Interaktivní prezentace 3D objektů

BcA. Jan HŮLA

Diplomová práce
2011

Univerzita Tomáše Bati ve Zlíně
Fakulta multimediálních komunikací

Univerzita Tomáše Bati ve Zlíně
Fakulta multimediálních komunikací
Ústav prostorového a produktového designu
akademický rok: 2010/2011

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**BcA. Jan HŮLA**
Osobní číslo:	**K08367**
Studijní program:	**N 8206 Výtvarná umění**
Studijní obor:	**Multimedia a design – Průmyslový design**
Téma práce:	**Interaktivní prezentace 3D objektů v internetovém prohlížeči**

Zásady pro vypracování:

1. **Historický vývoj vizualizace**
2. **Analýza současných technologií**
3. **Koncepce systému**
4. **Zpracování systému**
5. **Funkční prototyp systému**
6. **Vypracování doprovodné zprávy odůvodňující navržené řešení**

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/umělecké dílo**

Seznam odborné literatury:

GORDON,Ian, Theories of Visual Perception, New York: Psychology Press, 2004, ISBN: 0-203-50225-6
BUSS, Samuel, 3-D Computer Graphics-A Mathematical Introduction with OpenGL, New York: Cambridge University Press, 2003, ISBN-10: 0-511-07850-1
MOOCK, Colin, Essential ActionScript 3.0, Sebastopol: O?Reilly Media, 2007, ISBN-10: 0-596-52694-6
SANDERS, William, ActionScript 3.0 Design Patterns, Sebastopol: O?Reilly Media, 2007, ISBN-10: 0-596-52846-9

Vedoucí diplomové práce: **prof. ak. soch. Pavel Škarka**
Ústav prostorového a produktového designu

Datum zadání diplomové práce: **1. prosince 2010**

Termín odevzdání diplomové práce: **15. září 2011**

Ve Zlíně dne 15. února 2011

doc. MgA. Jana Janíková, ArtD.
děkanka

L.S.

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta multimediálních komunikací

MgA. Petr Stanický, MFA
ředitel ústavu

PROHLÁŠENÍ AUTORA
BAKALÁŘSKÉ/DIPLOMOVÉ PRÁCE

Beru na vědomí, že

- odevzdáním bakalářské/diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby [1)];
- beru na vědomí, že bakalářská/diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému a bude dostupná k nahlédnutí;
- na moji bakalářskou/diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3 [2)];
- podle § 60 [3)] odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- podle § 60 [3)] odst. 2 a 3 mohu užít své dílo – bakalářskou/diplomovou práci - nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- pokud bylo k vypracování bakalářské/diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tj. k nekomerčnímu využití), nelze výsledky bakalářské/diplomové práce využít ke komerčním účelům.

Ve Zlíně 24. 3. 2011 ..... JAN HŮLA .....

Jméno, příjmení, podpis

---

1) zákon č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, § 47b Zveřejňování závěrečných prací:

(1) Vysoká škola nevýdělečně zveřejňuje disertační, diplomové, bakalářské a rigorózní práce, u kterých proběhla obhajoba, včetně posudků oponentů a výsledku obhajoby prostřednictvím databáze kvalifikačních prací, kterou spravuje. Způsob zveřejnění stanoví vnitřní předpis vysoké školy.

(2) Disertační, diplomové, bakalářské a rigorózní práce odevzdané uchazečem k obhajobě musí být též nejméně pět pracovních dnů před konáním obhajoby zveřejněny k nahlížení veřejnosti v místě určeném vnitřním předpisem vysoké školy nebo není-li tak určeno, v místě pracoviště vysoké školy, kde se má konat obhajoba práce. Každý si může ze zveřejněné práce pořizovat na své náklady výpisy, opisy nebo rozmnoženiny.

(3) Platí, že odevzdáním práce autor souhlasí se zveřejněním své práce podle tohoto zákona, bez ohledu na výsledek obhajoby.

2) zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, § 35 odst. 3:

(3) Do práva autorského také nezasahuje škola nebo školské či vzdělávací zařízení, užije-li nikoli za účelem přímého nebo nepřímého hospodářského nebo obchodního prospěchu k výuce nebo k vlastní potřebě dílo vytvořené žákem nebo studentem ke splnění školních nebo studijních povinností vyplývajících z jeho právního vztahu ke škole nebo školskému či vzdělávacího zařízení (školní dílo).

3) zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, § 60 Školní dílo:

(1) Škola nebo školské či vzdělávací zařízení mají za obvyklých podmínek právo na uzavření licenční smlouvy o užití školního díla (§ 35 odst. 3). Odpírá-li autor takového díla udělit svolení bez vážného důvodu, mohou se tyto osoby domáhat nahrazení chybějícího projevu jeho vůle u soudu. Ustanovení § 35 odst. 3 zůstává nedotčeno.

(2) Není-li sjednáno jinak, může autor školního díla své dílo užít či poskytnout jinému licenci, není-li to v rozporu s oprávněnými zájmy školy nebo školského či vzdělávacího zařízení.

(3) Škola nebo školské či vzdělávací zařízení jsou oprávněny požadovat, aby jim autor školního díla z výdělku jím dosaženého v souvislosti s užitím díla či poskytnutím licence podle odstavce 2 přiměřeně přispěl na úhradu nákladů, které na vytvoření díla vynaložily, a to podle okolností až do jejich skutečné výše; přitom se přihlédne k výši výdělku dosaženého školou nebo školským či vzdělávacím zařízením z užití školního díla podle odstavce 1.

ABSTRAKT

Cílem této práce je vytvoření interaktivního programu pro prezentaci produktů v trojrozměrném prostoru. Na tomto programu bych chtěl demonstrovat praktičnost této prezentační formy v oblasti produktového designu. Teoretická část je tematicky rozdělena na problematiku týkající se vizualizace informací a problematiku spojenou s vývojem aplikací, v praktické části je provedena analýza jednotlivých technologií, které lze využít pro vývoj této aplikace a v poslední projektové části je popsán postup práce při tvorbě samotné aplikace.

Klíčová slova: vizualizace, počítačová grafika, vizuální myšlení, 3D, interaktivita, uživatelské rozhraní

ABSTRACT

The aim of thisworkis to createaninteractive program forthepresentation of products in threedimensions. On this program I wanted to demonstratethepracticality of thispresentationform in product design. The theoretical part isthematicallydividedintoissuesrelated to informationvizualization and problemsassociatedwith application development, thepractical part isananalysis of technologiesthatcouldbeused to developthis application and last part of projectdescribesthe work progress when creating the application itself.

Keywords:visualization, computer graphics, visual thinking, 3D, interactivity, user interface

V tomto místě bych rád poděkoval panu prof. akad. soch. Pavlu Škarkovi, za způsob jakým vedl a podporoval mé studium na Univerzitě Tomáše Bati. Jsem velice vděčný za to, že byl po celou dobu studia otevřený mému hledání a také za poznámky, které mi věnoval.

Prohlašuji, že odevzdaná verze bakalářské/diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

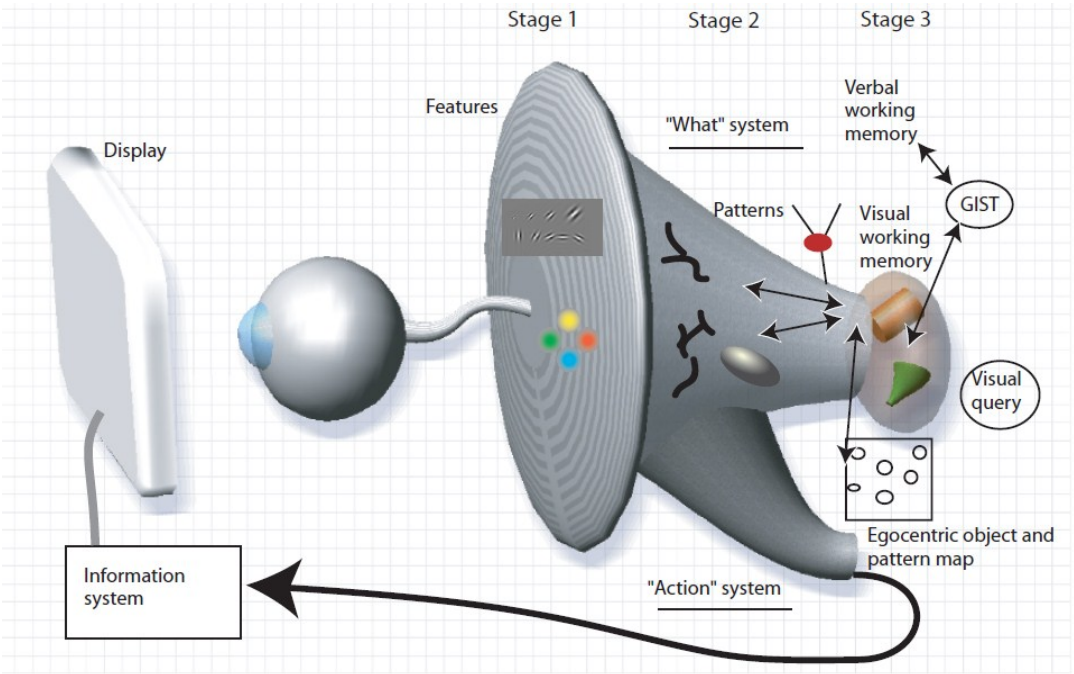
Cílem této diplomové práce je vytvoření interaktivního programu, který bude sloužit k vizualizaci trojrozměrných objektů s jeho funkcemi a vlastnostmi za účelem prezentace designu produktů. Tento program bude pouhou studií, na které bych chtěl demonstrovat efektivnost interaktivity u vizualizace objektů. Tento druh interaktivní vizualizace se obvykle označuje pojmem realtimová vizualizace a v mnoha oblastech, jako je medicína a podobné vědní obory, je realtimová vizualizace již ustáleným standardem. V medicíně je například využívána pro studium lidského těla, v oblasti chemie umožňuje realtimové vizualizace zkoumat struktury molekul a vůbec největší uplatnění získala v oblasti různých výukových simulátorů, jako jsou např. letecké simulátory. Na poli produktového designu se nicméně toto médium dodnes neuchytilo. K prezentaci designu produktů je v současné době nejvíce využíváno vyrenderovaných obrázků a animací. Důvodem, proč se v některých oborech uchytila realtimová vizualizace dříve a v některých vůbec, je zaprvé praktičnost a smysluplnost pro dané použití a za druhé naléhavost pro dané použití. Existuje mnoho způsobů, jak komunikovat informaci, a každý z těchto způsobů je nadřazený jiným pro určité účely. Tak jako je někdy efektivnější použít obraz, namísto slova, stejně tak je v některých případech efektivnější použít realtimovou vizualizaci místo předrenderované animace. Záměrem této diplomové práce je demonstrovat účinnost použití právě realtimové vizualizace pro prezentaci designu produktů. Teoretická část je věnována mnoha aspektům, které se dotýkají jak vizualizace samotné, tak i vývoje softwaru. V praktické části je vypracována analýza současných technologií, které se vývoje vizualizačního softwaru týkají. V závěru této části je uveden seznam kritérií pro vývoj samotné aplikace a odůvodnění zvoleného technologického řešení. Projektová část se skládá z popisu postupu práce samotného vývoje.

I. TEORETICKÁ ČÁST

1 PROBLEMATIKA TÝKAJÍCÍ SE VIZUALIZACE

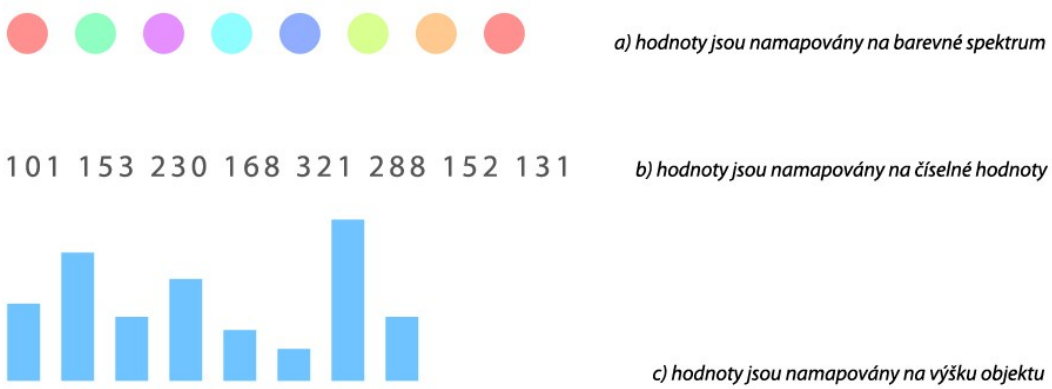
1.1 VIZUALIZACE INFORMACÍ

Jak už jsem nastínil v úvodu, každý druh komunikace je v určitých situacích výhodnější než ostatní. Náš mozek a mysl je vytvářen prostředím, ve kterém žijeme, a díky evoluci je přizpůsoben tak, aby co možná nejefektivněji zvládal úkoly, které vykonáváme. Funguje to stejně, jako u továrny vyrábějící určitý druh výrobků. Když se staví takováto továrna, tak je navržena právě taková výrobní linka, která dokáže vyprodukovat požadovaný výrobek s co možná nejmenším úsilím. Náš mozek je rovněž taková "linka" složitých procesů, která místo produktů doručuje informace z vnějšího světa. I když si toho nejsme vědomi, tak než k nám z vnějšího světa dojde určitá informace, tak v našem mozku proběhne mnoho složitých dekódovacích procesů. Lidé, kteří se zabývají designem uživatelských rozhraní a systémů vizualizací dat si jsou vědomi toho, že veškerá zařízení, která zobrazují informace, jsou, zjednodušeně řečeno, určitou nadstavbou našeho mozku a zobrazovaný výstup těchto zařízení je pouhým bodem přemostění. I když je náš mozek velmi flexibilní a dokáže se za nějaký čas přizpůsobit mnoha různým požadavkům, stejně je efektivnější, když se výstup zobrazovacích zařízení podřídí "vstupu" do našeho mozku.



Obrázek 1: zjednodušený model interakce mezi naším vizuálním systémem a informačním displejem.

To znamená, že by informace měly být podávané v takové podobě, v jaké je na to náš mozek zvyklý, aby tak cesta k zachycení informace byla co nejkratší. Když například mozek obdrží informaci, kterou nedokáže automaticky dekódovat, tak se tato informace přesune do našeho vědomí, kde se ji snažíme dekódovat vědomě. Pro zefektivnění komunikace je tedy dobré využívat veškerých ustálených konvencí a asociací a nevytvářet žádné nadbytečné procesy dekódování. Když například budu chtít vizualizovat data za účelem vyhodnocení nejvyšší či nejnižší hodnoty, nebudu je vizualizovat ani čísly ani barvou, ale schodovým grafem, z kterého se dají tyto hodnoty velmi jednoduše vyhledat. Pokud bych navíc věděl, že záměrem uživatele bude seřadit data podle výše hodnoty, tak mohu implementovat funkci, která data seřadí podle výše hodnoty.

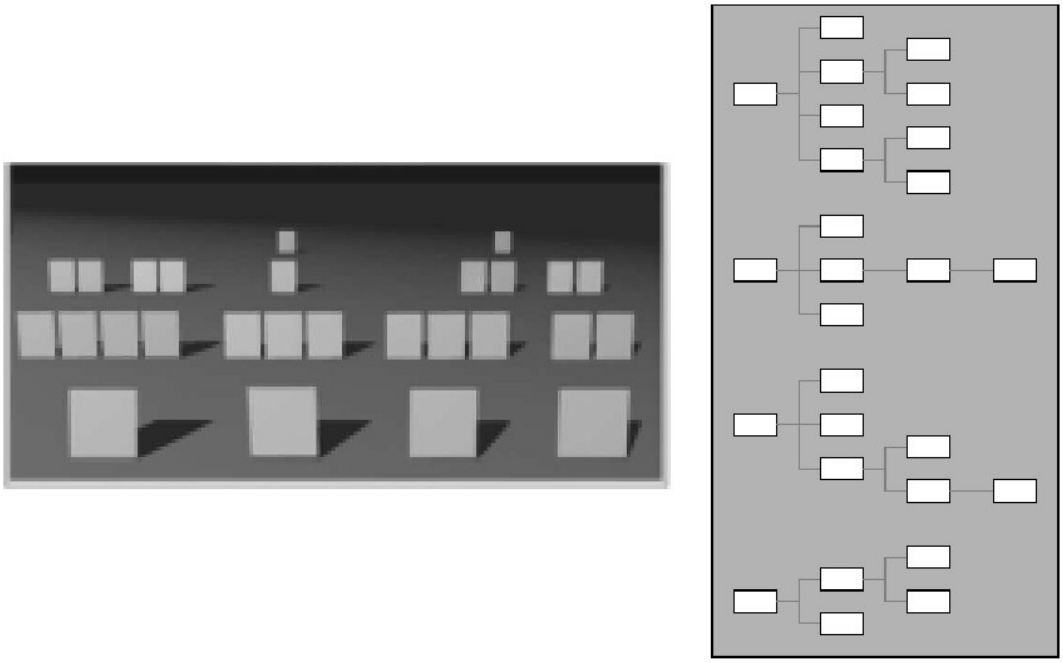


Obrázek 2: možnosti jak lze vizualizovat data za účelem porovnávání vztahů mezi jednotlivými hodnotami

Cílem všech výpočetních zařízení je co nejvíce eliminovat procesy probíhající v mozku a přenést je na tato zařízení. Pro každý záměr sdělení existuje účinný způsob a forma komunikace a cílem designera je potom tento způsob a formu rozpoznat. Většinou jde pouze o to, využít zažitých konvencí, symbolů a asociací a podat informaci v takové podobě, v jaké je na to člověk zvyklý. Čím bude kratší cesta mezi informaci z vnějšího světa a námi, tím menší je pravděpodobnost, že dojde k tzv. "informačnímu šumu".

Mnoho začínajících designerů propadá pokušení použít těch nejkomplexnějších prostředků, i v případech, kdy je to zcela kontraproduktivní. V těchto případech tyto prostředky pouze odvádějí pozornost. Když například budu chtít komunikovat abstraktní data, jako jsou různé statistiky, tak mi zcela vystačí dvourozměrné zobrazení. Třetí rozměr (hloubka) je v tomto případě nadbytečný a mohl by pouze mást. Naopak když budu chtít zorganizo-

vat prvky v prostoru tak, aby mohlo být jejich umístění zpětně vyvoláno z paměti, tak udělám lépe, když je zorganizuju do trojrozměrného pole, protože naše prostorová paměť je zvyklá operovat v trojrozměrném prostoru.(1)



Obrázek 3: způsoby prostorové organizace objektů

Jistě neexistuje nějaká obecná kategorizace toho, jak vizualizovat data podle způsobu použití. Vždy je nejprve potřeba znát konkrétní záměr komunikace a potom zvolit náležitou formu.

1.2 REALTIMOVÁ VIZUALIZACE A PŘEDRENDEROVANÁ ANIMACE JAKO FORMA PRO VIZUALIZACI DESIGNU PRODUKTŮ

Standardní formou vizualizace produktového designu i architektury je dnes statický obrázek, nebo předrenderovaná animace. S realtimovou vizualizací se v těchto oblastech můžeme setkat spíše zřídka a často jen pro interní účely firem (např. v automobilovém průmyslu). Jak už bylo zmíněno v úvodu, v mnoha oblastech se používá jako vizualizační médium právě realtimová vizualizace, pro svoje jedinečné možnosti. Pochopit, proč se v oborech, jako je například medicína, uchytila realtimová vizualizace dříve a mnohem silněji, je

jednoduché. V těchto oborech se na rozdíl od designu a architektury nejedná o estetičnost. Realtimová vizualizace v sobě obsahuje všechny aspekty předrenderované animace, či statického obrázku a tyto z ní jdou velice jednoduše vytvořit. Nejde to však naopak. Jediný aspekt, který realtimová vizualizace postrádá je vizuální dokonalost, neboli realističnost, kterou nabízí statické obrázky a předrenderované animace. Jedna z teorií estetiky říká, že míra estetičnosti je přímoúměrná míře realističnosti, v tom smyslu, jak moc se dané dílo podobá přírodě či životu. Když se zamyslíme nad tím, co je to umění, tak už samotné slovo *umění* naznačuje, že se jedná o něco umělého. Není to přírodní ani živé, ale napodobení přírodního a živého. Můžeme tedy říct, že čím skutečnější a věrnější, tím vyšší úroveň umění. Pravděpodobně by s tímto tvrzením mnoho lidí nechtělo souhlasit, neboť je to samozřejmě relativní záležitost a taky lze tuto větu pochopit mnoha způsoby. Když se ale například podíváme na historii vývoje uměleckých forem, tak uvidíme, že se vyvíjely právě tímto způsobem, tj. že jsou postupem času realističtější či živější. Každé umělecké medium v určité době dosáhne svých omezení a po technické stránce nejde posouvat dál. Úmyslně zde zmiňuji slovo *technické*. Vždy však budou existovat jiná média, která budou teprve na počátku vývoje a okolní svět je bude brát spíše jako cirkus, než umění. Tomuto pohledu zpočátku čelil například film nebo animace. Za poslední a nejrozvinutější uměleckou formu či médium považuje mnoho lidí počítačové hry. Tato forma zažívá v současnosti největší rozvoj a je ji věnována největší pozornost. Na rozdíl od filmu sice není tak vizuálně realistická, ale na druhou stranu je mnohem více "živá". Živá ve smyslu interaktivity. Inteligentně reaguje na současný okamžik. V tomto ohledu je film "mrtvý". V paralele k počítačovým hrám a filmu leží realtimová vizualizace a předrenderovaná animace. Pokud se bude vizuálně založený designer rozhodovat mezi realtimovou vizualizací a předrenderovanou animací, tak pravděpodobně oželí interaktivitu realtimové vizualizace a sáhne po hyperrealistické animaci. Nicméně v den, kdy bude realtimová animace stejně realistická jako dnešní rendery, nebude důvod zůstat u předrederované animace či obrázku.

Obrázek 4: srovnání předrenderované a realtimové vizualizace

1.2.1 DISPOZICE PRODUKTOVÉHO DESIGNU PRO POUŽITÍ REALTIMOVÉ VIZUALIZACE JAKO PREZENTAČNÍHO MÉDIA

U produktového designu k přechodu na realtimovou vizualizaci zajisté dojde dříve, než u architektury, protože scény pro vizualizaci produktů jsou výpočetně méně náročné, než

scény pro vizualizaci architektury, co se týče počtu polygonů a množství textur. U vizualizace produktů je možné využít veškerý výkon na malém prostoru a dosáhnout tak vyšší realističnosti, která je často dána komplexnosti *shaderu*[1]. Komplexností *shaderu* je myšleno počet instrukcí. Grafické karty dokážou současně zpracovávat pouze omezený počet instrukcí. Pokud budu mít grafickou kartu, která dokáže zpracovávat maximálně 1024 instrukcí a scénu s 10 různými materiály, budu moct vytvořit materiály s průměrným počtem instrukcí 100. Zjednodušeně řečeno, pokud mi budou stačit pouze 2 materiály, tak budu mít možnost vytvořit materiály 5x tak složitější.

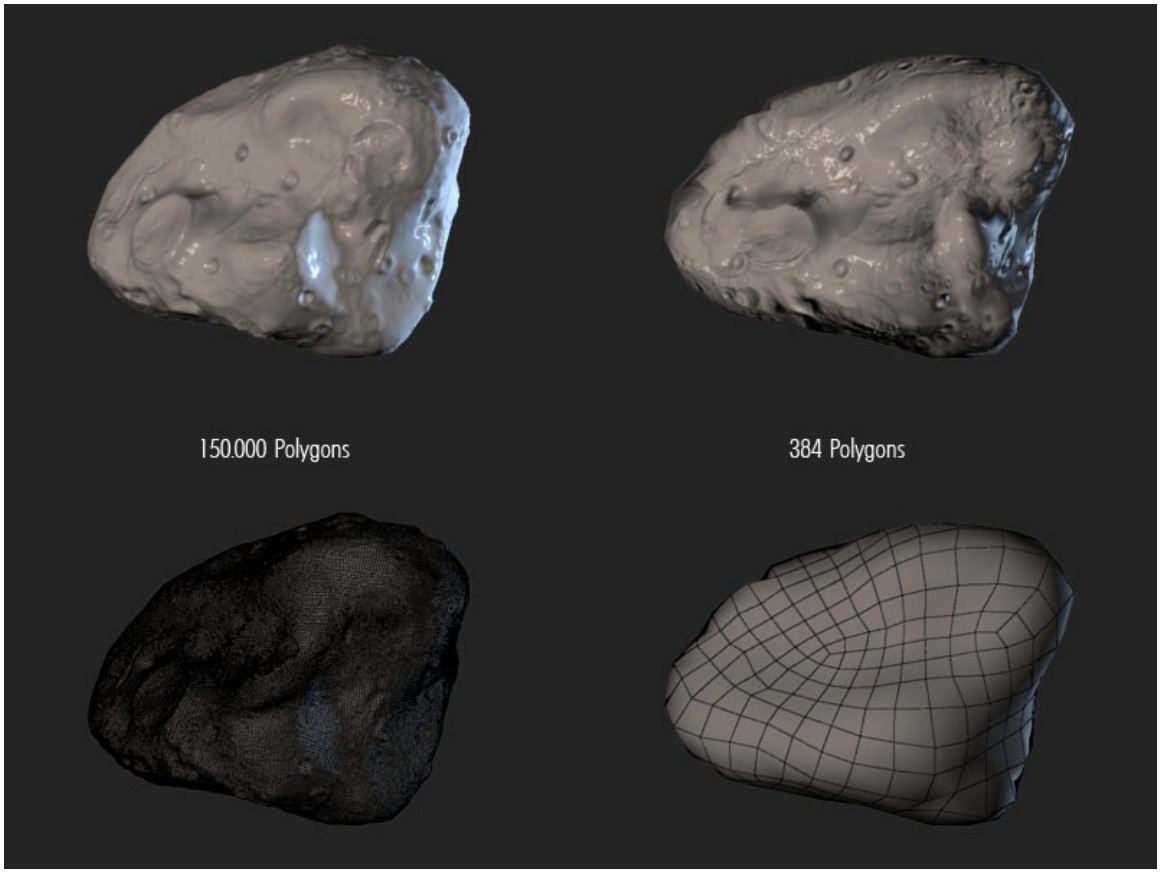
1.2.2 VÝPOČETNÍ NÁROČNOST REALTIMOVÉ VIZUALIZACE A PŘEDRENDEROVANÉ ANIMACE

Abychom dobře porozuměli problematice počítačové vizualizace, je dobré chápat diskrétní povahu počítačů. Logika počítačů je od základu postavena na diskrétní matematice, což je odvětví matematiky, které počítá pouze s jasně oddělenými hodnotami. Neexistují zde nějaké hodnoty "mezi". Mezi dvěma sousedními hodnotami není hladký průběh, ale skok. Vše musí být jasně definováno, jinak to nebude fungovat. Když si bude počítačový laik na obrazovce monitoru prohlížet kouli v trojrozměrném prostoru, tak bude mít pravděpodobně pocit, že prostorový obraz koule je uložen někde v počítači a on se na něj skrze virtuální kameru dívá z mnoha úhlů. V jistém smyslu by se to tak sice dalo chápat, ale je třeba mít na paměti, že ve skutečnosti se zhruba každou třicetinu sekundy generuje nový obraz, který vytváří iluzi prostoru a plynulého pohybu. Každou třicetinu sekundy probíhají znovu a znovu stejně náročné výpočty. Kolik snímků za sekundu počítač vygeneruje, určuje framerate a ten se u interaktivních aplikací pohubuje mezi 30-60 snímky za sekundu. U předrenderované animace, která může ve výsledku také běžet ve třiceti snímcích za sekundu, trvá často výpočet jednoho snímku až několik hodin. Například výpočet některých záběrů ve filmu *Transformers 2* trval údajně až 10 hodin na snímek. V takovém tempu by se na jed-

[1] **Shader** je specializovaný procesor (nebo část čipu), který zpracovává grafické informace grafickou kartou. Jako programovací jazyk je nejčastěji užíván CG (nVidia, univerzální), HLSL (Microsoft, pro DirectX), GLSL (pro OpenGL) a poté je přeložen pomocí překladače do assembleru přímo pro konkrétní grafickou kartu.

nom počítači vypočítávala sekunda animace necelých13 dnů. Kdyby se stejná scéna měla vypočítávat v reálném čase, tak by potřebný výkon musel být zhruba miliónkrát vyšší.

V praxi to tak však není. Vzhledem k tomu, že herní průmysl ročně vynakládá miliardy dolarů na to, aby bylo možno vytvářet co možná nejrealističtější scény v reálném čase, neplatí u realtimové grafiky a předrenderovaných animací přímá úměra u vztahu realističnost/výpočetní čas. Za prvé díky tomu, že realtimovou grafiku zpracovává grafická karta, která je na takové výpočty stavěna, na rozdíl od předrenderovaných animací, které se ještě stále vypočítávají přes procesor, a za druhé protože vývojáři věnují velkou část produkce optimalizaci, což se u předrenderované animace až tak nevyplatí.



Obrázek 5:počítačový model před a po optimalizaci

1.2.3 REALTIMOVÁ VIZUALIZACE A VIZUÁLNÍ MYŠLENÍ

Hlavní výhodou realtimové vizualizace je, kromě eliminace zdlouhavých výpočetních procesů, především interaktivita. Člověk, který si realtimovou vizualizaci prohlíží, není omezen lineárním průběhem předrenderované animace a sám může prozkoumávat prezentova-

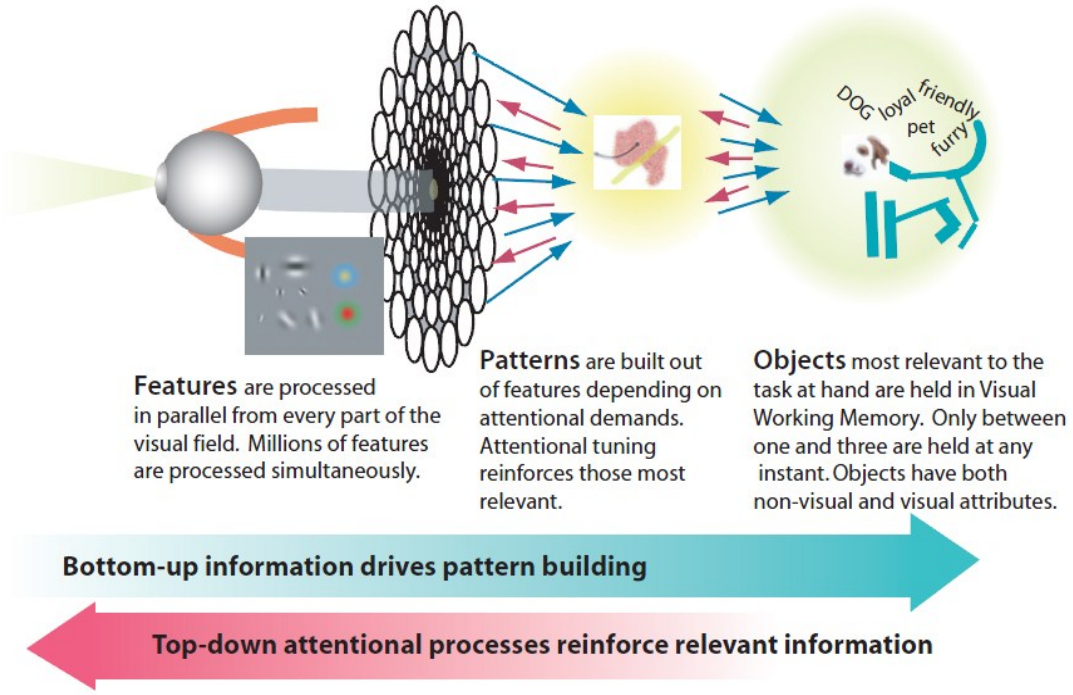
ný objekt, jak potřebuje. Může libovolně měnit úhel pohledu kamery, měnit barvy a jiné vlastnosti materiálů, skrýt některé objekty a soustředit se na konkrétní část, měnit osvětlení nebo dokonce simulovat chování objektu za určitých okolností. Tohle vše dokonale pasuje do modelu, jakým část našeho mozku pracuje. Tento model se nazývá vizuální myšlení, nebo také aktivní vidění.

1.3 VIZUÁLNÍ MYŠLENÍ

Dnešní posuzování fungování našeho mozku, se v mnohém liší od toho, jak jsme jej chápali v minulosti. Ještě nedávno si vědci mysleli, že ve svých hlavách nosíme rozmanité obrazy světa, které si skládáme z informací přicházejících skrze oči. I když dokážeme ve svých hlavách zformovat mentální obrazy, je pro nás mnohem výhodnější, když jsou tyto obrazy ve vnějším světě. My si potom z vnějšího prostředí bereme pouze ty informace, které jsou pro nás v daný okamžik důležité. I když můžeme podléhat iluzi, že vidíme detailní obraz celého světa, ve skutečnosti tomu tak není. V okamžiku, kdy potřebujeme z vnějšího světa získat určitou informaci, tak náš mozek připraví naše vidění takovým způsobem, aby snáze tuto informaci našlo (například hledáme-li červený objekt, tak budou receptory reagující na červenou barvu aktivnější). Když tuto informaci obdrží, tak vznikne nový vizuální dotaz a tento proces se opakuje. Takto se to opakuje neustále, když se díváme. Děje se to tak rychle, že si toho nejsme vědomi a máme dojem, že vidíme celistvý obraz světa se všemi detaily. Protože má mozek vysokou spotřebu energie, je evolucí optimalizován tak, že pracuje co nejúsporněji. Nosit kopii světa v našem mozku by bylo velmi neúsporné, a proto si „vypichujeme“ pouze to, co potřebujeme, v okamžik, kdy to potřebujeme. Nepotřebné informace jsou okamžitě nahrazovány novými.(2) V jeden okamžik dokážeme přijmout pouze jednu informaci, a proto vidíme pouze to, co potřebujeme. Dokonce, když se podíváme do jednoho místa, kde je více objektů, zaměřujeme naší pozornost pouze na jeden objekt a ostatních si nejsme vědomí. Kdyby nám někdo sdělil, že nám v rychlosti ukáže desku, na které budou barevné kuličky, u kterých si máme zapamatovat barvu a po tom, co by ji ukázal, se nás zeptal, jakou barvu měla ona deska, tak bychom si to pravděpodobně nedokázali vybavit. Jednoduše proto, že veškerá pozornost byla využívaná k získávání jiných informací.

Než dojde informace z vnějšího světa do našeho vědomí, tak proběhne mnoho procesů. Náš mozek pracuje modulárně, což znamená, že informace „putuje“ do určité části mozku, kde je zpracována a výsledný výstup je poslán do další části. Zjednodušený model by se

dal popsat třemi hlavními moduly. Tyto moduly jsou organizovány v určité hierarchii, od nejnižší úrovně, kde se zpracovávají jednotlivé "pixely" neboli body zachycené na sítnici, směrem do části, kde se z těchto pixelů dekódují vizuální vzory, jako například hrany, až k části, kde jsou identifikovány jednotlivé objekty. Postupně směrem nahoru jsou eliminovány přebytečné informace až do finální fáze, což je vizuálně operativní paměť, kde mohou být v jeden okamžik drženy maximálně tři objekty. Zároveň nám v mysli vyvstávají různé asociace k daným objektům, což jsou další informace, které jsou připraveny, aby s nimi bylo operováno.



Obrázek 6: Zjednodušený model vizuálního myšlení rozdělený do tří hlavních částí

Některé objekty se ve vizuálně operativní paměti zdrží pouhou desetinu sekundy, jiné déle. Limitovaná kapacita vizuálně operativní paměti z velké části omezuje naše poznání. Díky tomuto omezení se musíme, když přemýšlíme, spoléhat na externí pomůcky jakou jsou vizualizace, mapy, grafy a jiné.(3) Jelikož při naší dedukci, potřebujeme znát vzájemné

vztahy mezi věcmi. Nestačí pouze znát tyto věci samy o sobě. Kdyby tomu tak bylo, tak by všichni z nás již vymysleli spoustu vynálezů. Pokud chceme něco vydedukovat, tak potřebujeme mít určité informace postaveny vedle sebe, abychom mohli rozpoznat skrytý vzorec. Jelikož se do naší operativní paměti vejde najednou pouze pár objektů, je méně pravděpodobné, že se vedle sebe postaví zrovna ony zásadní informace, z kterých bychom dokázali něco vydedukovat.

Když například dostaneme sadu náhodných čísel a budeme je chtít zorganizovat od nejnižšího k nejvyššímu, tak bude jedna ze strategií, jak toho dosáhnout, taková, že najdeme nejdříve nejmenší hodnotu, tím způsobem, že první hodnotu porovnáme s druhou a z nich vybereme tu nižší, a tak budeme postupovat, až nám zbude pouze jedna hodnota. Stejný proces budeme opakovat i pro druhé nejnižší číslo a tak dále. Vždy si nejnižší číslo pravděpodobně zapíšeme, poněvadž bychom si jej za chvíli již nepamatovali. Zmiňované porovnávání se samozřejmě děje automaticky a nejsme si toho vědomi. I když se nám může zdát, že dokážeme vybrat z deseti jednomístných čísel to nejmenší v jeden okamžik, náš mozek musí tento úkol rozdělit na více menších úkolu. Kdybychom však do naší operativní paměti mohli postavit více čísel a naráz je seřadit, bylo by to mnohem snazší.

U porovnávání dvou objektů je často lepší, pokud je možné je vidět zároveň vedle sebe. Když máme porovnat dva objekty, které vidíme odděleně, tak to není tak snadné, jelikož naše vnímání je relativní. Uvidíme-li v časovém sledu s určitou prodlevou dva odstíny žluté barvy, které od se sebe nebudou výrazně lišit, budeme mít problém určit, který odstín byl světlejší. Když bude tímto odstínem vyplněn čtverec, který bude na bílém pozadí, tak to pro nás bude snazší. Pamatujeme si totiž velmi dobře rozdíly. Vždy pro nás však bude nejsnazší určit světlejší odstín, když budou oba čtverce postaveny vedle sebe.

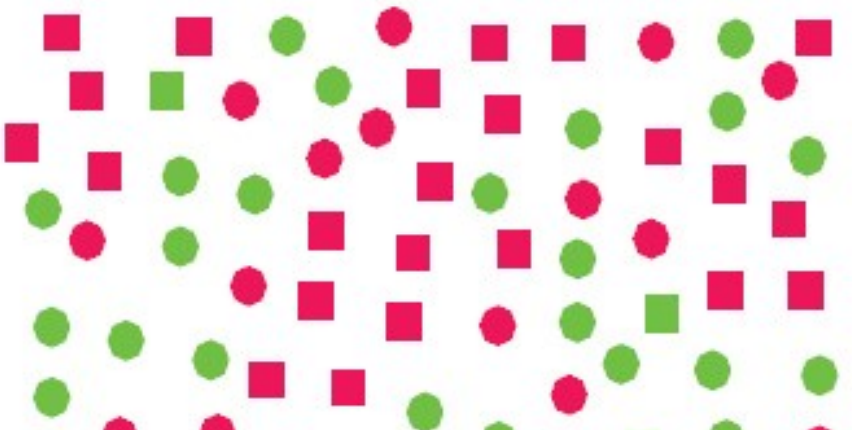
My jsme si naštěstí vyvinuli externí pomůcky, jako je počítač, který pro nás automaticky zpracovává vzájemné vztahy mezi informacemi a převádí je do intuitivních vizualizací. Člověk je tam, kde je, právě díky těmto externím pomůckám, které urychlují jeho procesy poznávání. Jak to poznamenal Donald A. Norman, vědec zabývající se kognitivními procesy: „Bez externích pomůcek by naše paměť, myšlenky a uvažování bylo omezené. Lidská inteligence je však velmi flexibilní a vyniká ve vynalézání procedur a objektů, které pomáhají překonat její vlastní limity. Skutečná síla tkví ve využívání externích pomůcek, které rozšiřují kognitivní aktivity. Čím to je, že máme lepší paměť a uvažování? Je to díky externím pomůckám. To je to, co nás dělá chytrými.“ (1)

1.3.1 IMPLIKACE PRO DESIGN

Abychom mohli navrhnout dobře fungující grafický design, musíme chápat, které kognitivní úlohy a vizuální dotazy má grafika zodpovídat. Například když budu navrhovat mapu metra, je potřeba uvědomit si jaké informace bude uživatel hledat. Jedna z informací bude například, jak dlouho trvá dostat se z bodu A do bodu B. Jelikož kombinací možných tras je nespočet, budu muset problém vyřešit tak, že mezi jednotlivé stanice napíšu čas jízdy a nechám uživatele, ať si délku své jízdy vypočítá. Pokud budu mít možnost navrhnout interaktivní aplikaci, tak ji navrhnu tak, aby se tyto časy počítaly automaticky po zadání bodů uživatelem.

Dobrý designer není ten, který dobře ovládá grafický software nebo umí dobře kreslit, ačkoliv jsou tyto dovednosti bezpochyby užitečné, nýbrž ten, který dokáže analyzovat, které informace je třeba zvizualizovat, aby byly zodpovězeny vizuální dotazy uživatele. Efektivní design musí začít s analýzou potřeb a s nimi spojenými vizuálními dotazy a potom je třeba použít barvy, tvary a prostor tak, aby bylo těmto potřebám „posloužено“.

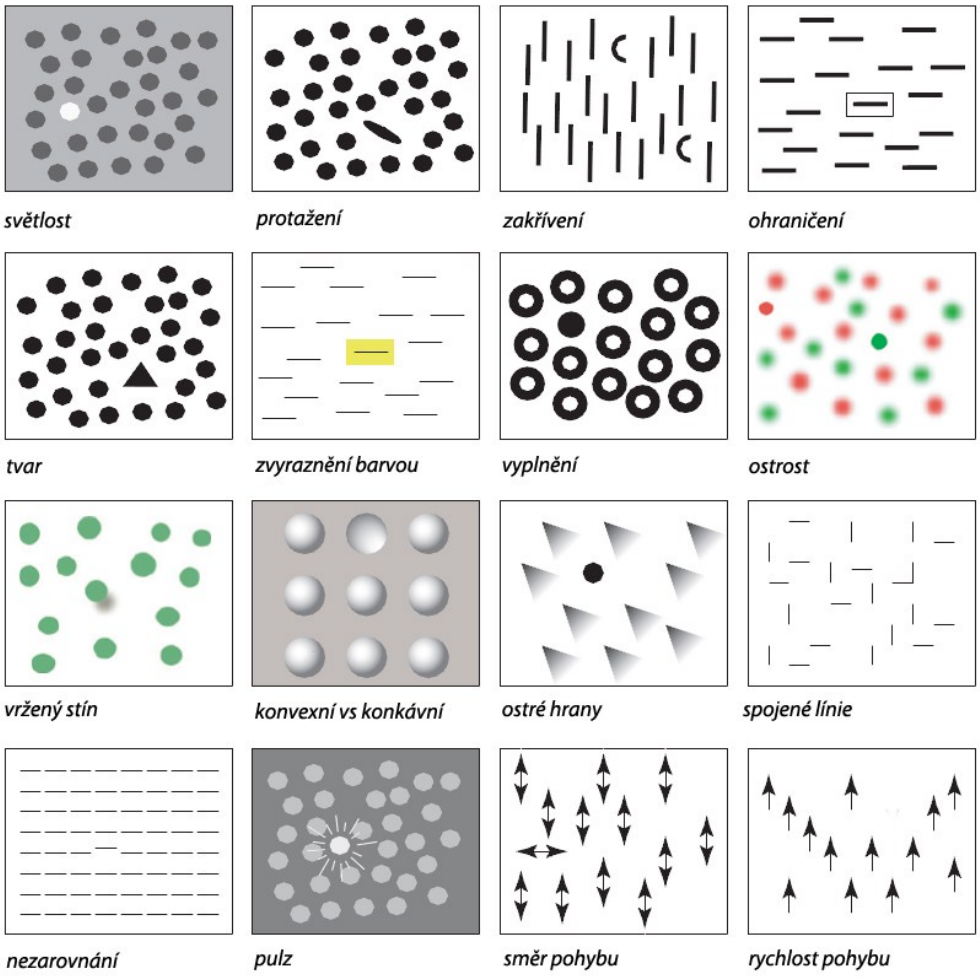
Další z funkcí mozku, kterou je třeba chápat, je schopnost adaptace receptorů při hledání určité informace. Když například hledáme rajče, budou receptory citlivé na červenou barvu vysílat intenzivnější signály a receptory snímající zelenou a modrou budou takříkajíc utišeny. V jeden okamžik se víceméně dokážeme vyladit pouze na jeden typ informace. Pokud budeme chtít, aby určitá informace byla rapidně vyhledatelná, budeme ji muset odlišit alespoň na jednom informačním kanálu, jako je barva, tvar, směr pohybu a podobně.



Obrázek 7: zelený čtverec nebude rapidně vyhledatelný, jelikož není odlišen od zbytku ani v jednom informačním kanálu (barva, tvar).

Určité prvky přitahují naší pozornost, aniž bychom je vyhledávali.

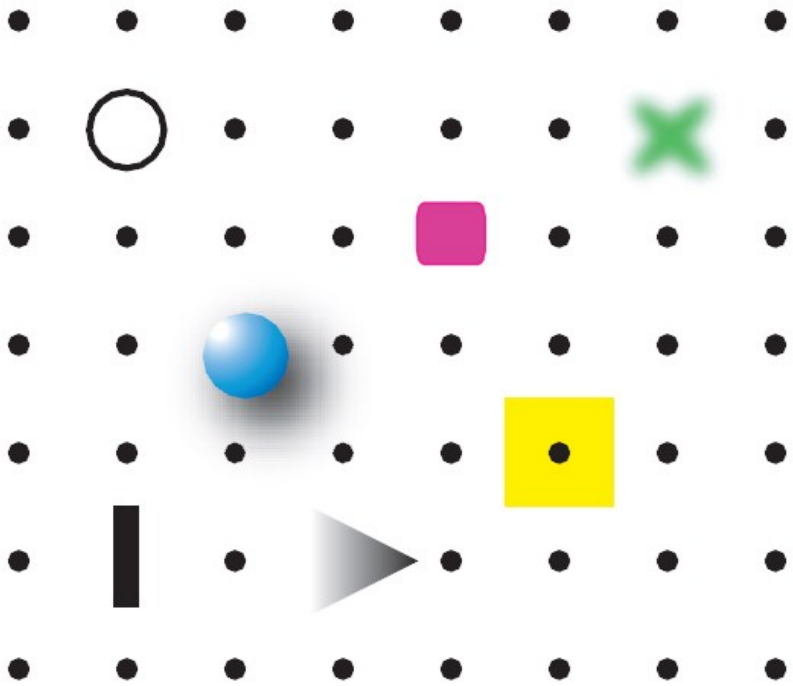
Mezi tyto patří:



Obrázek 8: nízko úrovňové prvky, které jsou rapidně rozpoznatelné

Dle toho, v které části mozku se určité prvky zpracovávají, lze usoudit, jestli budou snadno odlišitelné od ostatních či nikoliv. Pokud tyto prvky budou zpracovávat neurony v části zvané *vizuální kortex 1 (V1),* budou snadno identifikovatelné mezi ostatními prvky. *Vizuální kortex 1* je na nejnižší úrovni v řetězci mezi vnějším prostředím a naším vědomím, což znamená, že při vstupu do mozku jsou informace jako první zpracovávány zde. Prvky, které se zde zpracovávají, jsou například barva, pohyb a tvar. Výše v hierarchii, například v části zvané *vizuální kortex 2 (V2)* jsou z těchto prvků vyhodnocovány další, složitější vzory.(2)

Tyto prvky můžeme chápat jako kanály informací. Je obecně platné, že v čím více kanálech se objekt odlišuje, tím více vyniká. Pokud budeme chtít vytvořit systém obsahující 2 až 3 symboly, které budou moci být nalezeny během okamžiku, budeme je muset odlišit v rozdílných kanálech. Například velikostí, barvou a pohybem. Pokud budeme mít symbolů deset, bude to nemožné, jelikož nebudeme mít dostatečný počet kanálů, ve kterých se budou dát symboly odlišit. Pokud bude brán ohled na tyto poznatky, tak jedna věc, na kterou musí každý designer narazit, je problém při snaze udržet konzistentnost stylu a jasnost sdělení. V zájmu praktičnosti je často lepší upustit od estetických měřítek, jelikož ty jsou často pouze zažitými a přejatými vzory, které mohou být velice nepraktické pro dané řešení.

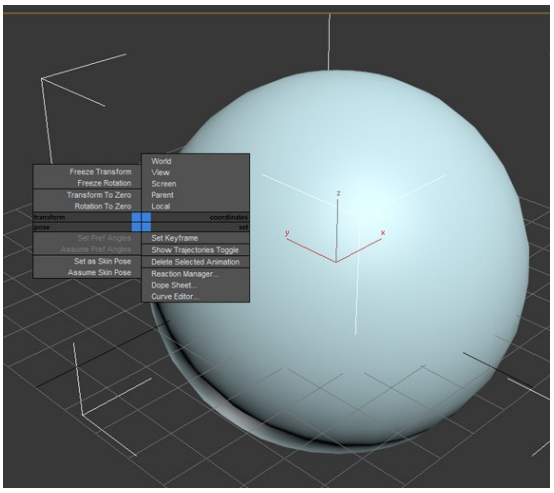


Obrázek 9: set symbolů, které jsou navrženy tak, aby byly rapidně vyhledatelné. Každý symbol je odlišen na více informačních kanálech. Například zelený kříž je odlišen barvou, tvarem i rozostřením.

1.4 GUI

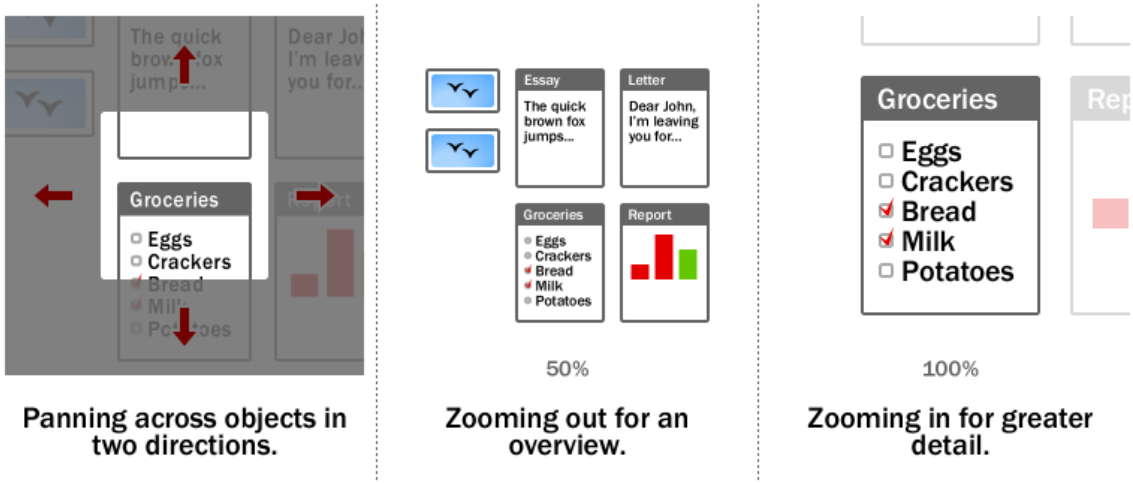
Člověk komunikuje s počítačem skrze uživatelské rozhraní. V počátcích mělo tohle uživatelské rozhraní pouze textovou podobu, později se však pro snazší orientaci a intuitivnost

vyvinulo grafické uživatelské rozhraní, které se často označuje pojmem *GUI (graphical user interface)*. S tímto uživatelským rozhraním byla současně navržena i počítačová myš, která práci s počítačem značně usnadnila. Dobře navržené uživatelské rozhraní dělá počítačový program snadno ovladatelný a úspěch programu na trhu s ním úzce souvisí. Cílem navrhování vizuální kompozice programu je usnadnit uživateli pochopení logiky a fungování procesů, které program v pozadí vykonává. Tudíž by grafické prvky měly reprezentovat úlohu, kterou vykonávají. Dobře navržené uživatelské rozhraní by nemělo odrážet architekturu systému, nýbrž požadavky uživatele. Což v konečném důsledku znamená, že když budu navrhovat ikonu, která bude sloužit k výpočtu úhlů mezi dvěma objekty, tak ji nepojmenuji $\sin^{-1}$, což může být funkce, která tento úhel vypočte, ale vytvořím symbol, který měření úhlu evokuje. Při tvorbě uživatelského rozhraní se často využívají poznatky kognitivní psychologie, jako různé asociační vztahy a zažité modely. Nejdůležitější části designu je analýza hierarchie potřeb podle frekvence použití. Cesta k vykonávání těch nejčastěji prováděných úkonů musí být co nejkratší, jako je pouhé najetí kurzoru na ikonu. Méně využívané úkony mohou být schovány v menu a cesta k nim může být delší, například až několik kliků. V závěru však může být jediný klik důvodem, proč si uživatel zvolí jinou aplikaci. U více sofistikovaných programů, které uživatelé mohou používat k mnoha odlišným účelům, se však často nabízí možnost, aby si uživatelé sami přizpůsobili uživatelské rozhraní podle svých potřeb. Nejrychlejším způsobem komunikace s programem jsou zpravidla klávesové zkratky. Pro začínající uživatele nicméně nejsou intuitivní a proto je používají spíše pokročilí uživatelé, kteří již jsou s funkcemi programu dobře obeznámeni. Další funkcionalitou, kterou často sofistikované programy nabízejí je kontextové menu. Kontextové menu je takové menu, které reaguje odlišně podle konkrétní situace. Faktory, které často ovlivňují chování kontextového menu, jsou například mód programu, přidržená klávesová zkratka, nebo pozice kurzoru na obrazovce. Jako příklad si uveďme program pro tvorbu 3D modelů. Tyto programy jsou často velmi složité, a proto vyžadují efektivní zpracování uživatelského rozhraní. Často vykonávané úkony jsou většinou v nabídce menu, která se zobrazí po kliknutí pravým tlačítkem myši na



Obrázek 10: kontextové menu

pracovní plochu. Uživatel si většinou sám může určit, které úkony se v tomto menu budou zobrazovat, nebo si může vytvořit jiné menu, které se zobrazí při přidržení jím určené klávesové zkratky. Toto menu také může zobrazovat jinou nabídku v závislosti na módu programu, nebo na pozici kurzoru. Novinkou v oblasti uživatelských rozhraní představuje tzv. *zoomovatelné uživatelské rozhraní (ZUI)*, které umožňuje oddalování pro celkový přehled a přibližování pro zobrazení detailnějších informací. Uživatelské rozhraní často mohou obsahovat grafy, mapy a jiné složité prvky, které uživateli usnadní jeho myšlenkové pochody.



Obrázek 11: zoomovatelné uživatelské rozhraní

1.4.1 STRUKTUROVANÉ HLEDÁNÍ

Když hledáme nějakou informaci, často nejprve vyhledáme místo, kde by se tato informace mohla nacházet, a potom toto místo "skenujeme“ detailněji. Nevyplatí se nám detailně skenovat celé okolní prostředí. To, kde by se daná informace mohla nalézat, si odvozujeme z předchozí zkušenosti. Když budu například hledat knihu v cizím prostředí, nejdříve vyhledám místa, kde by se mohla nacházet, jako jsou stoly, regály a poličky a ty pak detailně prozkoumám. Tohoto můžeme efektivně využívat při tvorbě uživatelského rozhraní. Pokud budeme mít mnoho objektů a chceme, aby se v nich uživatel dobře orientoval, bude vhodné si je rozdělit do větších celků, ve kterých budou náležitě zorganizovány. Uživatel si zapamatuje jejich relativní umístění v rámci skupiny. Jakmile naopak hledaný objekt nebude na svém místě, povede to k frustraci. Proto je důležité udržovat určitou konzistenci a například využívat organizaci již zažitých systémů. V některých případech, ve kterých je jiná organizace efektivnější, může být vhodné uživatele přinutit se přeorientovat.

1.4.2 Deset zásad uživatelského rozhraní

Jedná se o deset obecných zásad pro navrhování uživatelského rozhraní. Říká se jim "heuristika", protože jsou založeny spíše na odhadu a intuici, než na konkrétním používání směrnic.

Indikace stavu systému

Systém by měl vždy průběžně informovat uživatele o tom, co se děje, a to prostřednictvím odpovídající zpětné vazby a v přiměřeném čase.

Konzistence mezi systémem a skutečným světem

Systém by měl mluvit jazykem uživatelů, se slovy, větami a pojmy srozumitelnými pro uživatele, raději než jazykem systémových názvů. Měl by odrážet pravidla skutečného světa, takže se informace budou objevovat v přirozeném a logickém sledu.

Uživatelská kontrola a svoboda volby

Uživatelé často spouštějí funkce systému nějakým omylem, a proto potřebují možnost danou funkci zrušit. Toto musí být podporováno velmi zřetelně. Zároveň by systém měl nabídnout možnost vrátit se o krok zpět nebo dopředu.

Soudržnost a standardy

Uživatelé by si neměli lámat hlavu, zda různá slova, situace nebo akce indikují stejnou věc. Měly by se dodržovat konvence již zavedených platforem.

Prevence chyb

Mnohem lepší, než dobré chybové hlášení, je pečlivé provedení, které v první řadě zabraňuje potížím. Buďto by se mělo předejít situacím, které mohou být náchylné k chybám anebo by se uživatelům měla poskytnout možnost potvrzení dříve, než se určitá akce provede.

Rozpoznání versus vyvolávání z paměti

Systém by měl být navržen tak, že jednotlivé úkony budou snadno rozpoznatelné. Pokud se uživatel nachází v určitém dialogu, neměl by být nucen, aby si vybavoval informace z jiného dialogu. Návody k použití systému by měly být viditelné a snadno dohledatelné v jakýchkoliv situacích.

Flexibilita a efektivita využití

Akcelerátory - neviditelné pro nováčky - můžou často urychlit užívání programu odborného uživatele. Systém by měl být navržen tak, aby mohl sloužit oběma typům uživatelů, jak nezkušenému tak i zkušenému. Dále by měl uživatelům umožnovat přizpůsobit si systém, aby snadno vykonával často využívané akce.

Estetika a minimalistický design

Dialogy by neměly obsahovat informaci, která je irelevantní nebo zřídka potřebná. Každá další jednotka informace v dialogu soutěží s ostatními jednotkami informace a snižuje tak jejich relativní viditelnost.

Pomoc při rozpoznání, diagnostikování a zotavení se z chyb

Chybové zprávy by měly být vyjádřeny v jednoduchém jazyce (žádné kódy), přesně ukázat problém a konstruktivně navrhnout řešení.

Nápověda a dokumentace

I přestože je lepší, když je systém užíván bez jakékoliv dokumentace, občas je však nezbytné umožnit nápovědu a dokumentaci. Veškeré tyto informace by měli být jednoduše vyhledatelné, měly by se zaměřit se na jednotlivé úkony uživatelů a také by neměly být příliš velké.(5)

2 PROBLEMATIKA TÝKAJÍCÍ SE TECHNOLOGIE

2.1 OPENSOURCE VS PROPRIETÁRNÍ SOFTWARE

Když se jakýkoliv podnikatel v oblasti informačních technologii rozhodne nabízet nějakou službu či produkt, měl by dobře zvážit, jestli své podnikání postaví na opensourcovém nebo proprietárním softwaru. V případě, že by sám vyvíjel software, tak jestli jej vydá jako opensourcový nebo proprietární. O tom, který z těchto modelů je lepší, se vášnivě debatuje snad od počátku vývoje softwaru. Tento problém často přesahuje za hranice vývoje softwaru až do oblasti politických ideologií. Opensourcový a proprietární model by se dal s nadsázkou přirovnat ve stejném sledu ke komunistickému a kapitalistickému ekonomickému modelu, kdy v komunistickém modelu vynakládají lidé úsilí pro společné blaho a v kapitalistickém modelu jsou oproti tomu lidé motivováni, aby sami podnikali ve svůj prospěch, z čehož budou mít v konečné implikaci užitek i ostatní, jelikož se věří, že soutěživý trh produkuje kvalitnější produkty. Hlavní rozdíl mezi opensourcem a proprietárním softwarem je, že opensource má otevřený kód a proprietární software jej má uzavřený. To znamená, že uživatelé opensourcového softwaru mohou do kódu nahlédnout, zjistit jak funguje, pozměnit jej ke svým účelům a tento pozměněný kód šířit dál. Uživatelé proprietárního softwaru tuto možnost nemají, jelikož je jeho kód zakryptován nebo zkompilován do podoby, které rozumí počítač, nikoliv však člověk. Vývoj u obou těchto typů softwarů probíhá zcela odlišným způsobem. Proprietární software je vyvíjen a udržován úzkou skupinou lidí a naopak opensourcový software je vyvíjen kýmkoliv, kdo by se na vývoji chtěl podílet a má k tomu určité schopnosti. Často to probíhá tak, že tito dobrovolníci posílají svůj kód do centrálního repozitáře a osoba k tomu pověřená jej schvaluje a implementuje do verze určené pro veřejnost. Zjednodušeně řečeno.

Mojí snahou zde není obhajovat určitý model, ale spíše poukázat na výhody a problematiku, která s těmito modely souvisí. Oba modely se prokázaly jako úspěšné a vzniklo díky nim mnoho užitečného softwaru. Výběr modelu závisí na záměrech a situaci dané osoby či skupiny, a proto je třeba znát úskalí obou modelů.

Obrázek 12: Značky spojené s oběma typy softwarů

U opensourcu patří mezi největší výhody, podle mého mínění, svoboda a nezávislost v užívání. Pokud používám opensourcový kód, můžu s ním nakládat, až na výjimky, jak se mi zlíbí. Existuje více typů opensourcových licencí, které určují podmínky používání softwaru. U většiny typů je hlavní podmínkou, že uživatelé opensourcového kódu jej nemohou tzv. zavřít. To znamená, že pokud odněkud stáhnu opensourcový kód a upravím jej, tak v případě, že bych jej chtěl dále poskytnout veřejnosti, musí zůstat otevřený a přístupný ostatním. Samozřejmě mě nikdo nenutí, abych jej veřejnosti poskytl, nebo jej nezpoplatnil. To, že bych jej zpoplatnil, by však nemělo velký smysl, jelikož ten, kdo by si jej koupil, mohl by jej poskytnout veřejnosti a můj záměr by byl zmařen. S tématikou opensourcu je často spojován pojem *copyleft*, což je opak copyrightu. Existují i licence jako např. *BSD*, která dovoluje uživatelům kód uzavřít s podmínkou uznání autorství původního autora. Mezi nejznámější příklady opensourcového softwaru patří operační systém *LINUX*, internetový prohlížeč *MOZILLA FIREFOX*, nebo relační databáze *MYSQL*. U vývoje opensourcového softwaru se často stává, že se některé produkty štěpí do více odnoží, jak se to děje například s mnoha verzemi systému *Linux*. Například verze *FEDORA LINUX* je upravená

verze *RED HAT LINUXU*. Tvůrci *RED HAT LINUXU* později využili tuto odvozeninu a vytvořili z ní verzi *RED HAT ENTERPRISE LINUX*. Také dnes není výjimkou, že firmy, které byly zatvrzelými obhájci proprietárního modelu, jako např. *Microsoft*, začaly naplno využívat výhod opensourcové vývoje. Důvodem je právě ono zmiňované společné „blaho". Většina firem podnikajících ve stejné oblasti často čelí stejným problémům. Pokud nemá dostačující prostředky na vyřešení tohoto problému, muže se na vývoji podílet s ostatními firmami. V praxi se uchytil způsob, že se publikuje kód pro základní řešení problému, kterého se chytnou ostatní vývojáři a dále jej rozpracují. Dalším důvodem může být, že opensourcové projekty se často stávají standardem v daném průmyslu. Stejně jako existují standardizované rozměry různých strojírenských součástek, existují také standardní formáty dat nebo třeba programovací jazyky. Pro firmy je většinou výhodnější implementovat standardy, které podporuje zbytek světa, než něco, co není s ničím kompatibilní. Pokud firma vyvine určitý formát, nebo něco podobného a neposkytne jej veřejnosti, riskuje tím, že se vynoří jiné řešení, které se stane standardem a zbytek světa jej začne podporovat. Není výjimkou, že se i proprietární formáty stávají standardem, jako například formát *FBX*, který vyvinul *Autodesk* pro import a export dat. Vždy však existují tendence zbavit se závislosti na uzavřených formátech a přejít k otevřeným řešením.

Opensource je nejvíce využívaný, například, ve vědecké činnosti, kde je sdílení informací základním předpokladem vývoje. Zároveň i firmy, které se pouští do dlouhodobějšího podnikání větších rozměrů, si jsou často vědomy uvěznění proprietárního kódu. Pokud se například stane, že zkrachuje firma, jejíž uzavřený software používám, může se stát, že to fatálně naruší celé fungování mé firmy. Pokud daný software byl opensourcový, tak to přinejhorším znamená, že přijdu o budoucí podporu. Pokud je však kód užitečný ve větším měřítku, tak se jej s velkou pravděpodobností někdo ujme a bude jej vyvíjet dál, nebo pokud jsem k tomu technicky způsobilý, tak jej mohu udržovat sám. Je třeba si uvědomovat, že počítačový program je pouze určitá forma intelektuálního vlastnictví neboli „know-how". Oproti materiálním objektům má intelektuální vlastnictví či informace obecně tu výhodu, že když jej někomu předám, tak o něj nepřijdu. Mohu pouze přijít o užitek z toho, že jej vlastním, zatímco ostatní nikoliv a mám tak výhodu v soutěživém prostředí trhu. Hlavní výhoda kódu je to, že jej můžu nesčetně krát rozmnožit s nulovými náklady. Můžu tedy vyvinout užitečný program a poskytnout jej milionům lidí, aniž by mě to něco stálo, což je velká výhoda, například, oproti strojům. Richard Stallman, autor „čtyř softwarových svobod" a člověk, který se jako první zasadil o rozšíření opensourcu*(GNU),* často poukazu-

je na nevýhody uvěznění proprietárního softwaru. Ve svých přednáškách zdůrazňuje problémy, které vznikají s nenásilným vnucováním proprietárního softwaru dětem a počítačovým laikům obecně. Když se tito lidé v budoucnu rozhodnou zbavit se závislosti na uzavřeném softwaru a přejít k opensourcovým řešením, tak jim to často trvá dvakrát tak déle, než se odnaučí zažitý systém a přejdou na ten nový, než kdyby s ním začali od počátku.

Zde se dostáváme ke krátkodobé a dlouhodobé navratitelnosti obou modelů. Využití proprietárních softwarů bývá zpravidla efektivnější v krátkodobém časovém horizontu. Tento typ softwaru bývá často uživatelsky přívětivý a intuitivní s tzv. nablýskaným uživatelským rozhraním, jelikož jeho tvůrci vynakládají mnoho prostředků, aby se uživatelům zalíbil na první pohled a bylo pro ně snadné se na něj adaptovat. Tento typ softwaru mívá zpravidla kvalitní uživatelskou podporu, jelikož se k tomu při prodeji výrobci často zavazují. Z dlouhodobého hlediska však uživatelé mohou narážet na omezení, se kterým nejsou schopni nic udělat. Například se může stát, že by se jim hodila určitá funkčnost programu, která by značně zautomatizovala jejich práci. Nicméně u proprietárního softwaru nebudou mít přístup k nejnižší úrovni kódu, která jim umožní tuto funkci implementovat. Většina softwaru sice nabízí tzv. *API*, neboli aplikační rozhraní, skrze které mohou uživatelé přistupovat k funkcím na nižší úrovni programu. Toto *API* je zjednodušeně řečeno seznam funkcí, které budou deklarovány jako veřejné a uživatelé je tak mohou využívat. Nicméně se může stát, že funkce, kterou bude uživatel potřebovat, toto *API* nabízet nebude.

U opensourcu je tomu spíše naopak než u proprietárního modelu. Jelikož není výjimkou, že vývojáři nemají z vývoje opensourcového kódu žádný zisk, často se stává, že má opensourcový produkt nedostačující či žádnou dokumentaci. Toto však v žádném případě není pravidlem, jelikož některé firmy čerpají peníze právě z podpory svých opensourcových řešení. Nicméně u mnoha produktů tomu tak je a může to být z krátkodobého hlediska velmi nepříjemné. Velkou výhodou je fakt, že pokud uživateli chybí určitá funkcionalita, tak ji sám může implementovat. Jelikož je kód otevřený, může si změnit úplně cokoliv. Pokud by nastal případ, že jazyk, v kterém je kód napsán, nepodporuje určité vlastnosti, tak lze vytvořit tzv. *wrapper* a napojit si daný jazyk na jazyk jiný, který chtěné vlastnosti poskytuje.

Otázkou zůstává, proč tak mnoho lidí vynakládá úsilí na vývoj opensourcového softwaru, když jej nemohou přímo zpeněžit. Existují pro to dva zásadní důvody. Jedním je, že vývojáři zpeněží svůj software nepřímo, což může mít podobu například placené podpory, nebo díky financování z reklamy. Dalším častým důvodem je společenské uznání. Mnoho socio-

logických studií pojednává o tom, jaký význam pro lidi společenské má uznání, a co všechno pro něj dokážou udělat.(2)Proto opensourcový software nejčastěji vyvíjí lidé z určitých komunit. Zde platí, že mnoho problémů při vývoji opensourcového softwaru je vyřešeno díky tzv. programátorskému svědění, což je potřeba nenechat žádnou výzvu ladem. Mnoho lidí tráví celé noci programováním jen proto, že si potřebují dokázat, že zvládnou překonat určitý problém. Je to asi stejné, jako když někdo luští křížovku, nebo hraje počítačovou hru.

Z tohoto popisu obou modelů lze odvodit závěr, že volba modelu se odvíjí z velké části z dalekosáhlosti našich plánů. Pokud víme, že se naše nároky na software nebudou měnit, nebo pokud je používání programu pouze krátkodobé, což může být až několik let anebo bude tvořit pouze malou část našeho podnikání, může být efektivnější volba kompaktnějšího řešení, čímž je obvykle to proprietární. Naopak pokud požadujeme vysokou flexibilitu a očekáváme, že budeme na dané technologii v budoucnosti závislí, měli bychom se držet opensourcu. Pokud se podíváme na úspěšné korporace jako je *Google* či *Facebook*, tak vidíme, že tomu tak opravdu je. Sociální síť *Facebook* ve velké míře využívá relační databáze *MySQL*, kde také velkou mírou přispívá a u společnosti *Google* se tak děje stejně například s programovacím jazykem *Python*. V obou společnostech se používá převážně operační systém *Linux*.

Hnutí opensource se dnes rozšířilo daleko za hranice počítačového kódu do všech oblastí intelektuálního vlastnictví. Mezi známější počiny patří například systém dokumentace, kterou společně tvoří dobrovolníci, zvaný *WIKI*. Nejznámější ukázkou tohoto systému je online encyklopedie *WIKIPEDIA*. Dále například licence pro volné užití digitálních uměleckých počinu s názvem *CREATIVE COMMONS*. Obecně by se dalo říci, že zde patří veškeré systémy, které podporují volné šíření informací. Lze si snadno představit, že otázkou opensourcu se ve velké míře zabývají právní systémy. Je dobře známo, že kulturní rozkvět je přímo úměrný míře možností čerpat z jiných děl, či informací obecně. Pokud není možno čerpat z předchozích děl a konceptů, tak se vývoj z velké části zabrzdí. Kdyby například při evoluci nemohl být použit genetický kód již vyvinutých organizmů, tak by se vývoj zastavil. Toto se děje když některé korporace jako *IBM* kupují obrovské množství patentů, které drží nevyužité a čekají na jejich zhodnocení. Naopak pokud bude vývoj spočívat v neustálém přetváření jedné věci, tak dojde postupem času také k degradaci, jak to můžeme vidět například u některých rodů, kde bylo povoleno uzavírat sňatky pouze

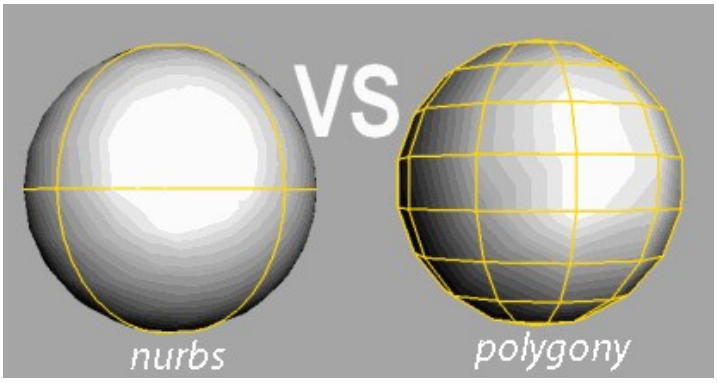
v rodinných kruzích. To, že u opensourcu nedochází k dostatečné inovaci je jedna z výtek jeho odpůrců. Úkolem zákonodárců, kteří stanovují zákony ohledně intelektuálního vlastnictví, je zajistit rovnováhu mezi těmito dvěma extrémy.(7)

2.2 POČÍTAČOVÁ GRAFIKA

Název počítačová grafika popisuje jakékoliv využití počítače k vytvoření nebo manipulaci obrázku. Díky rozvoji tohoto oboru je dnes interakce s počítači snazší a zároveň také dokážeme interpretovat mnoho typu dat. Dopad počítačové grafiky na současný svět je obrovský. Setkáváme se s ní takřka všude, ať už to je počítač, televize, magazíny, mobily, různé automaty nebo v neposlední řadě různé druhy reklamy. Počítačová grafika se zrodila jako podobor nauky o počítačích (*computer science*) v počátku 60. let 20. století. Často se rozděluje podle počtu dimenzí na 2D a 3D grafiku. V praxi se dělí na další podoblasti.

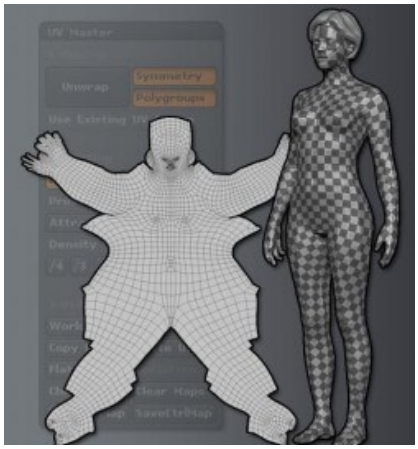
2.2.1 PODOBORY POČÍTAČOVÉ GRAFIKY

MODELING – jedná se o vytváření trojrozměrných počítačových modelů za pomocí metod, jako je polygonové modelování, modelování pomocí *NURBS* povrchů, či jiných parametrických metod. Polygonové modely se vytváří prostřednictvím polygonových plošek, které mohou být tří a vícehranné. Tato metoda je používaná zejména u veškeré interaktivní 3D grafiky. Metodou *NUBRS* se oproti tomu označuje parametrická tvorba ploch a křivek za pomocí různých interpolačních metod. Tato metoda se používá nejčastěji ve strojírenství pro svou přesnost a matematickou definovatelnost geometrických tvarů.



Obrázek 13: ukázka modelovacích metod

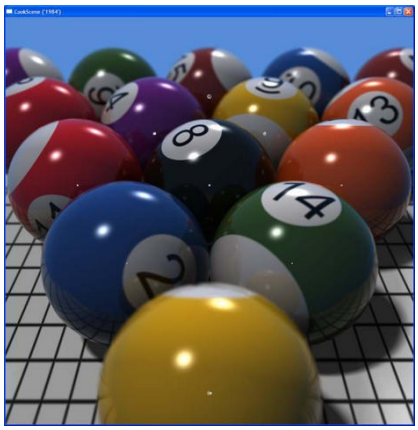
TEXTUROVÁNÍ – je metoda nanášení textur na počítačový model. Zatímco modely mohou být popsány pomocí třech souřadnic x, y, z, textury jsou popisovány v dvojrozměrném prostoru pomocí souřadnic u, v. Tyto dvourozměrné obrázky jsou pak přeneseny pomocí různých transformací na 3D model. S problematikou textu-



Obrázek 14: UV plát

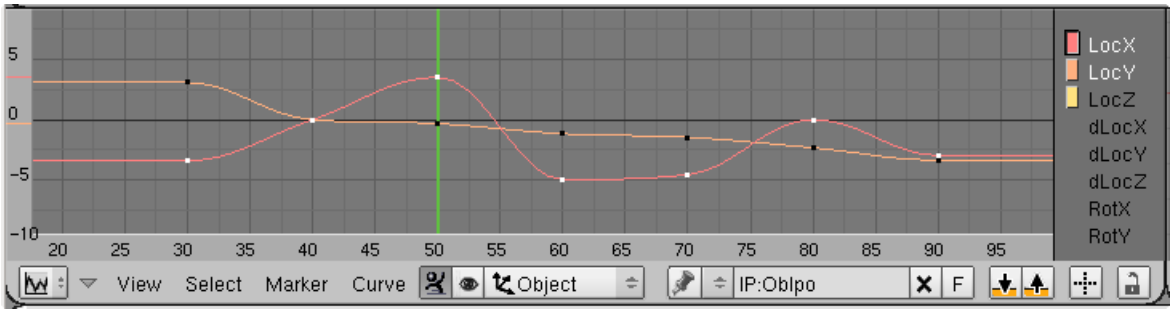
rování je spojeno vytváření tzv. *UV plátů*. Ty vzniknou nastříháním trojrozměrného modelu v určitých místech takovým způsobem, aby se dal rozvinout do plochy bez viditelných deformací.

RENDEROVÁNÍ – je pojem převzatý z klasického umění a v tomto kontextu znamená vytváření vystínovaného obrazu. Často se jedná o složité výpočty, ve kterých jsou započítávány světla, materiály a jiné vlastnosti a prvky renderované scény. Renderování se dělí na offlinové a realtimové. Offlinové renderování je opakem realtimového, což znamená, že se vykreslování obrazů neděje v reálném čase. Existuje více výpočetních modelů, jako například metoda „sledování paprsků“ (*raytracing*), kdy jsou sledovány světelné paprsky odrážející se ve scéně. Tato metoda se díky své náročnosti, až na pár výjimek, týká pouze offlinového renderování.



Obrázek 15: renderování pomocí metody sledování paprsků (raytracing)

ANIMACE – je technika vytváření iluze pohybu prostřednictvím sekvence obrázků. V 3D grafice se tento proces skládá z polohování 3D modelů do klíčových poloh podél časové osy. Tyto polohy jsou mezi sebou interpolovány, díky čemu vzniká plynulý pohyb.



Obrázek 16: interpolační křivky klíčových bodů animace (f-curves)

KOMPOZITOVÁNÍ – jedná se o proces skládání obrázků z mnoha zdrojů do jednoho obrazu, často takovým způsobem, že výsledný obraz působí zcela homogenně. Tato metoda spadá mezi postprodukční procesy a proto se netýká realtimové grafiky. Často jde o skládání dvojrozměrných a trojrozměrných prvků.

Obrázek 17: kompozitování, před a poté

ZPRACOVÁNÍ OBRAZU – týká se jakékoliv manipulace 2D obrazu.

2.2.2 HLAVNÍ OBLASTI VYUŽITÍ

Skoro každé odvětví do jisté míry využívá počítačovou grafiku. Mezi hlavní oblasti patří:

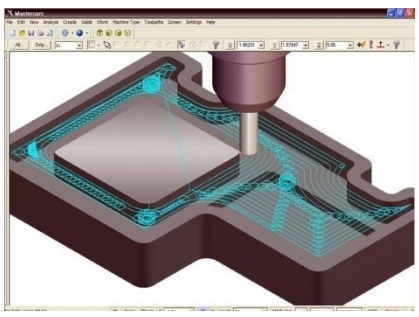
POČÍTAČOVÉ HRY – je dobře známo, že průmysl počítačových her se největším dílem zasadil o vývoj počítačové grafiky. V tomto průmyslu jsou ročně vynakládány miliardy dolarů za účelem posunutí hranice směrem k větší realističnosti.

Obrázek 18: počítačová hra Supermario Bros.

VIZUÁLNÍ EFEKTY A ANIMACE VE FILMU A REKLAMĚ – dalším velkým dílem se o vývoj počítačové grafiky zasadil filmový průmysl, protože se dnes vyplatí mnoho věcí udělat v počítači, než natáčet na živo.

Obrázek 19: záběr z filmu Avatár

CAD/CAM – tyto zkratky označují „navrhování pomocí počítačů“ *(computer aided design)* a „výroba za pomocí počítačů“ *(computer aided manufacture)*. Tyto oblasti využívají počítač k navrhování produktů a později



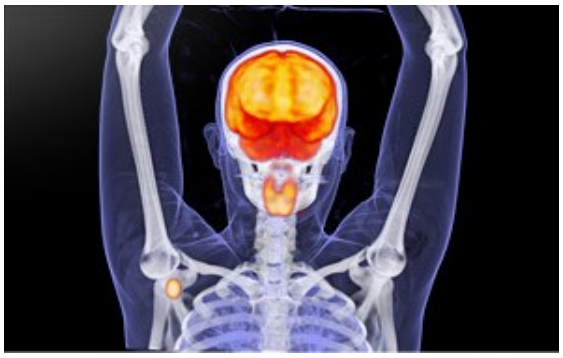
Obrázek 20: příprava formy výrobku pomočí CAM technologie

k výrobě těchto virtuálních produktů za pomocí instrukcí počítače.

SIMULACE – jde o využívání sofistikované 3D grafiky k simulování situací reálného světa jako je například řízení letadla. Simulace je nejčastěji využívaná jako výukový prostředek.

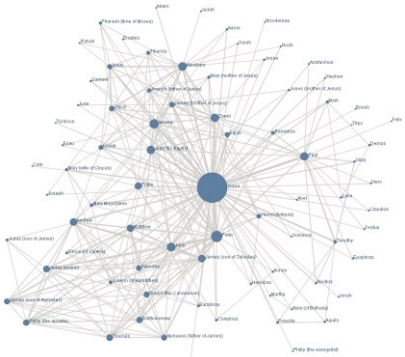
Obrázek 21: letecká simulace

ZOBRAZOVÁNÍ DAT V OBLASTI MEDICÍNY – vytváření obrazů z dat získaných při skenování pacienta, jako je například *MRI* (obrazy vytvořené pomocí magnetické rezonance). Vygenerovaný obraz pomáhá lékařům pochopit význam naskenovaných dat.



Obrázek 22: vizualizace lidského těla

INFORMAČNÍ VIZUALIZACE - zobrazování dat, které nemusí mít přirozeně vizuální podobu, jako například růst akcií na burze.



Obrázek 23: vizualizace vztahů na facebooku

2.2.3 HISTORIE VÝVOJE POČÍTAČOVÉ GRAFIKY

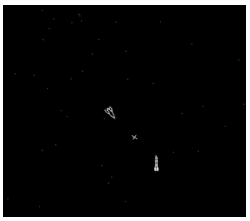
Pojem „počítačová grafika“ poprvé veřejně použil v roce 1960 grafický designer firmy Boeing William Fetter. Tento obor vznikl z nutností interakce mezi člověkem a počítačem, jelikož pro mnohé lidi byla komunikace prostřednictvím textových příkazů komplikovaná.

Mezi první počítače s uživatelským rozhraním patřil model *TX-2* vyvinutý v laboratořích Univerzity *MIT* roku 1958. S tímto počítačem mohli uživatelé komunikovat prostřednictvím péra. Revoluční program *SKETCHPAD* (1963), jehož autorem byl student *MIT* Ivan

Sutherland, uživatelům dovolil kreslit různé tvary po obrazovce počítače, které mohly být uloženy a později znovu otevřeny.

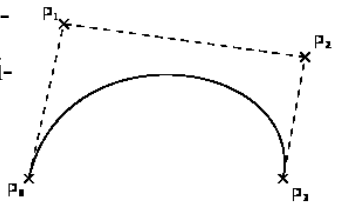
VÝVOJ V 60. LETECH 20. STOLETÍ

- **1961** V roce 1961 vytvořil SteveRussel, další student MIT, první počítačovou hru s názvem *SPACEWAR*.



Obrázek 24: hra SPACEWAR

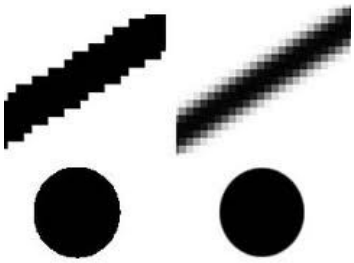
- **1962** Zaměstnanec firmy Renault Pierrem Baier publikoval své teoretické práce v oblasti parametrických křivek a povrchů.



Obrázek 25: beziérová křivka

- **1963** V laboratořích firmy *XEROXPARTC* byla vyvinutá počítačová myš.

- **1965** Objevení rychlé Fourierovy transformace umožnilo implementaci *antialiasingu*, což je metoda vyhlazování schodovitých hran.

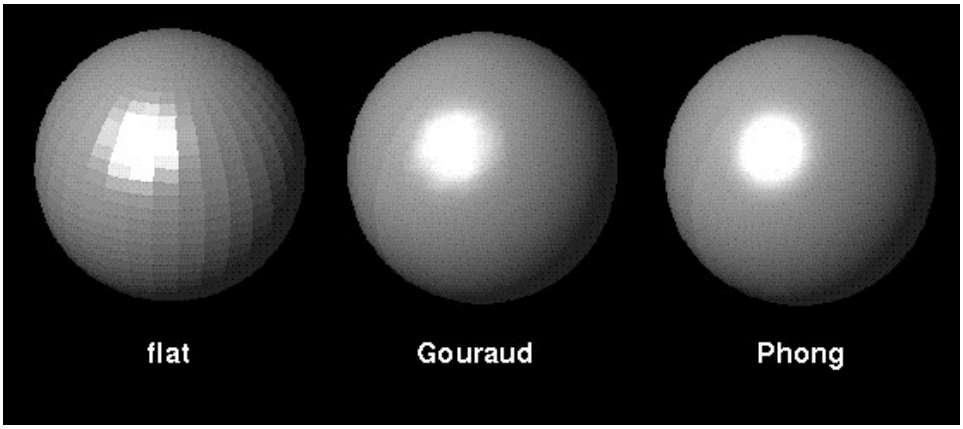


Obrázek 26: antialiasing

- **1969** V tomto roce vzniklo sdružení *SIGGRAPH*, které pořádá konference a vydává publikace v oblasti počítačové grafiky.

VÝVOJ V 70. LETECH

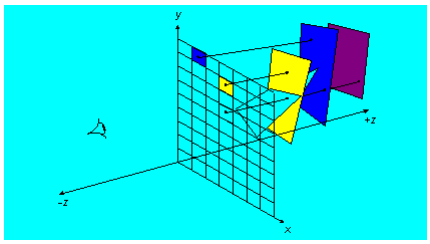
- **1971** Na univerzitě v Utahu byly objeveny nové renderovací modely (*PHONG, GOURAUD)*. Tyto modely byly schopné vypočítat reflexi a odlesky materiálů.



Obrázek 27: stínovací modely

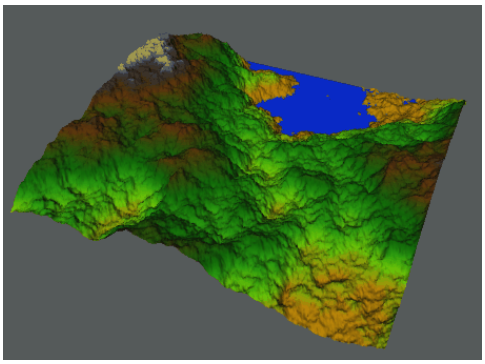
- Představení animace založené na klíčích (*keyframed animation*).

- **1974** Ed Catmull (zakladatel animačního studia *PIXAR*) představil renderovací metodu „*parametric patch rendering*“ a algoritmus zvaný *ZBUFFER*, který rozpozná pořadí objektů v závislosti na hloubce prostoru. Dalším jeho objevem bylo také mapování textur.



Obrázek 28: zbuffer

- **1975** V roce 1975 popsal matematik Mandelbrot fraktální geometrii. V počítačové grafice jsou fraktály využívány k realistické simulaci přírodních jevů, jako jsou hory, pobřeží a jiné složité struktury.



Obrázek 29: fraktální geometrie

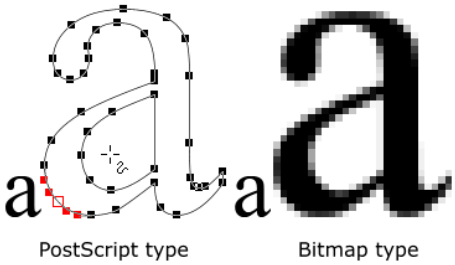
- **1976**James Blinn objevil metodu *enviroment mappingu* (metoda vypočítávání osvětlení objektů prostřednictvím falešného obrazu prostředí) a *bumpmappingu* (metoda vytváření iluze detailu na jinak jednoduché geometrii).

Obrázek 30: vytváření iluze prostoru pomocí bumpmappingu

- **1977**SteveWozniak vyvinul první osobní počítač s barevným monitorem *APPLE 2*.
- **1979** Turner Whitted vyvinul renderovací metodu „sledování paprsků“ (*raytra-cing*), která se stala standardem pro fotorealismus.

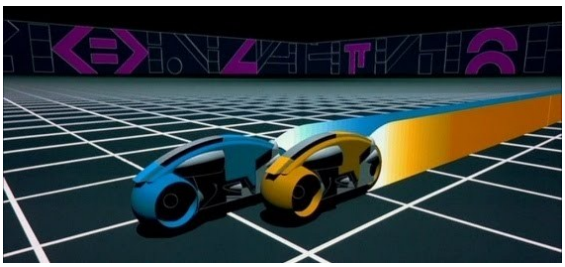
VÝVOJ V 80. LETECH

- **1982** Zakladatel firmy *ADOBE SYSTEMS* John Warnock vyvinul *PostScript*díky kterého je možno tisknout text. *ADOBE SYSTEMS* později vydává také program *PHOTOSHOP*.



Obrázek 31: Postscript

- **1982**Disney vydává první film, který je z velké části založen na počítačové grafice, s názvem *TRON*.

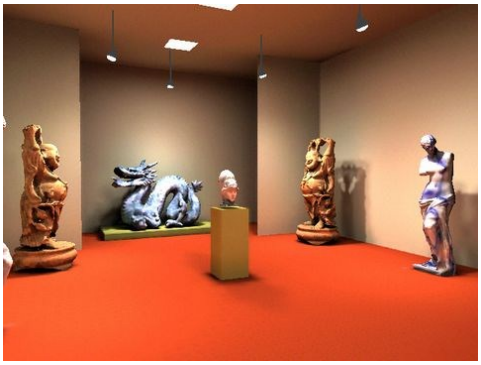


Obrázek 32: záběr z filmu Tron

- **1982** John Walker a Dan Drake program *AutoCAD*.

- **1984**vzniká první počítačový program pro tvorbu 3D grafiky *POLHEMUS*.

- **1985** je představen renderovací model *Radiosita*, který vytváří vysoce realistické obrazy 3d modelů tím, že do finálního osvětlení započítává rozptyl energie světelných paprsků.



Obrázek 33: vyrenderovaní obrázek pomocí metody Radiosita

- **1985**Tentýž rok vydává animační studio *PIXAR* svůj první film „*LUXO JR.*“

Obrázek 34: záběr z animovaného filmu Luxo Jr.

- **1989** společnost *IBM* vyvíjí předchůdce grafických procesorů, tzv. *VGA(video graphics array)*

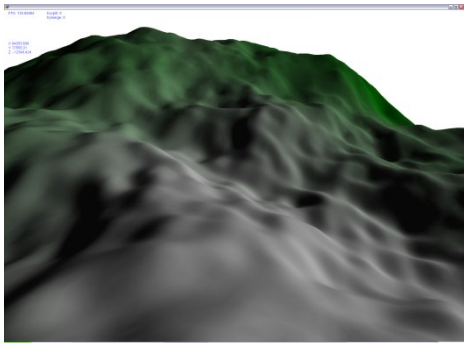
VÝVOJ V 90. LETECH

- **1990** Vývojáři studia *PIXAR* Pat Ranrahan a Jim Lawson vyvinuli renderovací software *RENDERMAN*, který se dodnes používá při produkci filmů nejvyšší kvality.

Obrázek 35: záběr z filmu Piráti z Karibiku vyrenderován pomocí softwaru Renderman

- **1991** studio *PIXAR* společně s *Disney* pracuje na animovaném filmu "Kráska a zvíře", kde je v mnoha záběrech *Renderman* využit.

- **1992** *Silicon Graphics* vydává specifikaci *OpenGL*, což je aplikační rozhraní, které umožňuje přístup k funkcím grafického procesoru.

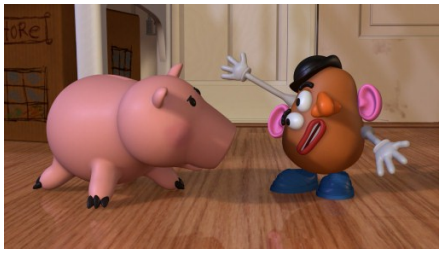


Obrázek 36: 3D model zobrazený přes OpenGL

- **1993** Steven Spielberg natáčí *"Jurský park"*, který se stal jedním z prvních úspěšných filmů, obsahujících ve velké míře počítačové efekty.

Obrázek 37: záběr z filmu Jurský park

- **1995** studio *PIXAR* natáčí první celovečerní počítačem animovaný film *"Příběh hraček"*.



Obrázek 38: záběr z animovaného filmu Příběh hraček

- **1995** *Microsoft* vydává alternativu *OpenGL, DirectX*

- **1995** rok velkého přelomu v odvětví 3D grafiky v herním průmyslu. John Carmack *(ID SOFT)* vydává hru *QUAKE*, která odstartovala velký zájem o 3D grafiku ve hrách. V této hře se používají skutečné modely místo 2D spritů (dvojrozměrné obdélníky s předrenderovanou grafikou).

Obrázek 39: hra Quake

- **1996** Společnosti zabývající se hardwarem začínají vydávat grafické karty s 3D akcelerátory.

Vývoj počítačové grafiky se později natolik urychlil, že se změny dějí takřka neustále. Díky vysokým nárokům počítačových her a výnosnosti herního průmyslu jsou investovány obrovské prostředky jak v oblasti algoritmů, tak v oblasti hardwaru.

2.2.3.1 TECHNOLOGICKÝ POKROK

Podle Mooreova zákona se počet tranzistorů v integrovaném obvodu zdvojnásobí jednou za rok při zachování stejné ceny. Gordon Moore byl jeden ze zakladatelů společnosti *INTEL* a svou předpověď vyslovil v roce 1965. V této době Moore předpokládal, že tato prognóza bude platit příštích 10 let, nicméně se ukázalo, že tento exponencionální růst bude platit ještě mnohem déle. Postupem času se rychlost růstu zpomalila a dnes se počet tranzistorů zdvojnásobuje zhruba jednou za 18 měsíců. Tato prognóza je odvozena vývoje technologie umožňující vyrábět čím dál tím menší součástky. Očekává se, že Mooreův zákon přestane platit v letech 2015-2020, kdy narazí na limity úrovně atomů. Mnozí výrobci hardwaru však předpokládají, že se tento růst nezastaví, jelikož bude možno vyrábět větší integrované obvody a bude je možno dále rozšířit o třetí rozměr, což znamená klást jednotlivé vrstvy tranzistorů na sebe. Také se očekává, že dnešní elektronické počítače budou nakonec nahrazeny kvantovými či optickými počítači. Dalším faktorem, který ovlivňuje rychlost a inteligenci počítačů je efektivně napsaný a optimalizovaný software. Ten závisí spíš na inteligenci člověka, než na fyzických možnostech. Dnes se velká část vědeckých činností v oblasti informačních technologií věnuje výzkumu umělé inteligence.(8) Ta je dnes v takové fázi, že dokáže dělat jakoukoliv racionálně definovatelnou činnost lépe než člověk. Očekává se, že v budoucnosti bude velkou část kódu psát právě umělá inteligence. Pomocí umělé inteligence fungují například také internetové vyhle-

dávače. Tyto vyhledávače jsou také jedním z hlavních faktorů, které mají vliv na technologický vývoj, jelikož díky nim dokážeme mnohem rapidněji nalézt řešení problémů. Obecně lze říci, že inteligentní technologie dělá inteligentnější lidi, kteří zpětně vytváří inteligentnější technologii, a takto se to dále opakuje v cyklu. Samozřejmě je zde myšlena pouze inteligence v určité oblasti. Významné osobnosti v oblasti výzkumu umělé inteligence, jako je např. Ray Kurzweil hovoří o příchodu tzv. technologické singularity, která je důsledkem exponenciálního technologického růstu. Tento pojem označuje dobu, kdy se technologický pokrok děje téměř instantně.(9)

2.2.4 TECHNOLOGIE PRO REALTIMOVOU GRAFIKU

Většinu interaktivní grafiky, kterou můžeme vidět na obrazovce monitoru, zpracovává grafický procesor *(GPU)*. Tento procesor je na rozdíl od klasického procesoru *(CPU)* velice specializovaný na úkony týkající se grafiky. *CPU* je navržen tak, aby mohly sloužit obecným účelům (např. editace textu atd.). Spouštějí se na něm aplikace, které jsou často napsány v jazyce pro obecné účely, jako je *C++* nebo *Java*. *GPU* jsou oproti tomu navrženy tak, aby dokázaly zpracovávat desítky miliónů *vertexů*[2]a rasterizovat stovky miliónů fragmentů[3] za sekundu. Toho je dosaženo díky vysoké paralelizaci procesů.

V počátcích vývoje bylo *GPU* tak úzce specializované, že programátoři byli nuceni psát kód v nízko úrovňovém jazyce (tzv.*Assembly*), obsahující fixní set příkazů, které zpracovával přímo hardware. Toto se nazývá *fixní pipeline*. Postupem času se *GPU* začaly vyrábět způsobem, že jednotky zpracovávající *vertexy* a fragmenty začaly být programovatelné. Toto se označuje jako *programovatelná pipeline* a dává programátorům mnohem větší svobodu, jelikož nejsou omezeni fixním setem příkazů a

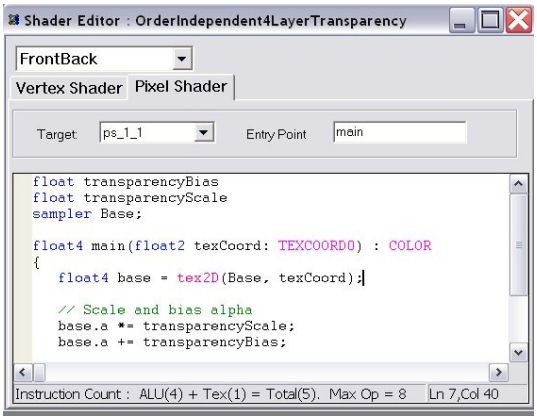
```
endpt:  mov     ax,3
        int     33h
        and     bx,01h
        mov     buttons,bx
        mov     endy,dx
        mov     ax,cx
        mov     bx,2
        mov     dx,0
        div     bx
        mov     endx,ax
        cmp     buttons,1
        jne     endptR  ;if nc
    mov endOrSwap,1
        jmp delta

endptR: mov     ax,3
        int     33h
```

Obrázek 40: ukázka assembly jazyka

[2] *Vertex* je struktura dat, které popisují bod v 2D nebo v 3D prostoru. Zobrazené objekty jsou často postaveny z plochých povrchů (typicky trojúhelníků) a vertexů (definujících rohy těchto ploch)

[3] Fragment lze považovat jako strukturu dat, která je potřebné k vystínování pixelů. V této struktuře jsou dále uloženy informace o tom, jestli fragment přetrvá nebo se z něj stane pixel.

mohou si vytvářet vlastní algoritmy. Programátoři mohli psát kód ve vysokoúrovňovém jazyce jako je *GLSL* nebo *HLSL*, což eliminovalo problémy s často až nekontrolovatelnou komplexností nízko úrovňového kódu.



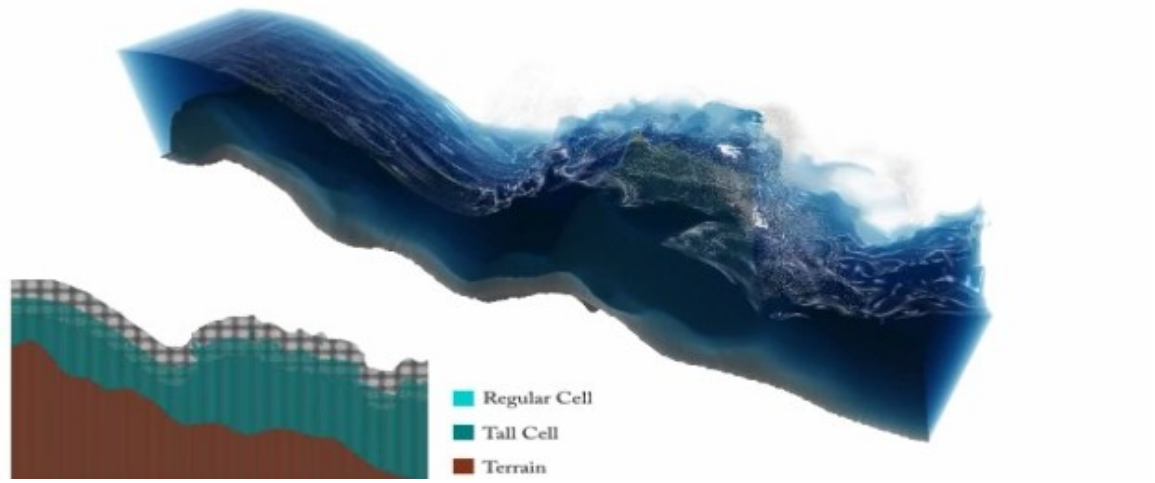
Obrázek 41: ukázka syntaxe jazyka HLSL

Jelikož existovalo více aplikačních rozhraní jako *OpenGL* a *DirectX*, které se staly velkými konkurenty, nastal problém s kompatibilitou kódu. Pro *OpenGL* se používal stínovací jazyk *GLSL*, zatímco *DirectX* se používal *HLSL*. Pro mnoho programátorů představovaly tyto dva světy velké dilema. Protože hlavní hybnou silou vývoje realtimové grafiky byl herní průmysl a *Microsoft Windows* nejrozšířenější platformou, vývojáři často zvolili *DirectX*, poněvadž ten byl v prostředí Windows poněkud napřed. Na druhou stranu však podporoval pouze Windows a tak aplikace vyvinuté pro *DirectX* nemohou být spuštěny v jiných prostředích jako *LINUX* a *OSX*. *OpenGL* bylo oproti tomu multiplatformní. Nicméně vývojáři často neodolali efektům, které umělo *DirectX*. I když je syntaxe obou jazyků *(HLSL,GLSL)* podobná, stále je porcování aplikací do jiných prostředí obtížné či v nejlepším případě neideální. Tento problém řeší stínovací jazyk *CG*, který je zacílen jak na *DirectX*, tak na *OpenGL*.

U programování obecně platí, že když se komplexnost kódu postupně zvyšuje, tak se začínají vyvíjet různé *Framework*[4],které automatizují řešení určitých opakujících se problémů. Většina vývojářů využívá těchto usnadnění, avšak někteří tento přístup odmítají a raději si všechno naprogramují sami. Dokonce existují i takoví, kteří si přejí programovat na té nejnižší úrovní, a využívat stoprocentní výkonnosti *GPU* (v případě grafiky), jelikož při používání vysokoúrovňového jazyka dochází při převodu do *assembly* jazyka (vysokoúrovňový jazyk je před zpracováním vždy potřeba převést do instrukcí, kterým rozumí hardware)

[4]Framework je softwarová struktura, která slouží jako podpora při programování a vývoji a organizaci jiných softwarových projektů. Může obsahovat podpůrné programy, knihovnu *API*, návrhové vzory nebo doporučené postupy při vývoji.

k určité ztrátě výkonu. Pro mnohé je však programování na nejnižší úrovni nereálné. Je třeba říci, že v dnešní době se využitelnost grafických procesorů posunula daleko za hranice vykreslování grafiky. Mnoho lidí si uvědomilo, že pokud by své výpočetní problémy převedli do na sobě nezávislých výpočtů, tak by mohli využít paralelnosti procesů grafického procesoru. Architektura grafického procesoru je od základu postavena na paralelních procesech, které umožňují rozdělit vykreslování obrazu do mnoha malých výpočetních problémů, které se počítají zároveň. Kdyby toto nebylo možné, tak by musela určitá část kódu počkat, než se vypočte ta před ní, což by znemožnilo vykreslování obrazu v reálném čase. Tímto způsobem funguje klasický procesor *(CPU)*. Sice se dnes procesory vyrábějí více jádrové (4-6 pro masový trh) a umožňují tzv. *multithreading*, což je právě ono využití paralelních procesů, nicméně ve srovnání s grafickým procesorem se o skutečné paralelní architektuře mluvit nedá. Grafické procesory mohou mít až 1000 jader, což může programátorům přinášet obrovské výhody, pokud výpočty dobře rozdělí. Těmto výpočtům přes grafickou kartu se obecně říká *GPGPU (General-purpose computing on graphics processing units)* a jsou častým nástrojem ve vědeckých kruzích. Jelikož byly grafické procesory vyvíjeny speciálně pro vykreslování grafiky, tak se problémy řešené přes *GPGPU* musely převádět na problémy 3D grafiky. To bylo samozřejmě velice neefektivní a tak společnosti



Obrázek 42: simulace vody vypočítaná přes grafický procesor pomocí fyzikálního enginuPhysX

produkující grafické karty začaly vyrábět nové architektury grafických karet, jako například architektura *CUDA*, kterou vyvinula společnost *NVIDIA*, a která umožňuje vypočítávat obecné problémy. Přes *GPU* se dnes vypočítávají mnohé fyzikální a jiné simulace, jež biologové využívají k dekódování genetického kódu a dokonce dnes vznikají i offline renderery, které vypočítávají obrázek právě přes *GPU*.

Výpočty přes *GPU* však mají stále jedno velké omezení a tím je paměť, kterou *GPU* disponuje. Zatímco *CPU* pracuje s operační pamětí (*RAM*), která dnes může mít kapacitu až 32GB (masový trh), *GPU* mívají tuto kapacitu nanejvýš 3GB. Proto se například u offline renderingu používá *GPU* jen pro menší scény, které mají nízký počet polygonů a textur. Vývoj *GPU* jde nicméně mnohem rychleji kupředu, než je tomu u *CPU*. Jednak proto, že *GPU* podléhá Mooreovu zákonu, stejně jako *CPU*. To znamená, že počet tranzistorů na integrovaném obvodu se zdvojnásobí jednou za 18 měsíců, při zachování stejné ceny, a také z toho důvodu, že doposud nebylo plně využito paralelismu, který *GPU* nabízí.

2.2.5 MATEMATIKA PRO 3D POČÍTAČOVOU GRAFIKU

Velká část kódu grafických aplikací je většinou matematika. Aby byl kód napsán efektivně, musí být postaven na čistých matematických konceptech. Matematika potřebná v počítačové grafice v sobě obsahuje mnoho konceptů a metod z různých matematických oblastí jako je geometrie, matematická analýza, numerická analýza, lineární algebra, diskrétní matematika, struktury dat a algoritmy. Pro mnoho matematických vzorců existují tzv. knihovny (soubor funkcí potřebných pro daná řešení), které vývojářům usnadňují práci. Vždy je však dobré chápat, co se děje tzv. "pod kapotou", což dává vývojářům větší kontrolu i možnost případné optimalizace kódu. Nejdůležitější koncepty, které by měl znát každý vývojář, jsou z oblasti trigonometrie a lineární algebry. Základní jednotkou v počítačové grafice je 2D nebo 3D vektor a veškeré transformace se provádějí za pomocí matic. Trigonometrické funkce na druhou stranu pomáhají určovat geometrické vztahy mezi objekty.

2.2.6 PROGRAMOVACÍ JAZYKY PRO 3D POČÍTAČOVOU GRAFIKU

Vývoj grafických aplikací v sobě často zahrnuje použití více jazyků. Nejčastěji se pravděpodobně můžeme setkat s použitím jazyka *C++* v kombinaci s aplikačním rozhraním *OpenGL*, nebo *DirectX*, kdy v *C++* je napsána logika programu a přes grafické aplikační rozhraní je vyřešeno zobrazování grafiky. Vývojáři často používají jazyky jako *GLSL,HLSL* nebo *CG*, díky kterým se vyhnou kryptické podobě nízko úrovňového kódu grafických aplikačních rozhraní. U aplikací, které zobrazují složité 3D modely a animace, produkci samotné aplikace často předchází tvorba těchto modelů a animací. Ty často vytvářejí umělci v různých specializovaných programech bez použití programovacího jazyka.

Nicméně mnoho pokročilých uživatelů si usnadňuje práci právě tím, že za pomocí kódu automatizují některé úkony nebo si díky kódu vyvíjí nové funkční prvky. Pro tyto účely je v oblasti počítačové grafiky nejvíce rozšířený skriptovací jazyk *Python*. Rozdíl mezi skriptovacími a programovacími jazyky je takový, že kód programovacího jazyka (např. *C++*) je po zhotovení programu zkompilován do podoby, která obsahuje příkazy přímo pro hardware, zatímco skriptovací jazyky kompilovány nejsou. Skriptovací jazyky jsou interpretovány za běhu. Zkompilované programy jsou díky tomuto faktu mnohem rychlejší, jelikož se za běhu nemusejí převádět, ale zároveň je jejich vývoj složitější. Jazyk *Python* je poněkud výjimkou, jelikož umožňuje dynamickou kompilaci, což znamená, že se určitým způsobem zkompiluje před každým spuštěním. Díky tomu, je dostatečně rychlý a zároveň jednoduchý.

II. PRAKTICKÁ ČÁST

V této části diplomové práce bude provedena detailní analýza současných technologií týkajících se realtimové grafiky. Na závěr bude uvedeno zdůvodnění a bližší specifikace zvolené technologie.

3 ANALÝZA SOUČASNÝCH TECHNOLOGÍÍ

Pro analýzu současných technologií jsem zvolil kategorizaci do tří hlavních kategorií, které v určitých případech obsahují další podkategorie. I když se tyto kategorie v některých případech prolínají, stále bude toto dělení užitečné. Hlavní kategorizace je založena na vývojovém prostředí, pro které je technologie určena. Dělí se tedy na: Technologie určené pro osobní počítače, technologie určené pro mobilní zařízení a technologie určené pro web.

3.1 TECHNOLOGIE URČENÉ PRO OSOBNÍ POČÍTAČE

Technologie pro osobní počítače jsou zpravidla nejsofistikovanější, jelikož osobní počítače jsou z těchto tří výše uvedených prostředí nejstarší a nejvýkonnější.

3.1.1 APLIKAČNÍ ROZHRANÍ GRAFICKÉHO PROCESORU

Mezi dvě hlavní technologie, které se dotýkají každého vývojáře v oblasti grafiky, patří aplikační rozhraní *OpenGL* a *Direct3D*. V minulosti, když v počítači neexistoval grafický procesor, tak všechny transformace *vertexů* a jiné úlohy vyžadované k vykreslení scény zpracovával procesor (*CPU*). Existovala pouze určitá část zv. *framebuffer*[5] určena ke grafickým účelům, která umožňovala ukládat a načítat pixely, které měly být zobrazeny na obrazovce monitoru. Programátoři museli k vykreslování grafiky implementovat vlastní algoritmy v samotném softwaru. V tomto smyslu byly veškeré procesy týkající se *vertexů* a fragmentů programovatelné. Bohužel procesor nebyl dostatečně výkonný, aby produkoval požadované 3D efekty. Dnes již 3D aplikace procesor k renderovaní nevyužívají. Namísto toho spoléhají na aplikační rozhraní *OpenGl* či *Direct3D*, prostřednictvím kterého komunikují s grafickým procesorem.

[5]Buffer, v překladu je vyrovnávací paměť, je část paměti, která je určena pro dočasné uchování dat před jejich přesunem na jiné místo.

3.1.1.1 OPENGL

OpenGL bylo vyvinuto v počátcích devadesátých let minulého století korporací *Silicon Graphics* ve spolupráci s organizací zvanou *OpenGL architecture review board (ABR),* která se skládala z hlavních výrobců grafických systému. Původně bylo *OpenGL* určeno pouze pro výkonné pracovní stanice založené na *Unixovém* operačním systému. *Microsoft,* který byl zakládajícím členem *ABR* později implementoval *OpenGL* jako podporu 3D grafiky pro operační systém *Windows NT*. Později byla přidána podpora *OpenGL* do všech operačních systému společnosti *Mirosoft*. Jeho hlavní výhoda tkví v tom, že není limitováno jedním typem operačního systému. Kromě podpory *Unixu* a systému *Windows* je *OpenGL* podporováno operačními systémy počítačů *Macintosh* a dnes standardně i operačními systému typu *Linux*. Tato flexibilita dělá z *OpenGL* nejlepší multiplatformní aplikační rozhraní pro 3D grafiku. *OpenGL* je dobře nadstavitelné, což znamená, že vývojáři mohou přidávat nové funkční prvky přírůstkově. Jakmile jsou tyto nadstavby ustálené, často se stávají součástí standardního jádra *OpenGL.*

Později byl pro *OpenGL* představen tzv. *stínovací jazyk GLSL*, který programátorům umožnil vyhnout se nízko úrovňovému kódu. V současnosti udržuje vývoj *OpenGL* konsorcium *KHRONOS*, jehož součástí je mnoho korporací, kterých se nějakým způsobem dotýká počítačová grafika. V době, kdy vzniká tato diplomová práce, je poslední verze určená veřejnosti označena číslem 4.2.

3.1.1.2 DIRECT3D

Microsoft začal vyvíjet aplikační rozhraní *Direct3D* okolo roku 1995 jako součást své multimediální iniciativy zvané *DirectX*. *Direct3D* je jedno z mnoha rozhraní, které vytvářejí *DirectX*. Za vývojem knihoven *DirectX* stál záměr společnosti *Microsoft*, kterým bylo vytvoření herního trhu pro platformu *Windows.* V počátcích se tato iniciativa setkala s velkým odporem. Mnoho významných herních vývojářů, jako např. John Carmack, napsalo *Microsoftu* otevřené dopisy, v kterých urgovali potlačení "této nedokonalé technologie v zájmu *OpenGL*".(10) V budoucnu se však ukázalo, že konkurenční prostředí prospělo oběma aplikačním rozhraním. Postupem času se *DirectX* stalo nejrozšířenějším aplikačním rozhraním pro vývoj her v prostředí *Windows*. Dokonce i jeho původní odpůrci později ocenili vlastnosti, kterými disponuje. *Microsoft* vydává nové verze v krátkých časových

horizontech, někdy až jednou ročně. Současná verze je *DirectX 11*, která obsahuje specifikaci stínovacího jazyka *HLSL*, což je alternativa výše zmíněného GLSL určená výhradně pro Direct3D. Za největší nevýhodu *Direct3D* považují mnozí lidé nekompatibilitu s ostatními systémy. Existují sice konvertory, které dokážou převézt *Direct3D* do *OpeGL*, to však není optimální, protože dochází k "osekání" mnoha funkcí.

Dnes jsou schopnosti obou aplikačních rozhraní velice podobné, jelikož jsou určeny pro tentýž hardware, který tyto schopnosti umožňuje. *OpenGL* je lehce ve výhodě, díky tomu, že je otevřené a výrobci hardwaru mohou skrze něj zpřístupňovat mnoho funkcí. To však může způsobovat problémy, protože tyto funkce mohou být využity pouze u specifického hardwaru.

Mnoho softwarových vývojářů volí aplikační rozhraní na základě cílového trhu, nebo preferencí programátora.

3.1.2 OFFLINE RENDEROVACÍ ENGINY ZALOŽENÉ NA GPU

Díky architektuře grafických procesorů *CUDA* a aplikačnímu rozhraní OpenCL, které umožňuje přístup k paralelním procesům, začalo v poslední době vycházet mnoho renderovacích enginu, které vypočítávají výsledný obraz prostřednictvím grafické karty. Tyto renderovací enginy jsou často, pravděpodobně z marketingových důvodů, označovány zavádějícím pojmem "realtimové renderovací enginy". Ve skutečnosti však mají k pojmu realtime daleko, jelikož se v nich mohou obrázky vykreslovat až několik hodin. Jsou sice mnohem rychlejší než klasické renderery fungující přes procesor a u některých objektů může být vykreslování téměř instantní, nicméně k interaktivitě to nestačí a proto tyto renderery slouží pouze k předrenderování obrázků, nebo animací. Mezi nejznámější renderovací enginy založených na *GPU* patří například *VrayRT GPU* vyvinutý společnosti *Chaosgroup* nebo *Arion* vydaný společností *Random control*. Obě tyto společnosti vydávají stejný renderovací engine i pro *CPU*. *Vray RT GPU* je v některých případech oproti klasické verzi *Vray* až 20x rychlejší, porovnáme-li standardní grafickou kartu se standardním procesorem. Největší nevýhoda těchto rendererů je, krom toho, že nepodporují některé funkce, jejich omezená kapacita operační paměti. To znamená, že je možno renderovat

pouze scény s menšími nároky na paměť. Scény pro vizualizaci produktů tuto podmínku splňují, větší architektonické scény však nikoliv.

3.1.2.1 NVIDIA OPTIX – NVIDIA SCENIX

Výjimkou mezi těmito renderovacími enginy je iniciativa společnosti *NVIDIA* s názvem *OPTIX*. *OPTIX* je engine, který umožňuje vypočítávat obraz pomocí metody *raytracingu* téměř v reálném čase. Stejně jako ostatní *GPU* renderery i zde hraje roli omezení velikosti scény. *NVIDIA OPTIX* funguje v tandemu s enginem pro správu scény *NVIDIA SCENIX*. Oba tyto enginy jsou však spíše určeny pro profesionální grafické karty vyšší cenové kategorie, jako například *NVIDIA QUADRO*, nebo *NVIDIA TESLA*. Ačkoli tyto enginy nejsou skutečně realtimové (trvá to pár sekund, než obraz tzv. vyšumí dočista), v oblasti vizualizace produktového designu, zvláště pak automobilového se uchytily s velkým úspěchem. Využívá je například program pro produktovou vizualizace *RTT DELTAGEN*.

Obrázek 43: obrázek vyrenderovaný pomocí technologie NVIDIA SceniX a OptiX

3.1.3 HERNÍ ENGINY

Vývojáři počítačových her zpravidla vyvíjí hry za pomocí herních enginů. Tyto enginy si buďto naprogramují sami, nebo použijí engine vytvořený někým jiným. Zjednodušeně řečeno je herní engin souhrn kódů, který je vyšší abstrakcí problémů herního světa, který se vývojář snaží vytvořit. Vývojáři nejprve vytvoří robustní systém, který v sobě obsahuje mnoho elementů potřebných pro samotnou hru. Herní enginy zpravidla obsahují renderovací engine, engine pro fyziku, systém pro animaci, umělou inteligenci, správu paměti a jiné funkce. I když je vývoj herního enginu často nákladný, z dlouhodobého hlediska je však ekonomický výhodný díky opětovného používání u dalších projektů. Tyto enginy jsou kromě počítačových her využívány i u jiných interaktivních aplikací, jako jsou marketingové dema, architektonické vizualizace a v neposlední řadě výcvikové simulace. Grafika těchto enginů stojí na jednom nebo obou aplikačních rozhraních grafického procesoru *(OpenGL, Direct3D)*. Některé nabízejí flexibilní kompilaci pro více druhů hardwaru, což je pro vývojáře velmi výhodné, jelikož se nemusí zabývat kompatibilitou. Přestože je pravděpodobně nejideálnější pracovat s vlastním enginem, mnoho úspěšných her bylo vytvořeno právě na enginu jiných firem nebo vývojářů. Mezi jedny s nejznámějších a nejrozvinutějších enginů patří například *Unreal Engine* vyvinutý společnosti *EPIC GAMES*, *Cryengine* vyvinutý společnosti *CRYTEK*, nebo novější *Unity3D* společnosti *UNITY TECHNOLOGIES*. Stejně jako je to s každou technologií i tyto mají své přednosti a nevýhody.

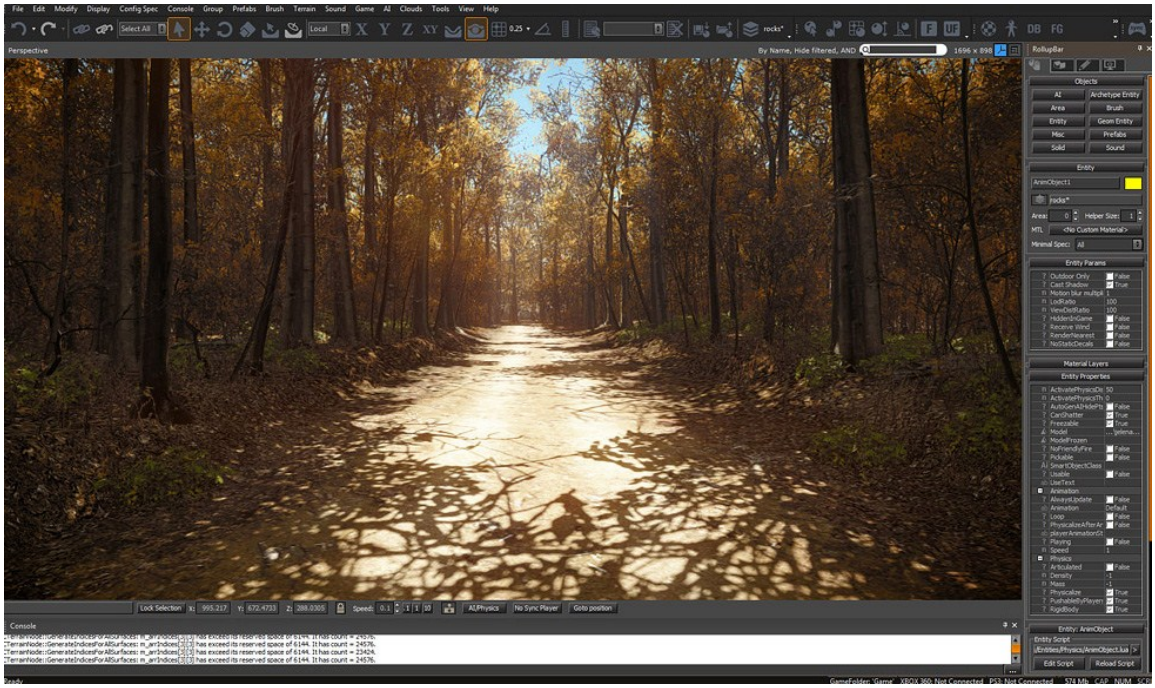
3.1.3.1 UNREAL ENGINE

Unreal engine byl vytvořen v roce 1998 firmou *EPIC GAMES*. Původně byl tento engine určen pro hry typu FPS (střílečka z pohledu první osoby) a první hra, ve které byl použit, nesla název *Unreal*. Tento engine je update-ován často až jednou měsíčně, a tak poskytuje ty nejmodernější technologické prvky herních enginů. Vedle *Unreal Engine* vydává *EPIC GAMES* vývojové prostředí *UDK* (*Unreal Development Kit*). Prvotním účelem *UDK* bylo umožnit hráčům tvorbu vlastních módů, což je termín označující hru v upravené podobě. Hráči si často vytvářeli zcela nové prostředí, které pak mezi sebou sdíleli. Toto se stalo tak populární až se z *UDK* stal plnohodnotný engine a má veškeré funkce *Unreal Engine*. Jediným jeho omezením je, že uživatelé nemají přístup ke zdrojovému kódu. Velkou výhodou *Unreal Engine* a tím pádem i *UDK* je velká uživatelská základna. Obecně platí, že čím větší uživatelská základna, tím snazší vývoj, jelikož na internetu bude existovat mnoho návodů a rad.

Obrázek 44: obrázek vyrenderovaný Unreal Enginem v reálném čase

3.1.3.2 CRYENGINE

Cryengine byl původně vyvinut jako technologické demo, které mělo demonstrovat schopnosti grafických karet *NVIDIA*. Po velkém úspěchu se z tohoto dema vyvinula celá hra s názvem *FAR CRY*. Mezi hlavní výhody *Cryenginu* patří v první řadě jeho vysoká míra realističnosti. Často je zmiňován jako engine s nejvyšším realismem. Je to také proto, že byl zacílen na nejvyšší grafické karty. Stejně jako *Unreal Engine* nabízí všechny prvky moderního herního enginu. Mimo to, že tento engine společnost *CRYTEK* licencuje mnoha studiím, v červnu roku 2011 byla uvolněna verze pro nekomerční účely ke stažení zdarma. Tato verze je identická s plnohodnotnou verzí. Nicméně ve srovnání s *Unreal Engine* nemá *Cryengine* tak rozsáhlou uživatelskou základnu, což je velká nevýhoda.



Obrázek 45: vysoce realistická scéna vytvořená v Cryenginu

3.1.3.3 UNITY3D

Unity3D je další z oblíbených enginů a také je to jeden z těch novějších. Mnoho vývojářů jej volí pro jeho multiplatformní přístup. *Unity3D* podporuje kromě standardních operačních systémů také operační systémy pro mobilní zařízení, jako je *iOS* (*ipad, iphone*), nebo *Android*. Velkou výhodou je také možnost implementovat hru do internetového prohlížeče,

díky speciálnímu pluginu, který společnost *UNITY TECHNOLOGIES* vyvinula. *Unity3D*nicméně neposkytuje tak komplexní vývojové nástroje a realistickou grafiku jako dříve zmiňované enginy.

Obrázek 46: hra v Unity web playeru

Existuje mnoho dalších enginů, které mají své specifické výhody a nevýhody. Pro produktovou vizualizaci jsou však nejvhodnější tyto příklady.

3.1.4 EUCLIDEON

Jednou z velmi zajímavých technologií poslední doby je renderovací engine *UNLIMITED DETAIL* společnosti *EUCLIDEON*. Tento engine není založený na polygonech jako všechny ostatní enginy, ale na tzv. atomech, jak je *EUCLIDEON* nazývá. Ve skutečnosti se jedná o technologií *voxelů*, se kterou se vývojáři snažili pracovat již v minulosti. Díky neúspěchům této technologie se pojem *voxel* stal v oblasti herního průmyslu spíše nepopulární. Voxel je oproti polygonům miniaturní bod a *EUCLIDEON* tvrdí, že dokáže zpracovávat scény až s trilióny takovýchto bodů. Toto je údajně možné díky chytrému vyhledávacímu algoritmu, který zobrazuje pouze to, co je potřeba. Tato technologie se nicméně setkává s velkým skepticismem, protože mnoho lidí věří, že je nepraktická pro interaktivní animace, která je u her potřebná. V oblasti produktové vizualizace by to nemusel být až takový problém.

Obrázek 47: technologie společnosti Euclideon

3.2 TECHNOLOGIE URČENÉ PRO MOBILNÍ ZAŘÍZENÍ

Očekává se, že kolem roku 2013 bude více lidí prohlížet internetové stránky z mobilů,(11) než z osobního počítače. To znamená, že bude mnohem více lidí, kteří budou mít mobily schopné zpracovávat složitou grafiku. Grafické procesory v dnešních *smartphonech* a přenosných tabletech jsou již schopny zobrazovat 3D efekty. Stále se však nemohou srovnat s výkonnosti stolních počítačů, a proto na nich nefungují aplikační rozhraní *OpenGL* a *Direct3D*. Jak *OpenGL*, tak *Direct3D* však poskytují specifickou implementaci pro mobilní zařízení *OpenGL ES* a *Direct3D* mobile.

Poněvadž *Microsoft* nemá na poli mobilních zařízení takové zastoupení jako u stolních počítačů je *OpenGL ES* v této oblasti nejvíce využívané. *OpenGL ES* je stejně jako *OpenGL* nízkoúrovňové aplikační rozhraní pro 2D a 3D grafiku. Ve skutečnosti je to verze *OpenGL*, která je „okleštěná“ o určité funkce, aby byla zajištěná kompatibilita se všemi mobilními zařízeními, které mají grafický procesor. Díky tomu mohou být aplikace vyvinuté pomocí *OpenGL ES* snadno spuštěny v systémech, které podporují klasické *OpenGL*. Při vývoji aplikací určených pro mobilní zařízení by mělo být pamatováno také na způsob interakce s těmito zařízeními. Mobilní zařízení dnes běžně obsahují dotykovou obrazovku s technologií *multitouch* umožňující velmi intuitivní a rychlou interakci, která může být dobře využitelná například u prohlížení 3D modelů. Stejně jako u osobních počítačů existují i pro mobilní zařízení herní *frameworky*, které usnadňují vývoj. Sem patří například výše zmiňované *Unity3D*, které podporuje většinu operačních systémů pro mobilní zařízení nebo *Unreal Engine*, který v současné době podporuje pouze operační systém produktů Apple iOS. Oba tyto enginy fungují za pomocí *OpenGL ES*.

3.3 TECHNOLOGIE URČENÉ PRO WEB

Každý dnes nepochybně chápe, jakou roli ve veškerém vývoji hraje internet. Je logické, že čím lépe se budou šířit informace, tím rychleji půjde vývoj kupředu. Díky internetu a vyhledávacím enginům je přístup k informacím víceméně instantní. V počátcích byl internet spíše statický, obsahoval mnoho textů a obrázků a nenabízel uživateli příliš mnoho možnosti k interakci. Časem se to však změnilo a internet jakoby ožil. Sami uživatelé začali tvořit obsah internetu a komunikovat skrze něj. Tato éra se začala označovat pojmem *Web*

2.0. V současnosti můžeme v mnoha diskusích zaslechnout termín *Web 3.0*. Tímto termínem někteří lidé označují dnešní prostředí internetu, které je charakterizováno jinými zvyky prohlížení a celkové interakce. Vše je dnes mnohem instantnější. *Web 3.0* je často charakterizován mimo jiné integraci třetího rozměru. Dalším často omílaným pojmem internetového žargónu je tzv.*WebOS* (*web operating system*), což označuje přechod od klasických aplikací spouštěných lokálně na počítači k aplikacím, které se spouštějí přes internetový prohlížeč. Tyto aplikace mají obrovskou výhodu, neboť uživatel nemusí nic instalovat a může okamžitě začít pracovat. Samozřejmě díky rychlosti internetu a omezenému přístupu prohlížeče k výkonu počítače nemůže být tento přístup (aplikace v prohlížeči) aplikován ve všech oblastech. Nicméně se očekává, že veškeré programy pro běžné uživatelé se budou v budoucnu spouštět právě přes internetový prohlížeč. Není tedy divu, že vzniklo mnoho iniciativ, které se pokouší implementovat 3D grafiku do internetového prohlížeče. Mezi tyto počiny patří například *WebGL*, které je moderními prohlížeči nativně podporováno a dále pluginy, jako je *Flashplayer, Silverlight, Unity Web Player*, nebo *Java*. Stejně jako ostatní technologie mají i tyto své výhody a nevýhody. Před popisem jednotlivých technologií je potřeba zmínit, co se při prohlížení internetových stránek odehrává. V počátcích internetu, když byly stránky pouze statické, tak k vytvoření stránky stačil jen jazyk *HTML*. Tento jednoduchý jazyk obsahuje příkazy pro internetový prohlížeč, který je interpretuje do podoby, kterou vidíme na obrazovce. Později byla díky kaskádovým stylům *(CSS)* umožněna větší kontrola nad vzhledem webových stránek. Ke skutečné interaktivitě nicméně bylo zapotřebí skriptovacího jazyka. Ten byl začleněn na dvou úrovních.

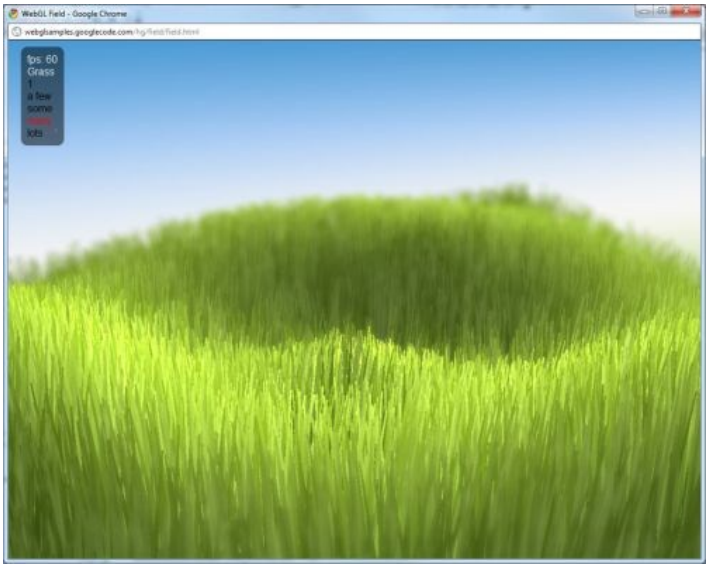
Když prohlížíme internetové stránky, kód těchto stránek je uložen někde na serveru, odkud se odesílá do našeho počítače, kde jej zpracovává a vykresluje internetový prohlížeč. Existují tedy dva druhy jazyků, pomineme-li *HTML* a *CSS*. Jeden zpracovává internetový prohlížeč a druhý se zpracovává na serveru, odkud se pak posílají již zpracované výsledky. Jako standardní skriptovací jazyk pro internetový prohlížeč se uchytil *Javascript*. Všechny populární prohlížeče v sobě mají zakomponovány *javascriptový* engine, který interpretuje *javascriptový* kód přicházející ze serveru. Většina animací a vyskakovacích oken byla dodnes vytvořena právě přes *javascript*, pomineme-li přídavné moduly prohlížečů. Pro serverovou stranu se nejčastěji používá jazyk jako je *PHP* nebo *Python*. Tyto jazyky většinou zpracovávají složitější logiku, jako je například práce s databázovými daty.

Další nedílnou součástí dnešního internetu jsou různé doplňky poskytující další funkční prvky. Tyto doplňky mají tu výhodu, že pracují nezávisle na internetovém prohlížeči. Jed-

nou napsaný kód pro tento doplněk bude fungovat v každém prohlížeči, pokud má uživatel daný doplněk nainstalovaný. Nejrozšířenější doplněk zvaný *Flashplayer* vyvíjí společnost *ADOBE*.

3.3.1 WEBGL

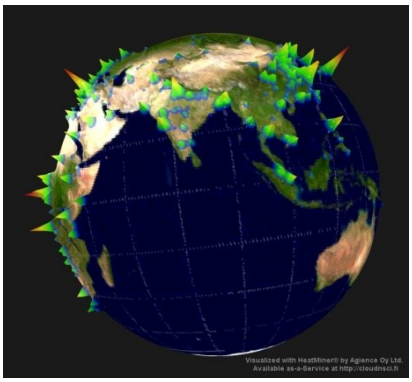
WebGL vyvinulo konsorcium *KHRONOS*, které mimo jiné vyvíjí také *OpenGL*. Je založeno na implementaci *OpenGL ES 2.0* a weboví vývojáři jej používají ve spojení s *Javascriptem*. *WebGL* je jediná technologie, která dokáže zobrazovat 3D grafiku v prohlížeči bez použití pluginu. Na druhou stranu je tak podporováno pouze u nových verzí prohlížečů, kromě *Internet Exploreru*, který jej nepodporuje vůbec. U mobilních zařízení je *WebGL* podporováno operačním systémem *Android*, nikoliv však systémem *iOS*. Vývoj *WebGL* je teprve v počátcích[6] a často se setkává s kritikou ohledně bezpečnostních děr, které mohou být pro uživatelé nebezpečné. *Microsoft* uvedl tento fakt, jako důvod proč se rozhodl *WebGL* neimplementovat do prohlížeče *Internet Explorer*.(12)



Obrázek 48: obrázek vytvořený v prostředí

3.3.2 JAVA APPLETY

Další možnou volbou, jak dostat 3D grafiku do internetového prohlížeče jsou tak zvané *Java applety*. K jejich spuštění je potřeba mít nainstalovaný doplněk *Java,* a jelikož *Java applety* neumožňují přístup k *GPU*, tak se 3D grafika zpracovává softwarově, což má velmi omezené možnosti. Proto je 3D grafika zobrazovaná pomocí *Java appletů* víceméně na



Obrázek 49: Java applet

[6]první specifikace tohoto jazyka vyšla v březnu 2011

ústupu.

3.3.3 MICROSOFT SILVERLIGHT

Doplněk *Silverlight* se dá považovat za přímou konkurenci *Flashplayeru*. V současné verzi je 3D grafika v *Silverlightu* vykreslována pouze softwarově, od verze 5 by však měla být vykreslována pomocí hardwarové akcelerace. Velkou výhodou má být podpora frameworků *XNA*, která se používá pro vývoj her. Naopak velkou nevýhodou je, že *Silverlight* má oproti *Flashplayeru* minimální uživatelskou základnu.

3.3.4 UNITY WEB PLAYER

Tento doplněk vyvinula společnost *UNITY TECHNOLOGIES*, aby bylo možné hrát hry vyvinuté prostřednictvím herního enginu*Unity3D* online přes internetový prohlížeč. Do nedávna to byla nejvyspělejší technologie pro sofistikovanější 3D grafiku v internetovém prohlížeči. Bohužel má tento doplněk ještě menší uživatelskou základnu, než dříve zmiňovaný *Silverlight*. To by se však podle prohlášení *UNITY TECHNOLOGIES* mělo v blízké době změnit. Engine *Unity 3D* by měl umožňovat export hry do formátu, který bude možné spouštět přes *Flashplayer*. Pokud by se tak stalo, tak by se vývojářům používající tento engine otevřely dveře k velmi silné uživatelské základně.

3.3.5 FLASH PLAYER

Tento doplněk určený pro přehrávání multimediálního obsahu si drží první místo v žebříčku nejvíce distribuovaných softwarů a to nejen mezi doplňky internetového prohlížeče, ale i softwarem obecně. Je nezávislý jak na operačním systému, tak na prohlížeči. Podle statistik má verzi *Flashplayer 10.0* nainstalováno 98-99% uživatelů internetu.(13) Nejnovější verze 10.3 je na tom přibližně o polovinu hůř, což je však u adaptací nových verzí běžné.

Schopnosti zobrazovat 3D grafiku v sobě *Flashplayer* má již dlouho. Ta byla doposud podporována pouze softwarově. Projekt z názvem *Molehill* spuštěným v prvním čtvrtletí tohoto roku vystavuje vývojářům nízko úrovňové aplikační rozhraní, díky kterému jim

bude umožněno naplno využít výkon grafické karty. Zatímco předchozí verze *Flashplayeru* umožňovaly zobrazovat polygony v řádech tisíců, díky tomuto aplikačnímu rozhraní může být zobrazovaný počet polygonů až stokrát vyšší. Na platformách *Windows* bude toto aplikační rozhraní komunikovat s grafickou kartou prostřednictvím *DirectX 9*, na *MacOS* a v *Linuxu* prostřednictvím *OpenGL 1.3* a u mobilních platforem bude podporováno *OpenGL ES 2.0*.

Vývojáři mohou k vývoji aplikací použít mnoho frameworků, které jim mohou vývoj značně usnadnit. Mezi nejpopulárnější frameworky patří *ALTERNATIVA3D, AWAY3D, COPPERCUBE, FLARE3D, MINKO, SOPHIE3D,* nebo v budoucnu snad dokonce *Unity3D*.

Obrázek 50: 3D grafika ve Flashplayeru

Mezi dvě nejsilnější technologie určené pro zobrazování 3D grafiky na webu patří nepochybně *WebGL* a *Flashplayer*. Výhody *Flashplayeru* jsou především uživatelská základna a také jeho rychlost. Ve srovnání s *WebGL* je rychlejší a díky tomu i mírně realističtější. *WebGL* má oproti tomu velkou výhodu v tom, že je to otevřená technologie a proto může být jasnou volbou mnoha vývojářů. V této chvíli jsou obě technologie v úplných počátcích a lze jen těžko předpovídat, jak se jim bude v budoucnu dařit.

3.3.6 TECHNOLOGIE CLOUDU

Další technologie, která umožňuje prohlížení 3D grafiky díky internetu, je tzv. *technologie Cloudu*. Tato technologie je poněkud ojedinělá a v jistém smyslu revoluční. Zatímco všechny předchozí technologie týkající se internetu byly příkladem *klient/server paradigmatu*, v tom smyslu, že ze serveru se posílá kód, který klient zpracovává a vykresluje výsledný obraz, u *technologie Cloudu* je tomu spíše naopak. Klient posílá příkazy serveru, který je zpracovává a odesílá již hotový obraz zpátky klientovi. Obecně nemusí mít *technologie Cloudu* nic společného s počítačovou grafikou. Jedná se o nové paradigma v oblasti informačních technologií, kdy je výpočetní kapacita brána spíše jako služba, než jako produkt. To znamená, že uživateli je přes síť poskytován pouze výstup aplikace, zatímco veškeré výpočty se dějí v *Cloudu*, což je zjednodušeně síť počítačů pracujících dohromady. V tomto *Cloudu* jsou většinou uložena i veškerá data. Uživatelům odpadají starosti se správou systému a zároveň nemusí utrácet peníze za drahý hardware. Platí pouze za výpočetní sílu. Tento koncept by se dal uvést do analogie k distribuci elektrické energie, která je vyráběna v elektrárnách a uživatel ji odebírá, bez toho aniž by věděl, jak vzniká. K aplikacím, které běží v *Cloudu* uživatelé často přistupují skrze internetový prohlížeč.

V poslední době se *technologie Cloudu* uchytila také v herním průmyslu. Dvě hlavní společnosti *GAIKAI* a *ONLIVE* slibují uživatelům možnost hrát ty nejnovější hry na jakémkoliv hardwaru. Projekt *OTOY*, který je více vizionářský, a prozatím pouze v podobě dema, si klade mnohem vyšší cíl. Chce díky superpočítačům posunout hranice realističnosti dnešních her, tím že bude v reálném čase vypočítávat výsledný obraz pomocí velmi náročné metody *raytracingu*. Nicméně tato technologie se *mainstreamovému* uplatnění v blízké době jistě nedočká. Projekty *GAIKAI* a *ONLIVE* jsou oproti tomu přístupné komukoliv, kdo bude mít dostatečně vysoký *bandwidth* (rychlost připojení).

Oba projekty nabízejí podobnou službu. Hlavní rozdíl je v tom, že díky službě *GAIKAI* mohou uživatelé hrát ty nejnovější hry přímo v internetovém prohlížeči (je třeba mít nainstalovaný *Flashplayer*, nebo *Javu*), zatímco u *ONLIVE* je potřeba stáhnout a nainstalovat aplikaci, díky které mohou uživatelé hry vyhledávat a následně hrát. *ONLIVE* zároveň nabízí hardware, který lze připojit k televizi a hrát hry bez použití počítače. Hlavním technologickým problémem u obou projektů je latence při přenosu dat. Jelikož pokud se přenáší celý obraz, tak je to mnohem větší objem dat, než kdyby se přenášel pouze kód pro vykreslování obrazu. Zároveň je také potřeba udržet latenci mnohem menší než u aplikací jiného

typu, jelikož by to mohlo narušit plynulost hry. Latence závisí na vzdálenosti mezi datacentrem a uživatelem. Proto se obě společnosti v současnosti snaží co nejlépe pokrýt spojené státy výstavbou nových datacenter. Společnost *ONLIVE* si klade za cíl také pokrytí Velké Británie a s menší latencí si lze službu *ONLIVE* vyzkoušet také v Česku. Tyto technologie mají jak své příznivce, tak i odpůrce. Mnoho významných postav herního průmyslu vidí tuto technologii jako velice praktickou jak v distribuci her, tak v schopnosti zamezení pirátství. Jestli bude výpočetní síla v budoucnu centralizovaná, jak tomu je dnes u různých druhů energií, však prozatím zůstává otevřenou otázkou.

Obrázek 51: hra *WorldofWarcraft*- spuštěná pomocí služby *GAIKAI*

3.4 ZVOLENÁ TECHNOLOGIE

Na začátku bych chtěl poznamenat, že jsem při vývoji aplikace dvakrát přešel na jinou technologii. Pokaždé, když jsem pronikl do problematiky dané technologie, tak jsem ji nakonec opustil a začal nanovo. Naštěstí se mnoho konceptů opakuje u všech technologií, a tak nepřišlo veškeré úsilí nazmar. V tomto místě bych chtěl popsat, jak a proč jsem se tak rozhodoval. Nejdříve však musím zmínit, jaká technologie by byla pro vývoj aplikace tohoto typu ideální.

Tato technologie by měla splňovat následující požadavky:

- vysoká míra realističnosti
- implementace do internetového prohlížeče
- opensource
- rozsáhlá uživatelská základna
- kvalitní dokumentace
- vývojové prostředí, které bude poskytovat funkce usnadňující práci

Taková technologie nicméně prozatím neexistuje, a proto je třeba obětovat některé body na úkor jiných.

První technologie, s kterou jsem začal experimentovat, bylo *WebGL*. Bylo to v létě 2010, když ještě nebyly žádné zmínky o projektu *Molehill*, který umožňuje hardwarově akcelerovanou grafiku ve *Flashplayeru*. Zároveň jsem také nevěděl nic o *Cloudových* řešeních. *WebGL* má výhodu hlavně v dokumentaci, jelikož je postaveno na verzi *OpenGL ES 2.0* a mnoho konceptů lze nastudovat i z dokumentace samotného *OpenGL*. Uživatelská základna je poměrně rozsáhlá, *WebGL* je podporováno všemi novějšími verzemi prohlížečů, kromě *Internet Exploreru*. Další výhodou je, že se jedná o opensourcový projekt a existuje mnoho frameworků, které usnadňují vývoj.

Když později na podzim 2010 vyšlo demo, které demonstrovalo schopnosti projektu *Molehill*, rozhodl jsem se přejít na tuto technologii, poněvadž otevírala dveře k mnohem větší uživatelské základně (je počítáno s adaptací nové verze *Flashplayeru*) a zároveň byla rychlejší a mírně realističtější. Podpora frameworků, které usnadňují vývoj, také nechybí a tyto frameworky jsou často sofistikovaňejší než ty u *WebGL*, což je důsledkem toho, že již v minulosti existovalo mnoho vývojářů, kteří vyvíjeli hry s 3D grafikou pro *Flashplayer*. Mezi nevýhody této technologie patří její uzavřenost (uživatelé nemají přístup k nejnižší úrovni kódu) a také úroveň dokumentace, která je v současnosti minimální. Vývojářům bylo slíbeno, že v první čtvrtině roku 2011 bude uvolněno aplikační rozhraní, které bude umožňovat vytváření hardwarově akcelerovaných 3D aplikací. Začal jsem se tedy učit programovací jazyk *actionscript 3*, který je potřeba k vývoji aplikací pro *Flashplayer* a frameworky pro 3D grafiku ve *Flashi*.

Další změna přišla na začátku roku 2011, když jsem objevil *technologie Cloudu* určené pro hry a vývojové prostředí *Unreal Development Kit* (*UDK*). I když jsem o *UDK* věděl, původně jsem jej zavrhl, protože to byla technologie, která neumožňovala spouštění přes internetový prohlížeč. Díky *technologii Cloudu* je však umožněno vyvíjet hry a aplikace pomocí herních enginů pro osobní počítače (které jsou mnohem výkonnější než technologie určené pro web) a později *streamovat* tyto hry a aplikace přes internet. Uživatel si je tak může spouštět přímo v prohlížeči. S přechodem na *UDK* jsem musel dočasně obětovat implementaci do internetového prohlížeče na úkor vysoce realistické grafiky, což bylo pravděpodobně nejkomplikovanější rozhodnutí.

3.5 UNREAL DEVELOPMENT KIT (UDK)

UDK je volně stažitelná verze *Unreal Enginu (UE)* s jediným rozdílem, že jeho uživatelé nemají přístup ke zdrojovému kódu. Jak *UE* tak *UDK* však mají dvě úrovně kódu. Nižší úroveň, která je napsána v *C++,* skrze které lze tento engine od základu přetvořit k obrazu svému a vyšší úroveň, která je napsaná v *Unrealscriptu*, který byl vyvinut se specifickými požadavky vývoje her. *Unrealscript* je komplexní objektově orientovaný jazyk s mnoha prvky moderních programovacích jazyků. Uživatelům *UDK* je k dispozici pouze *Unrealscript*. Součástí *UDK* jsou vizuálně založené editory, ve kterých uživatelé tvoří veškeré vizuální prvky aplikace, které pak propojí s vnitřní logikou aplikace pomocí *Unrealscriptu*. Mezi tyto editory patří například editor pro tvorbu materiálů, správu animací, tvorbu vizuálních a zvukových efektů a mnoho jiných pokročilých funkcí. Většina těchto editorů je založena na tvz. *nodech*, což umožňuje velmi intuitivní vývoj. *UDK* má dobře zpracovanou dokumentaci a internetová fóra jsou plna zkušených vývojářů. Výsledná aplikace je zkompilována do *exe* souboru a v této podobě může být distribuována. Na závěr je nutno říci, že v UE byly vytvořeny jedny z nejkvalitnějších počítačových her.

III. PROJEKTOVÁ ČÁST

4 Vývoj aplikace

V této části budou představeny funkce výsledné aplikace společně s postupem práce a jednotlivými metodami, které byly použity při tvorbě modelu i samotné aplikace.

4.1 POPIS PROJEKTU

Jako objekt, který bude pomocí výsledné aplikace prezentován, jsem zvolil model autobusu, na kterém jsem v poslední době pracoval. Jedná se o repliku dobového autobusu *Škoda*, z roku 1908, která je „napasována“ na podvozek dopravního vozu *Mercedes Sprinter*. Realizaci této repliky zaštiťuje společnost *Polyuni s.r.o.*, která má v úmyslu vyvážet tento vůz do zahraničí jako vozidlo pro turistický ruch. Původně jsem autobus vizualizoval pomocí statické vizualizace v podobě předrenderovaných obrázku. U této formy jsme při konzultacích narazili na určité problémy, které bych chtěl pomocí výsledné aplikace vyřešit. Mezi hlavní problémy patřilo zjištění určitých rozměrů a sladěnost barevných variant a v neposlední řadě vzhled vozidla z jiných úhlů pohledu a skrytých části. Výsledná aplikace bude řešit tyto požadavky, spolu s dalšími funkcemi, jako je animace funkčních prvků a zobrazení popisků jednotlivých části po najetí kurzoru.

Obrázek 52: Render modelu autobusu, který bude použit pro demonstraci aplikace

4.2 POSTUP PRÁCE

I když vývoj aplikace většinou nemá lineární průběh a mnoho úkonů lze vykonávat současně, pro snazší pochopení však budou jednotlivé úkony seřazeny lineárním způsobem. Celý vývoj by se dal dobře rozdělit na dvě části, z nichž jedna je tvorba obsahu a druhá programování logiky aplikace. Jelikož programování logiky aplikace je technicky nejnáročnější a je s ním spojeno mnoho nečekaných problémů, věnuje se první část vývoje právě jemu. Místo konkrétních objektů se dočasně používají tzv. "*Dummy*" objekty, na kterých se otestuje, zda kód funguje správně. Později se tyto objekty nahradí za ty konkrétní.

Zde však bude celý vývoj popsán, jak bylo zmíněno výše, lineárně, od tvorby modelů, přes animace a programování až po tvorbu uživatelského rozhraní. U každé části bude uvedena ukázka objasňující daný proces.

4.2.1 JEDNOTLIVÉ KROKY PŘI TVORBĚ APLIKACE

Samozřejmě prvním krokem při tvorbě takovéto aplikace bude vytvoření plánu a systému celého vývoje. Aby nedošlo k nečekaným problémům, je třeba vše promyslet dopředu, zvolit náležité jmenné konvence, organizace a formáty dat. Toto obvykle nelze učinit pouhým přemýšlením, a proto je potřeba otestovat všechny problematické body, v kterých nevíme, jak se data budou chovat. Nejčastější problémy se objevují při přechodu z jedné aplikace do druhé. Je potřeba znát všechny důležité konvence, aby se mohlo předejít zbytečným často velice náročným procesům předělávání. Pokud je zvolen náležitý postup práce a jsou zachovány všechny konvence, může být vývoj velice hladký a krátký. Při samotné invenci postupu práce je však potřeba počítat s nesčetným předěláváním a proto je z pravidla lepší jej dělat "nanečisto" a vyzkoušet chováni dat v nepředpokládaných situacích.

4.2.1.1 MODELOVÁNÍ OBJEKTU

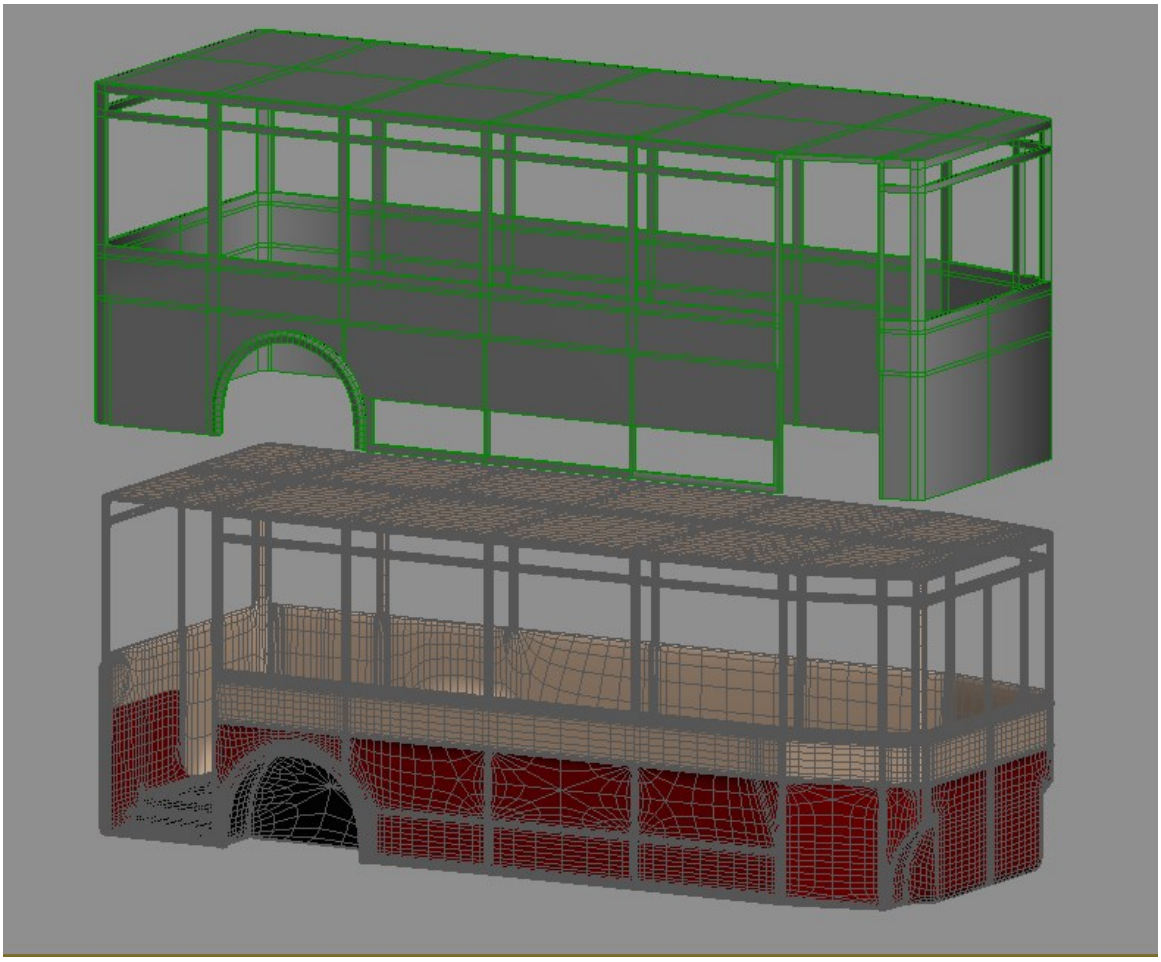
Při modelování objektů je nejdůležitější zachovat co nejméně komplexní a čistou topologii. Celý objekt je třeba prozkoumat a promyslet umístění hran polygonů. Z pravidla je nejideálnější pracovat pouze se čtyřstrannými polygony z důvodu snadné editace. To však často vede k vysokému počtu polygonů a u složitých modelů to může být značně komplikované, proto se často používají i trojstranné polygony. Je možno použít i vícestranné polygony,

neboť herní enginy veškeré polygony před vykreslením triangulizují. U animovaných objektů, kde dochází k deformaci polygonů, je lepší nastříhat vícestranné polygony na trojstranné a mít tak kontrolu nad tím, podél kterých hran se budou polygony při animaci lámat. U statických objektů toto není zapotřebí, jelikož to na výsledek nemá žádný vliv. Výsledný model by měl obsahovat co nejmenší počet polygonů. Modely určené k animaci by měli obsahovat maximálně kolem deseti tisíc polygonů. Vše se nicméně odvíjí od komplexnosti celé scény. V případě této aplikace bude scéna obsahovat pouze jeden objekt, složeny s jednotlivých komponentů a proto není počet polygonů až tak velkým omezením. V případě detailní geometrie, jako jsou různé struktury povrchu, se používají tzv. normálové mapy[7], které se vygenerují z highpoly modelů[8]. Tento highpoly model může obsahovat až několik milionů polygonů. Normálové mapy jsou pomocí UV souřadnic naneseny na lowpoly model, na kterém vytváří iluzi detailu.

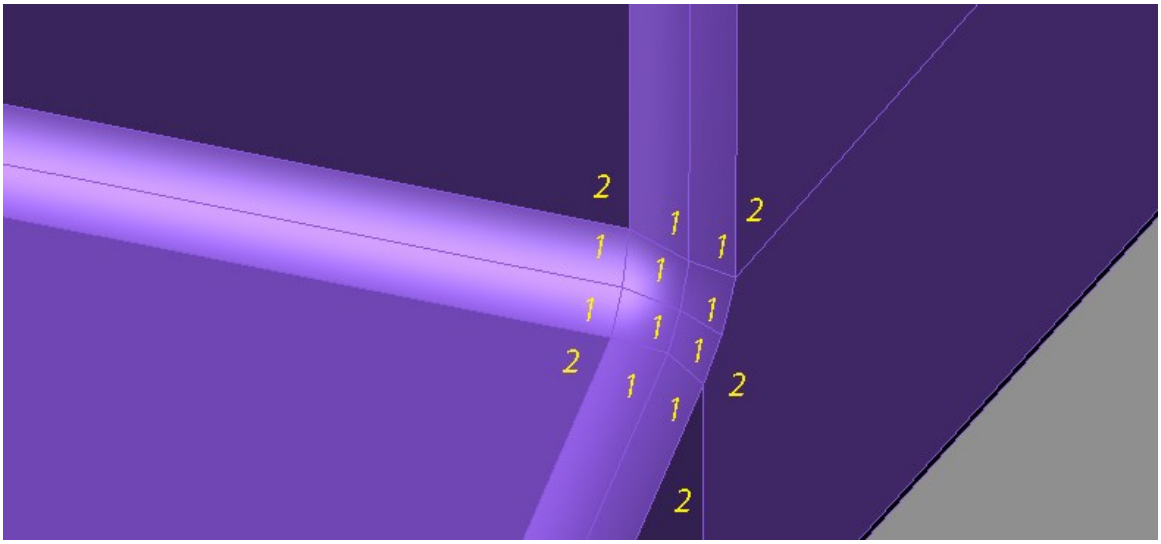
Zároveň je třeba pamatovat na duplicitní objekty. Pokud je ve scéně vícekrát stejný objekt, stačí jej vymodelovat pouze jednou. V herním enginu se pak vloží vícekrát jako instance, tím pádem bude zabírat místo v paměti pouze jednou. Dalším důležitým konceptem jsou tzv. "*smoothing groups*". Pokud bude skupině polygonů přiřazena stejná "*smoothing group*", tak budou mít přechody mezi těmito polygony hladký průběh. Pokud bude mít sousední polygon jinou "*smoothing group*", tak bude mezi těmito polygony ostrá hrana.

[7] Normálové mapy vytvářejí na modelech s nízkým počtem polygonů iluzi prostorového detailu. Tyto mapy jsou vytvořeny pomocí modelů s vysokým počtem polygonu a je v nich definováno odsazení geometrie, o které se má model s nízkým počtem polygonů odsadit, aby vypadal jako model s vysokým počtem polygonů.

[8] Highpoly model je název, který se používá pro detailní modely s vysokým počtem polygonů. Lowpoly model je naopak model s nízkým počtem polygonů



Obrázek 53: Srovnání modelu s nízkým a vysokým počtem polygonů. Jelikož byl model původně vytvářen pouze pro předrenderovanou vizualizaci, musel se později přemodelovat do verze s nízkým počtem polygonů.

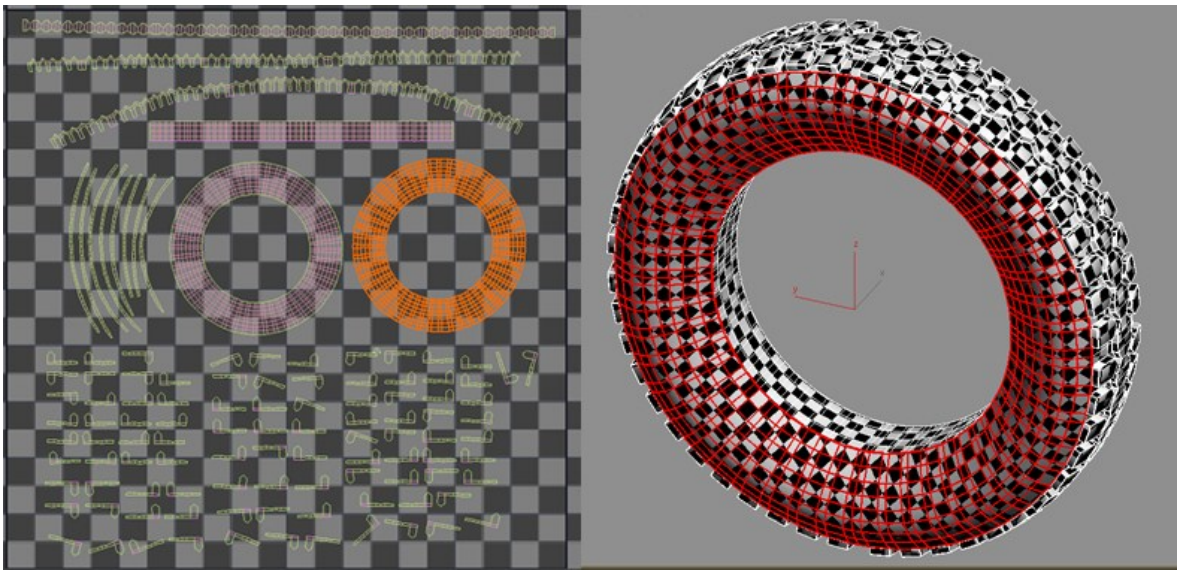


Obrázek 54: smoothing groups určující průběh mezi sousedními polygony

4.2.1.2 VYTVÁŘENÍ UV PLÁTŮ

Po vytvoření modelu je potřeba vytvořit UV pláty, které budou sloužit pro aplikaci textur. Pro tyto účely je obzvlášť výhodné mít čistou topologii polygonů. Existuje mnoho metod, jak lze tyto UV pláty vytvořit, přes automatické metody, které vytvoří UV pláty například na základě úhlu mezi sousedními polygony, až po ruční metodu, kdy jsou na modelu označeny hrany, podél kterých se mají vytvořit švy a v těchto místech se pak textury rozpojí. Jelikož u herních modelů je vždy nejideálnější mít co nejmenší počet švů, protože v těchto místech mohou vznikat problémy např. při zapékání normálových map, proto je lepší si vytvořit UV pláty ručně. Proces vytváření UV plátů je jedna z méně oblíbených rutin při vytváření 3D modelů a do budoucna se očekává, že bude potřeba tohoto procesu eliminována. Ve filmovém průmyslu se tak již stalo s příchodem formátu *PTEX*. Tento formát vyvinula společnost *Disney* jako opensourcový projekt, aby díky němu mohla zjednodušit produkci animovaných filmů. Textury v tomto formátu nejsou na model nanášeny pomoci UV souřadnic, ale pomocí jednotlivých polygonů. Každý polygon má svou vlastní část textury a tyto části jsou mezi sebou interpolovány na základě přilehlých polygonů. Z těchto důvodů je však nemožné vytvářet textury v 2D programu jako je *ADOBE PHOTOSHOP,* jelikož by dvojrozměrná podoba textury nedávala smysl. U tohoto formátu se textury kreslí přímo na 3D modelu ve specializovaných aplikacích, jako je např. *Mari*, kterou vyvinula společnost *The Foundry*. Očekává se, že se v budoucnu tento formát uchytí i v herním průmyslu, nicméně v současnosti je u herních modelů stále zapotřebí vytvářet textury pomocí UV souřadnic.

Často se také stává, že model může obsahovat více UV kanálů. Díky tomu lze model otexturovat za použití více metod. Pro některé části modelu je totiž výhodné použít stejné UV souřadnice. Když budu chtít například otexturovat krychli, aby všechny její strany obsahovaly stejnou texturu, tak UV pláty nastříhám na jednotlivé strany krychle a ty pak v UV prostoru poskládám nad sebe. Toto se provádí za účelem optimalizace velikosti textur. V některých případech je však zapotřebí, aby všechny strany měly svůj vlastní UV prostor. Proto existuje více UV kanálů, které mohou obsahovat jinak rozdělené UV pláty.

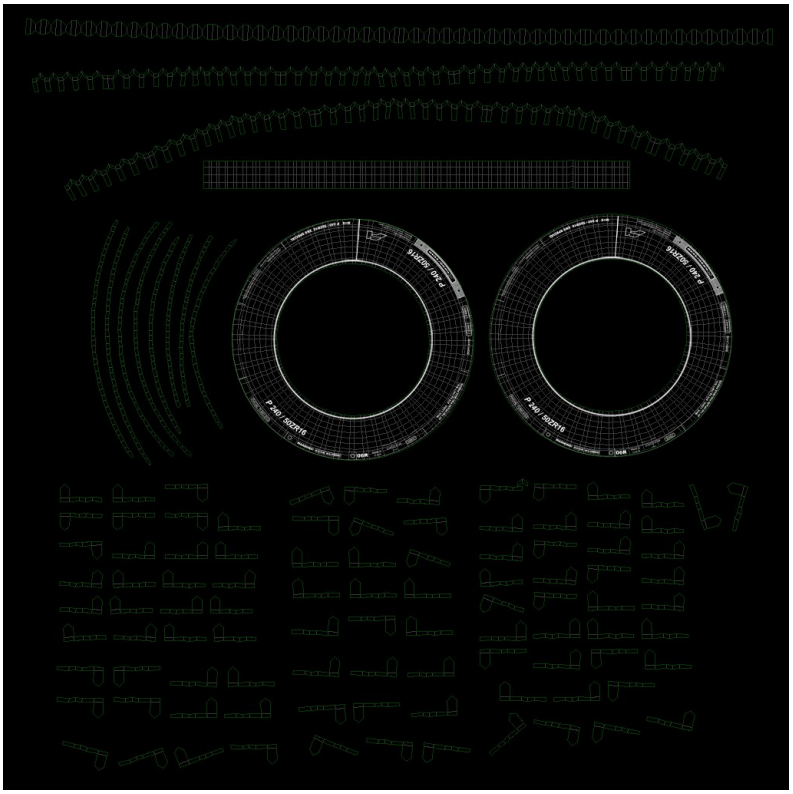


Obrázek 55: UV pláty

4.2.1.3 TEXTUROVÁNÍ

Jakmile jsou vytvořeny UV pláty, tak je možno začít s procesem texturování. To může mít podobu jak kreslení textur v 2D programu, tak kreslení textur přímo na 3D model. Kreslení na 3D model má tu výhodu, že přímo vidíme, jak textura na modelu vypadá a zároveň odpadá problém konzistentnosti textur v místech, kde se nacházejí švy rozdělující jednotlivé UV pláty. Pokud budeme vytvářet textury ve 2D programu a budeme mít ve švech složité vzory, tak je téměř nemožné zachovat mezi nimi plynulý přechod. Avšak v případě nanášení pouze pár jednoduchých textur, které nepotřebují být umístěny ve švech, je 2D program jako je *PHOTOSHOP* často jednodušším řešením. V tomto programu se jako podklad pro kresbu používá šablona vytvořených UV plátů, které slouží jako vodící linka při texturování. Pro realističtější vzhled se často předrenderovává textura s vystínováním modelu, která se sloučí dohromady s barevnou texturou. To lze nicméně provést pouze u statických modelů, u kterých se nemění poloha vzhledem k osvětlení scény. V *UDK* jsou

textury naneseny na model pomocí materiálu. Tyto materiály obsahují mnoho komponent, kterým může být přiřazena jiná textura. Jednotlivé komponenty ovlivňují různé vlastnosti materiálu jako je barva, svítivost, lesk průhlednost atd.

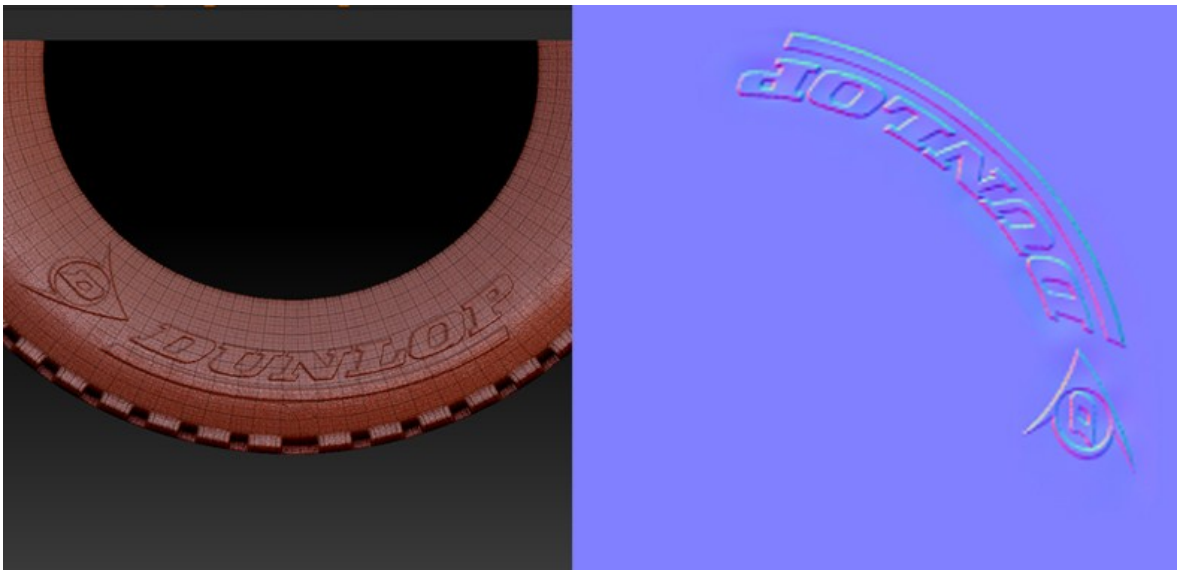


Obrázek 56: textura kola s šablonou UV plátů, které slouží jako vodící linka při texturování.

4.2.1.4 ZAPÉKÁNÍ NORMÁLOVÝCH MAP

Normálové mapy se používají pro vytvoření iluze prostorového detailu, který se v topologii objektu ve skutečnosti nenachází, jelikož by jeho vymodelování značně zvýšilo počet polygonů a výsledná polygonová síť by byla příliš hustá na to, aby se s ní dalo manipulovat v reálném čase. Formát normálové mapy obsahuje kanály pro červenou, zelenou a modrou barvu. Každý kanál náleží ve stejném sledu jedné z os X,Y,Z. Zastoupení těchto barev v určitém bodě určuje posunutí geometrie podél daných os. Tyto normálové mapy se vytvářejí z *highpoly* modelů, které obsahují žádané detaily. *Highpoly* model se zarovná s *lowpoly* modelem, z *lowpoly* modelu jsou vyslány paprsky ve směry normál jednotlivých ploch a do normálové mapy se zapíše vzdálenost bodu protnutí s *highpoly* modelem v jed-

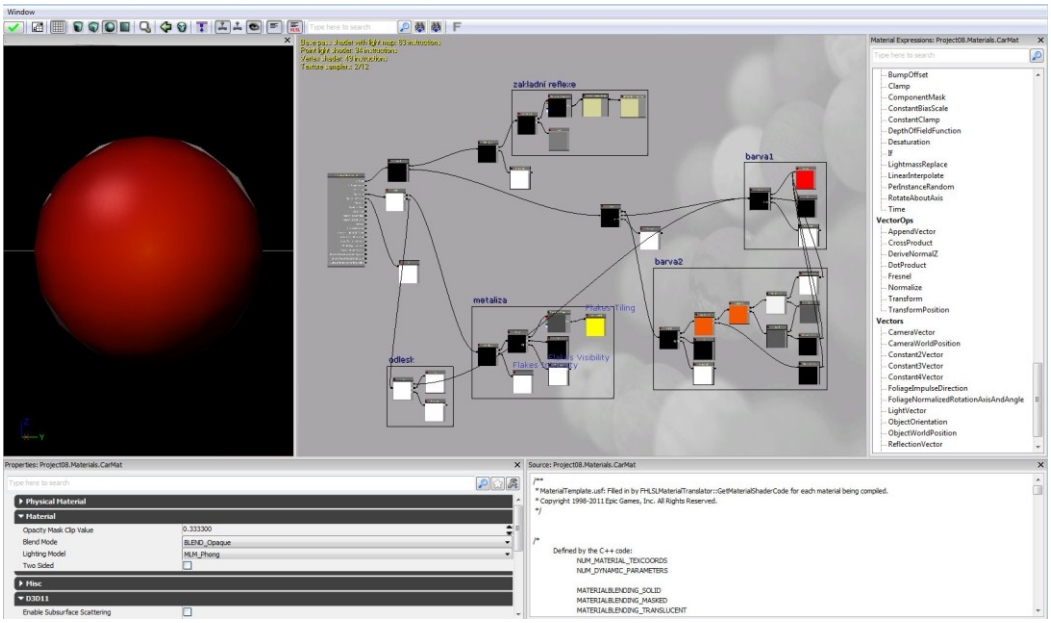
notlivých osách. Nakonec se normálové mapy aplikují na *lowpoly* model, na kterém vytvářejí iluzi detailu.



Obrázek 57: detail highpoly modelu, který obsahuje 1,9 milionů polygonů s detailem vygenerované normálové mapy

4.2.1.5 VÝTVÁŘENÍ MATERIÁLŮ

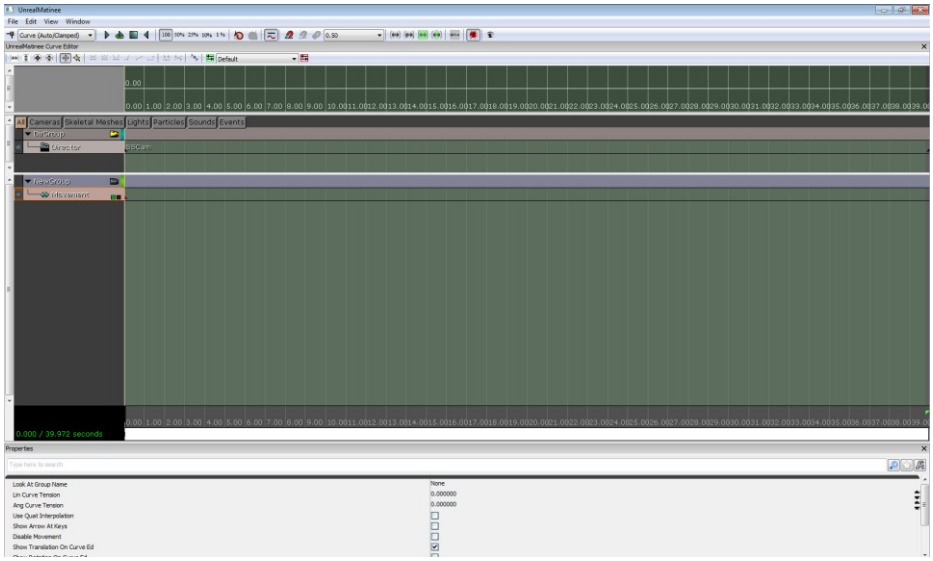
Vytváření materiálů je jeden z procesů, kterému je zapotřebí věnovat velkou pozornost, jelikož materiály mají na výsledný vzhled spolu s osvětlením největší vliv. *UDK* umožňuje vytvářet materiály pomocí sítí nodů, což je velice intuitivní a pro standardní účely dostačující. V případě, že by vývojáři tyto nody nestačily, může materiály vytvářet pomocí *HLSL* jazyka. Jednotlivé nody obsahují všechny základní matematické a jiné funkce, pomocí kterých se vytvářejí odlesky a jiné vlastnosti materiálů.



Obrázek 58: Síť nodů, které tvoří materiál karoserie autobusu

4.2.1.6 ANIMACE

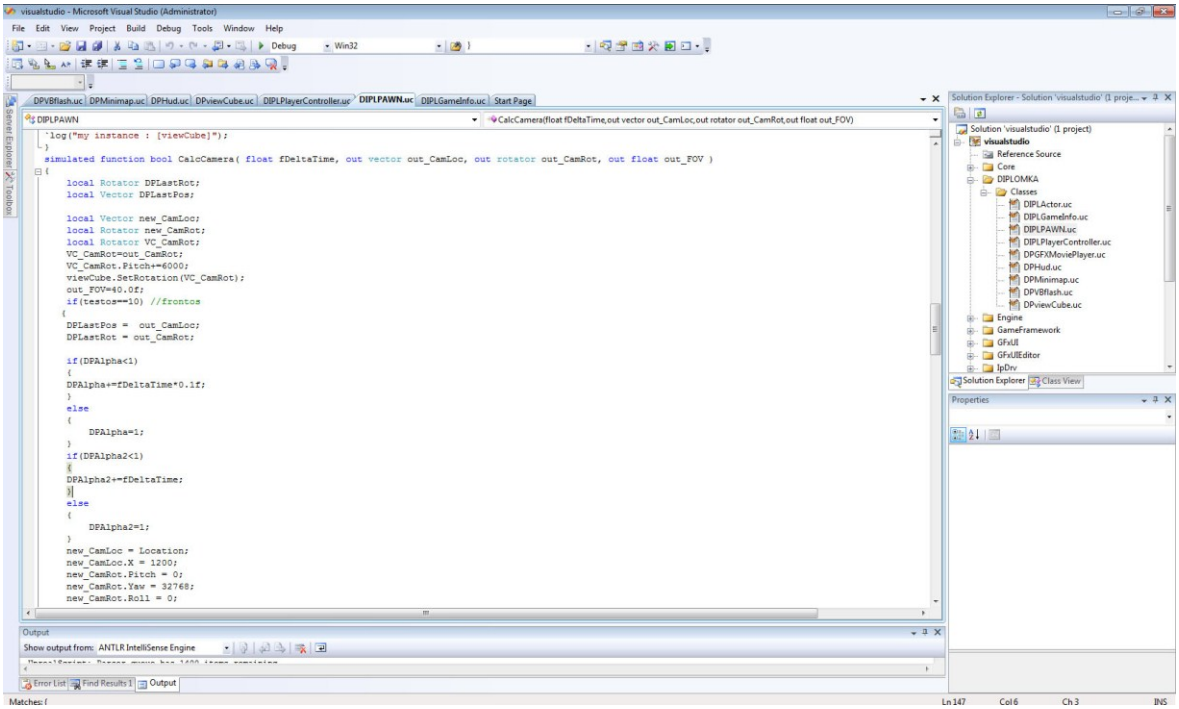
Animaci objektů lze provádět buďto v programu určeném pro animaci, nebo pokud se jedná o jednoduché transformace objektů, tak přímo v *UDK*, a to buď pomocí klíčů, nebo pomocí kódu. V případě této aplikace budou animace jednoduché, a proto se budou vytvářet přímo v *UDK*. Animace objektů jsou vytvořeny pomocí klíčů v animačním editoru *MATINEE*, který je součástí *UDK*. Animace kamer nicméně nemají lineární průběh a proto musely být vytvořeny pomocí *Unrealscriptu*.



Obrázek 59: animační editor Matinee

4.2.1.7 PROGRAMOVÁNÍ

Programování logiky je zajisté tím nejobtížnějším procesem v celém vývoji. *UDK* nabízí několik stovek *tříd*[9], z nichž některé obsahují přes 2000 řádek kódu. Protože často obtížné dohledat potřebné funkce. Z pravidla se při vývoji nadstavují již hotové třídy, které poskytují mnoho užitečných funkcí. Nová třída dědí všechny funkce svého předka i všechny ostatní funkce svých prapředků. Programovací jazyk *UDK* nese název *Unrealscript.*[10]



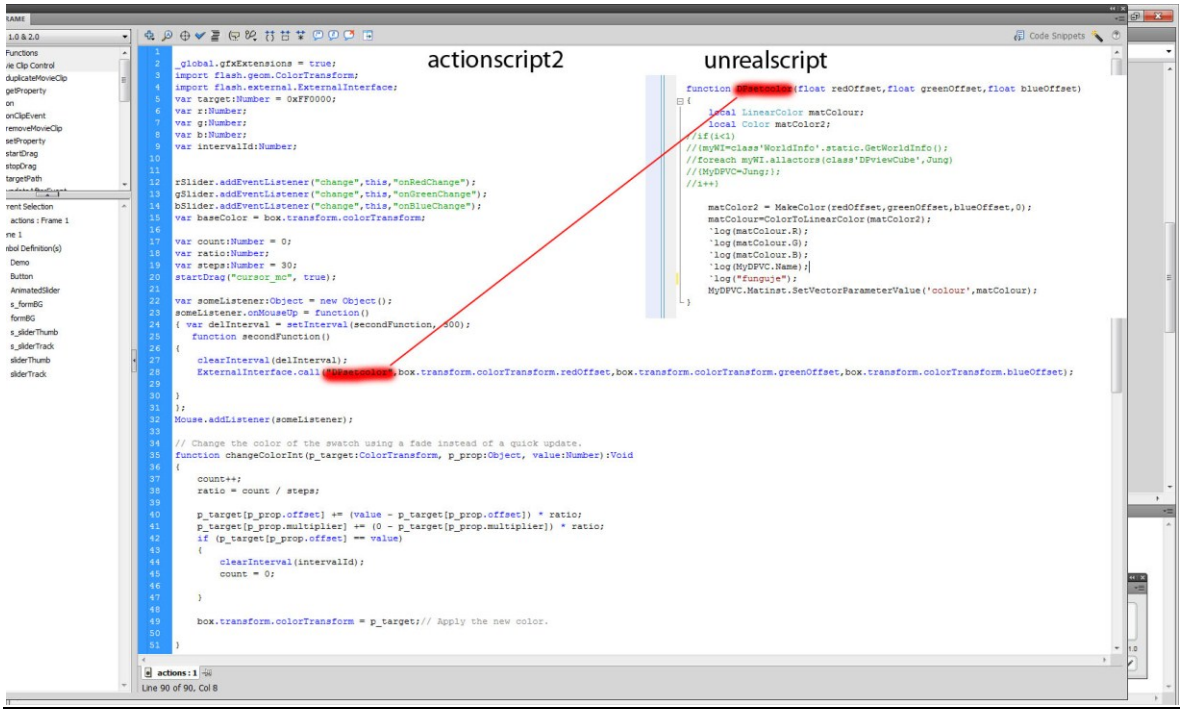
Obrázek 60: část kódu funkce, která řídí pohyb kamery

[9]Třída je základní konstrukční prvek objektově orientovaného programování sloužící jako šablona pro objekty. Definuje jejich vlastnosti a metody.

[10]Unrealscript je velice robustní programovací jazyk vyvinutý ke speciálním účelům produkce her a je na rámec této práce, aby zde byl alespoň z části popsán. viz. dokumentace unreal scriptu.

4.2.1.8 VYTVÁŘENÍ UŽIVATELSKÉHO ROZHRANÍ

Uživatelské rozhraní se v *UDK* vytváří pomocí *middlewaru*[11] třetí strany s názvem *SCALEFORM GFX* . *SCALEFORM GFX* je engine vykreslující vektorovou a rasterizovanou grafiku vytvořenou pomocí programu *ADOBE FLASH*, což je nejrozšířenější program pro tvorbu interaktivní 2D grafiky. Současná verze *SCALEFORM GFX* podporuje programovací jazyk *actionscript 3*. *UDK* bohužel obsahuje pouze starší integraci tohoto *middleware*, která podporuje pouze *actionscript* 2. Vytváření uživatelského rozhraní pomocí *SCALEFORM GFX* probíhá ve třech fázích. V první fázi je navržen vzhled uživatelského rozhraní v programu jako je *ADOBE PHOTOSHOP*, v druhé fázi je v programu *ADOBE FLASH* vytvořeno interaktivní chování prvků, pomocí časové osy a jazyka *actionscript* a v třetí fázi je skrze stejný jazyk vše propojeno s *unreal scriptem.*

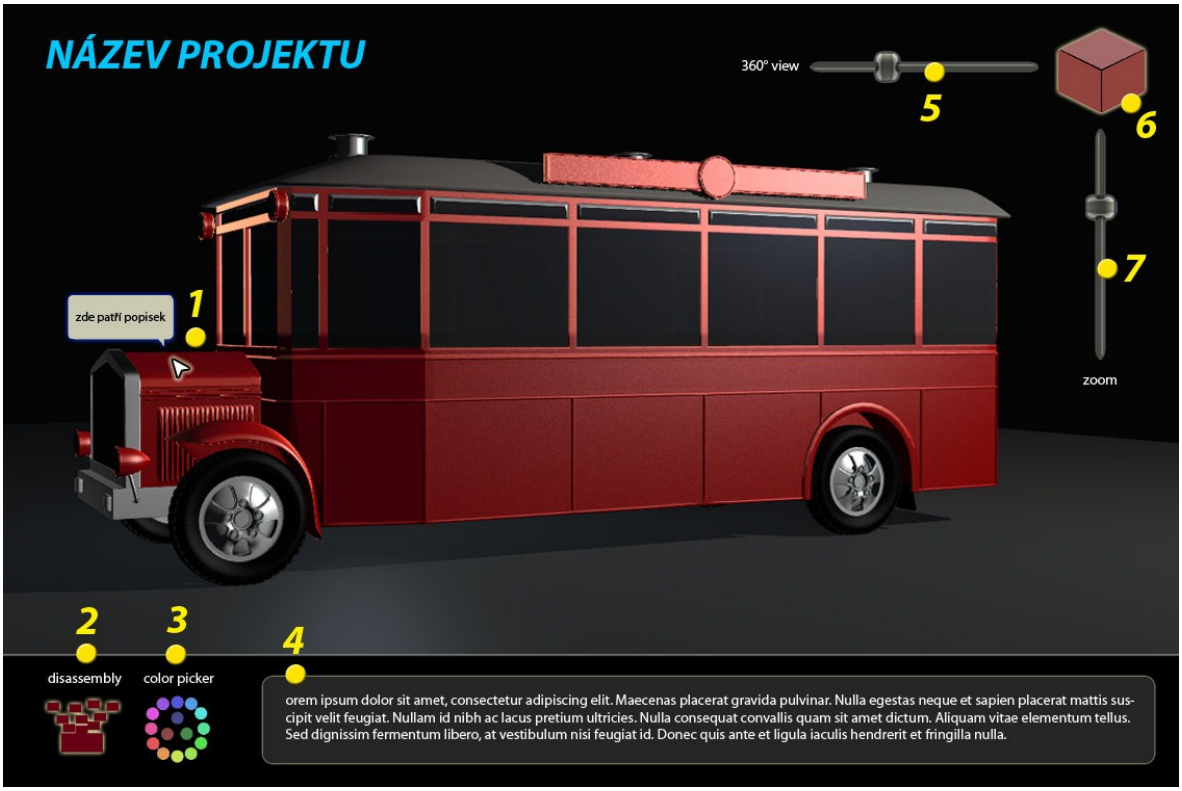


Obrázek 61: ukázka propojení kódu, který ovládá prvek uživatelského rozhraní, jehož funkcí je změna barvy, s unrealscriptem, který aplikuje výslednou barvu na model

[11] Middleware je software, který spojuje jiné komponenty softwaru.

4.3 VÝSLEDNÁ APLIKACE

V této chvíli není výsledná aplikace připravena k použití, proto zde uvádím pouze ukázky současné rozpracované verze. Finální verze bude hotova k datu obhajoby.



Obrázek 62: pracovní verze výsledné aplikace

1. **tooltip** - po najetí kurzorem na jednu z částí modelu se zobrazí název dané části
2. **ikona pro rozstřel dílů** - tato ikona zastupuje funkci, která rozloží model na jednotlivé díly
3. **ikona pro změnu barvy části modelu**- po kliknutí na tuto ikonu se otevře dialog se zásobníkem barev
4. **popis označeného dílu** - tento prostor je určen pro vložení delšího popisu právě označeného dílu
5. **360° view slider** - tento slider umožňuje rotovat scénu o 360°
6. **view cube** - tato krychle slouží pro rychlé přesunutí do jednoho ze tří hlavních pohledů (horní, přední, boční)
7. **zooming slider** - tento slider umožňuje přibližování kamery

ZÁVĚR

Cílem této práce bylo vytvoření aplikace pro produktovou vizualizaci, která bude zobrazovat 3D modely s jeho funkcemi v reálném čase. Tato aplikace není zamýšlena jako produkt, do kterého si uživatelé mohou nahrávat vlastní modely, ale jako služba, jejíž cílem bude prezentovat konkrétní objekt s jeho specifickými funkcemi. Výslednou aplikací bych chtěl demonstrovat praktické využití realtimové vizualizace, která se prozatím v oblasti produktového designu používá spíše výjimečně. Velká část obsahu této práce byla věnována zmapování problematiky jak vizualizace samotné, tak i technického zázemí, které je pro tvorbu takovéto aplikace nezbytné. Po zmapování hlavních technologických řešení vhodných pro tvorbu aplikace jsem na základě priorit zvolil technologii, která bude pro produktovou vizualizaci nejpraktičtější. Tato technologie s názvem *Unreal Developement Kit* je primárně určena pro vývoj počítačových her a jako taková v sobě již obsahuje spoustu užitečných funkcí, které celý vývoj usnadní. Jelikož tato aplikace bude sloužit pouze pro účely demonstrace, tak jsem zvolil pohodlnější proprietární řešení, což mi umožní soustředit se více na obsah aplikace než na samotný vývoj. Dočasným omezením této technologie je to, že ji nelze umístit do prostředí internetu a s tím spojená nutnost instalace. Tento problém však v budoucnu s velkou pravděpodobností vyřeší technologie *cloudu,* která umožňuje přes internet *streamovat* celou aplikaci, na rozdíl od ostatních technologií, které odesílají pouze kód, z kterého se výsledná aplikace vykresluje. Toto má velké výhody, jelikož výsledná grafika není omezená výkonností počítače, na kterém se aplikace prohlíží. Pro modelovou ukázku vizualizace jsem zvolil 3D model autobusu, na kterém jsem v poslední době pracoval. Jelikož funkcí, které by aplikace pro produktovou vizualizaci mohla obsahovat, je nespočet, zvolil jsem pouze pár obecnějších, jako je změna barvy a vlastností materiálu daného modelu, měření vzdáleností, popis jednotlivých komponentů a animace funkčních prvků samotného modelu. Realtimová vizualizace tohoto typu poskytuje mnohem více možností než předrenderovaná animace nebo statické obrázky, které jsou současným standardem, jelikož umožňuje rapidní vyhledávání informací o vizualizovaném objektu podle okamžitých potřeb daného člověka, což je při uvažování o problémech velmi důležité. Je pouze otázkou času a výkonu grafických procesorů, než se realtimová vizualizace sama stane standardem.

SEZNAM POUŽITÉ LITERATURY

1. **Cockburn, Andy.***Revisiting 2D vs 3D Implications on Spatial Memory.* Christchurch : Department of Computer Science University of Canterbury.

2. **Ware, Colin.***Visual Thinking: for Design* . s.l. : Morgan Kaufmann, 2008.

3. —. *Information Visualization.* s.l. : Morgan Kaufmann, 2004.

4. **Norman, Donald A.***Cognition in the head and in the world.* s.l. : Cognitive Science, 1993.

5. *http://www.useit.com.* [Online] září 2, 2011. http://www.useit.com/papers/heuristic/heuristic_list.html.

6. **Huizinga, Johan.***Homo Ludens: A Study of the Play-Element in Culture.* s.l. : Beacon Press, 1971.

7. **Lessig, Lawrence.***Code: And Other Laws of Cyberspace, Version 2.0.* s.l. : Basic Books, 2006.

8. **Russell, Stuart.***Artificial Intelligence: A Modern Approach (3rd Edition).* s.l. : Prentice Hall; 3 edition, 2009.

9. **Kurzweil, Ray.***The Age of Spiritual Machines: When Computers Exceed Human Intelligence* . s.l. : Penguin Bks 2000 Publishing edition, 2000.

10. *http://en.wikipedia.org.* [Online] září 2, 2011. http://en.wikipedia.org/wiki/Comparison_of_OpenGL_and_Direct3D.

11. **Tasner, Michael.***Marketing in the Moment: The Practical Guide to Using Web 3.0 Marketing to Reach Your Customers First.* s.l. : FT Press; 1 edition , 2010.

12. *http://news.cnet.com.* [Online] září 2, 2011. http://news.cnet.com/8301-30685_3-20071726-264/microsoft-declares-webgl-harmful-to-security/.

13. *http://www.adobe.com.* [Online] září 3, 2011. http://www.adobe.com/products/player_census/flashplayer/version_penetration.html.

14. *http://www.smcars.net.* [Online] září 1, 2011. http://www.smcars.net/forums/finished-work/28068-maserati-granturismo-s.html.

15. *http://www.freeorion.org.* [Online] 1. září 2011. http://www.freeorion.org/forum/viewtopic.php?f=26&t=2278.

16. *http://en.wikipedia.org.* [Online] http://en.wikipedia.org/wiki/B%C3%A9zier_curve.

17. *http://3dgrid.blogspot.com.* [Online] září 2, 2011. http://3dgrid.blogspot.com/2010/08/grid-tutorials-episode-1-modelling_5504.html .

18. *http://www.vizworld.com.* [Online] září 2, 2011. http://www.vizworld.com/2010/02/pixologic-releases-free-uv-master-plugin-zbrush/.

19. *http://www.nvidia.com.* [Online] září 3, 2011. http://www.nvidia.com/object/io_1249366628071.html.

20. *http://wiki.blender.org.* [Online] září 2, 2011. http://wiki.blender.org/index.php/File:ManAnimationEditorsIpoWindowDefaultViz.png.

21. *http://liammajor.com.* [Online] září 2, 2011. http://liammajor.com/production-line.

22. *http://downloads.khinsider.com.* [Online] září 2, 2011. http://downloads.khinsider.com/album_images/4911-mqgnmuvgwq.jpg.

23. *http://lawee.blog.cz.* [Online] 3. září 2011. http://lawee.blog.cz/1002/avatar-obrazzzky .

24. *http://news.thomasnet.com.* [Online] září 3, 2011. http://news.thomasnet.com/fullstory/CAD-CAM-Software-is-optimized-for-toolpath-control-513905.

25. *http://www.antycipsimulation.com.* [Online] září 3, 2011. http://www.antycipsimulation.com/solutions/projection-display-systems.

26. *http://hackingalert.blogspot.com.* [Online] září 3, 2011. http://hackingalert.blogspot.com/2011/05/what-is-social-graph-concepts-and.html.

27. *http://en.wikipedia.org.* [Online] září 1, 2011. http://en.wikipedia.org/wiki/File:ZUI_example.png.

28. *http://prosjekt.ffi.no.* [Online] září 4, 2011. http://prosjekt.ffi.no/unik-4660/lectures04/chapters/Introduction.html.

29. *http://csis.pace.edu.* [Online] září 4, 2011. http://csis.pace.edu/~marchese/CG/Lect1/Lecture_1.html.

30. *http://scienceblogs.com.* [Online] září 4, 2011. http://scienceblogs.com/goodmath/2007/09/fractal_mountains.php .

31. *http://blog.sixtimesnothing.com.* [Online] září 4, 2011. http://blog.sixtimesnothing.com/2010/10/bump-mapping-for-built-in-unity-terrain/ .

32. *http://www.xaraxone.com.* [Online] září 4, 2011. http://www.xaraxone.com/webxealot/workbook23/page_6.htm.

33. *http://johnkennethmuir.wordpress.com.* [Online] září 4, 2011. http://johnkennethmuir.wordpress.com/2009/12/04/cult-movie-review-tron-1982/.

34. *http://nis-ei.eng.* [Online] září 4, 2011. http://nis-ei.eng.hokudai.ac.jp/~doba/anime/pradio/README.html.

35. *http://pixaranimation.blog.cz.* [Online] 4. září 2011. http://pixaranimation.blog.cz/.

36. *http://www.awn.com.* [Online] 4. září 2011. http://www.awn.com/articles/3d/academy-honors-renderman-getting-point/page/2,1.

37. *http://michael-bien.com.* [Online] září 5, 2011. http://michael-bien.com/mbien/tags/opengl.

38. *http://www.femina.cz.* [Online] 5. září 2011. http://www.femina.cz/magazin/zabava/nejabsurdnejsi-sci-fi-filmy-vsech-dob.html.

39. *http://www.highdefdiscnews.com.* [Online] září 5, 2011. http://www.highdefdiscnews.com/?p=38932.

40. *http://dl.openhandhelds.org.* [Online] září 5, 2011. http://dl.openhandhelds.org/cgi-bin/caanoo.cgi?0,0,0,0,30,486.

41. *http://developer.amd.com.* [Online] září 5, 2011. http://developer.amd.com/archive/gpu/rendermonkey/sdk/Pages/default.aspx.

42. *http://www.gamephys.com.* [Online] září 5, 2011. http://www.gamephys.com/tag/physx-water.

43. *http://www.unrealengine.com.* [Online] září 5, 2011. http://www.unrealengine.com/news/epic_games_releases_march_2011_unreal_development_kit_beta/.

44. *http://cgeverything.co.uk.* [Online] září 5, 2011. http://cgeverything.co.uk/2011/08/18/free-cryengine%C2%AE3-sdk-for-non-commercial-use/.

45. *http://www.new-social.com.* [Online] září 6, 2011. http://www.new-social.com/?p=5685.

46. *http://www.farey.cz.* [Online] 5. září 2011. http://www.farey.cz/tag/voxel.

47. *http://www.geeks3d.com.* [Online] září 6, 2011. http://www.geeks3d.com/20101216/webgl-in-beta-in-google-chrome/.

48. *http://cloudnsci.fi.* [Online] září 6, 2011. http://cloudnsci.fi/wiki/index.php?n=HeatMiner.VisitorLocationHeatmapDemoApplet .

49. *http://sebleedelisle.com.* [Online] září 6, 2011. http://sebleedelisle.com/2011/06/webgl-and-molehill-an-overview-of-in-browser-gpu-3d/.

50. *http://www.wearetheinternetz.com.* [Online] září 6, 2011. http://www.wearetheinternetz.com/2010/05/03/ipad-running-world-of-warcraft-via-gaikai/.

51. *http://www.nvidia.com.* [Online] září 2, 2011. http://www.nvidia.com/object/io_1249366628071.html.

52. *http://www.variety.com.* [Online] září 4, 2011. http://www.variety.com/article/VR1118022620?refCatId=1050.

53. *http://www.cg-cars.com.* [Online] září 5, 2011. http://www.cg-cars.com/showthread.php/5739-VRED-Photo-Competition/page8.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

V1 - vizuální kortex 1

V2 - vizuální kortex 2

GUI - graphical user interface

ZUI - zoomovatelné uživatelské rozhraní

GNU – gnu´s not Unix

API – application programing interface (aplikační rozhraní)

CAD – computer aided design

CAM - computer aided manufacturing

MRI – Magnetic resonance imaging (magnetická rezonance)

MIT – Massachusetts Institute of Technology

VGA – video graphics array

GPU – graphics processing unit

CPU – central processing unit

GLSL–OpenGL Shading Language

HLSL – high level shading language

GPGPU – General-purpose computing on graphics processing units

CUDA - Computer Unified Device Architecture

RAM - Random-access memory

CG – C forgraphics

ABR – Architecture review board

FPS – frames per second

UDK – Unreal Development Kit

iOS- iPhone operating system

WebOS – Web operating system

UE – Unreal Engine

HTML – hypertext markup language

CSS – cascading style sheet

SEZNAM OBRÁZKŮ