Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
Ilkovičova 2, 842 16 Bratislava 4

Záverečná správa

Róbert Jačko

Študijný program: Informatika
Ročník: 3.
Predmet: Princípy počítačovej grafiky a spracovania obrazu
Ak. rok: 2020/2021

Téma: Príroda

1. scéna

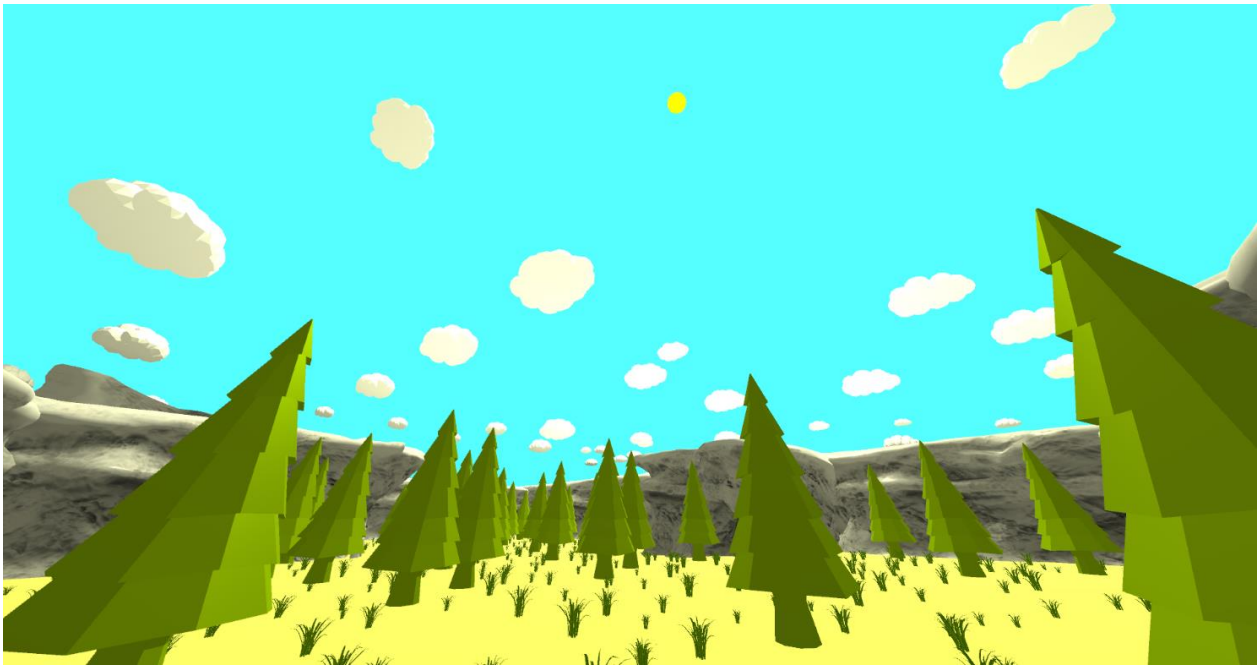
Prvá scéna je izba malého tučniaka. Je neskoro večer a tučniak sa chystá už spať. Po izbe sa dá voľne pohybovať. Je osvetlená lampou, ktorej farba sa dá meniť klávesou „F“. Keď sa priblíži k posteli, zobrazí sa okno, v ktorom je nápoveda. Ak je po zobrazení vyskakovacieho okna stlačená klávesa „F“, nastane animácia prechodu medzi scénami (obraz sa bude postupne stmavovať).

Izba osvetlená prostredníctvom spotlight svetelného zdroja, ktorým je lampa

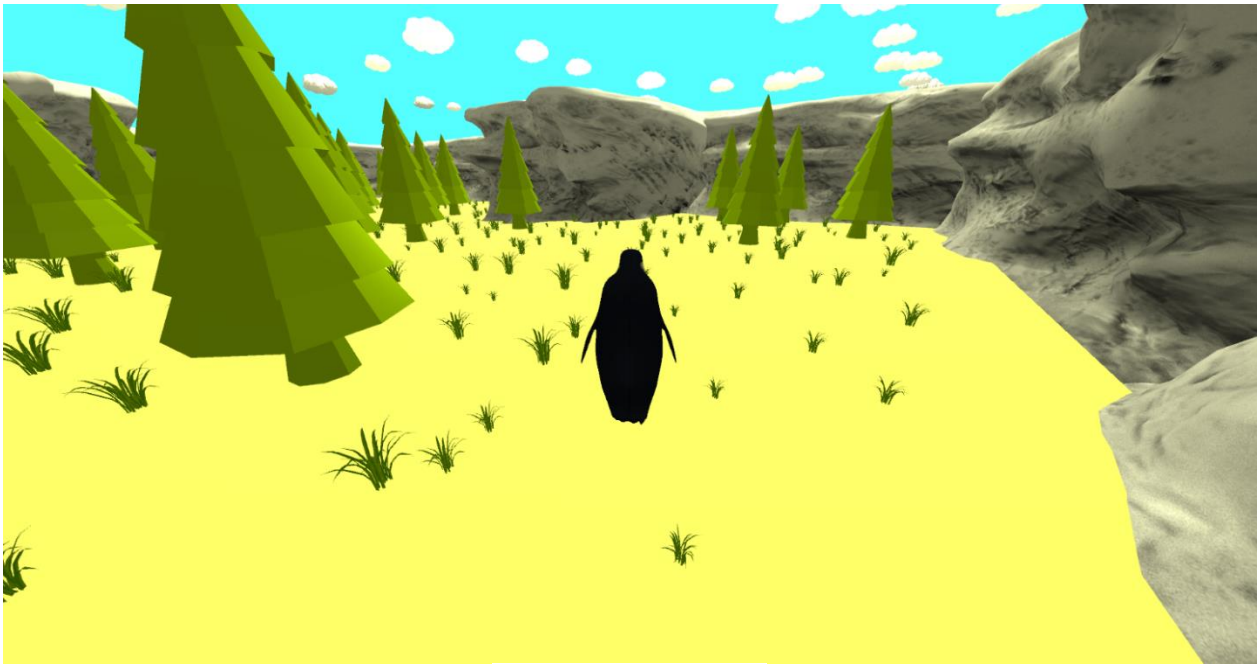
Okno s nápovedou. Pokial je hráč pri posteli tak pri pohybe alebo rotácii kamery okno ostáva stále v strede (rotuje a hýbe sa spolu s hráčom)

2. scéna

Druhá scéna je vlastne sen malého tučniaka, v ktorom prechádza čudným lesom. Začína sa animáciou prechodu medzi scénami (obraz sa postupne zosvetluje). Po lese sa dá voľne pohybovať a hádzať kamene všade okolo seba (stlačením ľavého tlačítka myši). Nachádza sa tu tiež jeden divný tučniak, ktorý pomocou animácie skáče zo skaly salto dozadu, potom sa trochu poprechádza a zrazu sa vráti pospiatočky naspäť. Ak malý tučniak vojde do jaskyne na konci lesa, zobrazí sa okno s nápovedou. Ak je potom stlačená klávesa „F“ tučniak sa zobudí zo sna a hra končí.



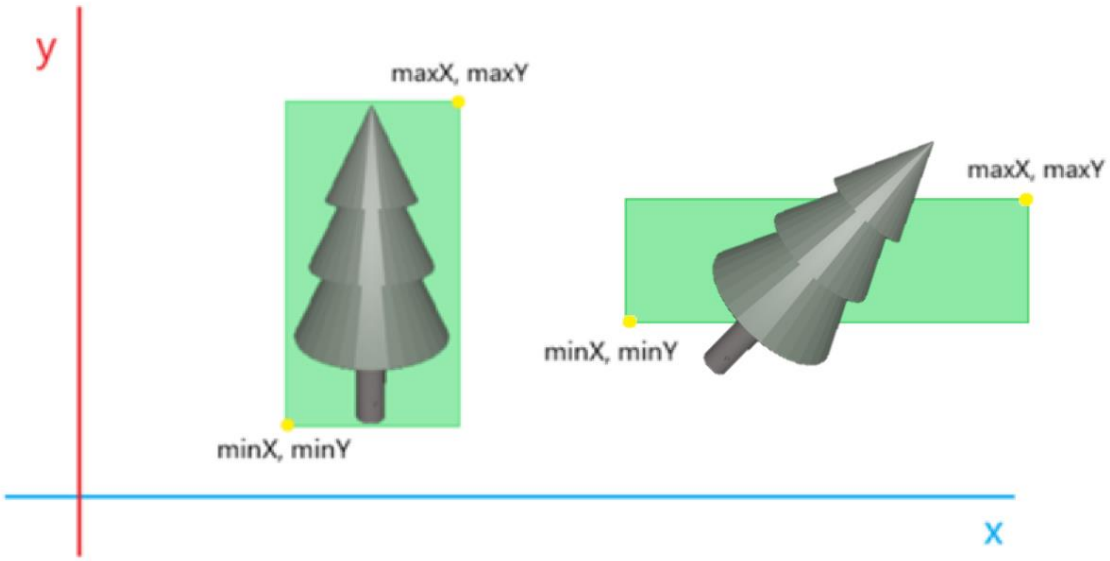
Pohľad z 1. osoby



Pohľad z 3. osoby

Kolízie

Každému objektu je priradený jeho BoxCollider. BoxCollider je reprezentovaný 2 bodmi, ktoré sa vždy transformujú spolu s ich objektami. Prvý bod si vždy pri škálovaní uchováva najmenšiu súradnicu x, y, z a druhý naopak tú najväčšiu. Kvôli tejto reprezentácii by mali byť objekty osovo zarovnané, pretože inak by BoxCollider nebol presný.



Pozn. Kedže sú oba body vždy transformované spolu s objektom, tak pri rotácii objektu rotujú oba body rovnako okolo stredu daného objektu. Preto ak objekt nie je osovo zarovnaný, nastane

Pri detekcii kolízie pre daný objekt sa v cykle prejdú všetky objekty, ktoré sa v scéne nachádzají a pri každom z nich sa použije jednoduchá funkcia na zistenie, či sa s daným objektom „pretínajú“.

```
bool Object::isColliding(glm::vec3 *objCollider) {
    return (objCollider[0].x <= collider[1].x && objCollider[1].x >= collider[0].x) &&
           (objCollider[0].y <= collider[1].y && objCollider[1].y >= collider[0].y) &&
           (objCollider[0].z <= collider[1].z && objCollider[1].z >= collider[0].z);
}
```

Pri kolízii sa pomocou dvoch bodov, ktoré definujú BoxCollider potom získava normála pre danú plochu objektu, s ktorým kolízia nastala. Normála sa vypočíta tak, že sa zistí, ktorá vzdialenosť medzi jednotlivými súradnicami BoxColliderov je najkratšia (tam sa objekty „pretli“).

Pozn.: ak je najkratšia vzdialenosť medzi **najmenším x cudzieho** objektu a **najväčším x tohto** objektu, cudzí objekt **prichádza sprava**. Normála je preto **{x=1, y=0, z=0}**, ak je najkratšia vzdialenosť medzi **najmenším y cudzieho** objektu a **najväčším y tohto** objektu, cudzí objekt prichádza **zhora**. Normála je preto **{x=0, y=1, z=0}**,...

```
glm::vec3 Object::getCollisionNormal(glm::vec3 *objCollider) {
    float diffs[6] = {
            std::abs( x: objCollider[0].x - collider[1].x),
            std::abs( x: objCollider[0].y - collider[1].y),
            std::abs( x: objCollider[0].z - collider[1].z),
            std::abs( x: objCollider[1].x - collider[0].x),
            std::abs( x: objCollider[1].y - collider[0].y),
            std::abs( x: objCollider[1].z - collider[0].z)
    };

    glm::vec3 normals[6] = {
            { a: 1, b: 0, c: 0},
            { a: 0, b: 1, c: 0},
            { a: 0, b: 0, c: 1},
            { a: -1, b: 0, c: 0},
            { a: 0, b: -1, c: 0},
            { a: 0, b: 0, c: -1}
    };

    int minIndex = 0;
    for (int i = 1; i < 6; i++) {
        if (diffs[i] < diffs[minIndex]) {
            minIndex = i;
        }
    }

    return normals[minIndex];
}
```

Jednoduché objekty

Objekty, ktoré slúžia len na dizajn alebo ako prekážky a nemajú žiaden pohyb sú definované prostredníctvom jedinej triedy SimpleObject. Ich meshe a textúry sú uložené v statickom poli (aby rovnaké objekty mohli zdielať tieto dáta). Rozoznávajú sa na základe ich ID.

```
class SimpleObject final : public Object {
private:
    uint8_t id;
    const static uint8_t OBJ_COUNT = 12;
    static std::unique_ptr<ppgso::Mesh> mesh[OBJ_COUNT];
    static std::unique_ptr<ppgso::Shader> shader;
    static std::unique_ptr<ppgso::Texture> texture[OBJ_COUNT];
    static char *textureName[OBJ_COUNT];
    static char *meshName[OBJ_COUNT];
```

```
char *SimpleObject::textureName[OBJ_COUNT] = {"wall.bmp", "ground.bmp", "floor.bmp", "ceiling.bmp", "g
char *SimpleObject::meshName[OBJ_COUNT] = {"cube.obj", "cube.obj", "cube.obj", "cube.obj", "tree.obj",
```

```
SimpleObject::SimpleObject(glm::vec3 p, glm::vec3 r, glm::vec3 s, uint8_t id) {
    if (!shader) shader = std::make_unique<ppgso::Shader>(color_vert_glsl, color_frag_glsl);
    if (!texture[id]) texture[id] = std::make_unique<ppgso::Texture>(ppgso::image::loadBMP(textureName[id]));
    if (!mesh[id]) mesh[id] = std::make_unique<ppgso::Mesh>(meshName[id]);
```

Rendering

Výpočet transformácie objektu

```
void Object::generateModelMatrix() {
    glm::mat4 t = glm::translate(position);
    glm::mat4 r = glm::yawPitchRoll( yaw: glm::radians(rotation.y),  pitch: glm::radians(rotation.x),  roll: glm::radians(rotation.z));
    glm::mat4 s = glm::scale(scale);

    modelMatrix = t * r * s;
}
```

Svetlo

- používa sa difúzna, ambientná a spekulárna zložka
- potom je implementovaný spotlight

```
FragmentColor = texture(Texture, vec2(texCoord.x, 1.0 - texCoord.y) + TextureOffset) * vec4((Diffuse + Ambient + Specular), 1.f);
float d = length(fragPos - LightPosition);
float attenuation = .1f + 0.07 * d + 0.01 * d * d;
FragmentColor = FragmentColor * max(dot(-lightDir, vec3(0, -1, 0)), 0.2f) / attenuation;  // spotlight
```

Transformácia do kamery

```
// Calculate the final position on screen
gl_Position = ProjectionMatrix * ViewMatrix * ModelMatrix * vec4(Position, 1.0);
```

Zhodnotenie

Špecifikáciu som úplne nedodržal, ale v mnohých smeroch som ju vylepšil. Nestihol som implementovať osvetlenie pomocou viacerých zdrojov svetla a nepoužíval som pri osvetlovaní materiály objektov. Taktiež som nestihol implementovať tiene, namiesto ktorých som pri vstupovaní do jaskyne aspoň postupne tlmil osvetlenie. Podarilo sa mi vytvoriť pomerne efektívne detegovanie kolízií, ale len pre objekty, ktoré sú osovo zarovnané.