

PV112 Java - 5. cvičení

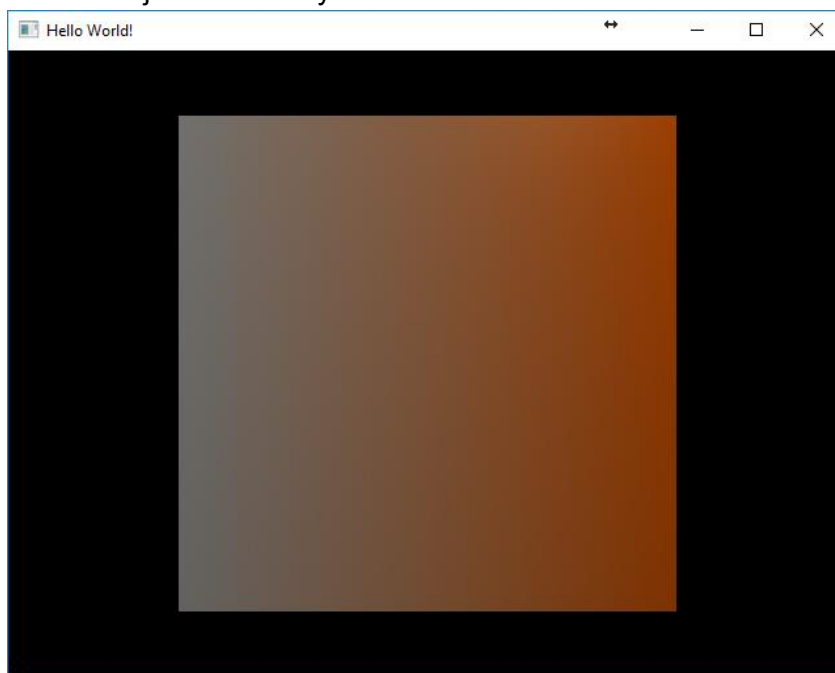
Stáhněte si cv5.zip ze studijních materiálů. Rozbalte a složku otevřete v NetBeans `File -> Open project...` (`Ctrl + Shift + O`). Po úspěšném otevření, zkompilování a spuštění projektu by se mělo zobrazit okno 640x480 pixelů uprostřed obrazovky s nadpisem "Hello World!" a šedou kostkou. Můžete ovládat pozici kamery; při držení levého tlačítka myši se můžete otáčet kolem středu, a při držení pravého tlačítka můžete přibližovat/oddalovat scénu.

Část 1.: generování barvy podle pozice (tzv. procedurální textura)

Dnes si vyzkoušíme více programování v shaderech. V první části tak, že budeme vypočítávat barvu objektu čistě z texturovacích souřadnic (umístění na povrchu) bez použití textury jako takové.

Task 1 - barevný přechod

Do fragment shaderu nám přichází texturovací souřadnice jako `vec2` od (0;0) do (1;1). Zároveň už známe funkci $mix(x, y, a) = (1 - a) * x + a * y$ (lineární interpolace). Pokud neznáte, najděte si její dokumentaci. Parametr a může být `float` nebo vektor odpovídající x a y . **Zkuste těmito prostředky vytvořit na stěně kostky gradient z šedé do oranžové.** Tyto barvy jsou již definované jako konstanty.

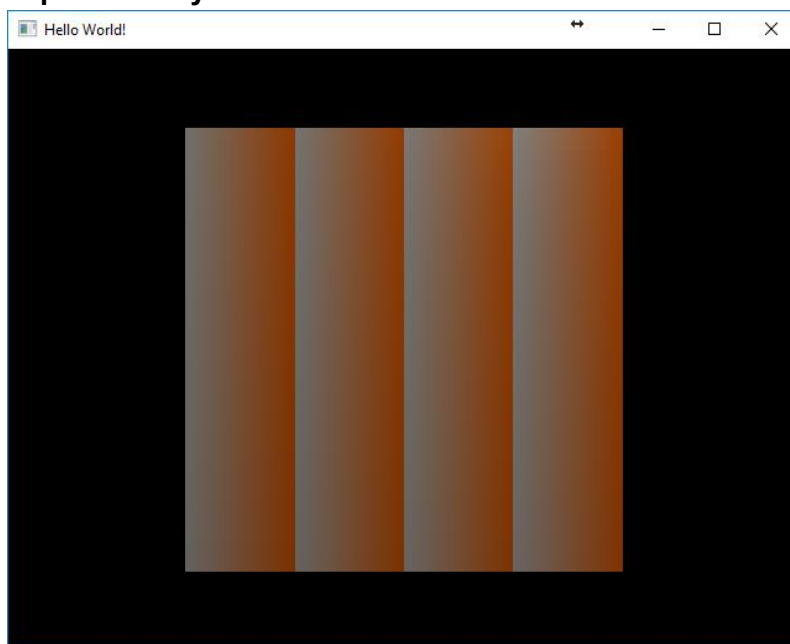


Task 2 - opakování vzoru

Na minulém cvičení jste využívali `GL_TEXTURE_WRAP` vlastnost textury a poté ji vzorkovali větším rozsahem souřadnic a tím docílili opakování/rozkopírování jedné textury vícekrát po povrchu. Podobný princip použijeme teď, ale tentokrát budeme mít celý systém plně pod kontrolou. Efektu docílíme přidáním

operace “modulo” do výpočtu. Mohli bychom použít přímo operaci modulo $x \bmod y$, kde $y = 1$. Na tento konkrétní případ má GLSL funkci $\text{fract}(x) = x - \lfloor x \rfloor$, která zachová z čísla pouze desetinnou část.

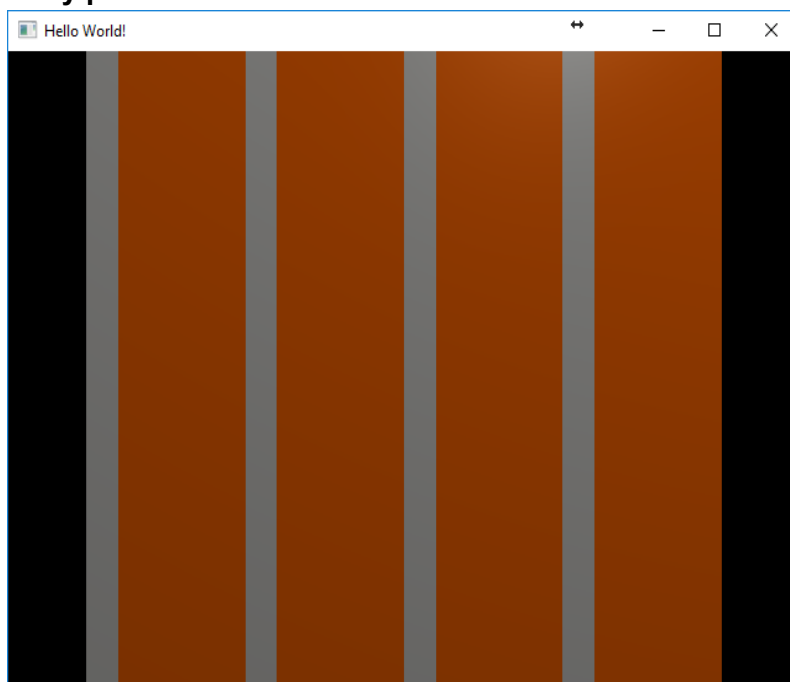
Zopakujte gradient na povrchu čtyřikrát za sebou.



Task 3 - ostrá změna

Pokusíme se postupně vybudovat efekt cihlové zdi; ta ale nemá moc plynulých přechodů, spíš se ostře střídá šedá malta ve spárách s oranžovými cihlami. Chceme tedy do určité souřadnice (řekněme 0.2) zobrazovat jednu barvu a dál druhou. To by šlo vytvořit nějakou if podmínkou, ale podmínkám je dobré se v sharedech vyhýbat. V tomto případě se jí můžeme vyhnout tak, že si z podmínky vygenerujeme hodnotu $\text{ratio} = \text{float}(x \geq 0.2)$, která ve vyhodnotí vždy a s tou dále pracujeme v nevětvěném výpočtu. Na toto lze použít funkci $\text{step}(\text{edge}, x)$.

Změňte gradient na ostrý přechod v 0.2.

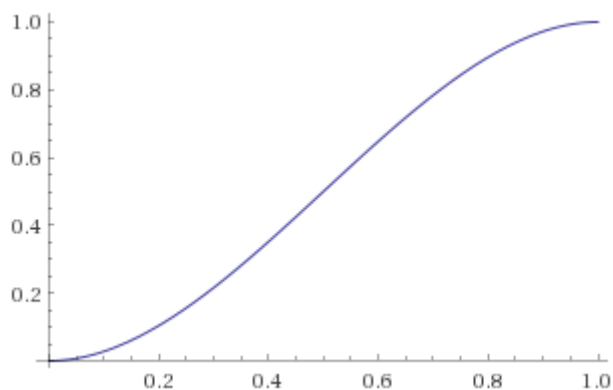


Task 4 - ostrá změna s malým přechodem

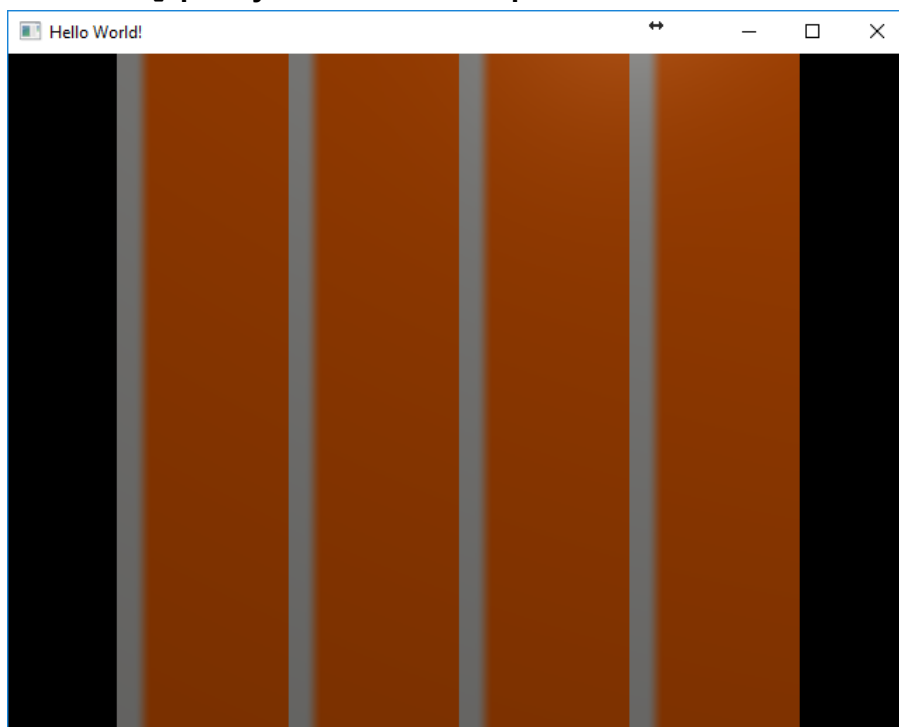
Ve skutečnosti ale cihly nemají přesně hladké okraje a maltu dělník nehází s mikroskopem. Vzhled vylepšíme tak, že na přechodu uděláme malý gradient, pro zjemnění přechodu. Chceme tedy aby do hodnoty 0.1 byla šedá, pak začal přechod do oranžové a od 0.2 dál už byla čistě oranžová. Jedná se o celkem často používaný výpočet, a existuje na něj opět funkce:

$$\text{smoothstep}(\text{edge0}, \text{edge1}, x) = \begin{cases} 0 & x < \text{edge0} \\ 1 & x > \text{edge1} \\ \text{Hermite}((x - \text{edge0})/(\text{edge1} - \text{edge0})) & \text{jinak} \end{cases}$$

, kde $\text{Hermite}(t) = 3t^2 - 2t^3$ vypadá v rozsahu $t \in [0; 1]$ následovně:

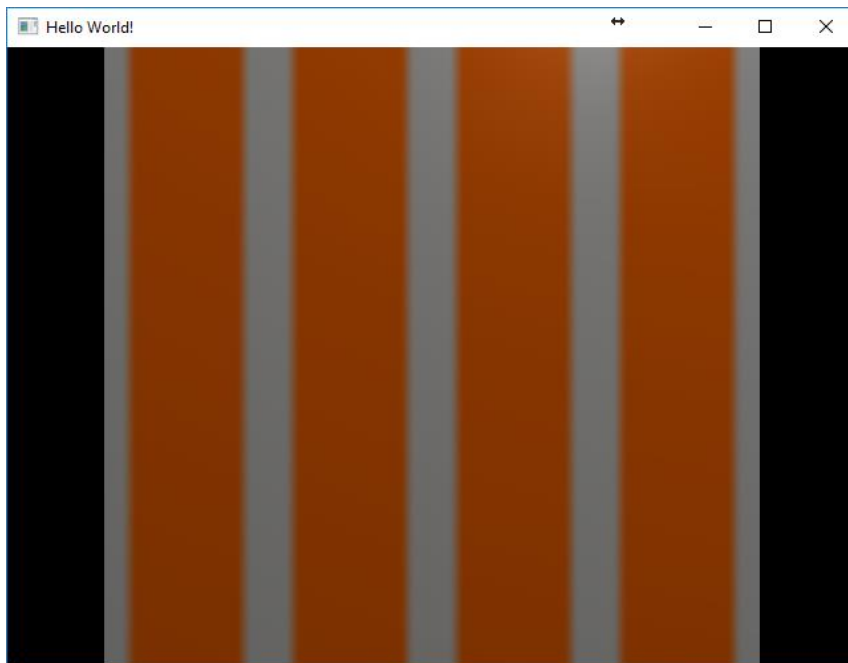


Využijte funkci `smoothstep` pro vytvoření hladkého přechodu v oblasti od 0.1 do 0.2.



Task 5 - přechod na obou stranách vzoru

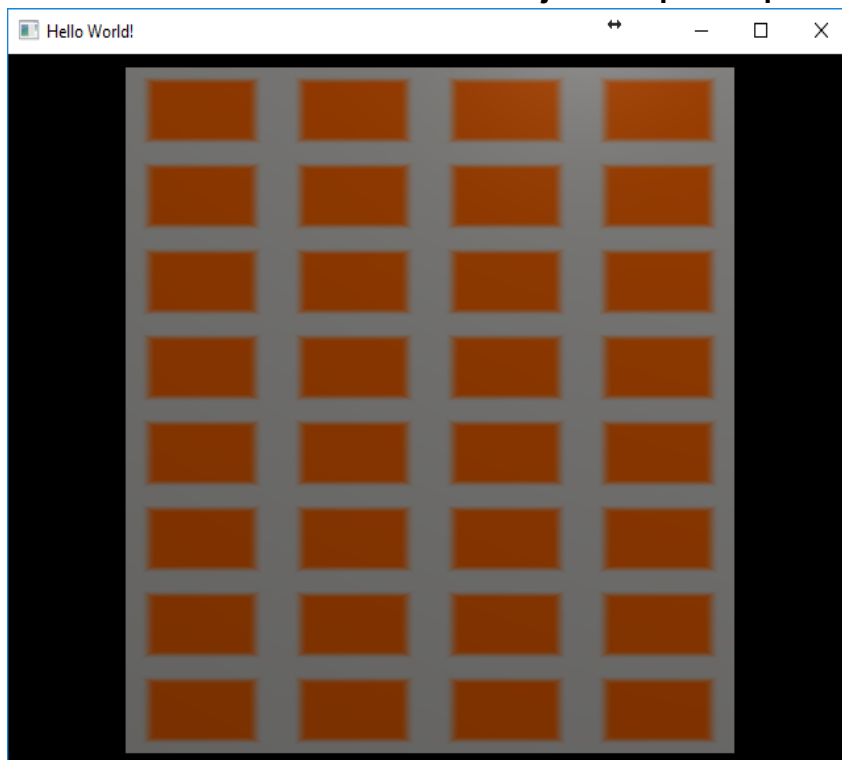
Dejme tomu, že již máme vzor vypadající jako spáry mezi cihlami – nicméně stále máme ostrý přechod na druhé straně. To vyřešíme tak, že na konec opakovaného vzoru přidáme zase přechod jako v předchozím případě, ale z oranžové do šedé. Pro symetrii určíme hranice jako 0.8 a 0.9. Využijte toho, co již víte o `smoothstep` a změňte výpočet parametru pro mix tak, aby ke konci zase klesal.



Task 6 - 2D vzor

Nyní máme celý barevný průchod, který chceme pro naše cihly. Nyní ho jen stačí použít pro ovlivnění barvy v obou osách. Všechny funkce, které jsme zatím použily dokáží pracovat i s celými vektory a svoje operace provádět na všech jeho složkách, takže to ani nebude tak náročné. Pokud si nebudete jistí, zkuste se podívat do dokumentace.

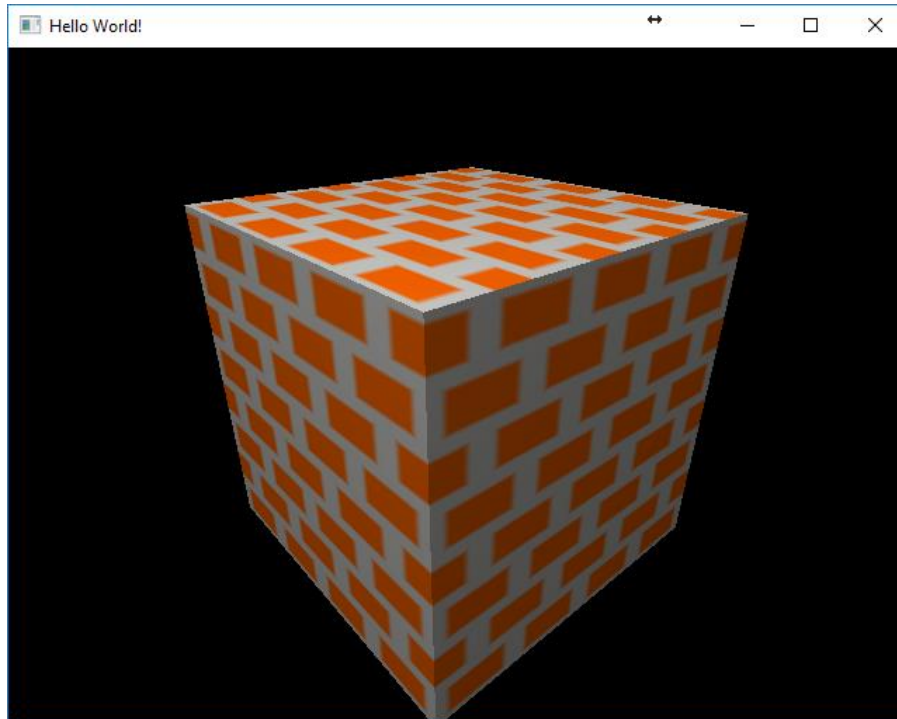
Zařídte generování cihlového vzoru v obou osách. Použijte větší počet opakování pro vertikální.



Task 7 - struktura zdi

Reálná zeď by takto vypadat neměla, protože by nebyla moc pevná a spára by byla slabým místem. Cihly se proto pokládají překládaně tak, že každá druhá řada je posunuta o půl cihly a spáry se tak rozloží. Pro nás to znamená, že musíme získat informaci o tom, kolikáté opakování vzoru právě kreslíme; následovně stačí jen posunout horizontální souřadnici a vygeneruje se nám správná barva.

Posuňte sudé řady cihel tak, aby zeď vypadala reálněji.



Task 8 - punt'á

Zkusíme si ještě jeden vzor – puntíkový. Opět začneme jedním prvkem, již zmíněným puntíkem. Zkuste namíchat červenou a bílou (jsou opět připravené jako konstanty) tak, aby uprostřed stěny kostky vznikl červený kruh o průměru 0.3. Přidejte i kousek hladkého přechodu mezi barvami pomocí `smoothstep`.

Udělejte červený puntík uprostřed “texture”.

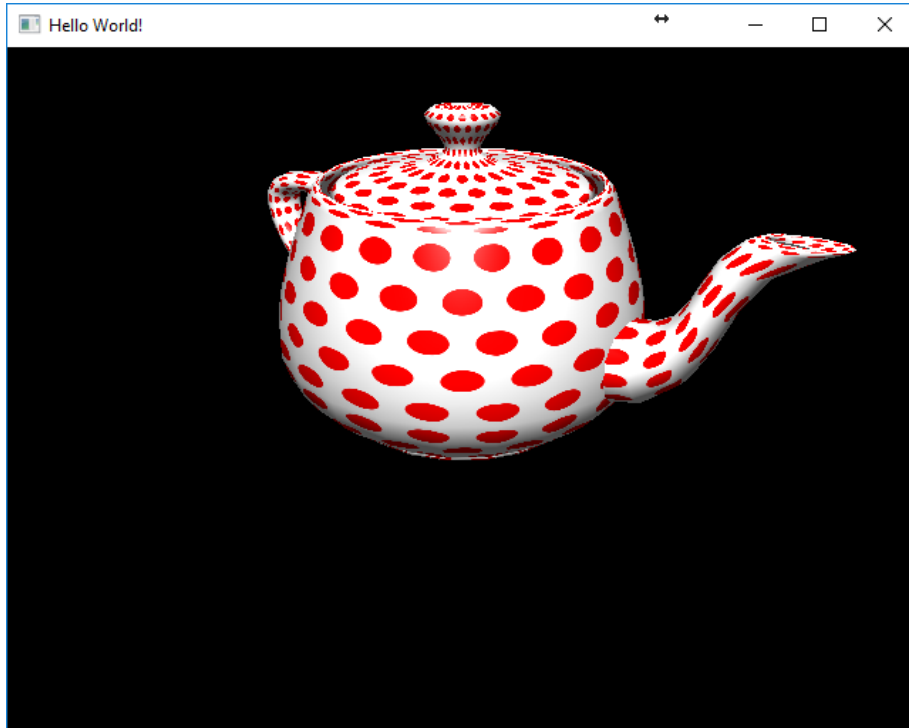
Teď to vypadá jako vlajka jednoho státu. Poznáte kterého?



Task 9 - Granko™ konvička

Opět vzor párkrát zopakujeme v obou osách a posuneme sudé řady o půlku. Pokud výsledek aplikujeme na konvičku, získáme základ pro reklamu na nejmenovaný český granulovaný kakaový nápoj.

Duplikujte puntík do 5x5 mřížky a posuňte některé řady. Zkuste vykreslit na konvičku.



Část 2.: Post-processing (kreslení do textury a z textury na obrazovku)

Velké množství vykreslovacích technik a efektů vyžaduje více kroků ke zpracování dat a vygenerování kýženého obrazu. K tomu se často využívá možnost vykreslit data přímo do textury a s ní dál pracovat/počítat. My si dnes ukážeme jednoduchý post-processing, kdy si vykreslíme scénu normálně jako doteď, ale ne přímo do paměti okna, ale do oddělené textury. Tento výsledek nezobrazíme rovnou, ale vykreslíme si další 2 trojúhelníky, které pokryjí celé okno a dovolí nám aplikovat na pixely nějaký filtr.

Task 10 - vykreslování mimo obrazovku do FBO a vykreslení textury na obrazovku přes fullscreen quad

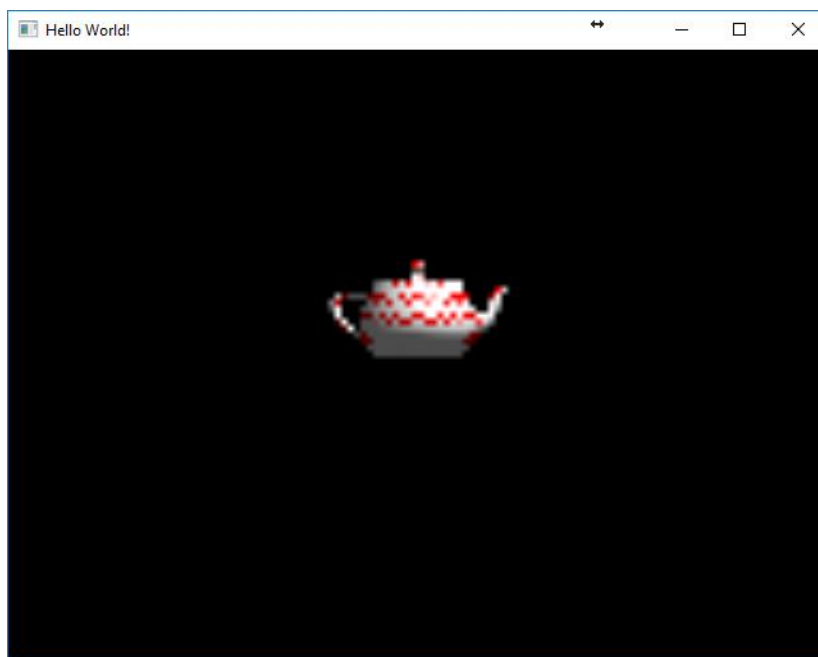
Na vykreslování do textury se používá Frame Buffer Object (FBO), který nám umožní vytvořit vedlejší paměť, do které se bude kreslit, a kterou máme více pod kontrolou - např. můžeme nastavit, jak má vypadat paměť pro výstup; tedy kolik textur (a jejich velikost) má být k dispozici pro zapisování a jaký je jejich formát (bitová hloubka jednotlivých složek, ...). Povinná je pouze textura pro hloubku; vytvoříme si ale i jednu texturu pro barvu, protože ta nás zajímá a chceme ji následně upravit a zobrazit. Do této paměti si tedy vykreslíme naši scénu a tím pádem její barevný výstup budeme mít v připojené textuře.

Nicméně takto vykreslená scéna do textury se sama nikde nezobrazí – je potřeba ještě tuto texturu zobrazit. Nejšikovnější způsob je vykreslit obdélník, který pokryje celou zobrazovanou plochu; když se potom vygenerují jeho fragmenty, tak budou odpovídat pixelům okna a můžeme je obarvit podle naší vykreslené textury. Na to jsou potřeba 2 hlavní věci: geometrie obdélníku a shadery, které správně umístí jeho vrcholy a obarví jeho fragmenty.

Zajistěte vše potřebné k získání mezikroku pro post-processing.

Další výhodou vedlejší paměti pro vykreslování je i její nezávislá velikost. Můžeme např. kreslit do pětkrát menší textury – je to textura jako každá jiná a při vzorkování se její hodnoty mohou dopočítat. Výsledek

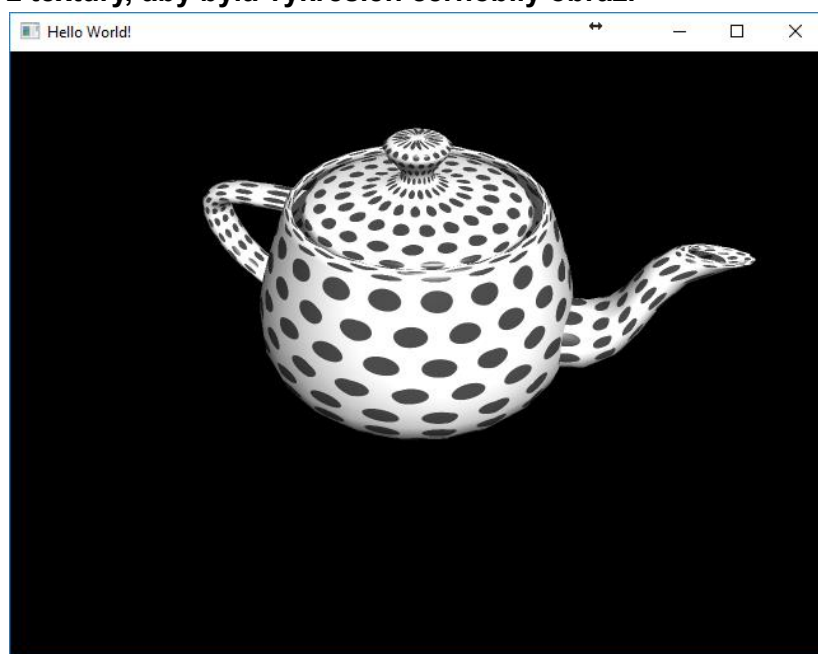
je vidět na následujícím obrázku. Zajímavější ale často je kreslit naopak do větší textury. K čemu to může být dobré?



Task 11 - úprava barev

Jedním z typických post-processing efektů je úprava výsledné barva. Lze například zobrazit jen jednotlivé barevné složky. Předtím, než to uděláte, zkuste se zamyslet – jak bude vypadat výsledek pro červenou, zelenou, modrou? Dalšími úpravami může být inverze barev nebo, zvýraznění některých barevných tónů. Vyzkoušíme si převod na černobílý obraz, který není nic jiného než zobrazení pouze jasové složky světla. Naivní implementace by bylo jednoduché zprůměrování barvy, ale lidé vnímají různé vlnové délky jinak intenzivně, proto je potřeba vážený průměr. Doporučované váhy jsou různé, často se ale používají ty ze standardu YIQ/NTSC a to 0.299, 0.587 a 0.114 pro R, G a B složky.

Zprůměrujte barvy z textury, aby byla vykreslen černobílý obraz.

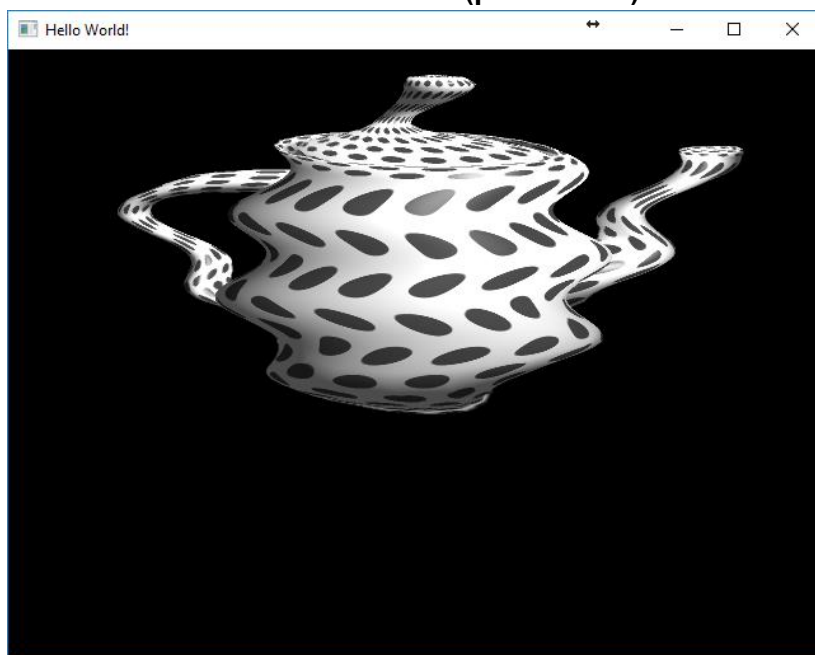


Task 12 - Vlnky™ (deformování obrazu posunem)

Předchozí efekt by šel naimplementovat i bez post-processingu; ale pro následující se bez toho něj už neobejdete. Vzhledem k tomu, že máme k dispozici všechny vykreslené pixely, můžeme si pomocí texturovacích souřadnic vzít i nějaký, který původně byl na jiném místě. Všechny tyto efekty můžeme

měnit v čase. GLSL ale neumožňuje nijak zjistit aktuální čas běhu programu, ani jinou obdobnou hodnotu – musíte si tedy předat hodnotu pro animaci do shaderu sami.

Vytvořte zvlnění obrazu. Zkuste ho animovat v čase (po stisku A).



Task 13 (bonus) - kreslení drátěného modelu

Co se stane, pokud teď stiskneme L pro vykreslování čar místo vyplněných trojúhelníků?



Proč se to děje? Rozhodně se nejedná o chtěný výsledek – chtěli bychom spíše vidět zvlněný a černobílý drátěný model konvičky. Jak toho docílit?