Souhlasím s vystavením této semestrální práce na stránkách katedry informatiky a výpočetní techniky a jejímu využití pro prezentaci pracoviště.

**Jméno:** Vít Novotný
**Datum:** 7. dubna 2025
**E-mail:** vnovotny@students.zcu.cz
**Studijní číslo:** A23B0412P

# Dokumentace k semestrální práci: 3D bludiště v OpenTK

## 1. Popis projektu

Aplikace implementuje základní 3D bludiště v prostředí OpenGL pomocí knihovny OpenTK, včetně kompletní fyziky pohybu hráče, kolizí, osvětlení a práce s texturami. Cílem bylo vytvořit interaktivní first-person prostředí, které je stabilní, fyzikálně konzistentní a odpovídá specifikaci zadání.

## 2. Klíčové části implementace

### Pohyb hráče a fyzika

Pohyb hráče je založen na jednoduché fyzikální simulaci – akcelerace je odvozena ze součtu sil působících na hráče. Ovládání reaguje na vstupy (WSAD, Shift pro běh, mezerník pro skok) a generuje síly, které určují směr pohybu.

Hráč se pohybuje konstantní rychlostí bez ohledu na kombinaci směrových kláves. Zároveň je implementována korektní simulace skákání s gravitací a zachycením dopadu.

### Kolizní systém

Byl implementován systém kolizí postavený na testování průniku koule s trojúhelníky pomocí metody "move and slide". Tento systém zajišťuje plynulý pohyb podél stěn bez zasekávání nebo poskakování a brání opuštění mapy, i v případě, že některé zdi chybí.

Po každém kroku je upravena pozice hráče podle kolizních korekcí a `Velocity` je znovu vypočtena podle skutečného posunu, čímž se udržuje fyzikální konzistence.

Následující metoda určuje nejbližší bod ke kouli na trojúhelníku ve 3D prostoru. Používá se při testu průniku:

```
private static Vector3 ClosestPointOnTriangle(Vector3 p, Vector3 a, Vector3 b,
    Vector3 c)
{
    Vector3 ab = b - a;
    Vector3 ac = c - a;
    Vector3 ap = p - a;

    float d1 = Vector3.Dot(ab, ap);
    float d2 = Vector3.Dot(ac, ap);
    if (d1 <= 0 && d2 <= 0) return a;

    Vector3 bp = p - b;
    float d3 = Vector3.Dot(ab, bp);
    float d4 = Vector3.Dot(ac, bp);
    if (d3 >= 0 && d4 <= d3) return b;

    float vc = d1 * d4 - d3 * d2;
    if (vc <= 0 && d1 >= 0 && d3 <= 0)
    {
        float v = d1 / (d1 - d3);
        return a + ab * v;
    }

    Vector3 cp = p - c;
    float d5 = Vector3.Dot(ab, cp);
    float d6 = Vector3.Dot(ac, cp);
    if (d6 >= 0 && d5 <= d6) return c;

    float vb = d5 * d2 - d1 * d6;
    if (vb <= 0 && d2 >= 0 && d6 <= 0)
    {
        float w = d2 / (d2 - d6);
        return a + ac * w;
    }

    float va = d3 * d6 - d5 * d4;
    if (va <= 0 && (d4 - d3) >= 0 && (d5 - d6) >= 0)
    {
        float w = (d4 - d3) / ((d4 - d3) + (d5 - d6));
        return b + (c - b) * w;
    }

    Vector3 n = Vector3.Cross(ab, ac);
    float denom = Vector3.Dot(n, n);
    float u = Vector3.Dot(Vector3.Cross(ap, ac), n) / denom;
    float v2 = Vector3.Dot(Vector3.Cross(ab, ap), n) / denom;

    return a + ab * u + ac * v2;
}
```

Další metoda kontroluje, zda koule zasahuje do trojúhelníku, a v případě kolize vrací vektor vytlačení:

```
public static bool SphereIntersectsTriangle(Vector3 center, float radius,
    Vector3 a, Vector3 b, Vector3 c, out Vector3 pushOut)
{
    pushOut = Vector3.Zero;

    // Najdi nejbližší bod na trojúhelníku k centru koule
    Vector3 closest = ClosestPointOnTriangle(center, a, b, c);
    Vector3 diff = center - closest;
    float distSq = diff.LengthSquared;

    // Pokud je vzdálenost menší než poloměr, nastala kolize
    if (distSq < radius * radius)
    {
        float dist = MathF.Sqrt(distSq);
        Vector3 normal = dist > 0 ? diff / dist : Vector3.UnitY; // pokud je stř
    ed přímo na trojúhelníku
        float penetration = radius - dist;
        pushOut = normal * penetration; // vektor, kterým se koule vytlačí z
    kolize
        return true;
    }

    return false;
}
```

### Osvětlení: Svítilna

Scéna je osvětlena reflektorem ve výšce 2,05 m s depresí 2°. Světlo sleduje pozici a směr hráče, připomíná baterku. Úhel světelného kužele je menší než FOV pozorovatele a osvětluje jen přímý výhled.

Implementace využívá OpenGL spotlight s nastavením směru a cutoff úhlu.

### Texturování

Zdi a podlaha jsou texturované pomocí klasického vzorkování. Načítání textur je provedeno s ohledem na HW podporu – ve výchozím nastavení je použito lineární filtrování s mipmapami.

### FPS Counter

Na obrazovce je zobrazeno počítadlo snímků za vteřinu, které počítá, kolik snímků proběhlo za poslední jednu vteřinu. Výstup je nezávislý na snímkové frekvenci a zajišťuje reálný odhad výkonu.

### First-person pohled a ovládání

Kamera sleduje hráče z pohledu první osoby. Myší lze otáčet pohled (vertikálně i horizontálně), přičemž rotace je omezena v rozsahu -90° až +90° ve vertikále. FOV lze dynamicky měnit kolečkem myši mezi 30° a 120°, výchozí hodnota je 90°.

## 3. Implementované prvky dle zadání

### Povinná část

- Pravoúhlá síť bludiště
- Fyzikálně založený pohyb hráče (rovnoměrná rychlost, vektorový směr)
- Kolize se stěnami bez poskakování a průchodů
- Zajištění hranic mapy i bez zdi
- Plynulé řízení pohledu myší, omezení přetočení
- Vše nezávislé na FPS
- Osvětlení svítilnou dle specifikace
- FPS counter
- Měřítko a geometrie dle zadání (1 jednotka = 1 m)

### Volitelná část

- Texturování zdí a podlahy
- Svítilna (projekční světlo)
- FPS měření
- Pokročilý kolizní systém

## 4. Poznámky k ovládání

- `W, A, S, D` – pohyb vpřed, vlevo, vzad, vpravo
- `Shift` – běh
- `Mezerník` – skok
- `Myš` – otáčení pohledu
- `Kolečko` – změna FOV (30°–120°)

## 5. Závěr

Implementace naplňuje požadavky zadání a zároveň ukazuje silnou stránku ve fyzikálně korektním chování hráče, precizních kolizích a světelném modelu. Kód je rozdělen do samostatných modulů, což usnadňuje rozšiřování o další funkce (např. minimapa, AI, grafické efekty).