Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Vlastnosti fyzikálního enginu Havok Physics

Plzeň, 2009 Pavel Karlík

Originální zadání práce

Prohlášení:

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, pokud není explicitně uvedeno jinak.

V Plzni dne ……

Pavel Karlík

……

Abstract

Characteristics of physical engine Havok Physics

Physical engines are important component in many programs as they want to simulate more and more often real physics - for example computer games, pharmacy, astronomy…

The deal of this bachelor thesis is to understand physical engine Havok Physics, make a set of demonstration scenes for beginners, a set of tests to discover how exactly can physical engines solve basic things from real physics, look for bugs and strange behavior and then (with experiences from previous tasks) find solution how to eliminate them during programming.

Obsah:

1. Úvod

S rostoucí komplexností a rozsahem herních titulů přišla postupně na řadu potřeba lehce implementovatelné abstrakce fyziky reálného světa. Samostatné vytváření vlastní herní fyziky se totiž časem stalo natolik náročným úkolem, že kdyby ji měl řešit každý projekt samostatně, tak vynaložené úsilí bude takřka vždy o stupeň horší (a časově náročnější), než řešení od specializovaného týmu, který se touto oblastí zabývá už několik let. A tak vznikly fyzikální enginy. Jejich využití je velmi rozmanité – od již zmíněných herních titulů, přes medicínu, strojírenství, až po astronomii.

Havok Physics je jedním z průkopníků a spolu s PhysX také leaderem v oboru fyzikálních enginů pracujících v reálném čase.

Úkolem mé bakalářské práce bylo seznámit se s fyzikálním enginem Havok Physics, dále s jeho pomocí vytvořit sadu ukázkových úloh spolu s dokumentací a jednoduchým „Hello World“ programem pro začátečníky.

Další částí mé práce bylo vytvořit ucelenou sadu testů k porovnávání fyzikálních enginů využitelnou do budoucna k přímému porovnání s konkurenty a poté těmito testy podrobit Havok Physics a najít jeho slabá místa.

Nakonec se pokusit tyto problémové části s nabitými zkušenostmi z předchozí práce vyřešit, aby se jich mohli ostatní vyvarovat a byli na ně při používání tohoto fyzikálního enginu připraveni.

2. Havok Physics

2.1 Obecné

Fyzikální engine Havok vznikl ve své prvotní verzi v roce 1998 v Irském Dublinu. Byl vyvinut dvěma studenty informatiky na Trinity College Hughem Reynoldsem a Stevenem Collinsem. Ve stejném roce založili společnost Telekinesys Research Ltd., která vlastnila na tento engine plná práva. Od této doby získal Havok několik ocenění:

- US National Academy of Television, Arts & Sciences Award, 2008 - (Technical Emmy)
- Best Choice of Computex - 2006 Winner
- FileFront - "Most Advanced Technology" of 2004
- Computer Graphics World - CGW 2003 Innovation Award
- Game Developer Frontline Award - 2002+2003 Best Game Component

Havok Physics prošel velkým vývojem - od prvotní verze až k aktuální verzi 6.5.0 (vydána 24.3.2009) a mohli jsme se s ním setkat v obrovském množství herních titulů, pro ilustraci např.: Assassin's Creed, Bioshock, Battlefield: Bad Company, Company of Heroes, Fallout3, Half-Life 2 (vzpomeňte na Gravity Gun), Halo 2+3, Test Drive Unlimited, Lost Planet, Oblivion, F.E.A.R.

Původní Havok se nám přetvořil v Havok Physics a stal se součástí
produktové nabídky Havok, která zahrnuje tyto součásti:

- Havok Behavior – systém pro událostmi řízené chování ve hře – AI
- Havok Physics – fyzikální engine her pro realtime detekce kolizí a simulace fyzikálních řešení
- Havok Animation – vývojové prostředí úzce propojené s Havok Physics pro jeho lepší integraci do různých herních enginů

- Havok Cloth – pohyb textilií, dále simulace pohybu objektů ve větru (stromy, tráva, vlajky…)
- Havok Destruction – výňatek z Havok Physics specializovaný na destrukci rigid bodies.
- Havok FX – aplikace Havok Physics na GPU grafických karet ATI, projekt byl okamžitě ukončen po odkoupení firmou Intel

V roce 2008 koupil veškerá práva na Havok engine Intel a okamžitě zastavil práce na nadějné implementaci Havok Physics na grafické karty ATI (AMD), šlo by o přímou konkurenci k PhysX a jejich Ageia PhysX kartám (I když momentálně prosvětlují informace, že nové AMD karty řady 4xxx mají v sobě zabudovanou podporu pro výpočty v Havok Physics). Zároveň se ale Intel zavázal, že poskytne Havok Physics k nekomerčním účelům zdarma, což se také po čtvrt roce stalo.
Intel dal k dispozici nejdříve Havok Physics a Havok Animation ve verzi 5.5, později uvolnil verzi 6.0, na které jsem pracoval do posledních momentů této bakalářské práce. V posledním měsíci jsem přešel na Havok 6.5 (vydání 24.3.2009), která je také nejaktuálnější dostupnou verzí na stránkách havok.com.

Havok Physics je multiplatformní – je možné ho implementovat jak na PC,
tak i na PlayStation 1,2,3, Xbox360, Nintendo Wii až po PSP.

2.2 Schopnosti

Havok Physics je technologie pro rychlé reálné simulace takzvaných rigid body (český fyzikální ekvivalent je tuhé těleso). Používá se převážně v aplikacích, kde je potřeba aby objekty v 3D prostoru na sebe interaktivně reagovaly. Byl již využit ve více než 150ti herních titulech napříč mnoha platformami a napříč mnoha typy her např. adventury, first a third person střílečky, sport a závody aut.
S Havok Physics SDK můžete vytvářet virtuální fyzický 3D svět, vytvářet v něm fyzikální objekty, přiřazovat jim parametry z reálného světa a měnit je během simulace.

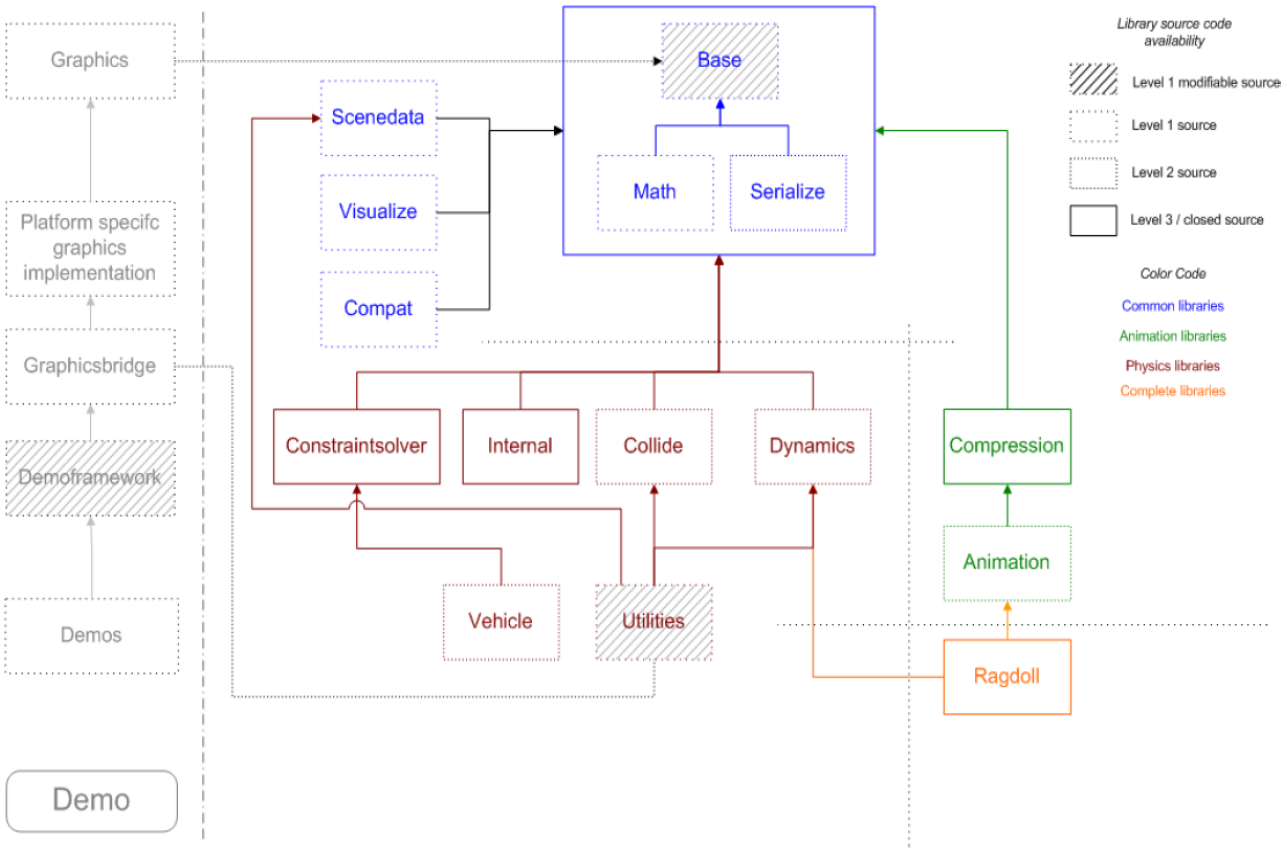
Můžete například měnit gravitaci, její směr, aplikovat síly a momenty sil na objekty, dynamicky přidávat a odebírat jednotlivé objekty. Nastavovat pružnost, hmotnost, tření, tlumení materiálů…

Havok Physics dále zajišťuje vysokoúrovňové řešení pro různé typické herní problémy jako simulace automobilů, bezvládných lidských těl (takzvaných ragdolls) - pro simulace zabitých nepřátel nebo např. crash testů, interakci klíčových částí objektu s herním prostředím (např. simulace nepřátel zasáhnutých do rozdílných částí jejich těla), dále obsahuje plnou kontrolu postavy - pro first a third person střílečky. Navíc má velmi bohaté API obsahující nástroje pro vytváření a ladění aplikací (např. Visual Debugger), disponuje možností integrace s third-party aplikacemi jako např. 3D Studio Max nebo Maya.

Havok ale určitě není:

- **Balíček pro vytváření hotových herních řešení** – Havok SDK je pouze a jen SDK. Zajišťuje pouze část konečného řešení.
- **Jednoduchá technologie** – potřebuje velkou investici na čas na plné proniknutí do všech možností, které využívá, a jejich efektivnímu využití.
- **Black box** – černá skříňka - Havok obsahuje spoustu nízkoúrovňových operací, které vám umožňují přistupovat přímo k jeho jádru, takže si ho můžete modifikovat přesně tak, jak potřebuje vaše konkrétní aplikace.

2.3 Architektura



Obr. 2.3 – 1 Struktura Havok Physics a Animation. Zdroj[1]

Knihovny:

- hkbase – obsahuje platformní a systémově nezávislou funkcionalitu pro ostatní knihovny, je vždy vyžadována! Samozřejmě je možné části této knihovny přepracovat tak, aby seděly přesně na použití vaší aplikace

- hkmath – matematická knihovna jádra, vždy vyžadována

- hkcompact – knihovna pro kompatibilitu verzí. Umožňuje načítat dohromady prostředky i ze starších verzí Havok Physics

- hkdynamics – zajišťuje propojení knihovny závěsů a os s knihovnou na detekci kolizí

- hkcollide – tento modul obsahuje všechnu detekci kolizí a jejich funkcionalitu používanou v Havoku

- hkserialize – knihovna pro export/import dat. Můžete pomocí ní exportovat nebo importovat svoje vlastní data ve svém vlastním formátu.

- hkvehicle – abstraktní vrstva a základní implementace vozidla pro použití Havok Vehicle Kitu

- hkutilities – obsahuje množství užitečných utilit usnadňující vývoj včetně příkladů

- hkcompresion – utilita pro kompresi dat

- hkanimation – centrální knihovna pro animace, včetně jejich ukládání a playbacku

- hkconstraintsolver – knihovna pro řešení os, jejich omezených pohybů, závěsů, propojených objektů, vždy vyžadována.

- hkinternal – rozhraní k interním třídám a strukturám Havoku

- hkscenedata – obsahuje data a jejich popis, který je extrahován z animation toolchain

- hkragdoll – obsahuje třídy k nastavení a manipulaci s bezvládnými těly

Knihovny technického dema:

- hkgraphics - obsahuje multiplatformní grafické jádro
- hkgraphicsbridge – spojení mezi fyzikou a grafikou
- hkgraphicsogl – OpenGL implementace hkgraphics
- hkgraphicsdx8 – DirectX 8 implementace hkgraphics
- hkgraphicsdx9 – DirectX 9 implementace hkgraphics
- hkgraphicsdx9s – DirectX 9s implementace hkgraphics
- hkdemoframework – multiplatformní framework dema

2.4 Wrappery

Havok jako takový je napsán v jazyce C++. V současnosti je možné ho používat pouze ve spojení C++ a Visual Studio 2005 popřípadě nově s verzí 2008 (Havok 6.5). Existují zatím individuální pokusy fanoušků o základní wrapper Havok na C#, ale bohužel jsou ve stádiu pokusů a jejich použití je zatím spíše ve fázi fikce.

2.5 Instalace SDK

Nejprve je potřeba se zaregistrovat na stránkách http://www.havok.com, poté je umožněn přístup ke stažení SDK balíčku se zdrojovými kódy, knihovnami, examply,

technickým demem, visual debuggrem a dokumentací. Stačí pouze rozbalit do libovolného adresáře na disku.
Poté je nutné nastavit v projektu cesty u linkeru (additional library a vybrat pro debug verzi knihovnu v adresáři **Havok/Lib/debug…,** stejně tak pro release vybrat odpovídající adresář.
Dále je potřeba v nastavení projektu v položce C/C++ General přidat adresář ze zdrojovými kódy **Havok/src**.
Pokud používáme technické demo, ve kterém je použita grafika, je také nutné dobře nastavit cesty k **DirectX SDK.**

Dále jsou zvlášť k dispozici HavokContentTools, které jsou určeny spíše pro práci s 3ds Max, Maya, XSI apod. Instalace je bezproblémová, jenom je potřeba poté ve Visual Studiu případně nastavit v linkeru projektu správnou cestu ke knihovnám, dále je potřeba v případě technického dema nastavit správně také cestu k DirectX SDK.

2.6 Alternativy

Jedním z nejvážnějších konkurentů Havok Physics je fyzikální engine PhysX od Ageia (v minulosti NovodeX). V současnosti je mezi těmito systémy velmi tvrdý konkurenční boj a rozhoduje se, který bude mít v budoucnosti nadvládu.

Výhody Havoku jsou v jeho zavedenosti, široké podpoře a propracované dokumentaci. Na druhou stranu nemá tak propracované efekty jako PhysX a vše musí počítat přes CPU.

Výhody PhysX jsou jeho kompletní otevřenost - je ke stažení kompletní SDK zdarma, ne pouze část a pro nekomerční použití jako u Havoku. Bohužel je PhysX omezen na Nvidia (a PhysX) karty. Při absenci těchto karet je výpočet odkázán pouze na CPU,
což má za následek velké rozdíly ve výkonu. Vzhledem k ceně Havok Physics
(cca 65 000 - 75 000 USD za projekt + možné poplatky z každého prodaného kusu), je otevřenost PhysX pro komerční sféru velkou výhodou.

Existují ještě dva fyzikální enginy, jež stojí za povšimnutí:

- **Newton Game Dynamics** – zdarma, ale ne open source, využívá plně deterministický algoritmus, což má za následek údajně daleko větší přesnost než konkurenti využívající iterační metody (defaultně Havok,ale umožňuje přepnutí i do deterministického módu). Bohužel na úkor rychlosti – oproti ostatním enginům je pomalejší (pracuje se ale stále na jeho urychlení). Údajně oblíbený v kombinaci s grafickými enginy Irrlicht popřípadě Ogre.

- **Open Dynamics Engine (ODE) –** zdarma, open source (dle licence BSD), C/C++ API, nejvíce se přibližuje k PhysX a Havoku, použit v mnoha známých herních titulech jako např. Call of Juarez, S.T.A.L.K.E.R, Xpand Rally, jeho nevýhodou je chybějící visual debugger

Dále existuje několik úzce specializovaných enginů pouze jako konkurence pro určitou část balíku Havok např.:

- ***AI* engine Kynapse od Autodesk** – placený, velmi populární (např.Unreal 3 engine), spíše konkurence k Havok Behavior

- **Digital Molecular Matter (DMM) -** umožňuje zničení předmětů pokaždé na jiné části a ne, jako u obyčejného destruction enginu, kde se daný předmět ničí vždy stejně. DMM je vyvíjeno LukasArts ve spolupráci s trikovým studiem Industrial Light&Magic.

- **Euphoria od Natural Motion** – další konkurence k Havok Behavior, velmi propracované AI. Nespoléhá na skripty - pracuje plně autonomně. Použití např. v GTA IV.

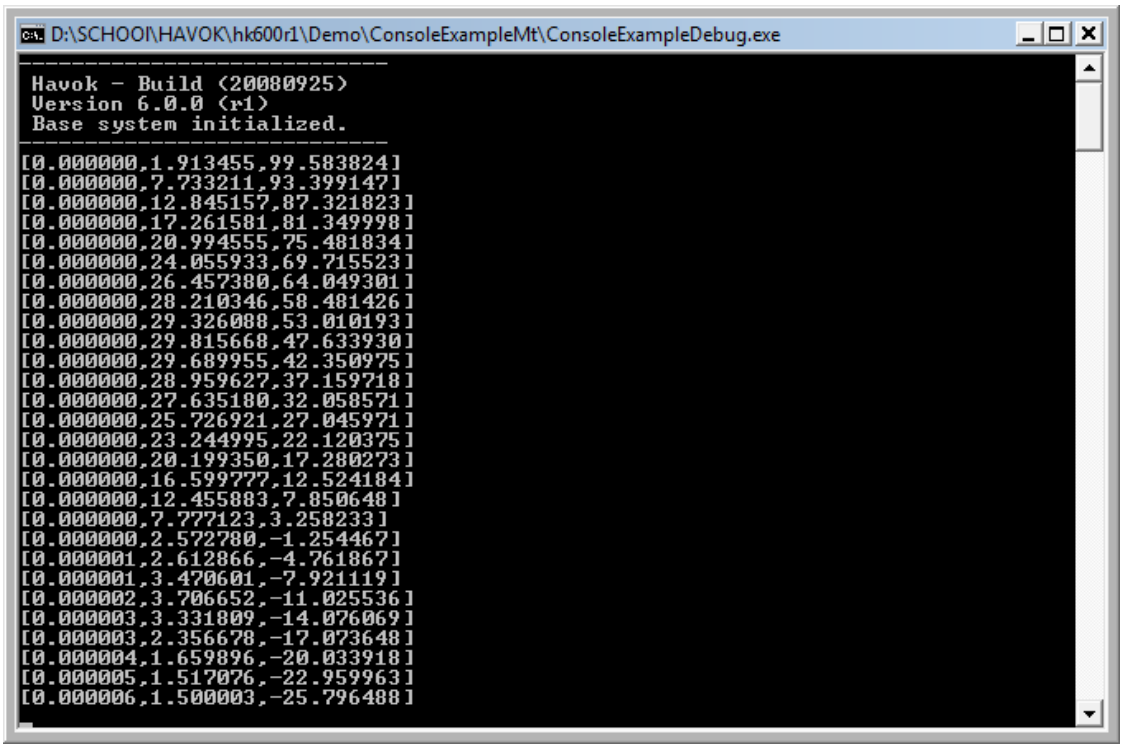
- **AI.implant** – další z řady konkurencí k Havok Behavior, ne tak rozšířený jako Euphoria

Je zde také možnost si koupit kompletní řešení, například v podobě CryEngine2, kde už je fyzika implementována.

2.7 První program

Úpravou standalone multithreaded dema přiloženého v SDK jsem se snažil o vytvoření co nejjednoduššího programu pro první začátky s Havok Physics. Program vytvoří pouze jednoduchou základnu, z které je na jednom jejím konci proveden šikmý vrh koule v úhlu 45°. Souřadnice koule se při tom vypisují v intervalu 250ms na konzoli. Nejlepší je si program celý postupně projít, snažil jsem se o co nejvíce komentářů ke každému kroku. Bohužel je nutné includovat každou použitou třídu, v dokumentaci je sice cosi psáno o globálním headeru pro začátečníky, ale v SDK bohužel nikde není. Pro kompilaci je nutné nastavit všechny cesty v projektu tak, jak je popsáno v předchozím kroku (kromě DirectX SDK).

Na obrázku 2.7-1 je vidět, jak Y-souřadnice roste a klesá jako u reálného šikmého vrhu. Z-souřadnice představuje vzdálenost šikmého vrhu.



Obr. 2.7 – 1 Šikmý vrh.

3. Ukázkové úlohy

3.1 Návrh

První návrh byl určitá souslednost na sebe navazujících kolidujících objektů, jako např. padající koule spustí domino efekt, ten spustí nějaký další efekt atd. Toto je implementováno v demu Domino. Dále je implementován rozstřel kulečníkových koulí – Pool. A poslední je jednoduchá implementace padesáti koulí – každá s jiným koeficientem vzpružnosti (restitution). Vytváří zajímavý vlnící se efekt.

3.2 Ukázky

3.2.1 Použité efekty

Friction – tření objektu, např. pilot domu z kostek v demu Domino má nastavené takřka nulové tření, aby šel „podseknout“ i v případě, že je zatížen tíhou celého domu

Dumping – útlum rychlosti, vynikající na simulaci tření vzduchu, linear dumping určuje jakým koeficientem má ustávat lineární rychlost. Angular dumping určuje ustávání rotační rychlosti objektu. Využité v kulečníkovém demu pro postupné zastavení koulí. Defaultně je linear dumping nastaven na nulu, takže se předměty chovají jako ve vakuu.

Mass – hmotnost, použitá prakticky všude

Gravity - v demu JumpingBalls je snížena gravitace světa na cca třetinu pro lepší efekt (zpomalení padání míčů)

HingeConstraint – vytvoření osy která je pevně svázaná s houpačkou u domu v třídě Domino

Restitution – koeficient restituce (vzpružnosti) - odrazivost objektů od překážek, využito hlavně v demu JumpingBalls

ActivationCallback – měnění barvy podle aktivace a deaktivace jednotlivých objektů, pozorovatelnost jak Havok Physics šetří výkon CPU, protože se nemusí zabývat nekolidujícími objekty, použité v Dominu

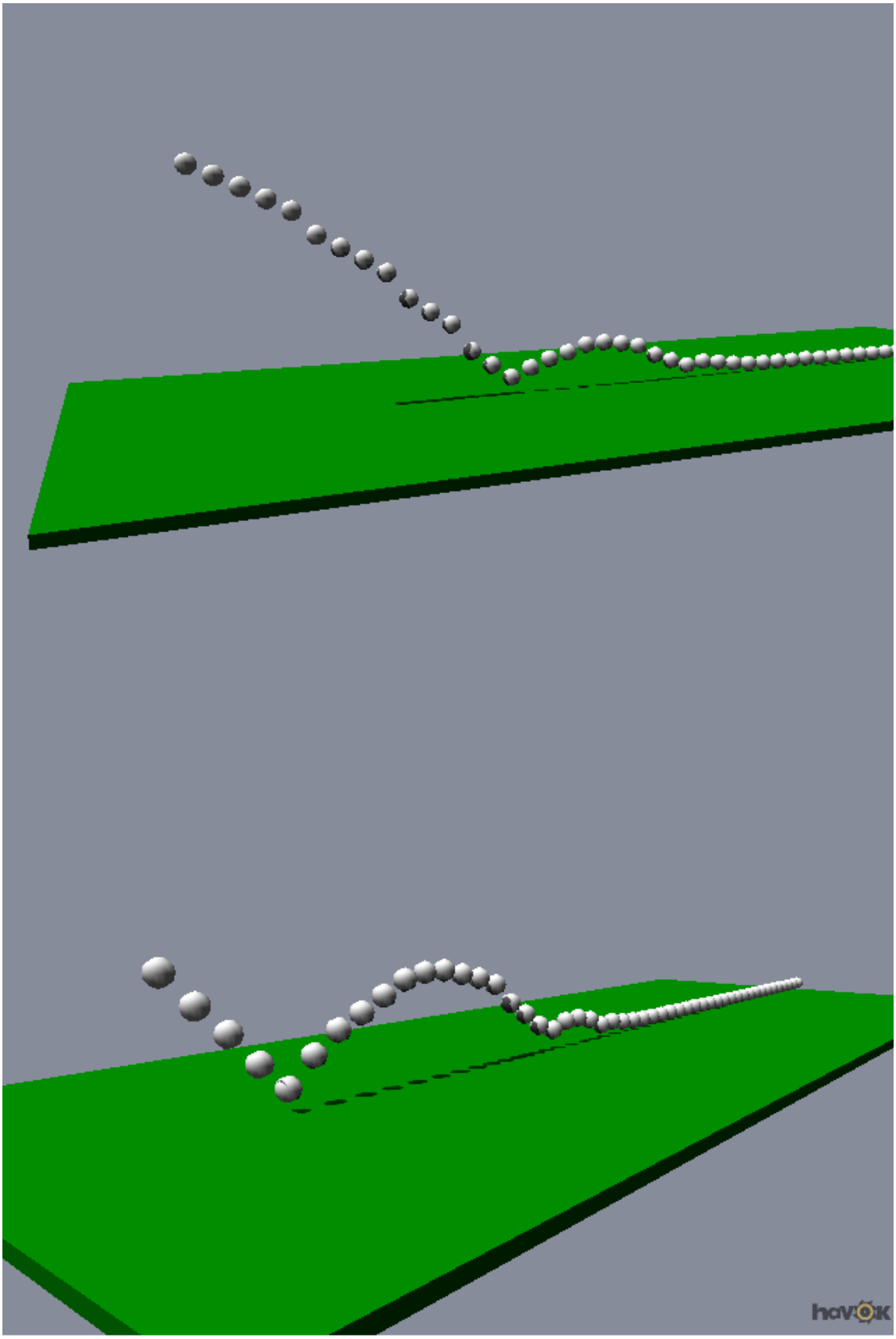
Ragdoll – figurka člověka padajícího ze schodů v úvodu Domino dema

3.2.2 Jumping Balls

Nejjednodušší ze všech ukázek. Padesát míčů vygenerovaných tak, že má každý proměnlivou odrazivost od povrchu - takzvanou restitution property. Vytváří efektní vlny. V tomto demu je využita možnost snížení gravitace pro docílení zpomaleného efektu. Tato třída je vhodná pro pochopení základních operací v Havok Physics.

Obsahuje pouze jedinou metodu a zároveň konstruktor a to je:
JumpingBalls::JumpingBalls(hkDemoEnvironment* env):hkDefaultPhysicsDemo(env)

Na obrázku 3.2.2–1 vidíme nepravidelné rozložení koulí, které naznačuje malý rozsah volby koeficientu restituce (popřípadě špatné zaokrouhlení někde uvnitř enginu). Pokud bychom přidali o pár koulí více a rozložili mezi ně rovnoměrně koeficient restituce od nuly do jedné, dostaneme se do fáze, kdy dva sousední objekty mají naprosto stejnou amplitudu, avšak koeficient restituce je u nich nastaven rozdílně.



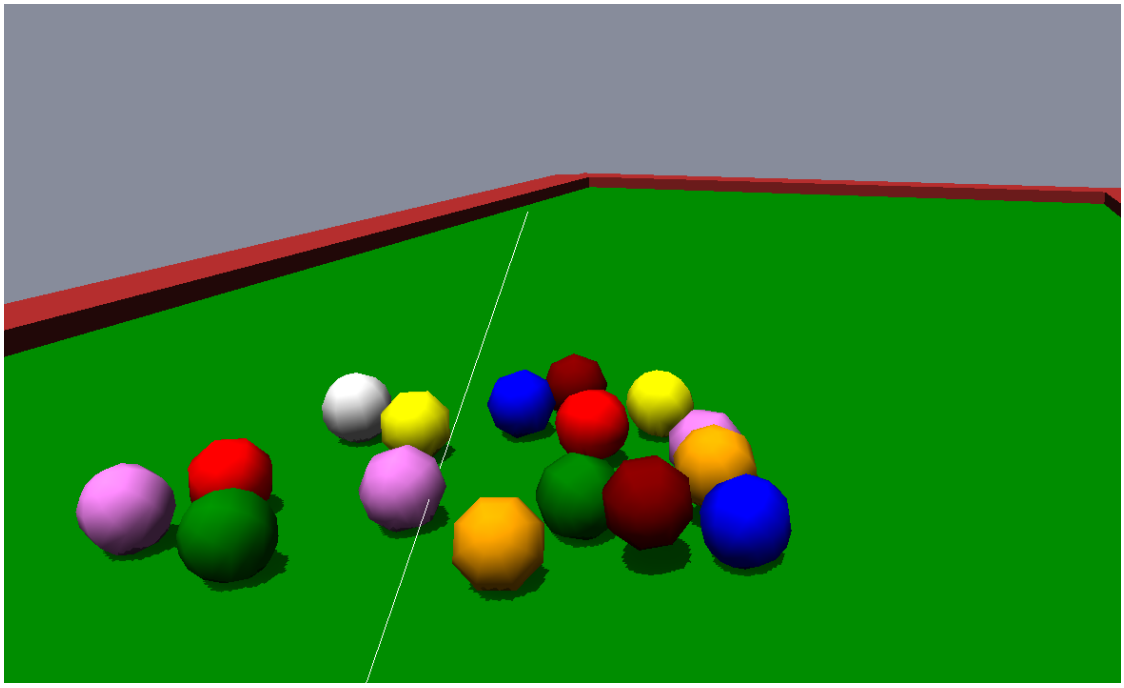
Obr. 3.2.2 – 1 Ukázka nestejnoměrnosti rozložení restituce.

3.2.3 Pool

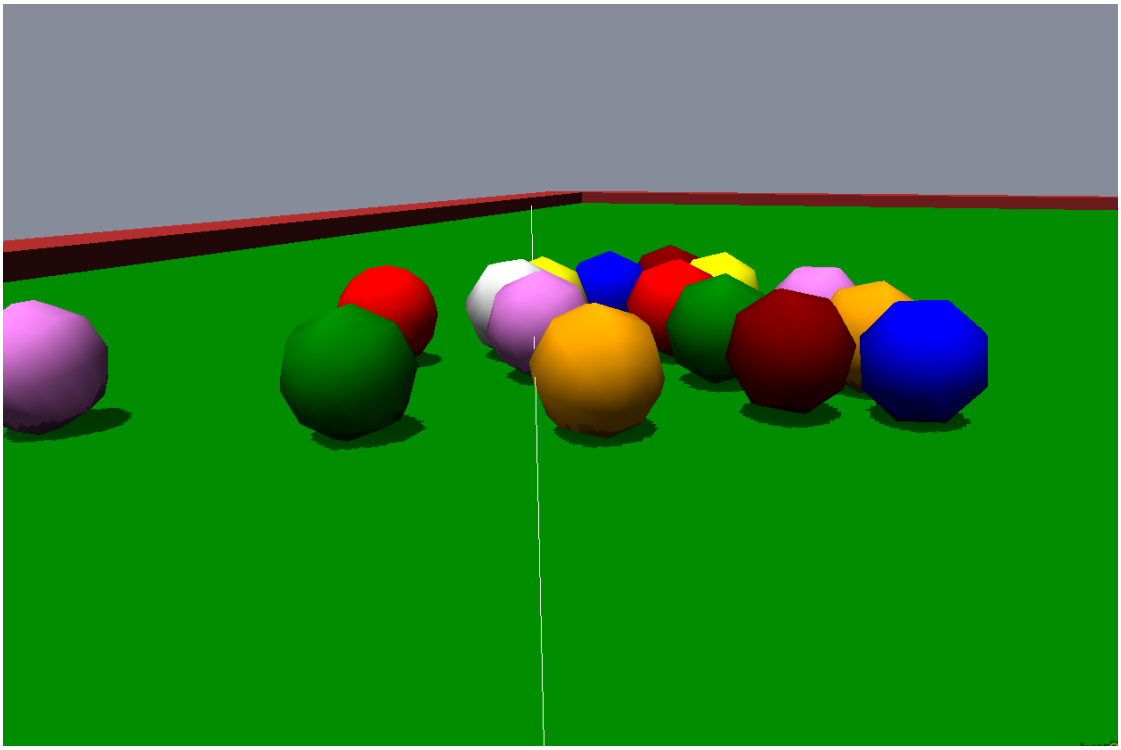
Tato třída simuluje základní problém kulečníku - rozstřel. Můžeme si nastavit libovolný úhel a rychlost jakou bílá koule zasáhne zbylých 15 koulí. Mimo jiné se zde potvrzuje, že je nevýhodné rozstřelovat přímo na nejbližší kouli, na rozdíl od šikmého šťouchu do druhé až třetí nejbližší řady. V budoucnu by byla možná např. kompletní implementace kulečníku s pomocí Havok Physics.
V této scéně bylo obtížné vyřešit restartování dema po každé úpravě kurzu a velikosti vektoru rychlosti (zobrazován šipkou). Je to vyřešené tak, že každé přidané rigid body se uloží do pole, abychom ho po provedení příslušné akce (úprava kurzu) mohli ze světa vyjmout a poté znovu rozmístit veškerá tuhá tělesa tak, jako tomu bylo na začátku. Je tu tedy poprvé použito ovládání dema za chodu a také grafické znázornění vektoru.

Metody:

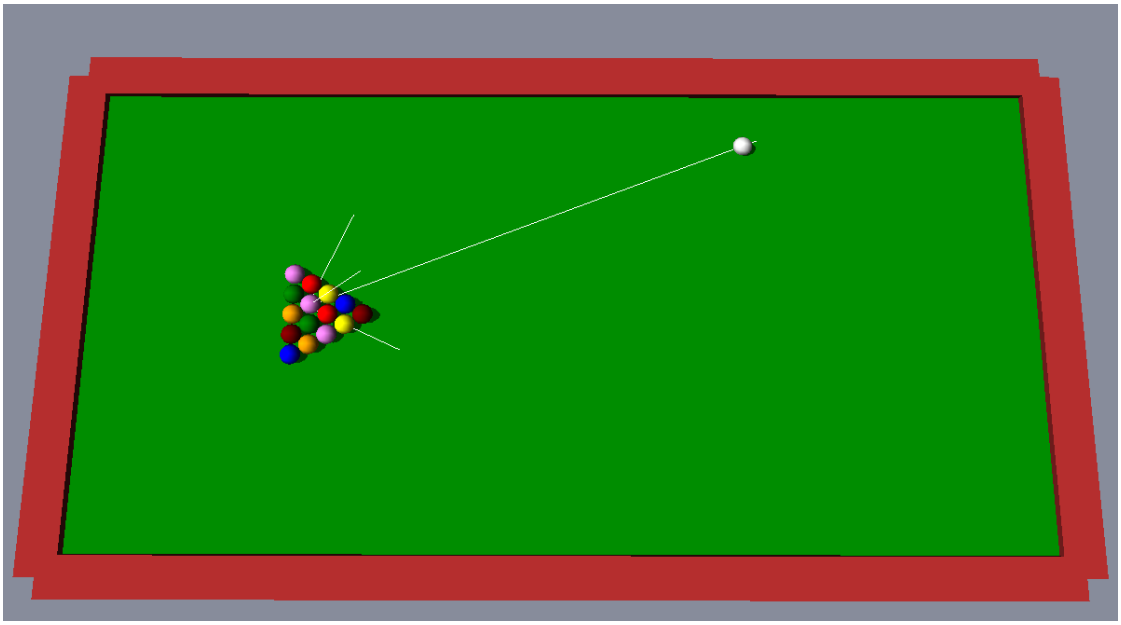
- **Pool::Pool(hkDemoEnvironment* env): hkDefaultPhysicsDemo(env)** – základní metoda, musí být v každé třídě, zde se vytváří svět, rigid bodies atd.
- **hkpRigidBody* createSphereFri(……)** – vytvoří kouli o zadané velikosti, hmotnosti, pozici, včetně nastavení součinitele tření. Zároveň je zde zmenšený inertia tensor kvůli dobré odrazivosti koulí od hran kulečníku
- **void Pool::createMovingBall(float velocity)** – vytvoří pohybující se míč směrem ke koulím o zadané rychlosti. Je nutný tento zápis vzhledem k tomu, že metodu voláme opakovaně po změně parametrů dema.
- **void Pool::createBalls()** – rozmístí znovu koule
- **void Pool::drawVector()** – vykreslí vektor rychlosti bílé koule
- **void Pool::createTable()** – vykreslí kulečníkový stůl
- **hkDemo::Result Pool::stepDemo()** – metoda,která se volá při každém překreslení grafiky



Obr. 3.2.3 – 1 Koule v okamžiku rozstřelu.



Obr. 3.2.3 – 2 Moment styku bílé koule s ostatními.



Obr. 3.2.3 – 3 Celkový pohled na scénu.

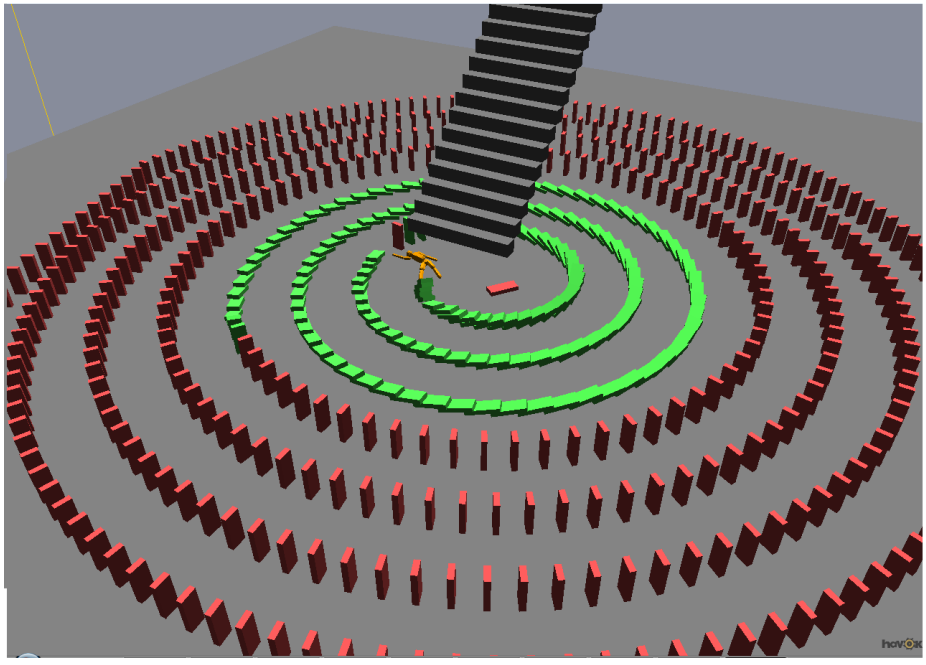
3.2.4 Domino

První scéna, kterou jsem vytvořil v Havok Physics. Nejprve byla implementovaná pouze do standalone dema a Visual Debuggeru, později jsem ji plně zakomponoval do Havok Demo Framework. Je to nejkomplexnější scéna této bakalářské práce – využil jsem zde takřka všech výše uvedených efektů, včetně využití třídy Ragdoll (pomocí třídy GameUtils, která zlehčuje vytváření rigid bodies díky velmi zjednodušeným příkazům).

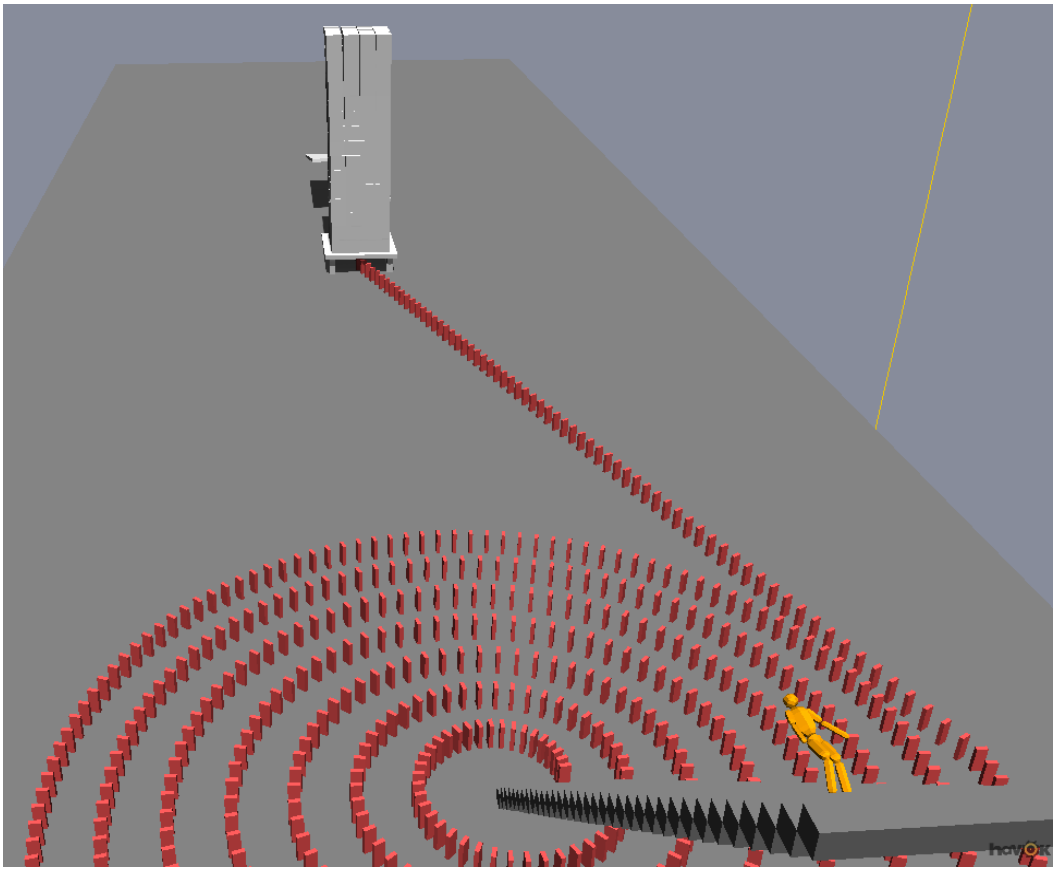
Jedná se o posloupnost na sebe navazujících akcí, kde nejdříve padá ragdoll ze schodů, poté navazuje domino efekt, na kterém je vidět postupná aktivace (zelená) a deaktivace (červená) fyziky pro každý objekt (šetří výkon CPU), dále bylo využito postupné přibývání hmotnosti posledních domino kostek, aby dokázaly podtrhnout pilíř domu. Pilíř musel mít tření snížené takřka na minimum, protože jinak ho hmotnost jednotlivých bloků (krychlí) domu tak zatížila, že nebylo možné ho podtrhnout. Dále navazuje houpačka s pevnou osou, která vyhodí do vzduchu poslední kostku. Po určité době nečinnosti na konci action listener deaktivuje všechny kostky a změní tak barvu na červenou.

Třída obsahuje tyto metody:

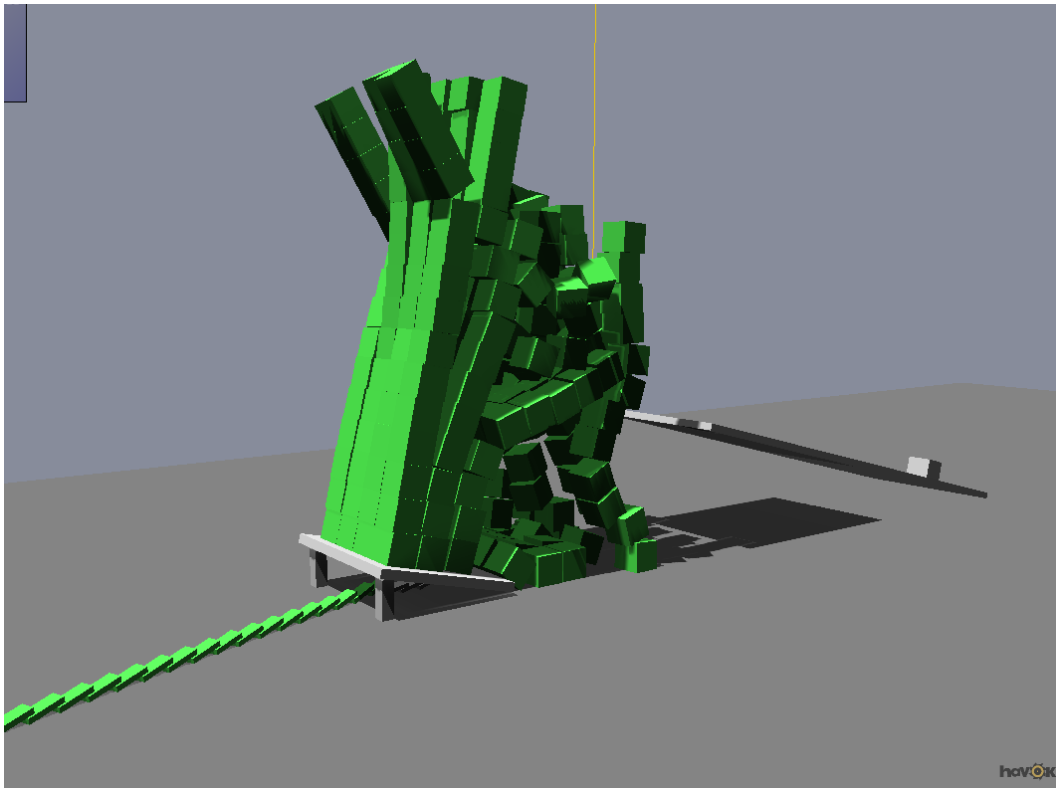
- **ActivationCallbacksDemoEntityActivationListener** – zajišťuje zpětnou vazbu od automatických aktivací a deaktivací objektů
- **hkpRigidBody* createBoxFastMoving(……)** – vytvoří kvádr o zadané velikosti, hmotnosti, a pozici s typem pohybu pro rychle pohybující se objekty
- **hkpRigidBody* createBoxFriction(……)** – vytvoří kvádr o zadané velikosti, hmotnosti, pozici, včetně nastavení součinitele tření
- **hkpRigidBody* createBoxRotated (……)** – vytvoří kvádr o zadané velikosti, hmotnosti, pozici, včetně nastavení rotace objektu
- **Domino::Domino(hkDemoEnvironment* env) –** konstruktor
- **void Domino::createFilter()** – filtr kolizí nutný pro implementaci ragdollu
- **void Domino::createBasePlatform()** – vytvoří základnu dema
- **void Domino::createDominoLine()** – vytvoří dominovou řadu - od spirály až po dům
- **void Domino::createHouse()** – vytvoří dům
- **void Domino::createTeeterTotter()** – vytvoří houpačku s připravenou kostkou za domem, houpačka má destruktibilní osu - je nastavena určitá síla, při které přestane osa houpačky existovat
- **void Domino::createRagdoll()** – vytvoří panáka padajícího se schodů, částečně převzato a upraveno z examplu technického dema



Obr. 3.2.4 – 1 Dopad panáka a reakce na domino.



Obr. 3.2.4 – 2 Padající panák.



Obr. 3.2.4 – 3 Destrukce domu složeného z kostek.

3.3 Instalace

3.3.1 Překlad

Aplikaci lze spustit ve Visual Studio 2005 popřípadě 2008 (s Havok 6.5) pouze pokud jsou správně nalinkované knihovny jak pro multithreaded debug, tak pro multithreaded release, tak pro DirectX SDK jak je popsáno v kapitole 2.5.

3.3.2 Spuštění

Aplikace se spouští souborem:
Demos_win32-net_9-0_debug_multithreaded.exe popřípadě
Demos_win32-net_9-0_release_multithreaded.exe vygenerovaným přímo do adresáře dema.

3.3.3 Ovládání

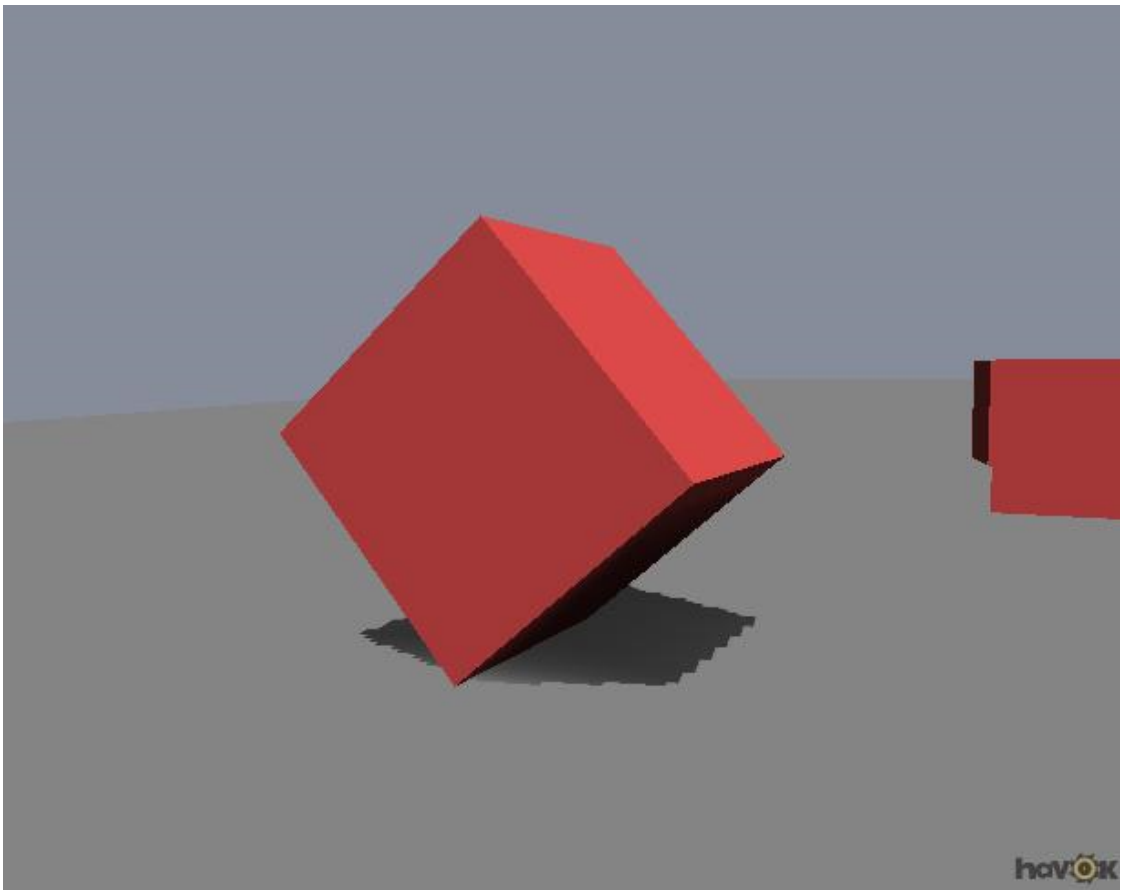
Ovládání jednotlivých programů je popsáno vždy v levém horním rohu. Havok Demo Framework se ovládá:
Escape – Exit
Enter – menu ve spuštěném demu pro restartování popř. jeho ukončení.
Šipky – pohyb v menu.

3.5 Problémy nestability

Při práci se enginem Havok jsem se setkal s několika ukázkami nestability. Avšak téměř vždy jsem je byl schopen vyřešit s pomocí fóra Havok. Např. rychle se pohybující kostka při dopadu z výšky penetrovala částečně do povrchu základny. Při velké rychlosti dokonce povrch penetrovala – stačilo změnit typ pohybu na specializovaný pro rychle se pohybující objekty a bylo po problému. Obecně vzato se dají různá podivná

chování eliminovat změnou nastavení parametrů objektu, přesnosti detekce jeho pohybu a nastavení typu jeho pohybu.

Jedním z nich byl např. mnou objevený problém, který se vyřešil až na Havok fóru. Byl u destrukce domu v ukázce Domino - při určitém nastavení se stalo, že se některé kostky deaktivovaly v nepřirozené poloze - balancováním na hraně. Bohužel se mi tento problém nepodařilo reprodukovat po změně pár parametrů v prvních úsecích dema a tak mám jenom printscreen:



Obr. 3.5 – 1 Chyba Havok Physics. Kostka stojí na trvale na hraně.

Naštěstí se ozval další člověk se stejnou zkušeností, takže problém pomohli vyřešit vývojáři z Havoku a to jenom prostým podhodnocením momentu hybnosti tělesa.

Dalším problémem bylo chování houpačky pod tlakem padajících kostek. Sice je houpačka udělaná z daleko lehčího materiálu, než jsou kostky, ale i tak mně její chování v množství kostek přijde dost nereálné. Stejně tak její chvění v klidovém stavu, když se na jejím povrchu nachází množství kostek. Tento problém by šel vyřešit zvýšením hmotnosti houpačky na reálnou hodnotu, avšak pak bychom byli ochuzeni o efekt vystřelené kostky, neboť kostky domu nejsou dostatečně těžké, aby dokázaly normálně vážící houpačku rozpohybovat.

4. Testy

Většina testů je tvořena porovnáním s předchozím verzí Havok 6.0, neboť jsem jich většinu na tomto enginu stvořil. Během mé práce se však objevila nová verze 6.5 a tak jsem se rozhodl přejít na aktuální a dát možnost porovnání s předchozí verzí.
Ve zdrojových souborech bude zřetelně označena stejná kompilace, ale ve verzi 6.0.

4.1 Test 1: Velmi rozdílné hmotnosti

4.1.1 Problém

Tato scéna byla vytvořena k otestování, jak si fyzikální enginy poradí s kolizí objektů o velkých rozdílech v hmotnosti a velikosti.

4.1.2 Scéna

Na základně (koeficient restituce k=0, součinitel smykového tření f=1) je umístěna krychle (a=1m, m=5kg, k=0, f=1), na kterou postavíme přesně na střed druhou krychli (a=10m, m=5000kg, k=0, f=1). Předpokládaným chováním je klidný stav, kdy velká krychle leží na malé a systém je v klidu.

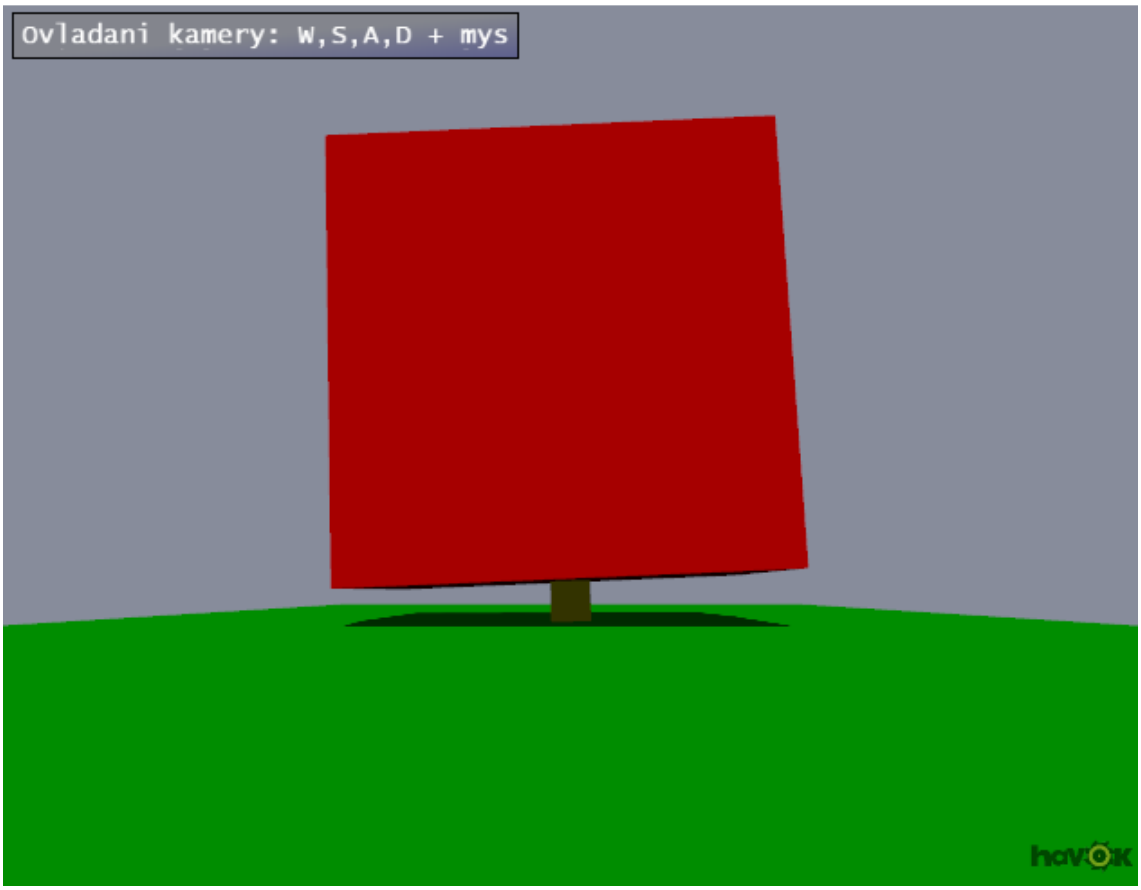
4.1.3 Výsledek

Velká krychle propadne skrz malou krychli na základnu s minimálním odporem, jakoby tam malá krychle takřka vůbec nebyla.

4.1.4 Řešení

Vylepšit (bohužel ne úplně vyřešit) toto chování pomohlo nastavit world collision solver na maximální přesnost (8 iterací hard oproti 4 iterace medium, které je nastaveno jako standard).
Po této úpravě se systém choval lépe, velká krychle zůstala na malé krychli tak, jak by každý očekával, ale bohužel začala nezvykle poskakovat a odrážet se, až spadla na stranu, kde pohyb po chvíli ustal.

Po bližším zkoumání je hranice do kdy se Havok 6.5 chová v tomto testu stabilně do cca 4000 kg pro horní krychli. Havok 6.0 se stabilně chová do cca 2500 kg, takže je zde patrné zlepšení od předchozí verze.



Obr. 4.1 – 1 Pohled na scénu a padající krychli na levou stranu.

4.2 Test 2: Zachování energie

4.2.1 Problém

Tento test byl vytvořen za účelem prověření, jak fyzikální enginy zvládají tak základní věc, jakou je fyzikální kyvadlo.

4.2.2 Scéna

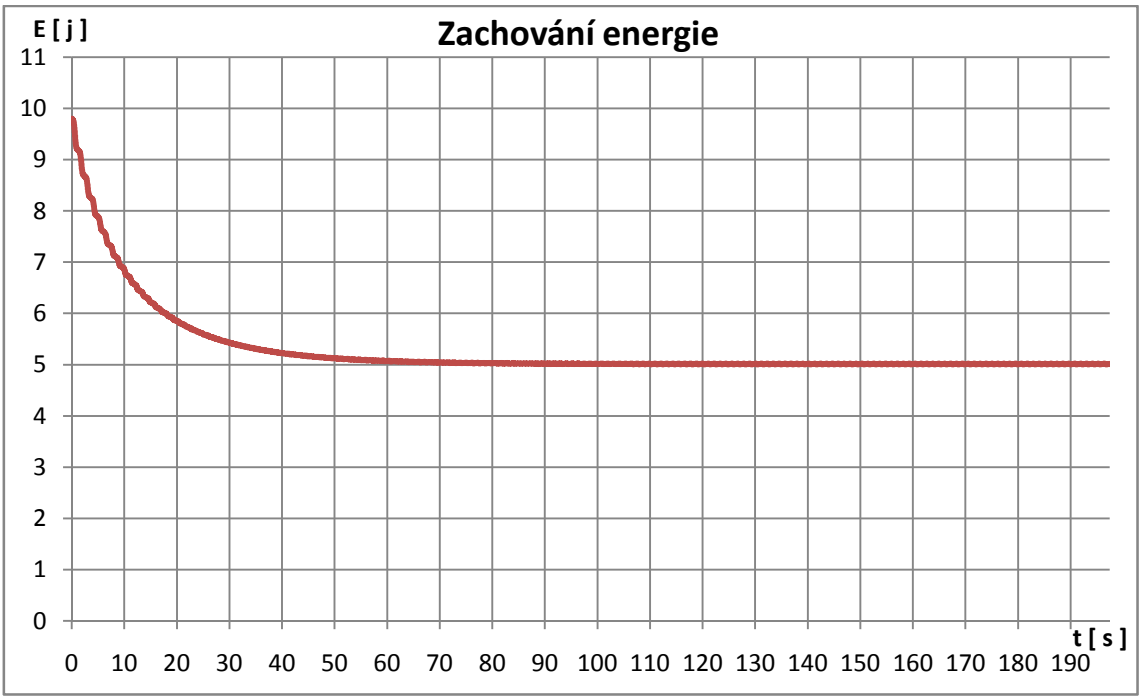
Kyvadlo (koule o poloměru r=5cm, m=1kg) je zavěšené na závěsu dlouhém 1m. Výchozí poloha je kolmo na tíhovou sílu (α=90˚).

4.2.3 Výsledek

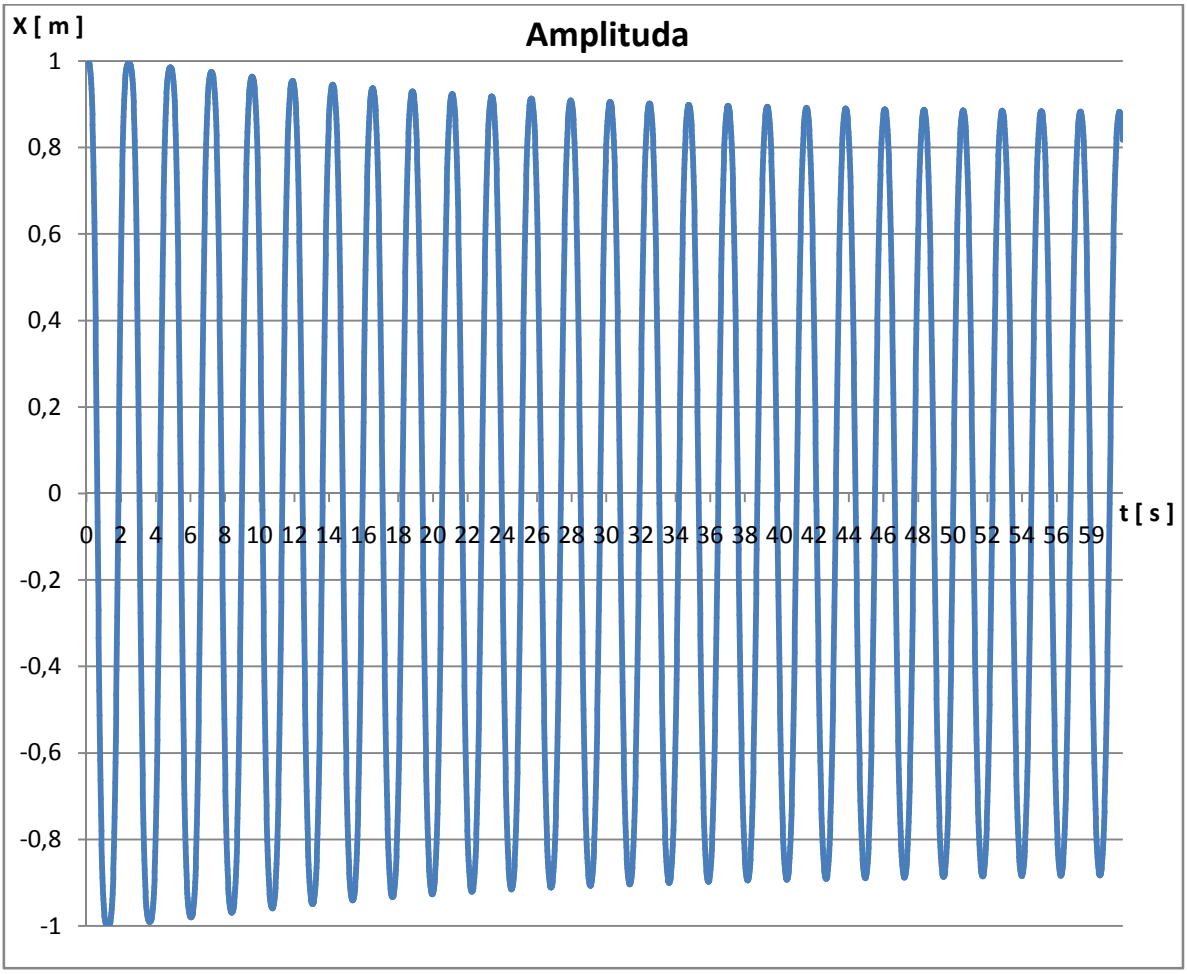
Tento test byl zklamáním, neboť Havok nedokázal zachovat energii kyvadla, tudíž jeho kmitání bylo tlumené. I když byl odpor prostředí nastaven na nulu (tudíž jsme simulovali vakuum), nastavena maximální přesnost, přesto nic nepomáhalo. Systém ztratil skoro 50% celkové počáteční energie (z 9.8 J na cca 5.1 J), poté se kmitání ustálilo.

Tento test generuje soubor Test2-out.txt, kde jsou tabulátorem oddělené postupně hodnoty: čas, celková energie kyvadla, X-ová výchylka kyvadla. Toto umožňuje kdykoliv zpětné porovnání přesných hodnot s jiným enginem. Celková energie kyvadla je počítána vztahem $E = E_p + E_k = mgh + {}^1\!/_2\, mv^2$

Havok 6.0 a 6.5 se v tomto testu chovají naprosto stejně.



Obr. 4.2 – 1 Chybná ztráta energie kyvadla.



Obr. 4.2 – 2 Chybně tlumená amplituda kyvadla při netlumeném kmitání.

4.3 Test 3: Osový systém

4.3.1 Problém

Tato scéna testuje osový a závěsový systém spolu s detekcí kolizí a odezvou systému.

4.3.2 Scéna

Na základně (koeficient restituce k=0, součinitel smykového tření f=1) je umístěna krychle (a=1m, m=5kg, k=0, f=1), z které vede pevná osa (délky l=2m) nahoru ve směru osy Y do středu podstavy horní krychle (a=10m, m=2000kg, k=0, f=1), kde se nachází kulový kloub s rozsahem α=3. Předpokládaným chováním je klidný stav, kdy velká krychle balancuje na pevné ose nad malou a systém je v klidu.

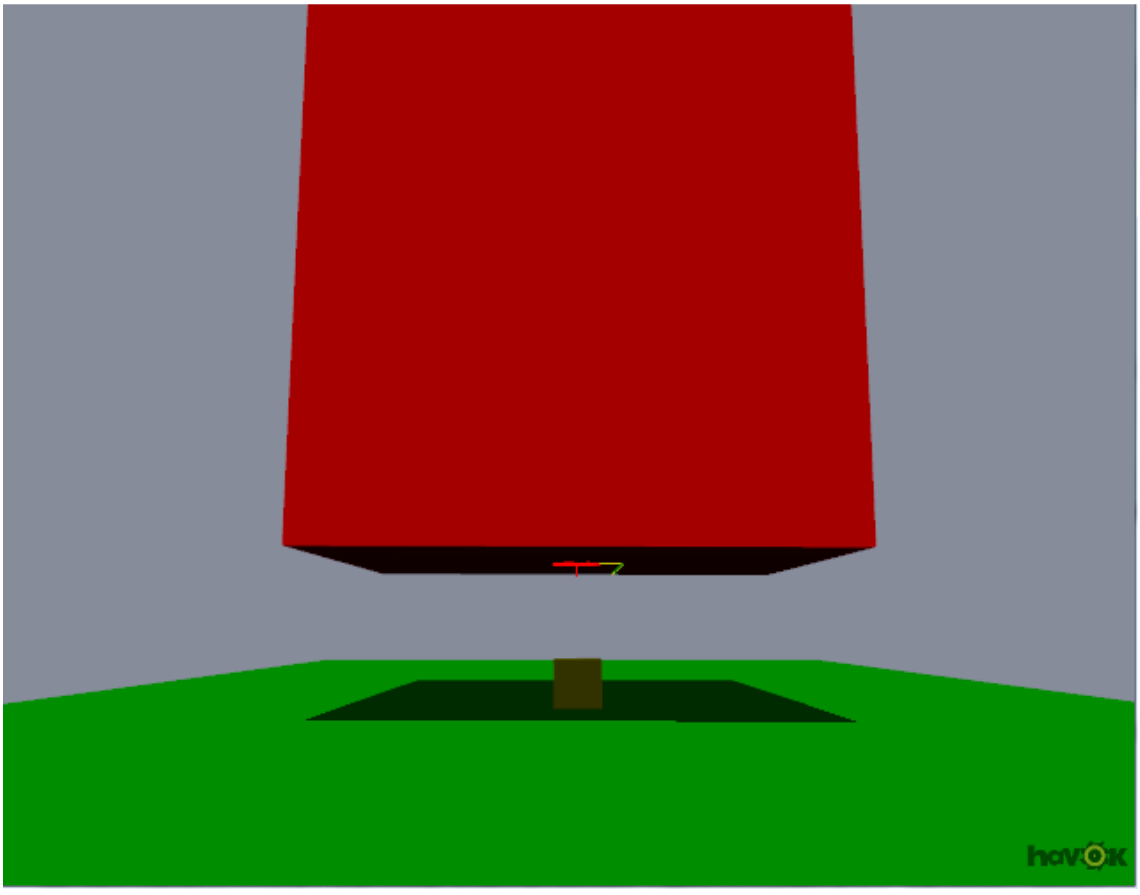
4.3.3 Výsledek

Dojde k totálnímu zhroucení systému, osy neudrží hmotnost velké krychle a ta spadne na základnu. Poté dojde k obnovení vazby a malá krychle je znovu pevně připoutaná k velké, ale ta osu stále svojí váhou deformuje. Nakonec systém deaktivuje obě tělesa v nepřirozené poloze.

4.3.4 Řešení

Tento problém jsem vyřešil nastavením world collision solveru na maximální hodnotu. Systém se poté choval stabilně a přesně podle očekávání.

Po bližším zkoumání je hranice do kdy se Havok 6.5 chová v tomto testu stabilně do cca 3000 kg pro horní krychli. Havok 6.0 se stabilně chová do cca 2500 kg, takže je zde taktéž patrné zlepšení od předchozí verze.



Obr. 4.3 – 1 Pohled na základní scénu testu 3.

4.4 Test 4: Stabilita závěsu

4.4.1 Problém

Tento test byl vytvořen za účelem prověření, jak fyzikální enginy dodržují přesně stanovenou délku osy. Jestli se vlivem větší hmotnosti osa neprodlouží a někde naopak nezkrátí.

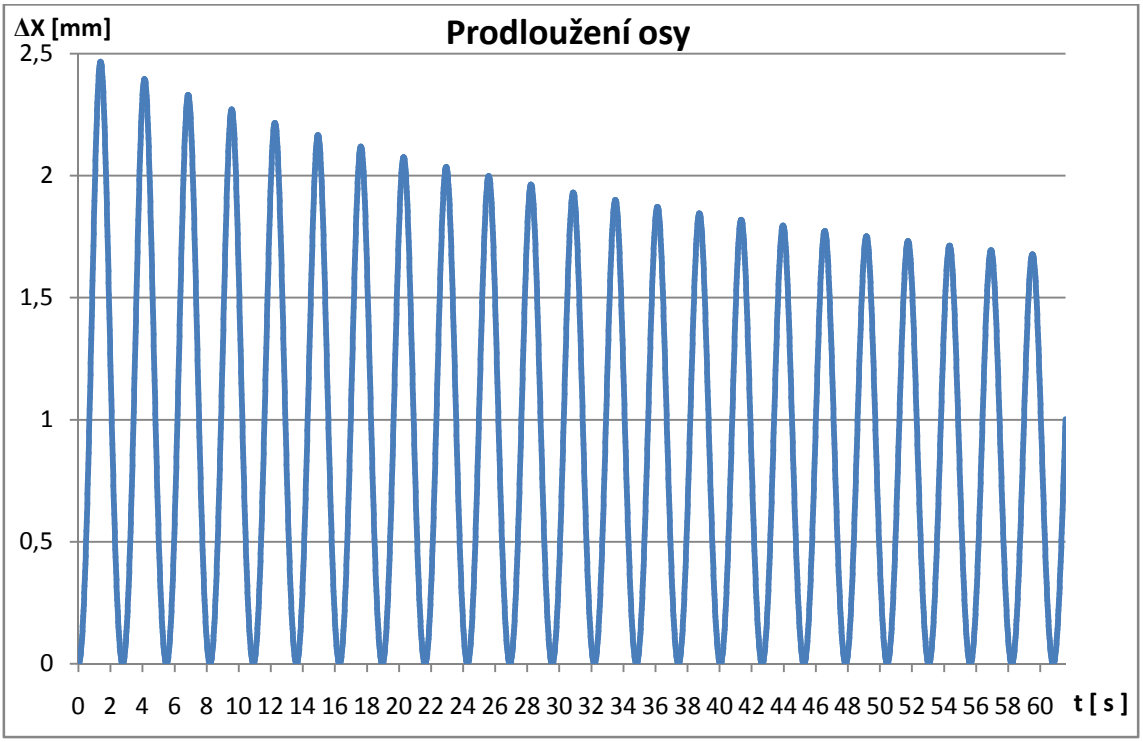
4.4.2 Scéna

Kyvadlo (koule o poloměru r=1m, m=5000kg) je zavěšené na závěsu dlouhém 5 m. Výchozí poloha je kolmo na tíhovou sílu (α=90˚). Pomocí konzole a výpisu do souboru měříme v každém vykreslení délku osy a odečítáme rozdíl od počáteční vzdálenosti. Výstupem je soubor Test4-out.txt, ve kterém je zaznamenán čas ve vteřinách a odchylka od původní polohy v milimetrech.

4.4.3 Výsledek

Objevilo se drobné prodloužení (v maximu 2.47mm). Které se poté ustálilo na 1.68 mm. Je zde vidět vliv tlumení popsaném v druhém testu. Testoval jsem různé hmotnosti a odchylka evidentně nezávisí na váze tělesa. Troufám si toto označit za velmi dobrý výsledek, neboť se jedná o zanedbatelnou chybu v porovnání se vzdáleností (0.00005%).

Havok 6.0 a 6.5 se v tomto testu chovají naprosto stejně.



Obr. 4.4 - 1 Prodlužování osy na základě kmitání kyvadla.

4.5 Test 5: Osový systém 2

4.5.1 Problém

Tato scéna testuje stabilitu osového a závěsového systému a hlavně jeho chování v návaznosti na velké rozdíly ve hmotnostech, včetně nekonečné hmotnosti pevné základny.

4.5.2 Scéna

Ze základny je vedena svisle dolů ve směru Y první osa (s kulovým kloubem u základny o stupni volnosti α=3° a délce l=2,5m) směrem doprostřed horní stěny malé krychle (a=1m, m=5kg, k=0, f=1), kde je pevně uchycená (nachází se zde takzvaný pivot). Ze středu dolní podstavy malé krychle vychází další osa, která má v tomto bodě také kulový kloub (α=3°, l=5m). Tato druhá osa vede dále dolů ve směru osy Y k druhé krychli (a=10m, m=5000kg, k=0, f=1), kde je opět napevno připojená ke středu horní stěny velké krychle.

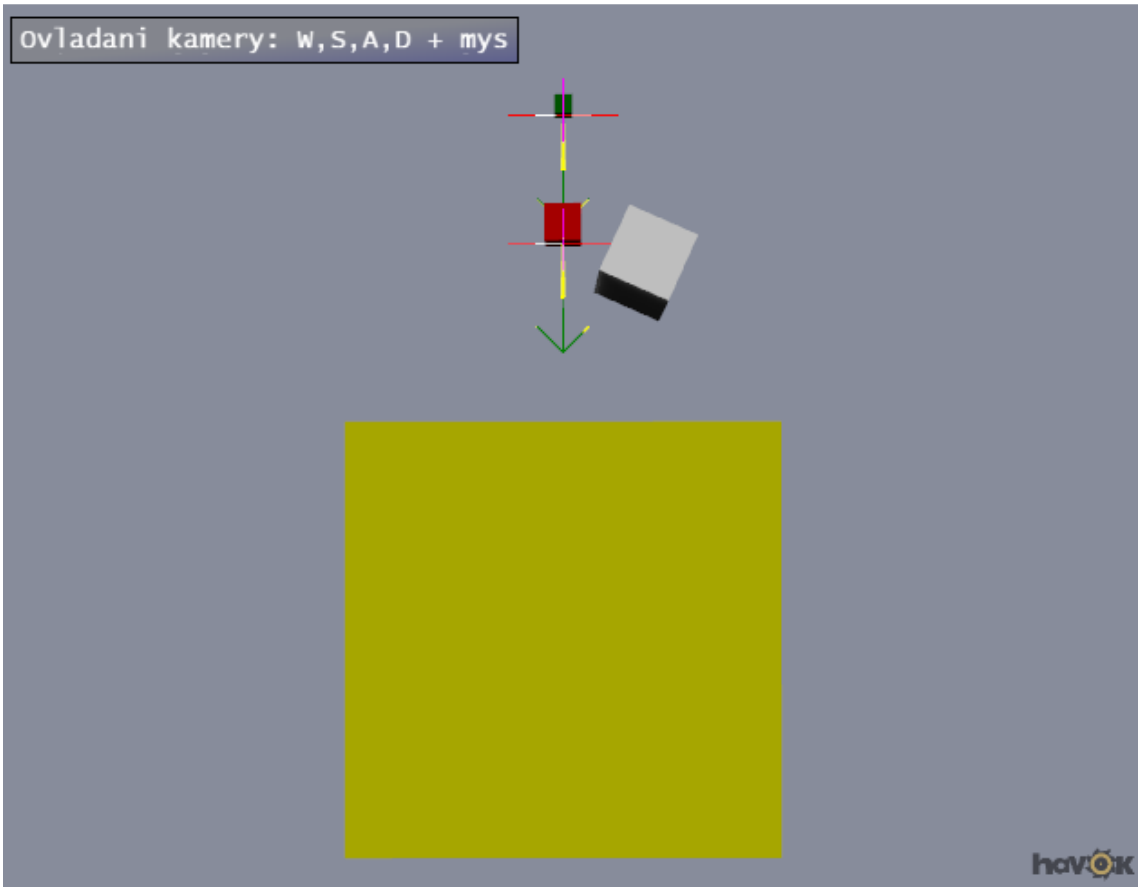
Do těchto krychlí spadne ještě jedna krychle o hraně 2 metry a hmotnosti 100kg, jenom abychom soustavu rozpohybovali a mohli sledovat, jak si s tímto Havok Physics poradí. Očekávané chování je stále stabilní závěs, který se bude minimálně pohybovat do stran vzhledem k omezení kloubů na 3.

4.5.3 Výsledek

Dojde k totálnímu zhroucení systému, osy neudrží hmotnost velké krychle, malá krychle začne kmitat, dokud nedosáhne kritické meze a celý systém se zhroutí.

4.5.4 Řešení

Vyřešit toto chování pomohlo nastavit world collision solver na maximální hodnotu. Systém se poté chová stabilně přesně podle očekávání.



Obr. 4.5 – 1 Ukázka základního rozmístění scény testu 5.

4.6 Test 6: Přesnost kolize + restituční model

4.6.1 Problém

Tato scéna byla vytvořena k otestování přesnosti výpočtu odrazu na základě dopadu kulového objektu na vodorovnou podložku. Druhotně také testuje přesnost chování při nastavení součinitele restituce.

4.6.2 Scéna

Na základnu (koeficient restituce k=1, součinitel smykového tření f=1) spustíme z výšky 10 metrů kouli (r=0.5m, m=10kg, k=1). Očekáváme volný pád koule a následný odraz koule zpátky přesně do původní polohy, z které jsme ji vypustili a tak opakovaně do nekonečna. Vzhledem ke koeficientu restituce, který jsme nastavili u podložky i koule na 1, by se podle teorie fyziky mělo jednat o naprosto elastickou srážku bez ztráty energie. Přesná poloha koule je vykreslovaná periodicky na konzoli.

4.6.3 Výsledek

Havok 6.0 : Koule evidentně každým svým odrazem ztrácí podstatnou část svojí energie, po šesti odrazech ztratí veškerou energii a zůstane nehybná na podložce. Navíc se neodráží přesně ve směru osy Y tak, jak jsme ji vypustili, ale i nepatrně ve směru osy X. Jenom za těchto šest odrazů se posunula po ose X o 0.2 mm.

Havok 6.5 : Koule každým svým odrazem ztrácí nepatrně svojí energie. Projevuje se to poklesem výšky o cca 20 cm po každém odrazu. Je zde ale velmi patrné zlepšení oproti předchozí verzi. Navíc byl odstraněn nepřesný odraz od podložky ve směru osy X.

4.6.4 Řešení

Vyřešit toto chování pomohlo nastavit world collision solver na maximální přesnost, dále nastavit u koule lineární a angulární tlumení na nulu a nastavit kvalitu pohybu koule na Bullet quality mode neboli na absolutní přesnost nezávisle na časovém kroku enginu. Po této úpravě se koule chová o dost lépe, dalo by se říct i podle očekávání, avšak i toto není dokonalé. Koule naopak po určité době začne nabírat na intenzitě a začne svoji výšku zvětšovat. Toto jde vyřešit ručním doladěním konkrétního objektu pomocí detektoru kolizí a periodické kontrole a upravování množství celkové energie, kterým zrovna objekt disponuje, ale je to vcelku komplikované řešení, které obchází samotný Havok a nedokážu si tyto úpravy představit v nějaké komplikované scéně.

4.7 Test 7: Mnoho stejných kolizí

4.7.1 Problém

Tato scéna testuje, jak si fyzikální enginy poradí s velkým množstvím kolizí tuhých těles a jejich vzájemným trvalým kontaktem.

4.7.2 Scéna

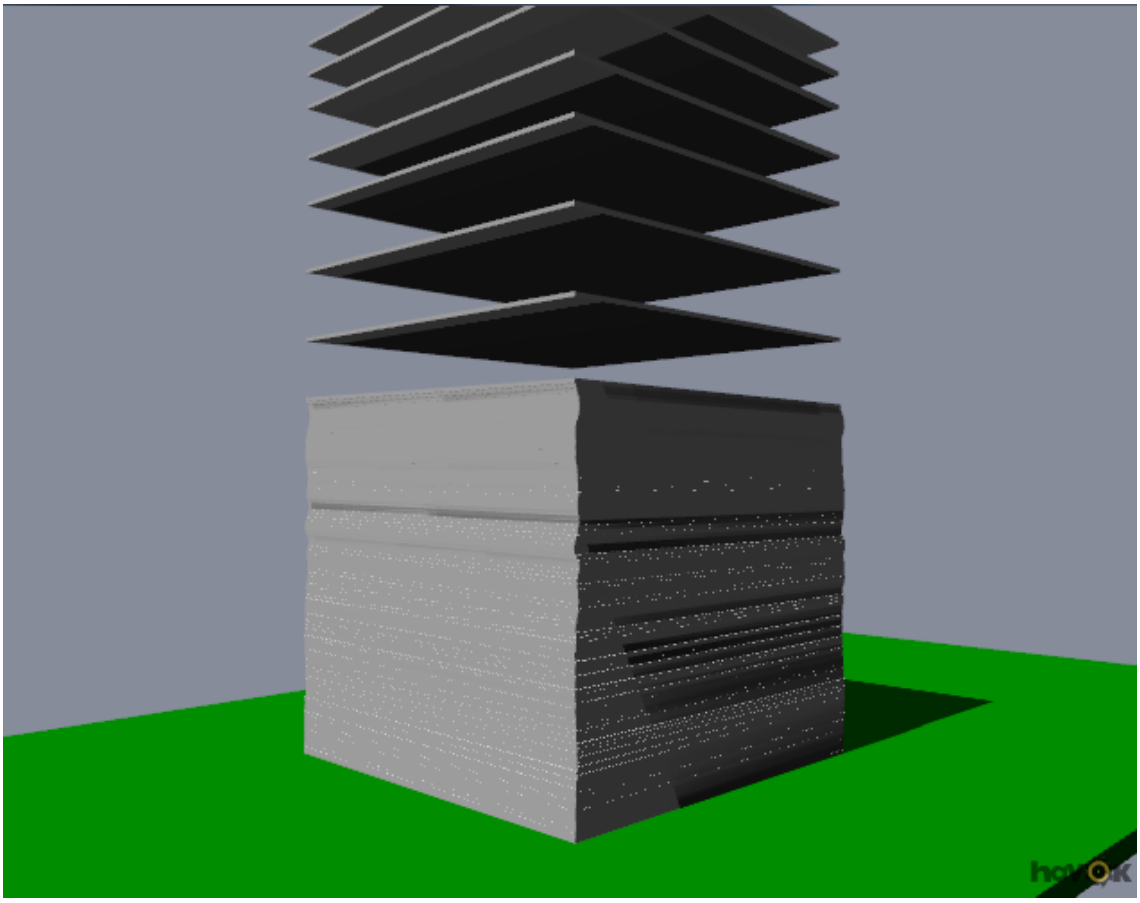
Na základnu (k=0, f=1) spouštíme opakovaně z výšky z bodu (0,20,0) desky o rozměrech 10x0.1x10m (m=10kg, k=0, f=1). Očekáváme klidný dopad desky přímo na podložku a tak na další a další desky až vytvoří stabilní sloupec. Desky vypouštíme šipkou dolů na klávesnici.

4.7.3 Výsledek

Netvoří se stabilní sloupec, desky jsou různě pootočené, občas je nějaká ze sloupce značně nerealisticky vytlačována (bez jakéhokoliv důvodu), častá destrukce celého sloupce ve výšce menší než 10 metrů. Celý sloupec se rozvlní a spadne.
Havok 6.0 dává lehce horší výsledky, ale styl chování je stejný.

4.7.4 Řešení

Vyřešit toto chování pomohlo nastavit world collision solver na maximální přesnost a dále Bullet quality mode u každé desky. Po této úpravě stavíme stabilní sloupce pokaždé minimálně 15 metrů vysoké, dále se nekoná žádné nerealistické vytlačování jednotlivých desek ze sloupce a nejsou nijak pootočené. Přesto se nežádoucí vlnění celého sloupce odstranit úplně nepodařilo. Tento problém by se dal řešit násilnou deaktivací celku po přidání poslední desky, které jsem implementoval do opravené verze na klávesnici šipku vlevo. Tímto deaktivujete aktuální objekt, popřípadě takzvaný celý jeho simulation island (jeho nejbližší okolí s kterým koliduje). Pokud tedy deaktivujete poslední dopadnutou desku, deaktivuje se tím celý sloupec a jeví se jako 100% stabilní.



Obr. 4.7 – 1 Ukázka scény testu 7 u opravené verze.

4.8 Test 8: Maximální rychlost

4.8.1 Problém

Tento test byl vytvořen, aby ověřil jakou maximální rychlost je schopen fyzikální engine zpracovat, aby nedocházelo k penetraci mezi jednotlivými objekty. Většina enginů totiž pracuje v určitých časových okamžicích (iterační princip). Pokud máme předmět s dostatečně velkou rychlostí, můžeme se trefit přesně do okamžiku, kdy systém detekuje objekt před překážkou, mezitím skrz ní objekt proletí, a v dalším kroku engine detekuje objekt za překážkou. Tím došlo k chybě a systému unikla jedna kolize.

4.8.2 Scéna

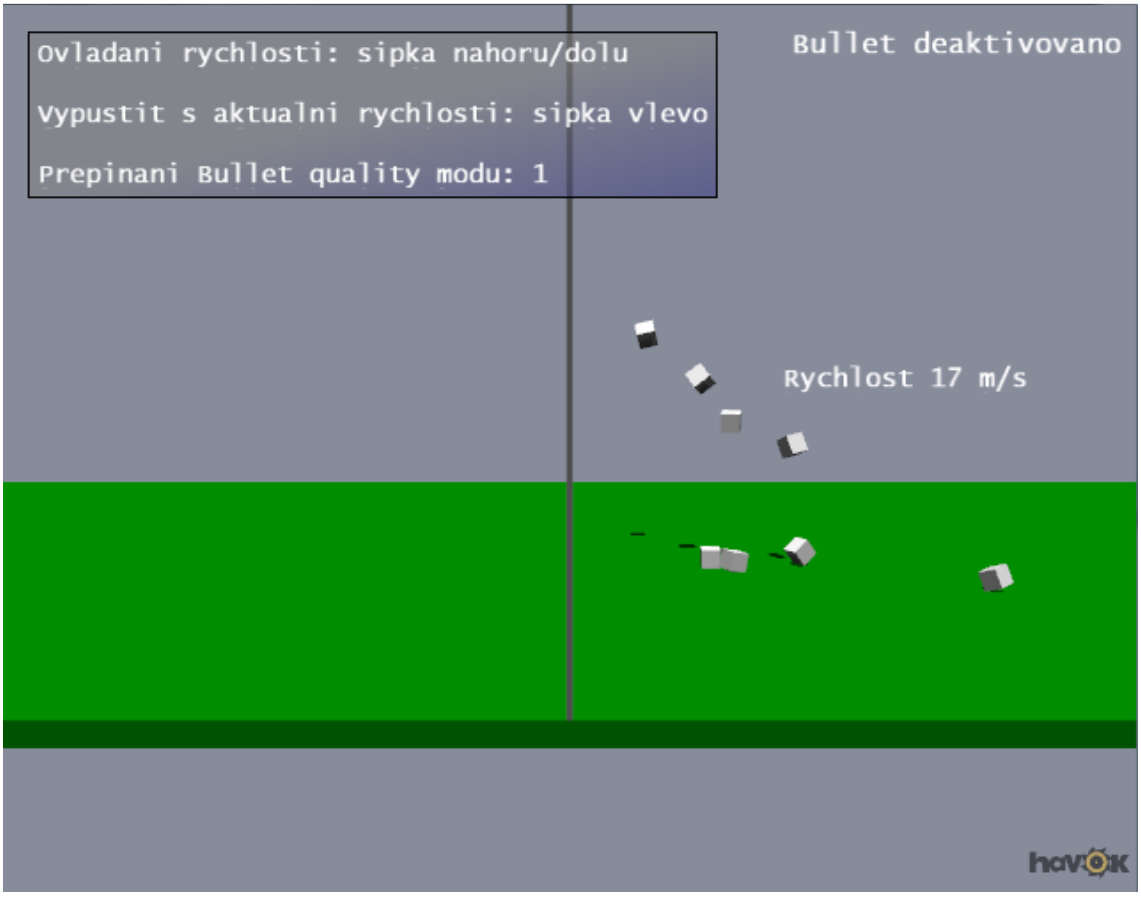
Jednoduchá základna, uprostřed které je svislá kolmá deska. Střílíme krychle (a=0.5m, m=1kg) ze vzdálenosti 0.5 - 2 metry (náhodně) kolmo proti desce. Šipkami nahoru a dolů regulujeme rychlost a zároveň střílíme krychle, šipkou vlevo vypouštíme krychle o právě nastavené konstantní rychlosti. Sledujeme při jaké rychlosti dochází k penetraci desky krychlí.

4.8.3 Výsledek

Systém se chová stabilně do 18 m/s. Při 19 m/s dochází k penetraci první kostky. Při rychlosti 32 m/s dochází k cca 50% penetrací. Rychlost 60m/s je už 70% a 100% nastává při 90 m/s.

4.8.4 Řešení

K vyřešení tohoto chování stačí nastavit Bullet quality mode u každé kostky. Po této úpravě jsem se dostal při 100% funkčnosti až na rychlost 350 m/s. Pro tuto rychlost je však nutné zvětšit maximální možnou lineární rychlost objektu, protože standardně je nastavena na 200 m/s. Bullet quality mód aktivujete v testu stisknutím klávesy 1.



Obr. 4.8 – 1 Ukázka scény testu 8.

5. Závěr

Nejprve jsem vytvořil tři ukázkové scény (Domino, Jumping Balls a Pool) spolu s jednou základní (pro začátečníky) pro implementaci Havok Physics do vlastních aplikací. Tato jednoduchá třída je jako jediná vytvořena mimo demo framework (GUI od Havoku) a to proto, aby byla možná implementovat na libovolné grafické rozhraní.
Dále jsem vytvořil set osmi testovacích scén, kde jsem v každé odhalil minimálně jednu chybu enginu Havok.
V prvním testu jsem se zabýval kolizí objektů o velmi rozdílných hmotnostech a rozměrech. Vzniklý problém jsem s pomocí vývojářů Havoku částečně vyřešil (bohužel ne zcela, v současnosti předávám moje kódy vývojářům pro podrobnější testování).
Druhý test zkoumá, jak Havok Physics pracuje s energií. V implementaci matematického kyvadla jsem zaznamenal útlum při každém kmitu, který se mně nepodařilo úplně eliminovat. Toto je druhý test, který je spolu s prvním stále v řešení s inženýry z Havoku. Zatím mně bylo řečeno, že je lepší aby systém energii postupně ztrácel, než aby se někde nerealisticky vytvářela (tak jako v reálném světě) a pracují na opravě problému.
Ve třetím testu zkoumám stabilitu osového systému spolu s velmi těžkými objekty. Vzniklý problém jsem úspěšně vyřešil navýšením přesnosti detekce kolizí na nadstandardní hodnotu.
Čtvrtý test se zabývá deformovatelností volného závěsu. Zkoumáme prodloužení závěsu vlivem velmi těžkého objektu. Tento test proběhl v pořádku, objevilo se pouze velmi nepatrné prodloužení (max. 0.00005%).
Pátý test zkoumá stabilitu osového a závěsového systému a hlavně jeho chování v návaznosti na velké rozdíly ve hmotnostech, včetně nekonečné hmotnosti pevné základny. Vzniklý problém jsem úspěšně vyřešil.
Šestý test se zabývá zachováním energie v návaznosti na restituční model a dále přesností detekce kolizí. V této scéně jsem objevil chybu, která byla po upozornění vývojářů Havok v nové verzi 6.5 úspěšně opravena (kolizní nepřesnost). Restituční model dosud stále dělá problémy, které jsem však takřka kompletně eliminoval. Bohužel ne úplně na 100% a tak je tento problém zaslán spolu s testem 1 a 2 vývojovému inženýrovi Havok k podrobnějšímu testování.

Scéna sedm testuje, jak si fyzikální enginy poradí s velkým množstvím kolizí tuhých těles a jejich vzájemným trvalým kontaktem. Vzniklý problém jsem částečně vyřešil. Po konzultaci se Seanem Thurstonem (Developer Support Engineer – Havok) jsem tento problém vyřešil úplně, ale bohužel ne zcela „čistým“ způsobem.

Test osm zkoumá, jakou si můžeme dovolit používat maximální rychlost, dokud se neobjeví problém „přeskočení momentu kolize“. Úpravou kvality výpočtu daného objektu jsem tuto rychlost zvýšil na takřka čtyřnásobek (350 m/s).

Troufám si tvrdit, že tyto scény jsou nastaveny tak obtížně, že budou dělat problémy i případným dalším testovaným fyzikálním enginům pro budoucí porovnání, kde vidím potenciál pro možné pokračování.

Tato bakalářská práce mně přinesla několik nových zkušeností. Mezi hlavními bych jmenoval zejména možnost pracovat s novou technologií, kterou nepoužívá nikdo z mého okolí. Díky tomu jsem byl většinu času odkázán pouze na uživatelskou dokumentaci (později i veřejné internetové Havok fórum) a byl nucen řešit vše naprosto samostatně. Dále bych vyzdvihl spolupráci přímo s vývojovými inženýry Havok, kteří mně pomohli s mnoha komplikovanými problémy v závěrečné testovací části. Zároveň jsem tím částečně přispěl k odhalení a dokonce už i opravení některých chyb.

Použitá literatura

[1] Havok [online]. 2009. Dostupné na http://www.havok.com/ [cit. 2009-01-28]

[2] TryHavok Discussion Board [online] Dostupné na: http://software.intel.com/en-us/forums/havok/ [cit. 2009-01-28]

[3] D. H. Eberly: Game physics. San Francisco. 2004. ISBN 1-55860-740-4

[4] Product Review of Physics Engines, Part One: The Stress Tests [online], Dostupné na http://www.gamasutra.com/features/20000913/lander_01.htm [cit. 2009-01-28]

[5] A. Seugling, M. Rolin: Evaluation of Physics Engines and Implementation of a Physics Module in a 3d-Authoring Tool. Diplomová práce. 2006. Dostupné na http://www.cs.umu.se/education/examina/Rapporter/SeuglingRolin.pdf [cit. 2009-01-28]