Počítačová grafika

Generátor kinetické typografie

4. prosince 2016

Autoři: Jan Hrdina, xhrdin10@stud.fit.vutbr.cz
Peter Gazdík, xgazdi03@stud.fit.vutbr.cz

Fakulta Informačních Technologií
Vysoké Učení Technické v Brně

Zadání

- Uživatel při spuštění aplikaci předá soubor s libovolným textem
- Aplikace postupně text zobrazí
  - animovaně s nelineárními časovými křivkami (Beziérovy kubiky)
  - ve 3D
  - vhodně esteticky rozdělený a rozmístěný
- Aplikace náhodně vybírá z možných animačních efektů a uspořádání
  - Každý efekt si může klást různé požadavky na čtený text, např. „začíná čtyřmi slovy, z nichž první je krátké a druhé je středně dlouhé“
  - Podle vhodnosti efektu pro zbývající text je efektu přiřazena váha, která ovlivňuje pravděpodobnost vybrání.

Použité technologie

- GLEW
- GLFW
- FTGL - Knihovna pro vykreslování textu v OpenGL - neoficiální verze přidávající podporu OpenGL 3 dostupná na https://github.com/lukexi/ftgl3

Použité zdroje

- Beziérovy kubiky [8]
- OpenGL tutorial [2]
- Fonty + licence
  - Roboto – Apache License, Version 2.0
  - Fira Mono – Open Font License, Version 1.1
  - DejaVuSerif – licence viz http://dejavu-fonts.org/wiki/License
- Texty
  - demos/dream - Martin Luther King, Jr., I Have a Dream [5]
  - demos/havel - První novoroční projev Václava Havla [3]
- Videa s kinetickou typografií [7] [12]
- Open-source herný engine Cocos2d-x [1]
- Kniha Game programming patterns [6]

Nejdůležitější dosažené výsledky

Estetika

Při návrhu efektů a podpůrných tříd jsme si dali záležet na výsledných estetických kvalitách.

- Využíváme nelineárních animačních křivek
- Snažili jsme se pečlivě volit kompozici textu, podpůrné metody kreslení textu umožňují neobvyklé požadavky jako „Automaticky nastav velikost písma tak aby šířka textu byla pokud možno přesně 600px a zároveň jeho výška nepřesáhla 800px“
- Volba vhodného efektu pro aktuální text - když například text začíná krátkým slovem, je větší pravděpodobnost, že se vybere efekt který toho dokáže využít a například zarovnat velké „I“ po straně řádků „I have a dream“

Obrázek 1: Efekt „LetterAside“

Kompletní animační framework

Pro zajištění snadné manipulace s textovými objekty jsme vytvořili 3D animační framework. Při jeho implementování jsme se inspirovali 2D herním enginem cocos2d-x[1] a návrhovými vzory publikovanými v knize Game Programming Patterns [6].

- Animace v čase – posuny, rotace, škálování, změna průhlednosti.

- Seskupování animací do skupin a jejich paralelní / sekvenční vykonávání.
- Invokace vlastních funkcí v průběhu vykonávání animací.
- Úprava časového průběhu animací pomocí beziérových kubik[8].
- Pozicování nezávislé na velikosti okna.
- Pozicování objektů relativně k jejich rodičovskému objektu v grafu scény a seskupování objektů.

Snadné přidávání nových efektů

Vše je navrženo tak, aby bylo možné snadno přidávat nové efekty. Pro maximální usnadnění této činnosti jsme dokonce vytvořili skript genEffect, který vytvoří šablonu pro efekt se zadaným názvem a rovnou jej zaregistruje v aplikaci.

Zvláštní použité znalosti

Rotace kolem pivotu

Aby bylo možné rotovat objekty v jiném bodě, než je jejich počátek (v tzv. pivotu), je potřeba před vykonáním rotace upravit pozici počátku na základě pozice pivotu. Nejjednodušší způsob, jak tohoto docílit je před samotnou rotací posunout celý objekt do tohoto bodu, vykonat rotaci a následně celý objekt posunout zpět. Toto má však za následek narušení relativního pozicování ostatních objektů, které jsou potomky aktuálního objektu.

Abychom se tomu vyhnuli, je potřeba upravit pozici počátku na základě hodnoty aktuální rotace a pozice pivotu a ne jen na základě pozice pivotu. K aktuální pozici tedy potřebujeme připočítat hodnotu danou následujícím výpočtem, kde matice odpovídá transformační matici rotace postupně v osách x, y a z a body x, y a z představují pozici pivotu.

$$\begin{bmatrix} \cos\beta\cos\gamma & -\cos\beta\sin\gamma & \sin\beta & 0 \\ \cos\alpha\sin\gamma+\sin\alpha\sin\beta\cos\gamma & \cos\alpha\cos\gamma-\sin\alpha\sin\beta\sin\gamma & -\sin\alpha\cos\beta & 0 \\ \sin\alpha\sin\gamma-\cos\alpha\sin\beta\cos\gamma & \sin\alpha\cos\gamma+\cos\alpha\sin\beta\sin\gamma & \cos\alpha\cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot (-1) \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

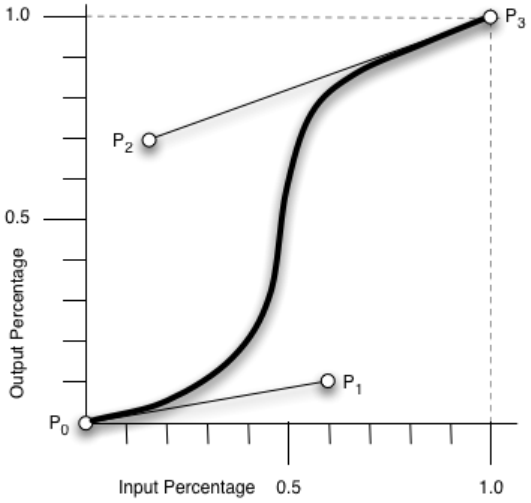
Úprava časového průběhu animací

Pro dosažení přirozeného vzhledu jednotlivých akcí animační framework umožňuje upravit jejich jinak lineární časový průběh. Nový časový průběh je parametrizovatelný pomocí Beziérovy křivky, která je definovaná čtveřicí bodů, viz obrázek 2. Algoritmus výpočtu Beziérovy kubiky je převzat z článku Bezier Curve based easing functions[8].

DoF a motion blur v OpenGL 3

Původně jsme plánovali do projektu začlenit podporu depth-of-field a motion blur. Oba algoritmy se dají snadno realizovat s využitím Accumulator Bufferu, proto jsme si jejich implementaci nechali až nakonec. Bohužel se nicméně pár dní před odevzdáním ukázalo, že accumulator buffer je od OpenGL v3.2 odstraněn z Core Profile [4] a je tedy nutné buďto využít Compatibility Profile, který nám nefungoval s FTGL, nebo využít jiných prostředků.

Pro DoF je například možné využít vlastních FBO, kdy se obrazová a hloubková složka rasterizované scény vyrenderují do textury, která se následně zpracovává ve fragment shaderu. Dá se tak dosáhnout zajímavých výsledků (viz [9]), nicméně z časových důvodů jsme tuto náročnější implementaci již nestihli. Podstatné je, že po tomto kurzu už to pro nás není španělská vesnice a kdybychom měli pár dní navíc, určitě bychom to zvládli.



Obrázek 2: Časový průběh definovaný čtveřicí hodnot $P_0 - P_3$[8]

Ladění OpenGL

Při ladění jsme si vyzkoušeli práci s nástrojem Intel Graphics Performance Analyzer, který umožňuje analyzovat stav pipeline OpenGL v čase, debugovat shadery a dokonce i rozebírat jednotlivé objekty ve scéně. Program je dostupný na https://software.intel.com/en-us/gpa

Náhodný výběr prvků z pole s vahami

Při náhodném výběru textového efektu využíváme algoritmus zohledňující váhy jednotlivých efektů. Algoritmus jsme převzali z http://softwareengineering.stackexchange.com/a/150678

Techniky vykreslování textu

Pro naši realizaci bylo přirozeně nutné nastudovat způsoby řešení vykreslování textů pomocí OpenGL.

Metody pracují většinou na jednom ze dvou principů:

- **Text jako bitmapa** – text se mimo OpenGL vykreslí do bitmapy a ta je následně v OpenGL použita jako textura. Přístupy se můžou lišit podle toho, jestli se vytvoří bitmapa se všemi glyphy a z nich se pak skládají řetězce nebo se renderuje každý řetězec zvlášť
- **Text jako vektor** – text se pomocí tesselátoru převede z polygonálního popisu na trojúhelníky a ty se rasterizují již pomocí grafické karty. Tuto metodu využíváme my.

Pro načítání fontů a jejich polygonální reprezentace slouží knihovna FreeType. O podporu tesselace, pozicování a použití v OpenGL ji doplňuje knihovna FTGL, kterou využíváme. Původní knihovna FTGL s poslední aktualizací v roce 2011 bohužel nepodporuje OpenGL 3, proto jsme použili neoficiální fork.

Práce na projektu

Rozdělení práce v týmu

- **Jan Hrdina**: Implementace efektů, vykreslování/pozicování textů, celková vize
- **Peter Gazdík**: Animační engine (animace, akce, graf scény, vykreslování, ...)

Co bylo nejpracnější

- **Jan Hrdina**
  - Pozicování a transformace textu v souřadném systému u efektů a vytvoření vhodných abstrakcí
  - Koordinace a návaznost animací
- **Peter Gazdík**:
  - Přechod na FTGL s podporou OpenGL 3.0
  - Pokročilejší transformace vyžadující znalosti lineární algebry.
  - Vytvoření souřadného systému simulujícího ortogonální projekci pro zabezpečení jednoduchého pozicování.

Zkušenosti získané řešením projektu

- Trello a Scrum se opět osvědčily na organizaci a plánování vývoje
- Overleaf na online psaní dokumentace
- Osvojení nových technologií a dovedností, viz Zvláštní použité znalosti

---

Autoevaluace

Technický návrh (80%): (analýza, dekompozice problému, volba vhodných prostředků, ...)

- Návrh architektury pomocí UML diagramu tříd
- oddělení OpenGL detailů od zbytku aplikace
- odstínění kódu efektů od implementačních detailů

Programování (70%): (kvalita a čitelnost kódu, spolehlivost běhu, obecnost řešení, znovupoužitelnost, ...)

- Snaha o intuitivní názvy identifikátorů
- Využití C++11 a moderních konstrukcí

Vzhled vytvořeného řešení (80%): (uvěřitelnost zobrazení, estetická kvalita, vhled GUI, ...)

- Pravidlo třetin v kompozici
- Pečlivá typografie
- Easing

Využití zdrojů (90%): (využití existujícího kódu a dat, využití literatury, ...)

- Architektura inspirovaná 2D herním enginem cocos2d-x
- Návrhové vzory uvedené v elektronické verzi knihy Game Programming

Hospodaření s časem (90%): (rovnoměrné dotažení částí projektu, míra spěchu, chybějící části řešení, ...)

- Práce na projektu v čase rozložena rovnoměrně
- Absence pokročilejších post-processing efektů - nechali jsme je na poslední chvíli, protože jsme spoléhali na jednoduchou implementaci pomocí accumulator bufferu, který, jak se ukázalo, již není podporován. Prozkoumali jsme techniky implementace v současných API, ale z důvodu nečekané časové náročnosti jsme ji vynechali.

Spolupráce v týmu (100%): (komunikace, dodržování dohod, vzájemné spolehnutí, rovnoměrnost, ...)

- Využití Gitu, Trella, Skype, chatu
- Rychlá komunikace v obou směrech

Celkový dojem (87%): (pracnost, získané dovednosti, užitečnost, volba zadání, cokoliv, ...)

Bavilo a motivovalo nás pracovat na vlastním zadání, bylo osvěžující pracovat na aplikaci čistě pro estetický požitek. Jsme rádi, že jsme si mohli osahat OpenGL, ke kterému by se člověk jinak nejspíš těžko nutil. Pracovali jsme tvrdě, ale výsledek (myslíme) stálo za to.

Ovládání vytvořeného programu

Technologie potřebné pro spuštění programu

- cmake
- GLEW
- GLFW
- FTGL
- GLM

Obsluha programu

Aplikace se spouští v příkazovém řádku. Přijímá minimálně jeden parametr - cestu k souboru se vstupním textem. Volitelně je možné s parametrem `-s` předat číselný seed pro generátor pseudonáhodné posloupnosti efektů.

Příklady spuštění aplikace:

- ./pgr-project demos/dream
- ./pgr-project -s 42 demos/dream

Před začátkem animace jsme přidali 4sekundovou prodlevu pro možnost zvětšit okno na celou obrazovku apod.

Doporučení pro budoucí zadávání projektů

Oceňujeme možnost realizace vlastního tématu. Na organizaci bychom nic neměnili.

Literatura

[1] Cocos2d-x - open-source game development platform. [Online]
http://cocos2d-x.org/.

[2] Joey de Vries. Learn opengl, extensive tutorial resource for learning modern opengl. [Online]
https://learnopengl.com/.

[3] Václav Havel. Novoroční projev prezidenta Čssr václava havla, 1990. [Online]
http://vaclavhavel.cz/showtrans.php?cat=projevy&val=327_projevy.html&typ=HTML.

[4] Kurt Akeley Mark Segal. The opengl graphics system: A specification (version 3.2 (core profile)), 2009. [Online]
https://www.opengl.org/registry/doc/glspec32.core.20091207.pdf.

[5] Jr. Martin Luther King. I have a dream, 1963. [Online]
http://www.americanrhetoric.com/speeches/mlkihaveadream.htm.

[6] Bob Nystrom. Game programming patterns, 2014. [Online]
http://gameprogrammingpatterns.com/.

[7] Agil Pandri Prakoso. Sky on eye - outraged! kinetic typography, 2012. [Video]
https://vimeo.com/50481715.

[8] Gaetan Renaudeau. Bezier curve based easing functions, 2014. [Online]
http://greweb.me/2012/02/bezier-curve-based-easing-functions-from-concept-to-implem

[9] Martins Upitis. Glsl depth of field with bokeh v2, 2011. [Online]
http://devlog-martinsh.blogspot.cz/2011/10/glsl-depth-of-field-with-bokeh-v2.html.

[10] Wikibooks. Opengl programming/motion blur, 2012. [Online]
https://en.wikibooks.org/w/index.php?title=OpenGL_Programming/Motion_Blur&oldid=2446136.

[11] Wikibooks. Opengl programming/depth of field, 2013. [Online]
https://en.wikibooks.org/w/index.php?title=OpenGL_Programming/Depth_of_Field&oldid=2562207.

[12] After Effects XYZ. Kinetic typography - text animation in after effects, 2015. [Video]
https://www.youtube.com/watch?v=qpT6UOQNEjY.