DELTA – Střední škola informatiky a ekonomie, s.r.o.

Ke Kamenci 151, Pardubice



MATURITNÍ PROJEKT

Knihovna pro simulaci 2D fyzikálního světa s demonstrační desktopovou aplikací

Příjmení, jméno: *Bien Lukáš*

Třída: *4.B*

Studijní obor: *Informační technologie 18-20-M/01*

Školní rok: *2023/2024*

Zadání maturitního projektu z informatických předmětů

Jméno a příjmení: *Lukáš Bien*

Pro školní rok: *2023/2024*

Třída: *4. B*

Obor: *Informační technologie 18-20-M/01*

Téma práce: *Knihovna pro simulaci 2D fyzikálního světa s demonstrační desktopovou aplikací*

Vedoucí práce: *Ing. Tomáš Pešek*

Způsob zpracování, cíle práce, pokyny k obsahu a rozsahu práce:

Cílem práce je vytvořit knihovnu pro fyzikální simulaci objektů ve 2D prostoru. Knihovna by měla umožnit přidávání, pohyb a interakce objektů v rámci simulovaného světa, jako jsou kolize a gravitace. Samotná knihovna by se měla zaměřit pouze na simulaci fyziky, zatímco demonstrační desktopová aplikace by měla ukázat, jak knihovnu využít.

Výstup:

Výstupem práce bude DLL knihovna a Demo desktopová aplikace, která bude demonstrovat, jak knihovnu použít. Demo bude jednoduše vykreslovat pomocí SFML to, co mu knihovna poskytne.

Stručný časový harmonogram (s daty a konkretizovanými úkoly):

Září: Seznámení s problematikou fyzikální simulace, rešerše existujících knihoven

Říjen: Analýza požadavků, návrh API knihovny a rozhraní aplikace

Listopad: Implementace jádra knihovny pro fyzikální simulaci

Prosinec: Implementace desktopové aplikace a integrace s knihovnou

Leden: zahájení zpracování dokumentace stávajícího řešení

Únor-Březen: Kompletace a finalizace dokumentace, dokončení dokumentace a teoretické části práce

Prohlášení

Prohlašuji, že jsem maturitní projekt vypracoval samostatně, výhradně s použitím uvedené literatury.

V Pardubicích dne.....

(vlastnoruční podpis)

Poděkování

Děkuji Ing. Tomášovi Peškovi za odborné vedení při zpracování maturitního projektu. Dále bych rád poděkoval rodině a přátelům za morální podporu.

Resumé

Tato práce dokumentuje tvorbu knihovny pro simulaci 2D fyzikálního světa a demonstrační desktopové aplikace, která slouží jako příklad, jak samotnou knihovnu použít.

Knihovna je psána v jazyce C++, desktopová aplikace je také psána v jazyce C++ za použití knihoven SFML, Dear ImGui a ImGui-SFML.

Klíčová slova

C++, SFML, ImGui, ImGui-SFML, CMake, CLion, knihovna, desktopová aplikace, polygon, API, triangulace, kolize, broad-phase, narrow-phase, uživatelské rozhraní

Summary

This thesis documents the creation of a library for simulating a 2D physical world and a demonstration desktop application that serves as an example of how to use the library itself.

The library is written in C++, the desktop application is also written in C++ using the SFML, ImGui and ImGui-SFML libraries.

Keywords

C++, SFML, ImGui, ImGui-SFML, CMake, CLion, library, desktop application, polygon, API, triangulation, collision, broad-phase, narrow-phase, user interface

Obsah

1. Úvod 8
2. Teoretická část 8
2.1. Způsob zpracování 8
2.1.1. Knihovna 9
2.1.2. Demo desktopová aplikace 9
2.2. Použité technologie 9
2.2.1. C++ 9
2.2.2. CMake 10
2.2.3. SFML 10
2.2.4. Dear ImGui 10
2.2.5. CLion 10
2.3. Pojmy 11
2.3.1. Knihovna 11
2.3.2. Desktopová aplikace 11
2.3.3. Polygon 11
2.3.4. Triangulace 12
2.3.5. Kolize 13
2.3.6. Broad-phase 13
2.3.7. Narrow-phase 13
2.3.8. Uživatelské rozhraní 13
2.3.9. API 13
3. Praktická část 14
3.1. Rešerše existujících řešení 14
3.2. Implementace – Knihovna 15
3.2.1. Newtonův druhý zákon 18
3.2.2. SAT (separating axis theorem) 18
3.2.2.1. MTV (minimální translační vektor) 20

3.2.3. Ear clipping algoritmus 23
3.2.4. Zjištění bodů doteku 25
3.2.5. Kalkulace impulsních sil 28
3.2.5.1. Setrvačnost 29
3.2.6. Použití knihovny 31
3.3. Implementace – Desktopová demo aplikace 32
3.3.1. Funkce 33
4. Závěr 35
5. Seznam použité literatury 36
6. Přílohy 37

1. Úvod

Od mého dětství mě fascinovaly počítačové hry a jejich mechanismy na pozadí. Zároveň mě v nedávné době oslovila open source knihovna Box2D, která se specializuje na simulaci 2D fyziky ve hrách. Knihovna Box2D je například použita v herním enginu Falling Everything, na kterém je postavena poměrně známá roguelike hra Noita. Box2D mě přivedla k otázkám, jak tyto knihovny fungují a jakým způsobem umožňují implementovat realistické fyzikální jevy.

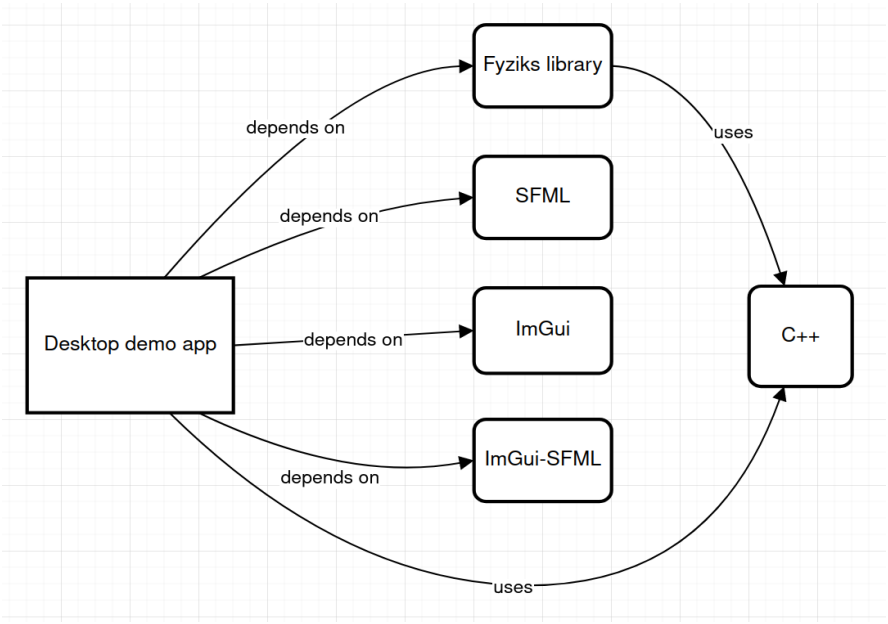
Rozhodl jsem se tedy zkusit naimplementovat základní knihovnu pro simulaci 2D fyzikálního světa, do kterého bude možnost přidávat objekty, na které budou působit vydefinované síly a pravidla, jako jsou váha objektů, tření, restituce a gravitace.

Součástí výstupu této práce je i desktopová aplikace, která používá vytvořenou knihovnu a ukazuje, jak knihovnu použít. Desktopová aplikace bude mít uživatelské rozhraní a uživatel bude moci přidávat a manipulovat objekty.

2. Teoretická část

2.1. Způsob zpracování

Svůj projekt jsem se rozhodl rozdělit do dvou částí – samotná knihovna a desktopová demo aplikace. Obě části budou psány v jazyce C++.



Obrázek 1 diagram struktury projektu

2.1.1. Knihovna

Knihovna se bude starat o veškerou simulaci fyziky bez použití jakékoliv další knihovny, bude implementovat algoritmy pro detekci kolizí mezi objekty, rovnice pro výpočet sil a chování mezi objekty. Zároveň bude knihovna poskytovat API pro uživatele, který bude moci tvořit svět a objekty v něm, kterým bude moci měnit vlastnosti. Knihovnu bude možno kompilovat do podoby sdílené (dynamické) nebo statické za pomoci již mnou vytvořených skriptů, ve kterých budou kroky pro kompilaci pomocí build systému CMake a zvoleného překladače.

2.1.2. Demo desktopová aplikace

Desktopová aplikace bude sloužit jako ukázková část projektu. Uživatel bude mít tu možnost sestavit kromě knihovny samotné i desktopovou demo aplikaci, ve které bude pár ukázek funkcí knihovny. Bude možnost měnit vlastnosti světa, přidávat objekty a nastavovat jejich vlastnosti. Desktopová aplikace bude využívat knihovny SFML pro vytvoření okna, kontextu okna a do něho následné vykreslování. Pro uživatelské rozhraní bude použita velice populární knihovna Dear ImGui. Tato knihovna potřebuje k fungování backend pro vykreslení uživatelského rozhraní, a tak bude použita malá knihovna ImGui-SFML pro tento účel.

2.2. Použité technologie

Tato část dokumentace se věnuje technologiím, které jsou v projektu použity. Knihovna i demo aplikace jsou psány v jazyce C++, přičemž demo aplikace využívá knihoven SFML pro vytvoření okna a následné kreslení do něho a ImGui pro uživatelské rozhraní. Jako build systém jsem zvolil CMake, což je nástroj pro automatizaci procesu konfigurace a kompilace projektu.

2.2.1. C++

C++ je programovací jazyk, který byl vyvinut jako rozšíření jazyka C. Byl vytvořen Bjarne Stroustrupem v 80. letech 20. století. C++ kombinuje výkonnost a efektivitu jazyka C s možnostmi objektově orientovaného programování.

Objektově orientované programování (OOP) je paradigmatický přístup k programování, který se zaměřuje na modelování programu jako kolekce objektů, které interagují mezi sebou. Každý objekt může mít vlastní data a funkce, které s nimi manipulují.

C++ nabízí další funkce oproti jazyku C, jako jsou třídy, polymorfismus, dědičnost, výjimky, šablony a mnoho dalších. Tento jazyk je široce používán v softwarovém vývoji pro

vytváření aplikací, her, operačních systémů, systémového softwaru a mnoha dalších aplikací. Je to silný a flexibilní jazyk, který umožňuje efektivní řešení různých programátorských problémů. [1]

2.2.2. CMake

CMake je open-source nástroj určený k automatizaci procesu kompilace a sestavení softwarových projektů. Je navržen tak, aby byl multiplatformní a umožňoval snadné a efektivní spravování složitých projektů v různých prostředích. CMake poskytuje jednoduchý a deklarativní způsob popisu konfigurace a sestavení projektu pomocí souborů CMakeLists.txt, které obsahují instrukce pro sestavení softwaru pro různé cílové platformy. CMake poté generuje platformě specifické soubory konfigurace (např. Makefiles pro Unixové systémy nebo projekty pro Visual Studio pro Windows), což usnadňuje proces kompilace a sestavení softwaru na různých platformách. [2]

2.2.3. SFML

SFML je zkratka pro „Simple and Fast Multimedia Library“. Jedná se o open-source knihovnu pro vývoj her a multimediálních aplikací v jazyce C++. Poskytuje sadu nástrojů pro práci s grafikou, zvukem, okny, networkingem a vstupy od uživatele. [3]

2.2.4. Dear ImGui

Dear ImGui je open-source knihovna psaná v jazyce C++, která umožňuje jednoduše vytvářet uživatelská rozhraní (GUI) pro desktopové aplikace. Jedná se o lehkou a snadno použitelnou knihovnu, která se často používá v herním vývoji nebo vývoji různých nástrojů.

Dear ImGui pracuje v „Immediate Mode“ stylu, což znamená, že uživatelské rozhraní se vykresluje a aktualizuje přímo ve hlavní smyčce vykreslování programu bez potřeby ukládání stavu GUI prvků.

Knihovna poskytuje širokou škálu GUI prvků, včetně tlačítek, textových polí, seznamů, rozbalovacích seznamů, sliderů apod. Knihovna je flexibilní a umožňuje uživatelům vlastní GUI prvky. [4]

2.2.5. CLion

CLion je integrované vývojářské prostředí (IDE) vyvinuté českou společností JetBrains pro programování v jazycích C a C++. Poskytuje vývojářům pomoc při psaní kódu, analýzu kódu a další nástroje pro produktivitu. CLion nabízí funkce jako syntax highlighting,

dokončování kódu, nástroje pro ladění, integraci s verzovacími systémy a podporu různých systémů pro sestavení projektu.

2.3. Pojmy

Tato část dokumentace se věnuje pojmům v dokumentaci a slouží jako stručná vysvětlivka pro každý z pojmů.

2.3.1. Knihovna

Knihovna je kolekce funkcí, které mohou být použity jinými programy. Knihovny mohou být staticky nebo dynamicky linkovány s ostatními programy.

Statické linkování připojí funkce knihovny rovnou při kompilaci, což vede k lepšímu výkonu a zároveň jsou tedy funkce knihovny součástí spustitelného souboru.

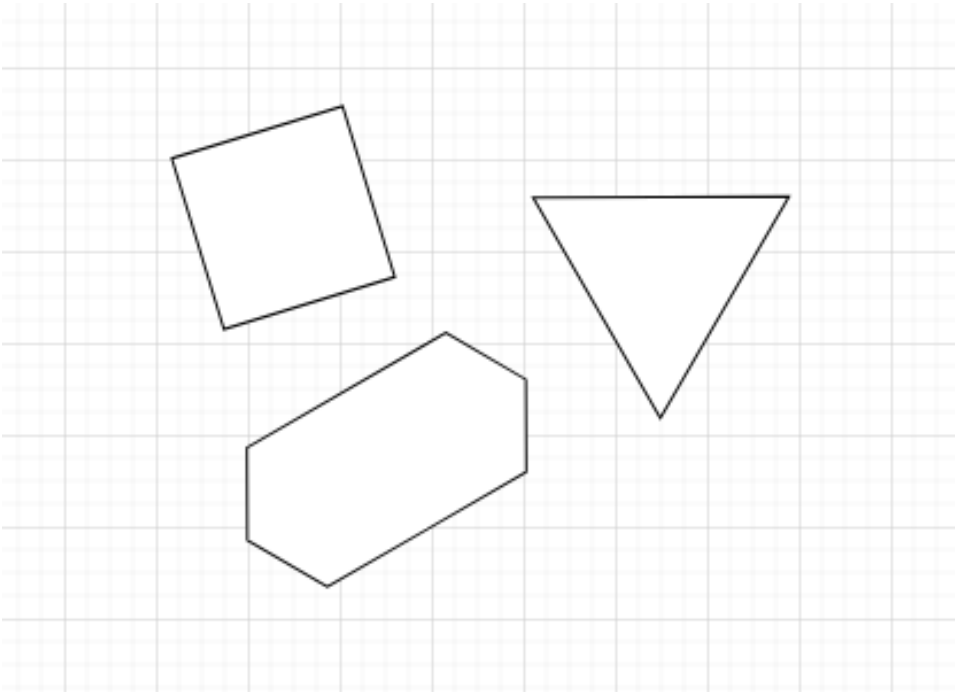
Dynamické linkování umožňuje programu odkazovat se na funkce knihovny za běhu, což má za výhodu menší velikost spustitelného souboru, možnost sdílet knihovnu mezi několika aplikacemi a menší využití paměti počítače. Dynamické linkování má ale za následek mírné zpomalení aplikace oproti statickému, které načte potřebné funkce jen jednou. Tyto rozdíly jsou ale v dnešní době s dnešním hardwarem zanedbatelné. [5]

2.3.2. Desktopová aplikace

Desktopová aplikace je software, který je určen k běhu na počítači a běží přímo na operačním systému. Může ve většině případů běžet kompletně bez přístupu k internetu na rozdíl od webových stránek. Desktopové aplikace mají uživatelské rozhraní pro interakci s uživatelem za pomocí myši nebo klávesnice. Mohou mít různé účely jako aplikace pro komunikaci, zábavní aplikace, aplikace pro správu souborů, aplikace pro úpravu médií a další.

2.3.3. Polygon

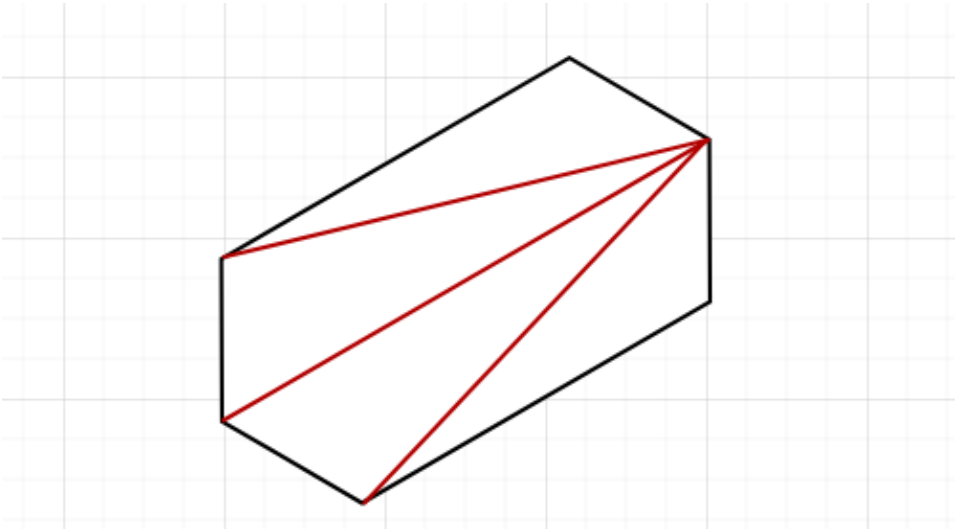
Polygon, také mnohoúhelník, je v geometrii uzavřená rovina obrazce, která je tvořena úsečkami spojující jednotlivé vrcholy nebo body. Tyto úsečky se nazývají hranami polygonu. Každý polygon má minimálně tři vrcholy a tři hrany.



Obrázek 2 ilustrace různých polygonů (mnohoúhelníků)

2.3.4. Triangulace

Triangulace je proces rozdělení oblasti roviny na trojúhelníky. Jedná se o techniku používanou často v počítačové grafice a geometrických algoritmech. Triangulace oblasti je provedena tak, že jsou vytvořeny trojúhelníky, jejichž vrcholy jsou body původního polygonu. Zkombinováním všech trojúhelníků dostaneme původní polygon.



Obrázek 3 ilustrace triangulace polygonu

2.3.5. Kolize

Kolize znamená srážka nebo střetnutí. V kontextu 2D simulace fyziky se bavíme o rázu těles, kdy vznikají při srážce těles síly, které změní jejich pohyb. Mírou rázu těles je restituce k. Při dokonale pružné srážce je $k = 1$, při dokonale nepružné naopak $k = 0$. Cokoliv mezi 0 a 1 předá zpět jen část energie tělesa. [6]

2.3.6. Broad-phase

Broad-phase je v kontextu detekce kolizí hrubé filtrování objektu ve světě, mezi kterými by mohla potencionálně být kolize. Účelem je eliminovat zbytečné kontroly mezi objekty, což může vést k optimalizaci simulace.

Mezi známé metody broad-phase patří grid-based metody – svět se rozdělí do mřížky a kolize se kontroluje mezi objekty, které leží ve stejné nebo sousední buňce mřížky, nebo například metoda AABB (axis aligned bounding box) – všechny objekty mají kolem sebe ohraničující boxy, testuje se potom nejdříve překrytí těchto boxů před reálným testem pro kolizi.

2.3.7. Narrow-phase

Narrow-phase je v kontextu detekce kolizí potvrzení kolizí mezi objekty poté, co proběhne broad-phase. Cílem je zjistit, jestli u dvojice objektů došlo ke kolizi, a pokud ano, tak zjistit detailní informace o kolizi jako například body dotyku, normály odrazu, síly odrazu a další.

2.3.8. Uživatelské rozhraní

Uživatelské rozhraní, také UI či GUI, slouží pro interakci uživatele s aplikací či webovou stránkou. Zahrnuje prvky jako jsou tlačítka, slidery, textové pole, menu a další.

2.3.9. API

API (application programming interface) je rozhraní pro programování různých aplikací. Jde o sbírku funkcí nějaké knihovny, které může vývojář zavolat z jeho aplikace či programu. Příklady API jsou OpenGL, Vulkan, Metal a DirectX pro práci s grafickou kartou.

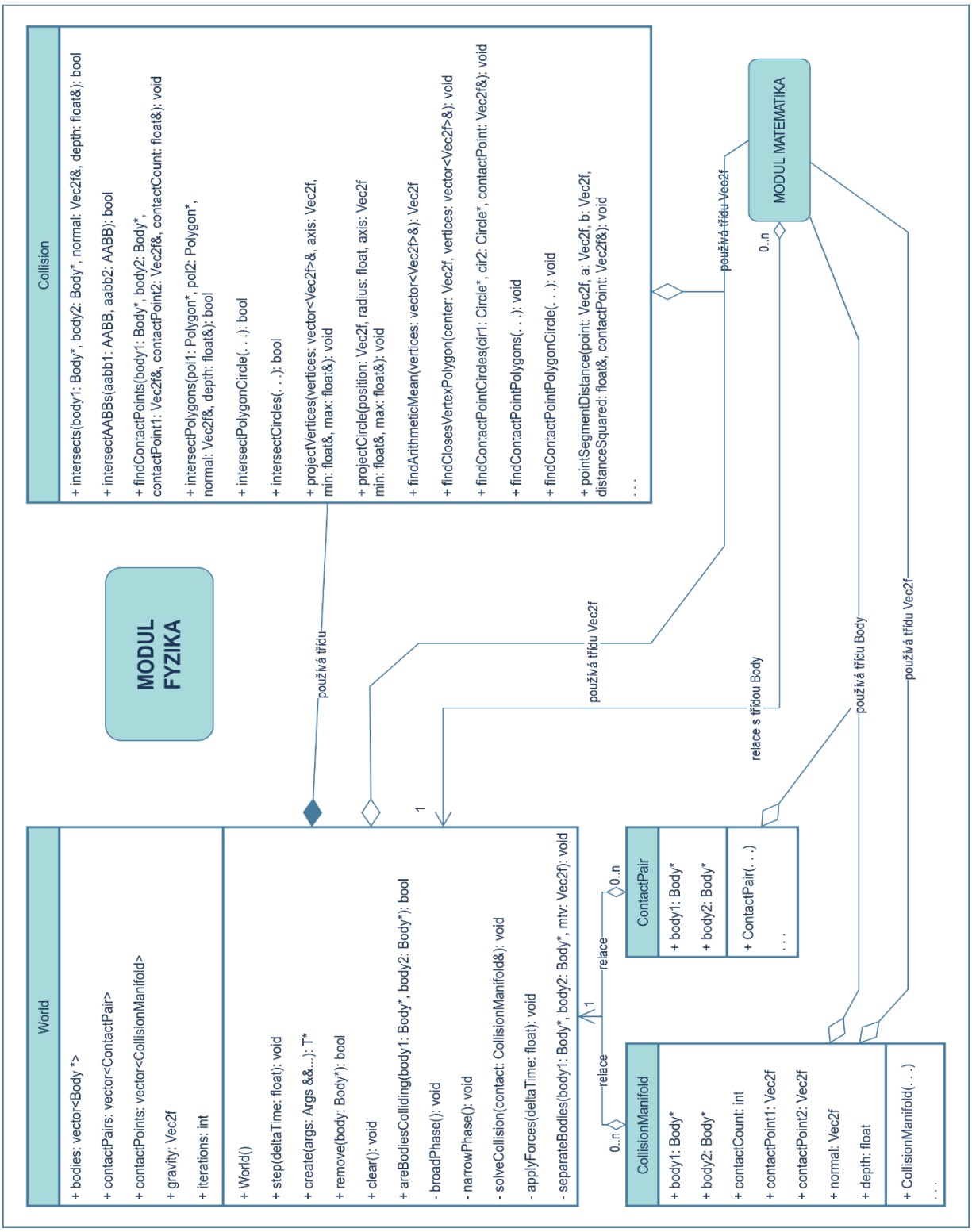
3. Praktická část

Tato část dokumentace se věnuje praktické části mé práce, popisuje, jak vývoj probíhal a jakých výsledků jsem svojí prací dosáhl.

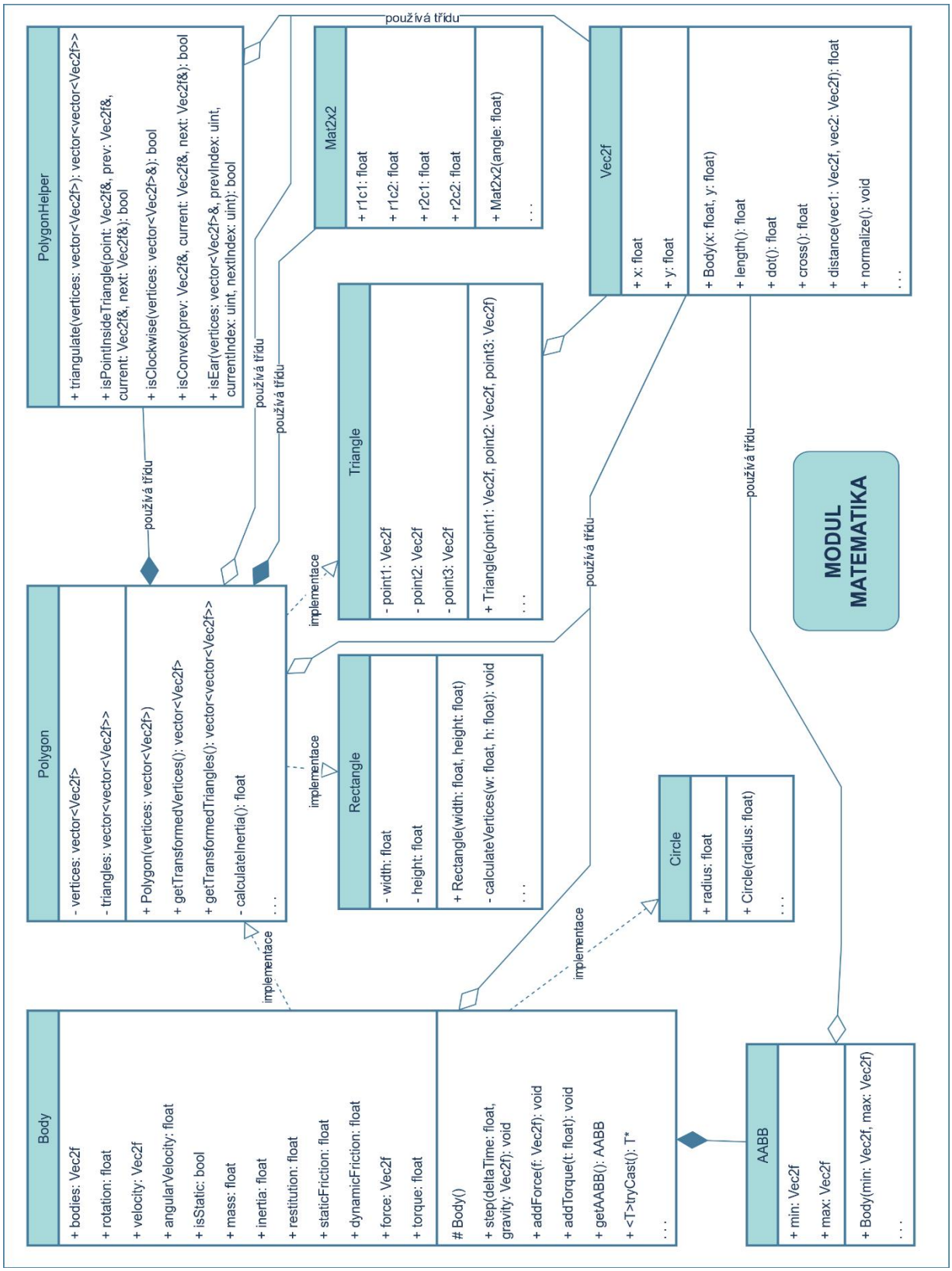
3.1. Rešerše existujících řešení

Jak je již zmíněno v úvodu, inspiroval jsem se knihovnou Box2D. Její autor má na svém GitHub profilu i odlehčenější verzi knihovny Box2D – Box2D-Lite, která je pro studijní účely. Zaujaly mě tutoriály uživatele Two-Bit Coding na YouTube na téma fyzických enginů, ve kterých odkazuje na články Chrise Heckera o simulaci fyziky. Podíval jsem se lehce i na knihovnu Matter.js

3.2. Implementace – Knihovna



Obrázek 4 diagram knihovny část 1



Obrázek 5 diagram knihovny část 2

Knihovna je rozdělena do dvou „modulů“. V první knihovně jsou třídy, co se starají o chod fyziky – definování světa, algoritmy pro kolizi a běh simulace. Druhý modul se stará o převážně matematické výpočty a například definování objektů.

Třída World slouží v knihovně k vytvoření světa, do kterého uživatel může přidávat objekty. Všechny objekty dědí ze třídy Body. Lze nastavit počáteční hodnotu gravitace, která bude na objekty přidané ve světě působit. Zároveň lze nastavit počet iterací který zajistí přesnost simulace. Svět pod sebou drží list ukazatelů na objekty a informace o objektech a zároveň má informaci o gravitaci, která bude na objekty při simulaci působit.

```
// hlavní hlavičkový soubor knihovny (zde je celý obsah knihovny)
#include "Fyziks.h"

int main() {
  World world; // vytvoření světa
  world.gravity = Vec2f(0.0f, -9.81f); // nastavení gravitace
  world.iterations = 16; // iterace (přesnost simulace)

  return 0;
}
```

Obrázek 6 kód - vytvoření světa

Jako hlavní způsob simulace je funkce step, kterou je možné zavolat na instanci světa. Přebírá parametr deltaTime, což vyjadřuje časový skok v simulaci, za který se mají objekty někam pohnout. V praxi je většinou deltaTime čas od minulého snímku po současný, aby simulace probíhala plynule, a aby i zásek byl dopočítán a knihovna tak běžela v reálném čase. Parametr deltaTime úzce souvisí s iteracemi a to tak, že deltaTime je vydělen iteracemi – vzniká tak časový pod-skok (substep). Simulace je o to přesnější, protože se kalkulace dělají v menších intervalech. Nevýhoda je, že simulace může být výrazně náročnější na výpočet čím vyšší je číslo iterací.

```
float timeStep = 1.0f / 60.0f; // časový skok
float elapsed = 0.0f; // počítadlo
Clock clock;
// hlavní smyčka simulace
while (true) {
  // resetujeme hodiny a zároveň získáme již uběhnutý čas od startu hodin
  deltaTime = clock.restart();
  elapsed += deltaTime;
  // dopočítání simulace, i když se zasekne
  while (elapsed >= timeStep) {
    world.step(timeStep); // hlavní metoda simulace
    elapsed -= timeStep;
  }
}
```

Obrázek 7 pseudo kód – hlavní smyčka simulace

V každém kroku se provede daný počet iterací, ve kterých se nejdříve vymažou informace o objektech v minulém kroku simulace. Následuje působení sil podle Newtonových pohybových zákonů, konkrétně druhý Newtonův zákon.

3.2.1. Newtonův druhý zákon

Také nazývaný jako zákon síly. Newtonův druhý zákon udává, že síla působící na těleso je úměrná součinu jeho hmotnosti a zrychlení, které mu uděluje. $\vec{F} = m\vec{a}$

Poté se všechny objekty ve světě hrubě přefiltrují pomocí broad-phase. Vznikne tak list dvojice objektů, které mohou mezi sebou potencionálně kolidovat. Po provedení broad-phase následuje narrow-phase. V této fázi dojde k detailní kontrole pro kolizi pro předem získané dvojice objektů a k vyřešení případných kolizí.

Aby bylo možné zjistit, jestli u dvojic objektů dochází ke kolizi nebo zda se třeba jen překrývají, je zapotřebí nějaká metoda či algoritmus, který tuto informaci dokáže poskytnout. Vybral jsem algoritmus SAT (separating axis theorem), protože je relativně jednoduchý na pochopení i na implementaci.

3.2.2. SAT (separating axis theorem)

Separating axis theorem, zkráceně SAT, je algoritmus nebo metoda na zjištění, zda dva konvexní polygony mají mezi sebou kolizi. Definice SAT je „Dva uzavřené konvexní objekty jsou disjunktní, pokud existuje čára („oddělující osa“), na kterou jsou projekce obou objektů disjunktní.“. [7]

```
Axis[] axes = // osy, na kterých bude projekce probíhat

// projdeme osy
for (int i = 0; i < axes.length; i++) {
  Axis axis = axes[i];

  // projekce obou objektů na osu
  Projection p1 = body1.project(axis);
  Projection p2 = body2.project(axis);

  // překrývají se osy?
  if (!p1.overlap(p2)) {
    // osy se nepřekrývají -> našla se mezera mezi objekty
    return false;
  }
}

// prošli jsme všechny osy a ani jedna projekce neukázala mezeru
// objekty mají mezi sebou kolizi
return true;
```

Obrázek 8 pseudo kód SAT 1

SAT funguje tak, že opakovaně projektuje objekty na osy. Za osy, na které se bude projektovat, se považují jednotlivé strany polygonu. V případě testovaní kolize mezi dvěma kruhy, stačí jen zkontrolovat, jestli jejich vzdálenost od středů je větší než součet obou poloměrů. Jestliže algoritmus najde takovou projekci na osu, kde vznikne mezera mezi objekty, objekty tak spolu nekolidují. Jestliže se nenajde žádná taková osa, objekty spolu kolidují. [8]

Kromě informace, jestli mezi sebou objekty mají kolizi, lze vyčíst i minimální translační vektor, o který lze objekty posunout tak, aby se nepřekrývaly.

3.2.2.1. MTV (minimální translační vektor)

```
float overlap = float.Max
Axis smallest = null;

Axis[] axes = // osy, na kterých bude projekce probíhat

// projdeme osy
for (int i = 0; i < axes.length; i++) {
  Axis axis = axes[i];

  // projekce obou objektů na osu
  Projection p1 = body1.project(axis);
  Projection p2 = body2.project(axis);

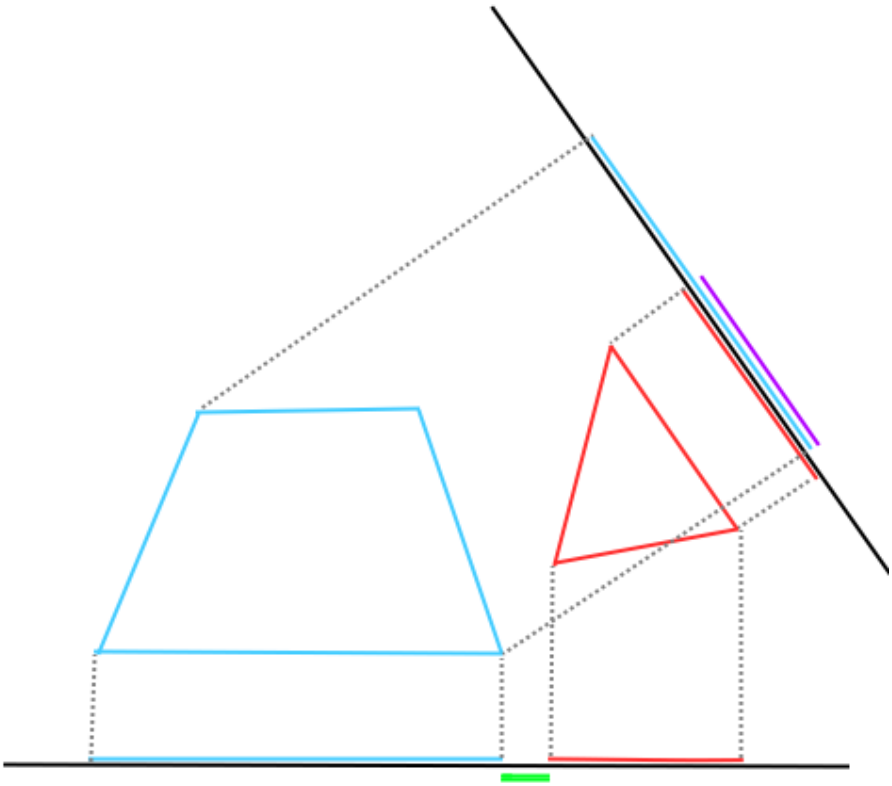
  // překrývají se osy?
  if (!p1.overlap(p2)) {
    // osy se nepřekrývají -> našla se mezera mezi objekty
    return false;
  } else {
    // zjistíme o kolik se překrývají
    double o = p1.getOverlap(p2);

    if (o < overlap) {
      // pokud jsme našli menší překrytí, nastavíme ho
      overlap = o;
      smallest = axis;
    }
  }
}

// prošli jsme všechny osy a ani jedna projekce neukázala mezeru
// objekty mají mezi sebou kolizi
return (true, smallest, overlap);
```

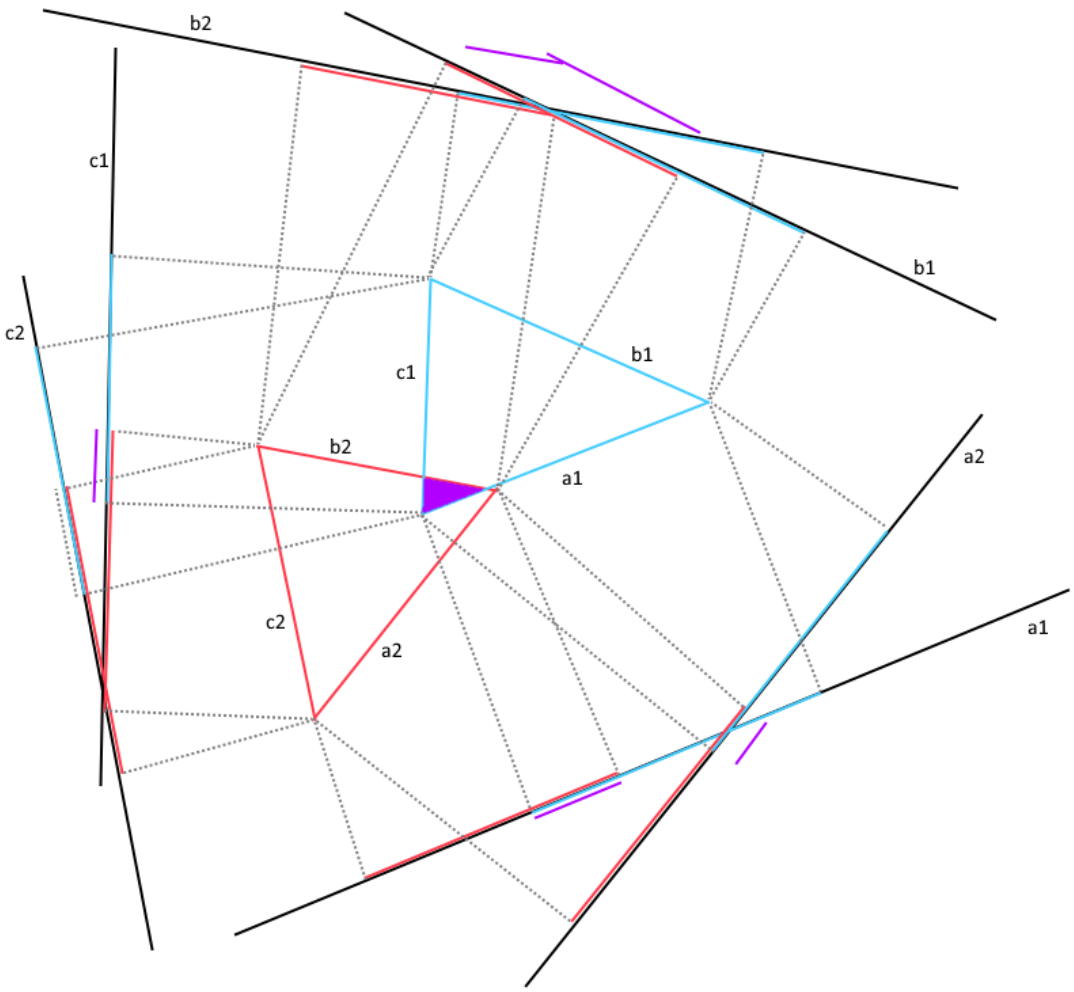
Obrázek 9 pseudo kód SAT 2

Při procházení všech os si můžeme zaznamenat minimální translační vektor (směr a vzdálenost), se kterým můžeme posunout objekty tak, aby se nepřekrývaly. Jedno těleso se posune o polovinu získané vzdálenosti ve směru vektoru, druhé se posune opačným směrem také o polovinu vzdálenosti.



Obrázek 10 ukázka algoritmu SAT 1

Na obrázku lze vidět, že se provedly celkem dvě projekce na dvě osy. Fialová barva značí překrývající se projekce a zelená barva představuje mezeru mezi polygony. Jelikož se našla mezera, je tento případ vyhodnocen jako nekolizní.



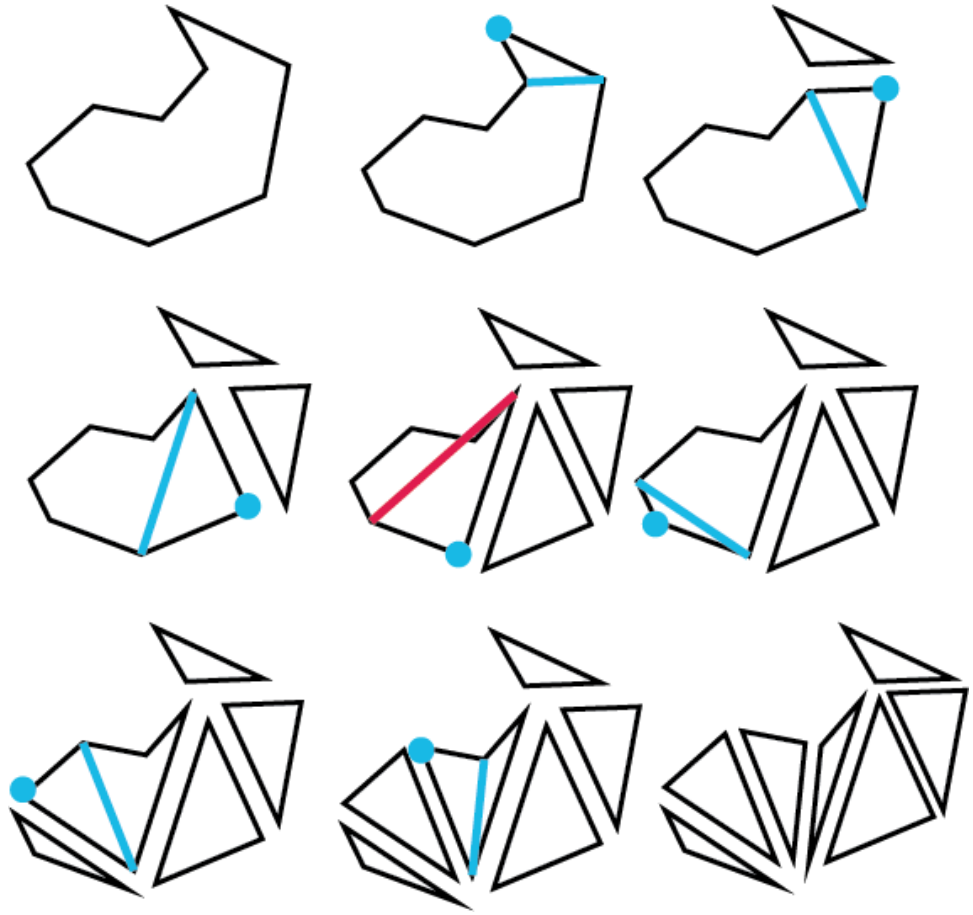
Obrázek 11 ukázka algoritmu SAT 2

Na obrázku lze vidět, že se pro dva trojúhelníky otestovalo celkem 6 os, na které byly provedeny projekce obou trojúhelníků – modrá a červená barva. Fialová barva ukazuje, kde se projekce překrývají. Jelikož se nenašla ani jedna osa, která by měla projekci bez mezery, mají tyto dva trojúhelníky mezi sebou kolizi. Jako MTV se použije nejmenší překrytí.

Zde ale nadešel problém. Algoritmus SAT nefunguje pro konkávní objekty. Musel být tedy naimplementován triangulační algoritmus, který rozdělí polygony na trojúhelníky, které budou vždy konvexní. Poté se bude algoritmus SAT provádět mezi jednotlivými trojúhelníky polygonů.

Jako algoritmus na triangulaci jsem zvolil Ear clipping algoritmus právě kvůli jeho jednoduchosti.

3.2.3. Ear clipping algoritmus



Obrázek 12 ear clipping algoritmus [9]

Z obrázku lze vydedukovat, že se jedná o triangulaci konkávního polygonu (konkávní polygony mají alespoň jeden vnitřní úhel větší než 180°). Ear clipping algoritmus se snaží useknout „ucho“ – trojúhelník, dokud nezbyl žádný další trojúhelník. Výsledné trojúhelníky by měly zpět tvořit původní polygon. Viz *obrázek 12 a 13*

```
Vertex[] vertices = // pole bodů
Triangle[] triangles = []; // pole výsledných trojúhelníků
// pokud je polygon definován body po směru hodinových ručiček
if (vertices.areClockwise) {
  vertices.reverse(); // obrátíme
}
// procházíme všechny body, dokud nezbyde poslední trojúhelník
while (vertices.length > 3) {
  bool earFound = false;
  // procházíme všechny body
  for (int i = 0; i < vertices.length; i++) {
    int previousIndex = (i - 1 + vertices.length) % vertices.length; // bod v
pořadí za hlavním bodem
    int currentIndex = i; // hlavní bod
    int nextIndex = (i + 1) % vertices.length; // bod před hlavním bodem
    Vertex previous = vertices[previousIndex];
    Vertex current = vertices[currentIndex];
    Vertex next = vertices[nextIndex];
    // pokud je vektorový součin mezi vektorama
    // z hlavního bodu do předchozího a dalšího bodu vetší než 0
    // -> vnitřní úhel u hlavního bodu < 180°
    if (isConvex(previous, current, next)) {
      // pokud žádný jiný bod neleží v trojúhelníku tvořeným z previous, current
a next
      if (isEar(previous, current, next, vertices)) {
        // našli jsme "ucho" -> přidáme do pole
        triangles.add({previous, current, next})

        // odstraníme hlavní bod z pole ostatních bodů
        vertices.removeAt(i);
        // našli jsme "ucho"
        earFound = true;
        break;
      }
    }

    if (!earFound) {
      break;
    }
  }
}
// přidání posledního trojúhelníku, který zbyl
triangles.add({vertices[0], vertices[1], vertices[2]});
return triangles;
```

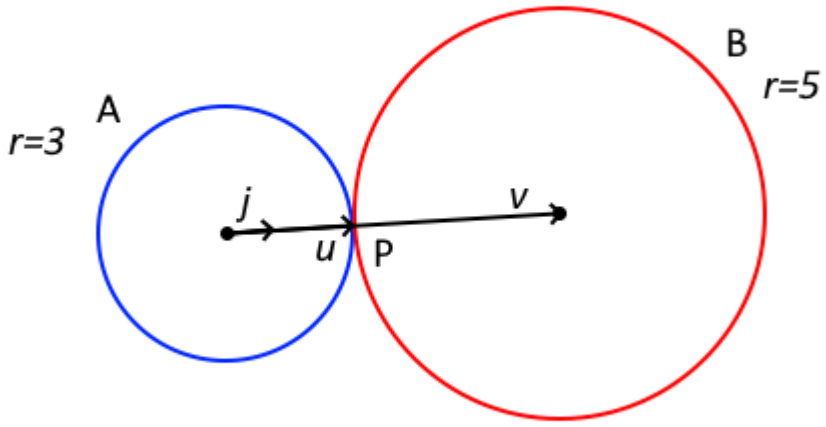
Obrázek 13 pseudo kód pro ear clipping algoritmus

Pokud objekty spolu kolidují, posuneme je minimálním translačním vektorem a zjistíme jejich body doteku. Poté už následuje jen vypočtení odrazových a třecích sil, které následně přeneseme na objekty.

3.2.4. Zjištění bodů doteku

Před výpočtem odrazových sil je potřeba znát, v jakých bodech se objekty při kolizi dotkly. Ty získáme následovně:

- Pro kolizi mezi dvěma kruhy:
  Spočítáme vektor vedený ze středu prvního kruhu do středu druhého, ze kterého uděláme jednotkový vektor. Tento jednotkový vektor poté vynásobíme poloměrem prvního kruhu a výsledný vektor nakonec posuneme o pozici prvního kruhu, abychom docílili souřadnic bodu dotyku.



Obrázek 14 bod doteku kruhů

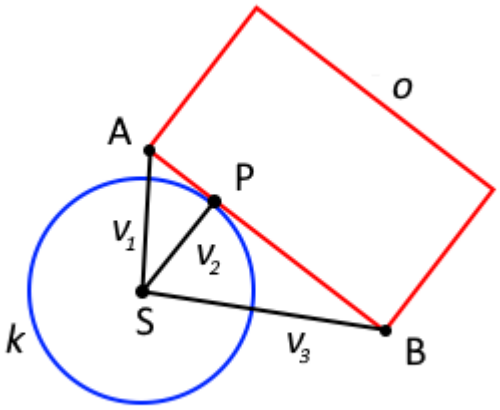
Na obrázku lze vidět bod dotyku P a kruhy A a B. Oba mají poloměr r. Vektor v je veden ze středu kruhu A do středu kruhu B, jeho jednotková verze je označena písmenem j. Finální vektor u je tedy vektor j vynásobený skalárem, což je poloměr r kruhu A.

- Pro kolizi mezi kruhem a polygonem:
  Algoritmus pro zjištění bodu dotyku mezi kruhem a polygonem opakovaně prochází všechny strany polygonu a hledá nejbližší bod na dané straně ke středu kruhu. Nejbližší bod se určí pomocí zjednodušené rovnice pro vzdálenost bodu od přímky, která vypadá následovně:

$$v = \frac{ax + by}{a^2 + b^2}$$

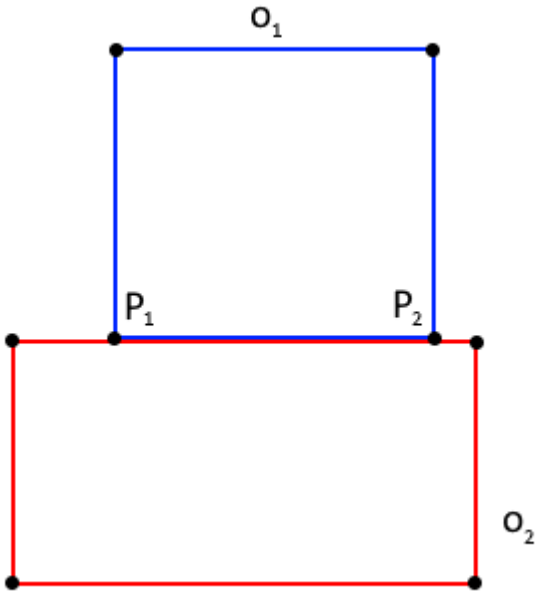
Obrázek 15 rovnice pro vzdálenost bodu od přímky

Ve jmenovateli je neodmocněná délka strany polygonu a v čitateli je skalární součin mezi vektorem strany polygonu a vektorem vedeného z jednoho bodu strany polygonu do středu kruhu. Pokud je vzdálenost v menší než 0, pak je nejbližší bod A, pokud je vzdálenost větší něž 1, pak je nejbližší bod B. V ostatních případech bod zjistíme jako $A + \overrightarrow{AB} * v$. Viz obrázek 16



Obrázek 16 bod doteku kruh a polygon

- Pro kolizi mezi polygony:
  Dva polygony mohou mezi sebou mít až dva body doteku, a to v případ, že jejich dotýkající se strany jsou vedeny stejnou přímkou. Tento algoritmus je velice podobný algoritmu pro kruh a polygon. Algoritmus prochází body jednoho polygonu a zároveň prochází strany druhého polygonu a naopak. Opět hledá nejbližší bod ke straně druhého polygonu za pomoci stejné rovnice jako na obrázku 15. Zároveň si algoritmus ukládá nejkratší vzdálenost a pokud se najde další bod s dostatečně podobnou vzdáleností (kvůli floating point aritmetice), je počet bodů dotyku roven dvěma. Viz obrázek 17

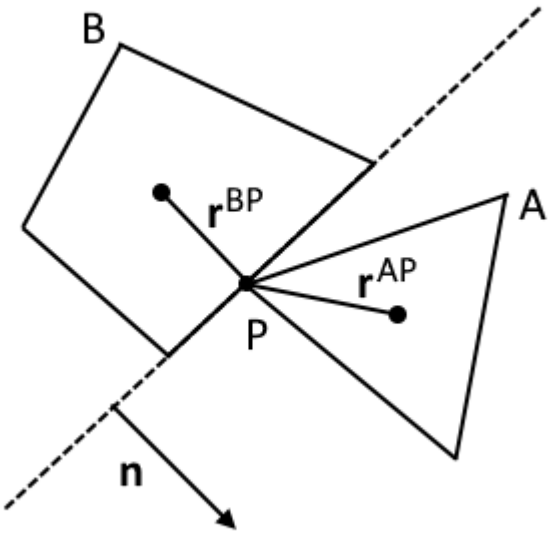


Obrázek 17 body doteku polygony

Na obrázku lze vidět dva polygony o_1 a o_2. Algoritmus prošel jejich strany a body a našel dva body doteku P_1 a P_2, protože jejich vzdálenost od strany druhého polygonu byla velice podobná.

3.2.5. Kalkulace impulsních sil

Samotná kalkulace pro výpočet impulsů, jak pro rotaci, tak pro změnu rychlosti objektů, je poměrně složitá. Pro ukázku máme dva objekty – A a B:



Obrázek 18 kalkulace impulsů – příklad

Na obrázku lze vidět dva objekty – A a B. Bod P je bod doteku, $\boldsymbol{n}$ je normálový vektor strany, kde se stala kolize. Vektory $\boldsymbol{r}^{BP}$ a $\boldsymbol{r}^{AP}$ jsou vedeny ze středů objektů k bodu dotyku v daný moment kolize.

Impulsní skalár j, prozatím jen pro změnu pozic objektů, nikoli i změnu rotace, se vypočítá následovně:

$$j = \frac{-(1+e)\boldsymbol{v}_1^{AB} \cdot \boldsymbol{n}}{\boldsymbol{n} \cdot \boldsymbol{n}\left(\frac{1}{M^A} + \frac{1}{M^B}\right)}$$

Obrázek 19 rovnice pro výpočet impulsního skaláru pro změnu rychlosti

kde e je koeficient restituce obou objektů, pro jednoduchost lze vzít menší hodnotu restituce jednoho z objektů, $\boldsymbol{v}_1^{AB} \cdot \boldsymbol{n}$ je relativní rychlost objektů po normále $\boldsymbol{n}$. Relativní rychlost lze vypočítat takto:

$$\boldsymbol{v}_1^{AB} = \boldsymbol{v}^B - \boldsymbol{v}^A$$

Obrázek 20 rovnice pro výpočet relativní rychlosti

V rovnici na obr. č.19 pro výpočet impulsu M představuje hmotnosti každého z objektů. Nutno podotknout že tato rovnice nebere v potaz okolní síly působící na tělesa v moment dotyku, jako jsou gravitace a tření. Pro jednoduchost to ale stačí.

Finální rovnice pro výpočet impulsního skaláru j pro změnu rychlosti i rotace vypadá následovně:

$$j = \frac{-(1+e)\boldsymbol{v}_1^{AB} \cdot \boldsymbol{n}}{\boldsymbol{n} \cdot \boldsymbol{n}\left(\frac{1}{M^A} + \frac{1}{M^B}\right) + \frac{(\boldsymbol{r}_\perp^{AP} \cdot \boldsymbol{n})^2}{I^A} + \frac{(\boldsymbol{r}_\perp^{\boldsymbol{BP}} \cdot \boldsymbol{n})^2}{I^B}}$$

Obrázek 21 rovnice pro výpočet impulsního skaláru pro změnu rychlosti a rotace

V rovnici nám přibylo pár výrazů - $\boldsymbol{r}_\perp^{AP}$ nebo $\boldsymbol{r}_\perp^{BP}$ jsou jen normály na vektory $\boldsymbol{r}^{BP}$ a $\boldsymbol{r}^{AP}$. Písmeno I značí setrvačnost objektů. [10]

3.2.5.1. Setrvačnost

Setrvačnost je vlastnost nebo veličina, kterou má každé hmotné těleso. Je to schopnost tělesa odolávat vnějším silám, které na něj působí.

Výpočet setrvačnosti pro kruh je jednoduchý, protože to je pravidelný objekt.

$$I = \frac{mr^2}{2}$$

Obrázek 22 výpočet setrvačnosti pro kruh

V této rovnici m značí hmotnost kruhu a r značí jeho poloměr.

Dlouho se mi však nedařilo najít správný výpočet setrvačnosti pro konkávní polygony, který by fungoval. Vypočítat setrvačnost pro pravidelné objekty v rovině je poměrně lehké, ale pokud jde o nepravidelné objekty, například konkávní polygon, nastane problém. Nakonec jsem se spokojil s výpočtem ze stránky Stack Overflow.

```
float calculateInertia() {
  float numerator = 0.0f;
  float denominator = 0.0f;

  for (int i = 0; i < vertices.size(); ++i) {
    Vec2f current = vertices[i];
    Vec2f next = vertices[(i + 1) % vertices.size()];

    numerator += Vec2f::cross(next, current) * (
      Vec2f::dot(next, next) +
      Vec2f::dot(next, current) +
      Vec2f::dot(current, current)
    );
    denominator += Vec2f::cross(next, current);
  }

  return (mass / 6) * (numerator / denominator);
}
```

Obrázek 23 kód pro výpočet setrvačnosti konkávního polygonu [11]

Výsledný impuls, který přičteme k rychlosti objektů je:

$$\frac{j}{M}$$

Obrázek 24 finální výpočet impulsu pro rychlost

kde j je skalár a M je hmotnost. Pro rotaci je to:

$$\frac{\boldsymbol{r} \times j}{I}$$

Obrázek 25 finální výpočet impulsu pro rychlost rotace

kde j je opět skalár, $\boldsymbol{r}$ je rychlost tělesa v době kolize a I je setrvačnost. Pro rotaci i pro rychlost platí, že pokud počet bodů dotyku je roven dvěma, výpočet probíhá dvakrát pro každý z bodů dotyků. Přidání výsledných impulsních sil proběhne také dvakrát.

3.2.6. Použití knihovny

Použití knihovny ve vašem vlastním projektu je relativně jednoduché. Základ je naklonování knihovny do vašeho projektu a jejího následného linkování s vaším projektem. Další krok je zahrnout hlavičkové soubory knihovny do vašeho projektu:

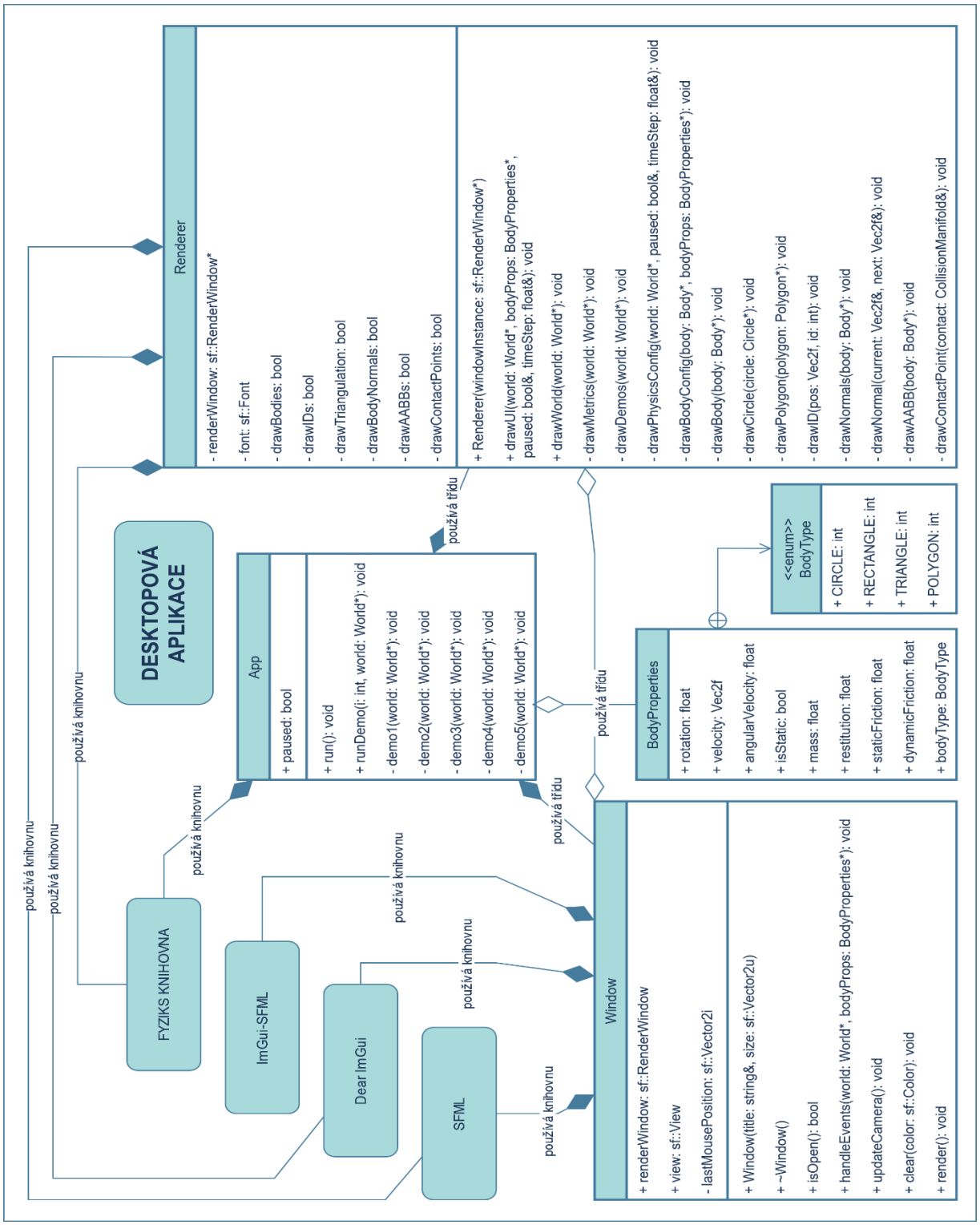
```
// Hlavní hlavičkový soubor
#include <Fyziks.h>
using namespace fy;

int main()
{
  // Vytvoření světa a definování vlastností
  World world;
  world.gravity = Vec2f(0.0f, -9.81f);
  world.iterations = 10;
  // Vytvoření kruhu a přidání vlastností
  auto body = world->create<Circle>(25.0f);
  body->position = Vec2f(100.0f, 100.0f);
  body->velocity = Vec2f(10.0f, 0.0f);
  body->rotation = Misc::toRadians(45.0f);
  body->restitution = 1.0f;
  // Vytvoření polygonu
  auto otherBody = world->create<Polygon>(std::vector<Vec2f>{
    Vec2f(0.0f, 0.0f),
    Vec2f(50.0f, 0.0f),
    Vec2f(25.0f, 50.0f)
  });
  otherBody->position = Vec2f(200.0f, 100.0f);
  // Hlavní smyčka
  while (true)
  {
    // Aktualizuje svět pomocí časového kroku
    world.step(1.0f / 60.0f);
    // Kontrola, zda nedochází ke kolizi dvou těles
    if (world.areBodiesColliding(body, otherBody))
    {
        // ...
    }
    // Vykreslení světa
    // ...
  }
  return 0;
}
```

Obrázek 26 kód použití knihovny

Na obrázku lze vidět jednoduché použití knihovny. Je zde vytvořen svět s počátečními vlastnostmi a do něj jsou přidané dva objekty. Simulace probíhá v hlavní smyčce a uživatel může zde i vyčíst jakékoliv informace o objektech v simulaci.

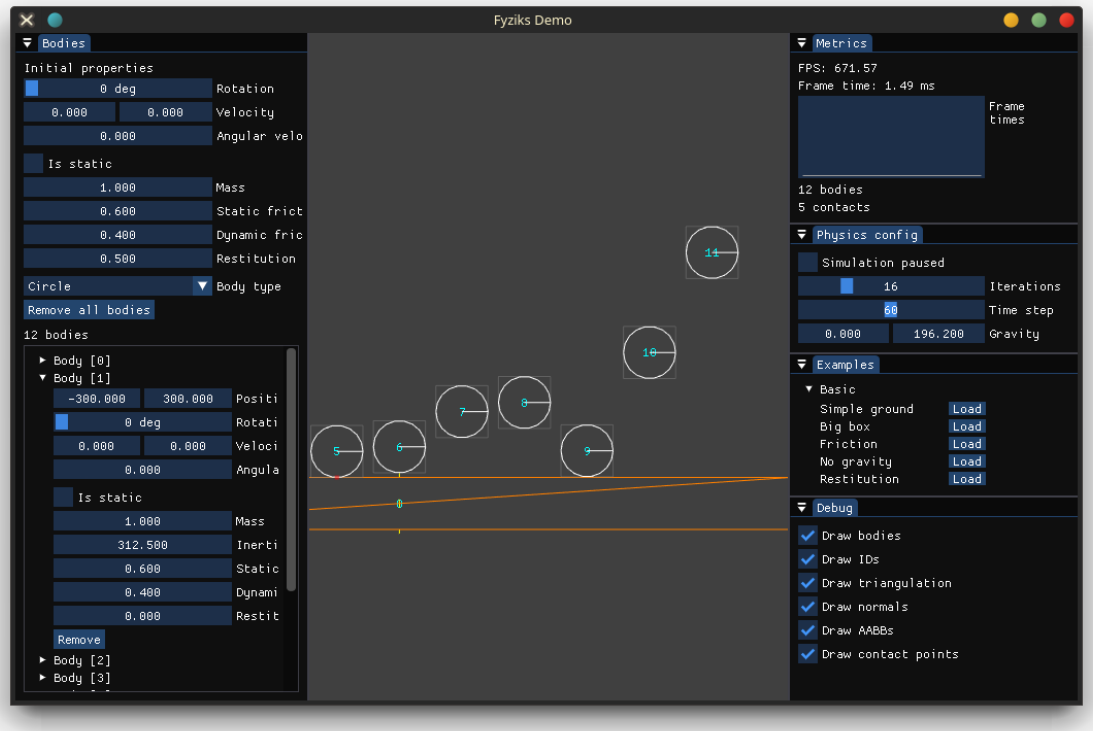
3.3. Implementace – Desktopová demo aplikace



Obrázek 27 diagram desktopové aplikace

Desktopová aplikace slouží jako ukázka, jaké má knihovna funkce a co umí. Poskytuje uživatelům interaktivní prostředí pro experimentování s knihovnou. Aplikace je rozdělena na několik tříd, které se starají o vykreslování, držení kontextu okna a například hlavní smyčka fyzické simulace.

3.3.1. Funkce



Obrázek 28 desktopová aplikace

- Celá aplikace se dá ovládat jen za pomoci myši. Kamera snímající svět simulace se ovládá jednoduše:
  - Levé tlačítko – přidání objektu s vlastnostmi definovaných v okénku „Bodies“
  - Pravé tlačítko – posouvání kamerou
  - Rolování kolečkem – přiblížení nebo oddálení kamery

  Okna knihovny Dear ImGui (modro-černé) se dají posouvat, zvětšovat i zmenšovat a dokovat na strany hlavního okna nebo do ImGui oken.
- Okénko „Metrics“ slouží jako zobrazení statistik o běhu simulace. Je zde FPS (frames per second) počítadlo, které zobrazuje, kolik snímků se vykreslilo za poslední sekundu. „Frame time“ znamená, kolik milisekund zabralo vykreslit jeden snímek.

Je zde i statistika, která říká, kolik je ve světě simulace dohromady objektů a kolikrát se dotýkají.

- V okénku „Physics config“ je možné změnit vlastnosti světa, jako je gravitace, počet iterací a časový skok simulace včetně zaškrtávacího políčka pro pozastavení simulace.
- V aplikaci je pět předdefinovaných příkladů, které je možné spustit tlačítkem „Load“ v okénku „Examples“. Je zde například test objektů s různými hodnotami tření nebo například scéna s objekty, které mají různý koeficient restituce.
- V debuggovacím okénku je možné nastavit, zda se mají vykreslovat samotné objekty, jejich ID ve světě, triangulace objektů, normálové vektory na jejich strany, AABBs (axis aligned bounding boxes) objektů a body kontaktu. Toto okénko slouží čistě pro debuggovací účely.
- Hlavní okénko „Bodies“ slouží pro konfiguraci objektů, které mají být přidány nebo jsou již ve světě simulace. Je zde sekce pro nastavení počátečních vlastností objektů. Lze nastavit rotaci, počáteční rychlost i rotační rychlost. Dále lze definovat, zda je objekt statický (nebudou na něho působit vnější síly), jeho hmotnost, hodnoty tření a koeficient restituce. Lze také vybrat tvar objektu ze čtyř předdefinovaných tvarů – kruh, trojúhelník, obdélník a konkávní polygon ve tvaru hvězdy. Druhá sekce slouží pro změnu vlastností již existujících objektů ve světě simulace. Je to pouhý seznam objektů identifikovaných podle identifikačního čísla.

4. Závěr

Toto téma projektu jsem si vybral primárně z důvodu zájmu o herní průmysl, simulaci fyziky a o 2D a 3D grafiku.

Tento projekt slouží jako dobrý studijní materiál o základech simulace fyziky ve 2D prostoru.

Při tvorbě tohoto projektu jsem se naučil základy a principy simulaci fyziky, dále jsem se naučil pracovat s jazykem C++ a prací s pamětí a ukazateli, ale také jak použít knihovnu SFML. Zdokonalil jsem svoje znalosti s knihovnou Dear ImGui.

Projekt je dostupný na webu https://github.com/4iwen/Fyziks ve formě zdrojového kódu a dá se sestavit pomocí build skriptů. Nebo je zde možnost stáhnout si již sestavenou verzi pro Windows, Linux nebo macOS ve sekci Releases.

Nabyté znalosti a zkušenosti z tohoto projektu v budoucnosti určitě použiji nejen v oblasti herního průmyslu.

5. Seznam použité literatury

[1] [Co je C++?] Online. In: ChatGPT. January 2022 version. Dostupné z: OpenAI, https://chat.openai.com/share/db0ff542-37d4-426d-931c-b7137c2bb924 [cit. 2024-02-24].

[2] [Co je to CMake?] Online. In: ChatGPT. January 2022 version. Dostupné z: OpenAI, https://chat.openai.com/share/4a8297d1-be3e-4439-84dc-2237a93ea2ed [cit. 2024-02-24].

[3] [Co je to SFML?] Online. In: ChatGPT. January 2022 version. Dostupné z: OpenAI, https://chat.openai.com/share/6a770f6b-dca0-49c4-a01e-b08ab4c3c33d [cit. 2024-02-24].

[4] [Co je to ImGui?] Online. In: ChatGPT. January 2022 version. Dostupné z: OpenAI, https://chat.openai.com/share/766e4b5a-ebde-4380-b0f1-23ccd2c60249 [cit. 2024-02-24].

[5] Knihovkna (programování). In: *Wikipedia: the free encyclopedia* [Online]. San Francisco (CA): Wikimedia Foundation, 2024, 2001-2024 [cit. 2024-03-19]. Dostupné z: https://cs.wikipedia.org/wiki/Knihovna_(programov%C3%A1n%C3%AD)

[6] Ráz těles. In: *Wikipedia: the free encyclopedia* [Online]. San Francisco (CA): Wikimedia Foundation [cit. 2024-03-19]. Dostupné z: https://cs.wikipedia.org/wiki/R%C3%A1z_t%C4%9Bles

[7] Hyperplane separation theorem. In: *Wikipedia: the free encyclopedia* [Online]. San Francisco (CA): Wikimedia Foundation [cit. 2024-03-19]. Dostupné z: https://en.wikipedia.org/wiki/Hyperplane_separation_theorem

[8] BITTLE, William. *Dyn4j – SAT (Separating axis theorem)*. Online. Dyn4j. 2010. Dostupné z: https://dyn4j.org/2010/01/sat/. [cit. 2024-03-17].

[9] *Ear clipping algorithm*. Online. In: Gamedev.net. 2014. Dostupné z: https://uploads.gamedev.net/monthly_06_2014/ccs-175326-0-96985400-1402010696.png. [cit. 2024-03-17].

[10] HECKER, Chris. *Physics, Part 3: Collision Response* [Online]. 1997, 18 [cit. 2024-03-19]. Dostupné z: http://www.chrishecker.com/images/e/e7/Gdmphys3.pdf

[11] Calculating the Moment Of Inertia for a concave 2D polygon relative to its origin. In: *Stack Overflow* [Online]. [cit. 2024-03-19]. Dostupné z: https://stackoverflow.com/questions/3329383/calculating-the-moment-of-inertia-for-a-concave-2d-polygon-relative-to-its-orgin

6. Přílohy

Obrázek 1 diagram struktury projektu 8
Obrázek 2 ilustrace různých polygonů (mnohoúhelníků) 12
Obrázek 3 ilustrace triangulace polygonu 12
Obrázek 4 diagram knihovny část 1 15
Obrázek 5 diagram knihovny část 2 16
Obrázek 6 kód - vytvoření světa 17
Obrázek 7 pseudo kód – hlavní smyčka simulace 18
Obrázek 8 pseudo kód SAT 1 19
Obrázek 9 pseudo kód SAT 2 20
Obrázek 10 ukázka algoritmu SAT 1 21
Obrázek 11 ukázka algoritmu SAT 2 22
Obrázek 12 ear clipping algoritmus [9] 23
Obrázek 13 pseudo kód pro ear clipping algoritmus 24
Obrázek 14 bod doteku kruhů 25
Obrázek 15 rovnice pro vzdálenost bodu od přímky 26
Obrázek 16 bod doteku kruh a polygon 26
Obrázek 17 body doteku polygony 27
Obrázek 18 kalkulace impulsů – příklad 28
Obrázek 19 rovnice pro výpočet impulsního skaláru pro změnu rychlosti 29
Obrázek 20 rovnice pro výpočet relativní rychlosti 29
Obrázek 21 rovnice pro výpočet impulsního skaláru pro změnu rychlosti a rotace 29
Obrázek 22 výpočet setrvačnosti pro kruh 29
Obrázek 23 kód pro výpočet setrvačnosti konkávního polygonu [11] 30
Obrázek 24 finální výpočet impulsu pro rychlost 30
Obrázek 25 finální výpočet impulsu pro rychlost rotace 30
Obrázek 26 kód použití knihovny 31
Obrázek 27 diagram desktopové aplikace 32
Obrázek 28 desktopová aplikace 33