3D bludiště - Semestrální projekt ZPG

Jakub Vokoun, 14. května 2025, javok@students.zcu.cz, A23B0235P

Spuštění

V příkazové řádce ve složce *src* spustit *dotnet run*. Lze předat prvním parametrem cestu k mapě (relativní vůči aktuální cestě). Při absenci parametru se načte výchozí mapa ve složce *src/Levels*.

Ovládání

Myš - směr pohledu kamery. WASD - chůze daným směrem. E - interakce s objekty (aktivace teleportu). shift - běh.

Životní cyklus aplikace

Vstupním bodem je třída App, která pouze zpracuje parametry příkazové řádky a předá kontrolu třídě Window. Ta připraví mapu na základě předaného parametru tak, že vytvoří instanci třídy Level, která načte mapu ze souboru.

Poté až do konce probíhá smyčka ve Window, kdy při volání funkcí se děje:

- OnRenderFrame: Vyrendruje scénu a upraví fps.
- OnUpdateFrame: Předá instanci třídy Camera stisklé klávesy pro zpracování, zpracuje klávesy ovlivňující celou aplikaci (např. Esc -> zavřít okno).
- OnMouseMove: Předá instanci třídy Camera pozici myši ke zpracování.

Kolize

Každý renderovaný objekt ve scéně a kamera má atribut typu *CollisionCube*, tedy kvádr se středem a třemi rozměry. Při každém pohybu kamery se kamery *CollisionCube* pokusí posunout na žádané místo. Jsou tři scénáře, které mohou nastat:

1. Místo je prázdné, lze na něj vstoupit.
2. Místo je obsazené (zdí), v tu chvíli nastupuje mechanismus kolizí a upraví pohyb tak, aby hráč šel žádaným směrem, ale pouze tak daleko, aby neprošel zdí.
3. Místo je prázdné, nemá ale ani podlahu. V tu chvíli zafunguje gravitace a přidá do pohybu negativní složku Y, úměrnou době pádu podle fyzikálních vztahů.

Mechanismus kolizí pro AABB

Celý projekt je osově zarovnaný, není proto problém při řešení kolizí použít poměrně jednoduššího mechanismu, tedy zkontrolovat kolizi v každém směru (X,Y,Z) a pokud nastává, upravit pohyb tak, aby nenastala. V obou horizontálních směrech os X a Z a zároveň ve směru vertikálním osy Y, a pokud posunutím v daném směru vzniká kolize, tak nastaví pohyb v tomto směru na maximální možný, který však nekoliduje. Tím zároveň vzniká pomalejší pohyb, pokud se hráč *otírá* o stěnu.

Snap-grid

Aby se předešlo kumulativní chybě vzniklé nasčítáním chyb ve floatových číslech, konkrétně v pozici kamery, tak je implementována "mřížka", po které se kamera pohybuje. Prakticky je to omezení přesnosti floatu na pouze několik málo desetinných míst tak, aby se nemohla chyba kumulovat, ale hráč si ničeho nevšiml.

FPS

Počítadlo snímků za vteřinu je umístěné v názvu okna, protože je to jednodušší, než implementovat zobrazování textu. Okno má vlastní stopky a při každém renderování scény si uloží aktuální čas do fronty. Poté z fronty odstraní všechny moc staré záznamy a vypíše FPS jako počet objektů ve frontě.

Načítání levelů

Pokud uživatel při spuštění aplikace předá prvním parametrem cestu k souboru s mapou, pokusí se tuto mapu aplikace načíst. Pokud to nebude možné, aplikace skončí a vypíše chybu. Pro zabránění hráči opustit mapu i v místech kde není stěna slouží přidání stěny z krychlí po takových okrajích mapy, kde chybí zeď. Tyto krychle mají jinou texturu, aby tak indikovaly konec mapy, interakce (sliding) s okrajem mapy je tedy již implementovaný v kolizích.

IObjectsStore - Grid

Přidaná abstrakce pro správu RenderObjectů. Třídy pracující s nimi nemusí řešit podrobnosti implementace, ale voláním funkcí *Add*, *GetAll* a *GetOnlyRelevant* jsou schopny je používat.

Tato abstrakce umožnila jednoduchou výměnu obyčejného Listu, za lehce sofistikovanější metodu ukládání - pravidelné mřížky ve slovníku. Tím se teoreticky snižuje složitost kontroly kolizí (nebo kdekoliv jinde se volá *GetOnlyRelevant*) z O(n) na O(k), kde n je počet všech objektů a k je počet *relevantních* objektů, v mřížce blízko. Pro malé mapy je to prakticky možná větší zdržení kvůli režii, ale nebylo testováno.

Vícepatrové bludiště

Implementace rozšíření zahrnovala tyto body:

- Umožnit načítání vícepatrových i jednopatrových map.
- Implementace gravitace.
- Přidání podlah.

Načítání bylo jednoduše rozšířeno o třetí rozměr, bloky už předtím měly svojí Y souřadnici, i když všechny stejnou. Nyní bylo třeba přidat akorát logiku během načítání souboru s mapou tak, aby se objekty pokládaly do správné výšky.

Gravitace byla implementována ve třídě Camera, kde je pomocí vzorečku $y(t) = \frac{1}{2} g t^2$ vyčíslena aktuální změna polohy v deltě času během pádu. Počítá se se standardním tíhovým zrychlením 9.81 m/s. Kvůli zaokrouhlovací chybě se však vzdálenost pro malé dT rovnala nule, proto se skutečně počítá ze vzorečku $y(t) = \frac{1}{2} g t$. Hráč začne padat ve chvíli, kdy pod jeho středem není nic - trochu rozdíl od klasických her, které hráčům více odpouští.

Aby se předešlo konstantnímu *poskakování kamery*, udržuje si kamera referenci na RenderObject, na kterém právě stojí. Pokud existuje, gravitace se neaplikuje. Pokud vyjde mimo objekt, na kterém stojí, pokusí se najít jiný, když ale žádný takový není, tak začne padat. Toto řešení stále způsobuje poskakování, ale pouze při přechodu mezi bloky.

Podlahy jsou implementovány jako kvádry s malou výškou. Nastal problém s propadáváním skrz podlahu, pokud byla moc nízká. Všechno řeší různé epsilony, které kontrují nepřesnost floatových čísel. Podlaha s nenulovou výškou je tedy zarovnána tak, že její vrchní část je ve stejné výšce jako případná kostka, aby nenastaly problémy s přechodem z jednoho bloku na druhý.

Teleport

Zahrnuje:

- Implementaci teleportačních plošin, jejich propojení a aktivaci.
- Vytvoření efektu *fade to white and back*.

Teleportační plošiny jsou velmi podobné podlahám, navíc však mají referenci na svůj *protějšek*. Ta se nastaví během načítání, není nijak kontrolována správnost, podle zadání se předpokládá validní vstup. V opačném případě by akorát nefungoval správně teleport, což není kritická chyba. Správnost spočívá v tom, že každý teleport má právě jednoho svého *protějška*, čímž je zaručena jednoznačnost místa přenosu při použití teleportu. Hráč může teleport využít tehdy, pokud na něj vstoupí a stiskne interakční klávesu E.

Efekt *FTW&B* byl vytvořen pomocí nového objektu se speciálním shaderem. Tento objekt je přidán do scény ve třídě Window po načtení Levelu. Objekt WhiteScreen si udržuje informaci o tom, jakou má mít průhlednost, probíhá-li právě teleport. Pokud ano, tak jak dlouho již probíhá a jaký je

celkový čas efektu. Právě tyto informace slouží k ovládání zvnějšku! Pouze průhlednost se předává fragment shaderu, který na jejím základě (ne)vykresluje.

Celý efekt je ovládán z Camery během události Window.OnRenderFrame -> Camera.ProcessKeyboard. Jelikož je výpočet aktuální průhlednosti overlaye počítán v závislosti na uplynulém času, nemá na něj fps žádný dopad.

Pro jednoduchost je efekt pozicován v prostoru kamery, tedy nehýbe se ve světě, ale je vždy automaticky před kamerou.

Dohromady je při tvorbě mapy možno vytvořit 0 až 10 teleportů, tedy 0 až 20 teleportačních platforem. Každá dvojice platforem má stejnou texturu, přičemž pro každou dvojici je vytvořena speciální, pro snadnost rozlišení. Teleportů by mohlo být i více, v závislosti na zvoleném rozsahu ASCII symbolů, ale bylo rozhodnuto, že deset stačí.

Sluníčko

Sluníčko je implementováno z textury. V shaderu se aplikuje Phongovo osvětlování, kde se odráží světlo podle toho, co je na textuře.

Textura

Přes třídu Texture, která načte obrázek a nastaví texturu v OpenGL. Každý RenderObject má dvě textury, difusní a spekulární mapu - rozlišení při osvětlování. Hardware assessment neimplementován.

Návod pro vytváření map

Na prvním řádku: {š}x{h} nebo {š}x{h}x{v} Kde {š} je integer určující šířku (X souřadnice), {h} hloubku (Z) a {v} výšku (Y).

Na dalších řádcích mapa, každý charakter je jeden blok v mapě. Šířka mapy se rovná šířce řádku, hloubka mapy je počet řádků na patro, výška je počet opakování pater. Dohromady by tedy soubor měl mít řádků: 1 + š * v (* h) Pokud není zeď na konci mapy, případně podlaha/strop v spodním/horním patře, tak se automaticky přidá *void* zeď.

Seznam symbolů a jejich významy:

- 'a' až 'n' je volný prostor, vůbec nic
- 'o' až 'z' je zeď, neprůchodné pole
- '@' je startovní pozice hráče na začátku
- '0' - '9' jsou teleporty

- '-' je podlaha
- '+' je strop
- '=' je podlaha i strop

Optimalizace

> V aplikaci se mnohokrát opakují identické objekty, například zdi, které se liší pouze v pozici. Každý takový objekt má ale vlastní hromadu informací, které jsou sice stejné, ale v paměti zabírají místo vícekrát. Je to primárně textura, pak i reference na Shader, Cameru. To je problém pro větší mapy, kde můj stroj už nezvládl mapu 50x80, protože došla GPU paměť (a RAMka používala 10 GB).
>
> Řešení existuje, minimálně se nabízí Instancing, který ovšem není implementován, ale problém by dost pravděpodobně vyřešil.

Tento problém byl vyřešel existencí pouze jedné textury na mapu, od které mají všichni referenci na instanci.

Povolení užití

Souhlasím s vystavením této semestrální práce na stránkách katedry informatiky a výpočetní techniky a jejímu využití pro prezentaci pracoviště. Nekontroloval jsem ale autorská práva textur, tuto zodpovědnost přenechávám katedře.