



Univerza v Ljubljani
Fakulteta
za računalništvo
in informatiko

DOKUMENTACIJA

Navidezna fakulteta v WebGLu

Avtorji:

Miha ZIDAR
Anže PEČAR
Aleksandra BERSAN

1. junij 2011

1 Opis rešitve

1.1 Kaj rešitev sploh je

Vsi vemo, da ima naša fakulteta zelo nelogično razporejene prostore in pogosto se zgodi, da obiskovalec ne najde prave predavalnice. Da bi obiskovalcem in brucem olajšali življenje, smo se odločili narediti 3-D model fakultete.

Ker želimo, da bi bila naša aplikacija čim lažje dostopna, smo jo postavili na internet s pomočjo odprtokodne knjižnice WebGL. Potrudili smo se, da aplikacija teče tekoče na sodobnih brskalnikih z WebGL podporo.

1.2 Uporabljene metode



Jeziki

- JavaScript - Delo z gl knjižnico
- GLSL - za Fragment in Vertex shaderja
- Python - Backend, pretvarjanje .obj datotek v json

Ogrodja

- webql-utils - Za funkcijo requestAnimationFrame
- glMatrix - Uporabne funkcije nad matrikami in vektorji
- jQuery - Izboljšanje uporabniške iskušnje
- Django - Backend za shranjevanje podatkov o mestih zanimanja v podatkovno bazo

Orodja

- Blender - Modeliranje faksa po gradbenih načrtih
- Gimp - Izdelava tekstur

Algoritmi

- Mipmap tekštura
- Phongov odbojni model
- Collision detection

2 Zabavni deli

2.1 Collision detection

V našem modelu imamo zaradi lažjega risanja v WebGL, vse ploskve predstavljene z trikotniki in ne z štirikotniki. To pa tudi nekoliko olajša detekcijo trkov med posameznimi površinami. Ponavadi se detekcijo trkov, hrani posebna podatkovna struktura, ki ima enako 3D predstavitev, vendar je načeloma razdeljena na manjše dele kot prvotna 3D mapa. Tukaj je tudi uprašanje, ali se nam splača hraniti dve strukturi ali samo eno. Mi smo se pri tej seminarski nalogi odločili da bomo imeli dve ločeni strukturi:

- faks: podatkovna struktura velikimi poligoni in manj oglisci ki se bo izrisovala
- coll: podatkovna struktura, ki je sestavljena iz veliko več manjših poligonov, ki se bojo uporabljali za detekcijo trkov.

Ponavadi so najbolj priljubljene strukture za zaznavanje trkov

- KD-tree - k dimenzionalno drevo ki prostor na vsakem nivoju razdeli na dva dela glede na trenutno dimenzijo.
- BSP-tree - struktura podobna kot KD-tree, vendar da na vsakem nivoju bolj optimalno razdeli prostor.
- R-tree - drevo ki prostor razdeli na bounding box-e in se nato išče točke znotraj teh.
- PR-tree - prioritetno R-tree, ki je le nadgranja R-tree da je iskanje hitrejše.

In tako kot so vse te strukture dobre za splošno iskanje trvko ene predmeta z ostalimi predmeti v prostoru, si lahko za naše potrebe izberemo kakšno, ki je malo bolj enostavna. To lahko rečemo, ker ne bomo veliko leteli (ko se leti, bo collision detection izklopjen), ali skakali. Torej to pomeni da bomo vedno v stiku z vsaj eno ploskvijo (tla), ter morda še z kakčno steno ali drugim objektom.

Torej ko smo se odločili za našo čollštrukturo, smo upoštevali kakšne so naše zahteve za detekcijo trkov in smo jo naredili tako da je največji trikotnik manjši ali enak objektu ki bo predstavljal človeka ki se sprehaja po faksu. V nadaljevnu bomo videli da ni potrebno da bi bili poligoni še manjši, čeprav so lahko nekateri, ki sestavlja majhne natančne dele modela.

Struktura - navadena 2D tabela

Na koncu smo videli da bo najlažje implementirati navadno 2D tabelo, ki bo imela naslednjo strukturo

Index	Value
0	$\{x_0^0 \dots x_m^0 \mid m \leq n\}$
1	$\{x_0^1 \dots x_m^1 \mid m \leq n\}$
2	$\{x_0^2 \dots x_m^2 \mid m \leq n\}$
:	:
n	$\{x_0^n \dots x_m^n \mid m \leq n\}$

V kateri Index predstavlja zaporedno število posameznega trikotnika, in v Value je zapisan seznam indeksov ostalih trikotnikov, ki so zelo blizu trenutnega. Podatke o posameznem trikotniku lahko preberemo iz naše prejšnje strukture čollin indeksom trikotnika o katerem želimo podatke.

Sedaj pa potrebujemo le še manjšo strukturo, v kateri je seznam trikotnikov z katerimi se že dotikamo. To strukturo bomo pa imenovali "Touching". Celoten algoritmom pa zgleda tako:

1. začetni položaj je na točno določenem trikotniku, in indeks tega trikotnika damo v Touching seznam.
2. shranimo trenutni položaj
3. premaknemo se na željeno mesto (kamor želi uporabnik).
4. za vsak indeks trikotnika ki je shranjen v Touching preverimo če se seka z našim osebkom.

5. preverimo tudi za vse indekse trikotnikov ki se nahajajo v seznamu trenutnega Touching trikotnika.
6. trikotnike katerih se dotikamo damo v Touching in tiste, ki se jih ne, damo ven.
7. popravimo premik, če je potrebno in nadaljujemo pri točki 2.

Algoritem - hitra detekcija trkov dveh trikotnikov

Za detekcijo ne pregledujemo bounding boxov posameznih trikotnikov, in struktur, saj smo pri postavljanju strukture zagotovili da ne bomo pregledovali nobenih trikotnikov ki so nam predaleč.

Ugotavljanje pa poteka tako:

1. izračunaj enačbo ravnine za vsak trikotnik.
2. naredimo skalarni produkt normale ravnine z vektorjem od poljubne točke na ravnini do vsake točke na drugem trikotniku. Tako bojo imele točke na eni strani ravnine en predznak, točke na drugi strani pa drug predznak.
3. pogledamo če imajo vse točke enak predznak, kar pomeni da en trikotnik leži v celoti nad ali pod ravnino drugega trikotnika. V temu primeru ne more priti do trka in vrnemo 0.
4. enako naredimo tudi za drugi trikotnik.
5. pogledamo če so vse razdalje toče enega trikotnika od ravnine drugega trikotnika enake 0. in vrnemo da ni prišlo do trka. ker se tako znebimo dodatnih težav.
6. če oba trikotnika sekata ravnino drugega trikotnika, pa lahko naredimo črto presečišča ravnin in vemo da se laho trikotnika sekata le na tisti črti.
7. izračunamo presečišče stranic trikotnika z črto ki je je na presečišču ravnin, za oba trikotnika.
8. sedaj pa le pogledamo na tej premici če se intervala posameznih trikotnikov sekata, in v temu primeru vrnemo da je prišlo do trka.

Obravnavanje trkov pa je tudi enostavno, saj imamo shranjeno zadnjo dovoljeno pozicijo in nimamo nobenih prožnih odbojev. Edina posebnost je le da ko izvemo za vse ploskve s katerimi je prišlo do trka, bomo naš objekt premaknili v smeri seštevka normaliziranih normal teh ploskev. to nam bo omogočalo da hodimo v klanec in po stopnicah.

2.2 Pretvarjanje iz .obj v json in parsanje

Za izrisovanja modelov iz Blenderja je potrebno prebrati datoteko v .obj formatu in iz zapisa izluščiti *vertexe*, *normale*, *face* in različne materiale. Za ta namen smo napisali preprosto python skripto, ki .obj datoteko prebere in prebrane podatke zapiše v json formatu. Na ta način smo pohitrili začetek izrisovanja, saj spletnemu brskalniku ni potrebno parsati .obj datoteke ob vsaki naložitvi strani.

2.3 Phongov odbojni model

2.4 Collision detection

3 Problemi

3.1 Blacklisted GPUs

Še preden smo začeli z razvojem naše seminarske naloge, smo naleteli na probleme. Tako Googlov Chrome, kot Mozillin Firefox sta imela na črni listi grafične kartice, ki so bile v naših računalnikih, kar je pomenilo, da je bil WebGL onemogočen. WebGL ni bilo možno prisiliti k delovanju iz nastavitev znotraj brskalnika pa čeprav smo šarili po nastavivah za developerje (Chromov: `about:flags` in FFjev: `about:config`). Po nekaj dneh Googlanja smo le naleteli na sistemsko spremenljivko `MOZ_GLX_IGNORE_BLACKLIST=1`, s katero smo omogočili WebGL znotraj FireFoxa in Chromovo `--ignore-gpu-blacklist` zagonsko stikalo.

Z najnovejšo različico Google Chromea nastavljanje zagonskega stikala ni več potrebno, FireFox4 pa še vedno potrebuje nastavljenou sistemskou spremenljivku.

3.2 WebGL

Začetnikom v OpenGLu iz lastnih izkušenj ne priporočamo dela z WebGLom. WebGL namreč temelji na OpenGL ES 2.0, ki je oskrunjena verzija OpenGLa, saj so iz specifikacije odstranili vse funkcije iz OpenGLa, ki so v pomoč začetnikom. Brez dodatnih knjižnic WebGL ne omogoča niti funkcij za skalarni ali vektorski produkt med vektorji, kaj šele funkcije za translacije in rotacije matrik.

3.3 Lepljenje tekstur

3.4 Zaplet v ekipi

Zaradi hujše prometne nesreče članica ekipe Aleksandra ni mogla opraviti svojega dela seminarske naloge v celoti, zato sva Anže in Miha sama nadaljevala projekt.

4 Nadaljne delo

4.1 Podrobnosti na modelu

Zaradi časovne stiske in zapletov v ekipi nam je zmanjkalo časa za dodajanje podrobnosti, kot so table, lijaki, koši za smeti in ostali predmeti v predavalnicah in na faksu. Prav tako nam ni uspelo dokončati vseh nadstropij fakultete.

4.2 Boljše tekture

Če bi imeli več časa, bi lahko ustvarili svoje tekture iz fotografij, ki bi jih posneli na fakulteti. Lahko bi pa tudi nadgradili izrisovanje iz Mipmappinga na kaj bolj modernega.

4.3 Druge ideje

Ena izmed idej je bila tudi, da bi omogočili prikaz jakosti WiFi signala v posameznih delih fakultete. Zabavno pa bi bilo tudi, da bi se prikazoval položaj vseh obiskovalcev, ki so trenutno znotraj navidezne fakultete.

5 Posnetki zaslonov





