1.5in 0.6in 1.0in 0.8in 20pt 0.25in 9pt 0.3in

University of Buea

Faculty of Science

Department of Computer Science

Implementation Of A Heart-shaped Primitive

A thesis submitted to the Faculty of Science
in partial fulfilment of the requirements for the degree of
Master of Science in Computer Science
By

Isaac Kamga Mkounga
(SC09B676)

Supervisor:
Prof. Emmanuel Kamgnia

Date:
August 2015

Declaration

I, Isaac Kamga Mkounga, Matriculation Number (SC09B676) declare that this thesis titled, *Implementation Of A Heart-shaped Primitive* and the work presented in it are my own. I confirm that this work was done wholly while in candidature for a Master of Science in Computer Science at the University of Buea. Where I consulted the published work of others, this has been clearly acknowledged. This work has not been previously submitted for a degree or any other qualification at this University or any other institution.

Signed: Date:

Isaac Kamga Mkounga
(SC09B676)

Dedication

To My Family

Certification

This is to certify that this research project titled *Implementation Of A Heart-shaped Primitive* is the original work of Isaac Kamga Mkounga, Matriculation Number (SC09B676), a Master of Science in Computer Science student of the Department of Computer Science in the Faculty of Science at the University of Buea.

Supervisor: _____ Date: _____

Prof. Emmanuel Kamgnia

HOD/CSC: _____ Date: _____

Dr Denis Nkweteyim

Dean/FS: _____ Date: _____

Prof Ayonghe Samuel

Acknowledgement

Firstly, I thank our Lord Jesus Christ for giving me life and good health during my stay at University Of Buea and for granting me the patience to go through this program.

I am grateful to the Vice Chancellor of the University Of Buea and her collaborators for maintaining an enabling environment where graduate students can learn and express their utmost research potential. I am especially indebted to Professor Joyce B.M. Endeley, currently the Director of the Higher Technical Teachers' Training College of the University of Buea in Kumba and Professor Theresa NkuoAkenji, the Deputy Vice Chancellor for Internal Control and Evaluation who authorized the connection of internet facilities to the Postgraduate Computer Science Laboratory during the implementation of this project. I wish to sincerely thank all the staff of the Department of Computer Science for their advice and stimulating conversations during the writing of this thesis. Special thanks go to Professor Emmanuel Kamgnia and Dr. Denis Nkweteyim for meticulously proofreading through and correcting this work while making their wide spectrum of research experience and books on graphics available to me.

My appreciation goes to the Google Open Source Programs Office (OSPO) – Mary Radomile, Carols Smith, Cat Allman and Stephanie Taylor and the United States Army Research Laboratory BRL-CAD team for their sponsorship of this project under the auspices of the 2013 Google Summer Of Code program. Special thanks go to Mr. Christopher Sean Morrison and Mr. Erik Greenwald for their valuable advice and mentorship during the implementation of this project. I am also indebted to *Nominal Animal* and the mathematician Dr. Titus Piezas for tips on ray-tracing the heart surface.

Many thanks go to my family for their assistance and encouragement throughout this program.This thesis is dedicated to them. I also thank the friends I made during the course of this program whether from University of Buea or the Developers community, for bringing all the fun they came along with– I appreciate your company and I'm happy to know I was not alone.

Abstract

In this thesis titled Implementation Of A Heart-shaped Primitive, we aimed at demonstrating the engineering of a heart-shaped primitive within BRL-CAD package.Using a case study approach, we designed the heart-shaped primitive's data structure, wrote necessary callback functions and tested them using BRL-CAD's testing infrastructure. We showed that the Laguerre-based root solver is indeed a sure-fire iterative method for finding roots of polynomials and ascertained it's stability on sextic equations. This work provides a guideline for the development of primitives within Computer-Aided Design (CAD) software by highlighting the implementation of geometrically-useful properties for any primitive within BRL-CAD.

Contents

List of Figures

List of Tables

Abbreviations

CAD	**C**omputer **A**ided **D**esign
BRL-CAD	**B**allistic **R**esearch **L**aboratory **C**omputer **A**ided **D**esign
CSG	**C**onstructive **S**olid **G**eometry
NURBS	**N**on-**U**niform **R**ational **B**-**S**plines
NMG	**N**on - **M**anifold **G**eometry
B-REPS	**B**oundary **R**epresentations

Chapter 1

INTRODUCTION

1.1 Historical Background

Throughout our long history, we humans have always sought for means to express our creativity–ways to communicate our ideas through writing, sculpting, painting, carving, architecture and drawing.As a matter of fact, paleolithic cave representations of animals at least 32,000 years ago in Southern France, ink drawings and paintings of human figures as well as writings in hieroglyphics on papyrus in the pyramids of ancient Egypt is indicative of the fact that our need to express our individuality goes back to antiquity.Before the renaissance, drawing was treated as a preparatory stage for painting and sculpting.The wide availability of drawing instruments such as pens and pencils and most especially paper made master draftsmen like Leonardo Da Vinci, Raphael and Michelangelo around the world to lift drawing to an art in its own right.Thus, drawing stood out as the most popular and fundamental means of public expression in human history and is one of the simplest and most efficient means of communicating visual ideas.[1]

The drawing board era where paper, pens, pencils, rulers and ink prevailed has been relegated to the background in this information age which

is powered by ubiquitous computer technology. Craving the unity of science and art, this essentially binary-sequenced revolutionary device called the computer married the artistic and engineering forms of drawing into an androgynous one called ComputerAided Geometric Design or geometric modeling for short. Geometric modeling currently involves the use of computers to aid in the creation, manipulation, maintenance and analysis of representations of the geometric shapes of two and threedimensional objects [2]. It is the outgrowth of convergent motivations and developments from several works of life as outlined below.

- In the 1950s, the need to automate the engineering drawing process led to electronic drawings which could be archived and modified more easily, could be easily verified and errors could be eliminated from mechanical designs without introducing new ones.These computer drafting systems allowed designers to produce drawings of objects by projecting threedimensional objects unto twodimensional surfaces.

- Then in the 1960s, there was a pressing need for software in the automobile, shipbuilding and aircraft industries to produce computer-compatible descriptions of geometric shapes which can be machined from wood and steel into stamps and dies for the manufacturing and assembling of car parts,ship hulls as well as wings and fuselages using computer numerically controlled tools.

- Later in the 1970s, the growing need for computers to render realistic images of objects as well as animate solid objects pushed research institutes like Xerox Palo Alto Research Center (Xerox PARC) and Apple Computers to make significant contributions to graphical user interface design and computer graphics.

These needs and problems could only be solved by research in fields such as graphics, animation and applications from algebraic geometry. The work of various computer scientists and mathematicians lead to the active development of several commercial packages sponsored by companies such as

Renault, Citroen, Ford and Boeing who could afford the computers capable of performing such lengthy calculations.

Today,geometric modeling is also referred to as Computer Aided Design (CAD), is pronounced "cad" and is routinely used in the design and manufacturing of engineering and architectural structures such as buildings, car parts, ship hulls and aircraft artillery as well as to specify special effects in cartoon movies, music videos and television shows. Indeed, CAD packages provide facilities for designing shapes of solid physical objects and specifying their motion in a way that art and science can unite to create cool designs.

Even though significant progress had been made in basic research and the functionality of commercially available solid modelers like Apple Computer's RenderMan, many solid modelers especially within the open source community are still limited in their geometric features. The open source community is a selforganizing collaborative social network of programmers driven by a passion to solve problems using computers. It has several thousands of its projects on sites that offer services like bug tracking, mailing lists and version control viz Github and Source Forge.These projects are constantly being improved upon by thousands of programmers putting in time and effort to write and debug software without direct monetary pay.

In this thesis, we document the process of developing a heart-shaped primitive, a set of callback functions and procedures which compute geometrically useful properties of solids such as wireframe plotting, database importation and exportation,ray tracing, bounding box calculations, just to name a few, within the Ballistic Research Laboratory Computer Aided Design (BRL-CAD) software package.

BRL-CAD was initiated by the United States Army Research Laboratory in 1983, the same agency which created the E.N.I.A.C., the world's first general purpose computer in the 1940s, to model military systems for the United States government. According to [3], BRL-CAD became born again in 2004 when it joined the open source community with portions of its source code licensed under the Lesser General Public License (LGPL) and Berkeley

Software Distributions (BSD) licenses and has been credited as being the oldest open source repository in continuous development. It supports a wide variety of geometric representations including an extensive set of traditional implicit primitive shapes as well as explicit primitives made from collections of uniform Bspline surfaces, Nonuniform Rational Bspline (NURBS) surfaces, Nonmanifold geometry (NMG) and purely faceted polygonal mesh geometry.

BRL-CAD also focuses on solid modeling aspects of Computer Aided Design. Figure 1.1 below shows a threedimensional model of a Goliath tracked mine, a German engineered remote controlled vehicle used during World War II. This model was created by students new to BRL-CAD in the span of about 2 weeks, starting from actual measurements in a museum.

1.2 Importance Of This Work

This work is significant to several stakeholders for several reasons ;

- By raytracing the heart's surface, it demonstrates to the scientific community that the Laguerre zerofinder is indeed a surefire iterative method for finding roots of polynomials and that Laguerre-based root solvers are stable on sextic equations.
- This work incorporates more geometric modeling functionality into the free and open source software community through BRL-CAD, the oldest open source repository in continuous development[3] by going beyond traditional CSG primitives shapes such as tori, spheres, boxes and ellipsoids towards the more complex heart-shape based on a sextic equation. This work provides a guideline for the development of primitives within open source CAD software.
- Given that BRL-CAD is used within governments to model military artillery and for engineering and analysis purposes within academia, this heart-shaped primitive gives BRL-CAD a more loving aura – an

environment where artists can produce cartoon animations as well as design cards, royal seals and banners, gifts and presents for family and communal celebrations such as weddings, family reunions and Valentine's day for entertainment purposes.

1.3 Thesis Organisation

This thesis is divided into five (5) chapters. Chapter 1 introduces the study and chapter 2 reviews the literature in the field of geometric modeling. In Chapter 3, we state the problems we intend to solve and our project design. In chapter 4, we discuss the interesting results which we obtained. Finally, in chapter 5, we state the contribution of our work and give possible research directions which can proceed from it.

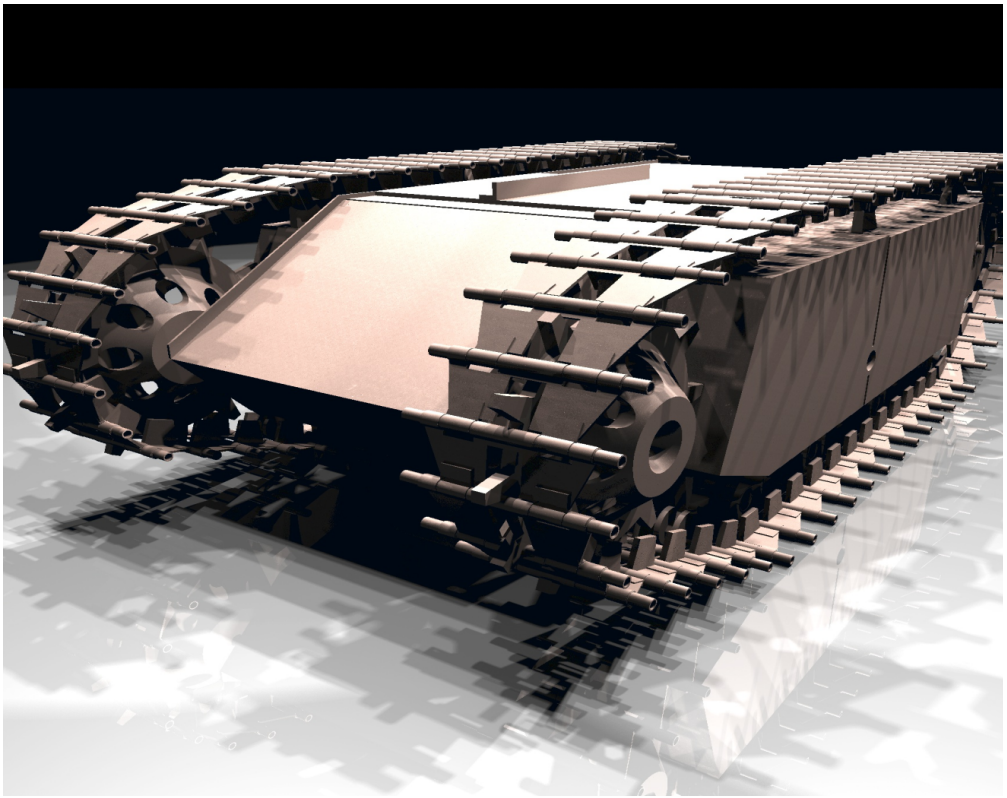

FIGURE 1.1: Model of a Goliath tracked mine

Chapter 2

LITERATURE REVIEW OF GEOMETRIC MODELING

With the advent of computers which could perform millions of floating point operations in unit time and which are still growing faster, researchers who believed computers could aid the processes of mechanical design and manufacturing were faced with a critical issue – how to represent physical reality using computer software. They sought the best data structures to represent this reality and the most appropriate algorithms to manipulate these representations.

BRL-CAD supports a wide variety of geometric representations including an extensive set of traditional implicit primitive shapes as well as explicit primitives made from collections of uniform Bspline surfaces, nonuniform rational Bspline (NURBS) surfaces, non-manifold geometry (NMG) and purely faceted polygonal mesh geometry. Consequently, in this chapter, we review the existing work done by scholars in the field of geometric modeling which have been applied to the development of BRL-CAD. First of all, it introduces the issue of representation and the notion of representation schemes.Then, it summarizes developments in wireframe modeling, surface

modeling, solid modeling and non-manifold modeling (aka nonmanifold geometry or nmg for short) with a keen eye on the algorithms underlying them.

As we progress in our literature review from older forms of geometric modeling to newer ones, we will discover that representation schemes were closely linked to algorithmic efficiency and that it has always been normal to expect designers to switch to newer ones in response to the improvements in algorithmic performance.Despite these enhancements in algorithmic efficiency within the designer community, we cannot say with complete certainty whether traditional representation schemes can be relegated to the background. We can only conclude that old and new representation paradigms coexist and that research led to representation schemes which supplemented the repertoire of geometric modeling.

2.1 Representation Schemes

A representation **R** of a solid or representation for short is a subset of three-dimensional Euclidean space denoted $\mathbb{E}^3$ which models a physical solid. According to [5], Requicha and Tilove stated that point set topology provided a formal language for describing the geometric properties of solids and they also threw more light on the mathematical characteristics of solids such as a solid's interior, boundary, complement, closure, boundedness and regularity. Requicha [4] insisted that to be computationally useful, a representation should formally capture the following properties ;

- ***Rigidity:*** Representations should have an invariant configuration irrespective of their location and orientation.
- ***Homogeneity:*** A representation should have an interior.
- ***Finiteness:*** A representation must occupy a finite amount of space.
- ***Boundary determinism:*** A representation must unambiguously determine the interior of that solid.

- ***Closure***: Representations of solids which are manipulated by rigid motions and regularized boolean operations should produce representations of solids too.

These formal characteristics leave representations no choice than to be bounded, closed, regularized and semianalytic, hence their coinage rsets according to [5]. An ***r-set*** is simply a regular and bounded set in $\mathbb{E}^3$.

A representation scheme is simply a relation between physical solids and their representations which can be characterized by the following properties;

- ***Domain***: A representation scheme must represent quite a number of useful geometric solids.
- ***Unambiguity***: A representation scheme should produce representations which intuitively capture the properties of the physical solid so that it can be easily distinguished from other representations.
- ***Uniqueness***: A representation scheme should uniquely represent a solid object within a software's database.
- ***Validity***: Representation schemes should yield representations of solids which do not exist or are valid.
- ***Closure***: A Representation scheme which transforms (reflects, scales, rotates) a representation should yield other representations too.
- ***Compactness***: Representation schemes should yield representations which save space and allow efficient algorithms to determine desirable physical characteristics.

2.2 Wireframe Modeling

For rectilinear objects whose edges are straight lines and whose faces are planar, the ordered pair of vertices $\mathbf{V} \in \mathbb{E}^3$ and edges $\mathbf{E} \in \mathbb{E}^3$ denoted

by $(\mathbf{V}, \mathbf{E})$ is the object's wireframe. In a practical sense, it is the skeleton of an object wherein joints are vertices and bones are edges. In [6], a six step algorithm to generate an object's wireframe was developed wherein an object's wireframe was represented by a vertex table and an edge table. Although the work in [6] had drawbacks such as not checking the validity of input data, wireframe modeling has always provided designers with a chance to experiment with the final result of their models through sketching and it is frequently used to preview complex models. However, the use of only edge information left wireframe models ambiguous on rectilinear polyhedra talk less of topological ones. Figure 2.1 below shows the wireframe of a sphere in greyscale.

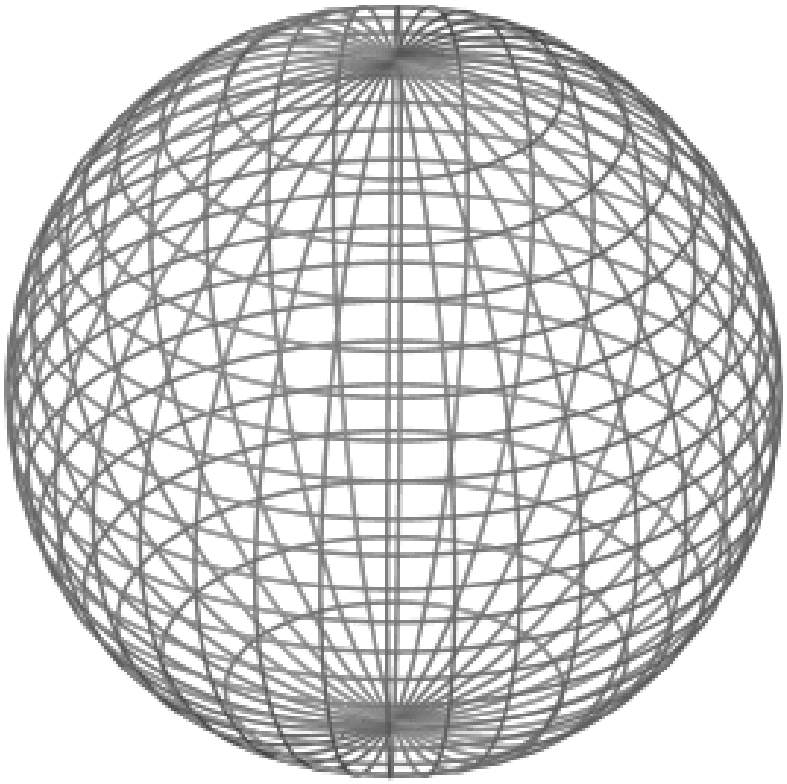

FIGURE 2.1: A wireframe of a sphere

2.3 Surface Modeling

After breakthroughs in wireframe modeling, research efforts in geometric modeling were directed towards extending the geometric coverage of CAD packages by incorporating complex freeform surfaces and curves. In this section, we emphasize on algebraic surfaces and curves used within BRL-CAD as it is the basis for Bezier surfaces and NURBS.

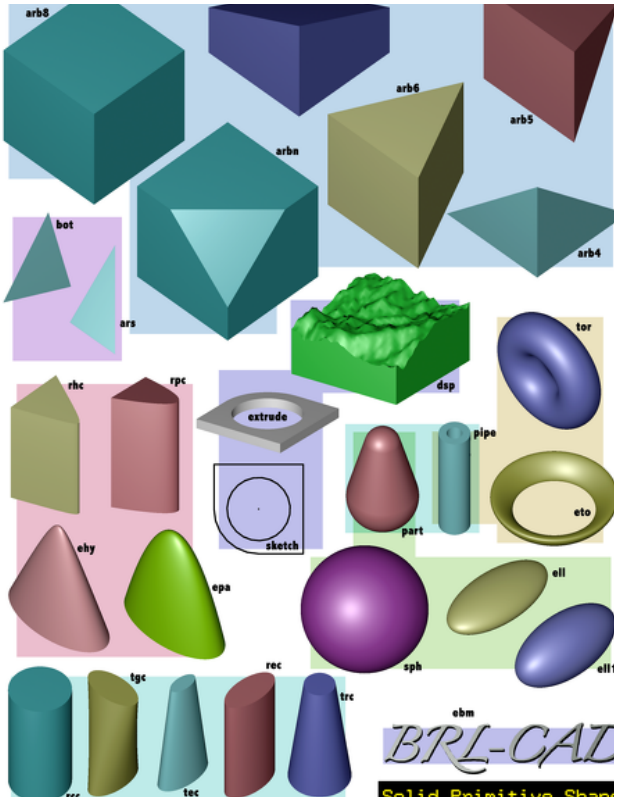

FIGURE 2.2: BRLCAD Solid Primitive Shapes

Figure 2.2 above shows a collection of some primitives used within the BRL-CAD package before the heart-shaped primitive was developed – several of which are implicitly and/or parameterically represented by algebraic equations.On the BRL-CAD's ideas page[7], there is a list of primitives which have not yet been implemented such as the Steiner surface, the ring cyclide surface, the quartoid, Wallis' conical edge solid, etc.

2.3.1 Implicit Representation

An algebraic surface in $\mathbb{E}^3$ is expressed as the set of points satisfying an irreducible polynomial equation

$$\mathbf{g(x,y,z) = 0}$$

in the unknowns x,y and z.
A polynomial ***f(x,y,z)*** over a field **F** is said to be irreducible over **F** if the degree of ***f(x,y,z)*** is positive and its only factors are c and *cf(x,y,z)* where c is a nonzero constant in **F**.

The requirement of irreducibility is so that a surface represented by an equation should not be decomposed into two separate surfaces, each of which can be described by an implicit equation.

2.3.2 Parametric Representation

Some algebraic surfaces possess a parametric representation which consists of a system of equations similar to the ones listed in (1) below;

$$\begin{aligned} \mathrm{x} &= \mathrm{h}_1(u, v) \\ \mathrm{y} &= \mathrm{h}_2(u, v) \\ \mathrm{z} &= \mathrm{h}_3(u, v) \end{aligned}$$

where h_i are rational functions and, u and v are restricted to particular closed intervals in $\mathbb{R}$.

As an example, the unit sphere given implicitly by $x^2 + y^2 + z^2$–1 = 0 can be parameterized by equation (2) viz;

$$\begin{aligned} \mathrm{x} &= (1 - \mathrm{s}^2 - t^2)/(1 + s^2 + t^2) \\ \mathrm{y} &= 2\mathrm{s} \;/\; (1 + \mathrm{s}^2 + t^2) \\ \mathrm{z} &= 2\mathrm{t} \;/\; (1 + \mathrm{s}^2 + t^2) \end{aligned}$$

Also, some algebraic curves possess parametric forms. A parameterization of the unit circle is given by the system of equations in (3) below;

$$x = (1 - t^2)/(1 + t^2)$$
$$y = 2t \ / \ (1 + t^2)$$

When the parametric representation is employed, it is easier to generate points on an algebraic surface or curve as compared to the implicit representation. Also, parametric equations are useful for interactive design since changes in their polynomial coefficients alter the surface's shape in an intuitive manner.

Lots of geometric operations could become faster if both aforementioned representations are made available within CAD packages. Thus, the problem of how to convert from one representation to the other is of great practical importance.

2.3.3 Implicitization

Implicitization is the process of converting a parametric representation into an implicit representation.

Sederberg[7] demonstrated in his thesis that, in principle, it is always possible to convert a parametric surface or curve into implicit form using classical elimination theory developed in the early 20^{th} century. In fact, Sylvester resultants require evaluating a determinant whose entries are coefficients of powers of the variable to be eliminated in several phases.

Although Sederberg's work stimulated lots of research interests in resultant-based methods, this sense of enjoyment within the research community was shortlived due to the following factors;

- Unfaithfulness: Polynomials derived using resultant-based methods give birth to phantom solutions.
- Ineffectiveness: Using floating point arithmetic, resultant-based methods become inaccurate.
- Inefficiency: Evaluating resultants entail huge amounts of computation and is expensive.

Another method of Implicitization is the Grobner basis technique introduced by Buchberger[8] where it was learned that expressing a polynomial as a linear combination of a Grobner basis facilitates finding the solution of the nonlinear system of equations as much as an **LU** - decomposition brings a system of linear equations to heel. A polynomial basis is a set of polynomials which can be used to express any polynomial and can be viewed as a vector space over the field of coefficients $\mathbb{F}$. In [9], Lazard constructed a Grobner basis with respect to a term ordering known as the elimination order. In [10], Hoffmann improved upon a basis conversion algorithm developed in [11] by first constructing a Grobner basis with respect to a term ordering different from the elimination order and finally built the final polynomial in which all variables have been eliminated term by term. This algorithm was known to be the fastest elimination technique for geometry applications that was implemented before the 1990s.

Although Grobner basis methods are more efficient and effective than resultant-based ones, implicitization is fairly expensive and limited in practice.

2.3.4 Parameterization

Parameterization is the process of converting an implicit representation of an object into its parametric equivalent, if it exists. Parameterization is not always possible since not all implicit surfaces can be expressed as rational parameterizations. According to Noether's theorem, a plane algebraic curve possesses a rational parameterization if and only if it has genus zero. [13] used a numerically stable Jacobi rotation adapted from [12] to parameterize several conics and parametric surfaces. Table 2.1 below shows a list of parameterizations of some popular conics – circle, ellipse, hyperbola and parabola.

TABLE 2.1: Implicit and parametric equations of some BRL-CAD primitives

	Implicit Form	Parametric Form
Circle	$x^2 + y^2 - r^2 = 0$	$x = \frac{r(1-t^2)}{(1+t^2)}$, $y = \frac{2rt}{(1+t^2)}$
Ellipse	$\frac{x^2}{a^2} + \frac{y^2}{b^2} - 1 = 0$	$x = \frac{a(1-t^2)}{(1+t^2)}$, $y = \frac{2bt}{(1+t^2)}$
Hyperbola	$\frac{x^2}{a^2} - \frac{y^2}{b^2} - 1 = 0$	$x = \frac{a(1+t^2)}{(1-t^2)}$, $y = \frac{2bt}{(1-t^2)}$
Parabola	$y^2 - 2px = 0$	$x = \frac{t^2}{2p}$, $y = t$

2.4 Solid Modeling

A solid can be represented explicitly by its boundary, its volume, or implicitly by specifying operations on volumetric properties that construct it. BRL-CAD focuses on solid modeling by emphasizing physical accuracy and fully describing $\mathbb{E}^3$.

There are 3 main well established paradigms for representing solids used within BRL-CAD based on their boundaries, volumes or constructing from primitives using regularized boolean operations, a boolean set theoretic operations applied to the context of rsets which were developed in [4].

In this section, we review the literature that has grown around classical representation paradigms such as *Boundary Representations*, *Constructive Solid Geometry* and *Spatial subdivision.*

2.4.1 Boundary Representations (B-REPS)

Just as Rome was not built in a day, the transition from surface modeling representation schemes to solid modeling representation schemes was a gradual process which started off with boundary representations. The boundary representations popularly known as as BREPS describe a solid as a set of surfaces which separate an object's interior from its exterior.

With BREPS, the boundary of a solid consists of vertices, edges and faces. For each vertex, edge and face, the geometric section of the representation holds the shape and location of the object in space possibly using an equation while the topological section of the representation holds the adjacency relationships between the vertices, edges and faces as well as their orientation. There are nine (9) ordered adjacency relationships between the three (3) aforementioned topological entities. In order to build a complete representation, it is normal for one to allow the retrieval of any topological entity and any of the nine (9) ordered adjacency relationships. Kevin Weiler [14] showed that only three (3) ordered adjacency relationships are sufficient

to obtain the others. From his work ,a space/time tradeoff arose: Maintaining all 9 adjacency relationships required more space but little time for retrieval while maintaining only sufficient adjacency relationships required little space but more time for retrieval. A more indepth comparison of the space/time tradeoffs of different boundary representations was conducted by Woo [15].

For solids with a manifold surface, several boundary representations exist, the earliest of which was Baumgart's wingededge data structure [16]. Here, an edge node holds information about the edge orientation, face adjacencies and the clockwise/counterclockwise successor and predecessor edges about adjacent faces. This work was based on the assumption that faces are simply connected. The wingededge data structure has to be modified to accommodate solids with multiply connected faces which contain holes (genus). In [17], Braid et al modified the wingededge data structure by introducing a fourth topological entity called a loop into a proposed data structure where each face consists of one or more edge loops surrounding each hole in that face. In the same vein, Yamaguchi and Tokieda [18] modified the wingededge data structure by introducing yet another topological entity called the bridge edge (or auxillary edge which is simply a double edge connecting two (2) edge cycles of a given face.

The reader is invited to explore other existing boundary representations such as the halfedge data structure by Mantyla [19], Hanharan's faceedge representation [20] and Ansaldi et al 's hierarchical face adjacency hypergraph [21]. In order to accomodate solids with internal cavities and permit single volumes to contain multiple faces, a new topological entity called the shell was introduced in [22]. As regards conversion, Shapiro and Vossler [23] developed an algorithm which converts a boundary representation to an equivalent CSG representation in two-dimensional space.

The Euler - Poincare characterization has been used as a necessary condition for the validity of a solid using its topological information viz edges (E), vertices (V), faces (F), loops (L), Holes (G) and shells (S). It is given

by the equation (5) below.

$$V–E + F – (L–F) – 2(S–G) = 0 \tag{5}$$

Mantyla showed that any topologically valid polyhedron can be constructed from an initial polyhedron by a finite sequence of Euler operations viz ***make*** and ***kill***.

Some representation schemes are hybrids such as the B-rep index which is a mixture of the boundary representation and cell decomposition schemes. BRL-CAD is one of the few solid modeling systems which supports the BREP and NURBS representation format. The image shown in Figure 2.3 below is the classic computer graphics Utah teapot model prepared for 3D printing and rendered via BRL-CAD ray tracing.

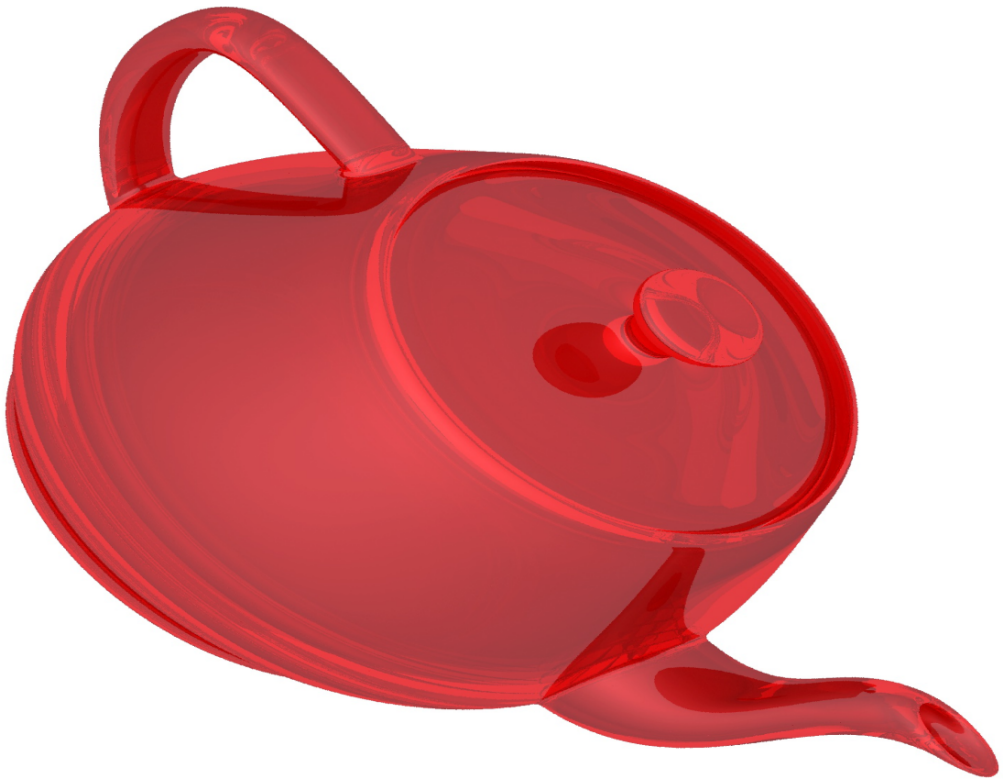

FIGURE 2.3: The Utah Tea Pot model

2.4.2 Constructive Solid Geometry (CSG)

During the 18^{th} century, George Boole's work on the algebra of logic gave birth to an algebra attributed to his name called Boolean algebra which possessed its own set theoretic concepts and operations like union, intersections, difference and complement. Although these operations seemed to be obvious candidates for combining solids, the conventional aforementioned boolean operations had been discovered to be inadequate to manipulate solids in $\mathbb{E}^3$ because the regularity and compactness of solids became compromised. In fact, the complement operator destroys the compactness of solids because the complement of an rset is unbounded. Thus, the need to develop new kinds of set theoretic operations which work well within the realm of solid modeling was of paramount importance. In [24], Requicha developed a new set of operations peculiar to rsets called *regularized union*, *regularized intersections* and *regularized difference*.

His work even went forward to show that the rsets and the regularized boolean operations formed not just a boolean algebra but also a ring. Besides these regularized boolean operations on solids, there also existed other valid operations on rsets which are linear transformations viz reflection, scaling, rotation, shearing, etc. The image in Figure 2.4 below illustrates how the regularized boolean operations result to new shapes.

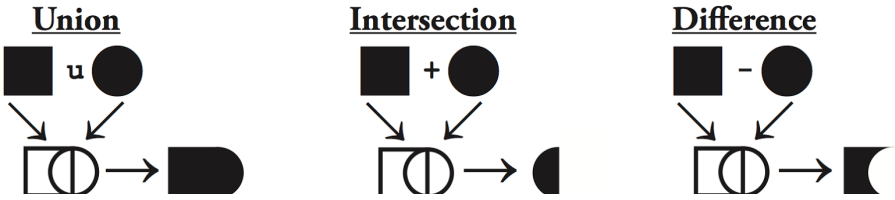

FIGURE 2.4: CSG boolean operations

One of the most widely used representation schemes of our time combines the volumes occupied by overlapping rsets using regularized boolean operations. This representation scheme is called Constructive Solid Geometry (CSG) and its signature is the CSG tree – a binary tree data structure whose leaves are primitives and whose internal nodes are either regularized boolean operations or linear transformations. By a primitive, we simply

mean a regular set of points in $\mathbb{E}^3$ satisfying an irreducible polynomial equation *f(x,y,z) = 0* in the unknowns x, y and z. Examples of primitives used in BRL-CAD include blocks, pyramids, circles, cones, cylinders,etc which can be seen in Figure 2.2 above and the recently developed heart-shaped primitive.

Although BRL-CAD has become a fully hybrid modeling system, it has always had its roots in CSG. The image below depicts a detailed M1A1 tank on a pedestal in a mirrored showcase room which is entirely constructed from implicit primitives and CSG boolean operations.

FIGURE 2.5: Model of a M1A1 tank on a pedestal in a mirrored showcase room

2.4.3 Cell Decomposition

Despite the wide applicability and attention received by boundary representations, it gradually became exciting to research new ways of representing objects explicitly by their volume – a collection of minute cells of a partition of $\mathbb{E}^3$. Some widely used data structures under this representational paradigm are hierarchical in nature and recursively decompose space until they obtain atomic blocks for which no further decomposition is necessary. Although this representation scheme is not widely used within the BRL-CAD package, we thought it important to highlight the prolific developments in this area of research. Due to the frequent use of hierarchical data structures in representing images in the domains of computer vision, image processing and computer graphics, the atomic blocks obtained from recursive decomposition are called *picture elements* or ***pixels*** for short.

The term *quadtree* has been used to describe hierarchical data structures which are based on the common property of recursive decomposition and can be differentiated by the type of data being represented, the principle guiding the decomposition process and the resolution of that decomposition (the number of times the decomposition is applied – whether fixed or variable). The most common quadtree is the region quadtree due to Klinger and Dyer[25] and was coined by Hunter[26].It is based on the successive subdivision of the image array into four (4) equal-sized quadrants which may be recursively subdivided until pixels are obtained. In the tree representation, the root node corresponds to the entire array, each child of a node represents a quadrant of the region the parent node represents while a leaf node represents an atomic block (usually a pixel). A leaf node may be black or white depending on whether its pixel is entirely inside or outside the represented region. It may be gray if it is an internal node of the tree.

As initiated by the aforementioned research carried out by Finkel and Bentley[27], hierarchical data structures can be used to represent images in threedimensional space and even higher. The ***octree*** data structure[26] is the three-dimensional analog of the quadtree. Here, an image is first represented in the form of a cubical volume and then recursively decomposed into

eight (8) congruent disjoint cubes called octants until cubes are obtained or a uniform or predetermined decomposition is reached. The original structure of a quadtree encodes it as a tree structure using pointers. This additional memory overhead posed a problem and prompted two (2) approaches to curbing it. Although it is difficult to put a finger on who exactly developed the first approach to solving this problem, it is an efficient method which treats the image as a collection of leaf nodes each of which is encoded by a base 4 number termed a *locational code*.

The second approach to cell decomposition due to Kawaguchi and Endo[28] is termed a ***DF-expression***. It represents the image in the form of a traversal of the nodes of its quadtree – a compact representation in which each node type is encoded with 2 bits. Although it is not easy to use when random access to nodes is required, it has been shown to be faster than the former for a static collection of nodes representing images in higher dimensional space using an efficient pointer-based implementation.

Rectangular data is often used to approximate objects in an image for which they serve as the minimum rectilinear enclosing object. Indeed, bounding rectangles are used in cartographic applications to approximate lakes, forests, hills, etc. The first representation of quadtrees holding rectangular data was due to Hinrichs and Nievergelt[29]. In this representation, each rectangle is seen as a cartesian product of two line intervals each analogous to an interval tree holding the interval's centroid and extent. The rectangles are reduced to points in threedimensional space and then the problem is treated as if dealing with a collection of points. Another such representation is the region-based *MXCIF quadtree* in which each rectangle is associated with the quadtree node corresponding to the smallest block containing it entirely. The subdivision ceases whenever a node's block contains no rectangles or is smaller than a predetermined threshold size. Yet another such data structure arose called the *R-tree* developed by Guttman[30] in which each node in the R-tree is a d - dimensional rectangle enclosing its child nodes. Its leaf nodes are the rectangles in the database. Each rectangle may be contained in several nodes each of which had to be visited before ascertaining the presence or absence of a particular rectangle. This

lead the retrieval of rectangles to be to slow;a problem which was alleviated by the implementation of an R^+-*tree* in which each rectangle is associated with with several nonoverlapping bounding rectangles. The retrieval time of the R-tree is sped up at the expense of an increase in the tree's height.

We now conclude our review of hierarchical data structures by looking at data structures which explicitly specify the boundaries of regions. They are called Polygonal Maps (**PM**) – a collection of polygons which can either be vertex-based or edge-based. The vertex-based Polygon Maps quadtree or PM_1 quadtree in two-dimensions was due to [31]. The PM_1 *quadtree* was based on a decomposition rule stipulating that positioning occurs as long as a block contained more than one line segment unless the line segments are all incident at the same vertex in the same block. The three-dimensional PM_1 octree data structure is quite useful as it is more forgiving on memory than the conventional octree when representing objects. The edge-based variant of the PM quadtree also dubbed the *PMR quadtree* uses a probabilistic splitting rule in which nodes contain a variable number of line segments.

Shown below in Figure 2.6 is a Sphere Flake drawn using BRL-CAD represented using an octree with five levels of recursion, specular reflections, multiple light sources, environment mapping, checkered texture synthesis, ambient occlusion, and soft shadows.

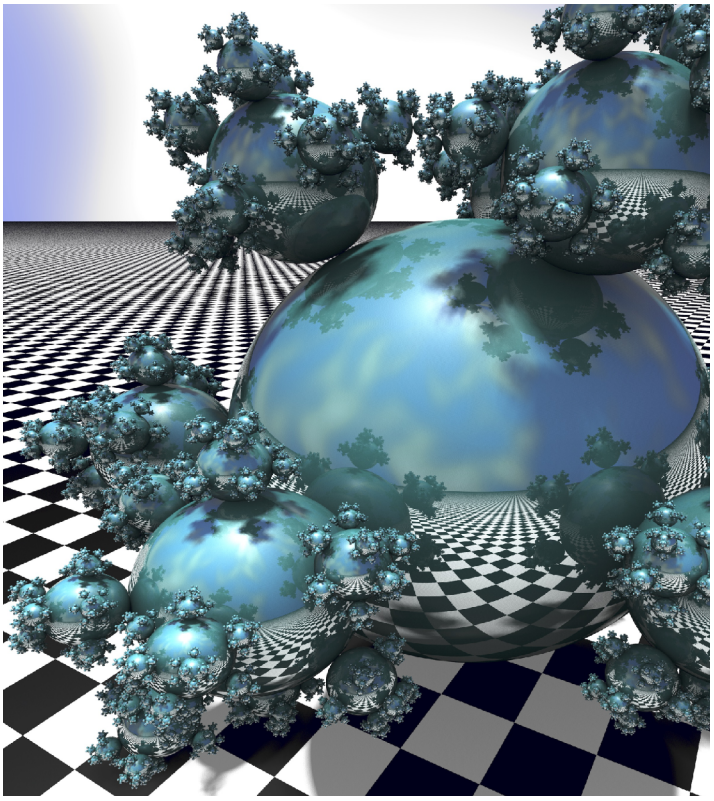

FIGURE 2.6: Model of a Sphere Flake

2.5 Non-manifold Geometry

So far, we have explored the modeling of manifold objects – objects with the property that each of its points have neighbourhoods which are homeomorphic to $\mathbb{E}^3$. These manifolds can also be seen as r-sets which were defined by Requicha in [5]. Lines and circles are one-dimensional manifolds while primitives such as tori, spheres, cones, cylinders,heart-shape,etc are examples of three-dimensional manifolds.

We now throw some light on geometric algorithms used to represent nonmanifolds objects which have points with neighbourhoods which are not homeomorphic to $\mathbb{E}^3$. While lines and circles are one-dimensional manifolds, figure eights are one-dimensional non-manifolds. Examples of two-dimensional non-manifolds may include a cone touching another surface at a single point, faces meeting along a common edge, etc. Non-manifolds usually arise when topological structures such as vertices, edges, and faces hangoff the mapped boundary graph of an object. They also naturally result from regularized boolean operations in CSG even when input is restricted to manifolds only.

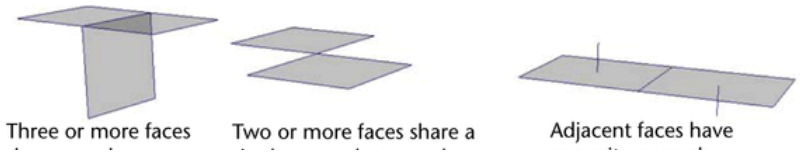

FIGURE 2.7: Examples of Non - Manifold Geometry

According to [2], non-manifold representations are representations which allow volume, manifold and non-manifold curves, surfaces and point elements in a single uniform environment. Thus, non-manifold modeling is seen as a hybrid representation paradigm which simultaneously encompasses wireframe modeling, surface modeling and solid modeling. It enables a smooth transition between several representation schemes and the automatic detection of solid enclosures without any need for restructuring or translation.

In [2], Weiler developed the *radial-edge* data structure which can be used to represent non-manifold geometry. NMG topological elements include vertices, edges, loops, faces, shells, regions and models. The topological information stored in a non-manifold model consists of adjacencies of topological elements. An adjacency relationship is the adjacency of a group of topological elements of one type around some other specific single element.

Thirty-six (36) adjacency relationships are possible in nonmanifold boundary representations. The radialedge data structure employs *uppointers* and *downpointers* to depict the relative *"hierarchy"* which exits between topological elements. The reader can find more information about data structures used for non-manifold geometric modeling at [32] and [33].

In this chapter, we reviewed the work done by researchers in the field of geometric modeling which has been applied in the development of the BRL-CAD viz wireframe modeling, surface modeling, solid modeling and non-manifold modeling.

Chapter 3

ANALYSIS & DESIGN

In this chapter, we state our aim of contributing to the BRL-CAD project and explain how we implemented a heart-shaped primitive in the project design section. Firstly, we introduce the concept of free and open source software. Secondly, we do an overview of the BRL-CAD software package. After, we state our aim of contributing to the BRL-CAD project. Finally, we give a detailed explanation of the design which we employed to implement the heart-shaped primitive for BRL-CAD.

3.1 The Open Source Community

Depending on how we choose to call it, *Free/Libre/Open Source Software (FLOSS)*, *Free and Open Source Software (FOSS)* or simply *Open Source Software OSS)* is software for which users have access to both the source code and binary executables and is licensed under a license which permits its users to read, edit and distribute the software to anyone and for any reason. This distinguishes open source software from commercial software which is distributed by giving away its binary executable version only. Usually, OSS is distributed at no cost with limited restrictions on how it can be used. According to Eric S. Raymond[34], one of the most prominent evangelists of the open source movement, hackerdom can be likened to what

anthropologists call a gift culture– a culture wherein members gain status and reputation by giving away their time, creativity and skills to reading, writing and debugging software, publishing useful information in blogs or documents like Frequently Asked Questions (FAQs) lists as well as handling unglamorous tasks like maintaining mailing lists, moderating news groups, etc without any monetary compensation. The word *hacker* was coined by a shared community of expert programmers and networking masters which traces its history back to the days of the earliest ARPAnet experiments and timesharing minicomputers who made the unix operating systems and the worldwide web work. As opposed to hackers, *crackers* who are more interested in breaking software and perturbing phone systems.

Today, the open source community has become a selforganizing collaborative social network of hackers driven by a passion to solve problems using free software with thousands of projects hosted on Sourceforge[35] and Github[36]. It has singularly developed some software packages and tools which are the best in the world such as the firefox web browser, Apache webserver, Linux operating systems like BSD, Ubuntu, Debian,etc, the MySQL database management system, the VLC media player, programming languages and tools like gcc, C, Perl, Python, Java, etc and much more. Some Examples of CAD packages within the open source community include BRL-CAD, Blender, FreeCAD, openSCAD and LibreCAD, etc.

3.2 Analysis Of The Work

BRL-CAD (pronounced Be-Are-El-CAD) was originally conceived and written by the late Mike Muss, a programmer and networking expert who also wrote the popular PING network program. In 1979, the United States Army's Ballistic Research Laboratory (BRL) (the agency responsible for creating ENIAC, the world's first generalpurpose electronic computer in the 1940s) identified a need for tools that could assist with the computer simulations and analysis of combat vehicle systems and environments. When

no existing CAD package was found to be adequate for this specialized purpose, Mike and fellow software developers began developing and assembling a unique suite of utilities capable of interactively displaying, editing, and interrogating geometric models. Those early efforts subsequently became the foundation on which BRLCAD was built.

The initial architecture and design of BRL-CAD began in 1979 and its development as a unified software package kicked off in 1983 with its first public release the following year. As a software package with a mature code base which has been actively developed for decades, BRL-CAD pays close attention to design and maintainability. Like other FLOSS packages, BRL-CAD's source code and most of its project data are stored in a subversion version control system for change tracking, collaborative development and is redistributed as free and open source software under the Open Source Initiative license terms. The design of its system architecture is based on a unixmethodology of command of the commandline services, providing many tools that work in harmony to complete a specific task. These tools include geometry and image converters,signal and image processing tools, various ray tracing applications,geometry manipulators, and much more. They will also be used to test that the geometric properties of the heartshaped primitive works as we will see in Chapter Four.

The basic layout of its code places public API headers in the toplevel ***include/*** directory and source code for both applications and libraries in the ***src/*** directory. The following is a partial listing of how BRL-CAD's source code is organised in a typical checkout or source distribution.

Applications and Resources

- ***db/*** for Example Geometry.
- ***doc/*** for project Documentation.
- ***include/*** for Public API headers.
- ***regress/*** for Regression test scripts

- ***src/*** for Application and library source.
- ***src/conv*** for Geometry converters.
- ***src/fb*** for Displaying data in windows.
- ***src/mged*** for the Multidevice geometry editor, the main GUI application.
- ***src/rt*** for Ray tracing applications.
- ***src/util*** for Image processing utilities.

Libraries

- ***src/libbn*** for Numerics library.
- ***src/libbu*** for Utility library.
- ***src/libgcv*** for Geometry conversion library.
- ***src/libged*** for Geometry Editing library.
- ***src/icv*** for Image conversion library.
- ***src/libpkg*** for Network Package library.
- ***src/librt*** for Raytracing library.
- ***src/libwbd*** for Geometry creation library.

The majority of BRL-CAD's source code is written in ANSI/POSIX C with the intent of strictly conforming with the C standard. The core libraries are all C API though several such as the Utility and Raytracing libraries use C++ for implementation details. Major components of the system are written in C, C++, Tcl/Tk, Bash and Php with source code files using extensions such as *.c, *.h, *.cpp, *.tcl, *.tk, *.sh and *.php. BRL-CAD uses the CMake build system for compilation and an inbuilt testing infrastructure in regress/ for unit testing.

BRL-CAD has a longlasting heritage of maintaining verifiable, validated and repeatable results in critical portions of the software package, particularly within the ray tracing library. It has an inbuilt testing infrastructure which compares all program output against benchmark results during each build. The ray tracing library is a multirepresentational library which lies at the heart of BRL-CAD and uses a suite of other libraries for other basic application functionality. Considerable attention is put into verification and validation throughout the package which includes regression tests that compare runtime behaviour against known results and reports any adverse variances from standard results as failures.

Despite this sophisticated infrastructure, performant design and longlasting heritage, many still coin BRL-CAD's aspiration of one day being the most widely used open source CAD package as rather lofty for the following reasons;

- With one of the fastest raytracers in existence (on several types of geometry) which is supported by an effective Laguerre-based root solver and used within academia for scientific instruction, computer graphics education and research, the stability of BRL-CAD's root solver on higher order polynomials such as quintics (of power 5) and sextics (of power 6) is still uncertain.
- As an open source CAD software which is deeply rooted in the Constructive Solid Geometry, BRL-CAD's set of primitives is still limited to traditional ones such as cones, cylinders, spheres, tori, etc.
- BRL-CAD is widely used within agencies within the United States Government for the modeling of military artillery and simulating combat vehicle systems and environments. This severely limits is user base to the military sector and gives it an "warring" flare which repels users who would have used it for more entertainment purposes.

In a bid to solve the aforementioned problems, we embarked on a journey to develop a heart-shaped primitive for BRL-CAD. In the following section, we document how we wrote various callback functions which compute

useful geometric properties for the heartshaped primitive such as formatted description, database importation and exportation, computation of the bounding box, plotting the wireframe and ray tracing.

3.3 Design

After having stated our goal of contributing to the BRL-CAD project, we now explain how we implemented the heartshaped primitive. Currently, BRL-CAD aspires to become the most widely used open source CAD software package in existence. Presently, it is mostly used by the United States of America's government agencies which fund its development as well as academic institutes which use use it for computer graphics educations and scientific research. Aljazeera news Channel's recent revelation that less than 1% of the world's population works in the military sector is indicative of the fact that BRL-CAD's usage must go beyond the military sector to break the status quo.

In a bid to increase BRL-CAD's user base by inviting designers and artists from the entertainment industry, we thought of developing a heart-shaped primitive. This heart-shaped primitive (or simply heart), a symbol of romantic love, would go a long way to entertain families and communities weddings, marriage anniversaries, valentine's day celebrations, etc. It would also be used by fashion designers to create magnificient embroidery on clothing. Lastly, it would also be used to design banners and royal coat of arms in cartoons movies as well as animate embroidery on clothing.

While adhering to its coding style, we incorporated the heart-shaped primitive into BRL-CAD's ray tracing library using the following steps;

- We made room for the heart-shaped primitive in BRL-CAD by hooking it unto the raytracing library.
- Wrote callback functions for the heart and tested them using BRL-CAD's inbuilt testing infrastructure in the ***regress/*** directory.

- Built support for typing in parameters for the heartshaped primitive in the display interfaces viz mged or archer.

In order to make room for the heart-shape in BRL-CAD, we tagged the heart primitive, designed a data structure for it and stubbed a skeletal *hrt.c* file into the source code repository in ***src/librt/primitives/hrt/hrt.c.***

3.3.1 Tagging the heart-shaped primitive

Given that each of the primitives in Figure 2.2 above is uniquely stored in BRL-CAD's database, it was necessary to tag the incoming heart-shaped primitive with a unique magic number, Ox6872743f, which is the hexadecimal equivalent of "?hrt?" and increment the maximum number of primitives in ***src/libbu/magic.c***, ***include/magic.h*** and ***include/raytrace.h***.

3.3.2 Data Structure of Heart-shaped primitive

The heart-shape has two lobes which are symmetric about the zaxis and meet at each of its two cusps as the picture in Figure 3.1 shows.

The heart-shape was stored in the ***include/****rtgeom.h* file as an Abstract Data Type (C-like structure) called *rt_hrt_internal* with the following fields representing its parameters.

- A magic number ***hrt_magic***
- A center point ***v***
- A vector in the direction of the X-axis ***xdir***
- A vector in the directions of the Y-axis ***ydir***
- A vector in the direction of the Z-axis ***zdir***
- Distance from center point to either cusps ***d***

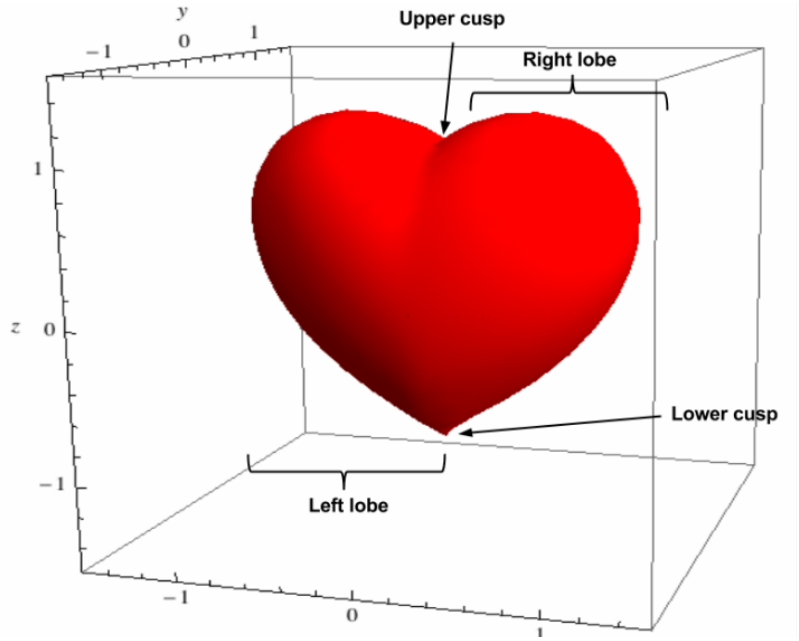

FIGURE 3.1: Diagram of the heart-shaped primitive

These 3 vectors viz ***xdir***, ***ydir*** and ***zdir*** are also called radial vectors because they radiate in the X, Y and Z directions and the aforementioned structure is also known in BRL-CAD's parlance as the heart-shape's internal format.

3.3.3 A bare bones heart-shape

The signatures of the functions which are called to compute the geometric properties for the heartshaped primitive were casted and enlisted in a function table in *src/librt/primitives/table.c*. Then, we added the *src/librt/primitives/hrt/* directory to the source code repository and committed the hrt.c file to it. The hrt.c file consisted of introductory comments, include files, structures for raytracing and storage of the heart-shaped primitive in the database as well as stubs for all callback functions which all print a ***rt_hrt_???:Not implemented yet!*** message when called. For surface representation and raytracing reasons, the heart-shape is stored in a more elaborate C-like structure called *hrt_specific* with the following parameters;

- A position vector for the heart-shape's center ***hrt_V***

- A unit vector in the X axis direction ***hrt_X***
- A unit vector in the Y axis direction ***hrt_Y***
- A unit vector in the Z axis direction ***hrt_Z***
- A unit vector in the Z axis direction ***hrt_d***
- A matrix for scaling and rotating ***hrt_SoR***
- A matrix for scaling and transposing a rotated heart ***hrt_invRSSR***
- A matrix for transposing a rotated heart ***hrt_invR***
- A vector for the inverse of the squared magnitudes of the three (3) aforementioned unit vectors ***hrt_invsq***

The location of the hrt.c file in the raytracing library was added to *src/librt/CMakeLists.txt* for compilation purposes and test driven development.

After hooking the heart-shaped primitive into BRL-CAD's source code repository, we wrote callback functions which compute geometrically useful properties for the primitive such as;

```
1. Formatted description
2. Database importation and exportation
3. Computing the bounding box of the heart-shaped primitive
4. Plotting the wireframe of the heart-shaped primitive
5. Implicit surface representation of the heart-shaped primitive
```

We now explain how the functions we wrote actually compute each of these properties. Each property was tested using BRL-CAD's commands[37] after having written the requisite function(s) that compute it. It is worth highlighting here that the *"rt_hrt_"* prefix is attached to the heart-shaped primitive's structure and its functions' signatures because *"rt"* and *"hrt"* denote the raytracing library and the heart-shape respectively.

3.3.4 Formatted description of the heart-shaped primitive

It often arises that engineers and scientists working on geometric models ask to know the exact values of the objects in question. In order to know a solid's type and the values of its key parameters, we wrote the *rt_hrt_describe()* function which simply prints the heart-shape's parameters in human readable format. The radial vectors in the X, Y and Z directions are printed alongside their magnitudes and the distance from the heart-shape's center to either of its cusps is also printed.

3.3.5 Database importation and exportation of the heart-shaped primitive

BRL-CAD provides CSG modeling features and its models are usually stored in it's Geometric Editing database called GED. For the heartshaped primitive to be used in CSG, we wrote functions that import and export data in between the database format and the internal format.

The *rt_hrt_import5()* function imports a heart from the database format to the internal format. Firstly, we check that the primitive being imported is the heart-shaped primitive by verifying that the primitive's magic number is the same as the heartshape's. Then, this function assigns integers held in an array buffer to the different parameters of the *rt_hrt_internal* structure.

We wrote the *rt_hrt_export5()* function to export the heartshape's internal format into the database format. The database format for the heartshape consists rather of only the heart's center and the 3 radial vectors in the X, Y and Z directions. It is not really useful storing value for d in the database as it is equivalent to some components of the radial vectors. After checking that the primitive in question is the heart, we store the values for *v*, *xdir*, *ydir* and *zdir* in an array buffer holding the integer values for the heart's parameters

3.3.6 Computing the bounding box of the heart-shaped primitive

The bounding box of a geometric model refers to the box with the smallest volume within which the model lies – more like the least upper bound of the set of all enclosing volumes. The spherical equivalent of the bounding box is the bounding sphere which is the smallest sphere within which a model lies. It is a useful property to compute for a model as it is used to appropriately orient an object (within its model coordinates) while positioning the camera (within its world coordinates) and casting lights towards it.

In order to compute the bounding box of the heart-shape, we wrote the *rt_hrt_bbox()* function which had two interesting three-dimensional points as parameters. These points are the minimal and maximal points located at the closest lower left hand corner and the furthest upper right hand corner of the bounding box. Obtaining the coordinates of these two points suffices to compute the bounding as they are located at opposite ends of the box, equidistant from the heart-shape's center and differ from the other 6 edges by a single component.

The X component of the minimal point (closest lower left hand corner point) is obtained by subtracting a factor of 2/3 from the unit vector from the heart-shape's center. The X component of the maximal point (closest lower left hand corner point) is obtained by adding a factor of 2/3 of the X component of the unit vector to the heart-shape's center.

The Y component of the minimal point (closest lower left hand corner point) and maximal point is obtained by subtracting and adding the unit vector from the heart-shape's center.

The Z component of the minimal point (closest lower left hand corner point) is obtained by subtracting the unit vector from the heart-shape's center while the Z component of the maximal point (furthest upper right hand corner point) is obtained by adding another factor of 1.25 to the unit vector to the heart-shape's center in a componentwise manner. The factor of 1.25 in the calculation of the Z component of the maximal point represents

the distance from the heart's center to either of its cusps (which is 1.0) and the additional 0.25 closely approximates the displacement from the upper cusp to the highest point on either of its lobes.

With the coordinates of the maximal and minimal points obtained, we obtain the bounding box of the heart-shape.

3.3.7 Plotting the wireframe of the heart-shaped primitive

As we earlier discussed, the wireframe of an object enables the sketching and preview of objects before adding colour and texture to them. In order to build the wireframe of the heartshaped primitive into BRL-CAD's functionality, we wrote the *rt_hrt_plot()* and *rt_hrt_24pts()* functions. The heart-shaped primitive's wireframe is made up of several ellipses aligned along the Z axis each of which consists of 24 edges. As we progress in the positive Z - axis direction, we use 8 ellipses (with decreasing radii) to frame the upper portions of the left and right lobes. The 24 required points for each ellipse are computed by the *rt_hrt_24pts()* function which computes 24 points which are 15° apart. Finally, the different ellipses are connected together to enrich the wireframe with more isocontours. Along the XY and XZ planes (when $Z = 0$ and $Y = 0$ respectively), the heart-shape's wireframe appears like an ellipse. Along the YZ plane (when $X = 0$), the heart-shape primitive's wireframe is indeed heart-shaped. The algorithm below was used to develop the wireframe of the heart-shape.

Algorithm 1. Plotting the wireframe of the heart-shape

```
1. for points in the +Z directions starting at the upper cusp
        locate centres of ellipses which constitute the upper half of the left lobe
        determine radial vectors in X, Y and Z directions as need be
        determine the 24 points for each ellipse
        draw each ellipse        // Connect the 24 points
        make connections between ellipses
        end at highest point of left lobe         // the maximum turning point
```

```
2. for points in the +Z directions starting at the upper cusp
        locate centres of ellipses which constitute the upper half of the right lobe
        determine radial vectors in X, Y and Z directions as need be
        determine the 24 points for each ellipse
        draw each ellipse        // Connect the 24 points
        make connections between ellipses
        end at highest point of right lobe         // the maximum turning point
3. for chosen levels in the +Z direction starting at the lower cusp
        locate centres of ellipses
        determine radial vectors in X, Y and Z directions as need be
        determine the 24 points for each ellipse
        draw each ellipse        // Connect the 24 points
        make connections between ellipses
        end at upper cusp level        //the maximum turning point
```

3.3.8 Surface representation and raytracing of the heart-shaped primitive

The heart-shaped primitive endowed with a surface representation by raytracing it. Raytracing at its very core consists of solving for the intersection points of a line and a surface. Many interesting surfaces have been written as polynomial functions of position and the heart-shape is not left out. The peculiarity of our work is that we proved that raytracing using the Laguerre-based rootfinder works for sextic equations (those with a degree of 6). In order to do this, we wrote *rt_hrt_prep()*, *rt_hrt_norm()*, *rt_hrt_shot()* and *rt_hrt_print()* functions.

The *rt_hrt_prep()* function prepares a heart-shape for raytracing by verifying that the object in question is a valid heart-shape. If the primitive is a valid heart, then a specific heart structure called *hrt_specific* is created and stored in memory for use by the *rt_hrt_shot()* function. To check the validity of the heart, we first check that atleast one of the three radial vectors has a positive magnitude. Secondly, we check that the value for parameter d is nonzero and is not too large. After, we check that the 3 radial

vectors are perpendicular to eachother. If they are indeed perpendicular to each other, then the heart-shaped primitive is a valid hear. Finally, we compute values for *d*, *v*, *hrt_invRSSR*, *hrt_invsq* and the heart-shape's bounding sphere centered at *v*. After computing these requisite parameters for the heart in *rt_hrt_prep()*, the *rt_hrt_print()* function prints the position vector of the heart-shape's center and the two (2) matrices for transposing, rotating and scaling just to make sure that are correct.

Millions of light rays are shot at the surface of the heart and the pixels where intersections occur are rendered. The *rt_hrt_shot()* function came in handy at this point of our development of the heart-shape. Each point in $\mathbb{E}^3$ is represented by a treble (x,y,z) and the heart-shaped primitive's surface is implicitly represented as a sextic equation as shown in (6) below.

$$f(x, y, z) = (x^2 + 9/4y^2 + z^2 - 1)^3 - z^3(x^2 + 9/80y^2) \tag{6}$$

Each light ray is modeled as a line in $\mathbb{E}^3$ written as a linear equation written in the form $W = Dt + P$ where W = (x,y,z), D = (a,b,c), P = (x_0, y_0, z_0) and *a*,*b*, *c*,x_0,y_0 and z_0 are constants in $\mathbb{R}$. The system of equations for the components of W is given below

$$\begin{aligned} \mathrm{x} &= \mathrm{at} + x_0 \\ \mathrm{y} &= \mathrm{bt} + y_0 \\ \mathrm{z} &= \mathrm{ct} + z_0 \end{aligned}$$

This equation (6) indicates that each point on the line is represented by a unique value for t. To find the points of intersection of the light ray and the surface of the heart-shaped primitive, we substituted x, y and z from (6) above into equation (5). This yielded a new sextic equation (7) in t shown below

$$S(t) = C_6t^6 + C_5t^5 + C_4t^4 + C_3t^3 + C_2t^2 + C_1t + C_0 = 0 \tag{7}$$

where Ci,i = 0÷6 are the coefficients of equation (7) which can be viewed in the Appendix.

The real zeroes of (7) indicate an intersection in $\mathbb{E}^3$. Even if complex roots are returned by the rootfinder, the roots with imaginary components sufficiently close to zero are considered to be real zeroes and are also used as values for t in intersection points. If there are no real solutions, then the ray does not intersect the heart's surface. We determine the coordinates of the intersection points by substituting the values for real t_i in W above.

The Algorithm below shows the intersection of an arbitrary light ray and the heart-shape's surface.

Algorithm 2. Intersect a ray with the heart-shape's surface

```
1. Normalize distance from the heart-shape
2. Generate sextic equation
3. Pass equation through Laguerre-based root finder
4. if (root finder returns other than 6 roots)
      throw exceptions  // root finder did not find roots
   else         // Real roots indicate an intersection in real space
      select real roots among the 6 //complex roots with very small imaginary parts
5. for each real root returned by root finder
       Determine the entry and exit points of the light ray
```

The normal to the surface N at the point of intersection is in the direction of the gradient of *f(x,y,z)*. N := (f_x,f_y,f_z) where f_x, f_y and f_z are the partials derivative functions of *f(x,y,z)* with respect to x, y and z respectively. The function *rt_hrt_norm()* computes the surface normal N to the surface of the heart-shaped primitive.

By substituting $w = x^2 + 9/4y^2 + z^2$–1, (6) becomes

$$w^3–z^3(x^2 + 9/80y^2) = 0$$

and the surface normal is given by the system of equations in below.

$$f_x(x, y, z) = 6\text{x}(\text{w}^2–z^3/3)$$
$$f_y(x, y, z) = 6\text{y}(12/27\text{w}^2–88/3z^3)$$
$$f_z(x, y, z) = 6\text{z}(\text{w}^2–z/2(x^2 + 9/80y^2))$$

Computing the exact algebraic expression for the coefficients of (7) above is cumbersome and error-prone. Instead, we use several polynomial variables to hold the coefficients of (7) and gradually build equation (7). Starting with polynomials for x^2, $9/4y^2$ and $z^2 - 1$, we obtain w. With $y^2 + 9/4x^3$ and z^3 , we obtain $z^3(x^3 + 9/80x^3)$. Hence, we obtain coefficients of (7).

Once the coefficients of (7) are determined, it is parsed through BRL-CAD's root finder. For polynomials with degrees less than five, there exists exact solutions in radicals. Indeed, linear and quadratic equations can be trivially solved by the substitution method and the quadratic formula respectively. A method for solving cubics was discovered by Cardan and a method for solving quartics was discovered by Ferrari. Although some methods exist to determine the exact solutions of higher order polynomials (with degrees greater than 4) in radicals from Galois theory, the equation (7) does not satisfy these conditions. Indeed, the Galois group of equation (7) is contained neither in the group of order 48 which stabilizes a partition of the set of the roots into three subsets of two roots nor in the group of order 72 which stabilizes a partition of the set of the roots into two subsets of three roots. As a result, only numerical methods can be employed to solve it.

Fortunately, BRL-CAD has a rootsolver which is based on a numerical method called the Laguerre method. Named after the French mathematician Edmund Laguerre, the Laguerre method is a rootfinding algorithm used to solve polynomials. Extensive empirical studies show that this method has come close to being a surefire method because it almost always converges to some root of the polynomial no matter what initial guess is chosen. This is in contrast to the other methods like the Newton-Raphson method which may fail to converge for poorly chosen initial guesses. Algorithm 3 below shows how the Laguerre method works;

Algorithm 3. The Laguerre Root-finding Method

1. Choose an initial guess x_0
2. while (k ¡ N or a is not sufficiently small)
2.1. for k = 0÷N

Compute G := f'(x_k)/f(x_k)
Compute H := G^2–f''(x_k)/f(x_k)
Compute a := $n/G + \sqrt{(n–1)(nH–G^2 f(x))}$,n = degree of f(x)
end for loop
2.2 Set $x_{k+1} := x_{k–a}$
end while loop

If a root is found, then the corresponding linear factor is removed from the polynomial. This deflation step reduces the degree of the polynomial by one and approximations for all roots of the polynomial are obtained.

BRL-CAD's rootfinder *rt_poly_findroot()* function is found in ***src/librt/roots.c***. We tested the root solver to show that it solves sextic equations by parsing an arbitrarily chosen equation (8) below

$$x^6 - 8x^5 + 32x^4 - 78x^3 + 121x^2 - 110x + 50 = 0 \tag{8}$$

Equation (8) is parsed through the rootfinder to obtain its roots $x_1 = 1–i$, $x_2 = 1 + i$, $x_3 = 2–i$, $x_4 = 2 + i$, $x_5 = 1 - 2i$, $x_6 = 1 + 2i$. This test is implemented in ***src/util/roots_example.c***. Besides this test, the call to the rootfinder by (6) worked and rendered the heart-shape as our discussion of results in the next chapter will show.

3.3.9 Type in support for the heart-shaped primitive

Finally, appropriate support for typing the parameters of the heart-shaped primitive using the keyboard into the mged or archer interfaces was implemented. To do this, we first wrote the *p_hrt[]* array and *hrt_in()* routine in ***src/libged/typein.c*** to prompt users to input parameters for the heart-shape. Then ,we added the *mk_hrt()* function to the include/wdb.h and ***src/libwdb/wdb.c*** files to assign values from **mged** or **archer** interfaces to the heart-shape's parameters.

In this chapter, we introduced the concept of open source software because our research was carried out within this context. Secondly, we described the procedure for constructing the heart-shaped primitive. After, we outlined the design of the heart-shaped primitive and the implementation of appropriate callback functions to compute geometrically important properties for the heart-shape.

Chapter 4

RESULTS & DISCUSSION

In this chapter, we present and discuss the results obtained from our work. We explain how we tested the different geometric properties of the heart-shaped primitive. Without loss of generality, we used a heart-shaped object centered at the origin (0,0,0), possesses 3 radial vectors (5,0,0), (0,5,0) and (0,0,5) as well as a distance to cusps of 4. Let's suppose this object called *amour* and is stored in the *heart_example.g* database. Given the situation of our work in the field of CAD, we use images to demonstrate that each of the heart-shape's geometric properties work.

4.1 Type-in support for the heart-shaped primitive

During the modeling process, a designer using BRL-CAD creates objects by typing its parameters into either the mged or archer graphical user interfaces. Having built this capacity into the heart-shaped primitive, we used BRL-CAD's ***in***[38] command to test its mettle.

The ***in*** command enables the user to type in the arguments needed to create a shape alongside its name and type. It supports various options and may be invoked with no arguments. The ***-s*** option invokes the primitive edit

mode on a new object immediately it is created. In order to test the type in support of the heart-shaped primitive, we create the *amour* object by typing its name, type and other parameters into the archer interface. After opening the *heart_example.g* database database destined to hold the *amour* object, we type ***"in amour hrt 0 0 0 5 0 0 0 5 0 0 0 5 4"*** into archer's command line interface. The object's name, *amour*, is printed on the command line indicating that the object has indeed been created. Figure 4.1 below shows that the *amour* heart-shape object can be created by typing in its parameters through the keyboard.

```
Archer> in
Enter name of solid: amour
Enter solid type: hrt
Enter X, Y, Z of the heart vertex: 0
Enter Y: 0
Enter Z: 0
Enter X, Y, Z of vector xdir: 5
Enter Y: 0
Enter Z: 0
Enter X, Y, Z of vector ydir: 0
Enter Y: 5
Enter Z: 0
Enter X, Y, Z of vector zdir: 0
Enter Y: 0
Enter Z: 5
Enter distance to cusps: 4
amour
Archer>
```

FIGURE 4.1: Testing type in support for the heart-shaped primitive in archer

4.2 Formatted description of the heart-shaped primitive

After having created objects for modeling, it sometimes becomes necessary to display a description of these objects. For us to test that we can describe the heartshaped primitive, we used BRL-CAD's ***l***[39] command on the amour object. The ***l (listing)*** command displays a verbose description of a specific list of objects. If the shape of the object is a primitive, then detailed parameters of that shape are displayed. If the object is a combination of other primitives, then the boolean formula for the combination is listed while indicating any accumulated transformations. If a shader and colour has been assigned to the combination, then all details will be listed. The ***-t*** (terse) option displays a shorter list of primitive shape parameters.

To describe the *amour* object, we print amour's parameters in both terse and verbose forms by running the ***"l -t amour"*** and ***"l amour"*** commands respectively in the archer command prompt. This is shown in Figure 4.2 below.

```
Archer> l -t amour
amour:  Heart (HRT)
        V (0, 0, 0)
        Xdir (5, 0, 0) mag=5
        Ydir (0, 5, 0) mag=5
        Zdir (0, 0, 5) mag=5
        d=4

Archer> l amour
amour:  Heart (HRT)
        V (0, 0, 0)
        Xdir (5, 0, 0) mag=5
        Ydir (0, 5, 0) mag=5
        Zdir (0, 0, 5) mag=5
        d=4
        Xdir direction cosines=(0, 90, 90)
        Xdir rotation angle=0, fallback angle=0
        Ydir direction cosines=(90, 0, 90)
        Ydir rotation angle=90, fallback angle=0
        Zdir direction cosines=(90, 90, 0)
        Zdir rotation angle=0, fallback angle=90
```

FIGURE 4.2: Testing the formatted description of the heart-shaped primitive

4.3 The bounding box of the heart-shaped primitive

As we earlier stated, the *rt_hrt_bbox*() function was implemented to calculate the bounding box of the heart-shaped primitive. In order to test that the bounding box of the heart-shaped primitive is computed, we use BRL-CAD's ***bb***[40] command.

The ***bb (bounding box)*** command reports dimensional information about objects using bounding boxes. It does this by calculating an axisaligned bounding box for an object and printing the dimensions of that box to the command prompt of archer. The ***bb*** command support various options, most of which control the type of information reported.

- The ***-e*** *(extent)* option reports the extent of the bounding box by printing its minimal and maximal points.
- The ***-d*** *(default)* option reports the length, width and height of the box.
- The ***-v*** *(volume)* option prints the volume of the bounding box by default too.
- The ***-q*** *(quiet)* option prints the properties of an object in quiet mode by disabling the printing of the default header.

Once more we use the amour object and the *bb* command to test how effectively the bounding box of the heart-shaped primitive was implemented. To report the extent of the bounding box of the amour object, we print its minimal and maximal points by running the ***"bb -qe amour"*** command in either the mged or archer command prompts. Then, we ran the ***"bb -qv amour"*** command in archer so that the volume of the amour object is reported in cubic millimeters. After, we reported the length, width and height of amour's bounding box by running the ***"bb -qd amour"***. Finally,

to report the volume and dimensions of the bounding box, we ran the ***"bb amour"*** command in archer's command prompt.

The image above in Figure 4.3 shows the results obtained after running the aforementioned commands.

```
Archer> bb -qe amour
min {-3.333333 -5.000000 -5.000000} max {3.333333 5.000000 6.250000}

Archer> bb -qd amour
X Length: 6.7 mm
Y Length: 10.0 mm
Z Length: 11.2 mm

Archer> bb -qv amour
Bounding Box Volume: 750.0 mm^3

Archer> bb amour
Bounding Box Dimensions, Object(s) amour:
X Length: 6.7 mm
Y Length: 10.0 mm
Z Length: 11.2 mm
Bounding Box Volume: 750.0 mm^3

Archer>
```

FIGURE 4.3: Testing the bounding box of the heart-shape

SECTION 4

4.4 Plotting the wireframe of the heart-shaped primitive

The process of modeling sometimes warrants the preview of the skeleton of a specific set of objects. To test that the wireframe of the heart-shaped primitive is working, we use the ***draw***[41] command in BRL-CAD. The ***draw*** command displays objects in either the mged or archer interfaces. It is synonymous to BRL-CAD's ***e*** command. The draw command's ***-C*** *(colour)* option enables the user to specify a colour that overrides all other previous colour specifications.

To draw the wireframe of the amour object using white wires, we run the ***"draw -C 255/255/255 amour"*** command. The image in Figure 4.4 below shows the wireframe of amour with red isocontours.

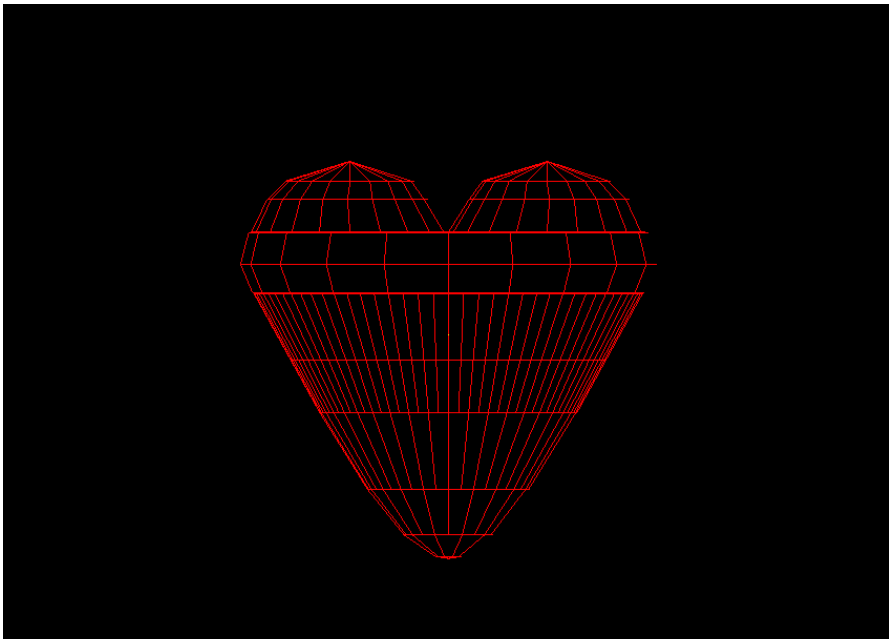

FIGURE 4.4: Testing the wireframe of the heartshaped primitive

4.5 Main Section 2

Sed ullamcorper quam eu nisl interdum at interdum enim egestas. Aliquam placerat justo sed lectus lobortis ut porta nisl porttitor. Vestibulum mi dolor, lacinia molestie gravida at, tempus vitae ligula. Donec eget quam sapien, in viverra eros. Donec pellentesque justo a massa fringilla non vestibulum metus vestibulum. Vestibulum in orci quis felis tempor lacinia. Vivamus ornare ultrices facilisis. Ut hendrerit volutpat vulputate. Morbi condimentum venenatis augue, id porta ipsum vulputate in. Curabitur luctus tempus justo. Vestibulum risus lectus, adipiscing nec condimentum quis, condimentum nec nisl. Aliquam dictum sagittis velit sed iaculis. Morbi tristique augue sit amet nulla pulvinar id facilisis ligula mollis. Nam elit libero, tincidunt ut aliquam at, molestie in quam. Aenean rhoncus vehicula hendrerit.

Appendix A

Appendix Title Here

Write your Appendix content here.