UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Zobrazování 3D scén na platformě Windows Mobile s využitím gravitačního senzoru

Štěpán Šonský

Bakalářská práce
2010

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2009/2010**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Štěpán ŠONSKÝ**
Osobní číslo:	**I07805**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Zobrazování 3D scén na platformě Windows Mobile s využitím gravitačního senzoru**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

V teoretické části bude popsána problematika tvorby 3D grafických aplikací pro platformu Windows Mobile s využitím OpenGL.
Cílem praktické části je návrh a implementace algoritmů pro:
- jednoduché modelování základních těles,
- 3D zobrazovací metody a
- transformaci scény na základě informací z gravitačního senzoru (simulující holografický efekt).

Implementační část bude realizována pro platformu Windows Mobile pomocí programovacího jazyka C ♯ s využitím API OpenGL—ES.
Součástí praktické části bude ukázková aplikace, demonstrující možnosti a využití implementovaných algoritmů.

Rozsah grafických prací:

Rozsah pracovní zprávy:

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

Seznam odborné literatury:

***SHARP, John. Microsoft Visual C 2008 Krok za krokem. Brno : Computer Press, 2008. 592 s. ISBN 978-80-251-2027-9.**

***SHREINER, Dave, WOO, Mason, NEIDER, Jackie, DAVIS, Tom. OpenGL. Průvodce programátora. Brno : Computer Press, 2006. 696 s. ISBN 80-251-1275-6.**

Vedoucí bakalářské práce: **Ing. Petr Veselý**
Katedra informatiky v dopravě

Datum zadání bakalářské práce: **15. ledna 2010**

Termín odevzdání bakalářské práce: **14. května 2010**

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Ing. Lukáš Čegan, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2010

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 12. 5. 2010 Štěpán Šonský

Poděkování

Děkuji především vedoucímu práce Ing. Petru Veselému za obstarání literatury, konzultace v průběhu zpracování a výuku počítačové grafiky.

Anotace

Práce se zabývá problematikou 3D grafiky na mobilních zařízeních s operačním systémem Windows Mobile, obsahuje seznámení se základními principy programování v OpenGL|ES prostřednictvím jazyka C#, vytváření těles a interakcí pomocí gravitačního senzoru, aby bylo možné pouhým nakloněním přístroje scénu prohlížet z různých úhlů nebo pohybovat jednotlivými objekty. Jsou popsány všechny vlastní třídy, jejich metody a algoritmy. Součástí je i podrobný popis ukázkových aplikací a jejich možností.

Klíčová slova

C#, OpenGL, Windows Mobile, grafika, akcelerometr, GPS

Title

Displaying 3D scenes using accelerometer on Windows Mobile platform

Annotation

The bachelor thesis deals with a topic of 3D graphics on the mobile devices running Windows Mobile operating system. Contains basic principles of OpenGL|ES programming using C# language, creating basic objects and interaction of scene depends on accelerometer. Own implemented classes, their methods and the algorithms are described there. Main goal is an ability to view scene from angle, that viewer want or moving objects, only by tilting the device. Descriptions of demo applications are also included.

Keywords

C#, OpenGL, Windows Mobile, graphics, accelerometer, GPS

Obsah

Seznam zkratek

3D	3 Dimensional - trojrozměrný
API	Application Programming Interface
FPS	Frames Per Second – počet snímků za sekundu
GAPI	Graphics Application Programming Interface
GLU	OpenGL Utility – rozšiřující nástroje OpenGL
GLUT	OpenGL Utility Library – knihovna rozšiřujících nástrojů OpenGL
GPS	Global Positioning System
GPU	Graphic Processing Unit
G-Senzor	gravitační senzor, akcelerometr
OpenGL|ES	Open Graphic Library for Embedded Systems
PNG	Portable Network Graphics – rastrový grafický formát
px	Pixel
SDK	Software Development Kit
SGI	Silicon Graphics Inc
WM	Windows Mobile – označení pro mobilní Windows do verze 6.5
WP	Windows Phone – označení pro mobilní Windows od verze 7
WVGA	Wide VGA rozlišení 800×480 px

Seznam obrázků

Seznam tabulek

1 Úvod

Hlavním cílem této bakalářské práce je, přinést inovativní řešení 3D zobrazení na zařízení s operačním systémem Microsoft WM 6.5. Za pomocí vestavěného G-Senzoru dosáhnout optické iluze 3D obrazu na běžném displeji. Pouhým nakloněním přístroje je možné prohlížet scénu z různých úhlů nebo pohybovat objekty. Pozorovatel nabývá dojmu plastického obrazu uvnitř displeje, ale je možné docílit i efektu, kdy objekty vystupují ven. Důležitou částí jsou také algoritmy a metody pro zjednodušené vytváření základních geometrických útvarů, které mají poskytnout programátorům vyšší vrstvu (nadstavbu) na OpenGL|ES. Zahrnuty jsou i některé pokročilé techniky jako např. texturování, dynamické osvětlení nebo blending, kterým lze docílit průhlednosti. Projekt nese název HG-Engine (od slova hologram) a v budoucnu by mohl najít uplatnění v různých odvětvích mobilní 3D grafiky, která se v dnešní době velmi rychle rozvíjí. Mezi příklady využití patří například pokročilý GPS software s modely budov, grafické uživatelské prostředí, kde může být naklánění využito pro efektivnější ovládání funkcí telefonu. Potenciál má ale i pro zábavní softwarový průmysl, kde je 3D grafika hojně využívána k tvorbě her. Nejmodernější senzory tak umožňují vývoj zábavního software nové generace, který přinese dosud nevídané možnosti interakce s realitou. V jednom z praktických příkladů je na ukázku implementována i jednoduchá fyzika, která s gravitací úzce souvisí.

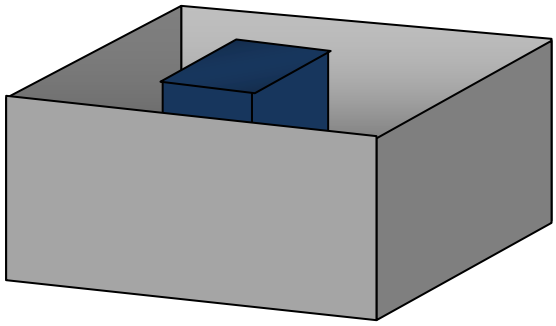
Programovací jazyk jsem zvolil C#, jelikož s ním mám zkušenosti z programování pro běžné počítače. Problém vzniknul až při volbě grafické knihovny. Začínal jsem s GAPI – integrovaným grafickým rozhraním, ale brzy jsem zjistil, že pokud budu chtít pracovat v 3D prostoru, budu si muset vybrat mezi OpenGL|ES a DirectX Mobile. K mému konečnému rozhodnutí přispělo hned několik důvodů. V první řadě dostupnost tutoriálů a zdrojů. Paradoxně Microsoft DirectX Mobile se při programování 3D grafiky skoro nepoužívá**,** protože i vývoj samotné knihovny se zastavil. WM 6.5 jsou také poslední verzí, která DirectX podporuje. V připravovaných WP 7 najdeme pouze podporu OpenGL|ES ve verzi 2.0 a jelikož projekt plánuji do budoucna rozvíjet, vybral jsem si jako grafické API právě OpenGL|ES.

Od počátku bylo mým cílem interagovat scénu pomocí gravitačního senzoru. Po zvládnutí základního úkolu, kdy se mi podařilo do projektu připojit potřebné knihovny, jsem naprogramoval jednoduchý posun úsečky závislý na akcelerometru a vykreslený pomocí GAPI. Postupně jsem začal tvořit nenáročné 2D aplikace v OpenGL|ES. K dispozici jsem měl pouze minimum zdrojových kódů, které mi umožnily pochopit některé metody v této knihovně. Trvalo několik dní, než jsem dokázal vytvořit první vlastní 3D aplikaci. Byl to trojúhelník, který se otáčel v závislosti na gravitačním senzoru. Dalším rozvíjením postupně vznikal projekt HG-Engine.

Při návrhu jsem vycházel z pozorování skutečného světa. Objekty kolem nás můžeme sledovat ze všech možných úhlů a mozek si tak vytvoří informaci o jejich celistvém tvaru. Orientaci v prostoru napomáhají naše dvě oči, každé z nich totiž snímá obraz z trochu jiného úhlu. Naproti tomu v 3D grafice i když je svět vymodelován do

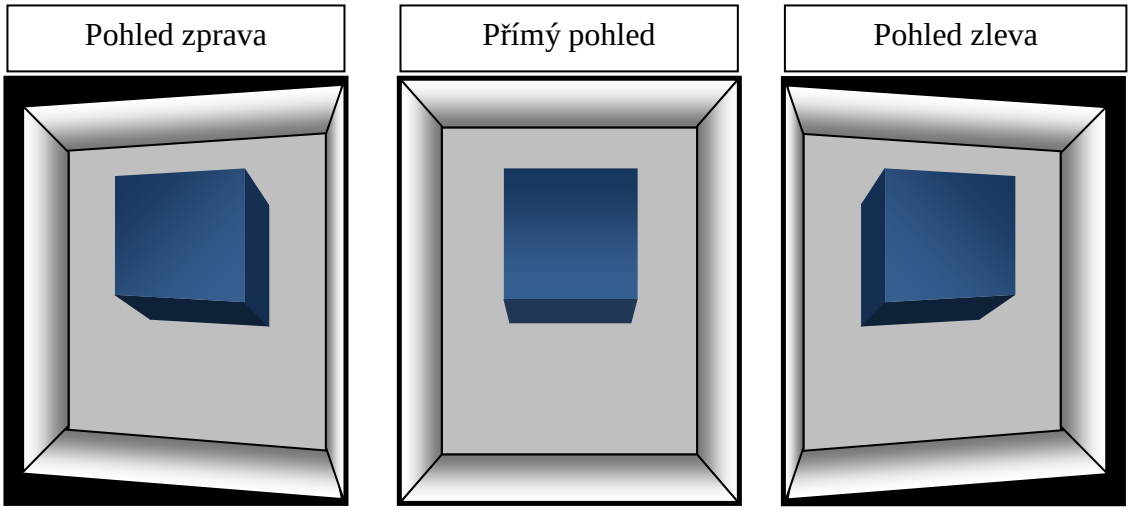
nejmenších detailů, přesto jeho zobrazení na běžném displeji nebude věrohodné z jednoho prostého důvodu. Poskytuje nám pouze jeden a ten samý obraz pro obě oči. Skutečného 3D zobrazení tak není možné dosáhnout bez použití 3D displeje nebo speciálních brýlí. Moderní technologie ale umožňují prostorový efekt simulovat i na běžných displejích různými metodami snímání polohy pozorovatele. Jednou takovou metodou je headtracking. Ten je založen na infračerveném senzoru, který snímá polohu světelného bodu v prostoru. Díky tomu pak lze měnit polohu kamery synchronně k pohybu diváka a k němu se tak dostává obraz, který odpovídá jeho poloze v prostoru. Mobilní telefony touto technologií nedisponují, ale některé z nich umožňují pomocí G-Sensoru určit polohu telefonu, z čehož se dá odvodit i přibližná poloha pozorovatele. A právě tohoto principu v této práci využívám.

2 Řešený problém



Obrázek 1 – Modelová situace, kvádr uvnitř krabice

Pro lepší představu to lze znázornit na modelovém příkladu, kvádru umístěném uvnitř pomyslné krabice bez vrchní stěny, viz obrázek 1. Zatímco v reálném světě jsou tyto objekty statické a mění se jen úhel pohledu pozorovatele, tak na 3D grafické scéně je situace složitější. Pokud by se změnila pouze poloha kamery pomocí standardních transformací, tak bude docházet k deformacím z důvodu perspektivního zobrazení. Mezi vertikálními a horizontálními hranami nebude pravý úhel a při pohledu zleva bude pravá hrana nepatrně kratší než levá, protože by byla v prostoru vzdálenější. V přímém pohledu je vše v pořádku a scéna působí na diváka tak, jak má. Jakmile se ale změní poloha kamery, dochází ke zmíněným deformacím. Vykreslovaný obraz vymezuje obdélník s černou výplní. Na obrázku 2 je znázorněna modelová situace tak, jak by se chovala při změně polohy kamery.

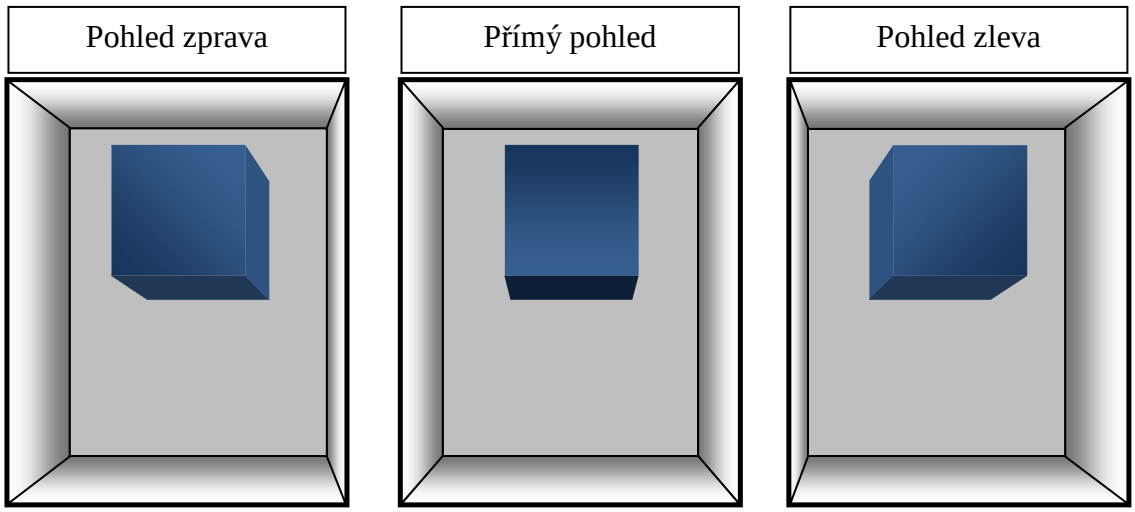


Obrázek 2 – Ukázka změny pohledu na scénu pomocí transformace

Aby se docílilo žádaného efektu virtuálního prostoru uvnitř přístroje, je třeba počítat ještě s displejem, který stojí v cestě jako pomyslné okno do virtuálního světa. Pro názornost si ho lze představit jako průhledné víko krabice. Z toho plyne, že ať už se na ní

pozorovatel bude dívat z jakéhokoliv směru a vzdálenosti, vždy budou korespondovat rohy displeje s horními vrcholy krabice. Čili vzhledem k displeji musejí být tyto body statické.

Z perspektivy je známo, že čím vzdálenější je objekt od pozorovatele, tím se jeví menší. Perspektiva také způsobuje, že dvě rovnoběžky se úměrně s přibývající vzdáleností sobě zdánlivě přibližují. S těmito optickými zákony OpenGL počítá, avšak některé z nich bylo nutné obejít, protože pouhou změnou polohy kamery nelze dosáhnout požadovaného efektu. Některé deformace způsobené perspektivou jsou na scéně nežádoucí, protože vzniknou až v reálném světě při pohledu na celý mobil. Stěny krabice tedy nyní vymezují virtuální prostor jakoby pod displejem. Důležité je si uvědomit, jak se zdánlivě pohybují jednotlivé body vzhledem k pozorovateli, když mění úhel pohledu. Následující obrázek 3 znázorňuje jednotlivé pohledy tak, jak by se měly vykreslovat na displej, aby bylo dosaženo žádaného efektu.



Obrázek 3 – Ukázka požadovaného vykreslení při různých pohledech

Aby bylo možné tohoto docílit, je zapotřebí přímo měnit polohu jednotlivých bodů po osách „x“ a „y“. Předem je ale nutné tento pohyb správně navrhnout. V základním pohledu, kdy je kamera nasměrovaná doprostřed scény, se všechny okolní stěny jeví stejně široké. Při pohybu kamery směrem doprava se ale pravá stěna zdánlivě zužuje a levá roztahuje, respektive zadní body se posunou doprava, stejně tak je tomu u všech ostatních směrů. Dalším faktem je to, že čím je bod od pozorovatele vzdálenější, tím více se pohybuje. Skutečnost je tedy taková, že se nemění poloha kamery, ale deformují se objekty na scéně. Jednotlivé vrcholy objektů se pohybují v závislosti na jejich hloubce vyjádřené pomocí „z“ souřadnice. Body se souřadnicí $z < 0$ se pohybují stejným směrem jako kamera. V místě, kde $z = 0$, čili tam, kde je virtuální prostor jakoby v hloubce displeje, jsou body statické (nepohybují se). Zajímavá situace ale nastává, když $z > 0$, v takovém případě se body pohybují v opačném směru vzhledem ke kameře a tím vzniká dojem, že objekt vystupuje ven z displeje.

3 Základy programování v OpenGL

Knihovna navržená firmou SGI, je v současné době standardem pro tvorbu 2D a 3D aplikací na platformách Windows, UNIX, Linux a MacOS X. Ale díky rychle se rozvíjejícímu mobilnímu průmyslu najdeme odlehčenou verzi OpenGL|ES i v kapesních přístrojích. Nativní podporou disponuje např.: Apple iPhone, zařízení s Windows Mobile 6 (zejména značky HTC, Sony Ericsson a Samsung), dále přístroje s novým mobilním operačním systémem Google Android, ale i vybrané modely Nokia s operačním systémem Symbian.Ve verzi 1.1, která je v rámci práce využívána chybí rozšiřující knihovny GLU a GLUT, které poskytují daleko širší možnosti a obsahují hlavně funkce pro vytváření těles, které bylo nutné pro tuto práci naprogramovat.

3.1 Inicializace OpenGL okna

Základním předpokladem pro programování v OpenGL je vytvoření GL okna, to znamená části obrazovky, kam se bude vykreslovat veškerý viditelný obsah scény. Tato inicializace se píše do konstruktoru hlavního formuláře, zajistí základní konfiguraci a inicializuje frame buffer. Ten uchovává aktuální informace o barvě jednotlivých pixelů a pokud je naplněn, odesílá se na výstup. Dále se nastaví depth buffer (nebo také z-buffer), do kterého se ukládají data o hloubce pixelů. Díky němu lze jednoznačně určit, jaká barva se vykreslí v případě překrývání objektů. Výsledkem následujících řádků kódu je OpenGL|ES okno s černým pozadím. Tuto barvu lze změnit pomocí parametrů `ClearColor(R, G, B, A)`.

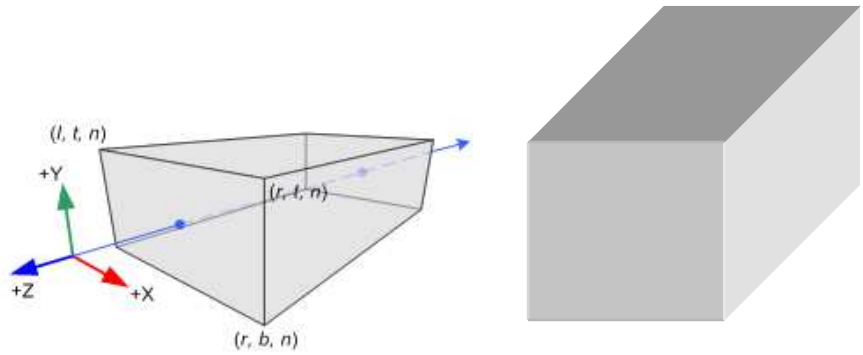
```
myDisplay = egl.GetDisplay(new EGLNativeDisplayType(this));

int major, minor;
egl.Initialize(myDisplay, out major, out minor);
EGLConfig[] configs = new EGLConfig[10];

int[] attribList = new int[]
{
      egl.EGL_RED_SIZE, 5,
      egl.EGL_GREEN_SIZE, 6,
      egl.EGL_BLUE_SIZE, 5,
      egl.EGL_DEPTH_SIZE, 16 ,
      egl.EGL_SURFACE_TYPE, egl.EGL_WINDOW_BIT,
      egl.EGL_STENCIL_SIZE, egl.EGL_DONT_CARE,
      egl.EGL_NONE, egl.EGL_NONE
};

int numConfig;
EGLConfig config = configs[0];
mySurface = egl.CreateWindowSurface(myDisplay, config, Handle, null);
myContext = egl.CreateContext(myDisplay, config, EGLContext.None, null);
egl.MakeCurrent(myDisplay, mySurface, mySurface, myContext);

gl.ClearColor(0, 0, 0, 0);
```

3.2 Matice

V OpenGL existují 3 různé transformační matice, které slouží k manipulaci s objekty, kamerou, perspektivou nebo souřadnicemi textur. Tyto matice o rozměrech 4×4 jsou interpretovány jako jednorozměrné float pole o 16 prvcích. Vynásobením aktuální scény těmito maticemi dochází k různým transformacím (viz podkapitola 3.3). Význam jednotlivých matic a jejich možnosti:

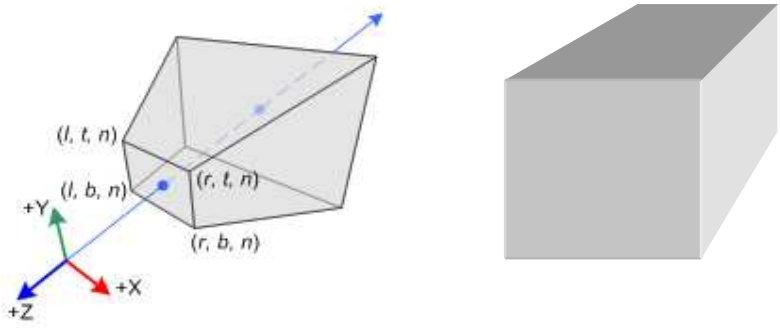
- **GL_PROJECTION** – Projekční (pohledová) matice, pomocí níž se nastavují vlastnosti pohledu jako například perspektiva, zorný úhel nebo výřez, ale lze také pohybovat nebo rotovat kamerou. Základními typy projekce jsou:

  - **Ortografická** – Rovnoběžná, neprojevuje se perspektivní zkreslení. Dva stejně velké objekty v různé vzdálenosti mají po vykreslení stejnou velikost. Ortografická projekce je vhodná například pro technické zobrazení, protože nedochází k deformacím vzdáleností, viz obrázek 4 [2].



Obrázek 4 – Znázornění viditelné oblasti a zobrazení kvádru v ortografické projekci

  - **Perspektivní** – Zobrazení se mnohem více podobá skutečnosti. Vzdálenější objekty jsou vykresleny menší než objekty vpředu. Rovnoběžné linie se s rostoucí vzdáleností opticky přibližují, viz kvádr na obrázku 5 [2]. Reálně počítá s optickými zákonitostmi perspektivy, v 3D grafice se používá nejčastěji.



Obrázek 5 – Znázornění viditelné oblasti a zobrazení kvádru v perspektivní projekci

- **GL_MODELVIEW** – Modelová matice, která slouží k základním transformacím objektů na scéně, například ke změně měřítka, posunu nebo rotaci.

- **GL_TEXTURE** – Tato matice se využívá při mapování textur na povrch objektů.

V průběhu programu je třeba mezi těmito maticemi přepínat a jasně tím specifikovat, zda se má transformovat jenom pohled, nebo přímo objekty. Přepnutí aktuální matice se provede následovně:

```
gl.MatrixMode(gl.GL_MODELVIEW);
```

Před každým vykreslením se musí transformační matice resetovat na matici jednotkovou. Ta způsobí, že při vynásobení nebudou prováděny žádné transformace.

```
gl.LoadIdentity();
```

3.3 Transformace

K přizpůsobení kamery a scény se používají metody, které k maticím přímo přistupují a mění jejich obsah. Samotná transformace potom probíhá násobením souřadnic bodů těmito upravenými maticemi.

- `void Translatef(float x, float y, float z);`
  Provádí posun o vzdálenost zadanou jako parametr.

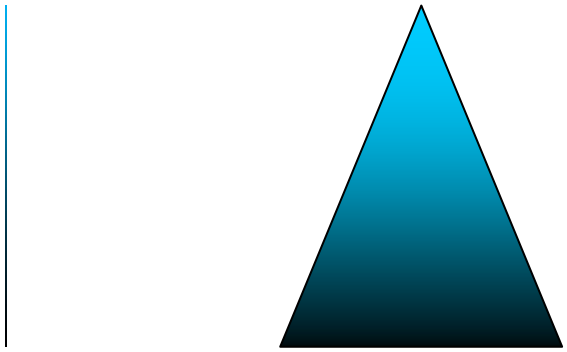
- `void Rotatef(float angle, float x, float y, float z);`
  Slouží k rotaci, angle udává úhel, ostatní parametry směr.

- `void Scalef(float x, float y, float z);`
  Změna měřítka, globálně mění velikost objektů na scéně.

3.4 Textury

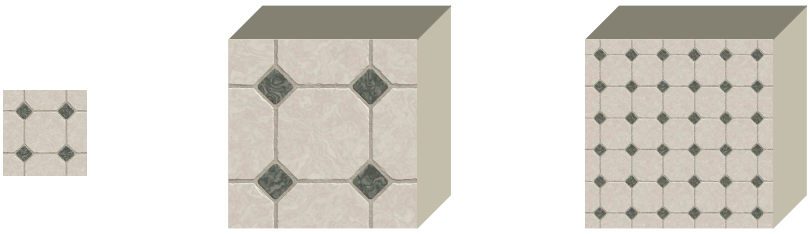
Textura je obecně bitmapa, která se nanáší na povrch objektu, aby se docílilo jeho reálnějšího vzhledu. Lze tak simulovat prakticky jakýkoliv povrch bez výrazných nerovností. OpenGL|ES podporuje pouze rastrové textury, které se dále dělí na:

- **GL_TEXTURE** – Jednodimenzionální textura o šířce 1 pixel. Nejčastěji se používá na barevné přechody (gradienty), které vzniknou jejím roztažením na šířku povrchu texturované plochy, viz obrázek 6. Výhodou tohoto typu textur je nenáročnost na grafickou paměť, nelze ale pomocí nich vytvářet na povrchu objektů složitější obrazce.



Obrázek 6 – Jednodimenzionální textura následně aplikovaná na trojúhelník

- **GL_TEXTURE_2D** – Za dvoudimenzionální texturu lze považovat jakýkoliv rastrový obrázek se dvěma rozměry, může jím být například i fotografie nebo geografická mapa. Tento typ textur se v 3D grafice používá nejčastěji. Texturu lze na povrch aplikovat různými způsoby. Může být roztažena do velikosti texturované plochy nebo jí lze nanášet několikrát vedle sebe.



Obrázek 7 – Dvoudimenzionální textura a možnosti její aplikace na objekt

Je nutné brát na vědomí, že čím větší rozlišení a barevnou hloubku textura má, tím náročnější bude vykreslení scény. A právě u mobilních telefonů je důležité šetřit prostředky, jelikož mobilní grafické čipy mají výkon velmi omezený. Použití textur je v 3D grafice výhodné, protože v řadě případů není nutné modelovat detailní objekt a zatěžovat tak GPU, ale stačí použít geometricky jednoduchý objekt s vhodnou texturou.

Samotnému použití textury předchází několik nezbytných kroků, především připojení souboru do RESOURCES a nastavení vlastnosti BUILD ACTION na EMBEDDED RESOURCE. Soubor s obrázkem se tak stane při sestavení programu součástí exe souboru. V práci jsou použity textury ve formátu PNG s barevnou hloubkou 8 bpp (256 barev). V programu se potom textura načte a použije takto:

1. V inicializaci je třeba zapnout podporu textur v OpenGL.

   ```
   gl.Enable(gl.GL_TEXTURE_2D);
   ```

2. Deklarace proměnné `tPict` typu `Texture`

   ```
   Texture tPict;
   ```

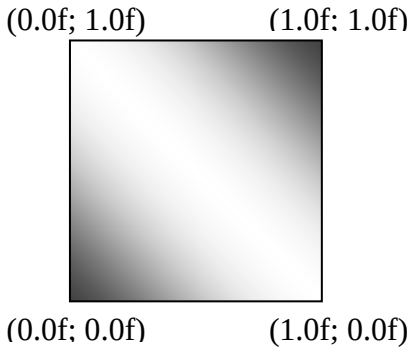
3. Načtení souboru PICT.PNG do této proměnné, druhý parametr udává, zda se má použít průhlednost (v tomto případě ne).

   ```
   tPict = Texture.LoadStream
   (System.Reflection.Assembly.GetExecutingAssembly().GetManifestRe-
   sourceStream("HGEngine.Resources.Pict.png"), false);
   ```

4. V místě, kde se má daná textura použít, se volá metoda `BindTexture`. Veškeré objekty vykreslované za tímto řádkem budou otexturovány souborem PICT.PNG.

   ```
   gl.BindTexture(gl.GL_TEXTURE_2D, tPict.Name);
   ```

Při texturování je možné nanést bitmapu na objekt v jakémkoliv směru. K tomu slouží pole `GL_TEXTURE_COORD_ARRAY`. Každému bodu ve `GL_VERTEX_ARRAY` (viz podkapitola 3.7 Vykreslování) náleží dva koordináty (souřadnice) textury, které dávají programu jasně najevo, jaká část bitmapy se má použít a jak bude na objekt umístěna. Souřadnice textur se řídí následujícím pravidlem na obrázku 8.



Obrázek 8 – Souřadnicový systém textur

```
texCoords = { 0.0f, 1.0f, 1.0f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f };
```

Takto naplněné pole znamená, že textura se bude vykreslovat na objekt v pořadí: levý horní, pravý horní, levý dolní a pravý dolní roh. Tedy ve stejném pořadí jako jsou

zadány body ve GL_VERTEX_ARRAY. Když jsou naplněna pole vertexů a koordinátů textury, předají se příslušným OpenGL proměnným pomocí ukazatelů. Po tomto programovém bloku je objekt definovaný v poli vertexů připraven k vykreslení.

```
fixed (float* vertexPtr = &vertex[0], texCoordsPtr = &texCoords[0])
{
      gl.EnableClientState(gl.GL_VERTEX_ARRAY);
      gl.VertexPointer(3, gl.GL_FLOAT, 0, (IntPtr)vertexPtr);
      gl.EnableClientState(gl.GL_TEXTURE_COORD_ARRAY);
      gl.TexCoordPointer(2, gl.GL_FLOAT, 0, (IntPtr)texCoordsPtr);
}
```

3.5 Systém osvětlení

V OpenGL existují základní 3 druhy světel s odlišnými vlastnostmi, které jsou zobrazeny na obrázku 9 [12]. Je možné definovat až 8 světel, jejich parametry se zapisují do symbolických proměnných GL_LIGHTx, kde $x \in \langle 0; 7 \rangle$. Intenzita jednotlivých barevných složek světla se udává na intervalu $\langle 0; 1 \rangle$ v modelu RGB (červená, zelená, modrá). Posledním parametrem je alpha a určuje celkovou intenzitu.

Obrázek 9 – Koule osvětlená pomocí ambient, diffuse a specular světla

- **Ambient** – Okolní světlo, osvětluje rovnoměrně všechny objekty na scéně, světelný zdroj vyzařuje světlo všemi směry stejnou intenzitou. Proto koule osvětlená tímto typem světla tvoří dojem 2D kruhu.

- **Diffuse** – Přímé světlo, vychází z jednoho bodu a v závislosti na směru, ve kterém na objekt dopadá, se daná stěna vykreslí světlejší či tmavší. Kromě pozice, kde se bude nacházet světelný zdroj, je třeba určit i směr, kterým se bude světlo emitovat.

- **Specular** – Složka světla, která určuje barvu a intenzitu odlesků. Ty zvýrazňují plasticitu objektů a výsledný dojem působí realisticky. Vypočty světelných odrazů u složitějších objektů jsou ale poměrně náročné na výkon.

Aby se světlo projevilo na scéně, je potřeba zapnout systém osvětlení příkazem:

```
gl.Enable(gl.GL_LIGHTING);
```

Následující kód zajistí nastavení dynamického difúzního světla čistě zelené barvy jako `GL_LIGHT0`. Je umístěné v bodě [20; 20; 30] (`light_position`) a světlo emituje ve směru osového středu (`light_direction`).

```
float[] light_diffuse = { 0f, 1.0f, 0f, 1.0f };

float[] light_position = { 20f, 20f, 30f, 1.0f };
float[] light_direction = { 0f, 0f, 0f};

fixed (float* light_difusePointer = &light_difuse[0],
              light_positionPointer = &light_position[0],
              light_directionPointer = &light_direction[0])
{
      gl.Lightfv(gl.GL_LIGHT0, gl.GL_DIFFUSE, light_difusePointer);
      gl.Lightfv(gl.GL_LIGHT0, gl.GL_POSITION, light_positionPointer);
      gl.Lightfv(gl.GL_LIGHT0, gl.GL_SPOT_DIRECTION,
                 light_directionPointer);
}
```

3.6 Blending

Blending je metoda, která prolíná barvu textury objektu s barvou pozadí na základě určité směšovací funkce. To znamená, že dojde k efektu zprůhlednění. Povolení blendingu se provede zavoláním metody:

```
gl.Enable(gl.GL_BLEND)
```

Dále je třeba nastavit funkci, podle které se budou barvy prolínat. Možných kombinací pro míchání je mnoho a umožňují různé variace průhlednosti. Jednotlivé barevné složky se násobí tímto faktorem a výsledná barva se použije při vykreslení.

```
void glBlendFunc(GLenum sFactor, GLenum dFactor);
```

Parametr sFactor (source) je zdrojový faktor, tedy pro objekt v popředí a dFactor (destination) je cílový faktor, ovlivňující pozadí. Možné parametry těchto faktorů a jejich matematický význam jsou uvedeny v tabulkách 1, 2 a 3.

Tabulka 1 – Funkce pro sFactor

Funkce	Význam
GL_DST_COLOR	barva cíle
GL_ONE_MINUS_DST_COLOR	1-barva cíle
GL_SRC_ALPHA_SATURATE	min (alpha zdroje, 1-alpha cíle)

Tabulka 2 – Funkce pro dFactor

Funkce	Význam
GL_SRC_COLOR	barva zdroje
GL_ONE_MINUS_SRC_COLOR	1-barva zdroje

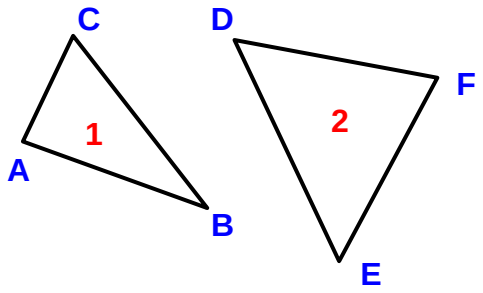
Tabulka 3 – Univerzální funkce

Funkce	Význam
GL_ZERO	0
GL_ONE	1
GL_SRC_ALPHA	alpha zdroje
GL_ONE_MINUS_SRC_ALPHA	1-alpha zdroje
GL_DST_ALPHA	alpha cíle
GL_ONE_MNUS_DST_ALPHA	1-alpha cíle

3.7 Vykreslování

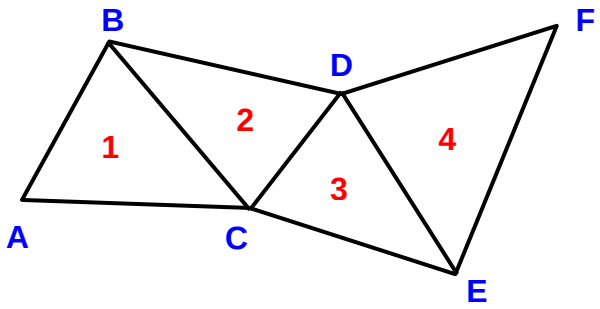
OpenGL může vykreslovat pomocí několika základních primitiv definovaných minimálně třemi vertexy. Vertex je obecně bod v prostoru, tedy definovaný třemi souřadnicemi (x, y, z). Základní stavební jednotkou plochy je trojúhelník a právě skládáním trojúhelníků vznikají objekty. Jednotlivé metody vykreslování se od sebe liší pořadím zadávaných vertexů a způsobem spojování. Před konstrukcí tělesa je proto třeba zvolit nejvhodnější z těchto způsobů.

- **GL_TRIANGLES** – trojúhelníky se vykreslují po třech bodech odděleně od sebe, nejsou automaticky spojovány, jak je vidět na obrázku 10.



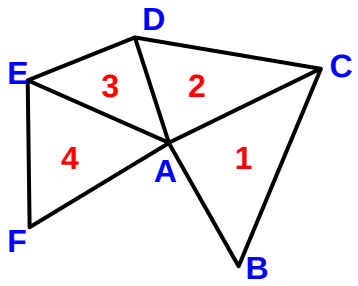
Obrázek 10 – Princip vykreslování pomocí GL_TRIANGLES

- **GL_TRIANGLE_STRIP** – pruh trojúhelníků se při vykreslování chová tak, že spojuje jednotlivé trojúhelníky vždy poslední vytvořenou hranou. Tento způsob se využívá i například k vykreslování čtyřúhelníků a různých mnohoúhelníků, znázornění viz obrázek 11.



Obrázek 11 – Princip vykreslování pomocí GL_TRIANGLE_STRIP

- **GL_TRIANGLE_FAN** – vějíř, všechny trojúhelníky mají společný první vrchol. Je vhodný pro vykreslování polygonů a jehlanů, viz obrázek 12.



Obrázek 12 – Princip vykreslování pomocí GL_TRIANGLE_FAN

Za předpokladu, že je `VERTEX_ARRAY` naplněno, respektive je mu předána reference na pole vertexů, lze jej vykreslit metodou `DrawArrays`, kde první parametr určuje typ primitivy, druhý udává pozici začátku ve vertexovém poli a třetím parametrem je počet vybíraných vertexů. Pomocí cyklů tak lze postupně iterovat a vykreslit například několik čtyřúhelníků z jednoho pole. Tento kód způsobí vykreslení prvních čtyř vertexů v poli pomocí `GL_TRIANGLE_STRIP`.

```
gl.DrawArrays(gl.GL_TRIANGLE_STRIP, 0, 4);
```

4 Použité knihovny a jejich metody

4.1 Windows Mobile 6 SDK

Aby bylo možné programovat PocketPC aplikace ve Visual Studiu, je nutné nainstalovat vývojářský balíček nástrojů Windows Mobile 6 SDK, který má 465 MB a je zdarma dostupný ke stažení na webu společnosti Microsoft [15].

Windows Mobile SDK existuje ve dvou verzích:

- **Standard:** Je určen pro smartphone zařízení bez dotykového displeje.
- **Professional:** Pro zařízení s dotykovým displejem – PocketPC, PDA, MDA, XDA.

Tato práce je zaměřena na dotykové přístroje, jejichž popularita prudce roste, dokazuje to podíl na trhu, který se za uplynulý rok 2009 zdvojnásobil. Do budoucna je v těchto zařízeních velký potenciál. Ovládání dotykovými gesty je intuitivní a aplikace takto ovládané jsou mnohem více interaktivní nežli ty ovládané klávesnicí. Pro vývoj tedy byla využívána verze Professional.

Po instalaci se ve Visual Studiu v nabídce vytvoření nového projektu objeví možnost Smart Device Project. Windows Mobile SDK umožňuje vyvíjené aplikace spouštět v emulátorech, ty ale nepodporují vykreslování pomocí OpenGL|ES a ani aplikace používající G-Senzor nebo GPS na nich není možné testovat. Pro takové případy Visual Studio nabízí možnost spustit a ladit program přímo na připojeném zařízení prostřednictvím spojení ActiveSync, tedy i bezdrátově přes Bluetooth.

4.2 GsensorSDK.dll

Pomocí moderního G-Senzoru lze určit, v jaké poloze se zařízení právě nachází, v praxi se využívá nejen v mobilních telefonech, ale i v digitálních fotoaparátech k otočení fotografie do polohy ve které byla pořízena nebo jako stabilizátor obrazu. V neposlední řadě ho najdeme také v krokoměrech a digitálních vodováhách. V mobilních telefonech umožňuje otáčet zobrazení na displeji v závislosti na poloze, ale navíc k němu mají prostřednictvím externí knihovny přístup i programátoři a využít se tak dá prakticky k jakýmkoliv účelům. Knihovnu GsensorSDK.dll vytvořil nezávislý programátor Scott Seligman [9]. V dnešní době tento senzor najdeme i v levnějších přístrojích bez operačního systému. V praktické části této práce jsou využívána výstupní data ze senzoru pro vhodnou transformaci scény, čímž je docíleno optické iluze 3D obrazu.

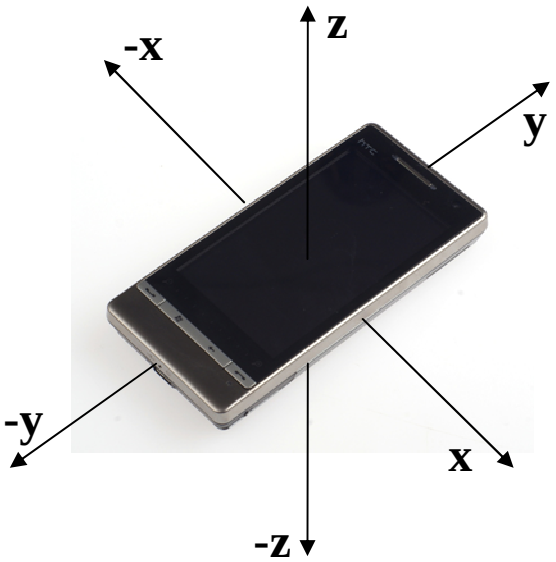
1. Aby bylo možné G-Senzor ve vlastním programu používat, musí se knihovna připojit do referencí a do programu includovat vložením následujícího řádku do hlavičky:

   ```
   using GSensorSDK;
   ```

2. Vytvoření instance třídy `HTCSensor`:

   ```
   HTCGSensor mySensor = new HTCGSensor();
   ```

Stěžejní metodou třídy `HTCSensor` je funkce `GetGVector()`, která vrací strukturu `GVector`, jenž nese atributy `X, Y, Z` typu `double`. Jejich hodnoty odpovídají aktuálnímu natočení přístroje. V našich fyzikálních podmínkách se jednotlivé hodnoty pohybují na intervalu <-9,8; 9,8> Ve standardní vodorovné poloze displejem nahoru například vrací funkce `GetGVector()` hodnoty x = 0, y = 0, z = -9,8. Osy akcelerometru jsou znázorněny na obrázku 13.



Obrázek 13 – Znázornění os akcelerometru

4.3 Location.dll

Tato knihovna slouží k získávání informací o poloze GPS. Technologie GPS se stává v mobilních telefonech standardem a navigační software tak směle konkuruje jednoúčelovým navigačním přístrojům. Pro určování polohy slouží 21 družic na oběžné dráze ve výšce 20183 km. Každá z nich obsahuje atomové hodiny s maximální odchylkou 3 ns a neustále vysílají signál nesoucí informace o jejich poloze a času odeslání. Každé místo na Zemi je pokryto šesti družicemi. K určení polohy stačí signály ze tří a pomocí čtvrté lze určit i nadmořskou výšku. Knihovna Location.dll je součástí Microsoft Windows Mobile 6 SDK a je implementována v ukázkové aplikaci GPS Demo. Rozšiřuje tak

možnosti využití HG-Engine i do oblasti navigačního software. Knihovna obsahuje několik tříd pro práci s GPS zařízením a polohou. Výpis základních z nich:

- **DegreesMinutesSeconds** – Slouží k uchování souřadnic ve formátu stupňů. Je možné získat i samostatné hodnoty stupňů, minut a sekund.
- **Gps** – Hlaní třída pro přístup ke GPS zařízení. Umožňuje otevřít komunikaci s GPS čipem a získávat od něho aktuální souřadnice.
- **GpsDeviceState** – Informace o výrobci, názvu a stavu GPS zařízení.
- **GpsPosition** – Třída pro geografické souřadnice. Pomocí jejích metod je možné získávat zeměpisnou šířku a délku ve stupních, ale i v desetinném formátu. Aktuální počet viditelných satelitů, rychlost nebo čas.
- **Satellite** – Informace o satelitu, jeho azimut a síla signálu.

Následující postup umožní ve vlastní aplikaci získávat data o aktuální geografické poloze a veškerých informacích, které GPS čip poskytuje:

1. Includování patřičné knihovny v hlavičce:

```
using Microsoft.WindowsMobile.Samples.Location;
```

2. Vytvoření instance objektu Gps a deklarace pomocné proměnné pro uchovávání aktuální pozice:

```
Gps gps = new Gps();
GpsPosition position = null;
```

3. Vytvoření procedury, která se bude volat při každé změně pozice. Zapíše aktuální pozici do proměnné position a pokud je GPS otevřené a souřadnice validní, zapíší se hodnoty zeměpisné šířky a délky do labelů.

```
protected void gps_LocationChanged(object sender,
LocationChangedEventArgs args)
{
position = args.Position;

if (gps.Opened)
{
      if (position != null)
      {
            if (position.LatitudeValid)
            {
                lblLatitude.Text = position.Latitude.ToString();
            }
            if (position.LongitudeValid)
            {
                lblLongitude.Text = position. Longitude.ToString();
            }
      }
}
```

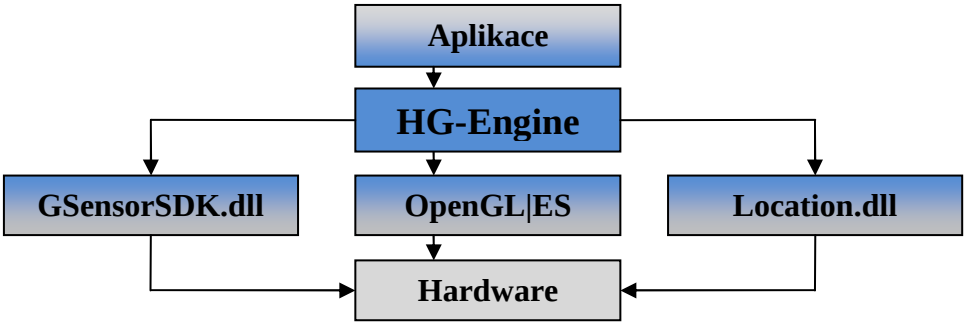
4. Napojení signálu změny polohy na vytvořenou proceduru a otevření GPS.

```
gps.LocationChanged += new
LocationChangedEventHandler(gps_LocationChanged);

if (!gps.Opened)
{
      gps.Open();
}
```

5 Vlastní implementace

5.1 Architektura

Na obrázku 14 je znázorněno blokové schéma umístění HG-Engine mezi jednotlivými vrstvami. Tento engine by měl umožnit programátorům tvoření 3D grafiky i s minimálními znalostmi OpenGL. Jedná se o jakousi šablonu, která obsahuje implementované další knihovny pro práci s 3D grafikou a komunikaci s vestavěnými senzory. Usnadní programátorům vývoj aplikací využívajících tyto technologie.



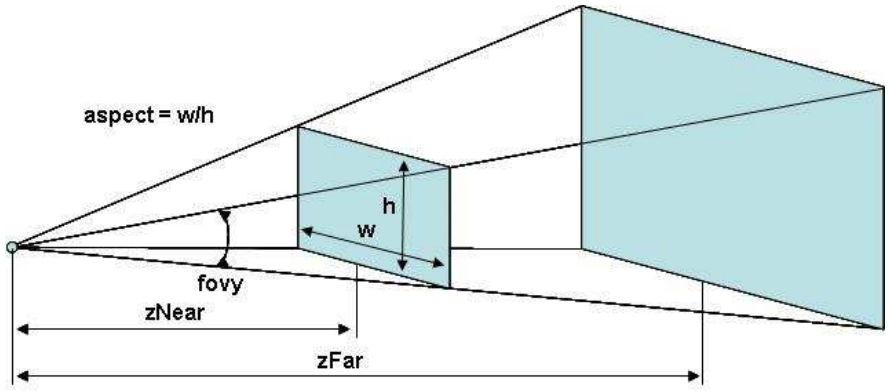
Obrázek 14 – Blokové schéma

5.2 Nastavení scény

Pro lepší orientaci byl zvolen osový střed na levý horní roh. Šířka a výška scény nastavena na setinu skutečného rozlišení displeje, čili X = 4,8 a Y = 8. Tím bylo dosaženo intuitivního zadávání souřadnic tak, jak je tomu u pozicování objektů na běžném formuláři pomocí atributů `Top` a `Left`. Protože pro OpenGL|ES neexistuje knihovna GLU, tak ze zdroje [1] byla použita procedura `gluPerspective`, která nastaví perspektivní projekci. Grafické znázornění jednotlivých parametrů je k dispozici na obrázku 15 [6].

```
void gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble zNear,
GLdouble zFar);
```

- **fovy** (field of view) – zorný úhel kamery, běžně se používá 45°
- **aspect** – poměr stran (šířka / výška)
- **zNear** – vzdálenost bližší ořezávací roviny ve směru osy Z
- **zFar** – vzdálenost vzdálenější ořezávací roviny



Obrázek 15 – Znázornění parametrů gluPerspective

Pokud není s kamerou manipulováno, tak se osový střed nachází v základní pozici, čili uprostřed obrazovky. Metoda `Translatef` způsobí posun scény o určitou vzdálenost, v tomto případě o polovinu šířky a výšky, aby se osový střed dostal do levého horního rohu a kamera se přesunula do vzdálenosti z = 9.

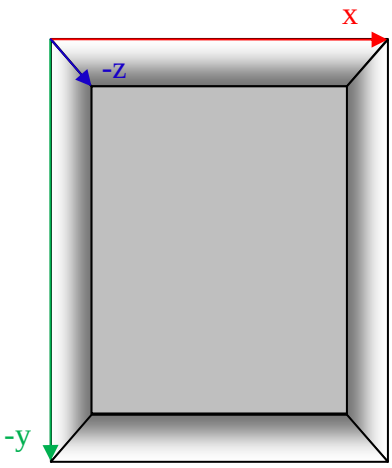
```
gl.Translatef(-width / 2, -height / 2, -9f);
```

Nyní je scéna nastavena tak, jak je znázorněno na obrázku 16 a lze jednoznačně určit hranice virtuálního prostoru. Podle nového souřadnicového systému se v jednotlivých rozích displeje nacházejí body:

- Levý horní [0; 0; 0]
- Pravý horní [x; 0; 0]
- Levý dolní [0; y; 0]
- Pravý dolní [x; y; 0]

Zadní stěna ve vzdálenosti „z“ je definována body:

- Levý horní [0; 0; -z]
- Pravý horní [x; 0; -z]
- Levý dolní [0; y; -z]
- Pravý dolní [x; y; -z]



Obrázek 16 – Umístění počátku souřadnicového systému

5.3 Objekty

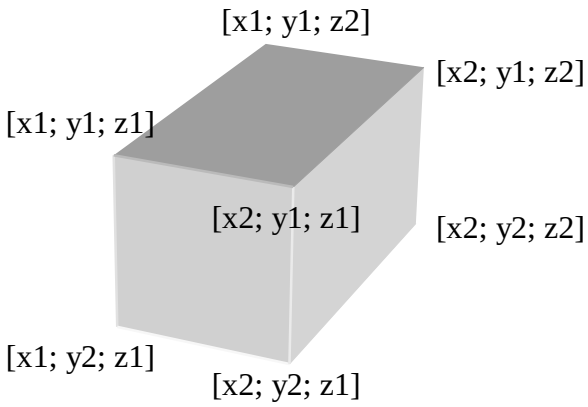
Pro zjednodušené modelování základních těles byly vytvořeny vlastní třídy, které umožňují vytvářet například kvádr zadáním pouze nezbytně nutných parametrů. O dopočítání polohy všech bodů se již starají vnitřní metody dané třídy. V mobilním OpenGL je totiž bez podpůrných knihoven i vytvoření základních těles pro programátora časově a početně velmi náročné. Navrhnout tuto architekturu tak byl základní kámen celého projektu.

5.3.1 HG_Block

`HG_Block` je třída pro obecný kvádr, která obsahuje následující atributy pro uchovávání základních dat o poloze a velikosti tělesa:

- `private float x1, x2, y1, y2, z1, z2;`
  základní souřadnice vrcholů, podrobnější vysvětlení nabízí obrázek 17.
- `private float width, height, deep;`
  vypočítaná šířka, výška a hloubka tělesa
- `private float transparency`
  průhlednost objektu použitá při blendingu

Přední stěnu tvoří body s hloubkou `z1`, zadní `z2`. Levý horní roh má vždy souřadnice [`x1; y1`], pravý dolní pak [`x2, y2`]. Z toho plynou jednoduché výpočty šířky (`x2-x1`), výšky (`y2-y1`) a hloubky tělesa (`z2-z1`).



Obrázek 17 – Označení vertexů objektu HG_Block

Metody pro základní operace poskytují jednoduchou manipulaci s kvádrem. Na základě předaných argumentů provedou přepočet atributů na novou polohu.

- `public HG_Cube(float x1_input, float x2_input, float y1_input, float y2_input, float z1_input, float z2_input)`
  Konstruktor, kterému se předávají parametry definující polohu a velikost kvádru.

- `public void Move(float x_move, float y_move, float z_move)`
  Procedura, která pohybuje objektem o určitý počet jednotek, tzv. relativní posun. V těle této procedury se pouze upraví současné souřadnice. Předávané parametry udávají velikost posunu na jednotlivých osách.

- `public void MoveToPos(float x_move, float y_move)`
  Zajišťuje pohyb objektem na přesné souřadnice, předané parametry, tzv. absolutní pozice.

- `public float[] Draw (float vcx_input, float vcy_input)`
  Funkce `Draw` vypočítá souřadnice všech vertexů daného objektu a vrací `float` pole pro vykreslení.

- `public bool IsInPosition(float x_input, float y_input)`
  Funkce vrací `true` nebo `false`, v závislosti na tom, zda se objekt nachází na vstupních souřadnicích.

- `public void CorrectPositions()`
  Tato procedura zajistí správné řazení atributů a to tak, aby platilo `x1<x2`, `y1<y2` a `z1<z2`. To v případě, že jsou souřadnice konstruktoru předány v jiném pořadí, aby se objekt vykreslil správně.

- `public void ApplyPhysics(float vcx_input, float vcy_input, float vcz_input)`
  Procedura, která na objekt aplikuje při každém vykreslení fyziku, její tělo je popsáno v kapitole 5.5 Fyzika.

Následuje podrobnější popis funkce `Draw`, která má za úkol vypočítat souřadnice jednotlivých bodů v závislosti na gravitačním vektoru. Jednotlivé proměnné se kterými pracuje (`x1, x2, y1, y2, z1, z2`) jsou atributy objektu `HG_Block`, vysvětlené výše. Vstupní proměnné `vcx` a `vcy` jsou upravené hodnoty gravitačního vektoru, podle následujícího vzorce, kde `sensitivity` určuje senzitivitu senzoru. Osvědčila se hodnota 20, při které se zobrazení jeví jako nejvíce věrohodné.

```
vcx = -(float)(mySensor.GetGVector().X) / sensitivity;
```

Každá souřadnice jednotlivých vertexů je ve funkci `Draw` přepočítána a posunuta o násobek gravitačního vektoru na příslušné ose (x nebo y) a hloubky (z) daného vertexu. Tím je zajištěno, že vzdálenější body se budou posouvat více, než ty umístěné blízko displeje. Z toho plyne, že body s nulovou souřadnicí „z“ jsou statické, čili nehýbou se.

Navržení a sestavení funkce `Draw` bylo časově velmi náročné a prošla mnoha změnami, než se dosáhlo její konečné podoby. Kvádr má 6 stěn, na každou z nich jsou potřeba 4 vertexy, tzn. 24 bodů a každý má 3 souřadnice (x, y, z), dohromady tedy kvádr definuje 72 číselných hodnot, které musejí být ve float poli v přesném pořadí, jinak nebude vykreslen správně. Každý řádek určuje souřadnice jednoho bodu. Tyto řádky tvoří bloky započaté komentářem, které definují jednotlivé stěny.

```
public float[] Draw (float vcx_input, float vcy_input)
{
float[] output = new float[]
{
      //Zadní
      x1 + z2 * vcx_input, y1 + z2 * vcy_input, z2,
      x2 + z2 * vcx_input, y1 + z2 * vcy_input, z2,
      x1 + z2 * vcx_input, y2 + z2 * vcy_input, z2,
      x2 + z2 * vcx_input, y2 + z2 * vcy_input, z2,


      //Přední
      x1 + z1 * vcx_input, y1 + z1 * vcy_input, z1,
      x2 + z1 * vcx_input, y1 + z1 * vcy_input, z1,
      x1 + z1 * vcx_input, y2 + z1 * vcy_input, z1,
      x2 + z1 * vcx_input, y2 + z1 * vcy_input, z1,


      //Vrchní
      x2 + z1 * vcx_input, y1 + z1 * vcy_input, z1,
      x1 + z1 * vcx_input, y1 + z1 * vcy_input, z1,
      x2 + z2 * vcx_input, y1 + z2 * vcy_input, z2,
      x1 + z2 * vcx_input, y1 + z2 * vcy_input, z2,
```

```
        //Levá
        x2 + z1 * vcx_input, y2 + z1 * vcy_input, z1,
        x2 + z1 * vcx_input, y1 + z1 * vcy_input, z1,
        x2 + z2 * vcx_input, y2 + z2 * vcy_input, z2,
        x2 + z2 * vcx_input, y1 + z2 * vcy_input, z2,


        //Spodní
        x1 + z1 * vcx_input, y2 + z1 * vcy_input, z1,
        x2 + z1 * vcx_input, y2 + z1 * vcy_input, z1,
        x1 + z2 * vcx_input, y2 + z2 * vcy_input, z2,
        x2 + z2 * vcx_input, y2 + z2 * vcy_input, z2,


        //Pravá
        x1 + z1 * vcx_input, y1 + z1 * vcy_input, z1,
        x1 + z1 * vcx_input, y2 + z1 * vcy_input, z1,
        x1 + z2 * vcx_input, y1 + z2 * vcy_input, z2,
        x1 + z2 * vcx_input, y2 + z2 * vcy_input, z2
};
return output;
}
```

Výstupem funkce je tedy `float` pole o 72 prvcích. Takto připravené pole se pomocí ukazatele předá do `VERTEX_ARRAY`, viz kapitola 3.7. Následující cyklus je používán pro vykreslení kvádru, kdy se volá procedura `DrawArrays` pro 0 <= i < 6. V této části je nutné správně navrhnout interval iterace, při špatném zvolení by při běhu programu docházelo k neočekávanému ukončení z důvodu nativní vyjímky.

```
for (int i = 0; i < 6; i++)
{
        gl.DrawArrays(gl.GL_TRIANGLE_STRIP, i * 4, 4);
}
```

5.3.2 HG_4Prism

Tato třída slouží k vykreslení nepravidelného čtyřbokého hranolu. Zadávání souřadnic se poněkud liší od kvádru. Hranol definují 4 body a dvě souřadnice hloubky. Do konstruktoru bylo navrženo zadávání bodů v pořadí podle směru hodinových ručiček. Atributy třídy HG_4Prism:

- `private float x1, x2, y1, y2, x3, y3, x4, y4, z1, z2;`
  Jelikož je hranol nepravidelný, tak pro každý vrchol existují samostatné souřadnice.

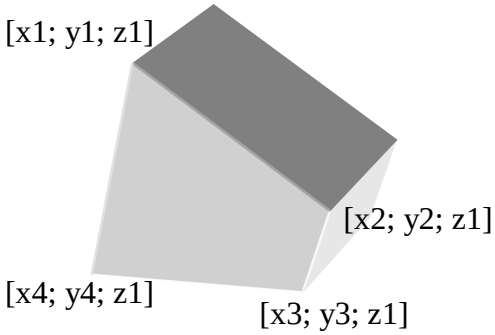
Metody hranolu jsou podobné těm, které má kvádr. Jejich těla se pouze nepatrně liší:

- `public HG_4Prism(float x1_input, float y1_input, float x2_input, float y2_input, float x3_input, float y3_input, float x4_input, float y4_input, float z1_input, float z2_input)`
  Konstruktor, kterému se předávají všechny potřebné souřadnice. Uvnitř se nastaví jednotlivé atributy objektu.

- `public float[] Draw (vcx_input, vcy_input)`
  Funkce pro vypočtení souřadnic vertexů k vykreslení.

- `public void Move(x_move, y_move, z_move)`
  Relativní posun objektu o určenou vzdálenost viz `HG_Block`.

- `public void MoveToPos(x_move, y_move)`
  Nastavení absolutní pozice, tedy konkrétních souřadnic.



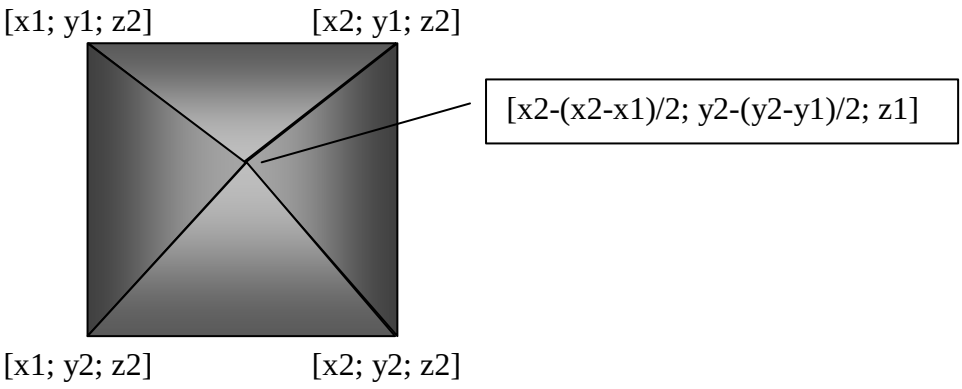
Obrázek 18 – Označení vertexů objektu HG_4Prism

5.3.3 HG_Pyramid

Třída pro definici čtyřbokého jehlanu s obdélníkovou podstavou, stěny jsou tvořeny trojúhelníky. Atributy definují pouze podstavu a výšku, poloha vrcholu se automaticky dopočítává podle vzorce na obrázku 19.

- `private float x1, x2, y1, y2;`

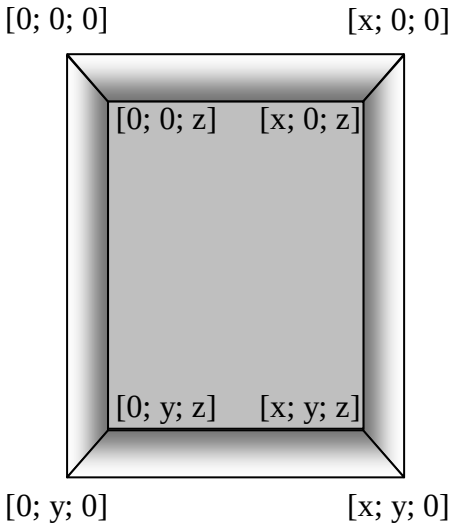
Metody jsou totožné jako u předchozích těles, proto není třeba je znovu uvádět.



Obrázek 19 – Označení vertexů objektu HG_Pyramid

5.3.4 HG_Environment

Okolní prostředí, které ohraničí scénu. Je tvořeno pozadím, což je stěna rovnoběžná s displejem v určené vzdálenosti. Okolní čtyři stěny jsou vzhledem k pozadí kolmé a spojují ho s rohy displeje.



Obrázek 20 – Označení vertexů objektu HG_Environment

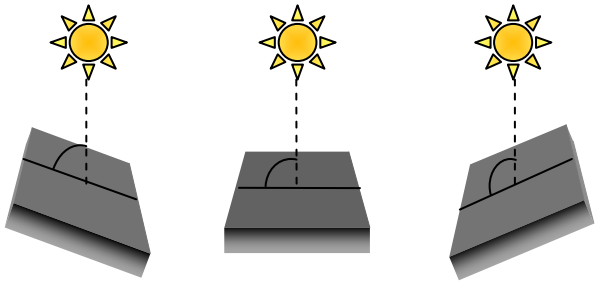
Funkce `Draw` nejprve vypočítá souřadnice pozadí a poté ostatní stěny v pořadí: levá, pravá, vrchní, spodní.

```
public float[] Draw (float vcx_input, float vcy_input)
{
float[] output = new float[]
{
//Pozadí
z * -vcx_input, z * -vcy_input, z,
x + z * -vcx_input, 0 + z * -vcy_input, z,
0 + z * -vcx_input, y + z * -vcy_input, z,
x + z * -vcx_input, y + z * -vcy_input, z,

//Levá
0, 0, 0,
0, y, 0,
0+z*-vcx_input, 0+z*-vcy_input, z,
0+z*-vcx_input, y+z*-vcy_input, z,

//Pravá
x, 0, 0,
x, y, 0,
x+z*-vcx_input, 0+z*-vcy_input, z,
x+z*-vcx_input, y+z*-vcy_input, z,

//Vrchní
0, 0, 0,
x, 0, 0,
0+z*-vcx_input, 0+z*-vcy_input, z,
x+z*-vcx_input, 0+z*-vcy_input, z,
```

```
        //Spodní
        0, y, 0,
        x, y, 0,
        0+z*-vcx_input, y+z*-vcy_input, z,
        x+z*-vcx_input, y+z*-vcy_input, z

        };

return output;
}
```

5.4 Osvětlení

Aby scéna vypadala reálněji, je zapotřebí zvolit vhodné osvětlení. Implementace systému světel byla jedna z náročnějších částí. Nicméně správné nasvícení scény je v 3D grafice velmi důležité. Jelikož HG-Engine úzce interaguje s okolím, tato interakce se dotkla právě i osvětlení. Zdroj světla vně zařízení, který je ve skutečnosti statický, se při náklonu mobilu vzhledem k němu pohybuje. Když se tedy mění úhel pohledu na scénu, měla by se změnit i poloha světla vůči scéně tak, jak je to vyobrazeno na obrázku 21. Je tak zřejmé, že použité světlo nemůže být statické, nýbrž dynamické a jeho pohyb musí být závislý na G-Senzoru. Důležité je také zmínit, že osvětlení a výpočet odlesků je výrazně náročný na výkon grafického čipu.



Obrázek 21 – Znázornění změny úhlu dopadajícího světla v závislosti na poloze

Zároveň je třeba nastavit vlastnosti materiálu, na který světlo dopadá. To probíhá obdobně jako nastavení světel. U texturovaných objektů jejich barvu určuje právě textura, proto je pro materiál nastaveno, že odráží veškeré světlo, které na něj dopadá.

5.5 Fyzika

Gravitační senzor umožňuje mimo jiné i pohybovat objekty na scéně nakloněním přístroje tak, aby pohyb respektoval fyzikální zákony. Dalším rozšířením tedy byla implementace jednoduché fyziky do jedné z ukázkových aplikací. Třída `HG_Block` byla rozšířena o další atributy:

- `private float ro, V, m;`

  Fyzikální vlastnosti: hustota, objem a hmotnost.

Třída také byla rozšířena o přetížený konstruktor pro fyziku, jenž má navíc argument `ro`, který nastaví objektu atribut hustoty rovnou při konstrukci. Vynásobením `ro` a objemu tělesa pomocí známého vzorce m=ρ*V se vypočítá hmotnost tělesa.

O vše ostatní se stará procedura `ApplyPhysics`, která na těleso aplikuje gravitaci. Předávají se jí 3 argumenty – gravitační vektor ve směrech x, y, z. Jejím úkolem je vypočítat v závislosti na hmotnosti tělesa jeho směr a rychlost pohybu. Výsledek výpočtu se předá proceduře `Move`, která na základě argumentů posune těleso o určenou vzdálenost.

Bylo třeba ošetřit, aby objekty nevstupovaly do stěn a nepropadávaly tak mimo viditelnou scénu. Proto se při každém aplikování fyziky kontrolují případné kolize se stěnami vnitřního prostředí. Tím se dosáhlo toho, že pokud se těleso dotkne jedné ze stěn, jeho pohyb v tom směru se zastaví, ovšem může se i nadále pohybovat v ostatních směrech, ve kterých není omezeno. Vzájemná kolize mezi objekty není naprogramována. Následuje tělo procedury `ApplyPhysics`, která se stará při každém vykreslení o změnu polohy ve směru gravitace.

```
public void ApplyPhysics(float vcx_input,float vcy_input,float vcz_input)
{

V = (height/100) * (width/100) * (deep/100);
m = ((V * ro)/100);

if (Physics == true)
{
      if ((x1 >= 0f) & (x2 <= 4.8f)) Move(vcx_input * m, 0, 0);
      if ((y1 <= 0f) & (y2 >= -7.48f)) Move(0, vcy_input * m, 0);
      if ((z1 <= 0f) & (z2 >= -2f)) Move(0, 0, vcz_input * m);

      //detekce kolizí

      if (x1 < 0f){ SetX1(0); SetX2(width);};
      if (x2 > 4.8f) { SetX1(480 - (width)); SetX2(480); };

      if (y1 > 0f) { SetY1(0); SetY2(height); };
      if (y2 < -7.48f) { SetY1(748 - height); SetY2(748); };

      if (z2 < -2) { SetZ2(-200); SetZ1(-200 + deep); }
      if (z1 > 0) Move(0, 0, -vcz_input * m);
}
}
```

6 Ukázkové aplikace

6.1 Tech-Demo

Technologické demo vždy prezentuje nejnovější implementované algoritmy. Nemá žádné konkrétní využití a v současné verzi disponuje následujícími funkcemi. Ukázková aplikace Tech-demo je k dispozici jako freeware, aby se otestovala kompatibilita s ostatními zařízeními. Funkčnost je zatím potvrzená na přístrojích: HTC Touch Pro 2, HTC Touch HD, HTC HD2, HTC Touch Pro, HTC Touch Diamond, HTC Touch Diamond 2, HTC Pure, Sony Ericsson Xperia X1, Samsung Omnia 2. Ke dni 9. 5. 2010 čítá 5281 stažení na serveru www.freewarepocketpc.net. Ohlasy jsou velice pozitivní a komunita žádá další vývoj.

- **Modelování těles**
    - **Kvádr** – Vytvoření kvádru probíhá ve třech krocích. Nejprve se stisknutím a tažením po displeji definuje podstava, dále tažením po ose x se určí vzdálenost přední stěny a v posledním kroku hloubka zadní stěny.
    - **Čtyřboký hranol** – Jelikož je nepravidelný, zadávají se postupně jeho čtyři vrcholy podstavy kliknutím ve směru hodinových ručiček. Hloubka přední a zadní stěny stejným způsobem jako u kvádru.
    - **Čtyřboký jehlan** – Podstava hranolu se vytváří totožným způsobem jako u kvádru. Následně tažením prstu po displeji ve směru osy x se určí vzdálenost podstavy a nakonec hloubka vrcholu jehlanu.
- **Manipulace s objekty**
    - **Výběr** – V programu je definována proměnná pro aktuální objekt, výběr probíhá kliknutím na požadovaný objekt. Program následně prochází souřadnice všech objektů a kontroluje, zda se některý z nich nachází na dané pozici. Pokud ano, takový objekt se nastaví jako aktuální.
    - **Posun** – Při kliknutí na objekt se automaticky vybere a stane aktuálním, tažením prstu po displeji je možné objektem libovolně pohybovat po osách x a y.
    - **Vymazání** – Pokud je vybrána tato funkce, tak kliknutím na objekt se nenávratně vymaže ze scény.
- **Funkce prostředí**
    - **Osvětlení** – Zapíná a vypíná dynamické osvětlení. Vypnutím světla při velkém počtu objektů na scéně se může výrazně zvýšit FPS.

- **Fyzika** – Aktivuje nebo deaktivuje fyziku. Při zapnutí se začnou objekty pohybovat ve směru gravitace, rychlost pádu závisí na hmotnosti tělesa a míře naklonění přístroje.

Na obrázku 22 je vlevo scéna s vypnutým osvětlením, uprostřed ta samá situace se zapnutým osvětlením. Vpravo je totožná scéna, kdy přístroj byl natočen doprava, čili pohled zleva.

Obrázek 22 – Ukázky implementace osvětlení v aplikaci Tech-Demo

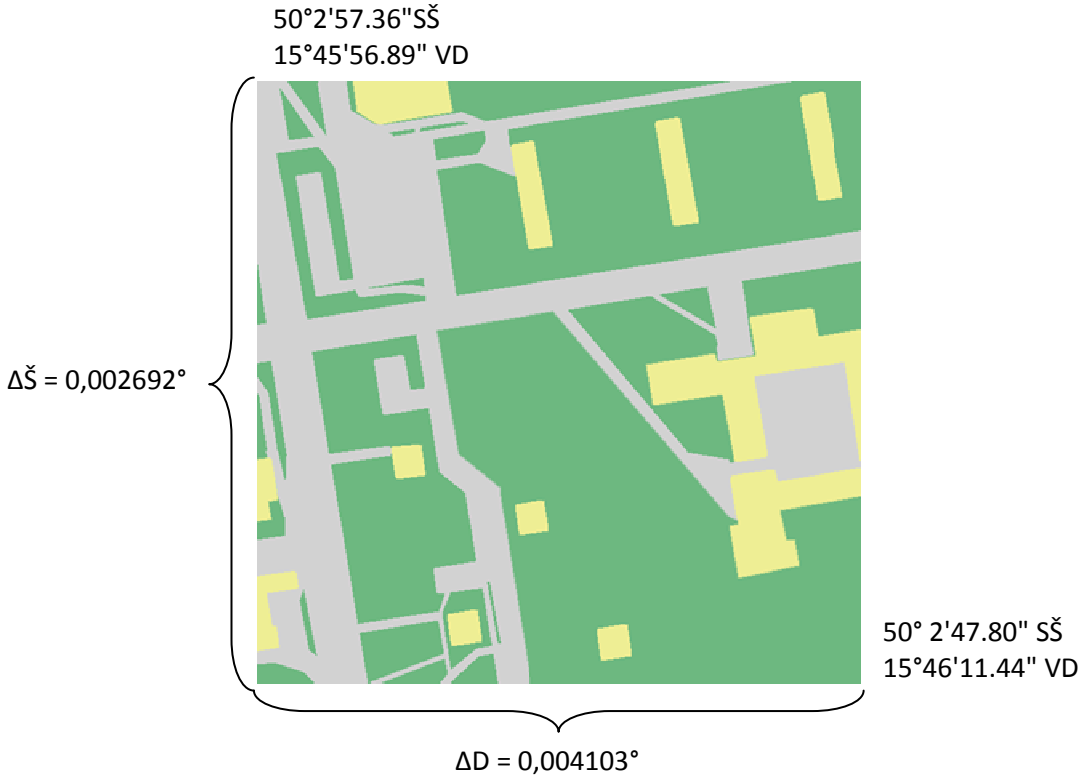
6.2 Pardubice GPS Demo

Tato ukázková aplikace má zabudovanou podporu GPS. Jako mapový podklad byl zvolen kampus Univerzity Pardubice pro jednodušší testování a kalibraci. Nebylo možné použít skutečné mapové podklady z důvodu přísných podmínek užití těchto dat. Jednotlivé budovy kolejí jsou vymodelovány ve 3D a úhel pohledu lze měnit nakloněním přístroje. Další funkce jsou přístupné z menu, viz obrázek 24.

- **Zoom** – Tažením prstu po ose x je možné pohled přibližovat a oddalovat.
- **Posun** – Umožňuje pohyb po mapě prstem v libovolném směru.
- **Moje poloha** – zaměří pohled na aktuální polohu a při pohybu se plynule mění tak, že bod je vždy uprostřed obrazovky.
- **G-Senzor** – Povolení/zakázání akcelerometru pro potlačení otřesů při chůzi.

Zásadním problémem byla kalibrace mapy, respektive přepočet reálných GPS souřadnic na pixely. Levý horní roh mapy (autobusová zastávka Polabiny, Hradecká) byl umístěn do bodu [0; 0; 0]. Pomocí programu Google Earth byly zjištěny přesné souřadnice

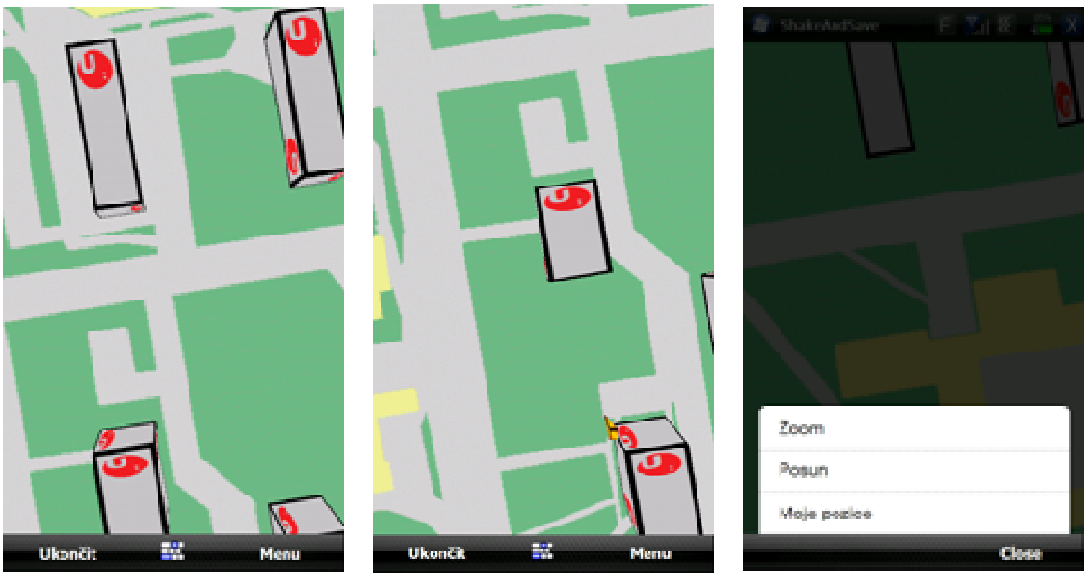
místa, kde mapa začíná a tyto jsou během výpočtu odečítány od aktuální polohy, výsledkem jsou relativní souřadnice. Geografický rozsah výřezu mapy je označen na obrázku 23. Dále bylo třeba určit poměr, kterým se relativní souřadnice budou násobit, aby se poloha správně zobrazila na mapě.



Obrázek 23 – Rozsah výřezu mapy, včetně delta hodnot

Rozlišení mapy bylo zvoleno 670×670 px, v OpenGL její velikost nastavena na 20,1×20,1 (20,1/670 = **0,03**, odůvodněno v následujícím odstavci). Poměry pro zeměpisnou šířku a výšku se vypočítají jednoduchým vztahem. Těmito hodnotami se poté násobí relativní souřadnice a výsledkem je poloha na virtuální mapě, kam se přesune jehlan žluté barvy (pomocí procedury `Move`), který pozici zobrazuje, viz obrázek 24 uprostřed.

poměr šířky = 20,1 / 0,002692 = 7467
poměr délky = 20,1 / 0,004103 = 4899



Obrázek 24 – Ukázky přímo z aplikace GPS Demo

Budovy kolejí jsou vymodelovány jako čtyřboké hranoly, pro umístění byl zvolen přepočet, aby nebylo složité ze souřadnic v pixelech na mapě vypočítat pozici daného bodu v prostoru OpenGL. Při vytváření jednotlivých budov se předávají konstruktoru souřadnice rovné trojnásobku souřadnic pixelu v souboru mapy. V těle konstruktoru se následně dělí stem, proto **0,03**. Zjednodušeně řečeno bod, který se nachází v bitmapě na souřadnicích [100; 200], se do konstruktoru objektu zadá jako [300; 600] a jeho skutečná poloha v prostoru OpenGL je [3; 6]. Plocha s mapou se nachází v hloubce z = 0, tudíž je statická a při pohybu gravitačního senzoru se mění pouze poloha bodů vrchních stěn budov, kde z > 0.

7 Shrnutí

Projekt se stále nachází ve fázi alpha verze, to znamená zatím uzavřený vývoj a vydávám pouze ukázkové demo aplikace pro testování kompatibility, odezvy uživatelů, atp. Neustále přidávám nové funkce a rozšiřuji možnosti využití. V rámci této práce jsem dosáhl všech předem stanovených cílů. Dokázal jsem realizovat veškeré algoritmy, které jsem předem teoreticky navrhl. Následující výčet vlastností je seřazen v pořadí, ve kterém byly jednotlivé prvky vyvíjeny.

- Implementace OpenGL|ES
- Implementace G-Senzoru
- Vlastní třídy základních geometrických těles
- Interakce scény na základě G-Senzoru
- Dynamické osvětlení
- Dynamické přidávání a odebírání objektů za běhu programu
- Implementace GPS
- Jednoduchá fyzika

Jako testovací zařízení pro vývoj mi posloužilo HTC Touch Pro 2 (Rhodium), tento přístroj disponuje ARM procesorem s frekvencí 528 MHz, 288 MB operační paměti. Čipová sada Qualcomm zahrnuje i dedikovaný grafický čip s 64 MB RAM a nativní podporou OpenGL|ES 1.1. K ovládání přístoje slouží dotykový displej s rozlišením WVGA.

Nejvýznamnějším pokrokem byla bezesporu implementace GPS, včetně zobrazení aktuální polohy na mapě vymodelované pomocí naprogramovaných algoritmů. Dalším rozšiřováním bude možno například měřit rychlost automobilu a pomocí ní vypočítat ideální pohled a přiblížení mapy. To znamená, aby se při vysoké rychlosti zobrazovala větší část mapy a naopak. Jedním z možných využití je tedy pokročilý 3D software pro navigaci. V současné době se stávají velmi populárními systémy rozšířené reality, které umožňují uživateli prostřednictvím fotoaparátu v reálném čase získávat informace o okolních budovách, obchodech, kulturních památkách, atd. HG-ngine by mohl získat uplatnění i v takto zaměřených programech.

Systém Windows Mobile je často kritizován kvůli grafickému rozhraní a ovládání, které není příliš uživatelsky přívětivé. Z tohoto důvodu výrobci do svých zařízení instalují nadstavby uživatelského prostředí, umožňující pohodlnější ovládání funkcí telefonu. HG-Engine by s dalším rozšiřováním mohl být využit i pro vývoj moderního 3D grafického rozhraní, ovládaného dotykovými a pohybovými gesty. Pro rychlý přístup k důležitým funkcím telefonu bude stačit naklonění přístroje do určitého směru.

Během vývoje jsem se potýkal s mnoha problémy, jelikož na začátku jsem neměl žádné znalosti a zkušenosti s programováním pro mobilní zařízení ani s OpenGL. Učil jsem se převážně z internetových zdrojů a pečlivým studováním zdrojových kódů, které nebylo vždy snadné pochopit. Většina použitých algoritmů prošla několika optimalizacemi. Když mě napadlo nějaké rozšíření, snažil jsem se ho implementovat co nejrychlejším způsobem, který ale nebyl vždy ideální a korektní. V případě, že jsem dosáhl úspěšného řešení daného problému, tak jsem přistoupil k optimalizacím a ladění výkonu. V této fázi již je možné postavit na enginu složitější aplikace a to měl být jeho hlavní účel. Práce především rozšířila mé obzory v oblasti 3D grafiky, která je mi blízká a chci se jí v budoucnu zabývat. Stejně tak programováním pro mobilní zařízení, kde se s příchodem nových operačních systémů otevírá softwarový trh.

Přiložený CD-ROM obsahuje kompletní zdrojové kódy aplikací Tech-Demo a GPS-Demo, včetně zkompilovaných programů.

Literatura

[1] *Xda-developers* [online]. 2009 [cit. 2010-05-11]. [App + Src + Game] Test OpenGL and OpenVG performance; Game using OpenGL Windows Mobile Software Development. Dostupné z WWW: <http://forum.xda-developers.com/showthread.php?t=511363>.

[2] **AHN, Song Ho.** *Song Ho Ahn* [online]. 2008 [cit. 2010-05-11]. OpenGL Projection Matrix. Dostupné z WWW: <http://www.songho.ca/opengl/gl_projectionmatrix.html>.

[3] **BABČANÍK, Jan**. *HW.cz* [online]. 2006 [cit. 2010-05-11]. Jak funguje GPS?. Dostupné z WWW: <http://hw.cz/Teorie-a-praxe/ART1634-Jak-funguje-GPS.html>.

[4] **DUTTA, Koushik**. *My Brain Hurts* [online]. 2008 [cit. 2010-05-11]. .NET Compact Framework wrapper for OpenGL ES. Dostupné z WWW: <http://www.koushikdutta.com/2008/08/net-compact-framework-wrapper-for.html>.

[5] **DUTTA, Koushik**. *My Brain Hurts* [online]. 2008 [cit. 2010-05-11]. Using HTC Touch Diamond's Accelerometer/Sensor SDK from Managed Code. Dostupné z WWW: <http://www.koushikdutta.com/2008/07/using-htc-touch-diamond-sensor-sdk-from.html>.

[6] **JARNICKI, Jacek**. *Zakład Systemów Komputerowych* [online]. 2005 [cit. 2010-05-11]. Ćwiczenie 9-10. Dostupné z WWW: <http://www.lab229.ict.pwr.wroc.pl/zsk/dydaktyka/gk/labz/cw_9_10_zao/>.

[7] **MILLER, Nate**. *GameDev.net* [online]. 2000 [cit. 2010-05-11]. OpenGL Texture Mapping: An Introduction. Dostupné z WWW: <http://www.gamedev.net/reference/articles/article947.asp>.

[8] **MOLOFEE, Jeff**. *NeHe Productions* [online]. 2006 [cit. 2010-05-11]. OpenGL Lessons. Dostupné z WWW: <http://nehe.gamedev.net>.

[9] **SELIGMAN, Scott**. *Scott's Weblog* [online]. 2008 [cit. 2010-05-11]. Comments. Dostupné z WWW: <http://scottandmichelle.net/scott/comments.html?entry=784>.

[10] **SHARP, John**. *Microsoft Visual C# 2008 Krok za krokem*. Brno : Computer Press, 2008. 592 s. ISBN 978-80-251-2027-9.

[11] **SHREINER, Dave, et al**. *OpenGL. Průvodce programátora*. Brno : Computer Press, 2006. 696 s. ISBN 80-251-1275-6.

[12] **SIDELNIKOV, Greg**. *Fallout Software* [online]. 2003 [cit. 2010-05-11]. OpenGL Lighting or How Light Sources Work. Dostupné z WWW: <http://www.falloutsoftware.com/tutorials/gl/gl8.htm>.

[13] **TIŠNOVSKÝ, Pavel**. *Root.cz* [online]. 2003 [cit. 2010-05-11]. Grafická knihovna OpenGL. Dostupné z WWW: <http://www.root.cz/clanky/graficka-knihovna-opengl-1>.

[14] **TUREK, Michal, et al**. *CZ NeHe OpenGL* [online]. 2008 [cit. 2010-05-11]. NeHe OpenGL Tutoriály. Dostupné z WWW: <http://nehe.ceske-hry.cz>.

[15] *Microsoft Corporation* [online]. 2007 [cit. 2010-05-11]. (14) Windows Mobile 6 SDK Refresh. Dostupné z WWW: <http://www.microsoft.com/downloads/details.aspx?FamilyID=06111A3A-A651-4745-88EF-3D48091A390B&displaylang=en>.