Computational Geometry for Playful Creation and Control

A capstone thesis submitted by

Stephanie Atherton
satherton@student.otis.edu

in partial fulfillment of the requirements for the award of the degree of

BFA in Toy Design,
Minor in Art & Design Education

Otis College of Art & Design
Liberal Arts & Sciences
December, 2024

Contents

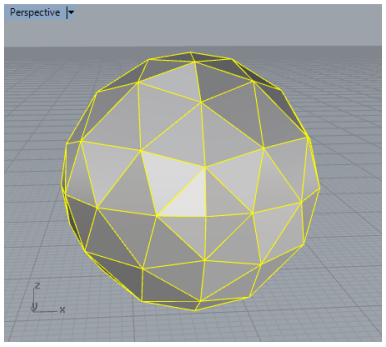

Figure 0.1: Triangulated sphere created via Rhino3D[Mar19]

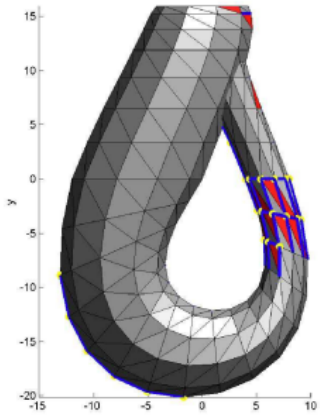

Figure 0.2: Triangulated 3D visualization of a klein bottle, via Duke DoMath 2021 [LMN21]

Preface

The season before this capstone was written, I spent the summer at the IU Bloomington NSF-funded "Research Experiences for Undergraduates" (REU) researching "combinatorial holonomy", as initially introduced by a previous summer research team at Duke University in 2021. While geometry has an obvious visual place in the body of art and design, I did not expect to be inspired by any direct applications during my summer off from Otis.

Yet in reading the Duke team's notes on triangulated surfaces in preparation for my own project, I couldn't help but recall the "triangulated surfaces" which populate a mesh in Rhino3D. It was a very interesting summer, and I came back to school wondering if there was something I could write that could make computational geometry - the driver behind programs like CAD or Rhino 3D - more interesting to other students.

Figure 1.1: The author's kindergarten pinch pot creation[Ath24k]

1 An Artistic Introduction

One of the simplest objects to mold from clay is the pinch pot, so we'll start there as an artistic analogue to computational geometry. Perhaps you'll recall from your elementary art class years how typically goes about making one:

1. Start with a ball of clay.
2. Push your thumb into the center of the ball, creating a hollow.
3. Pinch and turn the pot to create walls.
4. Press the pot down to create a base
5. Repeat 3.) for smooth surfaces until the pot is perfect! [Ben21]

One essentially always begins with a simple ball in claywork, and yet the sculptures which can arise in the forming process are infinite (when equipped with artistic inspiration). Recursively, any convex polyhedra P can be thought of as a "triangulated" version of the sphere S^2 by an embedding $f : P \hookrightarrow S^2$ in $\mathbb{R}^3$. A polyhedra $P := (V, E, F)$ is a three-dimensional solid consisting of a collection of polygonal faces $f \in F$ joined at edges $e \in E$ by vertices (i.e. corners) $v \in V$ invariant under the Euler characteristic $\chi = V - E + F = 2$. Now to mold our polyhedron into a sphere:

1. Start with any convex polyhedron.
2. Make one of two moves:
   (a) Merge 2 faces $f \in F$, consequently deleting an edge e and a face f.
   (b) Remove a terminal vertex $v \in V$, consequently deleting a vertex v and edge e.
3. Repeat 2a.) and/or 2b.) until you are left with a nice, smooth sphere $S^2 := (1, 0, 1)$ for the tuple (V, E, F).

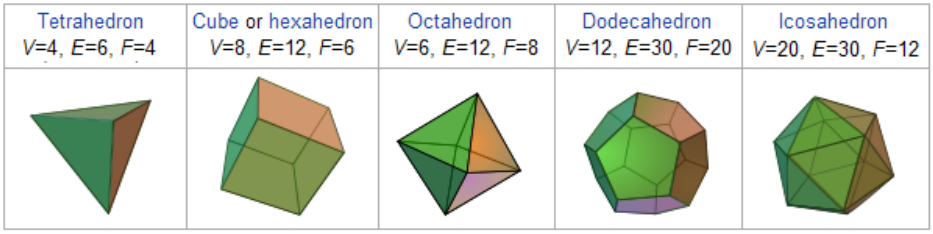

Figure 1.2: The convex polyhedra known as the *platonic solids* with vertices, edges, and faces enumerated. Try computing the Euler characteristic $\chi = V - E + F$ for yourself! [lim13]

4. Check the Euler characteristic of S^2 vs. that of the polyhedron you started with.

You should see that the Euler characteristic of the sphere S^2 is still $\chi = 2$ as an invariant, hence we say any polyhedral surface is homeomorphically equivalent to the sphere. By *homeomorphic*, we mean the shapes are the "same" despite their change in form. This may be difficult to initially grasp, but just as the unbaked pinch pot can be deformed back into the ball of clay reversing the steps as set earlier, we have just deformed a polyhedron (an octahedron let's say) into a sphere by carrying out a certain algorithmic procedure. Computational geometry, as applied to 3D programs, possesses a similar marvel in its ability to form and deform models for both visual and mathematical delight.

While subject to strict mathematical rules and procedures, computational geometry contributes creatively to the production of the more whimsical products you see on shelves. Hard toys for example have developed from handmade playthings to manufactured models that populate children's bedrooms and imaginations. As the rules for safe design of toys has evolved into practice, the mathematical rules for computational geometry have emerged alongside to aid in the production of joyful, juvenile products [HL23].

Students who may not be well-versed in the study of computational geometry may discover not only its accessibility for artistic shape creation and control, but also the pleasures it brings through the free-form design of hard toys. In a quirky approach to computational geometry, we hope to excite student curiosity in the subject through its playful interface with modern toy design.

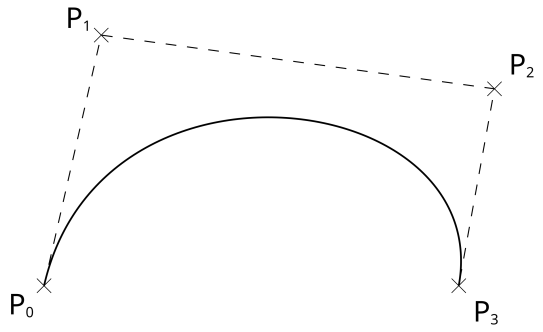

Figure 2.1: Cubic Bezier curve with 4 control points, which are used for modification of a curve [Com20]

2 Background and Relevance to Design

The 3D modeling program of choice for toy design students is Rhino 3D, a CAD tool known for its versatility in the design and rapid prototyping of hard toys. We define *geometric modeling* as the "mathematical representation of curves, surfaces, and solids" of a toy and *computational geometry* as its mathematical and algorithmic implementation [Pat13]. Rhino 3D depends on NURBS (Non-uniform Rational B-Splines) geometry in particular, and allows for the freedom of freeform modeling and precision for production within a single computer graphics application [Iss13]. We can digitally create, modify, and analyze the structure of our toy in a way that mimics the regular iterative process of design. Note that there are various geometric representations of surfaces. For polygon mesh representations used in animation programs such as Maya, consider our introductory example of turning a polyhedra into a sphere. The NURBS system on which Rhino 3D relies is based on *parametric curves*. Specifically, it extends B-splines, which themselves are composed of Bézier curve segments, to allow for greater control and flexibility through weights and non-uniform parameterization. Now imagine:

A cubic planar Bezier curve:

$$R(t) = \sum_{i=0}^{3} R_i B_{i,3}(t) \qquad 0 \leq t \leq 1$$

has the following control points

$$R_0 = [0,0], R_1 = [2,2], R_2 = [3,2], R_3 = [3,0]$$

A designer decides to subdivide (split) the curve at $t_0 = 1/2$ and $t_1 = 2/3$ in order to modify the curve in the interval $[t_0, t_1]$ and generate a particular shape feature required by his design [Pat13].

It is relevant to note that many toys, especially for younger children, abide by soft curves for both stylistic and safety standard purposes. Youth is often associated with softness of heart as well as of features, and indeed many drawings of and for youth attempt to echo that innocence. Further,

Figure 2.2: Whimsical sketches by a toy design student [oAD24]

Figure 2.3: Preschool toy car with rounded silhouette and chibi figures [FP24]

little hands can't be trusted with hard-edged toys. It is in fact a real design as well as a real math problem then that toy designers need to have creative control over curves in order to appeal to youth aesthetic and developmental sensitivities [Pla]. Referring back to our continuous Bezier curve represented by:

$$R(t) = \sum_{i=0}^{3} R_i B_{i,3}(t) \qquad 0 \leq t \leq 1,$$

note that the coefficients R_i serve as the control points for our curve $R(t)$ with the Bernstein polynomial as a basis. Given our Bézier curve is cubic, it is defined by a continuous curve formed using a discrete set of four control points $\{R_0, R_1, R_2, R_3\}$ which *we* (as designers) control. These points, combined with the cubic Bernstein basis functions $B_{i,n}$ determine the shape of the curve [PMC09].

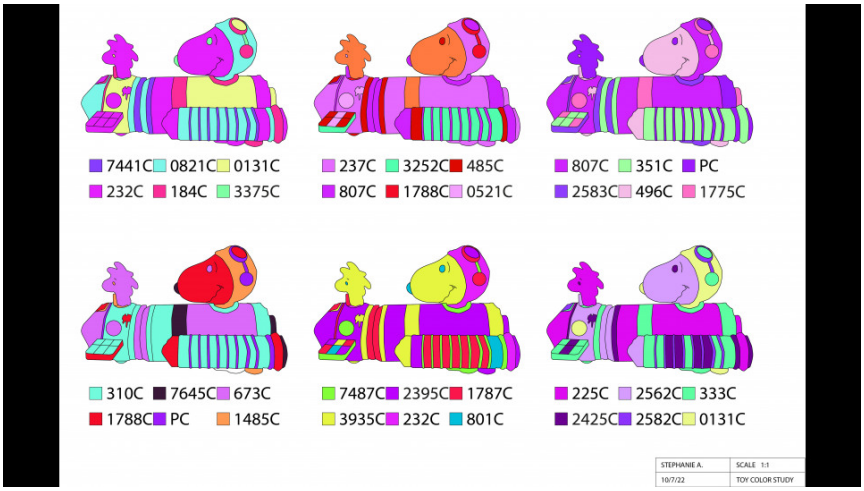

Figure 3.1: Color sheet of Snoopy as the WWI flying ace via vector-based software Adobe Illustrator, courtesy of the author[Ath22c]

3 A Geometric Approach to Vectors

Computer graphics enable the creation, manipulation, and visual display of geometry, art, and images, often using vector-based or raster-based software. Vector-based software is widely used to create and render 2D images, often referred to as "vector graphics", as seen in tools like Adobe Illustrator. It also forms the foundation for many 3D programs, enabling the creation and manipulation of scalable, mathematically defined geometries [Lib23]. Considering our earlier mention of the Bernstein polynomials as a basis for the Bezier curve, hopefully it comes as no surprise that mathematical vectors also go a long way in helping us to understand the computational geometry behind Rhino 3D.

You may recall a vector as an arrow with direction and force from a high school physics class. However, as functional analyst Berberian would argue, "That's an intuitive glimpse of vectors. The way mathematicians look at them (as a part of 'abstract algebra') leads to the idea of 'vector space'. In between this austere view and the intuitive idea, the example of 'geometric' vectors in 'ordinary 3-space' is fun and instructive; let's look at it" [Ber92]. It may also help to recall that any vector space can be considered an affine (and hence geometric) space should we fix the zero vector as an origin point o. Now all the other vectors can be viewed from the perspective of points too! Our vector v is the distance between two points p_1 and p_2 in $\mathbb{R}^3$, where any point p can be represented by coordinates (x, y, z). Then

$$\overrightarrow{p_1p_2} = [x_2 - x_1, y_2 - y_1, z_2 - z_1]$$

is the distance between p_1 and p_2, and we call $\overrightarrow{p_1p_2}$ the vector determined by the arrow from p_1 to p_2. Subtracting the components (x, y, z) of p_1 from p_2, the ordered triple $v = [a, b, c]$ recognizes the change in components along each axis and is a proper representation for any vector v.

Figure 4.1: A 3D printed Optimus Prime, created via a vector-based CAD program [Chu21]

4 Matrix Transformations and Modeling

Just as a ball of clay can be shaped into a pinch pot, we use linear transformations of form $T : \mathbb{R}^n \to \mathbb{R}^m$, mapping vectors from $\mathbb{R}^n$ to $\mathbb{R}^m$, to reshape geometry in Rhino 3D. Let c be a column vector with n entries. Then the transformation is given by $T(c) = Ac$, where A is the transformation matrix of T. Through matrix multiplication, A reshapes the column vector c, enabling transformations such as scaling, rotation, or translation to produce complex objects, including toys like Transformers).

In computer graphics we use the *homogeneous* coordinates (x, y, z, w) of projective space rather than the coordinates (x, y, z) of three dimensional Euclidean space. An artist might think of projective space as "perspective" space, where all lines meet at a point. The term "homogeneous" often implies "one", reflecting the scale invariance of homogeneous coordinates. Any Euclidean point can be represented by infinitely many homogeneous coordinates, as scaling by any non-zero value w leaves the point unchanged. See that:

$$(x, y, z, w) \iff (\frac{x}{w}, \frac{y}{w}, \frac{z}{w}, \frac{w}{w}).$$

[Ahn12]

Then a Euclidean point $p(x, y, z)$ can be represented by a column matrix $\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$ in the projective space of 3D computer graphics. By operating on 4x1 column vectors c representing points p with 4x4 transformation matrices, we transform points in projective space to "mold" new 3D shapes. These matrices allow for transformations such as identity (no transformation), rotation, scaling, translation, reflection, and shearing in projective space. We proceed to illustrate several examples of these 4x4 "molding matrices" relevant to any design program.

Identity: The "non-transformation" - no changes are made to elements of the space.

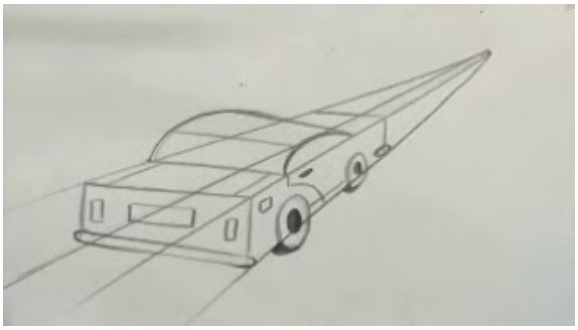

Figure 4.2: A car drawn in perspective (i.e. projective space) [Art20]

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about z axis by an angle k:

$$\begin{bmatrix} \cos(k) & \text{-}\sin(k) & 0 & 0 \\ \sin(k) & \cos(k) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about x axis by an angle k:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(k) & \text{-}\sin(k) & 0 \\ 0 & \sin(k) & \cos(k) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about y axis by an angle k:

$$\begin{bmatrix} \cos(k) & 0 & \sin(k) & 0 \\ 0 & 1 & 0 & 0 \\ \text{-}\sin(k) & 0 & \cos(k) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scale: Enlarges elements of the space.

$$\begin{bmatrix} \text{scale } x & 0 & 0 & 0 \\ 0 & \text{scale } y & 0 & 0 \\ 0 & 0 & \text{scale } z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translation: The "move" move - can typically be performed by simply clicking and dragging an object in the design program. Mathematically speaking, given a point $p(x, y, z)$ and a *translation vector* $v = [a, b, c]$, we construct a translation matrix of form

$$\begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

altering the location of a point p in projective space when applied to its column matrix.

Reflection: "Mirrors" elements of the space. To flip a 3D object across a plane, we flip the sign of a 1 (or multiple 1s) in a matrix to -1. For example, to flip a friendly robot "Fred" across the x axis as shown in figure 4.3, we apply the matrix:

$$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Similarly, to flip Fred across the y axis we would apply the matrix:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Shearing: Shearing essentially slants an object along one axis while keeping the other axes fixed. This transformation preserves the object's area but alters its shape. For example, here we shear along the y-axis while keeping the z-axis fixed [Iss13]:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & .5 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

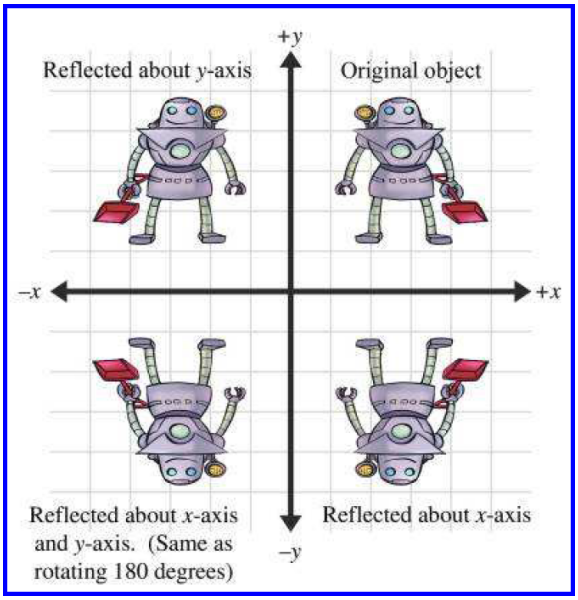

Figure 4.3: Reflecting a friendly robot character [Gao20]

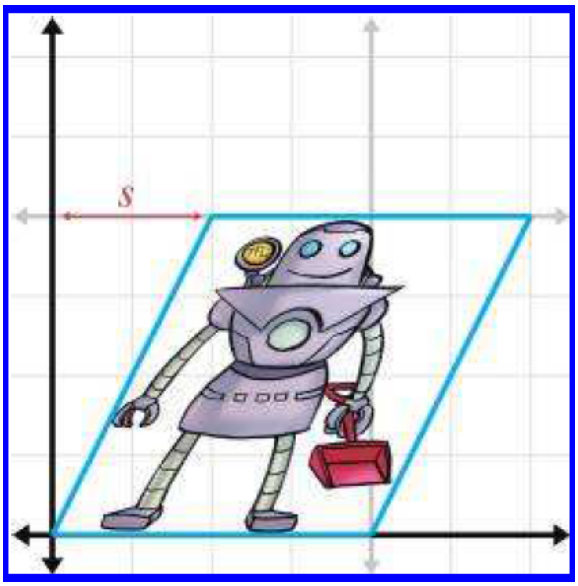

Figure 4.4: Shearing a friendly robot character [Gao20]

Figure 5.1: Hand-carved Persian toy chariot, circa 2000 B.C. [Cas16]

5 Moulds to Accessible Making

If any of those matrices in the prior section managed to "shear" your mind, let us remind you of their amusing application as a modern method to model hard toys. However, once upon a time in a faraway land such as ancient Persia, all toys were handmade, carved, or sculpted with love by elders in the community or by skilled children themselves. Yet as time and technology progressed, new methods to create toys eventually pushed traditional toymakers out of business [Goe18]. Nevertheless, there will never be a post-play era, and changing times and modern ways of making have also accomplished unprecedented creativity for new categories of novelties [TIE13].

Indeed, the toy industry divides its products into several mainstream categories, such as vehicles, dolls, preschool, and the smart toys emerging in the digital age. Many toys are considered

Figure 5.2: The author's GaliLEGO set is a prime example of a hard toy [Ath23a].

Figure 5.3: The author's American Girl doll Joonie, with "Jess" face mold [Ath22e]

Figure 5.4: Vacuum-formed parts of a toy cabin-camper [Ath22f]

"hard toys" (soft toys being plush) or at least have hard components sourced from plastic. American Girl dolls for example maintain huggable tummies but taut, slightly squishy limbs and heads derived from hollowed out plastic moulds. Even then, doll faces are made by traditional machinery methods rather than by direct modeling via 3D programs. American girl doll heads (like many doll heads) are formed using rotational molding, in which liquid plastic is rotated within a heated face mold. As the mold cools and plastic cures, one is left eventually with a full-cheeked, rounded face with two little buckteeth (as is signature of all American Girl dolls) [Pla22b].

Another popular machine-method of production is vacuum-forming, which follows the below steps:

1. A plastic sheet is placed in a rigid frame.
2. It is reheated to soften the plastic, but not to the point of melting it.
3. It is then sucked down into a mould which provides shape.
4. Now the plastic cools/hardens.

[Pla22a] Yet note that often a designer may have to sculpt the mould for the machine themselves - and that mould has to be nearly *perfect*, otherwise all toys produced will come out imperfect.

Hopefully not by the exposition of this paper, but perhaps there was ever a time you believed yourself not to be "mathematically inclined". The good news is that you don't have to be "mathematically inclined" to enjoy math, nor do you have to be a skilled artist to make art (otherwise our world would be devoid of kindergarten pinch pots). Not all designers can apply their fine motor skills with grand finesse, and likewise, not all artists figure themselves "creatively inclined" [Com13].

Remark. *Inclination by meaning of aptitude is a natural form of nonsense - we're really best off pursuing whatever it is (within reason) we feel inclined to do! Children really play games to play in the end, not for the "end-game" of triumph or talent that sometimes overshadows the fun of it all.*

Figure 5.5: Painted and finished vacuum-formed cabin
[Ath22d]

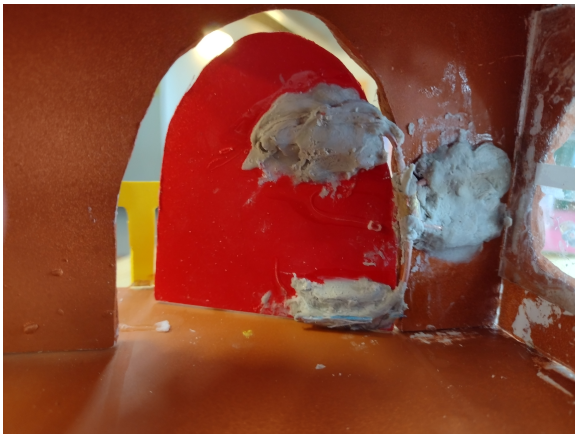

Figure 5.6: A heartfelt, handmade attempt at a working door. Good thing we have machines to help us in holding together our real houses and vehicles! [Ath22a]

Figure 5.7: An inside look at our cabin-camper, hand-constructed with a lack of finesse replaced by globs of clay [Ath22b]

An interesting combinatorics problem LEGO is working on at the time of this paper, one which the author was invited one summer to help solve along with the experience designer himself, is "which bricks to put in the box" in order to "optimize creativity" per se [Car23]. Indeed, LEGO and its respective Guided Creativity department prioritize not only making creativity accessible, but also making its playful products accessible for the physically disabled [Fou24].

While we continue to consider the prior remark, it is true that some are simply more "able" than others in art and craftsmanship. In the next section, we hope to illustrate (with at least argumentative finesse) that 3D modeling allows for a type of accessibility for designers to achieve what we denote as "parametric perfection".

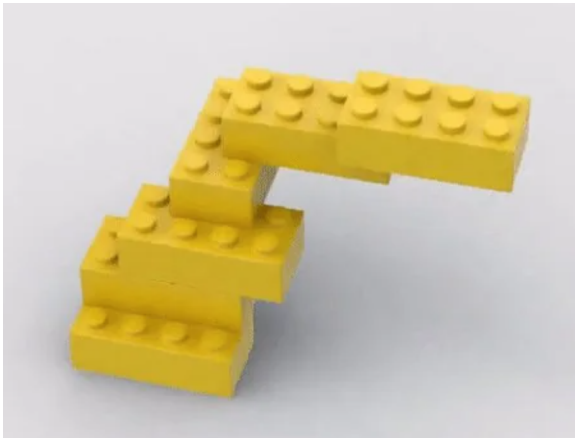

Figure 5.8: There are 102,981,504 combinatorial ways to stack six 2×4 LEGOs into a tower of height six [Kag21]

Figure 6.1: Nor is the visual design of a toy a toy [Mag29]

6 Parametric Precision for Play

A designer's (internal) need for parametric precision is further shaped by external standards - those upheld by regulatory bodies for juvenile products. A designer must consider several constraints (or *parameters*) then if they wish for their play product to pop up on toy store shelves one day:

1. Is this design suitable and safe for children?
2. How do I design my toy such that it can be physically manufactured?
3. Can the parts of my toy be properly assembled for play?

Now imagine the designer has limited control, and hence limited capacity to comply with these constraints:

1. If the designer cannot manage dull edges, children will scrape themselves!
2. A non-manfacturable toy is not a toy, as a toy that cannot be physically played with is not a toy.
3. For a toy that is made of several parts, they can never be joined to make a single plaything. Conversely, for single construction blocks used to create LEGO sculptures, they now have no play value!

Graphics programs such as Rhino 3D are a great aid in meeting the design constraints set forth as well as meeting the *design intent* of the individual behind the keyboard. The concept of design intent is closely associated with the *parametric design* methods certain CAD programs offer [Ass].

Figure 7.1: Rendering of a playset with lots of curves - check out the swirly slide! [Ath24b]

7 Control and Creation of Curves

Just as a "complicated" toy may be comprised of simpler, separately manufactured pieces that have been assembled, complicated curves (i.e. Bezier splines) are created by joining simpler curve pieces (i.e. Bezier curves). Bezier curves actually depend on *piecewise polynomials*. For all pieces to join smoothly, their tangent directions must match up at the endpoints. Then the designer needs to be able to easily control:

1. The starting and ending points of the curve,
2. The tangent direction at starting and ending points.

This level of control is made possible via the use of Bézier cubics. "With practice and experience, a designer can become proficient in using Bezier cubics to create a wide variety of curves. It is interesting to note that the designer may never be aware of equations . . .that are used to describe the curve" [COL18].

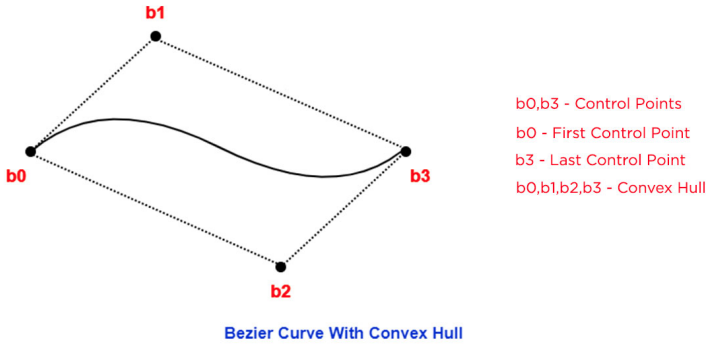

Figure 7.2: Convex hull property of a Bezier curve [Roy]

A Bézier cubic, defined by $n+1$ control points $\{R_0, R_1, R_2, R_3\}$, is a degree $n = 3$ curve represented by:

$$R(t) = \sum_{i=0}^{3} R_i B_{i,3}(t) \qquad 0 \leq t \leq 1,$$

where the curve is defined using Bernstein basis polynomials:

$$B_{i,n}(t) = \binom{n}{i}(1-t)^{n-i}t^i$$

restricted to the interval [0,1] where

$$n = 3 := B_0(t) = (1-t)^3, B_1(t) = 3(1-t)^2 t, B_2(t) = 3(1-t)t^2, B_3(t) = t^2.$$

Substituting into the curve equation, the Bézier cubic becomes:

$$R(t) = (1-t)^3 R_0 + 3(1-t)^2 t R_1 + 3(1-t)t^2 R_2 + t^3 R_3,$$

where $R(t)$ is an affine combination of the control points. Parametrically for points x, y where $x_0, y_0, x_1, y_1, x_2, y_2, x_3, y_3$ are constants visually pleasing to the designer, our curve can also be expressed by:

$$x(t) = (1-t)^3 x_0 + 3(1-t)^2 t x_1 + 3(1-t)t^2 x_2 + t^3 x_3,$$
$$y(t) = (1-t)^3 y_0 + 3(1-t)^2 t y_1 + 3(1-t)t^2 y_2 + t^3 y_3$$

for $0 \leq t \leq 1$. As t travels from 0 to 1, we describe a curve from beginning to end starting at control point (x_0, y_0) and ending at (x_3, y_3).

Bernstein polynomials are quite computationally and creatively considerate, possessing the attributes of :

- Non-negativity

Figure 7.3: How to parametize a pink donut - a toy torus [Muf]

  - Ensuring that the contributions of each polynomial are always positive, aiding stability

- Partition of Unity
  - Meaning the sum of all basis polynomials is always 1, giving rise to the *convex hull*. This ensures the Bézier curve remains contained within the set of its control points. This property also allows for invariance under geometric transformations.
- Symmetry
  - Making it easier to create symmetric curves
- Recursion
  - Allowing for efficient rendering algorithms [Nie12]

Surely, designers "need curves and surfaces that are varied in shape, easy to describe, and quick to draw. Parametric and rational functions satisfy these requirements . . . " [COL18]. When we represent curves and surface parametrically, we represent them in a way that our geometric

object maintains its shape over a parameter range. For Bézier curves in computer graphics, this parameter t is typically restricted to $0 \leq t \leq 1$. More concretely, one might parametize the surface of a pink toy donut with minor radius r and major radius R by

$$x = \cos(t)(R + r\cos(u)),$$

$$y = \sin(t)(R + r\cos(u)),$$

$$z = r\sin(u))$$

where parameters t, u vary from 0 to 2π. Then choosing any $0, \leq t, u \leq 2\pi$ which satisfy the above equations will always produce a donut. While neither *explicit* nor *implicit* representations, parametric systems are indeed the best candidate to carry out design intent. Our four control points coupled with the creatively considerate properties of the cubic Bezier curve allow designers to facilely manipulate and maintain shapes akin to their artistic vision *and* childrens' needs for play.

Figure 8.1: 3D-printed coder character Lovelace, originally modeled in Rhino 3D [Ath24l]

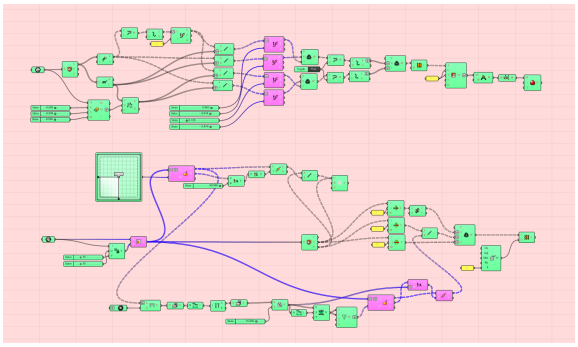

Figure 8.2: Grasshopper UI with sample code [eki18]

8 Interplay of Computational Creativity and Geometry

Not only does the computation behind 3D programs enhance the designer's ability to have creative control over their designs, but code can also be directly implemented to achieve a desired form. Beyond that, code can serve as a creative medium in its own right. Rhino 3D offers *computational design* capabilities via its grasshopper extension. Grasshopper allows for node-graph programming to parametrically create complex geometry - as particularly useful in disciplines like architecture and jewelry design. Additionally, one can script in rhino.python to achieve decorative product components. Code implementations may not offer the same "control-click" most designers are used to, but designing geometry algorithms offers a different "think-before-you-touch" approach conducive to clarifying new forms and efficient workflows.

Computational thinking has also made its way into STEM toys, pairing play with Python programming to create flashing rainbow light shows or fashionable keychain GIFs. Youth robotics

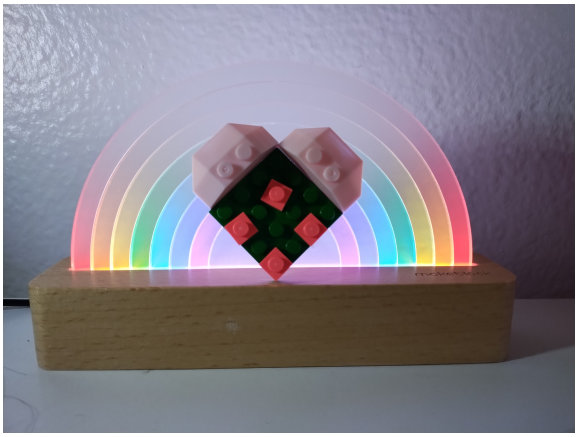

Figure 8.3: Makeblock xLight code-controlled rainbow lightshow [Ath23b]

Figure 8.4: Python-implemented sparkly heart GIF on imagiCharm [ima20]

also typically relies on block coding, and competitive leagues are a fantastic way to get children collaborating with each other. As a former robotics assistant coach, I have witnessed first-hand the convoluted creativity which drives elementary school girls to design the "Most Annoying Code" for their end user (viz. little brothers).

While modern algebraic geometry is often considered to be very abstract, classical applications arise (clearly) in 3D CAD programs as well as robotic automation, in which the possible "states" of robotic limbs r can be regarded as a tuple in $\mathbb{R}^{r^2}$. The subset of all *possible* states in $\mathbb{R}^{r^2}$ forms an affine variety.

If k is a field (like $\mathbb{R}$), and $f_1, ..., f_s$ are polynomials in $k[x_1, ..., x_n]$, then the affine variety $\mathbf{V}$ defined by $f_1, ..., f_s$ is:

$$\mathbf{V}(f_1...f_s) = \{(a_1...a_n) \in k^n \mid f_1(a_1..a_n) = 0 \,\text{for all}\, 1 \leq i \leq s\}.$$

Thus, an affine variety $\mathbf{V}(f_1...f_s) \subseteq k^n$ is the set of solutions of the system $f_1(x_1...x_n) = ... = f_s(x_1...x_n) = 0$. Visually, these solutions correspond to a shape or curve, sometimes intricate ones like the twisted cubic, defined by the variety $\mathbf{V}(y - x^2, z - x^3)$.

Conversely to the way that code or clicks can directly create shapes for toys in Rhino

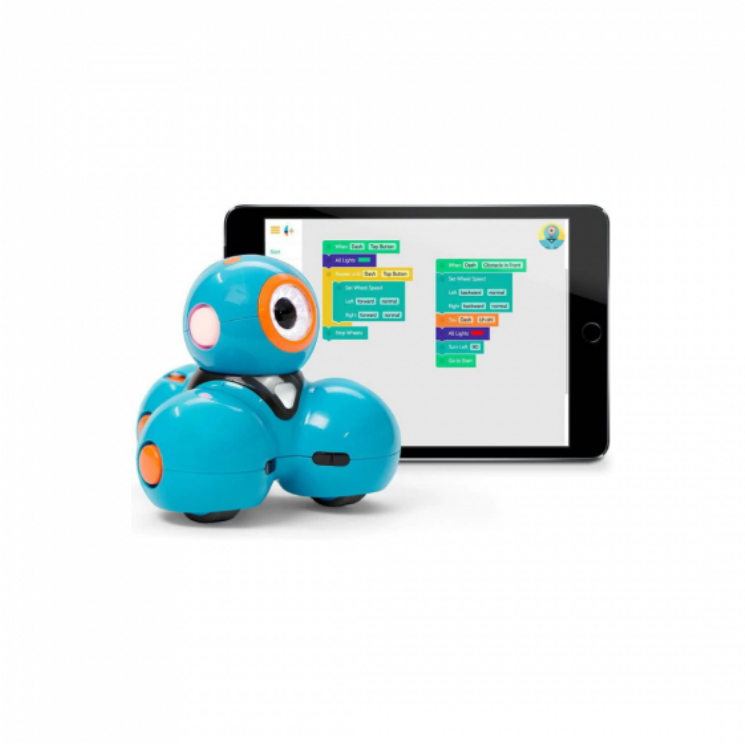

Figure 8.5: Dash robot operated via block-coding [Wor24]

Figure 8.6: LEGO Mindstorms Robotics EV3 UI [Gro24]

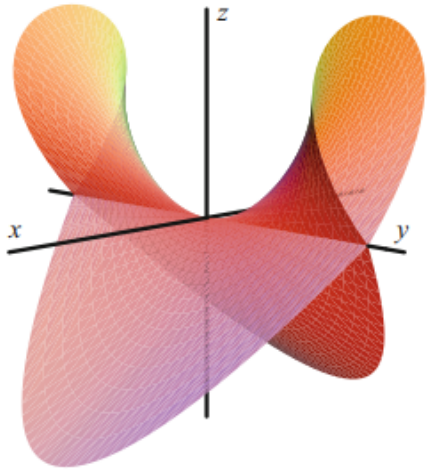

Figure 8.7: $\mathbf{V}(x^2 - y^2z^2 + z^3)$ [COL18]

3D, the computer algebra system and programming language Macaualay2 can be used to compute certain properties of a geometric object. Popularly used for research in algebraic geometry, one can easily determine Grobner bases, quotients, betti numbers and more of polynomial-derived curves and shapes applying just a few core algorithmic rules [COL18].

As academic and educationalist Peter Gray notes,

> "PLAY is a concept that fills our minds with contradictions when we try to think deeply about it. It is serious, yet not serious; trivial, yet profound; and imaginative and spontaneous, yet bound by rules. Play is not real, it takes place in a fantasy world; yet it is about the real world and . . . It is childish, yet it underlies many of the greatest achievements of adults." [Gra17]

Did you think this much math was underneath the toys we play with? It is the hard toys produced by the algorithmic rules of computational geometry which give rise to the games we play in real life (augmented by imagination of course). As Gray, this paper, and its project appendix suggest, "play is a powerful *vehicle* for learning" (author's emphasis) that drives creativity, synthesis, and accomplishment [Gra17].

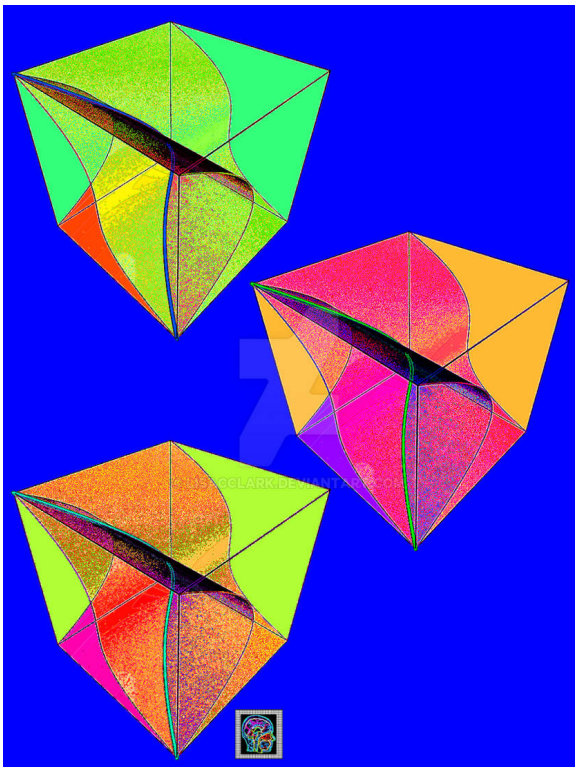

Figure 8.8: Pop-art interpretation of the twisted cubic [CC11]

References

[Ahn12] Song Ho Ahn, *Homogeneous coordinates*, 2012, September 30, 2024.

[Art20] Education Plus Art, *How to draw a car in one point perspective with/ instructions*, 2020, September 30, 2024.

[Ass] Robert McNeel & Associates, *Rhino in architecture, design, and engineering*, October 7, 2024.

[Ath22a] Stephanie Atherton, *Awkward cabin close-up 2*, 2022.

[Ath22b] ———, *Awkward cabin close-up 3*, 2022.

[Ath22c] Stephanie Atherton, *Flying ace color sheet*, 2022.

[Ath22d] Stephanie Atherton, *Front view of cabin*, 2022.

[Ath22e] Stephanie Atherton, *Joonie skateboarding*, 2022.

[Ath22f] Stephanie Atherton, *Vacuum-formed parts*, 2022.

[Ath23a] Stephanie Atherton, *Galilego*, 2023.

[Ath23b] Stephanie Atherton, *Makeblock light + lego heart*, 2023.

[Ath24a] Stephanie Athertom, *Hubble double bubble*, 2024.

[Ath24b] Stephanie Atherton, *Bananaland playset*, 2024.

[Ath24c] ———, *Hubble double bubble 3d printed model*, 2024.

[Ath24d] ———, *Hubble double bubble ghosted view*, 2024.

[Ath24e] ———, *Hubble double bubble pen view*, 2024.

[Ath24f] ———, *Hubble double bubble rhino render*, 2024.

[Ath24g] ———, *Hubble double bubble shaded*, 2024.

[Ath24h] ———, *Hubble double bubble tech view*, 2024.

[Ath24i] ———, *Hubble double bubble wireframe*, 2024.

[Ath24j] ———, *Hubble double bubble x-ray*, 2024.

[Ath24k] Stephanie Atherton, *Kindergarten pinch pot*, 2024.

[Ath24l] Stephanie Atherton, *Lovelace from kawaii+co.de*, 2024.

[Ben21] Cameron Bentley, *How to make a pinch pot*, 2021, September 23, 2024.

[Ber92] Sterling K. Berberian, *Linear algebra*, Oxford University Press, 1992, Sterling K. Berberian was a functional analyst and mathematics professor at The University of Texas Austin for many years. He is also known as for his expository works, including his textbook on linear algebra, as well as those related to hilbert spaces, real analysis, and measure theory. This particular textbook is written to cover topics for a first course in linear algebra, which is typically taken by second year of undergraduate of a mathematics major. Advances in mathematics are made all the time, but linear algebra serves as a mathematical foundation that has been standardized as a course for decades. One of the main objects of study in linear algebra are vectors, which are relevant in pure as well as applied contexts - such as in computer graphics.

[Car23] LEGO Careers, *Just imagine with nico vas*, 2023, October 7, 2024.

[Cas16] Laura Casely, *Instead of toy cars, there were toy chariots*, 2016, October 2, 2024.

[CC11] Lisa Clark C., *Twisted cubic curves - blue*, 2011.

[Chu21] Mike Chua, *Someone 3d printed a transformable transformers optimus prime figure*, 2021, September 24, 2024.

[COL18] David A. Cox, Donal O'Shea, and John B. Little, *Ideals, varieties, and algorthms*, Springer, 2018, David A. Cox is a Professor of Mathematics Emeritus at Amherst College, known for his work in algebraic geometry - including computational algebraic geometry. Donal O'Shea is a Professor of Mathematics and Statistics Emeritus at Mount Holyoke College, known for several bestselling books such as "Ideals, Varieties, and Algorithms: An introduction to Computational Algebraic Geometry" and "Using Algebraic Geometry". John B. Little is a Distinguished Professor of Science Emeritus at College of the Holy Cross with research interests in computational methods in algebraic geometry and commutative algebra and upon retirement, the history of mathematics. The preface of the book clearly articulates that the text is intended for an undergraduate class (or for self-study) with background in a first course in linear algebra and proofs. Following, the book primarily takes a computational or "algorithmic" approach to introductory algebraic geometry in focusing on the relation between ideals and varieties for upper division undergraduates. The text also expands upon applications of algebraic geometry, including to CAD modeling systems. Computer algebra systems (as well as CAD systems) began to appear in the 1960s, with the initial release of Macaulay2 (a CAS specific to algebraic geometry and commutative algebra) in 1993. The use of technology in education has also become more prevalent in the 21st century, inspiring computational components in companion to tradtional learning.

[Com13] Blender Community, *Artist without imagination?*, 2013, October 7, 2024.

[Com20] Wikipedia Commons, *Cubic bezier curve with four control points*, 2020, September 24, 2024.

[EGSS02] David Eisenbud, Daniel Grayson, Michael Stillman, and Bernd Sturmfels, *Computations in algebraic geometry with macaualay2*, Springer, 2002.

[eki18] ekimroyrp, *Palette*, 2018.

[Fou24] The LEGO Foundation, *Introducing lego braille bricks*, 2024, October 7, 2024.

[FP24] Fisher-Price, *Little people barbie toy car with music sounds and 2 figures, convertible, toddler toys*, 2024, September 24, 2024.

[Gao20] Gao, *Matrices in computer graphics*, 2020, September 30, 2024.

[Goe18] Liesl Goecker, *The makers of the best kids' toys are the ones going out of business*, 2018, October 2, 2024.

[Gra17] Peter Gray, *What exactly is play and why is it such a powerful tool for learning?*, Top Lang Disorders (2017), 217 – 228.

[Gro24] LEGO Group, *Retired lego® mindstorms® ev3 home edition software for pc and mac*, 2024.

[HL23] Jos Huxley and Joan Lawrence, *The astm f963 toy safety standard: What you need to know, part 1 - introduction mechanical and physical requirements*, Safety Education Course Lecture Slides, 2023.

[ima20] imagilabs, *The imagilabs app and imagicharm*, 2020.

[Iss13] Rajaa Issa, *Essential mathematics for computational design, third edition*, Robert McNeel & Associates, 2013.

[Kag21] Peter K. Kagey, *Stacking lego bricks*, 2021, October 7, 2024.

[Lib23] Boston Public Library, *Introduction to design software*, 2023, September 24, 2024.

[lim13] limsup, *From euler characteristics to cohomology (1)*, 2013, September 23, 2024.

[LMN21] Aram Lindroth, Alanna Manfredini, and Nathan Nguyen, *Holonomy of combinatorial surfaces*, Duke DOmath Project Notes, July 2021.

[Mag29] Rene Magritte, *The treachery of images*, 1929, October 16, 2024.

[Mar19] Earl Mark, *Computer aided architectural design*, 2019, September 20, 2024.

[Muf] My Sweet Muffin, *Le toy van wooden donuts*, October 28, 2024.

[NAS24] NASA, *Hubble space telescope*, 2024, October 29, 2024.

[Nie12] A.H. Niemi, *Bezier curves*, Aaalto University course notes, 2012.

[oAD24] Otis College of Art & Design, *Toy design student work*, 2024, September 24, 2024.

[Pat13] Nicholas M. Patrikalakis, *Computational geometry*, September 23, 2024, 2013.

[Pla] Drew Plakos, *Infant and juvenile products, 2024.*, Drew Plakos has been a professor teaching natural science courses in the toy design department for over 20 years, and has worked in various roles within the toy industry since the 1970s. He created these lecture slides for the Human Factors in Toy Design course, as taken by toy design students at Otis College of Art & Design. The slides pertain to the human factors of juveniles

specifically, under the constraints set by a US safety association, and provide an overview of juvenile safety considerations for product/toy designers. Indeed, toy designers need to meet certain safety specifications when designing for children, and the accessibility of 3d programs allow them to do so. However, safety standards can be updated yearly and new ones placed into effect.

[Pla22a] _____, *Hard good methods of production/plastics (part 1)*, Methods and Materials Lecture Notes, 2022.

[Pla22b] _____, *Hard good methods of production/plastics (part 2)*, Methods and Materials Lecture Notes, 2022.

[PMC09] Nicholas M. Patrikalakis, Takashi Maekawa, and Wonjoon Cho, *Shape interrogation for computer aided design and manufacturing*, Hyperbook Edition, 2009, All authors have been professors or research associates in the Dept. of Engineering at MIT, with the preface of the hyperbook clearly articulating the intended audience as graduate engineering students or professionals in industry. Fittingly, the text has a very applied focus towards industry regarding shape interrogation for design engineers/students. Specifically, the source provides a description of the Bernstein polynomials, bezier curves, and b-splines useful for our purpose. While primarily accurate to this day, technology is constantly developing and there could be new, relevant methods in computational geometry not taken into account in the hyperbook.

[Roy] Sushmita Roy, *Bezier curve with convex hull*, October 28, 2024.

[TIE13] TIE, *History of play*, Brochure, 2013.

[Wor24] Wonder Workshop, *Dash robot*, 2024.

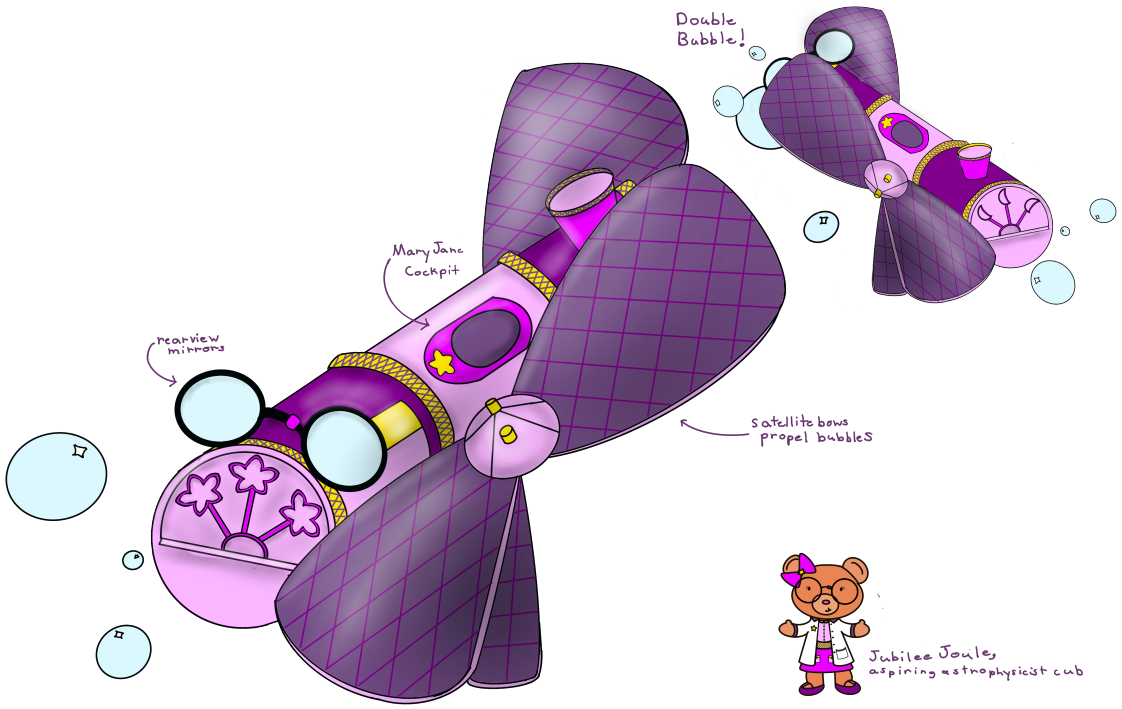

Figure 8.9: Vector graphics rendering of "Hubble Double Bubble Telescope" exploration toy for preschool girls [Ath24a]

Project Appendix

Concurrently with the writing of this paper, I was enrolled in my penultimate studio "Toy Design IV: Vehicle Design". While the originally proposed design concept consisted of a Kpop-inspired concert van with block-coding algorithmic music capabilities, it was vetoed under the belief that "8 year-old can't code" [1]. Nevertheless, I was still delighted to pursue the agreed-upon vehicle of what has become the Hubble Double Bubble Telescope, obviously inspired by the Hubble Space research Telescope that floats about in low-earth orbit [NAS24]!

Rhino.Python Code[2]

As we have learned, the shapes of the Hubble Double Bubble Telescope were all defined by parametric curves. Computationally, we would draw a curve as follows.

[1] Factually, all the coding toys referenced in the paper are indeed age-graded at 8+ or 6+. The elementary schoolers on the robotics team at the elementary school I work at are also all of at least upper-elementary school age and the girls *adore* Super Shy by New Jeans.

[2] Official code samples courtesy of https://developer.rhino3d.com/samples/

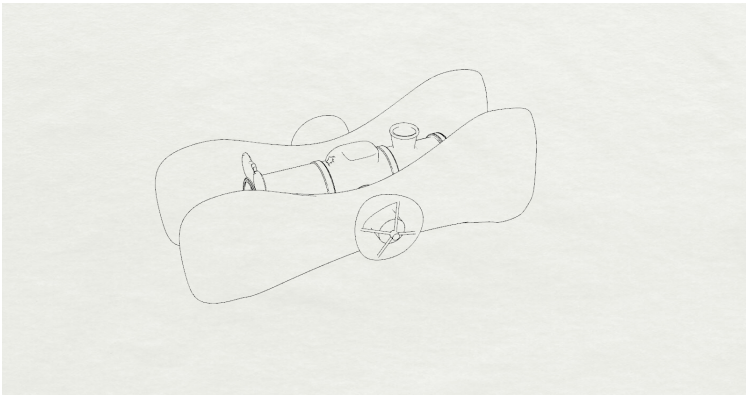

Figure 8.10: Pen-style 3D model view [Ath24e]

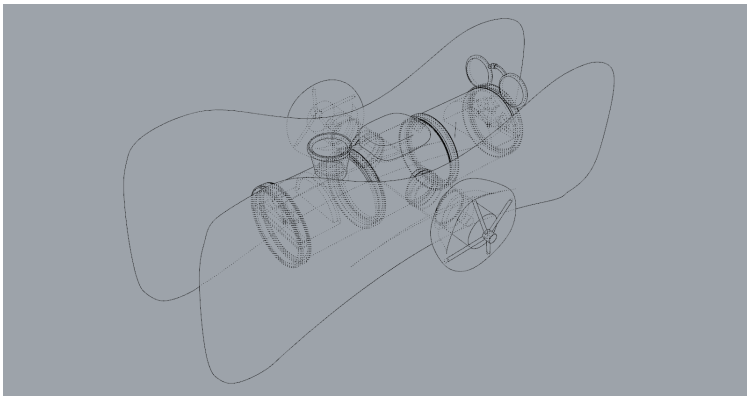

Figure 8.11: Technical-style 3D model view [Ath24h]

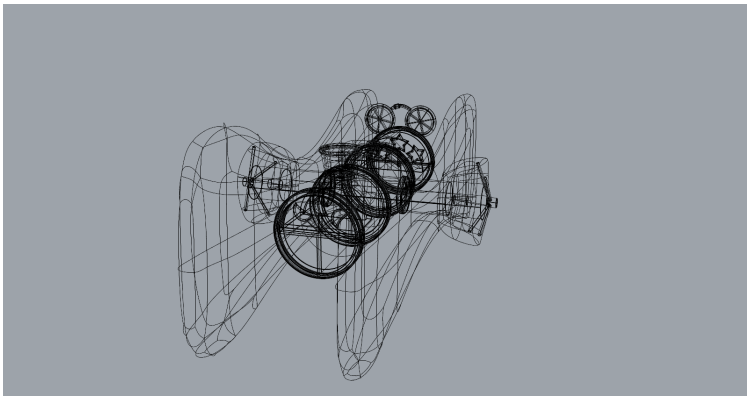

Figure 8.12: Wireframe 3D model view [Ath24i]

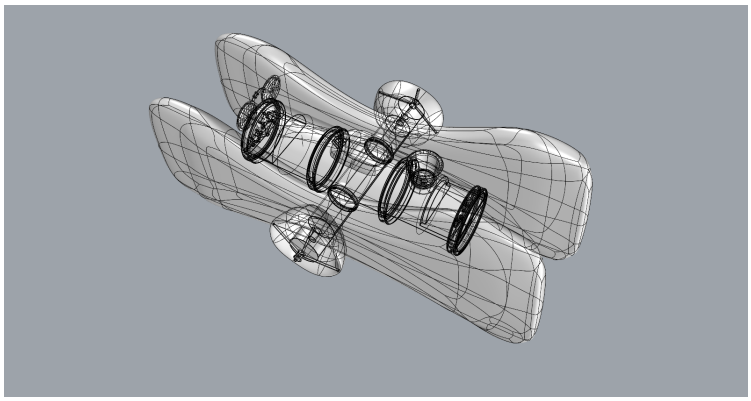

Figure 8.13: X-ray style 3D model view [Ath24j]

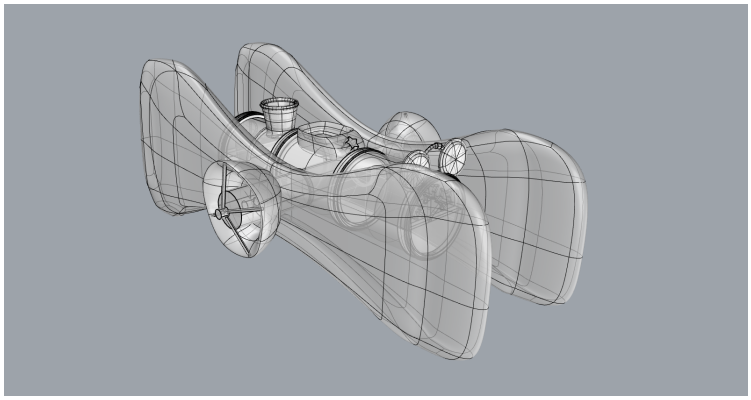

Figure 8.14: Ghosted 3D model view [Ath24d]

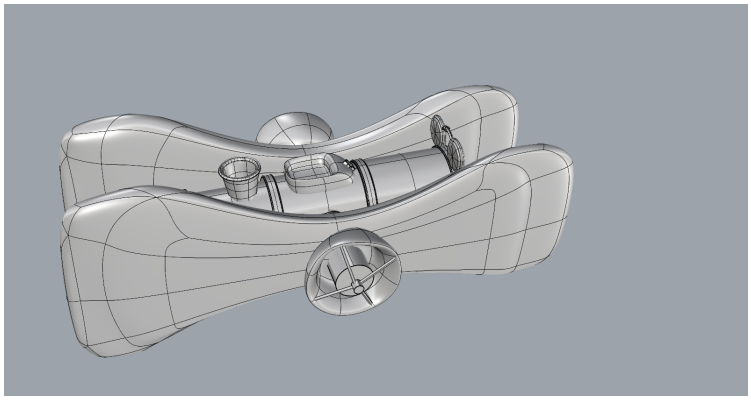

Figure 8.15: Shaded 3D model view [Ath24g]

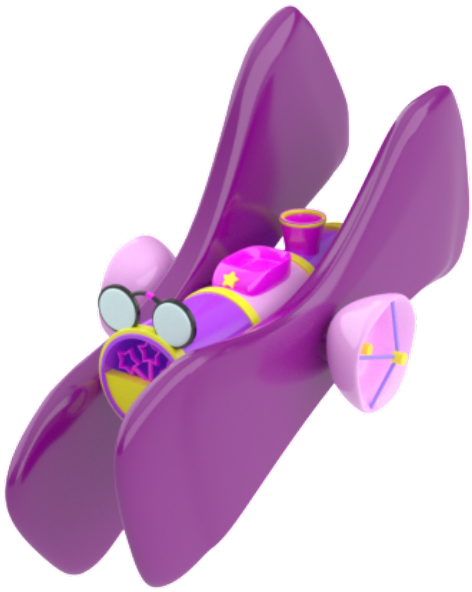

Figure 8.16: Final rendering of "Hubble Double Bubble Telescope", created using the NURBS geometry of Rhino 3D [Ath24f]

Figure 8.17: NURBS geometry must be converted to polygon meshes in preparation for 3D printing. Here is the tangible 3D printed and purple-painted model of the Hubble Double Bubble Telescope [Ath24c].

```python
#DRAWING A PARAMETRIC CURVE in Rhino.Python
import rhinoscriptsyntax as rs
import math

# Something really interesting about this script is
# that we are passing a function as a parameter
def DrawParametricCurve(parametric_equation):
    "Create a interpolated curve based on a parametric equation."
    # Get the minimum parameter
    t0 = rs.GetReal("Minimum t value", 0.0)
    if( t0==None ): return

    # Get the maximum parameter
    t1 = rs.GetReal("Maximum t value", 1.0)
    if( t1==None ): return

    # Get the number of sampling points to interpolate through
    count = rs.GetInteger("Number of points", 50, 2)
    if count<1: return

    arrPoints = list()
    #Get the first point
    point = parametric_equation(t0)
    arrPoints.append(point)

    #Get the rest of the points
    for x in range(1,count-2):
        t = (1.0-(x/count))*t0 + (x/count)*t1
        point = parametric_equation(t)
        arrPoints.append(point)

    #Get the last point
    point = parametric_equation(t1)
    arrPoints.append(point)

    #Add the curve
    rs.AddInterpCurve(arrPoints)


#Customizable function that solves a parametric equation
def __CalculatePoint(t):
    x = (4*(1-t)+1*t ) * math.sin(3*6.2832*t)
    y = (4*(1-t)+1*t ) * math.cos(3*6.2832*t)
    z = 5*t
    return x,y,z
```

```
if( __name__ == "__main__" ):
    #Call the function passing another function as a parameter
    DrawParametricCurve(__CalculatePoint)
```

We can also extract the length of the drawn curve.

```
#CALCULATING LENGTH OF CURVE in Rhino.Python
import rhinoscriptsyntax as rs

def CurveLength():
    "Calculate the length of one or more curves"
    # Get the curve objects
    arrObjects = rs.GetObjects("Select Objects", rs.filter.curve, True, True)
    if( arrObjects==None ): return
    rs.UnselectObjects(arrObjects)

    length = 0.0
    count  = 0
    for object in arrObjects:
        if rs.IsCurve(object):
            #Get the curve length
            length += rs.CurveLength(object)
            count += 1

    if (count>0):
        print("Curves selected:", count, " Total Length:", length)

if( __name__ == "__main__" ):
    CurveLength()
```

Two M2 Sessions

Algebraic geometry is a branch of mathematics which focuses on solutions of polynomial equations that can interpreted/visualized geometrically (often as curves). The subject can made more accessible through an algorithmic approach, "which has two principle aims:

- developing new tools for research within mathematics,
- and providing new tools for modeling and solving problems in science and engineering" (and design!, as we have seen) [EGSS02].

A third aim has developed in using Macaulay2 (abbreviated as M2) in the teaching of algebraic geometry and commutative algebra to advanced undergraduate and beginning graduate students. An M2 session takes place in a terminal, the web version of which can be experimented with here:

$$https://www.unimelb-macaulay2.cloud.edu.au,$$

and relies on inputs **i:** of mathematical objects/operations and outputs **o:** revealing information underlying our geometric entities. We have seen how one can implement a drawn curve via

Rhino.Python, and recursively, one can also compute properties of a curve in affine three-space. First, we define our algebraic and geometric objects.

```
#Polynomial Ring over rational coffeicients in 3 variables
i1 : R = QQ[x,y,z]
o1 = R
o1 : PolynomialRing
#Define ideal of our curve, which is an iaffine variety
i2 : curve = ideal( x^4-y^5, x^3-y^7 )
o2: ideal(-y^5+x^4, -y^7+x^3)
o2 : Ideal of R
```

We can then compute the Grobner basis (command "gb") of the ideal of our curve. A Grobner basis is a very "nice choice" of basis as it allows us to easily deduce further properties of an ideal once computed.

```
i3 : gens gb curve
o3 = (y5-x4 x4y2-x3 x8-x3y3)
#Find the dimension of the curve (see that we have a 1-dimensional curve)
i4 : dim curve
o4 = 1
#Find the degree of the curve
i5 : degree curve
o5 = 28
```

In Rhino 3D, there are many situations in which we union or intersect curves with surfaces in the process of creating new forms.

```
#Define the ideal of a surface
i6 : surface = ideal( x^5 + y^5 + z^5 - 1)
o6 = ideal( x^5 + y^5 + z^5 - 1)
o6 : Ideal of R
i7 : theirunion = intersect(curve,surface)
o7 = ideal (x^5y^5 + y^10 + y^5z^5 - x^9 -x^4y^5 - x^4z^5 - y^5x^4, · · ·
o7 : Ideal of R
i8 : curve*surface == theirunion
#The union of a curve and a surface is the intersection of their ideals
o8 = true
#The intersection of a curve and a surface is the sum of their ideals.
#We get a finite set of 140 points
i9 : ourpoints = curve + surface
o10 : ideal (-y^5+x^4, -y^7+x^3, x^5+y^5-z^5-1)
o10 : Ideal of R
i11 : dim ourpoints
o12 = 0
i13 : degree ourpoints
o13 = 140
```

We can also explore properties of more complicated shapes - like the twisted cubic, which we now consider in projective three-space $\mathbb{P}^3$ (the "space" artists draw in, you may recall).

```
i2 : ringP3 = QQ[x_0..x_3]
o2 = ringP3
o2 : PolynomialRing
```

Specifically, the twisted cubic is the image of the map $\mathbb{P}^1 \to \mathbb{P}^3, (s,t) \mapsto (s^3, s^2t, st^2, t^3)$.

```
i3 : ringP1 = QQ[s,t]
o3 = ringP1
o3 : PolynomialRing
i4 : cubicMap = map(ringP1,ringP3,{s^3, s^2*t, s*t^2, t^3})
o4 = map(ringP1,ringP3,{s^3, s^2t, st^2, t^3})
o4 : RingMap
```

We can find the mononomial ideal of its parameterization $r(t) = (t, t^2, t^3)$ by

```
i5 : idealCubic = monomialCurveIdeal(ringP3,{1,2,3})
o5 = ideal (x^2_2 - x_1x_3, x_1x_2 - x_0x_3, x^2_1 - x_0x_2)
o5 : Ideal of ringP3
```

and confirm the degree of our cubic curve.

```
i6 : degree idealCubic
o6 = 3
```

The ideal of the twisted cubic is actually generated by minors of the matrix $\begin{bmatrix} x_0 & x_1 & x_2 \\ x_1 & x_2 & x_3 \end{bmatrix}$, as shown by

```
i7 : M = matrix{{x_0,x_1,x_2},{x_1,x_2,x_3}}
o8 : Matrix
i9:  idealCubic2 = minors(2,M)
o9 : ideal(-x^2_1 +x_0x_2, -x_1x_2 +x_0x_3, -x^2_2 +x_1x_3)
o9 : ideal of ringP3
i10 : gens idealCubic
o10 : (x^2_2 - x_1x_3  x_1x_2 - x_0x_3 x^2_1 - x_0x_2)
o10 : Matrix
i11 : 0 == (gens idealCubic)%(gens idealCubic2)
o11 : = true
```

We have proven that the two ideals are equal [EGSS02].

The computational power of Macualay2 allows us to accessibly uncover relevant information regards our curves and surfaces (e.g. Grobner bases, dimensions, degrees, much more!) and prove properties of the shapes under consideration by boolean logic. The knowledge we gain via computation makes it possible to design algorithms in Rhino.Python for use in computational design practice, and further, to directly model youth products like the "Hubble Double Bubble Telescope" in CAD-based 3D modeling programs.