 Podpis avtorja:

Zahvalil bi se vsem, ki so mi pomagali pri izdelavi diplomske naloge, še posebej mentorju prof. Petru Peeru in asistentu Bojanu Klemencu. Zahvalil bi se tudi svojim staršem in punci Urški, ki so mi v času študija in pisanja diplomskega dela vedno stali ob strani, me podpirali in mi pomagali.

Kazalo

Seznam uporabljenih kratic

- OGRE (angl. Object-Oriented Graphics Rendering Engine) objektno usmerjen grafični pogon
- API (angl. Application programming interface) programski vmesnik
- HLSL (angl. High Level Shader Language) visokonivojski senčilni jezik
- GLSL (angl. OpenGL Shading Language) visokonivojski senčilni jezik
- CG (angl. C for Graphics) visokonivojski senčilni jezik podjetja Nvidia
- LOD (angl. Level of Detail) nivoji podrobnosti
- GPU (angl. Graphics processing unit) grafična procesna enota
- HUD (angl. Heads on Display) polprozoren prekrivni zaslonski prikaz; uporabniku ni potrebno zamenjati pogleda s točke, na katero je osredotočen, da vidi podatke na tem prikazu
- 3D trirazsežen
- OS (angl. operating system) opracijski sistem
- BSP (angl. binary space partitioning) binarno razdeljevanje prostora
- IDE (angl. integrated development environment) integrirano razvojno okolje
- LWGJL (angl. Lightweight Java Game Library) javina odprtokodna knjižnica za razvijanje iger

- FPS (angl. Frames per second) število izrisanih sličic na sekundo

Povzetek

Kadar se lotevamo razvoja računalniške igre, je koristno, če lahko začnemo razvoj na že obstoječem pogonu, ki nam prihrani programiranje najbolj osnovnih funkcij, ki jih srečamo v vsaki igri. V diplomskem delu smo tako opisali in primerjali različne odprtokodne 3D pogone za razvijanje računalniških iger z namenom izbire najustreznejšega za 3D računalniško različico namizne igre dama. Pri primerjavi odprtokodnih pogonov smo se omejili na pet pogosteje uporabljanih pogonov in zgradili odločitveni model, kjer smo vsak pogon ovrednotili in izbrali najbolj ustreznega. V igri smo prikazali različne lastnosti 3D pogona. Za občutek trirazsežnosti igre smo poskrbeli z uporabo dinamičnih animiranih senc, uporabili smo različne materiale in modele osvetlitve ter animacijo.

Ključne besede:

razvoj računalniške igre, igralni pogon, OGRE3D

Abstract

When we start to develop a computer game we can benefit a lot if we start the development with an existing game engine. A game engine spares us the development of most basic functionalities that need to be present in every game to function. In this thesis we evaluated different 3D open source game engines with the goal of choosing the most appropriate engine for developing a 3D computer version of the desktop game Checkers. We limited to comparison to five commonly used engines and devised a decision model to evaluate them and choose the most appropriate one. The game shows various functionalities of the 3D engine – for the feel of 3-dimensionality we employ dynamic animated shadows, different materials with lightning models and animation.

Key words:

Computer game development, game engine, OGRE3D

Poglavje 1

Uvod

V današnjem času so računalniške igre več milijard vreden trg, ki izziva filmski trg v velikosti in popularnosti [2]. Programska oprema, ki poganja 3D svetove – igralni pogoni kot na primer Quake, id Tech 4, Unreal Engine 3 in Source – so postali programsko-razvojni paketi, ki jih lahko licenciramo in uporabimo za izdelavo kakršnekoli igre.

Pogoni iger se v svoji arhitekturi in izvedbi med seboj razlikujejo. Pojavljajo pa se prepoznavni vzorci med licenčnimi igralnimi pogoni. Praktično vsi igralni pogoni vsebujejo zbirko ključnih komponent, kot so grafični pogon, fizikalni pogon, animacijski sistem, zvočni sistem, sistem za umetno inteligenco itd.

Kaj je igralni pogon? Izraz se je pojavil sredi devetdesetih let 20. stoletja v kombinaciji s popularno igro Doom podjetja id Software. Doom je bil zasnovan z jasno določeno mejo med glavnimi komponentami pogona (3D grafični sistem, sistem za zaznavanje trkov, zvočni sistem itd.), igralnimi svetovi in pravili igranja igre. Uporabnost oziroma vrednost pogona je s to ločitvijo postala vidna, ko so izdelovalci pogona izdali licenco za izdelavo novih iger drugim izdelovalcem iger. Nove igre so tako lahko nastale s spremembo vsebine, kot na primer igralnega sveta, orožja, vozil in le manjšo ali nično spremembo igralnega pogona. S tem so se pojavile tudi skupnosti za spreminjanje iger (angl. modding). Te skupnosti so sestavljali igralci

iger in majhna neodvisna podjetja, ki so ustvarjala nove igre z uporabo že obstoječih igralnih pogonov. Razmejitev med igro in igralnim pogonom je pogosto tanka. Nekateri pogoni jo jasno razlikujejo, medtem ko drugi niti ne poskušajo ločevati med njima. Igra lahko vsebuje nastavitve objektov v izvorni kodi (angl. hard coding) ali pa se poslužuje posebne kode za izrisovanje določenih tipov igralnih objektov, zaradi katerih je kodo nemogoče ponovno uporabiti v naslednjem projektu. Igralni pogon bi lahko določili kot programsko opremo, ki jo lahko razširimo in uporabimo kot temelj za različne igre brez velikih sprememb. Pogoni se razlikujejo glede na zvrst igre. Pogon, ki je zasnovan za igro boksa, se bo razlikoval od pogona za množično večigralsko spletno igro, prvoosebno strelsko igro ali realno-časovno strategijo. Vendar pa imajo pogoni tudi skupne lastnosti. Na primer vse 3D igre ne glede na zvrst potrebujejo nizko nivojski uporabniški vnos, izrisovanje 3D objektov, zvočni sistem, izrisovanje različnih pisav itd.

Odprtokodne 3D igralne pogone izdelujejo poklicni in nepoklicni izdelovalci iger. Ti pogoni so brezplačno dostopni na spletu. Izraz odprtokodni se nanaša na kodo, ki je prosto dostopna ter podpira odprt razvojni model, kar pomeni, da lahko vsakdo prispeva kodo. Če se lotimo razvoja igre in ne želimo pogona spisati sami, lahko uporabimo že enega od obstoječih pogonov. Če nimamo dovolj velikega proračuna za nakup plačljivega pogona, so odprtokodni pogoni prava rešitev. Dostopnost kode in brezplačna uporaba pogona sta tako ključna dejavnika pri izbiri igralnega pogona. Za našo 3D igro smo morali pogone primerjati, vrednotiti in izbrati ustreznega. Iskali smo stabilen igralni pogon z lastnostmi, kot so učinkovita organizacija objektov v sceni in prikaz senc ter z dodatnimi lastnostmi, ki bi omogočale hitrejšo izdelavo naše 3D igre.

Glavna tema diplomske naloge je primerjava nekaterih odprtokodnih pogonov med seboj in utemeljitev izbire določenega pogona za izdelavo predstavitvene računalniške igre. V sklopu naloge je bila izdelana igra `Dama` za operacijski sistem Windows 7.

V drugem poglavju bomo predstavili osnovne značilnosti pogonov in pre-

gledali, v katerih značilnostih se pogoni med seboj razlikujejo. Predstavili bomo naslednje igralne pogone: Panda3D, Irrlicht, OGRE3D, jME in Crystal Space.

V tretjem poglavju sledi primerjava igralnih pogonov med seboj. Predstavili bomo skupne lastnosti pogonov in lastnosti, po katerih se razlikujejo. Pogone bomo ovrednotili na podlagi tega, kateri najbolj zadovolji naše potrebe pri izdelavi igre in njegovem potencialu za uporabo pri večjih projektih.

V četrtem poglavju sledi podrobnejši opis izbranega pogona OGRE3D, kjer se bomo poglobili v teme, ki so pomembne za izdelavo naše igre.

V petem poglavju sledi predstavitev implementacije igre z izbranim pogonom. Na kratko bomo predstavili, kakšna je struktura razredov v igri, in si pogledali posebnosti pri izdelavi igre.

Šesto poglavje predstavlja ugotovitve dela. Predstavili bomo naše prispevke na temo naloge in podali nekaj izhodišč za nadaljnjo raziskovanje igralnih pogonov.

Poglavje 2

Igralni pogoni

2.1 Predstavitev igralnih pogonov

Za razvoj naše 3D igre smo hoteli uporabiti enega od že uveljavljenih igralnih pogonov. S tem bi pridobili na stabilnosti igre in skrajšali čas izdelave, saj nam ne bi bilo treba pisati svojega igralnega pogona. V nadaljevanju bomo predstavili naslednje odprtokodne 3D pogone, Panda3D, Irrlicht, jME Crystal Space in OGRE3D. Odprtokodnih pogonov je precej, vendar smo se odločili za zgoraj naštete na podlagi priljubljenosti pogona, ki jo merijo na spletni strani devmaster.net [1].

Pri opisovanju igralnih pogonov smo morali premisliti, katere komponente potrebujemo za izdelavo naše igre. Lastnosti, ki so bile prisotne pri vseh pogonih (npr. posebni učinki, senčilniki) ali pa za izdelavo naše igre niso tako pomembne (dodatni sistemi npr. fizika, umetna inteligenca, omrežje), smo izpustili, oziroma jim posvečamo manj pozornosti. Osredotočili smo se na področje upravljanja scene in na lastnosti, ki bi nam pri izdelavi prihranile čas in olajšale izdelavo 3D igre. Opis področja upravljanja scene je pomemben za izdelavo naše igre, saj imamo veliko povpraševanja po objektih v sceni v smislu, ali je določen objekt na določenem mestu ali ne itd.

Opisi pogonov nam bodo služili kot izhodišče primerjave pogonov med

[1] www.devmaster.net/engines

seboj. S podatki primerjave pogonov pa se bomo lahko odločili za pogon, s katerim bomo izdelali 3D igro dama.

2.2 Opis igralnih pogonov – Panda3D

Panda3D je igralni pogon, ki vsebuje grafični, zvočni, vhodni in izhodni sistem, sistem za detekcijo trkov ter druge lastnosti, ki so pomembne za izdelavo 3D iger [15, 16, 14]. Panda3D je odprtokodni pogon pod licenco BSD. Panda3D lahko uporabljamo za domačo ter tržno razvijanje iger brez stroškov licenciranja.

Jezik, v katerem pišemo igro, je jezik Python. Pogon sam je napisan v programskem jeziku C++ in vsebuje ovojnico, ki ponuja dostopnost pogona preko vmesnika za Python. Ta dostop omogoči razvijalcem prednosti razvoja, ki jih ponuja Python. Med njimi so hitro razvijanje programa in napredno upravljanje s pomnilnikom. Hkrati pa ohranja učinkovitost prevedenega jezika C++ v igralnem pogonu. Razvijalci v Panda3D najpogosteje pišejo programe v jeziku Python. Obstaja pa možnost neposrednega dostopa do pogona preko jezika C++.

2.2.1 Panda3D – upravljanje s sceno

Panda3D za upravljanje s sceno uporablja drevesno strukturo. Na vrhu drevesa imamo dve glavni vozlišči `Render` in `Render2D`. Razlika med njima je v mestu izrisovanja objektov, ki so pritrjeni nanju. Če je objekt pritrjen na vozlišče `Render`, se bo izrisoval kjerkoli v 3D prostoru. Če je vozlišče sin vozlišča `Render2D`, pa se bo izrisoval pred 3D sceno kot prekrivni zaslonski prikaz (angl. HUD). Vse izrisovalne lastnosti, ki jih določimo vozlišču, se prenašajo na sinove vozlišča. Za dostop do vozlišč pogon uporablja pomožni razred imenovan `NodePath`. Z njim lahko hranimo kazalec na vozlišče.

2.2.2 Panda3D – posebne lastnosti

Panda3D vsebuje lasten paketni sistem za distribucijo aplikacij. Aplikacijo je mogoče kot paket naložiti na spletno stran, s katere lahko obiskovalec požene aplikacijo.

Za spremljanje delovanja aplikacije vsebuje orodje za merjenje zmogljivosti, ki se imenuje Pstats. Z njim lahko merimo, koliko sličic se izriše na sekundo, koliko vozlišč je na sceni in koliko tekstur imamo. Orodje lahko naredi tudi graf, ki pokaže, katere operacije se najdlje izvajajo. Z njim lahko ugotovimo možna ozka grla aplikacije.

2.3 Opis igralnih pogonov – Irrlicht

Pogon Irrlicht deluje na različnih računalniških okoljih [15, 18, 14]. Uporabljamo ga lahko na okoljih Windows, Linux, OSX in Solaris. Napisan je v jeziku C++. Značilnosti so visoko nivojski programski vmesnik za izdelavo 3D aplikacij, kot so igre in znanstvene upodobitve. Ob pogonu dobimo še dokumentacijo ter različne sisteme, kot so sistem za sence, točkovne delce, animacije objektov in sistem za odkrivanje trkov. Vsi ti sistemi so dostopni preko vmesnika C++. Pri uporabi pogona pa lahko uporabljamo poleg jezika C++ tudi jezike Java, C# in Delphi. Pogon temelji na licenci zlib in je odprtokoden.

2.3.1 Irrlicht – upravljanje s sceno

Za izrisovanje scene je v pogonu Irrlicht uporabljena drevesna struktura. Imamo več tipov vozlišč, na primer za kamero, svetlobo, točkovne delce. Po potrebi lahko uporabnik dodaja nova vozlišča. Ob premiku vozlišča se potomci le-tega premikajo z njim. Pogon ima vgrajeno samodejno zaznavo, kdaj je določeno vozlišče v vidnem polju (angl. occlussion). Na voljo imamo dve tehniki za upravljanje s prostorom – BSP in Octree.

2.3.2 Irrlicht – posebne lastnosti

Aplikacije lahko razvijamo s podporo za različne jezike. To nam omogoča podpora za nize unikod (angl.unicode). Za hiter razvoj scen je k pogonu dodan tudi urejevalnik scen IrrEdit. Uporabljamo ga lahko za upravljanje grafa scene, upravljanje terena in ogled 3D modelov. Za 2D objekte scene skrbi lasten 2D sistem. Uporabljamo lahko različne gradnike, ki so pogosti pri grafičnih uporabniških vmesnikih, kot na primer gumbi, oznake in kombinirana polja.

2.4 Opis igralnih pogonov – OGRE3D

OGRE3D je odprtokoden grafični pogon, ki deluje na operacijskih sistemih Windows, Linux in Mac OS X [3, 15, 17]. OGRE3D temelji na licenci MIT. Izvorne kode aplikacij, ki jih naredimo s pogonom, ni treba objaviti. Prav tako ni potrebno objaviti, kakršnihkoli sprememb na pogonu, če smo jih naredili. Aplikacije lahko pišemo v jeziku C++. Narejen pa je tudi vmesnik za jezik Python. Poleg pogona dobimo tudi celotno dokumentacijo pogona.

2.4.1 OGRE3D – upravljanje s sceno

V OGRE3D lahko izbiramo med različnimi vtičniki za organizacijo scene. Na voljo imamo vtičnik za BSP, ki je primeren za zaprte prostore. Poleg vtičnika za BSP pa imamo še vtičnika za Octree in Terrain. V drevesno strukturo grafa scene lahko dodajamo ali iz njega odstranjujemo različna vozlišča scene. Vozlišča imajo lahko od nič do n sinov. Ob premiku očetovskega vozlišča se premaknejo vsi sinovi, relativno glede na očetovsko vozlišče. Vozlišča so lahko izločena iz izrisovanja glede na največjo prostornino, ki jo zasedejo (angl. bounding volume). V sceni lahko tudi poizvedujemo. Ugotovimo lahko, ali se na določeni lokaciji nahaja objekt.

2.4.2 OGRE3D – posebne lastnosti

V pogonu OGRE3D je za posamezne sisteme, na primer materiale, točkovne delce, prekrivke (angl. overlay), panoje, omogočena uporaba ukaznih datotek (angl. script). Z njimi lahko nastavljamo vrednosti med izvajanjem aplikacije in vidimo takojšne rezultate. Z ukaznimi datotekami tako pridobimo na času, saj ni potrebno ponovno prevajati izvorne kode.

Izvajanje v razhroščevalnem načinu vpliva na zmogljivost aplikacije, saj se izvajajo vgrajene funkcije za odkrivanje napak. Vgrajen je tudi modul za odkrivanje nepočiščenega spomina (angl. memory leak), ki nas opozori, če smo kje pozabili sprostiti spomin.

Ker je OGRE3D grafični pogon, nima določenih podsistemov. Manjkajo mu podsistem za zvok, fiziko in umetno inteligenco. Zato obstaja vmesnik `Reference Application Layer`, preko katerega lahko ločene podsisteme vključimo v pogon.

2.5 Opis igralnih pogonov – jMonkeyEngine

jMonkeyEngine (jME) je igralni pogon, narejen posebej za izdelavo modernih 3D aplikacij, saj precej uporablja senčilnike [11, 13]. Pogon jME je napisan v celoti v jeziku Java in za izrisovanje uporablja knjižnico LWJGL. Zaradi same narave jezika Java je pogon mogoče uporabljati na različnih operacijskih sistemih. jME vsebuje lastno integrirano razvojno okolje (angl. IDE), imenovano jMonkeyPlatform. Osnovano je na programskem okolju Netbeans in omogoča uporabo vtičnikov in grafično urejevanje. Pogon je izdan pod licenco BSD.

2.5.1 jMonkeyEngine – upravljanje s sceno in posebne lastnosti pogona

Na področju upravljanja scene pogon podpira drevesno strukturo za izrisovanje objektov. Drevesno strukturo lahko shranimo in naložimo preko lastnega sistema. Od prostorskih shem je na voljo shema za deljenje prostora Octree. Za hitrejše izvajanje pogon prilagaja nivo podrobnosti 3D objektov (angl. LOD). Z vidika posebnih lastnosti pogona je podprto delovanje aplikacije kot je Java applet. To za našo igro ni bistvenega pomena, a je lahko priročno, če hočemo v prihodnosti predstaviti svojo igro na spletu kot applet.

2.6 Opis igralnih pogonov – Crystal Space

Crystal Space je ogrodje za razvijanje 3D aplikacij v jeziku C++[10, 12]. Izdelava aplikacij pa ni vezana izključno na jezik C++, saj ogrodje ponuja vmesnik za skriptni jezik Python. Ponavadi se Crystal Space uporablja kot igralni pogon, vendar sama struktura Crystal Space omogoča uporabo za kakršnokoli 3D aplikacijo. Pogon lahko uporabljamo na operacijskih sistemih Windows in Mac OS X. Za to poskrbi z uporabo knjižnic OpenGL in SDL. Pogon lahko uporabljamo brezplačno pod licenco GNU Lesser General Public License.

2.6.1 Crystal Space – upravljanje s sceno in posebne lastnosti pogona

Crystal Space za upravljanje scene uporablja drevesno strukturo. 3D svet je shranjen v zapisu XML. S tem omogoča spreminjanje sveta izven izvorne kode. Za določanje vidnosti objektov je vgrajen algoritem portal. Na voljo imamo tudi upravljalnik `Sequence`, s katerim lahko objektom nastavimo sprožilce (angl. trigger), ki se sprožijo ob določenem dogodku. Za izdelavo naše igre je to dobrodošlo, lahko bi namreč nastavljali dogodke, kot na primer skrivanje žetonov, predvajanje zvoka ob dotiku z igralno površino itd.

Poglavje 3

Primerjava igralnih pogonov

Predstavljeni igralni pogoni ponujajo veliko funkcij, ki jih lahko uporabimo pri izdelavi 3D aplikacij. Lastnosti opisanih pogonov so zbrane v preglednici A.1 (glej dodatek A). V nadaljevanju si bomo pogledali, katere lastnosti so skupne vsem pogonom in pri katerih prihaja do razlik. Opisana področja smo izbrali glede na to, kaj bomo pri izdelavi naše igre potrebovali in kaj bi nam lahko koristilo pri nadaljnjih projektih. Področji pomembni za izdelavo naše igre sta področje posebnih učinkov ter področje upravljanja scene in materialov. Za ti dve področji smo se odločili, ker želimo izdelati čim lepšo igro, kar dosežemo s posebnimi učinki, in ker želimo imeti čim boljši nadzor nad upravljanjem objektov v sceni. Za uporabo pri nadaljnjih projektih bomo opisali naslednja področja: na katerih računalniških okoljih pogon deluje, katere programske vmesnike pogon podpira, ali podpirajo skriptne jezike ali ne in ali imajo podporo za vtičnike. Računalniška okolja, na katerih pogoni delujejo, so pomembna, ker želimo bodoče igre prodati na čim več različnih trgih. Ostala področja pa nam omogočajo hitrejše ali lažje razvijanje in nadgrajevanje projektov. Na koncu smo se glede na rezultate primerjave odločili za pogon, s katerim bomo izdelali predstavitveno 3D igro.

3.1 Skupne lastnosti igralnih pogonov

Ugotovili smo, da vsi pogoni delujejo na različnih računalniških okoljih. Delujejo v okoljih Windows, Linux in Mac OS X . Zato je razumljiva prisotnost podpore grafičnemu programskemu vmesniku OpenGL pri vseh pogonih. Na področju posebnih učinkov vsi podpirajo:

- panoje,
- točkovne delce,
- simulacijo megle,
- senčenje s šablonsko tehniko (angl. stencil shadow technique),
- odsev okolja.

Vsem pogonom je skupno, da lahko spreminjamo ali dodajamo materiale. Lahko jih namreč dodajamo ali popravljamo med izvajanjem aplikacije. Pri upravljanju scene vsi pogoni uporabljajo hierarhično drevesno strukturo, vendar pa se razlikujejo v podrobnostih implementacije.

3.2 Izbira pogona in utemeljitev izbire

Za izbiro ustreznega pogona smo ovrednotili posamezne lastnosti pogonov. Na podlagi teh ovrednotenih lastnosti oziroma sodil smo izbrali najustreznejši pogon. Pri tem smo uporabili orodje za večparametrsko odločanje DEXi. Lastnosti pogona smo razdelili v kategorije (slika 3.1) in na podlagi ovrednotenih lastnosti znotraj posamezne kategorije določili skupno oceno za posamezno. Primer odločitvenih pravil za skupno oceno pogona na podlagi ocen dveh glavnih podkategorij lahko vidimo na sliki 3.2. Na podlagi odločitvenih pravil lahko vidimo približno kolikšno utež ima posamezno sodilo v končni oceni (slika 3.1). Rezultati vrednotenja so podani v preglednici 3.3.

Sodila in povprečne uteži

Sodilo	Lokalne	Globalne
Igralni pogoni		
Nujno za razvoj naše igre	57	57
Posebni učinki - sklop 1	25	14
Teksturna tehnika senčenja	20	3
Šablonska tehnika senčenja	21	3
Simulacija megle	20	3
Simulacija vode	20	3
Simulacija neba	20	3
Posebni učinki - sklop 2	25	14
Panoji	20	3
Točkovni delci	20	3
Razpršena svetloba na objektivu	21	3
Zabrisano gibanje	20	3
Odsev okolja	20	3
Upravljanje s sceno	25	14
Povpraševanje po sceni	50	7
Prilagodljivost sistema	50	7
Materiali	25	14
Lasten jezik za materiale	50	7
Podpora LOD	50	7
Uporabna področja	43	43
Podprti operacijski sistemi	20	9
Windows	15	1
Mac OS X	15	1
IPhone	23	2
Linux	23	2
Android	23	2
Programski vmesniki	20	9
Vmesnik C++	28	2
Grafični API	22	2
Pogon za fiziko	16	1
Pogon za zvok	16	1
Sistem za umetno inteligenco	16	1
Ocena področja vtičnikov	20	9
Podpora za vtičnike	100	9
Ocena glede števila uporabnikov	20	9
Število registriranih uporabnikov	100	9
Ocena skriptnih jezikov	20	9
Skriptni jeziki	100	9

Slika 3.1: Kategorije sodil in njihove povprečne uteži, izračunane na podlagi odločitvenih pravil v programu DEXi.

	Nujno za razvoj naše igre	Uporabna področja	Igralni pogoni
	57%	43%	
1	manj primerno	<=primerno	manj primerno
2	<=primerno	manj primerno	manj primerno
3	<=primerno	najbolj primerno	primerno
4	primerno	>=primerno	primerno
5	najbolj primerno	manj primerno	primerno
6	najbolj primerno	>=primerno	bolj primerno

Slika 3.2: Odločitvena pravila za izračun skupne ocene posameznega pogona iz ocen dveh glavnih podkategorij (funkcionalnosti, nujne za izdelavo naše igre, in funkcionalnosti, ki nam lahko koristijo pri naslednjih projektih.)

Pogon OGRE3D je bil ocenjen kot bolj primeren. Ostali pogoni so dobili slabšo oceno, zato smo se odločili uporabiti pogon OGRE3D. Pomanjkljivost pogona OGRE3D je, da nima dodatnih podsistemov, ki jih pogosto srečujemo pri izdelavi 3D iger: podsistem za zvok, fiziko, omrežje itd. Vendar pa OGRE3D nadomesti to pomanjkljivost s tem, da vsebuje `Referrence Application Layer`. Z njim nam ponudi prosto izbiro kateregakoli podsistema in omogoča povezavo pogona OGRE3D z izbranim podsistemom. Pomembno vlogo pri izbiri pogona imajo tudi posebni učinki. Z njimi bo naša 3D igra izgledala lepše in bolj tehnično dovršeno. Z uporabo senc pa bomo še bolj poudarili 3D izgled igre. OGRE3D podpira vse posebne učinke naštete v preglednici A.1. S tem je zadovoljil naše zahteve po posebnih učinkih za našo igro. Precej pomembna je tudi licenca pogona, saj bi našo igro morebiti radi v prihodnosti prodali, kar pa nekatere licence ne omogočajo. V veliki meri pa je na odločitev vplival še en podatek – število registriranih uporabnikov na forumu OGRE3D. Pri ostalih pogonih je bilo število uporabnikov manjše, kar kaže na večjo aktivno uporabniško skupino pri OGRE3D.

Rezultati vrednotenja

Sodilo	Panda3D	Irrlicht	OGRE3D	jME	Crystal Space
Igralni pogoni (končna ocena)	**manj primerno**	**manj primerno**	***bolj primerno***	**manj primerno**	**manj primerno**
Nujno za razvoj naše igre	**manj primerno**	**manj primerno**	***najbolj primerno***	primerno	primerno
Posebni učinki - sklop 1	**manj primerno**	primerno	***najbolj primerno***	primerno	primerno
Teksturna tehnika senčenja	**ne**	**ne**	***da***	ni podatka	**ne**
Šablonska tehnika senčenja	***da***	***da***	***da***	***da***	***da***
Simulacija megle	***da***	***da***	***da***	***da***	***da***
Simulacija vode	**ne**	***da***	***da***	***da***	***da***
Simulacija neba	**ne**	***da***	***da***	***da***	***da***
Posebni učinki - sklop 2	primerno	***da***	***najbolj primerno***	***najbolj primerno***	***najbolj primerno***
Panoji	***da***	***da***	***da***	***da***	***da***
Točkovni delci	***da***	***da***	***da***	***da***	***da***
Razpršena svetloba na objektivu	**ne**	**ne**	***da***	***da***	***da***
Zabrisano gibanje	**ne**	**ne**	***da***	***da***	***da***
Odsev okolja	***da***	***da***	***da***	***da***	***da***
Upravljanje s sceno	primerno	**ni primerno**	***najbolj primerno***	primerno	**ni primerno**
Povpraševanje po sceni	***da***	**ne**	***da***	***da***	**ne**
Prilagodljivost sistema	ni podatka	**ne**	***da***	ni podatka	**ne**
Materiali	**manj primerno**	**manj primerno**	primerno	**manj primerno**	**manj primerno**
Lasten jezik za materiale	**ne**	**ne**	ni podatka	**ne**	**ne**
Podpora LOD	**ne**	**ne**	***da***	**ne**	**ne**
Uporabna področja	primerno	primerno	***najbolj primerno***	**manj primerno**	**manj primerno**
Podprti operacijski sistemi	primerno	primerno	primerno	***najbolj primerno***	primerno
Windows	***da***	***da***	***da***	***da***	***da***
Mac OS X	***da***	***da***	***da***	***da***	***da***
IPhone	**ne**	**ne**	***da***	***da***	**ne**
Linux	***da***	***da***	***da***	***da***	***da***
Android	**ne**	**ne**	**ne**	***da***	**ne**
Programski vmesniki	***najbolj primerno***	***najbolj primerno***	***najbolj primerno***	primerno	primerno
Vmesnik C++	***da***	***da***	***da***	***da***	***da***
Grafični API	***OpenGL in DirectX 9***	***OpenGL in DirectX 9***	***OpenGL in DirectX 9***	OpenGL	OpenGL
Pogon za fiziko	***da***	**ne**	**ne**	***da***	***da***
Pogon za zvok	***da***	**ne**	**ne**	***da***	***da***
Sistem za umetno inteligenco	***da***	**ne**	**ne**	**ne**	**ne**
Ocena področja vtičnikov	primerno	**ni primerno**	primerno	**ni primerno**	**ni primerno**
Podpora za vtičnike	***da***	**ne**	***da***	**ne**	**ne**
Ocena glede števila uporabnikov	**manj primerno**	primerno	***najbolj primerno***	**manj primerno**	**manj primerno**
Število registriranih uporabnikov	6000	13000	***26000***	ni podatka	4000
Ocena skriptnih jezikov	primerno	**ni primerno**	***najbolj primerno***	**ni primerno**	**ni primerno**
Skriptni jeziki	Python	**ne**	***Python, Lua***	**ne**	**ne**

Slika 3.3: Rezultati vrednotenja pogonov s pomočjo programa za večparametrsko odločanje DEXi.

Poglavje 4

Podrobnejši opis pogona OGRE3D

Ker bomo za izdelavo igre uporabili pogon OGRE3D, si ga bomo v nadaljevanju bolj podrobno pogledali. Opis pogona bomo razdelili na določene teme. Omenili bomo tiste, ki jih bomo srečali oziroma potrebovali skozi proces izdelave igre.

Začeli bomo z upravljalnimi razredi (angl. manager). V programu obstaja po en primerek posameznega razreda, ki mu pravimo edinec (angl. Singleton). Ponujajo nam nadzor nad določenim področjem, na primer viri, pisavo, animacijo itd. Upravljalni razredi so pomemben del arhitekture sistema v OGRE3D. Pri arhitekturi pogona bomo predstavili, katere vzorce načrtovanja objektov lahko najdemo v pogonu in opisali strukturo grafa scene. Nato bomo obravnavali kamero in pojasnili pojem svetovna geometrija. Pri prostorih bomo razložili, katere prostorske sheme lahko uporabimo v OGRE3D. Nadaljevali bomo z materiali, kaj lahko z njimi delamo in kako delujejo v pogonu. Sledi upravljanje z viri, predstavili bomo, kaj so viri v pogonu, kateri so podprti in kakšen je njihov življenjski cikel.

4.1 Upravljalni razredi

Upravljalni razredi v OGRE3D so razredi, ki upravljajo dostop do različnih funkcionalnosti sistema ali zagotavljajo obstoj drugih podobnih objektov[3]. Vsak upravljalni razred v OGRE3D obstaja kot enkraten, edinstven objekt, do katerega zagotovimo splošno dostopnost iz celotnega programa (angl. Singleton pattern). V pogonu OGRE3D najdemo veliko upravljalnih razredov. Pomembnejši med njimi so:

- LogManager: Pošilja sporočila na izhodni tok.
- ControllerManager: Upravlja razrede, ki menjavajo stanja glede na uporabniški vnos, ga pošilja naprej drugim razredom. Najbolj pogosta je raba pri razredih za animacijo tekstur ali materialov.
- DynLibManager: Skrbi za dinamično povezane knjižnice (DLL na okolju Windows ter deljeni objekti na okolju Linux).
- PlatformManager: Priskrbi nam dostop do podrobnosti računalniškega okolja in strojne opreme.
- CompositorManager: Omogoča dostop in upravljanje ogrodja, ki omogoča 2D kompozicijo in procesiranje opravil v zaslonskem prostoru.
- ArchiveManager: Upravljalnemu razredu za vire omogoči dostop do pravilnega razreda za upravljanje s posameznimi datotečnimi zabojniki kot npr. ZIP.
- ParticleSystemManager: Upravlja s sistemi za točkovne delce in objekti, ki spreminjajo točkovne delce.
- MaterialManager: Skrbi za nalaganje materialov v aplikaciji. Omogoča ponovno uporabo že naloženih materialov.
- SkeletonManager: Skrbi za nalaganje okostja 3D modelov v aplikaciji.

- MeshManager: Upravlja z nalaganjem mrež poligonov, ki sestavljajo 3D model.
- GpuProgramManager: Nalaga nizko nivojske programe (npr. v zbirnem jeziku) kot tudi visoko nivojske programe za GPE, ki so že bili prevedeni v zbirni jezik.
- FontManager: Nadzoruje in nalaga pisave. Te pisave uporabljamo za izrisovanje besedila na zaslon preko s upravljalnega razreda `Overlay` .
- ResourceGroupManager: Služi kot glavna točka za nalaganje virov in nadzor nad njihovim življenjskim ciklom.
- OverlayManager: Skrbi za nalaganje in izdelavo 2D objektov. Uporabljamo ga za HUD, uporabniški vmesnik in druge 2D vsebine, ki se izrisujejo na vrhu scene.
- TextureManager: Nadzira življenjski cikel in dostopnost vseh tekstur, ki jih uporabljamo v aplikaciji.

4.2 Arhitektura sistema v OGRE3D

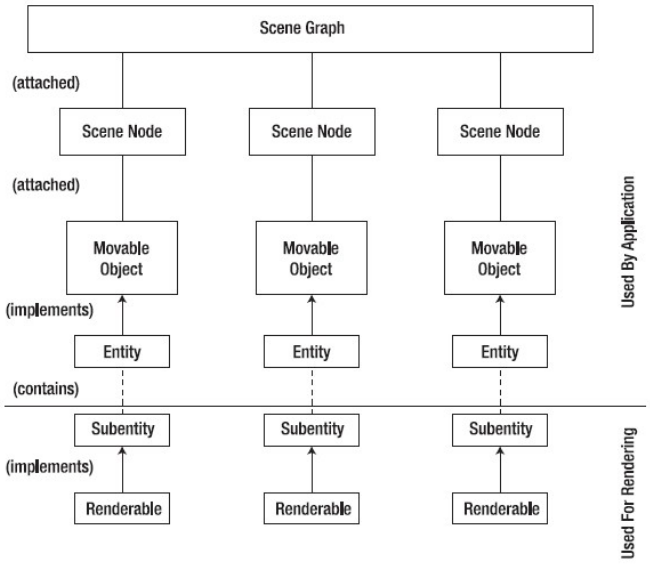
Sledi pregled vzorcev za načrtovanje objektov, ki jih lahko najdemo v pogonu OGRE3D. V nadaljevanju pa bomo razložili strukturo grafa scene. Pogledali bomo, kateri razredi sestavljajo graf, kakšne lastnosti imajo in razložili, kako graf sploh deluje.

OGRE3D uporablja mnogo načrtovalskih vzorcev, ki povečujejo uporabnost in prožnost pogona. Na primer OGRE3D obvešča odjemalsko aplikacijo o vsem, kar počne preko vzorca opazovalec (angl. Observer), s pomočjo katerega se odjemalčeva aplikacija naroči na sporočila dogodkov ali obvestila o spremembi stanja. Tako je aplikacija obveščena na primer, kdaj se je določeno okno začelo izrisovati, ali se je dokončno izrisalo itd. [4, 6, 7]. Vzorec edinec (angl. Singleton) se uporablja za ustvarjanje natanko enega primerka razreda. Vzorec obiskovalec (angl. Visitor) omogoča dodajanje operacij nad

objektom brez spreminjanja kode razreda oziroma objekta. Vzorec fasada (angl. Façade) je namenjen lažjemu dostopu do pogosto uporabljenih operacij, vgrajenih v drugih podsistemih, s pomočjo enega samega vmesnika. Nazadnje je tu še vzorec tovarna (angl. Factory), ki se uporablja za ustvarjanje primerkov razredov.

Ločitev grafa scene od vsebine scene je bila verjetno eno najbolj pomembnih odločitev v projektu OGRE3D [3]. Običajni grafi scen (takšni, ki so običajno uporabljeni v tržnih in odprtokodnih 3D pogonih) navadno povežejo vsebino scene z grafom scene v dedovalni hierarhiji, ki sili v izdelavo podrazredov vozlišč scene. To se dolgoročno izkaže kot zelo slaba odločitev, saj onemogoči spreminjaje algoritmov grafa scene brez spreminjanja ogromne količine kode. Poleg tega vsa vozlišča izhajajo iz osnovnega vozliščnega tipa, kar prinese podedovano neprožnost in nerazširljivost (vsaj s stališča vzdrževanja). Ta funkcionalnost sili postopoma v manjša spreminjanja osnovnega vozlišča, kar je slaba praksa objektno usmerjenega koncepta. V OGRE3D deluje graf scene na ravni vmesnika – do funkcionalnosti se dostopa preko metod grafa, ki ne razlikuje med vgrajenimi algoritmi. Nadalje je za vmesnik grafa scene pomembna le struktura grafa. Vozlišča ne vsebujejo dostopa vsebine. Namesto tega OGRE3D vse to pretvori v tako imenovan objekt `Renderable`, iz katerega izhaja vsa geometrija v sceni. Lastnosti za izrisovanje (znane tudi kot materiali) teh objektov so shranjene v objektih `Entity`, ki pa lahko hranijo enega ali več objektov `SubEntity`. Objekte `SubEntity` izrisujemo na zaslon. Na sliki 4.1 je opisana zveza med strukturo grafa scene in vsebino scene. Opazimo lahko, da so vozlišča scene tudi prikazana na grafu scene; graf scene neposredno ne upravlja s stanji vozlišč.

Vsa geometrija in lastnosti izrisa so na voljo grafu scene preko `Movable Object`. Namesto izdelave premikajočega objekta kot podrazreda vozlišča scene (angl. scene node) je le-ta pripet na vozlišče scene. To pomeni, da lahko vozlišče scene obstaja tudi brez kakršnega koli objekta, ki bi se moral izrisati. Tudi razširjanje in spreminjanje grafa scene ne vpliva na obliko in vgrajenost vmesnika vsebinskih objektov. Grafični vmesnik scene se lahko



Slika 4.1: Graf scene v OGRE3D.[3]

popolnoma spremeni, pa to ne bo vplivalo na razrede, ki vsebujejo vsebino scene. OGRE3D pa ponuja tudi drugo možnost. Graf scene ne potrebuje podatka o kakršnih koli spremembah vsebinskih razredov, vse dokler razredi vsebujejo enostaven vmesnik, ki je združljiv z grafom scene. Dovoljeno je dodajanje vsebine vozliščem scene. Ta zmožnost je uporabna, ko hočemo na vozlišču prenašati zvočne podatke, za kar ne potrebujemo nobenih podrazredov. Potrebujemo le enostaven vmesnik na po meri narejenem objektu, da ga prilepimo na vozlišče scene.

4.3 Kamera ter upravljanje s sceno

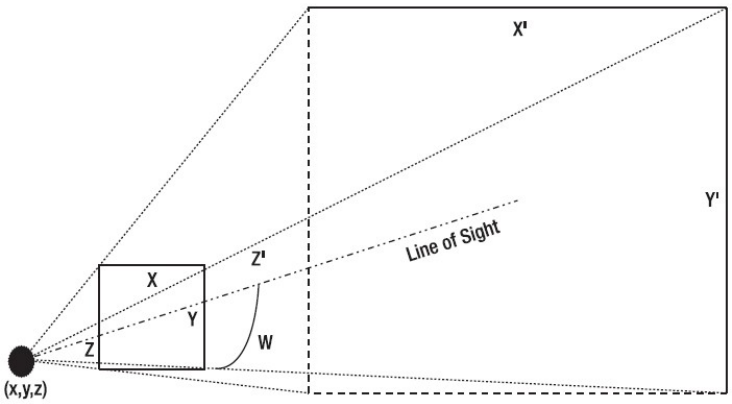
Ta dva razreda sta odgovorna za upodobitev naše scene. Preden jih lahko uporabljamo, ju moramo ustvariti. Ko OGRE3D naloži programske komponente, med katerimi so lahko različne oblike upravljalnika scene, se vsaka oblika upravljalnika prijavi kot določen tip upravljalnika [3].

- ST_GENERIC: Minimalna oblika upravljalnika scene. Ni najboljši za

določene strukture scene ali vsebino. Najbolj uporaben za nezapletene scene.

- ST_INTERIOR: Primeren za upodabljanje notranjosti in z objekti gosto napolnjenih scen.
- ST_EXTERIOR_CLOSE: Upravljalnik scene, najbolj primeren za upodabljanje zunanjih scen z majhno ali srednjo vidljivostjo.
- ST_EXTERIOR_REAL_FAR: Upravljalnik scene, ki je najbolj primeren za velike scene.

Razred kamera deluje natanko tako, kot deluje kamera v resničnem svetu. Zajame sliko naše scene na vsako sličico (angl. frame) z določene točke (ima torej smer in položaj). Običajno kamera ni upodobitveni objekt. Če imamo v vidnem polju pogled na drugo kamero, se le-ta ne upodobi. Kamere kot objekte lahko v OGRE3D obesimo na vozlišča scene (s tem lahko dosežemo učinek animacije) ali pa jih pustimo prosto v prostoru in jih premikamo ročno vsako sličico.



Slika 4.2: Vidno polje kamere.[3]

Na sliki 4.2 vidimo vidni stožec (piramida z odrezanim vrhom) kamere na položaju s koordinatami x, y, z. X in Y sta velikost oddaljene ploskve

(angl. clip plane), preko katere ne vidimo več. X' in Y' velikost ploskve, ki določa, na kateri najmanjši razdalji vidimo okolje. OGRE3D moramo posredovati naslednje podatke za pravilno izdelavo kamere: razdaljo med okvirjema, razmerje kamere (definirano kot X/Y) in navpični kot med linijo vidnega polja ter spodnjo ploskvo vidnega polja. Razred kamera samodejno izračuna vodoravni kot in velikost bližnje in oddaljene omejevalne ploskve.

4.4 Geometrija sveta

Geometrija sveta predstavlja naš navidezen svet, ki ga sestavljajo površje, rastlinje na površju ter notranjost statičnih objektov z zidovi, okni in vrati vred. Z drugimi besedami, svetovna geometrija je svet, v katerem naši objekti obstajajo.

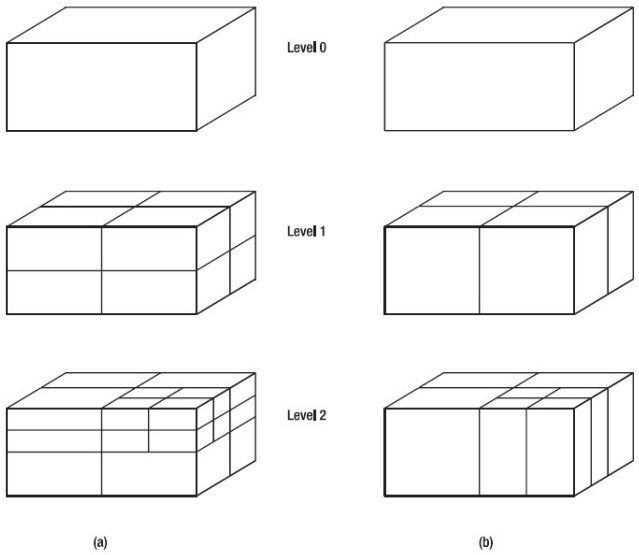
Geometrija sveta je običajno oblikovana z orodji za 3D oblikovanje, kot na primer Maya, 3D Studio Max ali Blender, ali pa v posebnih orodjih za izdelavo višinskih zemljevidov področja, kot na primer Terragen.

Upravljalni razred scene lahko naloži geometrijo sveta na dva različna načina: neposredno preko datoteke na disku ali skozi preko podatkovnega toka [3]. Druga možnost nam omogoča stiskanje poligonske mreže sveta ter nastavitvenih podatkov v po meri narejeno obliko zapisa, ki jo lahko pošljemo neposredno upravljalnemu razredu scene. Večina aplikacij uporablja prvo možnost.

4.5 Pogoste sheme delitve prostora

Quadtree je dvorazsežna delitvena shema. Vendar lahko to shemo kljub temu uporabljamo za bolj ravne 3D scene, predvsem pri površju. Shema Octree pa je bolj primerna za manjša zaprta in zunanja področja, kjer je precej pomembna navpična komponenta (tla nad in pod kamero). Na sliki 4.3 je prikaz shem Octree in Quadtree za tri korake prostorske razdelitve.

Binarno deljenje prostora (angl. Binary space partitioning; BSP) (slika

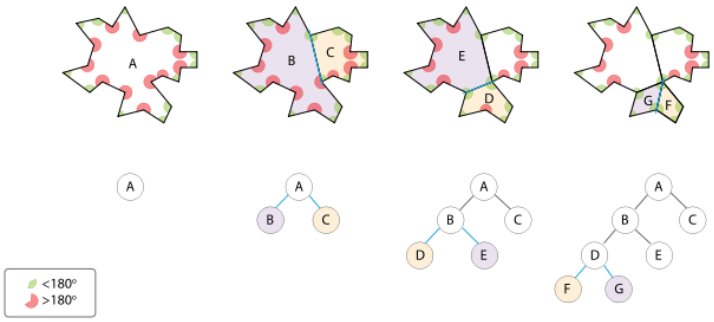


Slika 4.3: Prostorska vizualizacija prostorske delitvene sheme (a) Octree in (b) Quadtree.[3].

4.4) je starejša delitvena shema scene. Glavna prednost te sheme v modernih aplikacijah je hitra zaznava trka, ne pa organizacija poligonov. OGRE3D podpira upravljalnik scene BSP (zapis Quake 3 Arena), vendar ta zapis ni primeren za novejšo strojno opremo, ki procesira večje količine poligonov [3, 17]. Zato sta bila razvoj in podpora te sheme v OGRE3D opuščena. Kljub temu podpora zanjo še vedno ostaja za tiste, ki jo hočejo naložiti in uporabljati.

4.6 Materiali in odboji svetlobe v OGRE3D

Materiali določajo, kako se bo svetloba odbijala od objekta. V resnici se svetloba lahko odbija od objekta na druge objekte. V OGRE3D ta lastnost ni vgrajena, saj bi bilo zaradi zahtevnosti potrebnih algoritmov delovanje aplikacije prepočasno [3, 8, 1]. OGRE3D podpira štiri različne materiale glede na odboj/oddajanje svetlobe: neusmerjen, razpršen, oddajajoč in odbojen.



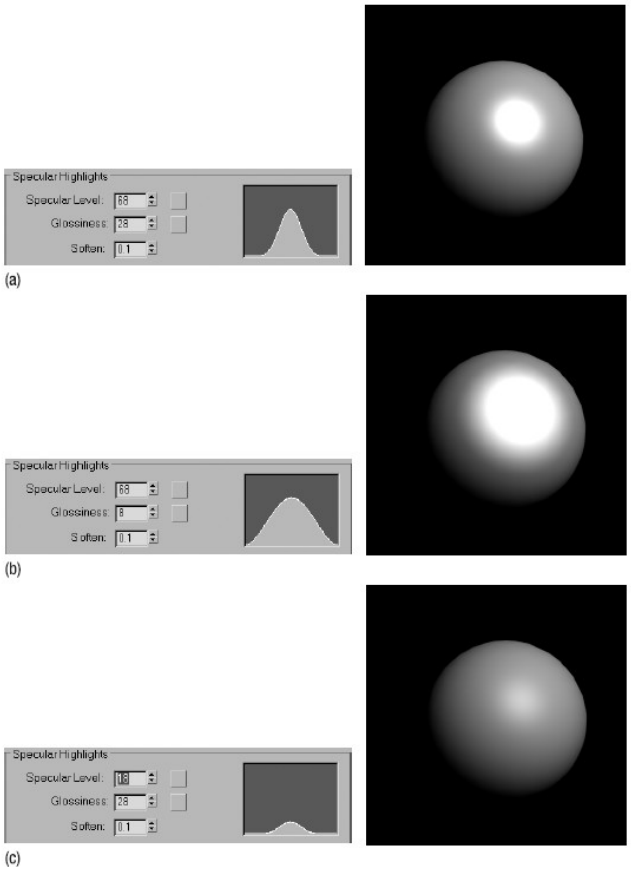
Slika 4.4: Binarno deljenje prostora.[19]

- Neusmerjena svetloba običajno služi kot približek celotni osvetlitvi scene. Neusmerjena barvna lastnost materiala na objektu določi splošno barvo objekta.
- Razpršena barva pomeni, da je svetloba odbita enakomerno v vse strani.
- Oddajajoča barva je barva svetlobe, ki jo objekt seva. To lahko vodi tudi k nenaravni podobi objekta.
- Odbojna barva opiše odboj svetlobnih žarkov od objekta zaradi neposredne osvetlitve objekta.

Na sliki 4.5 je prikazan urejevalnik za materiale, s katerim nastavimo odbojno lastnost objekta. Na levi strani imamo vrednosti, ki jih lahko nastavljamo. Odbojni nivo določa, koliko svetlobe se bo odbilo, bleščavost pomeni, kako zelo se bo objekt bleščal, ko bo obsijan s svetlobo.

Slika 4.6 prikazuje pregled nad zgradbo materiala. Material lahko vsebuje eno ali več tehnik osvetlitve. Vsaka izmed teh tehnik pa lahko vsebuje enega ali več obhodov. Samo ena tehnika je lahko aktivna v določenem času. Če je tehnika 0 aktivna, druge tehnike niso uporabljene v tem upodobitvenem obhodu.

Vsak upodobitveni objekt se sklicuje samo na en material. OGRE3D izbere tehniko iz tega materiala na različne načine. Ko enkrat izbere tehniko,



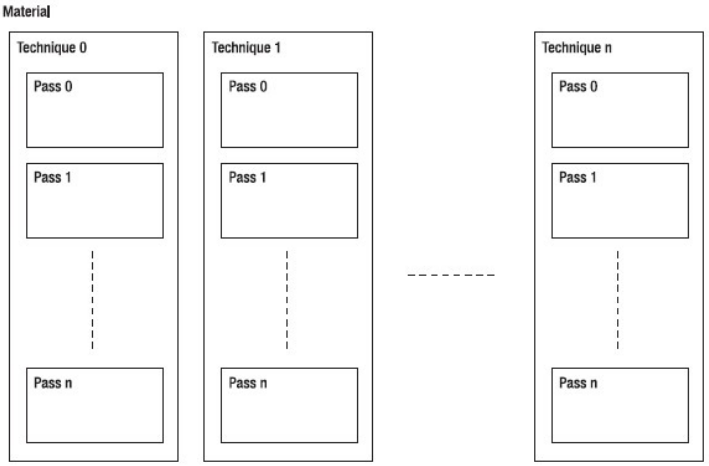
Slika 4.5: Urejevalnik za materiale v Autodesk 3D Studio Max.[3]

se vsak obhod izvrši natanko tako, kot je zapisano. Na primer, če OGRE3D izbere tehniko s tremi obhodi za upodobitveni objekt, bo narisal ta objekt trikrat za vsako sličico.

4.7 Upravljanje z viri v OGRE3D

Vse, kar OGRE3D uporabi za izris naše scene, obravnava kot vir [3, 17]. Pisave, poligonske mreže modelov, okostja modelov, materiali – vse so viri v OGRE3D. OGRE3D podpira naslednje vrste virov(slika 4.7):

- Material: Skriptne datoteke `.material` vsebujejo opise materialov (teh-



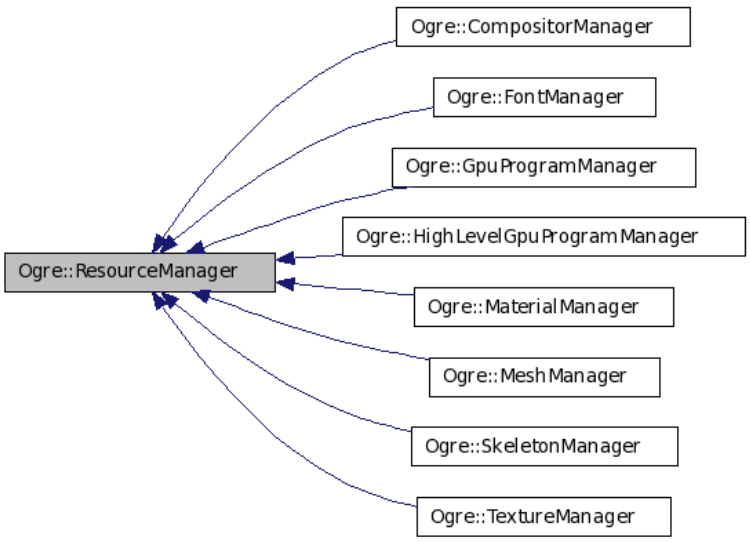
Slika 4.6: Razmerje med materiali, tehnikami in obhodi v OGRE.[3]

nike, obhode itn.).

- Poligonski mrežni model: Binarne datoteke `.mesh` vsebujejo vozlišča, geometrijo in nekatere podatke o animaciji.
- Okostja modelov: Binarne datoteke `.skeleton`, ki vsebujejo razvrstitev kosti in animacijske podatke, ki jih uporabljamo pri animaciji z okostjem v OGRE3D.
- Pisava: Tekstovne datoteke `.fontdef`, ki vsebujejo opredelitve pisav.
- Tekstura: podatke za teksture dobimo iz 2D slik. Te slikovne datoteke imajo svoje običajne končnice (.png, .jpg, .jpeg, .tga).

OGRE3D lahko upravlja z viri na dveh ravneh: kot s posameznimi viri ali kot s poimenovanimi skupinami. Poimenovane skupine imajo prednost pred posameznimi viri, da jih lahko naložimo in pospravimo po končani uporabi kot celoto. Namenjena je večji učinkovitosti pri organizaciji virov. Iskanje vira v skupini je časovne zahtevnosti O(1) [3].

Kazalec na vir v OGRE3D kaže pravzaprav arhiv datotek. Arhivi so lahko v kakršnikoli obliki zapisa, napisati moramo le razred, ki bo obdelal

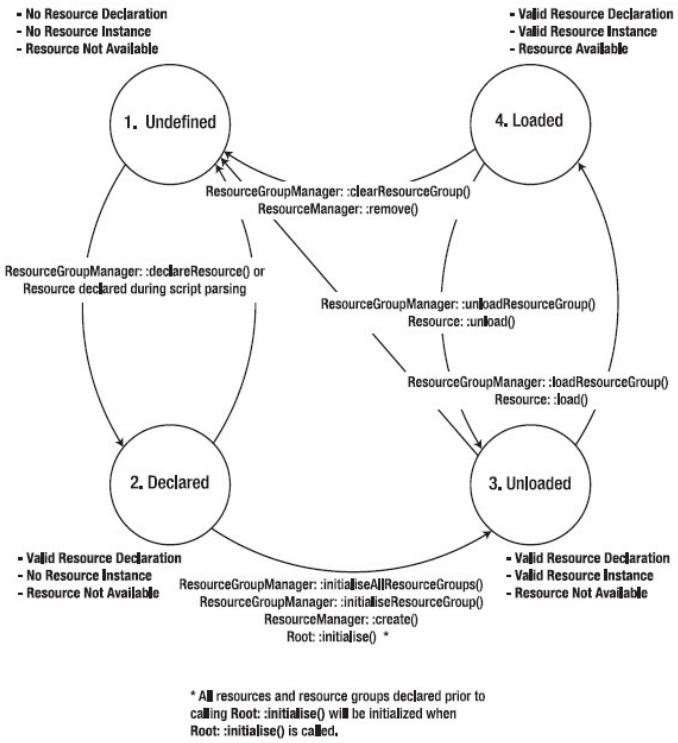


Slika 4.7: Pregled podprtih virov v OGRE.[17]

našo obliko zapisa. Iz arhivov lahko samo beremo, pisanje ni podprto. To še ne pomeni, da ne moremo sami dodati funkcije za pisanje podatkov v arhive. Na sliki 4.8 je prikazan življenjski cikel virov OGRE3D. Dogodki na sliki so klici API OGRE3D . Običajno nam ni treba skrbeti za stanja virov.

Spodnji izrazi so uporabljeni v OGRE3D, da razlikujemo, v kakšnem stanju se nahaja vir v določenem trenutku.

- Nepoznan: OGRE ne ve za ta vir. Datoteka je sicer shranjena v skupini virov, vendar OGRE ne ve, kaj naj naredi s to datoteko.
- Prijavljen: vir je bil označen za uporabo.
- Narejen: OGRE je izdelal prazen primerek vira in ga dodelil pravilnemu upravljalniku virov.
- Naložen: primerek je bil v celoti naložen. Vsi podatki vira so naloženi v računalniški spomin. Viri ne vzamejo veliko pomnilnika, vse dokler ni naložena njihova vsebina. Ni nobene škode, če dodamo vse vire upravljalniku virov, saj lahko potem po potrebi naložimo ali odstranimo vsebino vira.



Slika 4.8: Življenjski cikel virov.[3].

Poglavje 5

Opis 3D igre narejene v OGRE3D

Pogon OGRE3D smo izbrali, ker smo želeli izdelati 3D igro. Ustvarili smo 3D računalniško različico namizne igre dama. V tej igri lahko uporabimo veliko funkcij, ki jih ponuja OGRE. Med drugim smo uporabili točkovne delce (eksplozije), sence in animacijo z vozlišči.

5.1 Struktura in navodila igre

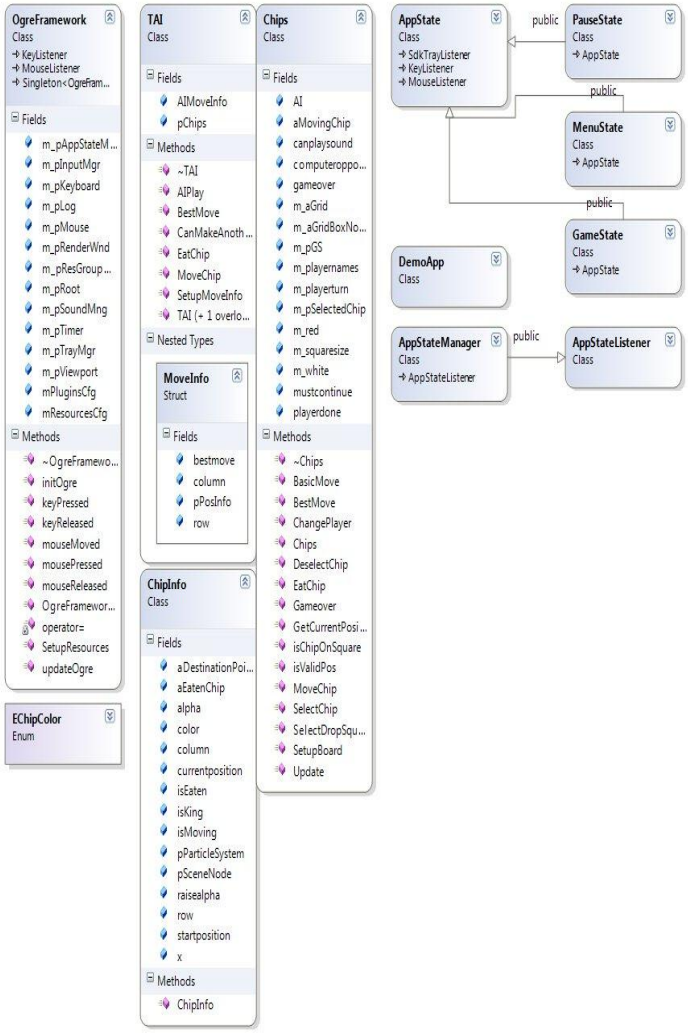
Igra se igra po pravilih namizne igre dama. Igra en igralec proti računalniškemu nasprotniku. Vsak igralec ima 12 žetonov, eden bele, drugi črne barve. Žetoni so navadno valjaste oblike. Začne igralec, ki ima bele žetone. Igralca vlečeta poteze izmenično. Začetni položaj je na črnih poljih v prvih treh vrstah. Žetoni se pomikajo samo diagonalno, vedno po črnih poljih. Žeton, ki še ni povišan v damo, se premika samo naprej. Poteza brez jemanja je poteza naprej na sosednje prosto polje. Če je na sosednjem polju nasprotnikov žeton, diagonalno za njim pa prosto polje, vzamemo nasprotnikov žeton s skokom čezenj, žeton pa odstranimo z deske. V tem primeru je jemanje obvezno. Cilj igre je vzeti (pojesti) čimveč nasprotnikovih žetonov ali pa blokirati nasprotnika. Žeton, ki pride do nasprotnikove zadnje vrste, postane dama.

Običajno na žeton položimo še enega, s tem se loči od navadnih žetonov. Dama se lahko premika naprej in nazaj, pri tem lahko preskakuje poljubno število praznih polj. Igro dobi igralec, ki vzame vse nasprotnikove žetone, ali pa jih blokira tako, da nasprotnik ne more napraviti nobene poteze.

Struktura aplikacije je sledeča (5.1). V razredu `OgreFramework` imamo navedene osnovne upravljalne razrede. Imamo upravljalne razrede za hranjenje stanj, za vnos podatkov, za vnos podatkov preko miške, za vire, zvok, števec idr. Pri metodah najdemo osnovne metode za spremljanje vnosa podatkov, pripravo virov, vzpostavitev igre in glavne metode za posodabljanje igre. Razred `Chips` upravlja z žetoni in skrbi, da se igra odvija po pravilih. V tem razredu lahko najdemo podatke o tem, kateri žeton se trenutno premika, ali je igra končana, kakšna je mreža in kje v mreži so postavljeni žetoni, kateri žeton je trenutno izbran, kdo je na potezi itd. Najdemo lahko naslednje metode: `BasicMove` skrbi za premik žetona, `BestMove` najde najboljšo možno potezo za žeton, `isValidPos` preveri ali je trenutni premik veljaven, `isChipOnSquare` preveri ali je žeton na polju, `SelectChip` izbere žeton, `SelectDropSquare` izbere polje, na katerega spustimo žeton, `SetupBoard` vzpostavi začetno stanje na šahovnici in metoda `Update` posodablja žetone. Razred skrbi tudi za animacijo žetonov, zvok, eksplozije itd. Razred `ChipInfo` je preprost razred, ki vsebuje podatke o posameznem žetonu. Pridobimo lahko naslednje podatke: kam se bo žeton premaknil, ali bo kak žeton na svoji poti pojedel drug žeton, barvo žetona, vrstico in stolpec, v katerem se trenutno nahaja in, ali je bil povišan v damo. V razredu `TAI` nimamo posebnih podatkov, imamo pa zato več metod, ki nam pomagajo pri odločitvi, kateri žeton naj premaknemo. Poklicati moramo le metodo `AIPlay` in metoda bo poskrbela za najboljši premik.

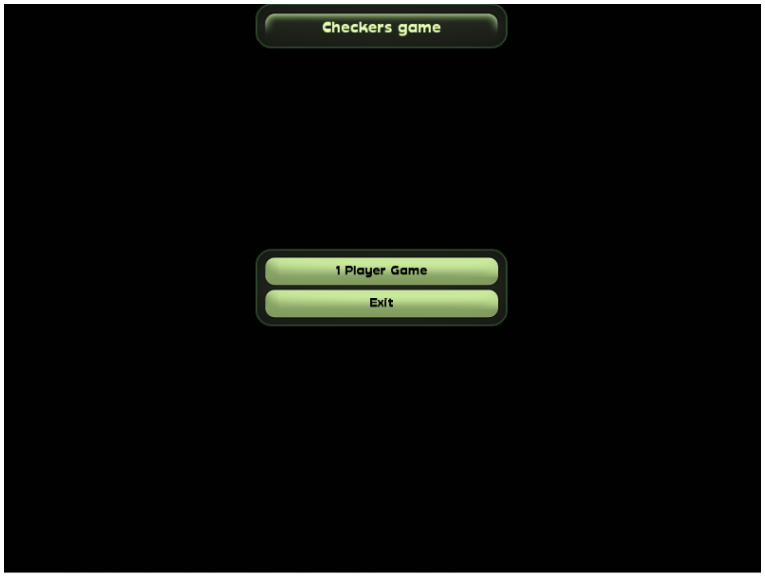
5.2 Uporabniški vmesnik

V igri smo uporabili uporabniški vmesnik OGRE3D, ki je enostaven za uporabo in več kot primeren za našo igro. Uporabili smo SDKTrays. Naredili



Slika 5.1: Razredni diagram za igro Checkers2012.

smo lasten upravljalni razred s stanji v uporabniškem vmesniku. Na voljo so tri stanja: `PauseState`, `MenuState`, `GameState`. Pri zagonu igre najprej pridemo v `MenuState` (slika 5.2).



Slika 5.2: Izgled uporabniškega vmesnika v stanju `MenuState`.

Po kliku na gumb za igro se naloži `GameState` (slika 5.3).

Razred `SdkTrayListener` obdela dogodke, ki jih lahko sprožijo gradniki. Dogodki, ki jih lahko procesira so naslednji:

- `buttonHit`: vrne kazalec na gumb, ki smo ga kliknili.
- `itemSelected`: vrne kazalec na element, ki smo izbrali iz menija.
- `labelHit`: vrne kazalec na oznako, ki smo jo kliknili.

5.3 Materiali

V igri smo uporabili naslednjo skripto za material.

```
material
{
```

Slika 5.3: Izgled igre v stanju `GameState`.

```
    technique
    {
        pass
        {
            ambient 0 0 0 1
            diffuse 1 1 1 1
            specular 0 0 0 1
            emissive 0 0 1

            texture_unit
            {
                texture chessboard_clr.tga
                tex_coord_set 0
                colour_op modulate
                scale 1 1
                scroll 0 0
                rotate 0
            }
        }
    }
}
```

S parametri `ambient`, `diffuse`, `specular` in `emissive` smo določili, kako naj določena vrsta svetlobe deluje na objekt iz določenega materiala. V

`texture_unit` smo določili, katero teksturo bomo uporabili. Ukaz `texture chessboard_clr.tga` pomeni, da smo si izbrali teksturo chessboard_clr.tga za naš material. Ukazi `Rotate`, `scale` in `scroll` pa vplivajo na postavitev te teksture. Vplivamo lahko na rotacijo teksture (ukaz `rotate`), velikost (ukaz `scale`) in zamik mesta teksture (ukaz `scroll`).

5.4 Kamera

Glavna naloga kamere je, da določi vidno polje [5, 9]. V igri smo kameri dodali še nekaj lastnosti, da je bolj uporabna: s statično kamero bi težko prikazali 3D izgled igre in s tem realističnost. Naredili smo pet metod, ki določajo delovanje kamere. Te metode so `enter`, `getInput`, `MoveCamera`, `Update` in `ChangePlayer`.

```
void Chips::ChangePlayer()
{
    if( m_playerturn == ccWhite )
    {
        m_playerturn = ccBlack;
        m_pGS->m_pCamera->setPosition(0.f, 20.2f, 18.2f);
        m_pGS->m_pCamera->lookAt(0.f, 0.f, 0.f);
    }
    else
    {
        m_playerturn = ccWhite;
        m_pGS->m_pCamera->setPosition(0.f, 20.2f, -18.2f);
        m_pGS->m_pCamera->lookAt(0.f, 0.f, 0.f);
    }
}
```

V metodi `ChangePlayer` postavimo kamero na določeno mesto in jo usmerimo proti določeni točki. V našem primeru je to koordinatno izhodišče.

```
void GameState::enter()
{
    ...
    m_pCamera = m_pSceneMgr->createCamera("GameCamera");
    m_pCamera->setPosition(Vector3(0.f, 20.268f, -18.272f));
    m_pCamera->lookAt(Vector3(0.f, 0.f, 0.f));
    //
```

```
    m_pCamera->setNearClipDistance(1);
    //
    OgreFramework *p=OgreFramework::getSingletonPtr();
    p->m_pViewport>setCamera(m_pCamera);
}
```

V funkciji `enter` ustvarimo kamero s klicem `createCamera`. S funkcijo `setPosition` jo postavimo v 3D svet in jo obrnemo proti točki, ki jo hočemo gledati s klicem `lookAt`. Zelo pomembno je dodeliti kamero razredu `Viewport`.Ta razred preslika našo 3D sceno, kot 2D sliko v obliki pravokotnika, na mesto kamere. To naredimo s klicem `setCamera`.

```
void GameState::getInput()
{
    OgreFramework *pof=OgreFramework::getSingletonPtr();
    if(m_bSettingsMode == false)
    {
        if(m_pKeyboard->isKeyDown(OIS::KC_A))
            m_TranslateVector.x = -m_MoveScale;

        if(pof->m_pKeyboard->isKeyDown(OIS::KC_D))
            m_TranslateVector.x = m_MoveScale;

        if(pof->m_pKeyboard->isKeyDown(OIS::KC_W))
            m_TranslateVector.z = -m_MoveScale;

        if(pof->m_pKeyboard->isKeyDown(OIS::KC_S))
            m_TranslateVector.z = m_MoveScale;
   //
    if(m_bRMouseDown)
    {
        m_pCamera->yaw(Degree(evt.state.X.rel * -0.1f));
        m_pCamera->pitch(Degree(evt.state.Y.rel * -0.1f));
    }
}
```

Podatke za premike in obračanje kamere moramo v igri prebrati. Zato smo ustvarili metodo `getInput()`, ki nastavlja vrednosti, kam naj se kamera premakne v naslednji sličici. Kamero lahko s pomočjo metod `yaw` in `pitch` tudi obračamo.

```
void GameState::moveCamera()
{
```

```
    OgreFramework *pof=OgreFramework::getSingletonPtr();
    if(pof->m_pKeyboard->isKeyDown(OIS::KC_LSHIFT))
    m_pCamera->moveRelative(m_TranslateVector);
    m_pCamera->moveRelative(m_TranslateVector / 10);
}

void GameState::update(double timeSinceLastFrame)
{
    ...
    m_TranslateVector = Vector3::ZERO;
    //
    getInput();
    moveCamera();
}
```

V metodi `update` ponovno nastavimo vektor za premik kamere in kličemo metodi `getInput`, ki nastavi translacijski vektor, in `moveCamera` za premik kamere.

5.5 Sence in animacija žetonov

Za prikaz senc v igri smo morali iz naših 3D objektov najprej dobiti podatke, kje imajo objekti robove. Podatke smo iz orodja za 3D oblikovanje pretvorili v lasten zapis za 3D objekte pogona OGRE3D. Pri tem smo uporabili orodje Ogre Meshes Exporter za izvedbo pretvorbe. S temi podatki smo morali v igri samo še poklicati metodo `createScene` in sence so se izrisovale na vseh 3D objektih v aplikaciji.

```
void GameState::createScene()
{
          Ogre::ColorValue color=Ogre::ColourValue(0.0f,0.0f,0.0f));
          m_pSceneMgr->setAmbientLight( color );

    pSceneMgr->setShadowTechnique(Ogre::SHADOWTYPE_STENCIL_ADDITIVE);
    Ogre::Light *l=m_pSceneMgr->createLight("Light");
    l->setType(Ogre::Light::LT_POINT);
    l->setPosition(0.f, 30.268f, -18.272f);
}
```

V zgornji metodi `createScene` smo morali za delovanje senc najprej ustvariti neusmerjeno svetlobo. Izbrati smo morali tudi tehniko senčenja.

Odločili smo se za tehniko senčenja "stencil additive". Ostala nam je le še postavitev neusmerjene svetlobe v sceno in sence so bile pripravljene za uporabo.

```
void Chips::Update(float dt)
{
    if(aMovingChip.size())
    {
        ChipInfo *info=aMovingChip.front();
        //
        float x = info->x;
        Ogre::Vector3 currentposition = info->currentposition;
        Ogre::Vector3 startposition = info->startposition;
        Ogre::Vector3 destination_position;
        destination_position=info->aDestinationPoint.front();
        //
        float distance = sqrt(startposition.squaredDistance(destination_position));
        x += dt/1000;
        Ogre::Vector3 direction = destination_position -       startposition;
        //
        currentposition.x = startposition.x + direction.x * (x);
        currentposition.y = sin( x * 3 );
        currentposition.z = startposition.z + direction.z * (x);
        //
        info->currentposition = currentposition;
        info->x = x;
        info->pSceneNode->setPosition(currentposition);
        ...
    }
}
```

Zgornja metoda določa animacijo žetona. V začetku metode najprej preverimo, ali sploh moramo kaj animirati. Če je pogoj izpolnjen, vzamemo prvo informacijo žetona v polju `aMovingChip` . Sledi vzpostavitev spremenljivk. Trenutni položaj žetona določa spremenljivka `currentposition`, s katerega polja se je žeton premikal, določa spremenljivka `startposition` in kam mora žeton premakniti, določa spremenljivka `destination_position`. Za izvedbo premika moramo izračunati razdaljo in smer gibanja žetona. Nato spremenljivki `currentposition.x` in `currentposition.z`, povečamo za zmnožek med smerjo gibanja in spremenljivke `x`. Spremenljivka X je sprememba časa `dt` deljeno s tisoč. Pri spremenljivki `currentposition.y` pa smo spreminjali vrednost s sinusno funkcijo, saj smo želeli prikazati valovno gibanje.

Poglavje 6

Zaključek

V diplomski nalogi smo predstavili in opisali pet bolj priljubljenih odprtokodnih igralnih pogonov. Priljubljenost smo določili glede na število registriranih uporabnikov na forumu vsakega igralnega pogona ter glede na število pregledov[1], ki so jih bili deležni. Po opisu pogonov smo zbrali skupaj njihove lastnosti in jih primerjali. Našteli smo skupne lastnosti in lastnosti, po katerih se pogoni med seboj razlikujejo. Na podlagi vrednotenja pogonov smo izbrali pogon OGRE3D, ki je najbolj ustrezal našim zahtevam, in z njim naredili računalniško različico namizne igre dama. Pri izbranem pogonu smo si bolj podrobno pogledali tiste lastnosti, ki smo jih potrebovali za izdelavo igre. Na koncu pa smo podrobneje predstavili samo izvedbo igre.

Primerjavo pogonov bi lahko še poglobili, oziroma razširili obseg pogonov, ki jih primerjamo. Vendar pa za izbiro pogona pri naši igri le-to ni bilo potrebno, predvsem pa smo se osredotočili na odprtokodne pogone. Dodatno bi lahko preverili, kako so dobro so posamezne funkcionalnosti implementirane in kako dobra je njihova časovna in prostorska zahtevnost. Bolj podrobna primerjava bi lahko vsebovala tudi študijo o težavnosti uporabe posameznega pogona.

Glavni namen pogonov je, da nam prihranijo velik delež programiranja, poleg tega pa je njihova koda relativno stabilna. Za neodvisne razvijalce

[1] www.devmaster.net/engines

iger so predvsem zanimivi zastonjski in odprtokodni pogoni, pri čemer so odprtokodni zanimivi predvsem zato, ker bi jih lahko po potrebi popravili. Tako smo v kratkem času lahko razvili 3D igro, za katero bi brez obstoječega pogona zagotovo porabili precej več časa.

Dodatek A

Primerjava lastnosti pogonov

		Pogoni				
Kategorija	**Lastnost**	Panda3D	Irrlicht	OGRE3D	jME	Crystal Space
podprti operacijski sistemi	Windows	da	da	da	da	da
	OS X	da	da	da	da	da
	iOS	-	-	da	da	-
	Linux	da	da	da	da	da
	Android	-	-	-	da	-
programski vmesniki	vmesnik C++	da	da	da	da	da
	grafični API	OpenGL, DX9	OpenGL, DX9	OpenGL, DX9	OpenGL	OpenGL
	pogon za fiziko	lasten, osnoven	-	-	da	da
	pogon za zvok	da	-	-	da	da
	pogon za omrežje	da	-	-	da	-
	sistem za umetno inteligenco	da	-	-	-	-
posebni učinki	panoji	da	da	da	da	da
	točkovni delci	da	da	da	da	da
	razpršena svetloba na objektivu[1]	-	-	da	da	da
	zabrisano gibanje[2]	-	-	da	da	da
	simulacija megle	da	da	da	da	da
	simulacija neba[3]	-	da	da	da	da
	teksturna tehnika senčenja	-	-	da	ni podatka	-
	šablonska tehnika senčenja	da	da	da	da	da
	simulacija vode	-	da	da	da	da
	odsev okolja[4]	da	da	da	da	da
materiali	lasten jezik za izdelavo materialov	-	-	da	-	-
	več izrisovalnih obhodov na objekt	ni podatka	ni podatka	da	da	da
	podpora LOD	-	-	da	ni podatka	-

Se nadaljuje na naslednji strani.

		Pogoni				
Kategorija	**Lastnost**	Panda3D	Irrlicht	OGRE3D	jME	Crystal Space
	materiale lahko dodajamo in popravljamo med izvajanjem aplikacije	da	da	da	ni podatka	ni podatka
upravljanje s sceno	prilagodljivost sistema	ni podatka	delno[5]	da[6]	ni podatka	-
	povpraševanje v sceni	da	-	da	da	-
	podprte prostorske sheme	ni podatka	BSP in Octree	Octree, BSP, tehnike Terrain	octree	portali
ostalo	podprti jeziki za senčilnike	Cg, HLSL	HLSL, GLSL	Cg, HLSL, GLSL	GLSL	Cg
	podpora za vtičnike	da	-	da	-	-
	skriptni jeziki	Python	-	Python, Lua, lastni[7]	-	-
	število registriranih uporabnikov	okoli 6.000	okoli 13.000	okoli 26.000	ni podatka	okoli 4.000
	datum začetka projekta	2001	2003	1999	2003	1997
	licenca	BSD	zlib	MIT	BSD	LGPL

Preglednica A.1: Primerjava lastnosti pogonov.

[1] angl. lens flare
[2] angl. motion blur
[3] angl. skybox
[4] angl. environment mapping
[5] lahko menjamo med zunanjo in notranjo vrsto scene
[6] angl. ni vezan na vrsto scene
[7] lastni skriptni jeziki za točkovne delce, materiale itd.

Seznam slik

Literatura

[1] D. H. Eberly, *3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics*, Morgan Kaufmann, ZDA, 2000

[2] J. Gregory, *Game Engine Architecture*, A. K. Peters, Wellesley, ZDA, 2009

[3] G. Junker, *Pro OGRE 3D Programming*, Apress, New York, ZDA, 2006

[4] S. B. Lippman, *C++ Primer Fourth Edition*, Addison-Wesley, London,Velika Britanija, 2005

[5] D. McMahon, *Linear Algebra Demystified*, McGraw-Hill Professional, ZDA, 2005

[6] R. Penton, *Data Structures for Game Programmers*, Muska & Lipman, ZDA, 2002

[7] D. Sanchez-Crespo, *Core Techniques and Algorithms in Game Programming*, New Riders Games, ZDA, 2003

[8] P. Walsh, *Advanced 3D Game Programming with Direct X 9.0*, Wordware Publishing Inc., ZDA, 2003

[9] J. Vince, *Mathematics for Computer Graphics*, Springer, 2005

[10] (2011) Crystal Space wikipedia,
Dostopno na: http://en.wikipedia.org/wiki/Crystal_Space.
(Datum zadnjega obiska: 20.9.2011)

[11] (2011) jMonkeyEngine wikipedia,
Dostopno na: http://en.wikipedia.org/wiki/JMonkey_Engine.
(Datum zadnjega obiska: 20.9.2011)

[12] (2011) Crystal Space,
Dostopno na: http://www.crystalspace3d.org.
(Datum zadnjega obiska: 20.9.2011)

[13] (2011) jMonkeyEngine,
Dostopno na: http://jmonkeyengine.com.
(Datum zadnjega obiska: 20.9.2011)

[14] "Dokument o odprtokodnih in nizkocenovnih pogonih",
Dostopno na:
http://ludocraft.oulu.fi/elias/dokumentit/open_source_game_engines.pdf.
(Datum zadnjega obiska: 20.9.2011)

[15] (2011) Source for game development,
Dostopno na: http://www.devmaster.net.
(Datum zadnjega obiska: 20.9.2011)

[16] (2011) Panda3D,
Dostopno na: http://www.panda3d.org.
(Datum zadnjega obiska: 20.9.2011)

[17] (2011) OGRE3D, Dostopno na: http://www.ogre3d.org.
(Datum zadnjega obiska: 20.9.2011)

[18] (2011) Irrlicht,
Dostopno na: htttp://www.irrlicht.sourceforge.net/features.html.
(Datum zadnjega obiska: 20.9.2011)

[19] (2011) Binary space partitioning,
Dostopno na:
http://en.wikipedia.org/wiki/Binary_space_partitioning.
(Datum zadnjega obiska: 20.9.2011)