Practica 2

Informàtica Gràfica

Jaume Pla Ferriol - *u1941090*

24 de novembre de 2024

Figure 1: Aquari: escena P2_script.js

1 Interacció

- **Teclat:**
  WASD: moviment endavant/esquerra/endarrere/dreta.
  ESPAI: flotar cap amunt. **Q** flotar cap avall.
  SHIFT: esprintar.
  1: shading mode **wireframe**.
  2: shading mode **color**.
  3: shading mode **color+wireframe**.
  4: shading mode **normal**.
  5: shading mode **normal+wireframe**.
  (actualment els modes de wireframe tenen molt mala performance)

- **Mouse:**
  MOVIMENT: enfocar camera.
  RODA: control de velocitat de la camera.

- **UI:**
  Shading mode: seleccionar el shading mode (wireframe / color / color+wireframe / normal / normal+wireframe).
  Wireframe opacity: opacitat del wireframe als shading mode 1/3/5
  Fog color: color de la boira/aigua.
  Fog amount: quantitat de boira/aigua
  Fog power: *sharpness* de la boira/aigua.
  Wireframe ignore fog: efecte de la boira/aigua al wireframe als shading mode 1/3/5.
  FOV: field of view (20^{o}-120^{o})
  Save camera POI: guarda una posició i orientació de la camera (point of interest)
  Load camera POI: recupera la posició i orientació de la camera guardada

2 Engine

Per poder anar afegint *features* al llarg del curs, he repertit les funcions en diferents scripts, i he utilitzat classes per poder simplificar la creació d'una escena.

/P2
P2_main.html: document HTML principal
P2_script.js: script d'escena actual
P2_style.css: document d'estil CSS
/WebGL_Engine
/Engine
camera.js *(controla la projecció i moviments de la camera com a mètodes de la classe Camera)*
engine.js *(inicialització de WebGL2.0)*
gameObject.js *(classes GameObject (inicialització dels buffers, model matrix, moviments i draw), ObjectInstance (instàncies de GameObject, s'han d'instanciar per poder-les renderitzar) i ObjectCollection(agrupa les instàncies per shaders, i les dibuixar))*
input.js *(defineix les tecles d'interacció, crea la classe InputParameters que crea els event listeners i actualitzen uniforms)*
player.js *(controla la velociat i estat d'input de moviment)*
renderer.js *(controla els draw modes)*
scene.js *(crea la classe Scene, ajunta les altres classes i té el* ***main draw loop****)*
shaderManager.js *(crea el shader program, inicialitza uniforms)*
ui.js *(fa de link entre els inputs/outputs d'HTML i JS)*
utils.js *(funcions d'utilitat)*
/Resources
blenderToJSON2.py *(script de blender per exportar en format JSON)*
[...]
/modelsJSON
[...]

3 Primitives

El meu objectiu principal en plantejar aquesta pràctica va ser fer una escena senzilla, i aprofitant la malla d'un tauró que ja tenia d'un altre projecte, vaig decidir fer un aquari senzill, fent 3 peixos i 2 malles de corall.
Al preparar aquestes malles els hi vaig assignar vertex colors, però vaig veure que el format OBJ no té suport per vertex colors, així que vaig fer un script amb Python per a Blender per a exportar les malles en format JSON.

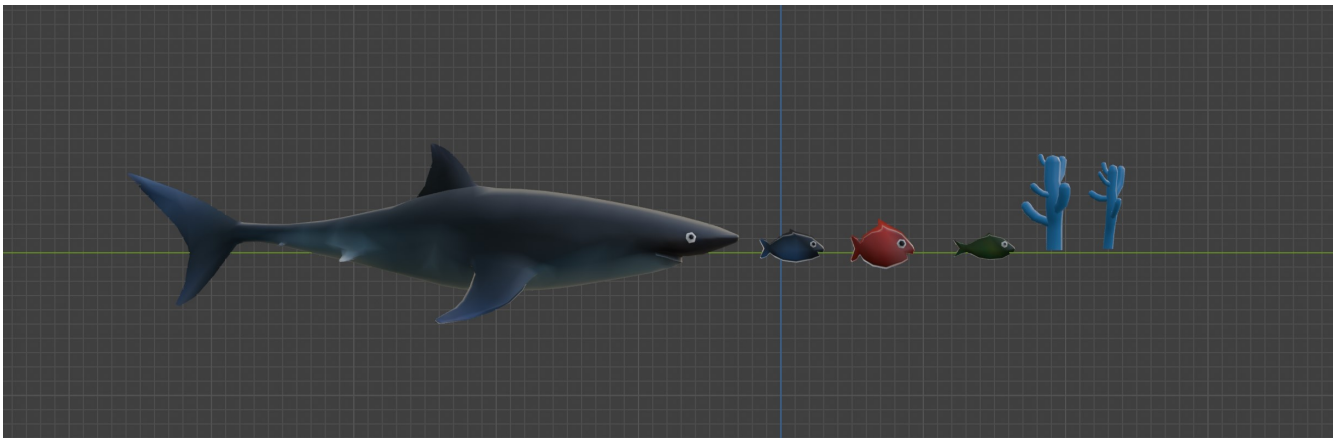

Figure 2: Malles

Figure 3: Malla del tauró d'un projecte anterior

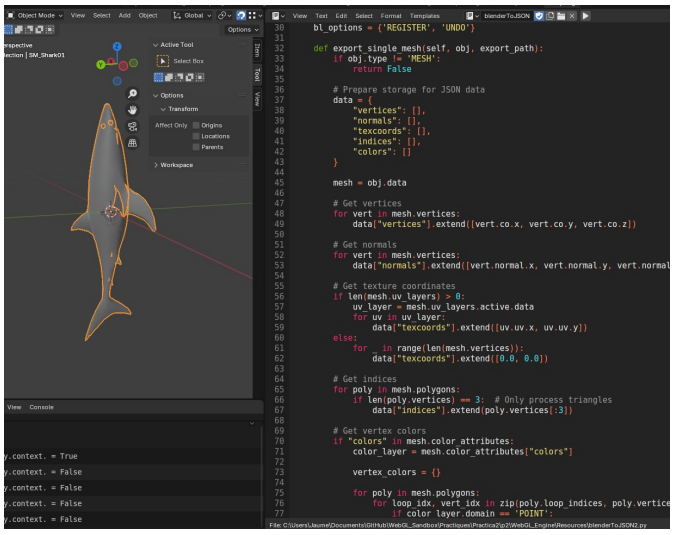

Figure 4: Utilització de l'script propi blenderToJSON2.py

4 Rendering i shaders

He utilitzat les *features* natives de *WebGL2.0* **gl.DEPTH_TEST**, **gl.POLYGON_OFFSET_FILL** i **gl.CULL_FACE** Hi ha un selector de *shading mode*, accessible amb les tecles 1-5 o des del menú *dropdown*, que controlen diferents *gl.drawElements* i canvien alguns *uniforms* dels shaders.

Aquest *shading mode* també controla diferents casos dins el propi shader. He utilitzat el *depth buffer* per a crear

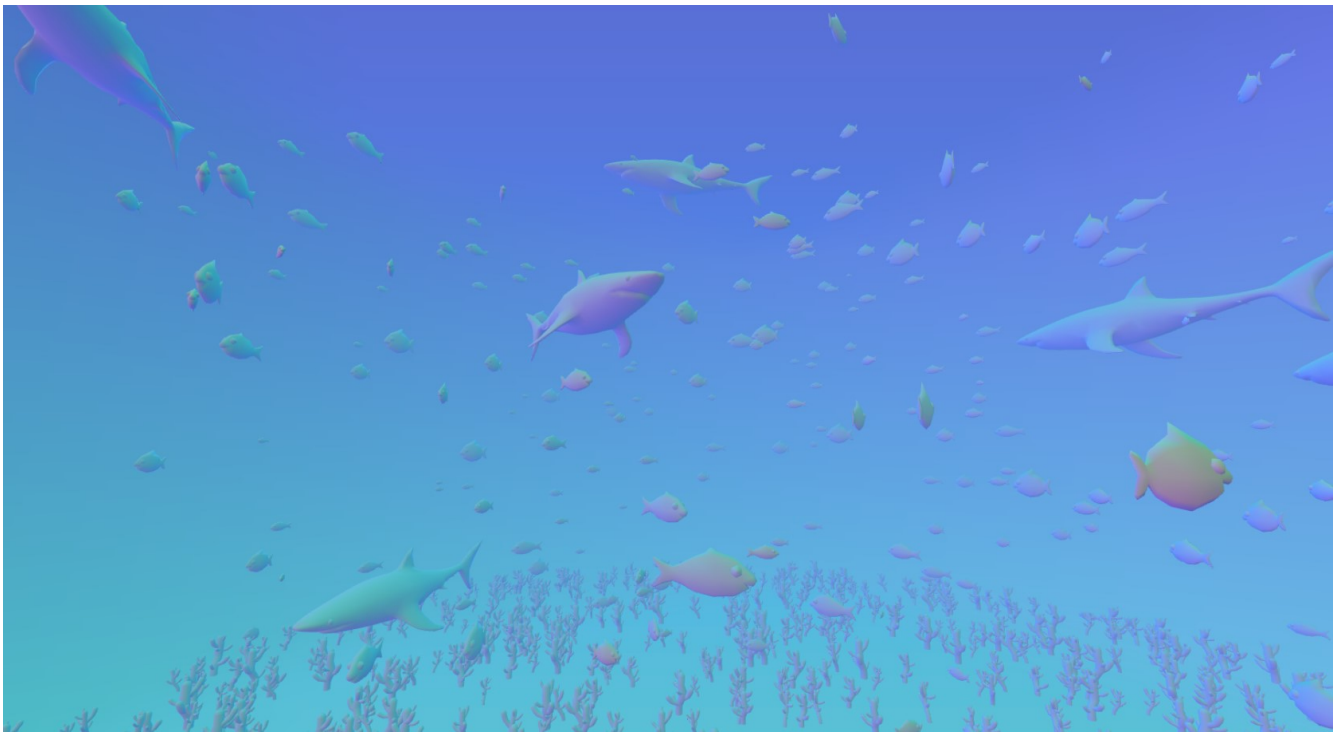

Figure 5: Normal shading

un efecte de boira, per simular de manera barata i estilitzada, una estètica d'interior d'un cos d'aigua.
A més, he aplicat un efecte de *cel shading* al *"base color"* del fragment shader.

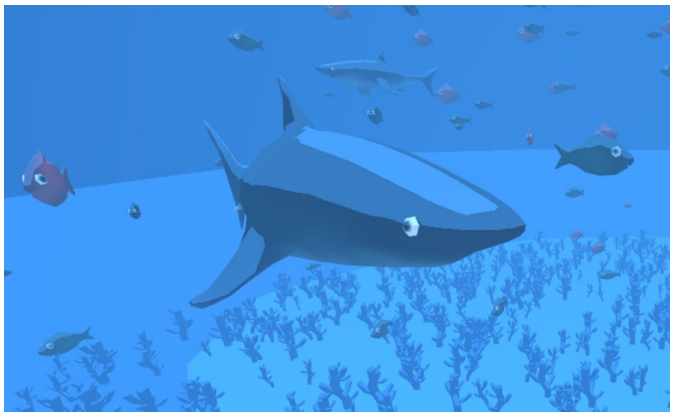

Figure 6: Cel shading amb 2 *steps* de llum

5 Conclusions

Un dels problemes que m'he trobat a l'afegir més malles a l'escena és que el mètode que havia plantejat per a renderitzar els wireframes fa baixar molt el rendiment. Caldria fer un treball d'optimització per a mirar de solucionar això. Diversos aspectes a optimitzar:

- *Uniform update:* com ho tenia plantejat, s'actualitzen diversos uniforms dins del main loop, s'hauria de separar en uniforms que necessiten aquesta actualització i els que només cal inicialitzar-los.

- *Shaders:* he utilitzat diversos *if*s dins del shader, i pel que tinc entès son bastant cars. Una bona solució seria tenir diversos fragment shaders dins del propi HTML, inicialitzar-los a l'script principal, i intercanviar-los a *renderer.js*.

- *Moviment circular:* el moviment dels peixos ara mateix es fa actualitzant la propia *model matrix* de cada objecte, es podria fer aquesta transformació dins del propi vertex shader, per rebaixar el cost de CPU.

- *Occlusion i frustum culling:* es podria implementar un sistema d'*occlusion culling* per amagar les cares amagades per altres objectes, i un sistema de *frustum culling* per descarregar els objectes fora del con de visió.