Schelet de cod tema 4 EGC
~ cu TODO-uri si explicatii ~

Asa mi-a iesit mie :)
http://www.youtube.com/watch?v=uzf6w8B5iOY

Cum am obtinut scheletul de cod:
-> din ultimele 2 laboratoare complete (lab4 si lab5 shadere)
-> lab4 e baza, iar din lab5 am luat *lab_texture_loader.hpp* si am copiat codul din shadere si partea din mai pt incarcarea texturilor (si evident imaginea din *resurse*)

Before you start:

- mi-am dat seama ca TODO-urile puse in cod sunt inutile, asa ca l-am scos
- in principiu le poti face in ce ordine vrei, sunt mai mult orientative
- codul efectiv e in mare parte hard-codare, deci va trebui sa te joci cu valorile pana obtii ceva care arata bine; in general grafica cam pe asta se bazeaza, sa obtii ceva care arata bine :)
- in caz ca am uitat sa mentionez ceva, ar fi bine sa te uiti si peste restul codului din laborator
- scheletul de cod e mai mult de forma, nu i-am facut mai nimic, e mai mult pentru cei care nu au laboratoarele facute :P

PRIMA PARTE :: Sistem de animatie

/* TODO [1] */ → main.cpp :: la initializare

- Incarca cele 3 capete .obj
- Ai nevoie doar de *vao* si *num_indices* pentru a desena obiectul. (pentru fiecare obiect)
- Poti sa iti faci o clasa pentru model sau 2 vectori globali, cum preferi; o sa-ti fie mai usor daca faci o clasa, pentru ca vor fi si alte chesti de adaugat

/* TODO [2] */ → main.cpp :: in notifyDisplayFrame()

- Deseneaza cele 3 capete .obj (sau unul e suficient) ca sa vezi ca iti merge
- Pentru desenare, sunt 2 functii: *glBindVertexArray* si *glDrawElements* , pentru asta aveai nevoie de *vao* si *num_indices*
- O sa trebuiasca sa modifici *model_matrix* inainte sa desenezi pentru ca headurile astea sunt foarte mici
- Foloseste *glm::scale(model_matrix,)* si *glm::translate(model_matrix,)* ca sa arate bine, si eventual si rotate daca ai nevoie
- Nu uita de glUniformMatrix4fv(glGetUniformLocation(gl_program_shader_curent, "model_matrix") sa trimiti la shader *model_matrix*

/* TODO [3] */ → main.cpp :: la initializare

- Incarca textura pentru fata (e in arhiva cu resurse pentru tema)
- O sa ai 2 texturi in total, pentru fata si pentru nota muzicala

/* TODO [4] */ → main.cpp :: in notifyDisplayFrame()

- Inainte de cele 2 functii pentru desenat , trebuie sa bindezi textura (in principiu cam totul de face inainte de functiile alea)

glActiveTexture(GL_TEXTURE0 + x); (unde x este un indice oarecare, poti sa pui si 0)
glBindTexture(GL_TEXTURE_2D, textura_ta);
glUniform1i(glGetUniformLocation(gl_program_shader_curent, "textura1"), x); (acelasi x de mai sus)

/* TODO [5] */ → shader_phong_fragment.glsl

- Acum ar trebui sa alegi cum anume pui culorile si lumina
- Varianta *out_color = vec4(light,light, light,1);* pune doar lumina
- Varianta *out_color = vec4(tex1, 1);* pune doar textura

- Incearca sa le combini cumva sa arate bine :)
- As fi putut sa-ti dau direct o formula, dar cred ca e mai bine sa faci tu una (stim ce se poate intampla daca seama prea mult :P)
- Daca iti iese asta ar trebui sa ai iluminarea phong facuta si cam asta e tot in shader, normalele de interpolare le poti calcula in *main.cpp* la initializare mai tarziu
- Eventual s-ar putea sa ai nevoie de o variabila sa-ti zica daca un obiect are sau nu textura; foloseste functia *glUniform1i(glGetUniformLocation(gl_program_shader_curent, "un_int_in_shader"), un_int_in_main);* ca sa trimiti int catre shader, iar in shader sa ai *uniform int un_int_in_shader*

/* TODO [6] */ → main.cpp :: la declarare

- Adauga o variabila pentru *timer* ca sa stii cand trebuie sa schimbi intre cele 3 fete
- Cam in asta consta animatia, trecerea de la o fata la alta, conform cu un *timer*
- Momentan fa sa treaca instantaneu intre obiecte, la un interval de timp, sa vezi ca merge
- Sunt asa multe posibilitati la cum sa faci automatul asta cu stari incat nu are rost sa detaliez

/* TODO [7] */ → lab_mesh_loader.hpp

- Daca ti-a iesit schimbarea intre fete, e timpul pentru animatia propriuzisa
- Codul e destul de simplu, mai greu e sa intelegi ce se intampla (o sa incerc sa explic cum pot, nu e mare lucru)
- Animatia consta in schimbarea pozitiei vertesilor (punctele de intersectie intre linii) la fiecare frame conform cu timer
- Cea mai simpla formula (liniara) este: *noua_pozitie = positia_initiala + timer * cat_se_muta_pe_frame*
- Formula pentru *cat_se_muta_pe_frame* este destul de evidenta (astia toti sunt vec3)
  cat_se_muta_pe_frame = (fata2.position – fata1.position) / durata_animatiei
- Explicatie formula: daca vrei sa ajungi din punctul A in punctul B in N secunde, faci (B-A)/N si adaugi asta la fiecare secunda, deci la momentul T ai pozitia = A + T * (B-A)/N; doar ca noi aici in loc de secunde avem frame-uri
- Asta se face in vertex shader, pentru fiecare vertex
- *durata_animatiei* o hardcodezi tu, e legata de timer (durata mare = slow motion)
- Faza cu vertex shaderul este ca are access doar la un singur vertex, pentru ca GPU lucreaza in paralel
- Deci va trebui sa calculezi la initializare vec3 *cat_se_muta_pe_frame* si sa il trimiti la vertex shader, iar pentru asta trebuie sa adaugi inca 3 floaturi (sau un *glm::vec3*) in *VertexFormat* (sau sa iti faci propriul tau format, whatever you like)

/* TODO [8] */ → lab_mesh_loader.hpp

- Ca sa poti sa calculezi diferenta intre vertexi (*fata2.position – fata1.position*), ai nevoie, evident, de vertexi
- Cand e incarcat modelul, e acolo *std::vector<VertexFormat> vertices* in care sunt stocati vertexi, si care apoi se pierde
- Va trebui sa stochezi undeva vectorul ala pentru fiecare fata (global sau intr o clasa)
- Pe langa asta, dupa ce calculezi diferenta, va trebui sa o trimiti la vertex shader, iar asta se face cu:
  glEnableVertexAttribArray(x); (unde x este layoutul sau pipe-ul pe care se trimite)
  glVertexAttribPointer(x ,3,GL_FLOAT,GL_FALSE,sizeof(VertexFormat), (void)(sizeof(float) * y));* (acelasi x, iar y este offsetul in structura *VertexFormat* de unde incepe stocarea vectorului *cat_se_muta_pe_frame*)
- mai tarziu va trebui sa adaugi layoutul x in vertex shader, TODO [11]

/* TODO [9] */ → main.cpp :: la initializare

- In momentul asta ar trebui sa faci initializarea: calculul *cat_se_muta_pe_frame* intre 2 cate 2 fete (surprise – sleep, annoyed – surprise, sleep - annoyed) pentru fiecare vertex din *vertices* (3 foruri in total sau unul mai mare)
- Partea asta e destul de simpla cu o exceptie: bindarea lui *vao* trebuie facuta dupa ce calculezi alea !!
- Mai exact, partea a2a a functie *void loadObj()* din *lab_mesh_loader.hpp* trebuie apelata dupa ce ai facut tu calculele, altfel nu o sa le ai in vertex shader (evident ca daca o apelezi de 2 ori, prima data o sa fie degeaba)
- Cum ai putea sa faci asta: fie spargi functia aia in 2 functii mai mici si le apelezi pe rand, fie te gandesti la alta varianta (im too lazy for that :P)

/* TODO [10] */ → main.cpp :: in notifyDisplayFrame()

- Dupa ce ai calculat alea, trebuie sa le trimiti la vertex shader
- Chestiile care sunt specifice fiecarui vertex (pozitia, normala, coord textura, diferenta_pe_frame) se numesc atribute si sunt trimise prin setarea de layout sau pipe, cand e generat vao (partea asta am facut-o in TODO [8])

- Chestiile care sunt generale (timer, imaginea textura, matricile) se numesc *uniform* si se trimit cu functiile:
  glUniform1i(glGetUniformLocation(gl_program_shader_curent, "un_int "), variabila_int);
  glUniform3f(glGetUniformLocation(gl_program_shader_curent, "un_vec3"), vec3.x, vec3.y, vec3.z);
- nu uita sa declari variabile cu FIX ACELASI nume din ghilimele in shadere

/* TODO [11] */ → shader_phong_vertex.glsl

- Daca ai trimis bine toate chestiile, ar trebui sa poti acum sa le folosesti in vertex shader sa obtii animatia
- Ar trebui sa ai: pozitia initiala (layout = 0), *cat_se_muta_pe_frame* (layout = x setat de tine), uniform *timer* si cam atat pentru animatie
- Formula de mai sus: *noua_pozitie = positia_initiala + timer * cat_se_muta_pe_frame*
- Abia dupa ce ai noua pozitie, faci si restul calculelor: *world_pos* si *gl_Position*
- Daca ti-a iesit pana aici si n-am uitat eu ceva, ar trebui sa ai o animatie draguta :)

/* TODO [12] */ → main.cpp :: la initializare

- Singura chestie care mai trebuie facuta la partea asta e calcularea normalelor
- Ce inseamna asta: daca iei de exemplu ploapele care se inchid si deschid, fata adormita are ploapele jos, deci normalele cam la 90 grade de orizontala, iar fata treaza are ploapele sus, iar normalele in sus
- Normala e un vec3 normalizat (de lungime 1), in case you didnt know
- Cum iluminarea se bazeaza pe normale, ar trebui sa le calculezi cumva pentru intervalul adormit-treaz
- Cea mai simpla varianta e la fel ca la calculul pozitiei, dupa aceeasi formula:
  *noua_normala = normala_initiala + timer * diferenta_pe_frame* (asta e in vertex shader)
- *diferenta_pe_frame* e tot un vec3, si e egala cu *(fata2.normala – fata1.normala) / durata_animatiei*
- la initializare, calculezi diferenta, o retii in *VertexFormat* si o trimiti la shader; asta inseamna sa mai adaugi un vec3 in *VertexFormat* si un layout = y in bindarea *vao* sa fie trimis la vertex shader

/* TODO [13] */ → shader_phong_vertex.glsl

- In vertex shader calculeaza noua normala, dupa formula de mai sus, si abia apoi calculezi si *world_normal*
- Nu uita sa normalizezi normala noua cu functia *normalize()*
- Si cam asta e tot cred … sper sa nu fi uitat nimic; in momentul asta ar trebui sa ai prima parte a temei completa :) well done

Few remarks:

- Tema nu e mare lucru, trebuie doar sa stii sa jonglezi intre shader si main
- In cazul in care nu esti sigur daca datele au fost trimise la shader, ce poti face e sa vezi ce valoare intoarce functia :
  glGetUniformLocation(gl_program_shader_curent, "variabila_ta")
- daca intoarce -1 inseamna ca nu a gasit *variabila_ta* din shader din 3 motive posibile:
  1) ai uitat tu sa o declari
  2) nu ai pus acelasi nume variabilei in shader si in main
  3) GPU a considerat ca variabila ta e inutila (nu modifica calculele) si a sters-o el la compilarea shaderului
- La unele chestii n-am intrat in detalii de implementare tocmai ca sa fie varietate (we know why)
- Partea a-2a se poate rezolva separat de prima parte, si chair sugerez asta (adica comenteaza codul de la prima parte) pentru ca dureaza foarte mult sa ruleze pana incarca cele trei fete, and we don't have all day
- Ar fi bine sa gandesti partea a2a ca si cum ar fi sincronizata cu prima, ca la sfarsit de tot sa le combini repede

A DOUA PARTE :: Sistemul de particule

/* TODO [1] */ → nicaieri :: chestii generale

- Care e faza cu sistemul de particule: sunt folosite la efecte speciale, unde le misca chestiile in mai multe directii (de ex: flacari, explozii etc.)
- Un mod oarecum simplu de a implementa particule este cu billboard si texturi
- Billboard inseamna ca obiectul e mereu cu fata spre tine, il vezi mereu la fel, oricum s-ar misca el sau tu, e reprezentat 2D
- Din moment ce e mereu 2D, inseamna ca poti sa-l desenezi ca o textura (textura == imagine/poza)
- Ca sa desenezi o textura, ai nevoie de un dreptunghi (pentru ca si poza e un dreptunghi)
- In plus, o sa vrei ca particula sa nu arate mereu ca un patrat, sa aibe unele parti transparente; asta inseamna ca imaginea sa aibe 4 valori (r,g,b,alpha), unde alpha e opacitatea (alpha = 0 nu se vede deloc)
- In fragment shader poti sa elimini un pixel daca are alpha = 0 (asta inseamna alpha discard) cu functia *discard()* ca sa nu mai faci calcule degeaba cu el
- Care e faza cu sistemul de particule (partea 2a): de obicei ai foarte multe particule (gen notele muzicale din tema), si ar dura foarte mult timp sa faci calculele in main, la fiecare frame, si sa mai ai si 60 FPS (adica sa nu se miste in slow motion)
- Asa ca mai bine faci calculele in vertex shader (sau geometry shader, dar asta e pentru avansati :P) pentru fiecare particula, pentru ca GPU lucreaza in paralel (ca la APD)
- Cam atat cu explicatiile, revenim la cod :)

/* TODO [2] */ → main.cpp :: la initializare

- Iti sugerez sa comentezi codul cu animatia fetei, pentru ca nu ai nevoie acum si se incarca foarte greu (dupa cum ai si observat probabil)
- Incarca textura pentru nota muzicala
- Cel mai simplu ar fi sa-ti faci o clasa pentru particula , sau cativa vectori globali, pentru ca o sa trebuiasca sa retii cateva chestii la fiecare particula
- Ai nevoie de:
  pozitia_initiala (*glm::vec3* sau 3 floaturi),
  velocity (e un vec3 care ii spune particulei in ce directie si cu ce viteza sa se miste, 2 in 1 ca la detergenti :P),
  culoarea (presupun ca o sa folosesti o singura poza, si nu sute de imagini doar ca sa schimbi culoarea notei muzicale; asta inseamna ca retii o culoare, vec3 sau vec4 daca vrei si semi-transparenta),
  vao si *num_indices* (ca si la fata, pentru desenare)
- Pe langa clasa de particula o sa mai ai nevoie de un *timer*, pentru animatia particulei
- Presupunem ca miscarea e liniara, deci formula e la fel ca la animatia fetei: *noua_pozitie* = *pozitia_initiala* + *timer* * *velocity*
- Aia o sa faci in vertex shader mai tarziu

/* TODO [3] */ → main.cpp :: la initializare

- Ce urmeaza o sa se faca intrun FOR, pentru fiecare particula
- Te uiti in functia *void loadObj()* din *lab_mesh_loader.hpp* si vezi cum e incarcat un mesh, si dai copy-paste la codul ala
- Singura diferenta este ca va trebui sa scrii tu de mana vectorii *vertices* si *indices* (cu *push_back()*)
- Vectorul *indices* pot sa ti-l dau eu, atat de simplu e: *[0, 1, 2, 1, 3, 2]* (daca ai 4 vertexi) sau *[0, 1, 2, 3, 4 ,5]* (daca ai 6 vertexi)
- In total ar trebui sa ai 2 tringhiuri care formeaza un patrat
- ATENTIE: in pdf-ul din tema zicea ceva de centrul particulei; cel mai simplu mod de a scapa de problema lui, este sa fie mereu in (0,0,0); asta inseamna sa desenezi simetric particula in jurul punctului (0,0,0)
- Si apropo, pe axa Z, particula 2D nu se modifica, deci poti sa pui 0
- Fii atent in ce ordine pui indicii la triunghiuri, trigonometric sau invers, sa fie la fel
- Dupa asta folosesti codul copy-paste ca sa bindezi *vao* si *num_indices*, pe care le retii undeva
- Ai nevoie doar de 2 layout : pozitia (vec3) si coord_textura (vec2)
- Normala in cazul acesta va fi inutila, pentru ca nu vom avea iluminare pe particula (nu are rost)

/* TODO [4] */ → main.cpp :: la initializare

- Daca ai terminat clasa pentru particula si crearea patratului, adauga in FOR o pozitie si o culoare
- Pentru moment pune *for (i=0; i<1; i++)* , pozitia *(0,0,0)* si velocity *(0.1f,0,0)* , iar culoarea nu conteaza
- O sa vrei sa vezi ca merge pentru o singura particula inainte sa treci mai departe
- Mai tarziu o sa trebuiasca sa revii si sa incerci sa hard-codezi valorile pana arata bine :)
- De preferat ar fi la le pui random pozitia, velocity si culoarea

/* TODO [5] */ → main.cpp :: in notifyDisplayFrame()

- La desenare o sa ai un FOR pentru fiecare particula, in care folosesti *vao* si *num_indices*
- Pe langa asta, o sa trimiti ca *uniform*:
  textura (mereu aceeasi, poti s-o pui inafara for-ului), *velocity* si *culoarea* si evident, variabila *timer*
  glUniform1i(glGetUniformLocation(gl_program_shader_curent, "un_int "), variabila_int);
  glUniform3f(glGetUniformLocation(gl_program_shader_curent, "un_vec3"), vec3.x, vec3.y, vec3.z);
- Ca sa modifici pozitia particulei, va trebui sa calculezi *model_matrix* cu functiile *glm::scale(model_matrix,)* si *glm::translate(model_matrix,)*
- Restul uniformelor ar trebui sa ramana la fel ca la animatia fetei
- In momentul asta, daca ai pus bine totul, ar trebui sa ai desenata in mijlocul ecranului nota muzicala, in 3D

/* TODO [6] */ → shader_phong_vertex.glsl

- E timpul pentru vertex shader, o sa faci 2 lucruri aici: sa se miste particula si sa fie 2D (asta inseamna billboard)
- Miscarea e simpla: *noua_pozitie* = *pozitia_initiala* + *timer* * *velocity*
- Nu uita sa declari in vertex shader toate variabilele, toate uniform-urile (cu FIX ACELASI nume ca in *main.cpp*)
- Pentru formula de billboard, va trebui sa cauti tu singur pe net (sorry, dar e un punct forte al temei si iti poate aduce restanta la materie, if you know what i mean :P)
- Ce pot sa fac e sa-ti dau cateva indicii despre cum sa-ti dai seama daca formula gasita e buna sau nu, si cum ar trebui s-o folosesti (este posibil sa fie mai multe formule, eu iti povestesc despre cea gasita de mine)
- Ai in vertex shader linia care ii spune fiecarui vertex cum sa fie afisat:
  *gl_Position = projection_matrix * view_matrix * model_matrix * vec4(in_position,1);*
- *in_position* este de fapt *noua_pozitie*
- formula ta de billboard ar trebui cumva sa schimbe linia aia cu ceva care sa includa cele 4 variabile, dar in alta ordine
- daca formula este buna, ar trebui sa vezi pe ecran efectiv un patrat (nedeformat in niciun fel, chiar un patrat) cu textura de nota muzicala (si care se misca conform cu *timer*, daca folosesti *noua_pozitie*)

/* TODO [7] */ → main.cpp :: la initializare

- in speranta ca ti-a iesit billboard-ul, si particula ta se misca si e 2D, revenim la initializare :)
- dupa cum era in video-ul din tema, particulele porneau din stanga ecranului, de la inaltimi diferite, mergeau spre drapta si aveau diferite culori
- nu e prea mult de spus aici, ori le hard-codezi ori le pui random (la fel si pentru culori)
- spor la treaba :P

/* TODO [8] */ → shader_phong_fragment.glsl

- mergem acum in fragment shader pentru ultimul retus la afisarea particulei
- nu am vem iluminare (pentru ca e inutila) , dar avem 2 uniforme : textura si culoarea
- daca ai deschis poza, ai observat 2 chestii: nota muzicala e neagra, iar backgroundul e alb
- faptul ca nota e neagra face treaba foarte usoara: negru inseamna (0,0,0) ca si vec3, deci poti sa aduni culoarea ta direct
- ce mai trebuie facut e partea cu alpha discard, care mai inseamna maxim 2 linii de cod: inainte sa aduni culoarea ta la textura, verifici ce valoare are textura (e un vec3); daca textura e alba , aplici functia *discard()* si gata
- functia *discard()* e un fel de *return;* si opreste fragment shaderul pentru pixelul respectiv
- si cam asta e tot, well done :)

A TREIA PARTE :: Sincronizare intre animatie si sistemul de particule

/* TODO [∞] */ → main.cpp

- Daca ai ajuns pana aici, si ti-au iesit primele 2 parti, this one is piece of cake :P
- Cateva chestii generale:
- Teoretic sunt 3 intervale de animatie, dar practic sunt 4, + [sleep, sleep] cea in care fata sta adormita un timp
- Particulele sunt desenate doar in intervalele [sleep, surprise] si [surprise, annoyed], deci mai baga un IF acolo
- Daca nu desenezi particulele, nu seta nici uniformele (ajuta la performante)
- Daca faci ceva gen *timer++* , asta NU inseamna + o secunda, ci inseamna + un frame (you can never trust him), iar in functie de cat de rapid e laptopul tau, s-ar putea sa ai surprize daca o sa prezinti pe calculatoarele din laborator (eu am avut la tema 2 :P)
- Cam asta e ... GAME OVER :)