Design and Implementation of Lathe-based and Cross-sectional Organic 3D Modeling and Animation via Sketching and Graphing

REDACTED

Abstract

Traditional polygonal modeling practices often pose trouble for 3D modeling novices or those who prefer 2D workflows. The purpose of this project is to innovate on previous sketch-based modeling techniques in building standalone sketch-based 3D modeling software to make organic 3D modeling/animation more accessible to beginners and 2D-centric artists. The program developed is primarily composed of a canvas/graph and an attached 3D model viewer updated in real time as the user draws. Two different algorithms are employed to generate modular mesh components that are assembled in the 3D viewer to create full models, including: lathing, where a 2D curve is rotated around an axis that is variably offset by other user-defined functions; and side-view cross section inflation, where raycasting is used to sweep out a mesh defined by a central and bounding curve. Each method also allows the addition of a modulatable secondary cross section to further define a shape. By interpolating between user-inputted curves, functions, and variables, this system presents an interesting form of animation that avoids complicated bone-based rigs altogether, by instead treating the sketches themselves as keyframes. The application has produced various compelling 3D creatures, solving the issues of limited topological complexity often seen in sketch-based modeling techniques through modularity. Furthermore, the non-destructive use of canvas-based input allows for unrestrained animation using variable levels of detail—demonstrated through the system's creation of a few short but flexible animations. The successes of this project suggest possibilities for increased 2D support in user-level, organic 3D modeling applications.

Introduction

3D modeling is a highly desirable skill in film, animation, game-development, and more—however, it has a very steep learning curve that intimidates newcomers, often resulting in many to quit due to its complex, time-consuming nature or from simple dislike of 3D workflows.

Today, polygonal 3D modeling, such as that seen in popular applications like Blender or AutoDesk Maya, dominates artistic modeling. While quite flexible when mastered, these types of programs are notoriously confusing for beginners. For example, users must manage vertices (3D points), edges, faces (polygons), surface normals, general mesh topology, animation rigging, and many other deep concepts. Unfortunately, the large amount of time one must invest upfront to develop these skills often prevents those who simply want to start creating from ever getting started.

Sketch-based modeling approaches these types of problems by generating 3D models from 2D input. This research area was popularized by the release of Teddy: a sketching interface for 3D

freeform design (Igarashi et al, 1999), which introduced a promising, small-scale program wherein drawn curves were "inflated" into models. Later on, a method of chaining together multiple cross sections in a 3D space (Hughes) came about. Unfortunately, although this field has seen great promise through the various techniques and demos created, few of these methodologies are seen in user-level applications. Today, the most 2D-to-3D features are typically seen with computer aided design (CAD) programs, which include features such as geometric push/pull-ing (SketchUp), surfaces of revolution, and lofts (AutoCAD). However, these are generally geared towards more precision-based, geometric modeling, with applications in digital fabrication or architecture—and are thus less optimal for organic models (those consisting primarily of natural-seeming curves) and artistic, freehand workflows.

This project sought to create standalone software to make organic 3D modeling/animation quick and accessible for 2D-artists—particularly through the use of cross-sectional and lathe-based modeling techniques via sketching and graphing.

Specifically, the program was designed to: maintain smooth, high performance to ensure instant feedback and viewing for artists; produce topologically desirable models (reliant on quads [4-vertex polygons]); allow flexibility by allowing access and animation for notable curves and variables used; investigate opportunities in within a functional, fully sketch-based system; enable intermediate users to rapidly model creatures of at least the complexity of a Charmander™ or equivalent creature; provide a solid user interface and input-scheme around the core generation techniques.

Materials

The implementation for this project was programmed in C++ (as to maximize speed). Outside dependencies used included: Open Graphics Library (OpenGL), OpenGL Extension Wrangler Library (Glew), Graphics Library Framework (GLFW), OpenGL Mathematics (glm), Boost, and Dear ImGui.

Methods

This program used sketch-based methods to generate modular "mesh components" to be assembled by the user to form more complex 3D creatures and objects.

The primary components that composed this project were its renderer, model generator, animation system, and user interface/input.

Renderer

The application used simple 2D and 3D renderers–both of which were written using the Glew, GLFW, and glm libraries in addition to GLSL (OpenGL's shader language). The 3D renderer displayed 3D-objects, shading them via the Phong shading model (which produces quick and efficient smooth shading by blending vertex normals); the 2D renderer drew the user's plot/canvas using flat-coloured lines and polygons.

Model Generation

The model-generation system worked primarily based on user-drawn curves (poly-lines) and user-drawn functions (piecewise). The application supported the following two sketch-based modeling methods: lathe-based and side-view cross-sectional. Lathing is the process of creating a 3D shape by rotating a curve around an axis—similar to Surfaces of Revolution, but with support for doubling back. In this program, 'lathes' were generated by sampling a sketched curve at regular intervals in accordance with the currently-set *sample-length*. Rings of points were then formed by rotating each (x, y) point in 3D space around the x-axis, treating y as the radius. Subsequently, the rings were manipulated by any auxiliary functions (such as through translations of y-offset/z-offset [wherein the rings were also rotated {around the z-axis/y-axis} by the offset-function's slope multiplied by the *lean-scalar* value]) according to their x-value. Finally, the rings were connected in sequence to create a polygonal mesh.

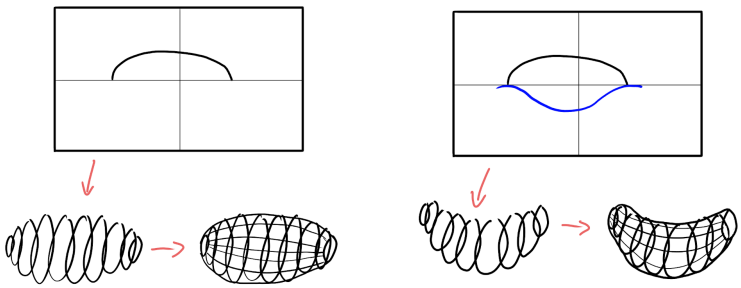

Figure 1. "Simple Lathe Generation Diagram"

Side-view cross-sectional models employed a similar process—but created rings differently—by instead using a bounding curve and central curve (which was either user-drawn or automatically generated [see Mesh Generation - Central Curve Auto-generation]). Along a fixed step (*sample-length*), the central curve shot rays in either direction perpendicular to the tangent line at that point. If both rays were caught by the bounding curve, a ring was created with its size and rotation determined by the resulting line segment's length and angle.

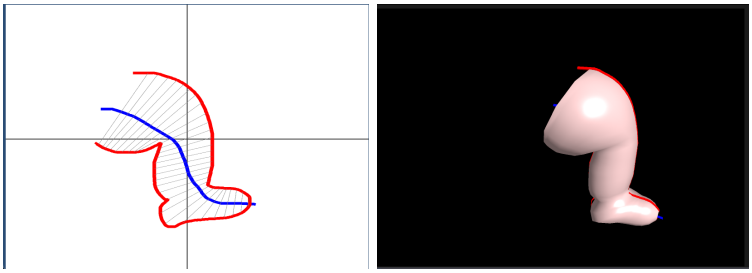

Figure 2. "Example of Cross-sectional Model Generation"

Both of the methods described also allowed the application of a secondary cross section in order to further define a shape. For this, another user-drawn curve replaced the standard ring shape used during the generation processes (after being sampled to the appropriate vertex count as if it were an enclosed polygon).

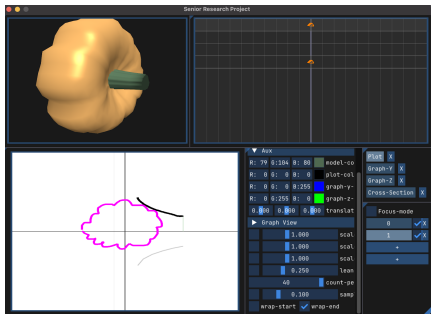

Figure 3. “Secondary Cross Sections Applied to Lathes to Create Pumpkin”

Animation System

Unlike in skeletal animation (using complicated rigs and "bones" to transform existing models), this program animated by generating a wholly new model every time-step. The animation system was built upon the concept of interpolating values used in the model-generation process between keyframes. These interpolated values could be both numbers (ex. sample-length, lean-scalar, etc.) or curves (poly-lines/piecewise-functions). To interpolate between two curves, both were re-parameterized in terms of the proportion of arc-length to the total length at a given point: on the scale of 0 to 1. Following this, the first curve’s points were linearly interpolated to the second’s depending on their normalized values.

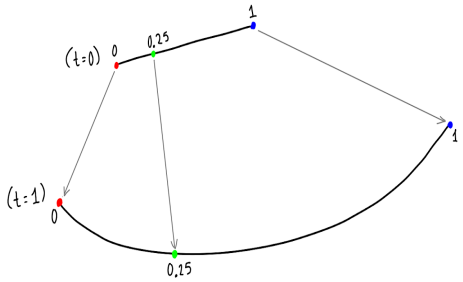

Figure 4. “Curve Interpolation Diagram”

Between keyframes, a t-value ($0 \leq t \leq 1$) was used to interpolate between the values stored within each keyframe. By default, the change was linear, but an easing function could be applied to remap t (these were either preset [linear, sinusoidal, elastic] or user-drawn). Finally, by updating the model each frame during which an animation was playing based on the new, interpolated values, this software’s curve-based generation allowed for what is essentially “morphing” between models.

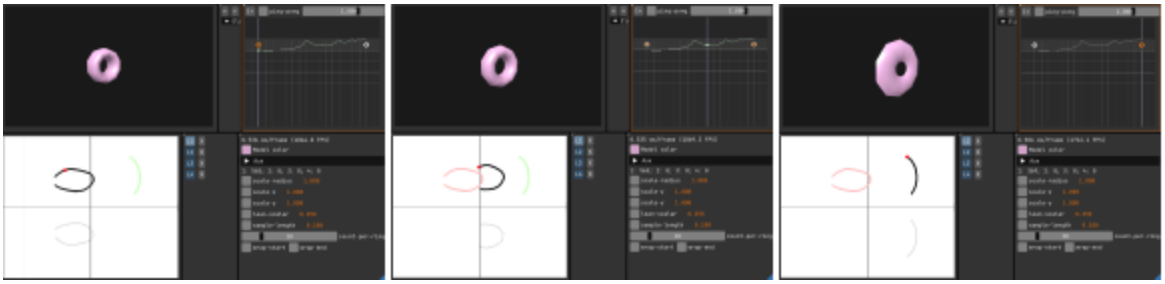

Figure 5. “Animating Between Two Lathes Using a Custom Easing Function”

Model Generation - Central Curve Auto-generation

Like animations, the automatic generation of central curves for cross-sectional models was also based on the interpolation of poly-lines. Since the desired central axis tended to be a curve in-between and through the bounds (see Figure 2), such was computed automatically in the absence of a manually-defined central curve. To do so, a ‘folding point’ was found along the bound line—approximated as the half-way mark of the bounding line. Said line was then bisected over the ‘folding point’, with each side being interpolated halfway ($t = 0.5$).

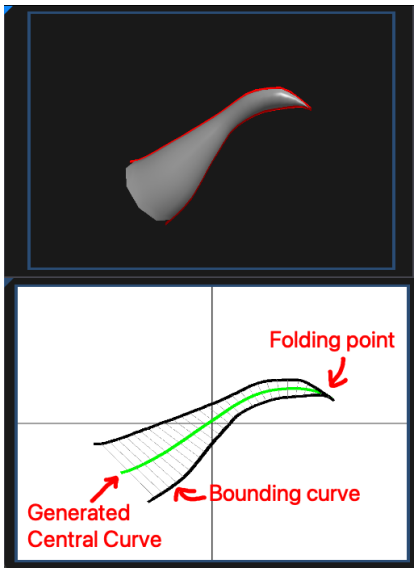

Figure 6. “Folding point demonstration”

Model Generation - Multiple Cross Sections

Multiple cross sections were applied to singular mesh components by treating their primary/central curve as a sort of timeline structure (where arc length is used instead of time), on which cross sections could be “keyed in” (as seen by the orange dots on Figure 7). Herein, each arc length step during the generation process was preceded by a sampling of the interpolated cross section at that point (linearly interpolating between these keyframe-like cross section-indicating points).

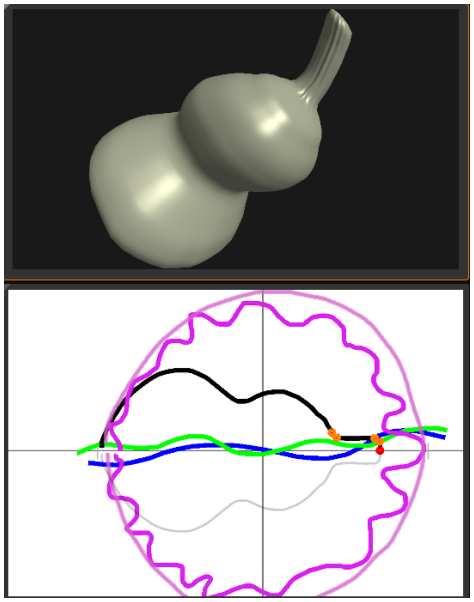

Figure 7. “Gourd with Textured Stem from Use of Multiple Cross Sections”

User Interface/Input

The application was brought together by its user interface and input system. The UI was built using Dear ImGui, an immediate-mode C++ GUI library. The model view, canvas, and timeline were rendered onto FrameBuffers using OpenGL, which allowed for them to be treated as displayable textures by ImGui. The rest of the UI consisted of buttons and the like to switch between models and curve-type in addition to changing parameters and such. Mouse input was used for purposes such as drawing (on the canvas), changing the model viewing angle, and manipulating the timeline/keyframes. Additionally, by leveraging 3D raycasting (wherein the position and normal [surface angle] of a ray’s intersection with models in the 3D scene is found), model selection and transformation via mouse pressing/dragging was accomplished. Keyboard input was leveraged to speed up the program workflow, with common actions (creating new models, clearing the canvas, and transforming [translating, scaling, & rotating via “G”, “S”, & “R”, respectively {like in Blender}]).

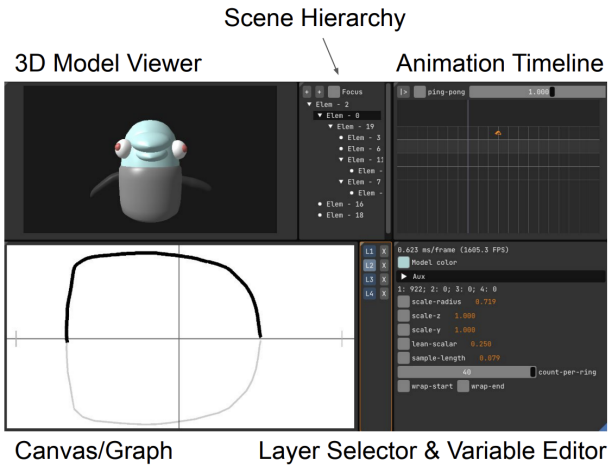

Figure 8. “Annotated Application Overview”

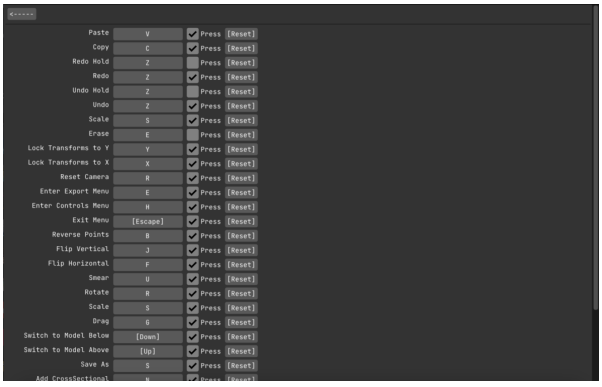

Figure 9. “Control Customization Menu”

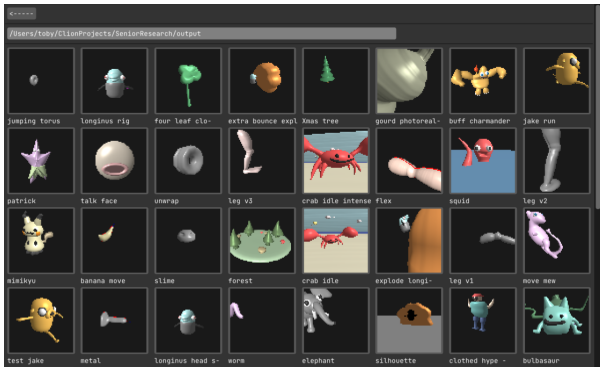

Figure 10. “File Loading Menu”

Additional UI and quality of life features included keyboard-control customization and project serialization (using text archives from boost::serialization)–which went a long way in increasing the software’s usability.

Results

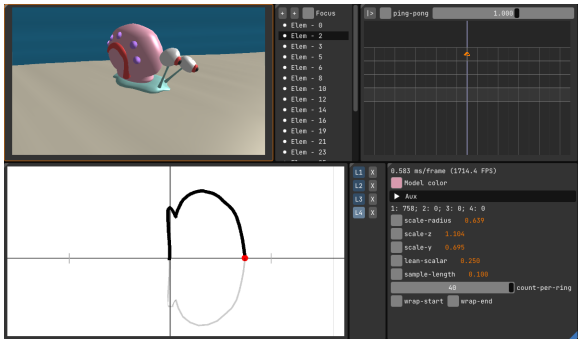

Figure 11. “Gary the Snail™”

Figure 11 presents a snail composed solely of lathes.

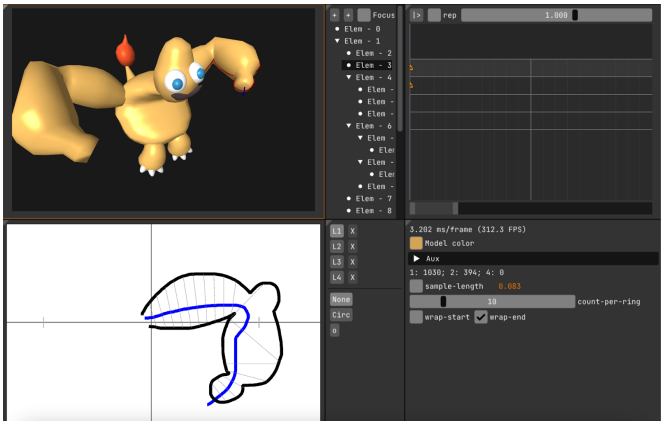

Figure 12. “Able-bodied Charmander™”

Figure 12 shows a Charmander™ created with a combination of lathes and side-view cross-sectional mesh components.

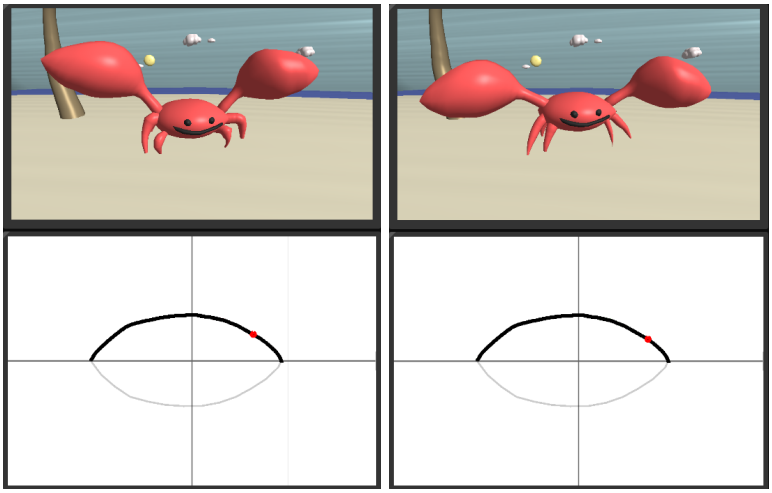

Figure 13. “Crab Rave v2”

Figure 13 demonstrates a crab jubilantly bouncing up and down. The scene consists of 21 mesh components, and yields frame rates of about 300 when 3D raycasting is active (for 3D drag-and-drop).

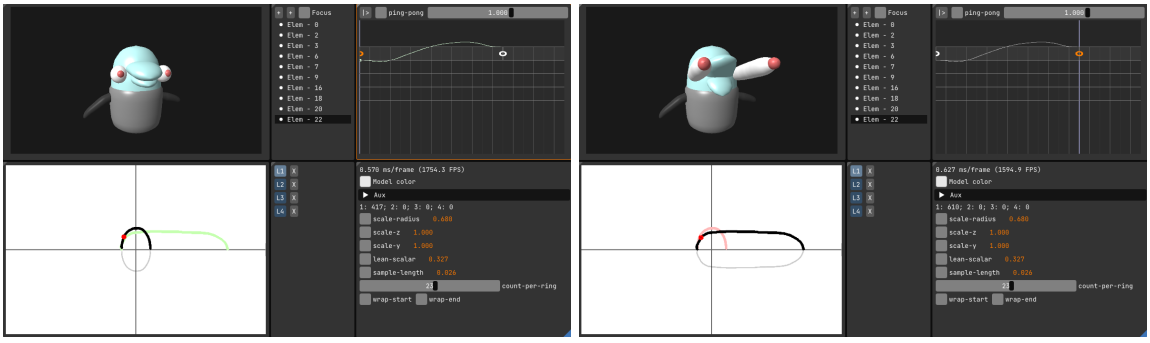

Figure 14. “Aghast Porpoise-like Figure”

Using only two keyframes and a user-drawn, bouncy easing function, the porpoise-like figure in this animated clip shifts from a calm to aghast state in a manner reminiscent of classical 2D squash and stretch.

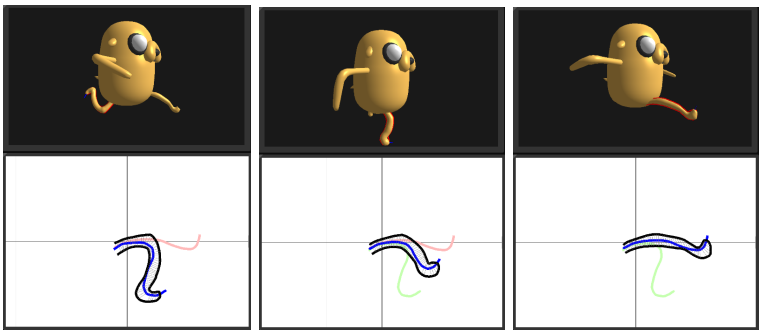

Figure 15. “2-Keyframe Running Animation using Interpolation”

Figure 15 shows three snapshots from a run cycle animation at points $t = 0$, $t = 0.5$, and $t = 1$. Keyframes were only used on frames 1 and 3, with the center frame being generated using only the interpolated data of its neighbor frames.

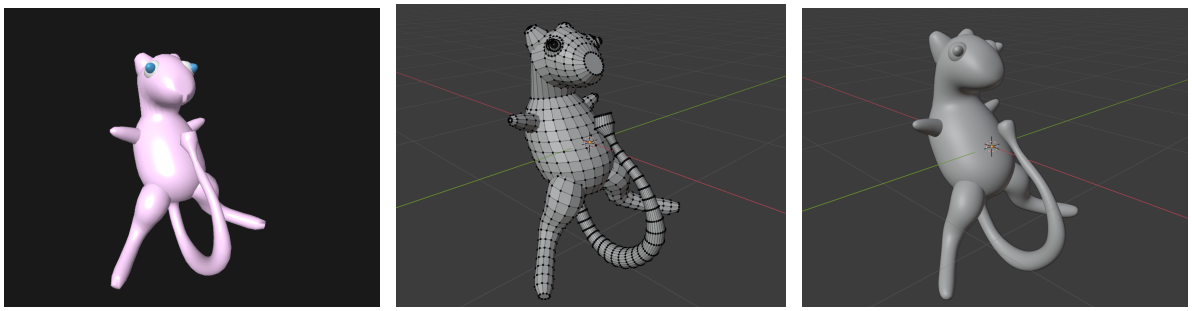

Figure 16 a, b, & c: "Suspiciously Rat-like Mew™"

Figure 16 shows a model in the program and loaded into Blender (with and without a subdivision modifier)—demonstrating the exporting capabilities and desirable, quad-based meshes created.

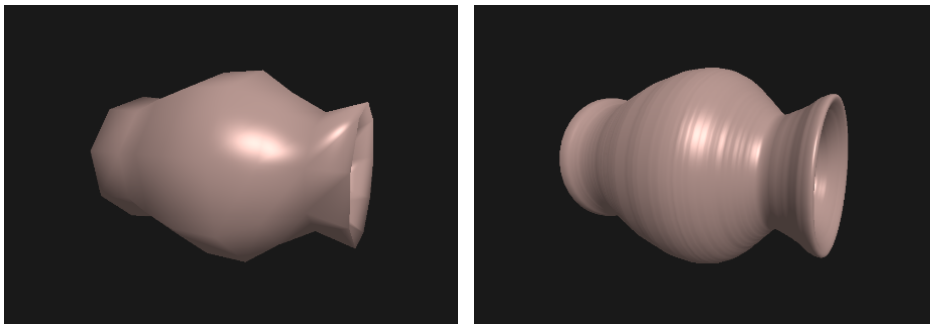

Figure 17 a & b: “Pot of Arbitrary Detail”

Figure 17 displays two versions of one pot-shaped lathe, with only the *sample-length* and *points-per-ring* values modified.

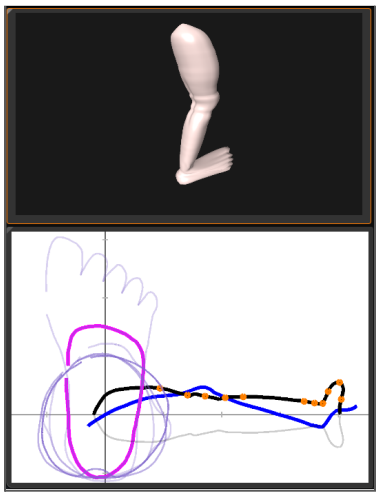

Figure 18. “Anatomically Realistic-ish Leg via Lathe of Multiple Cross Sections”

Figure 18 shows off the ability to bind multiple cross sections to a singular mesh component. It uses cross sections consisting of differing ovals/circles (some with indents or extrusions) as well as a foot shape to create the illusion of joints, tendons, and other general anatomical features.

Discussion and Conclusions

Having produced various compelling 3D models and short animations, the implementation of this project has shown promising results for the incorporation of sketch-based techniques into user-level organic modeling applications. By approaching modeling as a task based on a canvas, this project has demonstrated that common, pre-existing 2D art skills such as drawing and manipulating curves can be used to great effect in avoiding the complexities and 3D workflows of polygonal modeling and rig-based animation.

Generation Methods

While previous sketch-based demo programs often make the 3D view and canvas one and the same, this program separated them. Although this fact might make it appear as though the tie between user input and the 3D model would be less obvious, the meshes' real-time responses to input changes helped considerably to quickly teach how each factor relates. As such, it is quite possible that learning sketch-based modeling may provide a better experience for users, as they could discover things themselves by simply opening the program and scribbling on the canvas, switching layers, dragging a couple sliders, and hitting undo when unhappy with changes—a far cry from traditional polygonal modeling software, wherein users can easily spend hours getting lost in unfamiliar 3D spaces and pages upon pages of UI while merely trying to figure out the basics of manipulating and creating basic 3D forms.

Moreover, the addition of multiple cross sections, while slightly more challenging to intuitively comprehend, combines some of the flexibility and advantages of cross-section-to-cross-section techniques with the simplicity of side-view and lathe-based methods. In future extensions, the idea of parameterizing other variables along arc length, such as was done with cross sections, could be explored in more depth—and could quite easily be used for colour gradients, offsets/rotations, or other variables that depend on run-length instead of x-location or raycasting.

Topology

Although models within the program are represented using triangles (for performance reasons), they can easily be converted into quad-based meshes—and are, when exported. Quads make up all polygons except the two optional wrapping faces, which are N-gons of vertex counts specified by the currently-set 'points-per-ring' parameter.

The benefit of generating models primarily of quads is the increased compatibility with polygonal modifiers in outside programs, such as Blender's subdivision surface modifier, which smooths out a given mesh (see Figure 16).

Even while working with significant limits to topological complexity (lathes can not move parametrically; the surfaces of side-view cross-sectionals can not fold back on themselves; neither support the addition of holes or branches), relatively complex models (such as in Figures 11, 12, and 16a) were still able to be achieved using the hierarchical composability of the modular components—demonstrating this as valid approach to compensate for the shortcomings seen in many algorithmic sketch-based methods.

In this implementation, mesh components are treated as their own objects in some respect. This results in the final, assembled model consisting of many overlapping meshes. While working fine for bubbly, cartoonish styles, contiguous models would be preferable in many other cases. For this reason, a useful extension could remesh intersecting regions to form a single, fully-connected model, which would allow for smoother transitions along component boundaries while eliminating the possibility of clipping artifacts seen with intersecting geometry at lower vertex counts.

Non-Destructibility, Animation, & Level of Detail

3D modeling is often a destructive process—in Blender, subdividing a portion of a mesh may leave a model unworkable; in Teddy, sketch-based techniques are used to change a mesh itself—but it does not necessarily have to be that way. This project has shown some of the benefits of using non-destructive methods by moving the destructibility out of the model and down to the level of the input (here, the curves/functions and variables). With this important distinction between the input on the canvas and the mesh seen in the 3D viewer, the final model essentially becomes a function of the curves and parameters provided. As such, generating input (e.g., through interpolation/animation) or editing generation variables can have great effects, such as creating adjustable levels of detail (Figure 17) and stretchy, fluid, unrestricted motions (Figure 14).

Performance

Note: The following performance-related observations were made using a MacBook Pro (13-inch, M1, 2020) and would differ depending on computing power.

During typical modeling in the program, the frames per second (FPS)—which is uncapped for measurement purposes—well exceeded the necessary 60, and was generally found anywhere from 400-1000 FPS. It should be noted that lathe-based modeling was found to be significantly faster than cross-sectional modeling at scale (as the raycasting used by side-view models is of exponential complexity).

The only performance concerns truly arise during animation, wherein each model component is re-generated every frame. In scenes with many model components (especially those with short *sample-lengths*), the time used to generate each builds up and may begin to impact frame rate. Luckily, this problem remains fairly insignificant provided that the number of components used is somewhat reasonable, as even complex scenes like Figure 13 (which uses 21 components, many of which have maximized sampling rates) averaged frame rates around 300.

In its current state, the program is already suitable for providing the realtime output needed by artists. To further ensure a smooth, stable runtime for lower-end machines, changes could be made to only re-generate components whose inputs actually change frame-to-frame during animation and ignore models that cannot be seen by the camera.

References

Igarashi, T., Matsuoka, S., & Tanaka, H. (1999). Teddy: a sketching interface for 3D freeform design. SIGGRAPH '99.

hughes, D.S., & hughes, D.S. Model Creation by Sketching Cross-sections.

Gonen, O., & Akleman, E. (2012). Sketch based 3D modeling with curvature classification. Comput. Graph., 36, 521-525.

Cook, B. (n.d.). OpenGL + C++: Modern Graphics for groundbreaking games. Udemy. Retrieved February 18, 2022, from https://www.udemy.com/course/graphics-with-modern-opengl/

Welcome to OpenGL. Learn OpenGL, extensive tutorial resource for learning Modern OpenGL. (n.d.). Retrieved February 18, 2022, from https://learnopengl.com/

Appendices

Appendix A. Annotated Bibliography (9/20/21) of 3D Model Creation by Sketching Cross-sections

Citation:
hughes, D.S., & hughes, D.S. Model Creation by Sketching Cross-sections.

Purpose:
I chose this article because it is very close to what I had in mind for some of the modeling techniques I planned to implement. I wanted to look at articles that focus on non-machine-learning methods (since that's what I plan on doing), and of the selection, I found this one to use the most interesting/unique technique while also producing the most impressive results (in my opinion, of course).

Annotated Bibliography:
The central theme of this article is to develop a novel method of generating models from sketches (more particularly, cross-sections) in a way that would address problems from its predecessors while aiming for a more prototyping-based utility. By making use of a sequence of connected 2D-cross-sections along a 'spine', the method detailed in this paper sought to avoid prior sketch-based limitations, primarily restrictions in possible/conveniently-created shapes (since many previous works focused almost solely on straight lines and flat geometric shapes). Seeing as this paper was the thesis of a student at Yale University in the Department of Computer Science, I believe it's fair to say that the credibility of the author has been verified in this realm. The intended audience of this paper is mostly those looking to either use/develop sketch-based software for the development of 3D-models, especially as that applies to rapid prototyping. Another interesting work I researched was Teddy: a sketching interface for 3D freeform design, which has a plethora of notable similarities and differences. First off, unlike the more recent articles I've seen (which tend to use machine-learning), both demonstrate deterministic, non-AI methods of generating models from sketches. Unlike this paper, which operates via cross-sections and a 3D/2D mixed view, though, Teddy uses a more traditional 2D-drawing workflow wherein drawn shapes are 'inflated' (essentially working like lathes) while also employing gestures to edit (which is a feature lacked by 3D Model Creation by Sketching Cross-sections).

Reflection:
This article pertains quite directly to my research topic as it focuses on the use of cross-sections and sketching in the generation of 3D-models, something I plan to implement and expand upon. The article discusses concepts used to describe the setup's model-creation, implementation details such as sampling and interpolation, an evaluation of the user-experience (something I've

been worrying about quite a bit myself), example models made possible, and an array of extension ideas for future progress in this field (this is basically where I come in).

From this article, I took away much new and important information: it confirmed many ideas I had formed about already in addition to displaying methods I thought impossible and providing critical information. For example, I was previously considering whether it was plausible to allow the splitting of 3D-shapes such as these, but figured they simply wouldn't work, however, this article wrote about how said process could be accomplished in a more realistic way (interpolating into a n-faced shape before carrying on with singular interpolations of each). Something I found very important that was mentioned in this paper was how its implementation was unable to use smooth shading as the representation did not store it in a mesh-friendly format, instead generating individual, unlinked triangles. Knowing this, I can make sure I don't make the same mistake. As a result, I'm now ensuring that I keep in mind how my normals and indices will be generated, not just the vertices.

The information I picked up in this article will be easily applicable to my project (code) in a variety of ways.
Firstly, I may be able to directly apply some of the methods used in their implementation, such as the 'smart-sampling', a technique that was described in detail (which was kindly provided with some helpful, example pseudo-code). It also revealed a couple of good ideas for making a smooth user-interface and control-scheme. For instance, moving through the z-axis can be done easily by scrolling (which I hadn't yet considered [I was thinking more along the lines of simply using a length-timeline]), and this further inspired me as I realized the potential for adding more graphics-tablet friendly gestures (like ctrl-drag) to greatly improve this workflow.

Appendix B. Annotated Bibliography (9/28/21) of Sketch based 3D modeling with curvature classification

Citation:

Gonen, O., & Akleman, E. (2012). Sketch based 3D modeling with curvature classification. Comput. Graph., 36, 521-525.

Purpose:

I chose this paper in particular because it demonstrates the use of sketch-based modeling-techniques being used to generate more topologically complicated shapes, such as those with holes and branching paths. This is something I will surely need to learn much more about in order to allow for the generation of complex models. Additionally, I was rather curious how the Catmull Clark algorithm, something I had already been considering implemented, played a role here, specifically.

Annotated Bibliography:

This article describes a relatively simple yet effective solution to generating smooth, aesthetically-pleasing models with more complicated topologies from barebones-sketches. It separates out the stages it goes through into three categories, Curve construction (the process of turning user-input into curves), Curve partitioning, and Mesh construction (with sharing this process being the paper's central goal). This paper describes the partitioning step as categorizing parts of input-curves into tubular regions, junctions, and caps. Meanwhile, the mesh-construction phase details how the knowledge of what type of region a curve is can be used to generate topologically-correct meshes with holes or branched paths. The paper also describes how the Catmull Clark algorithm can be used to produce better-looking models since they are generated using primarily quads (polygons with 4-vertices). This article is certifiably credible owing not only to its presence and publication on ScienceDirect, a trusted research database, but also to the fact that its authors, Ozgur Gonen and Ergun Akleman are graduates of Texas A&M University with research specializations in rendering, modeling, and related fields (with Ozgur Gonen even focusing on 3D-modeling techniques for his college thesis)—making the two more than qualified to serve as authority figures in this realm. The audience for this paper is those looking to continue advancements in the field of sketch-based modeling, following its primary establishment by Teddy: a sketching interface for 3D freeform design over 20 years ago. Essentially, this article is less about exposing a groundbreaking method to the public, and more about revealing useful techniques to others working in this field to hasten progress. This can be seen by its lack of a user-aimed product/announcement and its heavy use of terminology and concepts that would only be grasped by those with a thorough understanding of prior research. Obviously taking much inspiration from Teddy: a sketching interface for 3D freeform design, I noticed some similarities and differences between these methods. First off, the main idea for generating the majority of a model matches that of Teddy, wherein sketched shapes are 'swept' or revolved in a circular fashion. Unlike Teddy's implementation, this one lacked user-facing

features like gesture-actions (likely because it wasn't for public use) while adding features sorely lacking in Teddy, ie. the ability to create topologically complex models.

Reflection:
This article pertains to my topic quite precisely, for it is a paper detailing a major-improvement to commonly used sketch-based techniques (which is something I ought to be aware of since I plan on using these sorts of methods to some extent). From this article, I believe the most significant contribution was its use of a Curve-partitioning step, something my current and planned implementations were sorely lacking. While it would be fine to avoid this step for simple revolved shapes or those without notable topological features, not having it could either bar me from the creation of more-complicated shapes or cause me to implement a janky-solution in its place. Luckily, now that I have learned the steps carefully-described in this paper, I am far more ready to approach ideas of adding holes and branching geometry to revolved or cross-sectional models. I think the addition of optional holes could be quite useful in cross-section views. I learned that a key curve-classification is that of the junction, which connects different tubular regions together, taking note of the vertex-rings in need of connection and the types of shapes that will allow for such smooth-connections. Without junctions, natural structures like trees would be infuriating to generate, or would produce lower-quality models (as they would likely suffer from clipping issues if model-stacking was used instead). Additionally, from this articles' use of the Catmull Clark subdivision algorithm, I learned that said process will work best with quads. And while I was already planning on targeting quad-based meshes for my program, with this knowledge, I will now be extra careful not to be careless and generate less-adaptable models using arbitrary raw-triangles or n-gons (polygons with more than 4-vertices). Something else insightful that I learned about the use of Catmull Clark is that it is best suited for increasing visual fidelity while decreasing noise when applied to well-sampled models, something I had naively hoped to gain from increasing sampling steps of raw-input.

Appendix C. Annotated Bibliography (10/21/21) of Teddy: A Sketching Interface for 3D Freeform Design

Citation:

Igarashi, T., Matsuoka, S., & Tanaka, H. (1999). Teddy: a sketching interface for 3D freeform design. SIGGRAPH '99.

Purpose:

Firstly, I elected to cover this paper as it was a major turning point in the primary field I'm researching for my project—that being sketch-based modeling (seeing as I've seen it either mentioned/referenced in nearly every paper since). Additionally, I was specifically drawn to this one as it seems to have more of a focus on the actual application created, whereas other articles tend to be more theoretical or only program a singular operation/algorithm demo. This ought to be helpful for me since I plan on creating a user-facing application myself.

Annotated Bibliography:

The central theme/topic of this article was the piece of software created using a variety of sketch-based techniques—Teddy (a Java Applet). A list of notable operations used in the application is given, including: creating objects, painting/erasing on surfaces, extrusion, cutting, and smoothing. Next, the paper briefly describes each operation and its implementation (providing various demonstrative figures) in addition to the initial algorithm/representation of user-strokes in generating models. The authority of the authors is backed up greatly by their portfolios and research backgrounds. For example, Igarashi Takeo, the lead researcher/visionary here has worked extensively in research fields relating to 2D-drawing and 3D-mesh creation (both of which are drawn upon here). It seems the primary audience of this paper was the sketch-based modeling researcher community, seeing as it goes fairly deep into the algorithms used while also noting that the primary source of testers for this application were among fellow undergraduates/scholars (even though it's written with rather understandable terminology, I can't really imagine people outside of those looking to advance the 3D sketch-based field being all too interested in the read). I think it's interesting to contrast this article with Morphological Analysis of Shapes—a paper cited in Teddy. Morphological Analysis of Shapes goes into great mathematical and descriptive depth with a variety of polygonal transformation/analysis techniques (MAT, CAT, morphological congruence, etc.) whereas Teddy only briefly mentions the CAT without much explanation outside its utility in its base-creation algorithm. Additionally, I found Morphological Analysis of Shapes to be significantly harder to read, since it makes heavy use of much technical jargon/lingo compared to the lehman's read that was Teddy.

Reflection:

This article is essentially a description of a more-or-less fully-featured sketch-based modeling program and the algorithms used in it. As such, it pertains quite directly to my research since my

entire goal is making a convenient user-friendly interface that allows for extensive use of 2D-to-3D methods for creating organic models (something Teddy was adept with [seeing as almost all the examples were stuffed-animal like figures]).

From this article, I picked up how similar operations can be represented very differently. For example, when I create SoFs, I make a quad-based model using a series of interconnected rings, however, Teddy, when inflating shapes, starts with a flat base that it lifts up and then fills in with triangles. Something else I learned about was model cutting, wherein a line is projected from the user's view, forming a planar polygon with the interior of the model it intersects with, allowing the polygons on either the top/bottom or left/right to be cut away using a simple inequality. I had done model-splicing in previous projects of mine, but I never thought to use it this way. Additionally, the paper's mention of the concept of a "chordal axis" prompted me to do more research on the subject, learning that it's like a central axis through a 2D-polygon with the ability to split off and curve. This will surely be useful once I get to more cross-sectional forms of modeling in my implementation.

After reading this article and playing around with a demo of the Applet I found online, I now have much I'd like to apply from the knowledge/experience I garnered. Although my experience using the demo Applet was a rather janky one, seeing and working with a functioning implementation has been quite eye-opening. Seeing that what I'm trying to achieve has been done in some respects—even though, with slightly different techniques from what I've been planning—I've realized that, to truly add something to the field, it might be worth diving in deeper with a specific feature (of course, only once my initial implementation is fairly sound). For this, I'm starting to think that sketch-based animation could be very interesting since it's not featured in this paper (or any other I've seen, for that matter). Additionally, some features demonstrated in the paper that I'm thinking about implementing myself now include smoothing (since my current plans have no way of smoothing rough edges of combined model-components) and drawing-on-surfaces (provided that I have time).