Irrlicht engine - podpora grafických technologií a management scény

Irrlicht engine- graphic technology and scene management

Filip Sadloň

Bakalářská práce
2008

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav aplikované informatiky
akademický rok: 2007/2008

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Filip SADLOŇ**
Studijní program:	**B 3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Téma práce:	**Irllicht engine – podpora grafických technologií a management scény**

Zásady pro vypracování:

1. **Vytvořte literární rešerši na zadané téma.**
2. **Podrobně se seznamte s enginem Irrlicht. Zaměřte se zejména na podporu grafických technologií a management scény.**
3. **Uvedenou oblast v předchozím bodu zadání podrobně popište. Zkrácenou verzi popisu uveďte v tištěné verzi práce, v elektronické příloze se bude nacházet popis celý.**
4. **Ve spolupráci se studentem řešící téma Irllicht engine – grafické rozhraní a práce se souboryproveďte návrh komplexnějšího programu využívající tento engine.**
5. **Vytvořte 3D modely pro tuto aplikaci v programu Blender.**

Rozsah práce:

Rozsah příloh:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

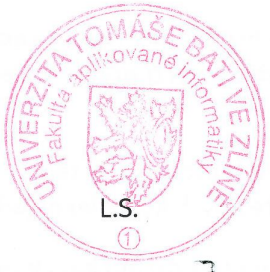
1. **Irrlicht engine – a free open source 3d engine [online]. 2007 [cit. 2008-01-22]. Dostupný z WWW: http://irrlicht.sourceforge.net/.**
2. **FINNEY, Kenneth C. 3D game programming all in one. [s.l.] : Premier press, 2002. s. ISBN 1-59200-136-X.**
3. **TEIXEIRA DE SOUSA, Bruno Miguel. Game programming all in one. [s.l.] : Premier press, 2002. 992 s. ISBN 1-931841-23-3.**
4. **MORRISON, Michael. Naučte se programovat počítačové hry za 24 hodin. 1. vyd. Brno : Computer Press, 2004. 421 s. ISBN 80-251-0371-4.**
5. **LIBERTY, Jesse. Naučte se C++ za 21 dní. Brno : Computer Press, 2007. 796 s. ISBN -80-251-1583-1.**

Vedoucí bakalářské práce: **Ing. Pavel Pokorný, Ph.D.**
Ústav aplikované informatiky

Datum zadání bakalářské práce: **20. února 2008**

Termín odevzdání bakalářské práce: **5. května 2008**

Ve Zlíně dne 20. února 2008



L.S.

prof. Ing. Vladimír Vašek, CSc.
děkan

doc. Ing. Ivan Zelinka, Ph.D.
ředitel ústavu

ABSTRAKT

Bakalárska práca je zameraná na tvorbu 3D interaktívnych aplikácií pomocou grafického enginu Irrlicht. Popisuje engine, jeho históriu a rozhranie pre programovanie aplikácií. Ďalej je popísané rozhranie pre management scény a rozhranie pre správu grafických technologií. Praktická časť popisuje použitie dostupných nástrojov spolu s externým fyzikálnym enginom pri tvorbe 3D interaktívnej aplikácie so zameraním na tvorbu a management scény a prácu s grafickými technológiami.

Cieľom práce je vytvoriť jednoduchú príručku o tvorbe aplikácií pomocou enginu Irrlicht a zároveň vytvoriť aplikáciu, ktorá ukazuje použitie funkcií enginu a jeho možnosti.

Kľúčové slová: Irrlicht, 3D engine, 3D počítačová grafika, 3D počítačová hra, 3D interaktívna aplikácia, Blender, Newton Game dynamics

ABSTRACT

The bachelor thesis deals with creation of 3D interactive applications using Irrlicht graphics engine. It describes the engine, its history and application programming interface. Then scene management and video technology management are described. Practical part of the thesis describes application of accessible tools and external physics engine in creation of a 3D interactive application focused on scene creation, management and graphic technology operations.

The aim of the thesis is to create a simple manual about creation of 3D applications using Irrlicht engine and also to create an application that demonstrates engine functions and its resources.

Keywords: Irrlicht, 3D engine, 3D computer graphics, 3D computer game, 3D interactive application, Blender, Newton Game dynamics

Úvodom by som chcel poďakovať môjmu vedúcemu, pánovi Ing. Pavlovi Pokornému, Ph.D za pomoc a rady pri tvorbe bakalárskej práce.

Prohlašuji, že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků, je-li to uvolněno na základě licenční smlouvy, budu uveden jako spoluautor.

Ve Zlíně dne 15.5.2008

…….

Podpis diplomanta

OBSAH

ÚVOD

Bakalárska práca je zameraná na tvorbu 3D interaktívnych aplikácií pomocou Irrlicht Engine. Irrlicht je open source multiplatformný grafický engine.

Každý program, ktorý zobrazuje priestorovú grafiku využíva nejaký 3D engine, je to tá časť programu, ktorá sa stará o grafickú realizáciu aplikácie, o vykreslovanie objektov, prípadne o ich správu. Pri tvobe aplikácie je možné vytvoriť vlastný 3D engine, čo vyžaduje komplexnú znalosť počítačovej grafiky. Druhá možnosť je použiť niektorý z dostupných grafických enginov, keďže väčšinu úloh na ktoré je engine potrebný využíva takmer každá 3D aplikácia. Hlavnou úlohou grafického enginu je vytvoriť vhodnú abstrakciu nad grafickým API (Aplication Programing Interface), teda nad OpenGL alebo nad Direct3D. Začlenením grafického enginu do aplikácie je možné pracovať s objektami, ktoré majú byť zobrazované na vysokej úrovni, programátor sa nemusí starať o tvorbu a vykreslovanie objektov, môže s nimi rovno pracovať a tvoriť napríklad hernú logiku. Keďže 3D grafika sa v súčasnosti využíva hlavne pri tvorbe hier je dostupných mnoho grafických a herných enginov. Rozdiel medzi grafickým a herným enginom je v tom, že herný engine poskytuje okrem API pre prácu s grafikou aj funkcie pre prácu s fyzikou objektov, zvukom, sieťovým pripojením a funkcie pre tvorbu umelej inteligencie. Irrlicht je čisto grafický engine, neobsahuje žiadne funkcie okrem grafických [1].

Medzi hlavné výhody enginu Irrlicht patrí jeho veľká rýchlosť, stabilita , to, že engine je ľahko rozšíriteľný a hlavne to, že je platformne nezávislý. Irrlicht môže byť použitý pre tvorbu aplikácií pre Windows, Linux aj pre MacOS. Výhodou je tiež nezávislosť na grafickom API. Irrlicht nepracuje len s Direct3D, ktorý je použiteľný len na sýstémoch s operačným systémom Windows, ale aj s OpenGL a dvoma softwareovými renderovačmi. Irrlicht Engine je vhodný na vytváranie rôznych druhov aplikácií ako napr. simulácií, hier typu first person shooter, third person shooter, real timeových strategických hier, hier typu RPG, renderovanie scén, ale aj na tvorbu 2D hier [1].

I. TEORETICKÁ ČASŤ

1 IRRLICHT ENGINE

1.1 Popis enginu

Irrlicht je multiplatformný 3D grafický engine, nie je to kompletný herný engine, pretože neobsahuje funkcie pre prácu s umelou inteligenciou, fyziku, ani funkcie na prácu so zvukom. Irrlicht je software s volne šíriteľným zdrojovým kódom pod licenciou Zlib, ktorá umožňuje komerčné využívanie bez nutnoti zverejnenia úprav v zdrojových kódoch. Kód aplikácie v ňom vytvorenej pracuje na operačných systémoch Windows, Linux, MacOS, Sun Solaris bez nutnosti úprav. V súčasnej dobe vývojári pracujú aj na portoch tohoto enginu pre Xbox, PlayStation a SymbianOS [1].

Irrlicht je písaný v C++, ale dostupné sú aj verzie pre Javu a .NET (C#,VB,Delphi.NET). Engine Irrlicht podporuje viacero renderovacích API: Direct3D8.1, Direct3D9, OpenGL 2.0, Irrlicht engine softwareový renderer, Apfelbaum softwareový renderer. Okrem 3D grafiky umožňuje Irrlicht použiť aj 2D ovládacie prvky užívateľského rozhrania ako sú tlačítka (button), posuvník (scroll bar), pole výberu (combo box), atď [12].

Scéna v Irrlichte je reprezentovaná grafom, objekty v nej sú reprezentované uzlami. Uzol v scéne môže byť napríklad kamera, v engine sú obsiahnuté okrem obyčajnej kamery aj kamera typu Maya a kamera typu FPS. Uzly ďalej môžu predstavovať svetlo, model, billboard (2D objekt s textúrou, ktorý je vždy natočený smerom ku kamere). Na optimalizovanie rýchlosti zobrazenia scény je vhodný uzol typu OctTree, ktorý sa využíva na zobrazenie komplexných scén[2].

Engine podporuje rôzne grafické technológie ako napríklad real-timeové tiene, kostrovú animáciu, časticové efekty a zabudovanú knižnicu materiálov s podporov vertex a pixel shaderov. V prípade, že zabudované materiály nepostačujú požiadavkam na aplikáciu je možne výtvoriť vlastné pomocou Pixel a Vertex Shaderov 1.1 až 3.0, HLSL, GLSL[12].

Irrlicht má tiež zabudovanú detekciu kolízií, ktorá je určená hlavne pre hry typu FPS (first person shooter) a umožňuje aby hráč nemohol prechádzať cez steny atď. Engine ďalej podporuje priame čítanie veľkého množstva formátov textúr a modelov, pretože rôzne

externé imortovače a exportovače sa už nachádzajú priamo v základnej knižnici. Engine je taktiež schopný modely v ktoromkoľvek podporovanom formáte prekonvertovať na iný podporovaný formát. Obsahuje tiež vlastný formát scény .irr, ktorý umožňuje uložiť celú scénu vrátane textúr a materiálov objektov do jediného súboru.

Fakt, že sa Irrlicht zameriava hlavne na grafickú stránku vývoja aplikácií, sa dá riešiť tak, že chýbajúce funkcie sa doplnia externými aplikáciami navrhnutými pre Irrlicht ako napr. irrKlang(2D a 3D zvukový engine), irrEdit(3D editor a lightmap generátor), irrXML(xml parser). Ďalej je možné do Irrlichtu integrovať nejaký fyzikálny engine, ktorý umožní použiť v aplikácii objekty s vlastnou dynamikou pohybu [12].

1.2 História Irrlichtu v bodoch

Uvedené dátumy popisujú kedy boli funkcie Irrlicht Enginu prvý krát uvdené v oficiálnej verzii, všetky tieto funkcie sa naďalej rozvíjali a v súčasnej verzii (1.4) sú už takmer všetky 100% hotové. Uvedených je prvých 10 verzií v ktorých sa vytvorilo jadro enginu, zmeny v ďaľších verziách neboli tak výrazné.

- rok 2001 - Irrlicht engine začal vyvýjať rakúsky programátor Nikolaus Gebhardt.
- verzia 0.1 - marec 2003 - prvá verzia vydaná pod licenciou Zlib pre platformu Windows32. Prvá verzia enginu podporovala:
    - šablóny typu string, polí, lineárnych zoznamov, vektory, matice
    - platformne nezávislé čítanie súborov a čítanie priamo z komprimovaných .zip súborov
    - GUI prostredie so systémom udalostí a množstvom elementov (obrázok, list box, statický text, scroll bar, tlačítko, check box atď.)
    - vykreslovanie 2D a 3D elementov pomocou Direct3D8
    - priame čítanie textúr vo formáte JPG, TGA, BMP, PSD
    - vykreslovanie priesvitných útvarov
    - dynamické svetlo
    - management všetkých uzlov v scéne, vrátane kamery
    - implementáciu typu OctTree
    - priame čítanie modelov vo formáte OBJ, MD2, BSP

- verzia 0.2 - máj 2003
    - vykreslovanie 2D a 3D útvarov pomocou OpenGL
    - dynamické tiene
    - billboardy
    - sky box
    - priame čítanie modelov vo formáte 3ds
- verzia 0.3 - júl 2003
    - priame čítanie modelov vo formáte MS3D
    - jednoduchú detekciu kolízií (bounding box)
    - výber uzlov v scéne pomocou kurzoru alebo 3d lúča
    - Linux
- verzia 0.4 - september 2003
    - nové GUI elementy - message boxy
    - multitexturing
    - časticový systém
    - uzol pre tvorbu realistickej vodnej hladiny
    - reálnu detekciu kolízií (na základe trianglov)
- verzia 0.5 - september 2003
    - nové GUI elementy – edit box, tab control, context menu
    - vykreslovanie 2D a 3D útvarov pomocou Direct3D9
    - hmlu
    - priame čítanie modelov vo formáte .X
- verzia 0.6 - marec 2004
    - platformne nezávislý zápis do súborov a získanie zoznamu súborov
    - integrovaný XML Parser a XML Writer
- verzia 0.7 - september 2004
    - Pixel a Vertex shadery pre Direct3D8, Direct3D9 a OpenGL
    - Irrlicht.NET s podporou C#,VB a Delphi.NET
- verzia 0.8 - december 2004
    - HLSL materiály a bump/normal mapy

- verzia 0.9 - marec 2005
    - priame čítanie modelov vo formáte OCT, CMS, LMTS, MY3D
    - nový renderer terénu
- verzia 0.10 - máj 2005
    - priame čítanie textúr vo formáte PNG
    - renderovanie do textúry
    - priame čítanie modelov vo formáte COLLADA, DMF
- súčasná verzia 1.4 - november 2007 [1]

1.3 Popis rozhrania aplikácie

Táto kapitola sa zaoberá popisom základných funkcií enginu Irrlicht pre tvorbu interaktívnych aplikácií so zameraním na popis funkcií pre management scény a podporu grafických technológií.

Irrlicht engine nepoužíva štandardné datové typy z dôvodu prenositeľnosti enginu, preto definuje množstvo vlastných typov. Z rovnakého dôvodu nepoužíva Irrlicht ani STL, takže definuje tiež vlastné pole, mapu a lineárny zoznam. Namiesto štandardného typu std::string definuje triedu irr::core::string, ktorá ponúka tiež nejaké užitočné funkcie navyše.

Engine používa namespace-y, hlavný namespace je irr, ktorý obsahuje ďalšie namespace-y:

- **irr**- celý engine je obsiahnutý v tomto namespace
    - základné triedy a rozhrania enginu, napr. event receiver, Irrlicht device, časovač atď.
    - definície vlastných datových typov Irrlichtu, napr. f32 je 32-bitový float, s32 je 32-bitový integer, c8 je 8-bitový char
    - definície typov enum, napr. typy udalostí, kódy kláves, tlačítok myši atď.
- **irr::core**- tu sa nachádzajú základné triedy ako napr. matice, vektory, priamky, lineárny zoznam, mapa
- **irr::gui**- obsahuje triedy vhodné na tvorbu grafického užívateľského rozhrania

- **irr::io**- obsahuje triedy pre prácu so vstupmi a výstupmi ako napr. čítanie súborov, ZIP archívov, XML súborov a zápis do nich
- **irr::scene**- obsahuje všetko potrebné pre management scény, pre tvorbu rôznych typov uzlov a pre prácu s nimi
- **irr::video**- obsahuje triedy, ktoré slúžia na prácu s grafickým ovládačom, tvorbu materiálov a textúr pre objekty scény

Irrlicht engine používa na popis objektov použitých v scéne hierarchickú štruktúru, objekty sú v scéne reprezentované uzlami. Uzly v scéne je možné spravovať pomocou scene managera. Uzly je možné vytvárať, odstraňovať, nadstavovať ich polohu, rotáciu, materiály alebo animácie [2].

1.4 Základné kamene aplikácie vytvorenej v engine Irrlicht

Irrlicht device je najdôležitejšia časť aplikácie, ktorá používa Irrlicht Engine, cez device sa dá pristupovať ku všetkému čo pomocou Irrlichtu vytvoríme, po zavolaní funkcie createDevice() alebo createDeviceEx() engine sám vytvorí okno aplikácie.

Video driver je jedno z najdôležitejších rozhraní Irrlichtu. Je to rozhranie pre ovládač grafickej karty, ktoré spravuje vykreslovanie 2D a 3D objektov vo vytvorenej scéne a manipuláciu s textúrami.

Scene manager umožňuje vytváranie a správu scénových uzlov, načítanie modelov z rôznych formátov ktoré sú enginom podporované. Umožňuje vytvárať rôzne druhy kamier, svetlá alebo aj celý terén.

Event receiver je rozhranie pre objekt, ktorý má reagovať na udalosti zo vstupných zariadení. Receiver, ktorý ponúka Irrlicht je len veľmi jednoduchý a pre praktické použitie ho treba rozšíriť podľa požiadavkov aplikácie [2].

Smyčka while slúži na to, aby aplikácia bežala až dovtedy kým ju nezastavíme, cyklicky sa v nej obnovuje render scény, prípadne simulácia fyziky objektov. Smyčka while zaručuje, že engine dostane maximum systémového času.

```
// setup parameters for Irrlicht device
SIrrlichtCreationParameters params = SIrrlichtCreationParameters();
params.AntiAlias=true;                                  // antialiasing flag
params.Fullscreen=false;                                // fullscreen flag
params.WindowSize=dimension2d<s32>(1400,900);           // window dimensions
params.DriverType=EDT_OPENGL;                           // driver type
params.Stencilbuffer=true;                              // stencil buffer flag
params.Bits=32;                                         // color depth

// create Irrlicht device
IrrlichtDevice* device= createDeviceEx(params);
// declaration of pointer to video driver
video::IVideoDriver *driver = device->getVideoDriver();
// ddeclaration of pointer to scene manager
scene::ISceneManager *smgr = device->getSceneManager();
// create custom event receiver
myEventReceiver* receiver = new myEventReceiver(device);
// set custom event receiver as receiver for Irrlicht device
device->setEventReceiver(receiver);

// while loop where it all happens
while(device->run())
{
    // call begin scene before any rendering is done
    driver->beginScene(true, true, video::SColor(255,255,255,255));
    // draw everything that's in the scene graph
    smgr->drawAll();
    // end scene
    driver->endScene();
}
// drop device on exit
device->drop();
```

Obr. 1: Základná časť programu

2 IRR::SCENE- ROZHRANIE PRE MANAGEMENT SCÉNY

Namespace irr::scene obsahuje všetky potrebné triedy pre management scény. ISceneManager, pomocou ktorého sa modely načítajú do pamäti a umiestňujú do scény. Triedy ktoré reprezentujú rôzne typy objektov od jednoduchých statckých modelov až po modely s kostrovou animáciou, emitory častíc a managera kolízií. Pomocou funkcií tohoto rozhrania je možné vytvoriť pre aplikáciu komplexnú scénu.

2.1 Vkladanie objektu scény

Pri vytváraní scény je potrebné vytvoriť modely pomocou 3D modelovacieho programu a vyexportovať modely do formátu ktorý je podporovaný enginom Irrlicht. Engine podporuje množstvo formátov a ak nie, je možné vytvoriť si vlastný loader. Irrlicht ponúka tiež možnosť poskladať scénu z hotových modelov v programe IrrEdit. Z programu IrrEdit je možné celú scénu vyexportovať do formátu .irr a potom ju v aplikácii načítať celú jediným príkazom.

2.2 Kamera

Kamera patrí do triedy ICameraSceneNode a pomocou scene managera je možné do scény vložiť kameru typu Maya, ktorá má v sebe už implementovaný pohyb podobný pohybu kamery v softvéri pre tvorbu 3D obsahu Alias Wavefront Maya. Ďalej je možné vložiť do scény kameru typu FPS (First Person Shooter), ktorá má implementovaný pohyb z pohľadu prvej osoby. Spolu s kolíznym systémom Irrlichtu je možné vytvoriť jadro aplikácie typu FPS veľmi jednoducho. Ďaľšou možnosťou je vložiť do scény obyčajnú statickú kameru, s ktorou je možné pracovať ako s bežným scénovým uzlom [2].

```
// create FPS type camera using scene manager(smgr)
scene::ICameraSceneNode* camera = smgr->addCameraSceneNodeFPS();
```

Obr. 2: Vloženie kamery typu FPS do scény

2.3 Jednoduchý model

Jednoduchý model je reprezentovaný triedou IMesh, pridaním modelu do scény získame IMeshSceneNode, čiže uzol v ktorom sa zobrazuje model (jeden model je možné použiť pre viacero uzlov). Každý mesh sa skladá z MeshBufferov, každý MeshBuffer obsahuje v sebe informácie o vertexoch. Jeden MeshBuffer obsahuje vždy vertexy s rovnakými vlastnosťami, napríklad textúrou. Verzia 1.4 enginu Irrlicht podporuje nasledujúce formáty modelov:

- 3D Studio modely (.3ds)
- B3D súbory (.b3d)
- Alias Wavefront Maya (.obj)
- Cartography shop 4 (.csm)
- COLLADA (.xml, .dae)
- DeleD (.dmf)
- FSRad oct (.oct)
- Irrlicht scény (.irr)
- Irrlicht statické modely (.irrmesh)
- Microsoft DirectX (.x)
- Milkshape (.ms3d)
- My3DTools 3 (.my3D)
- OGRE modely (.mesh)
- Pulsar LMTools (.lmts)
- Quake 3 levely (.bsp)
- Quake 2 modely (.md2)
- STL 3D súbory (.stl)

```
// load mesh from .b3d file
IMesh* mesh= smgr->getMesh("hero.b3d");
// add node for this mesh into scene graph
IMeshSceneNode* nodeForMesh = smgr->addMeshSceneNode(mesh);
```

Obr. 3: Vloženie modelu do scény

2.4 Animovaný model

Animovaný model reprezentuje trieda IAnimatedMesh a uzol pre animovaný model reprezentuje trieda IAnimatedMeshSceneNode. Všetky podporované formáty uvedené vyžšie môžu byť použité na vytvorenie animovaného objektu, za predpokladu, že daný formát animácie podporuje. Od verzie 1.4 obsahuje engine Irrlicht nový systém animácií, ktorý umožňuje lepšiu prácu so zakostrenými modelmi. Nový animačný systém umožňuje programátorovi veľmi jednoducho pristupovať priamo k jednotlivým kostiam animačnej kostry a priamo nadstavovať ich rotácie a polohy [2].

```
// load animated mesh from .b3d file
IAnimatedMesh* animMesh= smgr->getMesh("hero.b3d");
 // add node for this mesh into scene graph
IAnimatedMeshSceneNode* node = smgr->addAnimatedMeshSceneNode(animMesh);
```

Obr. 4: Vloženie animovaného modelu do scény

2.5 Testovacie objekty

Testovacie objekty môžu byť do scény vložené pomocou funkcie scene managera, funkcia addSphereSceneNode vloží do scény guľu, addCubeSceneNode vloží kocku. Ako testovací terén sa dá použiť za behu generovaný mesh ktorý získame volaním funkcie addHillPlaneMesh [2].

```
// add sphere into scene graph using scene manager (smgr)
scene::ISceneNode* node = smgr->addSphereSceneNode();

// create hill plane mesh
IAnimatedMesh* mesh = smgr->addHillPlaneMesh("myHillPlaneMesh",
    core::dimension2d<f32>(10,10),          //tile size
    core::dimension2d<u32>(10,10),          //tile count
    0,                                      //material
    1,                                      //height of hills
    core::dimension2d<f32>(5,5),            //number of hills
    core::dimension2d<f32>(10,10));         //texture repeat count
// create water surface node from hill plane mesh
IAnimatedMeshSceneNode* node = 0;
node = smgr->addWaterSurfaceSceneNode( mesh->getMesh(0), 3, 300, 3);
```

Obr. 5: Vloženie testovacích objektov do scény

2.6 Billboardy

Billboardy sú 2D objekty ktoré sú vložené v priestore a vždy smerujú normálou do kamery, je možné ich použiť napríklad ako model pre svetlo alebo strom. V Irrlichte je

tento typ reprezentovaný triedou IBillboardSceneNode a do scény sa vkladá funkciou scene managera addBillboardSceneNode, prípadne addBillboardTextSceneNode, ktorá do scény vloží text ako billboard [13].

```
// create light using scene manager(smgr)
ISceneNode* svetlo;
svetlo = smgr->addLightSceneNode(
0,                                                  //parent
core::vector3df(0,0,0),                             //position
video::SColorf(1.0f, 0.6f, 0.7f, 1.0f),             //color
1200.0f);                                           //radius
// cretae billboard for light attached to the light
ISceneNode* billboard;
billboard = smgr->addBillboardSceneNode(
svetlo,                                             //parent
core::dimension2d<f32>(50, 50));                    //size
```

Obr. 6: Vloženie billboardu spolu so svetlom

2.7 Časticový systém

Časticový systém zastupuje trieda IParticleSystemSceneNode, na to aby uzol vytvorený pre časticový systém naozaj častice emitoval treba vytvoriť navyše emitor. Irrlicht ponúka hneď niekoľko typov emitorov:

- IParticleBoxEmitter- emitor z kocky
- IParticleMeshEmitter- emitor z modelu
- IParticleRingEmitter- prstencový emitor
- IParticleSphereEmitter- emitor z guľe

Správanie častíc sa dá ovplyvniť pomocou afektoru, v Irrlichte nájdeme viac druhov afektorov:

- IParticleGravityAfector- ovplyvňuje častice gravitačnou silou s ľubovoľným smerom
- IParticleAttractionAfector- priťahuje alebo odpudzuje častice od určitého bodu, v osi X, Y alebo Z
- IParticleFadeOutAfector- spôsobuje pozvolné vyfarbovanie alebo miznutie častíc podľa zadaného času
- IParticleRotationAfector- ovplyvňuje rotáciu častíc tak, že častice rotujú podľa zadanej rýchlosti pre každú os rotácie okolo zadaného bodu [2]

```
// create particle system without default emitter
scene::IParticleSystemSceneNode* system = 0;
system = smgr->addParticleSystemSceneNode(false);
// set size of particles
system->setParticleSize(core::dimension2d<f32>(10.0f, 10.0f));
// create box emitter
scene::IParticleEmitter* emitter = system->createBoxEmitter(
    core::aabbox3d<f32>(-7,0,-7,7,1,7),     //box
    core::vector3df(0.0f,0.06f,0.0f),       //direction and speed
    80,                                     //min emitted per sec
    100,                                    //max emitted per sec
    video::SColor(0,255,255,255),           //min start color
    video::SColor(0,255,255,255),           //max start color
    800,                                    //min lifetime
    2000);                                  //max lifetime
// set our amitter as emitter for particle system
system->setEmitter(emitter);
// create fade out affector
scene::IParticleAffector* affector =
    system->createFadeOutParticleAffector();
//set our afector as particle affector of particle system
system->addAffector(affector);
```

Obr. 7: Vloženie časticového systému, emitoru a afektoru

2.8 Terén

Terén je možné vytvoriť ako ITerrainSceneNode, ktorý umožňuje zobrazovanie veľkých modelov krajiny. Terén sa tvorí na základe informácií získaných z výškovej mapy, tzv. heightmap. Výšková mapa je len jednoduchá čierno-biela textúra. Svetlé časti heightmapy predstavujú miesta terénu s väčšou nadmorskou výškou, tmavé časti miesta terénu s nižšou nadmorskou výškou. To, ako veľmi budú farby heightmapy ovplyvňovať výšku terénu sa dá nadstaviť. Terén používa algoritmus CLOD (Continuous Level of Detail), upravenú verziu algoritmu LOD(Level of Detail) , ktorý upravuje geometriu zobrazovaných predmetov v závislosti na ich vzdialenosti od kamery. Prametre LOD sa dajú nadstaviť [2].

```
// create terrain scene node using scene manager(smgr)
scene::ITerrainSceneNode* terrain = smgr->addTerrainSceneNode(
    "heightmap.bmp",                          // used hightmap
    0,                                        // parent
    -1,                                       // id
    core::vector3df(0.f, 0.f, 0.f),           // position
    core::vector3df(0.f, 0.f, 0.f),           // rotation
    core::vector3df(100.f, 10f, 100.f),       // scale
    video::SColor ( 255, 255, 255, 255 ),     // vertexColor
    15,                                       // max LOD
    scene::ETPS_17,                           // patch Size
    10);                                      // smooth Factor
```

Obr. 8: Vytvorenie terénu

2.9 Tieň

Tieň sa vytvára volaním funkcie addShadowVolumeSceneNode na objekt, ktorý má vrhať tieň, farbu aj priehľadnosť tieňa je možné nadstaviť. K tomu, aby Irrlicht mohol renderovať tiene je nutné aby bolo v scéne svetlo [4].

```
// create shadow
node->addShadowVolumeSceneNode(
0,                              // id
false,                          // zfail method flag
10000);                         // distance of light(inf)
//set alpha and color of shadow
smgr->setShadowColor(video::SColor(150,0,0,0));
```

Obr. 9: Vytvorenie tieňa pre uzol

2.10 Detekcia kolízií

Detekcia kolízií sa v Irrlichte ralizuje tak, že na základe objektu, s ktorým majú ostatné objekty v scéne kolidovať (napr. mapa) sa vytvorí ITriangleSelector. Keď je triangle selector vytvorený stačí už len na každý objekt, ktorý má s mapou kolidovať pridať tzv. collision animator, ktorý môže na objekt vplývať gravitačnou silou a zabráni tomu, aby objekt prechádzal cez steny triangle selectoru [4].

```
// create node for map using scene manager(smgr)
myMapNode = smgr->addOctTreeSceneNode(myMapMesh);
// create selector for map using scene manager(smgr)
scene::ITriangleSelector* selector = 0;
selector = smgr->createOctTreeTriangleSelector(
myMapMesh->getMesh(0),                  // mesh
myMapNode,                              // node
128);                                   // min polygons per node
// set selector as triangle selector of our map
myMapNode->setTriangleSelector(selector);
// example of adding collision animator to object(camera)
scene::ISceneNodeAnimator* anim = smgr->createCollisionResponseAnimator(
        selector,                       // triangle selector
        myCamera,                       // object node
        core::vector3df(30,50,30),      // elipsoid radius
        core::vector3df(0,-3,0),        // gravity force
        core::vector3df(0,50,0));       // elipsoid translation
// add colision to my object
myCamera->addAnimator(anim);
```

Obr. 10: Vytvorenie kolíznej mapy a kolízie pre uzol

2.11 Animátory

Animátory slúžia na animovanie uzlu, môžu meniť jeho pozíciu, rotáciu, veľkosť. Animátory su reprezentované triedou ISceneNodeAnimator, vytvárajú sa pomocou scene managera, ktorý podporuje tieto duhy animátorov:

- FlyCircleAnimator- rotuje s uzlom okolo zadaného stredu so zadanou rýchlosťou, polomerom a smerom
- FlyStraightAnimator- pohybuje s uzlom po priamke medzi dvoma bodmi za zadaný čas, pohyb je možné cyklicky opakovať
- FollowSplineAnimator- pohybuje uzol po dráhe ktorá je určená poľom bodov zadanou rýchlosťou po prepočítanej krivke, pomocou parametra tightness môže užívateľ určiť presnosť
- RotationAnimator- rotuje uzol okolo jeho osi zadanou rýchlosťou
- TextureAnimator- postupne mení textúry uzlu zo zadaného poľa textúr zadanou rýchlosťou, je možné opakovanie animácie

```
// create fly straight animator for node using scene manager(smgr)
scene::ISceneNodeAnimator* anim = 0;
anim = smgr->createFlyStraightAnimator(
core::vector3df(100,0,60),                    // start point
core::vector3df(-100,0,60),                   // end point
2500,                                         // time in ms
true);                                        // loop flag

node->addAnimator(anim);
```

Obr. 11: Vytvorenie a nadstavenie animátora pre uzol

2.12 Načítanie celej scény

Do aplikácie je možné použitím vlastného typu enginu Irrlicht .irr načítať objekt, ktorý má tvoriť scénu aj s vlastnosťami špecifickými pre tento engine. Súbory .irr vie na základne zostavenej scény generovať napríklad program IrrEdit, existuje aj exportér pre modelovací program Blender. Výhodou tohoto formátu je, že dokáže uchovať všetky informácie o scéne. Pomocou formátu .irr je preto možné načítať komplexnú scénu so všetkými objektami, nadstavenými materiálmi a textúrami, svetlami a časticovými systémami [12].

```
// loading of complete scene with scene manager(smgr)
smgr->loadScene("myCompleteScene.irr");
```

Obr. 12: Načítanie komplexnej scény z formátu .irr

3 IRR::VIDEO- ROZHRANIE PRE SPRÁVU GRAFICKÝCH TECHNOLÓGIÍ

Namespace irr::video obsahuje funkcie a triedy ktoré umožňujú dotvárať výsledný vzhľad aplikácie. Tu je možné dať objektom scény rôzne materiály, načítať rôzne textúry. Väčšina týchto operácií sa vykonáva pomocou triedy IVideoDriver, ktorá je jednou z najdôležitejších tried enginu. Poskytuje funkcie ako načítanie textúr, tvorbu textúr, vykreslovanie grafických primitív, nadstavenie zorného poľa aplikácie. Namespace ďalej obsahuje triedy, ktoré reprezentujú materiály, farby v rôznych formátoch, textúry atď.

Irrlicht Engine využíva namespace irr::video ako rozhranie medzi grafickými API. Prostredníctvom nižšie uvedených funkcií je možné pracovať s ktorýmkoľvek z podporovaných grafických API: Direct3D9, Direct3D8.1, OpenGL, Irrlicht renderer a Apfelbaum renderer. Zdrojový kód je úplne nezávislý na tom, ktoré API sa použije, všetko, čo je třeba spraviť aby aplikácia využívala iné grafické rozhranie je vytvoriť Irrlicht device s iným parametrom pre rozhranie [2].

3.1 Video Driver

Video Driver je jedným z najdôležitejších rozhraní enginu. Trieda IVideoDriver obsahuje 80 funkcií, pomocou ktorých je možné riadiť vykreslovanie scény a získavať o nej informácie. Pomocou funkcií beginScene a endScene sa začína resp. ukončuje vykreslovanie celej scény v hlavnej smyčke programu. Pomocou video driveru je tiež možné meniť projekčnú maticu, maticu pohľadu a transformačnú maticu. Je tiež možné nadstaviť jednotný materiál všetkým objektom scény, hmlu, svetlo, zorné pole. Pomocou video drivera tiež získavame informácie o zobrazovanej scéne ako napríklad zorné pole, rýchlosť vykreslovania (FPS), počet textúr, počet materiálov, počet svetiel atď. [2].

3.2 Grafické primitíva

Primitíva sa vykresľujú volaním funkcií triedy IVideoDriver. Je možné vykresliť 2D obrázok, 2D čiaru, 2D pravideľný mnohouholník, 2D obdĺžnik, 3D kocku, 3D čiaru, 3D trojuholník.

```
// draw 2D rectangle with custom color(ARGB) and dimensions
driver->draw2DRectangle(video::SColor(100,255,255,255),
                        core::rect<s32>(0, 0, 20, 20));
```

Obr. 13: Vykreslenie jednoduchého grafického prvku

3.3 Materiály

Materiály reprezentuje trieda SMaterial, každý objekt je vytvorený s defaultným materiálom typu solid a defaultnými farbami. K materiálu scénového uzlu sa pristupuje volaním funkcie uzlu getMaterial. Irrlicht engine podporuje nasledujúce druhy materiálov:

- EMT_SOLID- štandardný difúzny materiál
- EMT_SOLID_2_LAYER- difúzny materiál, ktorý používa 2 textúry, ktoré sa miešajú na základe alpha hodnoty vrcholov
- EMT_LIGHTMAP- materiál používajúci štandardnú lightmap techniku, jedna textúra je difúzna, druhá light mapa
- EMT_LIGHTMAP_ADD- štandardný lightmap materiál, farba druhej textúry sa pripočítava k farbe prvej
- EMT_LIGHTMAP_M2- štandardný lightmap materiál, farby textúr sa násobia dvomi, výsledok je väčšia svetlosť
- EMT_LIGHTMAP_LIGTNING, EMT_LIGHTMAP_LIGTNING_M2- verzie predchádzajúcich textúr, ktoré podporujú dynamické osvetlenie
- EMT_DETAIL_MAP- materiál, ktorý využíva dve textúry, jednu ako farebnú mapu a druhú, zvyčajne s väčším merítkom ako detail mapu
- EMT_SPHERE_MAP- materiál, ktorý simuluje na objekte odrazy okolia tým, že ako textúra sa použie tzv. sphere mapa
- EMT_REFLECTION_2_LAYER- materiál s odrazom
- EMT_TRANSPARENT_ADD_COLOR- priehľadný materiál, priehľadnosť sa počíta na základe farieb v obrázku
- EMT_TRANSPARENT_ALPHA_CHANNEL- priehľadnosť je počítaná na základe alfa kanálu textúry
- EMT_TRANSPARENT_ALPHA_CHANNEL_REF- obdoba predchádzajúceho materiálu, pixel sa vykreslí len ak je jeho alfa hodnota väčšia ako 127, čo spôsobuje ostré hrany, ale zároveň väčšiu rýchlosť vykreslovania

- EMT_TRANSPARENT_VERTEX_ALPHA- priehľadnosť na základe alfa hodnoty vrcholov
- EMT_TRANSPARENT_REFLECTION_2_LAYER- priehľadný materiál s odrazom, dá sa použiť napríklad ako sklo
- EMT_NORMAL_MAP_SOLID- difúzny materiál, ktorý využíva jednu textúru ako farebnú mapu a druhú ako normal mapu
- EMT_NORMAL_MAP_TRANSPARENT_ADD_COLOR- ako predchádzajúci materiál, ale navyše podporuje priehľadnosť
- EMT_NORMAL_MAP_TRANSPARENT_VERTEX_ALPHA- ako predchádzajúci materiál, priehľadnosť sa však určuje podľa alfa hodnoty vrcholov
- EMT_PARALLAX_MAP_SOLID- materiál, ktorý využíva parallax mapping na renderovanie normal mapy, výsledkom je realistickejší vzhľad ako u EMT_NORMAL_MAP_SOLID
- EMT_PARALLAX_MAP_TRANSPARENT_ADD_COLOR-ako predchádzajúci materiál, ale navyše podporuje priehľadnosť
- EMT_PARALLAX_MAP_TRANSPARENT_VERTEX_ALPHA- ako predchádzajúci materiál, priehľadnosť sa však určuje podľa alfa hodnoty vrcholov

Okrem druhu materiálu je možné nadstaviť ešte tzv. material flags, sú to boolean hodnoty pomocou ktorých je možné určiť ďaľšie vlastnosti materiálu. Dostupné flags:

- EMT_WIREFRAME- drátený model
- EMT_POINTCLOUD- renderuje len vrcholy
- EMT_GOURAD_SHADING- použitie shaderu typu Gourad
- EMT_LIGHTNING- určuje, či má byť model osvetlený
- EMT_ZBUFFER- použitie Z-bufferu
- EMT_ZWRITE_ENABLE- povolenie zápisu do Z-bufferu
- EMT_BACK_FACE_CULLING- vykeslovanie stien objektu, ktoré sú odvrátené od kamery
- EMT_BILINEAR_FILTER- bilineárne filtrovanie textúr
- EMT_TRILINEAR_FILTER- trilineárne filtrovanie textúr
- EMT_ANISOTROPC_FILTER- anisotropické filtrovanie textúr

- EMT_FOG_ENABLE- povolenie hmly na materiál
- EMT_NORMALIZE_NORMALS- vhodné při zväčšovaní aleo zmenšovaní dynamicky osvetleného modelu, lebo pri zmene veľkosti sa zmení aj veľkosť normál a osvetlenie by potom nebolo správne [2]

```
// disable dynamic lightning of node
node->setMaterialFlag(EMF_LIGHTING,false);
// enable anisotropic texture filter
node->setMaterialFlag(EMF_ANISOTROPIC_FILTER,true);
// set material type of material 0 to solid
node->getMaterial(0).MaterialType = EMT_SOLID;
// set material type of material 1 to 2 layer reflective
node->getMaterial(1).MaterialType = EMT_REFLECTION_2_LAYER;
```

Obr. 14: Nadstavenie materilálu uzlu

3.4 Vlastný materiál

Vlastný Materiál je možné vytvoriť použitím nového shaderu, čo vyžaduje znalosť tvorby shaderov. Irrlicht podporuje tzv. high level shadery (HLSL, GLSL) a low level shadery (ARB assembler) [7].

Keď je dostupný vlastný shader stačí použiť funkcie rozhrania IGPUProgrammingServices. Pomocou tohoto rozhrania je možné vytvoriť nový materiál použitím high level alebo low level shaderu buď zapísaného priamo v premenej typu string alebo zo súboru. Na to, aby bolo možné materiál použiť je treba zavolať jednu z funkcií addShaderMaterial, addShaderMaterialFromFiles, addHighLevelShaderMaterial a addHighLevelShaderMaterialFromFles, ktoré vracajú 32-bitové číslo nového typu materiálu [4].

```
// get GPU programing services
video::IGPUProgrammingServices* gpu = driver->getGPUProgrammingServices()
// id number for material 1
s32 material1 = 0;
// id number for material 2
s32 material2 = 0;

// create material from high level shader
newMaterialType1 = gpu->addHighLevelShaderMaterialFromFiles(
            "vertexShader.hlsl",     // vertex shader program
            "vertexMain",            // entry function of vertex program
            video::EVST_VS_1_1,      // shader version
            "pixelShader.hlsl",      // pixel shader program
            "pixelMain",             // entry function of pixel program
            video::EPST_PS_1_1,      // shader version
            0,                       // callback
            video::EMT_SOLID);       // base material

// create material from low level shader
newMaterialType2 = gpu->addShaderMaterialFromFiles(
            "vertexShader.vsh",      // vertex shader program
            "pixelShader.psh",       // pixel shader program
            0,                       // callback
            video::EMT_TRANSPARENT_ADD_COLOR);  // base material

 // set material1 for material 0
 node->getMaterial(0).MaterialType = (video::E_MATERIAL_TYPE)material1;
 // set material2 for material 1
 node->getMaterial(1).MaterialType = (video::E_MATERIAL_TYPE)material2;
```

Obr. 15: Použitie vlastného shaderu na vytvorenie materiálu

3.5 Textúry

Textúry reprezentuje trieda ITexture. Zo súboru sa textúry získajú volaním funkcie getTexture video driveru. Ako parameter pre funkciu getTexture je možé použiť buď ukazateľ na súbor textúry, alebo cestu k tomuto súboru. Textúry vytvorené pomocou jedného drivera nie sú kompatibilné s iným driverom. Takže napríklad textúra vytvorená pomocou OpenGL by po zmene driveru na Direct3D bola odmietnutá.

Irrlicht podporuje štyri farebné formáty textúr: štandardný 16-bitový, štandardný 32-bitový, 24-bitový formát (8 bitov na R, G, B, bez priehľadnosti), 16-bitový formát Irrlichtu, ktorý využíva 5 bitov pre každú zo zložiek R, G, B a jeden bit ktorý určuje priehľadnosť.

Ak veľkosti strán textúry nie sú mocninami 2, potom bude textúra Irrlichtom automaticky upravená na rozmery, ktoré sú mocninami 2. Pôvodná veľkosť textúry sa dá zistiť volaním funkcie getOriginalSize, aktuálnu veľkosť zistíme volaním getSize [2].

Irrlicht podporuje nasledujúce formáty textúr: BMP, JPG, TGA, PCX, PNG, PSD.

```
// set texture 0 of material 0
chassisN->getMaterial(0).setTexture(
0,driver->getTexture("diffuseMap.png"));
// set texture 1 of material 0
chassisN->getMaterial(0).setTexture(
1,driver->getTexture("reflectionMap.png"));
```

Obr. 16: Nadstavenie textúr

3.6 Práca s obrázkami

Na čítanie obrázkov slúži trieda IImageLoader, jednoducho pomocou funkcie loadImage je možné obrázok načítať. Pomocou boolean funkcií isALoadableFileExtension a isALoadableFileFormat je možné zistiť, či sa dá obrázok s danou príponou resp. formátom načítať. Na zápis slúži trieda IImageWriter a jej funkcia writeImage. Na kontrolu, či je možné zápis vykonať slúži boolean funkcia isAWriteableFileExtension. Obrázok reprezentuje trieda IImage, ktorá umožňuje nadstaviť farbu jednotlivým pixelom obrázku alebo získať kópiu obrázku so zmeneným merítkom. O obrázku sa tiež dajú získať informácie ako farba pixelu, rozmery obrázku, počet bytov na pixel, počet bitov na pixel, farebný formát atď.

3.7 Možnosti zobrazenia

Informácie o dostupných možnostiach zobrazenia (video módoch) predstavuje trieda IVideoModeList. Informácie je možné získať pomocou Irrlicht device zavolaním jeho funkcie getVideoModeList. Takto sa dajú získať informácie o farebnej hĺbke plochy, rozlíšenie plochy a počet dostupných možností zobrazenia. Každý video mód má vlastný index a pomocou funkcií getVideoModeDepth a getVideoModeResolution je možné získať informácie o jeho farebnej hĺbke resp. o rozlíšení [2].

II. PRAKTICKÁ ČASŤ

4 TVORBA INTERAKTÍVNEJ APLIKÁCIE

Nasledujúca kapitola popisuje tvorbu interaktívnej aplikácie(hry) pomocou enginu Irrlicht. Okrem vstavaných funkcií sú tiež popísané funkcie použitého externého fyzikálneho enginu a tvorba použitých modelov.

4.1 Fyzikálny engine

Irrlicht engine obsahuje veľmi jednoduchú a rýchlu detekciu kolízií, ktorá ale nepostačuje pre tvorbu fyzikálnej simulácie pohybu. Pre tieto účely je nutné použiť fyzikálny engine. Vytvorená hra demonštruje použitie fyzikálneho enginu Newton. Newton sa nepoužíva len na tvorbu hier, ale aj na tvorbu realtimeových fyzikálnych simulácií. Má jednoduchú a výstižnú dokumentáciu a je ľahko integrovateľný do Irrlichtu [6].

4.1.1 Newton world

V aplikácii využívajúcej tento fyzikálny engine je nutné vytvoriť fyzikálny svet. Newton world je kontajner, do ktorého sa vkladajú ostatné objekty. Na prácu s vloženými objektami slúžia tzv. callback funkcie, ktoré sa volajú automaticky pokiaľ nie je objekt zmrazený. Callback funkcia môže na objekt napríklad aplikovať silu alebo krútiaci moment.

Na to aby bolo možné prepojiť fyzikálne objekty a grafické objekty je nutné každému fyzikálnemu objektu nadstaviť tzv. user data, kde je možné uložiť pointer na grafický objekt alebo na nejakú triedu, ktorá ho obsahuje. Pohyb grafického objektu sa zaisťuje v transform callback funkcii. Táto funkcia je zavolaná vždy keď fyzikálny objekt zmení polohu alebo rotáciu. Aktualizácia polohy grafického objektu prebieha tak, že sa získajú user data z fyzikálneho objektu. Po získaní pointeru na user data sa grafickému objektu len jednoducho nadstaví rovnaká poloha a rotácia jako ma fyzikálny objekt [6].

4.1.2 Dynamické objekty

Každému newtonovskému objektu je možné nadstaviť množstvo parametrov, na základe ktorých sa bude počítať simulácia. Je možné nadstaviť pružnosť materiálu, koeficient trenia, hmotnosť, ťažisko. Dôležité je nadstaviť správny tvar objektu, je možné použiť dostupné grafické primitíva: kocka, guľa, ihlan, valec, elipsoid. Grafické primitíva sú vhodné pre jednoduché objekty alebo pre objekty, ktorých tvar je možné aproximovať. Pre

objekty, ktorých tvar nie je možné aproximovať slúži typ convex hull, kolízny objekt sa v tomto prípade vytvorí na základe množiny vertexov. Týmto spôsobom je tiež možné vytvoriť zložitejší grafický objekt a pre jeho reprezentáciu vo fyzikálnom svete použiť jednoduchší model, ktorý sa nebude vôbec zobrazovať. Ten nebude taký náročný na výpočet kolízií a zároveň presnejšie kopíruje požadovaný tvar ako niektoré dostupných primitív.

Komplexný objekt, napríklad terén je možné vytvoriť ako tree collision na základe množiny vertexov. Typ tree collision je optimalizovaný na vysoký výkon pre veľké modely [6].

```
// position matrix for physic body
matrix4 mat;
// get mesh for object
scene::IAnimatedMesh* c = smgr->getMesh("models/testcube.b3d");
// create scene node from mesh
scene::ISceneNode* cn = smgr->addAnimatedMeshSceneNode(c);
// set scale of scene node
cn->setScale(vector3df(0.05,0.05,0.05));
// create Newton body for object using cube primitive
NewtonBody* body=NewtonCreateBody(world,NewtonCreateBox(world,1,1,1,0));
// set scene node as user data for physic object
NewtonBodySetUserData(body, cn);
// set matrix
NewtonBodySetMatrix(body, &mat[0]);
// set mass matrix
NewtonBodySetMassMatrix(body, 20, 20,20,20);
// turn on autofreeze
NewtonBodySetAutoFreeze(body, 1);
// set transformation callback fuction
NewtonBodySetTransformCallback(body,setTransformNode);
// set force and torque callback fuction
NewtonBodySetForceAndTorqueCallback(body, addGravityForce);
// unfreeze physic body, so physics can be calculated
NewtonWorldUnfreezeBody(world, body);
```

Obr. 17: Vytvorenie jednoduchého objektu so simuláciou fyziky

4.1.3 Vozidlo

Newton engine ponúka základné funkcie pre tvorbu real-time simulácie jednoduchého vozidla. Newtonovské vozidlo neponúka žiadne pokročilé funkcie, je to len jednoduchý model vozidla s možnosťou pripojiť kolesá a nadstaviť parametre pruženia. Vozidlo tvorí jedno hlavné rigid body pre karosériu, k nemu sú pripojené kolesá tak, že je možné s nimi otáčať a simulovať funkciu podvozku [6].

Vozidlu je možné nadstaviť parametre rovnako ako iným dynamickým objektom, navyše je však možné nadstavovať rotáciu kolesám, tvrdosť podvozku, pozíciu kolies atď. Z vytvoreného vozidla je možné získavať informácie o rýchlosti, uhlovej rýchlosti kolies, o tom, či je niektoré z kolies vo vzduchu alebo či vozidlo stratilo trakciu s vozovkou.

4.2 Tvorba objektov pre scénu

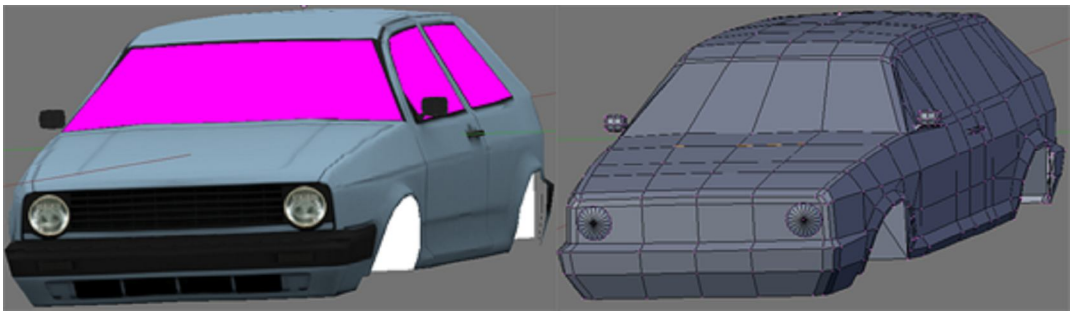
Súčasťou tvorby 3D aplikácie je okrem tvorby logiky a fyziky objektov tiež tvorba ich vzhľadu. Na tvorbu modelov, animácií a textúrovanie bol použitý program Blender. Blender je open source program pre tvorbu 3D obsahu, ktorý zahŕňa nástroje pre modelovanie, textúrovanie, animovanie, renderovanie, prácu s videom a zvukom atď. Keďže popis programu Blender nie je cieľom tejto práce, je popis tvorby modelov, textúr a animácií stručný, zachytáva len hlavné kroky pri ich tvorbe.

4.2.1 Modelovanie objektov

Pri tvorbe modelov pre realtime aplikáciu je nutné udržať detailnosť modelov na prípustnej úrovni. Vo všeobecnosti platí čím menej vertexov tým lepšie, záleží však tiež na dôležitosti daného modelu v scéne. Všetky použité modely sú low-poly a boli vytvorené štandardnými modelovacími technikami. Modely domov boli vytvorené prevažne technikou modelovania z kocky, zložitejšie modely technikou poly by poly teda extrudovaním a transformáciami jednotlivých vertexov. Počet vertexov resp. strán musí byť nízky nielen kvôli rýchlejšiemu vykreslovaniu scény ale aj kvôli náročnosti fyzikálnej simulácie a kolízií.

Všetky vytvorené objekty boli z Blenderu vyexportované do formátu .b3d (Blitz3D). Formát .b3d bol použitý pretože je veľmi rýchly, podporuje kostrovú animáciu objektov a má veľmi dobrú podporu ako pri exportovaní z Blenderu tak aj pri importovaní do Irrlichtu. Na export z Blenderu bol použitý skript Blitz3D Exporter v2.03.

Nepohyblivé objekty scény, teda domy, cesty, krajina boli pred použitím v Irrlichte importované do programu irrEdit. IrrEdit je voľne dostupný editor 3D sveta, nie je v ňom možné modely tvoriť, je možné z nich len poskladať mapu. IrrEdit bol použitý na nadstavenie príslušných materiálov a textúr všetkým objektom. Celá scéna bola vyexportovaná do formátu .irr, ktorý následne umožnil vytvoriť celú scénu automaticky jedným príkazom.



Obr. 18: Low poly objekt s textúrov a bez textúry

4.2.2 Textúrovanie objektov

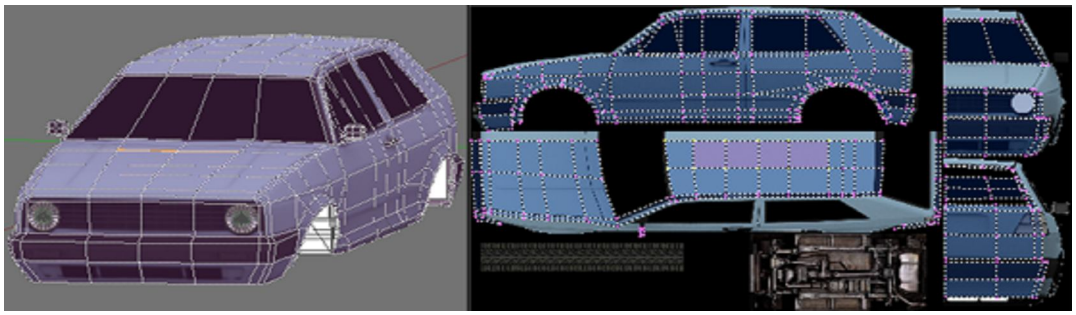
Každý vymodelovaný objekt je pred použitím v hre potrebné otextúrovať. Každý vertex modelu okrem informácie o jeho pozícii v 3D priestore uchováva tiež jeho 2D súradnice na textúre, tzv. UV koordináty. Upravovaním UV koordinátov jednotlivých vertexov boli všetky textúry v Blenderi namapované na vytvorené modely.

Na textúrovanie boli použité buď fotky, textúry vytvorené v programe Blender alebo voľne dostupné textúry z internetových repozitárov. Na získanie textúr boli použité repozitáre ako napríklad CG Textures (http://www.cgtextures.com), Forest Texture Library (http://textures.forrest.cz) , Mayang free textures (http://mayang.com/textures), TextureKing (http://www.textureking.com) a TurboSquid (http://www.turbosquid.com). Textúry boli pred mapovaním na objekty upravené pomocou 2D editoru. Ako 2D editor bol použitý open source program GIMPshop, ktorý je modifikáciou open source editoru GIMP. GIMPshop sa líši od GIMPu tým, že používa užívateľské rozhranie podobné komerčnému editoru Adobe Photoshop. Po namapovaní textúr na objekty pomocou UV koordinátov boli pomocou funkcie Bake programu Blender do textúr „zapečené“ tiene [14].

Textúry automobilov boli vytvorené podľa high-poly modelov a potom namapované na low–poly modely, takto bolo možné na jednoduchom objekte (menej ako 1000 vertexov) vytvoriť detaily na podobnej úrovni ako na pôvodnom modeli (viac ako 100 000 vertexov).

Pri textúrovaní scény boli vytvorené lightmapy. Lightmapy je možné vytvoriť v programe Blender funkciou Bake. Funkcia Bake umožňuje zapiecť do textúry vyrenderované informácie. Vytvorením lightmáp je možné oddeliť textúru, ktorá zobrazuje farby a detaily od osvetlenia. Pre lightmapu je možné vytvoriť vlastné UV koordináty. To

bolo s výhodou využité napríklad pri tvorbe textúry trávnika, kedy sa môže textúra s farbou a detailami opakovať, zatiaľ čo lightmapa je na trávniku namapovaná len raz.

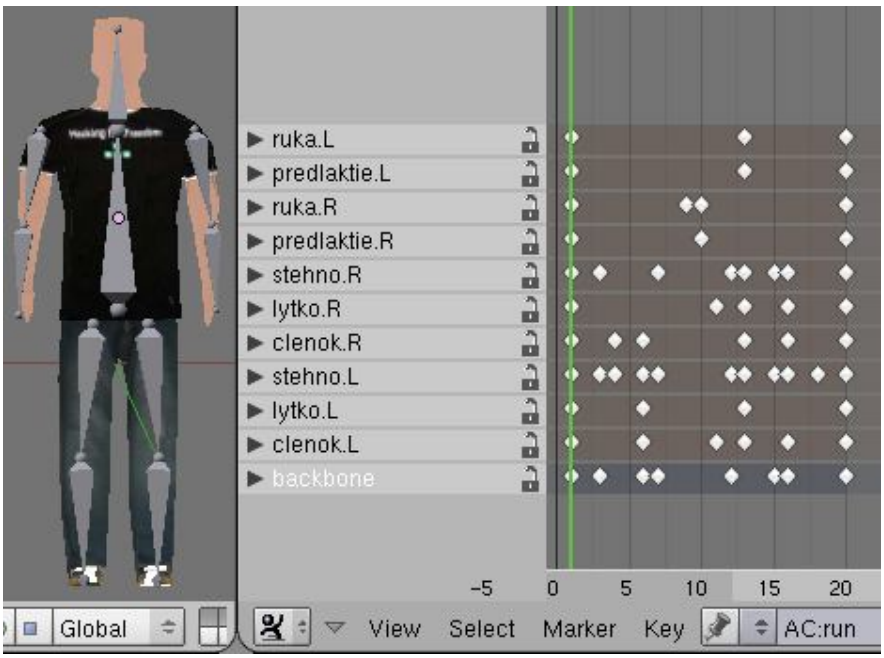


Obr. 19: Mapovanie textúry pomocou UV koordinátov

4.2.3 Animovanie objektov

Niektoré objekty bolo nutné pred použitím v hre naanimovať. Animácie boli taktiež vytvorené v programe Blender, pomocou kostrovej animácie. Pre objekty, ktoré majú byť animované je nutné vytvoriť kostru, kde sa každej hlavnej časti tela priradí kosť, jednotlivé kosti sú spojené kĺbami. Aby bolo možné animovať model je nutné každej kosti priradiť skupinu vertexov, ktoré má ovplyvňovať. Podľa váhy, aká sa jednotlivým vertexom nadstaví budú tieto vertexy ovplyvnené. Je tak možné docieliť prirodzeného a plynulého pohybu vertexov v oblasti kĺbov.

Jednotlivé animácie boli vytvorené pomocou Action editoru. V niektorých framoch animácie sa vložia tzv. kľúče, ktoré určujú polohu alebo rotáciu kostí. Action editor potom automaticky interpoluje polohu a rotáciu kostí medzi dvoma po sebe nasledujúcimi kľúčami. Takýmto spôsobom boli vytvorené animácie chôdze, behu, výskoku a čupnutia pre hrdinu a animácie otvorenia dverí pre autá.

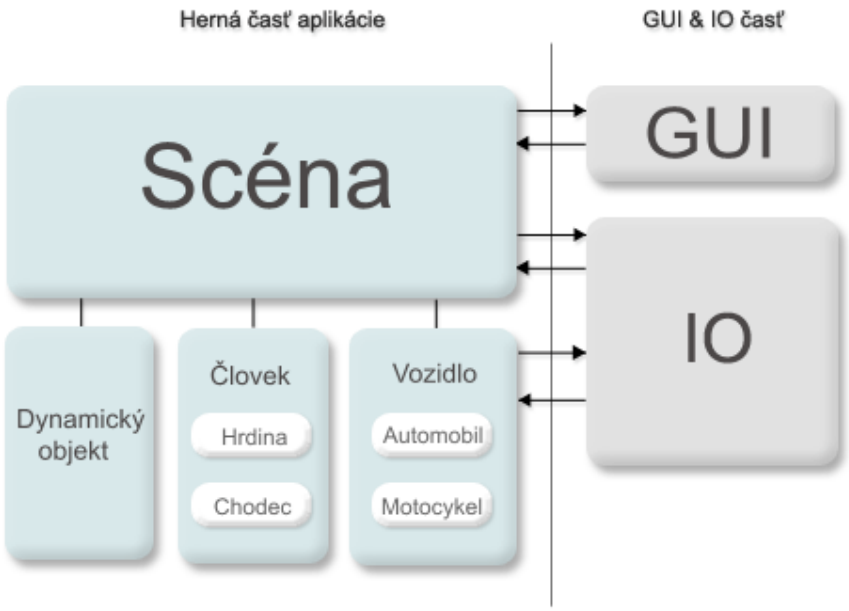


Obr. 20: Zakostenie hrdinu a Action editor programu Blender

4.3 Architektúra aplikácie

Aplikácia bola vytvorená objektovo, hlavný objekt hry definuje trieda Scene, ktorá spravuje všetko, čo bude v scéne vytvorené. Na začiatku sa vytvorí pre scénu grafický objekt typu OctTree, potom fyzikálny svet NewtonWorld. Podľa parametrov uložených v súbore .xml sa vytvoria v scéne vozidlá, chodci a dynamické objekty. Parametre vozidla sú taktiež uložené v súbore typu .xml. Pretože použitý fyzikálny engine Newton nie je písaný objetkovo, sú callback funkcie pre objekty s fyzikou definované staticky.

Srdcom aplikácie je smyčka while, v ktorej sa cyklicky obnovuje grafický svet, fyzikálny svet a spracovávajú sa vstupy z klávesnice a myši. V tejto smyčke sa tiež nachádza čakacia smyčka, ktorá zaisťuje maximálnu rýchlosť hry 80 fps. 80 obrázkov za sekundu bolo zvolených preto, že takú rýchlosť aplikácia dosiahne aj na menej výkonnom počítači a zároveň preto, že výpočty fyziky podávaju pri tejto rýchlosti postačujúce výsledky.



Obr. 21: Hierarchia objektov v aplikácii

4.3.1 Vytvorenie scény

Tvorba scény sa spustí zavolaním konštruktoru triedy Scene, ktorému sa ako prvý parameter predá string s cestou k mape. Následne sa vytvorí grafická mapa zo súboru .irr.

Na základe grafického modelu mapy sa tiež spustí tvorba fyzikálnej mapy. Fyzikálna mapa sa tvorí tak, že sa postupne prechádzajú všetky mesh buffery modelu. Pre každý triangle sa vytvorí pole, ktoré obsahuje informácie o polohách vertexov, ktoré sa predá Newtonu, ten si na základe týchto súradníc vytvorí vlastnú mapu. Pri tvorbe mapy je tiež použitá funkcia Newtonu, ktorá optimalizuje mapu tak, že sa snaží odstrániť nadbytočné vertexy.

Po vytvorení statickej časti scény sa začne tvorba dynamických objektov, na základe informácií zo XML súboru sa postupne vytvoria chodci, automobily, motocykle, hrdina a pohyblivé predmety. Tiež sa vytvorí kamera typu 3rd person.

4.3.2 Event receiver

Event receiver je objekt, ktorý reaguje na udalosti. Trieda IEventReceiver, ktorá v Irrlichte reprezentuje receiver ma jedinú funkciu onEvent, ktorá je volaná vždy keď nastane udalosť. Receiver použitý v hre vznikol dedením z receiveru enginu Irrlichtu, trieda je pomenovaná myEventReceiver. Vytvorený receiver uchováva inforácie otom, či bola

pohnutá myš, ak áno ktorým smerom a či bolo stlačené tlačítko myši. Navyše obsahuje dve polia typu boolean, ktoré zaznamenávajú súčasný a predchádzajúci stav všetkých kláves aby bolo možné rozoznať nábežnú a zostupnú hranu na klávesách.

4.3.3 Kamera

Keďže Irrlicht neponúka žiadnu šablónu pre kameru z pohľadu tretej osoby bolo nutné vytvoriť vlastnú implementáciu. Bola vytvorená trieda thirdPersonCamera, ktorá používa ako základ uzol pre kameru typu FPS z Irrlichtu (ICameraSceneNode). Kamere je nutné nadstaviť uzol scény ktorý má sledovať a požadovanú vzdialenosť od sledovaného objektu.

U kamery typu 3rd person je dôležité, aby následovala svoj cieľ plynule. Kamera by nemala prenášať na pozorovateľa jemné pohyby a otrasy sledovaného objektu, ktoré spôsobuje fyzikálna simulácia. Plynulosť kamery bola docielená tak, že nová poloha kamery sa po pohybe sledovaného objektu počíta s určitým stupňom voľnosti. V každom kroku sa na základe žiadanej a aktuálnej pozície vypočíta vektor posunutia a ten je predelený podľa toho, ako rýchlo má kamera sledovať cieľ.

```
// calculate new position of camera
// from rotation and positon of target
core::vector3df targetPos = target->getPosition();
targetPos.Z+=cos(targetRot*(2*3.14/360))*distToTarget;
targetPos.X+=sin(targetRot*(2*3.14/360))*distToTarget;
targetPos.Y+=2.5;          // camera is 2.5 units above target
// get vector from current to new position
core::vector3df deltaPos=targetPos-node->getPosition();
// get distance from current to new position
float distance= sqrt(deltaPos.X*deltaPos.X+deltaPos.Z*deltaPos.Z);
// divide distance to delay movement
distance= distance/250;
// calculate new position depending on distance
vector3df finalPos = deltaPos*distance+node->getPosition();
```

Obr. 22: Výpočet novej pozície kamery

Je tiež dôležité, aby kamera neprechádzala cez steny objektov v scéne. Vďaka raycastingu je možné zistiť kolízne body priamky v priestore. Vždy predtým ako sa má kamera umiestniť na novú pozíciu je do požadovanej pozície vyslaný lúč. Keď lúč koliduje s nejakým objektom, kamera bude umiestnená do bodu kolízie.

```
// ray starts from target
core::vector3df rayStart = target->getPosition();
// create line from target to calculated position
line3df ray = line3df(rayStart,finalPos);

// vector for storing intersection position
core::vector3df intersection;
// vector for storing colliding triangle
core::triangle3df tri;

// check for collision with raycasting
if (smgr->getSceneCollisionManager()->getCollisionPoint(
ray, selector, intersection, tri))
// if collision occures final position will be adapted
{
    // get desired change in position
    deltaPos = intersection - node->getPosition();
    // calculate new position depending on distance
    finalPos = node->getPosition() + deltaPos/50;
}
// set new position
this->node->setPosition(finalPos);
```

Obr. 23: Detekcia kolízií pomocou raycastingu a korekcia pozície kamery

4.3.4 Hrdina

Trieda hero predstavuje hlavného hrdinu hry a je potomkom triedy Human, teda človek. Graficky hrdinu reprezentuje animovaný uzol scény IAnimatedMeshSceneNode, fyzikálne reprezentuje hrdinu NewtonBody, ktoré má tvar elipsoidu. Elipsoid bol zvolený pretože dobre vystihuje tvar ľudského tela a zároveň umožňuje lahší pohyb po nerovnom teréne a schodoch na rozdiel od kvádru. Na to, aby sa elipsoid, ktorý predstavuje telo hrdinu nezačal kotúľať bolo nutné nadstaviť funkciu fyziky, obmedzenie typu UpVector. UpVector dovolí objektu zmeniť rotáci len okolo osi Y (zvislá os). Ako potomok hrdinu je do scény vložený ešte uzol, ktorý predstavuje tieň hrdinu. Tieň je vytvorený staticky pretože dynamické tieňe sú náročné na systémový čas. Trieda hero implementuje nasledujúce funkcie:

- move- slúži na pohyb, hrdina môže kráčať alebo bežať, dopredu a dozadu, podľa typu pohybu je spustená animácia a aplikovaná sila
- trackTo- funkcia, ktorá nasmeruje hrdinu na bod v priestore, výšková súradnica je ignorovaná, funkcia rotuje hrdinu len okolo osi Y

- strafeLeft- úkrok vľavo
- strafeRight- úkrok vpravo
- turnLeft- rotuje hrdinu doľava, parameter funkcie je rýchlosť rotácie
- turnRight- rotuje hrdinu doprava, parameter funkcie je rýchlosť rotácie
- stay- zastaví všetky animácie, po jej zavolaní stojí hrdina na mieste
- goToCar- volaním funkcie nastúpi hrdina do najbližšieho vozidla, ktoré získa zo scény, pohyb do vozidla prebieha tak, že hrdina sleduje waypointy, ktoré sú umiestnené v rohoch vozidla
- fire- po zavolaní funkcie hrdina vystrelí

Udalosti zo vstupných zariadení pre hrdinu spracováva trieda heroEventAdapter vo funkcii handleEvents. Na to, aby bolo možné elipsoidom, ktorý je fyzikálnou reprezentáciou hrdinu pohybovať slúžia callback funkcie. Funkcia addForceHuman aplikuje na elipsoid silu a krútiaci moment, funkcia setTransformHuman aktualizuje polohu a rotáciu grafického objektu.

4.3.5 Chodec

Trieda Pedestrian predstavuje chodca, je potomkom triedy Human. Grafická reprezentácia je animovaný uzol a fyzikálna elipsoid, rovnako ako u hrdinu. Aj u chodca je podobne ako u hrdinu použitý statický tieň.

Trieda Pedestrian je len jednoduchý model chodca, implementuje len dve funkcie, jednou je stay, ktorá objekt zastaví, druhá je walkAround. Jedným s parametrov pre vytvorenie chodca je pole waypointov, ktoré predstavujú trasu po ktorej má chodiť. Jednoduchým volaním funkcie walkAround postupne obchádza všetky waypointy v poli, keď príde na posledný začne od začiatku. Funkcia walkAround pozostáva z časti track to, ktorá upravuje rotáciu objektu a walk, ktorá mení polohu objektu. Sily sú opäť aplikované vo funkcii addForceHuman a poloha vo funkcii setTransformHuman.

4.3.6 Vozidlo

Trieda Vehicle slúži jako rodičovská trieda pre všetky vozidlá použité v hre. Funkcie, ktoré sú pre jednotlivé typy vozidiel špecifické sú definované ako virtual. Pomocou dedičnosti je tak možné spravovať všetky typy vozidiel rovnakým spôsobom. Trieda Vehicle implementuje nasledujúce funkcie:

- init- funkcia, ktorá inicializuje vozidlo, vytvorí grafický aj fyzikálny objekt
- run- vykonáva sa v každom cykle programu, tu sa aplikujú udalosti z klávesnice na vozidlo
- getWaypoints- funkcia, ktorej sa ako parameter predá poloha hrdinu, funkcia vráti pole waypointov, po ktorých sa hrdina dostane do vozidla
- addTire- pridá na vozidlo koleso
- freeze- zmrazí vozidlo, čím sa pozastaví fyzikálna simulácia, z vozidla sa stane len grafický objekt a nezaťažuje tak CPU
- unfreeze- funkcia, ktorú je možné použiť na rozmrazenie vozidla

Udalosti z klávesnice pre objekt vozidlo spracováva trieda vehicleEventAdapter. Všetky vozidlá používajú rovnaké callback funkcie na prácu s fyzikou. Na aktualizáciu pozície grafického vozidla slúži funkcia setTransformVehicle, na aplikovanie gravitačnej sily addGravityForce a na aktualizáciu polôh kolies setTorqueVehicle.

4.3.7 Auto

Auto implementuje trieda Car, ktorá je potomkom triedy Vehicle. Táto trieda je založená na simulácii vozidla fyzikálneho enginu Newton. V konštruktore auta sú nadstavené počiatočné parametre a zavolaná funkcia init. Vo funkcii init sa vytvorí animovaný uzol, ktorý reprezentuje auto graficky. Uzlu sú následne nadstavené textúry, materiály a počiatočná pozícia.

Keď je vytvorený grafický objekt, vytvorí sa newtonovský objekt. Pre jednoduchosť je vo fyzikálnom svete hry auto reprezentované ako kváder. Vlastnosti auta, ktoré sú špecifické pre rôzne typy áut ako napríklad poloha kolies, hmotnosť a rozmery sú uložené v súbore XML. Po tom ako je celé auto vytvorené sa zaregistrujú callback funkcie. Auto

využíva na aplikovanie gravitačnej sily funkciu addGravityForce, na aktualizovanie pozície grafického objektu slúži funkcia setTransformVehicle. Na aktualizáciu polôh kolies slúži callback funkcia setTorqueVehicle.

Automobil používa niekoľko materiálov, karoséria je tvorená materiálom EMT_REFLECTION_2_LAYER, ten používa dve textúry, textúru materiálu auta a textúru, na základe ktorej sa generujú fiktívne odrazy. Sklá automobilu používajú materiál EMT_TRANSPARENT_REFLECTION_2_LAYER, ktorý používa taktiež dve textúry, textúru na základe ktorej sa určí priehľadnosť a textúru na základe ktorej sa generujú fiktívne odrazy. Interiér auta, kolesá a svetlá používajú základný materiál EMT_SOLID s jednou textúrou.

```
// turn off lighting
chassisN->setMaterialFlag(EMF_LIGHTING,false);
// turn off back face culling, faces will be double-sided
chassisN->setMaterialFlag(EMF_BACK_FACE_CULLING,false);
// turn on anisotropic texture filtering
chassisN->setMaterialFlag(EMF_ANISOTROPIC_FILTER,true);

// lights material texture, type is default solid
chassisN->getMaterial(0).
setTexture(0,driver->getTexture( (path + "lights.png").c_str() )   );
// windows material textures
chassisN->getMaterial(3).
setTexture( 0, driver->getTexture("textures/cars/alphatest.jpg") );
chassisN->getMaterial(3).
setTexture( 1, driver->getTexture("textures/cars/reflect.jpg") );
// windows material type
chassisN->getMaterial(3).MaterialType =  EMT_TRANSPARENT_REFLECTION_2_LAYER;
// chassis material textures
chassisN->getMaterial(1).
setTexture(0,driver->getTexture(  (path + "golf2_orange.jpg").c_str()  ));
chassisN->getMaterial(1).
setTexture(1,driver->getTexture("textures/cars/reflect2.jpg"));
// chassis material type
chassisN->getMaterial(1).MaterialType = EMT_REFLECTION_2_LAYER;
// interier material texture, type is default solid
chassisN->getMaterial(2).
setTexture( 0, driver->getTexture("textures/cars/black.jpg") );
```

Obr. 24: Nadstavenie materiálu automobilu

4.3.8 Motocykel

Ďalším potomkom triedy vehicle je trieda Bike, ktorá predstavuje motocykel. Trieda Bike je veľmi podobná triede Car. Simulácia motocykla prebieha rovnako ako simulácia auta, rozdiel je v počte a umiestnení kolies. Aby sa simulácia určená pre automobil správala

podobne jako reálny motocykel stačí umiestniť ťažisko objektu pod úroveň kolies, teda mimo objekt. Vďaka ťažisku pod objektom sa motocykel neprevráti na bok a v zákrutách sa nakláňa proti odstredivej sile.

Vytvorenie motocykla prebieha v rovnakých krokoch ako u automobilu, v konštruktore sa nadstavia počiatočné parametre a následne vo funkcii init sa vytvorí grafický objekt, materiály, textúry a newtonovský objekt.

4.3.9 Streľba

Vytvorená aplikácia tiež ukazuje jeden z možných spôsobov vytvorenia streľby. Pri realizácii streľby s použitím externého fyzikálneho enginu je opäť nutné vytvoriť grafickú aj fyzikálnu reprezentáciu letiacej guľky. Fyzikálne simulovať let veľmi rýchleho a malého objektu sa nedoporučuje pretože sa môže stať, že engine nezaznamená kolíziu. V tomto prípade je vhodné použiť opäť raycasting.

Test kolízie prebieha tak, že po stlačení tlačítka na myši sú pomocou funkcie Irrlichtu zistené súradnice priamky, ktorá vychádza z obrazovky z miesta, kde je práve kurzor. Následne je pomocou raycastingu enginu Newton vyslaný lúč so súradnicami tejto priamky. Keď Newton zistí kolíziu s nejakým telesom vykoná sa callback funkcia, ktorá vyhodnotí kolíziu s grafickým objektom. Kolízia s grafickým objektom sa zisťuje pomocou raycastingu enginu Irrlicht, lúč s rovnakými súradnicami ako má priamka, ktorá vychádza z kurzoru je vyslaný aj na grafický objekt. Vyslať lúč aj na grafický objekt je nutné preto, že fyzikálna a grafická reprezentácia objektu sa môžu líšiť. Ak lúč koliduje aj s grafickým objektom, do miesta kolízie sa umiestni tzv. decal objekt, ktorý napodobňuje stopu po guľke. Následne je v mieste, kolízie na objekt aplikovaná sila, ktorá simuluje silu, ktorou by pri náraze na objekt pôsobila guľka.

Po tom, ako bol zistený objekt, ktorý leží v dráhe strely a bola vytvorená stopa po guľke a sila, ktorou pôsobí guľka na objekt stačí vytvoriť grafickú reprezentáciu letiacej guľky. Guľka bola vytvorená ako primitívum pomocou scene managera Irrlichtu. Aby sa guľka pohybovala bol pre ňu vytvorený FlyStraightAnimator so štartom v zbrani hrdinu a cieľom na konci priamky, podľa ktorej sa simuloval let guľky. Aby guľka po určitom čase po výstrele zanikla bol použitý DeleteAnimator.

Výhodou takto vytvorenej streľby je, že je vďaka raycastingu veľmi rýchla a jednoduchá, nevýhodou je, že neberie do úvahy pohyb objektov. Môže sa preto stať, že objekt medzi tým ako guľka k nemu priletí zmení pozíciu, ale kolízia bola napriek tomu vyhodnotená. Tento jav je možné odstrániť tým, že na pohybujúci objekt sa vyšle pomocou raycastingu ešte jeden lúč v čase kolízie ktorý zistí, či sa tam objekt stále nachádza alebo prípadne zistí ďalšiu kolíziu.

ZÁVER

V teoretickej časti bakalárskej práce bola vypracovaná literárna rešerš o 3D grafickom engine Irrlicht. Teoretická časť bola rozdelená na tri menšie celky.

Prvá časť popisuje základné rysy enginu, jeho vlastnosti, schopnosti a štruktúru rozhrania. Nasleduje história enginu, postupný popis vývojových verzií a zmien, ktoré nové verzie priniesli.

Druhá časť popisuje rozhranie Irrlichtu pre management a prácu so scénou. Pomocou rozhrania irr::scene je možné pre aplikáciu tvoriť a upravovať komplexnú scénu. Je možné načítať zo súboru modely v rôznych formátoch, spúšťať animácie na animovaných modeloch alebo animovať pohyb objektov. Ďalej je možné vytvoriť kamery, emitory častíc, použiť kolízny systém. Pre tvorbu scény je tiež možné použiť objekty enginu optimalizované na vykreslovanie veľkých a zložitých modelov.

Tretia časť popisuje rozhranie pre prácu s grafickými technológiami. Toto rozhranie poskytuje nástroje pre tvorbu vzhľadu aplikácie. Pomocou rozhrania irr::video je možné zobraziť aplikáciu pomocou hociktorého z dostupných grafických API, teda pomocou Direct3D, OpenGL a pomocou dvoch softwareových renderovačov. Ďalej je možné z rôznych formátov načítať textúry, vykreslovať základné geometrické útvary a nadstavovať objektom rôzne materiály.

Praktická časť bakalárskej práce ukazuje možnosti použitia popísaných funkcií enginu Irrlicht pri tvorbe 3D interaktívnej aplikácie, počítačovej hry. V tejto časti je popísaná samotná architektúra aplikácie, materiálové nadstavenia, nadstavenia textúr, využitie kolízneho systému, spôsoby animovania objektov. Okrem použitia vstavaných funkcií praktická časť tiež popisuje základné techniky pri tvorbe modelov pomocou programu Blender a demonštruje nové možnosti, ktoré sa dajú získať použitím externého fyzikálneho enginu. Aplikácia používa na smulovanie fyziky objektov engine Newton.

Ako súčasť praktickej časti práce bola vytvorená k enginu Irrlicht užívateľská príručka. Príručka popisuje základné funkcie Irrlichtu a funkcie pre management scény, správu grafických technológií, funkcie pre tvorbu GUI a prácu so súbormi. Časť príručky popisujúcu užívateľské rozhranie a prácu so súbormi vypracoval študent, ktorý riešil bakalársku prácu na túto tému.

ZÁVER V ANGLIČTINE

In the theoretical part of the bachelor thesis, a literature retrieval about 3D graphics engine Irrlicht was created. The theoretical part is divided into three parts.

The first part describes basic features of the engine and structure of programming interface. Then history of the engine, development and changes were described.

The second part deals with the Irrlicht‘s scene management interface. Using the interface irr::scene complex scene can be created and managed. Models can be loaded in many different formats, model animations and movement animations can be played. Also cameras, particle emittors and collision system can be used. There are also several objects optimized to handle large and complicated meshes.

The third part describes Irrlicht’s graphics technology interface. This interface offers tools for creation of application design. Using interface irr::video, the application can be rendered with Direct3D, OpenGL or with one of two built in software renderers. Textures can be loaded in many formats, basic graphic primitives can be drawn and materials can be created.

The practical part of the thesis demonstrates application of described functions of Irrlicht engine in the 3D interactive application creation, computer game. In this part the created application is described. Material settings, texture settings, collision system usage and animations are described. In the addition to the Irrlicht functions, some basic modeling techniques and the usage of external physics engine are described. For physics simulations, the Newton physics engine is used.

A simple user manual about creation of 3D applications using the Irrlicht engine was created. The manual describes the functions of Irrlicht for scene management, graphic technology management, tools for creating GUI and file operations. The part of manual, which describes user interface creation and file operations was written by a student, who created bachelor thesis on this topic.

ZOZNAM POUŽITEJ LITERATÚRY

[1] GEBHARDT, Nikolaus. *Irrlicht Engine* [online]. 2003 [cit. 2008-03-10]. Dostupný z WWW: <http://irrlicht.sourceforge.net/>

[2] GEBHARDT, Nikolaus. *Irrlicht Engine 1.4 API documentation* [online]. 2007 [cit. 2008-03-10]. Dostupný z WWW: <http://irrlicht.sourceforge.net/docu/index.html>

[3] *Irrlicht Engine Forum* [online]. Dostupný z WWW: <http://irrlicht.sourceforge.net/phpBB2/index.php>

[4] GEBHARDT, Nikolaus. *Irrlicht tutorials* [online]. 2003 [cit. 2008-03-12]. Dostupný z WWW: http://irrlicht.sourceforge.net/tutorials.html

[5] *Newton Game Dynamics : Physics forum* [online]. Dostupný z WWW: http://www.newtondynamics.com/forum/

[6] JEREZ, Julio. *Newton Documentation*, c2005. 91 s

[7] *Shader : from Wikipedia, the free encyclopedia* [online]. 2007 [cit. 2008-03-19]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Shader#Types_of_shader>

[8] *Fórum* [online]. Dostupné z WWW: http://www.blender3d.cz/forum

[9] PARISI, Diego. *Blitz3D Toolset : for Blender* [online]. Dostupný z WWW: <http://www.gandaldf.com/>

[10] *Blender* [online]. Dostupný z WWW: http://www.blender.org/

[11] *IrrEdit* [online]. Dostupný z WWW: http://www.ambiera.com/irredit/

[12] GEBHARDT, Nikolaus. *Irrlicht : Features* [online]. 2003 [cit. 2008-03-08]. Dostupný z WWW: http://irrlicht.sourceforge.net/features.html

[13] *Wikipedia* [online], 2001. Dostupný z WWW: http://wikipedia.org/

[14] *GIMPshop* [online]. [2007] [cit. 2008-04-20]. Dostupný z WWW: <http://www.gimpshop.com/>

ZOZNAM POUŽITÝCH SYMBOLOV A SKRATIEK

API	Rozhranie pre programovanie aplikácií.
ARB	Nízkoúrovňový jazyk na programovanie shaderov pomocou OpenGL
FPS	First Person Shooter, zobrazenie pohľadom prvej osoby.
fps	Frames per second, počet vykreslených obrázkov za sekundu, udáva rýchlosť akou beží 3D aplikácia.
Fyzikálny engine	Rozhranie pre simuláciu fyzikálneho správania objektov.
GLSL	OpenGL Shading Language, multiplatformný jazyk určený na tvorbu shaderov.
Grafické primitíva	First Person Shooter, zobrazenie pohľadom prvej osoby.
Grafický engine	Rozhranie pre prácu s grafikou.
GUI	Grafické užívateľské rozhranie.
HLSL	High Level Shading Language, jazyk určený na tvorbu shaderov vyvinutý firmou Microsoft.
Lightmap	Stromová dátová štruktúra, ktorá sa využíva na optimalizovanie vykresľovania 3D priestoru rekurzívnym rozdelením na oktanty.
OctTree	Stromová dátová štruktúra, ktorá sa využíva na optimalizovanie vykresľovania 3D priestoru, rekurzívne rozdeľuje priestor na oktanty.
RPG	Role-Playing Game, hra na hrdinov.
Shader	Súbor inštrukcií, ktoré slúžia na tvorbu efektov pri renderovaní, vytvárajú sa programovaním GPU.
UV koordináty	UV súradnice, sú to 2D súradnice vertexov modelu na textúre. Definujú akým spôsobom sa má textúra namapovať na objekt.
Vertex	Bod v priestore, vrchol objektu (geom.).

ZOZNAM OBRÁZKOV

ZOZNAM PRÍLOH

PRÍLOHA P I: SCREENSHOTY Z VYTVORENEJ APLIKÁCIE


0
20
40
60
80
100
120
140
160
mph
km/h

PRÍLOHA P II: ADRESÁROVÁ ŠTRUKTÚRA PRILOŽENÉHO DVD

- /praca

Obsahuje text bakalárskej práce vo formáte DOC a PDF.

- /projekt

Obsahuje vytvorenú aplikáciu, aplikácia sa spúšťa súborom game.exe

- /prirucka

Obsahuje vytvorenú príručku vo formáte PDF