Tibor Zalabai

Princípy počítačovej grafiky a spracovania obrazu

Tuneler

cvičenie: Štvrtok o 16:00

DOS Tuneler 1991

Princíp tejto DOS hry, staršiu už viac ako 28 je jednoduchý. Hru hrali dvaja hráči, ktorý vidia svoje tanky z hora. Každý z nich má svoju stanicu, na ktorej sa dobíja ich palivo. Pointou hry je sa ,,predierať“ neznámym terénom (musia ničiť prekážky), aby našli nepriateľa a zneškodnili.

DOS Tuneler 1991 vs My Tuneler

Oproti staršej verzií, som sa rozhodol pomeniť viaceré aspekty hry:

- Staršia verzia bola kvôli technológiam obmedzená na 2D a pohľad zhora. Moja verzia je vytvorená v 3D prostredí.
- Kamera v DOS verzií je z vrchu na dol a je rozdelená na 2 časti (2 hráči), ja som vytvoril pohľad, ktorý je z vyvýšeného boku (popísané nižšie). Ďalej nerozdeľujem kameru, je jedna, ktorá je statická.
- V starej verzií nepoznáme lokáciu nepriateľa. V mojej verzií som sa rozhodol, že nepriatelia sa budú vidieť navzájom, ale na to, aby sa k sebe dostali, budú musieť ničiť prekážky (kocky) – taktiež sa medzi nimi môžu ukrývať.
- V mojom tuneleri nie je palivo, ani health bar. Aby hra netrvala príliš dlho, a aby hráči neboli limitovaný palivom, som sa rohodol, že hra bude fungovať na štýl – One shot One kill (jedna rana a príde koniec kola).
- V hre spadajú z neba tzv. Power Ups, ktoré im po ,,picknutí“ pomôžu zvíťaziť. (opísané nižšie).
- Hrá sa na 3 víťazstvá. Prvý, kto dosiahne 3 výhry, je automaticky víťaz.

Ovládanie a inštrukcie hry

Hru hrajú 2 hráči. Prvý hráč (biely) sa pohybuje všetkými možnými smermi pomocou šípkami Hore,Dole,Vpravo,Vľavo a strieľa pomocou Space. Druhý hráč (zelený) sa pohybuje pomocou W,S,A,D a strieľa s tlačítkom Left Shift.
Princíp hry je jednoduchý:
Hráči sa snažia streliť jeden druhého, zvíťaziť kolo, a následne zaznamenať 3 víťazstvá skôr ako nepriateľ. Prekážky môžu použiť ako úkryt, no častokrát treba dbať na to, že aj tieto prekážky sa dajú zničiť. Napomôcť hráčom môžu power ups, ktoré sa náhodne objavujú na mape, a ponúkajú rôzne vylepšenia ako zrýchlenie, protekciu, alebo zväčšenie projektilov.

Štruktúra hry

V hre využívam viacero objektov, ktoré spolu interagujú. Nižšie sa ich pokúsim zhrnúť, a v krátkosti opísať:

Player

- Najdôležitejší objekt v projekte. Je v ňom zahrnutá veľká časť logiky.

- Pomocou scene.keyboard[GLFW_KEY_X] sa hráč pohybuje (pri pohybe sa mení pozícia hráča a jeho rotácia. V hre sú vyriešené aj diagonálne pohyby vpravo-hore, vľavo-dole..
- V tomto .cpp sa riešia aj kolízie. Vymenujem ich a v krátkosti opísem:
1. kol. s projektilom – Pokiaľ do hráča narazí náboj, hra končí a vytvorí sa explózia.
2. kol. s PowerUpom – Ak hráč ,,pickne“ powerup, spustí sa funkcia, ktorá identifikuje, o aký powerup ide, a vykoná príkazy, ktoré daný powerUp aktivujú.
3. kol s nepriateľom – Ak hráč narazí do druhého, automaticky sa zastaví a neprejde ďalej. Algoritmus pre kolízie bude opísaný nižšie.

PowerUp

- Gulička, ktorá padne ,,z neba“ na náhodné miesto na mape. Používam 3 typy powerupov. Každý z nich má účinnosť po dobu 10 sekúnd. Modrý PowerUp zrýchli hráča (pomocou funkcie v Player.cpp, kedy sa zväčší posun hráča pri stláčaní tlačidiel). Zelený vytvorí hráčovi ochranný tórus okolo jeho osi, ktorý pohltí všetky prichádzajúce strely. Červený PowerUp rapídne zväčší veľkosť projektilov (vždy keď sa inicializuje projektil, posielame mu veľkosť projektilu).
- Power Up sa zničí, pokiaľ ho zasiahne projektil (viď. Update v PowerUp.cpp).

Projectile

- Inšpiráciu pre túto triedu som si zobral z projektu gl9_scene od p.Drahoša, ale musel som vymyslieť smer strely (keďže v spomínanom projekte je len strela zdola hore). Preto pri inicializovaní strely definujem premennú directionOfProjectile, podľa ktorej vie, akým smerom má ísť gulička.

HandGun

- Objekt, ktorý je využívaný na to, aby zobrazoval skóre. Príklad: 2 pištole symbolizujú 2 víťazstvá. Ak sa jedná o hráča č.1, tak zbraň bude mať šedé podfarbenie (ak o druhého – pôjde o zelenú farbu).

House

- Objekt reprezentujúci hlavnú prekážku. Pred štartom hry sú náhodne rozhodené kocky po ploche.
- Každá kocka má náhodnú farbu, a pri každej kolízii s projektilom (viď.update) sa jej zmenší veľkosť. Ak je veľkost menšia ako 0.8f, tak sa kocka zničí.

Screen

- tento objekt používam pri zobrazovaní rôznych textov (či už pri výhre hráča v jednom kole, alebo pri výpise totálneho víťaza). Pri potrebe inicializácie tohto textu (napr. v triede Player.cpp) ho zavolám s príslušným parametrom (ktoré potom použijem na to, aby som vedel, aký .obj mám načítať).

Shadow

- objekt, ktorý využívam na tiene. Hlavným atribútom je int_typeOfObj, podľa ktorého vieme, aký .obj mám načítať. Pokiaľ existuje viac inštancií objektu, používam ID (každý z inštancií má unikátne id, podľa ktorého ho vyhľadáme v scéne). Ak už daný objekt získame, zostáva nám zosynchronizovať pohyb objektu a nášho tieňu. To riešime pomocou hierarchických transformácií, ktoré riešim nižšie.

Torus

- ,,Protekcia“ tanku sa vytvorí pri picknutí powerUpu. Tento torus sa zrýchluje exponencionálne s časom (na začiatku je pomalý, tesne pred upršaním rotuje najrýchlejšie). Vo funkcií Update taktiež neustále dostávame objekt hráča, ktorý powerUp získal. Následný synchronizovaný pohyb už riešim pomocou hierarchických transformácií.

Desk

- Je to v podstate stôl, na ktorom sa hra odohráva. Spawne sa na začiatku a celý čas má statickú pozíciu.

Toto bol zoznam všetkých využívaných objektov (a tried). Ostatné triedy sú nevyhnutné pre hru, a ich vytvorenie je inšpirované projektom gl9_scene. Jedná sa napríklad o Camera.cpp, Explosion.cpp, Generator.cpp, Object.cpp, Scene.cpp alebo Tuneler.cpp.

Splnené body zadania

V tejto časti budem opisovať body, ktoré som splnil v zadaní, a postup, akým som ich tvoril.

Use of Texture mapping on 3D geometry.

1. Unique 3D meshes.

- v každej triede, ktorú využívam na tvorbu objektov, som používal buď mnou vytvorené objekty (napríklad objekty v triede Screen.cpp alebo House.cpp), alebo voľno dostupné objekty zo stránok ako free3d.com, turbosquid.com. Hra by nabrala ešte krajší vzhľad, pokiaľ by som použil objekty, ktoré som chcel, problém bol, že veľa z nich bolo platených, alebo nepodporovalo formát .obj.

2. Unique texturing using UV Coordinates

- Tento bod mi zabral väčšiu námahu, pretože nie každý objekt mal k dispozícií textúru. Napokon som do projektu pridal 2 objekty s textúrou. Prvým je HandGun.cpp a druhým Desk.cpp.

Hierarchical object transformation and camera

1. Animated or interactive camera with perspective projection

- Pokiaľ jeden hráč zničí druhého, prichádza naňho spomalený zoom, ktorý trvá až dokým nie je kamera v určitej pozícií. Tento algoritmus sa nachádza v triede Player.cpp

```
scene.camera->position.x = this->position.x ;
scene.camera->position.y = this->position.y - 20.0f;
scene.camera->position.z += 0.05f;
if (scene.camera->position.z > -25.0f){
    runZooming = false;
```

2. Use of hierarchical tranformations in the game scene

- Riešim to napríklad v update (pri getovaní objektu zo scény vo for cykle), pri objekte typu Shadow, alebo Torus. Model Matrix, ktorý prislúcha daného objektu, zmením na ModelMatrix objektu, ktoré má nasledovať. Ešte ho vynásobím posunom, rotáciou a veľkosťou druhého objektu. Príklad:

```
if (player->playerId == this->playerId){
    modelMatrix = player->modelMatrix * glm::translate(mat4(1.0f),
position)
                        * glm::orientate4(rotation)
                        * glm::scale(mat4(1.0f), scale);
```

3. Use of multiple camera view presets.

V triede tuneler.cpp, vo funkcií onKey() zaznamenávam stlačenia používateľov (inšpirácia z gl9_scene). Pokiaľ používateľ stlačí tlačidlo ,,U“, alebo ,,V“ zmení sa pohľad kamery nasledovne:

```
if (key == GLFW_KEY_U&& action == GLFW_PRESS){
  scene.camera->position.x =-0.08;
  scene.camera->position.y  = -6.199;
  scene.camera->position.z  = -61.199;

  scene.camera->back.y =-0.390002;
  scene.camera->back.z = -5.100025;
}
if (key == GLFW_KEY_I&& action == GLFW_PRESS){
  scene.camera->position = { 0.000000, -39.000000, -50.000000};
scene.camera->back = {0,-2.950000 , -3.900000};
}
```

Interaction and simple Game Logic

1. Implementation of basic game logic, game has multiple scenes, is playable and has an ending

- Myslím si, že game logic mám splnenú (keďže hráči sa pohybujú a snažia sa zostreliť jeden druhého)
- Pokiaľ príde k ďalšiemu kolu, znova sa náhodne roztiedia prekážky.
- Koniec hry príde, pokiaľ má hráč 3 víťazstvá.

2. Effective object to object collisions and interactions

- Kolízie riešim vo funkcií update, kde vo for cykle getujem objekty. Ak sa objekt getne, a je to nevyhnutné, tak zisťujem, či prišlo ku kolízií.
- Pri niektorých funkciách musím pridať konštantu pri zisťovaní zrážky, pretože objekty stiahnuté zo stránok neboli vždy konzistentné a ich hranice boli častokrát posunuté. Taktiež je pri niektorých zrážkach objektoch pár-milimetrový zásah do objektu (kvôli tomu, že niektoré stiahnuté objekty mali zložitejšie tvary, a ťažko sa mapovali ich hranice).

Kolízie som riešil nasledovne:

```
if (distance(position, powerUp->position) < powerUp->scale.y)
```

Taktiež by som chcel upozorniť na zaujímavý prístup k riešenia pohybu hráča (napríklad do kocky, alebo do druhého hráča).

- Každý pohyb si uložím do pomocného vector3 temp_position, následne prechádzam do For-cyklu, v ktorom si getnem objekt, a otestujem, či temp_position a position objektu (napr. Kocky) je menšia ako veľkosť objektu. Ak áno, tak viem, že by prišlo ku zrážke, a tento temp_position NEpriradím k reálnemu position. Ak by ani po tomto FOR-cykle neprišlo k zrážke, tak position = temp_position

2. Dynamics cene with game objects being created and destroyed during gameplay

- Objekty vytváram v rôznych častiach hry (pri strelách, pri powerUpoch, pri generovaní kociek, pri vytváraní skóre alebo vypisovaní výsledného textu.
- Ako príklad uvádzam generovanie PowerUpov, ktoré sa spawnú na náhodnú pozíciu. Ako inšpiráciu som si zobral projekt gl9_scene

```
if (time > 7.0f) {
  int randomColor = linearRand(1.0f, 4.0f);
  auto obj = make_unique<PowerUp>(randomColor,i);
  obj->position = position;
  obj->position.x = linearRand(-22.5f, 22.5f);
  obj->position.y = linearRand(-24.5f, 22.5f);
  obj->position.z = -30.0f;
  scene.objects.push_back(move(obj));
  auto shadow = make_unique<Shadow>(2,i);
  scene.objects.push_back(move(shadow));
  i = i + 1;
  time = 0;
}
```

- Taktiež si môžeme všimnúť, že sa v tom momente spawne aj Shadow, ktorý bude nasledovať náš PowerUp (kvôli jeho typu – 2, a unikátnemu ID – i)
- Odstraňujem objekty pomocou return false, alebo pri projektiloch, mu nastavím vek na 100 (podmienka je, že ak je vek väčší ako 2, tak vráti false).

Animation using keyframes, splines or simulation

1. Data driven animations

- V projekte využívam keyframes najmä pri nasledovaní Playera Torusom, alebo nasledovaní Objektu tieňom. Ako pomocné linky a inšpiráciu som študoval http://old.mbsoftworks.sk/index.php?page=tutorials&series=1&tutorial=27, https://www.khronos.org/opengl/wiki/Keyframe_Animation,

- Riešim to tak, že vo for cykluse si získam objekt, ktorý potrebujem a následne nad ním vykonám nasledovný algoritmus:

```
modelMatrix = player->modelMatrix * glm::translate(mat4(1.0f), position)
                    * glm::orientate4(rotation)
                    * glm::scale(mat4(1.0f), scale);
```

kde modelMatrix patrí objektu, ktorý chceme posúvať podľa daného objektu (v tomto prípade player)

2. Basic simulated animation

- Tento bod som vyriešil prostredníctvom gravitácie. Gravitácia sa spustí na začiatku, po inicializácií.
- Kód je nasledovný:

```
void PowerUp::gravity(Scene &scene, float dt, float floor){
    if (!iamDown && position.z <= floor){       //ked klesam
        gravityDown += dt*16;                   //rychlost sa exponen. zvysuje
        position.z += 0.02*gravityDown;         //menim poyiciu na zaklade rychlosti
        if (position.z >= 0){                   //ak nie som na zemi
            iamDown = true;
            gravityUp = gravityDown/1.3;        //rychlost smerom hore by uz bola mensia
        }
    }
    else if (iamDown){                          //ked stupam
        position.z -= gravityUp*0.015;          //zmensuje sa rychlost
        gravityUp -=dt*21;
        if (gravityUp < floor + 0.03){          //ak som uz hore ( a idem zas klesat )
            iamDown = false;
            gravityDown = 0;
        }
     }
}
```

(kod je popísaný v riadkoch)

3. Procedurally driven animation

- Pohyby rieším v Player.cpp, a každý príklad pohybu je riešený nasledovne:

```
...
...
else if (this->playerId == 1 && scene.keyboard[GLFW_KEY_DOWN] &&
scene.keyboard[GLFW_KEY_LEFT]) {
  temp_position.x += speedOfPlayer1 * dt;
  temp_position.y -= speedOfPlayer1 * dt;
  rotation.y = -PI / 1.5f;
} else if (this->playerId == 1 && scene.keyboard[GLFW_KEY_LEFT] ) {
  temp_position.x += speedOfPlayer1 * dt;
  // rotation.z = -PI/4.0f;
  rotation.y = -PI / 2.0f;
} else if (this->playerId == 1 && scene.keyboard[GLFW_KEY_RIGHT] ) {
  temp_position.x -= speedOfPlayer1 * dt;
  rotation.y = PI / 2.0f;
...
...
```

Ako som spomínal, kvôli nárazom do kocky, rieším to, aby sa hráč nedostal do kocky. Preto sa všetko najprv zapamätá do premennej, ktorú neskôr otestujem, či sa dostala do kolízie. Ak nie, priradím temp_position do Player->position.

Lighting with multiple light sources

1. Diffuse scene lighting with materials

- Tento bod riešim pomocou materiálov, ktoré sú priradené objektom.
- Na začiatku si načítam .mtl súbor (ktorý obsahuje 6 položiek (Ns,Ka,Kd,Ke,Ni), ktoré sa skladajú z vektorových zložiek x, y, z. (viď súbor testAhoj.mtl)
- Potom si v triede (pre objekt načítam tento súbor:

```
ifstream myStream ("testAhoj.mtl", std::ifstream::binary);
tinyobj::LoadMtl(material_map, material, myStream);
```

Vo funkcií render si vytvorím pomocné zložky (vypracoval som ich po dlhšom študovaní stránky https://learnopengl.com/Lighting/Materials s K. Tóthom)

```
vec3 ambient = vec3(material.data()->ambient[0], material.data()->ambient[1], material.data()->ambient[2]);
vec4 diffuse = vec4(material.data()->diffuse[0], material.data()->diffuse[1], material.data()->diffuse[2], 1.0f);
vec3 specular = vec3(material.data()->specular[0], material.data()->specular[1], material.data()->specular[2]);
float shininess = material.data()->shininess * 128;
shader->setUniform("MaterialAmbient", {ambient.x, ambient.y, ambient.z});
shader->setUniform("MaterialDiffuse", {diffuse.x, diffuse.y, diffuse.z, 1.0f});
shader->setUniform("MaterialSpecular", {specular.x, specular.y, specular.z});
shader->setUniform("MaterialShininess", shininess);
```

Následne tieto zavolania vyvolajú metódy v shadery diffuse_frag.glsl, kde som doplnil údaje podľa tutoriálu už zo spomínaného linku.

- Double lightning mám vyriešené tak, že si v scéne definujem druhé svetlo, a následne ho vo funkcií Render v triedach zavolám.

```
Scene.h
glm::vec3 lightDirection2{-0.92f, 0.36f, -0.37f};
glm::vec3 lightColor2 = glm::vec3(0,1,0);

Trieda.cpp
shader->setUniform("LightDirection2", scene.lightDirection2);
shader->setUniform("LightColor2", scene.lightColor2);
```

2. Correct Phon lighting with multiple light sources

- Tento bod som považoval za najťažší zo všetkých, pretože jeho štúdia a implementácia mi zabrala najviac času. Phong zo začiatku vôbec neštudoval, ale po viacerých odkomunikovaných mailov s p. Hudecom sme nakoniec našli riešenie. Z pomocných linkov sme si naštudovali teoretickú, a aj implementačnú časť (links - https://www.opengl.org/sdk/docs/tutorials/ClockworkCoders/lighting.php https://en.wikibooks.org/wiki/GLSL_Programming/GLUT/Smooth_Specular_Highlights http://www.ozone3d.net/tutorials/glsl_lighting_phong.php). Museli sme upraviť súbor diffuse_frag aj diffuse_vert, a podopĺnať určité funkcie a príkazy (u niektorých stále neviem na 100% povedať ich funkcionalitu).

3. Shadows or reflections using any approach you can think of

- Snažil som sa vytvoriť tiene pomocou metódy Shadow Mapping (sledovali sme postup v tutoriáloch openGL – Shadow Mapping). Problém bol v tom, že mi zatienilo celú mapu, a nie konkrétne objekty, a keďže sme si s tým nevedeli dať rady, išli sme nasledovnou cestou: Vytvorili sme si triedu Shadow, ktorá bude reprezentovať tiene. Pri každej inicializácií tieňu sa do parametre konštruktora dostane číslo, ktoré hovorí, aký .OBJ súbor má mať daný tieň. Následne mu nastavím scale.z = 0, keďže tieň je 2D. Synchronizovaný pohyb s objektom (ktorému robí tieň) je robený tak, že si vo FOR-cykle getnem ten objekt, ktorý potrebujem (pokiaľ je viac inštancií toho objektu, tak využívam uniqueID – môj typ a môj unique_id sa musí rovnať objektovému typu a objektovému id). Potom je už pohyb zosynchronizovaný pomocou hierarchických transformácií.

Zhrnutie

Tento projekt sa mi páčil, pretože som sa naučil aspoň z časti pracovať s knižnicou OpenGL, dokázal som si viac predstaviť fungovanie hier (pretože veľká časť je vytvorená prostredníctvom tejto knižnice). Taktiež sa mi páčilo, že sme na cvičeniach pracovali s vecami, ktoré sme neskôr mohli využiť v projekte. Ako jednu z výhod považujem to, že sme mali vzorovú hru – gl9_scene, podľa ktorej sme mohli nájsť súvislosti a model hry. Tým, že táto knižnica je náročná, som vďačný za to, že sme mali povolené spolupracovať (a tým si zazdielať nejaký algoritmus). Preto som celý semester pracoval na

projekte spolu s K. Tóthom, s ktorým som objavoval a zisťoval aj najťažšie podmienky projektov. Taktiež vďačím p. Hudecovi, ktorý v relatívne rýchlo odpisoval na naše maily a otázky, a vždy sa snažil opísať nám riešenie, a často k nim aj priložiť užitočné linky.

Obrázok je na poslednej strane