QUICK EVALUATION OF SMALL BODY GRAVITATION

BY

ANDREW C. COLOMBI

B.S., Bucknell University, 2002

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2008

Urbana, Illinois

Doctoral Committee:

Assistant Professor Anil N. Hirani, Chair
Assistant Professor Benjamin F. Villac, U. California, Irvine
Professor John C. Hart
Professor Michael T. Heath
Abstract

Numerical integration of ordinary differential equations resulting from the gravitation of nearby celestial small bodies is the subject of this thesis. We present three methods that alleviate the computational burden of evaluating gravitational force near a small body: i) adaptive polynomial interpolation, ii) adaptive polynomial least squares approximation, and iii) acceleration via specialized, commodity hardware. Each method is evaluated on its quantitative accuracy with respect to a reference model, and its observance of qualitative features of gravity. We conclude with a summary of methods available for computing small body gravitation, and recommendations for different scenarios.
To my father and mother,
For the gift of opportunity.
I would like to acknowledge my advisor, Anil Hirani, for his guidance throughout my research. I am equally indebted to Benjamin Villac; whose knowledge of astrodynamics has been an invaluable resource.

To all the faculty in the Scientific Computing group: Thank you for your tutelage.

I would also like to acknowledge my colleagues and friends: Nathan Bell for Octrees, Keenan Crane for GPUs, Jared Hoberock for BVHs, and Mike Katelman for BBQs.

Finally, my research would not have been possible without the Computational Science and Engineering (CSE) program’s Turing compute cluster.
# Table of Contents

List of Symbols .................................................. viii

Chapter 1 Introduction .......................................... 1
  1.1 Introduction .................................................. 1
  1.2 Overview ..................................................... 2
  1.3 Summary of Contributions ................................... 3

Chapter 2 Background .............................................. 4
  2.1 Mathematical Properties of Gravitation .................... 4
    2.1.1 Modeling Small Body Gravitation ....................... 5
  2.2 Previous Work: Small Body Gravitation ..................... 6
    2.2.1 Mascon Method ........................................... 6
    2.2.2 Polyhedral Method ....................................... 7
    2.2.3 Modified Polyhedral Method ............................ 9
  2.3 Previous Work: Large Body Gravitation .................... 9
    2.3.1 Spherical Harmonics .................................... 9
    2.3.2 Polynomial Approximation ............................. 11

Chapter 3 Adaptive Approximation by Polynomial Interpolation 13
  3.1 Polynomial Force Interpolation Near Irregular Bodies .... 13
    3.1.1 Interpolation Scheme ................................... 13
    3.1.2 Numerical Experiments .................................. 15
  3.2 Adaptive Spatial Partitioning ................................ 24
    3.2.1 Octrees .................................................. 25
    3.2.2 Spherical Harmonics Far Away ........................ 27
  3.3 Example Octree Construction and Performance ............... 27
    3.3.1 Octree Model Construction and Error Analysis ......... 28
    3.3.2 Single Evaluation Speed Tests ........................ 28
  3.4 Performance Analysis Using Trajectory Integrations ....... 31
    3.4.1 Experiment Design ....................................... 31
    3.4.2 Close Retrograde Orbits ................................ 32
    3.4.3 Mid-Range Orbits ....................................... 33
    3.4.4 Random Trajectories .................................... 34
    3.4.5 Ejecta Trajectory ....................................... 35
List of Symbols

General

$i, j$ An index to a collection; e.g. B-splines $B_{j,k,\tau}$ or points $\mathbf{x}_i$.

$x$ A point in $K$ dimension.

$U$ The true gravitational potential.

$\hat{U}$ An approximation to gravitational potential.

$F$ The true gravitational force acting on a unit point-mass.

$\hat{F}$ An approximation to gravitational force acting on a unit point-mass.

$G$ Universal gravitational constant, taken as $6.6742 \times 10^{-11} \text{m}^3\text{kg}^{-1}\text{s}^{-2}$ for all experiments.

$\rho(\mathbf{x}), \rho$ Mass density.

$\nabla$ Gradient operator.

$O$ Big-O notation describing run times of algorithms.

Chapter 2

$\Omega$ A volume representing a small body, e.g. asteroid.

$\text{div}$ Divergence operator.

$\Delta$ Laplace operator: $\text{div} \cdot \nabla$. 
Chapter 3

\( k \)  The degree of a polynomial, and consequently, order of interpolation used in a cell.

\( \ell_k \)  A polynomial, the zeros of which define \( k+1 \) abscissa of Gauss-Lobatto-Legendre interpolation.

\( P_k \)  The Legendre polynomial of degree \( k \).

\( \mathcal{N} \)  A collection of interpolation abscissa.

\( \delta_{ij} \)  The Kronecker delta.

\( p_{\mathcal{N}}^{x_i} \)  A Lagrange polynomial over the set \( \mathcal{N} \).

\( x, y, z \)  The cartesian coordinates of a point in 3 dimensions.

\( f \)  A function to approximate.

\( \mathcal{P}, \mathcal{D} \)  Power spectral density, and normalized power spectral density.

Chapter 4

\( n \)  An index of iterations of refinement.

\( k \)  The degree of a B-spline. And, briefly an index of functions in section 4.4.2.

\( \tau \)  A long knot vector.

\( \sigma \)  A small knot vector used to precisely define a single B-spline.

\( B_{j,n,\tau} \)  An element of the B-spline basis formed with knot vector \( \tau \).

\( B_\sigma \)  A single B-spline function of order \(|\sigma| - 2\) using the knots of \( \sigma \).

\( \mathbb{P}_{n,\tau} \)  The space spanned by a B-spline basis using the knot vector \( \tau \).

\( S \)  A vector space of functions.

\( S_n \)  The vector space of functions after \( n \) iterations of refinement.

\( \mathcal{B} \)  A spanning set for the vector space \( S \).

\( \mathcal{B}_n \)  A spanning set for the vector space \( S_n \).

\( \phi \)  An element of the spanning set \( \mathcal{B} \), also known as a basis function.
support of a function.

\[ C(\phi) \] The children of \( \phi \).

\[ \Omega \] The domain of the approximation.

\[ T^p \] The \( p \)th level of refinement within \( B_n \).

\[ R^p \] The set of spanning functions in \( T^p \) that are refinements of those in \( T^{p-1} \).

\[ N^p \] The set of spanning functions in \( T^p \) that are not refinements of those in \( T^{p-1} \).

\[ M, \, \tilde{M}, \, \bar{M} \] Mass matrices.

\[ b, \, \tilde{b}, \, \bar{b} \] Right hand side of a linear system.

\[ x_n \] Coefficients from iteration \( n \).

\[ P \] A patch of B-splines.

**Chapter 5**

\[ A \] An array.

\[ N \] Number of elements in the array.

\( A_{idx}, B_{idx} \) Indexes of an array.
Chapter 1

Introduction

1.1 Introduction

A common task in simulation of natural phenomena is the integration of Ordinary Differential Equations (ODE) resulting from harmonic scalar fields. The dynamics of such fields, which include gravitation and electromagnetism, are described by a vector field derived from potential energy. Simulation of free particles under the influence of these vector fields is the subject of this thesis. Specifically, we investigate efficient methods of evaluating these vector fields that are suitable for ODE integration.

Motivating this research is the computational complexity of computing these vector fields when the field’s source does not fit an idealized shape. For example, the gravitational force of a point mass and the electrostatic force due to an “infinite plane” are both captured by simple equations. The force due to an oddly shaped asteroid, however, is not so simply captured. In this case current techniques require extensive computation, which is not well suited to the kind of analysis required before exploring such asteroids with spacecraft.

To illustrate how unwieldy these simulations can be consider mission design for spacecraft visiting a small body (e.g. asteroid, comet, or other massive spacecraft). A typical task during mission design is to explore the dynamics of a trajectory using Monte Carlo analysis. The designer starts by plotting a “nominal” trajectory using idealized initial conditions (i.e. pretending as if they know the exact start state of the craft). This is conventionally done using a high order embedded Runge-Kutta method such as Runge-Kutta Prince-Dormand. Then thousands of trajectories with nearby starting conditions are also generated and compared with the nominal trajectory. The purpose of this analysis is to determine how sensitive the nominal trajectory is to perturbation, and to predict worst case scenarios should the
spacecraft drift during a mission. Putting numbers to the process reveals how cumbersome it can be:

1. one phase of Monte Carlo analysis uses, minimally, 10000 perturbed trajectories,
2. one trajectory requires approximately 1000 integration steps,
3. one step (of Runge-Kutta Prince-Dormand) requires 14 evaluations of gravity,
4. one evaluation of gravitation for a typical asteroid takes 0.005s.

In sum, it requires approximately \textit{8 CPU days for one round of Monte Carlo analysis}.

Simulations of this sort are becoming more and more frequent as interest in small bodies blossoms. There are presently four active missions to small bodies: DAWN, ROSETTA, Hayabusa and New Horizons; and an equal number that have already completed their mission: Stardust, NEAR SHOE-MAKER, DEEP SPACE 1 and Deep Impact. As our missions become more aggressive (Hayabusa and ROSETTA involve \textit{landing} on their targets) the simulations become more extensive. Compounding all this is that as simulation demands grow so too do the models of our asteroids. The recently imaged Toutatis asteroid, the largest model yet, is $3 \times$ larger than the asteroid in our studies, 1998 ML14 [28]. Put differently, Toutatis would require approximately 24 CPU days for one round of Monte Carlo analysis.

Twenty-four days is a long time to wait.

To produce efficient gravitational models two approaches are considered: adaptive function approximation and massively parallel Graphics Processing Unit (GPU) implementations. The work described herein focuses solely on gravitation; however, the techniques discussed are applicable to electromagnetism as well.

\subsection{1.2 Overview}

Chapter 2 introduces the gravitation, gravitation’s mathematical properties, and summarizes the state-of-the-art for computing gravitation of small bodies. Relevant methods for large bodies (e.g. planets and planetoids) are also presented.
Chapter 3 begins our contributions by introducing an adaptive polynomial approximation to gravitational force of a small body. Chapter 4 develops a more sophisticated algorithm that offers several qualitative improvements over that of chapter 3.

Chapter 5 introduces the GPU as a computational tool. Section 5.2.1 develops a GPU algorithm for close encounters to a small body, and section 5.2.2 develops an algorithm for distant interactions.

Finally, chapter 6 gives concluding remarks, and future work.

### 1.3 Summary of Contributions

The contributions of this research are itemized below:

**Function Approximation:**

- Introduction of polynomial approximation for small body gravitation (Chapter 3).
- Development of a general use, order $n$, $C^{n-1}$, adaptive function approximation (Chapter 4).

**GPU applications for small body missions:**

- Development of GPU algorithm for small body gravitation (Chapter 5).
- Development of GPU algorithm for distant small body gravitation (Chapter 5).
Chapter 2

Background

2.1 Mathematical Properties of Gravitation

As the subject of this thesis is numerical models of small body gravitation, some time reviewing the fundamental features of gravity are well spent. Gravitational potential $U : \mathbb{R}^3 \to \mathbb{R}$ is a scalar field satisfying Poisson’s equation,

$$\Delta U(x) = 4\pi G \rho(x), \quad (2.1)$$

where $\rho(x)$ is the mass density, and $G$ is the gravitational constant. As such, anywhere away from mass (i.e. where $\rho(x) = 0$) gravitational potential follows Laplace’s equation,

$$\Delta U(x) = 0.$$

The gravitational potential of a unit point-mass centered at the origin is $U(x) = \frac{G}{|x|}$, and the potential due to an arbitrary mass $\Omega$ is given by,

$$U(x) = -G \int_\Omega \frac{\rho(y)}{|x-y|} dy. \quad (2.2)$$

Gravitational force is defined by the negated gradient of potential, $F(x) = -\nabla U(x)$. By applying this relation it follows that the force due to a unit point-mass centered at the origin is $F(x) = -G \frac{x}{|x|^3}$, and the force of gravity due to an arbitrary mass is,

$$F(x) = -G \int_\Omega \frac{\rho(y)}{|x-y|^3} (x-y) dy. \quad (2.3)$$

Furthermore, as a consequence of equation 2.1, gravitational force has non-
trivial divergence only inside regions of mass,

\[ \Delta U(x) = \text{div} \cdot \nabla U(x), \]
\[ \Rightarrow \text{div} F(x) = G4\pi \rho(x), \]
\[ \Rightarrow \text{div} F(x) = 0, \text{ for } \forall x \notin \Omega. \]

The discontinuity in \( \rho(x) \) at the interface between \( \Omega \) and space means gravitational potential and gravitational force are smooth everywhere but at that boundary. There the potential is \( C^1 \) and the force is \( C^0 \).

### 2.1.1 Modeling Small Body Gravitation

Now we define precise quantitative and qualitative objectives for our models of gravity.

Throughout this research we use the asteroid 1998 ML14. It is roughly 1.0 km in diameter at its widest, and its density is estimated to be 2.5 g cm\(^{-3}\). The surface model used contains 16320 triangular faces and 8612 vertices. 1998 ML14 was picked for two reasons: \( i \) 1998 ML14 is similar to 4769 Castalia and 25143 Itokawa, two other asteroids for which we have detailed shape models; \( ii \) to compare results to previous work also using 1998 ML14 [10]. Whenever true gravitational force is required, for example when measuring the error of a method, a double precision implementation of the polyhedral method (section 2.2.2) is used.

Navigation near the surface of a small body requires accelerations to be known to \( 10^{-12} \text{ km/s}^2 \) [10]. The average acceleration on the surface of 1998 ML14 is approximately \( 10^{-7} \text{ km/s}^2 \), thus, the required relative error is \( 10^{-5} \). Farther away from the surface, acceleration becomes weaker, and the relative error bound becomes looser. Henceforth we take \( 10^{-5} \) relative error in the force as our quantitative goal.

On the qualitative side true gravitation exhibits several important features. Foremost is \( C^0 \) continuity. True gravitational force is \( C^0 \) continuous everywhere; therefore a model of gravitational force should, ideally, also be continuous. If the model fails to be continuous trajectory optimization, which relies on continuity for well posedness, may fail [9].

Another qualitative feature of gravitational force \( F \) is exactness, i.e. \( F \) is a conservative force and \( \exists U \) such that \( F = -\nabla U \). While this is clearly true
for true gravitation – after all \( \mathbf{F} \) is defined as \(-\nabla U\) – an approximation \( \mathbf{\hat{F}} \) may be constructed that does not obey this property. Such approximations may lead to trajectories with different qualitative characteristics. Specifically, trajectories simulated under such an \( \mathbf{\hat{F}} \) may not conserve energy.

Lastly, gravitation is divergence free. This is a concern when working with geometric integrators. Geometric integrators preserve intrinsic properties of the underlying vector field, such as volume preservation in divergence free vector fields. Only models that exhibit divergence free vector fields (or where nontrivial divergence is part of the physics) stand to benefit from these integrators; therefore, it is desirable to reproduce this feature of gravitational force when modeling it.

Although by no means an exhaustive list, these three qualitative features (continuity, exactness, and divergence freedom) shall henceforth be referred to as the qualitative features of gravitational force.

### 2.2 Previous Work: Small Body Gravitation

#### 2.2.1 Mascon Method

The *mass concentration* method (or mascon for short) is one of the first methods developed for modeling the gravitation of irregularly shaped bodies, and is still in use today for particular applications [10]. Mascons are derived by observing that gravitation is additive and small irregular shapes act like point masses at large distances [57]. Thus, a large irregular shape may be split into many small shapes each of which are approximated by a point mass; see figure 2.1. The mass of the asteroid is usually equally distributed among the mascons; however, in some circumstances it is desirable to distribute the density unevenly.

The mascon method provides a compelling and simple approximation for gravitational potential, and it is accurate even very close to the surface. Unfortunately the process of differentiation – to produce gravitational force – reveals the quantization errors of the approximation [57]. Even far from the surface (1 to 2 asteroid radii from its center of mass) the relative error is several orders of magnitude too large for trajectory simulation. Furthermore, mascons do not scale well: as the model increases in complexity the total cost of evaluation increases linearly. Hence, the mascon model should not be used
for trajectory simulation.

### 2.2.2 Polyhedral Method

The polyhedral method of Werner and Scheeres [57] calculates force by performing an exact integration (equations 2.2 and 2.3) over a polyhedral approximation to the mass. First, the mass is split into volumes of constant density. Then, each volumes’ boundary is approximated by a polyhedral surface, typically as triangles. For each volume the three dimensional integral is transformed into a surface integral by application of Green’s Theorem. This integral is further manipulated to yield a summation over all faces and edges of the surface description. Finally, the contribution from each volume is summed to yield the total force on a free particle. Figures 2.2 and 2.3 diagram this, and equations 2.4 and 2.5 give details:

\[
U(x) = \frac{1}{2} G \rho \sum_{e \in \text{edges}} \mathbf{r}_e \cdot \mathbf{E}_e(r_e \ln(v_e) - \frac{1}{2} G \rho \sum_{f \in \text{faces}} \mathbf{r}_f \cdot \mathbf{F}_f(r_f \tan(w_f)),
\]

\[
F(x) = -\nabla U(x) = G \rho \sum_{e \in \text{edges}} \mathbf{E}_e \cdot \mathbf{r}_e(x) \ln(v_e) - G \rho \sum_{f \in \text{faces}} \mathbf{F}_f \cdot \mathbf{r}_f(x) \tan(w_f).
\]

The notation of equations 2.4 and 2.5 is completely described by Werner and Scheeres [57], suffice it to note that there are two summations: one of faces (triangles) and the other over edges. The term within each summation
Figure 2.2: Each shaded area is a volume of constant density. The red disk is a free particle. Each volume has its own contribution, and the sum is taken for the net effect.

Figure 2.3: A simplified asteroid model shows the faces and edges of that comprise the asteroid’s surface model.

evaluates a transcendental function, and herein lies the expense of small body gravitation. A typical surface model has 10000s of faces and edges, and each costs the evaluation of either tan or ln. Despite these performance issues polyhedral gravitation is the standard technique for simulating trajectories near small bodies. This is primarily due to its accuracy.

In terms of accuracy the polyhedral method cannot be surpassed: a direct integration of the polyhedral mass is exactly the gravitation of said mass. For this reason the polyhedral method is considered the reference gravitation throughout this work. One may be suspicious of the constant density restriction, but in practice this is rarely a limitation for two reasons: i) Models of the internal mass distribution are never known at the mission design phase anyway. ii) The constant density assumption appears to be sufficient for simulation near many asteroids [18, 47, 58].
2.2.3 Modified Polyhedral Method

Several performance enhancements to the polyhedral method have been proposed [10]. Three approaches are investigated by Cangahuala [10]: replacing transcendental functions with Taylor series approximations, simplifying the mesh, and caching individual terms in the sum. They found that all three methods have a significant impact on the running time of a simulation. Specifically, in ideal circumstances a calculation could be sped up 100 fold without compromising the model’s accuracy; and more general orbits still benefited from a 10 times speed-up [10].

It is worth noting that qualitative features of gravitational force are not necessarily preserved under this approximation. For example, gravitation is continuous, but due to caches the Modified Polyhedral Method is not: In order to be useful a cache is “hit” whenever the input is within some epsilon of the cached input, but not necessarily exactly the cached input. This means whenever a cache is missed a discontinuity results. Exactness is also broken by caching as not all cache lines will miss or hit simultaneously; hence, the result will be a Frankenstein of previous computations that do not all come from the same query. Disabling caching restore exactness and continuity, but Taylor approximations mean the force field will contain non-trivial divergence\(^1\). Of course merely replacing the mesh does not compromise any qualitative features.

2.3 Previous Work: Large Body Gravitation

2.3.1 Spherical Harmonics

Spherical harmonics are a series of scalar-valued functions defined in spherical coordinates over \(\mathbb{R}^3\):

\(^1\)This has been verified by experiment. Simply replace the transcendental functions with their approximations and compute the Laplacian using Werner and Scheeres [57].
\[ U(r, \phi, \lambda) = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} A_n^m r^{-n-1} Y_n^m(\phi, \lambda) + \sum_{n=0}^{\infty} \sum_{m=-n}^{n} B_n^m r^n Y_n^m(\phi, \lambda), \quad (2.6) \]

\[
Y_n^m(\phi, \lambda) = \begin{cases} 
  P_n^m(\sin \phi) & \text{when } m = 0, \\
  P_n^m(\sin \phi) \cos m\lambda & \text{when } m > 0, \\
  P_n^m(\sin \phi) \sin |m|\lambda & \text{when } m < 0,
\end{cases}
\]

where \( r, \phi \) and \( \lambda \) represent radial distance, latitude and longitude respectively, and \( P_n^m \) are the associated Legendre functions \([13, 32]\). Two properties make spherical harmonics particular apt to modeling gravitational potential:

1. When the domain is confined to a constant radius (i.e. the surface of a sphere), spherical harmonics form a basis for all continuous functions on that surface.

2. Spherical harmonics in \( \mathbb{R}^3 \) are a solution to Laplace’s equation.

Below these properties are used to create an approximation to gravitational potential of an arbitrary body.

Spherical harmonics models the gravitational potential as a boundary value problem. The boundaries are taken to be infinitely far away and a sphere that encompasses the source. We know the potential approaches zero at infinite distance from the mass, and potential along the encompassing sphere is measured empirically or calculated using an exact method. Using property 1 it is possible to pick coefficients that bring a finite number of spherical harmonics arbitrarily close to the measured potential along the encompassing sphere. To solve the second boundary constraint set all \( B_n^m \) of equation 2.6 to zero. Functions solving Laplace’s equation have a unique solution given boundary conditions. Thus, by property 2, having solved the boundary value problem provides the potential in all space outside the encompassing sphere.

To obtain a force approximation take the negated gradient of the approximated potential. Thus, force is indirectly rendered via a representation for gravitational potential. Unfortunately, only domains with spherical boundary can be accurately represented. For example, a uniformly dense cube’s potential could only be accurately represented outside its circumsphere. Note
that taking a smaller sphere for the inner boundary (e.g. the one with radius half the edge length) is no longer Laplace’s problem: Mass outside the sphere gives rise to a non-zero left hand side of equation 2.1; hence, spherical harmonics are no longer appropriate.

Recent work on spherical harmonics in astrodynamics has focused on: Regular representations; although spherical harmonics have no inherent singularities the classic representation with Associated Legendre Functions has problems at the poles [13, 42]. Producing higher fidelity models; GRACE is an active mission measuring Earth’s gravitational field, and the Lunar Prospector recently produced two new models [33, 55]. Wavelet enhancement; recently wavelet approximations have been proposed to augment spherical harmonic models of terrestrial gravitation [20]. In this configuration spherical harmonics are used to model low frequency variation and spherical wavelets are used to model high frequency effects.

Outside astrodynamics high-speed implementations have been researched: ccSHT [11] is a free package designed for use on parallel computers, and Rokhlin and Tygert [44] is designed for a serial machine. However, these performance enhancements were not designed with trajectory simulation in mind.

Spherical harmonics is the preferred method for approximating gravitational force as many sources are spherical (e.g. planets) [13, 32, 42]. Recent interest in non-spherical bodies (e.g. asteroids and comets) with close proximity interactions challenge spherical harmonics, and motivate other approaches.

2.3.2 Polynomial Approximation

Interpolation based methods, as developed in Junkins [30] and Engels and Junkins [17], compute gravitational forces by differentiating a polynomial fit of the geopotential in a small region of space. With their approach, accurate gravity models around spherical bodies can be represented with relatively low degree polynomials; and force evaluations are reduced to efficient polynomial evaluations.

Given the nearly ellipsoidal shape of the Earth, Junkins [30] divides space with a uniform grid in ellipsoidal coordinates. Orthogonal polynomials are used as an interpolation basis for the interpolation of the geopo-
potential. In that setting, each component \( F_j(x) \) of the gravitational force, \( \mathbf{F}(x) = -\nabla U(x) \), is approximated as a linear combination of the derivative of the polynomial interpolation basis, \( p_i(x) \), i.e., \( F_j(x) \approx \sum_{i=1}^{n} a_i \frac{\partial}{\partial x_j} p_i(x) \), where the coefficients \( a_i \) depend on the interpolation nodes within an interpolation domain.

While the virtues of the interpolation approach (efficient force evaluation and ease of local model updating) are attractive for simulation near irregular bodies, a uniform grid is impractical for reaching the accuracy required for precise trajectory determination around such bodies. For example, in section 3.1.2 we give a rough calculation that between 6 and 74 Petabytes of storage would be required for a uniform grid, depending on the order of interpolation. Thus, any approach to modeling gravitation with polynomial approximation must use an adaptive grid.
Chapter 3

Adaptive Approximation by Polynomial Interpolation

Results

A piecewise polynomial approximation to gravitational force is developed. We use polynomial interpolation as a local approximation, and piece the local approximations together in an octree data structure. The result is an approximation of gravitational force that is both accurate and fast.

3.1 Polynomial Force Interpolation Near Irregular Bodies

This chapter begins our exploration of polynomial approximations to small body gravitation. Before developing a complete solution we describe how gravitational force can be interpolated locally. Following this description are numerical experiments exploring the errors of our local interpolation.

3.1.1 Interpolation Scheme

Given our initial goals of accelerating force evaluations for numerical integration applications, we opted to directly interpolate the force rather than the potential. Although this makes our force not globally exact, (i.e. not the gradient of a potential), this choice proved adequate for the purpose of this research.

While several shapes for interpolation domains, henceforth referred to as cells, have been considered, a cubic region was finally selected for its simplicity and sufficiency. Below we give a more detailed account of the interpolation used.

To interpolate gravitational force we pick Gauss-Lobatto-Legendre (GLL) interpolation points with the barycentric form of Lagrange polynomials: GLL
Figure 3.1: Gauss-Lobatto-Legendre nodes. Left: one dimensional Gauss-Lobatto-Legendre nodes. Right: two dimensional cartesian product of the one dimensional nodes.

points have a low Lebesgue constant, which translates to being close to the best uniform approximation of the interpolated function [31]; and the barycentric form of Lagrange polynomials are known for their superior numerical conditioning and computational efficiency [8].

Constructing GLL points in a cubic domain is achieved by cartesian product of the 1-dimensional case. The 1-dimensional GLL points of order $k$ are the $k + 1$ zeros of

$$\ell_k(x) := (x - 1)(x + 1)P'_k(x),$$  \hfill (3.1)

where $P_k(x)$ is the Legendre polynomial of degree $k$. The multi-dimensional case is then built by computing these zeros for each axis separately and forming their cartesian product, as illustrated in figure 3.1. We shall refer to this collection of points as $\mathcal{N}$ and use $|\mathcal{N}|$ to denote the number of points in $\mathcal{N}$, which is $(k + 1)^3$.

Lagrange polynomials satisfy

$$p^N_{\mathbf{x}_i}(\mathbf{x}_j) = \delta_{ij} \text{ for } \mathbf{x}_i, \mathbf{x}_j \in \mathcal{N},$$  \hfill (3.2)

and can be computed as the products of the univariate Lagrange polynomials centered at the coordinates of $\mathbf{x}_i$. In three dimensions this yields

$$p^N_{\mathbf{x}_i}(\mathbf{x}) = \frac{\ell_k(x)}{\ell'_k(x_i)(x - x_i)} \frac{\ell_k(y)}{\ell'_k(y_i)(y - y_i)} \frac{\ell_k(z)}{\ell'_k(z_i)(z - z_i)},$$  \hfill (3.3)

which is of degree $3n$. Here $\ell_k(x)$, $\ell_k(y)$ and $\ell_k(z)$ are as defined in equa-
tion (3.1) and each factor in the product above is the barycentric form of the univariate Lagrange polynomial [8]. The factors corresponding to each dimension are of the same degree \( k \). For this reason, we will call the interpolation scheme based on such polynomials “order \( k \) interpolation” and refer to interpolating polynomials as “order \( k \) polynomials”. Note that the quantities \( \ell'_k(x_i), \ell'_k(y_i), \) and \( \ell'_k(z_i) \) only depend on the locations of the interpolation points. Thus these can be precomputed and stored in a table, which costs \( O(|\mathcal{N}|^2) \) (each \( \ell'_k \) costs \( O(|\mathcal{N}|) \), and we must compute one for each \( x_i \in \mathcal{N} \)). The evaluation of the interpolation polynomial, however, is an \( O(|\mathcal{N}|) \) operation: All \( p^{N}_{x_i} \) share the term \( \ell_k(x)\ell_k(y)\ell_k(z) \), so this can be computed once per evaluation at cost \( O(|\mathcal{N}|) \). Finally, each \( p^{N}_{x_j} \) also uses \( 1/((x - x_i)(y - y_i)(z - z_i)) \) for a total across all \( p^{N}_{x_i} \) of an additional \( O(|\mathcal{N}|) \). Thus, interpolation using barycentric form of Lagrange polynomial basis requires \( O(|\mathcal{N}|^2) \) setup, but only \( O(|\mathcal{N}|) \) work per evaluation.

Now we can construct a polynomial approximation of function \( f \) as a linear combination of the function’s value at the interpolation nodes times the Lagrange polynomial centered at that point:

\[
f(x) \approx \sum_{x_j \in \mathcal{N}} f(x_j)p^{N}_{x_j}(x). \tag{3.4}
\]

In our case the function \( f \) corresponds to a component of the gravitational force around an asteroid. The forces per unit mass are computed via the polyhedral method, but can be obtained from other sources as well, such as measured gravimetric data, or by numerically solving Poisson’s equation at the interpolation points.

The interpolation error of the above scheme can be quantified via Ciarlet’s formula [22] which in our case depends on an integral of derivatives of the components of the force. This suggests that the error will increase as we approach the asteroid (since force depends inversely on the second power of distance) and decrease as the interpolation domain becomes smaller.

### 3.1.2 Numerical Experiments

Now we study interpolation errors using numerical experiments. To find the error in a cell we took random samples of the approximate force \( \hat{\mathbf{F}} \) and exact force \( \mathbf{F} \) (computed via the polyhedral method), we computed the relative
error as $\|\hat{F} - F\|/\|F\|$. The maximum such error was taken as the error bound of a cell. Note that this measure is dimensionless. In the following we explore the change in interpolation error as parameters of our local model are varied: distance from the asteroid, effect of surface irregularities, size of the cell, and order of the polynomial interpolation. Cangahuala [10] suggests that the relative error in acceleration that is acceptable for mission design is $10^{-5}$; thus, we use this number or a smaller error as our target error. All tests were done with three dimensional bodies and cells. For simplicity, many of the figures depict a two dimensional slice. The banding that is visible in the error graphs in figures 3.3-3.6 and figure 3.10 is due to the location of the interpolation points and the shape of the asteroid. In each of these plots the logarithm (base 10) of the error is shown.

Distance From an Asteroid

Our first experiment was designed to test the hypothesis that gravity in cells closer to the asteroid would be more difficult to approximate than in cells farther away. We setup an interpolation cell with 400 m sides and order 6 polynomials at progressively closer locations. To keep our test simple we used a cuboid with the approximate dimensions of Castalia [29] as our asteroid. The cells used were in two types of locations. One set was at varying distances from a flat face of the asteroid at 0 m, 500 m, and 1000 m between the closest part of the interpolation region and the surface of the asteroid. A second set was placed similarly at 0 m, 707 m, and 1414 m from an edge of the asteroid. Figure 3.2 shows the test cases, and table 3.1 summarizes the results.

In both cases we can see that bringing the cell closer to the asteroid increases the error in approximation. For the edge on case we see a dramatic difference, though this is probably due to the presence of the edge, which is a high curvature feature. To investigate the shape effect further we repeated this experiment with a point-mass approximation instead of the polynomial interpolant while keeping the test regions identical. The results of this experiment are also summarized in table 3.1. The point-mass approximation is very ill-suited to approximating our cuboid asteroid. Thus, the errors made by the polynomial interpolant are not merely a result of misrepresenting a $1/r^2$ term because if the $1/r^2$ term were dominant a point mas approximation would suffice. From this we conclude that shape effects significantly impact
the errors present at this range.

<table>
<thead>
<tr>
<th>Distance</th>
<th>Face approach max. error</th>
<th>Point-mass</th>
<th>Edge approach max. error</th>
<th>Point-mass</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interpolation</td>
<td>2.29 $\times 10^{-8}$</td>
<td>0.168</td>
<td>3.89 $\times 10^{-9}$</td>
<td>0.0819</td>
</tr>
<tr>
<td>Far</td>
<td>2.95 $\times 10^{-7}$</td>
<td>0.400</td>
<td>7.67 $\times 10^{-8}$</td>
<td>0.192</td>
</tr>
<tr>
<td>Medium</td>
<td>9.77 $\times 10^{-7}$</td>
<td>1.85</td>
<td>5.37 $\times 10^{-3}$</td>
<td>0.634</td>
</tr>
</tbody>
</table>

Figure 3.2: Context for computing error as a distance from body. The body is the larger square and the smaller squares are test regions. Left figure is the context for face approach in which the distances are 0 m, 500 m and 1000 m. Right figure is context for edge approach in which distances are 0 m, 707 m and 1414 m. These are labeled adjacent, medium and far in table 3.1 which shows the errors.

### High Curvature Features

To further investigate the effects of high curvature features we performed another test. In this experiment we used the same Castalia-like cube, but added a tetrahedron to one face to act like a small hill on the surface. We started with a tetrahedron with 200 m edge lengths and scaled down from there. For each configuration we used a cell with 25 m sides and order 6 interpolation, placed such that the center of the closest face was aligned with
the tip of the tetrahedron. Figure 3.3 shows some test cases, and table 3.2 summarizes the results.

<table>
<thead>
<tr>
<th>Feature size (m)</th>
<th>Max. error</th>
<th>Feature size (m)</th>
<th>Max. error</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>$6.84 \times 10^{-4}$</td>
<td>100</td>
<td>$6.00 \times 10^{-4}$</td>
</tr>
<tr>
<td>175</td>
<td>$6.62 \times 10^{-4}$</td>
<td>75</td>
<td>$5.79 \times 10^{-4}$</td>
</tr>
<tr>
<td>150</td>
<td>$6.41 \times 10^{-4}$</td>
<td>50</td>
<td>$5.62 \times 10^{-4}$</td>
</tr>
<tr>
<td>125</td>
<td>$6.19 \times 10^{-4}$</td>
<td>25</td>
<td>$5.37 \times 10^{-4}$</td>
</tr>
</tbody>
</table>

As we want errors beneath $10^{-5}$, it seems that even a slight bump can give polynomial interpolation significant problems. Given that the 1998 ML14 model has edge lengths as small as 9 m, this is especially troubling. The solution, as we shall see, is to use smaller cells that wrap around the feature instead of one large cell.

**Varying the Size of a Cell**

Given the poor performance of polynomial interpolation for cubes at the 25 m scale, we wanted to know how small our cubes would need to be in order to achieve the desired error. In this experiment we studied the effect of varying the size of a cell on the accuracy of our interpolation. To test this we placed a cell with order 6 interpolation centered near a tip of a 1998 ML14 model. Starting with an edge length of 250 m, we halved each dimension of the cube for every subsequent experiment. Figure 3.4 shows some test cases, and table 3.3 summarizes the results.

<table>
<thead>
<tr>
<th>Size (m)</th>
<th>Max error</th>
<th>Size (m)</th>
<th>Max error</th>
</tr>
</thead>
<tbody>
<tr>
<td>250</td>
<td>$5.75 \times 10^{-2}$</td>
<td>3.91</td>
<td>$9.03 \times 10^{-4}$</td>
</tr>
<tr>
<td>125</td>
<td>$2.85 \times 10^{-2}$</td>
<td>1.95</td>
<td>$4.63 \times 10^{-4}$</td>
</tr>
<tr>
<td>62.5</td>
<td>$1.46 \times 10^{-2}$</td>
<td>0.977</td>
<td>$2.28 \times 10^{-4}$</td>
</tr>
<tr>
<td>31.3</td>
<td>$7.24 \times 10^{-3}$</td>
<td>0.488</td>
<td>$1.14 \times 10^{-4}$</td>
</tr>
<tr>
<td>15.6</td>
<td>$3.63 \times 10^{-3}$</td>
<td>0.244</td>
<td>$5.66 \times 10^{-5}$</td>
</tr>
<tr>
<td>7.81</td>
<td>$1.86 \times 10^{-3}$</td>
<td>0.122</td>
<td>$2.82 \times 10^{-5}$</td>
</tr>
</tbody>
</table>
Figure 3.3: Effect of small features on error. The shaded plots show the logarithm of relative error of gravitational force along the face of the cell touching the tetrahedron feature. Top left shows an image of our test body. In this image the tetrahedron had 200 m edge lengths. Error plots shown are for tetrahedra with edge lengths of 175 m (top right), 100 m (bottom left) and 25 m (bottom right). Each cell used order 6 interpolation. See also table 3.2.
Figure 3.4: Error as a function of cell size. The shaded plots show a cross section of the logarithm of relative error of gravitational force. Top left shows the context for the test. A silhouette outline shows the boundary of asteroid 1998 ML14; the squares are slices of cells we tried. Errors are shown for cells with edge lengths 250 m (top right), 62.5 m (bottom left) and 3.91 m (bottom right). The saturated white regions in the first two error plots show that large errors are present. The interior of the asteroid is not measured and left black. Each cell used order 6 interpolation. See also table 3.3.
We can use these results to motivate the case for an irregular grid. First, assume the domain of interest is a cube surrounding 1998 ML14 with edge length 2500 m; this is a fair assumption because anywhere outside that range we can use low order spherical harmonics as a fast approximation. From these results we conclude that we will need resolution down to 12 cm to capture the fine detail near the surface. With a regular grid this would require dividing the domain into \((2500/0.12)^3 \approx 9 \times 10^{12}\) cells. If each cell contains order 6 interpolation, we need \(7^3 \times 3 = 1029\) double precision coefficients, or 8232 bytes per cell (the power of 3 comes from 3 dimensions, and the factor of 3 from each component of force). The total memory cost for such a model is about 74 Petabytes. Even at order 2, we would need around 6 Petabytes to store a regular grid.

### Varying the Polynomial Order

Our next experiment focused on varying the order of approximation; specifically, we explored the interaction between distance from the asteroid and order of approximation. Two cells centered at (441 m, 231 m, 0 m) (near surface) and (750 m, 231 m, 0 m) (about 1.5-radii away) with 250 m edge length and polynomial interpolants between orders 1 and 7 were tested. Figures 3.5 and 3.6 show some test cases, and table 3.4 summarizes our results.

<table>
<thead>
<tr>
<th>Order of interpolation</th>
<th>Distant case max. error</th>
<th>Close case max. error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7.94 \times 10^{-2}</td>
<td>3.48 \times 10^{-1}</td>
</tr>
<tr>
<td>2</td>
<td>5.37 \times 10^{-3}</td>
<td>1.40 \times 10^{-1}</td>
</tr>
<tr>
<td>3</td>
<td>8.91 \times 10^{-4}</td>
<td>1.13 \times 10^{-1}</td>
</tr>
<tr>
<td>4</td>
<td>1.20 \times 10^{-4}</td>
<td>7.59 \times 10^{-2}</td>
</tr>
<tr>
<td>5</td>
<td>2.34 \times 10^{-5}</td>
<td>7.16 \times 10^{-2}</td>
</tr>
<tr>
<td>6</td>
<td>3.23 \times 10^{-6}</td>
<td>5.75 \times 10^{-2}</td>
</tr>
<tr>
<td>7</td>
<td>6.03 \times 10^{-7}</td>
<td>5.37 \times 10^{-2}</td>
</tr>
</tbody>
</table>

Far away there is a clear benefit to using high order polynomials for approximating the gravitation. Closer in, however, these results show diminishing returns for higher order polynomials. We conclude that the appropriate strategy for polynomial approximation uses few high order cubes far away,
Figure 3.5: Error as a function of order of interpolation in a distant cell. The shaded plots show a cross section of the logarithm of relative error of gravitational force. Top left shows the context for the test. The silhouette outline shows the boundary of 1998 ML14, the square is a projection of the cell. Error plots shown are for cells with interpolation orders 2 (top right), 4 (bottom left) and 6 (bottom right). The interior of the asteroid is not measured and left black. See also table 3.4.
Figure 3.6: Error as a function of order of interpolation in a cell close to the asteroid. The shaded plots show a cross section of the logarithm of relative error of gravitational force. Top left shows the context for the test. The silhouette outline shows the boundary of 1998 ML14, the square is a projection of the cell. Error plots shown are for cells with interpolation orders 2 (top right), 4 (bottom left) and 6 (bottom right). As we can see all errors reported are outside the acceptable range. The interior of the asteroid was not measured and left black. See also table 3.4.
and many low order cubes closer to the asteroid. This is somewhat counterintuitive as one may expect high order approximations to yield the most significant benefits where the field changes most rapidly.

With all these results in mind, we conclude that an efficient representation of gravitational force near a small body must be adaptive in both size and interpolation order of each cell.

3.2 Adaptive Spatial Partitioning

Our approach to modeling the gravitational force near a small body separates the problem into two tasks: dividing the domain into cells, and approximating the force in a cell. Having already settled on a solution for the latter, we now discuss the former.

Subdividing the domain into manageable cells is accomplished with an adaptive octree data structure [46]. This data structure is constructed by recursively splitting the domain into a hierarchy of different sized cuboids that each contain a local model of gravitational force. We initialized the process with a local model for the highest level cuboid. From here we estimate the error relative to the polyhedral method; if the error is too large the cuboid is divided, and the process is repeated recursively on each of the pieces. Otherwise the cuboid is retained and becomes a leaf cell in the octree. The interpolation information is retained only for leaf cells and it is discarded for the rest. In building the octree data structure, we conservatively choose a relative error threshold of $5 \times 10^{-7}$. This is almost two orders of magnitude less than the error in acceleration advised by Cangahuala [10]. We choose the more conservative threshold during octree construction because error estimation during that stage is based on random sampling as explained in section 3.1.2.

The use of such an adaptive structure in combination with the selected interpolation scheme produces a configuration known as a T-junction. At T-junctions continuity is frustrated by misaligned interpolation points, as depicted in figure 3.7. In this example the blue and red cells use the same order interpolation at a shared face. Since they do not share interpolation points – only the black points are shared – the polynomial at the boundary may be different depending on the cell you approach it from. Hence, a
discontinuity may form at T-junction. In our experiments with trajectories this was not an issue, as shown by results of section 3.4, though it remains to be seen how discontinuity affects other kinds of simulation (e.g. trajectory optimization).

3.2.1 Octrees

We now provide practical detail regarding the octree data structure. Octrees are more easily illustrated by their 2D analogue; hence, the following description is given for “quadtrees”. Quadtrees are adaptive tree data structures for organizing localized data in a rectangular domain. Two operations characterize the function of a quadtree: Subdivide, and Find. Subdivide is a constant time operation that splits a rectangle into four quadrants by splitting each dimension in half. A quadtree is built by beginning with a single rectangle and subdividing recursively until the desired tree structure is created [46]. Figure 3.8 shows a quadtree as a tree and as a collection of rectangles.

Find recalls localized data associated with query points by recursively traversing the tree. At each level Find picks the child quadrant containing our query; this is a constant time operation as each cell has at most four children. After every level is traversed a leaf is reached, and the data it
Figure 3.8: Left: A quadtree viewed geometrically. Right: A quadtree viewed as a tree. Labels show the mapping between geometric and tree views. The point represents a query; the tree view shows the query being resolved. Cells 3 to 9 are leaf cells.

contains is returned. As balanced trees have at most \( \log N \) levels the run time complexity of \texttt{Find} is \( O(\log N) \), where \( N \) is the total number of cells [21]. Thus, subdividing our model to improve accuracy incurs only a sub-linear run time penalty. This compares favorably to the polyhedral method which requires a linear cost increase to improve accuracy.

To illustrate the practical benefit of a sub-linear run time consider the following example. For the sake of argument we shall examine moving from polyhedral models with 1000 elements to 10000 elements, and compare that to moving from 10000 octree cells to 100000. Following this scenario the polyhedral method would cost \( 10 \times \) more computation when moving to the new model, whereas an octree method only costs \( 1.2 \times \) the former computation. Taking this out another factor of 10 we find costs rising \( 100 \times \) and \( 1.4 \times \) respectively. In other words, methods with asymptotically better performance have dramatically superior run times as problems scale up to take advantage of newest computational power available.

Note that the computational complexities of polyhedral method versus our method depend on different things (asteroid mesh complexity versus number of cells, respectively). The same gain in accuracy by the two methods may require different refinements. Thus comparing complexities as we do above is simplistic. However, our experiments described in section 3.4 show that trajectory integration using our method is much faster than the polyhedral method while producing extremely accurate trajectories. This is made more precise in section 3.4.
While quadtrees can be implemented in a straightforward manner, there are several low level improvements which are worth noting [21]. For example, a typical implementation associates a rectangle with each cell in the quadtree. Queries made to the quadtree recursively traverse each level by performing a containment test on every rectangle the next level down. This implementation is functional, but substantial savings in time and memory can be made by taking advantage of the quadtree’s special structure [21].

Octrees follow the same design, but use eight cuboids in 3D instead of four rectangles in 2D.

3.2.2 Spherical Harmonics Far Away

The octree structure described herein must exist in a bounded cuboid region; this precludes it from producing approximations to acceleration everywhere outside the body. To provide such approximations we employ spherical harmonics. We place a sphere centered at the origin and just large enough to enclose the body. Then coefficients to spherical harmonics that fit the gravitational potential are computed. In theory, one could take sufficient coefficients and produce an accurate field for all space outside the sphere; in practice, computing to such a degree of accuracy is too costly. Instead, we satisfy ourselves with accurate results outside the octree domain. This is easier to achieve as the higher order components of spherical harmonics fall off quickly as distance from the body increases.

3.3 Example Octree Construction and Performance

To investigate the errors in our approximate gravity field we constructed an octree model in a region around asteroid 1998 ML14, and numerically analyzed the interpolation error. The experiments were done using a triangle mesh surface model of the asteroid 1998 ML14 with 8,162 vertices, 24,480 edges, and 16,320 triangles.
3.3.1 Octree Model Construction and Error Analysis

The domain of the octree began at (-1250, -1250, -1250) m and extended 2500 m in each direction, where the origin is the center of mass of the asteroid model. (As a point of reference the radius of 1998 ML14 is \(\approx 500 \text{ m}\).) The octree was limited in depth to 9 levels; therefore the smallest cell size was about 10 m. The first three levels of cells used order 6 polynomials, the last two levels order 2, and the rest used order 4. Each cell was tested with 10000 sample points, and subdivision continued until the maximum of these errors was beneath \(5 \times 10^{-7}\). The error was measured as described in section 3.1.2. This model was created in 1150 CPU hours on a parallel computer using the Message Passing Interface (MPI) [53] (64 processors for approximately 18 hours), and occupies 653 Megabytes of memory. To cover the region outside the octree spherical harmonics of degree and order 12 were employed. We name this model the cubetree model.

Part of an octree structure for 1998 ML14 is shown in figure 3.9 and figure 3.10 shows cross sections of the error in the \(xy\)-plane. The first error plot starts at (400 m, -250 m) and extends 500 m; subsequent plots magnify the region near (400 m, 100 m) by 2 and 4 times. As we can see, errors are very well behaved for the majority of the plotted regions. In fact, only when we get very close (within a few meters) to the surface are we in danger of violating our goal of \(10^{-5}\). This is due to the bound on the octree depth imposed on this particular experiment, and can be improved by building a tree with smaller cells close to surface. Note that even though the error bound is violated at the surface, the results from our experiments on ejecta trajectories in section 3.4.5 show that this is not a problem in practice.

3.3.2 Single Evaluation Speed Tests

In this experiment we measured the comparative performances of the competing models by measuring the time required for a single force evaluation. Table 3.5 summarizes the relative speeds (1.0 being polyhedral method) of several methods. We can see that the polynomial interpolation scheme compares favorably with other methods. Compared to [10] we have similar performance, but better understanding of the errors. Specifically, while Cangahuala [10]
Figure 3.9: An octree model for part of space around asteroid 1998 ML14. The cube in the top figure shows the region for which this octree was constructed. The cubes in the bottom two rows are visualized as translucent to reveal the hidden structure. In the sequence shown in bottom 2 rows, larger cubes are incrementally removed to reveal the finer structure of the octree.
Figure 3.10: Error of an octree model for asteroid 1998 ML14. The plots show the logarithm of relative error in gravitational force. Top plot shows the error along the $xy$-plane at $z=0$ m cutting across many cells. Lower left and right show a zoom of 2 and 4 times. The error in most of the cells is less than $10^{-5}$ as desired. The only exceptions are in the cells very close to the asteroid. These are cells of size about 10 m. Cells used a variable order interpolation depending on their size, orders ranged from 2 to 6. The interior of the asteroid was not measured and left black.
only reports errors for orbits at 3 radii from the body, our error estimates go all the way to the body.

Table 3.5: Relative speeds of available methods for single force evaluation.

<table>
<thead>
<tr>
<th>Model</th>
<th>Speed Factor</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>polyhedral</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>order 6 polynomial</td>
<td>0.0104</td>
<td></td>
</tr>
<tr>
<td>order 4 polynomial</td>
<td>0.0038</td>
<td>Estimated from order 6</td>
</tr>
<tr>
<td>order 2 polynomial</td>
<td>0.0008</td>
<td>Estimated from order 6</td>
</tr>
<tr>
<td>degree and order 12 spherical harmonics</td>
<td>0.0145</td>
<td></td>
</tr>
<tr>
<td>coarse shape, Taylor series</td>
<td>0.091</td>
<td>Cangahuala [10]</td>
</tr>
<tr>
<td>coarse shape, Taylor series, histories</td>
<td>0.01</td>
<td>Cangahuala [10]</td>
</tr>
</tbody>
</table>

3.4 Performance Analysis Using Trajectory Integrations

While the measurement of a single force evaluation described in previous section gives some idea of the order of speed-up obtained with the octree method, this factor is actually a function of space. For example, cells closer to the asteroid are usually deeper in the octree, so they receive lower order interpolants. To get a notion for actual speed improvements we have to integrate trajectories. This was done for four different classes of trajectory: close retrograde orbits, mid-range orbits, random trajectories, and ejecta.

3.4.1 Experiment Design

In each of the experiments below we generate several trajectories with different force models and parameters.

*Cubetree* trajectories are generated using the cubetree force model. Integration is done with the Embedded Runge-Kutta Prince-Dormand (order 8,9) method using relative error tolerance $10^{-13}$ and absolute error tolerance $10^{-6}$. Going beneath $10^{-6}$ can caused the adaptive time stepping routine to over refine the step size.

A *reference* trajectory refers to a simulation done with the polyhedral method as described by Werner and Scheeres [57]. Integration is done with
the same method as above, however, the absolute error threshold is set to $10^{-10}$. These trajectories are used as a baseline to measure the accuracy of cubetree trajectories.

Finally, trajectories generated with the augmented polyhedral model use a mix of polyhedral and spherical harmonics: spherical harmonics are used where ever they would be used in a cubetree trajectory. Furthermore, augmented polyhedral trajectories use the same tolerances as cubetree trajectories. The augmented polyhedral method is used to measure timing performance of the cubetree model.

The trajectory integrations were done in rotating coordinates. The period of rotation for 1998 ML14 was assumed to be 14.93 hours and the moment of inertia tensor was computed from the triangle mesh surface of the asteroid. This was used to compute the principal axis. The local coordinate system for the asteroid was used as the rotating coordinate system; the $z$-axis turned out to be close to but not exactly same as the principal axis. The computed normalized principal axis, in the coordinate system of the asteroid mesh was $(0.0636, 0.0008, 0.9356)$. Thus the $xy$-plane was close to, but not the same as the equatorial plane.

For the semimajor axis calculation in section 3.4.6, the mass value was calculated from an assumed density of $2.5 \times 10^3 \text{ kg/m}^3$ and computed volume of approximately $511,320,552 \text{ m}^3$.

### 3.4.2 Close Retrograde Orbits

For our first experiment we chose a known family of stable orbits. Initial conditions were chosen randomly within a band of retrograde orbits close to the asteroid. Specifically, we placed initial conditions near the equatorial plane with randomly chosen radii between 600 m and 1000 m from the center of the asteroid. Velocities were always chosen to place the orbiter in a retrograde orbit. Initial speeds were chosen between 0.45 and 0.75 of escape speed. The only force simulated was gravitation in rotating coordinates. Simulations ran for 30 days of ballistic motion with each model (cubetree, augmented polyhedral, and reference), where impacting trajectories were thrown out. Impacting trajectories are addressed in section 3.4.5. The position and velocity of the orbiter was recorded every 5 minutes of simulated time.

This experiment was repeated for 1,111 trajectories. For each trajectory
we measured the maximum difference in position and velocity between the cubetree trajectory and reference trajectory. Figure 3.11 is a histogram of the errors in position and velocity. Clearly the vast majority of trajectories fall within 2 m of the reference trajectory; in fact, only 4 trajectories were outside a 2 m range. The maximum position error was 3.56 m, and the minimum was 9.76 mm. On average integrations with the cubetree method were 112 times faster than the augmented polyhedral method.

Figure 3.11: Histograms of errors in position and velocity for 1,111 close retrograde cubetree trajectories integrated for 30 days and observed at 5 minute intervals.

\[ \text{Figure 3.11: Histograms of errors in position and velocity for 1,111 close retrograde cubetree trajectories integrated for 30 days and observed at 5 minute intervals.} \]

### 3.4.3 Mid-Range Orbits

Another region of interest in our model is the jump between octree and spherical harmonics. To investigate this domain we performed a similar experiment. Initial conditions were placed between 1250 m and 1500 m from the center; velocities were randomly picked between 0.67 and 0.8 of the escape speed with both prograde and retrograde orbits. Inclination of the orbits was limited by choosing initial positions near the equatorial plane and initial velocities with a small component outside the equatorial plane. Otherwise, this experiment was identical to the previous.

We repeated the simulation for 1,487 trajectories. Figure 3.12 is a histogram of the errors in position and velocity. Again the majority of trajectories differ by less than 2 m; only 15 have position error greater than 2 m. The minimum error in position is 3.2 mm, median 12.22 cm and maximum
10.24 m. These 3 orbits are shown in figure 3.14. Note that the maximum error is significantly larger than in our previous experiment. We expect these worse case trajectories to have inherently sensitive dynamics; a hypothesis we shall revisit and show evidence for in section 3.4.6. On average integrations using our cubetree method were 90 times faster than using the augmented polyhedral method. Recall that both methods used spherical harmonics outside a certain range.

Figure 3.12: Histogram of errors in position and velocity for 1,487 mid-range cubetree trajectories integrated for 30 days and observed at 5 minute intervals.

### 3.4.4 Random Trajectories

Next we explored more of the phase space. Positions were taken between 600 m and 1500 m from the center of the asteroid, and velocities were chosen from the plane passing through the initial position and tangent to the sphere centered at the origin. Magnitude of the velocity was clamped to within 0.45 and 0.75 of escape speed. Simulation was performed identically to the previous two, and repeated 911 times. The histograms in figure 3.13 summarize the results; the last column in each histogram represents all differences greater than or equal to the 20 m or mm/s. In this experiment even more cubetree trajectories diverge from the reference: 38 trajectories have error greater than 2 m, and of those 8 have greater than 100 m error. On average the integration time of these trajectories was accelerated by 111 times.
3.4.5 Ejecta Trajectory

The final experiment focused on ejecta and impacts. Initial positions were chosen with a uniform random distribution on the polyhedron’s surface; initial velocities were chosen from the hemisphere above the surface and with magnitude uniformly distributed between 0.1 and 0.9 of escape speed. Otherwise the experiment is the same as the previous; 2,440 trajectories were generated this way. Since most ejecta trajectories are short lived the differences are small: only one trajectory had more than 0.5 m difference in position (it had 4 m difference). As such, histograms have been omitted. Ejecta trajectory were calculated 169 times faster using the cubetree approximation. This improvement reflects the lower order interpolation used near the surface of the asteroid.

3.4.6 Dynamical Error Analysis

In order to better understand the errors found in the previous experiments, the underlying dynamical properties of a few sample trajectories have been considered. This analysis indicates that the large errors obtained for particular trajectories are not an intrinsic limitation of the approximation scheme used, but rather of the sensitive nature of the trajectories in chaotic regions. This result is achieved via frequency analysis and Monte Carlo sensitivity
analysis.

**Frequency Analysis**

Given that the physical system we are modeling is conservative (Hamiltonian), Fourier analysis of trajectories is a powerful tool to discriminate between regular and chaotic motion [35, 36]. In particular, regular (quasi-periodic) trajectories in Hamiltonian systems – which correspond to stable trajectories and present only linear sensitivity with respect to the initial conditions – exhibit a discrete spectrum of frequencies corresponding to the natural frequencies of the torus on which they lie. On the other hand, chaotic trajectories, which present exponential divergence between neighboring trajectories present a continuous spectrum, and appear as noise on the power spectrum of some coordinates.

In order to test the hypothesis that trajectories presenting large discrepancies between the numerical integration in the two models are sensitive, we applied a fast Fourier transform (FFT) on sample trajectories in the mid-range test case of section 3.4.3. We chose three sample trajectories corresponding to the minimum, median, and maximum error cases; and applied the FFT on the semi-major axis, $a$, for an integration time span of 30 days with sampling period of 5 minutes. The results are shown in figure 3.15 and the actual position paths corresponding to those trajectories are shown in figure 3.14. Figure 3.15 shows the power spectral density $P$ of the trajectories for both the cubetree approximation (continuous line) and reference method (dots), as well as the normalized difference in power spectral density, $D$:

$$D(f) = \frac{|P_{\text{cubetree}}(f) - P_{\text{polyhedral}}(f)|}{P_{\text{polyhedral}}(f)}$$

As can be observed, the minimum difference trajectory (top plot of Fig. 3.15) shows a “discrete” spectrum where a few main frequency peaks are apparent. Thus, this trajectory is likely to be a regular one. On the other hand, as the discrepancy between the integration results based on the two different models increases, the number of frequencies in the power spectrum tends to increase. In the median case the main frequencies are still dominant, but show a small instability. For the largest error case, the noise is significant,
Table 3.6: Sensitivity with respect to initial condition: maximum Hausdorff distance between reference trajectories and their perturbations for mid-range orbits.

<table>
<thead>
<tr>
<th>Trajectory</th>
<th>Difference in Position (m)</th>
<th>Trajectory</th>
<th>Difference in Position (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>40.03</td>
<td>9</td>
<td>16.53</td>
</tr>
<tr>
<td>2</td>
<td>35.36</td>
<td>10</td>
<td>19.00</td>
</tr>
<tr>
<td>3</td>
<td>256.0</td>
<td>11</td>
<td>30.04</td>
</tr>
<tr>
<td>4</td>
<td>22.34</td>
<td>12</td>
<td>346.9</td>
</tr>
<tr>
<td>5</td>
<td>19.95</td>
<td>13</td>
<td>12.28</td>
</tr>
<tr>
<td>6</td>
<td>13.00</td>
<td>14</td>
<td>20.86</td>
</tr>
<tr>
<td>7</td>
<td>31.65</td>
<td>15</td>
<td>16.99</td>
</tr>
<tr>
<td>8</td>
<td>60.99</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

suggesting a chaotic trajectory. Similar results have been obtained for the other action elements (eccentricity, inclination). This indicates that in the cases where large errors between the polyhedral and cubetree model have been obtained, the discrepancy is likely to be a result of the intrinsic sensitive nature of the trajectories rather than due to the approximation method.

**Sensitivity analysis**

In order to better test the extent of the sensitive nature of the trajectories corresponding to the largest errors in the mid-range experiment, we performed a Monte Carlo simulation over perturbations around initial conditions. For seed initial conditions we used the 15 worst case trajectories from the mid-range experiment. Using only reference trajectories, trajectories from each perturbation were compared with the trajectory of the nominal initial condition by taking the Hausdorff distance. Perturbations were chosen from a normal distribution, with the mean centered at the nominal initial condition and standard deviations 1 mm and 1 $\mu$m/s for position and velocity. Each initial condition was run with 10 perturbations and the maximum Hausdorff distance was recorded. Table 3.6 shows the results, trajectory 1 was the worst case trajectory, trajectory 2 the second worst, etc.

As can be clearly observed from this table, small perturbations in the ini-
Figure 3.14: Three mid-range 30 day cubetree trajectories sampled at 5 minute intervals. Each plot corresponds to a different initial condition in the experiment described in section 3.4.3. The top, middle, and bottom have the smallest (3.2 mm), median (12.22 cm), and maximum (10.24 m) errors respectively. Frequency analysis for these is shown in Fig. 3.15. The asteroid model shown is a simplified version. The one used for actual trajectory propagation in all experiments had 16,320 triangles.
Figure 3.15: Frequency analysis of semi-major axis of mid-range 30 day cubetree trajectories shown in Fig. 3.14. The plots, from top to bottom are for the trajectory with the smallest, median, and maximum errors in the set of mid-range trajectories.
tial conditions lead to significant variations in the resulting trajectories which shows that the dynamics is highly sensitive to initial conditions. These results quantify the previous observation obtained via frequency analysis: since both models can be considered small perturbations of each other, the discrepancies between two integrated trajectories in chaotic regions may present large variations, independently of the approximation method used. The cubetree method captures this high sensitivity as seen in the frequency analysis, which shows its overall consistency.

3.5 Discussion

The cubetree technique combines an adaptive spatial data structure with polynomial interpolation to cover the entire domain with an approximation. The decomposition of the domain can be quite coarse in some regions, and it also varies in order of interpolation. This flexible spatial hierarchy enables us to refine our model in regions that are difficult to approximate (e.g. near high curvature regions), and enables error guarantees on the model. Even lacking continuity and exactness, this technique produces extremely accurate results in the vast majority of the trajectories tested.

Sparse Interpolation Points

An alternative to tensor product multidimensional polynomial interpolation is the sparse representation of Smolyak [52]. This formulation uses only a subset of the tensor product points to produce approximations that are nearly as accurate. Following their prescription we could reduce the memory footprint and computational cost of our algorithm.

Model Fitting

Finally, different techniques for capturing the interpolated values should be considered. This data could come from physical experiments, or one might, for example, wish to use Poisson’s equation as a starting point. One of the draw backs of the polyhedral method is that incorporating direct measurements of the force field requires updating the physical description of the small body. On the other hand, interpolation techniques should be more apt at making corrections because they are based on data to begin with.
Aside from the speed of polynomial approximations, this is potentially their greatest strength.

**Qualitative Features of Gravity**

The discontinuities across cells do not seem to have a major impact on trajectory propagation. However, it remains to be seen if this discontinuity affects optimization of, say, low-thrust trajectories [9]. Furthermore, the chosen representation is not exact, i.e. it is not the gradient of a potential. Such a representation is desirable for simulations that need gravitational potential and gravitational force.
Chapter 4

Adaptive Approximations that are $C^n$ and Exact

Results

A piecewise polynomial approximation that is $C^n$ and exact is developed. Our first contribution is a method for solving a least squares data fitting problem even when $\phi \in \mathcal{B}_n$, which form the approximation space, are not linearly independent. Then we implement this technique to approximate the gravity of asteroid 1998 ML14. Implementing such a large scale approximation requires extending the data structures of Grinspun [26]. Specifically, our data structures improve memory efficiency during model creation and computational efficiency during model usage. The result is a fast, continuous and exact method for representing gravitational force.

4.1 Continuous and Exact Polynomial Approximation

In the conclusion of chapter 3 it was identified that polynomial interpolation model did not preserve any of the qualitative features of gravity: continuity, exactness or divergence freedom. As noted in chapter 2 these features have practical significance to certain kinds of trajectory simulation. It is the goal of this chapter to explore methods that preserve continuity and exactness.

To create a continuous and exact approximation we will employ the following strategy: i) Exactness will come from modeling potential rather than force. Taking the negated gradient of our model will provide force. In this case exactness is automatic as $\tilde{F} := -\nabla \tilde{U}$. ii) Model $C^1$ continuity by constructing $\tilde{U}$ in a function space that includes only functions with the desired continuity. Put more concretely, choose a function space $S$ with spanning
set $\mathcal{B}$ such that $\phi \in \mathcal{B}$ observes,

$$\phi(x) \in C^1, \quad \phi(x) \text{ and } \nabla \phi(x) \text{ have compact support.} \quad (4.1)$$

An example of such a spanning set $\mathcal{B}$ with elements satisfying these properties is one generated by Basis Refinement of a B-spline basis.

### 4.2 Basis Refinement

Basis Refinement, or CHARMS [26], is an adaptive refinement framework developed for physical simulation. In the CHARMS framework adaptive refinement is performed on spanning functions\(^1\), a set of functions that span $S$, rather than the elements that describe the domain. Once the spanning functions are refined the domain elements are adapted to fit the needs of the spanning functions (i.e. for the purpose of integration). Figure 4.1 gives an example of refining degree 1 B-splines in 1D.

![Figure 4.1: Example of refining a degree 1 B-spline. The top B-spline is a linear combination of the bottom B-splines. From left to right the coefficients would be 1/2, 1 and 1/2.](image)

The refinement relation for spanning functions $\phi \in \mathcal{B}$ is the fundamental unit of adaptivity in CHARMS. A refinement relation gives a recipe for representing $\phi$ with a linear combination of translated and scaled versions of $\phi$, known as $\phi$'s children $\mathcal{C}(\phi)$; see figure 4.1. At the $n^{\text{th}}$ iteration of refinement the spanning set $\mathcal{B}_n$ is constructed by replacing some elements $\phi \in \mathcal{B}_{n-1}$ with

\(^1[26]\) refers to these functions as scaling functions.
$C(\phi)$. Since linear combinations of $C(\phi)$ can represent $\phi$, linear combinations of $B_n$ can represent $B_{n-1}$. Thus, the approximation space of the $n^{th}$ iteration $S_n$ always contains the previous space: $S_n \supseteq S_{n-1}$. Note that this procedure does not always produce a basis. For some applications – including this one – the linear independence property of a basis is not necessary. In this case $B_n$ are merely spanning sets of the space $S_n$.

The advantage of CHARMS is that one does not need to develop machinery for handling discontinuities at T-junctions. (See figure 3.7 for an example of a T-junction discontinuity.) T-junctions are seamlessly handled by the virtues of the spanning set: if its span does not contain discontinuous functions then no discontinuities can arise. Furthermore, as long as the refinement observes the continuity conditions no subsequent spanning set will permit discontinuities. Thus, CHARMS is the foundation upon which we build a continuous adaptive approximation.

Before going into more detail we must introduce our spanning function $\phi$, the B-spline Basis function.

### 4.3 B-spline Basis

Our presentation will begin with a review of the 1-dimensional case [14, 15], followed by a focused treatment of B Splines in $K$ dimensions [43]. At the end a brief discussion of other $K$-dimensional B-spline techniques is presented.

#### 4.3.1 1D B-spline Basis

Let $\tau := \{t_j\}$ be a nondecreasing sequence in $\mathbb{R}$ with $N$ elements, $t_0$ to $t_{N-1}$, called the knot vector. The $j^{th}$ B-spline of order $k$ for the knot vector $\tau$ is designated $B_{j,k,\tau}$, and defined as,

$$
B_{j,0,\tau}(x) := \begin{cases} 
1 & \text{if } t_j \leq t < t_{j+1} \\
0 & \text{otherwise,}
\end{cases}
$$

$$
B_{j,k,\tau}(x) := \frac{t - t_j}{t_{j+k} - t_j} B_{j,k-1,\tau}(t) + \frac{t_{j+k+1} - t}{t_{j+k+1} - t_{j+1}} B_{j+1,k-1,\tau}(t). \tag{4.3}
$$

Figure 4.2 diagrams several B-splines with various knot vectors and of various orders. Notice that in each case at least $k + 2$ knots are required to define
B_{j,k,\tau}. Additionally, every knot beyond the first \( k + 2 \) defines one additional \( B_{j,k,\tau} \). Put in another way, there are \( N - (k + 1) \) B-splines in \( \tau \) of length \( N \). Thus, the set of B-splines produced by a knot vector \( \tau \) is \( \{ B_{j,k,\tau} : j \in \{1, \ldots, N - (k + 1)\} \} \), which we will abbreviate henceforth to \( \{ B_{j,k,\tau} \} \).

For the moment let us consider a knot vector \( \sigma \) small enough to define only one B-spline function, i.e. \( \sigma := \{ t_j \} \) for \( j \in [0, k + 1] \subset \mathbb{Z} \). (\( j \) and \( k \) are implicitly defined when \( \sigma \) is exactly the right size: \( j := 0 \) and \( k := |\sigma| - 2 \).) Henceforth we denote this special case by \( B_\sigma \), and \( \sigma \) will always refer to a knot vector just big enough to define one B-spline\(^2\) [48]. The continuity of \( B_\sigma \) at \( t_j \) depends on the multiplicity of \( t_j \). Let \( v_j \) be the multiplicity of \( t_j \), then \( B_\sigma \) at \( t_j \) is \( C^{k-v_j} \) and smooth everywhere else. For example, a B-spline of degree 3 with knots of multiplicity 1 is \( C^2 \) at the knots and smooth elsewhere.

Figure 4.2 gives several more examples.

Now consider the vector space \( S \) spanned by \( B := \{ B_\sigma \} \), where \( \sigma_j \) is one knot vector in a collection of knot vectors. In general \( S \) is a vector space, but \( B \) is not a basis [48, 49]; fortunately we can restore the basis property by looking at a restricted case. Fix \( k \) and consider a knot vector \( \tau \) with \( N > 2k + 1 \) elements. From \( \tau \) we can define the B-spline basis \( B = \{ B_{j,k,\tau} \} \), which forms a well known vector space: the spline space \( \mathbb{P}_{k,\tau} \) [15]. Note that we can always return to the previous notation by splitting \( \tau \) into \( N - (k + 1) \) smaller \( \sigma_j \); but for the moment let us continue with \( \tau \) and explore \( \mathbb{P}_{k,\tau} \).

The space \( \mathbb{P}_{k,\tau} \) is of piecewise polynomials of order \( k \) covering \([t_{k+1}, t_{N-k}] \subset \mathbb{R}\) with breaks at \( t_j \); see figure 4.3, which diagrams elements from different \( \mathbb{P}_{k,\tau} \) and their respective basis. Note that we ignore the first and last \( k \) intervals in \( \tau \) when defining \( \mathbb{P}_{k,\tau} \). This is necessary to “bootstrap” the spline: For example, in figure 4.3 (b) notice that in the third and fourth intervals 3 elements of the basis have support. As 3 linearly independent quadratics are necessary to cover all possible quadratic polynomials, only those intervals have sufficient coverage to be considered part of \( \mathbb{P}_{k,\tau} \). Frequently this bootstrap process is handled by starting and ending the knot vector with a knot of multiplicity \( k + 1 \), but this is not strictly necessary [15]. Just as before, the degree of continuity between breaks in \( \mathbb{P}_{k,\tau} \) is defined by the multiplicity of \( t_j \), as figure 4.3 shows.

\(^2\)This formulation is known as a point based spline and developed in more detail by Sederberg et al. [48].
Figure 4.2: Several examples of B-splines. × below the axis indicate locations of knots and the presence of multiple knots. From top to bottom: 

(a) An order 2 B-spline, the 4th knot has multiplicity 2 making the breakpoint only \( C^0 \). 
(b) An order 3 B-spline using the same knot vector as (a). The \( C^0 \) discontinuity is now \( C^1 \) and one less B-spline is defined. 
(c) An order 3 B-spline with knots of multiplicity 1 and uniform spacing.
Figure 4.3: Several examples of the space $\mathbb{P}_{k,\tau}$. The shaded green region marks the domain of $\mathbb{P}_{k,\tau}$. a) Degree 1 space with uniform knots. b) Degree 2 space with a knot of multiplicity 2. c) Degree 3 space with uniform knots of multiplicity 1.
B-spline Refinement

For B-splines to operate in the CHARMS framework they must observe a refinement relation, and, in general, they do not. Although any $B_\sigma$ can be represented with a linear combination of finer $\{B_{j,k,\tau}\}$, this process does not always yield a refinement relation because $\{B_{j,k,\tau}\}$ may not be made by translating and scaling $B_\sigma$.\(^3\) Fortunately one common case does keep a refinement relation: $\sigma$ with uniform knot spacing and knot multiplicity 1. To refine $B_\sigma$ add a knot between every pair of knots in $\sigma$; more precisely, insert $(t_{j+i} + t_{j+i+1})/2$ after $t_{j+i}$ in $\sigma$ for $i \in 0, 1, \ldots, k$; see figure 4.4.

Figure 4.4: Refinement relation for a degree 3 B-spline. × below the axis indicate locations of knots. In this refinement every pair of knots in the original is split by a new knot half way between them. The result is a new set of B-splines made by translating and scaling of the original.

Refinement rules for a single B-spline can be applied to any element of $\mathcal{B}$, but doing so does not produce a new $\mathbb{P}_{k,\tau}$, in fact the new spanning set is not even a basis. At this point we abandon $\mathbb{P}_{k,\tau}$ in favor of our looser framework where each element of $\mathcal{B}$ is individually defined by a unique $\sigma$. Our reason for introducing $\mathbb{P}_{k,\tau}$ was to describe the familiar space in which our algorithm starts, but once refinement begins a new notation is required. To this end we return to the CHARMS notation: the space at iteration $n$ is $S_n$ and its spanning set is $\mathcal{B}_n$. For example, $S_0 = \mathbb{P}_{k,\tau}$ and $\mathcal{B}_0 = \{B_{k,j,\tau}\}$.

\(^3\)A necessary and sufficient condition is that $\tau$ contain all the knots of $\sigma$. Proof: The only difference between $\sigma$ and $\tau$ is the presence of “extra” knots. Remove these knots by picking coefficients that enforce “not-a-knot” [15]. Sederberg et al. [49] gives a formula for the coefficients for degree 3 B-splines in section 4.1.
4.3.2 B-spline Basis in $K$-Dimensions

Multidimensional spline theory is an active area of research. In fact, even the fundamental properties that define a multidimensional spline are in active debate. We shall discuss briefly some of these ideas at the end of this section, but for now we will only consider the simplest multidimensional B-spline: the tensor product B-spline.

The $K$-dimensional tensor product B-spline $B_{\tau}$ is defined by the product of $K$ 1-dimensional B-splines – each with an individual knot vector – where each spline handles one dimension:

$$B_{j,k,\tau}(x) = \prod_{d=1}^{K} B_{j_d,k,\tau_d}(x_d).$$

(4.4)

In equation 4.4 $\tau$ is simply a list of knot vectors, one for each dimension, and $\tau_d$ is the knot vector for dimension $d$. Similarly, $j$ is a multi-index where $j_d$ specifies using the $j_d^{th}$ B-spline of $\tau_d$ for dimension $d$. For example, in 3 dimensions equation 4.4 is

$$B_{j,k,\tau}(x) = B_{j_1,k,\tau_1}(x_1)B_{j_2,k,\tau_2}(x_2)B_{j_3,k,\tau_3}(x_3).$$

Multidimensional B-splines bases constructed via tensor product share many traits with their 1-dimensional counterpart. First, constructing the multidimensional $P_{k,\tau}$ follows the same procedure: Fix $k$ and choose $\tau$ such that each $\tau_d$ has $N > 2k + 1$ elements. Then $B = \{ B_{j,k,\tau} \}$ will define a basis that spans all piecewise polynomial patches over the $K$-dimensional patch $\prod_{d=1}^{K} [t_{d,k+1}, t_{d,N-k}] \subset \mathbb{R}^K$, where $t_{d,j}$ is the $j^{th}$ element of the knot vector $\tau_d$.

Refinement in multiple dimensions treats each dimension independently; and in our case we refine each dimension by halving the distance between knots in the knot vector, as described at the end of 4.3.1. The second experiment in section 4.4.2 gives an example of refinement in 2 dimensions. We denote subsequent spaces of refined $K$-dimensional B-splines with the $S_n$ and $B_n$ notation.

Now we can define the spanning functions $\phi$ which constitute a spanning set $B_n$ for an adaptive approximation of gravitation. Let $\phi$ be a 3-dimensional B-spline of degree $k = 3$ with uniformly spaced knots of multiplicity 1. Mod-
eling potential with \( \sum_i \alpha_i \phi_i \) means force is modeled by \( \sum_i \alpha_i \nabla \phi_i \). Thus, the modeled force \( \hat{F} \), having lost one degree of continuity from differentiation, is \( C^1 \) and exact.

**Other Spline Methods for \( K \) Dimensions**

In 1 dimension splines have two important properties that make them valuable for approximation: as piecewise polynomials they are simple to manipulate, and as solutions to certain variational problems they guarantee bounds on their curvature [4]. Unfortunately, when in \( K \) dimensions both properties cannot be preserved by the same theory. As such \( K \)-dimensional spline theory has grown from two seeds: piecewise polynomials [15, 16, 43], and approximations that minimize curvature [3, 4].

Creating \( K \)-dimensional splines from piecewise polynomials leads to the development of Box splines and Simplex splines [16, 43]. These theories construct the \( K \)-dimensional equivalent to a B-spline: a \( C^n \) piecewise polynomial in \( K \) dimensions that has compact support. The tensor product B-spline we use is a special case of Box splines. Starting from the supposition that a spline minimizes some energy in a Hilbert space produces the theory of minimizing splines, or \( D^m \)-splines [3, 4]. This theory produces thin plate splines and the multiquadric.

### 4.3.3 Quick Evaluation of B-splines and Their Derivative

To evaluate B-splines efficiently we use de Boor’s Algorithm [15]. de Boor’s Algorithm’s efficiency comes from leveraging redundant computations between nearby B-splines: Consider evaluating a degree \( k \) B-spline from the recursive definition in equation 4.3. Starting from the bottom of the recursion, at degree 0, only one B-spline is non-zero. At degree 1 two B-splines are non-zero, and both derive their value from the 0th degree B-spline with non-zero value. At degree \( k \) there are \( k+1 \) non-zero B-splines, and they derive their values from \( k \) degree \( k-1 \) B-splines. In this way, the B-splines spread their value to subsequent levels in a tree; see figure 4.5. Evaluating \( B_{j,k,\tau} \) with equation 4.3 computes each path from \( B_{j,k,\tau} \) to \( B_{j,0,\tau} \) independently, repeating many calculations along the way. De Boor’s Algorithm computes each
element of the pyramid once, thereby avoiding these redundant calculations.

Figure 4.5: A tree diagraming de Boor’s Algorithm. The central nodes are non-zero contribution’s to the levels beneath. The grayed nodes are zero terms of equation 4.3. Arrows indicate the direction of contribution.

The derivatives of degree $k$ B-splines are simply linear combinations of degree $k - 1$ B-splines [12]:

$$
\frac{d}{dx} B_{j,k,\tau}(x) = \frac{k}{t_{j+k} - t_j} B_{j,k-1,\tau}(x) - \frac{k}{t_{j+k+1} - t_{j+1}} B_{j+1,k-1,\tau}(x).
$$

Since de Boor’s Algorithm evaluates the $k - 1$ degree B-splines to compute $k^{th}$ degree B-splines, the derivatives are easily computed in line with the rest.

The appendix gives pseudo-code that evaluates uniformly spaced B-splines of degree $k$, and its first derivative.

## 4.4 Approximation Using B-spline Basis and Basis Refinement

At the confluence of high order multidimensional B-splines and Basis Refinement lie possible approximation methods. This section begins with a discussion of the idiosyncrasies in our approximation space $S_n$. Then an approximation method is proposed and tested with small experiments.
4.4.1 Linear Independence of $\mathcal{B}_n$

Refinement by substitution with high order B-splines does not guarantee linear independence of the spanning set $\mathcal{B}_n$ [26, 34]. Even in 1 dimension simple refinements can lead to linear dependence in $\mathcal{B}_n$. Figure 4.6 gives several 1-dimensional examples.

As figure 4.6 indicates, the tricky cases are between different levels of refinement – where each level is the B-splines sharing a common knot spacing. Linear independence between levels can be guaranteed with additional bookkeeping [34], but for function approximation the extra effort is not necessary. In fact, all we need for function approximation is linear independence within each level of $\mathcal{B}_n$. Within a level the only difficulty is to capture the repeated children between nearby B-splines. As figure 4.6, (a) (compare the second and third row) shows, the adjacent B-splines contribute to the children of one another. The simplest way to avoid over adding a given child is to record the included spanning functions of each level of $\mathcal{B}_n$ with a bitmap or set [26].

The problem with the bitmap approach is that it is not adaptive: every function, included or otherwise, is explicitly represented. A set is adaptive, but in practice a set data structure is too inefficient to store individual spanning functions. We will revisit this problem in section 4.5.

The following sections describe an approximation method that does not require linear independence between the levels of $\mathcal{B}_n$.

4.4.2 Least Squares Approximation

Let $U : D \subset \mathbb{R}^3 \rightarrow \mathbb{R}$ and pick a basis $\mathcal{B} = \{\phi_j\}$ for some function space covering $D$. A least squares approximation chooses coefficients to minimize the distance between $U$ and its approximation in $\mathcal{B}$ given some norm. In many applications – particularly scattered data interpolation [19] – a discrete norm is made by the collected data to be fit [15]. In our application data points can be placed where ever convenient. This gives us the flexibility to pick any norm we want, e.g. $L^2$ or $H^1$. Before going further let us formalize the minimization problem.

---

4For example, the C++ Standard Template Library set has 24 bytes of overhead per function (3 pointers). In our models that would cost about 860 MB of overhead. Given the functions themselves cost about 860 MB (3-dimensional location) we would waste over 1.5 GB.
Figure 4.6: Examples where refinement does not produce a basis. In each diagram the top row shows $\mathcal{B}_0$; red B-splines are to be refined and black B-splines are not.  

\(a\) The second and third rows are the left and right refinements respectively, and the final row shows the combination of these refinements. Green shading indicates the support of the middle spanning function in $\mathcal{B}_0$, notice the shaded region is completely covered by a basis capable of representing the unrefined B-spline. 

\(b\) This example shows that restricting the domain of the approximation can make otherwise linearly independent examples, linearly dependent. The green shaded region is the domain of the approximation. The refinement of the red functions would not normally completely represent the black function because the children of the red functions do not cover the support of the black function – which is depicted as the union of red and green shaded regions. In this case, however, the restriction to the green region means that they do.
Keep $U$ and $B$ as described above and pick an inner product space in which we wish to minimize $\|U - \sum_j \alpha_j \phi_j\|$, or equivalently

$$\|U - \sum_j \alpha_j \phi_j\|^2,$$  \hspace{1cm} (4.5)

where $\|\cdot\|$ is the norm induced by the inner product $\langle \cdot, \cdot \rangle$. First we rewrite the quantity in 4.5 in terms of the inner product,

$$\|U - \sum_j \alpha_j \phi_j\|^2 = \langle U - \sum_j \alpha_j \phi_j, U - \sum_j \alpha_j \phi_j \rangle,$$

$$= \langle U, U \rangle - 2\langle U, \sum_j \alpha_j \phi_j \rangle + \langle \sum_j \alpha_j \phi_j, \sum_j \alpha_j \phi_j \rangle,$$

$$= \langle U, U \rangle - 2\sum_j \alpha_j \langle U, \phi_j \rangle + \sum_j \sum_k \alpha_j \alpha_k \langle \phi_j, \phi_k \rangle.$$

The last two manipulations are by bilinearity of inner product. To minimize this with respect to $\alpha_i$ we take the derivative and set it to zero,

$$\frac{\partial}{\partial \alpha_i} \left[ \langle U, U \rangle - 2\sum_j \alpha_j \langle U, \phi_j \rangle + \sum_j \sum_k \alpha_j \alpha_k \langle \phi_j, \phi_k \rangle \right] = 0.$$

The first term obviously goes to zero, the second term remains though only one element of the sum (the one multiplied by $\alpha_i$) persists, and the last term is differentiated using the product rule. This produces,

$$2\sum_k \alpha_k \langle \phi_i, \phi_k \rangle = 2\langle U, \phi_i \rangle,$$

$$\implies \sum_k \alpha_k \langle \phi_i, \phi_k \rangle = \langle U, \phi_i \rangle.$$  \hspace{1cm} (4.6)

Equation 4.6 defines a linear system, $Mx = b$, where

$$M_{i,j} = \langle \phi_i, \phi_j \rangle,$$  \hspace{1cm} (4.7)

$$b_i = \langle U, \phi_i \rangle.$$  \hspace{1cm} (4.8)

In this linear system $M$ is often called a mass matrix.

Solving this linear system can also be phrased as a projection operation. In this case we project $U$ down to $S$ using the inner product $\langle \cdot, \cdot \rangle$.

The next natural question is with what norm shall we minimize $\|U -$
\[ \sum_j \alpha_j \phi_j \|. \] For our application we wish to approximate both \( U \) and \( F \); thus, the most obvious choice is the \( H^1 \) norm,

\[
\langle f, g \rangle = \int_\Omega f(x)g(x) + \sum_i \frac{\partial}{\partial x_i} f(x) \frac{\partial}{\partial x_i} g(x) dx ,
\]

\[\|f\| = \sqrt{\langle f, f \rangle} , \] (4.9)

as it incorporates both gravitational potential and force. Alternatively, we could choose the \( L^2 \) norm,

\[
\langle f, g \rangle = \int_\Omega f(x)g(x) dx ,
\]

\[\|f\| = \sqrt{\langle f, f \rangle} .\] (4.10)

In section 4.6.2 we will explore both possibilities.

Implementing either inner product requires numerical integration. Our approximation uses cubic polynomials; thus \( \langle \phi_i, \phi_j \rangle \), i.e. creating the mass matrix, will necessitate integrating degree 6 polynomials. A degree 6 polynomial is exactly integrated with Gauss-Legendre quadrature of order 4 [25], making it a natural choice for creating the mass matrix. The right hand side, \( \langle U, \phi_i \rangle \), will also use order 4 Gauss-Legendre quadrature. We justify this decision with our goal: to approximate gravitation with cubic polynomials. Therefore, order 4 Gauss-Legendre quadrature will be accurate for the right hand side at the same time as the approximation is accurate. Finally, the integration domain, \( \Omega \), is split into a hexahedral mesh that mimics the break points of the piecewise polynomials defining \( \phi_i \). Whenever \( \phi_i \) overlaps with another spanning function \( \phi_i \) from a finer level the hexahedral mesh follows the break points of \( \phi_j \). This hexahedral mesh defines the quadrature domains over which we evaluate equations 4.7 and 4.8.

### 4.4.3 Solving \( B_n \) with Linear Dependence

This process works very well for tensor product B-splines, but will fail once our refinement process begins as the mass matrix is singular when \( B_n \) has linear dependence. To accommodate linear dependence between levels of \( B_n \) we will solve \( B_n \) hierarchically: Each \( B_n \) is formed of \( P \) nested tiers \( T^p \) for \( 0 \leq p \leq P \leq n \). They are nested in the following sense: \( T^0 \) is the coarsest
and contains exactly the spanning functions necessary to cover the domain \( \Omega \). \( T^p \) contains spanning functions from the \( p^{th} \) level that are refinements of spanning functions unique to \( T^{p-1} \), and everything else from \( T^{p-1} \). In other words,

\[
\phi \in T^p \implies \phi \in T^{p-1} \lor (\exists \phi' \in T^{p-1} \land \phi \in C(\phi') \land \phi' \notin T^{p-2}),
\]

where \( T^{-1} := \emptyset \). Although \( B_n \) is defined similarly to \( T^p \), they are not the same. In \( B_n \) there is no restriction on which spanning functions from \( B_{n-1} \) are refined; however, \( T^p \) may only refine \( \phi \in T^{p-1} \) that are the result \( p-1 \) refinements. Figure 4.7 gives a 1-dimensional example, and demonstrates the difference between \( B_n \) and \( T^p \).

To find coefficients for \( B_n \) we iterate through \( T^p \), solving one tier at a time. First we solve \( T^0 \) using equations 4.7 and 4.8. To solve \( T^p \) assume we know the coefficients of \( T^{p-1} \). Then separate \( T^p \) into \( N \) and \( R \) such that,

\[
\phi_j \in N \implies \phi_j \in T^{p-1}, \\
\phi_j \in R \implies \exists \phi' \in T^{p-1} \land \phi_j \in C(\phi').
\]

In other words, \( R \) are spanning functions that come from refining \( T^{p-1} \), and \( N \) are the spanning functions we carry without refinement from one tier to the next. Organize \( \phi_j \in T^p \) such that \( \phi_j \in R \) when \( j < |R| \) and \( \phi_j \in N \) otherwise, i.e. the first \( |R| \) spanning functions of \( T^p \) are the new spanning functions, and the rest are from the previous tier, \( T^{p-1} \). Next, create the mass matrix \( M \) of \( T^p \) as usual, equations 4.7; and remove the spanning functions that come from earlier levels by taking their coefficients from the solution to the previous tier, \( T^{p-1} \). To do this remove the last \( |N| \) rows of \( M \) and \( b \), as these equations have already been solved in the in \( T^{p-1} \). This leaves us with under determined system \( \tilde{M}'x = \tilde{b} \). Using the coefficients from \( T^{p-1} \), we remove the last spanning functions in \( N \) by subtracting their contribution from the right hand side; thus, creating \( \tilde{b} \). Finally, we solve
Figure 4.7: Examples of $\mathcal{T}^p$ that constitute $\mathcal{B}_n$. Red B-splines are to be refined at the next tier. Notice the difference between $\mathcal{T}^1$ and $\mathcal{B}_1$ in the third iteration. This demonstrates that $\mathcal{B}_n$ may refine any spanning function from $\mathcal{B}_{n-1}$, but $\mathcal{T}^p$ only refines those spanning functions unique to $\mathcal{T}^{p-1}$. 
\( Mx^p = \bar{b} \). Expressed as equations the process is:

1) \( \bar{M}'_{i,j} = M_{i,j} \) for \( i < |\mathcal{R}|, j < |\mathcal{T}^p| \),
2) \( \bar{b}_i = b_i - \sum_j \bar{M}'_{i,j}x^{p-1} \) for \( i < |\mathcal{T}^p|, j < |\mathcal{T}^p| \),
3) \( \bar{M}_{i,j} = \bar{M}'_{i,j} \) for \( i < |\mathcal{R}|, j < |\mathcal{R}| \),
4) \( x^p = M^{-1}\bar{b} \),

where \( x^{p-1} \) is a vector of zeros for the first \( |\mathcal{R}| \) entries and coefficients gleaned from solving \( \mathcal{T}^{p-1} \) for the last \( |\mathcal{N}| \) entries. Finally, the coefficients for \( \mathcal{T}^p \) are created by replacing the \( |\mathcal{R}| \) zeros in \( x^{p-1} \) by \( x^p \). This process is continued until every tier \( \mathcal{T}^p \) of \( \mathcal{B}_n \) is solved, the coefficients of \( \mathcal{T}^{P-1} \) become the coefficients for \( \mathcal{B}_n \).

It is worth noting that not only does this process permit linear dependence among the levels of \( \mathcal{B}_n \), it also reduces the size of any given linear system solve: In practice we never need the last \( |\mathcal{N}| \) rows of \( M \), so they should never be created. This savings can as much as halve the total size of the mass matrix being solved, which is not insignificant. For example, the final iteration of our model required 69 GB, doubling this makes for a very large matrix indeed. See section 4.5.2.

**Numerical Experiments**

Now we perform some simple numerical tests to demonstrate that this hierarchical solve works even in the presence of linear dependence. For this simple demonstration we will focus on approximating \( U \) rather than \( F \), leaving that discussion to a much more detailed analysis in section 4.6. Two tests are performed: a simple 1-dimensional problem and then a 2-dimensional problem.

In both cases we use \( U = \frac{1}{||x||} \) and the \( L^2 \) norm. In the 1-dimensional case we model the domain \( \Omega = [2, 3] \), and in 2 dimensions we use \( \Omega = \prod_{i=1}^2 [2, 3] \). In either case \( \phi_i \in \mathcal{B}_0 \) uses uniform knot spacing of 1 unit. For the 1-dimensional test \( \mathcal{B}_1 \) is created by refining every spanning function whose knot average (i.e. center knot) is \( \leq 3 \); this creates the second example in figure 4.6. For the 2-dimensional case we create \( \mathcal{B}_1 \) by refining every basis whose knot average’s \( x \)-coordinate is 2 or 3. Then we create \( \mathcal{B}_2 \) by refining \( \phi \in \mathcal{T}^1 \subset \mathcal{B}_1 \) whose knot average’s \( x \)-coordinate is 2; figure 4.8 diagrams the setup.
Figure 4.8: 2-dimensional least squares experiment setup. Crosses mark locations of the peaks in each B-spline spanning function: Red crosses mark elements at the first level of refinement, green crosses are the second level, and blue ones are the last level. The center green square marks the region of approximation.
These experiments were run using the hierarchical solve and a flat solve using Conjugate Gradient [45]. Table 4.1 gives the maximum relative error between $U$ and $\hat{U}$ after each iteration, and the condition number of the matrices solved in each experiment. Note that the condition numbers for the flat solve are numerically singular when $n > 0$. In both cases hierarchical solves reduce the maximum error after a step of refinement; on the other hand, in 2 dimensions flat solves actually increase maximum error after the second round of refinement. It is also worth noting that during first round of refinement the flat solve produces a slightly better result. This indicates the optimal coefficients for previously solved levels change as additional levels are added; we will return to this observation in section 4.7. In 1 dimension the flat solver finds a similar solution to the hierarchical solve, as depicted in figure 4.9. This reflects the fact that poor conditioning merely means you should not trust your solution, not that it is necessarily defective.

Table 4.1: Comparison of Hierarchical and Flat Solve

<table>
<thead>
<tr>
<th>$K$</th>
<th>$B_n$</th>
<th>Flat Conditioning</th>
<th>Max Rel. Error</th>
<th>Hierarchical Conditioning</th>
<th>Max Rel. Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$B_0$</td>
<td>$6.97 \times 10^4$</td>
<td>$3.79 \times 10^{-4}$</td>
<td>$6.97 \times 10^4$</td>
<td>$3.79 \times 10^{-4}$</td>
</tr>
<tr>
<td></td>
<td>$B_1$</td>
<td>$4.59 \times 10^{17}$</td>
<td>$7.87 \times 10^{-5}$</td>
<td>$2.31 \times 10^3$</td>
<td>$7.84 \times 10^{-5}$</td>
</tr>
<tr>
<td>2</td>
<td>$B_0$</td>
<td>$4.85 \times 10^7$</td>
<td>$8.48 \times 10^{-5}$</td>
<td>$4.85 \times 10^7$</td>
<td>$8.48 \times 10^{-5}$</td>
</tr>
<tr>
<td></td>
<td>$B_1$</td>
<td>$6.47 \times 10^{19}$</td>
<td>$1.56 \times 10^{-5}$</td>
<td>$5.32 \times 10^6$</td>
<td>$1.73 \times 10^{-5}$</td>
</tr>
<tr>
<td></td>
<td>$B_2$</td>
<td>$5.79 \times 10^{19}$</td>
<td>$4.92 \times 10^{-5}$</td>
<td>$1.57 \times 10^6$</td>
<td>$5.54 \times 10^{-6}$</td>
</tr>
</tbody>
</table>

4.4.4 Interpolation

Another B-spline approximation technique commonly used in 1 dimension is interpolation. In B-spline interpolation abscissa are selected such that when $B = \{B_{j,k,r}\}$ interpolates $U$ at the abscissa, $B$ forms a good approximation to $U$. One could apply the hierarchical solve described above to develop an approximation based on interpolation; however, picking the abscissa is a crucial step. de Boor [15] notes that least squares approximation is, in effect, an interpolation technique that picks its own abscissa; and goes on to say,

---

5 As measured by 100000 random samples.
6 In this case $B_n$ may be identified with $T^p$ because $B_n$ is refined one tier at a time.
Figure 4.9: A comparison between the 1-dimensional approximations to $1/x$. On the left is the flat solver, on the right is the hierarchical solver. The top row is with the initial B-spline configuration, and the bottom row is after one round of refinement. For this simple example they have very similar solutions. Relative error is plotted on the $y$-axis in log scale.
“In a way, least-squares approximation is a very reasonable way to choose the interpolation points.” For this reason we prefer least square approximation to interpolation.

4.5 Implementation Details

In the subsequent sections we describe implementation details that allow our model to efficiently represent small body gravitation.

4.5.1 Representation of $B_n$

This section focuses on representing $B_n$ in a way suitable to preparing and using our model of small body gravitation. Two concerns dominate the design: memory efficiency during model creation and computation efficiency during force reconstruction. The former is the primary bottleneck of our coefficient fitting stage and dictates the number of iterations of refinement the algorithm will permit. Ideally as much memory as possible is given to the mass matrix, meaning our representation of $B_n$ should be as lean as possible. The latter is the whole reason for doing this in the first place! Fortunately both problems are addressed by the same data structure.

From a pedological point of view our algorithm is more clearly presented using a collection of individual $\phi_j := B_\sigma$; however, from an implementation standpoint representing $B_n$ as a collection of tensor product B-spline patches is advantageous. Each patch $P_i$ is constructed by a tensor product B-spline representing many B-splines, $P_i := \{B_{j,3,\tau}\}$, with a single data structure. For example, to represent a single $B_\sigma$ requires its $K$-dimensional location and the uniform step size of $\sigma$. On the other hand, we can represent any number of B-splines in a cuboid layout with only:

```c
struct Patch {
    double lo [K]
    double hi [K]
    double knot_distance
}
```

where $K$ is the dimension of the patch, and $lo$ and $hi$ are the upper and lower bounds for the support of the patch.

Representing $B_n$ this way has two advantages. First, it makes representing $B_n$ far more compact. For example, the model developed in section 4.6.1
requires 1.5 MB of storage when represented this way; however, representing each \( \phi_i \) explicitly would have required 850 MB. Our second advantage is in evaluating \( B_n \). The most efficient B-spline evaluation procedures compute neighboring B-splines of a knot vector simultaneously, as described in section 4.3.3 [15]. If we represent B-splines with small, isolated knot vectors \( \sigma \) we cannot leverage de Boor [15] B-spline evaluation algorithm. Using the B-spline patch model instead captures redundant computation of neighboring \( \phi_i \) whenever they belong to the same patch.

Working with patches, instead of individual spanning functions (as in Grinspun [26]), creates new bookkeeping challenges. For example, it is no longer reasonable to think about refining individual spanning functions, instead we consider refining patches. The following sections describe algorithms and data structures for efficient use of B-spline patches.

**Refining Patches**

Refining a patch is, semantically, the same as refining each B-spline within the patch individually. As each patch is represented by a tensor product B-spline, the new patch is constructed by inserting an additional knot between the knots forming each knot vector of the tensor product. In terms of our data structure this amounts to dividing the knot distance by 2. However, just as with individual spanning functions of figure 4.6 (a), some patches will share children. To accommodate redundancy in the children patches must be resized to avoid each other, we call this process *patch splitting*. Consider the 2-dimensional example in figure 4.10. From \( a \rightarrow b \) two adjacent patches are refined; oblivious to one another, they generate redundant children. To extinguish the repeated children one patch is selected to avoid the other, e.g. \( b \rightarrow c \). The number of additional patches created by this process – something we wish to keep low – is reduced by always splitting the patch that contains fewer spanning functions.\(^7\)

Therefore, our algorithm for refinement is: \( i \) Loop through patches and refine those that need refinement. \( ii \) Pair-wise compare patches to avoid overlapping refinements using patch splitting. The pair-wise comparison may be efficiently implemented with an octree data structure (see section 3.2.1), but

\(^7\)This heuristic was found to globally produce fewer patches than always splitting the patch that locally creates fewer patches.
Figure 4.10: The patch splitting process. Shaded regions show the support of each patch, and markers show the peaks of the spanning functions. At the start, (a), there are two patches – red and green. Each patch is refined by replacing the present spanning functions with their children, producing (b). A $3 \times 5$ block of spanning functions are redundant, so one patch is split to avoid the redundant region, producing the three patches in diagramed in (c).
\begin{verbatim}
V ← BoundingVolume(L)
if |L| > 1 then
  (Lℓ, Lr) ← Split(L)
  left ← Build(Lℓ)
  right ← Build(Lr)
  data ← ∅
else
  left ← ∅
  right ← ∅
  data ← L
end
return (V, left, right, data)
\end{verbatim}

Function Build(L) Produces a node of the tree.

in our experience the cost of the straightforward \(O(N^2)\) pair-wise comparisons is small compared to, solving a matrix with 20 million unknowns.

Computing Gravity with \(B_n\)

We have already spoken to the favorable efficiency of evaluating the B-splines of \(B_n\) with patches, but we still need a mechanism for finding the patches that affect a given query. For this we will use a bounding volume hierarchy (BVH) [24].

Bounding volume hierarchies are tree data structures for organizing objects with spatial extent. In this sense they fill the same roll as octrees, but there is an important difference: While octrees subdivide space, a BVH subdivides groups of objects. Two recursive operations characterize a BVH: Build and Find. Build takes as input a list \(L\) of objects and creates one node of the tree; see Function Build(L). BoundingVolume finds the upper and lower bounds of the objects in \(L\). Split divides \(L\) into two sublists \(L_ℓ\) and \(L_r\), which form the input to the next level of the tree. A poor split will mean the bounding volumes of the subsequent levels do not shrink rapidly, and result in poor Find performance. Our Split sorts \(L\) by the widest dimension of its bounding volume; then \(L_ℓ\) and \(L_r\) split the sorted \(L\) evenly. Figure 4.11 diagrams the process of building a BVH.

Find recalls all the objects containing a point query by recursively traversing the BVH. At each level Find recursively follows all children that contain the query point. When Find reaches a leaf the object contained is appended to the output. Following the Split described above, the tree will always be
balanced; hence, most queries will only require $O(\lg N)$ effort. Note that the running time will depend on the query and how the bounding volumes are organized within the tree. For example, a perfectly bad Split can force Find to visit every node in the BVH.

To evaluate $B_n$ efficiently its constituent patches are organized in a BVH. The bounding volume of a patch $P$ is the $\cup \text{supp} [\phi_i]$ for $\phi_i \in P$. Find retrieves the relevant patches with the BVH and each patch is evaluated.

### 4.5.2 Model Creation

This section concludes the description of model creation.

#### Domain Representation and Creating $B_0$

The domain of our approximation, $\Omega$, plays three roles during model creation: i) It provides the domain over which we evaluate our inner product (equation 4.9); ii) seeds $B_0$ with B-spline patches necessary to cover $\Omega$; and iii) it defines the regions over which we will measure the error in our approximation. The simplest solution is to represent the region around an asteroid as a cube, but this needlessly includes the interior of the asteroid. To avoid as much of the interior as possible we use an octree (see section 3.2.1), and Subdivide whenever a cuboid region intersects the asteroid. To avoid subdividing forever a lower bound on the cuboid size is provided.

Once a domain octree $\Omega$ has been constructed $B_0$ is created to cover $\Omega$. Each octree cuboid is considered in isolation and a patch is placed to cover it with a basis for $P_{k,\tau}$. This creates redundant B-splines near the boundaries of octree cells, which are removed using patch splitting. See figures 4.12, and 4.10 b and c.

#### Matrix Representation and Iterative Solver

The matrices $\tilde{M}'$ and $\tilde{M}$ in section 4.4.3 are very big. By the final iteration in our experiments $\tilde{M}$ had 18 million rows and each row had, on average, 310 non-zero entries. A mass matrix of this size requires 86 GB stored naïvely. For our implementation we used the Portable, Extensible Toolkit for Scientific Computation (PETSc) [5, 6] to store and solve the linear system. Using PETSc’s Compressed Sparse Row format our matrix was 69 GB. Additional
Figure 4.11: A bounding volume hierarchy. The objects organized are the colored boxes. On the left is a geometric view. The first level shows the bounding box in black, the second level shows it in black and gray, and the last level only the underlying objects remain. On the right is a tree view of the bounding boxes.
Figure 4.12: Detailed view of initializing $\mathcal{B}_0$. Left: $\Omega$ is the interior of the two boxes. Right: Patches are placed so the domain of each patch covers the cuboid it came from. This creates redundancies, which the overlapping markers (crosses and squares) show. Shaded regions show the domain of each patch’s $\mathcal{P}_{k,\tau}$.

savings could be made using their block symmetric format, but in our experiments the next iteration would still have been out of reach even after a savings of half.

The linear system we solve is symmetric positive definite; therefore, Preconditioned Conjugate Gradient [45, 50] is a reasonable choice for solving our system. For a pre-conditioner we used symmetry preserving diagonal rescaling; i.e. we solved

$$D^{-1}\tilde{M}D^{-1}x' = D^{-1}\tilde{b},$$

and scaled the result,

$$x = D^{-1}x'.$$

$D$ in these equations is the matrix of zeros everywhere but the diagonal, where it is the square root of the diagonal of $M$.

**Error Estimation and Creating $\mathcal{B}_{n+1}$**

To estimate relative error we sample the octree domain $\Omega$. Within each cuboid $\omega \in \Omega$ the error is randomly sampled at a rate of one sample every $10$ m$^3$, which is about $10 \times$ fewer samples than used for the densest error estimates in the cubetree model (chapter 3). Evaluating the exact gravitation is by far the dominant time cost of our algorithm, and fewer error samples
does not appear to change the model significantly. Any time the relative error of a sample in $\omega$ exceeds some threshold every patch intersecting $\omega$ is marked for refinement. Any time no samples in $\omega$ exceed the threshold $\omega$ is removed from the list of domains to check at the next iteration.

At the end of the error checking process all patches requiring refinement are refined and patch splitting keeps them from producing redundant spanning functions. Still, there lurks another form of redundancy: Refined patches produce new children that leave $\Omega$ completely; for example the far left B-spline in figure 4.6 (b), this B-spline has no support in the green shaded $\Omega$. One way to deal with these is patch splitting, but it is far simpler (and almost completely free) to leave them in $B_n$. As they are not part of $\Omega$ they will produce a 0 row in $\bar{M}$ and 0 in $\bar{b}$. PETSc’s diagonal scaling preconditioner automatically replaces 0 rows with the corresponding row of an identity matrix, so recording the extra spanning function is virtually free. By comparison, patch splitting to precisely fit $\Omega$ was found to dramatically increase the number of patches, which complicates the BVH and slows performance.

4.6 Numerical Experiments with 1998 ML14

Now we turn our attention to approximating the gravitational potential and gravitational force of 1998 ML14.

4.6.1 Notes on Model Creation

The domain of our approximations began at (-1250, -1250, -1250) m and extended 2500 m in each direction, where the origin is the center of mass of the asteroid model. (As a point of reference the radius of 1998 ML14 is $\approx 500$ m.) The smallest allowed cube in the octree domain was $2500/2^7 \approx 19.5$ m to an edge. We also limited the largest cube to $2500/2^4 \approx 156$ m to an edge. The relative error threshold for refinement was set to $5 \times 10^{-7}$, which is well beneath the requirement for mission design [10]. The Conjugate Gradient solver’s relative convergence tolerance [7] was set to $10^{-16}$. This model was created in 4500 CPU hours on a parallel computer using the Message Passing Interface (MPI) [53] (64 processors for approximately 80 hours) and occupies 210 MB of memory. $B_0$ is composed exclusively of patches with knot spacing
2500/2^7 m, \( B_2 \) ends the iterations with a mix of knot spacings at 2500/2^9 m to 2500/2^7 m. To cover the region outside the octree spherical harmonics of degree and order 12 were employed. We name this model the Patches of Uniform B-splines Tree, or pubtree for short.

4.6.2 \( L^2 \) versus \( H^1 \) Norm

Before producing our full scale model we compared the \( L^2 \) and \( H^1 \) projections to one another. The domain, \( \Omega \), was a varying sized cube centered about the \( x \)-axis and beginning at \( x = 1000 \) m from the center of 1998 ML14. \( B_0 \) was created to exactly cover \( \Omega \) with \( 4 \times 4 \times 4 = 64 \) 3-dimensional degree \( k = 3 \) B-splines. Each test measured the maximum relative error in \( \hat{U} \) and \( \hat{F} \) created by \( L^2 \) or \( H^1 \) using 10000 random samples in \( \Omega \). To simulate the effect of refining \( B_0 \) we quartered the edge length of \( \Omega \) in subsequent tests.

Table 4.2 gives the maximum relative errors for each case. In the beginning both norms produce similar results, but as we subdivide \( \Omega \) the \( H^1 \) projection surpasses the \( L^2 \) projection. Ultimately \( H^1 \) performs slightly better for small body gravitation, so it is the route we will take when creating a large scale model.

Table 4.2: Efficacy of \( L^2 \) and \( H^1 \) projection.

<table>
<thead>
<tr>
<th>Approximate size of ( \Omega ) (m(^3))</th>
<th>Error in ( \hat{U} ) ( L^2 )</th>
<th>Error in ( \hat{F} ) ( L^2 )</th>
<th>Error in ( \hat{U} ) ( H^1 )</th>
<th>Error in ( \hat{F} ) ( H^1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>400.</td>
<td>( 2.55 \times 10^{-4} )</td>
<td>( 1.12 \times 10^{-2} )</td>
<td>( 2.55 \times 10^{-4} )</td>
<td>( 1.12 \times 10^{-2} )</td>
</tr>
<tr>
<td>100.</td>
<td>( 1.45 \times 10^{-6} )</td>
<td>( 2.34 \times 10^{-4} )</td>
<td>( 1.44 \times 10^{-6} )</td>
<td>( 2.34 \times 10^{-4} )</td>
</tr>
<tr>
<td>25.0</td>
<td>( 6.19 \times 10^{-9} )</td>
<td>( 3.98 \times 10^{-6} )</td>
<td>( 5.71 \times 10^{-9} )</td>
<td>( 3.88 \times 10^{-6} )</td>
</tr>
<tr>
<td>6.25</td>
<td>( 4.04 \times 10^{-11} )</td>
<td>( 1.28 \times 10^{-7} )</td>
<td>( 2.13 \times 10^{-11} )</td>
<td>( 5.18 \times 10^{-7} )</td>
</tr>
<tr>
<td>1.56</td>
<td>( 5.74 \times 10^{-12} )</td>
<td>( 6.59 \times 10^{-8} )</td>
<td>( 1.26 \times 10^{-13} )</td>
<td>( 1.13 \times 10^{-9} )</td>
</tr>
<tr>
<td>0.39</td>
<td>( 2.01 \times 10^{-12} )</td>
<td>( 8.66 \times 10^{-8} )</td>
<td>( 4.92 \times 10^{-14} )</td>
<td>( 2.86 \times 10^{-10} )</td>
</tr>
</tbody>
</table>

4.6.3 Continuity at T-junctions

Now we test continuity between levels of refinement. In figure 4.13 (a) we plot the error in the cubetree’s force at a T-junction; the discontinuity is readily apparent at 935 m. Compare this to the same plot for a T-junction in our B-spline model, figure 4.13 (b). No discontinuities exist, in fact the
only evidence of a T-junction is a slight shift in the oscillatory pattern at 770 m.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{pic.png}
\caption{Continuity comparison of cubetree and B-spline method. In each case the absolute error in the $x$-component of the force is measured along a 50 m ray. The dashed line at $y = 0$ represents the actual force. As the T-junctions do not occur in the same places for each model, different rays are measured. The left graph shows cubetree, and the right graph shows B-spline method.}
\end{figure}

\subsection{4.6.4 Error Measurements}

This section explores the structure of refinement by plotting the relative error in potential and force. Portions of the domain are examined in figures 4.14 and 4.15, which plot relative error in gravitational potential and gravitational force along the $z = 0$ plane. Note that $\hat{U}$ is about three orders of magnitude more effective than $\hat{F}$. Therefore, for applications where an approximation to potential is not needed a direct approximation to force (as in chapter 3) may be easier to create. A white band, extending approximately 60 m from the surface, is visible near the interface of the asteroid and space. This band is where the force approximation fails to make $10^{-6}$. The same band for $10^{-5}$ error it is 40 m from the surface. Future improvements to patch refinement may shrink the band further, though we feel comfortable with accuracy up to 40 m from the surface.
Figure 4.14: Relative error of pubtree across the complete domain. The interior of the asteroid is not measured and left black. Error is plotted on a log scale. Left: Error in the potential. Right: Error in the force.

Figure 4.15: Relative error of pubtree abutting the asteroid. The interior of the asteroid is not measured and left black. Error is plotted on a log scale. Left: Error in the potential. Right: Error in the force.
Quantitative Comparison to Cubetree

Compared to the model in chapter 3 this model appears to be less accurate despite using similar error bounds. Figure 4.16 plots cubetree force’s relative error in the same region as figures 4.14 and 4.15. This effect comes from the convergence rate of cubetree compared to pubtree: In the cubetree model most cells contain degree 6 or degree 4 polynomials, whereas pubtree only uses degree 3 piecewise-polynomials. Therefore, a single refinement of cubetree has a higher rate of convergence – especially away from the boundary where gravity is smooth. This is illustrated in the plots of figure 4.13. Both plots show the error of $\hat{F}$ along a ray as that ray crosses two levels of refinement. Pubtree approximately halves the error at the refinement whereas cubetree error falls about two orders of magnitude.

Figure 4.16: Relative error in the force produced by cubetree. The interior of the asteroid is not measured and left black. Error is plotted on a log scale. Left: The error over the entire domain; compare this to figure 4.14. Right: Sample of the error in a region abutting the asteroid; compare this to figure 4.15.

Cubetree’s aggressiveness means a direct comparison between pubtree and cubetree in terms of memory footprint and speed is not entirely fair. Also, quantitative comparisons between trajectories of the models will tend to favor cubetree; as we will see in the following section. Of course quantitative improvements over cubetree was not our goal. Furthermore, both models easily achieve the original goal of $10^{-5}$.
Refinement Sequence

Figure 4.17 shows the model developed over 3 iterations. After each iteration regions with high error are reduced. For example, the band of white surrounding the asteroid, which shrinks after each iteration.

Figure 4.17: Relative error over the sequence of refinement. The interior of the asteroid is not measured and left black. Error is plotted on a log scale.

4.6.5 Trajectory Integrations

Our trajectory simulations use the models from section 3.4, and add one additional trajectory type:
Pubtree trajectories are generated using the pubtree force model. Integration is done with Embedded Runge-Kutta Prince-Dormand (order 8,9) method using relative error tolerance $10^{-13}$ and absolute error tolerance $10^{-6}$.

We integrated a family of retrograde orbits starting between 600 m and 1000 m from the asteroid center. Initial positions were chosen close to the equatorial plane, and initial velocities did not contain large components outside the plane. The magnitude of the velocity was clamped to within 0.65 and 0.75 of escape speed. Simulations ran for 5 days of ballistic motion with each model (pubtree, cubetree, augmented polyhedral, reference), where impacting trajectories were thrown out. The position and velocity of the orbiter was recorded every 5 minutes of simulated time.

This experiment was repeated for 993 trajectories. For each trajectory we measured the maximum difference in position and velocity between the cubetree trajectory and reference trajectory. Figure 4.18 is a histogram of the errors in position and velocity. For comparison sake the same histograms for cubetree are presented in figure 4.19. Cubetree’s superior relative error is evident in the histograms, however, the shape of the histograms are similar. On average pubtree trajectories were $270 \times^8$ faster than the augmented polyhedral, compared to $301 \times$ for cubetree.

![Histograms of errors in position and velocity for 993 retrograde pubtree trajectories integrated for 5 days and observed at 5 minute intervals.](image)

Figure 4.18: Histograms of errors in position and velocity for 993 retrograde pubtree trajectories integrated for 5 days and observed at 5 minute intervals.

---

$^8$See section 4.7.
Figure 4.19: Histograms of errors in position and velocity for 993 retrograde
cubetree trajectories integrated for 5 days and observed at 5 minute intervals.

4.7 Discussion

The CHARMS framework, coupled with our hierarchical coefficient fitting,
has produced an adaptive approximation to gravitational potential and force.
The force is exact and continuous, and the accuracy is within the recommenda-
dations set forth by NASA for the vast majority of the domain.

Trajectory Integration

During the trajectory integration experiments we found pubtree was approx-
imately 0.9× the speed of cubetree. However, single evaluation tests actually estimate the speed at 2× cubetree! The discrepancy comes from the integr-
grator’s refinement strategy: On average pubtree trajectories required 1.8×
more samples than cubetree. We believe this comes from the higher ampli-
tude of the oscillations in pubtree’s force model; see figure 4.13. Reducing
the amplitude by tightening the error threshold would resolve this, thereby
increasing the speed of pubtree.

Patch Refinement

As we have described it, pubtree employs patch refinement: if the error is bad in one part of the patch the whole patch is refined. This approach becomes problematic once the regions with high error are smaller than the patches. In this case every iteration will require that complete patches are
refined, effectively destroying adaptivity. To wit, our iterations of pubtree ended exactly for this reason.

The solution to this problem is sub-patch refinement implemented with patch splitting and hierarchical error estimation. First, error estimation should follow an octree structure: Each time a cuboid of the error octree is found to contain error it should be subdivided before being checked again. In this way we localize the error. Then, instead of refining whole patches we split the patch so only its spanning functions affecting the localized error are refined. Following this algorithm would reintroduce adaptivity and permit further iterations on pubtree.

Fixed Point Solution

Another avenue of improvement is in the coefficient fitting stage. Take, for example, \( B_n \) composed of two tiers \( T^0 \) and \( T^1 \). Presently the coefficients for \( T^1 \) are influenced by the coefficients to \( T^0 \), but not the other way around. As noted at the end of section 4.4.3, the optimal choice of coefficients for \( T^0 \) is impacted by \( T^1 \). Therefore we propose to use a fixed point iterative scheme for finding the coefficients to \( T^0 \) and \( T^1 \) simultaneously. In such a scheme, we find the coefficients to \( T^1 \) in the usual way, and then reverse the roles of \( T^1 \) and \( T^0 \); i.e. fix \( T^1 \) and solve \( T^0 \). With more tiers one would travel up and down the refinement hierarchy fixing all but one tier and solving the coefficient fitting problem. This scheme would continue until the coefficients had reached a fixed point. Our method already has a flavor of multigrid, and adding this fixed point iteration would bring it even further into the multigrid fold [45].

Linear Independence

Although we have pursued a linearly dependent spanning set \( B_n \), with sufficient implementation effort a linearly independent basis could also be devised. A comparison between the methods would be a valuable analysis.

Sparse Quadrature

Presently nearly all the time required by our algorithm is spent calculating gravitational potential and force for the right hand side, \( b \), of equation 4.8.
Using multidimensional sparse quadrature [52] would improve our running time significantly.
Chapter 5

Gravitation on Graphics Processing Units

Results

Two classic algorithms for computing gravitational force of a small body are adapted for the parallel architecture of Graphics Processing Units. In each case we find an order of magnitude speed-up.

5.1 Graphical Processing Units

In recent years Graphics Processing Units (GPUs) have moved from real-time graphics accelerators to general purpose parallel computers. When compared with conventional CPUs, GPUs may offer significant advantages in performance per cost and performance per watt. This chapter explores using the GPU to efficiently compute the gravitational force of a small body. Specifically, we implement gravitation by spherical harmonics and the polyhedral method for the GPU, and compare their performance to their CPU counterparts.

5.1.1 GPU Computing

GPU computing is still a young field, yet it has already delivered results in a variety of high performance computing domains. The astrophysics community has begun exploring GPUs as an alternative to the GRAvity piPE (GRAPE) special purpose computers for N-body simulation [51]. Researchers in computational biophysics have explored accelerating molecular modeling using GPUs [54], and realized up to 100× speed improvement. General scientific computations, e.g. the finite element method, have also been adapted to the GPU [23]. For a broader review of GPU computing and discussion of fundamental algorithms – e.g. sorting, summation, reduction – see Nguyen [38], Pharr and Fernando [41].
5.1.2 Hardware Overview

The following section describes the major features of the GPU in terms of classic problems in high performance computing. Interested readers are referred to the CUDA Programming Guide [39] for more information.

Memory Latency

In modern high performance computing a significant bottleneck is memory latency. For example, on a typical x86 platform the time it takes to fulfill a memory request is on the order of 200× longer than the time it takes to execute a machine instruction. CPUs use large caching hierarchies to mitigate latency, which occupy significant area on the chip. The GPU, on the other hand, has no hierarchical cache. This feature has a significant impact on the design of the hardware. Foremost, removing cache opens up space for additional floating point processors. Of course removing the cache forces GPUs to use a different strategy for hiding memory latency. In cache’s place the GPU uses a massively multithreaded architecture [2, 39].

As threads are fundamental to GPU programming, we shall take this moment to establish a definition. A thread is a sequence of machine instructions paired with sufficient state to track progress through the sequence. Threads begin at the start of their sequence, and are destroyed when they reach the end. At any point during execution of the instruction sequence a thread may be temporarily halted; in this circumstance the thread’s state is used to store its progress so it may be later resumed where it left off.

Graphics Processing Units are designed to track many threads per processor and switch between them rapidly. This enables the hardware to select from a large pool of threads, in particular threads not delayed by memory requests. To illustrate this idea consider a scenario where 1 thread, executing on a single processor, makes a memory request. If the request takes the time of 10 instructions then the processor will waste 10 opportunities to do work. Now augment this example with an additional 9 threads waiting to execute the same code. Each time a thread makes a memory request it is identified as stalled, and the hardware quickly switches to an unblocked thread. Thus, the system cycles through all 10 threads making a single memory request each time. By the time every request is submitted the first thread’s access will have completed and the block on it will lift. This strategy is employed
Figure 5.1: GPU two tiers of shared memory. Device memory permits communication from any thread to any thread, but the synchronization cost is high. Shared memory enables threads of the same block to communicate at little cost.

on the GPU but at a much larger scale. The GPU in our experiments, a NVIDIA GeForce 8800 GTX, has 128 thread processors. Thus, to effectively hide memory latency we need at least several thousand concurrent threads.

Communication

In the presence of multiple processing units one inevitably wonders how they may communicate. GPUs employ a two tier shared memory model, as depicted in figure 5.1. At the higher level is device memory; this memory is analogous to conventional system memory, and enables any thread to communicate with any other. To communicate via device memory all threads must: i) Write their communication to device memory. ii) Reach an agreed upon synchronization point; threads executing simultaneously may not be at the same point in execution, hence synchronization points may result in “waiting” threads. iii) Read from device memory the communications of other threads. The turn around time for this kind of communication is on the order of 1µs (write latency + synchronization + read latency).

Beneath device memory is shared memory. This is a relatively small local memory shared by threads of the same block [39]. A block is a logical grouping
of threads that the programmer may use to organize up to 768 threads at a time. Threads in the same block may communicate via shared memory in the same way threads communicate via device memory, i.e. follow the three steps above replacing “device memory” with “shared memory”. There are two advantages of using shared memory: First, rather than synchronize across the whole device, communication via shared memory synchronizes only at the block level. Furthermore, reads and writes to shared memory have zero latency. Thus, turn around time for shared memory communication is much smaller than device memory communication. Total turn around time can be as small as 1 ns using shared memory communication.

Simplicity is essential to a communication scheme. The process of reformulating a serial task into parallel ones is already taxing, when a programmer must also consider a particular network topology the effort becomes prohibitive. Fortunately GPU communication is relatively straightforward: there are no sophisticated network topologies to consider. Furthermore, communication on the GPU – when using shared or device memory – is orders of magnitudes faster than Beowulf clusters using Myrinet or TCP (which pass small messages in about 10 $\mu$s) [1].

5.2 GPU Implementations

Now we present two GPU implementations of gravitational force. First we focus on the polyhedral method, which will give us an opportunity to introduce fine grained parallelism. Then we will explore gravitation via spherical harmonics, where we employ task level parallelism.

5.2.1 Polyhedral Method

Let us reintroduce equation 2.5,

$$F(x) = -\nabla U(x) = G\rho \sum_{e \in \text{edges}} E_e \cdot r_e(x) \ln(v_e) - G\rho \sum_{f \in \text{faces}} F_f \cdot r_f(x) \tan(w_f).$$

where $r_e$ and $r_f$ represent vectors, $E_e$ and $F_f$ matrices, and $v_e$ and $w_f$ represent scalars. GPUs are well suited to this kind of computation. Foremost, the equation is readily split into parallel threads by considering each term in the sum as a thread. In a traditional cluster environment such paralleliza-
tion is not practical because the time required to transmit data between processing units rivals the time required by a serial implementation. On the GPU, however, we have already noted that communication latency is orders of magnitudes faster than on a cluster.

To implement the polyhedral method we split calculation into 4 phases. In the first phase we construct $E$ threads, one for each edge, and in parallel compute each term of

$$\sum_{e \in \text{edges}} E_e \cdot r_e(x) \ln(v_e).$$

The next phase uses parallel reduction – which we shall discuss shortly – to perform the summation. The final two phases use the same technique to calculate the sum over faces. Once face and edge contributions are determined the CPU collects both results and adds them to compute the final value.

Parallel reduction is a classic parallel computing algorithm used to evaluate the associative operator $\oplus$ on a sequence of terms [41]. Reduction evaluates the sequence by performing $\oplus$ on pairs of terms until there is only one left. Algorithm 5.1 details the sequence of instructions each thread follows to reduce an array $A$ of $N$ elements, where $N$ is a power of two, with operator $\oplus$; and figure 5.2 diagrams this process. As we halve the number of terms at each pass the total number of passes is $\log N$. To perform reduction on non-power of two arrays simply pad the array with sufficient zeros to reach the next power of two.

```
\begin{algorithm}
    $i \leftarrow N/2$
    \textbf{while} $i > 0$ \textbf{do}
    \hspace{1em} \textbf{if} threadID $< i$ \textbf{then}
    \hspace{2em} Aidx = threadID
    \hspace{2em} Bidx = Aidx + i
    \hspace{1em} \textbf{end}
    \hspace{1em} $i = i/2$
    \hspace{1em} \textbf{barrier}()
    \textbf{end}
\end{algorithm}
```

\textbf{Algorithm 5.1:} Parallel Reduction. Let \texttt{threadId} be an integer between 0 and $N - 1$ that uniquely identifies each thread, and \texttt{barrier()} be a synchronization point that blocks threads from advancing before all threads have reached \texttt{barrier()}.

In cluster environments parallel reduction is only worth the communica-
Figure 5.2: \texttt{Aidx} and \texttt{Bidx} refer to the indices in algorithm 5.1. The numbers inside each box refer to the thread that works on that entry. Each thread combines one \texttt{Aidx} with one \texttt{Bidx} and places the result in \texttt{Aidx}.

...tion cost if the $\oplus$ operator is sufficiently expensive. With a GPU, communication costs are so small that even floating point addition can be parallelized in this way.

**GPU Implementation Idiosyncrasies**

The GPU implementation was written in CUDA [39] – NVIDIA’s C++ extensions for programming the GPU – and follows the algorithm outlined above with one exception. To optimize performance the first and third phase are partially combined with the second and forth phases respectively. Specifically, threads (i.e. terms) are organized into blocks of 256 elements. Each thread computes a term and writes the result to shared memory. Then each block performs a reduction of its 256 terms before writing the result to device memory. At this point device memory contains an array whose elements are the reduction of 256 terms. Now phase two (four) is invoked to reduce this array using shared memory. Given the maximum size of a block is 768 threads, we can reduce up to $256 \times 768 = 196,608$ terms in only two passes through device memory. Staging reductions through shared memory is a worthwhile effort as device memory is much slower than shared memory.
5.2.2 Spherical Harmonics

Our GPU implementation of the polyhedral method uses fine grained parallelism: our task was to calculate gravitational force, and we split this task into sub-tasks for the GPU. For spherical harmonics we use task level parallelism: Instead of parallelizing by dividing the task to thousands of threads, we parallelize by replicating the task to thousands of threads. Thus, each thread independently calculates the force of gravity for some unique point in space. In terms of trajectory simulation this means that thousands of trajectories will run simultaneously.

No special parallel algorithm is required for this implementation. We wrote a simple serial spherical harmonics code and let the GPU run it on thousands of threads.

GPU Implementation Idiosyncrasies

To reduce time spent on data access we implemented two optimizations. First, we let threads share coefficients. Blocks of 64 threads load 64 coefficients into shared memory; then those same threads calculate 64 terms of equation 2.6. Once every thread in the block consumes every shared coefficient the next 64 are loaded. This process repeats until every coefficient is loaded. Figure 5.3 diagrams this process. Next, we incorporated the classic 4th order Runge-Kutta integrator into the CUDA implementation. By putting force calculation and ODE integration on the GPU we reduce data transfers between GPU and CPU.

5.3 Trajectory Experiments

To verify the performance and correctness of our GPU implementations we compared trajectories generated by the GPU with several CPU implementations.

5.3.1 Polyhedral Method

To assess the GPU implementation of the polyhedral method code we integrated trajectories near the asteroid 1998 ML14 [37, 40]. Simulations ran for 1 day of simulated ballistic motion, positions and velocity were recorded.
Start

Load x, y, z location

Load a set of up to 64 coefficients

Loop up to 64 times to evaluate each coefficient we loaded

More coefficients?

NO

Write x, y, z force components

Stop

YES

Wait for other threads to finish using coefficients

Figure 5.3: The execution path of a single thread computing spherical harmonics. Squares represent a thread’s independent effort. Parallelograms indicate a synchronized effort and imply barrier synchronization within thread blocks. We then wrap this with an order 4 Runge-Kutta integrator to produce a trajectory.
every 5 minutes of simulated time. Trajectories were randomly seeded with
initial positions near the equatorial plane and between 1.2 km and 1.35 km
from the center of mass of the asteroid. The inclination of trajectories was
restricted by reducing the out of plane velocity to near zero. The trajectory
integrations were done in rotating coordinates. The period of rotation for
1998 ML14 was assumed to be 14.93 hours and the moment of inertia tensor
was computed from the triangle mesh surface of the asteroid. This was used
to compute the principal axis. The local coordinate system for the asteroid
was used as the rotating coordinate system; the z-axis turned out to be close
to but not exactly the same as the principal axis. The computed normalized
principal axis, in the coordinate system of the asteroid mesh was (0.0636,
0.0008, 0.9356). Thus the xy-plane was close to, but not the same as the
equilateral plane.

Our comparison used three trajectories; italics denote the shorthand we
give for each trajectory. The GPU trajectory used the method described
above to compute force, and the Runge-Kutta-Fehlberg embedded method of
order 4-5 to integrate the trajectory. The absolute error bound was $10^{-6}$ and
the relative error bound was $10^{-13}$. Our reference trajectory used a double
precision CPU implementation of the polyhedral method to compute force,
and the Runge-Kutta Prince-Dormand embedded method of order 8-9 to
integrate the trajectory. Absolute error was bound at $10^{-10}$ and relative error
at $10^{-13}$. Finally, the speed comparison trajectory used the force computation
from the reference trajectory with integrator of the GPU trajectory.

This experiment was repeated for 1000 trajectories. For each trajectory
we measured the maximum difference in position and velocity between the
GPU trajectory and reference trajectory. Figure 5.4 is a histogram of the
errors in position and velocity. Most trajectories fall within 10 m of the
reference trajectory. The maximum position error was 68.79 m, and the
minimum was 34.52 cm. On average integrations with the GPU method
were about 50 times faster than the speed comparison trajectory running on
a 2.2GHz Athlon CPU.

---

1Reducing the error tolerance or increasing the method order led to unacceptably small
time steps. We believe this is due to the limited precision available on the GPU.
2We also implemented a single precision CPU polyhedral method; it ran at the same
speed as our double precision code, so we do not report on it.
5.3.2 Spherical Harmonics

Spherical harmonics on the GPU was tested in a similar way. Order and degree 12 spherical harmonics were fitted to the asteroid 1998 ML14; and simulations ran for 1 day of simulated ballistic motion, recording position and velocity every 5 minutes. Randomly selected initial positions were near the equatorial plane and between 2 km and 2.5 km from the center of the asteroid. The inclination of trajectories was restricted by reducing the out of plane velocity to near zero. In this case the integrator was fixed, hence, we only produced to two trajectories: GPU and reference. Both used the classic order 4 Runge-Kutta method with time step 0.5 s.

This experiment was repeated for 973 trajectories (27 of 1000 escaped or impacted). For each trajectory we measured the maximum difference in position and velocity between the GPU trajectory and reference trajectory. Figure 5.5 is a histogram of the errors in position and velocity. Most trajectories fall within 20 m of the reference trajectory. The maximum position error was 53.40 m, and the minimum was 3.22 m. On average integrations with the GPU method were about 75 times faster than the reference trajectory running on a 2.2GHz Athlon CPU.

5.3.3 Error Analysis

Ideally single precision should offer up to $10^{-7}$ relative error, but due to numerical error neither method lives up to this standard. Both methods suffer
cancellation in their summations, and the polyhedral method has additional sources of error as distance to the body increases [27]. Figure 5.6 illustrates the effect of distance to a body on error; relative error is measured with respect to a double precision CPU implementation. Note that toward the edges of the domain the GPU implementation does not even reach $10^{-4}$ relative error.

Figure 5.6: Relative error in the force produced by our GPU implementation of the polyhedral method. The interior of the asteroid is not measured and left black. Error is plotted on a log scale. Left: The error over the entire domain. Right: Sample of the error in a region abutting the asteroid.
5.4 Discussion

Performance

Our experiments have shown GPUs to be significantly faster than CPUs for typical simulations, albeit suffering some loss in accuracy. Furthermore, if we consider “bang for buck” the GPU has an even larger lead. For example, the GPU in these experiments cost less than half of a single node on the cluster computer that produced the reference trajectories. On top of that a single GPU well outpaced a single node.

Accuracy

The excellent performance, however, is marred by limited accuracy of the GPU in our tests. This can be attributed to the combination of hardware (single precision) and algorithmic (cancellation) deficiency. GPUs with double precision are becoming available, and should provide enough precision to overcome cancellation effects.\(^3\)

Qualitative Features

GPU methods borrow the qualitative features of their CPU counterparts. In this case both methods preserve all qualitative features.

Task Level Parallelism

Our experiments suggest that that task level parallelism may be quicker for trajectory integration. A task level polyhedral method would follow a similar strategy as presented for spherical harmonics: Thousands of force queries are satisfied simultaneously rather than a single query being split into thousands of subtasks. This strategy may be suitable for Monte Carlo simulation near small bodies, or full two body interactions (i.e. the dynamics of two polyhedra orbiting one another) [56].

\(^3\)Intel and AMD chips work in 96 bits of precision internally, and expose 64 bits of precision to the programmer i.e. double precision.
Chapter 6

Summary & Conclusion

6.1 Summary

We have presented several efficient methods for computing gravity of small, irregular bodies that are accurate enough for planning missions near them. To conclude we summarize and compare the computational characteristics of each available method.

Polyhedral Method

Memory: The primary memory cost of this method is storing the polyhedral model and its associated data. As most models have only tens of thousands of elements the polyhedral method require a small amount of memory. Speed: Calculation requires iterating over every edge and face; furthermore, each edge and face calculation includes a transcendental function. Hence, polyhedral methods are slow to compute gravitational force; our techniques range from $50 \times$ to $400 \times$ faster. Error: As long as the polyhedral model and density assumptions are not far from the truth this method produces exact results. Of course, any errors in this method would spoil models using it as a base line; e.g., cubetree and pubtree. Qualitative Features: All the qualitative features are preserved.

Modified Polyhedral Method

Memory: In addition to the memory costs of the polyhedral method, modified polyhedral method [10] caches prior calculations; this, however, can only take a constant factor beyond the unmodified requirement. Hence, modified polyhedral methods also have small memory requirements. Speed: Depending on the orbit this varies between $10 \times$ and $100 \times$ faster. In ideal circumstances this may reach our performance, but in general $100 \times$ speed
ups are not achieved. **Error:** Errors at 3-radii are close to or below $10^{-3}$; closer trajectories may experience more or less error. **Qualitative Features:** Depending on the efficiencies employed Modified Polyhedral Method may preserve some qualitative features.

**Spherical Harmonics**

**Memory:** Spherical harmonics have a small memory footprint: even the most sophisticated models only use $10^5$ coefficients to store the whole model. **Speed:** Depending on the order and degree used, spherical harmonics can be up to $100\times$ faster than the polyhedral method. At best, spherical harmonics approaches the speed of our method, and compares favorably with other methods. **Error:** No matter what order is chosen errors near non-convex regions of the asteroid render this method useless for missions close to a small irregular body [57]. This puts spherical harmonics at a significant disadvantage to both the polyhedral method and the methods presented in this paper. (Excluding spherical harmonics on the GPU, which suffers equally.) **Qualitative Features:** All the qualitative features are preserved.

**Mascons**

**Memory:** Mascons use memory linear to the number of point masses used. In practice this number is in the thousands; thus, mascon memory footprint is very small. **Speed:** Mascons are faster than polyhedral methods, however, they are still subject to a linear run-time complexity. In practice their performance is irrelevant as they have non-trivial error. **Error:** Errors for this method have not been theoretically bounded, and experiments show that large errors do exist [57]. **Qualitative Features:** All the qualitative features are preserved.

**Cubetree (Chapter 3)**

**Memory:** Cubetree has moderate memory requirements: every octree leaf cell requires $(\text{order} + 1)^3 \times 3$ coefficients. The octree used in our experiments described in sections 3.3 and 3.4 had 386,880 leaf cells and occupied 653 Megabytes. **Speed:** Using interpolating polynomials permits a constant time reconstruction of the force within a cell, and finding the correct octree cell is a $O(\log (\text{Number of cell}))$ operation. In practice this gives about $100\times$
speed up over the polyhedral method. **Error:** Errors can be controlled to within user tolerances. In our experiments we reached our $10^{-5}$ goal quite near the asteroid (further than 4 m from the surface), and surpassed it at points 2 and 3 radii away (from the center). **Qualitative Features:** None of the qualitative features are preserved, however, this did not impact our trajectory tests.

### Pubtree (Chapter 4)

**Memory:** Pubtree has similar memory requirements to cubetree. The model we produced required 210 Megabytes of memory, however, it did not have the same accuracy as cubetree. A few more iterations would solve this, but at a higher memory cost. **Speed:** Once again similar to cubetree: The average runtime of a bounding volume hierarchy query is $O(\log (\text{Number of patches}))$, and once we have a patch only a constant amount of work is required to compute gravity. In practice we see over two orders of magnitude speedup compared to the polyhedral method. **Error:** Errors can be controlled to within user tolerances. Our model reached the goal of $10^{-5}$ in the majority of the modeled volume. Further from the surface we passed this goal by at least an order of magnitude. **Qualitative Features:** Continuity and exactness are preserved, but divergence is still present.

### GPU Methods (Chapter 5)

**Memory:** GPU methods have no specific memory overhead; therefore, their costs mimic the cost of the corresponding CPU method. **Speed:** GPU methods are very fast for easily parallelized tasks, such as those presented in chapter 5. $50 \times$ speed-ups are possible when the frequency of communication and size of communication between the GPU and CPU is kept small. **Error:** The GPUs we tested only offer single precision floating point, which – as demonstrated in chapter 5 – is clearly a limitation. GPUs with double precision floating point are becoming available at the time of this writing. **Qualitative Features:** GPU methods observe whatever qualitative features are present in the underlying algorithm. In this case both methods observed all qualitative features of gravitation.
6.2 Conclusion

All of the methods introduced herein are quick, accurate methods for calculating small body gravitational force. Which method is best for a particular circumstance depends on other requirements of the project.

When simplicity is key GPU methods are king. Unlike the polynomial approximations, programming for a GPU has a relatively low barrier to entry: 

\text{i}) The required hardware is cheap and readily available, and 
\text{ii}) the software is easy to use and straightforward to implement. That said, they are still a maturing platform and present GPU solutions suffer vendor lock-in. Coming standards – e.g. OpenCL, DirectX 11 – will provide portability in coming years. GPUs provide over an order of magnitude speed-up, and the recent availability of double precision makes error a non-issue.

Where gravitational force is the only requirement cubetree offers an extremely fast model, albeit with some upfront cost. Software development effort is on-par with traditional methods, but the bootstrapping process requires intense computation. Constructing cubetree with a GPU providing the reference force would enable cluster free production of cubetree models.

Scenarios where continuity or gravitational potential are required, such as trajectory optimization, require the full effort of pubtree. Pubtree software is the most complex of these methods, and fitting the coefficients requires cluster computing as the linear solves are too large to fit on a single machine. Furthermore, the extra effort does not appear to impact Monte Carlo trajectory simulation near small bodies. Hence, pubtree is only recommended when continuity is absolutely required.

All of these methods vastly decrease computation time, and bring Monte Carlo simulation near small bodies off of cluster computers and onto workstations.
Appendix

Fast B-spline Evaluation with Derivatives

The following algorithm evaluates a B-spline with uniform knots and its derivative. The last iteration of de Boor’s algorithm [15] is separated and derivative calculations are made based on the degree $k - 1$ B-splines.

\[
\begin{align*}
B[0] & \leftarrow 1 \\
\text{for } f \leftarrow 1; f < k; f \leftarrow f + 1 \text{ do} \\
& \quad B[f] \leftarrow \frac{t}{f} B[f - 1] \\
& \quad \text{for } g \leftarrow f - 1; g > 0; g \leftarrow g - 1 \text{ do} \\
& \quad \quad B[g] \leftarrow \frac{t-(g-f)}{f} B[g - 1] + \frac{(g+1)-t}{f} B[g] \\
& \quad \text{end} \\
& \quad B[0] \leftarrow \frac{1-t}{f} B[0] \\
\text{end} \\
f \leftarrow k \\
B'[f] \leftarrow \frac{B[f-1]}{\Delta} \\
B[f] \leftarrow \frac{t}{f} B[f - 1] \\
\text{for } g \leftarrow f - 1; g > 0; g \leftarrow g - 1 \text{ do} \\
& \quad B'[g] \leftarrow \frac{B[g-1] - B[g]}{\Delta} \\
& \quad B[g] \leftarrow \frac{t-(g-f)}{f} B[g - 1] + \frac{(g+1)-t}{f} B[g] \\
& \quad \text{end} \\
B'[0] \leftarrow -\frac{B[0]}{\Delta} \\
B[0] \leftarrow \frac{1-t}{f} B[0] \\
\text{return } B, B'
\end{align*}
\]

Function \text{Evaluate}(t, \Delta, k) Evaluates the uniformly spaced B-spline with knot spacing $\Delta$. The return value is a tuple; $B$ contains the values and $B'$ contains the derivatives.
References


Curriculum Vitae

Academics

University of Illinois at Urbana-Champaign  8/04 – 8/08

• Ph.D. Computer Science, specializing in Scientific Computing with application to Aerospace Engineering.

Bucknell University  8/98 – 05/02

• Graduated May 2002 Summa Cum Laude - Double Major: Mathematics & Computer Science.

Industry

NVIDIA - Intern  5/07 – 11/07

• Application developer for NVIDIA’s high performance computing platform, CUDA.
• Researched, developed, and evaluated CUDA implementations of common networking and security algorithms.

Publications

Honolulu, Hawaii.


