Tumbler Animation Project Report

Michael Ferguson

12/18/2002

Project Description

The goal of the project is to create an animation tool which can "unroll" and "slice" a three-dimensional model composed only of triangles. This work was motivated by Lilla LoCurto and Bill Outcault and completed in part as an undergraduate independent study, CS 490, at Cornell University, under the guidance of Kavita Bala.

Use Cases (Goals)

In a use case, a user creates a movie of a person unrolling. She creates a scene and imports a large PLY file. She specifies that the model should be fully unrolled at a later time and export a high-resolution QuickTime move of the scene.

In a second use case, users interested in the calligraphic forms of sections of people import two models. From these two models, they create a variety of sections and reposition them to make interesting movement. They export this animation to a high-resolution QuickTime movie.

Project Status

Figure 1 shows the interface that Tumbler presents to its users. This window contains a tool palette, which can be used to rotate, translate and scale a selected object or to rotate, translate, or zoom the camera. There is a timeline at the bottom of the window with which a user can change the current time. The model displayed in the center has an "unroll" effect attached to it, and the grey disk controls how the unrolling is done. Note that this version does not include support for the "slice" effect. However, it can unroll a model in spherical or cylindrical coordinates with and without "spines", import SMF and PLY files, and export fixed-resolution QuickTime movies and SMF files.

Technical Implementation

3D Objects

A scene stores a list of 3D objects which are arranged into a hierarchy of coordinate systems. Reorienting an object at the top of the hierarchy will also reorient its children. An object also may contain cells which are always drawn in an object's local coordinate system.

The grey disk described above is currently the only example of a control object. A control object represents a setting for an effect with its position, and it can be translated, rotated, and scaled in the same way as any other object in the scene. I implemented this control disk simply by adding it as a child of the model object in the hierarchy.

Level of Detail

In order to render large models quickly, they need to be simplified. I investigated two methods, *propslim* and *plycrunch*, for this project in my accompanying Level of Detail Survey. Both of these methods are a publicly-available command-line programs. Currently, to reduce a model, one of these programs must be run from a command line on an input file in the correct format.

The "Unroll" Effect

The "unroll" effect animates between two positions of the model: the start state and the unrolled state. Imagine projecting each point in the model onto a cylinder or sphere of a certain radius, and then unrolling that cylinder or sphere. In the start state, the effect makes no change to the model. In the unrolled state, each point in the model is converted into cylindrical or spherical coordinates and the resulting rectangular surface is drawn. See Figure 2 for an example of this transformation. The radius of the line from the origin to the point on the model becomes the height of the projected surface at that point. The in-between states are created simply by linearly interpolating between the start and end states.

The "unroll" effect sends the points at the theta angles 0 and 2π to diverging locations (see Figure 2), and so all triangles with a positive x-value and which intersect the xz plane are split by the xz plane and each new triangle is sent to the correct location. Finally, for simplicity, I implemented changes in the position of the projective cylinder by transforming all of the vertices before and after the "unroll" is processed. Also, because a user usually wants a body to be unrolled from its "spine" which may not lie upon the z-axis, for cylindrical projections, I vary the center of the imaginary cylinder so that it matches the center of a vertical slice of the model - see Figure 3. Figure 4 shows an example of this effect applied to a person.

Effects List Processing

In order to enable a user to combine effects, I represent models with effects applied to them as a source object together with a list of effects that the object goes through before being drawn. Effects might include "unrolling" or "slicing." The source object exposes its triangles for modification to each effect. So to apply the effects in order, a "scratch" copy of the original model is made. Then each effect is applied to this scratch copy, which is then stored and used whenever the object is drawn. If the effects list and the current parameters for each effect have not changed, the scratch copy can be reused instead of being recomputed. Although I completed the back-end for the effects list, currently it is not possible to compose several effects because the user interface is not complete.

Lessons Learned

This project has been the first large-scale application that I have developed independently. I have learned a few software engineering lessons:

• Plan time for integration. My project schedule left adequate time to finish some of the features themselves, but I had not planned on the time I needed to integrate the features as well.

• Don't think *too* hard about design. Programming is like writing sometimes - to get started it might be better to create a working copy than to design it "right the first time." There are two reasons: it is easy to get stuck on design problems when it is unclear what issues will arise, and at the end of the day if you've only done design work, you will have not created the program. If your aim is to create a working copy, you can make a design choice just for the sake of simplicity. These design choices can be revisited later if they need to be. As my sculpture professor once told me, "Don't do it all in your head-bone;" it's often easier to make design decisions in the act of creating the program or the sculpture. When adding a new class to do one thing, it probably doesn't *need* to be all that flexible. Consider putting off adding flexibility until you need it.

• For intensely mathematical sections of the program,I created quite a few notes in my book. I think that it would help to create descriptions of these sections complete with pictures as need be so that they can be preserved with the source code. Although

computers can compile and execute C, it is not a terribly legible mathematical notation.
• Using existing libraries and frameworks can be useful. However, sometimes it is more trouble to use a library than to grow your own code. In this project, I thought about using the QuickTime movie format to store all of my parameters for effects and for model positions. After doing a lot of research on QuickTime, I concluded that it was a bad idea, mostly because the documentation was not clear about exactly how I would do that. Although it would have been an interesting project, the functionality I would gain from QuickTime would be questionable; without QuickTime, I only needed to create a class which performed linear interpolation. I think it would have been a nightmare to use QuickTime to store all of my data. The lesson is that even when a library claims to be able to do what you would like it to, you should only use a library for its supported use.
• User interface code is not trivial to implement. Although an interface is only a way a person can use some back-end functionality, it takes quite a bit of work to create a good interface. For research projects on back-end parts, the simplest user interface to implement is probably the best one.
• There is a wealth of knowledge in the computer science literature. I was very glad that I could draw upon previous level-of-detail research instead of creating my own haphazard method. Furthermore, the oldest and most-cited papers are not necessarily the best ones for every reader; I learned more about quaternions from a Google search than from the "authoritative" original paper, because the original paper was difficult to follow while the page Google found was actually written for undergraduates like me.

Future Work

As I mentioned, the effect list user interface is not complete. And although I intended to support a second effect - slicing a model - I did not get to slicing in this version. Furthermore, I still need to add support for specifying lights and animating camera movements. In addition, the graph of key-frames and the stored parameters at these frames is not yet completed. The translate, rotate, and scale tools should allow constraints, and these tools currently only manipulate one object (instead of all of the objects in a selection). I would like to add a mechanism to "look at" a certain object in case the camera is aimed in the wrong direction, a window to choose the output movie resolution, and support for anti-aliasing.

Figure 1
the Tumbler user interface

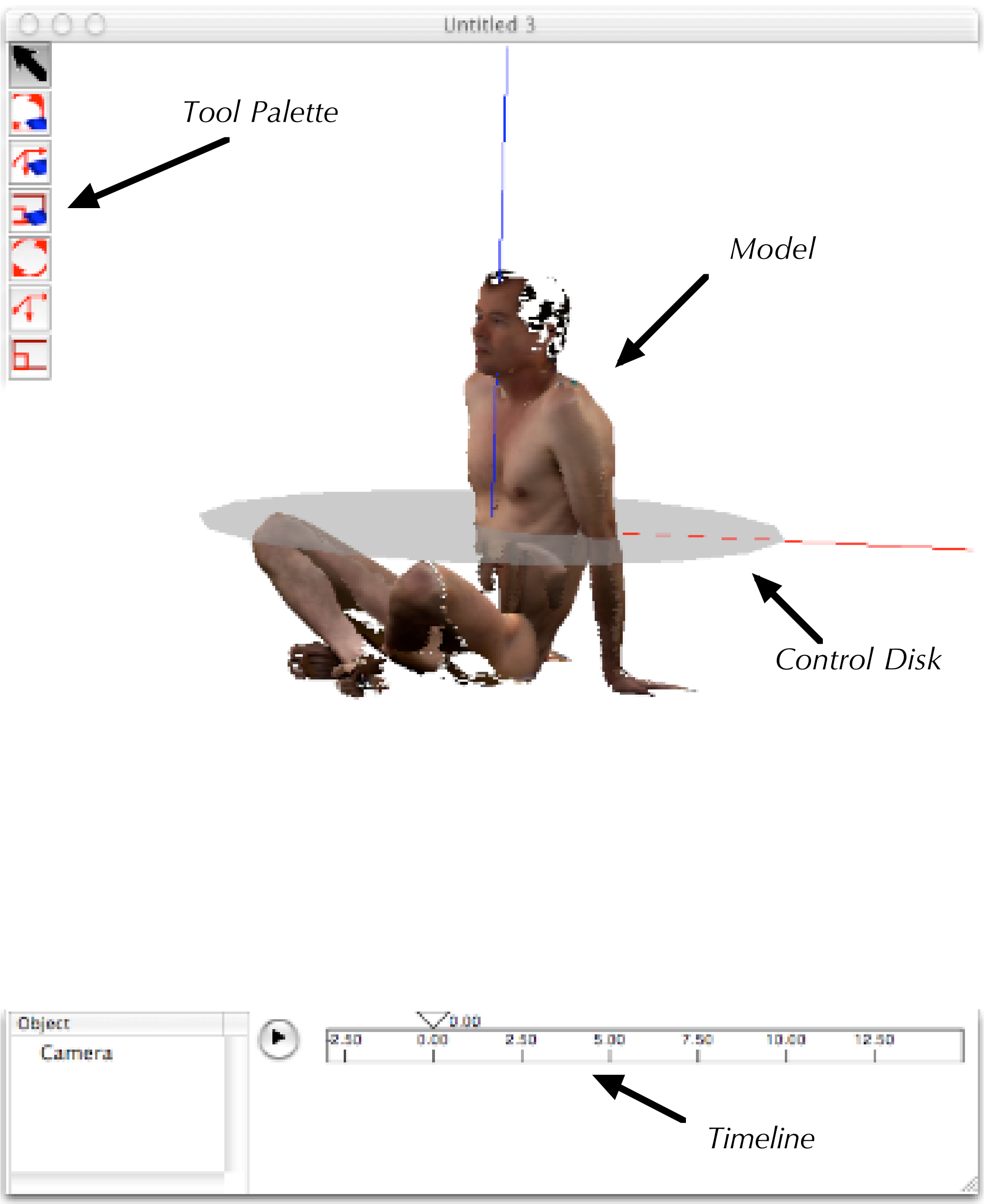
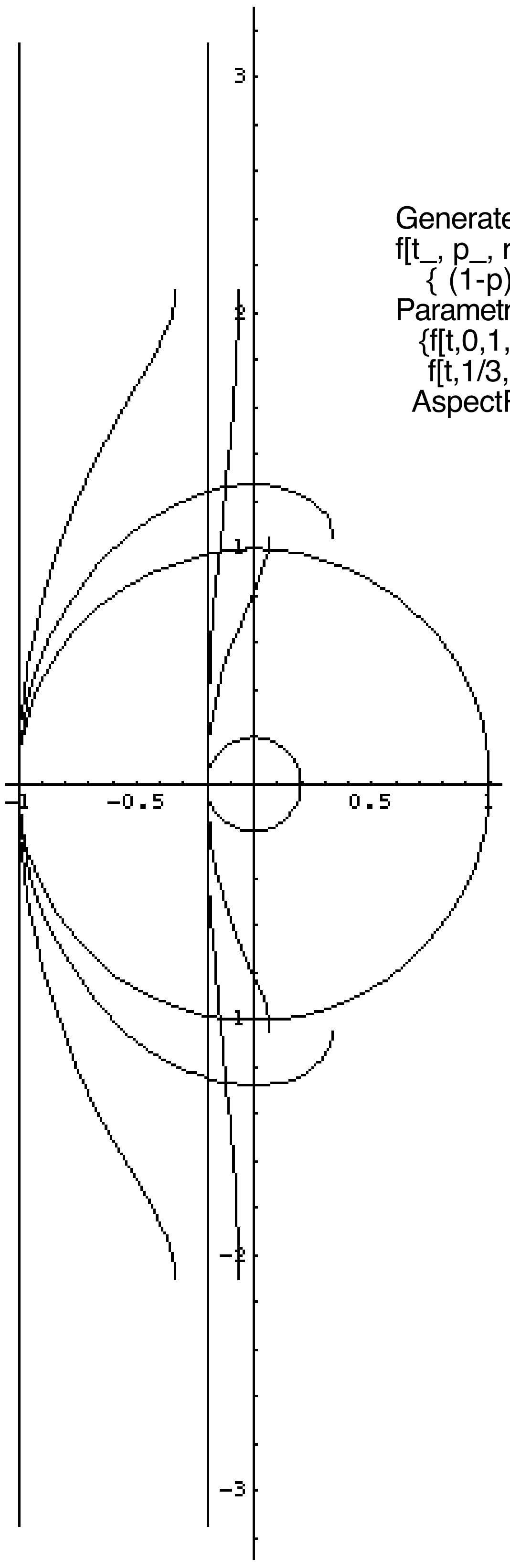


Figure 2

Example of interpolation between starting locations and cylindrical projection



Generated in *Mathematica* with:

```
f[t_, p_, r_, rmax_] :=
   { (1-p)*r*Cos[t]-p*r, (1-p)*r*Sin[t]+p*rmax*(\[Pi]-t)}
ParametricPlot[
  {f[t,0,1,1], f[t,1/3,1,1], f[t, 2/3,1,1],f[t,1,1,1],f[t,0,0.2,1],
   f[t,1/3,0.2,1], f[t, 2/3,0.2,1], f[t,1,0.2,1]  }, {t, 0, 2*\[Pi]},
  AspectRatio\[Rule]Automatic]
```

Figure 3

Example of cylindrical "spine"

Each cylinder represents a section of an object with center at the center of the cylinder. The dark line is the center used in the projection transformation; it is interpolated between the values of the centers of the cylinders.

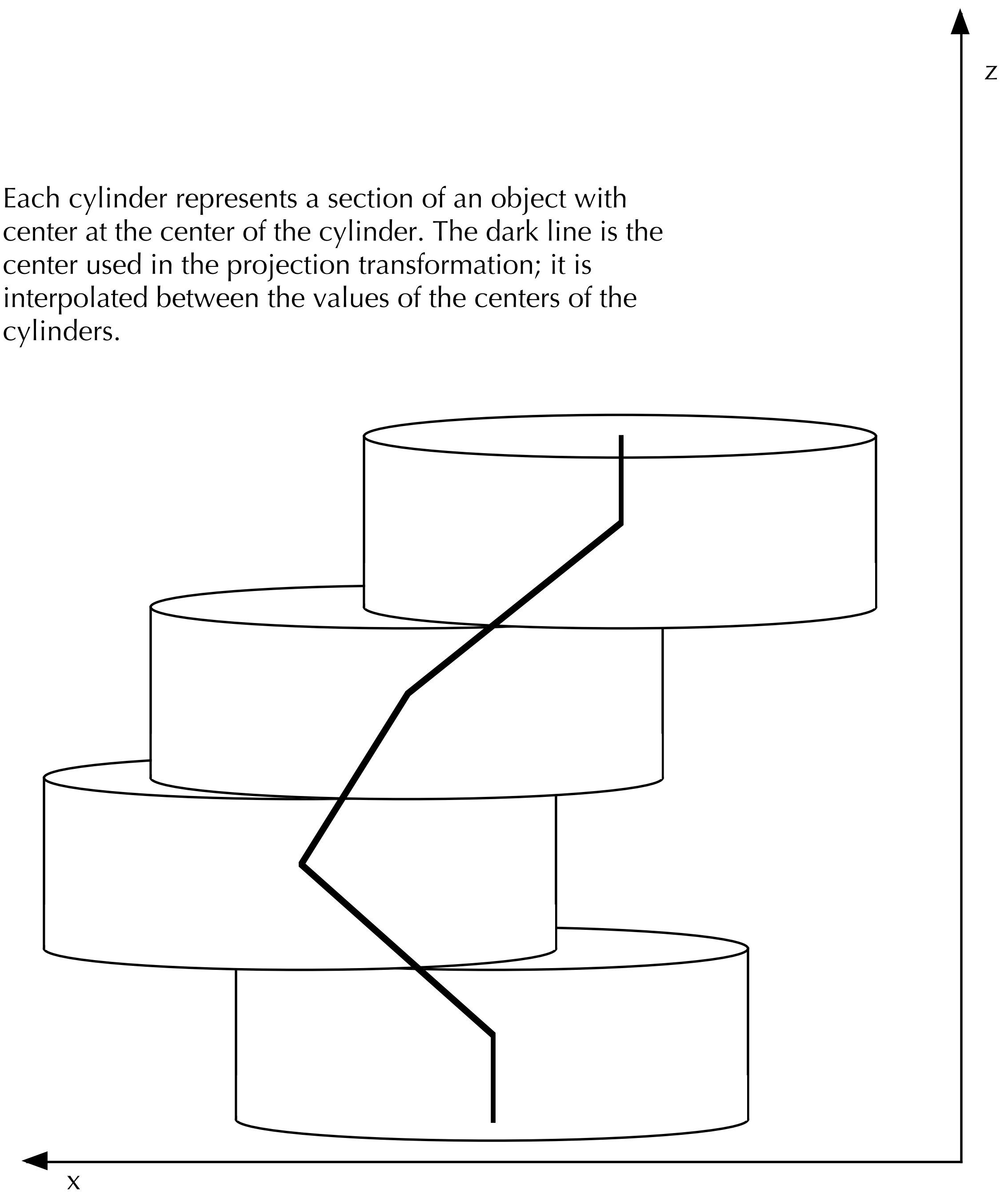
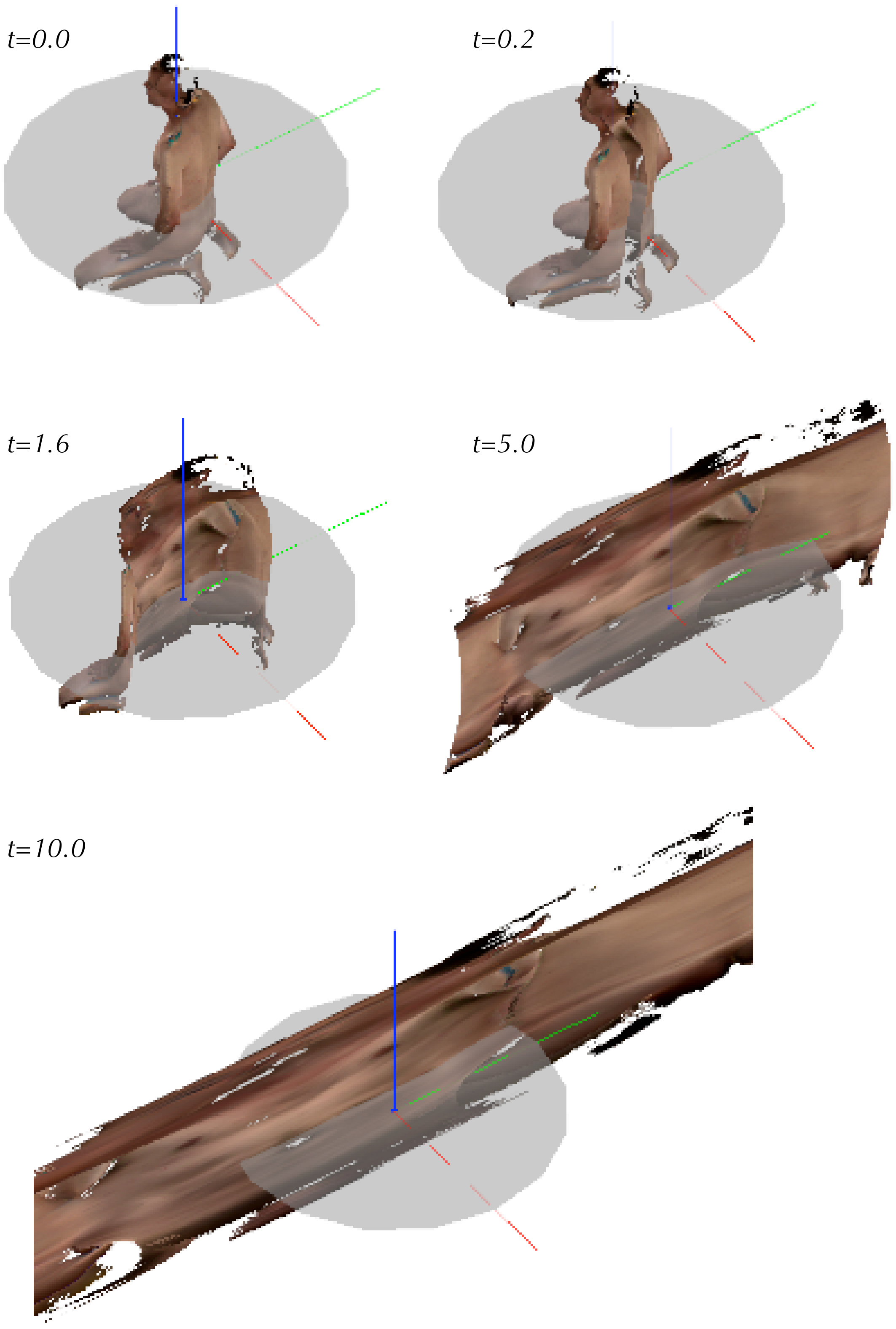


Figure 4

Example of the "unroll" effect


t=0.0
t=0.2
t=1.6
t=5.0
t=10.0