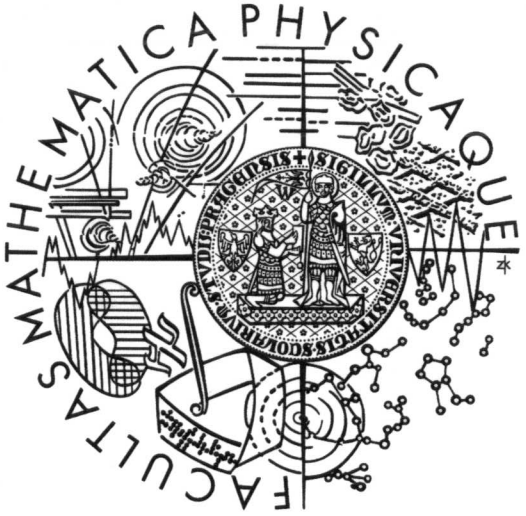
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Petr Soukup

Videozáznam pádu do černé díry

Ústav teoretické fyziky

Vedoucí bakalářské práce: Mgr. Tomáš Ledvinka, Ph.D.

Studijní program: Fyzika, obecná fyzika

2006

Poděkování

Na tomto místě bych chtěl poděkovat všem, kteří napomohli vzniku této práce, zejména pak vedoucímu mé bakalářské práce Mgr. Tomáši Ledvinkovi, Ph.D., za obětavou podporu, vedení a spoustu podnětných nápadů.

Dále děkuji autorům stránek http://nehe.ceskehry.cz/ i jejich anglické předlohy za vynikající a názorný soubor tutoriálů k OpenGL a 3D grafice.

V neposlední řadě děkuji svému příteli Robertu Goldweinovi za mnoho cenných rad ohledně programování v jazyce C/C++ a v prostředí Microsoft Visual C/C++.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 18.8.2006

Petr Soukup

Obsah

Název práce: *Videozáznam pádu do černé díry*
Autor: *Petr Soukup*
Katedra: *Ústav teoretické fyziky*
Vedoucí bakalářské práce: *Mgr. Tomáš Ledvinka, Ph.D.*
Email vedoucího bakalářské práce: ledvinka@mbox.troja.mff.cuni.cz

Abstrakt: Tato práce se zabývá popisem efektů, které mají vliv na to, jak vidí okolní svět pozorovatel pohybující se relativistickými rychlostmi v Minkowského prostoročase a problematikou jejich vizualizace v rámci počítačové grafiky. Díky Lorentzově transformaci jsou směry šíření světelného paprsku v různých inerciálních soustavách různé. Pozorovatel postupně zrychlující na relativistickou rychlost vidí, jak se všechny okolní předměty přibližují k ose, po které se pohybuje vpřed – aberace. Metoda použitá v této práci k vizualizaci těchto efektů se nazývá relativistický ray tracing – jedná se o zpětné sledování paprsku vycházejícího z konkrétního místa na projekčním plátně kamery pozorovatele a berou se v úvahu efekty speciální relativity při přechodu z jedné inerciální soustavy do druhé. Grafický engine vytvořený v této práci je univerzální nástroj, který lze použít k vizualizaci většiny relativistických efektů a to i v rámci křivočarého prostoročasu Obecné teorie relativity.

Klíčová slova: *simulace vizuálních efektů ve Speciální teorii relativity, aberace, Dopplerův jev, Minkowského prostoročas*

Title: *Movie - One flight into a black hole*
Author: *Petr Soukup*
Department: *Department of theoretical physics*
Supervisor: *Mgr. Tomáš Ledvinka, Ph.D.*
Supervisor's email address: ledvinka@mbox.troja.mff.cuni.cz

Abstract: There are several effects that determine the way in which an observer traveling with relativistic speed in Minkowski spacetime sees the surrounding world. In this paper, I describe these effects and focus on implementation of visualization of such effects on PC. Photons travel in different directions in different Lorentz frames. An accelerating observer when reaching relativistic speeds sees all surrounding objects approaching the axis of his forward movement. This effect is known as relativistic aberration. I use relativistic ray tracing to visualize these effects which means tracing photons backwards from the place of their impact on projection screen of the camera to the place of their origin while taking Lorentz transformation from one Lorentz frame to another into account. Graphical engine created in this work is a universal tool which can serve as a basis for visualizing all relativistic effects and is easily extendable to curved spacetime of General relativity.

Keywords: *simulation of visual effects in Special Relativity, aberration, Minkowski spacetime, Doppler effect*

Úvod

Informace o okolním světě jsou pozorovateli zprostředkovány světlem - tj. fotony. Ve Speciální i Obecné teorii relativity světlo ovšem, jako všechny fyzikální objekty samo podléhá zákonům těchto teorií. Existuje tedy zřetelný rozdíl mezi tím, co se děje a to co pozorovatel vidí. Tato práce se zabývá tím jak vidí pozorovatel okolní svět v rámci Speciální teorie relativity. Dnešní moderní hardware poskytuje rozsáhlé možnosti vizualizace fyzikálních problémů na osobních počítačích, které dříve nebyly možné nebo byly neskonale obtížnější. V této práci je využito těchto možností k vizualizaci výše popsaných efektů.

1. KAPITOLA

Úvodní poznámky

Formalismus použitý v této práci odpovídá běžnému formalismu používanému dnes v Speciální a obecné teorii relativity – dále jen STR a OTR, tak jak se vyskytuje v moderních učebnicích Teorie Relativity – např. – [1]. Zejména pak kovariantní zápis rovnic pomocí vektorů a 1-forem, tak je pospán např. v [1] – Part II.

Velikost konstanty c – rychlosti světla je položena rovna 1.

V oddílech 2.1, 2.2, 2.3 jsem čerpal z [1] – částí I a II.

2. KAPITOLA

Efekty Speciální relativity na pohybujícího se pozorovatele

2.1 Světočáry a vektory

Primitivní idea vektoru je, že vektor je spojnice dvou bodů, zde řekněme spojnice dvou prostoročasových událostí A a B. Pro účely Teorie Relativity je vhodné o vektoru uvažovat jiným způsobem. Spojnici dvou bodů můžeme reprezentovat parametrizovanou úsečkou $P(\lambda) = A + \lambda(B - A)$ kde $\lambda = 0$ odpovídá počátku vektoru a $\lambda = 1$ jeho koncovému bodu – viz obr. 1.1. Derivací podle parametru λ získáme:

$$\frac{d}{d\lambda}P(\lambda) = B - A = P(1) - P(0) \tag{2.1.1}$$

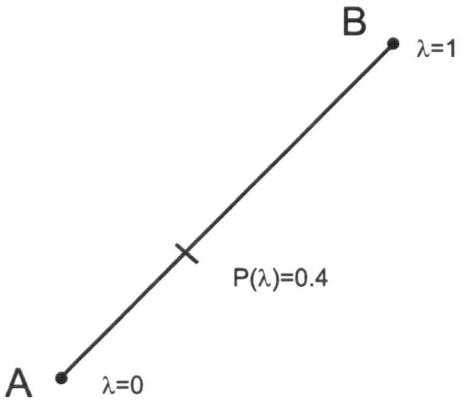
a tedy namísto

$$\vec{AB} = B - A \tag{2.1.2}$$

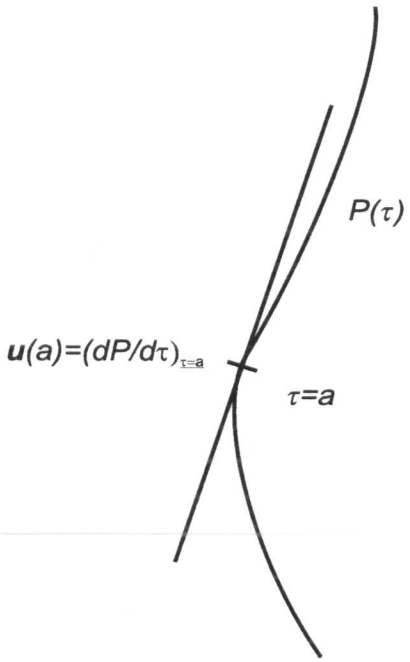
můžeme psát

$$\vec{AB} = \left(\frac{dP}{d\lambda}\right)_{\lambda=0} \tag{2.1.3}$$

Tento koncept nám umožňuje zapsat vektor jakožto tečný vektor ke křivce $P(\lambda)$ v daném bodě. Křivka $P(\lambda)$ v obecném případě nemusí být přímka, ale libovolná diferencovatelná parametrická křivka.



Obr. 1.1.: Parametrická úsečka



Obr. 1.2.: 4-rychlost v daném bodě je derivací světočáry podle parametru τ.

Toto je obvyklý způsob zápisu vektoru v Teorii Relativity. Pokud je $P(\tau)$ světočára pohybujícího se objektu, kterou parametrizujeme jeho vlastním časem, pak derivací $dP/d\tau$ v libovolném bodě získáme 4-rychlost $\boldsymbol{u}$ tohoto objektu – viz. obr 1.2. Výhodou tohoto přístupu je naprostá nezávislost na jakýchkoliv souřadnicích. Světočára $P(\tau)$ a vektor 4-rychlosti $\boldsymbol{u}$ jsou čistě geometrickými objekty bez vázanosti na souřadnice. Souřadnice vstupují do hry teprve volbou konkrétní vztažné soustavy reprezentované ortonormálními bázovými vektory $\boldsymbol{e_0}$, $\boldsymbol{e_1}$, $\boldsymbol{e_2}$, $\boldsymbol{e_3}$. Světočáru v rozvoji do této báze zapsat vztahem:

$$P(\tau) = x^0(\tau)\boldsymbol{e}_0 + x^1(\tau)\boldsymbol{e}_1 + x^2(\tau)\boldsymbol{e}_2 + x^3(\tau)\boldsymbol{e}_3 = x^\mu(\tau)\boldsymbol{e}_\mu \tag{2.1.4}$$

a podobně pro 4-rychlost :

$$\boldsymbol{u} = \frac{dP(\tau)}{d\tau} = u^0\boldsymbol{e}_0 + u^1\boldsymbol{e}_1 + u^2\boldsymbol{e}_2 + u^3\boldsymbol{e}_3 = \frac{dx^\mu}{d\tau}\boldsymbol{e}_\mu \tag{2.1.5}$$

Podobně lze zapsat jakýkoliv další vektor.

2.2 Hyperbolický pohyb

Mějme pozorovatele který se pohybuje v Minowského prostoročase s konstantním zrychlením vůči nějaké inerciální soustavě. 4-rychlost pozorovatele splňuje vztah

$$u^\nu u_\nu = -1 \tag{2.2.1}$$

a jeho 4-zrychlení

$$\boldsymbol{a} = \frac{d\boldsymbol{u}}{d\tau} \tag{2.2.2}$$

je kolmé na 4-rychlost neboť

$$a^\nu u_\nu = \frac{1}{2}\frac{d}{d\tau}(u^\nu u_\nu) = 0 \tag{2.2.3}$$

díky (2.2.1). Vezměme nyní okamžitou klidovou soustavu pozorovatele – tj. soustavu, která se v daném okamžiku pohybuje stejnou rychlostí jako pozorovatel. V takové soustavě v daném okamžiku jistě platí

$$\boldsymbol{e_0} = \boldsymbol{u} \tag{2.2.4}$$

Kde $\boldsymbol{e_0}$ je nultý bázový vektor. Z (2.2.3) a pak plyne vztah a^0=0 pro tuto vztažnou soustavu. Prostorové složky 4-zrychlení se pak redukují na obyčejné zrychlení

$$a^i = \frac{d^2x^i}{dt^2} \tag{2.2.5}$$

Invariantní čtverec velikosti zrychlení pak můžeme zapsat vztahem

$$\boldsymbol{a}^2 = a^\nu a_\nu = \left(\frac{d^2x}{dt^2}\right)^2 \tag{2.2.6}$$

Nechť se pozorovatel pohybuje s konstantním zrychlením z ve směru x^1. Pohybové rovnice pak budou dány vztahy:

$$u^0 = \frac{dt}{d\tau} \qquad u^1 = \frac{dx^1}{d\tau} \tag{2.2.7}$$

$$a^0 = \frac{du^0}{d\tau} \qquad a^1 = \frac{du^1}{d\tau} \tag{2.2.8}$$

Z (2.2.6) plyne

$$a^\nu a_\nu = z^2 . \tag{2.2.9}$$

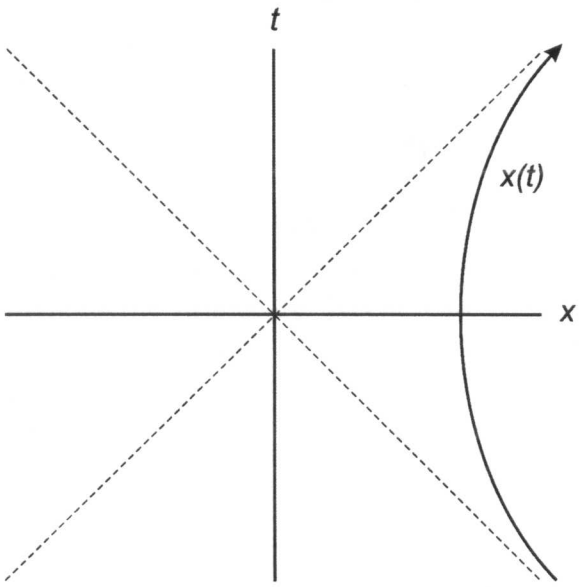
S využitím vztahů (2.2.1) a (2.2.3) dostaneme soustavu lineárních diferenciálních rovnic,

$$\frac{du^0}{d\tau} = zu^1 \qquad \frac{du^1}{d\tau} = zu^0 \tag{2.2.10}$$

Jejím řešením dostáváme vztahy

$$x = \frac{\cosh z\tau}{z} \qquad t = \frac{\sinh z\tau}{z} \tag{2.2.11}$$

Při znázornění světočáry tohoto pozorovatele v prostoročasovém diagramu dostáváme hyperbolu – odtud název hyperbolický pohyb. Vše je vidět na obr. 1.3.



Obr. 1.3: Hyperbolický pohyb
Světočára pozorovatele, který cestuje s konstantním zrychlením vůči inerciální soustavě, která se v daném okamžiku pohybuje stejnou rychlostí jako pozorovatel je hyperbola.

2.3 Vztažná soustava pozorovatele pohybujícího se s konstantním zrychlením. Tetráda

Zavést systém souřadnic urychleného pozorovatele je z různých důvodů problematické. Lze ovšem zavést lokální systém souřadnic/vztažnou soustavu reprezentovaný tetrádou neboli systémem ortonormálních bázových vektorů $\boldsymbol{e}_0', \boldsymbol{e}_1', \boldsymbol{e}_2', \boldsymbol{e}_3'$. Pro bázový vektor $\boldsymbol{e}_0'$ vztažné soustavy vůči které je pozorovatel v daném okamžiku v klidu bude platit $\boldsymbol{e}_0' = \boldsymbol{u}$, kde $\boldsymbol{u}$ je 4-rychlost pozorovatele. Tedy z (2.2.11) dostaneme

$$\boldsymbol{e}_0' = (\cosh z\tau, \sinh z\tau, 0, 0) \tag{2.3.1}$$

Vektory $\boldsymbol{e}_2'$ a $\boldsymbol{e}_3'$ nejsou pohybem pozorovatele nijak dotčeny – pozorovatel se pohybuje pouze ve směru 1, tyto vektory jsou totožné s bázovými vektory $\boldsymbol{e}_2$ a $\boldsymbol{e}_3$ vztažné soustavy, která splývala se vztažnou soustavou pozorovatele v okamžiku, kdy začal zrychlovat. Nazývejme tuto vztažnou soustavu dále globální vztažnou soustavou. Poslední bázový vektor $\boldsymbol{e}_1'$ musí být ortogonální vůči zbývajícím třem a tedy

$$\boldsymbol{e}_1' = (\sinh z\tau, \cosh z\tau, 0, 0) \tag{2.3.2}$$

a celkem tedy

$$\begin{aligned} \boldsymbol{e}_0' &= (\cosh z\tau, \sinh z\tau, 0, 0) \\ \boldsymbol{e}_1' &= (\sinh z\tau, \cosh z\tau, 0, 0) \\ \boldsymbol{e}_2' &= (0, 0, 1, 0) \\ \boldsymbol{e}_3' &= (0, 0, 0, 1) \end{aligned} \tag{2.3.3}$$

2.4 Lorentzova transformace

Mějme dvě inerciální soustavy, které se vůči sobě pohybují v daném okamžiku rovnoměrně přímočaře. Nechť čárkovaná soustava se vůči nečárkované pohybuje rychlostí v kladným směrem ve směru *1*. Pak transformační vztahy pro libovolné polohové 4-vektory ve zmíněných soustavách budou dány tzv. speciální Lorenzovou transformací

$$x^{\mu'} = \Lambda^{\mu'}{}_{\nu} x^{\nu} \qquad x^{\nu} = \Lambda^{\nu}{}_{\mu'} x^{\mu'} \tag{2.4.1}$$

Připomeňme, že platí

$$\boldsymbol{u} = \frac{dx^{\mu}}{d\tau} \boldsymbol{e}_{\mu} \tag{2.4.2}$$

Pro vektory bází obou soustav lze z faktu, že 4-rychlost $\boldsymbol{u}$ je nezávislá na souřadnicích, (viz oddíl 2.1 a vztahy 2.1.4 a 2.1.5) s použitím (2.4.2) snadno odvodit transformační vztahy:

$$\boldsymbol{e}_{\mu'} = \boldsymbol{e}_{\nu} \Lambda^{\nu}{}_{\mu'} \qquad \boldsymbol{e}_{\nu} = \boldsymbol{e}_{\mu'} \Lambda^{\mu'}{}_{\nu} \tag{2.4.3}$$

a pro libovolný vektor s rozkladem do báze

$$\boldsymbol{v} = \boldsymbol{e}_\mu v^\mu \tag{2.4.4}$$

pak

$$v^{\mu'} = \Lambda^{\mu'}{}_{\nu} v^\nu \qquad v^\nu = \Lambda^\nu{}_{\mu'} v^{\mu'} \tag{2.4.5}$$

Matice $\Lambda^{\mu'}{}_{\nu}$ a $\Lambda^{\nu}{}_{\mu'}$ jsou navzájem inverzní což plyne s faktu, že se jedná o relativní pohyb v opačných směrech. **Chyba! Nenalezen zdroj odkazů.Chyba! Nenalezen zdroj odkazů.Chyba! Nenalezen zdroj odkazů.Chyba! Nenalezen zdroj odkazů.**

Není překvapující zjištění, že matice jejíž řádky tvoří vektory báze (2.3.3) je vlastně maticí Lorenzovy transformace $\Lambda^\nu{}_{\mu'}$. Především – soustava s bází (2.3.3) se pohybuje vůči globální soustavě v daném okamžiku rovnoměrným přímočarým pohybem a Lorentzova transformace v daném okamžiku mezi oběma systémy je tedy možná. Pokud vyjdeme z levého vztahu (2.4.3) a za nečárkovanou bázi vezmeme ortonormální bázi globálního souřadného systému

$$\begin{aligned} \boldsymbol{e}_0 &= (1,0,0,0) \\ \boldsymbol{e}_1 &= (0,1,0,0) \\ \boldsymbol{e}_2 &= (0,0,1,0) \\ \boldsymbol{e}_0 &= (0,0,0,1) \end{aligned} \tag{2.4.6}$$

Pak dle levého vztahu (2.4.3) můžeme psát:
$(\cosh z\tau, \sinh z\tau, 0, 0) = (1,0,0,0)\Lambda^0{}_{0'} + (0,1,0,0)\Lambda^1{}_{0'} + (0,0,1,0)\Lambda^2{}_{0'} + (0,0,0,1)\Lambda^3{}_{0'}$ a tedy 1. řádek matice L.T. je $(\cosh z\tau, \sinh z\tau, 0, 0)$. Podobně můžeme postupovat pro zbývající tři řádky a dostáváme

$$\Lambda^\nu{}_{\mu'} = \begin{bmatrix} \cosh z\tau & \sinh z\tau & 0 & 0 \\ \sinh z\tau & \cosh z\tau & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4.7}$$

$$\Lambda^{\mu'}{}_{\nu} = \begin{bmatrix} \cosh z\tau & -\sinh z\tau & 0 & 0 \\ -\sinh z\tau & \cosh z\tau & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4.8}$$

kde 1. matice je totožná s (2.3.3) což jsme chtěli ukázat.

2.5 Šíření signálu. Vlnová klubka, rovinné vlny a fotony

Nechť nějaký zdroj elektromagnetického záření – např. hvězda vyšle do prostoru krátký signál – vlnové klubko. Vlnové klubko je příčně i podélně ohraničený svazek vln. Takový svazek je možné vytvořit superpozicí nekonečných rovinných, lineárně polarizovaných harmonických vln ve tvaru

$$\boldsymbol{E}(\boldsymbol{x},t) = \boldsymbol{E}_0 \sin 2\pi\nu(t - e^j x_j) \tag{2.5.1}$$

$$\boldsymbol{H}(\boldsymbol{x},t) = \boldsymbol{H}_0 \sin 2\pi\nu(t - e^j x_j) \tag{2.5.2}$$

kde ***E, H*** jsou intenzity elektrického a magnetického pole, ***e*** je jednotkový směrový vektor šíření vlny a ν je frekvence v daném inerciálním systému; s frekvencemi ležícími v nějakém malém okolní frekvence ν a se směry šíření, který se rozkládá v jistém malém prostorovém úhlu okolo vektoru ***e***. V takovém případě se totiž vlnění až na jistou malou ohraničenou prostorovou oblast interferencí téměř vyruší. Odchylky od monochromatičnosti a paralelnosti vlnového klubka lze pro účely těchto úvah zanedbat, jak tvrdí Votruba v [3] – kapitola VI – oddíl 4.3. Dále budeme tedy vlnové klubko považovat za harmonickou rovinnou vlnu s konečným prostorovým rozsahem. Této vlně pak můžeme přiřadit konkrétní frekvenci ν a vlnový vektor

$$\boldsymbol{k} = \frac{2\pi}{\lambda}\boldsymbol{e} = 2\pi\nu\boldsymbol{e} \tag{2.5.3}$$

což můžeme jednotně zapsat jako vlnový 4-vektor k^μ

$$k^\mu = (2\pi\nu, \boldsymbol{k}) = (2\pi\nu, 2\pi\nu\boldsymbol{e}) \tag{2.5.3a}$$

kde $2\pi\nu = \omega$ je kruhová frekvence. Této vlně můžeme taktéž přiřadit konečnou energii E a hybnost ***p***, jež zapíšeme ve formě 4-hybnosti

$$\boldsymbol{P}^\mu = (E, \boldsymbol{p}) \tag{2.5.4}$$

Pro prostorové složky hybnosti pak platí

$$\boldsymbol{p} = E\boldsymbol{e} \tag{2.5.5}$$

kde ***e*** je jednotkový směrový vektor šíření vlny: $\boldsymbol{e}^2 = 1$.

Pro 4-hybnost pak platí

$$\boldsymbol{P}^\mu \boldsymbol{P}_\mu = 0 \tag{2.5.6}$$

Mějme dvě inerciální soustavy vzájemně se vůči sobě pohybující jako v oddílu 2.4. Nechť $\boldsymbol{P}^\mu$ je 4-hybnost vlnového klubka v nečárkované soustavě. Pak pro hybnost klubka v čárkované soustavě bude platit vztah

$$P^{\mu'} = \Lambda^{\mu'}{}_\nu P^\nu \tag{2.5.7}$$

kde $\Lambda^{\mu'}{}_\nu$ je matice (2.4.8). Pro jednotlivé komponenty pak bude platit:

$$P^{0'} = \cosh(z\tau)P^0 - \sinh(z\tau)P^1 \tag{2.5.8}$$

$$P^{1'} = \cosh(z\tau)P^1 - \sinh(z\tau)P^0 \tag{2.5.9}$$

$$P^{2'} = P^2 \qquad P^{3'} = P^3 \tag{2.5.10}$$

Z (2.5.5) dostaneme vztah pro směr šíření signálu

$$e^{i'} = \frac{P^{i'}}{P^{0'}} \tag{2.5.11}$$

Jestliže $\boldsymbol{e_0}' = \boldsymbol{u}$ v (2.3.3), kde $\boldsymbol{u}$ je 4-rychlost pozorovatele, pak vztah pro obyčejnou rychlost čárkované vůči nečárkované soustavě $\beta = \frac{dx}{dt}$ bude dán, vezmeme-li v úvahu definici vektoru 4-rychlosti, podílem 1. a 0-té složky $\boldsymbol{u}$ tedy

$$\beta = \frac{\sinh z\tau}{\cosh z\tau} = \tanh z\tau \tag{2.5.12}$$

pro γ pak dostáváme výraz

$$\gamma = (1-\beta^2)^{-\frac{1}{2}} = \cosh z\tau \text{ a} \qquad \gamma\beta = \sinh z\tau \tag{2.5.13}$$

a pro jednotkový vektor směru šíření signálu pak z (2.5.8) – (2.5.10) a s pomocí (2.5.12) a (2.5.13) dostáváme

$$e^{1'} = \frac{e^1 - \beta}{1-\beta e^1} \qquad e^{2'} = \frac{e^2}{\gamma(1-\beta e^1)} \qquad e^{3'} = \frac{e^3}{\gamma(1-\beta e^1)} \tag{2.5.14}$$

$$E' = E\gamma(1-\beta e^1) \tag{2.5.15}$$

Transformační vztah pro frekvenci harmonické rovinné vlny mezi dvěma inerciálními soustavami je

$$\nu' = \nu\gamma(1-\beta e^1) \tag{2.5.16}$$

kde e^1 je směr šíření vlny v nečárkované soustavě. Podrobné odvození zde provádět nebudeme – lze ho nalézt např. v [3] - kapitola VI – oddíl 4.1. Z (2.5.15) a (2.5.16) plyne invariantní vztah

$$\frac{E'}{\nu'} = \frac{E}{\nu} = K \tag{2.5.17}$$

a pomocí (2.5.5) také

$$\boldsymbol{p} = K\nu\boldsymbol{e} \tag{2.5.18}$$

Podle kvantové elektrodynamiky a kvantové teorie pole musí světelná vlna šířící se směrem $\boldsymbol{e}$ o frekvenci ν obsahovat vždy celý počet světelných kvant – fotonů, pro jejichž energii a hybnost platí vztahy

$$E = \hbar\omega \qquad\qquad \boldsymbol{p} = \hbar\boldsymbol{k} \tag{2.5.19}$$

kde $\hbar = 1{,}054\,571 \,.\, 10^{-34}$ J.s je redukovaná Planckova konstanta. Z (2.5.17) a (2.5.18) plyne invariance Planckovy konstanty a tím pádem i vztahů (2.5.19) a můžeme pro foton tím pádem psát

$$P^\mu = \hbar k^\mu \tag{2.5.20}$$

a pro foton tím pádem platí i vztahy (2.5.4) – (2.5.11) a (2.5.14.), (2.5.15).

Při odvozování vztahů pro elektromagnetické vlny a fotony v tomto oddíle jsem čerpal z [3] – kapitola VI – oddíly 3.3 – 4.3.

2.6 Relativistická aberace

V dalším textu budou veškerými nositeli informace fotony. Směr kterým pozorovatel vidí předmět bude tedy dán směrem příletu fotonů vyzářených nebo odražených od předmětu.

Mějme dvě inerciální soustavy vzájemně se vůči sobě pohybující jako v oddílu 2.5. Nechť $\boldsymbol{P}^{\mu}$ je 4-hybnost fotonu v nečárkované soustavě. Pak pro foton, jak již bylo řečeno v minulém oddíle platí vztahy (2.5.4) – (2.5.11) a (2.5.14.), (2.5.15). Vztahy (2.5.14) představují vlastně relativistickou teorii aberace.

Přímým výpočtem můžeme získat vztahy pro inverzní transformaci:

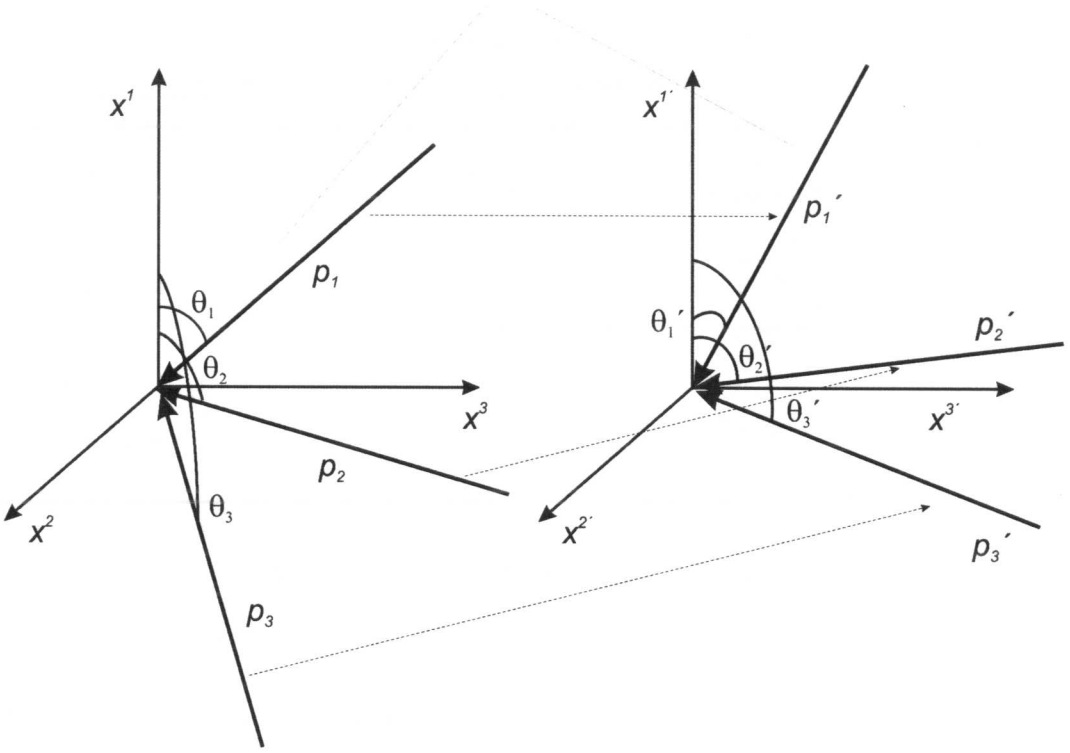
$$e^1 = \frac{e^{1'} + \beta}{1 + \beta e^{1'}} \qquad e^2 = \frac{e^{2'}}{\gamma(1 + \beta e^{1'})} \qquad e^3 = \frac{e^{3'}}{\gamma(1 + \beta e^{1'})} \tag{2.6.1}$$

Nechť směr pohybu fotonu v nečárkované soustavě svírá s kladným směrem osy x^1 úhel θ. Pak bude $e^1 = \cos\theta$. Přímým výpočtem pak můžeme odvodit vztah pro úhel α, který svírají směry šíření fotonů v obou soustavách, který s přesností do řádu β^2 bude

$$\tan\alpha \cong \beta \sin\theta (1 + \frac{1}{2}\beta\cos\theta) \tag{2.6.2}$$

Jak tedy vypadá okolní svět pro pozorovatele, který je v klidu vůči čárkované soustavě? Z (2.5.14) je vidět: Pokud položíme e^3 rovno 0 (osová symetrie celého problému kolem osy 1 je zřejmá), pak bude e^2=sinθ a lze se pro libovolný úhel přesvědčit, že úhel θ', který svírá směr příchozího fotonu s osou $x^{1'}$ v čárkované soustavě je pro libovolný směr menší než úhel θ, který svírá směr příchozího fotonu s osou x^1 v nečárkované soustavě, tj. směry všech příchozích fotonů k pozorovateli se v čárkované soustavě „přimykají" k zápornému směru osy $x^{1'}$ – viz obr. 2.4.

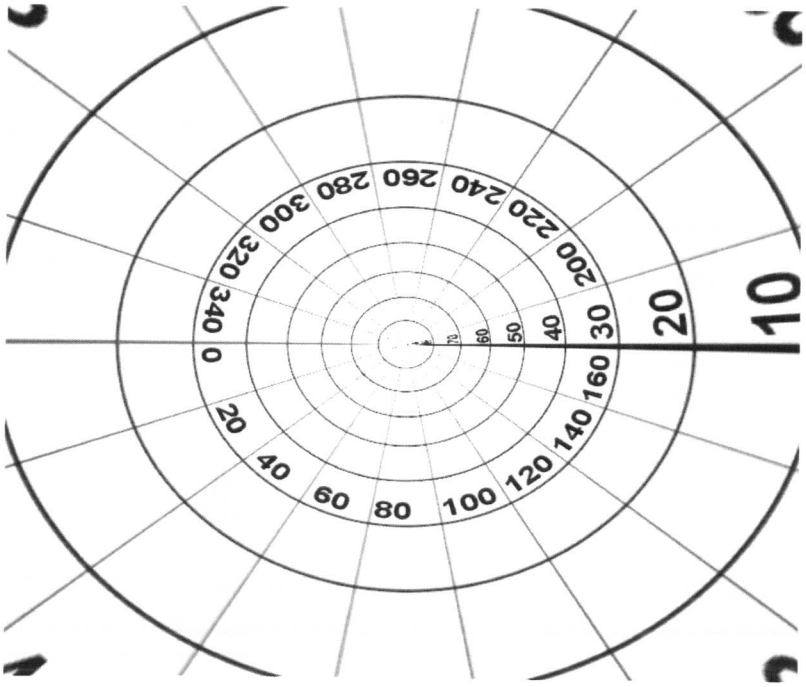
Směr v jakém vidí pozorovatel daný předmět je dán tím, z jakého směru do jeho oka dopadají fotony z předmětu. Tj. pokud se pozorovatel letící relativistickou rychlostí dívá před sebe, všechny objekty které jsou v klidu vůči nečárkované soustavě se přibližují k ose $x^{1'}$, pokud za sebe, všechny objekty se od ní vzdalují – viz též obr. 2.4. Čím více se rychlost pozorovatele vůči nečárkované soustavě blíží rychlosti světla, tím je efekt znatelnější. Na obr. 2.5 – 2.13 jsou vyobrazeny efekty aberace pro pozorovatele letícího vesmírem – jak by pozorovatel viděl okolní vesmír. Tyto obrázky jsou vygenerovány programem, který je předmětem této práce. Při tvorbě pohybu vzad a do stran byl pro zjednodušení pouze změněn směr pohybu pozorovatele daný transformační maticí, čili pozorovatel pořád kouká do kladného směru 1, ale pohybuje se různými směry. Nicméně ilustraci toho, jak by viděl okolní vesmír to dává dostatečnou.



Obr. 2.4: Příchozí fotony se v čárkované soustavě přimykají k zápornému směru osy $x^{1\prime}$.
Směry všech příchozích fotonů k pozorovateli se v čárkované soustavě „přimykají" k zápornému směru osy $x^{1\prime}$. Zajímavý je osud paprsku p_2, který pozorovatel v nečárkované soustavě hledící ve směru 1 nevidí, kdežto pozorovatel v čárkované soustavě ano. Čili druhý pozorovatel vidí jakoby „za sebe".

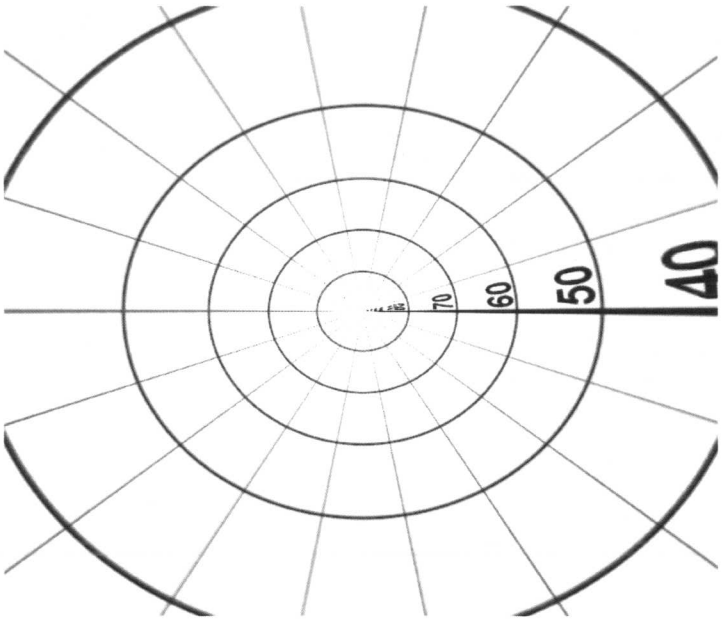
Obr. 2.5.: Pozorovatel v klidu

Takto čelním okénkem ze své lodi vidí okolní vesmír pozorovatel v klidu. Pro pohled vzad bude obrázek stejný.



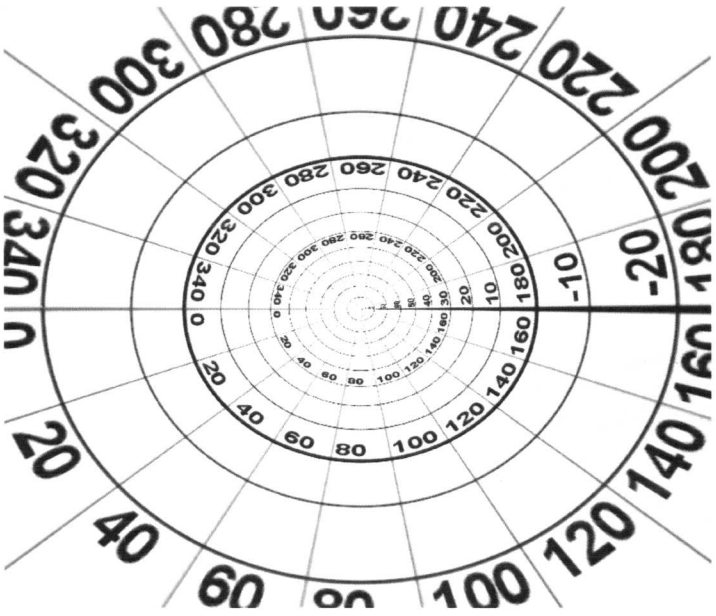
Obr. 2.5a.: Pozorovatel při rychlosti β=0.3 pohled vpřed

Zorné pole pozorovatele se při pohledu vpřed rozšiřuje – množství informace o okolním vesmíru je větší. Celý vesmír se jakoby „stěhuje" před pozorovatele.



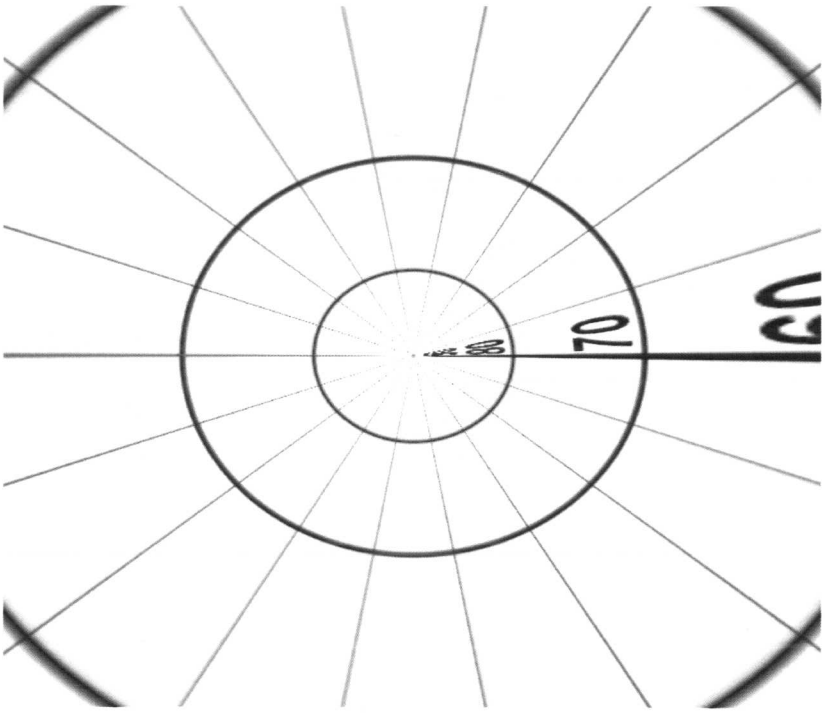
Obr. 2.6.: Pozorovatel při rychlosti β=0.3 pohled vzad

Zorné pole pozorovatele se při pohledu vzad zužuje – množství informace o okolním vesmíru je menší. Celý vesmír se jakoby „stěhuje" za pozorovatele (tj. ve smyslu směru, kterým se pozorovatel dívá).



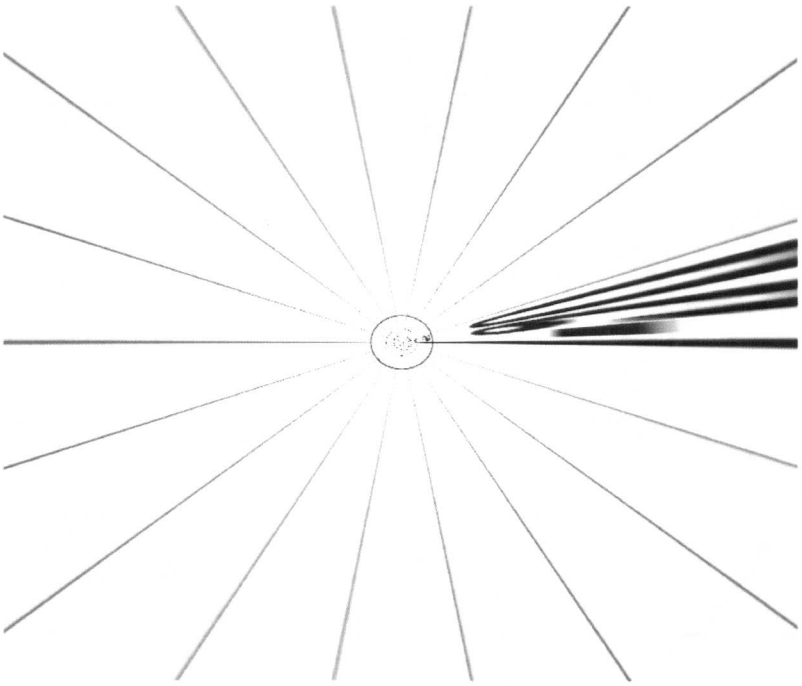
Obr. 2.7.: Pozorovatel při rychlosti β=0.7 pohled vpřed

Efekt se prohlubuje pozorovatel již vidí pod rovník, čili jakoby „ za sebe".



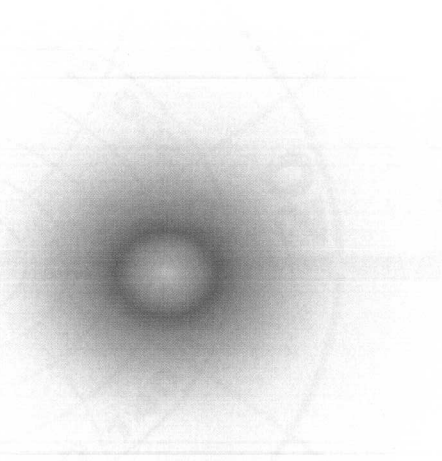
Obr. 2.8.: Pozorovatel při rychlosti β=0.7. Pohled vzad

Prohlubující se efekt z předchozího obrázku pro pozorovatele hledícího vzad. Ze zorného pole toho již mnoho nezbývá.

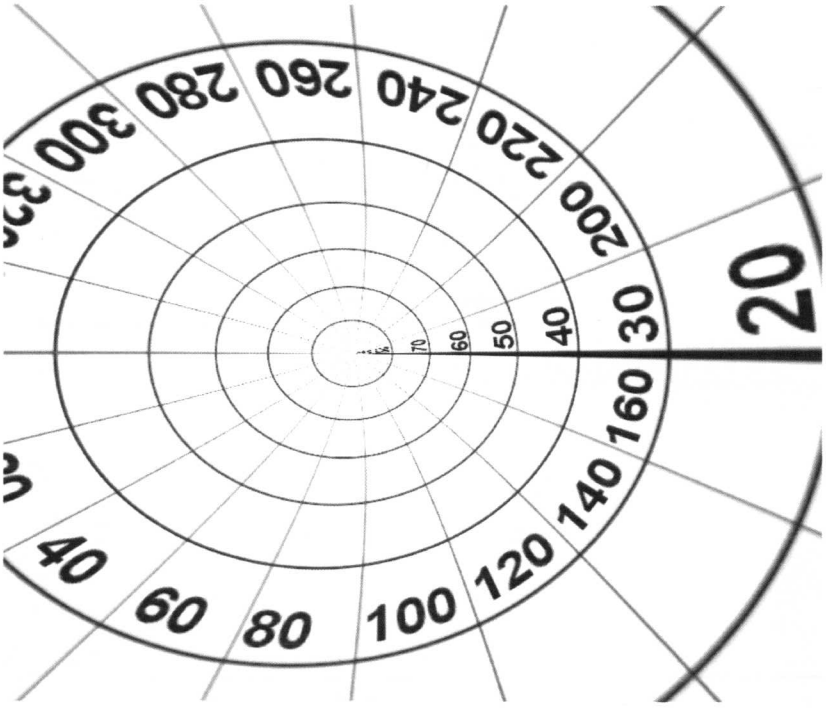


Obr. 2.9.: Pozorovatel při rychlosti β=0.9999 pohled vpřed

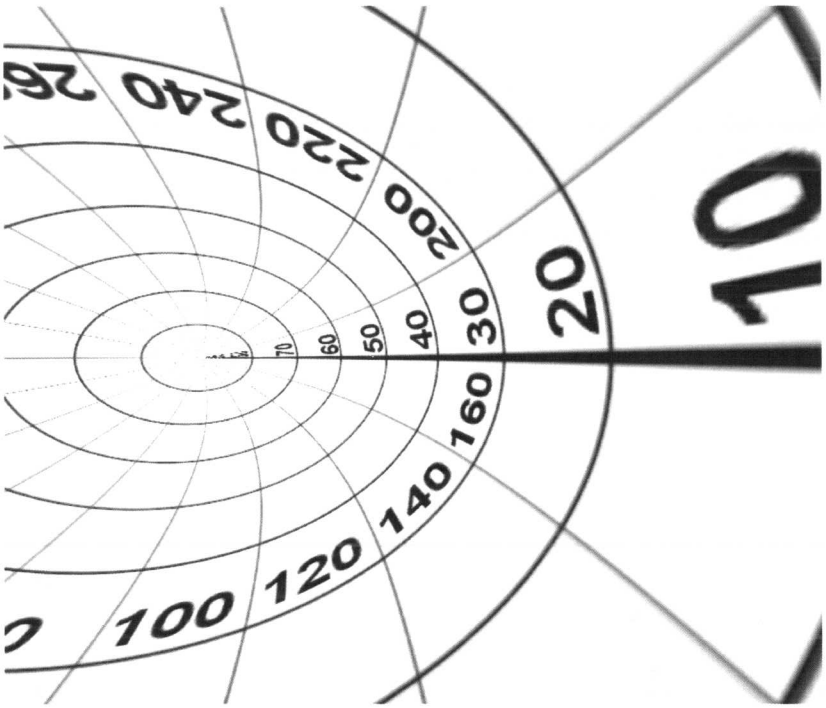
Téměř celý vesmír se přestěhoval před pozorovatele. Rozmazané číslo je -90, čili je vidět téměř až na jižní pól.



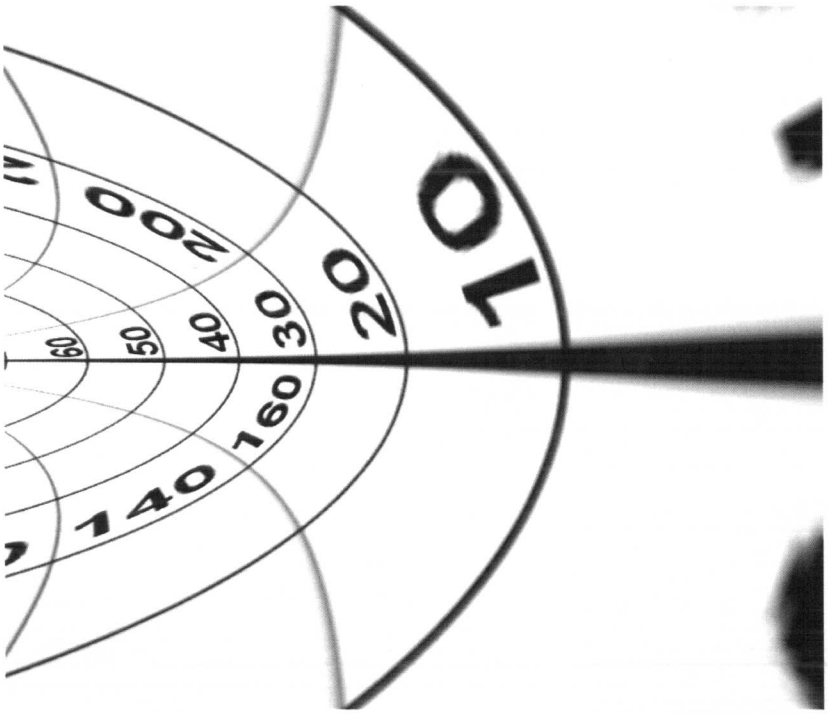
Obr. 2.10.: Pozorovatel při rychlosti β=0.9999 pohled vzad
Pozorovatel toho již mnoho nevidí.



Obr. 2.11.: Pozorovatel při rychlosti β=0.3. Pohled do strany
Zde je okolí mírně deformované



Obr. 2.12.: Pozorovatel při rychlosti β=0.7 pohled do strany
Okolí se postupně deformuje



Obr. 2.13.: Pozorovatel při rychlosti β=0.9. Pohled do strany

2.7 Relativistický Dopplerův jev

Vlnová délka fotonu / vlny je pochopitelně také ovlivněna volbou inerciální soustavy a proto se vlnové délky jeví různým pozorovatelům různě. Mějme pozorovatele pohybujícího se stejným způsobem jako v předchozím oddílu. Pak vztah pro transformaci vlnových délek mezi oběma soustavami bude

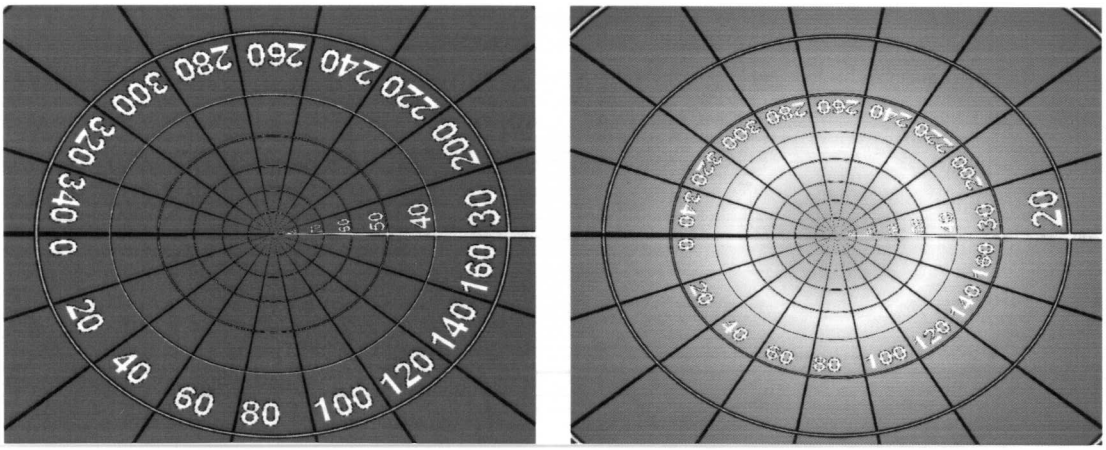
$$\lambda' = \lambda D \tag{2.7.1}$$

kde faktor D je dán vztahem

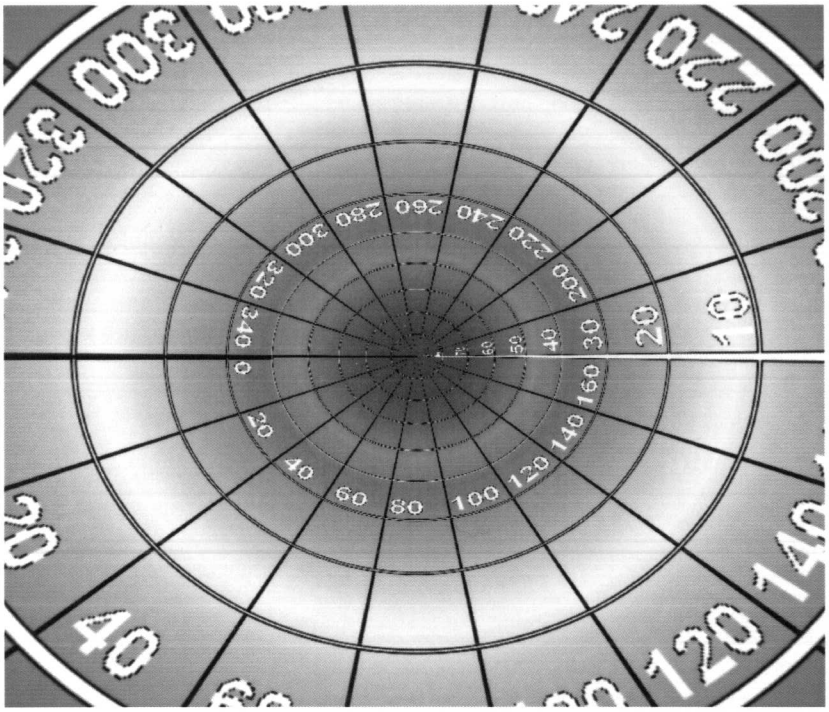
$$D = \frac{1}{\gamma(1-\beta\cos\theta)} \tag{2.7.2}$$

a úhel θ je brán ve stejném smyslu jako v předchozích oddílech. Podrobné odvození těchto vztahů je dostatečně známé, lze jej nalézt např. v [3] – kapitola VI – oddíl 4.1 – 4.2.

Pro pozorovatele, který letí stejným směrem jako přilétající foton z (2.7.1) a (2.7.2) plyne, že vlnové délky se v jeho soustavě prodlužují, ve viditelné části spektra se stěhují k červenému okraji – *rudý posuv*. Vlnové délky fotonů, které letí vstříc pozorovateli se naopak zkracují – *modrý posuv*. Velikost změny vlnové délky je samozřejmě ovlivněna úhlem, pod kterým foton v nečárkované soustavě letí. Pro foton letící libovolným směrem se jedná o složení tzv. *podélného* a *příčného* Dopplerova jevu. Jak vizuálně vypadá rudý a modrý a Dopplerův jev pro obecný směr je vidět na obr. 2.14 – 2.21. Z důvodů názornosti je celé okolí vybarveno jednou barvou. Na obrázcích je pak vidět jak konkrétní část spektra (v tomto případě červené) vypadá v různých úhlech při různých rychlostech. Pro ostatní barvy spektra to bude vypadat podobně, červenou složku jsem zvolil proto, že je vidě nejdéle. Obrázky v tomto oddílu jsou opět vygenerovány programem, který je předmětem této práce.

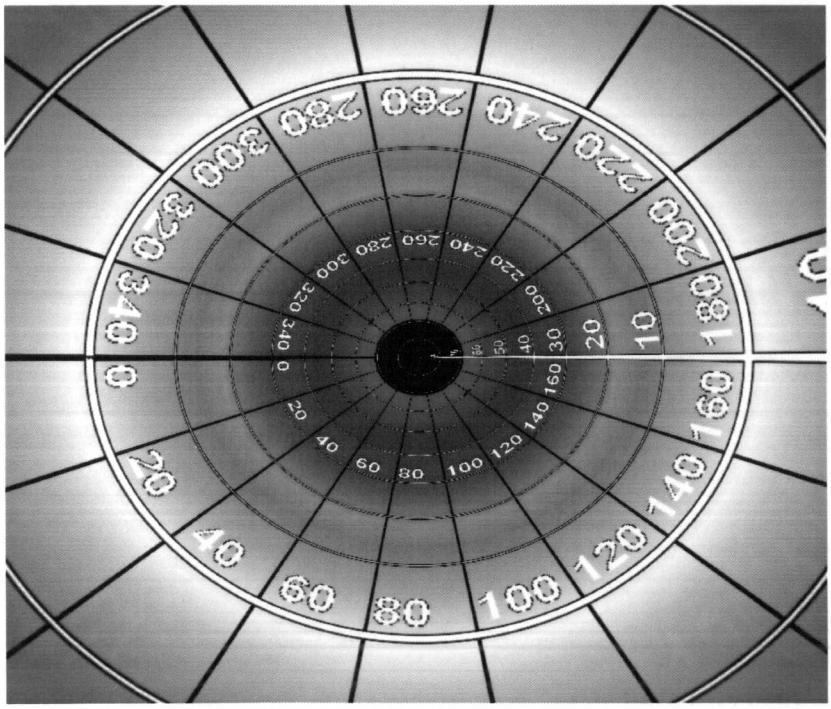


Obr. 2.14a,b: Dopplerův efekt pro nízké rychlosti:
Na levém obrázku je vidět, jak vypadá červená část spektra pro stojícího pozorovatele, na pravém pak jak se postupně mění pro pozorovatele pohybujícího se rychlostí β=0.1. Je vidět, že posun je větší pro fotony nalétávající v čelním směru než pro fotony přilétávající ze stran.



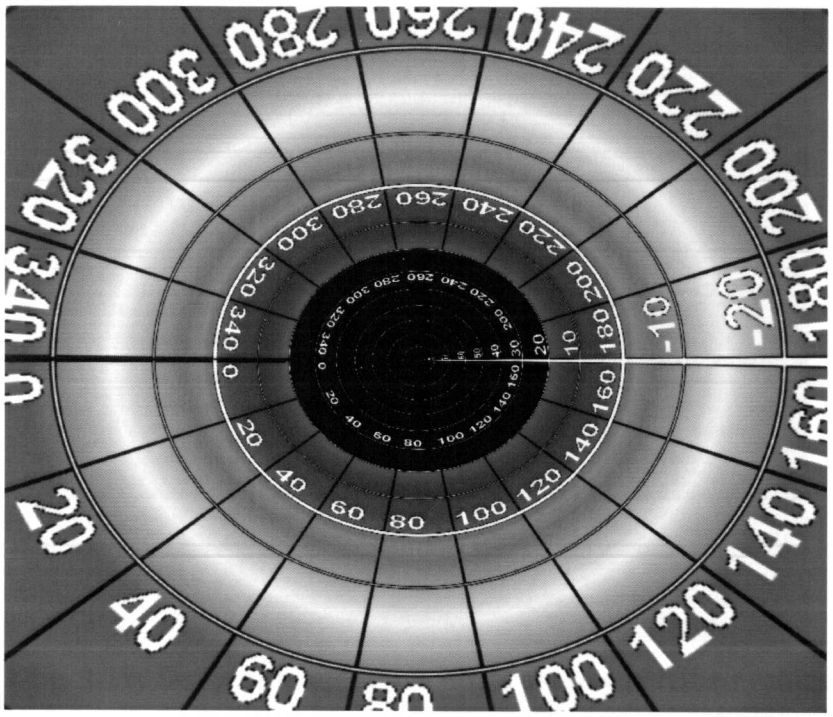
Obr. 2.15: Dopplerův efekt pro pozorovatele letícího rychlostí β=0.2:

Zde se efekt prohlubuje, vznikají zajímavé barevné přechody. Je vidět, že pro čelně nalétávající fotony je posun vln. délky již do modré oblasti spektra – tj. zhruba přes 2/3 viditelného spektra a to i při relativně malé relativistické rychlosti.

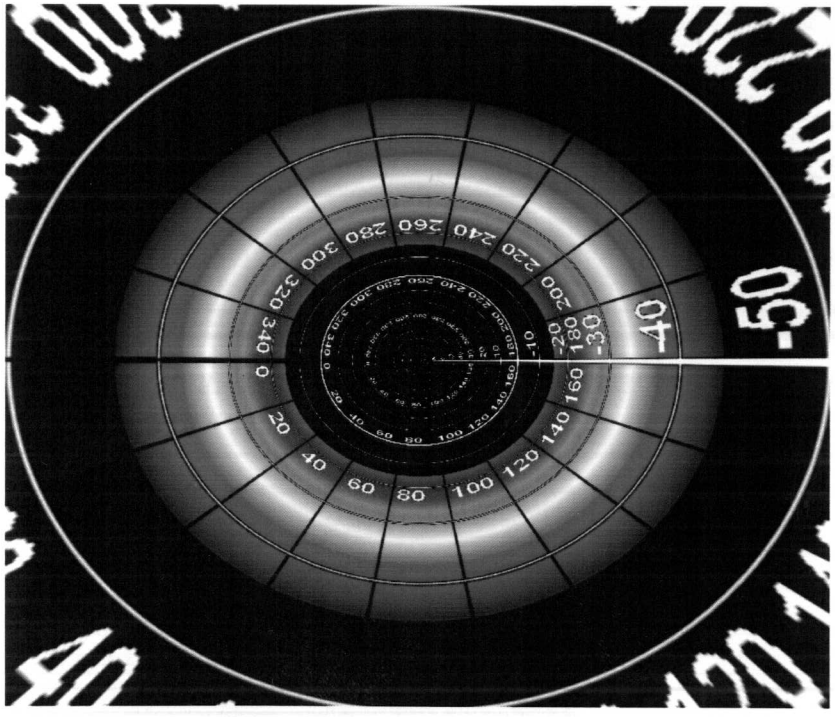


Obr. 2.16: Dopplerův efekt pro pozorovatele letícího rychlostí β=0.4:

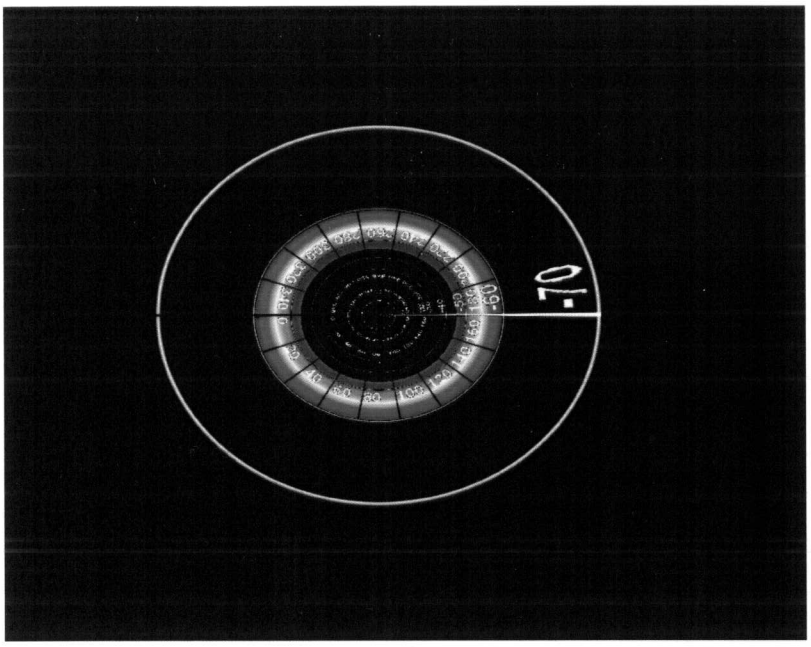
Čelní fotony se již přesunuly do ultrafialové oblasti - pozorovatel je nevidí. V ostatních částech obzoru dochází k prohlubování efektů



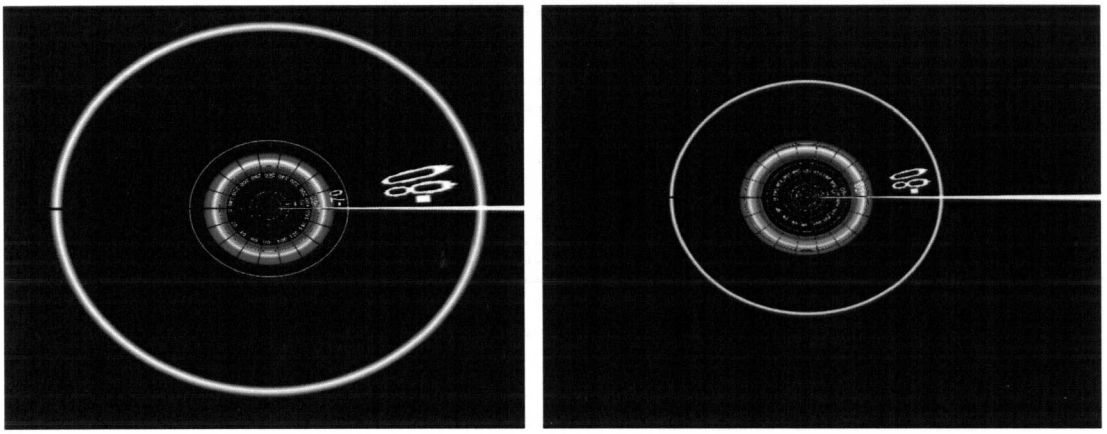
Obr. 2.17: Dopplerův efekt pro pozorovatele letícího rychlostí β=0.55:
Stále větší část čelní oblasti je pro pozorovatele neviditelná.



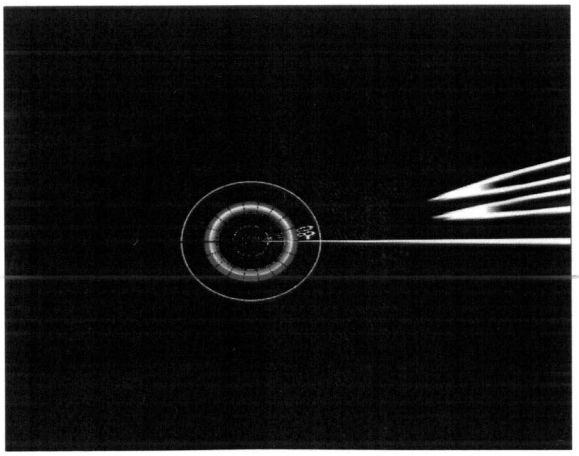
Obr. 2.18: Dopplerův efekt pro pozorovatele letícího rychlostí β=0.7:
Zde je navíc vidět, že ani fotony které pronásledují pozorovatele nejsou již vidět, zde ovšem dochází k rudému posuvu – prodlužování vlnových délek – čili se nacházíme v infračervené oblasti spektra. Fotony, jejichž směr svírá s osou 1 v nečárkované soustavě úhel mezi 40 – 70 ° jsou stále vidět.



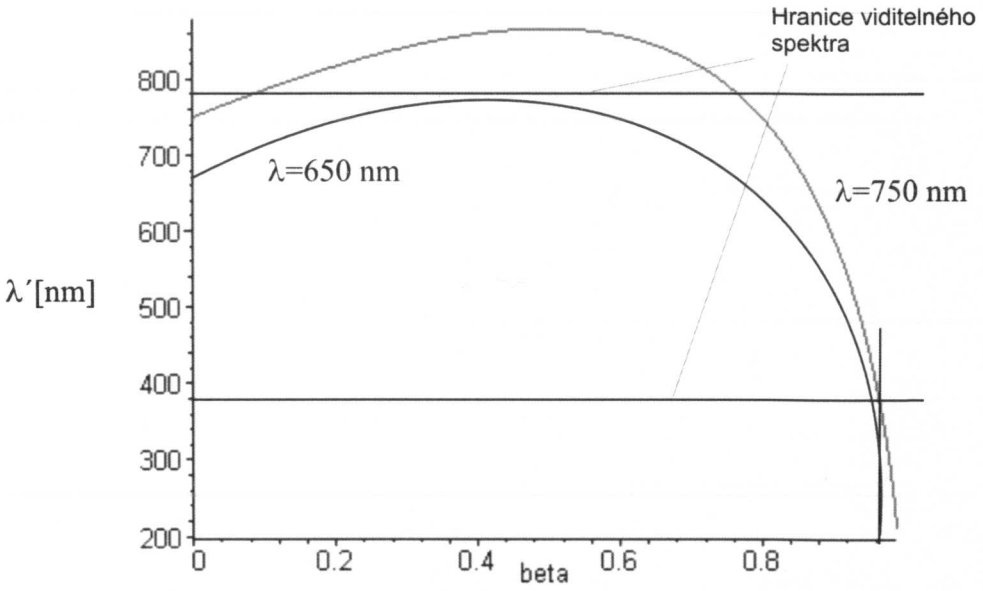
Obr. 2.19: Dopplerův efekt pro pozorovatele letícího rychlostí β=0.9:
Většina glóbu již není vidět, ale některé úhly si udržují své privilegované postavení.



Obr. 2.20: Dopplerův efekt pro pozorovatele letícího rychlostmi β=0.98 (vlevo) a β=0.995 (vpravo)



Obr. 2.21: Dopplerův efekt pro pozorovatele letícího rychlostí β=0.999

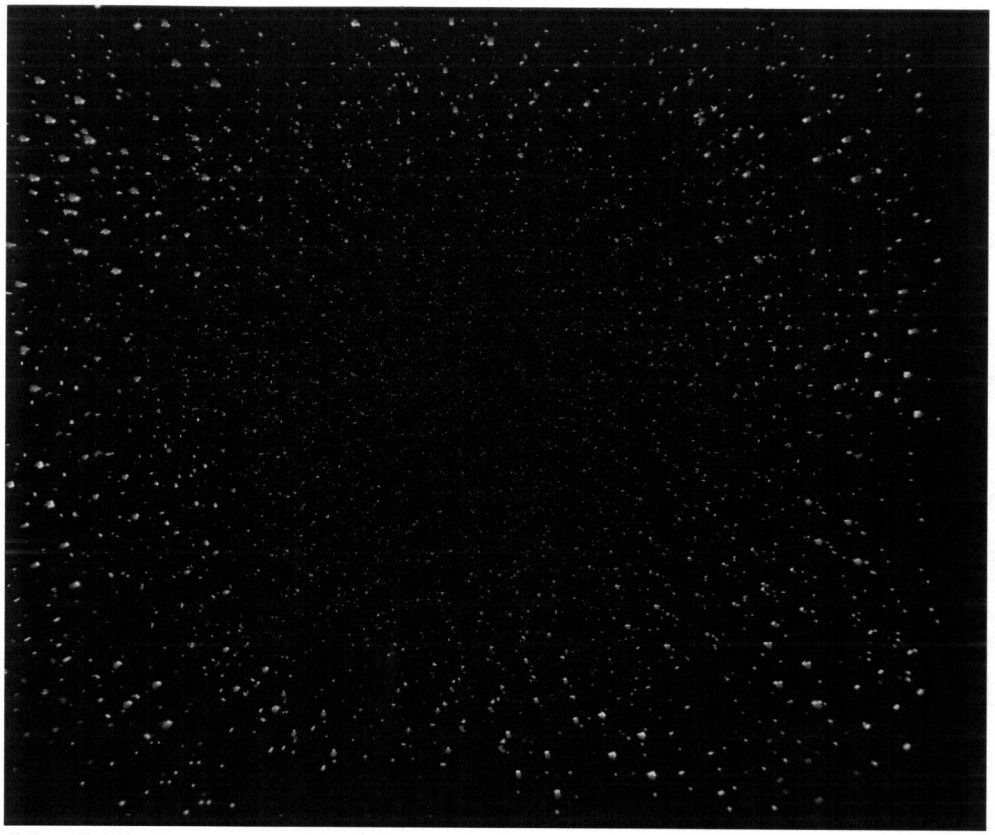


Obr. 2.22: Závislost vlnové délky v soustavě pozorovatele na jeho rychlosti pro nečárkované vlnové délky 650 a 750 nm a $\theta = 30°$.

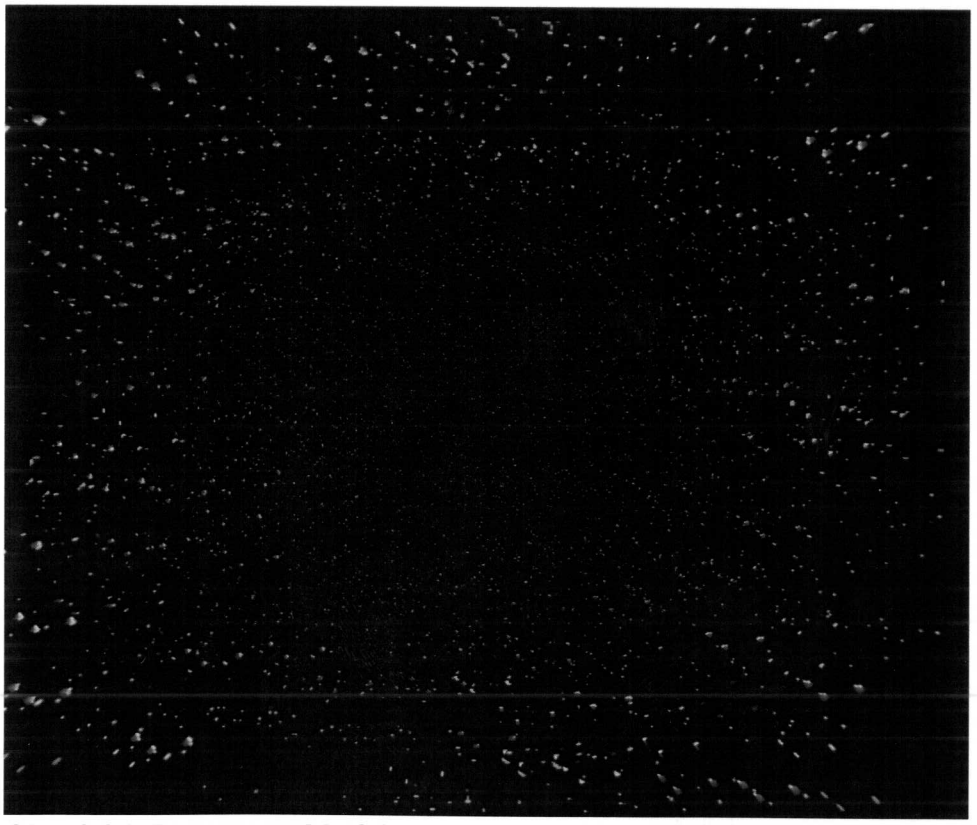
750 nm foton zmizí a při vysoké rychlosti se opět objeví. Vidíme, že foton je viditelný až téměř do rychlosti světla. 650 nm foton je vidět po celou dobu zrychlování.

Zajímavý jev nastává pro fotony jejichž směr svírá s osou 1 v nečárkované soustavě úhel mezi cca 30 – 50°. Fotony z těchto směrů jsou vidět i při velmi vysokých rychlostech – viz obr. 2.19 – 2.21. V těchto oblastech se po počátečním nárůstu vlnové délky (rudý posuv) vlnová délka začíná opět pozvolna zkracovat (modrý posuv) a světlo se tak udrží ve viditelné oblasti i při velmi vysokých rychlostech. Příklad pro konkrétní vlnovou délku a konkrétní úhel je na obr. 2.22.

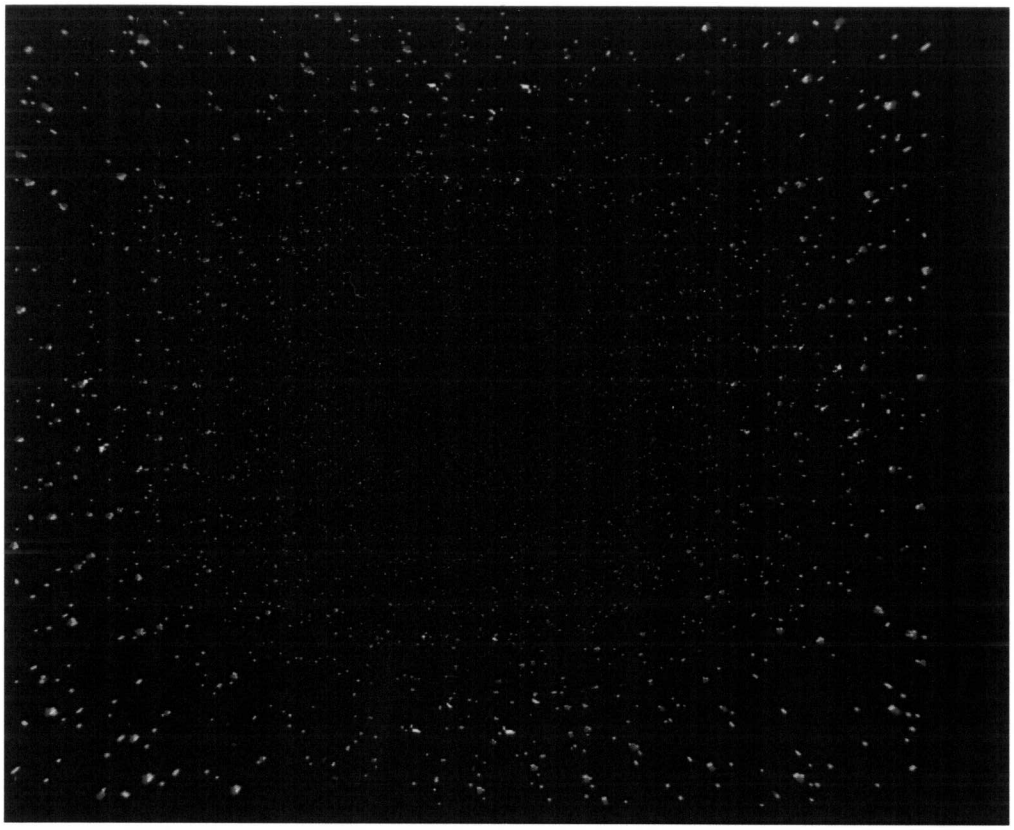
Na obr. 2.23 – 2.29 je vidět, jak by vypadal Dopplerův efekt a aberace pro let vesmírem.



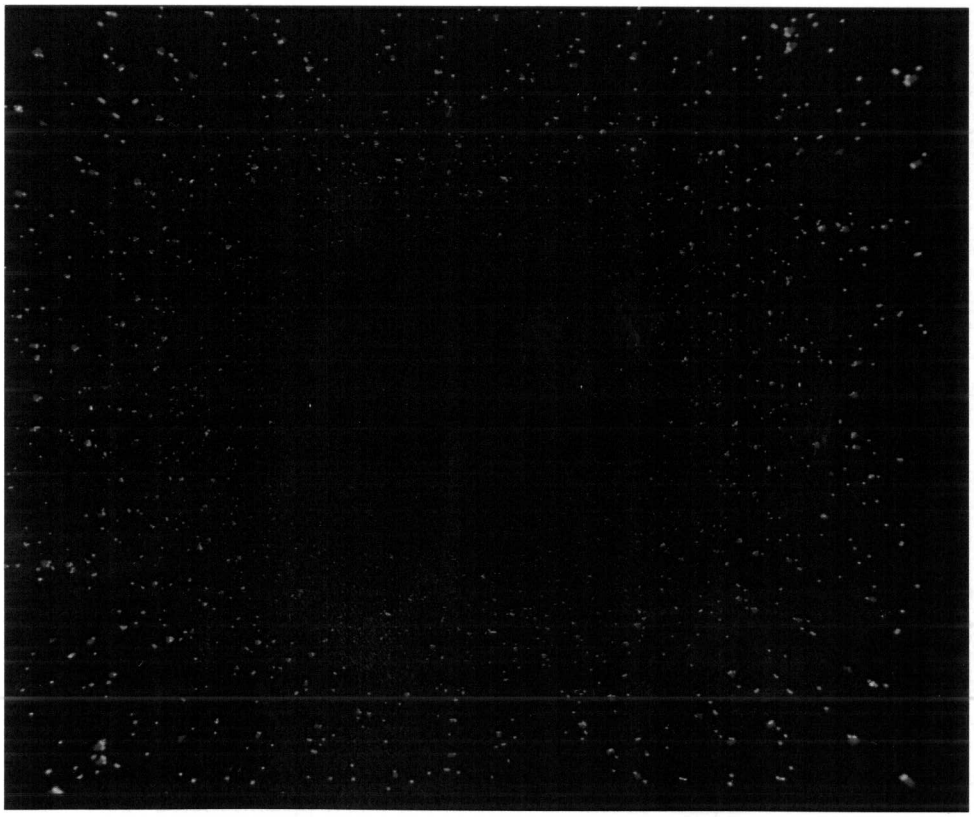
Obr. 2.23: Pozorovatel letící vesmírem rychlostí β=0



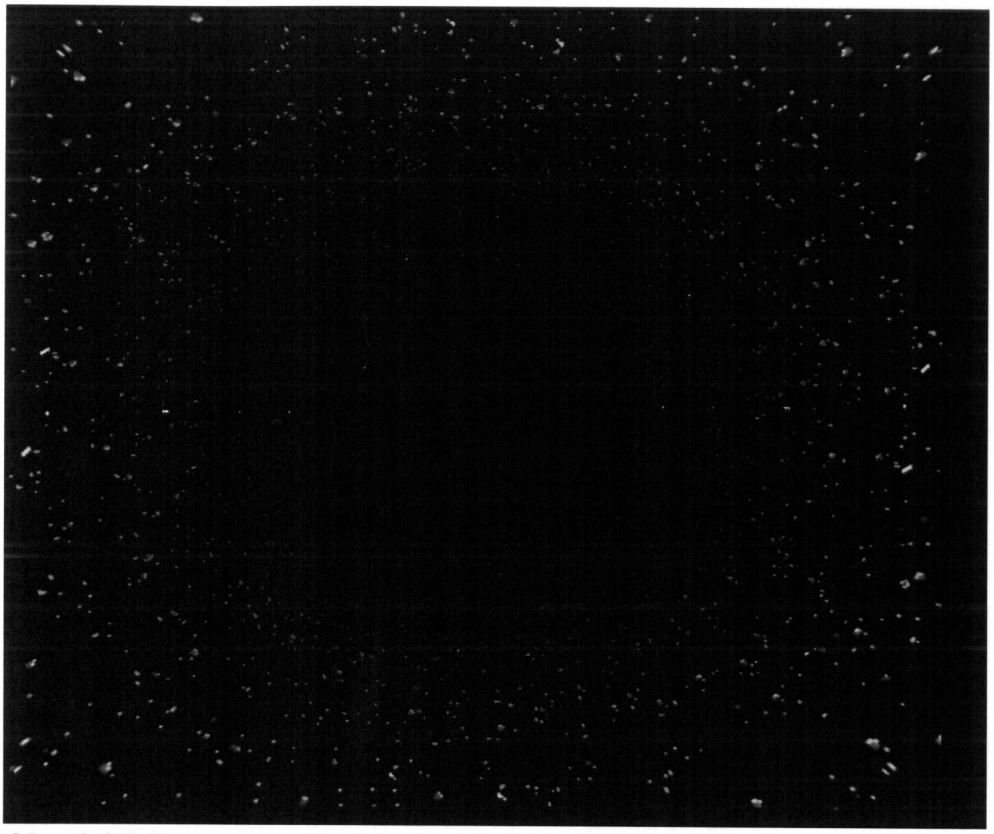
Obr. 2.24: Pozorovatel letící vesmírem rychlostí β=0.1



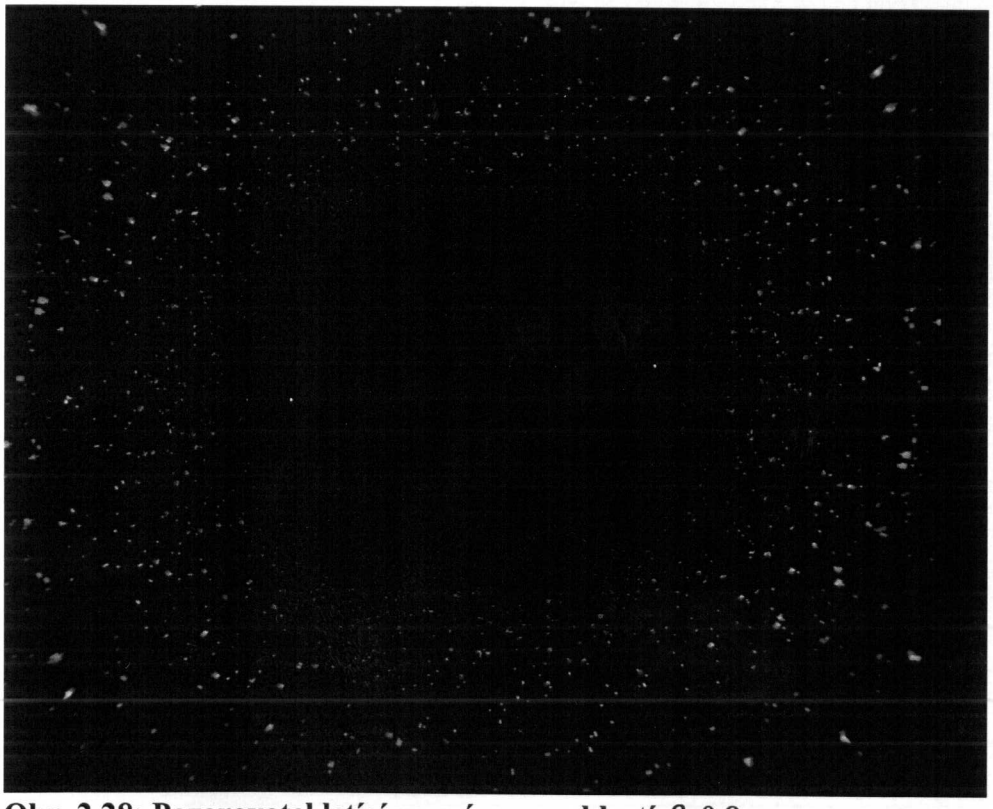
Obr. 2.25: Pozorovatel letící vesmírem rychlostí β=0.2



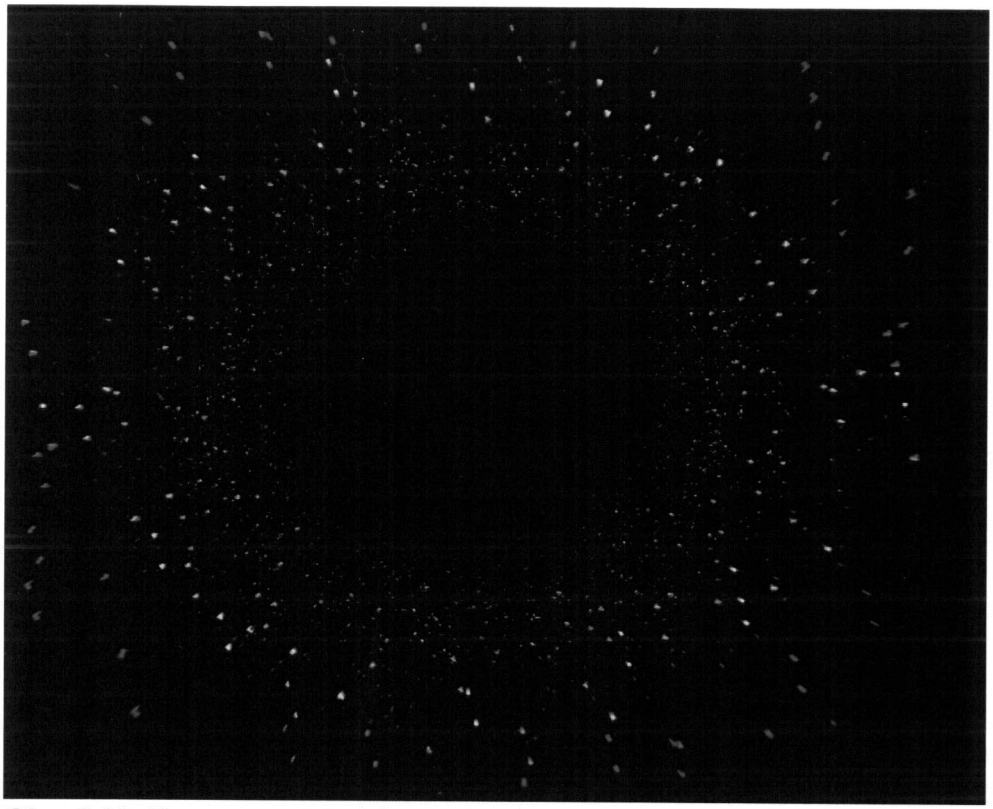
Obr. 2.26: Pozorovatel letící vesmírem rychlostí β=0.4



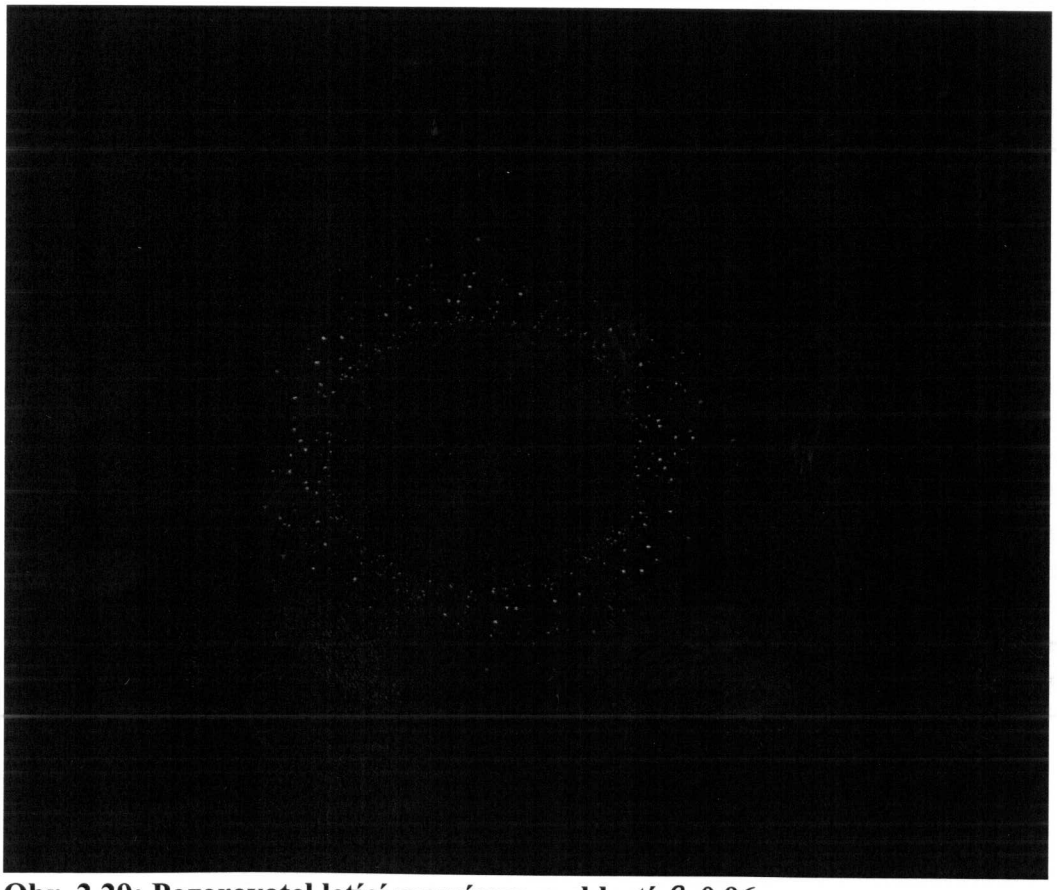
Obr. 2.27: Pozorovatel letící vesmírem rychlostí β=0.6



Obr. 2.28: Pozorovatel letící vesmírem rychlostí β=0.8



Obr. 2.29: Pozorovatel letící vesmírem rychlostí β=0.9



Obr. 2.29: Pozorovatel letící vesmírem rychlostí β=0.96

3. KAPITOLA

Implementace problému. Relativistický ray-tracing

3.1 Stručně o ray tracingu

Ray tracing je metoda používaná v počítačové grafice k zobrazování 3D scenérií. Jedná se o metodu, při které se zpětně sleduje paprsek vycházející s optického systému – např. oka nebo kamery a zjišťuje se, na které místo v okolním prostoru by dopadl. Jedná se vlastně o opačný proces než se v realitě ve skutečnosti děje – paprsek vychází z nějakého místa na okolní scenérii a dopadá do optického systému. Výhoda tohoto přístupu je zřejmá – programátora nezajímají všechny paprsky nebo fotony, které prostorem projdou, ale jen specifické paprsky, které dopadnou do optického systému. Tím se celý problém zobrazení okolní scenérie značně zjednoduší.

Relativistický vs. obyčejný ray tracing
Relativistický ray tracing je lehkou modifikací předchozího přístupu. Jde o to, vzít v úvahu relativistické efekty pro pozorovatele pohybujícího se relativistickou rychlostí. Směry šíření paprsku/fotonu jsou různé v různých vztažných soustavách – viz *relativistická aberace* – předchozí oddíl. Stanovím si směr paprsku v soustavě pohybujícího se pozorovatele – provedu transformaci do globální soustavy okolního vesmíru a mám směr kterým se bude šířit paprsek/foton v soustavě vůči které se pozorovatel pohybuje rychlostí β. Pak si spočtu na jaké místo okolní scenérie foton dopadne a při obrácení chodu času v celém vesmíru vím z jakého místa vylétl foton, který dopadl do optického systému letícího pozorovatele v jeho klidové inerciální soustavě ve mnou stanoveném směru.

3.2 OpenGL

OpenGL je průmyslový standard, který označuje multiplatformární rozhraní (API), které slouží k tvorbě aplikací pokročilé počítačové 3D grafiky. Implementace OpenGL existují pro prakticky všechny počítačové platformy, na kterých je možno vykreslovat grafiku. Kromě implementací hardwarových, které jsou přítomny přímo na grafické kartě, existují i implementace softwarové, které umožňují provozovat OpenGL i na hardwaru, který ho sám o sobě nepodporuje. Většina moderních grafických karet nějakou implementaci OpenGL obsahuje. OpenGL se ovládá pomocí volání funkcí a procedur, kterých je cca 250. Tato práce byla psána v *Microsoft Visual C/C++ 6.0,* který obsahuje vlastní knihovnu OpenGL. Po zadání direktiv preprocesoru

```
#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glaux.h>
```

jsou přístupné veškeré funkce OpenGL a vše ostatní již řeší překladač.

3.3 Mapování trojúhelníků

Z projekčního plátna kamery probíhá tracování paprsku skrz objektiv a zjišťuje se, kam paprsek dopadne na sféře nekonečného poloměru, která obklopuje kameru a reprezentuje okolní vesmír / hvězdnou oblohu. Nejzákladnější jednotky, na které lze rozdělit projekční plátno na kameře i sféru reprezentující okolní vesmír jsou trojúhelníky. Toho lze s výhodou využít, protože tracovat paprsek pro každý pixel na plátně zvlášť by bylo výpočetně příliš náročné. Místo toho lze plátno rozdělit na malé trojúhelníčky a provést relativistický ray tracing pouze pro jejich vrcholy. Pro ty se pak spočte, na jaké souřadnice na nekonečné sféře reprezentující okolní vesmír tyto paprsky dopadnou. Původní trojúhelník z plátna zůstane i na sféře stále trojúhelníkem, byť třeba s jiným poměrem délek stran. Body, které leží uvnitř trojúhelníku na plátně budou ležet uvnitř velkého trojúhelníku na sféře – viz obr. 3.1. Pak lze s úspěchem provést projekci trojúhelníku ze sféry na trojúhelník na plátně. Hvězdná mapa okolního vesmíru je reprezentována bitmapou. Projekční plátno kamery tvoří výstup na obrazovku – tzv. framebuffer do kterého OpenGL maluje. Pokud znám souřadnice na bitmapě – spočtu si odpovídající souřadnice na nekonečné sféře a pak mohu zavolat funkci OpenGL která mi namapuje trojúhelník na bitmapě na trojúhelník na plátně / obrazovce. Tento postup se nazývá *texture-mapping – mapování textur*. Detaily tohoto mapování si řeší OpenGL samo, ale mělo by docházet k co nejlepšímu zachovávání proporcí. Při volbě dostatečně malých počátečních trojúhelníků na plátně odpovídá vizualizace s dostatečnou přesností skutečnosti. Ray tracing pro celé plátno je realizován pomocí 2 for cyklů pro osu x a y na plátně s pevně nastavenou volbou kroku. Funkce, která mapuje trojúhelník může v případě potřeby rekurzivně volat sebe samu, čímž dochází k zpřesnění vizualizace. Hloubku rekurze lze nastavit číselným parametrem od 1 do N. Rekurze se využívá ve středu obrazovky, kde je potřeba jemnější volby kroku, jinak je obraz ve středu rozmazaný a křivý. Funkce programu, která zajišťuje mapování trojúhelníků se jmenuje *DrawTriangle* a lze ji nalézt ve zdrojovém kódu programu v příloze této práce. Tato funkce je „srdcem" celého programu.

3.4 Vlastní ray tracing

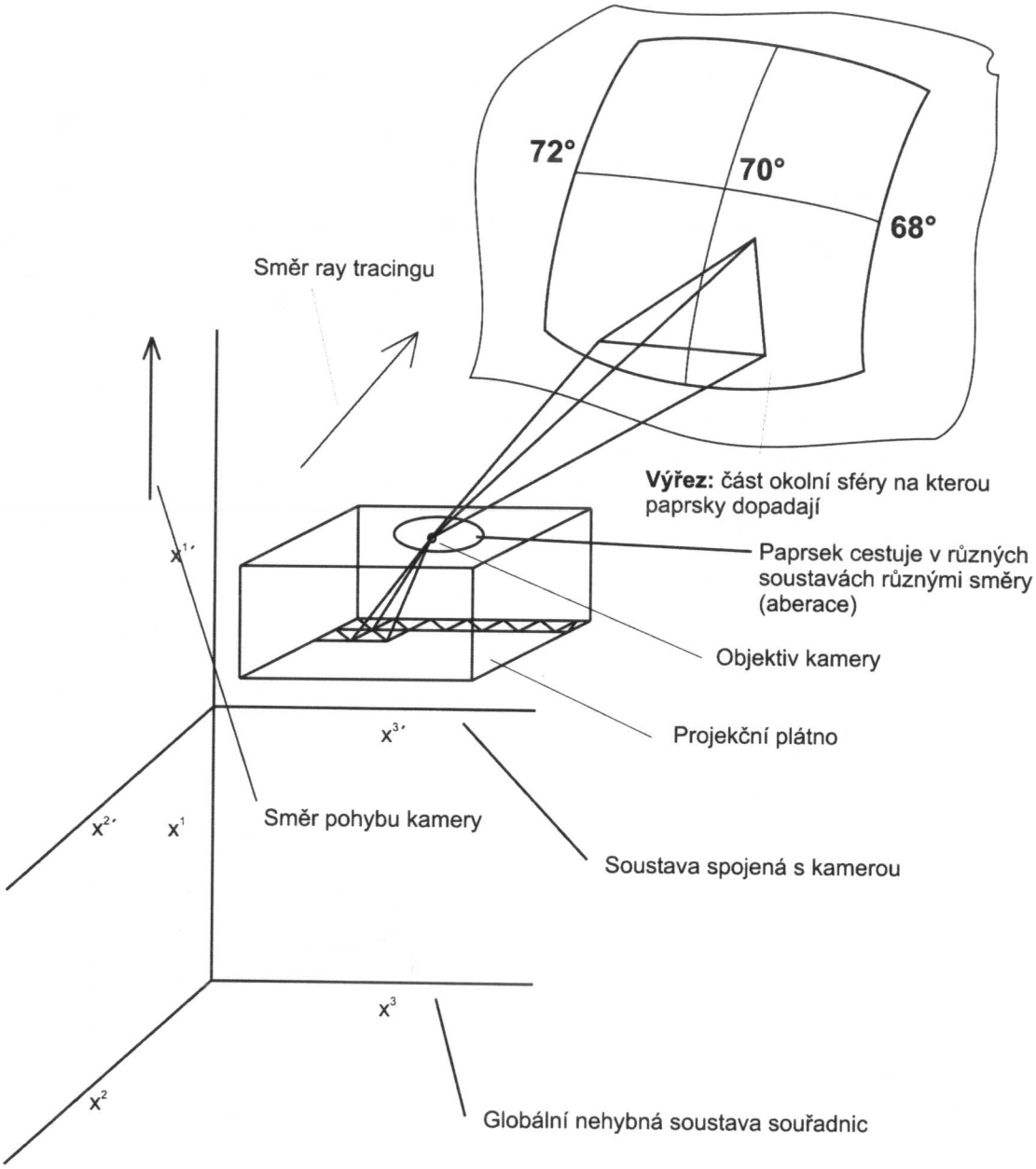
Kamera má projekční plátno velikosti ***x.y*** pixelů na které dopadají paprsky. Při ray-tracingu vyjdu z konkrétního bodu na plátně a s pomocí jednoduchých geometrických vztahů si spočtu směr fotonu, který spojuje konkrétní bod na plátně s objektivem kamery. Nechť ***a*** a ***b*** jsou vzdálenosti bodu na plátně od středu kamery a ***c*** je výška kamery – viz obr. 3.2. Pak platí

$$d = \sqrt{a^2 + b^2} \qquad e = \sqrt{c^2 + d^2} \qquad \cos\alpha = \frac{d}{e} \tag{3.4.1}$$

Pro foton platí

$$E = p \tag{3.4.2}$$

kde E je jeho energie a p hybnost – viz oddíl 2.5 – a směr hybnosti je shodný se směrem šíření fotonu – tj. směrem daným úsečkou **e**, která spojuje bod na plátně, ze kterého foton vylétává s objektivem kamery – viz obr 3.2. Energii si mohu zvolit



Obr. 3.1: Relativistický ray tracing

Trojúhelník z plátna kamery zůstává i po projekci na sféru trojúhelníkem. Na obrázku je taktéž vidět jak je plátno kamery rozdělené na malé trojúhelníky, které se poté projektují na sféru. Výsledkem projekce celého plátna je souvislé pokrytí části sféry. Je taktéž vidět, že fotony cestují v čárkované soustavě jiným směrem než v nečárkované. Není sice úplně správné malovat do jednoho obrázku cesty fotonu ve dvou různých soustavách, ale pro názornost jsem se nakonec rozhodl použít tento přístup. Velký trojúhelník na sféře se pak v OpenGL namapuje na malý trojúhelník na plátně – texture mapping.

---

libovolnou. Směr šíření fotonu to neovlivní. Pro projekci velikosti hybnosti do roviny plátna p_d pak bude platit

$$p_d = E.\cos\alpha \tag{3.4.3}$$

Pro složky 4-hybnosti fotonu pak bude platit

$$P^{0'} = E \qquad P^{2'} = \cos\beta . p_d \qquad P^{3'} = \cos\gamma . p_d \tag{3.4.4}$$

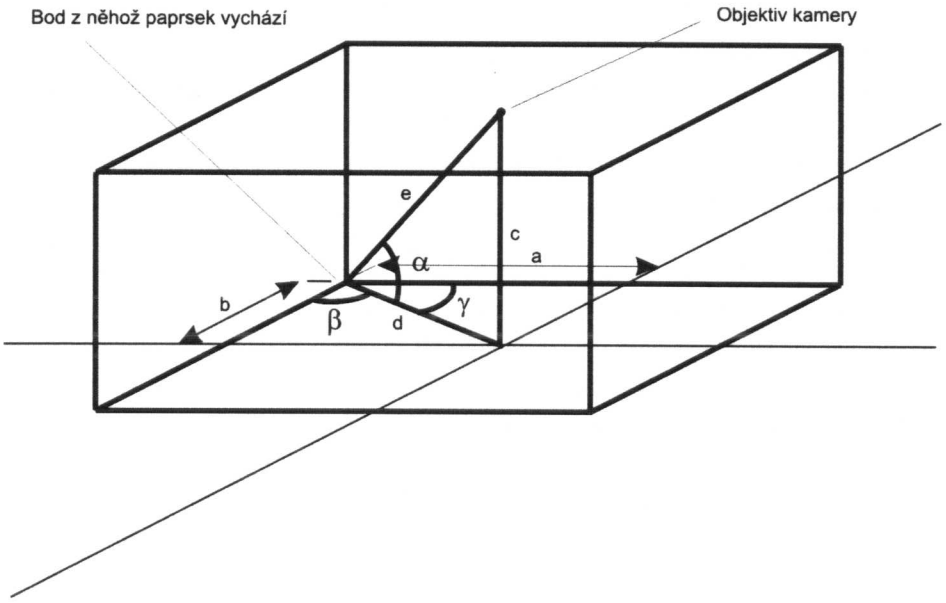
$$P^{1'} = \sqrt{(P^{0'})^2 - (P^{2'})^2 - (P^{3'})^2} \tag{3.4.5}$$

Funkce programu, která zajišťuje právě popsaný postup se jmenuje *ComputeImpulse* a lze ji nalézt ve zdrojovém kódu programu v příloze této práce.

Pro 4-hybnost v nečárkované soustavě platí

$$P^{\nu} = \Lambda^{\nu}{}_{\mu'} P^{\mu'} \tag{3.4.6}$$

kde $\Lambda^{\nu}{}_{\mu'}$ je matice (2.4.7). Doba po kterou je pohyb simulován je velmi krátká v porovnání s časem, který by potřeboval foton na cestu k jednotlivým hvězdám, pokud by se prováděl ray tracing v reálném čase. Taktéž vzdálenost, kterou pozorovatel urazí po dobu simulace, je zanedbatelná vzhledem ke vzdálenostem k okolním hvězdám. Z těchto důvodů je použit zjednodušený model – kamera se po celou dobu pohybu nachází ve středu sféry s nekonečným poloměrem a z P^{ν} se pomocí vztahu (2.5.11) spočte směr šíření fotonu v nečárkované soustavě a pak se



Obr. 3.2: Stanovení směru pohybu fotonu, který vychází z daného místa na plátně

jednoduše spočítá, kam by dopadl na nekonečné sféře obklopující kameru – tj. dvojici úhlů θ a ϕ ve sférických souřadnicích. Funkce, která výpočet v programu zajišťuje, se jmenuje *WhereOnSphere* a lze ji nalézt ve zdrojovém kódu v příloze. Sféra je reprezentována čtvercovou bitmapou, čili na konci funkce ještě musí přepočítat souřadnice θ a ϕ do souřadnic vlastních OpenGL, kde je to značeno tak, že levý horní roh má souřadnice 0,0 a pravá dolní 1,1.
Tento postup se provede pro všechny 3 body trojúhelníku na plátně – tím se získají souřadnice třech bodů na sféře, a pak se provede *texture mapping* tak jak byl popsán výše.

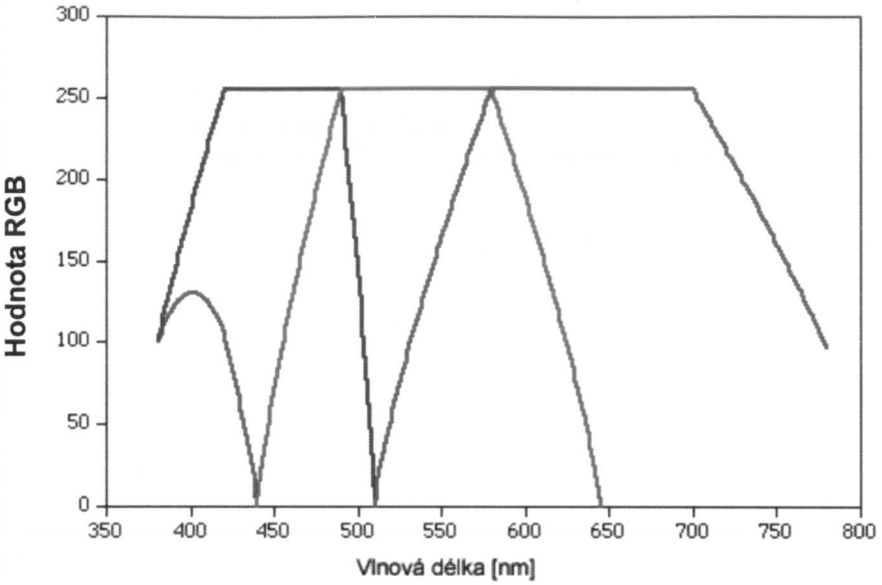
Objevuje se domnělá námitka, že pokud se použije vztah (3.4.6), tak mu odpovídá transformace (2.6.1) pro směr fotonu a úhel, který svírá směr šíření fotonu v nečárkované soustavě s osou x', není větší než v čárkované, jak bychom na první pohled čekali pro aberaci při dopředném pohybu (viz oddíl 2.6 a obr. 2.4.), ale naopak menší – tj. pozorovatel by při pohledu vpřed viděl, jak objekty cestují za něj a nikoliv před něj, jak by to při aberaci mělo správně být. Je nutné si ovšem uvědomit, že ray tracing probíhá – viz. oddíl 3.1 – proti proudu času. Tj. při ray tracingu běží čas v celém vesmíru jakoby „obráceně". Pokud pak otočíme čas v celém vesmíru a pustíme film ve smyslu „správného" plynutí času pak pozorovatel i fotony cestují opačným směrem a pozorovatel vidí vesmír „správně" tedy tak, jak to odpovídá relativistické aberaci – celý vesmír se při pohledu vpřed „stěhuje před pozorovatele".

3.5 Implementace Dopplerova jevu

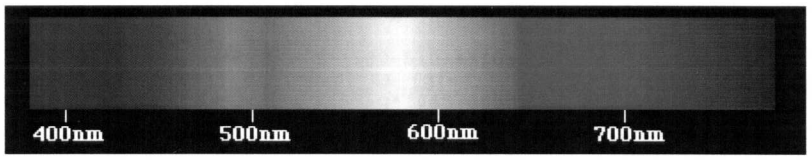
Realizace Dopplerova jevu v počítači skýtá mnohá úskalí. Barvy v počítači jsou uloženy pomocí barevných palet – např. RGB, CMYK a je třeba nalézt nějakou spojitost s vlnovými délkami. Světlo dopadající do oka pozorovatele není monochromatické, ale obsahuje více vlnových délek, které vytvářejí výsledný barevný dojem. Palety barev v počítači nemusejí odpovídat a neodpovídají tomu, jak obraz z dopadajících vlnových délek vytváří lidské oko. Tím vzniká problém s reprezentací barev pro tuto simulaci. Buď by se muselo vytvořit pole, do kterého by se uložila pro každý pixel informace o všech vlnových délkách na kterých vyzařuje a pak nalézt nějaký způsob jak tuto informaci převést do RGB barev. Tím bychom se ovšem připravili o komfort pouhého vložení libovolné bitmapy / textury (např. s pěkným obrázkem mlhoviny) do programu s tím, že o realizaci vizualizace Dopplerova posuvu se postará sám program. Použití opačného postupu – tj. vypočítávání vlnových délek z RGB palety by sice tento problém odstranilo, ale zdá se neúměrně složitým a zřejmě by vyžadovalo zdlouhavé studium této problematiky. Na druhé straně astronomové s úspěchem používají lineární aproximace jednotlivých vln. délek viditelného spektra pomocí RGB barev. Kdybychom tedy použili bitmapu, na které každý pixel má barvu odpovídající pouze jediné vlnové délce, mohli bychom jednoduše převádět RGB barvy na vlnové délky a zpětně tyto převádět na RGB barvy. Právě tohoto přístupu se autor této práce rozhodl použít. Efekt sice není nijak oslňující, ale dává velice dobrou ilustraci toho co se děje při Dopplerově efektu v rámci STR. Mírnou nevýhodou je, že nelze simulovat bílou barvu (všechny vlnové délky).

Lineární model RGB

Model, ze kterého jsem vycházel[1)] je popsán na obr. 3.3. Vzhled spektra, který se takto získá, je vidět na obr. 3.4. Tato aproximace dává vzájemně jednoznačné zobrazení z prostoru vlnových délek do prostoru RGB. Jak je vidět, jedna barva se vždy vytáhne na maximální hodnotu (255), jedna se položí rovna nule a pro zbývající se použije lineární funkce. Parabola na obr. 3.3 je rovněž nahrazena přímkou. Kód funkcí, které zajišťují převod je pak poměrně triviální:



Obr. 3.3 Lineární aproximace vlnových délek pomocí RGB barev
Maximální hodnota barvy je 255, minimální 0.



Obr. 3.4: Vzhled spektra, jak ho vytvoří lineární aproximace pomocí RGB barev
Je vidět, že vzhled je celkem věrný.

```
void WlToRGB(tripole rgb, double lambda) {
        double r=0,g=0,b=0;
        if ((lambda >= 380) && (lambda < 440))
        {
                r=-1*(lambda-440)/(440-380);
                g=0;
                b=1;
        }
        if ((lambda >= 440) && (lambda < 490))
        {
                r=0;
                g=(lambda-440)/(490-440);
                b=1;
        }
        if ((lambda >= 490) && (lambda < 510))
        {
                r=0;
                g=1;
                b=-1*(lambda-510)/(510-490);
        }
        if ((lambda >= 510) && (lambda < 580))
        {
                r = (lambda-510)/(580-510);
                g = 1;
                b = 0;
        }
        if ((lambda >= 580) && (lambda < 645))
        {
                r = 1;
              g = -1*(lambda-645)/(645-580);
                b = 0;
        }
        if ((lambda >= 645) && (lambda < 700))
```

```
        {
                r=1;
                g=0;
                b=0;
        }
        if ((lambda >= 700) && (lambda <= 780))
        {
                r = -(lambda-810)/(810-700);
                g = 0;
                b = 0;
        }
        if(lambda==0)
        {
                r=0; g=0; b=0;
        }
        if (lambda == 1000)
        {
                r=1; g=1; b=1;
        }
        r=r*255;
        g=g*255;
        b=b*255;
        rgb[0]=r;
        rgb[1]=g;
        rgb[2]=b;
}

double RGBToWl(tripole rgb) {
        double r,g,b;
        double lambda=0;
        r = rgb[0]/255;
        g = rgb[1]/255;
        b = rgb[2]/255;
        if ((g==0.0) && (b==1.0))
                lambda = 440-r*(440-380);
        if ((r==0) && (b==1))
                lambda = g*(490-440)+440;
        if ((r==0) && (g==1))
                lambda = 510 - b*(510-490);
        if ((g==1) && (b==0))
                lambda = r*(580-510)+510;
        if ((r==1) && (b==0) && (g!=0))
                lambda = 645 -g*(645-580);
        if ((r==1) && (g==0) && (b==0))
                lambda = 675;
        if ((r!=1) && (g==0) && (b==0))
                lambda = 810 - r*(810-700);
        if ((r==0) && (g==0) && (b==0))
                lambda=0;
        r=r*255;
        g=g*255;
        b=b*255;
        if ((r>216) && (g>216) && (b>216))
                lambda=1000;
        return(lambda);
}
```

Máme tedy způsob jak přecházet mezi barvami a vlnovými délkami. Je třeba ovšem získat přístup k datům obrázku, ze kterého je vyrobena textura, která se v programu používá, aby bylo možné spočítat vlnovou délku. K tomuto účelu se výborně hodí grafický formát RAW, což je holá bitmapa, bez hlaviček obsahující pouze informace o barvách. Každý pixel je reprezentován 3 byty po jednom na každou R,G,B barvu – hodnota 0 – 255. Pak stačí soubor s obrázkem otevřít a jeden po druhém si pixely přečíst. Pro každý pixel se pak spočte, pod jakým úhlem ho foton opouští, pokud chce dopadnout do objektivu kamery. Připomeňme, že čtvercová bitmapa odpovídá nekonečné sféře – viz kapitola o implementaci ray tracingu. Z koordináty y na bitmapě lze potom vypočítat úhel θ s pomocí elementárních trigonometrických vztahů. Připomenu ještě, že úhel θ ve vztahu (2.7.2) je úhel, který svírá směr letu fotonu s osou 1. Jakmile tedy známe úhel, je možné si podle vztahů (2.7.1) a (2.7.2) spočíst vlnovou délku fotonu, který vidí pozorovatel / kamera ve své inerciální soustavě – vlnovou délku v nečárkované soustavě známe – viz výše. Následně se pomocí funkce `WlToRGB` spočítá barva, tak jak ji uvidí pozorovatel a jediné co zbývá, je to někam zapsat. Pro tento účel se vytvoří textura, což stejně vyžaduje Open GL, výhodou je, že neztratíme původní informaci o výchozích vlnových délkách fotonů,

což je informace, kterou budeme potřebovat pro výpočet Dopplerova posuvu pro ostatní rychlosti. Vytváření vlastních textur namísto používání hotových bitmap je jedna z výhodných funkcí, které nabízí OpenGL. Zjednodušeně řečeno se někde alokuje dynamická paměť, tam se postupně uloží informace – např. v RGB paletě a pak se řekne programu – „Tohle bude textura s číslem 1." Nevýhodou je, že pro každou rychlost musíme provádět výpočet nové textury, což vyžaduje výpočetní čas, ale i tak je výpočet vcelku rychlý a simulaci je možné pustit dokonce v reálném čase. V podobě kódu vypadá vše následovně:

```
int DopplerTexture ( char *filename, Uk_TEXTURE_buffer)
{
        FILE *f, *g;
        int i,j,k =0;
        int sirka = buffer->sirka * buffer->format;
        unsigned char *p = NULL;
        tripole rgb;
        double lambda, lambdac;
        double beta=tanh(Tau), gamma = cosh(Tau), doppler, phi;

        f = fopen(filename, "rb");


        for( i = buffer->vyska-1; i >= 0 ; i-- )  // Procházení obrázků pomocí for
cyklů - raw formát je uložen obráceně, řádky jsou číslovány odspodu
                {
                        phi = pi+(pi*i)/1024;     // Zde vypocet uhlu pod jakym foton
opousti pixel
                        doppler  =  1/(gamma*(1-beta*(-1*cos(phi))));  // Zde vypocet
Dopp. posuvu
                        p = buffer->pixels + (i * sirka );
                        for ( j = 0; j < buffer->sikra ; j++ )
                        {
                                for ( k = 0 ; k < buffer->format-1 ; k++)
                                {

                                        rgb[k] = fgetc(f);
                                }

                                lambda=RGBToWl(rgb);
                                if (lambda == 1000)
                                        lambdac = 1000;
                                else
                                        lambdac = lambda*doppler;
                                WlToRGB(rgb, lambdac);

                                for ( k = 0 ; k < buffer->format-1 ; k++, p++)
                                        *p = rgb[k];

                                }
                fclose(f);
        }
        return done;    // Vraci pocet nactenych bytů

1                                                                          }
```

[1]) Tento model jsem převzal z [4], odkud jsem si taktéž vypůjčil obrázek 3.3

PŘÍLOHA A

Zdrojový kód programu

Program byl psán v jazyce C/C++, použitý překladač byl *Microsoft Visual C/C++ 6.0*. Program běží pod Microsoft Windows XP jako standardní aplikace Win32, čili je potřeba zajistit funkce pro vytvoření a správu okna a vykreslování grafiky v okně. V této části je pouze zdrojový kód pro vizualizaci, funkce pro správu oken jsou vynechány, protože se jedná o obecně známé a dostupné informace, které pro účely této práce nejsou zajímavé. Při psaní funkcí pro správu oken, což je samo o sobě dosti složitá problematika, jsem z části vycházel z Open GL tutoriálů v [2], kód pro vizualizaci je čistě mým dílem.

Tvorba videa je poté realizována pomocí knihovny Avigenerator.h která zajišťuje výstup toho, co se děje na obrazovce do videosouboru. Veškerý zdrojový kód zde uvedený je původní prací autora této práce. Běh programu začíná voláním funkce *StartHole*. Funkce, které již byly uvedeny v předchozím textu zde jsou vynechány.

Zdrojový kód programu

```
#include <stdio.h>
#include <math.h>
#include <windows.h>
#include <gl\gl.h>  // Header File For The OpenGL32 Library
#include <gl\glu.h> // Header File For The GLu32 Library
#include <gl\glaux.h>      // Header File For The Glaux Library
#define CAMSIZE 40
#define PlanckConst 6.6260693E-34

int x2, y2; //
GLuint texture[2];
static float XSIZE=1000;
static float YSIZE=1000;
static float CAMHEIGHT=10;
double pix_size;
double PhotonEn,Tau;
static double PhotonFreq=6E-7;
bool Recursion;
typedef double vect [4];
typedef vect matrix [4];

void ComputeImpulse(int x, int y, vect PhotonImpulse);  // pokud neni mineno
jinak, jsou vsechny vektory 4vektory

void copy(vect r,vect r0);
void TransformVector(vect Vector,matrix TrMatrix);
void WhereOnSphere(vect Impulse, GLfloat *pTheta, GLfloat *pPhi);
void VectCopy(vect a,vect b);
double ScalarProduct(vect x,vect y);
void DrawTriangle(int xt1,int yt1,int xt2,int yt2,int xt3,int yt3, int
RecLev, bool TEST, bool OnTop);
void StartHole(void);
int DrawGLScene(GLvoid);

void copy(vect r,vect r0) {
```

```
        int i;
        for ( i = 0; i < 4; i++ )
                r[i]=r0[i];
}

void ComputeImpulse(int x, int y, vect PhotonImpulse) {
        extern int x2,y2;
        extern double pix_size;
        int i,j;
        double xc, yc, ch ;
        extern double PhotonEn;
        double alpha, beta,  gamma, p_pro,vsize, p_pro_l,t;
        i=(x2-x);
        ch=CAMHEIGHT;
        PhotonEn=PlanckConst/PhotonFreq;
        j=-(y2-y);
        t=0;
        xc=(i*pix_size); yc=(j*pix_size); // Tyto souradnice jsou v cm
        vsize=sqrt(pow(xc,2)+pow(yc,2)+pow(ch,2)); // Spojnice bodu na platne
a vrcholu kamery
        p_pro=sqrt(pow(xc,2)+pow(yc,2));
        alpha=acos(p_pro/vsize);
        p_pro_l=PhotonEn*cos(alpha); // Slozka 4-hybnosti v rovine detektoru
        beta=acos(xc/p_pro);
        gamma=acos(yc/p_pro);
        PhotonImpulse[0]=PhotonEn;
        PhotonImpulse[1]=cos(beta)*p_pro_l;
        PhotonImpulse[2]=cos(gamma)*p_pro_l;
        PhotonImpulse[3]=sqrt(pow(PhotonEn,2)-pow(PhotonImpulse[1],2)-
pow(PhotonImpulse[2],2));
}

void DrawTriangle(int xt1,int yt1,int xt2,int yt2,int xt3,int yt3, int
RecLev, bool TEST, bool OnTop) {
        int i;
        extern double Tau;
        double zs;
        static double phisize=1;
        static double thetasize=1;
        extern bool Recursion;
        div_t div_result;
        double xrel, yrel;
        vect PhotonImpulse;
        matrix TrMatrix;
        GLfloat phi [3];
        GLfloat theta [3];
        GLfloat xs=XSIZE;
        GLfloat ys=YSIZE;
        GLfloat xtect [3];
        GLfloat ytect [3];
        GLfloat radius;
        int xp1,xp2,xp3,yp1,yp2,yp3;
        GLfloat side [3];
        struct coord2D {
                GLfloat x;
                GLfloat y;
        } ;
        struct coord2D coord [3];
        coord[0].x=xt1; coord[0].y=yt1;
        coord[1].x=xt2; coord[1].y=yt2;
        coord[2].x=xt3; coord[2].y=yt3;

        RecLev--;

        TrMatrix[0][0]=cosh(Tau);       TrMatrix[0][1]=0;       TrMatrix[0][2]=0;
TrMatrix[0][3]=-1*sinh(Tau);
        TrMatrix[1][0]=0;           TrMatrix[1][1]=1;           TrMatrix[1][2]=0;
TrMatrix[1][3]=0;
```

```
      TrMatrix[2][0]=0;          TrMatrix[2][1]=0;          TrMatrix[2][2]=1;
TrMatrix[2][3]=0;
      TrMatrix[3][0]=-1*sinh(Tau);    TrMatrix[3][1]=0;    TrMatrix[3][2]=0;
TrMatrix[3][3]=cosh(Tau);

      for ( i = 0; i < 3; i++ ) {  // Tady si uz jenom nanormuju souradnice
na jednicku
            xrel=coord[i].x/xs; // Puvodni souradnice posilane funkci
            yrel=coord[i].y/ys; //
            xtect[i]=xrel*phisize;     // Transformace do souradnic GL 0..1
            ytect[i]=yrel*thetasize;
      }

      if  (TEST)  radius=sqrt(pow((xtect[2]-0.5),2)+pow((ytect[2]-0.5),2));
// Pocita se to jenom jednou.

      if ((radius<0.5) && (RecLev>-1) && (Recursion)) {
            if ((radius<0.35) && (radius>0.25) && (OnTop)) {
                  OnTop=false;
                  RecLev=1;
            }
            else if ((radius<0.25) && (radius>0.15) && (OnTop)) {
                  OnTop=false;
                  RecLev=1;
            }
            else if ((radius<0.15) && (radius>0.07) && (OnTop)) {
                  OnTop=false;
                  RecLev=2;
            }
            else if ((radius<0.07) && (OnTop)) {
                  OnTop=false;
                  RecLev=2;
            }

            if (yt1!=yt2) {                                          //
Je to horni nebo spodni trojuhlenik?
                  div_result=div((yt2-yt1), 2);          //       Spocte
souradnice stredu vsech stran trojuhelnika
                  yp1=yt1+div_result.quot;
                  xp1=xt1;
                  div_result=div((xt3-xt2), 2);
                  xp2=xt2+div_result.quot;
                  yp2=yt2;
                  xp3=xp2;
                  yp3=yp1;
            }

            else { // spodni
                  div_result=div((xt2-xt1), 2);          //       Spocte
souradnice stredu vsech stran trojuhelnika
                  xp1=xt1+div_result.quot;
                  yp1=yt1;
                  xp2=xt2;
                  div_result=div((yt3-yt2), 2);
                  yp2=yt2+div_result.quot;
                  yp3=yp2;
                  xp3=xp1;
            }
                  DrawTriangle(xt1,  yt1,  xp1,  yp1,  xp3,  yp3,  RecLev,
false, OnTop);      // rekurzivni volani kresleni trojuhelnika, parametr
rekurze o jednu nizsi, test radiu uz v dalsim kole nedelej.
                  DrawTriangle(xp1,  yp1,  xt2,  yt2,  xp2,  yp2,  RecLev,
false, OnTop);
                  DrawTriangle(xp1,  yp1,  xp3,  yp3,  xp2,  yp2,  RecLev,
false, OnTop);      // prostredni trojuhelnik
                  DrawTriangle(xp3,  yp3,  xp2,  yp2,  xt3,  yt3,  RecLev,
false, OnTop);
      }
```

```
      else {              // Pokud se nevola rekurze, dochazi k vlastnimu
kresleni...

            for ( i = 0; i < 3; i++ ) {
                  ComputeImpulse(coord[i].x,coord[i].y,PhotonImpulse);
                  //zs=PhotonImpulse[3]/PhotonImpulse[0];
                  TransformVector(PhotonImpulse,TrMatrix);              //
transformace 4vect pomoci tr. matice
                  //zs=PhotonImpulse[3]/PhotonImpulse[0];
                  WhereOnSphere(PhotonImpulse,&theta[i],&phi[i]);  //  Kde
skonci foton na nekonecne sfere, tj. rovnobezky se v nekonecnu protinaji na
tom, kde se kamera nachazi vuci stredu. Z hlediska nek sfery je kamera vzdy
ve stredu sfery.
            }
            side[0]=sqrt(pow((theta[0]-theta[1]),2)+pow((phi[0]-phi[1]),
2));
            side[1]=sqrt(pow((theta[1]-theta[2]),2)+pow((phi[1]-phi[2]),
2));
            side[2]=sqrt(pow((theta[2]-theta[0]),2)+pow((phi[2]-phi[0]),
2));

            if (!((side[0]>0.5) || (side[1]>0.5) || (side[2]>0.5))) {
                  glBindTexture(GL_TEXTURE_2D, texture[0]);
                  glBegin(GL_TRIANGLES);
                        glTexCoord2f(phi[0],     theta[0]);     glVertex3f(
xtect[0], -ytect[0],  0.0f);
                        glTexCoord2f(phi[1],     theta[1]);     glVertex3f(
xtect[1], -ytect[1],  0.0f);
                        glTexCoord2f(phi[2],     theta[2]);     glVertex3f(
xtect[2], -ytect[2],  0.0f);
                  glEnd();
            }

            else {
                  for ( i = 0; i < 3; i++ ) {
                        phi[i]=phi[i]-0.5;
                        if (phi[i]<0) phi[i]=phi[i]+1;
                  }
                  glBindTexture(GL_TEXTURE_2D, texture[1]);
                  glBegin(GL_TRIANGLES);
                        glTexCoord2f(phi[0],     theta[0]);     glVertex3f(
xtect[0], -ytect[0],  0.0f);
                        glTexCoord2f(phi[1],     theta[1]);     glVertex3f(
xtect[1], -ytect[1],  0.0f);
                        glTexCoord2f(phi[2],     theta[2]);     glVertex3f(
xtect[2], -ytect[2],  0.0f);
                  glEnd();
            }
      }
}

void TransformVector(vect Vector, matrix TrMatrix) {
      int i;
      vect OrVect;
      VectCopy(OrVect,Vector);
      for ( i = 0; i < 4; i++ ) {
            Vector[i]=ScalarProduct(OrVect,TrMatrix[i]);
      }
}

double ScalarProduct(vect x,vect y) // obycejny
{
      double a=0;
      int i;
      for (i=0; i<4; i++) a=a+x[i]*y[i];
      return(a);
}
```

```
void VectCopy(vect a,vect b) {
       int i;
       for ( i = 0; i < 4; i++ ) {
              a[i]=b[i];
       }
}

void WhereOnSphere(vect Impulse, GLfloat *pTheta, GLfloat *pPhi) {
       double cross;
       double x,y,z;
       double pi = 3.14159265359;
  GLfloat textx, texty;
       x=Impulse[1]/Impulse[0];
       y=Impulse[2]/Impulse[0];
       z=Impulse[3]/Impulse[0];
       *pTheta=fabs((180/pi)*asin(z));
       cross=sqrt(pow(x,2)+pow(y,2));
       *pPhi=(180/pi)*asin((y/cross));
       if (z<0) *pTheta=-*pTheta;              // Zde se provadeji potrebne
upravy v zavislosti na znamenku souradnic
  if (*pPhi<0)                                                    //    aby    bylo
vypoctena spravna hodnota sferickych souradnic
       {
              if (x>0) *pPhi=360.0-fabs(*pPhi);
              else *pPhi=180.0-*pPhi;
       }
       else if (x<0) *pPhi=180-*pPhi;
  if (*pTheta>0) texty=0.5+(*pTheta/(90))*0.5;      // Transformace sour x a
y do GL
  else texty=0.5+(*pTheta/(90))*0.5;
  textx=(*pPhi/(360))*1.0f;
  *pPhi=textx;
  *pTheta=texty;
}


void StartHole(void) {
       extern int x2, y2;
       extern bool Recursion;
       bool OnTop=TRUE;
       int RecLev=1;       // Hloubka rekurze pri vykreslovani
       extern double Tau=-5.0; // 1.5
       extern double pix_size;
       int i,j,a,b,xbase,ybase,dd;
       div_t div_result;
       div_result=div(XSIZE, 2);
       x2=div_result.quot+1; // Souradnice prostredku od kraje kamery
       div_result=div(YSIZE, 2);
       y2=div_result.quot+1;
       pix_size=CAMSIZE/XSIZE; // Rozmer pixelu bude v cm.
       dd=20;
       Recursion=true;
       a=(XSIZE-1)/dd; b=(YSIZE-1)/dd;

       for (j=0;j<=b; j++)                        // osa y
              for (i=0; i<=a; i++)                // osa x
              {
                            xbase=i*dd; ybase=j*dd;

       DrawTriangle(xbase,ybase,xbase,ybase+dd,xbase+dd,ybase+dd,RecLev,
true, true); // Nakresli trojuhelnik s temito vrcholy

       DrawTriangle(xbase,ybase,xbase+dd,ybase,xbase+dd,ybase+dd,RecLev,
true, true);
              }

       // Tau += 0.2;
}      // Konec hlavni funkce
```

Použitá literatura

[1] Misner C. W., Thore K. S., Wheeler J. A. (1973): Gravitation. W. H. Freeman and Company, San Francisco.

[2] Webové stránky v češtině věnované Open GL a 3D grafice – http://nehe.ceskehry.cz/

[3] Votruba V. (1969): Základy speciální teorie relativity. Academia, Praha

[4] Dan Bruton: Color science http://www.midnightkite.com/color.html