Rymdgymnasiet

Gravitation

Och dess lag simulerad

Karl Andersson

GYMNASIEARBETE

KIRUNA 2016

Handledare: Camilla Nilsson

# Sammanfattning

# Innehållsförteckning

# Inledning

1.1 Introduktion

I ett samhälle som blir allt mer beroende av teknologin både i privatlivet som arbetslivet

Jag gör simuleringen för gymnasiearbetet med inriktningen teknik och fysik.

1.2 Syfte

Syftet med programmet är att simulera gravitationen med hjälp utav Newtons gravitationslag och sedan visualisera det på datorskärmen.

1.3 Frågeställningar

1. Går det att upptäcka några liknelser mellan simuleringen och universum?
2. Vad skulle kunna simuleringen bättre?

# Bakgrund

# Material

3.1 Programspråk

Ett programspråk är ett språk där man ger instruktioner till datorn med hjälp av text något som vi människor har lätt att förstå. Texten kompilerar (översätts) sedan till maskinkod (binärt), något som t.ex. en dator förstår men vi människor har svårare för, det går men är inte optimalt för effektivitet.

Som programspråk valde jag C++ p.g.a. de kunskaper jag redan har över det och att det är nära besläktad med programspråket C som brukas räknas som modern till alla de nu moderna språken. C är dessutom väldigt kraftig över hur nära den bestämmer maskinens processer och är ett utav det vanligaste språken som man använder när man programmerar en maskin, men även också datorer, fast där brukar C++ och JAVA ligga över de andra språken, vilket också är en anledning jag valt C++ då den förekommer vanligen.

3.2 IDE (Intergrated Development Environment)

IDE är där du skriver programmet som brukar bestå av en textredigerare, kompilator och debugger.

Jag valde Microsoft Visual Studio som mitt IDE då de först av allt har en gratis version. IDE:n är lätt att använda och kräver inte så mycket avancerade kunskaper för att komma igång. Nackdelen med denna IDE:n är att den inte är den bästa för multiplattform och det faktumet att man inte kan ändra kompilator.

3.3 Grafik Bibliotek

Ett grafik bibliotek är ett program bibliotek med optimerade funktioner som hjälper en att kommunicera med sitt grafikkort och på så sätt rita på sin skärm.

Jag har valt OpenGL, främst för de insamlade kunskaper jag redan har utav det men annars du det är brett stött av flera grafikkort och stödjer multiplatform. OpenGL står för Open Graphics Language, är gratis och uppdateras regelbundet via grafikkorts drivrutiner. OpenGL är ett av de största grafik biblioteken bredvid DirectX som är Microsofts eget grafik bibliotek och stödjer inte multiplatform. OpenGL har flera versioner med den senaste versionen just nu 4.5. Jag har valt versionen 3.3 då den är tillräckligt modern och stöds just nu av de flesta grafikkorten och är den version jag är mest van vid.

För att kunna programmera enklare med OpenGL har jag tagit hjälp av ett bibliotek som kallas GLEW (OpenGL Extension Wrangler Library) som hjälper en med bl.a. optimering för ens system och operativsystem.

3.4 Övriga Bibliotek

Övriga bibliotek som jag använde var GLFW och GLM. GLFW står för OpenGL Framework och är det bibliotek som läser in data från tangentbord och mus och är essentiellt om man ska kunna interagera med programmet under den tid det körs. GLFW är även ansvarig för att skapa ett fönster som är viktigt för att kunna använda OpenGL då när man ska rita så måste det ritas på ett fönster, vilket är hur operativsystem funkar (Windows).

GLM används vid lite mer avancerad matematik och används främst inom grafik programmering för dess vektorer och matriser. GLM står för OpenGL Mathematics.

Förutom de externa biblioteken som måste hämtas från nätet använder jag mig av flertalet interna bibliotek såsom <iostream> som använder kommandotolken för att ta in och ge ut data i form av text och siffror, används främst till debugging. <stdlib> som jag använde främst för funktionen rand som genererar ett slumpat tal. <ctime> för funktionen time som är ger ett värde räknat i sekunder sedan första januari 1970 UTC vilket är användbart då funktionen rand använder ett så kallat seed för att få fram sitt ”slumpade tal” och om inte seedet ändras så blir det slumpade värdet densamma varje gång då programmet körs, men om man använder time funktionen som byter värde varje sekund så kommer det slumpade värdet variera varje gång. Även <algorithm> används för algoritmer och <vector> för att lagra liknande information i flera mängder så som massa till olika objekt.

# Metod

4.1 Shader

En shader är ett program som gör skuggning och skrivs i programspråket GLSL som är redan för installerat med OpenGL. GLSL, OpenGL Shading Language, är ett språk likt C men används för att ge instruktioner till grafikkortet om hur den ska gå tillväga för att skriva på skärmen. Det finns tre olika shaders i OpenGL; Vertex, Fragment och Geometry shader, varav de två första är obligatoriska för att kunna skriva något överhuvudtaget.

OpenGL använder än fast väg för att kunna skriva ut något på skärmen som går ungefär så här: Vertex Shader -> Shape Assembly -> Geometry Shader -> Rasterization -> Fragment Shader -> Tests and Blending.

Vertex shadern kan kort beskrivas med att den tar emot en mängd angivna tre dimensionella positioner och med angivna instruktioner har möjlighet att göra något speciellt med dem men i vårt fall har vi bara skickat vidare dem till Shape Assembly som monterar ihop dem beroende på hur vi valt att de ska monteras t.ex. som trianglar. Geometry shader är frivillig men har möjligheter att kunna skapa nya former som t.ex. två trianglar från en. Efter det så rastreras formen, den med andra ord pixeleras och skickas sedan till fragment shader där vi kan föra lite allt möjligt men kan man sammanfatta att vi ger vår form en eller flera färger. Sedan testas det om vår form ska blandas med eventuella andra former om t.ex. om en är transparent och är framför den andra.

4.2 Globalt

Det globala fältet är det fält som inte involverad i någon funktion och är tillgängligt för alla inom filen.

Förutom de bibliotek som var tvungna att inkluderas, inkluderade jag en annan C++ fil vars syfte är att öppna de shader filerna som behövs och läsa in dem till vårt grundprogram. Dessutom finns det några globala variabler så som höjden och längden på fönstret och en pekare som pekar mot det fönstret som skapas senare. Det finns även en funktion som ger ett värde för den relativa mus hjulpositionen som rings varje gång den får input från användaren.

I det globala fältet finns även strukturen Point som innehåller viktig data som varje partikel kommer att erhålla. Datan innehåller 4 variabler och två vektorer: mass, massgain, density, radius, position och velocity.

4.3 Initialisering

Inne i main funktionen börjar programmet med att initialisera GLFW och skapar ett fönster med den givna dimensionen i det globala fältet (standard: 1280x720), därefter initialiseras GLEW. En visuell oktogon konstrueras utav åtta trianglar med en array som ställer upp värden efter varandra, det finns alltså åtta tringlar och varje triangel har tre punkter och varje punkt har tre dimensioner även fastän vi bara använder två.

En vertex array object och vertexbufferobject genereras

4.4 Programslinga

# Resultat

# Diskussion och Slutsatser

# Källförteckning