TEMA 5 - Space Shuttle -
Documentație Dincă Vlad Andrei grupa 344

Pentru proiectul 3D am ales să realizez o scenă “din spațiu” în care este reprezentată o navetă spațială prinsă într-o ploaie de meteoriți, scena fiind “iluminată” de către un corp ceresc surprins în dezintegrare.

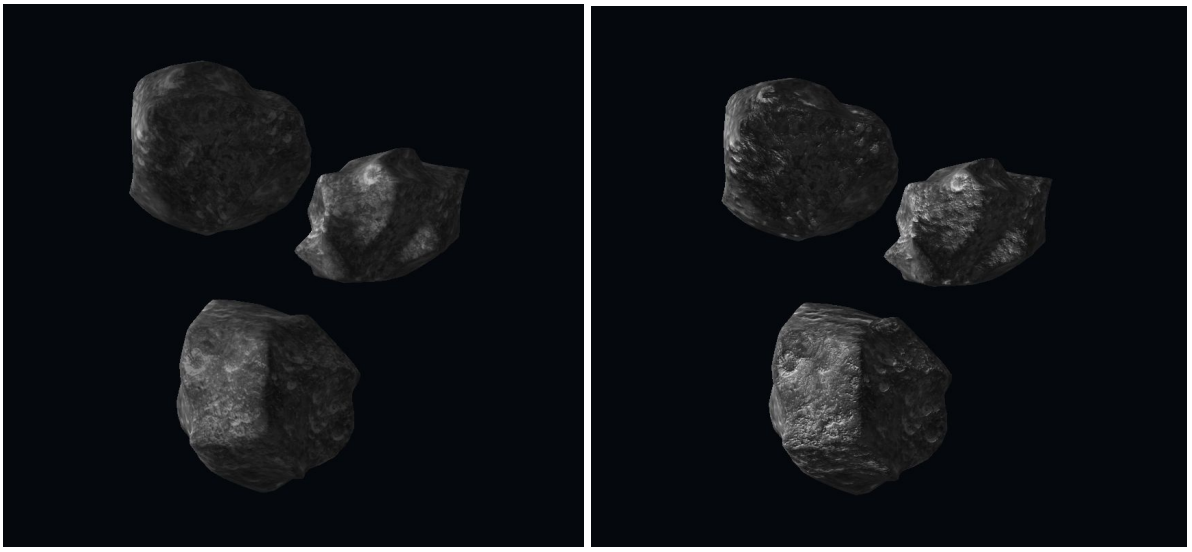
(prezentarea scenei)

Naveta poate fi controlată cu ajutorul săgeților, sus / jos, iar la apăsarea săgeților stânga / dreapta este simulată deplasarea printr-o translație treptată și o rotație, racheta înclinându-se stânga / dreapta, apoi revenind în poziția inițială(unde rotația nu este aplicată).

Clasa *Camera* se ocupă de matricele de vizualizare și proiecție, urmărește game object-ul principal și îi permite jucătorului să se uite în jur, modificând punctul de referință din matricea de vizualizare odată cu deplasarea cursorului mouse-ului.

Stelele din fundal sunt puncte albe, generate cu ajutorul funcției *glm::diskRand(rază)* pentru poziționarea random și au normalele $n = (0,\ 0,\ 1)$ “îndreptate” spre “cameră” și spre sursa de lumină.

Meteoriții sunt generați la poziții *(x, y)* aleatorii și avansează pe axa z “spre cameră” oferind iluzia că racheta înaintează în spațiu. Când aceștia ies din aria vizuală a jucătorului, sunt eliminați din vectorul de obiecte și memoria este eliberată.
Meteoriții sunt aleși aleator dintre 3 tipuri (forme diferite), iar normalele lor sunt calculate la nivel de fragment, fiind mapate în texturi (***bump maps***).



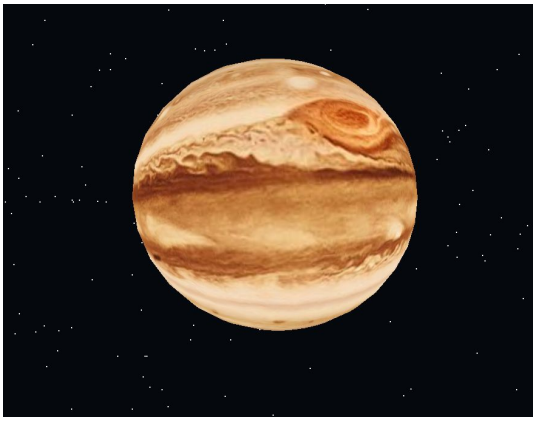
(stânga, cei 3 meteoriți fără bump map. dreapta, meteoriții cu bump map-ul aplicat)

Modelele din scenă sunt salvate în formatul .obj și încărcate cu ajutorul librăriei Assimp, apoi sunt “desenate” cu mai multe shadere specifice:

- *ship_vertex_shader* și *ship_fragment_shader*: țin cont de poziția și normalele la vârfuri, cât și coordonatele de texturare, texturează și aplică formula de iluminare folosind drept normală la suprafață (la nivel de fragment) normala interpolată.

- *meteor shaders*: la fel ca shaderele folosite pentru desenarea navetei, doar că shader-ul de vârfuri primește în plus coordonatele tangentei și bitangentei.
  Aici este calculată matricea schimbării de reper TBN (inversa ei, trecând vectorul de poziție al sursei de lumină, poziția vârfului și poziția observatorului în reperul reprezentat de *(T, B, N)*), iar în *fragment shader* normala la fragment este preluată din textura de normale.

- *planet shaders*, shaderele care nu țin cont de lumină, ci doar texturează obiectul, fiind colorat la “intensitate” maximă. Astfel este oferită impresia că sursa de lumină este chiar corpul ceresc, poziția luminii din scenă fiind aceeași cu centrul planetei.

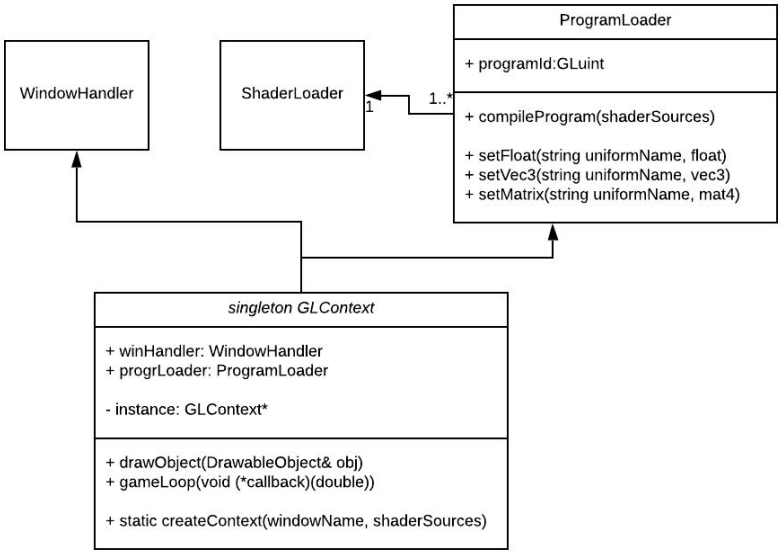


(planeta, “sursa de lumină”, desenată cu shader-ul care nu calculează formula de iluminare)

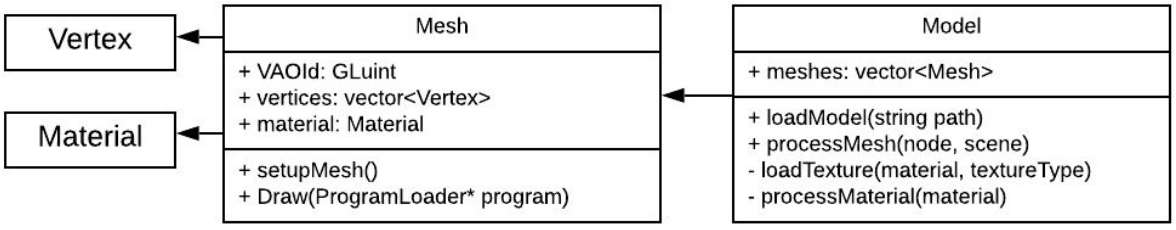
Arhitectura proiectului:

Clasele aplicației pot fi împărțite după funcționalitățile îndeplinite:

- Interacțiunea directă cu OpenGL și GLFW:
  - ***ProgramLoader***: pentru compilarea cu ușurință a unui grup de shadere și schimbarea rapidă a programului curent activ (prin apelarea metodei *program.use()*). De asemenea, permite setarea variabilelor uniforme din shadere (de exemplu *program.setVec3(uniformName, value)*) .
  - ***WindowHandler***: inițializează fereastra, setează modul fullscreen și trimite callback-urile pentru evenimentele de tastatură și de mouse.
  - ***GLContext***: apelează funcțiile de inițializare a celorlalte clase în ordine și setează main game loop-ul



- Manevrarea modelelor 3D, încărcarea lor și a texturilor:
  - ***Model***: apelează funcția de încărcare a librăriei Assimp pentru un model, procesează datele și stochează rețelele de triunghiuri ale modelului într-un vector de *Mesh*
  - ***Mesh***: stochează și desenează o rețea de triunghiuri, păstrând un *VAO* propriu ce va fi activat înainte de apelarea *glDrawElements*.
    Vârfurile sunt ținute într-un vector de structuri, unde structura ***Vertex*** stochează poziția, normala, coordonatele de texturare ale vârfului, cât și tangenta și bitangenta unde este cazul.
    De asemenea, pentru rețelele de triunghiuri ce nu sunt texturate, clasa *Mesh* stochează proprietățile materialului.



- Clase specifice obiectelor din scenă:
  - ***MeteorHandler***: clasă ce se ocupă de manipularea meteoriţilor din scenă, generarea lor, verificarea îndeplinirii condiţiei de eliberare a memoriei, funcţia de update şi draw
  - ***Ship***: clasă handler pentru naveta spaţială
  - ***Camera***: clasă ce updatează matricele de vizualizare şi proiecţie ţinând mereu cont de poziţia observatorului şi a punctului de referinţă. Conţine două metode ce uşurează mişcarea “camerei”: *moveCamera(direction, step)* ce deplasează atât punctul de referinţă, cât şi observatorul în acelaşi timp (util pentru urmărirea navetei) şi *panCamera(deltaX, deltaY)* ce modifică doar poziţia punctului de referinţă atunci când utilizatorul foloseşte mouse-ul pentru a se uita în jur