VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

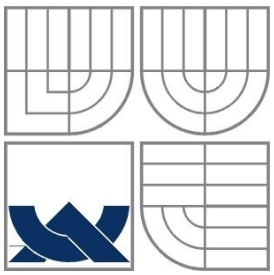
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS

# GRAFICKÉ INTRO 64KB S POUŽITÍM OPENGL

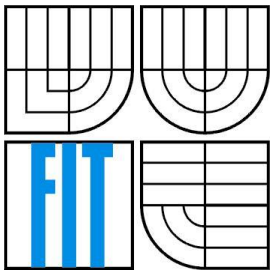
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE MARTIN PEŇÁZ
AUTHOR

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS

# GRAFICKÉ INTRO 64KB S POUŽITÍM OPENGL

GRAPHICS INTRO 64KB USING OPENGL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN PEŇÁZ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. TOMÁŠ MILET

BRNO 2015

# Abstrakt

Tato bakalářská práce se zabývá problematikou řešení grafického intra s omezenou velikostí do 64kB. Práce popisuje metody použité při vytváření grafického intra. Zabývá se procedurálním generováním, nastavením kompilátoru a finální komprimací exe packerem.



# Abstract

This bachelor’s thesis deals with the solution of graphics intro with limited size of 64kB. The work describes methods used for creation of graphics intro. It deals with procedural generation, compiler settings and final compression by exe packer application.



# Klíčová slova

OpenGL, 64kB, intro, skybox, Perlinův šum, komprese, omezená velikost, částicový systém, procedurální generování, GLSL, Phongův osvětlovací model, stínovací techniky, konstruktivní solidní geometrie

# Keywords

OpenGL, 64kB, intro, skybox, Perlin noise, compression, limited size, particle system, procedural generation, GLSL, Phong lighting model, shadow mapping, constructive solid geometry


# Citace

Martin Peňáz: Grafické intro 64kb s použitím OpenGL, bakalářská práce, Brno, FIT VUT v Brně, 2015

# Grafické intro 64kb s použitím OpenGL

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Tomáše Mileta. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……

Martin Peňáz
20. 5. 2015

## Poděkování

Rád bych poděkoval Ing. Tomáši Miletovi za jeho odborné vedení, trpělivost a cenné rady. Dále bych také poděkoval rodině a přátelům za podporu při tvorbě práce.

# Obsah

# 1 Úvod

Cílem této práce je vytvoření grafického intra s omezenou velikostí 64KB. Grafické demo je lineární a neinteraktivní, uživatel tedy nemůže nijak ovlivnit jeho děj. Grafické intro je video vytvořené programem. Intro slouží jako ukázka schopností programátora nebo skupiny, které intro vytvořili. První intra s omezenou velikostí se začala objevovat na počátku 80. let, kdy se jimi podepisovaly pirátské skupiny, které prolamovaly ochrany proti kopírování počítačových her. V dnešní době jsou intra vytvářena jako umělecká díla. Autoři mohou soutěžit v různých velikostních kategoriích od 1KB až po 64KB. Důraz je především kladen na maximální využití grafických možností a různých efektů se zachováním minimální velikosti intra.

Velikost 64KB se může zdát velmi malá při představě, že jde o grafickou aplikaci. Velká část uživatelů se setkává s grafickými aplikacemi, jako jsou počítačové hry, kde velikost může sahat až do gigabajtů dat. Proto není možné používat statická data pro grafický obsah, ale až za běhu aplikace je generovat. Modely a textury si tedy neukládáme, ale uložíme si pouze postup, jak je vygenerovat. Následně je vygenerujeme až při spuštění aplikace.

V práci jsou popsány možnosti generování grafického obsahu, jako jsou textury a geometrie. Dále jsou zde popsány grafické techniky pro výpočet stínu a osvětlení scény, způsoby jak vygenerovat a skládat šumové funkce, tvorba částicového systému, skyboxu a generování terénu.

OpenGL (Openg Graphics Library) je v práci použita pro kreslení grafického výstupu. OpenGL knihovna byla v roce 1992 vytvořena společností Sillicon Graphics inc. Od svého vydání prošla knihovna velkou řadou změn, tou největší bylo představení programovatelné kreslící pipeline, kdy od verze OpenGL 2.0 je možné za pomoci GLSL jazyka kreslící pipeline naprogramovat.

Důležitou součástí intra je i jeho příběh. Chtěl jsem vytvořit intro, které by ukazovalo přírodu. Scéna bude umístěna na golfovém hřišti, na kterém rostou stromy a tráva. Nad scénou je zobrazen procedurálně generovaný skybox s dynamickou oblohou. Po hřišti se pohybuje robot, který právě hraje golf. Výsledkem práce je grafické intro, které nepřesahuje svojí velikostí 64kB.

# 2 Teorie

V této kapitole nahlédneme na použité metody procedurálního generování, jsou zde popsány také metody pro osvětlení scény a tvorbu stínu. Seznámíme se rovněž s pojmy skybox, billboarding a L-systémy.

## 2.1 Procedurální generování

Jelikož má grafické intro omezenou maximální velikost, nebylo možné mít uložené textury ani modely jako statická data. Proto jsou všechny textury, případně modely generovány pomocí algoritmů (procedurálně). Poté je velmi jednoduché změnit výsledný vzhled za použití jiných parametrů objektů. Je také důležité, že výsledek práce je pseudonáhodný, tedy pro stejné parametry dostaneme vždy stejný výsledek. Nevýhodou generování je delší čas potřebný pro inicializaci intra.

## 2.2 Generátor náhodných čísel

Generátory náhodných čísel se používají v mnoha odvětvích informatiky. Jedná se buď o funkci nebo hardwarové zařízení, jenž má za úkol generování náhodných čísel. Pokud budeme pracovat pouze se softwarovými generátory, tak v dnešní době není možné dosáhnout opravdu skutečné náhodnosti. Deterministické algoritmy, které se k tomu používají, vytvářejí pouze pseudonáhodné posloupnosti čísel. Tyto posloupnosti nejsou tedy skutečně náhodné, ale pouze tak vypadají. Pro zlepšení této iluze se často využívá hodnota času. Pro realisticky vypadající objekty takový postup není vhodný. Největším problémem při používání prostého generátoru je nerealistický vzhled. Obraz vypadá uměle a to proto, že mezi hodnotami, které obdržíme z generátoru, není žádná spojitost. Další potencionální nevýhodou je, že u každého výpočtu obrazu dostaneme odlišný výsledek, což je v našem případě spíše překážkou.

S řešením tohoto problému přísel Ken Perlin, který vyvinul metodu zvanou Perlinův šum. Základem jeho metody je funkce, která přijímá reálné číslo, a vrací pseudonáhodnou hodnotu. Tato funkce, ale bere rozdíly na jejím vstupu, v případě, že funkce obdrží dvě velice podobné hodnoty na vstup, pak jejím výsledkem jsou náhodně vypadající, ale opět dvě velmi podobná čísla.

Existuje velké množství různých implementací takovéto funkce, zde byl použit generátor definovaný takto [1]:

$$Noise(x) = G((x \ll 13)^x) \tag{2.1}$$

$$G(x) = 1 - \frac{(x \cdot (x \cdot x \cdot 15731 + 789221) + 1376312589)\ \&\ 7fffffff}{1073741824} \tag{2.2}$$

Hexadecimální maska 7fffffff odpovídá maximální hodnotě 32bitového integeru a slouží proti možnému přetečení hodnoty. Takto definovaný generátor vrací hodnoty v rozsahu $< -1,1 >$. Jedná se o pseudonáhodný generátor, což znamená, že pro každý vstup vrátí vždy stejný výsledek. Definice odpovídá jednodimenzionálnímu generátoru. Pokud bychom chtěli generátor pro více dimenzí, můžeme toho dosáhnout například tím, že hodnotu pronásobíme prvočíslem.

$$Noise(x, y) = G((x + y \cdot 57) \ll 13)^{(x+y \cdot 57)}) \tag{2.3}$$

### 2.2.1 Interpolace

Dnešní počítače pracují pouze s diskrétním signálem, nicméně svět okolo nás je spojitý. Interpolace je matematický proces, kdy z dané diskrétní množiny, za pomoci aproximace vypočítáme zbylé hodnoty funkce. Existuje několik typů interpolace, které se liší svojí přesností a složitostí výpočtu.

**Lineární interpolace**

Nejjednodušší a nejméně náročný způsob interpolace, je lineární interpolace [2]. Jestliže jsou zadány dva body: $a_1 = (x_0, y_0)$ a $a_2 = (x_1, y_1)$, tak lineární interpolace je přímka mezi těmito dvěma body. Hodnota y je dána tedy rovnicí.

$$\frac{y - y_0}{y_1 - y_0} = \frac{x - x_0}{x_1 - 0} \tag{2.4}$$

Pokud vyřešíme tuto rovnici pro y, dostaneme rovnici, což je rovnice lineární interpolace.

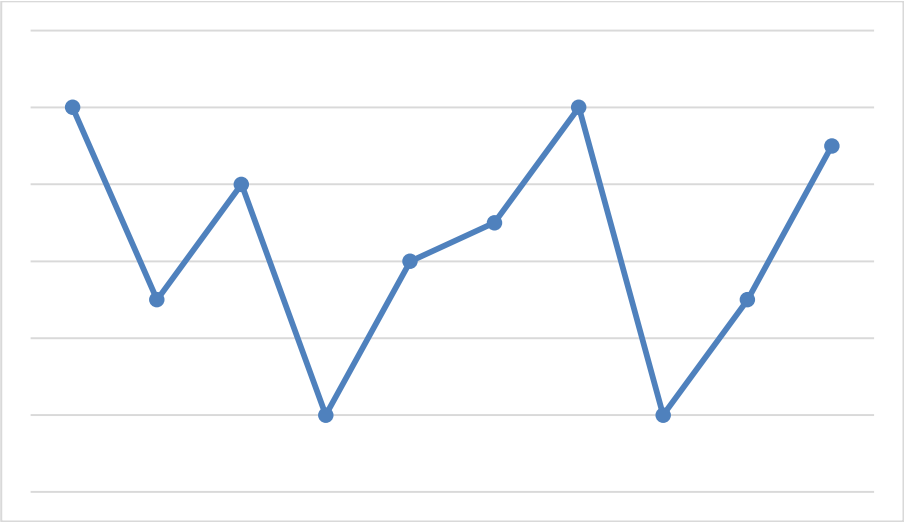
$$y = y_0 + (x + x_0)\frac{y_1 - y_0}{x_1 - x_0} \tag{2.5}$$

Tuto rovnici je možné dále zjednodušit.

$$I_l(y_1, y_2, x) = y_1 \cdot (1 - x) + y_2 \cdot x \tag{2.6}$$

Kde $y_1$ a $y_2$ jsou hodnoty získané přímo z generátoru (2.1) a $x$ je pozice hledané hodnoty mezi těmito dvěma body. Kdy pro první bod $x = 0$, pro druhý bod $x = 1$ a pro interpolované hodnoty mezi těmito dvěma body je x v rozsahu $< 0{,}1 >$.

Výhodou lineární interpolace, je její malá výpočetní náročnost, a proto je v praxi velmi využívaná, zejména v případech, kdy je nutný rychlý výpočet. Její velkou nevýhodnou je velmi nepřesný výpočet, přechody mezi body jsou velmi ostré, a tak jsou velmi nevhodné například pro goniometrické funkce.



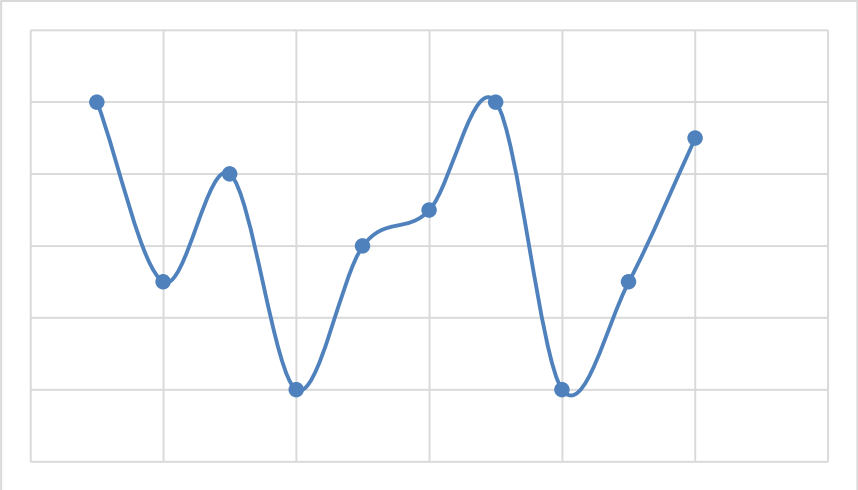
Obrázek 2.1: Ukázka lineáni interpolace

**Kosinová interpolace**

Vylepšením lineární interpolace, je interpolace kosinová [2], která částečně řeší problém velmi ostrých přechodů mezi body. Řešení je založeno na použití části funkce kosinus mezi dvěma body. Úpravou rovnice lineární interpolace (2.6), dostaneme interpolaci kosinovou. Tato metoda je mírně výpočetně náročnější, nicméně dosahuje znatelně lepších výsledků.

$$f(x) = \frac{(1 - \cos(x \cdot \pi))}{2} \tag{2.7}$$

$$I_k(y_1, y_2, x) = y_1 \cdot \big(1 - f(x)\big) + y_2 \cdot f(x) \tag{2.8}$$



Obrázek 2.2: Ukázka cosinové interpolace

### 2.2.2 Perlinův šum

Perlinův šum představil roku 1985 Ken Perlin, popsaný v [3] [4] a [1]. Jedná se o metodu vhodnou pro generování grafického šumu. Perlinův šum nám generuje pseudonáhodné hodnoty v rozsahu $\langle -1,1 \rangle$. I přestože generátor náhodných čísel bere v potaz rozdíly v hodnotách na vstupu, tak by pouhé vyhlazení pomocí interpolace nemuselo být dostačující. Například při použití dvojrozměrného šumu by na obraze při použití interpolace mohly být patrné ostré přechody. Tento problém je řešen tak, že ještě před interpolací, je krok vyhlazení. Místo toho, aby byla interpolována hodnota získaná přímo z generátoru, budeme interpolovat průměr hodnoty daného bodu a hodnot v jejím okolí. Takovéto vyhlazení zabrání funkci dosahovat lokálních extrémů nevyhlazené funkce a zároveň sníží její amplitudu zhruba na polovinu. V jedno dimenzionálním prostředí, to nemá moc praktického využití, ale ve více dimenzích docházích ke zmenšení ostrých přechodů mezi extrémy.

Vyhlazovací funkci pro jednodimenzionální prostředí je možné popsat takto:

$$SmoothNoise(x) = \frac{Noise(x)}{2} + \frac{Noise(x-1)}{4} + \frac{Noise(x+1)}{4} \tag{2.9}$$

Podobně je možné zapsat i funkci pro vyhlazení ve dvou dimenzích. Pro zjednodušení zápisu si definujeme pomocné proměnné $\alpha, \beta, \gamma$. Proměnná $\alpha$ představuje vyhlazení v diagonálním směru, $\beta$ ve směru vertikálním a horizontálním, a $\gamma$ představuje bod, který je právě zpracováván.

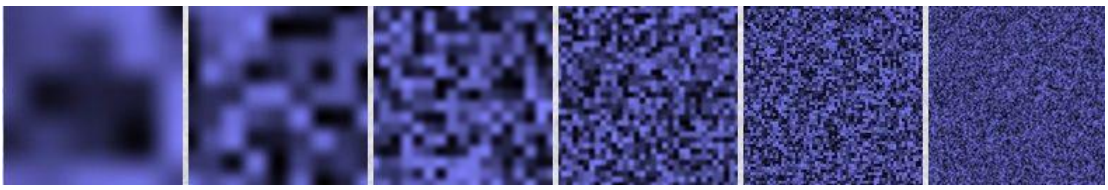
$$\alpha = \frac{Noise(x-1,y-1) + Noise(x+1,y-1) + Noise(x-1,y+1) + Noise(x+1,y+1)}{16} \tag{2.10}$$

$$\beta = \frac{Noise(x-1,y) + Noise(x+1,y) + Noise(x,y+1) + Noise(x,y+1)}{8} \tag{2.11}$$

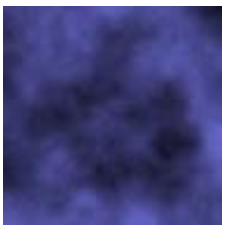
$$\gamma = \frac{Noise(x,y)}{4} \tag{2.12}$$

$$SmoothNoise2D = \alpha + \beta + \gamma \tag{2.13}$$

Takto upravený šum je možné dále vyhladit za pomoci interpolace. Můžeme použít lineární, cosinovou či kubickou interpolaci. Základní představa při generování Perlinova šumu je v tom, že vytvoříme několik funkcí $f(x)$, každá taková funkce má různou amplitudu a frekvenci a každá tato funkce představuje rozdílnou úroveň detailu výsledného obrazu. Výsledná hodnota je poté spočítána jako součet šumových funkcí o různých frekvencích a amplitudách. Taková složená šumová funkce se nazývá Perlinův šum. Výsledek šumu je závislý na zvolených parametrech: frekvence, amplitudy, perzistence a počtu oktáv. Na obrázku 2.3 jsou znázorněny 2D šumové funkce, každá s jinou frekvencí a amplitudou a na obrázku 2.4 je součet těchto funkcí, neboli Perlinův šum. Jednotlivé šumové funkce při sčítání se nazývají oktávy.



Obrázek 2.3: Několik různých funkcí šumu (převzato z [1])



Obrázek 2.4: Výsledný Perlinův šum po sečtení několika vrstev šumu (převzato z [1])

Perlinův šum má tu vlastnost, že hodnoty sousedních pixelů na sebe navazují, díky tomu je možné vytvářet pěkně plynulé přechody.

Frekvenci každé oktávy můžeme vyjádřit jako

$$f = 2^i \tag{2.14}$$

a amplitudu jako

$$a = p^i \tag{2.15}$$

Poté je možné Perlinův šum v jedné dimenzi definovat takto:

$$Perlin(x) = \sum_{i=0}^{n} p^i \cdot Noise(a \cdot x) \tag{2.16}$$

A ve dvou dimenzích jako:

$$Perlin(x, y) = \sum_{i=0}^{n} p^i \cdot Noise(a \cdot x, a \cdot y) \tag{2.17}$$

Perlinův šum počítá hodnotu každého pixelu zvlášť, proto může být výpočetně velice náročný. Základní složitost je $O(2^n)$.

### 2.2.3 Kongruentní generátory

Asi nejznámějším a velice často používaným pseudonáhodným generátorem čísel, je lineární kongruentní generátor [5] popsaný vzorcem:

$$x_{i+1} = (ax_i + c) \bmod m \tag{2.18}$$

Kde $x_i$ je vstupní hodnota, $x_{i+1}$ je další člen řady, seed je pak $x_0$. Generátor generuje pseudonáhodná čísla v intervalu $< 0, m)$, počet možných hodnot v tomto rozsahu je omezen, proto se nejpozději po $m$ vygenerovaných číslech začne opakovat stejná posloupnout (perioda). Konstanty $a$, $c$, a $m$ je nutné vhodně zvolit. Maximální možné periody je možné dosáhnout za těchto tří předpokladů.

$c$ a $m$ jsou nesoudělná

$a - 1$ je dělitelné všemi prvočíselnými děliteli čísla $m$

$a - 1$ je násobkem čísla 4, pokud je $m$ násobkem čísla 4

Takovéto generátory jsou velice jednoduché na implementaci. Mezi jejich hlavní výhody patří rychlost a paměťová nenáročnost.

### 2.2.4 Částicový systém

Pojmem částice označujeme v počítačové grafice hmotné těleso, na které z vnějšku působí různé síly.

Pro zjednodušení výpočtů uvažujeme o částici jako o hmotném bodě. Jedná se tedy o těleso s nenulovou hmotností ale s nulovým objemem. Částicový systém je složen z velkého množství takových částic a každá částice se chová dle vlastních, určených pravidel. Vlastnosti částice se mění v závislosti na čase. Částice může mít svoji hmotnost, v daném čase je umístěna v určitém místě prostoru a pohybuje se určitou rychlostí.

V intru jsou částice využívány pro generování textury trávy. Každá taková částice má svoji rychlost $\vec{p}$ a pozici $\vec{v}$. Na částice mohou působit různé síly. Při výpočtu nové rychlosti a pozice částice používáme Newtonův druhý zákon dynamiky:

$$\vec{F} = m\vec{a} \tag{2.19}$$

Síla $\vec{F}$ se skládá z gravitační síly $\vec{F}_{\mathrm{g}}$

$$\vec{F_g} = m\vec{g} \tag{2.20}$$

kde $m$ je hmotnost tělesa a $\vec{g}$ je tíhové zrychlení. Tíhové zrychlení udává rychlost, kterou těleso nabude na povrchu kosmického tělesa, v našem případě Země, za jednu sekundu volného pádu.

# 2.3 Osvětlovací model

Osvětlovací model popisuje, jak povrch objektu odráží dopadající světlo. Osvětlovacích modelů existuje několik, ale nejznámnější a nejpoužívanější jsou empirické modely, tedy modely, které nejsou založeny na fyzice. Jedná se o Lambertův a Phongův osvětlovací model a těmto dvěma, jsou věnovány následující podkapitoly.

## 2.3.1 Lambertův osvětlovací model

Lambertův osvětlovací model [6] počítá s ideálním difúzním odrazem, který závisí na úhlu dopadu světelného paprsku. Difúzní složka udává intenzitu světla, které se rovnoměrně odráží do všech směrů.

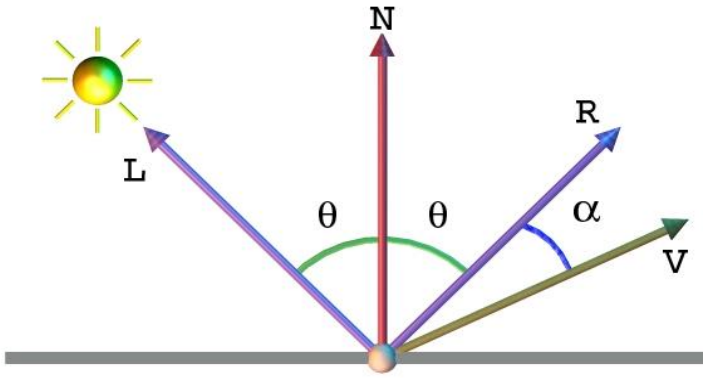
$$C_d = I_d K_d\left(\vec{N} \cdot \vec{L}\right) \tag{2.21}$$

$I_d$ představuje intenzitu světla, které dopadá na objekt z jednoho světelného zdroje. Intenzita je závislá na vzájemné poloze normály $\vec{N}$ a vektoru dopadu světla $\vec{L}$. Pokud je skalární součin těchto dvou vektorů menší nebo rovný nule, tak je povrch objektu odvrácen od světelného zdroje.

### 2.3.2 Phongův osvětlovací model

Phongův osvětlovací model [6] je rovněž empirický osvětlovací model pro výpočet odraženého světla. Tento model je jakousi nadstavbou Lambertova osvětlovacího modelu. Přestože není založen na fyzikálním modelu, dosahuje velice dobrých výsledků. Je výpočetně nenáročný, proto je často používán pro real-time grafiku.

Odraz světla v určitém bodě na povrchu materiálu se spočítá za pomoci čtyř vektorů. Vektor $\vec{L}$ je určen směrem dopadajícího světla, $\vec{N}$ je normála v daném bodě na povrchu, $\vec{V}$ je vektor k pozorovateli a $\vec{R}$ je odražený paprsek světla.




Obrázek 2.5: Model Phongova osvětlovacího modelu

Phongův model se skládá ze tří základních složek. Celková intenzita odrazu je dána jako součet jednotlivých složek, ambientní, spekulární a difúzní.

$$C = C_a + C_d + C_s \tag{2.22}$$

**Ambientní svělo**

Ambientní složku vyjádříme jako:

$$C_a = I_a K_a \tag{2.23}$$

$I_a$ zde představuje celkovou intenzitu ambientní složky světla, což je ta část světla, která dopadá na těleso ze všech stran rovnoměrně. Intenzita odraženého světla je nezávislá na vzájemné poloze zdroje světla, tělesa a pozorovatele. $K_a$ je odrazivý koeficient materiálu objektu.

**Difúzní světlo**

Difúzní složka Phongova osvětlovacího modelu je shodná s difúzní složkou v Lambertově osvětlovacím modelu, platí pro ni tedy stejný vztah.

**Spekulární světlo**

Spekulární složka udává intenzitu té části světla, které dopadá na těleso z jednoho světelného zdroje a odráží se od tělesa také v jednom směru dle zákona odrazu.

$$C_s = I_s K_s \left(\vec{V} \cdot \vec{R}\right)^n \tag{2.24}$$

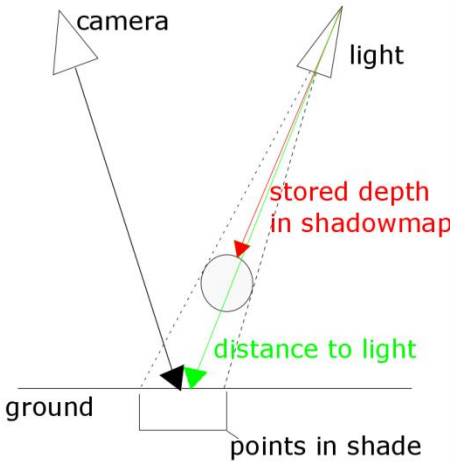
Kde $\vec{R}$ je odražený paprsek světla, který spočítáme jako:

$$\vec{R} = 2\left(\vec{N} \cdot \vec{L}\right)\vec{N} - \vec{L} \tag{2.25}$$

a $n$ je Phongův exponent určující míru lesklosti.

# 2.4 Shadow Mapping

Shadow mapping je technika pro tvorbu stínu ve 3D počítačové grafice. S touto technikou přišel v roce 1978 Lance Williams [7]. Jedná se o dvou průchodovou metodu. Při prvním průchodu se vykreslí scéna z pozice světla a do hloubkové mapy se zapíše vzdálenost jednotlivých fragmentů od světla. Při druhém průchodu se scéna kreslí z pohledu kamery. Pomocí lineární transformace se pozice objektů promítne do pozice objektů z pohledu světla, určí se její umístění v hloubkové mapě a z ní se vypočítá vzdálenost. Pokud má fragment menší vzdálenost, než je vzdálenost zapsaná v hloubkové mapě, pak je osvětlený. V případě, že je vzdálenost větší, tak se mezi fragmentem a světelným zdrojem nachází jiný objekt, fragment tedy leží ve stínu. Lépe je to patrné z obrázku 2.6.




Obrázek 2.6: Shadow Mapping

V případě, že tedy mluvíme o shadow mapě, jedná se o paměť hloubky (z-buffer) zkonstruovanou z pohledu světelného zdroje. Taková metoda sama o sobě není dostačující, neboť stíny, které produkuje, jsou velmi ostré. Ostrost stínu navíc záleží na rozlišení stínové mapy a na vzdálenosti světla od objektů. Na obraze se tato skutečnost projevuje jako kostrbatý stín.
Byly vytvořeny tedy další techniky, které se snaží tomuto jevu zabránit a výsledný stín zjemnit. Jednou z těchto metod je Percentage Closer Filtering (PCF) [8]. Technika PCF je založena na čtení několika vzorků z hloubkové mapy, pro každý vzorek je následně proveden test, zda leží ve stínu, poté je výsledek zprůměrován a výsledná hodnota představuje procentuální zastínení fragmentu.

$$P(x) = \frac{1}{n}\sum_{i=1}^{n} f(x_i) \tag{2.26}$$

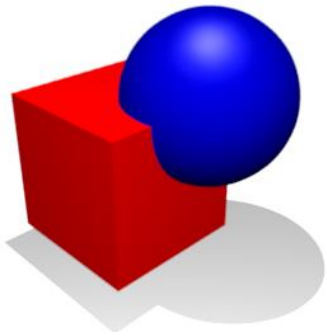
Nejčastěji se provádí průměr přes všechny hodnoty vzorkovaného jádra (2.26). Pokud použijeme jádro o velikosti 3x3 tak musíme zprůměrovat devět hodnot. Pokud by i takto vyhlazené výsledky stínové mapy pro nás nebyly postačující, tak je možné rozšířit obyčejné PCF s průměrovacím jádrem o nějaké vhodnější, jako je například poissonovo filtrovací jádro.

## 2.5 Constructive Solid Geometry

Constructive solid geometry (CSG) [9] je metoda velice často používaná pro modelování pevných objektů. CSG nám dovoluje vytvářet velice složité povrchy, případně objekty za pomoci Booleanovských operatorů pro kombinování objektů. Často tedy CSG reprezentuje vizuálně velmi komplikovaný model, který je ve skutečnosti pouhou kombinací velmi jednoduchých objektů. Pro lepší představu velmi dobře poslouží obrázek 2.10.

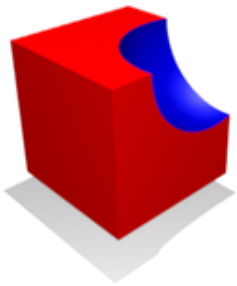
Základem v CSG je primitivum, ve 3D to může být například krychle, válec nebo koule, vždy záleží na konkrétní implementaci CSG. Primitivum může být typicky popsáno procedurou, která může jako parametry přijímat například u koule, koordináty středového bodu a poloměr. Nad těmito primitivy je možné provádět booleanovské operace sjednocení, průniku a rozdílu.

Operace sjednocení představuje výsledný objekt jako takový objekt, ve kterém se nachází všechny body těles A i B. $F = A \cup B$. Kde F je výsledný objekt, A je v našem případě krychle a B je koule.



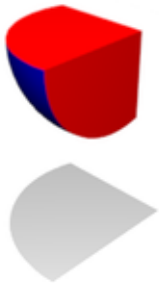
Obrázek 2.7: booleanovská operace sjednocení (Zdroj: http://en.wikipedia.org/wiki/Constructive_solid_geometry)

Operace rozdílu, kterou zapíšeme jako $F = A - B$, reprezentuje všechny body tělesa A, s vyjímkou těch, které jsou shodné s body z tělesa B.



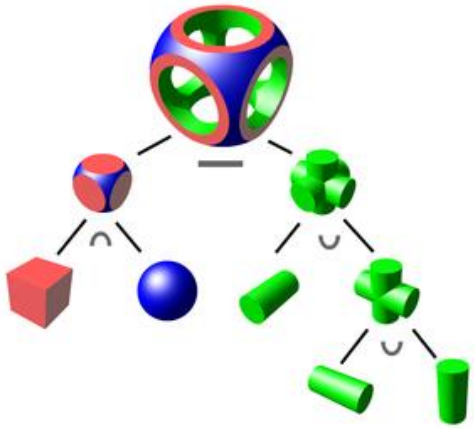
Obrázek 2.8: booleanovská operace rozdílu (Zdroj: http://en.wikipedia.org/wiki/Constructive_solid_geometry)

Operace průniku, zapsaná jako $F = A \cap B$, reprezentuje právě ty body, které mají tělesa A a B společné.



Obrázek 2.9: booleanovská operace průniku (Zdroj: http://en.wikipedia.org/wiki/Constructive_solid_geometry)

Pomocí těchto tří základních operací, je možné vytvářet velice komplikované objekty. CSG má velkou škálu praktických využití, je velmi rozšířené mezi CAD systémy, ale svoje využití si může najít například i v procedurálně generované grafice.



Obrázek 2.10: binární strom reprezentující CSG objekty (Zdroj: http://en.wikipedia.org/wiki/Constructive_solid_geometry)

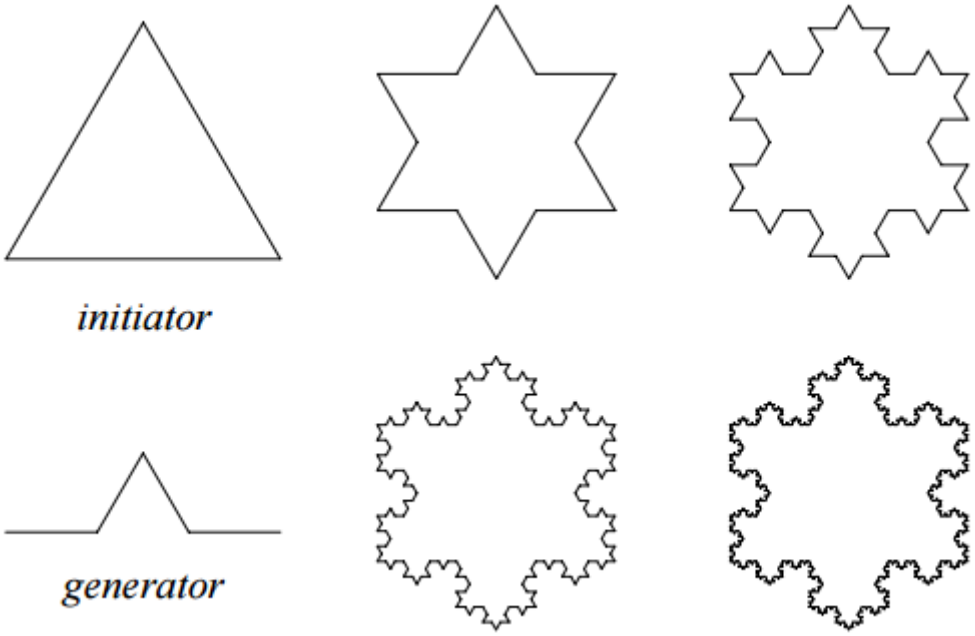
## 2.6 Billboarding

Billboarding je technika, která upravuje orientaci objektu tak, že je neustále natočen čelem ke kameře. Billboarding může být použit například k tomu, abychom nahradili složitou geometrii, jednou texturou nanesou například na dva trojuhelníky tvořící čtverec a tím ušetřili výpočetní výkon. Tato technika tedy zajišťuje, že textura je neustále natočena směrem ke kameře a uživatel si díky tomu neuvědomí, že například strom, na který se dívá, je jen plochá textura namapovaná na čtverec. Samozřejmě pokud nad takovýmto stromem přeletí, tak dojem iluze je ztracen. Billboarding může být tedy využit k ušetření počtu polygonů, ale obecně se dá použít ve všech situacích, kdy je vyžadováno, aby byl daný objekt natočen v jistém směru, například jako fotbalista k míči.

## 2.7 L-systémy

L-systémy vytvořil v šedesátých letech maďarský biolog Aristid Lindenmayer [10]. Nejprve sloužily pro popis jednoduchých buněčných struktur, ale později se ukázalo, že jejich možnosti jsou mnohem větší, a že jdou použít i pro mnohem složitější úlohy.

L-systém je druh formální gramatiky, jejímž základním konceptem je přepisovatelnost. Technika přepisování se tedy používá pro definování složitých objektů, za použití množiny přepisovacích (produkčních) pravidel. Asi nejznámnějším příkladem L-systému je Kochova sněhová vločka na obrázku 2.11. Základem pro vygenerování vločky je iniciátor, na který je postupně aplikované produkční pravidlo, a tak vzniká finální objekt od první, až do čtvrté iterace.

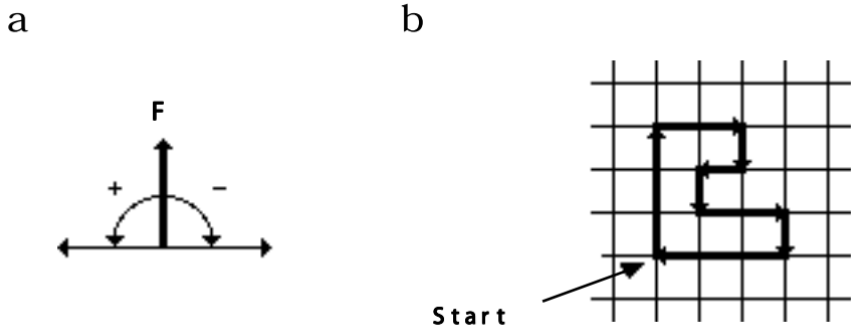



Obrázek 2.11 Kochova sněhová vločka (převzato z [10])

L-systém můžeme tedy definovat jako trojici $G = \{V, \omega, P\}$, kde $V$ je konečná, neprázdná množina symbolů. $P$ je konečná množina produkčních pravidel a $\omega \epsilon V$ je počáteční řetězec (axiom).

### 2.7.1 Želví grafika

Základem želví grafiky je virtuální želva. Stav želvy je definován trojicí $(x, y, \alpha)$, kde $(x, y)$ představují kartézské souřadnice a $\alpha$ je úhel, jímž je definováno natočení želvy.




Obrázek 2.12: (a) Želví reprezentace symbolů F,+,-. (b) Interpretace řetězce (převzato z [10])

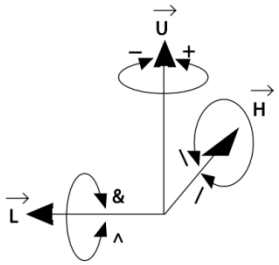
Příkazy ve formě znaků, které želva interpretuje:

F Želva provede posunutí v před, dle aktuálního natočení a pozice, vykreslí tedy úsečku.

\+ Želva provede rotaci do prava o úhel $\boldsymbol{\delta}$.

\- Želva provede rotaci do leva o úhel $\boldsymbol{\delta}$.

Pokud želva takto interpretuje řetězec znaků: FFF-FF-F-F+F+FF-F-FFF, tak vykreslí obrázek 2.12b. Nyní tedy víme, jak pomocí želví grafiky modelovat ve 2D, nicméně pro nás je mnohem důležitější modelování ve 3D prostoru. Proto je nutné rozšířit množinu znaků o znaky, které provádějí operace ve 3D. Základem je reprezentace aktuální orientace želvy v prostoru za pomoci tří vektorů $\vec{U}, \vec{L}, \vec{H}$, kde $\vec{U}$ značí vektor up, $\vec{L}$ značí left a $\vec{H}$ představuje heading.




Obrázek 2.13: Orientace želvy v prostoru (převzato z [10])

Rotaci želvy, je pak možné vyjádřit vztahem

$$\left[\overrightarrow{H'}\,\overrightarrow{L'}\,\overrightarrow{U'}\right] = \left[\vec{H}\,\vec{L}\,\vec{U}\right] R \tag{2.27}$$

$\vec{H}\,\vec{L}\,\vec{U}$ představují aktuální souřadný systém želvy a $\overrightarrow{H'}\,\overrightarrow{L'}\,\overrightarrow{U'}$ představují nově vypočtený souřadný systém po aplikování rotací a $R$ představuje výsledek složení transformačních matic $R_U R_H R_L$.

$$R_U(\alpha) = \begin{pmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.28}$$

$$R_L(\alpha) = \begin{pmatrix} \cos(\alpha) & 0 & -\sin(\alpha) \\ 0 & 1 & 0 \\ \sin(\alpha) & 0 & \cos(\alpha) \end{pmatrix} \tag{2.29}$$

$$R_H(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{pmatrix} \tag{2.30}$$

Následující symboly kontrolují pohyb želvy ve 3D prostoru na obrázu 2.13.

***F*** Želva provede posun v před dle aktuálního natočení a pozice

+ Želva provede kladnou rotaci okolo osy $\vec{U}$

− Želva provede zápornou rotaci okolo osy $\vec{U}$

& Želva provede zápornou rotaci okolo osy $\vec{H}$

^ Želva provede kladnou rotaci okolo osy $\vec{H}$

\ Želva provede zápornou rotaci okolo osy $\vec{L}$

/ Želva provede kladnou rotaci okolo osy $\vec{L}$

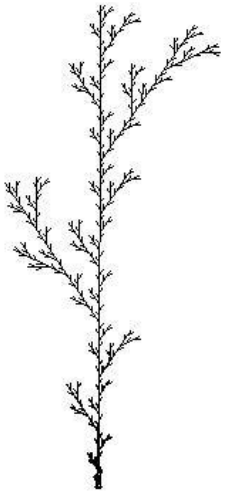
### 2.7.2 Závorkové L-systémy

Pomocí klasických L-systému jsme schopni popsat lineární objekty. Naším cílem je popisovat stromové, větvící se struktury. Z tohoto důvodu je nutné přidat další symboly do gramatiky, které budou odlišovat větev a želva je bude interpretovat takto:

[ Uloží aktuální stav želvy na zásobník. Na zásobník tedy uloží natočení a pozici želvy a může dále uložit další atributy, jako je barva případně šířka větve.

] Vyjme z vrcholu zásobníku atributy popisující stav želvy.

S takto rozšířenou množinou znaků, jsme již schopni kreslit složité, větvící se struktury, jako je například strom, křoví nebo květina.



Obrázek 2.14: Stromová struktura ve 2D

Příklad takového závorkového L-systému je na obrázku 2.14. Gramatika popisující takovou strukturu je následující:

$$V = \{F, +, -, [, ]\}$$

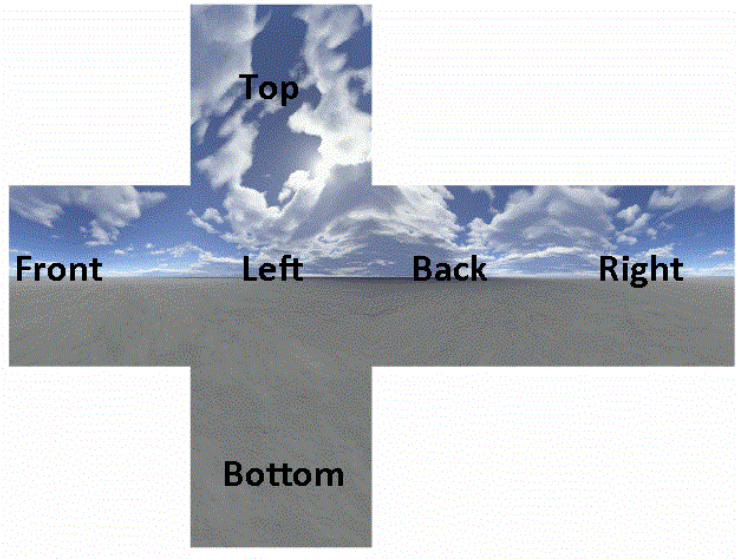
$$\omega = F$$

$$P = \{F \rightarrow F[+F]F[-F]F$$

Úhel o který se želva ve 2D natáčí je $\delta = 25.7°$ a počet iterací $n = 5$.

## 2.8 Skybox

Skybox představuje reprezentaci okolní scény. Tato metoda bývá velice často používána v počítačových hrách, díky tomu nám připadá scéna větší, než ve skutečnosti je. Skybox je v podstatě krychle, která obklopuje celou scénu a na jejíž strany je namapována textura okolí. Textura může být libovolná a může nám vytvářet okolí hor, města, vesmíru, mraků a jiné. Okolní objekty tedy nejsou reprezentovány polygony, ale pouze obrázkem, který je zobrazován na stěnách krychle. Je důležité, aby na sebe textura plynule navazovala, poté se k namapování textury na krychli použije cube mappingu. Kamera je následně umístěna do středu krychle a její pozice se vůči skyboxu nemění.




Obrázek 2.15:Ukázka skyboxu (převzato z [11])

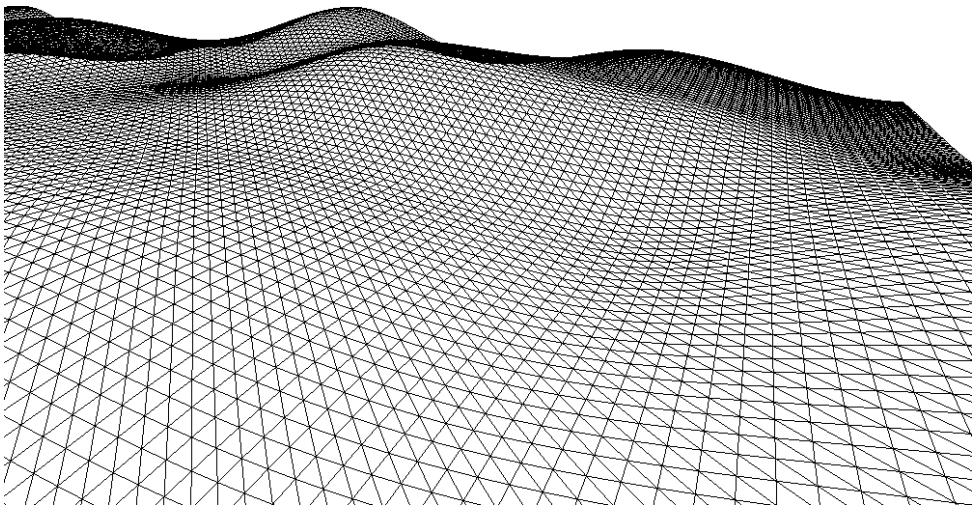
V dnešních aplikacích se již používají i skydomy. Nejedná se tedy už o krychli, ale o polokouli. Se skydomem se dá dosáhnout reálnější oblohy, na úkor počtu vykreslených polygonů, jelikož geometrie koule je mnohem složitější, než geometrie obyčejné krychle.

# 3 Implementace

Tato kapitola je zaměřena na vlastní implementaci intra. Půjde převážně o implementaci jednotlivých metod, které jsou popsány v teoretické části této práce.

## 3.1 Generování terénu

Pro vytvoření terénu je použita výšková mapa. Nejprve je tedy nutné vytvořit mřížku bodů, která bude reprezentovat pozice vrcholů jednotlivých primitiv, konkrétně trojúhelníků. Souřadnice jsou generovány v cyklu. Souřadnice x a z, jsou následně vycentrovány tak, že střed výškové mapy má souřadnice $(0, height, 0)$. Souřadnice y představující výšku je dopočítávána za pomoci 2D Perlinova šumu (2.17). Šum je nastaven tak, aby vygeneroval mírný kopcovitý terén, který by reprezentoval golfové hřiště. Geometrie terénu je tedy generována na CPU. Pro generování mírného kopcovitého terénu, byla použita kosinová interpolace (2.8).



Obrázek 3.1: Výšková mapa vytvořena pomocí 2D Perlinova šumu

Spolu s vrcholy jsou spočítány jednotlivé indexy trojúhelníku, aby poté bylo možné správně vykreslit celý terén jediným příkazem funkce glDrawElements(). Dále je pak nezbytné spočítat normály jednotlivých primitiv, které jsou následně použity pro výpočet Phongova osvětlovacího modelu. Do shaderů se tedy společně s jednotlivými vrcholy zasílají i normály. Při výpočtu normály pro daný vrchol nestačí použít pouze normálu daného trojuhelníku, ale je důležité použít normály všech sousedních trojúhelníků.

## 3.2 Generování textur

Textura slouží pro detailní popis povrchu tělesa. Z textury můžeme nahrávat data do shaderů, kde je můžeme namapovat na výsledný objekt, anebo ze shaderů generovat data do textury místo na obrazovku. Textury se skládají z jednotlivých texelů, které se při kreslení mapují na jednotlivé pixely. Jelikož při implementaci intra je pro nás zachování minimální velikosti největší prioritou, budeme používat procedurální textury. Procedurální textura je textura, která se vytváří až za běhu programu. Není nutné, mít tedy texturu někde předem připravenou, jelikož si ji můžeme vytvořit, až za běhu aplikace.

### 3.2.1 Textura trávy

Textura trávy je generována po spuštění aplikace sadou shader programů. Tráva pokrývá celou plochu golfového hřiště a byla generována pomocí částicového systému. Generování výsledného stébla trávy se skládá ze tří základních částí.

Ve vertex shaderu byla za pomoci Perlinova šumu vypočtena startovní pozice částice $P(x, y)$ a úhel, kterým bude následně vystřelena. Jelikož Perlinův šum vrací hodnoty v rozsahu $\langle -1,1 \rangle$ bylo potřeba si tyto hodnoty přemapovat do rozsahu $\langle 0,1 \rangle$. Přemapování hodnoty provedeme tak, že původní hodnotu vynásobíme hodnotou 0.5 a následně k ní přičteme 0.5.

Nejdůležitější částí při generování trávy je geometry shader, kde stéblo získává svoji výslednou podobu. Na začátku tedy máme počáteční bod a směr, kterým částici vystřelíme. Trajektorii částice si vyjádříme pomocí diferenciální rovnice, kterou následně vypočítáme za pomoci Eulerovy metody.

K vektoru gravitační síly ještě přičítáme vektor náhodné síly, která nám zajistí různé úhly zakřivení jednotlivých stébel. Dále bylo velice důležité ošetřit krajové situace, kdyby docházelo k vystřelení částice z textury. Pokud by došlo k vystřelení částice mimo hranice textury do leva, tak budeme pokračovat s generováním té samé částice zprava. Toto nám zajistí, že nebudou pozorovatelné přechody mezi jednotlivými okraji textury.

Poslední částí je obarvení textury ve fragment shaderu, kde se každému stéblu trávy určuje jeho výsledná barva. Každé stéblo má jinou barvu, od světle zelené po tmavě hnědou barvu. Takto nagenerovaný obrázek nejprve vykreslíme do frame buffer objectu a uložíme do textury. Tímto máme vytvořenou výslednou texturu trávy.

Výsledná textura trávy je rozprostřena po celém terénu. Aby nedocházelo k aliasingu, bylo třeba vygenerovat mipmapy pro texturu trávy. Nejprve musíme nastavit potřebné parametry textury:

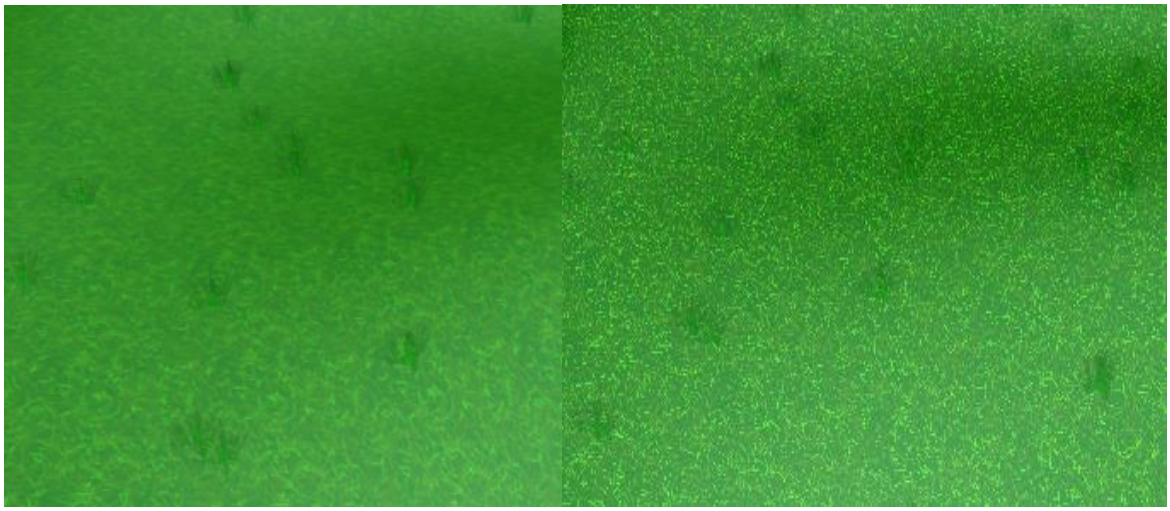
```
glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,
GL_LINEAR_MIPMAP_LINEAR);
```

```
glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,
GL_LINEAR_MIPMAP_LINEAR);
```

Poté již můžeme vygenerovat mipmapu:

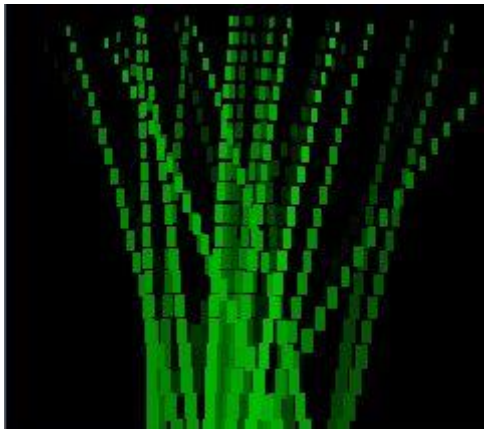
```
glGenerateMipmap(GL_TEXTURE_2D);
```

Výslednou texturu namapovanou na terén, můžeme vidět na obrázku 3.2, kde v pravo je textura bez použití mipmap, u které dochází k aliasingu a na levo je textura se zapnutými mipmapami. Aliasing se projevuje nepěkným zrněním.



Obrázek 3.2: Mipmapa textury na terénu zabraňující aliasingu

Textura trávy pro billboardy se vytvářela velice podobným způsobem s tím rozdílem, že ve vertex shaderu se generovala pro počáteční pozici pouze souřadnice x a úhel při použití Perlinova šumu, souřadnice y byla konstantní rovna nula, jelikož jsme nechtěli, aby se nám začala generovat jednotlivá stébla trávy někde ve vzduchu (viz. obrázek 3.3).



Obrázek 3.3: Textura pro billboardy s trávou

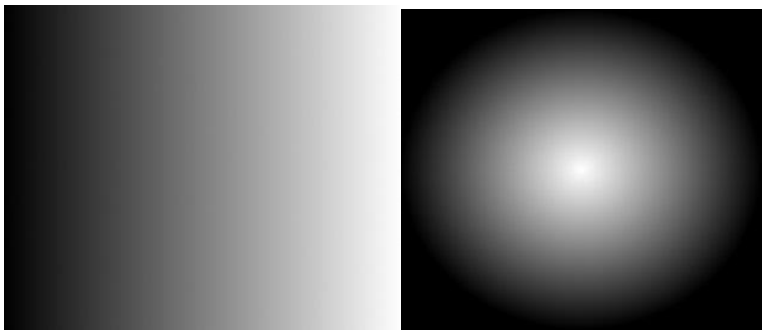
Na obrázku 3.3 je velmi dobře patrné, jak částicový systém funguje. Na pozici $y = 0, \quad x \epsilon < 0{,}1 >$ je vždy umístěn emitor, který vystřeluje částici do prostoru a po její trajektorii jsou vykreslovány malé obdélníky, které reprezentují finální stéblo trávy. Tyto obdélníky jsou generovány v geometry shaderu následujícím způsobem.

```
for(int k = 0; k < 4; ++k)
{
      gl_Position = vec4(Position + Size * (vec2(-1+2*(k%2), -1+2*(k/2)),
      0, 1);
      EmitVertex();
}
EndPrimitive();
```

Position je pozice ve středu obdélníku, Size je jeho velikost a poté v cyklu postupně určíme 4 vrcholy obdélníku.

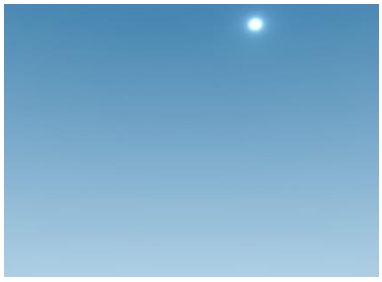
# 3.3 Skybox

Skybox se skládá ze tří základních prvků a to jsou mraky, slunce a obloha. Jelikož je skybox animovaný, je naprogramován za pomoci shaderů a počítán na GPU. Obloha a slunce jsou vytvořeny pomocí gradientu. Gradient je barevný přechod, který je tvořen plynulým přechodem mezi jednotlivými barvami podél daného vektoru. Barevný přechod mezi dvěma sousedícími barvami je počítán jako lineární interpolace. Využíváme dva druhy gradientů lineární a radiální gradient 3.4.



Obrázek 3.4: lineární gradient a radiální gradient

Barevný přechod je použit pro vytvoření oblohy a slunce. Skládá se z barvy slunce, barvy horizontu a barvy oblohy. Barevný přechod se počítá jako lineární interpolace mezi sousedními barvami. Vytvoření gradientu oblohy probíhá ve dvou krocích, nejprve se vytvoří barevný přechod oblohy s horinzontem a následně se připočte barva slunce. V OpenGL používáme pro míchání dvou barev funkci $mix(color1, color2, y)$, která smíchá barvy podle $y$.

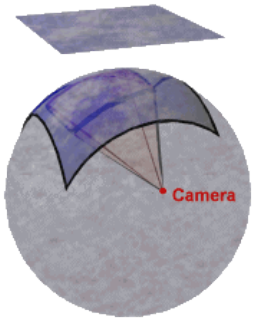


Obrázek 3.5: Barevný přechod horizontu

Nejdůležitější částí skyboxu jsou mračna, která se pohybují po obloze. Existují různé techniky vytváření mračen. Pro naši aplikaci nepotřebujeme 3D mračna, a tak si vystačíme s 2D mračny. Ušetříme tím výpočetní výkon a při zvolení vhodných parametrů lze dosáhnout pěkných výsledků. Mraky jsou reprezentovány nekonečnou rovinou s Perlinovým šumem znázorněnou na obrázku 3.6. Jelikož používáme 3D šum, tak čas představuje třetí rozměr. Hodnota Perlinova šumu pixelu se zjistí jako průsečík $P(x, y)$ směrového normalizovaného vektoru $v$ s rovinou v daném čase. Souřadnice $coords(x, y, time)$ se poté použijí jako vstupní hodnota pro funkci šumu. Přičtením času k tomuto vektoru ovlivníme také pohyb mraků. Výpočet průsečíku určeného směrovým vektorem $\vec{v}$ s rovinou ve výšce $h$ je určen vztahem (3.1).

$$vec2\ P\ =\ vec2\left(v.x\frac{h}{v.y}, v.z\frac{h}{v.y}\right) \tag{3.1}$$

$$vec3\ coords = vec3(P, time) \tag{3.2}$$




Obrázek 3.6: Rovina s perlinovým šumem (převzato z [12])

Následně se tyto souřadnice (3.2) použijí pro výpočet 3D Perlinova šumu (3.3). Vypočtená hodnota Perlinova šumu je poté použita pro míchání barvy gradientu oblohy s bílou barvou, která reprezentuje mraky (3.4). Kde $colgrad$ je již vytvořený gradient oblohy, $vec4(1)$ představuje bílou barvu mraků. Tyto dvě barvy smícháme lineární interpolací podle parametru $x$.

$$x = 3DNoise(coords, frequency, persistance, amplitude, octaves) \quad (3.3)$$

$$col = mix(colgrad, vec4(1), x) \quad (3.4)$$

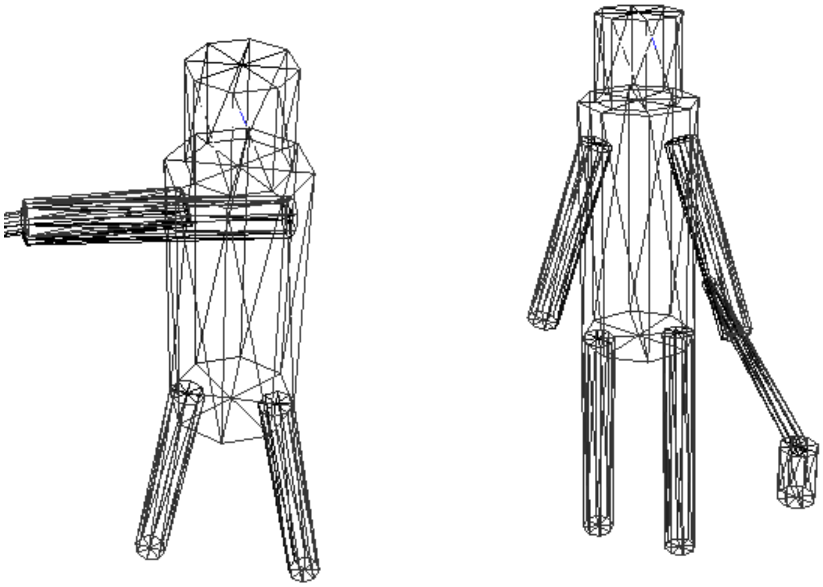
Obrázek 3.7: Oblačná a polojasná obloha

## 3.4 Postavička a míček

Mezi dynamické prvky intra patří robot hrající golf. V této podkapitole se tedy budeme věnovat geometrii robota, tvorbě jeho obličeje a také fyzikálnímu modelu sloužícímu pro popis pohybu míčku po terénu.

### 3.4.1 Geometrie robota

Robot se skládá ze šesti osmibokých hranolů, každý takový hranol představuje jednu část jeho těla jako je hlava, ruce, tělo a nohy (viz. obrázek 3.8).



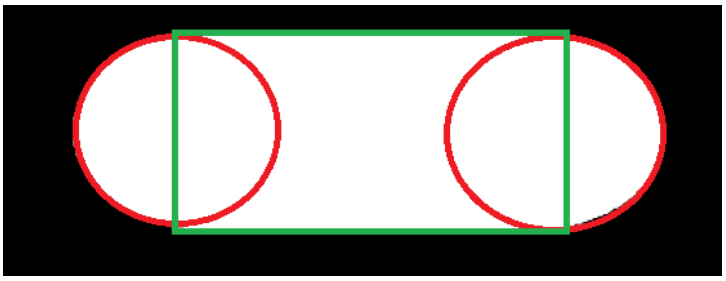
Obrázek 3.8: V levo robot při odpalu, v pravo při chůzi

Dále je nutné pro každou část jeho těla sestavit určitou transformační matici $R$, která buď slouží k vytvoření animace robotova pohybu, nebo se použije k natočení robota do určitého směru, například tak, aby byl robot čelem k míčku, který se chystá odpálit. Pokud tedy chceme, aby robot švihnul golfovou holí a tím simuloval odpal, je nutné všechny vrcholy, které tvoří geometrii robotových rukou, násobit transformační maticí a to tak, že postupně s časem měníme úhel, o který chceme rotovat. Geometrie robota je počítána na CPU.

### 3.4.2 Obličej robota

Robotův obličej je vykreslován v reálném čase při běhu programu. Naprogramován byl pomocí vertex a fragment shaderu, kdy ve vertex shaderu se provádí mapování, kam na robotovo tělo se má obličej vykreslit a ve fragment shaderu se provádí již samotné vykreslení obličeje. Pro zjednodušení tvorby obličeje bylo využito metody konstruktivní solidní geometrie, popsané v teoretické části 2.5. Nejprve bylo nutné definovat základní operace sjednocení (union), průniku (intersect) a rozdílu (substract) v jazyce GLSL. Dále bylo nutné definovat základní primitiva, s jejichž pomocí budou vytvářeny složitější konstrukce, pro naše potřeby byl definován čtyřúhelník a kruh. S využitím dvou základních primitiv byl dále vytvořen tvar oválu (viz obrázek 3.9).

$$oval = Union(Union(circle, circle), rectangle) \tag{3.5}$$



Obrázek 3.9: Použití CSG ve fragment shaderu

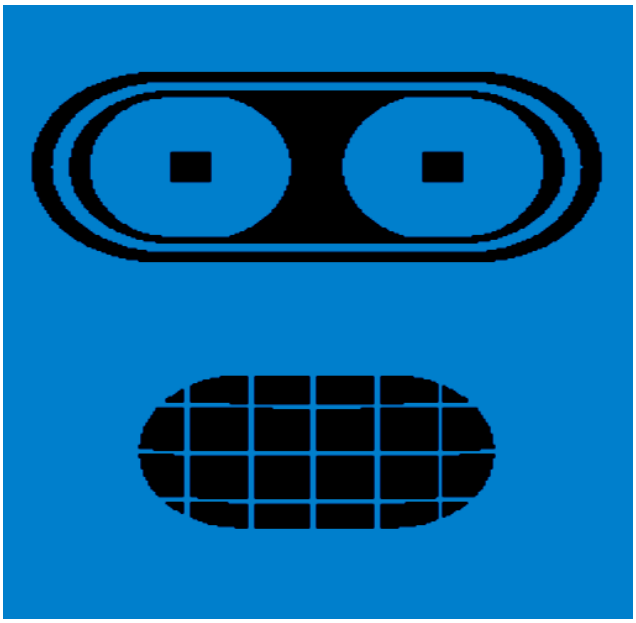
Postupným skládáním byl vytvořen celý obličej. Pro zjednodušení popisu rozdělíme obličej na dvě části, jedna část kde se nachází oči a druhá kde jsou ústa. Následuje zjednodušený popis, jak pomocí CSG vygenerovat oční část robotova obličeje.

$$eyeRects = Union(eyeLeftRectangle, eyeRightRectangle) \tag{3.6}$$

$$pasek = Substract(ovalL, ovalS) \tag{3.7}$$

$$pruh = Substract(pasekL, pasekS) \tag{3.8}$$

$$\begin{aligned} eyeSet = Union(&Substract(Union(Substract(background, oval),\\ &Substract(Union(eyeRightCircle, eyeLeftCircle),\\ &eyeRects)), pasek), pruh) \end{aligned} \tag{3.9}$$



Obrázek 3.10: Obličej robota vygenerovaný ve fragment shaderu

### 3.4.3 Fyzika míčku

Pro míček je implementován základní fyzikální model. Míček má určenou svoji rychlost $\vec{v}$, hmotnost $mass$ a svůj poloměr $radius$. Fyzika míčku je počítána na CPU, tedy do vertex shaderu se přidá jeden další uniform, který bude určovat pozici míčku. Ve vertex shaderu se pak už nová pozice míčku spočítá jako $gl\_Position = MVP * vec4(vertex + ballposition, 1)$. gl_Position je výsledná pozice vrcholu, $MVP$ je již složená model-view-projekční matice, $vertex$ je vrchol primitiva a $ballposition$ je aktuální pozice míčku. Na míček působí dvě síly a to gravitační síla $\vec{F}_g$ a síla větru $\vec{F}_w$ výslednici těchto sil vyjádříme takto:

$$\vec{F}_v = \vec{F}_g + \vec{F}_w \tag{3.10}$$

Zrychlení míčku poté spočítáme podle druhého Newtonova pohybového zákonu.

$$\vec{a} = \vec{F}_v / mass \tag{3.11}$$

Nyní můžeme vyjádřit rychlost a pozici těmito dvěmi rovnicemi:

$$v(t) = v(0) + \int_0^t a(t)dt \tag{3.12}$$

$$p(t) = p(0) + \int_0^t v(t)dt \tag{3.13}$$

Tímto byl vyřešen pohyb míčku v prostoru. Jelikož se míček pohybuje po terénu, kde se různě odráží, bylo nutné implementovat kolizní model míčku, který by určoval jakým směrem a jak rychle se bude míček odrážet. Pro určení zda došlo ke kolizi, či nikoliv, potřebujeme znát aktuální pozici míčku a výšku terénu v daném bodě. Pokud dojde ke kolizi, je nutné vypočítat nový vektor rychlosti $\vec{v}$. Ten vyjádříme následujícím vztahem:

$$\vec{v} = reflect(\vec{v}, \vec{n})(1 - collisionLoss) \tag{3.14}$$

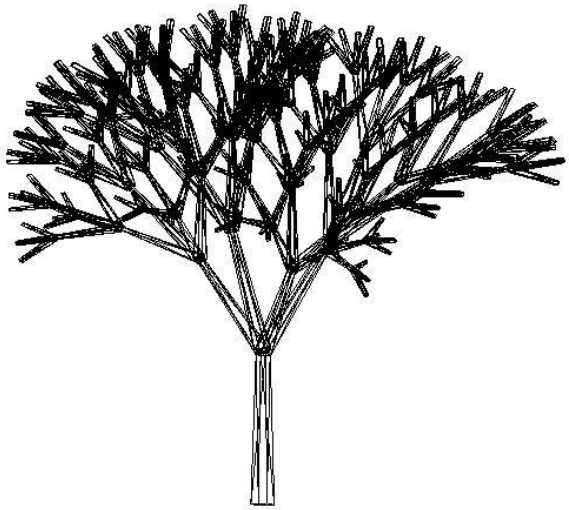
Kde $collisionLoss$ představuje ztrátu rychlosti míčku při vícenásobných odrazech, $reflect$ je funkce, která počítá vektor odrazu pro vektor $\vec{v}$, který se odráží v místě s normálou $\vec{n}$, dle vztahu (2.25).

## 3.5 Generování vegetace

Následující kapitola se zabývá generováním stromů a trsů trávy, které jsou rozmístěny po celé scéně.

### 3.5.1 Stromy

Stromy jsou generovány s využitím závorkovaných L-systémů popsaných v kapitole 2.7.2 a následně interpretovány s použitím želví grafiky ve 3D (viz. kapitola 2.7.1). Nicméně pro vykreslení stromu by bylo nedostatečné vykreslit pouhé úsečky, místo toho jsou vykresleny větve, každá větev je reprezentována komolým čtyřbokým jehlanem. Při interpretaci řetězce představujícího strom, se pro každou větev ukládá její pozice, velikost, aktuální natočení a úroveň zanoření na zásobník. Podle těchto parametrů se následně vypočítá geometrie větve.



Obrázek 3.11: Geometrie stromu

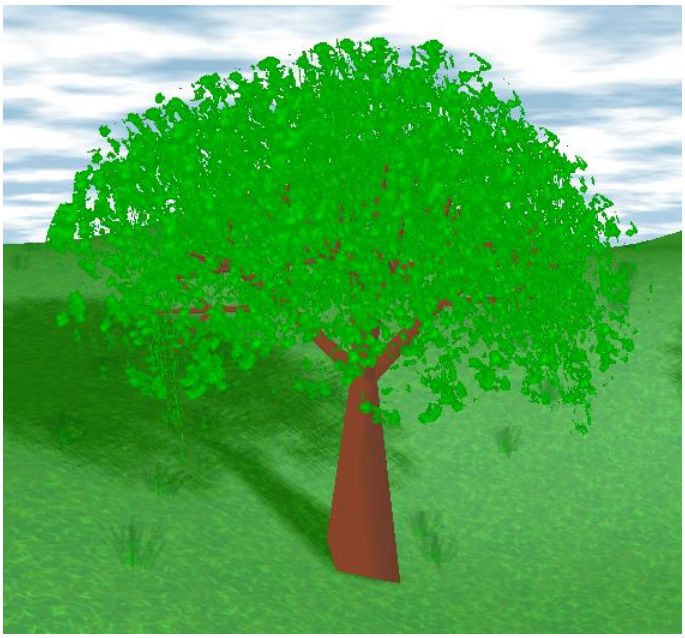
Na obrázku 3.11 je vygenerován strom s využitím závorkového systému definovaného takto:

$$V = \{F, +, -, <, >, \wedge, \&, [, ]\}$$

$$\omega = F$$

$$P = \{F \rightarrow F[+F][-F][< F][> F]\}$$

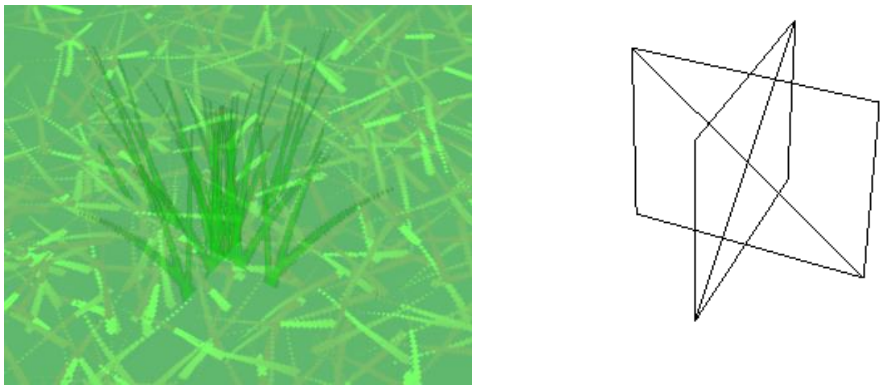
Následně je pro každou koncovou větev uložena pozice, na kterou bude nagenerována textura listí na náhodně natočený čtverec. Textura listí je vygenerována ve fragment shaderu s využitím Perlinova šumu a prahování, kdy od určité hodnoty jsou fragmenty zahazovány s použitím funkce discard. Tímto je docíleno vzniknutí ostrůvků, které připomínají jednotlivé listy (viz. obrázek 3.12).



Obrázek 3.12: Finální vygenerovaný strom

### 3.5.2 Tráva

Pro generování trsů trávy je použita textura popsaná v kapitole 3.2.1, kdy textura je nanášena na dva zkřížené čtverce, které představují trs trávy.



Obrázek 3.13: Trs trávy

Před samotným vykreslením trsů trávy je třeba nastavit blending funkci.

```
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA).
```

## 3.6 Vytvoření scény

Tvorba finální scény se skládá ze dvou kroků, nejprve jsou vykresleny pouze stromy, z této scény je následně vytvořena shadow mapa. Následně se vykreslí již celá scéna se skyboxem, otexturovaným terénem, na kterém jsou již patrné stíny, robotem, míčkem a praporkem s vlající vlajkou.

Obrázek 3.14: Výsledná scéna golfového hřiště

## 3.7 Použité knihovny

Projekt byl vytvořen ve vývojovém prostředí Microsoft Visual Studio 2012 a byl napsán v jazyce C++. Zdrojové kódy projektu jsou rozděleny do tří hlavních částí a to: zdrojové soubory, shadery a hlavičkové soubory. Pro práci s oknem bylo použito WinAPI. Dále byla použita knihovna GLM, která přidává datové typy pro vektory a matice, které jsou podobné datovým typům v GLSL, dále také usnadňuje manipulaci s těmito daty a přidává základní funkce pro manipulaci s těmito datovými typy. Není použita žádná knihovna, která by inicializovala OpenGL kontext verze 3.0 a vyšší. Proto bylo nutné si všechny nové funkce OpenGL vytáhnout z hlavičkového souboru glext.h. To je realizováno za pomoci funkce *wglGetProcAddress*, která vrací adresu OpenGL extension funkce.

```
glGenBuffers = (PFNGLGENBUFFERSPROC)wglGetProcAddress("glGenBuffers");
```

Takto bychom například získali funkci glGenBuffers. Díky tomuto přístupu nemusíme do projektu přidávat další rozšiřující knihovny jako je například glew. Zároveň tímto způsobem ušetříme na místě, jelikož nemusíme přikládat celou knihovnu, ale vybereme si jen funkce, které opravdu použijeme.

# 4 Metody pro omezení velikosti aplikace

Jak již bylo dříve zmíněno, při tvorbě grafického intra, je velký důraz kladen na zachování minimální velikosti apliace. Jsou různé možnosti jak zmenšit výslednou aplikaci. Například můžeme nastavit překladač tak, aby byl překlad optimalizován na minimální velikost a aby překladač odstranil nepoužívaná data. Mezi další způsoby, jak zmenšit velikost je zkomprimovat výslednou aplikaci za pomoci speciálních Exe packerů. Mezi další možnosti, jak uspořit místo, je používat vhodný programátorský styl a konstrukce. Je tedy vhodné psát kód pokud možno co nejobecněji, abychom použité konstrukce mohli využít někde znovu, za minimálních změn. Mezi další dobré programovací zvyklosti je používání rekurze a cyklu pokud je to možné, aby se velice podobná konstrukce nemusela rozepisovat vícekrát pod sebou. Samozřejmostí je generování grafického obsahu, které je popsáno v předcházejících kapitolách.

## 4.1 Nastavení překladače

Pro nastavení překladače a linkeru se používají přepínače, pomocí kterých se zapínají různé optimalizace pro rychlost, velikost a jiné. Parametry Microsoft Visual C++ překladače, které jsou vhodné pro minimalizaci velikosti výsledné apliakce:

- /Ox Full optimization
- /Os Tento přepínač optimalizuje velikost výsledného exe souboru tím, že uspořádá kód do jednodušších konstrukcí.
- /O1 Minimalizuje velikost, zapíná řadu dalších přepínačů a ve většině případů vygeneruje nejmenší možný kód. Zapíná například i přepínač /Os.
- /O2 Maximalizuje rychlost, také zapíná řadu dalších přepínačů a ve většině případů vygeneruje nejrychlejší kód.

## 4.2 Exe packery

Jsou to aplikace, které mají za úkol minimalizovat velikost výsledné aplikace. Těchto komprimačních programů existuje celá řada, některé z nich byly přímo vyvinuty pro 64kB intra. Exe packery výslednou aplikaci zkomprimují a na její začátek vloží kód pro rozbalení a spuštění aplikace. Mezi známé komprimační programy patří kkrunchy, který byl vyvinut Německou skupinou Farbrausch. Kkrunchy je určený pro 32 bitové aplikace a používají ho i jiné skupiny, které jsou součástí

demoscény. Je to konzolový exe packer. Pro komprimaci intra, byl kkrunchy použit následujícím způsobem:

```
Kkrunchy.exe --best --refsize 64 BAKALARKA.exe --out intro.exe
```

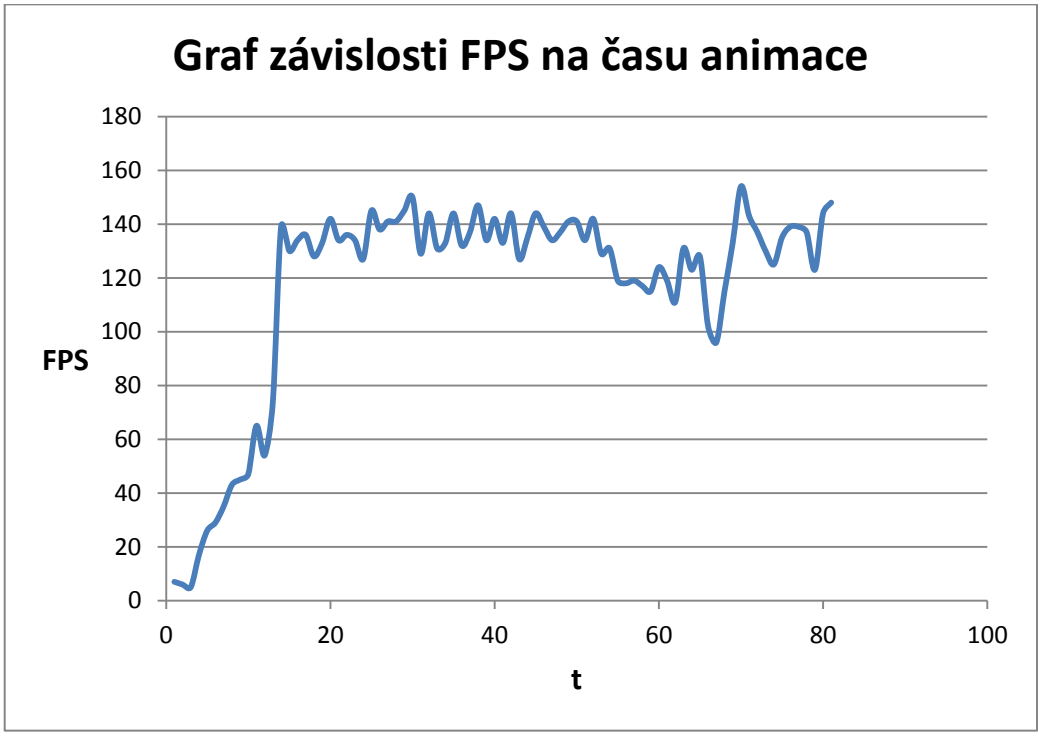
Kde *--best* je parametr pro nejlepší způsob komprimace a *--refsize* 64 znamená, že požadovaná velikost by neměla překročit 64kB, *--out* udává jméno výstupního souboru v našem případě intro.exe. Mezi další známé komprimační programy patří UPX, neboli ultimate packer for executables. UPX je multiplatformní, tedy je možné ho využít i pro linuxová intra. Při vývoji bakalářské práce byly otestovány oba komprimační programy a jejich srovnání je uvedeno v tabulce 1.

| komprimace | původní intro | kkrunchy | UPX |
|---|---|---|---|
| velikost [bytů] | 126 976 | 45 056 | 49 152 |

Tabulka 1: Porovnání komprimačních programů

# 5 Měření

Na výsledném demu byla otestována závislost FPS na průběhu času. Hodnoty FPS jsou relativně vysoké, tím pádem bychom mohli i nadále přidávat další zajímavé prvky do aplikace. Hodnoty FPS kolísají v průběhu intra, to je ovlivněno počtem objektů, které jsou právě ve scéně vykreslovány a jsou v zorném poli kamery. Na počátku jsou FPS výrazně nižší neboť je zabírána velká část scény. Test byl proveden na notebooku Acer Aspire 5745G s operačním systémem Windows 7, na kterém byl i celý projekt vytvářen. Notebook disponuje grafickou kartou NVIDIA GeForce GT330M, která podporuje OpenGL verze 330 a GLSL verze 3.3.




Graf 1: Graf závislosti FPS na času animace

# 6 Závěr

Cílem této práce bylo vytvoření grafického intra, které nepřesáhne velikost 64kB. Výsledné intro mělo velikost 44kB, tedy s rezervou splnilo požadovanou velikost. Během práce na tomto projektu bylo nastudováno a implementováno několik různých způsobů procedurálního generování:

- Perlinův šum – generování terénu,
- Constructive solid geometry – generování obličeje robota ve fragment shaderu,
- L-systémy – generování stromů,
- Částicové systémy – generování textury trávy s pomocí geometry shaderu.

Dále byly nastudovány a implementovány techniky, které zlepšují vizuální přitažlivost celé scény:

- Phongův osvětlovací model,
- Shadow mapping,
- Skybox.

Při tvorbě práce jsem se seznámil s knihovnou OpenGL a důkladně jsem si prohloubil znalosti počítačové grafiky. Jako rozšíření do budoucna by mohlo být přidání dalších přírodních prvků a objektů jako je voda, případně přidání částicového systému, který by reprezentoval spadnutí míčku do vody a následné šplouchnutí. Další možností rozšíření je předělání intra na počítačovou hru, což by vzhledem k současnému návrhu znamenalo pouze úpravu kamery a přidání několika ovládacích prvků.

# Literatura


[1] Elias, Hugo. Perlin Noise. [Online] http://freespace.virgin.net/hugo.elias/models/m_perlin.htm.

[2] Bourke, Paul. Interpolation methods. [Online] http://paulbourke.net/miscellaneous/interpolation/.

[3] Perlin, Ken. Making noise. [Online] http://www.noisemachine.com/talk1/.

[4] Perlin, Ken. *ACM Transactions on Graphics: proceedings of the ACM SIGGRAPH 2002, July 2002.* New York : Association for Computing Machinery, c2002. str. [570] s. ISBN 1-58113-521-1.

[5] Kapadia, Apu. Linear Congruential Generators. [Online] https://www.cs.indiana.edu/~kapadia/project2/node7.html.

[6] Macey, Jon. Illumination models. [Online] http://nccastaff.bournemouth.ac.uk/jmacey/CGF/slides/IlluminationModels4up.pdf.

[7] *Casting curved shadows on curved surfaces.* Williams, Lance. New York, New York, USA : ACM Press, 1978. Proceedings of the 5th annual conference on Computer graphics and interactive techniques - SIGGRAPH '78. stránky 270-274.

[8] BUNELL, Michael a Fabio PELLACINI. *Shadow Map Antialiasing* [online]. 2007 [cit. 2015-05-15]. Dostupné z: http://http.developer.nvidia.com/GPUGems/gpugems_ch11.html

[9] *Constructive solid geometry: a symbolic computation approach.* Leff, L. a Yun, D. Y. Y. New York, New York, USA : ACM Press, 1986. Proceedings of the fifth ACM symposium on Symbolic and algebraic computation - SYMSAC '86. stránky 121-126. ISBN 0897911997.

[10] Prusinkiewicz, Przemyslaw a Lindenmayer, Aristid. *The algorithmic beauty of plants.* New York : Springer-Verlag, c1990. stránky xii, 228 p. ISBN 35-409-7297-8.

[11] Bubnar, Michal. Skybox. [Online] http://www.mbsoftworks.sk/index.php?page=tutorials&series=1&tutorial=13.

[12] Elias, Hugo. Cloud Cover. [Online] http://freespace.virgin.net/hugo.elias/models/m_clouds.htm.

# Seznam příloh

## CD

- Zdrojové soubory
- Spustitelné soubory
- Video
- Readme