Asteroid Dynamics

New Mexico
Supercomputing Challenge
Final Report
April 5, 2006

Team #89
Santa Fe High School

Team Members:
Luwen Huang

Teacher:
Anita Gerlach

Project Mentor:
Charles McClenahan, Ph.D

Table of Contents:

Executive Summary:

In Earth's history, stretching over billions of years, asteroids have collided with Earth, leaving deep impact craters and destruction as evidence. Hundreds of thousands more asteroids orbit in the Main Belt, dangerously close to Earth. Today, scientists know that the question is not if, but when an asteroid will impact, and are still trying to accurately predict the event. This *n*-Body[1] task is complex, involving thousands of objects with many factors, such as gravitation and energy, often requiring supercomputing. I present a unique approach that will better this prediction process, for the sake of catastrophe prevention and scientific interest. This project attempts to calculate the trajectory of any given asteroid deviated from standard orbit and the likelihood of this event.

To accurately create this visual program, I needed a language with graphical and powerful numerical and object-oriented properties. Therefore, I chose Microsoft Visual Studio C++ Express. It provided all of the projects requirements and allowed for more structural organization and graphical expansion using OpenGL.

The program uses a unique conservative integration scheme in which it integrates over a transformed time step polynomial. This mathematical transformation allows for exact conservation of energy and momentum. For every asteroid, the program defines and sets variables in the Hamiltonian equations, integrates over a time step and calculates the properties of that asteroid. It then projects the asteroids onto a dynamic graphics interface.

The program accurately modeled real life and gave some interesting solutions. It indeed matched real world statistics and predicted asteroid close calls. As expected, perturbed asteroids were rare but not implausible, and certainly plausible to cross Earth's orbit. Since the results closely matched real world statistics, the program can certainly be used as an accurate probability predictor for asteroid collisions, a major achievement.

Introduction:

The *n*-Body problem has many applications, ranging from kinetic theory problems in chemistry to modeling hundreds of thousands asteroids in the Main Belt. The groundwork for the *n*-Body problem is pure mathematics, yet mathematics so complex in calculation that, for centuries, the problem was not applied to any real world situations. Recent developments in supercomputing, however, have allowed for the application of *n*-Body mathematics. In choosing a specific application, I considered the complexity of each scenario and how well it could be suited to the new, conservative algorithm. Deciding on the case of asteroids in the Main Belt for its complexity, ideal[1] space environment, and most importantly, its modern appeal and danger, I set out to predict and model the thousands of asteroids orbiting in the Main Belt. The goal is to calculate the likelihood of any asteroid deviating from standard orbit, impacting Earth, **and** its trajectory.

The Main Belt, home of hundreds of thousands of asteroids, is situated between Mars and Jupiter. On a regular basis, through gravitational interactions, asteroids are driven from their standard orbits and travel a different trajectory, sometimes nearing Earth. Asteroids vary greatly in size, from Ceres the largest, comprising almost 50% of all the mass in the belt, to most asteroids the size of specks of dust. For realistic purposes, the program utilizes 1000 asteroids. Most asteroids in the Main Belt are the size of dust particles. Also, the threshold size for an asteroid not burning up in the atmosphere is 100 m wide, minimum. Finally, with 1000 asteroids, the program can be easily checked for accuracy and quickly run.

The *n*-Body problem for this situation has traditionally been solved through numerical integration schemes, polynomials involving time-steps. The key point is that the system of asteroids must conserve energy and momentum, or the system will cease to be one and fall apart. It is known that numerical integration algorithms lead to drifts in accuracy. Even symplectic[2] integration, a popular Hamiltonian method, loses energy over time. It is with this new conservative integration that I conserve the true Hamiltonian and momentum of these asteroids. With this algorithm, I exactly solve this system and give

the trajectory and probability of any deviation from orbit. This contributes to determining the likelihood of asteroid impacts in a certain time frame in the future.

Method:

This program strives to provide a wholly accurate model and calculations of the asteroids in the Main Belt. It utilizes a unique conservative integration scheme which has not been applied to a real life *n*-Body problem until now. Therefore, this program is structured around the algorithm's specific mathematics. To explain fully the math would be a lengthy interruption, so only the goal and point of the complicated math is explained.

The Hamiltonian describes a system by stating its total energy, kinetic and potential. It is a complex partial derivative that is commonly used to solve *n*-Body problems through symplectic integration. But, this is not exact integration because the time step ("integration rectangle") is not infinitesimal, but finite. The only way to achieve an exact integration would be to transform the Hamiltonian into a linear function, so that the integration is simply in a form similar to distance equals velocity times time. This is achieved through defining two different sets of variables and transforming them. Once the integration is done, the variables are transformed back to the original state, fit to be displayed. The total energy is now known, and the other properties are calculated from this. The equations are simplistic in nature; potential energy, kinetic energy, rotational kinetic energy, etc., but the process of transformation and integration is complex.

This program is organized by classes that call from written functions, and event handlers that call from constructors in these classes. It is built upon 7 classes, namely, Phase Space, CM Rewrite, Vector Rewrite, Phase Space Transformation, Integration, Back Vector Transformation, and Back Phase Space Transformation. These classes further draw from source code files that contain the methods. Phase Space defines the asteroid properties radius, angle, mass, momentum in the radial direction, and angular momentum. It also includes the call to display the dynamic asteroids. The class also includes random number generators to randomly pick initial conditions and properties of the asteroids. This random choosing of initial conditions is repeated with each simulation, ensuring that results are not drastically affected by initial conditions. A noteworthy point

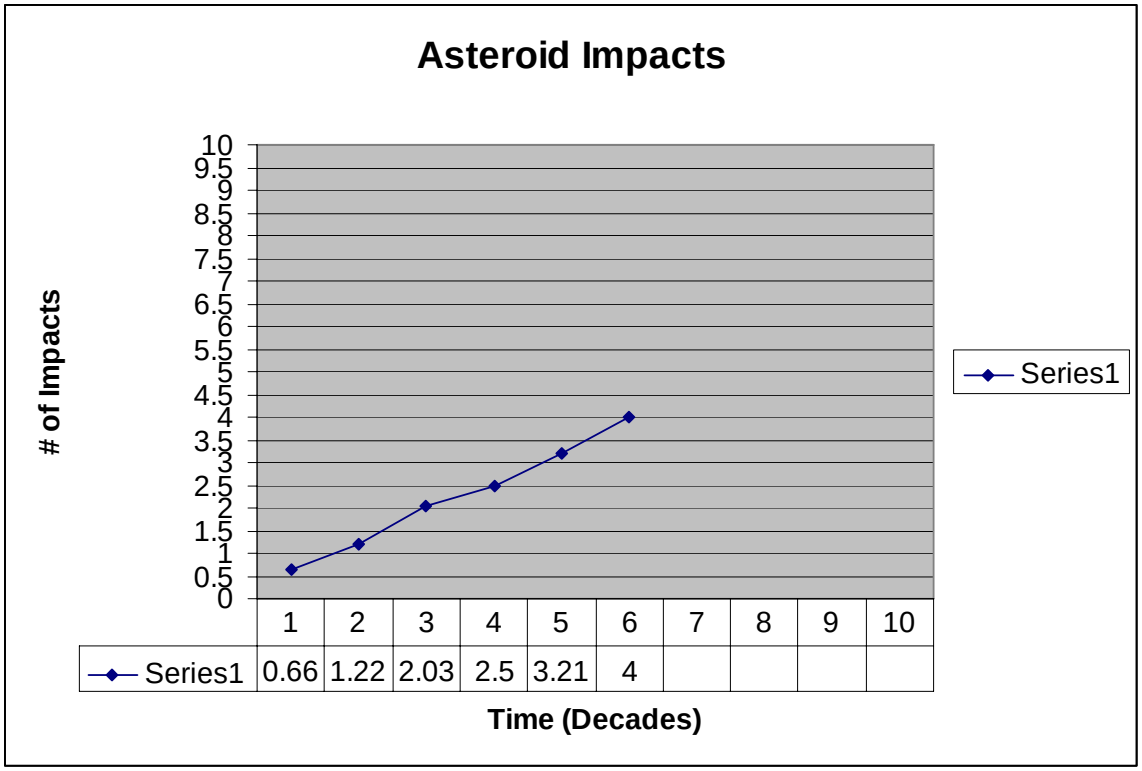
is that even though the conditions do not resemble that of the Main Belt, integration over the time steps narrow the differences, and provides accurate probabilities of asteroid impacts. CM Rewrite and Vector Rewrite are classes that define the center of mass as the origin and radius vectors of the *i*th asteroid. Phase Space Transformation defines a new set of variables that turn the Hamiltonian into a linear function by assigning new variables to quadratic components. The last two classes transform the variables back by inverting the steps taken to transform them and thus converting them to their original, displayable form. A loop is constructed so that after a display, the program immediately goes back to Hamiltonian linear transformation for a new energy value, a new time step. This continues until the user chooses to end the loop.

To check the conservative algorithm, the program calculates the angular momentum of any given asteroid with the basic mechanics equations.

The program naturally has a GUI that displays a dynamic viewing screen of the changes in the moving bodies. It is created so that it is an actual application that accepts user input, such as momentum checking. The GUI is very user-friendly and easily understood.

Results:

The program performed well to expectations. For the 1000 asteroids, it conserved momentum, matched statistics and gave interesting solutions. The program ran for many short, time efficient runs to make sure it conserved momentum before simulating a 60 year period. The short runs were necessary since I had to make sure the program was accurate before embarking upon a long iteration. This 60 year period was the peak of the simulation in which it iterated over many small time steps and displayed a dynamic screen.



Graphed in Microsoft Excel. Time Intervals are adjusted on a large scale to provide for possible graphing -- time steps are significantly smaller and values are rounded. This is an average graph of all data collected during 60 year iteration to summarize results.

This provided satisfying results. I found that 4 explosion-worthy but not globally dangerous asteroids hit Earth. These asteroids impacted Earth and as expected, did not push it out of orbit, as common sense would dictate. This is one of the many indications that the 60 year iteration was correct. I found no global scale impacts, consistent with the rarity of such events. Near the end of the run, the program showed slight but not continuous deviations in some asteroids. The asteroids shifted slightly out of their normal orbits but returned to a stable orbit. These slight perturbations could be remote possibilities of asteroid impacts in the future. The program recorded a wealth of data, graphs, charts, etc. Please see appendixes for an important note.

Conclusions:

The results are consistent with recorded statistics. A small explosion-sized asteroid hits Earth every 1-2 years, usually in the remote Siberian lands. Also, global scale collisions occur about once every 1000 centuries, so it is expected that the program will not find one in 60 years. The project also showed possibilities of more deviations near the end of 60 years. NASA predicts a slim chance of an asteroid impact in the next 40-60 years, coinciding with this program's prediction. There are many checks made on the accuracy of this program, including the momentum checker, consistency with known statistics, and common sense, such as the fact that the planets and the sun behaved within normal parameters. Though this program is generated from initial conditions, integration over the time steps evens out discrepancies, making plausible solutions.

Recommendations:

To ever more perfect this program, more asteroids need to be added. Eventually, the perfect model would include the hundreds of thousands of asteroids in the Main Belt, ranging from the size of small planets to specks of dust. With more asteroids, realistic results could improve, but the program may break down with lack of conservation of momentum. Also, integration over a larger time would be interesting as it may give a solution for the destruction of Earth when a very large asteroid hits. It can also provide further checks for conservation of momentum. Expanding it to 3-D will of course be more accurate and appealing, but such expansions and the above said will take significantly more processing time.

Achievements:

This project demonstrated an accurate conservative integration algorithm with the Hamiltonian. It applied a purely mathematical method to a real-life situation, demonstrating how the power of computing can be used for scientific interest and public concern. Exact conservative integration can now be implemented widely, replacing time-consuming and inaccurate symplectic schemes. This program adds to and confirms the probabilities of asteroid impacts predicted by NASA. Also, the program records useful

data and has a user-friendly GUI, appealing to the general public. As this is an active area of observation and research, this project contributes to the scientific community and provides interesting predictions of when asteroids will collide with Earth.

Acknowledgements:

I would like to thank first and foremost my project mentor Dr. McClenahan. This project would not be without his guidance. Also, I thank also Supercomputing Challenge judge Tom Laub and my teacher Anita Gerlach for their helpful suggestions and willingness to help me whenever I am in need.

Bibliography:

Books

Marion, Jerry B., and Stephen T. Thornton. *Classical Dynamics of Particles and Systems.*

Tipler, Paul. *Physics for Scientists and Engineers*. 7th Edition.

Deitel, Harvey. *Visual C++ Programming*.

Scholarly Articles

Kotovych, Oksana., and John C. Bowman. *An Exactly Conservative Integrator for the n-Body Problem*.

Websites

www.wikipedia.org

http://neat.jpl.nasa.gov/neofaq.html

www.nasa.org

http://nssdc.gsfc.nasa.gov/planetary/factsheet/asteroidfact.html

Software

Microsoft Visual C++ Express Edition

Appendixes:

*Note: **All** data spanning over **all** iterations too large to include.

*Note: Entire code not possible to include. File too large, including all cpps, headers, forms, and dlls.

These missing components will be presented along with the executable program in the Exposition.

Form:

```
pragma once

#include "asteroid.h"

namespace MyProject {

	using namespace System;
	using namespace System::ComponentModel;
	using namespace System::Collections;
	using namespace System::Windows::Forms;
	using namespace System::Data;
	using namespace System::Drawing;

	/// <summary>
	/// Summary for Form1
	///
	/// WARNING: If you change the name of this class, you will need
to change the
	///          'Resource File Name' property for the managed
resource compiler tool
	///          associated with all .resx files this class depends
on.  Otherwise,
	///          the designers will not be able to interact properly
with localized
	///          resources associated with this form.
	/// </summary>
	public ref class Form1 : public System::Windows::Forms::Form
	{
	public:
		Form1(void)
		{
			InitializeComponent();
			//
			//TODO: Add the constructor code here
			//
		}
```

```
	protected:
		/// <summary>
		/// Clean up any resources being used.
		/// </summary>
		~Form1()
		{
			if (components)
			{
				delete components;
			}
		}
	private: System::Windows::Forms::Button^  GoButton;
	protected:

	private:
		/// <summary>
		/// Required designer variable.
		/// </summary>
		System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
		/// <summary>
		/// Required method for Designer support - do not modify
		/// the contents of this method with the code editor.
		/// </summary>
		void InitializeComponent(void)
		{
			this->GoButton = (gcnew
System::Windows::Forms::Button());
			this->SuspendLayout();
			//
			// GoButton
			//
			this->GoButton->Location = System::Drawing::Point(98,
231);
			this->GoButton->Name = L"GoButton";
			this->GoButton->Size = System::Drawing::Size(75, 23);
			this->GoButton->TabIndex = 0;
			this->GoButton->Text = L"Go";
			this->GoButton->UseVisualStyleBackColor = true;
			this->GoButton->Click += gcnew
System::EventHandler(this, &Form1::GoButton_Click);
			//
			// Form1
			//
			this->AutoScaleDimensions = System::Drawing::SizeF(6,
13);
			this->AutoScaleMode =
System::Windows::Forms::AutoScaleMode::Font;
			this->ClientSize = System::Drawing::Size(292, 266);
			this->Controls->Add(this->GoButton);
			this->Name = L"Form1";
			this->Text = L"Form1";
			this->ResumeLayout(false);

		}
#pragma endregion
```

```
      private: System::Void GoButton_Click(System::Object^  sender,
System::EventArgs^  e)
                  {
                        //For Earth, r = 1 AU
                        double r4 = 23;        //radius in pixels
                        double theta4 = Math::PI/6;

                        double x4 = r4*Math::Cos(theta4);
                        double y4 = r4*Math::Sin(theta4)*(-1);

                        //For Mars, r = 1.5 AU
                        double r2 = 34.5;            //radius in
pixels
                        double theta2 = Math::PI/6;

                        double x2 = r2*Math::Cos(theta2);
                        double y2 = r2*Math::Sin(theta2)*(-1);

                        //For Jupiter, r = 5.2 AU
                        double r3 = 120;
                        double theta3 = Math::PI/6;

                        double x3 = r3*Math::Cos(theta3);
                        double y3 = r3*Math::Sin(theta3)*(-1);

                        Pen ^myPen = gcnew
Pen(System::Drawing::Color::Black);
                        Graphics ^myGraphics = this->CreateGraphics();

                        //translate origin to (40,40)
                        myGraphics->TranslateTransform(140,120);


                        //sun
                        myGraphics->DrawEllipse(myPen,0,0,20,20);

                        //earth
                        myGraphics->DrawEllipse(myPen,x4,y4,8,8);

                        //mars
                        myGraphics->DrawEllipse(myPen,x2,y2,8,8);

                        //Jupiter
                        myGraphics->DrawEllipse(myPen,x3,y3,10,10);

                        for (int i = 1; i<1000; i++)
                        {
                              new asteroid();
                              void display();
                              new CMtransform();
                              new VectorTransform();

                              if (i = 2, i<1000, i++)
                              {
                                    new LinearTransform();
                                    new Integration();
                                    new BT();
```

```
                                    new BTDisplay();

                                }
                            }



                        }
        };
}
```

Header:

```
        public class asteroid
{
public:
        asteroid();
        void display();         //draws

private:
        double r,
                 theta,
                 pr,
                 ptheta,
                 mass;
    class ^asteroid next;
};

public class CMtransform
{
public:
        CMtransform();

};


public class VectorTransform
{
public:
        VectorTransform();

};

public class LinearTransform
{
public:
        LinearTransform();

};

public class Integration
{
public:
```

```
      Integration();

};

public class BTransform
{
public:
      BTransform();

};

public class BTdisplay
{
public:
      BTdisplay();

};
```

Source:

```
asteroid::asteroid()
{
      Random ^randomObject = gcnew Random();

      r = randomObject->Next(53,76);
      theta = randomObject->Next(0,2*Math::PI);
      double randomMass = randomObject->Next(950*Math::Pow(10,18));

}

void asteroid::display()
{
      Pen ^myPen = gcnew Pen(System::Drawing::Color::Black);
      Graphics ^myGraphics;
      myGraphics->TranslateTransform(140,120);  //translate origin

      double x = r*Math::Cos(theta);
      double y = r*Math::Sin(theta);
      myGraphics->DrawEllipse(myPen,x,y,20,20);
}
+ user protected code □
```