VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

3D APLIKACE PRO PLATFORMU ANDROID

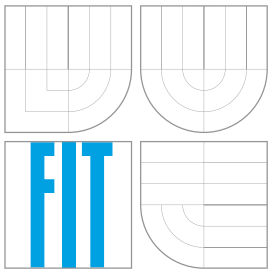
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAKUB SMEJKAL

BRNO 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

3D APLIKACE PRO PLATFORMU ANDROID

3D APLICATION FOR ANDROID PLATFORM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAKUB SMEJKAL
AUTHOR

VEDOUCÍ PRÁCE — Ing. ALEŠ LÁNÍK
SUPERVISOR

BRNO 2010

Abstrakt

Tato bakalářská práce řeší implementaci 3D aplikace pro platformu Android. Aplikace realizuje simulátor rubikovy kostky. Ovládání je realizováno s pomocí inerciální jednotky plus dalších druhů rozhraní. Pro podporu uživatelova řešení problému Rubikovy kostky je naimplementována i umělá inteligence.

Abstract

This bachelor thesis addresses the implementation of 3D application for platform Android. Application implements simulator of Rubik's cube. Control is implemented using an inertial unit, plus other types of interfaces. To support the user's solving Rubik's cube problem is implemented an artificial intelligence.

Klíčová slova

Platforma Android, OpenGL ES, Quaterniony, ArcBall rotace, Rubikova kostka, Vrstvový algoritmus.

Keywords

Platform Android, OpenGL ES, Quaternion, rotation ArcBall, Rubik's cube, Layered algorithm .

Citace

Jakub Smejkal: 3D aplikace pro platformu Android, bakalářská práce, Brno, FIT VUT v Brně, 2010

3D aplikace pro platformu Android

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Aleše Láníka

.....
Jakub Smejkal
17. května 2010

Poděkování

Na tomto místě bych rád poděkoval Ing. Aleši Láníkovi, vedoucímu bakalářské práce, za odbornou pomoc, ochotu a čas, který mi při tvorbě práce věnoval.

Obsah

1 Úvod **2**

2 Platforma Android **4**
2.1 Historie 4
2.2 Přehled technologie 6

3 Techonologie zobrazení, OpenGL ES **12**
3.1 Popis standartu 13
3.2 Zobrazování 3D prostorů v OpenGL ES 14

4 Umělá inteligence **17**
4.1 Popis Rubikovy kostky 18
4.2 Algoritmy 19

5 Řešení uživatelského rozhraní **22**
5.1 Možnosti ovládání kostky 22
5.2 ArcBall rotace 24

6 Implementace aplikace **28**
6.1 Možnosti vývoje 28
6.2 Kooperace Androidu a OpenGL ES 30
6.3 Konečná podoba aplikace 32

7 Měření 3D výkonu platformy **35**
7.1 Testy 35
7.2 Výsledky 38

8 Závěr, možnosti rozšíření **39**

A Přílohy **42**

Kapitola 1

Úvod

V posledních letech dochází k rozvoji elektronické komunikace, proto se stále více věnují výrobci telekomunikačních zařízení sféře mobilních telefonů a jejich platforem. Trendy v tomto odvětví jsou podobné jako v jiných technologických větvích, tedy urychlovat vývoj, snižovat jeho cenu a zkvalitňovat výrobky pro koncového uživatele. Tyto snahy postupně vedou k tvorbě nových technologii a celkově filozofií přístupu k jejich designu. Proto se na tomto poli začíná čím dál více prosazovat model open source, z něhož si jeho zástupci slibují naplnění výše popsaných trendů. Mezi jeho zástupce patří také Google a jeho platforma Android. Android nepatří mezi první open source systém na mobilních zařízeních, ale v současné době se mu připisuje největší význam a to především do budoucna.

Tato bakalářská práce se zabývá tvorbou aplikace pro tuto platformu s využitím inerciání jednotky. V práci jsem také kladl důraz na uživatelskou přívětivost ovládání, kterou jsem považoval jako velmi podstatnou při zpracování tématu programu. Tím je simulace Rubikovy kostky. K realizaci vizuální složky aplikace jsem se rozhodl použít Opengl ES, které je ostatně jedinou variantou pro zobrazení 3D objektů na platformě Android. V době vývoje práce byla dostupná pouze implementace verze 1.0. Tato verze obsahuje určitá omezení oproti plnohodnotné verzi na stolních počítačích a je také popsána v kapitolách níže. V současné verzi s příchodem Androidu 2.0 a s tím i NDK(native development kit) kitu, došlo k zpřístunění verze 2.0, která značně ulehčuje práci vývojáře. I když tento kit byl zpřístupněn až při uvedení výše zmíněné verze platformy, je možné ho používat již na verzi 1.5. K zobrazení samotné kostky a transformací s ní prováděných bylo potřeba připojit i zbytek uživatelského rozhraní, tedy menu. V tomto směru se nejlépe jevily přímo předtvořené widgety pro UI od Google, které jdou snadno střídat s Opengl zobrazením. Jako poslední část bakalářské práce jsem zahrnul i zjednodušený algoritmus umělé intelegince od Herberta Kociemby[5], který plní funkci nápovědy pro uživatele. Zmíněný algoritmus jsem mírně upravil pro jeho použití na mobilní platformě, kdy bylo potřeba upravit jeho výkonové požadavky. Samotné detaily budou popsány níže.

Celou technickou zprávu jsem rozdělil do několika částí, kde v první popisuji vznik platformy, její fylozofii a rozebírám základní principy fungování aplikací. Následně v druhé se věnuji popisu OpenGL ES a teorii ohledně vykreslování a vůbec reprezentace 3D prostorů. V další přichází na řadu rozbor Rubikovy kostky a algoritmů, pomocí nichž se dá řešit. V části „Řešení uživatelského rozhraní“ popisuji možnosti ovládání kostky a principy zakomponování zvoleného přístupu ArcBall rotace. V předposlední pak sumarizuji poznatky

získané při práci s platformou Android, tedy možností při vývoji aplikaci a samotnou realizaci. Poslední část reprezentuje měření výkonnosti na zapůjčeném zařízení HTC Desire v porovnání s prací z roku 2009 „3D aplikace pro mobilní telefony“ od Marka Vyorala[1]. Závěrem pak uvádím průběh studia, zhodnocení výsledků a několik námětů na rozšíření programu.

Kapitola 2

Platforma Android

2.1 Historie

Google a Android Historie operačního systému Android se datuje od roku 2005, kdy v červenci společnost Google koupila společnost Android, která se profilovala na poli tvorby softwaru pro mobilní zařízení. V té době to byla malá začínající firma, kterou založila čtveřice Andy Rubin, Rich Miner, Nick Sears a Chris White. Tímto krokem započal vývoj budoucí mobilní platformy Android. Vedoucím týmu se stal Andy Rubin, který měl vyplnit vizi platformy, která by měla být pružná a snadno aktualizovatelná. Systém měl stát na linuxovém jádře. Další kroky Google vedly ke spolupráci s různými softwarovými ale i hardwarovými firmami. Tato snaha následně vyústila v roce září 2007 k založení Open Handset Alliance. V tomtéž roce byl následně vypuštěn první SDK kit, který se během několik měsíců stal velmi žádaný u uživatelů. Následující rok Google představil první oficiální verzi systému Android 1.0.

Celá filozofie systému byla prezentována v usnadnění vývoje pomocí otevřeného kódu platformy a tím vytvoření standartu, který by vytvořil prostředí na poli mobilních zařízení po vzoru stolních počítačů se standarty jako PC, Macintosh, tedy rozšířit trh a vyřešit problém s přílišnou fragmentací platforem. Otevřenost by měla také zajistit lepší odladitelnost zákaznických aplikaci, tedy zvýšit jejich konkurenceschopnost a produktivitu. Android měl také změnit poměry u některý operátorů, kteří často omezovali nabídku aplikací třetích stran s odůvodněním, že by mohli zavléci do jejich sítí škodlivý software. Tento přístup by měl nahradit jediný repozitář takzvaný Android Market, odkud by si uživatelé mohli stahovat aplikace přímo z mobilního telefonu po vzoru AppStore od společnosti Apple. Software v tomto centrálním uložišti by měl být samozřejmě testovaný proti závadnosti. V současnosti je k dispozici implementace Androidu 2.1. Této verzi postupně předcházelo několik vývojových verzí:

- 0.9

  Vydána dne 18. srpna 2008. Tato verze obsahuje aktualizované a rozšířené API. Proběhla také úprava vývojových nástrojů.

- 1.0

  Vydána dne 23. září 2008. Aktualizace přinesla především opravu chyb ze starší verze, plus také některé změny v API.

- 1.1

  Vydána dne 9. března 2009. Verze upravuje vzhled uživatelského rozhraní. Poprvé je také přidána podpora pro vyhledávání hlasem. Došlo k úpravě některých vestavěných aplikací jako budík, emailová komunikace a jejich aktualizačních intervalů, plus změna funkcionality map. Podlední důležitá změna se týkala vývojářů a jejich „Dev“ zařízení, které mohou nyní pracovat s komerčními aplikacemi.

- 1.5 (Cupcake)

  Vydána v květnu 2009. Android v této verzi podporoval již nahrávání videa, stereo Bluetooth profily, přizpůsobitelnou virtuální klávesnici a systém pro rozpoznávání hlasu. Dále byl zpřístupněn AppWidget framework vývojářům softwaru třetí strany.

- 1.6 (Donut)

  Vydána v září 2009. Zahrnovala upravené vyhledávání, implementovaný indikátor baterie a VPN ovládací prvek. Přibyli nové technologie, jako „Text to Speech engine“, gesta a úpravě komunikace uživatele se systémem.

- 2.0/2.1 (Eclair)

  Vydán dne 26. října 2009. V této poslední verzi byla provedena optimalizace rychlosti systému, následně byla přidána podpora pro více velikostí obrazovky a rozlišení, provedena úprava uživatelského rozhraní prohlížeče, seznamu kontaktů a přidána podpora HTML5. Změnou prošly Google Mapy, přidána podpora Microsoft Exchange, plus implementována funkce blesku pro fotoaparáty a digitálního zoomu. Jako poslední podporuje Eclair standart Bluetooth 2.1

V roce 2008 se take dostal na český trh první telefon se systémem Android. Název přístroje zvolila firma T-mobile, která ho i distribuovala. Komunikátor G1 bylo první komerční zařízení pro Android. Jeho výrobu obstarala firma HTC pod kodovým označením Dream. Tento model má dnes už svého nástupce v podobě G2 a trend do budoucna je jednoznačný. Paleta telefonu se bude významě rozšiřovat. V tomto směru nezůstává ani Google pozadu. V roce 2010 hodlá na trh vypustit vlastní telefon pod názvem Nexus One.

Open Handset Alliance Sdružení bylo ohlášeno 5. listopadu 2007. V prohlášení společnosti byly definovány následující cíle. Uskupení si vzalo za cíl podporu vývojářů v umožnění co možná největší možné spolupráci na produktech a s tím spojené zlepšení kvality produktů a zrychlení jejich vývoje. Mezi další činnosti Open Handset Alliance patří přispívání vlastní tvorbou software a jinných duševních komodit do open source fondu. Tyto artikly jsou následně přístupné komunitě vývojářů pod licencí Apache V2. Tato licence umožňuje jejich použití, modifikování a následně buď ponechání, nebo produkování dále do open source obce. Produkce je přístupná na stránkách developerů Google: http://developer.android.com. Mezi členy Alliance patří:

- Výrobci mobilních zařízení.

  HTC, LG, Motorola, Samsung

- Mobilní operátoři.

  China Mobile Communications, KDDI, DoCoMo, Sprint/Nextel, T-Mobile, Telecom Italia, Telefonica

- Výrobci polovodičů.

  Audience, Broadcom, Intel, Marvell, Qualcomm NVidia, SiRF, Synaptics

- Softwarové společnosti.

  Ascender, eBay, esmertec, Google, LivingImage, Livewire, Nuance, Packet Video, SkyPop, SONiVOX

- Komercializační společnosti.

  Aplix, Noser, TAT, Wind River

Předešlé informace čerpány z [11].

2.2 Přehled technologie

Mobilní platforma Android se skládá ze samotného operačního systému, který poskytuje podporu pro vývojáře aplikací, a několika aplikací, sám obsahuje již pár základních pro potřebu uživatele. Google k tomuto systému také poskytuje SDK, které se dá aplikovat na různá vývojová prostředí. Primárně je podporován Eclipse. V samotném SDK je ještě skromná databáze příkladu, která tvoří vedle dokumentace hlavní podporu developera. Jako poslední produkt od Google přichází NDK, které tvoří most mezi prostředím v javě a nativním kódem, který lze díky němu do aplikace zahrnout a tím zvýšit její výkon.

Architektůra platformy Struktura Androidu se dá rozdělit na několik částí:

- Programové vybavení:

  Čistý systém podporuje několik základních funkcí přímo od výrobce, které poskytují klasickou telefoní funkcionalitu plus rozšíření ohledně internetu a map, které jsou s ním úzce svázány. Do této množiny také patří aplikace třetích stran, které nainstaluje uživatel.

- Rozhraní pro aplikace:

  Hlavním úkolem této části je poskytnout vývojáři široké možnosti interakce jeho produktu se systémem a jinými nainstalovanými aplikacemi. Vývojář může jejím prostřednictním využívat uživatelské rozhraní systému pro komunikaci s uživatelem, komunikovat s hardwarem daného zařízení, používat jeho kapacity a prosperovat z filozofie modularity systému a tím v případě widgetů také standartizovat ovládání svých aplikací.

- Knihovny:

  Platforma obsahuje množinu knihoven, které jsou dostupné vývojáři přes aplikační rozhraní. Knihovny jsou psány v jazycích C a C++. Patří sem především systémové knihovny, knihovny pro podporu medií, správci pro řízení zobrazení, implementace 3D knihoven a jiných grafických technologií, podpora pro databáze a některé úzce specializované jako řešení webového rozhraní a jinné.

- Virtuální stroj:

  Ten je implemetován pomocí Dalvik VM. Zde běží jednotlivé aplikace psané v javě, které využívají zakladního setu knihoven. Každá aplikace má svoji vlastní instanci

Dalvikova virtuálního stroje a ten následně komunikuje s linuxovým jádrem celého systému.

- Linuxové jádro:

  Tvoří vrstvu mezi softwarovým vybavením systému a hardwarem zařízení. Android je postaven na verzi 2.6.

Aplikační filizofie Samotný program pod Dalvikovým virtuálním strojem by se dal rozdělit do dvou základních částí, které spolu velmi úzce spolupracují:

- Část výpočetní:

  Zde by se daly zařadit aktivity(Activites) a služby(Sevices). Kdy oba zmíněné prvky systému provádějí výpočet aplikace, ale liší se tím, že aktivity jsou spjaty především s uživatelským rozhraním, tedy poskytují uživateli nástroje pro ovládání aplikace plus ho infromují o změnách vlastního stavu. Naproti tomu služby jsou spíše navrženy pro výpočet časově náročných operací, proto jsou pro ně vyhrazena separátní vlákna, ve kterých pracují bez toho, aby byl uživatel omezen v manipulaci s aplikací. Typicky to jsou akce, které nejsou aktuálně esenciální pro provoz aplikace, ale zkracují reakční dobu softwaru v budoucnu.

- Část komunikační

  Tato sekce je více neobvyklejší vzhledem k běžným zvyklostem chování programů a naplňuje filozofii platformy Android. Komunikační část totiž poskytuje nejen informace aplikacím a systému, ale i výpočetní výkon a data. Souhrně sem jdou zařadit následující pojmy: poskytovatelé obsahu(Content providers), přijímače plošných zpráv(Broadcast receivers) a komunikační nástroj účely(Intents). Pro poskytování dat slouží primárně Content providers. Zde operuje dvojice provider a resolver, kdy od providera jsou žádána data přes resolver, jehož metody jsou volány a on sám volá implementované metody providera, který již poskytuje finální data. Tyto data jsou dostupná jak v jedné tak mezi různými dvěma aplikacemi. Následně pojem broadcast receriver je spojen s ostatními ve smyslu, že využívá služeb Intentů. Tuto formu komunikace používájí i aktivity a služby. Intenty jsou souhrně řečeno objekty, které reprezentují více známé zprávy, jejichž obsah se následně liší podle druhu komunikace, tedy jaký druh dvojice mezi sebou komunikuje. Pro aktivity a služby to znamená přenos akce, která je požadována, a odkaz na data, nad kterými má proběhnout, a pro broadcast receivery pouze signalizace události. Samotné broadcast receivery slouži k informování aplikací na různé systémové události. Například pokud dochází baterie. Aplikace pak sami rozhodují které broadcasty budou přijímat a úprava jejich chování už je pak na tvůrci daného softwaru.

- Manifest

  Jako poslední a neméně důležitý prvek v komunikaci vystupuje soubor Manifest.xml, který se jmenuje konstatně pro každou aplikaci. Představuje informování systému o jednotlivých částech aplikace, tedy prostředky, jež mu může nabídnout, nebo také jiným aplikacím. Soubor souhrhně identifikuje aktivity, služby, broadcast receivery, content resolvery a také specifikuje filtry intentů. Tyto elementy následně slouží k orientaci systému při vybírání správného poskytovatele služeb. Tato struktura je využita především v případě, pokud zdroj pro intent žadatele není přesně specifikován,

proto systém pak následně pomocí těchto filtrů porovnává, která aplikace je schopna poskytnout nejlepší možný výstup.

- Úkoly

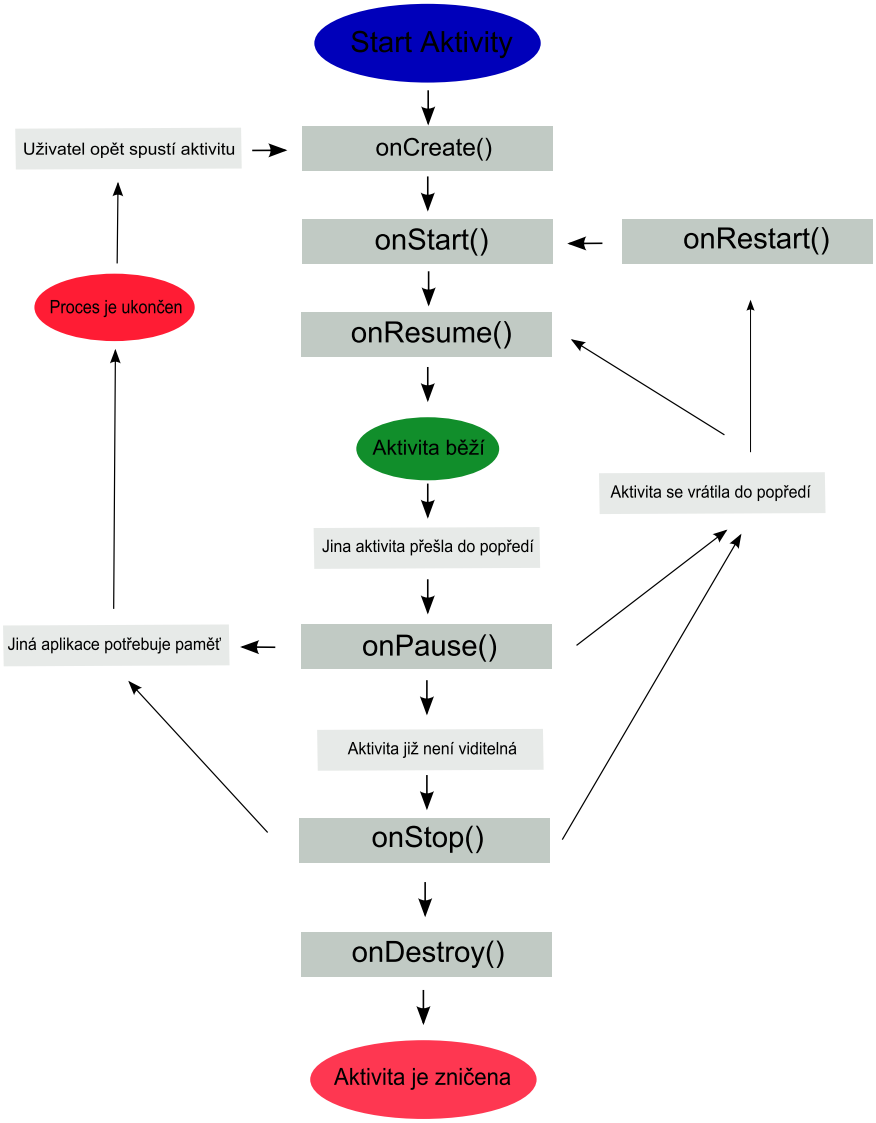
  Toto základní rozdělení s elementárními komponentami systému se pak váže do struktur, které se distribuuje ve formátu apk. Po spuštění se pro každou aplikaci vytvoří samostaný proces a má také vlastní instanci virtuálního stroje. Tyto procesy následně tvoří logiku kooperace aplikací v systému na základě potřeb uživatele. Tato logika se nazývá Úkol(Task), která schraňuje jednotlivé aktivity v daném procesu. Fungování této struktury prakticky odpovídá modelu zásobníku, kdy aktuální aktivita , tedy aktivita, která se vyskytuje na vrcholu, je měněna podle akcí uživatele. Pokud je potřeba je také možné toto chování pozměnit. Samotné aktivity mouhou také přecházet nebo se rozhodovat, kterému zásobníku budou přiřazeny. Toto chování ovlivňuje nastavení „flagů“ a atributů danné aktivity.

- Správa úkolů

  Vzhledem k tomu že android je určen pro mobilní zařízení je třeba nějak řídit množství úkolů na pozadí, jinak by mohla být značně omezena rychlost systému a také výdrž zařízení na baterii. Proto je definována množina druhů chování pro nakládání s aplikacemi, které byli přesunuty na pozadí. Standartně android ukončí všechny aktivity, které nebyly delší dobu použity a nejsou na vrcholu zásobníku daného úkolu. Tento model je opět modifikovatelný od hranice, kdy se uchovají všechny aktivity neustále po případ, kdy jsou všechny zahozeny, výjma kořenové, i po pouhém opuštění aplikace a následným návratem.

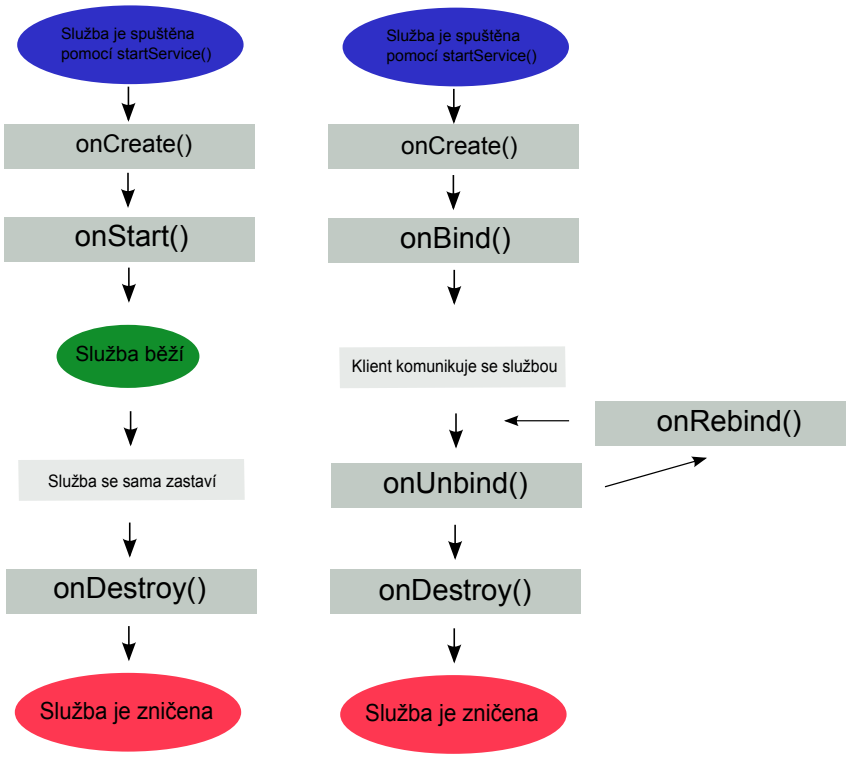
Životní cykly:

Aktivita: Její životní cyklus se dá dělit do tří stavů podle současného stavu aktivity vzhledem k systému. V prvním z nich je instance běžící nebo aktivní, z pohledu uživatele je plně viditelná a je nastavená pro přijmutí jeho příkazů, tedy má „focus“. Tento stav je uvozen voláním „onResume“ a ukončen „onPauzed“. V druhém je aktivita pozastavena, tedy systém neustále uchovává její data, ale má možnost je v případě akutního nedostatku paměti uvolnit a aplikaci tím ukončit. Z pohledu uživatele je aplikace stále viditelná, ale nemá již „focus“, tedy není úplně na popředí a je částečně překryta aktivní aplikací. Tento cyklus se odehrává mezi voláními „onStart“ a „onStop“. Poslední možnost nastává, když je aplikace zastavena. V tomto případě jsou neustále uchovávána její data, ale pro systém má už nižší prioritu, proto bývá ukončena při nedostatku paměti dříve než aktivita, která je pouze pozastavena. Pokud dojde k tomuto stavu, tak uživatel již není schopen s aplikací komunikovat a ani ji vidět na obrazovce, dokud se nevrátí alespoň do „pozastaveno“, nebo „běžící“ čí „aktivní“. Toto chování je realizováno mezi voláním „onCreate“ a „onDestroy“. V modelu zobrazeném níže je vidět ještě volání „onRestart“, které se využívá pouze po „onStop“ a je vždy následováno „onStart“. Pokud dojde k zničení aktivity, nemusí to ještě nutně znamenat ztrátu dat pro uživatele. Pro programátora jsou připraveny volání „onSavaInstaceState“ a „onRestoreInstanceState“, které jsou využity systémem tehdy, když nastane možnost zničení aplikace. Tvůrce pak dostává možnost doimplementovat záložní rutiny.



Obrázek 2.1: Životní cyklus aktivity.

Služba: V případě služby je množina stavů cyklu jednoduší, než u aktivity. Existuje zde pouze možnost, kdy je běžící nebo ne, a danný model chování se liší podle toho, jestli ji ovládáme přímo, nebo pomocí nějakého rozhraní. Pokud potřebujeme službu spustit, je třeba ji vytvořit a následně spustit. V případě jejího zastavení se pak přístup liší podle výše zmíněného druhu kontroly, kdy při přímém přístupu se služba je schopna zastavit sama, naproti tomu při vzdáleném je třeba se odpojit a následně službu manuálně zrušit.



Obrázek 2.2: Životní cyklus služby.

Broadcast receiver: Tato komponenta má pouze dva stavy a to aktivní a neaktivní. Obojí je iniciováno funkcí příjímání zpráv, která svoji aktivitou určuje stav samotného receiveru. Pokud je komponenta aktivní, není umožněné systému proces ukončit.

Funkce cyklů z pohledu správy paměti a procesů: Android operuje s pamětí způsobem, který by měl minimálně ovlivnit práci uživatele daného zařízení. Proto implementuje strukturu hierarchie, podle které značí prioritu procesů a rozhoduje, který bude ukončen a který ne. Důležitost procesu se dále určuje podle jeho obsahu. Tedy rozlišujeme pro tyto účely procesy na popředí, viditelné procesy, procesy hostující služby, procesy na pozadí a

prázdné procesy. Pro ty které se vyskytují na popředí je přidělena maximální priorita, proto musí obsahovat běžící aktivitu, aktivní službu, která je využívána výše zmíněnou aktivitou a broadcast receiver, který je aktivní. Nižší priorita je následně přidělena procesu, který obsahuje pozatavenou aktivitu a nějakou službu k ní přidělenou. Třetí v pořadí je proces obsahující službu, která není vázána k aktivitě, ale provádí nějakou uživatelsky důležitou činnost, která je momentálně aktuální. Předposlední priorita platí pro procesy na pozadí, které přímo neovlivní práci uživatele, a proto mohou byt ukončeny při nedostaku paměti. Jelikož je těchto procesů větší množství je pod platformou implemetová LRU seznam, který je identifikuje a tedy přiděluje další prioritu pro ukončení. Seznam je seřazen podle data posledního využití uživatelem. Nejnižší prioritu v celém systému pak mají prázdé procesy, které neobsahují žádnou aktivní komponentu, proto slouží jen pro funkci cache, tedy optimalizace rychlosti spuštění. Systém se následně rozhoduje jestli daný proces ukončit podle výkonu systému jako celku. Předešlé informace čerpány z [14].

Kapitola 3

Techonologie zobrazení, OpenGL ES

Pohled na OpenGL ES, stejně jako na OpenGL lze brát z několika úhlů. Tedy z pohledu vývojáře je to sada funkcí pro reprezentaci grafických objektů, v roli implementátora je to sada funkcí ovlivňující práci hardwaru cílové platformy a z pohledu tvůrce stavový stroj, který řídí množinu vykreslovacích operací. Přístup OpenGL je tedy vrstvový, to umožňuje kooperaci na úrovni modelu client - server, což má určité výhody při výpočtu náročnějších úloh. Samotné kreslení se provádí do frame bufferů, přičemž OpenGL pracuje se dvěma buffery. Jeden je aplikační a druhý je vytvořen okenním systémem platformy. Přičemž GL může buffer vlastněný tímto systémem ovlivňovat,ale sám je v plné režii platformy, proto konečné slovo při vykreslování má cílový okenní systém. Technologie je designována jako multiplatformní, proto je použitelná téměř na jakékoliv technologii a pod jakýmkoliv jazykem. Nicméně nejvhodnější je použití jazyků C a C++, kde napojení na technologii je již popisováno ve standartu.

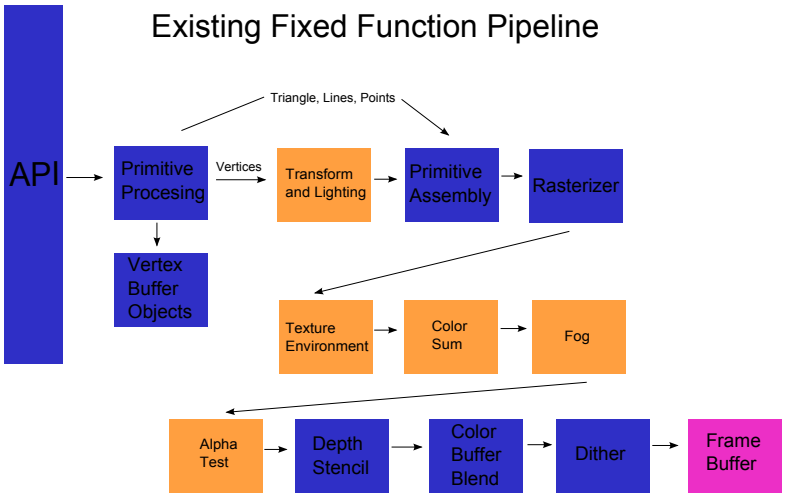
Technologie OpenGL ES vychází z desktopového standartu OpenGL a je upravena pro použití na mobilních platformách. OpenGL ES muselo být značně zjednodušeno, kvůli limitaci cílových zařízení, nicméně v současné době je již situace lepší než v předešlých letech, proto následující verze jsou čím dál bohatší na funkce dostupné programátorovi. První verze standartu byla značena jako 1.0, kdy v současné době je k dispozici již 2.1, proto je tu široká škála implementací i pro zařízení s menším hardwarovým potenciálem. Toto je také filozofie tvůrců technologie, kdy je celý systém navržen tak, aby byl minimální zátěži pro paměť a s tím spojenou baterii mobilního zařízení. Směrem k aplikaci na trhu s velkým rozptylem zařízení jsou také designovány výpočty s možností běhu jak na grafickém hardwaru, tak v podobě softwaru bez asistence dedikovaných zařízení. Standart také podporuje mechanismus rozšíření, který je schopen reagovat na nové trendy v oblasti hardware. OpenGL ES také definuje množinu profilů, které specifikují potřebné komponenty platfomy pro použití standartu. Existují dva profily. První je cílen na oblast handheldů, telefonů a větších herních zařízení, který se vyznačuje maximální podporou pro 3D a minimálními nároky na hardware. Druhý profil směřuje na trh letectví a automobilového průmyslu, kde je primární požadavek na spolehlivost systému. V neposlední řadě je třeba zmínit, že standart je dostupný zdarma.

3.1 Popis standartu

Technologie se dělí na dvě hlavní větve. První reprezentuje verzi 1.X, která je primárně designovaná pro zařízení s fixní desetinnou čárkou, tedy zařízení obsahující levnější variatu hardware, kde bude převládat spíše softwarová realizace výpočtu. Tento fakt platí především pro verzi 1.0, kdy od verze 1.1 se začíná již více počítat s hardwarovým zázemím mobilního zařízení, ale je zpětně kompatibilní s 1.0. Druhá větev představuje 2.X, která je určena pro zařízení obsahující 3D pipeline, tedy produkty, které disponují hardwarovou podporou prostorového zobrazení. Součástí OpenGL ES je také vrstva EGL, která je realizována jako interface mezi vyktreslovací technologií a platfotmě specifickým okenním systémem.

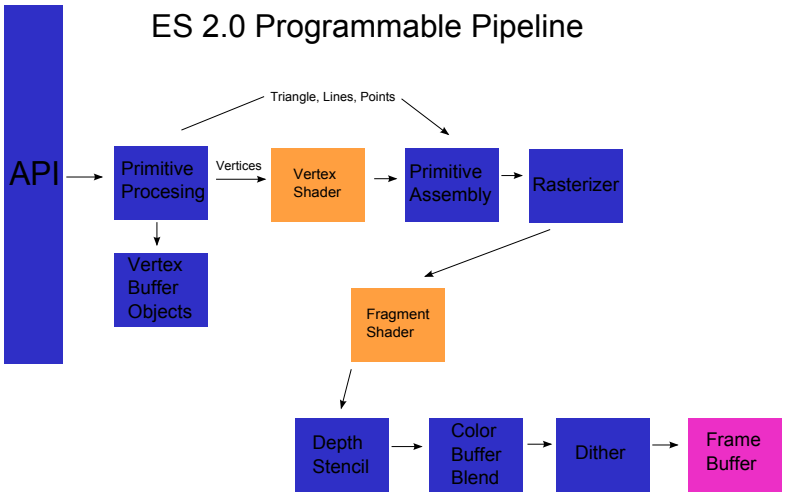
Větev 1.X: Verze 1.0 vychází z desktopového OpenGL 1.3. Standart obsahuje implementaci pro zpracování geometrických úloh, rasterizaci objektů, mapování, filtrování a jiné operace s texturami, zpracování částic, práce s frame bufferem.

Verze 1.1 oproti 1.0 se inspiruje u PC verze 1.5. V této verzi je podpora rozšířena o nové prvky. Jako jsou například objekty pro ukládání elementů do paměti, použitých v renderování scény. Automatické generování mipmap objektů. V této revizi již není nutné tuto funkcionalitu implementovat v programu, je možné ji nechat na dedikované zařízení. Podpora zpracování multi textur a implementace logiky zpracování jejich kombinací. Implementace palet matic, které jsou využity pro plynulou animaci složitějších objektů. Zahrnutí renderování pouze viditelných částí scény pro zvýšení výkonnosti aplikací. Podpora realističtějšího vykreslování častic, používá se především u scén s efekty. Implementace dotazů jak na různé prvky platformy, tak na její datové elementy. Cílem je možnost návrhu aplikace pomocí vrstvového modelu. Podpora rychlého vykreslování částí textur do specifikovaných části obrazovky, používá se při renderingu pozadí, fontů, případně texturovaných prvků v hrách. Plus další přídavky různých strukturovaných typů. K této verzi byla ještě vydáno rozšíření OpenGL ES 1.1 Extension Pack, které rozšiřuje programtorovi možnosti v oblastech jako: práce s texturami, maticemi, úprava frame bufferu



Obrázek 3.1: OpenGL ES 1.0 pipeline.

Větev 2.X: Tato větev podporuje především high-tech zařízení, které podporují zpracování pomocí 3D pipeline. Technologie umožnuje také tvorbu shaderů pomocí OpenGL ES Shading Language. Oproti starším verzím již není podporována transformace ve fixní desetinné čárce a částicový pipeline. Tento postup je právě nahrazen shadery.



Obrázek 3.2: OpenGL ES 2.0 pipeline.

Předešlé informace dle [13].

3.2 Zobrazování 3D prostorů v OpenGL ES

Poloha tělesa v prostoru je v OpenGL ES určována pomocí souřadnic v souřadnicové soustavě. Soustava souřadnic je soustava elementárních hodnot , umožňujících určovat polohu objektu ve zvolené vztažné soustavě, nebo také vzájemně jednoznačné zobrazení mezi množinou bodů n-rozměrného prostoru a uspořádanou n-ticí čisel.Touto n-ticí čísel jsou myšleny jednotlivé souřadné osy. Proto je potřeba pro polohu objektu v n-rozměrném prostoru použít právě n-bodů. Samotné souřadnice se dají definovat jako skupina čísel, které určují polohu v daném systému souřadnic. Ty mají pak různý význam podle toho, jaký typ souřadné soustavy zvolíme. Danou soustavu charakterizuje volba počátku, typ jednotlivých souřadných os, tedy jejich směr, počet a charakter, dále pak jednotky celého systému pomocí nichž se určuje hodnota samotných souřadnic. Mezi nejznámější druhy souřadných systémů patří: kartézská, polární, kruhová, válcová. Charkteristiku jednotlivých os pak můžeme dělit podle jejich počáteční orientace, tedy pokud jsou souřadné osy na sebe kolmé, tak se celá soustava nazývá ortogonální. Dalším znakem je tvar samotných os. Pokud jsou všechny osy přímkami, pak se mluví o přímočaré soustavě, pokud se dají nazvat křivkami, pak se o danné soustavě hovoří jako o křivočaré. Příklad přímočaré soustavy souřadnic je například kartézská soustava, křivočarou nám může reprezentovat třeba polární soustava. Pro naše použití je především důležitá reprezentace pohybu v systému. Vzhledem k tomu, že v OpenGL pracujeme s 2D a 3D prostory a v rámci této bakalářské práce výhradně v 3D, tak požadujeme popis v rámci soustavy prostorových souřadnic. Ta je definovaná jako soustava souřadnic v trojrozměrném prostoru. Mezi nejběžnější pak patří : Afinní, Kartézská,

Sférická, Válcová soustava souřadnic.

Afinní soustava souřadnic:
Tuto soustavu reprezentuje trojice os, která jsou charakterizovány jako různoběžky. Ty se pak protínají v jednom bodě, tedy v počátku. Současně platí, že nemusí být na sebe kolmé. Určení samotných souřadnic pak probíhá pomocí vedení rovnoběžek s jednotlivými souřadnicovými rovinami. Hodnotu složek souřadnice bodu pak určí protnutí dané rovnoběžky se souřadnou osou. Například při určení x-ové složky se vede rovnoběžka s rovinou yz a protnutí s osou x značí hodnotu složky x souřadnice.

Kartézská soustava souřadnic:
Je pouze speciálním případem afinní soustavy, kdy platí, že souřadné osy jsou vždy na sebe kolmé.

Sférická soustava souřadnic:
Představuje rozšíření polárních souřadnic. Souřadnice je určena trojicí, kde r značí vzdálenost od počátku soustavy, pak zde vystupuje úhel mezi průvodičem a osou x a úhel mezi osou y a dříve zmíněným průvodičem.

Válcová soustava souřadnic: Jeví se jako sférická, je opět rozšířením polárních souřadnic. Určení polohy představuje trojice r, tedy vzdálenost od osy z, pak úhel průmětu průvodiče bodu do roviny xy k nejčastěji ose x a z určuje polohu na z-tové ose.
Předešlé informace dle [12].

OpenGL ES pracuje s klasickou kartézskou soustavu souřadnic. Kde každý bod je určen trojicí skalárů, které značí pozici v 3D vesmíru. V rámci zobrazování a práce s tímto prostorem, je potřeba často tyto body transformovat do nových pozic. Těmto operací se říká transformace. OpenGL ES je aplikuje na osy souřadnic tudíž dochází k posunu celého prostoru a objektů v něm. Samotné souřadnice těles ve své matematické podobě nejsou příliš vhodné pro 3D transformace, proto byly zavedeny souřadnice homogenní. V tomto tvaru mají podobu čtveřice, která se skládá z kartezských souřadnic plus určitého váhového elementu.

Definice 3.2.1 *Pro homogenní souřadnice platí:* $\mathbf{S} = [x, y, z]$
$X = \frac{x}{\omega}$ $Y = \frac{y}{\omega}$ $Y = \frac{z}{\omega}$
Pak:
$\mathbf{S_h} = [\omega X, \omega Y, \omega Z, \omega]$

Důvod zavedení těchto souřadnic je prostý. Díky nim je možné provádět všechny transformace stejným způsobem, tedy je možné pokaždé použít operaci násobení. V případě klasických by bylo třeba rozlišovat mezi operací posuvu, kdy by bylo třeba aplikovat sčítání a ostatními, které jdou realizovat pomocí násobení. Pak můžeme zavést množinu transformačních matic pro základní transformace, které můžeme snadno využít pro naše zobrazení. Jejich základní podoby jsou vypsány níže.

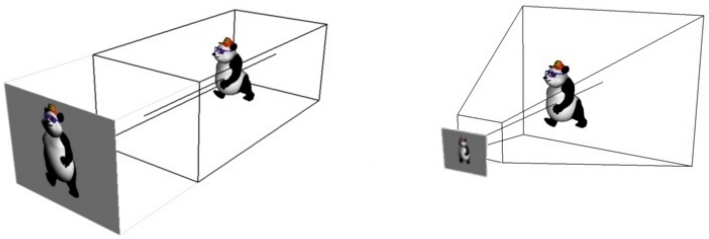
$$\textbf{Rotace:} \quad \mathbf{R_z} = \begin{pmatrix} cos(\alpha) & sin(\alpha) & 0 & 0 \\ -sin(\alpha) & cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.2.1}$$

$$\textbf{Posunutí:} \quad \mathbf{R} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ d_x & d_y & d_z & 1 \end{pmatrix} \tag{3.2.2}$$

$$\textbf{Měřítko:} \quad \mathbf{R} = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_y & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.2.3}$$

$$\textbf{Zkosení:} \quad \mathbf{R} = \begin{pmatrix} 1 & S_(hy) & S_(hz) & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.2.4}$$

Pro reprezentaci scény je také důležitá projekce scény. OpenGL ES umožňuje pro zobrazení 3D grafiky použít buď Orthografickou, nebo Perspektivní projekci. Každá scéna v OpenGL ES je určena jistým oběmem tělesa, které určuje danou projekci. Kde na jedné z hraničních ploch je objekt kamery, která celou scénu snímá. V případě ortografické je to jednoduchý kvádr a pro perspektivu je to komolý jehlan, kde kamera je obvykle umístěna na stěně tak,aby směřovala k větší z protilehlých stěn. V případě kvádru je jedno z které z definovaných stěn bude kamera snímat scénu, protože nedochází během posuvu tělesa v prostoru k žádné deformaci. Tato vlastnost vyplývá z geometrické podstaty kvádru. Naproti tomu vypadá situace u hranolu jinak, pokud máme kameru nastavenou pohledem na větší stěnu, tak při přibližování se obraz zvětšuje a naopak. Více ukazují následující obrázky:

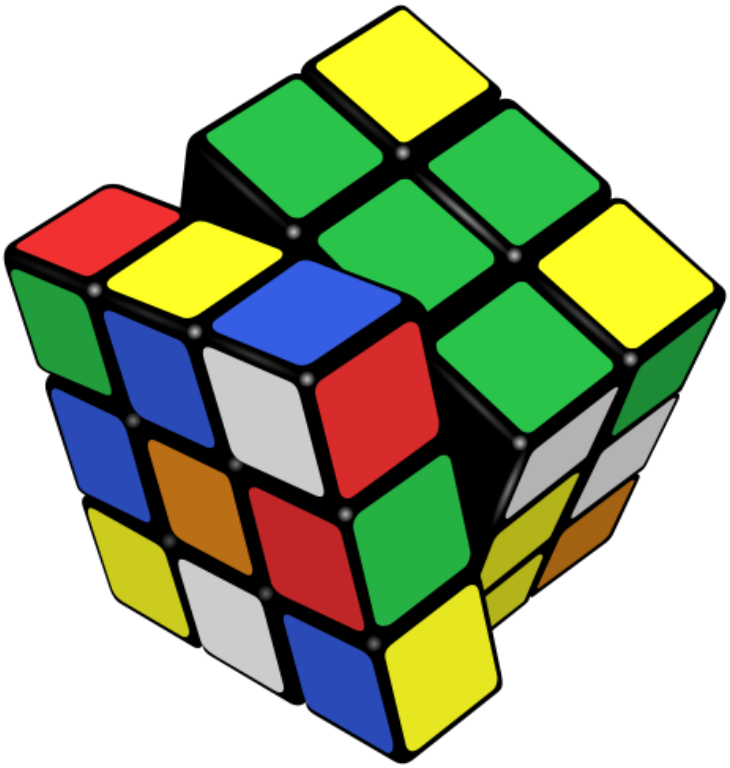


Obrázek 3.3: Ortografická a perspektivní projekce. Dle [9]

Kapitola 4

Umělá inteligence

K samotné realizaci Rubikovy kostky jsem se rozhodl přidat umělou inteligenci, která řeší posloupnost kroků vedoucích k poskládání dané kostky. Podle mého mínění tím aplikace získá značně na atraktivitě, vzhledem k tomu, že v současné době již tento hlavolam neprožívá zlatou éru, jako tomu bylo v osmdesátých letech.



Obrázek 4.1: Rubikova Kostka.

4.1 Popis Rubikovy kostky

Rubikova kostka je objekt tvaru krychle, která má na každé straně 9 barvou odlišených čtverců. Na první pohled se jeví jako poskládaná z 27 menších kostek, ale skutečná realizace je odlišná a vyhrazuje její teoretická specifika. Reálně se skládá z šesti středů, dvanácti hran a osmi rohů. Tyto části umožnují manipulaci s jejími řadami a sloupci, pokaždé o 90 stupňů, nebo násobek této hodnoty. Jediná statická část kostky jsou středy, které díky této vlastnosti určují barvu jednotlivých stěn. Sedm rohů může být napozicováno nezávisle, kde současně osmý roh je závislý na předchozích sedmi. Toto produkuje 40320 kombinací pro nezávislé rohy plus 2187 pro závislý. Další stavový prostor tvoří pak jednotlivé hrany. Pro jedenáct z nich, které jsou na sobě nezávislé existuje 239,500,800 kombinací. Poslední hrana je opět závislá na předchozích jedenácti a tvoří 2048 stavů. Tyto všechny prvky tvoří množinu kombinací kostky, těch je následně 43,252,003,274,489,856,000. Toto číslo platí pro hlavolam, který byl manipulován pomocí standartních mechanismů určených stavbou kostky. Pokud by byl rozložen, tak je výsledná množina přibližně desetkrát mohutnější.

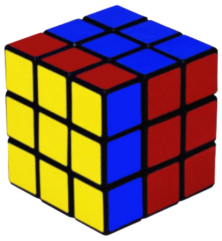
Počátkem problému Rubikovy kostky je stav, kdy jsou barvy na stěnách rozházeny pomocí jednotlivých operací s řadami a sloupci. Vyřešená kostka pak má na každé ze stěn pouze jednu barvu. Možných řešení kostky je velké množství. Každý postup obsahuje v sobě sadu algoritmů pro přechod mezi různými stavy kostky. Každý z nich ji přibližuje k cílenému řešení. Jednotlivé algoritmy jsou použity v rozličných etapách řešení problému, tedy prakticky řečeno, kdy bude který použit. Toto určují jejich vlastnosti, jelikož jednotlivé algoritmy mohou mít různé vedlejší účinky. Takové jsou následně použity v brzkých fázích řešení, kdy nemají takový vliv. Některé algoritmy následně popíšu v sekci níže. Velikost standartní kostky 3x3x3 je 5,7 cm. Krom této varianty existují také pokročilejší a jednoduší varianty. Nejsnazší je kostka 2x2x2, dále existují obtížnější jako: 4x4x4, 5x5x5, 6x6x6 a 7x7x7. Na trhu lze najít také hlavolamy, které se neomezují na podobu krychle a je jich velké množství. Krom těchto fyzických variant jsou dostupné také programové implementace kostek ve více rozměrech než ve třech.

Historie kostky: Hlavolam byl vynalezen v roce 1974 Ernő Rubikem. Vynálezce se zabývá sochařstvím a vyučuje jako profesor architektury na vysoké škole Uměleckoprůmyslové v Budapešti. V letech, kdy vytvořil Rubikovu kostku, byl zaměstnán jako architekt. Kostka kterou původně nazval kouzelná kostka, měla sloužit jako učební pomůcka studentům pro lepši pochopení prostorových objektů. Rubik nebyl původně úplně první, kdo takový hlavolam vynalezl. Podobnou kostku stvořil již v roce 1970 Larry Nichols, který postavil kostku se stejným herním účelem, ale o rozměrech 2x2x2. Samotná fyzická realizace byla ovšem postavena na principu magnetů, tedy se lišila od Rubikovy verze. Ta vešla v celosvětové povědomí počátkem roku 1980, kdy byla prezentována na veletrhu hraček v Londýně, Paříži, Norimberku a New Yorku. V tomto roce si ji nechal i Ernő Rubik mezinárodně patentovat. Dva roky poté byl ale soudně napaden ze strany Larryho Nicholse, kvůli původu nápadu hlavolamu. Žaloba byla uznána pouze v případě kostky 2x2x2.

4.2 Algoritmy

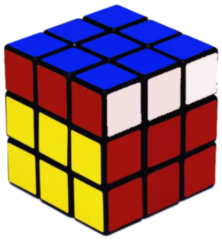
Vlastních postupů pro řešení problému kostky je poměrně velké množství. Obecně je ale lze rozdělit podle jejich plánovaného použití. Existují algoritmy určené primárně pro člověka, tedy v tomto případě se vyznačují menší náročností na paměť, nicméně na druhou stranu není možné pomocí nich dosáhnout lepších, tedy kratších řešení. Druhou variantou jsou postupy určené primárně pro počítač, tedy podstatně náročnější, ale generující značně kratší posloupnost kroků k řešení. Takové algoritmy jsou zamýšleny ke generování optimálních řešení. Pokud se chceme bavit konkrétněji, je třeba zavést způsob popisu kostky.

Nejběžněji se používá značení podle stěn, tedy šestice: F(a),B(b),L(d),R(e),U(c),D(f). Jednotlivá písmena popisují orientaci stěny, tedy Front, Back, Left, Right, Up, Down. Tímto jsou stěny identifikovány, ale v popisu řešení se již využívají k pojmenování jednotlivých operací s kostkou, kde například Front znamená rotací přední stěny o devadesát stupňů ve směru hodinových ručiček, kde se orientujeme tak, že tuto operaci bereme při pohledu na danou stěnu. Vycházíme ze stavu, kdy je stěna Up orientována vzhůru a díváme se na Front. Pak při operaci s dalšími provedeme rotaci kolem osy y o násobek devadesáti stupňů. V případě spodní a vrchni stěny rotujeme z výchozího stavu podle x opět o násobek devadesáti stupňů. K těmto reprezentacím ještě přibývá rotace o stoosmdesát stupňů, která je značena dvojicí: identifikace stěny a číslem dvě. Tedy například: F2,D2, a tak dále. Poslední variantou pohybu je otočení proti směru hodinových ručiček. Ten se značí dvojicí identifikace stěny a apostrof. Například F', D'. V rámci jednotlivých algoritmů je také potřeba definovat popis menších částí kostky, tedy jednotlivých kostiček. Princip popisu se opírá opět o barvy jednotlivých stěn, proto pro hrany platí například dvojice UL, ta značí kostičku v horní vrstvě v levém sloupci uprostřed, zbytek je analogicky. Následně pro rohy je situace stejná, akorát přibývá identifikace další barvy stěny. Například levý horní přední roh se značí jako ULF.

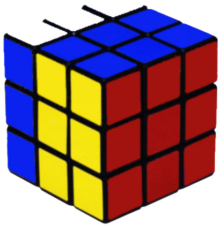


(a) Rotace Front

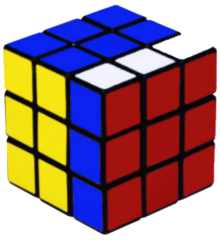
(b) Rotace Back



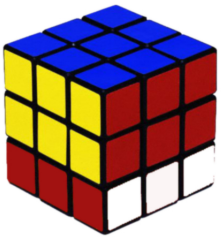
(c) Rotace Up



(d) Rotace Down



(e) Rotace Down



(f) Rotace Down

Algoritmy určené lidskému uživateli: Mezi nejznámější patří vrstvový algoritmus vynalezený Davidem Singmasterem v roce 1981. Jeho princip spočívá v řešení kostky postupně podle jejich vrstev.

Vrstvový algoritmus: Kdy se postupuje od vrchní(v našem značení je to Up) a končí se u spodní(tedy Down). Při praktickém postupu je v první etapě úkolem dostat horní vrstvu do stavu, kdy se na ni nachází kříž z jedné barvy. Plus tato barva je spojena s barvou středů na přilehlých stranách, tedy pokud je ve středu přilehlé strany modrá barva, tak ve výsledné fázi této etapy musí být i na hraně spojené s křížem. Na konci této fáze by měli být správně nastaveny čtyři horní hrany, potom je možné přejít na druhou. Zde je úkolem řešitele poskládát tři horní rohy, kde čtvrtý je později použit pro zjednodušení operací s kostkou. Po jejím dokončení máme vyřešenou jednu stěnu kostky. Třetí fáze má za úkol správně nastavit hrany na prostřední vrstvě opět vyjma jediné a to té, která leží pod volným rohem z horní vrstvy. Po ní následuje fáze čtvrtá, kde je potřeba vyřešit zbylé hrany. V současném stavu je jich pět, kdy prvně se řeší spodní tři a pak poslední vespod a zbývající ze středu. Následuje předposlední část a to je nastavení zbývajících rohů kostky. Po jejím úspěšném absolvování zbývá už pouze nastavit správnou orientaci rohů a kostka je vyřešena.

Krom tohoto algoritmu exitují i rychlejší varianty. Sem patří například Fridrichova metoda, která spočívá ve spojování fázi předchozího postupu. Konkrétně to jsou fáze dvě a tři a na poslední vrstvu připadají pouze dvě fáze. Jako další jsou například metody ZB,ZZ,VH. Tyto postupy jsou rychlejší než prvně zmíněný, ale jsou náročnější na řešitele. Všechny předešlé metody jsou zařazovány do skupiny vrstvové, což vyplývá z jejich principu. Dále pak existují ještě blokové a postupy, které řeší jako první rohy.

Předešlé informace dle [7].

Algoritmy určené počítači: Tyto postupy produkují značně kratší sekvenci kroků, než ty předešlé, ale mají velké nároky na řešitele problému, proto jsou implementovány na počítači. Sem se dají zařadit následující algoritmy: Thistlethwaiteruv, Kociembuv, Korfuv.

Thistlethwaiteruv algortimus: Je založen na rozdělení množiny možných kombinací kostky na časti. Toto rozdělení je provedeno pomocí omezení povolených operací. Ty jsou určeny následujícími skupinami:

$$G0 =< L, R, F, B, U, D > \tag{4.2.1}$$

$$G1 =< L, R, F, B, U2, D2 > \tag{4.2.2}$$

$$G2 =< L, R, F2, B2, U2, D2 > \tag{4.2.3}$$

$$G3 =< L2, R2, F2, B2, U2, D2 > \tag{4.2.4}$$

$$G4 = I \tag{4.2.5}$$

Zde G0 představuje celý stavový prostor kostky. Řešení je prováděno pomocí přechodu mezi množinami, až se dojde k poslední, která obsahuje pouze vyřešenou kostku.

Kociembuv algortimus: Tento postup představuje vylepšení předchozího, tak že redukuje seznam množin nutných k vyřešení kostky. Prohledávání množin je realizováno pomocí obdoby IDA*. Algoritmus řeší kostku obvykle v 21 krocích oproti 52 u Thistlethwaiterova.

$$G0 =< L, R, F, B, U, D > \tag{4.2.6}$$

$$G1 =< L, R, F2, B2, U2, D2 > \tag{4.2.7}$$

$$G2 = I \tag{4.2.8}$$

Sám Kociemba ho nazývá dvoufázový, i když na internetu se používá pro jeho reprezentaci spíše pojmenování podle autora. Algoritmus skutečne probíhá ve dvou fázích, kdy v první hledá přechod do G1 a v druhé se obnoví kostka za použití pouze povolených pohybů v ní. Dle [8]

Kapitola 5

Řešení uživatelského rozhraní

V této části se již dostanu k praktickému řešení bakalářské práce. Jako první problém se vyskytlo ovládání kostky, které muselo být dostatečně pohodlné a intuitivní pro uživatele. Manipulací kostky je prakticky tráven téměř veškerý herní čas a jelikož uživatelé mohou mít rozdílné nároky na ovládání, rozhodl jsem se také zahrnout více druhů. V aplikaci je tedy podpora jak pro inerciální jednotku zařízení, tak i pro klasickou interakci pomocí tahů na dotykovém displeji. Protože ovládání kostky pomocí polohování celého zařízení se ukázalo poměrně snadné na implementaci, proto v kapitolách níže uvádím především problémy spojené s interakcí přes dotykový displej.

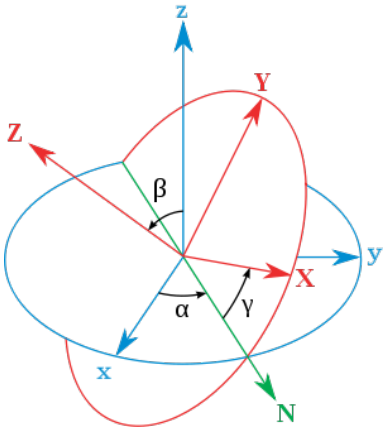
5.1 Možnosti ovládání kostky

Kromě vhodností ovládání jednotlivých metod je třeba řešit převod mezi vstupem od uživatele, tedy 2D prostor pohybu na obrazovce, v kterém zobrazujeme daný objekt. Tento převod realizují pak následující metody. Ovládat tento objekt pomocí dotykového displeje lze pomocí více možností. První možností jsou Eulerovy úhly, které lze implementovat pomocí před definované funkce glRotatef:

Eulerovy úhly: [1]

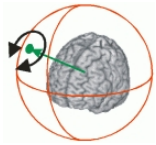
Rotace je tedy definována pomocí osy a úhlu. Je možné je také skládat za sebe a tedy teoreticky lze dosáhnout požadovaného aplikačního chování. Problém ovšem nastane právě při jejich skládání. Kdy tvůrce musí počítat s vedlejšími efekty jako jsou například inverzní rotace ke složené rotaci, kdy je třeba provést operaci nejen pomocí opačného úhlu, ale také pomocí opačného pořadí os. Nicméně existuje tu zásadnější problém jménem „gimbal lock“, který vzniká právě z daného skládání rotací. Tedy osy se při jednotlivých operacích se systémem navzájem ovlivní a proto již další rotace nejsou intuitivní, protože uživatel nepočítá s tímto problémem. Jediná výhoda je pro programátora, že implementace je velmi snadná.

[1] Leonhard Paul Euler se narodil 15. dubna 1707 v Basileji a zemřel 18. září 1783 v Petrohradu. Je považován za nejvýznamějšího švýcarského matematika, zabýval se ale také fyzikou. Mezi jeho dílo patří 60 až 80 svazků obsahující jeho spisy. Eulerův přínos je především v oblasti diferenciálních počtů a teorie grafů. Zavedl také nové pojmy v matematické analýze a významné byly také jeho práce v mechanice, optice a astronomii.Dle [6]



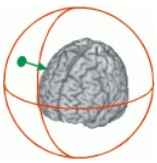
Obrázek 5.1: Eulerovy úhly. Převzato z [4]

Rotace pomocí vektoru: Další možností je rotace pomocí vektoru, kterou plně implementuje již zmíněná funkce glRotatef, která přijímá argumenty určující jednotkový vektor rotace a úhel. Vektor se skládá ze složek x,y,z. Výhodou je že daný systém umí popsat všechny pohledy na objekt, ale není dostatečně intuitivní pro uživatele. Jako daleko vhodnější se jeví následující metoda.



Obrázek 5.2: Rotace pomocí vektoru a úhlu. Dle [2]

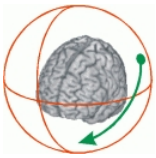
Ovládání pomocí elevace a azimutu: Tato metoda má výhodu intuitivnosti jednotlivých proměných, které určují požadovaný pohled na objekt. Nastavení elevace a azimutu vychází přímo z životní praxe. Nicméně tento přístup nedovoluje zobrazení všech možných pohledů na kostku, a proto jsem jej v závěrečné implementaci zavrhl.



Obrázek 5.3: Rotace pomocí elevace a azimutu. Dle [2]

ArcBall rotace: Vzhledem k zamýšlenému druhu ovládání, tedy manipulace s kostkou pomocí dotykového displeje, je tento přístup pravděpodobně nejlepší vzhledem k pohodlí uživatele. Princip totiž umožňuje prakticky objekt na displeji „uchopit“ a orotovat s ním cestou zvolenou pomocí myši. Dochází totiž k vytvoření specifické osy pro každou rotaci. Tu specifikuje přímo uživatel operací „uchopení“ objektu a táhnutím po displeji, z čehož se také určuje úhel rotace. Přístup využívající logiky Arcballu ma výhody například eliminující

potřebu inverzního skládání rotací pokud se rozhodneme vrátit se z orotovaného stavu zpět do počátku. Prakticky jde o tvorbu jistého virtuálního trackballu, který je znám z použití na starších mobilních zařízeních, kde ovšem typicky sloužil k pohybu myši, nikoliv k rotaci prostorových objektů. My tuto myšlenku využijeme k manipulaci se sférou, kde tedy bude plnit oproti své hardwarové implementaci přesně opačnou funkci, tedy převod z vektoru zadaného ve 2D prostoru do vektoru rotace v 3D prostoru. Tento princip využívám ve své aplikaci a věnoval jsem mu samostaný prostor v sekci níže, kde je podrobně vysvětlen.



Obrázek 5.4: Rotace pomocí ArcBallu. Dle [2]

Metody nesouvisející s dotykovým displejem: Vzhledem k tomu že jsem uživateli chtěl nabídnou více možností ovládání mé aplikace, nemohl jsem zůstat pouze u jedné možnosti. Proto jsem zahrnul také manipulaci s kostkou pomocí inerciální jednotky, kde stačilo využít standartizované api přímo od Google, které je založené na principu instance managera, který již provádí komunikaci s hardwarem zařízení a aplikaci předává finální data. Jediný problém, který bylo třeba odladit, byla funkcionalita tohoto manageru, který funguje na principu událostí, které se ovšem můžou skládát historicky do fronty, pokud je aplikační vlákno momentálně zaneprázdněné. Tento jev je pak pro uživatele značně kontra-produktivní, jelikož si přeje provádět pouze aktuálně zadané manipulace. Řešení spočívalo v časové kontrole přijmutí jednotlivých událostí. Ty se pak následně provedou v programu a po nich jsem vložil krátkou pauzu v běhu programu, která měla pouze za úkol zahodit zbývající události inerciální jednotky ve frontě. Tu jsem nastavil na desetinu sekundy, tedy dobu, kterou uživatel nepostřehne a odezva aplikace na jeho požadavky se značně zlepšila.

Nyní také nastala otázka, jestli je dobré tyto přístupy kombinovat. Tedy ovládání přes displej a pomocí inerciální jednotky. Došel jsem k názoru, že nejlepší bude dát uživateli na výběr, tedy možnosti, že si může vybrat mezi čistým ovládáním pouze pomocí displeje, nebo pouze inerciální jednotky, a jako poslední kombinace obou. Ta spočívá v podpoře výše zmíněné jednotky a současně dotykového ovládání, buď ve standartní verzi, nebo v upravené. Tato úprava spočívá v tom, že manipulování kostkou pomocí displeje slouží jen k orientaci na kostce, tedy prakticky řečeno, pouze k prohlédnutí, jaké barvy jsou aktuálně na jaké straně. Uživatel kostku pouze „uchopí“ a podívá na stav na jednotlivých stěnách a po jejím „puštění“, tedy uvolnění dotyku na displeji, se kostka vrátí pomocí animace do výchozího stavu odkud uživatel začínal rotaci. Myslím si, že tento přístup má výhodu pro zařízení s menším displejem ve smyslu manipulace s jednotlivými řadami a sloupci, pokud se nacházíme ve stavu, kdy je k nám cílová stěna orientována jinak než kolmo na paprsek vyslaný ze snímací kamery.

5.2 ArcBall rotace

Jak již bylo zmíněno výše, tento přístup manipulace s 3D objekty poskytuje maximálně efektivní výsledky. Základní princip rotace je vytvoření sféry okolo rotovaného objektu.

Odtud se da odvodit, že tato metoda poskytuje nejvěrnější ovládání pro kouli a její podobným tvarům. Nicméně pro variantu koskty je také velmi postačující. Samotná rotace pak probíhá po zadání alespoň dvou bodů uživatelem, tedy situace kdy uživatel táhl prstem po displeji alespoň o jeden bod. V tomto případě se vytvoří ve sféře dva vektory spojující bod na ní a její střed. Ty mají pak dvojí funkci a to první, která určuje osu rotace a druhou, která určuje úhel, o který se má těleso otočit. Osa podle které se pohyb objektu realizuje se pak vytvoří jako vektor kolmý na vektory zadané uživatelem. Úhel je pak dán intuitivně úhlem mezi výše zmíněnými uživatelovými vektory. Takto je popsán princip Arcballu pomocí geometerie. Na tomto místě bychom tento přístup mohly zkombinovat v již známým přístupem pomocí osy a úhlu rotace, který byl zmíněn již výše a dosáhly bychom poměrně uspokojivého výsledku. V mé práce jsem se ale rozhodl využít možností quaternionů, které poskytují jisté výhody, především v plynulosti animace a také jejich implementace naproti tomu není až tak náročná. Proto se v následující části krátce rozepíši o quaternionech jako takových.

Quaterniony: Dají se definovat jako rozšíření komplexních čísel do čtyř rozměrů. Sama jsou částí obecnějšího celku hyperkomplexních čísel a jsou představovány čtveřicí, kde ta je definována trojicí, která značí imaginární část, a jedním členem, který značí reálnou část. V matematickém pojetí se pak zapisuje takto.

$$H = a * 1 + b * i + c * j + d * k \tag{5.2.1}$$

Komponenty a,b,c,d představují reálná čísla. Operace s nimi nejsou komutativní, ale jsou asociativní.

$$H_1 * H_2 \neq H_2 * H_1 \tag{5.2.2}$$

$$(H_1 * H_2) * H_3 = H_1 * (H_2 * H_3) \tag{5.2.3}$$

Základní rovnice, která definuje quaterniony pak zní.

$$i^2 = j^2 = k^2 = ijk = -1 \tag{5.2.4}$$

Mezi další vlastnosti quaterniony pak patří nasledující.

$$i * j = -j + i = k \tag{5.2.5}$$

$$j * k = -k + j = i \tag{5.2.6}$$

$$k * i = -i + k = j \tag{5.2.7}$$

V zobrazování trojdimenzionáchních objektů pak využíváme jiné reprezentace quaternionů. Tedy jako vektor o čtyřech členech, které jsou shodné jako reprezentace výše.

$$q = [w, x, y, z] \tag{5.2.8}$$

Předešlé informace dle [10]

Zde jsem zvolil jiné značení identických členů pro lepši orientaci v problematice. W představuje skalár, a x, y, z jsou imaginární složky. Pro naši potřebu budeme tyto komponenty identifikovat dále jinak a to x,y,z budou tvořit osu rotace, w pak bude značit úhel.

Samotný proces tvorby od teorie k praxi vypadá následovně. V programu na úplném začátku odchytávám události dotyku na displej, kde následně nastavuji první vektor určující požadovanou rotaci. Ten je vytvořen pomocí souřadnic v okně aplikace. Ty se ovšem musí převést na naší reprezentaci ve sféře, pomocí které rotujeme. Vzhledem k tomu, že jsem pro moji aplikaci zvolil poloměr sféry jedna, tak jsou převedeny na interval od mínus jedné do jedné, kde střed je v nule. Po tomto převodu je třeba zjistit polohu vektoru v systému, tedy jestli je již mimo sféru, nebo v ní. To se provede jednoduchým porovnáním délky vektoru a rozměrů sféry. Zde připomínám že vektor se vytvoří pomocí bodu zadaného uživatelem a středu systému, který má všechny složky rovny nule. V případě že vektor leží mimo sféru, provedeme jeho normalizaci a tím pádem pokud se pohybujeme mimo sféru, tak budeme tělesem rotovat pouze pomocí kružnice, tedy na nejkrajnějším okraji vzhledem k mapování z dvojdimenzionáního prostoru displejových souřadnic na trojdimenzionální, proto zde nastavíme složku z na nula. V případě polohy vektoru ve sféře je pak pouze třeba dopočítat opět souřadnici z, která je závislá na rozdílu délky vektoru a rozměru sféry. Zde nás zajímají jen kladné souřadnice, tedy ty které jsou na ploše naší imaginární sféry orientované směrem k naší kameře.

Tento postup pak aplikujeme pro další bod, který nám zadá uživatel a pomocí něho už můžeme vytvořit požadovaný quaternion. Jako první vezmeme dva vektory namapované na naši sféru a spočítáme pro ně osu rotace, tedy vektor, který je na ně kolmý. Jako poslední složka, která nám pro konstrukci quaternionu schází, je w, tedy reprezentace úhlu rotace. Tu následně získáme pomocí skalárního součinu vektorů zadaných uživatelem. Nyní máme již všechny složky a můžeme je přiřadit do třídy quaternionu a vytvořit reprezentaci naší rotace. Jako poslední krok, který zbývá provést, je převod tohoto quaternionu na rotační matici, kterou můžeme použít už přímo v OpenGL ES pro manipulaci se scénou. Tato matice je definována následujícím způsobem.

Matice pro rotaci pomocí quaternionů vypadá následovně:

$$\mathbf{R} = \begin{pmatrix} w^2 + x^2 - y^2 - z^2 & 2xy + 2wz & 2xz - 2wy & 0 \\ 2xy - 2wz & w^2 - x^2 + y^2 - z^2 & 2yz + 2wx & 0 \\ 2xz + 2wy & 2yz - 2wx & w^2 - x^2 - y^2 + z^2 & 0 \\ 0 & 0 & 0 & w^2 + x^2 + y^2 + z^2 \end{pmatrix} \tag{5.2.9}$$

Vzhledem k tomu, že pracujeme s jednotkovým vektorem, díky tomu, že jsme zvolili poloměr sféry jedna, tak platí:

$$w^2 + x^2 + y^2 + z^2 = 1 \tag{5.2.10}$$

Proto můžeme tuto matici zjednodušit na následující.

Po úpravě:

$$\mathbf{R} = \begin{pmatrix} 1-(2y^2+2z^2) & 2xy+2wz & 2xz-2wy & 0 \\ 2xy-2wz & 1-(2x^2+2z^2) & 2yz+2wx & 0 \\ 2xz+2wy & 2yz-2wx & 1-(2x^2+2y^2) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{5.2.11}$$

Nyní už jen stačí dosadit hodnoty z našeho quaternionu a rotační matice je hotová. Tu pak vezmeme a roznásobíme jí modelovou matici OpenGL ES a dostaneme aktuální souřadnicový systém do požadovaného stavu. Samotná logika animace je pak ještě o něco složitější, protože s aktuální rotací si nevystačíme, a proto je potřeba je skládat. To se provede snadno pomocí uchování poslední rotace, která je dána součtem předešlých a k ní připočteme aktuální. Výsledná matice je pak naší finální rotací.Dle [3].

Kapitola 6

Implementace aplikace

V této kapitole popíši zkušenosti a poznatky, které jsem získal během tvorby aplikace na platformě Android. Plus také podrobnější popis realizace vykreslování kostky a manipulace s jednotlivými řadami.

6.1 Možnosti vývoje

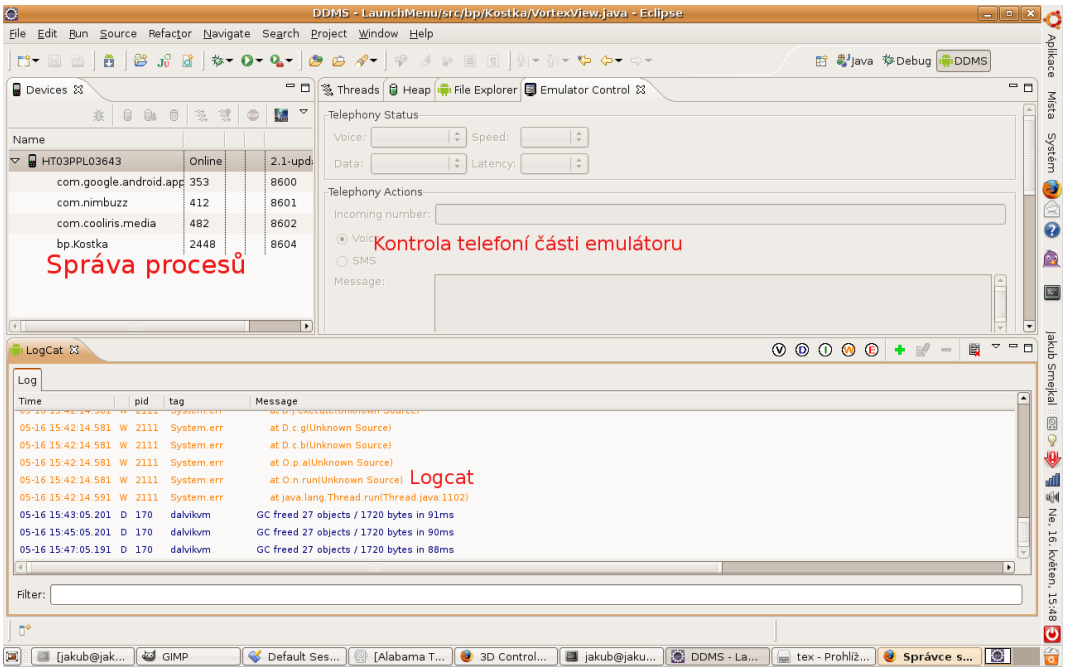
Na stránkách projektu Android je k dispozici SDK a v dnešní době i NDK kit. Google standartně poskytuje největší podporu pro vývojové prostředí Eclipse, které je navíc poskytované zdarma, a proto jsem ho zvolil i já. Mezi další možnosti ještě patří neoficiální podpora Netbeens. Samotná kooperace mezi SDK a vývojovým prostředím probíhá pomocí pluginu, který se do něj nainstaluje. Nyní je možné již vytvářet projekty pro platformu Android. Google poskytuje na svých stránkách také dokumentaci a jistý tutoriál, který je ovšem vzhledem k bouřlivému vývoji platformy občas neaktuální. Větší hodnotu pro vývojáře tedy mají opět relativně aktuální ukázkové kódy, které jsou distribuované rovnou s SDK a samozřejmě internetová komunita, která se od startu projetku značně rozrostla. Existuje tedy poměrně velké množství specializovaných webů a diskuzní skupin.

Po úspěšné instalaci a vytvoření prvního kódu je třeba hledat možnosti jeho ladění a spouštění. K této potřebě můžeme využít buď emulátor, který byl v raných fázích jedinou možností, nebo zakoupit hardware od Google, které umožňuje i výstup do ladící konzole a běh přímo na hardware, což má svoje výhody, především u tvorby grafických aplikací, které využívají OpenGL ES. Jelikož na emulátoru jsme odkázáni na softwarové vykreslování scény, které je nesrovnatelně pomalejší oproti hardwarová akceleraci na zařízení. Pokud by jsme volili cestu hardwarem, tak máme možnost zakoupit přímo vývojářský telefon produkovaný Googlem, nebo nějaký přístroj určený přímo na trh. V tomto případě je ale třeba zkontrolovat seznam kompatibilních zařízení, protože ne všechny podporují již zmiňovaný ladící výstup.

Osobně jsem pracoval většinu času s emulátorem, který je ze zmiňovaných možností nejdostupnější. Jeho možnosti jsou ostatně celkem široké. Před jeho samotným spuštěním je nutné nejdřív vytvořit jeho instanci, kde je bohatá škála specifikací. Pomocí ní můžeme dosáhnout prakticky jakékoliv konfigurace zařízení, která je podporována platformou. Krom nastavení skinů emulátoru, kde tato možnost je spíše estetická, lze nastavit možnosti paměti zařízení, datového uložiště, velikost displeje, komunikační možnosti a další ještě specifičtější

parametry. Tento výčet vypadá poměrně optimisticky, ale přesto samozřejmě nejde emulátorem plně nahradit finální testování, protože ne vždy je tu plná korespondence mezi ním a zařízením, které používá uživatel.

Jakmile máme instanci emulátoru vytvořenou, můžeme využívat jeho potencinálu pro lazení aplikací. Zde považuji za nejužitečnější prvek Logcat, který obsahuje několik úrovní výstupu, podle závažnosti sdělení. Tedy každá zpráva, která se v něm objeví má svoji důležitost, podle které lze výstup filtrovat. Při vývoji aplikace pak máme standartně přístup k třídě, pomocí které si můžeme vytvořit vlastní výstup do Logcatu. Krom tohoto je tu ještě klasiká možnost lazení pomocí breakpointů a poměrně nestandartní tracování aplikace. To je využíváno především k určení kritických bodů v programu, s ohledem na výkonnost, kde je následně po jejich lokalizaci můžeme optimalizovat. K dispozici je také modul informující o stavu vláken a hromady, kde si můžeme manuálně volat garbage collector. Eclipse podporuje i testování aplikace při simulaci příchozích hovorů. Tyto všechny možnosti jsou realizovány pomocí komunikace nástrojů v SDK s emulátorem. Komunikace probíhá formou standartizovaných zpráv, které mají jistou logickou posloupnost. Pomocí těchto zpráv by měly jít simulovat i specifické hardwarové události. Některé jsou snadno odvoditelné, jako například stav baterie, jiné složitější. Tuto komunikaci jde provozovat i manuálně pomocí telnetu, a proto jsem ji zkoumal vzhledem k realizaci simulace inerciální jednotky, kde se nakonec ukázalo, že již bylo vytvořeno snažší řešení, které je označováno jako Sensor simulátor pod uskupením Open Intents. Z názvu již lze poznat, že aplikace je poskytována zdarma a s otevřeným kódem. Celý princip je pak postaven na architektuře klient server, kdy server běží jako javovská aplikace na stolním počítači, kde je lazení prováděno a klient, který od něj přijímá informace, je spouštěn pod emulátorem. Aby tento princip fungoval, bylo třeba vytvořit speciální třídy, které simulují fungování standartní verzí. Prakticky se tedy nastavují data pro emulátor na desktopu a podle potřeby interpretují v aplikaci.



Obrázek 6.1: Ladící rozhraní v prostředí Eclipse.

6.2 Kooperace Androidu a OpenGL ES

K samotné realizaci vykreslování a interakce s kostkou, jsem využil poskytované api pro OpenGL ES 1.0, které je dostupné přímo v SDK. Práce s touto verzí má svoje omezení, ale v době vývoje ještě nebyl k dispozici NDK kit, který umožňuje přístup k implementaci 2.0.

Celé api je zastřešováno jednou třídou, pomocí které se provádí samotné vykreslování. Nazývá se GLSurfaceView a obsahuje rozhraní pro vykreslování, které je pak třeba implementovat. To můžeme realizovat v další třídě, kterou instanciujeme v konstruktoru třídy, která dědí od GLSurfaceView. Technologie nás nenutí k jejímu využití, ale je to výhodné, jelikož je podporována v platformě a poskytuje minimálně srovnatelný výkon, pokud by jsme provedli implementaci ručně. Struktura pomocí které vykresluji mou aplikaci pak vypadá následovně. Na úplném vrcholu je třída dědící od aktivity, kde se instanciuje v konstruktoru třída dědící od GLSurfaceView, která implementuje rozhraní pro vykreslování. V terminologii platformy se nazývá renderer. Zde pak využívám možnost definice jednotlivých pohledů, tedy v androidu nazvaných View, v XML souboru. Pomocí nich lze snadno nadefinovat vzhled aplikace, podle jejich stavů. Při realizaci je třeba je propojit s kódem pomocí speciální třidy, která se nazývá inflater a vytváří již finální funkční realizaci. Tudíž tyto soubory slouží pouze pro popis.

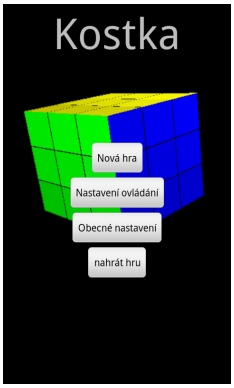
Jakmile je vytvořena hierarchie těchto tříd, je třeba vytvořit algoritmus vykreslování a interakce. To se provede v třídě, která implementuje rozhraní pro renderer. Při jeho tvorbě je možné nastavit poměrně důležitý parametr a to frekvence vykreslování. Zde je na tvůrci zda potřebuje vykreslovat neustále, nebo zvolí ekonomičtější možnost překreslení jen pouze pokud došlo k změnám. V mé aplikaci tento přístup bohatě postačuje. Dále v rozhraní je několik metod, které je nutné naimplementovat. Jsou to funkce pro vytvoření vykreslovací plochy(OnCreateSurface), pro řešení změn(OnChangeSurface), které mohou nastat pokud dojde k změně plochy samotné a konečně funkce, která provádí vykreslení(OnDraw). Osobně provádím ve vytvářecí části nastavení OpenGL stroje a v kreslící vykreslování podle aktuálního stavu kostky. Tento stav je možné změnit buď pomocí transformace systému, tedy rotace, nebo manipulací s jednotlivými řadami. Jelikož jsem se rozhodl pro maximální intuitivnost ovládání, zvolil jsem dotykovou manipulaci. Tedy bylo třeba určit jakou část kostky, tedy kostičku v názvosloví umělé inteligence, má uživatel na mysli. Implementace OpenGL na stolních počítačích poskytuje pro toto nástroj, který funguje na principu vykreslení scény do bufferu, obecně se používá stencil buffer. Nicméně nedojde k standartnímu kreslení, ale je možné si rasterizaci objektu upravit tak, že výsledná reprezentace v bufferu je složena z podle nás definovaných symbolů. Pokud se následně uživatel dotkne displeje, tak podle jím daných souřadnic si spojíme náš objekt, který zvolil, přečtením z tohoto bufferu, který samozřejmě musíme aktualizovat, podle prováděných změn v bufferu, jež se používá pro vizuální reprezentaci. Tento přístup ale z kapacitních důvodů není ve verzi ES podporován, proto musíme pracovat s vykreslovacím bufferem. Tato metoda staví na odlišné volbě barev pro jednotlivé rozlišitelné objekty. Jinak je principielně stejná. Pozor je ale ovšem třeba dát správné nastavení barev, které je nejpřesnější formou jednoho čísla, konktrétně integer. Ten ji reprezentuje pomocí čtyř bajtů, kdy nejvíce významný značí složku alpha, další pak červenou, zelenou a modrou. Tento údaj je třeba rozložit do čtyř dalších integerů, které budou obsahovat uvedené informace na druhém nejméně významném bajtu. Takto lze pak velmi přesně určit požadovanou kostičku.

Jakmile máme určenou kostičku, je třeba rozhodnout jestli chce uživatel pohybovat s řadou nebo sloupcem kostky. Zde jsem zvolil metodu vektoru, podle rozdílu souřadnic. Tedy jestli je větší rozdíl v x složce, nebo v y. Tento vektor opět vytváří uživatel, tahem prstu po dotykovém displeji. Zde je potřeba ještě vzít v potaz rotaci kostky, jelikož metoda je funkční pouze v případě pokud známe její orientaci. Tento problém jsem vyřešil pomocí inverzní transformace k finální transformaci s kostkou. Tou pak aktualizuji body a rozhodnu, jestli rotovat se sloupcem, nebo s řadou a také určím znaménko. Přístup má ovšem jednu nevýhodu a to tu, že ztrácí přesnost vzhledem k natočení stěny k obrazovce. Pokud je natočena kolmo k paprsku od kamery, tak je metoda absolutně přesná, ovšem čím více se úhle mění jakýmkoliv směrem, tak se zmenšuje i přesnost. Nicmémě manipulování s kostkou dochází nejčastěji v prvním případě, proto považuji tento neduh za nepříliš závažný.

Jako poslední prvek aplikace jsem se rozhodl ještě do aplikace přidat umělou inteligenci, který by uživatele přiblížil k řešení. Pro její realizaci jsem se rozhodl využít Kociembův dvoufázový algoritmus, který poskytuje zdarma na svých stránkach již naimplementovaný v javě. Přesto jej nešlo jednoduše přidat do projektu a použít. Jelikož původně byl cílen na stolní počitače, nebyla provedena žádná optimalizace chodu. V původní verzi trval výpočet kostky kolem třiceti minut, což bylo samozřejmě neúnosné. Jako další možnost bylo použít implementaci v C, která by běžela na vzdáleném serveru a poskytovala i řešení symetrie kostek. Nicméně to by znamenalo značnou závislost na internetovém připojení. Proto jsem se rozhodl optimalizovat algoritmus a provést ho lokálně. Po prvním prozkoumání činnosti, bylo jasné že drtivou většinu výpočetního času konzumuje tvorba tabulek sloužících pro vyhledávání. Jelikož jsou tyto tabulky statické, nechal jsem si je vytvořit a připojil je k aplikaci. Krom této akce bylo třeba přepsat v algoritmu přístupy do nich a pokud se jednalo vícedimenzionální tabulky, tak vytvořit mapovací funkci. Nyní se tabulky načtou dávkově do bitových vektorů, odkud jsou pak čteny. Po této úpravě počítá algoritmus do několika sekund, což považuji za dostatečné.

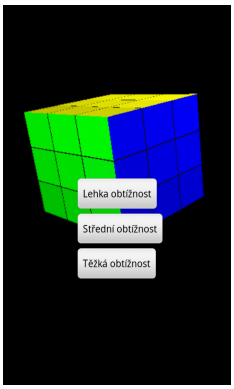
6.3 Konečná podoba aplikace

Hned jak byl hotový funkční základ programu, tedy metody ovládání a zakomponování umělé inteligence. Zbývalo ještě dodělat menu pro uživatele a následně ho propojit s aplikací. Android zde nabízí oficiálně podporovanou cestu formou Aktivit, kde každá Aktivita obsahuje jednu obrazovku menu. Na tvorbu jednotlivých komponent jsem pak využil standartních widgetů platformy, které poskytují požadovanou funkcionalitu. Úvodní rozcestník v programu měl podle mých představ obsahovat klasicky volbu nové hry, nastavení ovládání a jiných specifik programu a následně možnost návratu k dříve rozehrané hře, jak zobrazuje obrázek níže.



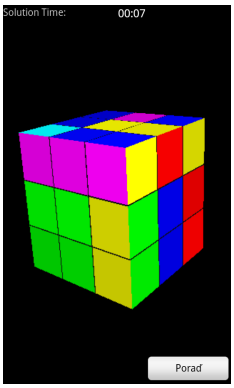
Obrázek 6.2: Hlavní nabídka.

Pokud uživatel zvolí možnost nové hry, zobrazí se mu pak výběr obtížnosti. Tu jsem definoval podle takzvaného „zamíchání“ kostky, jelikož pokaždé vycházím ze složeného tvaru, který následně jednotlivými operacemi pro uživatele pozměním. Operace jsou voleny pseudonáhodně a v tomto menu si hráč volí jejich počet, tedy obtížnost celé hry.Odtud se má uživatel možnost vrátit zpět na hlavní nabídku, pak již následuje samotná hra.



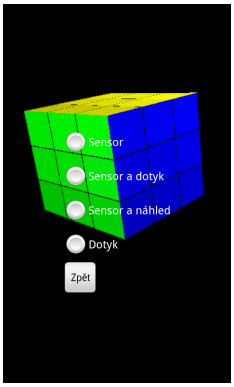
Obrázek 6.3: Nabídka obtížnosti.

Jakmile již hraje jsou pouze dvě možnosti jak tuto obrazovku opustit a to, že složí kostku ať již s pomocí umělé inteligence, nebo bez ní, nebo hru opustí a ta se mu pak následně uloží. Tento stav aplikace nabízí uživateli kromě samotné kostky také dobu řešení problému, tedy zjednodušené stopky a podle jeho volby také tlačítko na radu od algoritmu umělé inteligence.



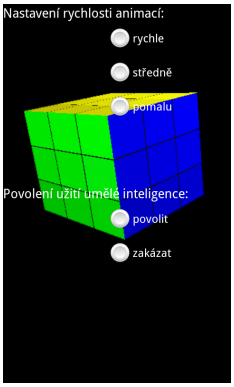
Obrázek 6.4: Rozhraní hry.

Při volbě nastavení ovládání v hlavním menu se uživateli zobrazí všechny podporované přístupy. Tedy ovládání pomocí sensoru, dotyku, sensoru a dotyku, sensoru a náhledu. Zmíněné postupy byli popsány již výše. Pokud uživatel tuto volbu vynechá, je implicitně nastaveno ovládání pomocí dotyku.



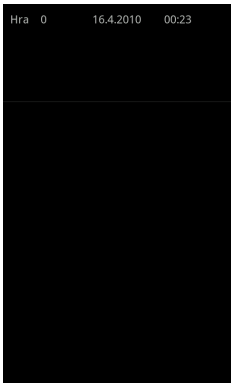
Obrázek 6.5: Nabídka ovládání.

Další možností je nastavení samotného programu, tedy povolení umělé inteligence a nastavení prodlev jednotlivých rotací.



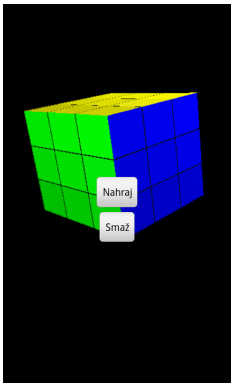
Obrázek 6.6: Nabídka obecného nastavení.

Jako poslední se uživateli může zobrazit seznam uložených her, které jsou označeny pomocí herního identifikátoru a času, kdy byla hra založena, jako poslední je aktuální čas od začátku řešení kostky. Zde si hráč může vybrat uloženou hru, nebo se vrátit zpět.



Obrázek 6.7: Nabídka uložených her.

Po volbě, mu aplikace nabídne možnosti buď nahrát, nebo smazat.



Obrázek 6.8: Nabídka voleb pro manipulaci s uloženou hrou.

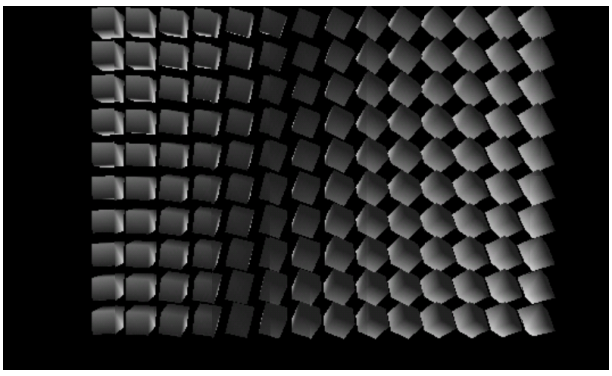
Kapitola 7

Měření 3D výkonu platformy

Jako poslední část bakalářské práce jsem se rozhodl zahrnou porovnání výkonosti této stále relativně nové platformy s klasickými telefony a jejich implementací 3D grafiky. Jako referenci k výsledkům mého měření jsem použil bakalářskou práci Marka Vyorala, která byla na téma 3D aplikace pro mobilní telefony[1]. Provedl jsem tedy stejné testy jako ve výše zmíněné práci s výjimkou těch jednoduších, jelikož zde byly výsledky neprůkazné, vzhledem k internímu omezení počtu snímku v implementaci OpenGL ES na platfomě Android, kde lze dosáhnout pouze hodnoty šedesát. Plus také testu, který využívá reprezentace grafických objektů pomocí 3ds max, jelikož na jeho tvorbu již nezbýval čas. Tyto změny v testech jsem pak zohlednil ve výsledcích přepočtem tabulky, takže porovnání neutrpělo na újmě. Předpokládané výsledky vzhledem k použitému zařízení HTC Desire, který je výpočetně poměrně výkoný a podpoře OpenGL ES, by měli být značně nad porovnávanou technologií. Všechny testy byli prováděny až po delší době, kdy došlo k ustálení počtu zobrazení a následně byl odebrán vzorek v délce deseti sekund a hodnotou počtu snímků za tuto dobu. V následujících řádcích představím jednotlivé testy a pak porovnám výsledky s výše zmíněnou prací.

7.1 Testy

Test rotace 140 krychlí s jedním světlem. Tato scéna je ekvivalentní s testem číslo šest z porovnávané práce.



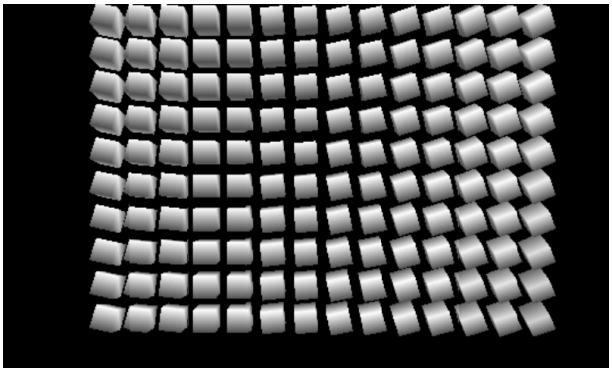
Obrázek 7.1: Test s jedním světlem.

Po ustálení počtu snímků, lze z výstupu Logcatu odvodit průměrnou zaokrouhlenou hodnotu 36 vykreslení za sekundu. Tedy za deset sekund souhrně 357 snímku.

```
05-15 15:57:11.461      I 1858 FPS                    36
05-15 15:57:12.481      I 1858 FPS                    36
05-15 15:57:13.491      I 1858 FPS                    35
05-15 15:57:14.511      I 1858 FPS                    36
05-15 15:57:15.531      I 1858 FPS                    35
05-15 15:57:16.541      I 1858 FPS                    36
05-15 15:57:17.551      I 1858 FPS                    36
05-15 15:57:18.571      I 1858 FPS                    36
05-15 15:57:19.591      I 1858 FPS                    36
05-15 15:57:20.601      I 1858 FPS                    35
```

Obrázek 7.2: Výsledky testu s jedním světlem.

Test rotace 140 krychlí se čtyřmi světly. Zde porovnávám test číslo sedm.



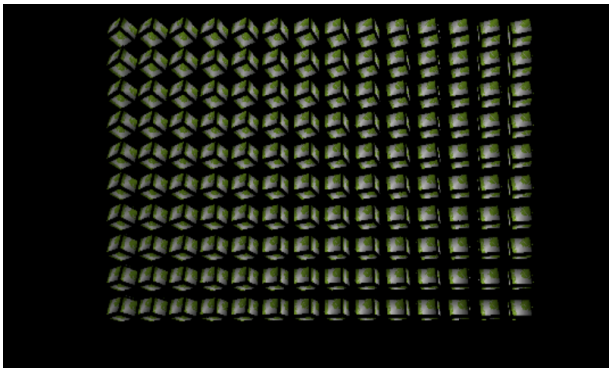
Obrázek 7.3: Test se čtyřmi světly.

Tato scéna je již náročnější a je to znát na výsledcích, kde telefon zvládá vykreslovat kolem 29 snímků za sekundu. To znamená souhrně 287 vykreslení za deset sekund.

```
05-15 15:36:23.611      I 1064 FPS                    28
05-15 15:36:24.631      I 1064 FPS                    28
05-15 15:36:25.651      I 1064 FPS                    29
05-15 15:36:26.681      I 1064 FPS                    29
05-15 15:36:27.711      I 1064 FPS                    29
05-15 15:36:28.711      I 1064 FPS                    28
05-15 15:36:29.741      I 1064 FPS                    29
05-15 15:36:30.761      I 1064 FPS                    29
05-15 15:36:31.791      I 1064 FPS                    29
05-15 15:36:32.831      I 1064 FPS                    29
```

Obrázek 7.4: Výsledky testu se čtyřmi světly.

Test rotace 140 krychlí se jedním světlem a texturou. Test odpovídá číslu osm v porovnávané práci.



Obrázek 7.5: Test s texturou a jedním světlem.

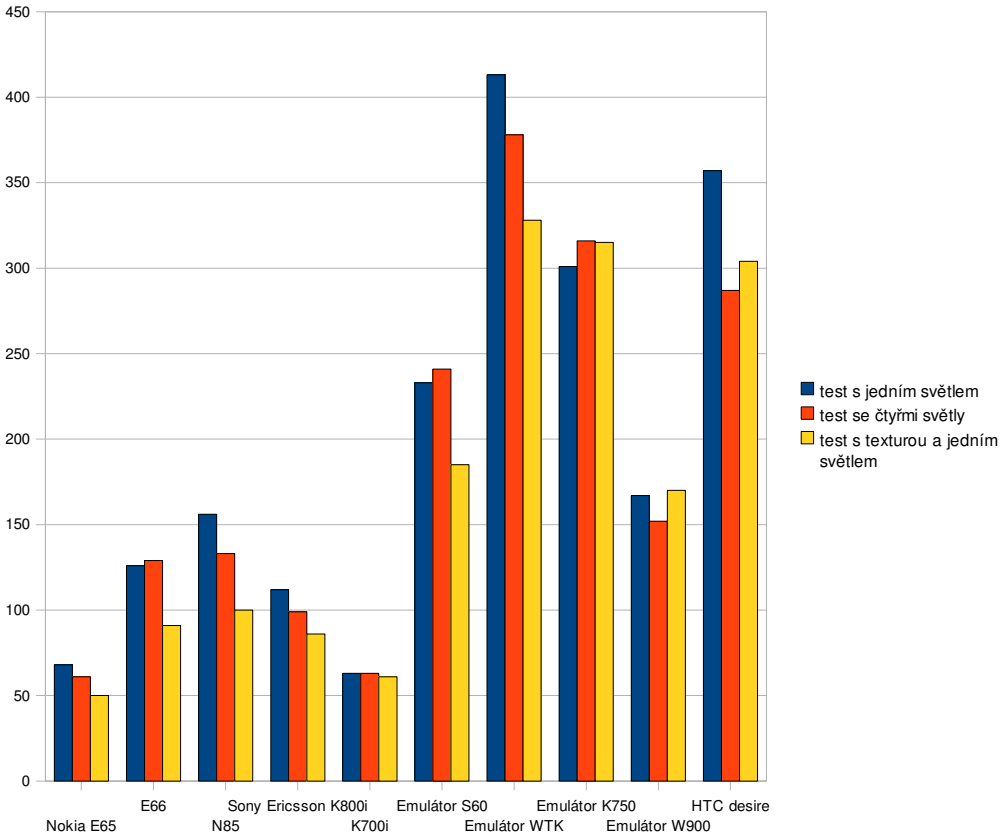
Zde je patrné menší zlepšení, vzhledem k sníženému počtu světel ve scéně. Nicméně náročnost vykreslení textury se i tak projevila. Aplikace se pohybuje mezi 30 a 31 snímky za vteřinu, souhrně 304 vykreslení za deset sekund.

```
05-15 15:51:03.161      I 1565 FPS                31
05-15 15:51:04.161      I 1565 FPS                30
05-15 15:51:05.161      I 1565 FPS                30
05-15 15:51:06.171      I 1565 FPS                30
05-15 15:51:07.191      I 1565 FPS                31
05-15 15:51:08.221      I 1565 FPS                31
05-15 15:51:09.231      I 1565 FPS                30
05-15 15:51:10.241      I 1565 FPS                30
05-15 15:51:11.271      I 1565 FPS                31
05-15 15:51:12.281      I 1565 FPS                30
05-15 15:51:13.301      I 1565 FPS                31
```

Obrázek 7.6: Výsledky testu s textůrou a jedním světlem.

7.2 Výsledky

Výstupy testů jsou shrnuty v následujícím grafu, kde jsou zobrazeny všechna zařízení z porovnávané práce ve třech realizovaných testech. Výsledné počty snímku jsou počítány za dobu deseti sekund. Předpoklad favorita HTC se potvrdil až na případ emulátorů, které vykazovaly velkou výkonost.



Obrázek 7.7: Souhrn výsledků za deset sekund.

Kapitola 8

Závěr, možnosti rozšíření

Cílem bakalářské práce byla tvorba trojrozměrné aplikace s podporou inerciální jednotky pod platformou Android. K její realizaci bylo třeba nastudovat platformu android, práci s OpenGL a se specifiky varianty ES, prostudovat možnosti testování. Průběh celé práce bych rozdělil na tvorbu reprezentace kostky v aplikaci a tedy i její ovládání, plus jsem se rozhodl přidat navíc podporu umělé inteligence, kde jsem využil již vytvořeného dvoufázového algoritmu od Herberta Kociemby. Ve finální části pak bylo nutné aplikaci odladit na fyzickém zařízení, tedy především chování inerciální jednotky. Při tvorbě reprezentace kostky bylo nutné vytvořit, kromě algoritmu vykreslení, její interní reprezentaci v programu, kterou reprezentuje samostatná třída. Do této fáze také patři realizace rotací, kde bylo potřeba zvolit správnou metodu. Jako poslední pak přišel na řadu algoritmus umělé inteligence, který bylo nutné optimalizovat pro použití na mobilním zařízení. S výsledkem práce jsem poměrně spokojený, kromě proměnlivé přesnosti ovládání jednotlivých řad a sloupců kostky. Zde ještě existují mezery, které by šlo vylepšit, nicméně se domnívám, že na ovládání aplikace tento nedostatek nemá až tolik zásadní vliv.

Jako možnosti rozšíření bych uvedl přidání podpory pro nahrávání časů řešení kostky na internet, nebo zdokonalení vizuální stránky kostky směrem k lepšímu ovládání. Například umístění reflexní plochy do prostoru za kostku, tedy uživatel by měl lepší přehled o jejím stavu. Nebo přidání sportovní varianty řešení kostky pomocí gps.

Literatura

[1] Vyoral, M.: 3D APLIKACE PRO MOBILNÍ TELEFONY. 2009.

[2] WWW stránky: 3D ovládání.
http://www.cabiatl.com/mricro/obsolete/graphics/3d.html, [cit. 5. 2. 2010].

[3] WWW stránky: ArcBall na stránkách rainwarrior.
http://rainwarrior.thenoos.net/dragon/arcball.html, Vystaveno roku 2008 [cit. 4. 3. 2010].

[4] WWW stránky: Wikipedie.
http://en.wikipedia.org/wiki/File:Eulerangles.svg, Vystaveno roku 2008 [cit. 6. 1. 2010].

[5] WWW stránky: Stránky Herberta Kociemby, autora Dvoufázového algoritmu.
http://kociemba.org/cube.htm, Vystaveno roku 2010 [cit. 10. 4. 2010].

[6] WWW stránky: Wikipedie. http://cs.wikipedia.org/wiki/Leonhard_Euler, Vystaveno roku 2010 [cit. 11. 1. 2010].

[7] WWW stránky: Rubikova kostka na Wikipeii.
http://en.wikipedia.org/wiki/Rubik%27s_Cube, Vystaveno roku 2010 [cit. 12. 3. 2010].

[8] WWW stránky: Algoritmy pro optimální řešení Rubiovy Kostky na Wikipeii.
http://en.wikipedia.org/wiki/Optimal_solutions_for_Rubik%27s_Cube, Vystaveno roku 2010 [cit. 13. 3. 2010].

[9] WWW stránky: Tutorial k projekcím na droidnově.
http://www.droidnova.com/android-3d-game-tutorial-part-vi 436.html, Vystaveno roku 2010 [cit. 15. 2. 2010].

[10] WWW stránky: Quaterniony na stránkách Wolfram mathworld.
http://mathworld.wolfram.com/Quaternion.html, Vystaveno roku 2010 [cit. 21. 2. 2010].

[11] WWW stránky: Platforma Android na Wikipedii.
http://en.wikipedia.org/wiki/Android_(operating_system), Vystaveno roku 2010 [cit. 4. 2. 2010].

[12] WWW stránky: Souřadné systémy na Wikipeii.
http://en.wikipedia.org/wiki/Coordinate_system, Vystaveno roku 2010 [cit. 7. 1. 2010].

[13] WWW stránky: OpenGL ES na stránkach Khronos.
`http://www.khronos.org/opengles/`, Vystaveno roku 2010 [cit. 8. 2. 2010].

[14] WWW stránky: Platforma Android na stránkach Google.
`http://developer.android.com/guide/topics/fundamentals.html`, Vystaveno roku 2010 [cit. 8. 2. 2010].

Dodatek A

Přílohy

Obsah CD Na CD přikládám zdrojové soubory programu, které se nacházejí v adresáři /bp. Adresářová struktůra programu odpovídá klasiké aplikaci pod platformou Android. Dále krátký manuál s popisem ovládání aplikace a plakát prezentující moji práci.