Finálna správa k projektu z PPGSO

Adam Rybanský

FIIT STU

Akademický rok 2020/2021

Čas cvičení : utorok 18:00

Stručný opis riešenia

Názov projektu je Go-kart aréna. Projekt pozostáva z 2 scén. Scéna je reprezentovaná triedou, ktorá obsahuje všetky objekty, svetlá, sily , a ostatné veci, ktoré sa používajú v danej scéne.

Triedy / moduly

Trieda main spúšťa celý program , prepína medzi scénami, zachytáva input z klávesnice a myši a posiela ho do scén, odkial ho berú a spracovávaju triedy Camera a Player.

Existujú dve abstraktné triedy, Camera a Object. Z triedy Camera dedia triedy ThirdPesonCamera a FirstPersonCamera. Z triedy Object dedia skoro všetky ostatné triedy, okrem triedy ObjectSpawnShuffler, ktorá iba pomáha pri vytáraní normálnych objektrov. Všetky ostatné triedy teda reprezentujú fyzické objekty viditeľné v scéne.

Projekt je štruktúrovaný tak, že triedy sú rozdelené do priečinkov Scene1 a Scene2, podľa toho, do ktorej scény patria. Iba triedy main, Scene, Object a Camera nie sú ani v jednom, lebo sa využívajú v oboch.

Do priečinku so shadermi som pridal myshader_frag.glsl, myshader_vert.glsl , light_cube_frag.glsl a light_cube_vert-glsl

Ovládanie

1.Scéna :

pohyb – šípky

otáčanie – myš

interakcia s objektami – F

2. Scéna :

pohyb – šípky doľava a doprava

skok – SPACE

menenie pohľadu kamery – E

zapnutie a vypnutie vetra – W

návrat do scény 1 – B

znovunačítanie scény 2 - R

Opis scén

Scéna 1 je štvorcová miestnosť, ohraničená 6 stenami (trieda Wall). Kamera je z pohľadu 1. osoby, je teda aj hráčom (trieda FirstPersonCamera) Pohybuje sa šípkami, a otáča myšou.

V scéne je uplatnený Phongov osvetľovací model, nachádzajú sa tu 2 lampy (trieda PointLight), na pravej a ľavej stene .

Ďalej tu máme 3 kusy nábytku, každý z iného materiálu (triedy Crate, Barrel a Table). Tieto objekty sa nachádzajú vždy na inom mieste (ale na zemi) , o to sa stará trieda ObjectSpawnShuffler. Na zadnej stene sa nachádzajú 3 gule triedy Sphere, ktoré sa pohybujú navzájom okolo seba, uplatňujú hierarchickú transformáciu. Na prednej stene sa nachádzajú 2 objekty, jeden je vypínač (LightSwitch) , ktorý mení farbu svetla. Druhým sú dvere (Door) ktoré prepnú scénu na scénu 2. Aby tieto objekty vykonali svoje funkcie, musí hráč stáť vedľa nich, byť otočený smerom ku nim, a stlačiť F.

Scéna 2 je dráha, na ktorej sa nachádza hráč (trieda Player)– červené auto. Pohybuje sa doľava a doprava po ceste (trieda Road). Cesta je vpravo a vľavo ohraničená stenami (trieda Mantinel). Textúra na ceste a stenách sa posúva, aby to vyzeralo že hráč sa hýbe dopredu. Na opačnom konci cesty je tma (trieda Darkness), aby scéna budila dojem tunelu. V tme sa nachádza neviditeľný objekt Generator, ktorý spawnuje prekážky.

Scéna obsahuje jednoduché directional light, ako v gl9 (a používa diffuse_vert.glsl a diffuse_frag.glsl).

Kamera je reprezentovaná triedou ThirdPersonCamera. Nedá sa ovládať hráčom, ale dá sa zmeniť uhol pohľadu, stačením E. Prednastavené uhly pohľadu sú : zo šikma, zo zadu, zhora, a zhora s animáciou, kedy kamera robí kruhy okolo hráča.

Generátor každú polsekundu spawnne prekážku typu TrafficCone alebo Tire. TrafficCone je dopravný kužel, ktorý stojí na mieste a posúva sa smerom k hráčovi. Keď doňho hráč narazí, odletí (závisí či narazí z predu, z ľava alebo z prava). Tire je kotúľajúca sa pneumatika, ktorá okrem toho, že sa posúva smerom k hráčovi, sa aj hýbe doprava

a doľava a odráža sa od stien. Keď do nej hráč narazí z predu, tiež odletí, a keď narazí z boku, odrazí sa ako od steny.

Hráč môže stlačiť SPACE aby vyskočil, a kým je vo vzduchu, nemôže naraziť do prekážok. To čo ho pritiahne naspäť je gravitácia.

Hráč môže stlačiť W aby zapol alebo vypol vietor. Vietor ho posúva doprava alebo doľava, náhodnou silou.

Po minúte od začiatku scény Generátor vygeneruje cieľ (trieda Finish), ktorý sa tiež posúva k hráčovi. Keď doňho hráč narazí, hra sa skončila. Hráč ale môže kedykoľvek stlačiť R aby scénu znova načítal alebo B aby sa vrátil do prvej scény.

Obrázky riešenia

Ku každému bodu zo zadania vysvetlím, kde v kóde sa nachádza, alebo dám rovno screenshot. Screenshoty toho, ako to vyzerá v hre, dávať nemusím, to bude vidno na videu.

- Použitie mapovania textúr na 3D geometriu **[4p]**
  - Unikátne 3D trojuholníkové siete (2p)
  - Unikátne textúrovanie pomocou uv koordinátov (2p)

takmer v každom objekte, tu je príklad:

```
if (!shader) shader = std::make_unique<ppgso::Shader>(diffuse_vert_glsl, diffuse_frag_glsl);
if (!texture) texture = std::make_unique<ppgso::Texture>(ppgso::image::loadBMP( bmp: "TrafficCone.bmp"));
if (!mesh) mesh = std::make_unique<ppgso::Mesh>("TrafficCone.obj");
```

- Transformácie kamery **[6p]**
  - Kamera s perspektívnou projekciou (1p)
  - Animovaná kamera (2p)
  - Použitie viacerých prednastavených pohľadov kamery (1p)

V triede ThirdPersonCamera:

```
//pressing E swaps the camera positon
if (keyboard[GLFW_KEY_E]) {
    if (glfwGetTime() - cooldown > 0.5)
    {
        cameraMode++;
        cooldown = glfwGetTime();
    }
}

if (cameraMode > 3)
    cameraMode = 0;

if (cameraMode == 0) {              //default position
    position = { a: 0.0,  b: -10.0,  c: -10.0};
    front = { a: 0.0,  b: 1.0,  c: 1.0};
} else if (cameraMode == 1) {       //camera is behind player and follows him left and right
    position = { a: 0.0,  b: -12.0,  c: -5.0};
    front = { a: 0.0,  b: 2.0,  c: 1.0};
    position.x = playerPosition.x;
} else if (cameraMode == 2) {       //camera is above player
    position = { a: 0.0,  b: -2.5,  c: -15.0};
    front = { a: 0.0,  b: 0.15,  c: 1.0};
} else if (cameraMode == 3) {       //camera is above player and rotates around him in circle
    float t = (float) glfwGetTime();
    position = { a: -2 * sin(t) + playerPosition.x,  b: 3 * cos(t) - 4.5f,  c: -12.0};
    front = { a: 0.0,  b: 0.15,  c: 1.0};
    if (position.x > 4.5)
        position.x = 4.5;
    else if (position.x < -4.5)
        position.x = -4.5;
}
```

- Transformácie kamery **[6p]**
  - Interaktívna kamera (2p)

-V triede FirstPersonCamera

```
void FirstPersonCamera::update(float dt) {
    processKeyboardMovement(dt);
    processMouseMovement(dt);
    updateCameraVectors();

    //cant move up and down
    position.y = 1;

    //cant move through walls
    if (position.x > 9)
        position.x = 9;
    else if (position.x < -9)
        position.x = -9;
    else if (position.z > 9)
        position.z = 9;
    else if (position.z < -9)
        position.z = -9;

    viewMatrix = lookAt(position, center: position + front, up);
}
```

- Logika scény **[5p]**
  - Implementácia virtuálnej scény s vzájomným vzťahom medzi objektami (1p)
  - Zmena scény a viacero virtuálnych oblastí (2p)
    - Aspoň 2 rôzne scény
  - Scéna bude mať logické spodok a pozadie (obloha, strop, steny…) (1p)
  - Prezentované demo musí mať logický začiatok a koniec (1p)

-K tomuto nemám moc čo dodať, bude to najlepšie vidno na videu. Možno dám príklad na vzájomné vzťahy medzi objektami – Generátor vytvára pneumatiky, pneumatiky sa odrážajú od mantinelov, hráč môže naraziť do pneumatiky.

- Objekty a interakcia **[6p]**
  - Dynamická scéna s objektami, ktoré sú vytvorené a zaniknú v čase(1p)
    - Aspoň 2 rôzne typy objektov

TraficCone a Tire sa vytvárajú takto:

```
if (time > 0.5) {

    //what object will generate - 0 for Tire, 1 for TrafficCone
    int object_type = static_cast<int>(glm::linearRand(0, 2));

    if (object_type == 0) {
        auto obj = std::make_unique<Tire>();
        obj->position = position;
        obj->position.x += glm::linearRand(-6.0f, 6.0f);
        obj->position.z -= 1.0f;
        scene.objects.push_back(move(obj));
        time = 0;
    }
    else if (object_type == 1) {
        auto obj = std::make_unique<TrafficCone>();
        obj->position = position;
        obj->position.x += glm::linearRand(-6.0f, 6.0f);
        scene.objects.push_back(move(obj));
        time = 0;
    }
}
```

a zanikajú takto :

```
bool TrafficCone::update(Scene &scene, float dt) {
    // Delete when alive longer than 10s or out of visibility
    age += dt;
    if (age > 10.0f || position.y < -10) return false;
```

```
// Use iterator to update all objects so we ca
auto i = std::begin( & objects);
while (i != std::end( & objects)) {
    // Update and remove from list if needed
    auto obj = i->get();
    if (!obj->update( & *this, dt))
        i = objects.erase(i); // NOTE: no need
```

- Objekty a interakcia **[6p]**
  - Procedurálne generovaná scéna (2p)
    - Obmedzenia a lokácia objektov definivaná deterministickým algoritmom

Trieda ObjectSpawnShuffler (a asi aj Generator) :

```
void ObjectSpawnShuffler::shuffle(Scene& scene){
    srand ( _Seed: static_cast <unsigned> (time( _Time: 0)));
    //choose random spawn positions for every object, if they collide, try again
    while (true) {
        for (auto &obj : objects) {
            obj->position.x =  -7.0f + static_cast <float> (rand()) /( static_cast <float> (RAND_MAX/(7.0f-(-7.0f))));
            obj->position.z =  -7.0f + static_cast <float> (rand()) /( static_cast <float> (RAND_MAX/(7.0f-(-7.0f))));
        }
        bool good = true;
        for (auto &obj1 : objects) {
            for (auto &obj2 : objects) {
                if (obj1.get() != obj2.get()) {
                    if (distance(obj1->position, obj2->position) < 4) {
                        good = false;
                        break;
                    }
                }
            }
        }

        if (good == true)
            break;
    }
    //add objects to the scene
    for (auto &obj : objects) {
        scene.objects.push_back(move(obj));
```

- Objekty a interakcia **[6p]**
  - Efektívna kolízia medzi objektami(3p)
    - Dynamická odozva na kolíziu

Kolízia môže nastať medzi hráčom a pneumatikov alebo dopravným kuželom. Efektívna je preto, že tieto objekty majú svoj BoundingBox, ktorý keď je prekročený, zavolá sa metóda collide() danej triedy.

```
//bounding box for collision purposes
struct boundingBox{
    float min_x;
    float min_y;
    float max_x;
    float max_y;
}boundingBox;

//constants to help determine size of object
float ySizeConst;
float xSizeConst;
```

```
void Object::updateBoundingBox() {
    boundingBox.min_y = position.y + ySizeConst;
    boundingBox.max_y = position.y - ySizeConst;
    boundingBox.min_x = position.x + xSizeConst;
    boundingBox.max_x = position.x - xSizeConst;
}
```

```
// If player hits an obstacle, call the collide metod of the obstacle
auto trafficCone = dynamic_cast<TrafficCone*>(obj.get());
if(trafficCone && airborne == false && trafficCone->collided == false) {
    if (trafficCone->boundingBox.max_y <= boundingBox.min_y && trafficCone->boundingBox.max_y >= boundingBox.max_y)
        if ((trafficCone->boundingBox.min_x >= boundingBox.max_x && trafficCone->boundingBox.min_x <= boundingBox.min_x ) ||
        (trafficCone->boundingBox.max_x >= boundingBox.max_x && trafficCone->boundingBox.max_x <= boundingBox.min_x))
            {
                if (trafficCone->boundingBox.max_y >=  boundingBox.min_y -1.0)
                    trafficCone->collide( collisionType: "FRONT");          //collision from front
                else {
                    if (trafficCone->position.x < position.x)
                        trafficCone->collide( collisionType: "LEFT");       //collision from left
                    else
                        trafficCone->collide( collisionType: "RIGHT");      //collision from right
                }
            }
}
```

```
void TrafficCone::collide(const std::string& collisionType) {
    collided = true;
    if (collisionType == "FRONT"){
        position.z -= 0.75f;
        speed.z -= 3.0f;
        rotMomentum.x = -5.0f;
    }
    else if (collisionType == "LEFT"){
        position.x -= 0.5f;
        speed.x -= 2.0f;
        rotMomentum.z = 3.0f;
    }
    else if (collisionType == "RIGHT"){
        position.x += 0.5f;
        speed.x += 2.0f;
        rotMomentum.z = -3.0f;
    }
}
```

Táto metóda spĺňa aj ďalší bod:

- Transformácie a animácie **[9p]**
  - Procedurálna animácia (2p)
    - Uzavretá metóda s parametrami
    - Logiké vetvenie

- Transformácie a animácie **[9p]**
  - Základná simulácia s aspoň 2ma silami s použitím vektorovej algebry (2p)
    - Napr. gravitácia + vietor

Presne gravitáciu a vietor mám implementovanú, Jediný object na ktorý obe pôsobia je hráč.

```
//Wind
glm::vec3 wind = {0,0,0};
int windDirection = 1;

//Gravity
glm::vec3 gravity = {0,0,5};
```

```
// If wind is toggled, adjust position
if (countWithWind == true)
    position += scene.wind * dt;

//Jump
if((scene.keyboard[GLFW_KEY_SPACE] || goingUp == true) && falling == false) {
    if (goingUp == true)
        position.z -= 10 * dt;
    else
        goingUp = true;
    airborne = true;
}
if (position.z < -3) {      //this is the "peak" of the jump
    position.z = -3;
    goingUp = false;
    falling = true;
}

if (airborne == true && goingUp == false)       //when player reaches the "peak of the jump", gravity will start working on h
    position += scene.gravity * dt;

if (position.z > 0) {       //when player reaches the ground, gravity will stop working again
    position.z = 0;
    airborne = false;
    falling = false;
}
```

- Transformácie a animácie **[9p]**
  - Hierarchická transformácia objektov
    - Využitie hierarchickej transformácie v scéne (2p)
      - Aspoň 2 levely hierarchie s 3mi objektami
      - Použitie kompozície maticových transformácii

Trieda Sphere

```
//There are 3 objects, each has its own update function
int sphereNum;
void myUpdate1(float dt);
void myUpdate2(float dt, glm::mat4 modelMatrixParent);
void myUpdate3(float dt, glm::mat4 modelMatrixParent);
//object depending from this one
std::unique_ptr<Sphere> child;
```

```
bool Sphere::update(Scene &scene, float dt) {
    //the first object updates all 3
    myUpdate1(dt);
    child->myUpdate2(dt, modelMatrixParent: modelMatrix);
    child->child->myUpdate3(dt, modelMatrixParent: child->modelMatrix);
    return true;
}
```

```
//update for the second object
void Sphere::myUpdate2(float dt,glm::mat4 modelMatrixParent) {
    auto t = (float) glfwGetTime();
    position = { a: 0, b: 4, c: 0};
    rotation = { a: 0, b: -4,  c: t*2};

    glm::mat4 ROTATION_MATRIX = translate( m: glm::mat4{ s: 1.0f}, v: {rotation.x, rotation.y, c: 0})
                              * rotate( m: glm::mat4{ s: 1.0}, rotation.z, v: { a: 0, b: 0, c: 1})
                              * translate( m: glm::mat4{ s: 1.0f}, v: {-rotation.x, -rotation.y, c: 0});

    modelMatrix = modelMatrixParent *
                  translate( m: glm::mat4{ s: 1.0f}, this->position)
                  * ROTATION_MATRIX
                  * glm::scale( m: glm::mat4{ s: 1.0}, this->scale);
}
```

- Transformácie a animácie **[9p]**
  - Animácia na základe dát uložených v kľúčových snímkoch a interpolácie (3p)
    - Kľúčový snímok reprezentujúci dátovú štruktúru uchovávajúcu transformácie

Nemám.

- Osvetlenie za pomoci viacerých svetelných zdrojov **[7p]**
  - Správne kombinovať scénu s difúznymi materiálmi a viacerými zdrojov svetla (2p)

Mám na to vytvorený vlastný shader myshader_vert.glsl a myshader_frag.glsl

- Osvetlenie za pomoci viacerých svetelných zdrojov **[7p]**
  - Menitelná farba osvetlenia (1b)

Trieda LightSwtich volá metódu changeColor() triedy PointLight

```
void PointLight::changeColor() {
    colorMode++;
    if (colorMode > 3)
        colorMode = 0;

    if (colorMode == 0)
        color = { a: 1.0, b: 1.0, c: 1.0};          //white
    else if (colorMode == 1)
        color = { a: 0.0, b: 0.0, c: 1.0};          //blue
    else if (colorMode == 2)
        color = { a: 1.0, b: 0.0, c: 0.0};          //red
    else if (colorMode == 3)
        color = { a: 0.0, b: 0.0, c: 0.0};          //no light
}
```

- Osvetlenie za pomoci viacerých svetelných zdrojov **[7p]**
  - Správny Phongov osvetlovaci model s viacerými zdrojmi svetla (2p)
    - Správne tlmenie svetla na základe hĺbky
    - Aspoň tri materiály a tri komponenty svetla
    - Správne kombinovať materialy s svetelné komponenty

Príklad materiálu:

```
//Brass
material.ambient = { a: 0.329412f,  b: 0.223529f,  c: 0.027451f};
material.diffuse = { a: 0.780392f,  b: 0.568627f,  c: 0.113725f};
material.specular = { a: 0.992157f,  b: 0.941176f,  c: 0.807843f};
material.shininess = 0.21794872f;
```

Príklad napĺňania shaderu v metódach render():

```
//Light1
auto lightSource1 = dynamic_cast<PointLight*>(scene.pointLights.front().get());
shader->setUniform( name: "light.position",lightSource1->position);
shader->setUniform( name: "light.color",lightSource1->color);
shader->setUniform( name: "light.ambient",  lightSource1->ambient);
shader->setUniform( name: "light.diffuse",  lightSource1->diffuse);
shader->setUniform( name: "light.specular", lightSource1->specular);
shader->setUniform( name: "light.constant", lightSource1->constant);
shader->setUniform( name: "light.linear", lightSource1->linear);
shader->setUniform( name: "light.quadratic", lightSource1->quadratic);
//Light2
auto lightSource2 = dynamic_cast<PointLight*>(scene.pointLights.back().get());
shader->setUniform( name: "light2.position",lightSource2->position);
shader->setUniform( name: "light2.color",lightSource2->color);
shader->setUniform( name: "light2.ambient",  lightSource2->ambient);
shader->setUniform( name: "light2.diffuse",  lightSource2->diffuse);
shader->setUniform( name: "light2.specular", lightSource2->specular);
shader->setUniform( name: "light2.constant", lightSource2->constant);
shader->setUniform( name: "light2.linear", lightSource2->linear);
shader->setUniform( name: "light2.quadratic", lightSource2->quadratic);
//Material
shader->setUniform( name: "material.ambient",  vector: material.ambient);
shader->setUniform( name: "material.diffuse",  vector: material.diffuse);
shader->setUniform( name: "material.specular",  vector: material.specular);
shader->setUniform( name: "material.shininess", material.shininess);
//Camera
shader->setUniform( name: "viewPosition",scene.camera->position);
shader->setUniform( name: "ProjectionMatrix", scene.camera->projectionMatrix);
shader->setUniform( name: "ViewMatrix", scene.camera->viewMatrix);
// render mesh
shader->setUniform( name: "ModelMatrix",  matrix: modelMatrix);
shader->setUniform( name: "Texture", *texture);
```

- Osvetlenie za pomoci viacerých svetelných zdrojov **[7p]**
  - Správne tiene a odrazy implementované ľubovoľným spôsobom (2p)

Nemám.

Zhodnotenie špecifikácie

Špecifikáciu som zhruba dodržal, iba som ju viac „špecifikoval“. Herné zmeny, čo nastali sú v tom, že na dráhe nie sú zákruty ani lampy, a hráč sa nespomalí keď narazí na prekážku. Dôvod je jednoduchý, keď som písal špecifikáciu, nevedel som odhadnúť zložitosť prvkov v hre, a implementovať tieto veci by ma stálo zbytočnú námahu, a žiadne body navyše by mi nepriniesli.

Všetky triedy ktoré som špecifikoval som aj implementoval, a pridal som aj niektoré ďalšie.

Interakcia používateľa ostala taká akú som plánova, (ale tiež som pridal nejaké ďalšie veci ako menenie uhľa pohľadu kamery, alebo zapínanie vetru).

Vytvoril som 2 vlastné shadere, jeden pre Phonga a jeden pre object lampy. Toto v špecifikácii nebolo, lebo keď som ju písal, nerozumel som ako shadere fungujú.

Čo sa týka grafov v špecifikácii (class diagram a zapojenie komponentov do funkčného celku) , tie naďalej platia ale keďže sú dosť úbohé, aktuálnu funkcionalitu zďaleka nevystihujú.

To isté platí aj pre sekciu Pseudokódy.

Oprava špecifikácie

Úvod

Téma projektu je Go-kart aréna. Moja predstava je taká, že hráč začne v nejakej úvodnej miestnosti, kde sa bude môcť pohybovať, budú tam nejaké predmety, s ktorými môže interagovať , a budú tam dvere, do ktorých keď vstúpi začne hra samotná.

Vtedy sa zmení pohľad kamery a hráč sa ocitne na go-karte a bude môcť jazdiť po dráhe. Dráha bude ohraničená mantinelmi (nebude sa dať ísť mimo nej), budú na nej prekážky v podobe dopravných kuželov a kotúlajúcich sa pneumatík. Keď dôjde ku kolízii hráča a prekážky, prejaví sa to tak, že prekážka odletí.

V go-kart aréne je kamera z tretej osoby, v úvodnej miestnosti z prvej osoby.

V go-kart aréne je jednoduché directional light z ľavého dolného rohu obrazovky. V úvodnej miestnosti sú dve lampy, implementújuce Phongov osvetlovací model.

Výhodou takejto implementácie je podľa mňa to, že ak mi bude treba pridať dodatočnú funkcionalitu, môžem do úvodnej miestnosti ľubovoľne pridávať predmety a zabezpečiť funkcionalitu s nimi.

1. Návrhy dátových Štruktúr v projekte

Main: tato trieda bude nejaký celkový manažér pre chod programu . Bude spracovávať input od používateľa a posielať ho ďalej, inicializovať scény a premínať medzi nimi.

Scene: Zapuzdrenie všetkých objektov, síl, svetiel a kamery pre jednu scénu hry. Sú 2, jedna pre úvodnú miestnosť a druhá pre Go-kart arénu.

Object: Spoločný interface pre všetky objekty.

Camera: Spoločný interface pre obidva typy kamery.

Časť Úvodná miestnosť:

FirstPersonCamera: Objekt kamery a hráča v jednom. Vo svojej funkcii update() spracováva vstup od používateľa ako pohyb (WASD + hýbanie myšou). Nerenderuje sa.

ObjectSpawnShuffler: Trieda ktorá pridelí objektom nábytku náhodné súradnice, na ktorých sa spawnnú v scéne, ešte predtým ako sa scéna spustí.

PointLight: Objekt lampy, ktorý vyžaruje svetlo. Má vlastný shader light_cube_vert.glsl a light_cube_frag.glsl .

Fyzické objekty – nábytok v miestnosti všetky majú vlastné textúry a meshe, a všetky využívajú shadere myshader_frag.glsl a myshader_vert.glsl :

-Barrel, Crate a Table – tieto objekty sa vždy spawnnú na inom mieste. Zároveň majú definované aj materiály pre Phongov osvetlovací model.

-Sphere – ide o agregáciu 3 rovnakých objektov, ktoré sa pohybujú a využívajú hierarchickú transformáciu a násobenie matíc, vo svojich funkcíach update1(), update2() a update3().

-LightSwtich – objekt ktorý vo svojej funkcii update() spracováva vstup od používateľa (F) ,a volá funckiu changeColor() objektov PointLight.

-Door - objekt ktorý vo svojej funkcii update() spracováva vstup od používateľa (F) , a prepne obrazovku na scénu 2.

Časť Go-kart aréna:

ThirdPersonCamera: Objekt kamery, má viacero prednastavených pohľadov. Vo svojej update() funkcii zachytáva vstup od používateľa (E), a prepína medzi nimi. Nerenderuje sa.

Generator: Objekt, ktorý dynamicky vytvára iné objekty a dáva im počiatočné pozície. Nerenderuje sa.

Fyzické objekty – hráč, prekážky a pozadia. Všetky majú vlastné textúry a meshe, a všetky využívajú shadere diffuse_frag.glsl a diffuse_vert.glsl :

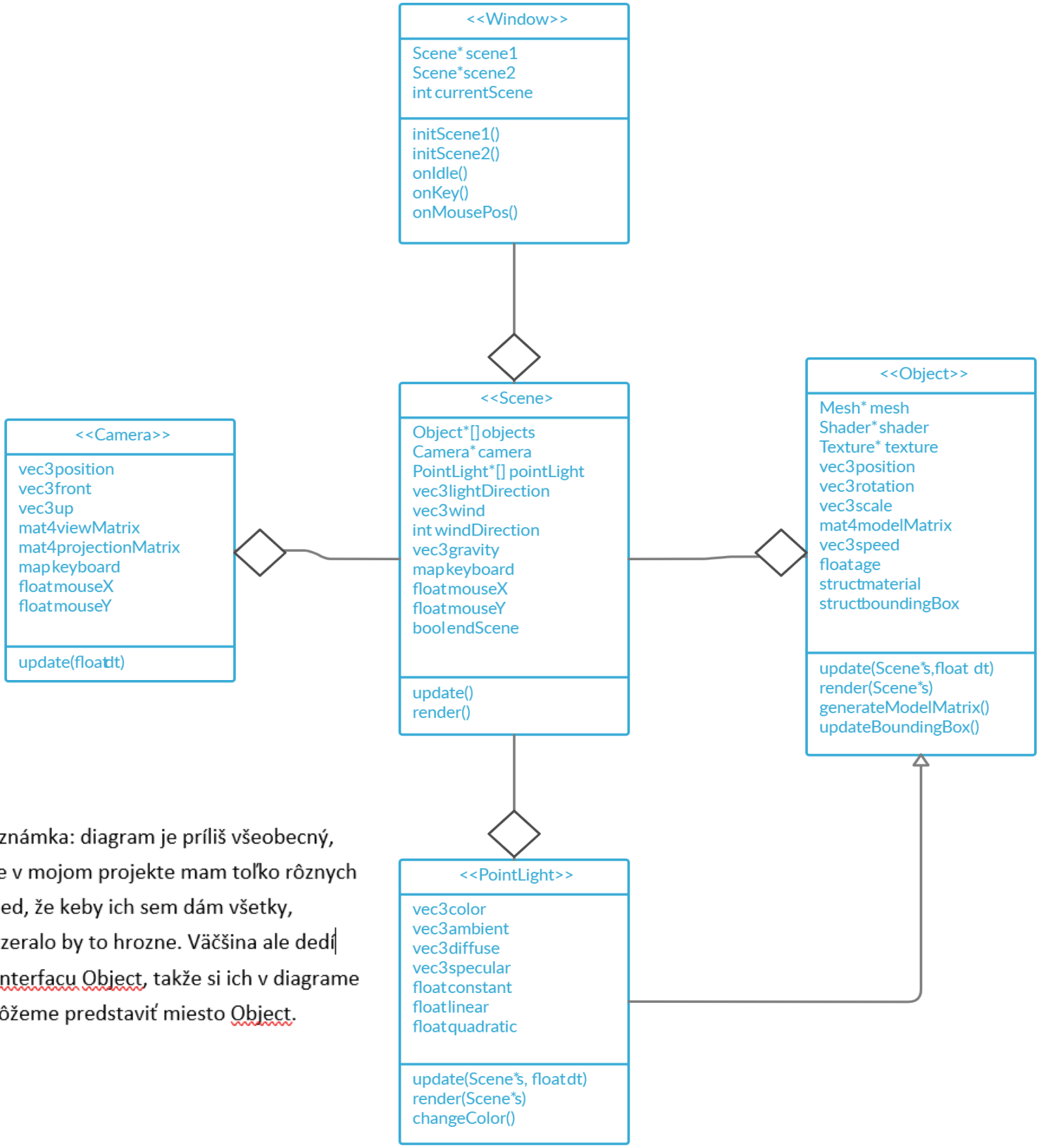
-Road, Mantinel – textúra na nich sa posúva aby hráč nadobudol dojem že sa hýbe dopredu.

- TrafficCone,Tire – prekážky generované objektom Generator. Spawnú sa v diaľke a pohybujú sa smerom k hráčovi. Keď do nich hráč narazí (pretne sa boundingBox prekážky a hráča), zavolá sa metóda collide(), ktorá dá prekážke procedurálnu animáciu.

-Player – objekt hráča, . Vo svojej update() funkcii zachytáva vstup od používateľa ako pohyb (A,D), skok (SPACE), vypínanie a zapínanie vetra (W). Zároveň prechádza ostatné objekty v scéne a kontroluje či nedošlo ku kolízii s prekážkou. Ak áno, zavolá metódu collide() danej prekážky.

- Finish – objekt cieľu, Generátor ho spawne po minúte od začatia scény, tiež sa pohybuje smerom k hráčovi a v momente ako sa ho dotkne, scéna sa prestane animovať.

2. Class diagram



Poznámka: diagram je príliš všeobecný, ale v mojom projekte mam toľko rôznych tried, že keby ich sem dám všetky, vyzeralo by to hrozne. Väčšina ale dedí z interfacu Object, takže si ich v diagrame môžeme predstaviť miesto Object.

3. Pseudokódy

Časť Úvodná miestnosť

```
While (doorEntered == false){
        if (input == „w,a,s,d“) {
                player.update()
                camera.update()
        }
        else if (input == „mouse“)
                camera.update()

        else if (input == “f”)
                for(object in objects)
                        if (camera.lookingAt(object)) {
                                player.interact(object)
                                if (object == ‘door’)
                                        doorEntered = true
                        }

        scene.update()
}
main.switchScene()
```

Časť Go-kart aréna

```
While (foundGoal == false){
        if (input == „a,d“) {
                player.update()
                camera.update()
        }
        If (player.checkForCollision() == true){
                If (object == ‚goal‘)
                        foundGoal = true
                else
                        player.update()
        }
        generator.addObjects()
        scene.update()
}
```

4. Návrh interakcie používateľa s projektom

V úvodnej miestnosti : pohyb: WASD, otáčanie myšou, interakcia s objektami: F.

V go-kart aréne : pohyb doľava a doprava AD, skok: SPACE, menenie uhľa pohľadu kamery: E, vypínanie a zapínanie vetra: W.

5. Metódy renderovania štruktúr na obrazovku

V úvodnej miestnosti objekty používajú shadere *myshader_vert.glsl* a *myshader_frag.glsl,* kde je uplatnený Phongov osvetlovací model. Do shadera sa pošlú ako vstupné parametre:

-zložky svetla zo zdroju svetla, spolu s jeho pozíciou farbou a vzdialenostnými konštantami (2x, pre oba zdroje svetla v scéne).

- zložky matériálu daného objektu (každý objekt má buď defaultný, alebo definovaný materiál)

- pozícia kamery + PositionMatrix + ViewMatrix

- modelMatirx objektu + jeho textúra

Výnimkou sú objekty, ktoré sa nerenderujú, a samotné zdroje svetla, ktoré majú vlastný shader *light_cube_vert.glsl* a *light_cube_frag.glsl*. Tieto shadere spravia iba to, že objekt bude vyzerať ako keby vyžaruje svetlo v každom smere v maximálnej intenzite. Potrebujú tieto parametre :

-farbu svetla

-projectionMatrix a viewMatrix kamery

-modelMatrix objektu svetla

V go-kart aréne objekty používajú predpripravený shader *diffuse_vert_glsl.h* a *diffuse_frag_glsl.h*. Funguje rovnako ako gl9, zdroj svetla je definovaný v triede Scene.

Všetky predmety budú využívať triedy *Mesh*, *Texture* a *Shader* definované v adresári *ppgso.*

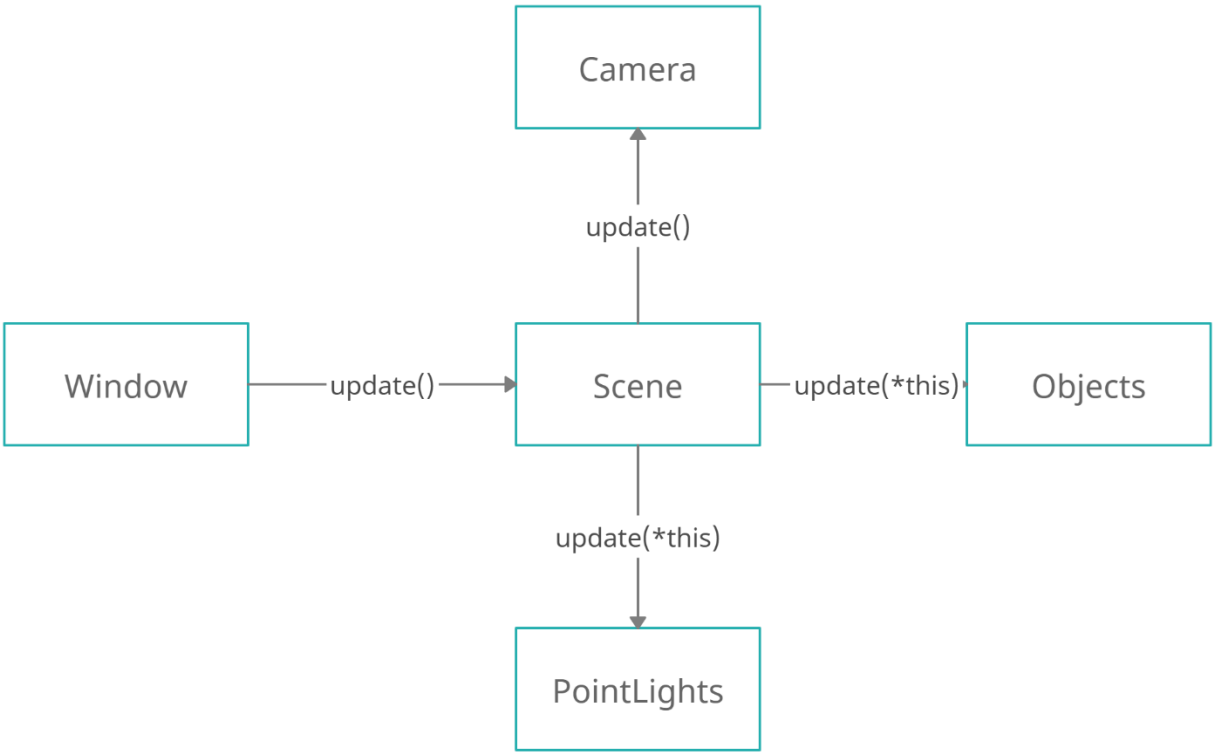
6.Zapojenie jednotlivých komponentov aplikácie do funkčného celku

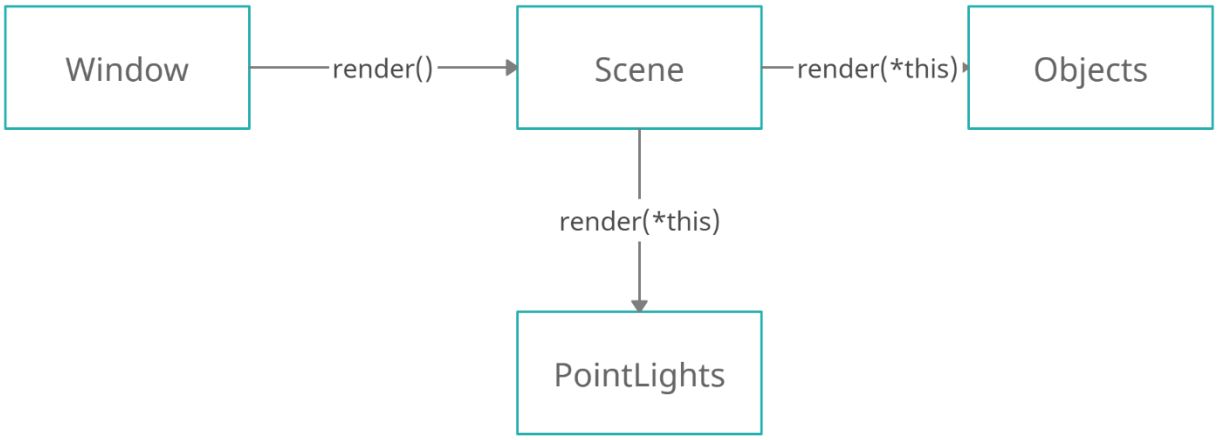
Trieda Scene agreguje všetko, čo sa nachádza v jednej „scéne“ projektu: Kameru, svetlá , všetky fyzické objekty, všetky externé sily, používateľský vstup formou klávesnice a myši.

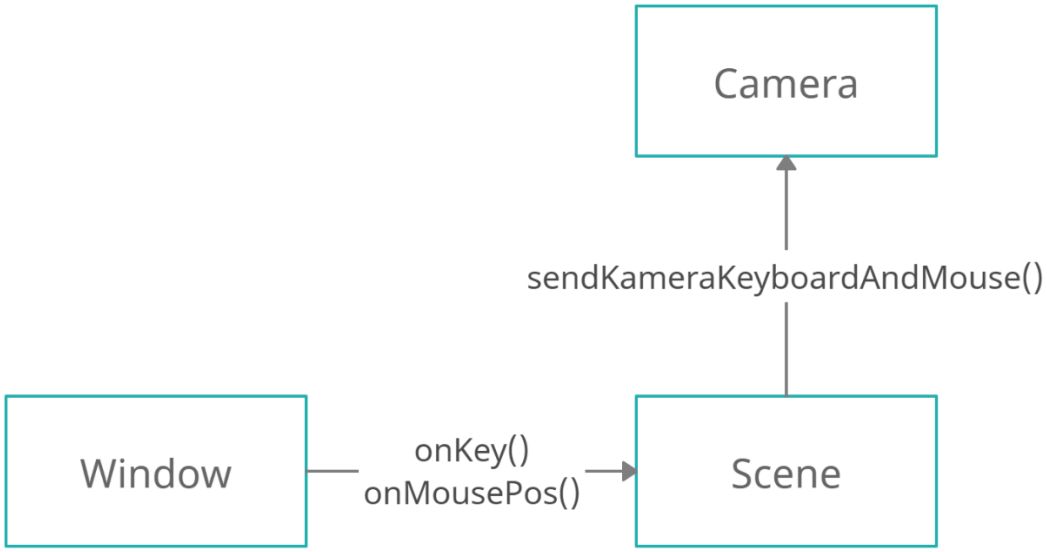
Window vo svojej funkcii onIdle() volá funkciu update() aktuálnej scény. (A tiež do nej posiela používateľský input). Scéna vo svojej update() funkcii volá update() funkcie všetkých objektov v nej, a ako argument posiela pointer na samu seba. Vďaka tomu má každý objekt v scéne prístup ku všetkému, čo by mohol potrebovať.

Poznámka: Kamera sa správa trochu inak, jej funckia update() nedostáva pointer na scénu, lebo žiadne údaje z nej nepotrebuje (lebo sa nerenderuje). Potrebuje však prístup k vstupu od používateľa, preto jej ho scéna musí explicitne posielať.

Poznámka 2: diagramy sú príliš všeobecný, ale v mojom projekte mam toľko rôznych tried, že keby ich sem dám všetky, vyzeralo by to hrozne. Väčšina ale dedí z interfacu Object, takže si ich v diagrame môžeme predstaviť miesto Object.




Window
render()
Scene
render(*this)
Objects
render(*this)
PointLights


Camera
sendKameraKeyboardAndMouse()
Window
onKey()
onMousePos()
Scene