Szakdolgozat feladat

Demeter Ábel Bence
Mérnökinformatikus hallgató részére

Fizikán alapuló ütközésszimuláció procedurálisan generált terepen

Mai modern számítógépes játékonak egyik fontos központi komponense a fizikai motor. Ennek a rendszernek a feladata, hogy szimulálja az objektumok mozgását, egymással való interakcióját, ideális esetben reprodukálva a valóságban tapasztalható eseményeket. Ennek érdekében a rendszer egyik alapvető feladata az objektumok közti ütközések detektálása. Itt fontos szempont a hatékonyság hiszen igényektől függően akár nagyon sok objektum kezelésére is képesnek kell lennie.

Hasonlóan fontos elem maga a terep ami a játékpálya jelentős részét alkotja. Ez gyakran procedurális módszerek segítségével készítik el már csak a méretéből adódóan (például nyílt világú játékok esetén). Ez utóbbi két elemet összekapcsolja a statikus terep és a dinamikusan mozgó objektumok közti interakció, ütközés detektálás. A szakdolgozat célja ezt a két elemet ötvöző megoldás elkészítése és megoldás nyújtása az ütközések hatékony kezelésére.

A hallgató feladatának a következőkre kell kiterjednie:

- Tekintse át és ismertesse a fizikai motorok ütközés detektáló komponensének működését!
- Mutassa be a procedurális terepgenerálás alapjait!
- Tervezzen meg és implementáljon egy fizikai motort és 3D megjelenítő rendszert!
- Egészítse ki a rendszert procedurálisan generált tereppel!
- Demonstrálja és tesztelje az objektumok tereppel való ütközését!
- Értékelje az elkészült rendszert teljesítmény szempontjából!

Tanszéki konzulens: Fridvalszky András Máté, PhD hallgató

Budapest, 2024. október 3.

/ Dr. Kiss Bálint /
egyetemi docens
tanszékvezető

Budapest Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Irányítástechnika és Informatika Tanszék

H-1117 Budapest, Magyar tudósok krt 2.
Telefon: 463-2699. Fax: 463-2204
www.iit.bme.hu

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
IIT Tanszék

Demeter Ábel Bence

FIZIKÁN ALAPULÓ ÜTKÖZÉSSZIMULÁCIÓ PROCEDURÁLISAN GENERÁLT TEREPEN

KONZULENS

Fridvalszky András Máté,

PhD hallgató

BUDAPEST, 2024

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Demeter Ábel Bence**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2024. 12. 08.

...…….

Demeter Ábel Bence

Összefoglaló

A szakdolgozat egy valós idejű szimulációs szoftvernek a megvalósítását hivatott véghezvinni. Ez a szimuláció konvex testeknek a fizikán alapuló mozgását és ütközését modellezi egy generált terepen.

A szimuláció indításakor a testeknek van egy kezdeti fizikai állapota, ami az időt előre szimulálva folyamatosan változik. A szimuláció során szerepet játszanak az ütközések, amik kis idő alatt nagy változást fejtenek ki. Továbbá testreszabható a terep geometriája, ami szintén a fizikiai állapot változását eredményezi.

A színtér egy felhasználó által, billentyűkkel mozgatható kamera segítségével megfigyelhető. A szimuláció pedig irányítható, az időt előre és hátra lehet szimulálni. Illetve változtathatóak az testek fizikai tulajdonságai, a változtatásokkal pedig a szimuláció újrajátszható.

Az említett feladatokat ellátó szoftver C++/OpenGL környezetben készült pár segédkönyvtár felhasználásával: GLEW a grafikus függvények linkeléséhez, GLFW az ablakozáshoz/eseménykezeléshez, GLM a matematikai adattípusokhoz/függvényekhez és végül a Dear ImGui könyvtár a grafikus kezelési felülethez.

Abstract

The thesis aims to implement a real-time simulation software. This simulation models the physics based movement and collision of convex bodies on a generated terrain.

In the beginning of the simulation, the bodies have an initial physical state, which is continuously changing as time is being simualted forward. During the simulation collisions occur, these collisions exert a greater change on the bodies' states. Moreover, the terrain geometry can be customized, this also has an effect in changing the states.

The user can navigate through the scene with a camera, which can be controlled with the keyboard. The simulation is controllable in the sense that time can be simulated forward and backward. Also, the physical property of the bodies can be customized, and the result can be viewed by replaying the simulation.

The software which handles the mentioned tasks is written in a C++/OpenGL environment using a few supporting libraries: GLEW for linking the graphical functions, GLFW for window/event handling, GLM for mathematical datatypes/functions and last but not least Dear ImGui for a graphical user interface.

1 Bevezetés

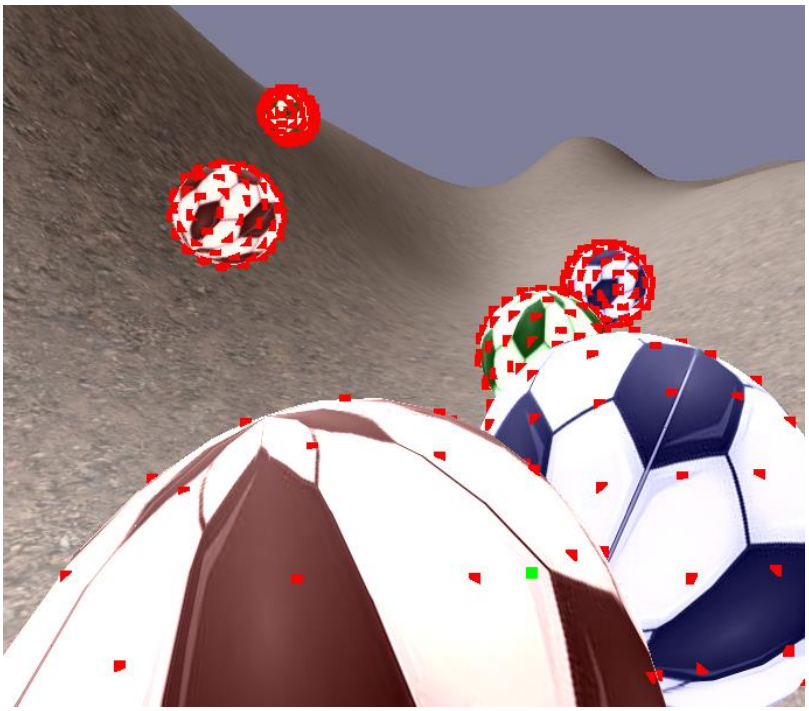
Az alkalmazás egyedi dimenziókkal és geometriával rendelkező objektumokat képes megjeleníteni, ahol az egyes objektumoknak különféle fizikai tulajdonságai vannak. A szimulációban az időtöl függő fizikán alapuló mozgás alapból be van fagyasztva, de a jobb és bal nyíl billentyűkkel előre illetve hátra szimulálható a mozgás.

A szimuláció kezdetében a testek rendelkeznek egy kezdeti lineáris és forgási sebességgel, majd az időt előre szimulálva gravitáció, súrlódás és ütközési impulzusok hatnak rájuk. Ennek következtében a nem végtelen tömegű testek esetén a lendületük és perdületük módosul.

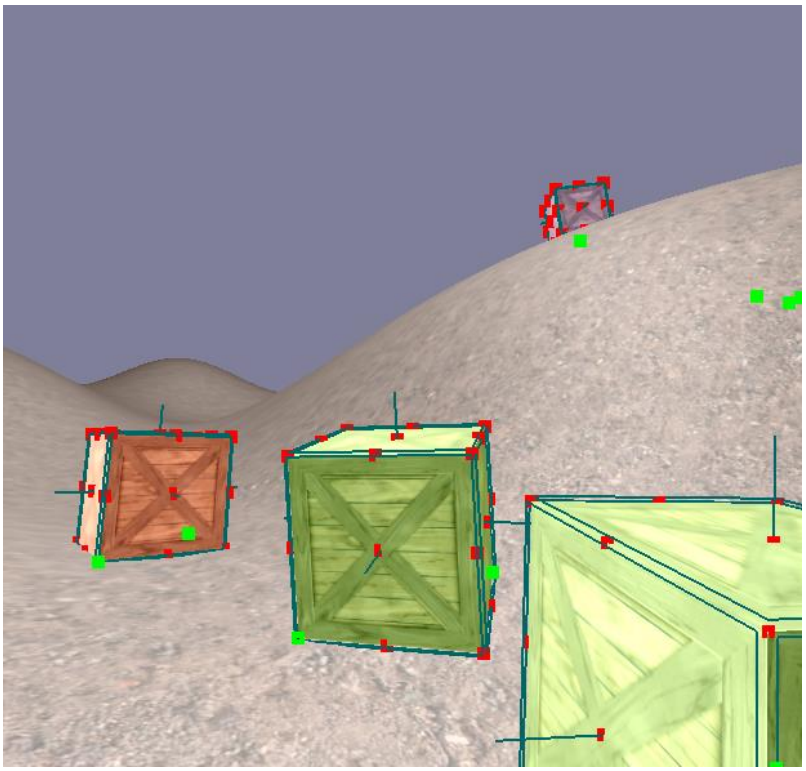
A terep Fourier sorral procedurálisan számított magasságértékekből tevődik össze. Az amplitúdó, frekvencia, fázis és a különféle frekvencián található szinuszgörbék összeadásához szükséges iterációk száma megadható, így valós időben változtatható a terep amin a testek ütköznek.

A kezelőfelületen a felhasználó beállíthat bizonyos paramétereket, amikkel a testek fizikai tulajdonságai és a szimuláció más paraméterei változtathatóak. Például a testek rugalmassága (mekkora ellenirányú impulzust kapnak), csúszóssága (mekkora mértékben hat a testekre súrlódás), az időlépcső (milyen gyors legyen a szimuláció) vagy a gravtitáció erőssége.

Összesen három típusú objektum kezelésére van felkészítve az alkalmazás, ebből két konvex geometriára, gömbökre és téglatestekre, illetve a nem konvex terepre. A terep a szimuláció során végig statikus marad, azaz nem mozdul el az interakciók hatására. A gömbök és téglatestek mozgása pedig különféle alapelrendezést követően szimulálható, illetve az alapelrendezésbe visszaállítható esetlegesen más fizikai paraméterekkel.



Ábra 1: Labdák egy generált terepen



Ábra 2: Dobozok egy másik generált terepen

1.1 Könyvtárak

A projekt felhasznál pár külső könyvtárat, amik segítenek a fejlesztés során a feladatok elvégzésében. A szükséges feladatok közé tartozik a megjelenítés, egy felhasználói felület létrehozása, beviteli eszközök kezelése, illetve matematikai függvények és adattípusok felhasználása különféle számításokhoz.

1.1.1 OpenGL és GLEW

Az OpenGL [6] (Open Graphics Library) egy platformfüggetlen, alacsony szintű API, amelyet 2D és 3D grafikai alkalmazások fejlesztésére használnak. A grafikus hardver közvetlen vezérlését teszi lehetővé.

A GLEW [7] (OpenGL Extension Wrangler Library) egy segédkönyvtár, amely megkönnyíti az OpenGL kiterjesztések kezelését és használatát. A GLEW automatikusan inicializálja az OpenGL funkciókat, így azokat egyszerűen lehet meghívni a programban.

1.1.2 Dear ImGui

A Dear ImGui [8] egy könnyen használható, bővíthető grafikus felhasználói felület (GUI) könyvtár, amely elsősorban fejlesztőeszközök és hibakereső felületek létrehozását szolgálja. Támogatja az OpenGL, Vulkan, DirectX és más megjelenítő motorokat, így szinte bármilyen platformon futhat. Könnyen integrálható és jó néhány már elkészített összetevőt tartalmaz, például gombokat, csúszkákat és grafikonokat.

1.1.3 GLFW

A GLFW [9] (Graphics Library Framework) egy nyílt forráskódú könyvtár, amelyet grafikus alkalmazások fejlesztésére használnak. Fő funkciói közé tartozik az ablakok és grafikus kontextusok létrehozása, valamint a beviteli eszközök (billentyűzet, egér) kezelése.

Az események részét képezi, a billentyűkkel történő, objektumok és a kamera térbeli mozgatása, illetve a szimuláció irányítása. Ezek az események kezeléséhez tartalmaz megfelelő függvényeket a GLFW könyvtár, ami egyúttal a szimulációs ablak létrehozásáért is felel.

1.1.4 GLM

A GLM [10] (OpenGL Mathematics) egy nyílt forráskódú C++ könyvtár. A könyvtár olyan matematikai típusokat és függvényeket biztosít, mint a vektorok, mátrixok, valamint ezekkel végzett műveletek. Az OpenGL Shading Language (GLSL) szintaxisát követi, így könnyen átvihető a kód a CPU és GPU műveletek között.

A GLM matematikai műveleteinek a saját implementációja egy időigényes feladat lenne. A könyvtár használatával számos rendkívül hasznos függvény állnak rendelkezésre, mint például a mátrix invertálás, projekciós mátrix számítás, vektorok skalárszorzata, vektorok keresztszorzata, vektorok normalizálása stb.

1.2 Könyvtárak használata

A megjelenítendő objektumok közé tartozik egy fényforrás, egy terep, gömbök és téglatestek. Az egyes objektumok pedig mind saját vizuális tulajdonságokkal rendelkeznek, ezek közé tartoznak a textúrák és a megvilágítási paraméterek. A megjelenítés ezek a tulajdonságok alapján történik az OpenGL grafikus keretrendszer felhasználásával. Az objektumok térbeli adatait és az előbb említett tulajdonságaikat a keretrendszer tárolóival fel lehet tölteni a GPU memóriájába, hogy az árnyalóprogramok elvégezzék a grafikus csővezeték feladatait.

A terep egy nagy objektum, optimalizálatlanul és magas részletességen sok csúcspontot tartalmaz, amit nem minden GPU képes elfogadható sebességgel feldolgozni. Ebben a problémában segít a tesszelációs csővezeték, amit megfelelő árnyalóprogramokkal, az OpenGl keretrendszeren keresztül lehet konfigurálni

A szimuláció elsődleges célja, hogy minél jobban legyenek a valós fizikai interakciók modellezve, amik előfordulhatnak a mozgásban levő testek esetén a generált terep fölött. Viszont a fizikai tulajdonságok, a terep paraméterei és a szimuláció konstansai sokféle értéket vehetnek fel. Ezeket az értékeket kell megfelelően kiválasztani, a valós idejű módosításuk pedig sok időt megspórolhat és segít a legéletszerűbb működést biztosító konfiguráció megtalálásában. Ebben a kalibrációs feladatban nyújt segítséget a Dear ImGui keretrendszer, lehetővé válik az említett értékeket futásidőben módosító felhasználói felület elemek létrehozása. Szintén ez a felület segítségével kerülnek beállításra a terep tulajdonságai (amplitúdó, frekvencia, fázis, iterációk) és annak optimalizálási paraméterei (hogyan történjen a terep tesszelációja).

2 Fizikai állapot követése

Mivel a testek fizikai állapota az idő múlásával folyamatosan változik, ezért szükséges egyrészt egy struktúra bevezetése amivel követhető az állapot, másrészt pedig ennek az állapotnak a változását is számon kell tartani. Az állapotvektorról található további leírás a [1] csatolmány 1. számú fejezetében.

2.1 Test lineáris mozgása

A mozgás két kategóriába sorolható. Az egyik a lineáris mozgás, ami közvetlenül a pozíció, sebesség és gyorsulás függvénye, amelyek egyenes vonalú mozgást írnak le.

Ezzel szemben a másik kategória a testek forgása, ami nemlineáris művelet. Például többek között a kommutativitás tulajdonsága nem teljesül, ha két külön tengely mentén kell forgás műveletet végrehajtani, akkor számít hogy először melyik tengely mentén történik meg (felcseréve más forgást írna le a két művelet).

2.1.1 Állapot tárolása

Egyelőre tekintsünk minden testet pontszerűnek, ekkor a térbeli mozgás szimulálásához elegendő tudni a test pozícióját és sebességét. Ekkor egy adott erő hatására egyszerűen meghatározható az új pozíció ha kiszámoljuk a gyorsulást, abból pedig a sebességet. A fenti tulajdonságok alapján a következő *S* állapotvektor és annak az idő szerinti deriváltja írható fel.

$$\frac{d}{dt}\mathrm{S}(t) = \frac{d}{dt}\big(\, x(t);\; v(t)\big) = \left(v(t); \frac{F(t)}{m} \right)$$

Ezt az összefüggést felhasználva, a térbeli elhelyezkedése egy pontszerű testnek követhető ha ismerjük a tömegét. Ha tudjuk az állapotvektor függvény értékeit minden időpontban, akkor az idő függvényében szimulálhatóvá válnak a testek.

2.1.2 Állapot változása

A szimuláció során a végleges eredmény amire szükségünk van, az egy állapotvektor függvény. De ennek a függvénynek csak az *S(0)* kezdeti értékeit tudjuk a szimuláció kezdetén.

A függvény kiszámolható az idő szerinti állapot derivált numerikus integrálásával. Felhasználjuk azt, hogy tudjuk mennyit fog változni a következő időpillanatban az állapotvektor. Minden diszkrét időlépcsőben a változást hozzáadjuk az előző állapothoz. Ezt az eljárást hívják Euler integrálásnak.

$$S(t + \Delta t) = S(t) + \frac{dS}{dt} \cdot \Delta t$$

Integrálásra léteznek pontosabb alternatív technikák, mint például a Verlet integrálás vagy a Runge-Kutta módszerek, de a szakdolgozat keretében az Euler integrálás kerül felhasználásra.

2.2 Test forgása

Ha a testek csupán csak pontok lennének akkor elgendő lenne minden iterációban tudnunk a sebességüket és a pozíciójukat, de mivel akár tetszőleges formájú konvex térbeli testeket szeretnénk szimulálni ezért ez már nem lesz elég. Így a térbeli állapotinformáció kiegészül még egy változóval ami leírja melyik irányba fordul egy test az adott pillanatban, ez lesz az orientáció. Az orientáció idő szerinti változása pedig hasonló a pozíció sebesség általi változásához.

2.2.1 Orientáció leírása

A test fordulását leíró orientációt egy 3x3 mátrix jellemzi (jellemezhető kvaternióval is aminek van előnye a mátrix reprezentációval szemben, ez a későbbiekben részletezésre kerül). Ezt az orientációt reprezentáló mátrixot rotációs mátrixnak hívják.

$$\mathrm{R(t)} = \begin{pmatrix} \mathrm{r_{xx}} & \mathrm{r_{yx}} & \mathrm{r_{zx}} \\ \mathrm{r_{xy}} & \mathrm{r_{yy}} & \mathrm{r_{zy}} \\ \mathrm{r_{xz}} & \mathrm{r_{yz}} & \mathrm{r_{zz}} \end{pmatrix}$$

Mivel nem pontszerűek a testek ezért lesz tömegközéppontjuk, aminek jelentős szerepe lesz a szimuláció során. Ha a testeket úgy hozzuk létre, hogy modellezési térben az origó a tömeg középpont, akkor ehhez egy bizonyos szemantika is társítható.

Az előbbi modellezési esetben az orientációt leíró mátrix első oszlopa azt az irányt adja meg amerre a test *x* tengelye mutat, hasonlóan a második oszlop az *y* tengely irányát, a harmadik oszlop pedig a *z* tengely irányát. Ilyenkor a pozícióhoz is társul többlet szemantika, minden test esetén az *x(t)* állapotváltozó egyúttal a tömeg középpont pozíciója.

2.2.2 Orientáció változása

A lineáris eset jó kiinduló analógiaként szolgál az orientáció idő szerinti változásának a jellemzéséhez, viszont ez nem lesz elég. A pozíció deriváltja egy vektor, de az orientációé nem lehet egy vektor, mert az egy mátrix.

A forgás jellemzéséhez egy olyan vektor definiálható aminek az iránya megadja a forgásnak a tengelyét, a nagysága pedig a forgás mértékét, ez a vektor a szögsebesség. Ezt a vektort felhasználva az orientáció idő szerinti deriváltja a következő. A szögsebesség pontos meghatározása majd később kerül részletezésre.

$$\frac{d}{dt}\mathrm{R}(t) = \omega(t) * \begin{pmatrix} r_{xx} & r_{yx} & r_{zx} \\ r_{xy} & r_{yy} & r_{zy} \\ r_{xz} & r_{yz} & r_{zz} \end{pmatrix}$$

Tehát az orientáció változását a szögsebesség és az aktuális orientáció csillag operátorral kapott erdeménye szolgáltatja. Itt a csillag operátor egy *a* sorvektor és egy *b* oszlopvetkor(ebben az esetben mátrix) keresztszorzatát jelöli a következőképpen.

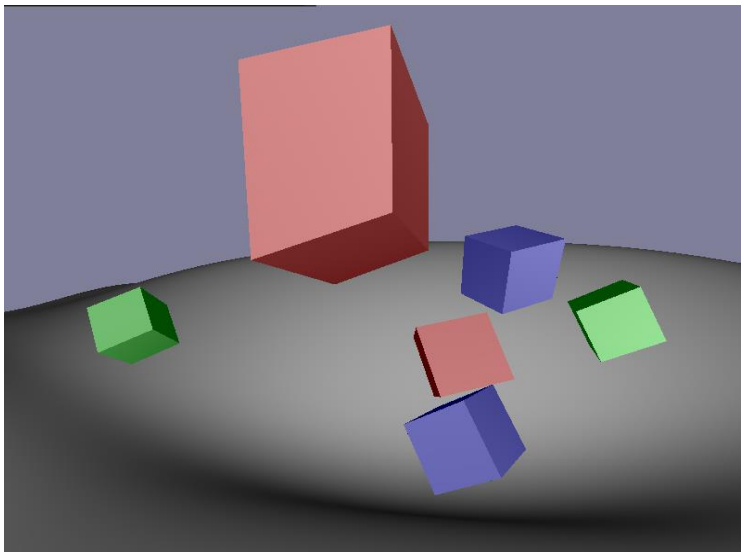
$$a^{*}\mathrm{b} = \begin{pmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{pmatrix} \begin{pmatrix} b_x \\ b_y \\ \mathrm{b_z} \end{pmatrix} = \begin{pmatrix} a_y b_z - b_y a_z \\ -a_x b_z + b_x a_z \\ a_x b_y - b_x a_y \end{pmatrix} = \mathrm{a} \times b.$$

Tehát a végeredmény az lesz, hogy a rotációs mátrix minden oszlopa felülíródik az oszlopok szögsebességel vett keresztszorzatával. Az előbbiekhez található levezetés a csatolt irodalmak közül az [1] tanulmányban, abban pedig a 2.3 számú fejezetnél.

2.2.3 Rotációs mátrix pontatlansága

Az orientációt reprezentáló mátrix másnéven egy rotációs mátrix, amit a számítógépes grafikában számos helyen használnak transzformálásokhoz. Ennek a mátrixnak a sajátossága, hogy minden oszlopvektor ami a mátrix oszlopaiból áll merőleges egymásra. Ha a mátrix változása során elveszti ezt a tulajdonságot akkor már nem térbeli rotációt fog leírni.

Az orientáció idő szerinti integrálásához sok mátrixösszeadás szükséges. Egy 3x3 mátrix pedig 9 darab lebegőpontos számot tartalmaz, viszont ez az adattípus tartalmaz pontatlanságot ami hibát eredményez. A hibák gyors forgások esetén összeadódnak és a mátrix elveszti a fent leírt tulajdonságát. Ez azt eredményezi, hogy a testek nemdeterminisztikus jelleggel elkezdenek nem kívánatosan szétnyúlni.



Ábra 3: Deformált piros doboz és lassabban forgó kisebbek

A fenti képen a pontatlanság következménye látszódik, minden kocka azonos dimenziókkal rendelkezett a szimuláció kezdete során, viszont az egyik piros kocka elkezdett szétnyúlni.

2.2.3.1 Ortonormalizálás

A rotációs mátrix tulajdonsága, aminek elvesztése okozza a nem kívánatos transzformációkat, visszaállítható abban az esetben ha a hiba túlságosan deformálja a mátrixot. Ezt a visszaállítási folyamatot nevezik ortonormalizációnak.

Az ortonormalizáció elvégzéséhez több módszer is létezik, az egyik legegyszerűbb eljárás ha egyenként normalizáljuk a rotációs bázisvektorokat és keresztszorzatokat számítunk. Elsősorban egységvektort képezünk a rotáció *x* tengelyének vektorából, majd a *z* tengely és az új *x* tengely vektorával kapott keresztszorzatot is egységvektorrá képezzük, végül pedig az újon számított vektorok keresztszorzatát szintén egységvektorrá alakítva egyenlővé tesszük az *y* tengely vektorával. Így a három új tengely vektorból alkotott mátrix rotációs tulajdonsága teljesülni fog.

Az előbbi eljárásnál létezik bonyolultabb, de pontosabb eljárás, ami a hibát egyenlő mértékben osztja szét a bázisvektorok között, ez a Gram-Schmidt ortonormalizációs eljárás. Az eljárás elvégzéséhez előbb szükséges definiálni a vektor projekció műveletét.

$$\mathrm{proj}_u(v) = \frac{u \cdot v}{u \cdot u}\mathrm{u}$$

A fenti művelet *u* és *v* vektorok esetén a *v* vektor *u* szerinti projekcióját eredményezi. Ezt felhasználva a következő a Gram-Schmidt ortonormalizációs eljárás *k* darab vektor esetén ha az első javított vektor megegyezik az eredetivel.

$$u_1 = v_1;\ u_k = v_k - \sum_{j=1}^{k-1} \mathrm{proj}_{u_j}(v_k)$$

Minden kapott javított vektort jelölő *u* vektor egységvektorrá képzése után, egymásra merőleges bázisvektorok fognak előállni amik a rotációs mátrix oszlopaiba helyezhetőek.

2.2.3.2 Kvaterniók

A rotációs problémában segítséget nyújthatnak a kvanterniók. Számos területen használják a kvaterniókat, többek közt a számítógépes grafikában is, mivel alkalmasak térbeli orientáció leírására. Mivel ezt az információt egy kvaternió 4 darab lebegőpontos számmal tárolja a 9 helyett, ezért sokkal kisebb hibával lehet integrálni az orientációt.

Egy kvaternió a komplex számnak négy dimenzióba kibővített formája, a komplex szám egy darab imaginárius komponense helyett három darab *i, j* és *k* imaginárius komponenst tartalmaz. A kvaternió esetén az *i, j* és *k* egységhosszú bázisvektorok, ezt a részt hívják a kvaternió vektor részének, a negyedik komponenst pedig a kvaternió skalár részének.

$$[s, v] = s + v_x i + v_y j + v_z k$$

Ezt a jelölést felhasználva, két kvaternió szorzata a következőképpen van definiálva.

$$[s_1, v_1][s_2, v_2] = [s_1 s_2 - v_1 \cdot v_2, s_1 v_2 + s_2 v_1 + v_1 \times v_2]$$

Ha azt szeretnénk, hogy tetszőleges forgatási információt tároljon egy *Q* kvaternió, akkor a skalár részébe a forgatási szög felének a koszinuszát kell helyezni, a vektor részébe pedig a forgatási szög felének a szinuszával beszorzott forgatási tengelyt.

$$Q = \left[\cos\left(\frac{\theta}{2}\right), \sin\left(\frac{\theta}{2}\right)\omega(t)\right]$$

A rotációs mátrixal történő reprezentációhoz hasonlóan, itt is szükség van az idő szerinti változás követésére. Az orientációs reprezentációt lecserélve és az alábbi összefüggést bevezetve a testek forgása szimulálhatóvá válik.

$$\frac{d}{dt}Q(t) = \frac{1}{2}[0, \omega(t)]Q(t)$$

A kvaterniókkal ki lehet fejezni egy tetszőleges térbeli *P* pontnak egy adott tengely körüli adott szöggel való forgatását is. A művelethez a kvaternió konjugáltját kell kiszámolni (a konjugált lényegében a vektor rész negálva és a skalár rész változatlanul hagyva), majd az alábbi szorzást elvégezni.

$$P_r = Q_c * [0, P] * Q$$

2.2.4 Tehetetlenségi tenzor

Eddig részletezésre kerütek olyan eszközök amikkel le lehet írni a testek aktuális orientációját és annak az idő szerinti változását a szögsebesség ismeretében. De pontosan hogyan fog a szögsebesség megfelelő értéket kapni? Elvégre nem mindegy, hogy egy testet melyik pontján éri forgatónyomaték, mert a dimenzióiból adódóan az más forgást eredményezne. Illetve az egyes testek esetén más-más geometriák szerint kell a forgást meghatározni.

A fenti probléma feloldására és a testek forgásának szabályozására érdemes bevezetni a tehetetlenségi tenzor fogalmát. Ez egy 3x3 mátrix ami leírja hogyan oszlik szét a tömeg a testen belül a tömeg középponthoz képest. A tenzor függ a testek orientációjától és tömegétől, de nem függ a térbeli eltolásuktól. Jelölje $r_i' = r_i(t) - x(t)$ a test felületén található pontnak a tömeg középpontból való eltolás vektorát. Ekkor a test tehetetlenségi tenzora a következő:

$$I(t) = \sum \begin{pmatrix} m_i(r_{iy}'^2 + r_{iz}'^2) & -m_i r_{ix}' r_{iy}' & -m_i r_{ix}' r_{iz}' \\ -m_i r_{ix}' r_{iy}' & m_i(r_{ix}'^2 + r_{iz}'^2) & -m_i r_{iy}' r_{iz}' \\ -m_i r_{ix}' r_{iz}' & -m_i r_{iy}' r_{iz}' & m_i(r_{ix}'^2 + r_{iy}'^2) \end{pmatrix}$$

Viszont erre az értékre minden iterációban szükség van, a fenti szumma kiszámítása pedig minden testre költséges volna iterációnként. Átrendezés után a fenti kifejezés a következő alakra hozható a rotációs mátrix felhasználásával (levezetés az [1] számú csatolt tanulmányban található a 2.10 számú fejezetnél).

$$I(t) = R(t) I_{body} R(t)^T$$

Az I_{body} egy adott geometriához tartozó konstans érték, ami a testek létrehozásakor kiszámítható. Így költséghatékonyabban meg lehet határozni a tehetetlenségi tenzor értékét, az I_{body} ismeretében a tenzor csupán két mátrixszorzással kiszámítható.

Az alábbi mátrixok felelnek meg a téglatest és a gömb I_{body} mátrixának, ahol *M* a testek össztömege. I_{cuboid} esetén a w_c, h_c, l_c értékek a szélesség, magasság és hosszúság dimenziók, I_{sphere} esetén pedig *r a* gömb sugara.

$$\mathrm{I_{cuboid}} = \frac{1}{12} M \begin{pmatrix} \mathrm{h_c^2 + l_c^2} & 0 & 0 \\ 0 & \mathrm{w_c^2 + l_c^2} & 0 \\ 0 & 0 & \mathrm{w_c^2 + h_c^2} \end{pmatrix} \qquad I_{sphere} = \frac{2}{5} M \begin{pmatrix} r^2 & 0 & 0 \\ 0 & r^2 & 0 \\ 0 & 0 & r^2 \end{pmatrix}$$

2.3 A végleges állapotvektor

Az eddigi állpotvektor nem tartalmaz minden, a testeket jellemző térbeli tulajdonságot. Ezt a végleges állapotvektort pedig már momentumok bevezetésével érdemes felírni.

2.3.1 Momentumok

A mozgás szimulálása során a testekre erők (gravitáció, impulzusok) hatnak. Az erők hatására pedig megváltozik az állapotvektor, ezeket a változásokat pedig célszerű momentumokkal leírni. Ugyanúgy lehetne használni gyorsulást, de momentumokkal a képletek valamelyest lerövidíthetőek. Tehát a legfontosabb állapotváltozó a momentum lesz, ezt integrálva megkaphatóak lesznek a további változók az állapotvektorban.

A momentumnak is van lineáris meg forgási változata, ezeket a továbbiakban a lendület és perdület kifejezések fogják jelenteni. Egy test lendületére az alábbi egyenletek vonatkoznak, jelölje *P* a lendületet.

$$\mathrm{P}(t) \;=\; M \cdot v(t) \qquad\qquad \frac{\mathrm{d}}{\mathrm{dt}} P(t) \;=\; F(t)$$

A perdület esetén szintén skálázódik a szögsebesség, de ebben az esetben nem egy tömeg skalárral, hanem egy mátrixal, ami a tehetetlenségi tenzor. Az idő szerinti derivált teljesen analóg, viszont itt az erő egy forgatónyomaték (az erővektor a geometria egy felületi pontjának relatív helyvektorával keresztszorozva kapható meg). Jelölje *L* a perdületet és τ a forgatónyomatékot.

$$L(t) \;=\; \mathrm{I(t)} \cdot \omega\mathrm{(t)} \qquad\qquad \frac{\mathrm{d}}{\mathrm{dt}} L(t) \;=\; \tau\mathrm{(t)}$$

Most már minden tulajdonsága ismert egy térben mozgó testnek, és az állapotának változása is követhető. A tehetetlenségi tenzor, a szögsebesség és a lineáris sebesség segédváltozóként szolgálnak az új állapot kiszámításában. A végleges állapotvektor pedig az alábbi módon írható fel.

$$\frac{d}{dt}\mathrm{S}(t) = \frac{d}{dt}\begin{pmatrix} x(t) \\ R(t) \\ P(t) \\ L(t) \end{pmatrix} = \begin{pmatrix} v(t) \\ \omega(t)^{*}R(t) \\ F(t) \\ \tau(t) \end{pmatrix}$$

2.3.2 Testek tömege

Eddig minden testnek fix tömege volt, ami egy skalár értékkel megadható volt és ezzel a mozgás egyenletei felírhatóak voltak. Problémát okoz az, hogy így nem lehet ábrázolni statikus testeket, mivel egy statikus test azt jelentené, hogy közel végtelen tömege van. Egy nagyon nagy értékkel történő közelítést elkerülve, ha a tömeg tárolása helyett az inverz tömeg lenne a testeket jellemző változó, akkor annak nulla értékre beállítása a végtelen tömegű testet jelentené(azaz a statikus test esete). Egy nulla inverz tömeggel rendelkező testre ugyanúgy kiértékelődik minden egyenlet de változásokat nem fognak eredményezni.

Lineáris esetre elég az inverz tömeg nullára állítása, de ez csak azt eredményezi, hogy a test nem tolódik el, forgást ugyanúgy fognak rajta az erők kifejteni. A tehetetlenségi tenzor nullmátrixra cserélése ezt az esetet is kiküszöböli, ehhez pedig az I_{body} értékét kell kinullázni.

Statikus test lesz a terep, ezt semmiképp nem lesz kívánatos mozgatni a szimuláció során.

3 Ütközési impulzusok

A testek mozgása ütközések nélkül az idő függvényében tekinthető folytonosnak, viszont ezt a folytonosságot megtörik az esetleges ütközések, illetve megkövetelik az állapotvektor azonnali módosítását. Az életszerű mozgás szimulálásának elérése mellett, a másik fontos szempont a testek egybemosódásának az elkerülése, ami elkerülhető egyes ütközési esetekben a momentum, impulzus alapú, azonnali módosításával. Azokban az esetekben, ahol az egybemosódás nem kerülhető el impulzussal, ott más módszereket kell alkalmazni.

Az új állapotvektor kiszámításához elengedhetetlen az ütközés pontos helyének és az ütközési felület normálisának az ismerete. A továbbiak azzal a feltételezéssel részletezik a működést, hogy ez az információ már rendelkezésre áll, ezeknek az adatoknak a meghatározását majd a következő fejezet tartalmazza.

3.1 Pont sebessége

A térben mozgó testek lényegében felületüknek bármely pontján ütközhetnek, a tömeg középpont sebességének ismerete viszont nem elegendő a probléma megoldásához. Például, nem mindegy, hogy egy baseball ütő a labdát a markolattól távol vagy közel találja el.

Tehát szükséges ismerni egy felületi pont hogyan mozog a térben, másképp mondva, szükséges kiszámolni a pontnak a sebességét. Erre pedig az alábbi összefüggés használható.

$$\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{p}(t_0) = \mathrm{v}(t_0) + \omega(t_0) \times \big(p(t_0) - x(t_0)\big)$$

Itt a pont sebességét a t_0 időpillanatban a $\frac{d}{dt}p(t_0)$ jelöli, ami megegyezik a pont tömeg középpontól való eltolásvektorának a szögsebességgel történö keresztszorzata és a test sebessége összegével.

3.2 Relatív sebesség

A pontbeli sebesség ismeretében, már meg lehet határozni a relatív sebességet egy *A* és *B* test ütközése esetén. Jelölje $\frac{d}{dt}p_A(t_0)$ és $\frac{d}{dt}p_B(t_0)$ a két sebességet, megjegyzendő, hogy a térben ugyanarról a pontról van szó mindkét esetben, arról a pontról ami az ütközés detektálása esetén meghatározásra került. De szükséges elkülöníteni, mert a két test esetén eltérő pontbeli sebesség fog keletkezni.

$$\frac{d}{dt}p_A(t_0) = v_A(t_0) + \omega_A(t_0) \times \left(p_A(t_0) - x_A(t_0)\right)$$

$$\frac{d}{dt}p_B(t_0) = v_B(t_0) + \omega_B(t_0) \times \left(p_B(t_0) - x_B(t_0)\right)$$

Felhasználva az előbbieket és az ütközési normál vektort, egy nagyon fontos mennyiséget lehet felírni, ami majd az ütközések osztályozásában fog segíteni. Ez a mennyiség a testek relatív sebessége, ami kiszámolható a két pontbeli sebesség különbsége és a normális skalárszorzataként az alábbi módon.

$$v_{rel} = \hat{n}(t_0) \cdot \left(\frac{d}{dt}p_A(t_0) - \frac{d}{dt}p_B(t_0)\right)$$

A v_{rel} azt mondja meg, hogy az együttes sebesség ellentétes, merőleges vagy azonos irányítottságú az ütközési normálissal.

Ha a v_{rel} pozitív, akkor az ütközést követően a testek távolodni fognak egymástól, tehát nem szükséges kezelni a helyzetet, mert nem fogja egybemosódás követni az ütközést.

Ha pontosan nulla az érték, abban az esetben lép fel a testek nyugalmi állapota, tehát ha a következő időpillanatban nem fog erőhatás kifejtődni a testekre (például gravitáció a talaj és egy arra eső doboz esetén), akkor nem fognak sem eltolódni, sem egymásba mosódni.

Ha negatív, akkor ellentétes irányú lesz az eredő sebesség a normálissal. A következő időpillanatban a testek egymás fele fognak mozogni, vagyis egybemosódás megy végbe. Ebben a helyzetben a helyes működés, ha mind a két test momentuma megváltozik (a lendület és perdület egyaránt), bizonyos fizikai tulajdonságok alapján. Ezt a működést pedig impulzusok bevezetésével lehet modellezni.

3.3 Az impulzus

Ütközések során a testek fizikai állapota kis idő lefolyása alatt nagy változáson megy keresztül, ennek a változásnak a leírásához pedig bevezetésre kerül egy új mennyiség.

3.3.1 Mennyiség definíciója

Ez a mennyiség lesz felelős ütközést követően a testek momentumának módosításáért. Az impulzus az erőhöz hasonlóan egy vektormennyiség, de a mértékegysége momentumban van megadva. Egy impulzus testre gyakorolt hatásának jellemzéséhez, egy nagy erő hatását kell elképzelni ami kis ideig tart. Jelölje *J* az impulzus mennyiségét.

$$\mathrm{F}\,\Delta\mathrm{t} = \mathrm{J}$$

Az impulzus testre gyakorolt hatása pontosan a test momentumának a változását jelenti az alábbi módon a lendületre.

$$\Delta\mathrm{P} = \mathrm{J}$$

Perdület esetén az impulzus hatásának leírásához szükség van a testnek egy *p* felületi pontjához, aminek ismeretében meg lehet határozni az impulzív forgatónyomatékot, ezt pedig τ_{imp} jelöli. Az ütközések kontextusában ez a pont az ütközés detektálás során kapott felületi pont.

$$\tau_{\mathrm{imp}} = \left(p - \mathrm{x(t)}\right) \times \mathrm{J}$$

$$\Delta\mathrm{L} = \tau_{imp}$$

Most hogy rendelkezésre áll egy mennyiség, amivel a momentum változtatása véghezvihető, már csak az maradt, hogy az impulzus iránya és mérete meghatározásra kerüljön. Az iránya az ütközési normál vektor lesz, a mérete pedig egyelőre egy *j* skalár amit a következő alfejezet részletez.

3.3.2 Ütközési impulzus mértéke

A mérték meghatározásához az alábbi összefüggést és jelölést fogjuk használni, jelölje v_{rel}^{-} az impulzus momentumhoz történő hozzáadása előtti relatív sebesség értékét és v_{rel}^{+} az értéket ami a hozzáadás után a testek relatív sebessége. Az alábbi empirikus összefüggés kell érvényesüljön az ütközés után.

$$v_{rel}^{-} = -\epsilon v_{\mathrm{rel}}^{+}$$

Itt az ϵ az ütközés rugalmasságának leírására alkalmas konstans, az értékét pedig nulla és egy között veszi fel. Az ütközés tökéletesen rugalmas, ha az ϵ egy, és tökéletesen rugalmatlan, ha nulla értéket vesz fel.

Ezt az összefüggést felhasználva fel lehet írni kényszereket amiknek teljesülnie kell az impulzusvektor nagyságának kiszámolásához, azaz a korábban említett *j* skalár meghatározásához. Tehát az új relatív sebességnek ellentétes előjelűnek kell lennie az ütközés előtti értékkel és a nagyságát az ϵ konstans szabályozza. A *j* értéke függ a testek pontbeli sebességétől, tömegétől, illetve azok tehetetlenségi tenzorától. A pontos levezetés a [2] számú csatolt tanulmány nyolcadik fejezeténél található, és az alábbi összefüggést tartalmazza.

$$\mathrm{j} = \frac{-(1+\epsilon)\mathrm{v}_{\mathrm{rel}}^{-}}{\frac{1}{\mathrm{M_a}} + \frac{1}{\mathrm{M_b}} + \hat{\mathrm{n}}(\mathrm{t_0}) \cdot \left(\mathrm{I_a^{-1}(t_0)}\left(\mathrm{r_a} \times \hat{\mathrm{n}}(\mathrm{t_0})\right)\right) \times \mathrm{r_a} + \hat{\mathrm{n}}(\mathrm{t_0}) \cdot \left(\mathrm{I_b^{-1}(t_0)}\left(\mathrm{r_b} \times \hat{\mathrm{n}}(\mathrm{t_0})\right)\right) \times \mathrm{r_b}}$$

Az *A* testhez tartozó mennyiségek alsó indexében az *a* jelölés, a *B* test esetén pedig a *b* jelölés van használva. Itt az r_a és r_b értékek a p ütközési pont és a tömeggközéppontok között képzett vektorok a következő módon $p - \mathrm{x_a(t_0)}$ és $p - x_b(t_0)$.

Most már ismert az impulzusvektor nagysága és iránya, a működés biztosítása végett az *A* testre módosítatlan előjellel kell alkalmazni, a *B* testre pedig ellentétes előjellel. Így megfelelő mértékben fog módosulni az új lendület és perdület az ütközést követőleg.

3.3.3 Súrlódás

A testek kontaktus esetén veszítenek sebességükből, ezt a sebességbeli változást pedig egy súrlódást reprezentáló pontbeli impulzussal lehet modellezni. Az ütközés pontjában, úgy ahogy az imént részletezett normálissal megegyező vagy ellentétes irányú impulzust kapnak a testek, itt is hasonlóan érdemes eljárni. De ebben az esetben az impulzus a tangenciális irányban érvényesül.

A tangenciális irány megkapható ha a két testhez tartozó pontbeli sebességek különbégéből egy relatív sebesség vektort képzünk, majd ebből a vektorból elhagyjuk az ütközési normálissal megegyező irányú kompnenseket. Ez eredményezi azt a vektort ami merőleges a normálisra, a végleges irány pedig ennek egységvektorrá képzésével kapható meg.

A súrlódási impulzus iránya tehát a tangens meghatározásával már megvan, mivel a súrlódás ellentétes irányban hat a test sebességére, ezért ezt még negálni szükséges. A mértéke pedig ismét egy új skalárérték kiszámolását követeli meg, legyen ez az érték j_t.

A j_t pedig meghatározható az előző fejezetben bemutatott skalárképlettel, de ebben az esetben nem a normál vektort, hanem a tangens vektort kell használni. Illetve ezt a skalárt szükséges szabályozni egy újabb γ konstans bevezetésével, ahol a γ szintén egy nulla és egy közötti tartományban mondja meg, hogy mennyire legyen csúszós a mozgás. Nulla érték esetén egyáltalán nem csúsznak a testek, tehát egy ragadós mozgás lesz megfigyelhető, egy érték esetén pedig tökéletesen fognak tovább csúszni a testek.

A maximális j_t érték nem haladhatja meg a normális irányába használt skalárét, ezért az alábbi kényszer lesz rá alkalmazva.

$$j_{t_{\max}} = (1 - \gamma) \cdot j$$

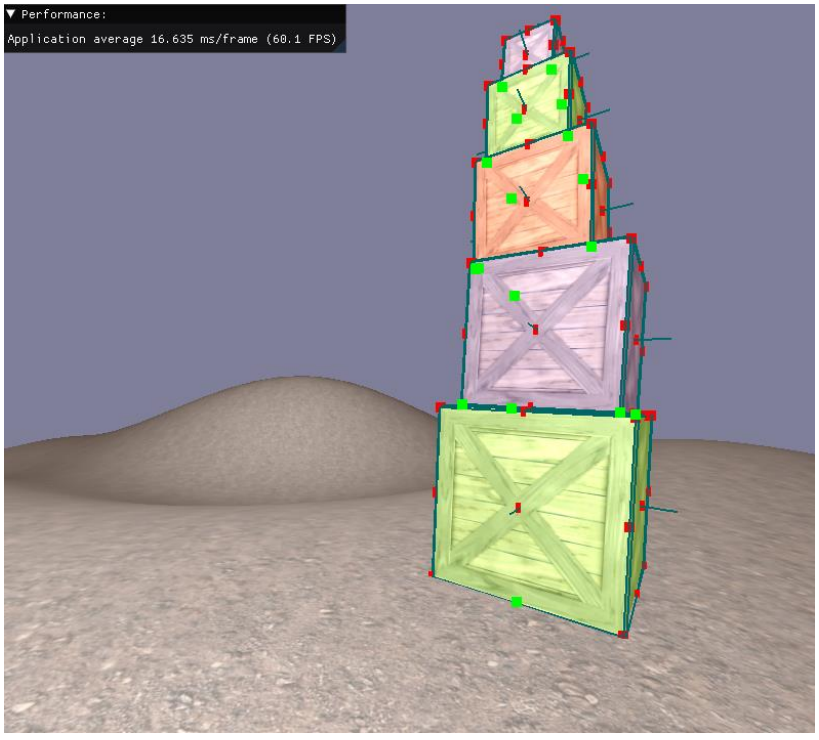
Továbbá, ha a kiszámolt j_t nagysága a maximális értéket mehaladná akkor az életszerűtlen mozgást eredményezne. Ezért be kell szorítani a $[-j_{t_{\max}}, +j_{t_{\max}}]$ tartományba. Ezt követően rendelkezésre áll a súrlódási impulzus iránya és nagysága, amit a testek momentumára alkalmazva a mozgás modellezhetővé válik.

3.4 Egybemosódás kezelése

Mivel a testek tipikusan állandó gravitációs nyomás alatt vannak, ezért az egybemosódást nem akadályozza meg az impulzusok alkalmazása. Ebben az esetben lép fel a nyugalmi állapot helyzete is.

A szimulációs motorok a nyugalmi állapotot minden értintekzési pont esetén kiszámolt ellenerőkkel garantálják, ehhez pedig meghatározzák a szükséges mennyiséget amivel minden érintkezési pontban a másik test válaszol a rá ható erőkre. Ez egy egyenletrendszernek a megoldását jelenti amiben bizonyos kényszerekkel szűkítik a megoldáshalmazt (ezt az eljárást a [2] számú csatolmány kilencedik fejezete részletezi).

A szakdolgozat keretében nem ez az eljárás kerül bemutatásra, minden ütközéspár esetén egy egybemosódás vektor nagyságával kerülnek a testek széttolásra. Ez az empirikus eljárás eredménye nem egyezik meg a tanulmányban bemutatottal, de ez is elfogadható életszerűséget biztosít.



Ábra 4: Dobozok oszlopa, kontakt pontok zöld színnel

3.4.1 A vibrációs probléma

A következő probléma merül fel a szakdolgozatban alkalmazott egybemosódás elkerülési eljárás esetén. Ha egy vektor formájában rendelkezésre áll az átlapolódás iránya és mértéke, akkor ezzel megegyező irányba történő azonnali eltolással két test esetén (például a talaj és egy rajta fekvő doboz) ez elkerülhető.

Csupán két test szimulációja esetén ez elegendő is volna, de ha képbe jön egy harmadik test, márpedig számos test mozgására kell kondicionálva legyen a projekt, akkor lép fel a vibrációs probléma. Ha egy testből eltolásra kerül egy másik akkor az lehet, hogy beletolódik egy harmadikba. A harmadik pedig ismét ki kell tolja magából és visszatolja az elsőbe. Iteratív eltolások esetén pedig ez azt eredményezi, hogy a középső test ide-oda fog mozogni, más kifejezéssel élve vibrálni. Ezt elkerülendő, a következő kényszer bevezetése szükséges.

Az egybemosódási kényszer a következő, ebben a kontextusban eltolni testeket csakis előre meghatározott bázisvektorokkal megegyező irányba lesz lehetséges. A bázisvektorok pedig lehetnek az *x, y* és *z* irányba mutató egységvektorok. Jelölje $\bar{e}$ a legnagyobb egybemosódást reprezentáló vektort, ez a vektor felbontásra kerül a bázisvektorok irányába az ortonormalizáció során bemutatott vektor projekciós összefüggéssel.

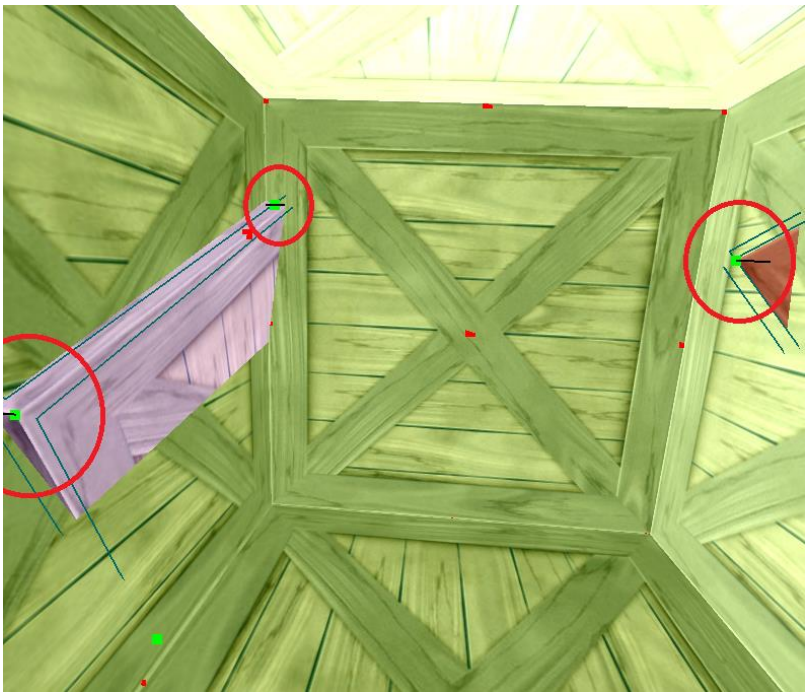
$$\bar{e_x} = \mathrm{proj}_{\bar{x}}(\bar{e})$$

$$\bar{e_y} = \mathrm{proj}_{\bar{y}}(\bar{e})$$

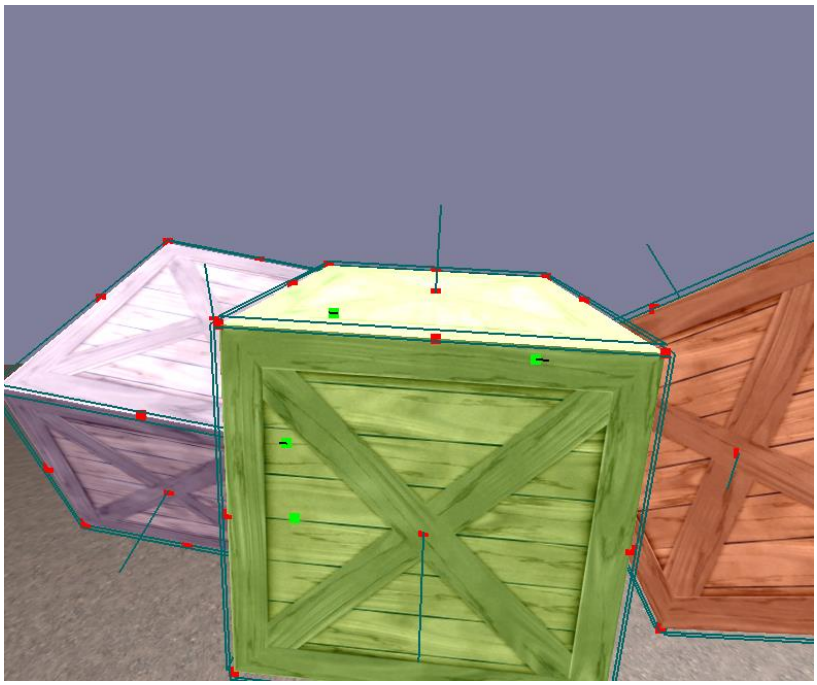
$$\bar{e_z} = \mathrm{proj}_{\bar{z}}(\bar{e})$$

Ha a fenti vektorok irányába fog az átlapolódott test eltolódni, akkor az ha beletolódik egy harmadik testbe, akkor majd a harmadik testet lesz szükséges eltolni ugyanígy felbontva annak egybemosódás vektorát. Ez azt is jelenti, hogy érintkezésben levő párok esetén el kell dönteni melyik kerüljön javításra.

Tehát a következő kényszer az, hogy mindig a bázisvektorok irányába távolabb elhelyezkedő test kerül eltolásra. Így ha a testeknek egy hosszú lánca is található a térben akkor iteratívan véghezvíve a vektorprojekciós műveleteket majd pedig az eltolásokat, akkor nem fog felmerülni a vibrációs probléma.



Ábra 5: Kameranézet a doboz belsejéből, a fekete vonalak az $\bar{e}$ vektorok



Ábra 6: Kameranézet a dobozon kívül, testek lánca

4 Ütközések detektálása

Az előző fejezetben használt ütközési adatoknak a meghatározásával fog ez a fejezet foglalkozni. Szükséges megalkotni egy olyan struktúrát, ami tetszőleges geometriák ütközésével kapcsolatos információt tartalmaz. Ezek a konvex geometriák vagy ütközőtestek pedig többféle reprezentációval vehetőek fel, akár valamilyen térbeli pontokat összefoglaló képlettel (például gömb esetén középpont és sugár által definiált térbeli pontok) illetve egy valamelyest magasabb kifejezőerőt biztosító egyedi háromszöghálóval.

A háromszögháló csúcspontokból, élekből és oldalakból tevődik össze. Ebben az esetben, feltételezve, hogy két háló érintkezése van vizsgálat alatt, meg lehet határozni egy generikus algoritmust ami a megfelelő információt fogja szolgáltatni az ütközési adatokról. Ez az eljárás nem lesz annyira gyors, mint a képlettel megadott ütközőtestek esetén, mivel itt meg kell vizsgálni minden élet minden éllel és minden csúcspontot minden oldallal. De ez alapján már felírható lesz az ütközési információt tartalmazó struktúra, ami a lényegi részt a gömb és téglatest ütközése esetén egyaránt szolgáltatja.

Az alábbi adatok kerülnek az ütközési struktúrába:

- Ütközési felület pontja
- Ütközési felület normál vektora
- Él-él ütközés esetén ez egyik él
- Él-él ütközés esetén a másik él
- Csúcspont-oldal ütközés történt vagy sem

Gömbök ütközése esetén nyilván nem érdemlegesek a csúcspontokkal, éleket és oldalakkal kapcsolatos adatok, de ezek csak szimplán nem kerülnek felhasználásra, csupán az ütközési pont és normális.

A téglatestek esetén nem a generikus háromszögháló ütközés detekció kerül felhasználásra, mert ha csak téglatestek vannak, akkor lehet egyszerűsíteni egyes számításokat, viszont nagyon hasonló így is az eljárás a generikus változathoz (él-él ütközésnél ugyanaz).

4.1 Téglatestek ütközése

Két téglatest, de akár két tetszőleges konvex háromszögháló értintkezéseit lefedik a csúcspont-oldal és él-él érintkezések, a továbbiakban ez a két eshetőség lesz kifejtve a téglatestek esetén.

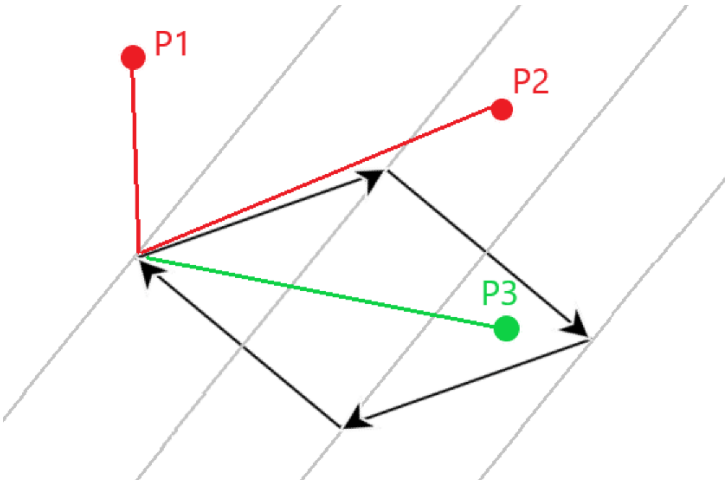
4.1.1 Csúcspont-oldal ütközés

Legyen a két vizsgálandó téglatest *A* és *B*. Így az *A* doboz oldalai és a *B* csúcspontjai között kell az érintkezéseket ellenőrizni. Ehhez pedig a vektorok skalárszorzatának tulajdonságai kerülnek felhasználásra.

A téglatestek oldalai négy csúcspontot tartalmaznak, az *A* test esetén ezek a csúcspontok alapján négy oldal vektor határozható meg, amik az óramutató járásával megegyező irányba, körbe mutatnak. Ezt követően, a *B* test csúcspontjain végig kell haladni és megnézni, hogy az ebbe a pontba állított vektorok skalárszorzata az oldal vektorokkal pozitív előjelű lesz vagy sem. Ha sehol nem negatív, akkor az azt jelenti, hogy az oldal által leírt térrészben van az adott csúcspont.

Már csak azt kell megvizsgálni, hogy a térrészen belül mekkora távolságra van a *B* csúcspontja az *A* oldalának síkjától. Ha ez a távolság egy bizonyos érték alatt van, akkor egy csúcspont-oldal ütközést találtunk. Ez a művelet elvégezhető az oldal normál vektorára projektálásával a csúcspontba állított vektornak, majd ennek a projektált vektornak a hossza lesz a síktól való távolság. Az ütközési normális lesz az oldal síkjának normál vektora, az ütközési pont pedig az adott csúcspont.

Ha az előző két lépés közül egyik sem teljesül, akkor tovább lépünk a következő oldalra vagy csúcspontra és a leírt lépéseket meg kell ismételni. Ez az eljárás egyszerűen kibővíthető tetszőleges konvex háromszöghálóval megadott ütközési geometriákra is.



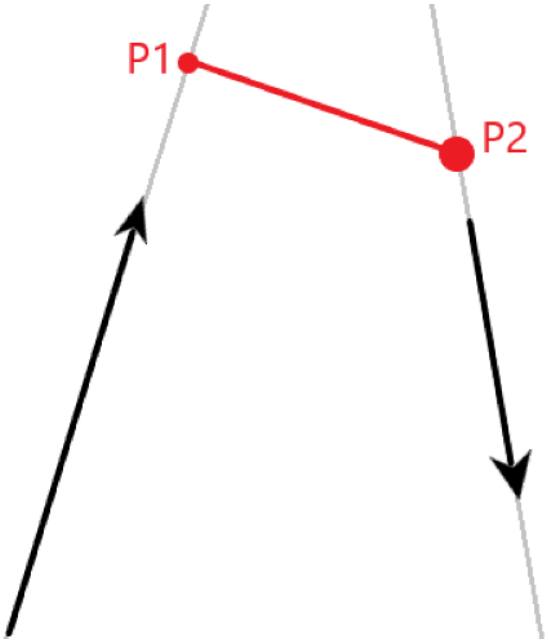
Ábra 7: P1 a térrészen kívül, P2 nem de nincs a síkban, P3 érintkezési pont

4.1.2 Él-él ütközés

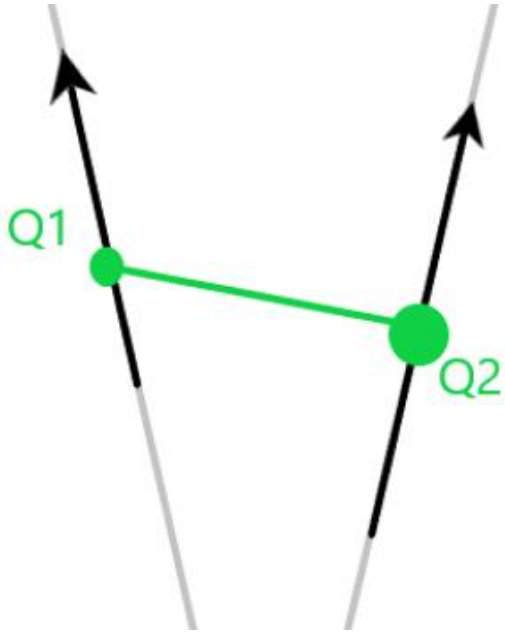
Ismét a két *A* és *B* téglatest van vizsgálat alatt. Minden élet az *A* testen összevetjük a *B* test minden élével. Két egyenes metszéspontja egy nagyon egyszerű képlettel meghatározható, de ez több okból is problémás, egyrészt azért, mert az élek szakaszok. Másrészt, mert a számítógépen a lebegőpontos pontatlanság miatt nem lehet tökéletes metszéspontot meghatározni, valós értékekkel dolgozó alakzatok esetén. A csúcspont-oldal ütközéséhez hasonlóan itt is érdemes távolságokat kiszámolni. Ha pedig a távolság egy adott érték alatt van, akkor él-él ütközés történt.

Felhasználva az egyenes egyenletét, két egyenes távolsága egy kétváltozós egyeneletrendszerrel meghatározható. Két pontot kell megtalálni, amiknek a különbségéből alkotott vektor merőleges mind a két egyenesre. A két változó ami kérdéses, hogy mennyit kell előre lépni az egyik és a másik egyenesen, hogy a két pontra teljesüljön a fenti tulajdonság.

A megfelelően kicsi távolság meghatározása után azt is meg kell nézni, hogy mind a két pont a szakaszokon belül található vagy sem, mert lehet, hogy az elfogadható távolság az egyenes szakasz részén kívül található. Ezt követően az ütközési pont a két szakaszpont átlaga, az ütközési normális a pontok különbsége.



Ábra 8: P1 és P2 vektora kicsi de nincs szakaszok közt



Ábra 9: Q1 és Q2 esetén érintkezés

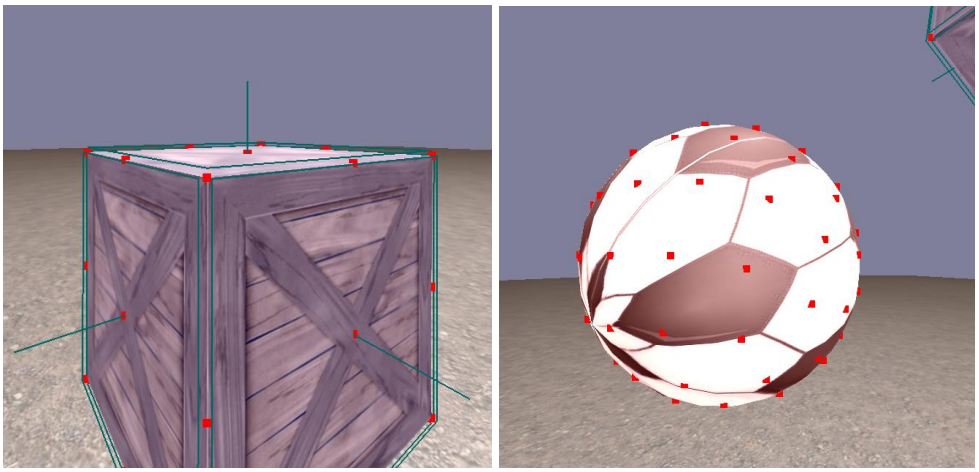
4.2 Gömbök ütközése

A következő ütközőgeometria a gömb, egy *A* és *B* gömb sugarának ismeretében az érintkezés egyszerűen meghatározható. Lényegesen költséghatékonyabb, mint a téglatestek ütközése esetén, mert itt nem kell végig iterálni minden élen, oldalon és csúcsponton. Ez okból a gömbök szolgálhatnak egy úgynevezett elsődleges ütközőként minden elképzelhető ütközési geometria esetén, még ha nem is közelítik megfelelően az objektumok megjelenített alakját (csupán egy előütközés vizsgálatára szolgálnak, mert ha befoglaló gömbök nem értintkeznek, akkor a tényleges ütközők sem fognak).

A és *B* esetén akkor lép fel ütközés, ha a két középpont közti vektor hossza kisebb, mint a sugarak összege. Az ütközési pont a vektor irányába történő sugárnyi ellépés, az ütközési normális pedig a vektor egységvektorrá alakított változata.

4.3 Ütközés a tereppel

A terep lényegében egy képlettel megadott magasságértékek sorozata, ha minden ütközési geometria felületén egyenletes eloszlással és kellő sűrűséggel ellenőrzési pontok találhatóak, akkor a pontok magassága alapján eldönthető, hogy az adott ütköző a terep alatt található vagy sem.



Ábra 10: Piros ellenőrzési pontok a dobozon és labdán

4.3.1 Transzformációs probléma

Ha a terepet leíró képlet (ami egy *x* és *z* koordináta ismeretében szolgáltatja a terepet egy *y* magassággal) argumentuma a pont *x* és *z* koordinátái, akkor ez vissza fog adni egy magasságot ami összevethető a pont magasságával. Ebben az esetben az ütközési pont az ellenőrzési pont lesz, az ütközési normális pedig a terep gradiense az adott pontban.

Problémát okoz, hogy a terep tipikusan át van méretezve vagy el van tolva transzformációs mátrixokkal, hogy életszerűen nagy legyen és a megfelelő pozícióban legyen található. Ezeket a transzformációkat írja le a rotációval együtt a számítógépes grafikában használt model mátrix, ami egy objektum minden térbeli transzformációját tartalmazza. Ennek a mátrixnak kell az inverzét felhasználni, hogy az ellenőrzési pontokat áttranszformáljuk a terep koordináta rendszerébe, itt pedig már könnyen elvégezhető a magasság alapú detekció.

5 Megjelenítés

Jelenleg részletezésre került minden szimulálandó objektum esetén hogyan szükséges eltárolni a fizikai állapotukat, illetve hogyan rendelhető hozzájuk ütköző geometria. Ezen a ponton a szimuláció jól lefutna, csak nincsen látható nyoma annak, mit produkálnak a háttérben folyó számítások. Tehát az utolsó lényeges lépés az objektumok megjelenítése, viszont érdemes megjegyezni, hogy a megjelenített objektum nem feltétlen egyezik meg az ütköző geometriával. Annak az elsődleges szerepe, hogy esetlegesen egy kevésbé részletes megközelítése legyen a látható objektumnak, így egy egyszerűbb modellen elvégezhető az ütközések kezelése.

5.1 Modellek megjelenítése és grafikus csővezeték

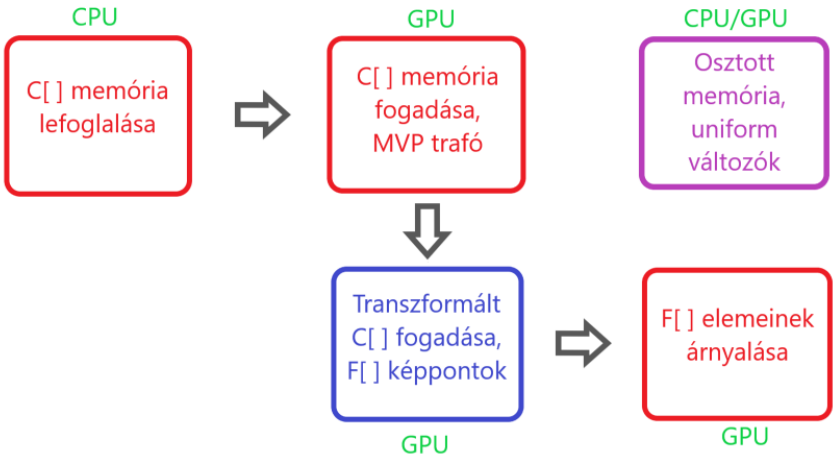
Sokféle objektum képzelhető el, de minden esetben egy háromszögháló csúcspontjai kerülnek feltöltésre a CPU memóriájából a GPU memóriájába. Pontosabban, a csúcspontokhoz tartozó attribútumok amik a legtöbb esetben pozíciók, normál vektorok és textúra koordináták. Ez a feltöltött memória az objektumok modell térben létrehozott formája, ami a szimuláció kezdetén kerül feltöltésre és ott is marad amíg az adott objektum meg nem szűnik. Ezt követően kezdetét vehetik a grafikus csővezeték feladatai az árnyalóprogramok segítségével. Az árnyalóprogramok osztott erőforrások az objektumok között, azaz egy programot több objektum is felhasználhat a kívánt megjelenítésre.

A grafikus csővezetékben a csúcspont árnyaló az első állomás, a GPU memóriába feltöltött csúcspontokat áttranszformálja a végleges világkoordinátás alakjukba (*M* mátrix), ezek a transzformációk pedig a fizikai állapot alapján számolódnak ki a szimuláció során. Ugyanitt történik a nézeti térbe történő transzformálás is (*V* mátrix), ez a kamera paramétereinek segítségével történik meg, így a kamera szemszögéből látható világ kerül előtérbe. Végül a csúcspont árnyalóban kerül végrehajtásra a projekciós transzformáció (*P* mátrix), ami levetíti a csúcspontokat a képernyő térbe.

A levetített csúcspontokat átveszi a raszterizációs hardver egység, ami a csúcspontok által alkotott háromszögeket kitölti fragmensekkel, amikkel a következő program már dolgozhat.

A fragmens árnyaló kifejti a hatását a képernyő minden elemére, az adott megjelenítési modell szerint kiszínezi azokat.

Az imént felsorolt grafikus csővezeték lépéseit az alábbi ábra szemlélteti, a hardver által végzett lépések kékkel vannak jelölve, a szoftveres feladatok pedig pirossal. A lilával jelölt dobozban található az osztott memória, amihez a CPU és GPU is egyaránt hozzáfér, például a kamera paraméterei, a csúcspont árnyaló transzformációi vagy a fényforrások adatai.



Ábra 11: Standard grafikus csővezeték

5.1.1 Csúcspontok attribútumai

Minden feldogozott csúcspontnak van egy sajátos szerkezete, az egyszerűbb árnyalóprogramoknak elegendő a csúcspontok pozícióinak számontartása (későbbi árnyalóprogram hibakereséshez), de ha összetettebb megvilágítási modellt szükséges eszközölni, akkor a csúcspontok kiegészülnek még az adott pozícióban találahtó normál vektorokkal illetve a textúra koordinátákkal.

A normál vektoroknak a fragmens árnyalóban van kiemelt jelentősége, az egyes árnyalási technikáknak szüksége van ezek ismeretére, hogy meghatározható legyen a megvilágított felület végleges színe. A normál vektorok modellezési térben vannak értelmezve, ezért ezeket is a csúcspont árnyalóban az éppen aktuális modellezési transzformációval el kell mozgatni. A textúra elemek kételemű vektorok, ezek segítségével adható meg, hogy az egyes textúrák melyik pixelét kell az adott fragmenshez rendelni. Ezek a további attribútumok a pozícióval együtt CPU számításokkal kerülnek meghatározásra.

5.1.2 Phong-Blinn árnyalási technika

Mint a legtöbb árnyalási technikának, ennek is szüksége van anyagtulajdonságokra az egyes megjelenítendő objektumok esetén, hogy a végleges radiancia meghatározható legyen. Ezek a tulajdonságok közé tartoznak az objektumok ambiens, diffúz és spekuláris paraméterei. A korábban említett osztott memóriába ezek az adatok feltölthetőek, így az árnyalóprogramok könnyen elérhetik ezeket.

Az ambiens tulajdonság nem függ a fénynek a beesési szögétől, minden fragmens ezt az alap értéket fogja felvenni, ezért tipikusan alacsony értéket érdemes használni, mert másképp életszerűtlenül intenzív lesz a radiancia.

Egy diffúz felület független a nézeti iránytól, de függ a fény beesési szögétől. A merőlegeshez minél közelebb van a beesési irány, annál intenzívebb a diffúz hatás, egy tisztán diffúz felület optikailag durva felület.

Spekuláris felületről beszélünk, ha a megvilágítás függ a nézeti iránytól is, illetve egy hatványtényezőtől ami a fényességet szabályozza.

$$\mathrm{L} = \mathrm{m_a} + \mathrm{m_d} \cdot \mathrm{l_n} + \mathrm{m_s} \cdot \mathrm{h_n}^{\gamma}$$

$$\mathrm{l_n} = max\left(0, \hat{l} \cdot \hat{n}\right)$$

$$\mathrm{h_n} = max\left(0, \hat{h} \cdot \hat{n}\right)$$

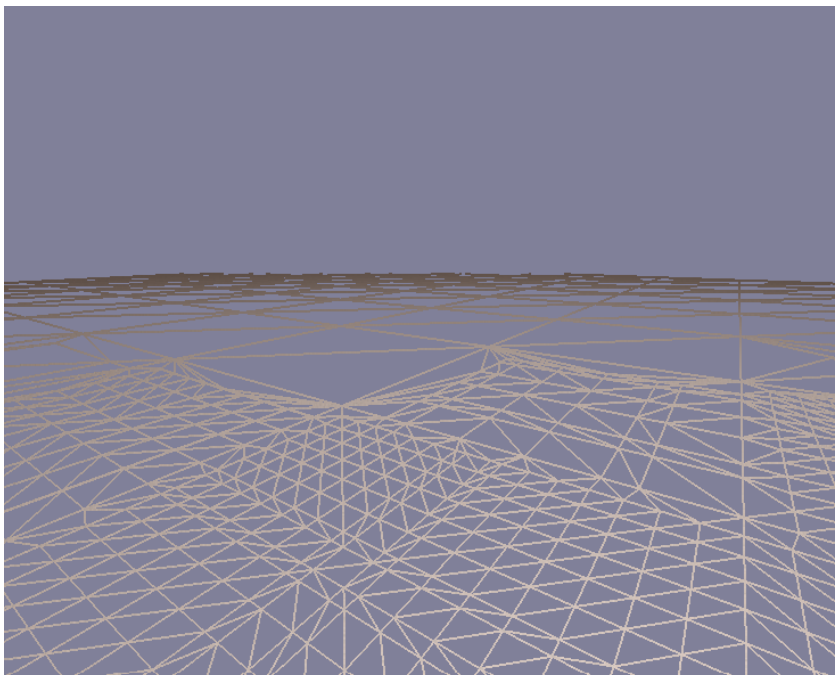
$$\hat{h} = \frac{\hat{l} + \hat{v}}{\left|\hat{l} + \hat{v}\right|}$$

A fenti L radiancia képlet szolgáltatja a Phong-Blinn árnyalást, ahol m_a, m_d és m_s az ambiens, diffúz és spekuláris anyagtulajdonságok. $\hat{v}$ és $\hat{l}$ a szem és a fényforrás irányába mutató vektorok, γ pedig a spekuláris csillogást szabályozó hatványkitevő.

5.2 Terep megjelenítése

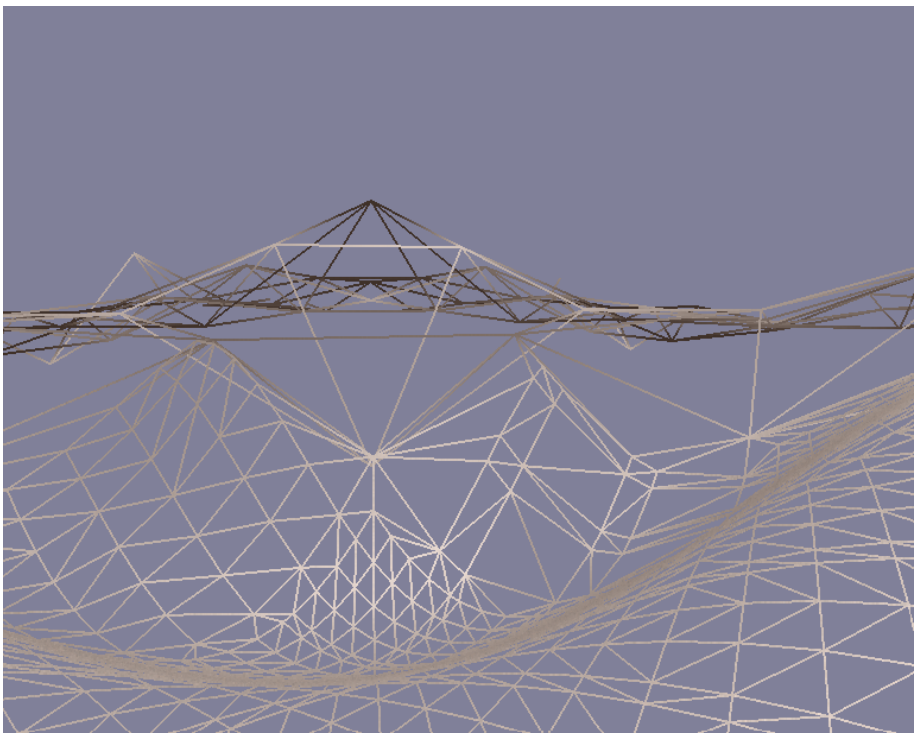
A terep egy nagy objektum, lényegesen több csúcspontból tevődik össze, mint a szimulálandó testek. De az OpenGL által támogatott tesszelációs csővezeték lehetőséget nyújt optimalizációra procedurálisan generált terepek esetén. Ez az optimalizációs technika megköveteli további árnyalóprogramok létrehozását és a grafikus csővezeték kiegészítését. Az optimalizáció azt használja ki, hogy a kamerától mért *z* mélység fordítottan arányos a részletességgel, a részletesség pedig a tesszelációval szabályozható.

A tesszelációs optimalizáció alapja a patch megjelenítési primitív, ez a vonalhoz és a háromszöghöz hasonlóan a grafikus megjelenítés során beállítható. Az egyes primitívek azt mondják meg hogyan legyenek a GPU memóriába feltöltött csúcspontok értelmezve, a patch esetén egy négyszöget kell elképzelni, azaz négy csúcspont van egy primitívként értelmezve. A szimuláció kezdetén létrehozásra kerül egy síkfelület ami NxN patch primitívet tartalmaz, majd ezek mélység alapján változó számú új, a tesszelációs hardver által létrehozott csúcsponttal fognak felosztásra kerülni a kamera aktuális elhelyezkedése szerint.



Ábra 12: A z mélységben közeli patchek részletesek, a távoliak nem

Na de miért egy sík a GPU memóriába feltöltött patch háló? Azért, mert nem lehet tudni, hogy mennyi új csúcspont lesz létrehozva, illetve, hogy azok hol lesznek véglegesen a modellezési térben. Tehát a dinamikus tesszeláció megköveteli, hogy az árnyalóprogramokba legyen a terep képletével megadva az újon létrehozott csúcspont magasságértéke. A patch csúcspontok attribútumaihoz csak a pozíció tartozik, a textúra koordináták és a normál vektor majd az árnyalóprogramokban kerül meghatározásra.

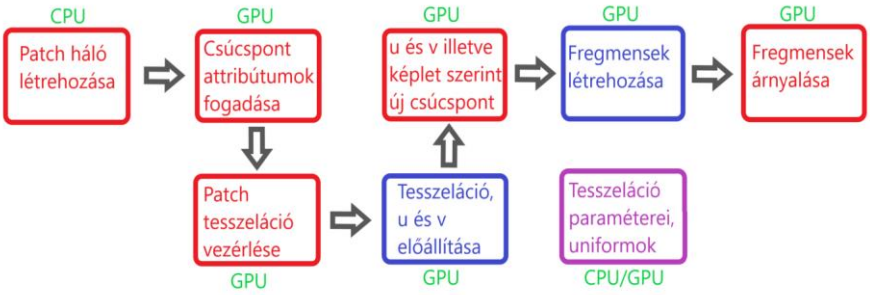


Ábra 13: A végleges magasságértékek nemnulla terep amplitúdóval

Az hogy mi számít pontosan elegendően mélynek, illetve, hogy mi számít alacsony és magas tesszelációnak, azt osztott uniform memóriába elhelyezett változókkal meg lehet mondani az árnyalóprogramoknak. Ezek a paraméterek pedig akár az alkalmazás kezelő felületéről is beállíthatóak, de mindenképp szükségesek a tesszeláció szabályozásához. A paraméterek közé tartozik a minimális/maximális mélysége, illetve a minimális/maximális tesszeláció.

5.2.1 A kibővitett grafikus csővezeték

Az imént részletezett optimalizációs technika két további árnyalóprogramot és egy hardver egységet használ fel a tesszeláció elvégzéséhez. A következő ábra szemlélteti az új csővezeték struktúráját. A hardver által végzett feladatok kékkel, míg a szoftveresek pirossal vannak jelölve. A tesszelációs csővezetékről részletes és gazdagon ilusztált cikk található az [5] számú csatolmányban



Ábra 14: Kibővített grafikus csővezeték

Az attribútumok fogadását elvégzi a korábban említett csúcspont árnyaló, de ezúttal semmi transzformációt nem végez. A következő a tesszeláció vezérlési árnyaló, ez az árnyaló felelős azért, hogy megadja a hardver egységnek az egyes belső vagy külső patch oldalakat hány részre ossza fel. A hardver pedig megad minden új csúcsponthoz egy *u* és *v* értéket, ami a patchen belüli koordinátái az adott csúcspontnak. A következő a tesszelációs kiértékelési árnyaló, ez képes kiolvasni az *u* és *v* patch koordinátákat, a terep képlete alapján a hozzuk tartozó magasságot meghatározza illetve minden csúcspontot levetít a képsíkra. Innen a további lépések megegyeznek a kibővitetlen grafikus csővezeték lépéseivel.

A következő kétváltozós Fourier sor képlet felelős az egyes csúcspontok *x* és *z* koordinátája alapján a terep magasságát visszaadni. Az *A* és *P* függvények adják vissza a terep amplitúdóját és fázisát, az *f* konstans a frekvencia. A_0 *m* és *n* függvényében egy pszeudorandom szám.

$$T(x,z) = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} A(m,n) \cdot \xi(x,z,m,n)$$

$$\xi(x,z,m,n) = cos(f(m \cdot x + n \cdot z) + P(m,n))$$

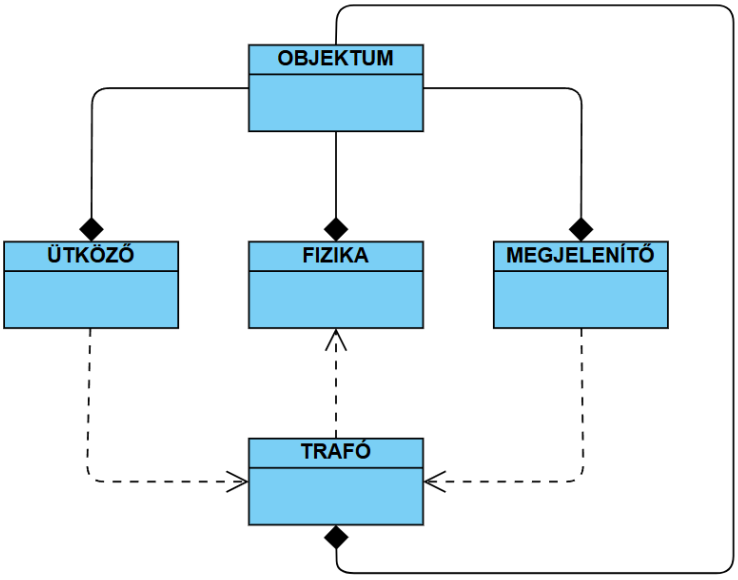
$$\mathrm{A(m,\ n)} = \frac{A_0(m,n)}{\sqrt{m^2+n^2}}$$

6 Projekt felépítése és implementáció

Programozási szempontból egy objektum osztály felelősségeit érdemes osztályokra bontani, ezzel betartva az OO programozásnak az elveit. Tehát az objektum osztályt négy másik osztállyal alkotott kompozíciós viszonya definiálja, ez azt jelenti, hogy nem fordulhat elő objektum a következők nélkül:

- Megjelenítésért felelős osztály
- Fizikai állapotért felelős osztály
- Ütközések kiszámolásáért felelős osztály
- Transzformációkért felelős osztály

Az objektum térbeli elhelyezkedését a transzformációkért felelős osztály tárolja, annak értékeit pedig a fizikai állapotért felelős osztály szolgáltatja. A transzformációkat megkapják a megjelenítési és az ütközési feladatokat ellátó osztályok, tehát az egyes kompozíciós elemeket ugyanaz a transzformáció köti össze.



Ábra 15: Objektum UML diagram

6.1 Implementáció

Az ütközésért és megjelenítésért felelős osztályok egyaránt absztraktak, tehát az objektumok csak a leszármazottaikat tudják adattagként példányosítani. Öröklés bevezetése szükséges, mivel számos megjelenítési forma, illetve számos ütköző test képzelhető el. Az eddigi implementáció pedig bővíthető további leszármazottakkal (például tetraéder geometria megjelenítésre meg ütközésre).

Az ütköző osztályok esetén kell gondoskodni arról, hogy minden leszármazott minden más leszármazottal egyedi módon ütközik. Ez azt vonja maga után, hogy szükség van minden ütköző pár esetén egy új függvény bevezetésére, viszont az ütköző osztály egyetlen interfészen keresztül kommunikál a külvilággal.

Továbbá, maga az objektum osztály is absztrakt, a leszármazottak egyedien határozzák meg az adattagjaikat a konstruktorukban. Például egy doboz objektum beállítja fizikai tulajdonságként a megfelelő I_{body} mátrixot és a nemnulla inverz tömeget, illetve létrehoz egy téglatest ütköző példányt és egy téglatestet megjelenítő példányt. A terep objektum pedig beállít egy nullmátrixot az I_{body} helyén és egy nulla inverz tömeget, illetve létrehoz egy terep ütköző példányt és egy terep megjelenítő példányt.

A fizikai állapot követését a fizika osztály kezeli, minden iterációban kiszámolja az állapotvektor változását, de az impulzusok és az ütközésdetektálás egy másik osztály feladata. Ez az osztály a statikus ütközéskezelő, végig iterál minden objektum páron és megvizsgálja ha ütköznek, az ütközés adatait pedig egy *std::tuple* segítségével elmenti. Az ütközéskezelő tartalmazza a szimulációs konstansok értékeit is, amik a felhasználói felületről változtathatóak (ϵ rugalmassági és γ csúszóssági konstansok), majd ezek és az ütközésadatok alapján elvégzi az impulzusok hozzáadását.

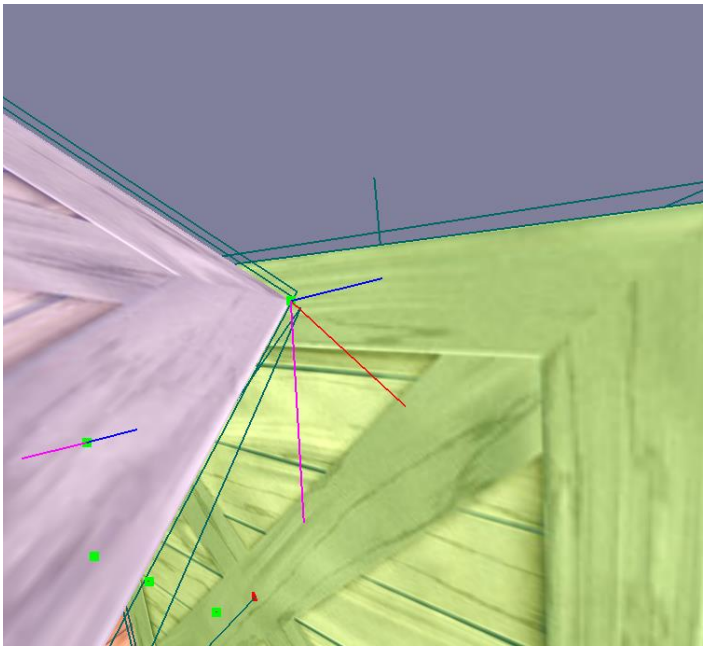
6.2 Hibakeresés

A fejlesztés során nem volt elegendő a szokásos felület a hibakereséshez, nagy segítség volt egy hibakereső osztály bevezetése, ami tetszőleges számú és színű vonalat vagy pontot tudott megjeleníteni. De mivel az OpenGL limitált számban kínál tárolókat memória feltöltésére, ezért minden rajzolás előtt közvetlen létrejön egy tároló, rajzolás után pedig felszabadulásra kerülnek. A gyakori létrehozás és felszabadítás teljesítmény szempontjából nem előnyös, de hibakeresési célokra megfelelő.

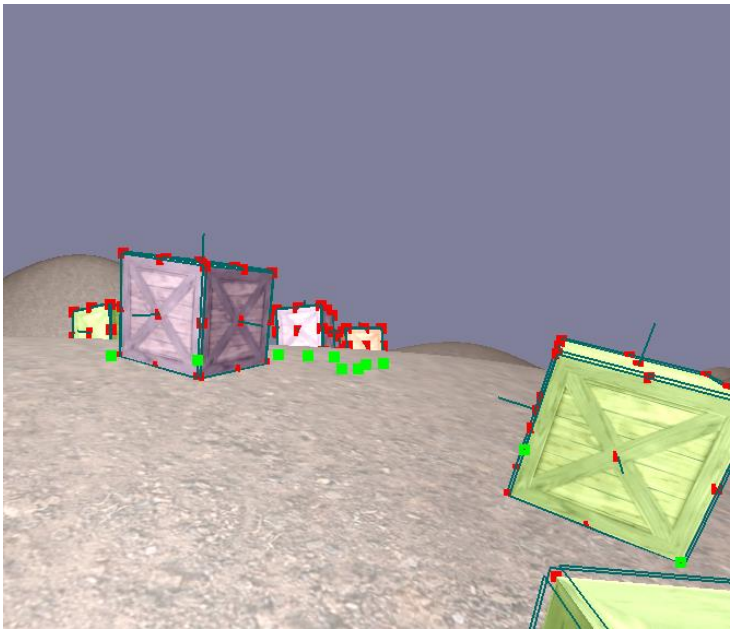
Az osztály statikus (egy példány van csak belőle) és állapotgépszerűen működik, csúcspontok feltöltése esetén az előzőek törlésre kerülnek, illetve minden rajzolás előtt be lehet állítani hogyan történjen a rajzolás:

- primitív beállítása (pont vagy szakasz)
- primitívek színe
- Logikai állapot, hogy a primitívek *z* mélységben minden más előtt legyenek

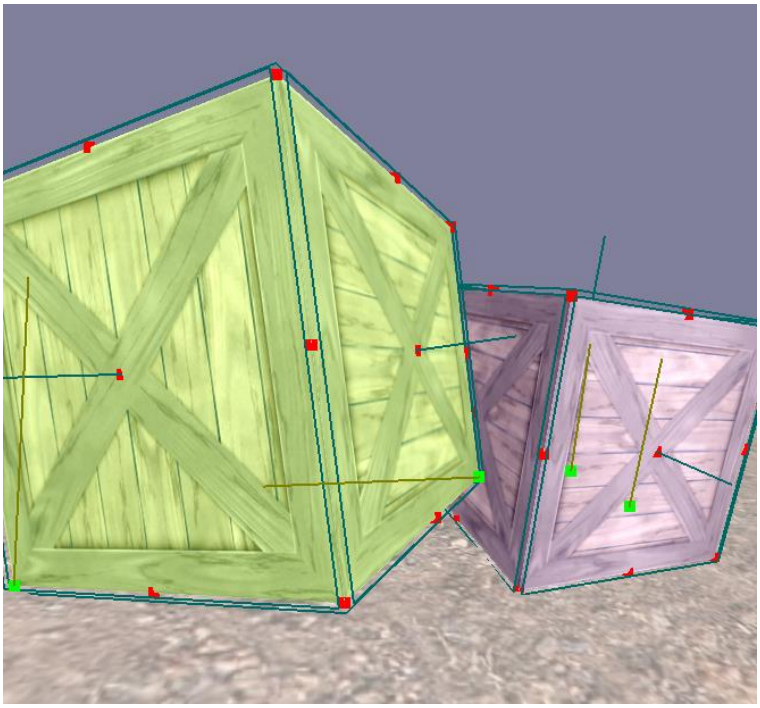
A megfelelő beállítások után az osztály használható ellenőrzési pontok, ütközési pontok, ütközési normál vektorok, sebesség vektorok stb. megjelenítésére.



Ábra 16: Pontbeli sebességek(kék és piros) és eredő sebesség(magenta)



Ábra 17: Ütközési pontok(zöld) és kontrol pontok(piros)

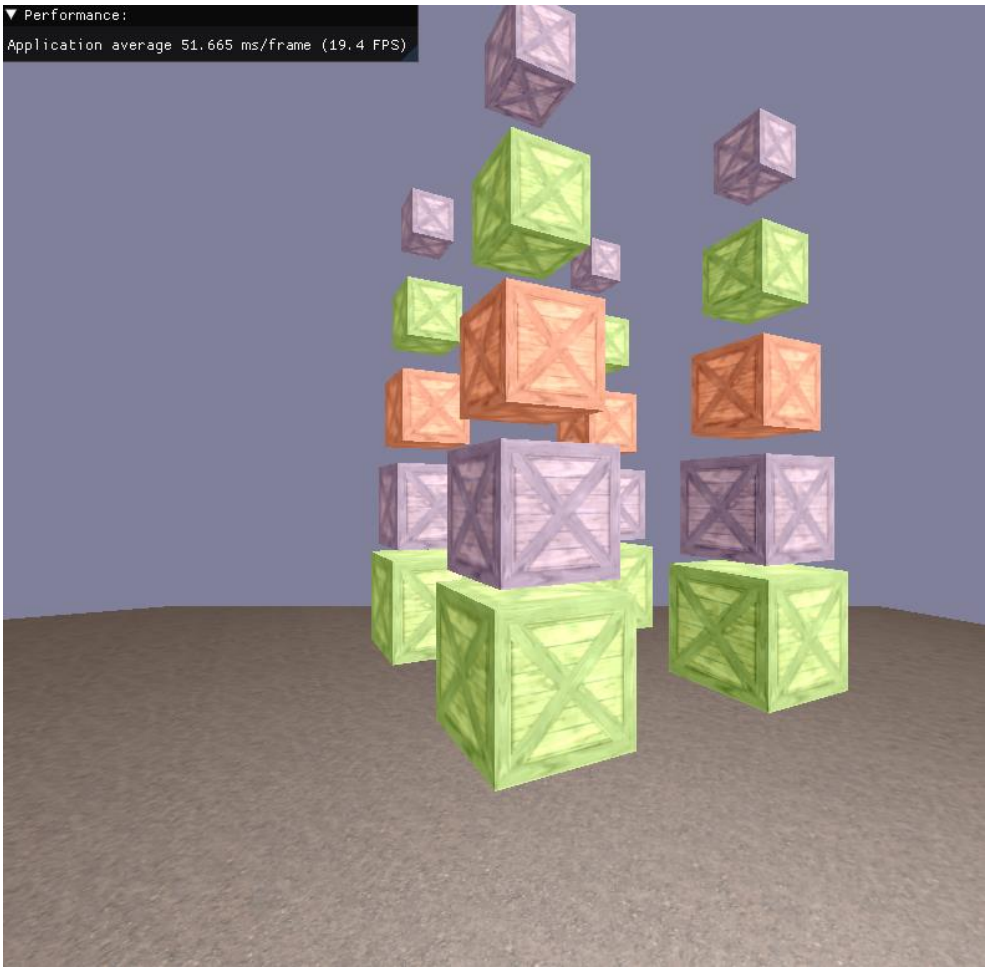


Ábra 18: Ütközési normál vektorok(zöld vonal) és ütközési pontok(zöld pontok)

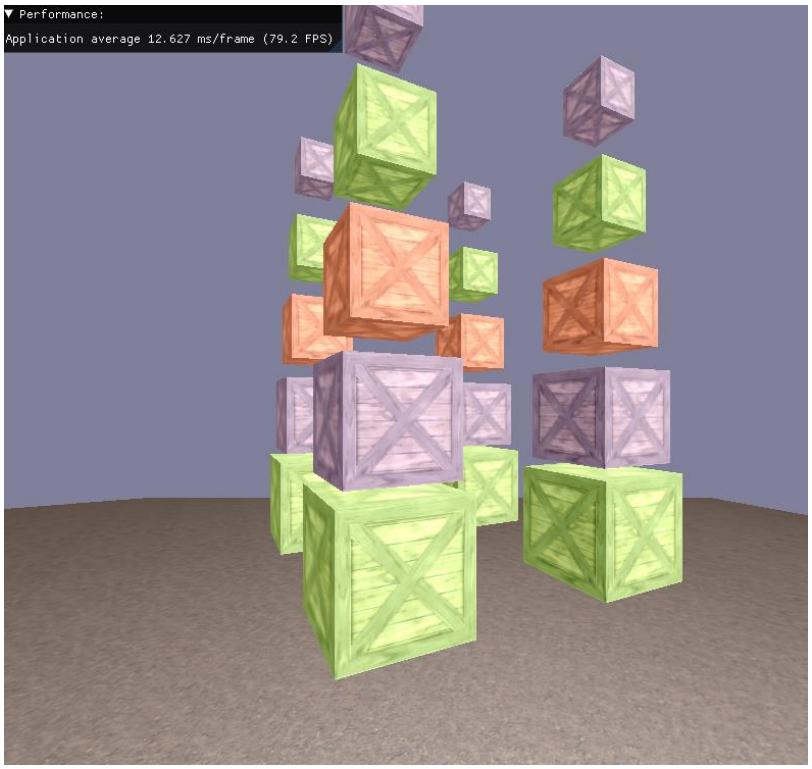
7 Teljesítménymérés

A korábban említett elsődleges ütközők használata téglatestek esetén segít a költséges csúcs-él-oldal iterációk elkerülésében. Ha a téglatesteket befoglaló gömbök nem érintkeznek (ennek ellenőrzése egy viszonylag olcsó eljárás), akkor a téglatestek sem fognak, ezzel pedig elkerülhető, hogy az ütközésdetektálás végigiteráljon minden csúcson, élen és oldalon.

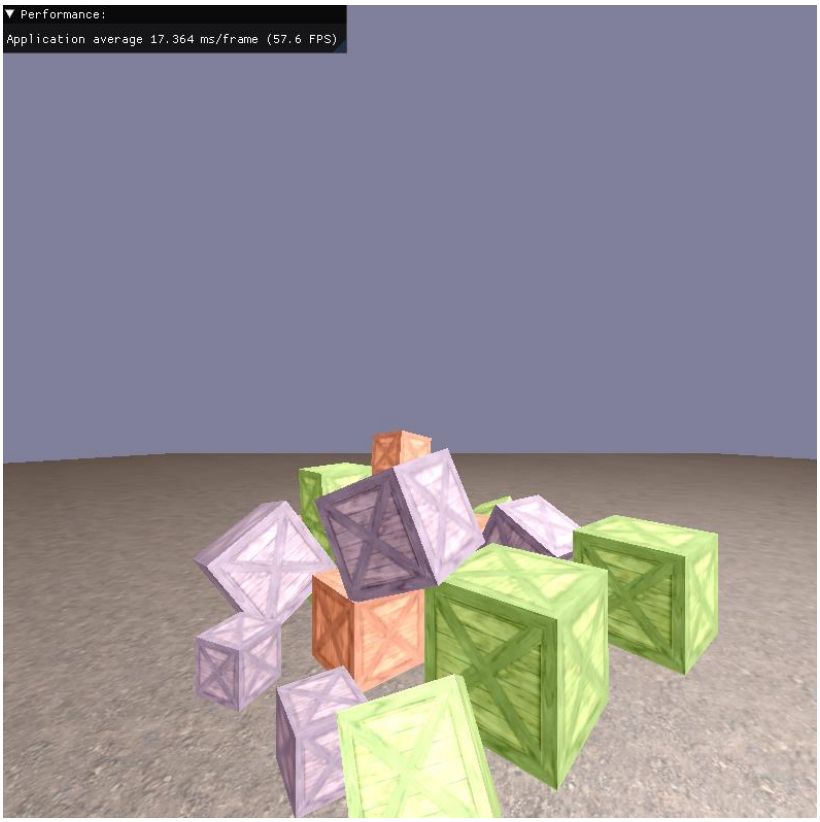
Az alábbi ábrákon látható, hogy optimalizálatlan formában a teljesítmény csupán ≈19.4 képkocka másodpercenként. Optimalizációval a szimuláció kezdetén, amikor még nem történik sok ütközés, akkor ≈79.2 képkocka. Végül amikor már egy ideje fut a szimuláció és sok ütközés számítás megy végbe, akkor ≈57.6 képkocka.



Ábra 19: Teljesítmény optimalizáció nélkül (20 doboz)



Ábra 20: Teljesítmény optimalizációval a szimuláció elején



Ábra 21: Teljesítmény optimalizációval és a sziumláció folytatásával

8 Továbbfejlesztési lehetőségek

Ebben a fejezetben olyan lehetőségek kerülenk bemutatásra amik a szimuláció életszerűségén kifejezőerején vagy teljesítményén segíthetnek. A szimuláció pontosságán pedig például a 2.1.2 fejezet során említett alternatív integrálási technikák vagy kvaterniók használata segíthet.

8.1 Kvadratikus programozás

A 3.4 fejezetnél említésre került egy ellenerőkön alapuló megközelítés az egybemosódás megakadályozására. Ez a megoldás életszerűbb testek közti reakciókat fog eredményezni, mivel itt a tényleges ellenerők kerülnek kiszámolásra amik a testekre hatnak, ellentétben az azonnali eltolások módszerével.

Az ellenerők meghatározásához N darab kontak pont esetén N darab egyenletnek a felírása szükséges. A megoldásvektor pedig N hossú lesz, aminek minden eleme az egyes erőknek a nagysága az ütközési normlisok mentén. Az egyenletekre bizonyos kényszereket kell alkalmazni, mint például az, hogy az erőknek taszító hatása kell legyen, vagy a szeparációs távolság negatív kell legyen (ebben az esetben lép fel egybemosódás).

Ennek az egyenletnek a megoldására, pontosabban az optimalizálására egy eszköz a kvadratikus programozás. A módszer során nem feltétlen a pontos megoldás kerül meghatározásra, előfordul, hogy az eredményt legjobban közelítő vektort szolgáltatja a kvadratikus programozás.

Az egyenletrendszer és a kényszerek pontos matematikai felírása a [2] számú csatolmány kilencedik fejezete tartalmazza, illetve további leírást tartalmaz a kvadratikus programozásról.

8.2 Generikus algoritmus háromszögháló ütközéshez

A korábban említett téglatest ütközések esetén használt detekciós eljárások túl specifikusak, mivel egy téglatest geometriáját veszik figyelembe. Ha az oldalak négyszög alapú megközelítése helyett háromszög alapú lenne, akkor generalizálhatóvá válik a detekciós probléma. Ezzel a megközelítéssel megváltoznának a csúcspont-oldal értintkezési esetek, de az él-él ütközések változatlanok maradnak.

A számítógépes grafikában az objektumokat tipikusan háromszöghálókkal modellezik, ezért háromszögek használata az ütközésdetektálás esetén nagyobb kifejezőerőt biztosítana az objektumok mozgásának szimulálása során.

Sajnos a háromszög alapú eljárás esetén is le kell ellenőrizni minden él-él és csúcspont-oldal ütközést ezért túlságosan részletes ütközőgeometriák esetén költségessé válik a generikus algoritmus. Egy optimalizációs lehetőség lehet, hogy csak azok a háromszögek kerülnek feldolgozásra, amelyek síkja az ütköző objektum felé mutat. További teljesítmény optimalizációra lehetőségeket a következő fejezet tartalmaz.

8.3 Optimalizáció térfelosztással

Az előütközők használata eredményezett teljesítmény növekedést, de mégtöbb test esetén ez nem lesz elegendő egy stabil teljesítmény megtartásához. Az eljárás sorá az objektumok kisebb csoportokba szerveződnek, így pedig csak a csoporton belülieket kell ütköztetni. A csoportok létrehozása pedig a tér felosztása alapján történik, ahol egy csoport a térnek egy bizonyos szegmense. Így egy adott objektum esetén nem szükséges egy másik csoport objektumait figyelembe venni ütközésdetektálás során, mivel azok máshol vannak a térben.

8.4 Optimalizáció GPU használatával

A grafikus kártya tulajdonsága, hogy nagyon sok párhuzamos számítás végezhető el rajta. Ezt a tulajdonságot kihasználva, ez egyes párhuzamosítható detekciós eljárásokat érdemes a GPU segítségével elvégezni. Illetve a tereppel történő ütközés esetén segíthet, ha egy képre raszterizájuk a terep alól a színteret és megvizsgáljuk mi került a terep alá.

9 Irodalomjegyzék

[1] David Baraff, Robotics Institute, Carnegie Mellon University, Unconstrained Rigid Body Dynamics
https://www.cs.cmu.edu/~baraff/sigcourse/notesd1.pdf

[2] David Baraff, Robotics Institute, Carnegie Mellon University, Nonpenetration Constraints
https://www.cs.cmu.edu/~baraff/sigcourse/notesd2.pdf

[3] Jan Bender, Kenny Erleben, Jeff Trinkle and Erwin Coumans, Interactive Simulation of Rigid Body Dynamics in Computer Graphics
https://animation.rwth-aachen.de/media/papers/2012-EG-STAR_Rigid_Body_Dynamics.pdf

[4] Johnathon Selstad, Orthonormalization
https://zalo.github.io/blog/polar-decomposition/

[5] Dr. Jeffrey Paone, Tesselation
https://learnopengl.com/Guest-Articles/2021/Tessellation/Tessellation

[6] OpenGL
https://www.opengl.org/

[7] GLEW
https://glew.sourceforge.net/

[8] Dear ImGui
https://github.com/ocornut/imgui

[9] GLFW
https://www.glfw.org/

[10] GLM
https://glm.g-truc.net/0.9.8/