UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Domen Grabec

UPORABA RAČUNALNIŠKEGA VIDA ZA IZBOLJŠANJE PRISOTNOSTI V RAČUNALNIŠKIH IGRAH

DIPLOMSKO DELO NA UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Saša Divjak

Ljubljana, 2009

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01593/2009
Datum: 15.10.2009

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

Kandidat: **DOMEN GRABEC**

Naslov: **UPORABA RAČUNALNIŠKEGA VIDA ZA IZBOLJŠANJE OBČUTKA PRISOTNOSTI V RAČUNALNIŠKIH IGRAH**

USAGE OF COMPUTER VISION TO INCREASE THE FEELING OF PRESENCE IN COMPUTER GAMES

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Podajte pregled sistemov, ki uporabljajo računalniški vid za izboljšanje občutka uporabnikove prisotnosti pri računalniških igrah. Analizirajte različne možnosti spreminjanja točke gledanja v programski opremi, s katero so podprte računalniške igre. Predstavite knjižnico DirectX in postopek, po katerem ta izrisuje 3D objekte na zaslon, Razvijte in predstavite aplikacijio , ki poskuša ustvariti ustvariti iluzijo, da igralec gleda v 3D svet in ga dojema na isti način, kot resnični svet. Zato naj aplikacija s pomočjo spletne kamere sledi premikom glave uporabnika, nato pa te premike prevede v spremembo kamere znotraj igre.

Mentor:

prof. dr. Saša Divjak

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Franc Solina

Kazalo vsebine

1 UVOD

Računalniške igre postavijo uporabnika v alternativni svet, ki je definiran z določenimi pravili, omejitvami in cilji. Primer, kjer lahko vidimo na zaslonu predstavljen igralni svet, z izrisom, v dveh dimenzijah, v dveh barvah ter s tremi premikajočimi objekti, je ena prvih računalniških iger – Pong. Leta 1972, ko je igra izšla, so bile računske sposobnosti računalnikov in igralnih konzol grafično relativno skromne - v primerjavi z današnjimi igralnimi postajami, ki so sposobne izrisovati skoraj foto-realistično grafiko v realnem času. Sklepamo lahko, da kompleksnejša in bolj prepričljiva grafika med drugim ustvari boljšo iluzijo igralnega sveta in s tem privlači več uporabnikov. Skladno s to ugotovitvijo se industrija računalniških iger od svojega nastanka z razvojem tehnologije razvija in nadgrajuje, kar se posledično vidi v grafično bolj zahtevnih igrah.

Glavna dejavnika, ki sta dolgo časa vplivala na kakovost izkušnje pri igranju iger, sta igralnost in grafična podoba igre. Če pogledamo nazaj, lahko vidimo številne poskuse, katerih cilj je bil izboljšati uporabniško izkušnjo, bodisi s prikazovanjem stereoskopske slike (Virtual boy [1]), z uporabo na premik občutljive rokavice (Power Glove[2]), z uporabo na premik in pospešek občutljivega kontrolerja (Wii remote[3]) itd., od katerih so bili nekateri uspešni, drugi pa popoln polom.

Pri projektih, ki želijo izboljšati uporabniško izkušnjo, je v zadnjem desetletju vse večji poudarek na računalniškem vidu. To tezo potrdi podrobnejši pregled teh projektov, predstavljen v prvem poglavju. Večini predstavljenim projektom sta skupni dve pomanjkljivosti, in sicer težka dostopnost in slaba podpora pri računalniških igrah. Projekti so težje dostopni, ker smo poleg programske opreme primorani kupiti tudi strojno opremo, kar se cenovno odrazi na izdelku. Drugi problem je v tem, da avtorji teh projektov to omogočijo samo v (zelo redkih) igrah, ki podpirajo premike kamere ali ponudijo orodje razvijalcem iger, ki naj podporo direktno vključijo v svojo računalniško igro, kar je danes zelo težko doseči.

Zgoraj opisani glavni pomanjkljivosti sta obenem tudi izhodiščni točki za izbiro ciljev tega dela. Diplomsko delo je rezultat raziskave, ki želi ugotoviti, ali je mogoče izboljšati uporabniško izkušnjo zgolj z uporabo spletne kamere, in to prav v igri, ki te funkcionalnosti ne predvidi. Od aplikacije se zahteva, da je sposobna slediti premikom glave uporabnika in te premike prevesti v premike kamere znotraj igre.

1.1 Zgradba naloge

Uvodno poglavje vključuje predstavitev širše problematike področja izboljšave uporabnišle izkušnje pri igranju iger. V drugem poglavju sledi pregled bolj ali manj uveljavljenih

sistemov, ki rešujejo problematiko izboljšanja uporabniške izkušnje. Tretje poglavje predstavlja način delovanja izvršljivih datotek in pristope k spreminjanju njihovega privzetega obnašanja. V četrtem poglavju je predstavljena knjižnica DirectX in postopek po katerem izrisuje 3D objekte na zaslon, peto in hkrati zadnje poglavje pa je posvečeno opisu aplikacije, ki ustreza postavljenim ciljem iz uvodnega poglavja. Delo se zaključi s sklepom in opisom težav, ki bolj ali manj pestijo opisano aplikacijo.

2 PREGLED OBSTOJEČIH PROJEKTOV, KI Z RAČUNALNIŠKIM VIDOM IZBOLJŠAJO IZKUŠNJO PRI RAČUNALNIŠKIH IGRAH

2.1 Project Natal

Project Natal [4] [5] je izdelek podjetja Microsoft, ki poskuša ustvariti igralsko izkušnjo brez potrebe po igralni palici ali kontrolerju. Ta nadgradnja je namenjena igralni konzoli Xbox 360, ki je prav tako produkt podjetja Microsoft. Ideja projekta je omogočiti uporabniku popolno interakcijo z igralno konzolo, uporabo telesnih kretenj, govornih ukazov in objektov, ki jih prepozna Project Natal. Microsoft se je odločil za tako potezo, ker želi povečati svoj krog uporabnikov igralne konzole Xbox 360 in jo približati tudi bolj nezahtevnim igralcem.

2.1.1 Tehnične podrobnosti:

Project Natal uporablja senzor v obliki podolgovate tablice velikosti 23 cm. Senzor postavimo nad ali pod zaslon, na katerega smo priključili igralno konzolo. Naprava je sestavljena iz kamere, sposobne zajemati video posnetek, senzorja globine in polja mikrofonov. Vsebuje tudi CPE, na katerem se izvaja namenska programska oprema, ki omogoča 3D zajem človeškega skeleta, glasovno prepoznavanje in prepoznavanje obraznih izrazov. Polje mikrofonov omogoča, da senzor lahko razbere izvor zvoka in s tem tudi odstrani okoliški šum.

Senzor globine je sestavljen iz infrardečega projektorja, ki dopolnjuje črno-beli CMOS senzor. Projektor omogoča, da CMOS senzor zajema kakovostno sliko tudi ob slabih svetlobnih pogojih. Senzor je po globini nastavljiv, s programsko opremo pa se tudi sam nastavi glede na uporabnikovo telesno zgradbo in prisotnost okoliških predmetov – pohištva.

Slika 1: Senzor projekta Natal

2.2 Eye Toy

Eye toy[6] je digitalna kamera, podobna spletnim kameram, namenjena igralni konzoli Playstation 2. Kamera omogoča uporabniško interakcijo z igralno konzolo s pomočjo računalniškega vida in razpoznavanjem kretenj. Naprava je bila javnosti predstavljena leta 2002 in se lahko pohvali s kopico naslovov iger, ki podpirajo njeno delovanje.

2.2.1 Tehnične podrobnosti

Naprava je po svoji sestavi skoraj identična spletnim kameram s CMOS čipom za zajem video posnetkov. Posebnost je samo rdeča LED dioda, ki se prižge, kadar je prostor preslabo osvetljen in takrat kamera nima zadostnih pogojev za pravilno delovanje. Za priklop na igralno konzolo uporablja USB priključek in USB 1.1 protokol. Za izostritev slike poskrbi vrtljiv objektiv, ki ga je treba ročno nastaviti.

Slika 2: Senzor projekta Eye Toy

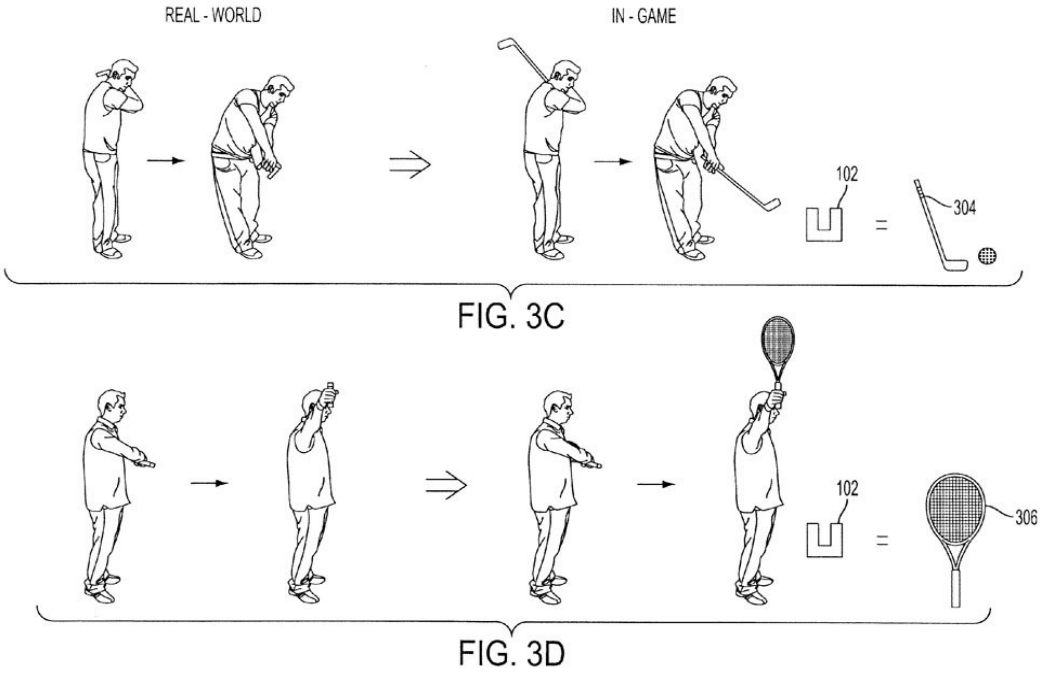
2.3 Sony Motion Sensor Controller

Sony je v navezi z napravo Eye Toy na E3 konferenci predstavil programsko opremo, ki je sposobna naprednega sledenja in razpoznavanja objektov[7]. Namen projekta je sledenje objektu, s katerim uporabnik manipulira v realnem svetu, in njegovo obnašanje pretvoriti v akcije znotraj igre.

Sony je delovanje naprave ponazoril s sledečim primerom. Igralec želi igrati igro, v kateri je potrebno mečevanje. Namesto da igralec uporabi kontroler za interakcijo z igralno konzolo, lahko Motion Sensor Controllerju poda nek objekt iz svoje okolice (kuhinjski kozarec, knjigo, steklenico itd.) in naprava obnašanje tega predmeta prevede v obnašanje meča v igrici. To pomeni, če igralec zamahne z steklenico v roki, bo lik v igri zamahnil z mečem.

2.3.1 Tehnične podrobnosti

Tehnične podrobnosti Motion Sensor Controller-ja so enake kot pri Eye Toyju. Razlika je v programski opremi, ki teče na igralni konzoli. Zaradi zahtevnosti algoritmov računalniškega vida bo Motion Sensor Controller podprt samo na Playstation konzolah tretje generacije.



Slika 3: Delovanje Sony motion sensor controller-ja

2.4 Head tracking by Johnny Chung Lee

Head tracking [8] je raziskovalni projekt, ki omogoča učinkovito sledenje uporabnikovi glavi v prostoru. 3D prostor na zaslonu se izrisuje glede na relativno pozicijo glave od zaslona. Ko uporabnik premakne svojo glavo, se njegov premik upošteva kot premik točke kamere v igri. Z drugimi besedami, če uporabnik v realnem svetu premakne glavo v-stran, se bo grafika v igri izrisala, kot bi lik znotraj igre premaknil glavo v-stran.

2.4.1 Tehnične podrobnosti:

Johnny Lee je za sledenje uporabil napravo Wii Remote, ki vsebuje infrardečo kamero visoke ločljivosti. Naprava je sposobna v realnem času zaznati več izvorov infrardeče svetlobe in jim

slediti. Za sledenje glavi je Johnny Lee uporabil očala, ki imajo na okviru dve infrardeči LED diodi, pozicionirani 20 cm narazen. Wii Remote je postavljen pod zaslon in sledi LED diodama, pozicioniranima na uporabnikovih očalih. Nato podatke v realnem času pošilja preko Bluetooth protokola računalniškemu programu. Program iz dobljenih podatkov lahko izračuna relativni položaj glave glede na Wii Remote, ki se nahaja pod zaslonom.

Slika 4: Delovanje Johhny Lee-jevega headtracing projekta

2.5 TrackIR

TrackIR [9] [10] je naprava, namenjena sledenju premikom glave. Sposobna je slediti premikom v vse tri dimenzije in tudi zaznati rotacijo glave okrog vseh treh osi. Položaj in smer glave opazuje samo za ta namen zgrajena infrardeča video kamera. Video kamera deluje s pomočjo treh točk, ki odbijajo infrardečo svetlobo in so nameščene na ogrodju, ki ga uporabnik namesti na pokrivalo. TrackIR na trgu obstaja že od leta 2001 in se je uveljavil predvsem v igrah, ki simulirajo letenje, kot npr. Microsoft Flight Simulator. Kasneje je svoj nabor iger, ki jih podpira, razširil na dirkaške igre in prvoosebne streljanke.

2.5.1 Tehnične podrobnosti

TrackIR vsebuje infrardečo kamero, ki je sposobna zajemati video v eni barvi. Programska logika v čipovju kamere obdela sive odtenke v video posnetku na način, da zmanjša velikost video posnetka. Posledično se zmanjša prenos podatkov med kamero in računalnikom. Programska oprema, nameščena na računalniku, obdela podatke in iz njih razbere položaj in rotacijo glave.

Slika 5: Izgled namestitve TrackIR naprave

3 PRISTOPI K SPREMINJANJU TOČKE GLEDANJA

Tehnološki cilj, ki ga želimo doseči, je dokaj optimističen, in sicer potrebujemo komponento, ki omogoča spreminjanje točke gledanja pri čim večjem številu računalniških iger, po možnosti pri vseh. Po dolgotrajnem raziskovanju vidimo, da je možnih več pristopov, vendar z nobenim ni mogoče v popolnosti doseči zastavljenega cilja. Za razumevanje različnih načinov pristopa je potrebno osnovno poznavanje pojmov (programska) knjižnica in

(računalniški) proces.

3.1 (Programska) knjižnica

Programska knjižnica [12] je zbirka podprogramov ali funkcij za pomoč pri razvoju programske opreme. Knjižnice vsebujejo kodo in podatke, ki jih je mogoče uporabiti v neodvisnih programih. Nekatere izvršilne datoteke so obenem samostojni programi in knjižnice. Izvršilne datoteke in knjižnice se med sabo povezujejo s procesom, imenovanim povezovanje, ki ga opravlja povezovalnik (ang. *linker*). Glede na način povezovanja ločimo statično in dinamično povezovanje.

3.1.1 Dinamično povezovanje

Pri dinamičnem povezovanju se podprogrami knjižnice naložijo v program ob njegovem izvajanju. V datotečni strukturi na disku izvršilna datoteka programa in knjižnica predstavljata različni datoteki. Povezovalnik ob prevajanju programa zapiše, le katere knjižnice in imena njihovih podprogramov potrebuje. Knjižnice pa se naložijo v naslovni prostor programa ob njegovem zagonu ali med njegovim izvajanjem. Za to povezovanje poskrbi del operacijskega sistema, imenovan **loader**.

3.1.2 Statično povezovanje

Pri statičnem povezovanju povezovalnik poveže ustrezne programske knjižnice in objektne datoteke, nato pa jih prevajalnik prevede v izvršilno datoteko. Za razliko od dinamičnega povezovanja se vsi podprogrami nahajajo samo v izvršilni datoteki programa.

3.1.3 Sklep

Če povzamemo, da preučevanje programske kode, ki se je nahajala v statičnih knjižnicah poteka s pregledovanjem strojne kode. Le-ta je namenjena za izvajanje na CPE in posledično je preučevanje te kode zamudno in naporno. O delovanju programske kode, ki se je nahajala v dinamičnih knjižnicah, pa lahko izvemo dosti več. Pri vsaki funkciji (podprogramu) v dinamični knjižnici lahko vidimo njeno ime, s katerimi parametri jo pokličemo in vrednost, ki

jo vrača. Nemalokrat pa obstaja na internetu bodisi uradna bodisi neuradna dokumentacija, ki opisuje uporabo in delovanje teh funkcij.

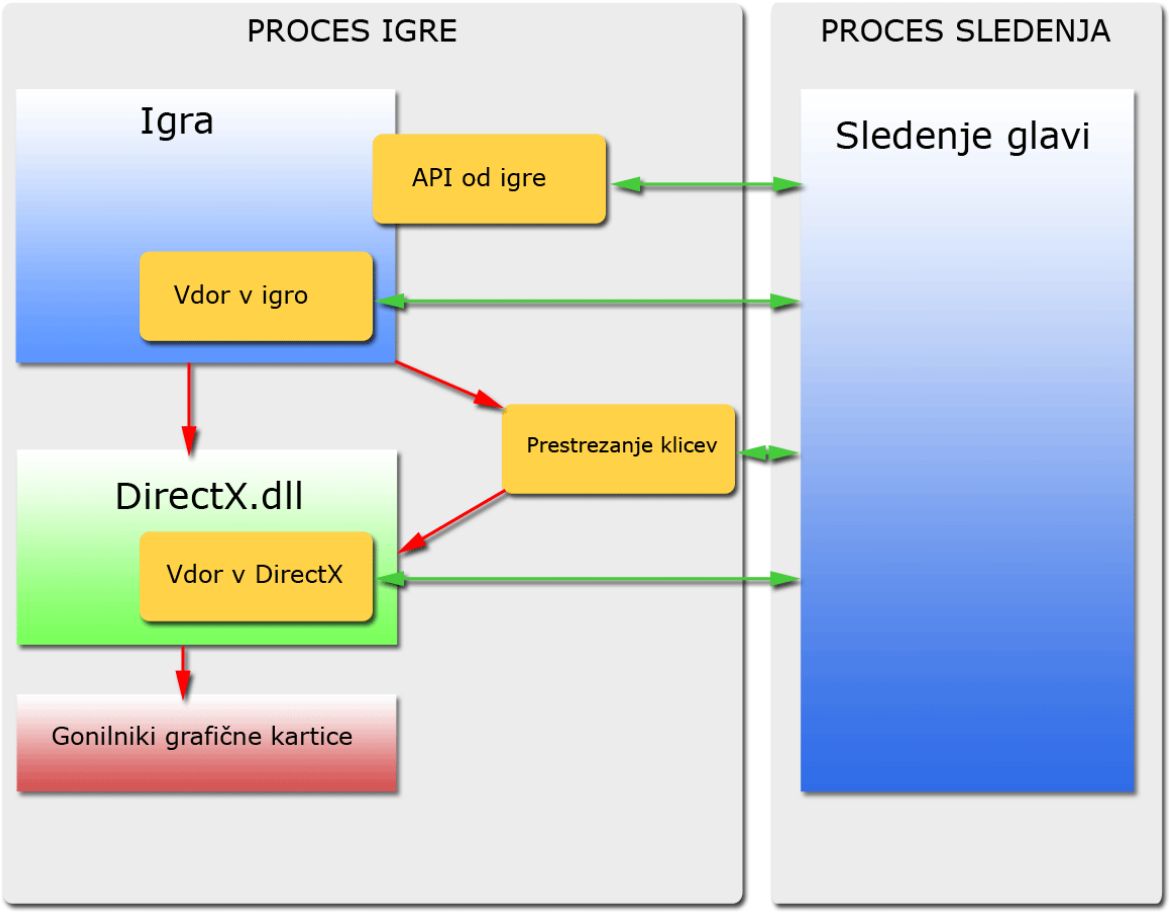
3.2 (Računalniški) proces

Ob ustvarjanju procesa [11] sistem le-temu dodeli naslovni prostor. To je območje v pomnilniku, kjer je proces naložen in kjer ima pravice branja in zapisovanja. Proces v izvajanju nima dostopa do naslovnih prostorov drugih procesov, vendar ti med sabo lahko komunicirajo s pomočjo medprocesne komunikacije (ang. *IPC*).

Če želimo v igri spreminjati točko gledanja, moramo poskrbeti, da lahko spreminjamo določene podatke v pomnilniku, ki so bili dodeljeni igri. Podatke lahko spreminjamo samo pod pogojem, da se ta koda izvaja v istem procesu kot igra.

3.3 Pregled različnih načinov pristopa

V spodnji shemi [13] so povzeti različni pristopi glede na izbrano vstopno točko.



Slika 6: Prikaz različnih načinov implementacije prestrezanja klicev

3.3.1 API od igre

Določene igre, kot npr. Microsoft Flight Simulator X, ponujajo API (programski vmesnik), ki nam omogoča spreminjanje točke gledanja. Poskrbeti moramo samo, da na API pošiljamo pravilne in točne podatke o položaju glave.

3.3.1.1 Prednosti

- Implementacija funkcionalnosti v igro je zelo enostavna.
- Edini pristop, kjer ni potrebno skrbeti, v katerem procesu se izvaja programska koda, ki spreminja točko gledanja.

- Ker je API podprt s strani razvijalcev igre, premiki glave ne bodo vplivali na hitrost in stabilnost igre.

3.3.1.2 Slabosti

- Premike glave lahko implementiramo samo v igre, ki imajo temu namenjen API in teh ni prav veliko. Največ jih izvira iz vrst dirkanja z avtomobili in simulacij letenja.
- Vsaki igri, ki jo želimo podpreti, se moramo posebej posvetiti in določiti na kakšen način aplikacija komunicira z igrinim API-jem.

3.3.2 Vdor v DirectX

DirectX je grafična knjižnica podjetja Microsoft. Njen namen je olajšati in skrajšati razvojni cikel izdelave računalniških iger. Večina (če ne skoraj vse) iger na trgu danes uporablja DirectX knjižnico za izris grafike na osebnem računalniku. Na žalost ta knjižnica ni odprtokodni izdelek in s tem je vdor vanjo zelo otežen.

3.3.2.1 Prednosti

- Implementacija premikov glave v knjižnico bi delovala za veliko večino iger.

3.3.2.2 Slabosti

- Ker izvorna koda od knjižnice ni na voljo, podrobnosti o njenem delovanju ugotovimo samo s preučevanjem strojne kode. Vdor je zato otežen in bi zanj potrebovali ogromno časa in truda.
- Treba je vdreti v vsako novo verzijo DirectX knjižnice, ki izide.
- Paziti moramo, da ne kršimo pogojev uporabe.

3.3.3 Vdor v igro

Strojno kodo igre je mogoče preučiti s in jo spremeniti tako, da igra podpira premike kamere.

3.3.3.1 Prednosti

- To je mogoče narediti za katero koli igro.

3.3.3.2 Slabosti

- Podobno kot vdor v DirectX, bi ta podvig zahteval veliko časa, truda in znanja.
- Vdreti bi bilo potrebno v vsako igro posebej.
- Paziti moramo, da ne kršimo pogojev uporabe.

3.3.4 Prestrezanje klicev

Lahko izkoristimo dejstvo, da izvršilna datoteka igre dinamično naloži DirectX.dll knjižnico. Obstajajo metode, ki nam omogočajo, da te klice ob pravilni uporabi prestrežemo, ne da bi glavni program vedel za to. Pri prestreženih klicih lahko vidimo vrednosti njihovih argumentov in jih poljubno spreminjamo.

3.3.4.1 Prednosti

1. Implementacija premikov glave s prestrezanjem bi delovala za vse igre, ki uporabljajo knjižnico, katere klice prestrezamo.
- Srednje težavna implementacija.

3.3.4.2 Slabosti

- S to metodo ne moremo podpreti iger, ki kodo za izris 3D grafike povežejo z uporabo statičnih knjižnic.

3.4 Analiza različnih pristopov

Sedaj imamo na voljo štiri različne načine pristopa in za enega se moramo odločiti. Pri tem najpomembnejšo vlogo igra faktor, koliko iger lahko podpremo z (enkratno) implementacijo spreminjanja točke gledanja.

Upoštevajoč to ugotovitev, lahko takoj izločimo pristop “Vdor v igro”, saj zahteva preveč vloženega časa in na koncu podpremo delovanje samo v eni igri. Pristop “Vdor v DirectX” je dosti bolj zanimiv, saj bi z eno uspešno implementacijo podprli veliko število iger. Vendar pa bi bilo verjetno pregledovanje več deset tisoč vrstic strojne kode večmesečni projekt, kar tudi presega okvire tega diplomskega dela.

Sedaj nam preostane še prestrezanje klicev in uporaba API-ja igre. Lažji od teh pristopov je vsekakor uporaba API-ja, vendar to lahko naredimo samo pri igrah, ki to podpirajo. V primeru prestrezanja klicev pa bomo s tem pristopom podprli več iger, vendar bo sama implementacija znatno bolj težavna.

Po tehtnem premisleku sem se odločil za težji pristop – prestrezanje klicev. Z njim bo lažje doseči cilj podporo večim igram, po drugi strani pa pristop predstavlja tudi večji izziv.

3.5 Različne implementacije prestrezanja

Običajno se prestrezanje uporablja, ko je program že v izvajanju, vendar pa je mogoče prestrezanje vpeljati, še preden se ta zažene.

3.5.1 Prestrezanje pred zagonom

Prestrezanje klicev izvršilni datoteki ali knjižnici lahko dosežemo z uporabo postopka vzvratnega inženirstva. To se najbolj pogosto uporablja za prestrezanje klicev funkcij, s tem da klice opazujemo ali jih nadomestimo s svojimi.

Z uporabo razgrajevalnika (ang. *dissasembler*) lahko vidimo, kje se v določenem modulu (knjižnici) nahaja vstopna točka določene funkcije. To vstopno točko je mogoče spremeniti in s tem dosežemo, da se dinamično naloži kak drug modul in pokliče določena funkcija tega modula.

Prestrezanje lahko dosežemo tudi na drug način. Izvršilni datoteki lahko spreminjamo tabelo vnosov (ang. *import table*). S spremembami lahko dosežemo nalaganje dodatnih dinamičnih knjižnic, lahko pa tudi spremenimo, kateri del kode se izvede, ko program pokliče določeno funkcijo.

Tretji pristop za prestrezanje klicev se imenuje “proxy dll”. To implementiramo tako, da ustvarimo svojo verzijo knjižnice, ki izgleda identična knjižnici, ki jo potrebuje izvršilna datoteka igre. Poskrbeti moramo, da naša verzija knjižnice vsebuje vse funkcije, ki jih vsebuje prvotna knjižnica.

3.5.2 Prestrezanje med izvajanjem

Operacijski sistem in programska oprema lahko dopuščata, da je mogoče prestrezati klice med izvajanjem programa. Vendar je potrebno, da ima proces, ki želi izvajati prestrezanje, dovolj pravic za to početje. V grobem so ti pristopi zelo podobni pristopom prestrezanja pred zagonom, razlika je v tem, da spreminjajo klice in strukture v sistemskem spominu programa (igre), ko je ta že v izvajanju.

Za naš primer so ustrezni vsi načini prestrezanja klicev in ni očitnih prednosti ali slabosti. Za eno se je treba odločiti, zato bomo implementirali metodo "proxy dll". Preden to metodo uporabimo, moramo vedeti, klice katere funkcije je treba prestreči, in za dosego tega je potrebno malo boljše poznavanje DirectX knjižnice.

4 DIRECTX KNJIŽNICA

Microsoftova DirectX [14] knjižnica je v svojem bistvu zbirka programskih vmesnikov, namenjenih ukvarjanju z multimedijo. Osredotoča se zlasti na izdelavo računalniških iger in obdelavo video vsebin na Microsoftovih platformah. Imena programskih vmesnikov se vsa začnejo z "Direct", npr.: Direct3D, DirectDraw, DirectMusic, DirectPlay, DirectSound itd.

Slika 7: DirectX logo

4.1 Zbirka DirectX programskih vmesnikov

- **DirectX Graphics** je sestavljen iz naslednjih programskih vmesnikov:

    - DirectDraw: je namenjen izrisu 2D grafike. Ta vmesnik ni več priporočeno uporabljati, saj ga je nadomestil Direct2D vmesnik. V DirectX je vključen zaradi vzvratne združljivosti s prejšnjimi verzijami DirectX knjižnice.
    - Direct3D (D3D): za izris 3D grafike.
    - DXGI: za lažje upravljanje z večimi zasloni. Prvič predstavljeno v DirectX 10.
    - Direct2D: za izris 2D grafike.
    - DirectWrite: za različne sloge pisave.
- DirectCompute: za izvajanje računskih ukazov na grafični procesni enoti (ang. GPU).
- DirectInput: namenjen kot vmesnik za vhodne naprave, kot so tipkovnica, miška, igralna palica in kontroler.
- DirectPlay: namenjen komunikaciji preko lokalne mreže ali svetovnega spleta.
- DirectSound: namenjen predvajanju in snemanju zvoka.
    - DirectSound3D (DS3D): namenjen predvajanju zvoka v 3D.
- DirectMusic: namenjen predvajanju skladb, ustvarjenih v DirectMusic Producer.
- DirectX Media: sestavlja DirectAnimation za 2D/3D animacijo v spletnem okolju.
- DirectX Diagnostics (DxDiag): orodje, namenjeno reševanju težav in generiranju poročil o komponentah, od katerih je DirectX odvisen.
- DirectX Media Objects: prinaša podporo prenašanju objektov, kot so kodirniki in dekodirniki.
- DirectSetup: namenjen namestitvi DirectX komponent in zaznavanju trenutno nameščene DirectX verzije.

Direct3D (programski vmesnik za 3D grafiko) je ustaljen standard pri izdelavi video iger za Microsoft Windows, Microsoft Xbox in Microsoft Xbox 360 platforme. Ker je Direct3D najbolj znana in oglaševana komponenta DirectX-a, velikokrat pride do zamenjave imen

DirectX in Direct3D. V Direct3D se med drugim računa tudi točka gledanja, ki jo želimo spremeniti, in ga je za to treba bolj podrobno pogledati.

4.2 Direct3D

Direct3D [15] je na voljo samo v Microsoft Windows operacijskem sistemu in odprtokodni distribuciji programske opreme Wine. Direct3D se uporablja za izris 3D grafike v programih, kjer je hitrost ključnega pomena, predvsem v igrah. Direct3D omogoča delovanje v celozaslonskem načinu namesto prikazovanja v namiznem oknu. Vsekakor pa imajo programi možnost teči v okenskem načinu, če so tako programirani. Direct3D uporablja strojno pospeševanje na grafični kartici, če le-ta to omogoča. Strojno je sposobna pospešiti tako celotni izrisovalni cevovod (ang. rendering pipeline), kot tudi samo posamezne dele. Direct3D zna izkoristiti napredne zmožnosti grafičnih kartic, kot so: z-buffering, anti-aliasing, alpha blending, mipmapping, atmospheric effects in perspective-correct texture mapping.

4.2.1 D3DX

Direct3D pride z D3DX, knjižnico, ki pomaga pri izračunu mnogih matematičnih operacij na vektorjih, matrikah, računa matriko pogleda, projekcijsko matriko. Pomaga tudi pri bolj težavnih opravilih, kot je prevod senčnikov, uporabljenih za 3D programiranje in shranjevanje zgoščene animacije okostja.

4.2.2 Direct3D matrike in prostor

V Direct3D moramo nastaviti 3 različne matrike [16], da se lahko tri-dimenzionalni prostor pravilno izriše. Te matrike se uporabljajo za transformacijo 3D objektov, iz njihovih začetnih položajev v končno 2D sliko na ekranu.

4.2.2.1 Transformacijski koraki

- Na začetku ustvarimo 3D objekt v urejevalniku (npr. 3D studio max). Običajno nastavimo vrednosti položaja objekta na 0, 0, 0. Položaj je definiran z **x**, **y** in **z** vektorji, ki določajo pozicijo objekta v relativni odmaknjenosti od središča. Ti položaji so določeni v modelnem prostoru (ang. *model space*).
- Sedaj želimo te objekte prestaviti, zavrteti in povečati/pomanjšati v svetu, v katerem bo delovala igra. To dosežemo tako, da Direct3D-ju podamo matriko, ki transformira naš objekt iz modelnega prostora v prostor sveta (ang. *world space*). Ta matrika je

imenovana matrika sveta (ang. *world matrix*).

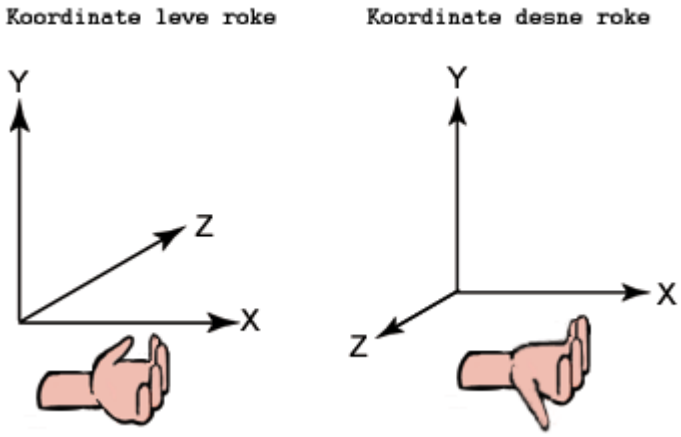
- Ta svet si pa želimo ogledati z različnih možnih točk. Tem točkam rečemo tudi točka kamere ali točka gledanja. Da dosežemo to transformacijo, podamo Direct3D-ju matriko, imenovano matrika pogleda (ang. *view matrix*), ki transformira prostor sveta v prostor pogleda (ang. *view space*). Včasih se prostor pogleda imenuje tudi prostor kamere.
- Nazadnje moramo Direct3D-ju povedati še, kako naj naš 3D prostor prikaže na 2D zaslonu. Podamo mu matriko, ki določa, kako prostor pogleda spremeniti v zaslonski prostor (ang. *screen space*). Matrika se imenuje projekcijska matrika (ang. *projection matrix*).

4.2.2.2 Povzetek vseh treh matrik

1. Matrika sveta – transformira 3D podatke iz modelnega prostora v prostor sveta. To matriko moramo nastaviti vsakič, ko želimo prikazati kak objekt v našem 3D svetu.
2. Matrika pogleda – transformira prostor sveta v prostor pogleda. To moramo nastaviti vedno, ko se točka gledanja spremeni.
3. Projekcijska matrika – transformira prostor pogleda v zaslonski prostor. To običajno nastavimo samo enkrat med inicializacijo.

4.2.3 Koordinatni sistem

Kot zanimivost velja omeniti, da imata DirectX in OpenGL drugačen sistem označevanja osi. Direct3D uporablja princip leve roke, kjer X os kaže desno, Y os navzgor in Z navznoter proti zaslonu. Pri OpenGL kaže X desno, Y navzgor in Z proč od zaslona. Pretvorba iz enega sistema v drugega zahteva, da eno od osi pretvorimo v njeno inverzno inačico. Običajno Z os pretvorimo v njeno inverzno vrednost.



Slika 8: Različna orientacija koordinatnega sistema

4.2.4 Programska struktura Direct3D matrika

Za upravljanje z zgoraj omenjenimi matrikami Direct3D predstavlja programsko strukturo, ki nam olajša delo z njimi. Struktura se imenuje D3DXMATRIX. Ta struktura vsebuje 4-krat po 4 elemente kot vrednosti matrike in predstavlja matriko velikosti 4x4. Tip lahko deklariramo enostavno z:

D3DXMATRIX worldMatrix;

To ustvari spremenljivko tipa D3DXMATRIX, imenovano worldMatrix.

Obstaja mnogo funkcij v Direct3D, ki jih lahko pokličemo in s pomočjo njih manipuliramo z matriko.

4.2.4.1 Matrika sveta

Matrika sveta se uporablja za transformacijo iz modelnega prostora v prostor sveta. Uporabljamo jo za premik, povečanje/pomanjšanje, obračanje objektov v svetu. Pri tem nam Direct3D ponuja vrsto funkcij, ki nam to olajšajo:

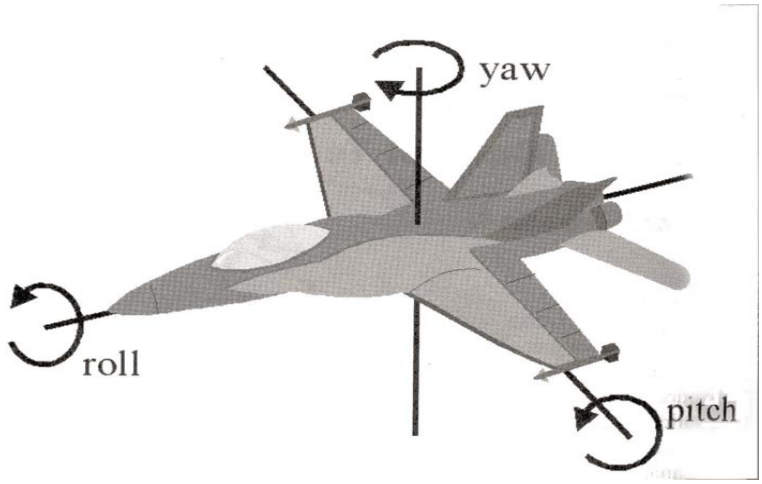
D3DXMatrixTranslation(D3DXMATRIX *out, FLOAT x, FLOAT y, FLOAT z);

Ta funkcija, kot prvi argument sprejme kazalec na matriko, v katero se bo shranil rezultat, drugi argumenti so uporabljeni za pozicioniranje objekta v 3D prostor.

Ko objekt pozicioniramo na pravo mesto, ga želimo tudi obrniti v želeno smer. V ta namen uporabimo sledeče funkcije:

D3DXMatrixRotationX(D3DXMATRIX *pOut, FLOAT angle); //ang. pitch
D3DXMatrixRotationY(D3DXMATRIX *pOut, FLOAT angle); //ang. yaw
D3DXMatrixRotationZ(D3DXMATRIX *pOut, FLOAT angle); //ang. roll

Zgoraj omenjene tri funkcije nam omogočajo, da lahko objekt zavrtimo okoli katere koli osi. Rezultat se shrani v matriko, ki jo funkcija sprejme kot prvi argument. Funkciji pa moramo podati tudi kot, za katerega želimo element obrniti, izražen v radianih.



Slika 9: Rotacije okoli vseh treh smeri

Na koncu želimo naš objekt še povečati/pomanjšati. V ta namen uporabimo:

D3DXMatrixScaling(D3DXMATRIX *out, FLOAT sx, FLOAT sy, FLOAT sz);

Rezultat se shrani v matriko, katere kazalec je podan kot prvi argument. Ostali argumenti določajo razteg/krčenje v x, y in z smeri.

Sedaj je naš objekt pravilne velikosti, na pravem mestu v svetu in obrnjen v želeno smer. Čas je, da si izberemo točko gledanja.

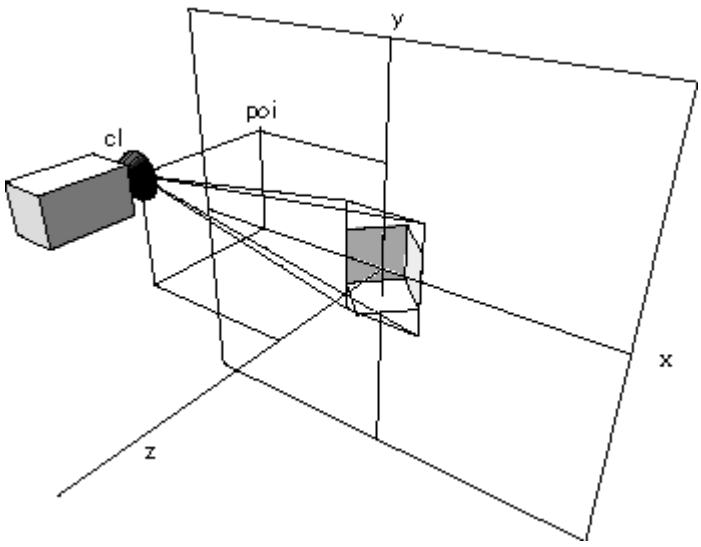
4.2.4.2 Matrika pogleda

Matriko pogleda uporabljamo za transformacijo iz prostora sveta v prostor pogleda. Matrika omogoča, da določimo, iz katere točke na svetu gledamo in v katero smer gledamo. To matriko moramo spreminjati vedno, ko se položaj ali smer gledanja kamere spremeni. Matriko kreiramo z naslednjo funkcijo:

D3DXMatrixLookAtLH(D3DXMATRIX *out, CONST D3DXVECTOR3 *oko, CONST D3DXVECTOR3 *tocka, CONST D3DXVECTOR3 *navzgor);

kjer je:

- out – kazalec na naslov matrike pogleda, v katero se bo shranil rezultat
- oko – točka gledanja v svetu (na kateri poziciji se nahaja naše oko/kamera)
- tocka – točka, kamor je pogled v svetu usmerjen
- navzgor – moramo določiti, katera smer je navzgor.



Slika 10: Točka pogleda na postavljen objekt

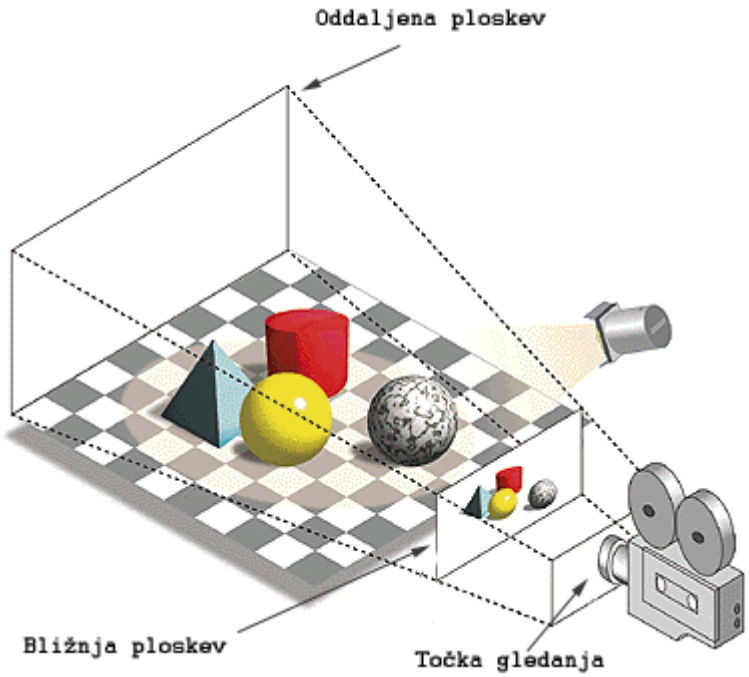
4.2.4.3 Projekcijska matrika

Projekcijska matrika določa, kako transformiramo 3D svet v 2D obliko, primerno za naš zaslon. Za konstrukcijo projekcijske matrike uporabimo sledečo funkcijo:

D3DXMatrixPerspectiveFovLH(D3DXMATRIX *out, FLOAT fovY, FLOAT Aspect, FLOAT zn, FLOAT zf);

kjer je:

- out - kazalec na naslov projekcijske matrike, v katero se bo shranil rezultat
- fovY – kakšno polje gledanja potrebujemo; najpogostejša je vrednost D3DX_PI/4
- Aspect - določimo zaslonsko razmerje, ki ga potrebujemo. Zadostuje, da delimo širino zaslona z višino zaslona.
- zn – to je z vrednost, s katero določimo bližnjo ploskev. Karkoli bo bližje tej vrednosti, ne bo izrisano. To vrednost običajno nastavimo na 1.0f.
- zf – to je z vrednost, s katero določimo oddaljeno ploskev. Karkoli bo bolj oddaljeno od te ploskve, ne bo izrisano.



Slika 11: Spreminjanje 3D sveta v 2D prikaz

4.2.4.4 SetTransform funkcija

Po tem ko vsako od zgoraj omenjenih matrik izračunamo, moramo njene vrednosti podati Direct3D napravi. V ta namen obstaja funkcija [17]:

void SetTransform(TransformType state, Matrix matrix);

Kjer je:

- state [18] – podatkovna struktura enum, s katero določimo, kateri tip matrike nastavljamo. Zaloga vrednosti: "world" (matrika sveta), "view" (matrika pogleda), "projection" (projekcijska matrika).
- matrix – matrika, ki jo podamo funkciji.

4.2.5 Povzetek

Iz zgornjih ugotovitev, kako Direct3D deluje, lahko zaključimo, da je mogoče točko gledanja spreminjati na dveh položajih. Lahko prestrezamo klice funkcije **D3DXMatrixLookAtLH** ali klice funkcije **SetTransform.** Zopet smo v položaju, kjer je potrebno pregledati prednosti in slabosti obeh pristopov:

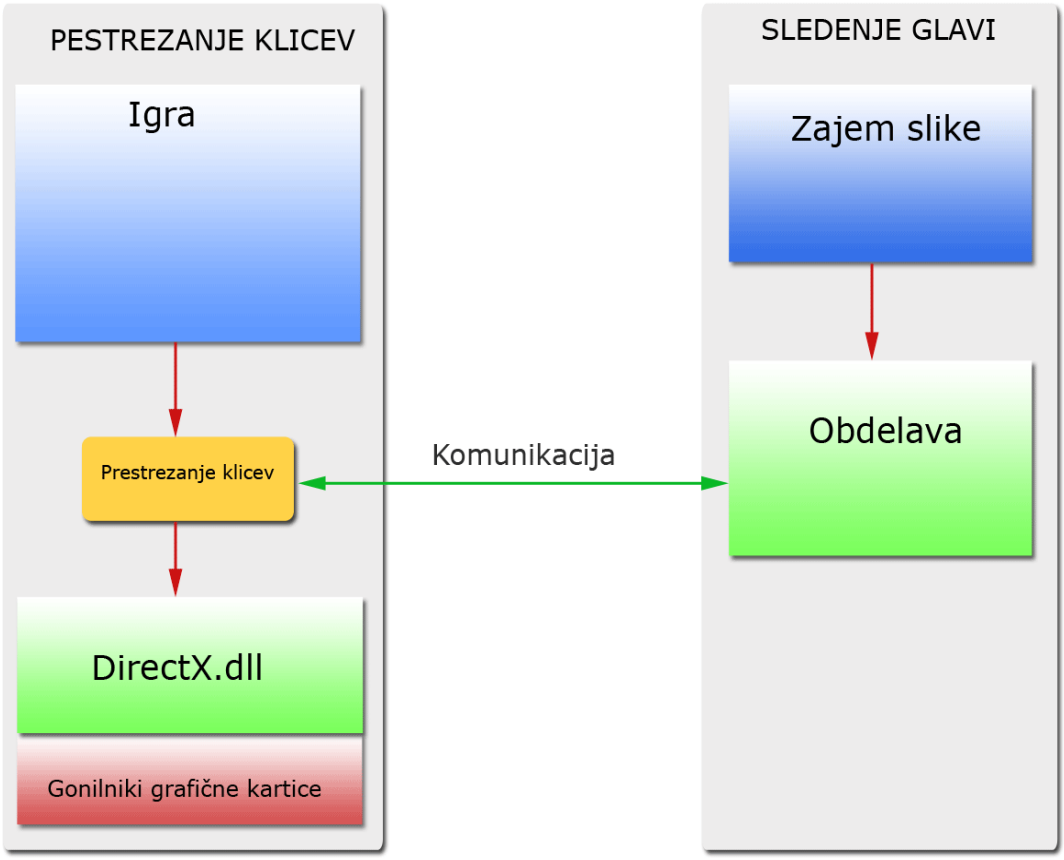
- **D3DXMatrixLookAtLH:** vhodni podatki o točki in smeri gledanja so podani v vektorski obliki in je manipulacija z njimi enostavna. Lahko se zgodi, da pisci igre ne uporabljajo te funkcije in so v ta namen napisali svojo.

- **SetTransform**: vhodni podatki o točki in smeri gledanja so podani v matrični obliki in je manipulacija z njimi malo otežena. SetTransform funkcija praviloma vedno sledi D3dXMATRIXLookAtLH funkciji. Tudi če so pisci igre uporabili svojo implementacijo D3dXMATRIXlookAtLH funkcije in Direct3D implementacijo setTransform funkcije, bo prestrezanje klicev pri slednji še vedno uspešno.

S prestrezanjem **setTransform** funkcije obstaja večja verjetnost, da je mogoče spreminjati točko gledanja. Iz tega razloga je slednje prestrezanje klicev bolj smotrno, to pa bomo dosegli z uporabo prej opisane metode "proxy.dll".

5 OPIS REŠITVE

Zaradi okoliščin, v katerih mora aplikacija delovati, lahko strukturo sistema razdelimo na dva večja modula, ki morata med sabo komunicirati. Prvi modul skrbi za prestrezanje klicev med igro in DirectX.dll knjižnico. Spreminjati mora točko gledanja, podatke za pravilno spremembo pa dobi od drugega modula. Drugi modul se poveže s spletno kamero, od katere dobiva v živo video posnetek, na katerem se nahaja uporabnik. Dodatna logika analizira video posnetek in določa položaj glave uporabnika.



Slika 12: Arhitekturni diagram predstavljene aplikacije

5.1 Modul prestrezanja klicev

Direct3D knjižnica (d3d8.dll) izvozi za zunanjo uporabo samo eno funkcijo, imenovano Direct3DCreate8, ki vrača kazalec na vmesnik IDirect3D8. Poskrbeti moramo, da pri klicu te funkcije vrnemo svojo izvedenko IDirect3D8 vmesnika, prvotno verzijo vmesnika pa shranimo. Pri prestrezanju se bodo podatki spreminjali v naši verziji vmesnika, nato pa se spremenjeni poslali naprej na prvotni vmesnik. Pa poglejmo programsko kodo tega modula.

5.1.1 proxy.cpp

(pomenben del izvorne datoteke, kjer se nahaja glavna funkcija knjižnice)

```
...
// globalne spremenljivke
#pragma data_seg (".d3d8_shared")
myIDirect3DDevice8* gl_pmyIDirect3DDevice8;
myIDirect3D8*       gl_pmyIDirect3D8;
HINSTANCE           gl_hOriginalDll;
HINSTANCE           gl_hThisInstance;
#pragma data_seg ()
#pragma comment (linker,"/section:.d3d8_shared,RWS")

// glavna metoda pri dll knjižnici
BOOL APIENTRY DllMain( HANDLE hModule, DWORD  ul_reason_for_call, LPVOID
lpReserved)
{
   // s tem se izognemo lvl4 opozoril pri prevajalniku
   LPVOID lpDummy = lpReserved;
   lpDummy = NULL;

   switch (ul_reason_for_call)
               {
                  case DLL_PROCESS_ATTACH: InitInstance(hModule); break;
                  case DLL_PROCESS_DETACH: ExitInstance(); break;

      case DLL_THREAD_ATTACH:  break;
                  case DLL_THREAD_DETACH:  break;
               }
   return TRUE;
```

```
}


// edina izvožena funkcija d3d8.dll knjižnice
IDirect3D8* WINAPI Direct3DCreate8(UINT SDKVersion)
{
   if (!gl_hOriginalDll) LoadOriginalDll(); // poiščemo originalni d3d8.dll

   // prestrežemo IDirect3D objekt iz prvotne knjižnice
   typedef IDirect3D8 *(WINAPI* D3D8_Type)(UINT SDKVersion);
   D3D8_Type D3DCreate8_fn =
(D3D8_Type)GetProcAddress( gl_hOriginalDll,"Direct3DCreate8");

      // izpis
      if (!D3DCreate8_fn)
      {
            OutputDebugString("PROXYDLL: Pointer to original D3DCreate8 function not
             received ERROR ****\r\n");
            ::ExitProcess(0); // izhod iz programa
      }

      // dobimo kazalec na originalni dll
      IDirect3D8 *pIDirect3D8_orig = D3DCreate8_fn(SDKVersion);

      // ustvarimo svoj IDirect3D8 objekt in vanj shranimo kazalec na originalni objekt
      gl_pmyIDirect3D8 = new myIDirect3D8(pIDirect3D8_orig);

      // vrnemo kazalec na našo verzijo vmesnika namesto na originalno
      return (gl_pmyIDirect3D8);
}

void InitInstance(HANDLE hModule)
{
      OutputDebugString("PROXYDLL: InitInstance called.\r\n");

      // inicializacija
      gl_hOriginalDll        = NULL;
      gl_hThisInstance       = NULL;
      gl_pmyIDirect3D8       = NULL;
```

```
        gl_pmyIDirect3DDevice8 = NULL;

        // shranimo referenco od instance med globalne spremenljivke
        l_hThisInstance = (HINSTANCE)  hModule;
}

void LoadOriginalDll(void)
{
   char buffer[MAX_PATH];

   // pridobimo pot do sistemskega direktorija
  ::GetSystemDirectory(buffer,MAX_PATH);

  // na koncu pripnemo ime dll-ja
  strcat(buffer,"\\d3d8.dll");

  // v primeru, da je kazalec prazen poskušaj naložiti d3d8.dll
  if (!gl_hOriginalDll) gl_hOriginalDll = ::LoadLibrary(buffer);

  // izpis
  if (!gl_hOriginalDll)
  {
      OutputDebugString("PROXYDLL: Original d3d8.dll not loaded ERROR ****\r\n");
      ::ExitProcess(0); // v primeru da ne moremo naložiti d3d8.dll je programa konec
  }
}

void ExitInstance()
{
   OutputDebugString("PROXYDLL: ExitInstance called.\r\n");

   // osvobodimo d3d8.dll knjižnico
   if (gl_hOriginalDll)
   {
       ::FreeLibrary(gl_hOriginalDll);
        gl_hOriginalDll = NULL;
   }
}
```

5.1.2 myIDirect3D8.cpp

(pomenben del izvorne datoteke, kjer nastane objekt Direct3DDevice)

```
myIDirect3D8::myIDirect3D8(IDirect3D8 *pOriginal)
{
   m_pIDirect3D8 = pOriginal;  // shranimo kazalec na prvotni objekt
}

HRESULT __stdcall myIDirect3D8::QueryInterface(REFIID riid, void** ppvObj)
{
   *ppvObj = NULL;

   // poglejmo, če ustrezni vmesnik obstaja
   HRESULT hRes = m_pIDirect3D8->QueryInterface(riid, ppvObj);

   if (hRes == NOERROR) // če je vse vredu pošljemo ven naš “lažni” naslov
   {
      *ppvObj = this;
   }

   return hRes;
}

ULONG __stdcall myIDirect3D8::AddRef()
{
   // pokličemo prvotno funkcijo
   return(m_pIDirect3D8->AddRef());
}

ULONG __stdcall myIDirect3D8::Release()
{
   extern myIDirect3D8* gl_pmyIDirect3D8;

   // pokličemo prvotno funkcijo
   ULONG count = m_pIDirect3D8->Release();

   // v primeru, da je število referenc 0, pomeni, da se je originalni objekt zbrisal
   // zato tudi mi zbrišemo svoj objekt
   if (count == 0)
```

```
    {
        gl_pmyIDirect3D8 = NULL;
        delete(this);
    }

return(count);
}

HRESULT __stdcall myIDirect3D8::CreateDevice(UINT Adapter,D3DDEVTYPE
DeviceType,HWND hFocusWindow,DWORD
BehaviorFlags,D3DPRESENT_PARAMETERS*
pPresentationParameters,IDirect3DDevice8** ppReturnedDeviceInterface)
{
    // globalna spremenljivka
    extern myIDirect3DDevice8* gl_pmyIDirect3DDevice8;

    // ta klic prestrežemo in podamo svoj  "lažni" Direct3dDevice objekt
    HRESULT hres = m_pIDirect3D8->CreateDevice( Adapter, DeviceType, hFocusWindow,
                                        BehaviorFlags, pPresentationParameters,
ppReturnedDeviceInterface);

    // ustvarimo svoj Direct3Ddevice objekt in ga shranimo v globalno spremenljivko
    gl_pmyIDirect3DDevice8 = new myIDirect3DDevice8(*ppReturnedDeviceInterface);

    // programu vrnemo naslov od svojega Direct3DDevice objekta
    *ppReturnedDeviceInterface = gl_pmyIDirect3DDevice8;

    return(hres);
}
```

5.1.3 myIDirect3Device8.cpp

(pomenben del izvorne datoteke mojega Direct3DDevice objekta)

```
myIDirect3DDevice8::myIDirect3DDevice8(IDirect3DDevice8* pOriginal)
{
    HANDLE Handle_Of_Thread_1 = 0;
    // v konstruktorju ustvarimo novo nit, ki bo prejemala podatke o položaju glave
    Handle_Of_Thread_1 = CreateThread(NULL, 0, Thread_no_1, NULL, 0, NULL);
```

```
    // shranimo naslov prvotnega objekta
    m_pIDirect3DDevice8 = pOriginal;
}

DWORD WINAPI Thread_no_1(LPVOID lpParam)
{
    HANDLE  hStdout = NULL;

    try{
        // preveri dosegljivost imenskega voda (ang. named pipe)
        FILE* fpread = fopen("\\\\.\\pipe\\ttrace","r");
        if(fpread == NULL)
        {
            // v primeru, da je imenski vod nedosegljiv zaključi nit
            return 0;
        }
        // vstopi v neskončno zanko, kjer poskušaj podatke prebrati vsakih 50 milisekund
        while(true){
                        int i = 0;
                        int j = 0;
                        int k = 0;

                        // poskušaj prebrati podatke z imenskega voda
                        fscanf(fpread, "%d %d %d", &i, &j, &k);
                        // zakleni možnost spreminjanja globalnih podatkov
                        lock();
                        // shrani prebrane vrednosti v globalne podatke
                        x = i;
                         y = j;
                        w = k;
                        // odkleni možnost spreminjanja globalnih podatkov
                        release();
                        Sleep(50);
                        }
    fclose(fpread);
    }
    catch(exception e){
    }
    return 0;
```

```
}

// inicializacija matrike s samimi ničlami
inline D3DMATRIX ZeroMatrix(void)
{
   D3DMATRIX ret;
   for (int i=0; i<4; i++)
      for (int j=0; j<4; j++)
      ret.m[i][j] = 0.0f;
   return ret;
}

// množimo dve metriki
inline D3DMATRIX MatrixMult(const D3DMATRIX a, const D3DMATRIX b)
{
   D3DMATRIX ret = ZeroMatrix();
   for (int i=0; i<4; i++)
      for (int j=0; j<4; j++)
         for (int k=0; k<4; k++)
         ret.m[i][j] += a.m[k][j] * b.m[i][k];
   return ret;
}

// funkcija v kateri spreminjamo točko gledanja
HRESULT __stdcall
myIDirect3DDevice8::SetTransform(  D3DTRANSFORMSTATETYPE State,CONST
D3DMATRIX* pMatrix)
{
   // vrednosti spreminjamo samo takrat, kadar je v obdelavi matrika pogleda
   if(State == 2){

      D3DMATRIX* matrix = new D3DMATRIX(*pMatrix);
      // zaklenemo dostop do globalnih spremenljivk
      lock();

      // matriki, ki določa položaj točke gledanja prištejemo premike v smeri x, y in z
      float x_fl = (float)(x)/40;
      matrix->_41 = matrix->_41 + x_fl;
```

```
float y_fl = (float)(y)/40;
matrix->_42 = matrix->_42 + y_fl;

float z_fl = (w-80)/20;
matrix->_43 = matrix->_43 + z_fl;


// določimo, za kakšen kot želimo zavrteti točko gledanja
double angleY = -((double)(x)/300);

// ustvarimo prazno matriko in jo napolnimo s podatki za rotacijo pogleda okrog y osi.
// ko bomo to matriko množili z matriko pogleda, se bo ta zavrtela za določen kot.
D3DMATRIX *viewY = new D3DMATRIX();
viewY->_11 = cos(angleY);
viewY->_33 = cos(angleY);
viewY->_13 = sin(angleY);
viewY->_31 = -sin(angleY);
viewY->_22 = 1;
viewY->_44 = 1;

// ustvarimo prazno matriko in jo napolnimo s podatki za rotacijo pogleda okrog x osi.
double angleX = ((double)(y)/300);
D3DMATRIX *viewX = new D3DMATRIX();
viewX->_22 = cos(angleX);
viewX->_33 = cos(angleX);
viewX->_32 = sin(angleX);
viewX->_23 = -sin(angleX);
viewX->_11 = 1;
viewX->_44 = 1;

// odklenemo dostop do globalnih spremenljivk
release();

// matriki, ki obračata pogled v X in Y smeri, med seboj  zmnožimo
D3DMATRIX* rot = &MatrixMult(*viewX, *viewY);
// rotacijsko matriko zmnožimo z matriko pogleda
matrix = &MatrixMult(*rot, *matrix);

// spremenjeno matriko pošljemo naprej originalnemu Direct3DDevice objektu
```

```
        return(m_pIDirect3DDevice8->SetTransform(State, matrix));
        }
// v primeru, da ne nastavljamo matrike pogleda, jo pošljemo prvotnemu Direct3Ddevice
objektu
return (m_pIDirect3DDevice8->SetTransform(State, pMatrix) );
}
```

5.2 Modul sledenja glavi

Naloga modula sledenja glavi je odčitavanje trenutne pozicije glave uporabnika in sporočanje teh podatkov modulu, ki prestreza klice DirectX knjižnice. Pri tem modulu nismo omejeni na določen programski jezik in je zaradi hitrosti razvoja izbira C# jezika smiselna.

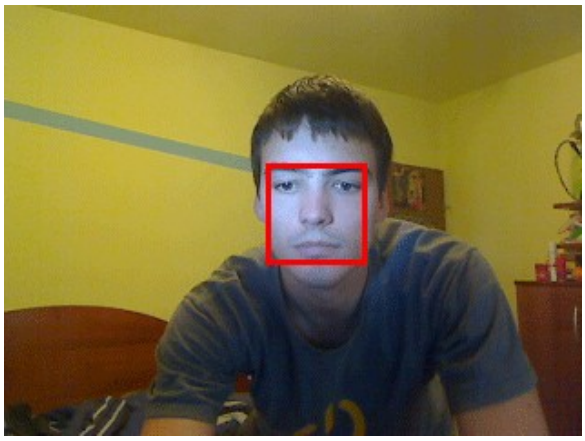
Za zajem slike uporabimo touchless SDK [19] knjižnico, napisano v C# jeziku. Namen te knjižnice je sledenje objektom iste barve, vendar vsebuje enostaven in učinkovit vmesnik za dostop do spletne kamere, ki ga uporabimo za zajem slike.

Za prepoznavanje obrazov na sliki je primerna fdlib [20] knjižnica, ki jo je razvil Wold Kienzle v C++ jeziku. Dejstvo, da je ta knjižnica napisana v drugem programskem jeziku, k sreči ne predstavlja velike ovire, saj Microsoft na .Net platformi (na kateri teče tudi C#) dobro podpira klicanje v C in C++ jeziku napisanih programskih modulih.

5.3 Rezultat

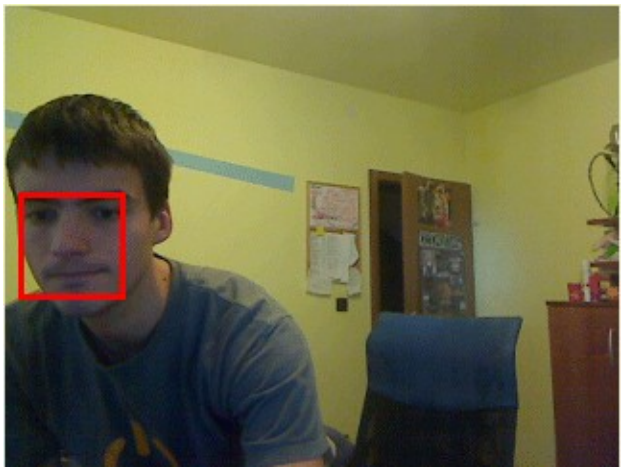
S programom želimo doseči, da gledanje dogajanja v igri na zaslonu ne bo več tako statično. Privzeto v igri, ne glede na to, kje sedimo za zaslonom, je dogajanje in izgled igre enak. Želimo doseči učinek, ki ga izkusimo npr., ko pogledamo skozi okno. Če premaknemo glavo proti levi, vidimo več zunanjosti na desnem robu okna. Če premaknemo glavo proti desni, vidimo več na levem robu okna. Podobno velja za premike v smeri navzgor, navzdol, naprej in nazaj. Spodaj se nahaja zbirka slik, kjer prva predstavlja položaj glave za zaslonom, druga pa učinek (spremembo točke gledanja) v igri.

5.3.1 Glava v sredinskem položaju



Slika 13: glava v sredinskem položaju

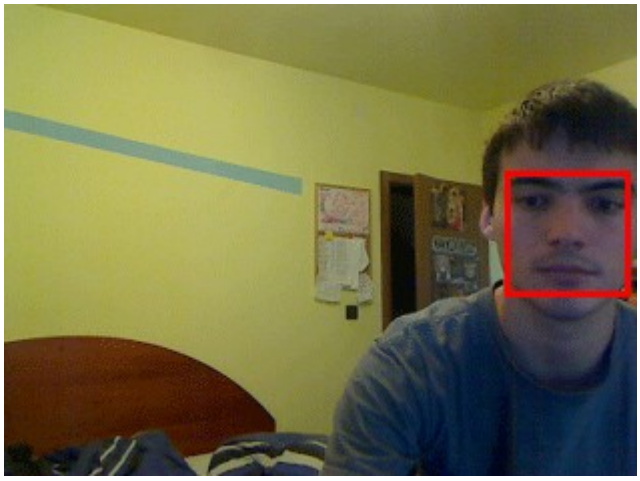
5.3.2 Glava, premaknjena na desno (iz zornega kota uporabnika)



Slika 14: Točka gledanja, premaknjena v desno in smer gledanja, premaknjena v levo

Slika 15: V igri se vidi večji del leve strani sobe

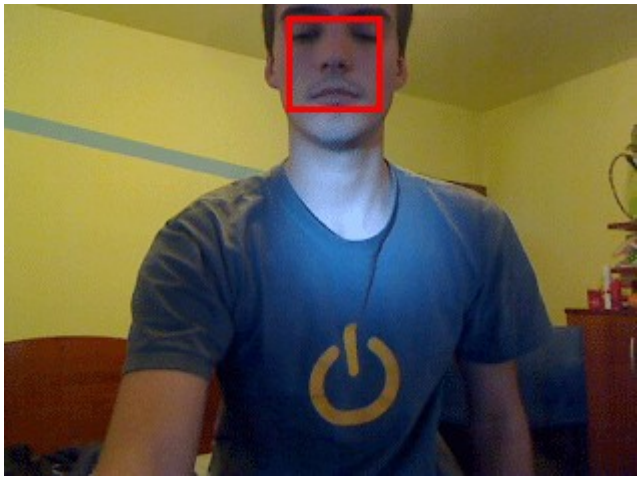
5.3.3 Glava, premaknjena na levo (iz zornega kota uporabnika)



Slika 16: Točka gledanja, premaknjena v levo in smer gledanja, premaknjena v desno

Slika 17: V igri se vidi večji del desne strani sobe

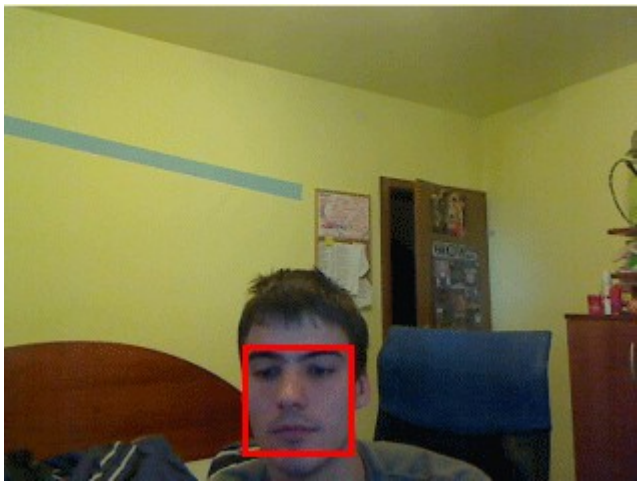
5.3.4 Glava, premaknjena navzgor.



Slika 19: Točka gledanja, premaknjena navzgor in smer gledanja, premaknjena navzdol

Slika 20: V igri se vidi večji del spodnje strani sobe

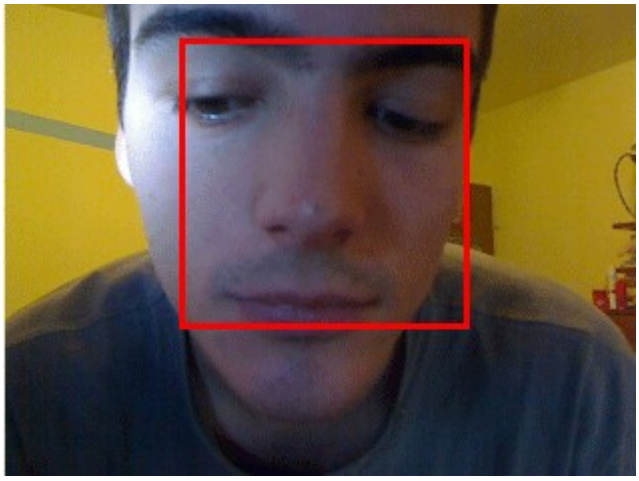
5.3.5 Glava, premaknjena navzdol.



Slika 21: Točka gledanja, premaknjena navzdol in smer gledanja, premaknjena navzgor

Slika 22: V igri se vidi večji del zgornje strani sobe

5.3.6 Glava, premaknjena bližje ekranu.



Slika 23: Glava, premaknjena bližje ekranu

Slika 24: Pogled v igri približan

5.3.7 Glava, premaknjena proč od ekrana.

Slika 25: Glava, premaknjena proč od ekrana

Slika 26: Pogled v igri oddaljen

6 Sklep

Po nekajurni uporabi lahko vidimo, da je želeni učinek pogleda v virtualni svet delno dosežen. Uporabnik lahko v igri za nekaj trenutkov res dobi občutek, da gleda v 3D svet skozi okno, vendar samo izkušnjo spremlja preveč težav, da bi bil ta način gledanja splošno uporaben. Te težave so tako programske kot strojne narave in nekatere so premostljive (predvsem programske), medtem ko druge ne (predvsem strojne).

6.1 Težave

6.1.1 Zamik spletnih kamer (latenca)

Spletne kamere najprej zajamejo sliko, potem pa programska oprema na kameri zajete podatke obdela, jim doda barve, izravna belo svetlobo ipd. Ko je obdelava končana, obdelane podatke skrči in jih pošlje po USB vodilu. Na drugi strani CPU skrčene podatke razširi in tako iz podatkov lahko razbere sliko.

Na žalost ves ta proces nekaj časa traja in to se pozna v zamiku spletnih kamer. Od zajema slike na kameri in prikaza te slike na monitorju vedno preteče nekaj časa. Velikost zamika je odvisna predvsem od osvetljenosti prostora in kakovosti spletne kamere. Ta vrednost se giblje med 100 ms in 300 ms.

Ta zamik je dovolj velik, da škoduje uporabniški izkušnji. Premik glave v realnem svetu se v igri upošteva v najboljšem primeru šele po 100 milisekundah. Dovolj, da uporabnik to opazi in da ga to zmoti.

Logitech je za določeno serijo svojih kamer razvil možnost, da kamere pošljejo neobdelano črno-belo sliko preko USB vodila [21]. Pozitivna posledica je občutno zmanjšan zamik (skoraj ničen), vendar na žalost to deluje na zelo omejenem naboru spletnih kamer višjega cenovnega razreda.

To je največja težava, ki se je pojavila v razvoju tega projekta, saj je nemogoče običajno spletno kamero pripraviti do manjših časovnih razlik pri zamiku.

6.1.2 Premik vseh predmetov

Kadar v igri spremenimo točko gledanja, spremenimo položaj vseh predmetov, ki se nahajajo v igri. Na žalost pa se izkaže, da želimo biti glede premikanja predmetov selektivni. Sprememba točke gledanja premakne izris vseh objektov v 3D svetu, vendar to pomeni, da premakne tudi 3D text in menije, ki so postavljeni tako, da se izrišejo točno pred zaslonom. In ker smo premaknili točko gledanja, teh menijev ne vidimo več. To lahko zmoti uporabnika do te mere, da ne more dostopati do menijev in določenih funkcij v igri.

Kako zaobiti to težavo, zaenkrat še ni znano, vendar bi verjetno z dodatnim raziskovanjem v tej smeri lahko prišli do rešitve.

6.1.3 Identifikacija objekta pod miškinim kazalcem

Sam and Max igra spada v kategorijo "pustolovskih iger" in pri tem tipu iger je potrebna interakcija med različnimi predmeti, ki se nahajajo v igri. Običajen vmesnik za izbiro predmeta v teh igrah je prav miških kazalec in Sam and Max ni nobena izjema.

Če želimo nad nekim predmetom izvesti določeno akcijo, postavimo miških kazalec nad predmet in kliknemo. Težava se pojavi, ko s programom spremenim točko gledanja brez vednosti igre. Sedaj se predmet ne nahaja več na isti lokaciji na zaslonu kot prej in igra nima informacije o novi lokaciji predmeta.

Posledično ob premiku točke gledanja klik na predmet ne izvede več želene akcije. Ta težava je tako moteča, da uporabnik po vsej verjetnosti ne bo mogel nadaljevati z igro. Obstaja verjetnost, da je tudi ta problem mogoče premostiti, vendar bi zahteval dobršno mero raziskovanja.

7 LITERATURA

[1] (2009) Virtual boy. Dostopno na: http://en.wikipedia.org/wiki/Virtual_Boy

[2] (2009) Power glove. Dostopno na: http://en.wikipedia.org/wiki/Power_glove

[3] (2009) Wii Remote. Dostopno na: http://en.wikipedia.org/wiki/Wii_Remote

[4] (2009) Project Natal. Dostopno na: http://en.wikipedia.org/wiki/Project_Natal

[5] (2009) Project natal. Dostopno na: http://www.xbox.com/en-US/live/projectnatal/

[6] (2009) Eye Toy. Dostopno na: http://en.wikipedia.org/wiki/EyeToy

[7] (2009) Sony motion sensor controller. Dostopno na: http://www.siliconera.com/2009/07/02/sony-patents-a-motion-control-system-that-uses-ordinary-objects-as-controllers/

[8] (2009) Johnny Chung Lee Wi remote. Dostopno na: http://johnnylee.net/projects/wii/

[9] (2009) TrackIR. Dostopno na: http://www.naturalpoint.com/trackir/

[10] (2009) TrackIR. Dostopno na: http://en.wikipedia.org/wiki/Trackir

[11] (2009) Računalniški proces. Dostopno na: http://en.wikipedia.org/wiki/Process_%28computing%29

[12] (2009) Programska knjižnica. Dostopno na: http://sl.wikipedia.org/wiki/Programska_knji%C5%BEnica

[13] (2009) Vdor v izvršilno datoteko. Dostopno na: http://en.wikipedia.org/wiki/Hooking

[14] (2009) DirectX. Dostopno na: http://en.wikipedia.org/wiki/Directx

[15] (2009) Direct3D. Dostopno na: http://en.wikipedia.org/wiki/Microsoft_Direct3D

[16] (2009) Direct3D matrike. Dostopno na: http://www.toymaker.info/Games/html/matrices.html

[17] (2009) SetTransform funkcija. Dostopno na: http://msdn.microsoft.com/en-us/library/microsoft.windowsmobile.directx.direct3d.device.settransform.aspx

[18] (2009) Zaloga vrednosti enum setTransform funkcije. Dostopno na: http://msdn.microsoft.com/en-us/library/microsoft.windowsmobile.directx.direct3d.transformtype.aspx

[19] (2009) Touchless knjižnica. Dostopno na: http://touchless.codeplex.com/

[20] (2009) Fdlib knjižnica. Dostopno na: http://www.kyb.mpg.de/bs/people/kienzle/fdlib/fdlib.htm

[21] (2009) Zajem neobdelane slike s spletne kamere. Dostopno na: http://www.quickcamteam.net/documentation/how-to/how-to-enable-raw-streaming-on-logitech-webcams