|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CS 6491 Project # 6  Asymmetric Revolution |  | Macintosh HD:Users:donovanhatch:Downloads:RevBend:RevBend:data:deep_donovan.jpg |  |  |
|  |  | Donovan Deep  Hatch Ghosh | Sumit Khetarpal | Nick  Olive |

# Objective

Figure

Implement design system for non-symmetric revolution of a solid around curved spine

# Features Supported

* Select curves and modify edges. Press ‘1’ to select left, ‘2’ to select right curve
* To modify middle axis, press ‘0’
* Insert Edges (Press ‘c’, and click on location of desired control point)
* Drag control points to change the shape of the curve
* To modify radius, press ‘r’, and drag the radius of control points
* Press ‘m’ to toggle between normal, radial, ball offsets
* Press ‘a’ in 2D or 3D mode to animate
* Press ‘s’ to toggle using ball morph vs lerp
* Press ‘\’ in ball morph mode to show arcs during 2d animation
* For more key options, please reference controls.txt in base folder

# Part 1: B-Spline

*Major contributor: Sumit Khetarpal*

**Details of implementation**

* Quintic B-Spline Curve: Given a minimum of 5 control points, the quintic B-Spline curve can be generated by refine, and four dual operations in sequence
* Clamping B-Spline Curve: The dual operations lead to reduction of one vertex. As a result of four dual operations, the curve can shrink significantly. In order to avoid user confusion, we clamped the B-spline curves forcing it to pass through the first and the last point. This was done by adding a duplicate of the control point at both beginning and end, before the rd4 operation, and then removing them after the rd4 operation.
* Radius Profile via Quintic B-Spline: The user can change the radius at any control point. To generate a smooth radius interpolation, we again used the quantic B-Spline subdivision. We know the radius, and we know the index of each of those control points. Therefore, we can combine the information into point, pt (index, radius). The points can then be subdivided to get a smooth quantic B-spline curve.
* Offset Calculation: Three different offsets were used. To find the normal offset, we calculated the tangent of the curve, and then obtained the normal by rotating the tangent ninety degrees. The normal offset is then simple displacement along the normal vector. For radial offset, we first calculated the normal at the offset. The ball offset is a combination of first calculating normal offset at half the desired distance, followed by calculating radial offset using remaining half distance.

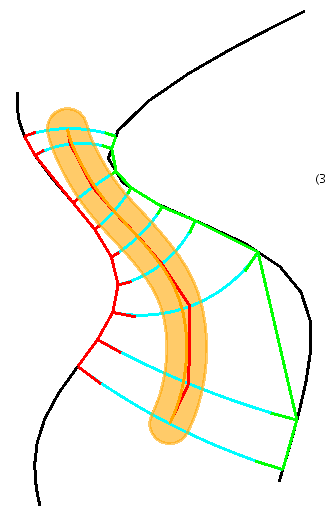
Figure

* Figure 1, shows normal (red), radial (green), and ball (blue) offset curves. The spine of the curve is shown in the black, and the control points are bigger circles that can be dragged around.

# Part 2: Ball Morph Interpolation

*Major contributor: Donovan Hatch*

**Details of implementation**

* Ball Morph and Lerp: User can switch between using a ball morph transformation or a lerp transformation on the 2d B-spline control curves.
* Ball Morph: Ball morph finds corresponding points on the two different user generated curves (after mirroring c1).
* Once it’s found the corresponding points on the two lines, it finds the tangents of each match, creates a circle where those tangent lines meet, and then creates an arc between those two points based off of the circle that was created.
* The corresponding point intermediate position is calculated from this arc.
* At the same time, a the radii for each corresponding point is calculated by performing a lerp between the two different radii values.

**What Works**

* Can create proper ball morph transformation arcs
* Can still switch to lerp mode instead of ball morph
* Can show arcs for all morph

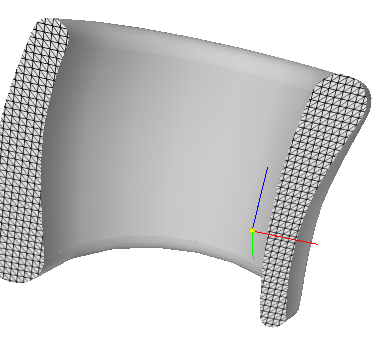
**What Doesn’t Work**

* Lines drawn that intersect on the mirror will probably not work correctly in most cases. My implementations of trying to fix that broke other things and in the end I decided to leave it without being able to intersect.
* Curves that are wrapped around find an out of order correspondence resulting in improper transformations. This is a result

# Part 3: Revolute Sweep

Major contributor: Nick Olive

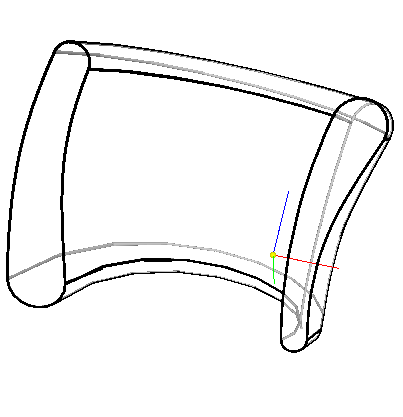
## Controls

* 3 - Return to 2D
* a - animate frames
* v - show the volume of the shape in console
* w - show silhouette mode
* space - change camera position
* c - change focal position
* x - tilt camera
* z - zoom camera
* t - toggle textured mode
* , - decrease aplha
* . - increase alpha
* { - decrease subdivisions
* } - increase subdivisions
* [ - decrease rotational resolution
* ] - increase rotational resolution

## Details of implementation

* Connect sequential points morphed by part 2 in quads
* Subdivide each quad if needed (controlled by curly braces)
* If smooth mode is set calculate the normal at each vertex by averaging the adjacent faces
* If wire frame mode is on the shape draws a silhouette in light grey, discussed later, then the shape draws with 0 alpha value to update the Z-Buffer while not changing the image, then draws the silhouette again in foreground colors.
* The hatching pattern is a 10x10 bitmap that is then texture mapped on to the cross-sections
* The silhouette edges are calculated by checking if the dot product of the two adjacent faces to the edges and the vector from the vertex to the camera are opposite.
* Sharp corners are just calculated as a threshold of the adjacent face angles
* To find the volume each quad on the face is calculated as two triangles, each has its volume calculated with respect to the origin.

## What Works

* smooth shading
* wire frame
* animating the stroke morphing while rotating on the spine
* rotating the view, and updating the swept angle
* flat shading
* volume calculation

## What Doesn’t Work

* Interaction with bending sometimes causes visual bugs
* Some of the radius code for the stroke causes an assertion in opengl to fail crashing the program when it is 3d rendered

# Part 4: Spine Bending

*Major contributor: Deep Ghosh*

**Details of implementation**

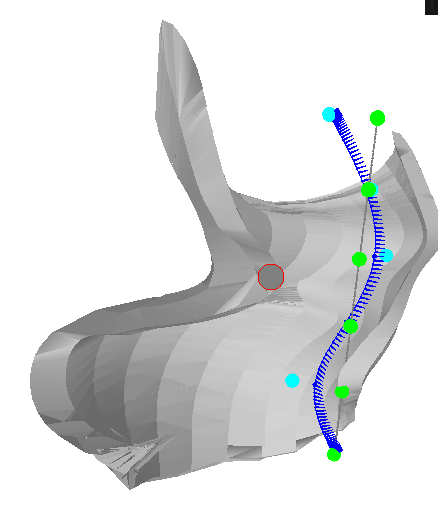
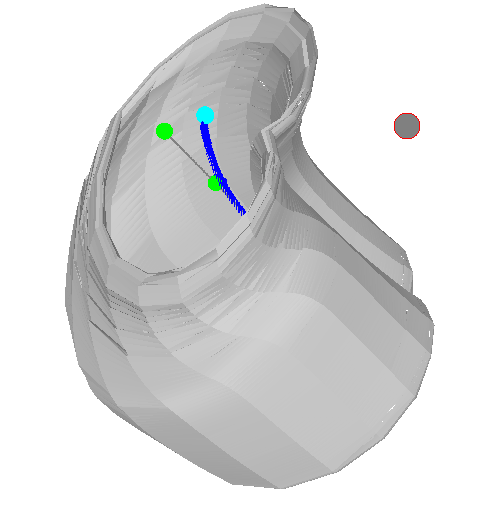
* The central spine is created with a set of 6 control points and a curve generated using quintic b-spline implementation.
* When switching over to 3D mode the code uses the sample point generated for generating the quantic b spline to calculate a twist free advection vector. The process starts by registering a normal, tangent and binormal at the base point. A point is registered with on the normal. For every subsequent point the registered point is propagated along the curve parallel to tangents at each point. The registered point is re registered at each point by against the local frame. To calculate the local frame at each point we have used an exterior product in place of a det product to calculate the local x,y,z co,ordinates.
* By pressing 4 we can show the twist free advection at any point.
* “j” toggles bending.
* When bending the model, each point is checked against the bent spine for a corresponding projection. This is done in 2 steps. One we find the projection of the original point against the fixed control polyline(toggled by 5). This projected point is then transformed to the bent spine by travelling a equal distance down the quintic spine to locate a point on the spine. Using the local frame of the located point we calculate the new point location against the bent spine.
* Shading is achieved via the implementation of part 3 as bent shape is generated by the same code implementation.
* Bending is possible in 3D by dragging the mouse around. To move in vertical axis, press down ‘b’ and drag mouse. Points are selected by clicking.
* Added code to open Help note using ‘H’.
* Used concepts from reference 2.

**What Works**

* Bending of shape
* Bending in 3D
* Showing of twist free advection
* Toggling control polyline
* Shading of bent shape

**What Doesn’t Work**

* There is no limit to bend of spine. This can lead to unpredictable shapes.
* Picking of points in 3D can inadvertently pick a different point than the user intended. Occasional occurrence.
* Large amount of sample points in the 2D quintic b-spline can lead to the 3D part of the applications slowing down considerably.
* After integration with the bending code we found that the 3D models edges are becoming jagged. We are yet to find the resolution.



# References

1. <http://disney-animation.s3.amazonaws.com/library/bmorph.pdf>
2. <http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=B6F6CDEBD7D269290A262C37CF299939?doi=10.1.1.90.4139&rep=rep1&type=pdf>
3. <http://freespace.virgin.net/hugo.elias/graphics/x_polygo.htm>
4. <http://www.ntu.edu.sg/home/ehchua/programming/opengl/cg_basicstheory.html>