Project 5. CS3451, Fall 2012. Instructor: Jarek Rossignac

Due by midnight on Nov 30

Teams of up to 3 people: If you do not have a team, please email all 3 TAs.

The objective is to allow the user to create and orient 4 polygonal solids of revolution and to create a smooth 3D animation that morphs between two of them or between all 4 of them. Each solid j is a polygonal approximation of a solid of revolution (http://en.wikipedia.org/wiki/Solid_of_revolution)defined by a profile polyline Pj, an axis direction Dj, and a sampling count Kj. Provide simple but non-trivial defaults for these all parameters. For example, set K1=3, K2=4….

PART A (Editing a solid): Should be done by Nov 13

1) Let the user switch between the 4 solids by pressing '1'…'4' to select the one that is being displayed and edited
2) For the selected solid j, show two things: the PROFILE (a polygon Pj as a 2D curve near the left border of the window) and a 3D view of the corresponding SOLID (the polygonal approximation Sj of the rotational sweep of the profile Pj around some arbitrarily oriented axis of direction Dj).
3) The user should be able to use the mouse to select points of Pj, move them by dragging to a new location, delete them by keeping 'd' pressed while clicking on them, and insert new ones by keeping 'i' pressed and clicking near the center of an edge and dragging the newly created point to a new location. The first and last point of Pj should be on the axis (left border of the canvas). Provide simple, yet different defaults for each Pj.
4) The surface of the solid Sj is defined as the rotational sweep of the area between the profile and the left edge of the canvas. The sweep is a 3D sweep around an axis that passes through a point that projects at the center of the screen and has direction Dj. The integer Kj defines the number of sample profiles curves regularly distributed around the axis in 3D. Each edge of the profile sweeps a portion of a cone during its rotation around the axis from one instance of the profile to the next instance. Your 3D model should approximate that cone portion by a quad and should triangulate the quad into 2 coplanar triangles. For example, if Pj has only 3 vertices (A,B,C) such that the angle at B is 90 degrees and of course A and C are on the left border (axis) and if Kj=4, then Sj should look like a diamond (the union of two pyramid glued together at their quad floor). If you increase Kj to 32, the pyramids will resemble cones. If you subdivide Pj several times, the whole think may resemble a sphere.
5) The orientation of the solid in 3D is defined by the direction Dj of the axis of rotation and (optionally by a rotation of that solid around that axis). Provide a simple GUI through which the user can easily and intuitively change the direction Dj (and optionally the orientation of Sj around the axis). For example, then the user keeps 'o' pressed and moves the mouse, the horizontal and vertical displacements may change (increment/decrement)

the x and y components of Dj, while keeping them between -1 and +1. The z component may be obtained via normalization. As an option, you may also want to provide the ability to edit the rotation angle wj of the solid around its axis.

6) The solid Sj should be shown centered so that the middle point of its axis (midpoint of the first and last vertex of Pj along the axis is at the center of the screen.
7) The user should be able increment/decrement Kj by pressing ',' or '.'
8) You should provide an option (pressing 'C') that makes Pj convex. Make sure that you clearly explain your algorithm for converting Pj into a convex polygon (assuming that there is an edge, along the left border, between the last and first vertex. Most algorithms that do so, delete some vertices. In your write up, explain which vertices you delete, how do you test which ones to delete, and which order you delete them in. Make sure to test your algorithm thoroughly to ensure that it always works for a valid (non-self intersecting) input Pj. If you do not manage to provide such an algorithm, at least warn the user by painting in red the non-convex vertices of Pj.
9) Provide a key action ('W') to save (write) the parameters for all 4 solids (Pj, Kj, Dj, wj) and to load it back. This will help you debug and demo your project.
10) Once you can render each solid, notice that its faces are either quads or triangles. Write a function or method that creates a triangle mesh represented using the corner table (G[], V[], O[]) discussed in class and representing the boundary of the same solid and replace your original display by the display of that triangle mesh. Provide a button 'm' that toggles between showing the original faces and showing the triangle mesh. The two images should be identical (show the triangle mesh in a different color to clarify which is which).
11) Compute a triangle normal Ti for each triangle of the mesh as the cross-product of two of these edges. Make sure that these normal point outwards by drawing them as short line segments out of the centroid of Ti. Do not normalize these normal, so that their magnitude is proportional to the area of Ti. Then, compute vertex normal as sums of the normal of the incident triangles. Finally, normalize the vertex normal. Provide a toggle 'g' between using flat or Gouraud shading for the mesh. Include images of the original faces, of the flat shaded triangle mesh with triangle normal, and of the smooth shaded mesh with vertex normal.

PART B (Morphing between two solids): Should be done by Nov 20

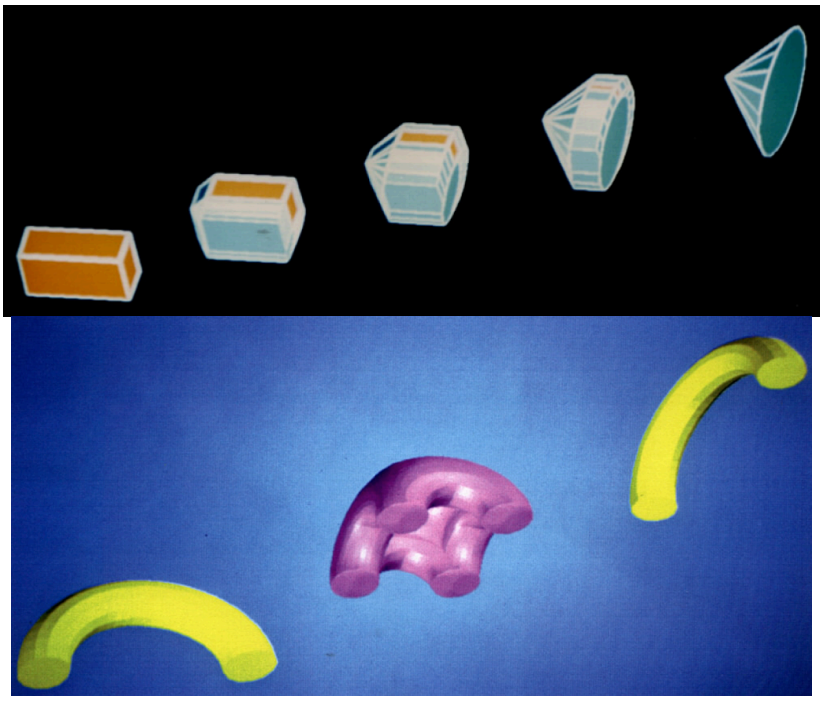
1) Implement a Minkowski morph between S1 and S2 first. Then implement an animation that interpolates between the four solids, one after the other using Neville's blending of the results of the Minkowski morphs. The approach has been discussed in class, and a paper has been provided Below, we provide some suggestions and details.
2) Your Minkowski morph from S1 to S2 will keep the evolving solid centered. To debug the morph and to show it, I suggest that you offer a mode where in

addition to showing the animated morph, you also show S1 (translated by (-width/4,0,0)) and S2 (translated by (width/4,0,0)) or something like this. So that a viewer can see the initial and final shape constraints (left and right) and the animation (center).

3) Let variable t (or "currentAnimationTime" for those who like longer variable names) be either incremented and decremented automatically (in animation mode) or controlled by the user. Provide a toggle 'a' between these two modes and , when not in animation mode, let the user change the value of t by keeping 't' pressed and moving the mouse left or right.
4) First make sure that S1 and S2 are convex. Select manually (in your program) a single face of S1 and (automatically) identify the unique matching vertex of S2. Highlight them both in red and verify that the triangle of S1 properly animates to become the corresponding vertex. Check that you have a single matching vertex. Provide yourself a key to move to the next face of S1 and make sure that your computation of the matching vertex works for all faces one by one. Then, try it on non-convex solid S2, in which case you could have more than one matching vertex. In such cases, produce an animated instance of the face of S1 for each matching vertex on S2. Hence, during animation, the same face will become 1 or more copies, and each copy will fly towards a different vertex of S2, remaining parallel to itself while shrinking. Stop the animation midcourse and snap a picture showing such a case. Debug it to verify that each face does have at least one matching vertex. You may run into some numeric issues (for example, an edge of S2 is almost parallel to a face of S1). Identify such a case (save the parameters and the face number), so that you can more easily debug it. Then, implement a key 'e' that perturbs each vertex of each solid by a very small random amount (between -0.0001 and +0.0001 or something like that). Also try using <= instead of < or vice versa in your test for deciding whether a vertex is matching. By principle, you want each face of S1 to match with at least one vertex of S2 (exactly one if the solids are convex).
5) Once the above works, put it in a loop so that you can animate all faces of S1 as they morph with their corresponding vertices in S2. Verify that this works. Color all triangles of S1 in green and preserve their color during the animation. Do the reverse by swapping the role of S1 and S2, but during the animation of the faces of S2, change the time parameter to be 1-t. Use red for the faces of S2. Now you should have a morph with all triangles. But the animated shape will have some missing quad faces. Make a picture of the mid-course situation, which should show green and red triangles and missing quads.
6) Now temporarily comment out the display in the morph of the green and red faces and prepare to debug the computation and animation of the quads. Select the first corner c of S1 to identify an edge of S1 (facing c in c.t). Show it in blue on the left rendering of S1. Write a program that will identify on S2 all matching edges and show them in blue on S2 (right). The criteria for the testing whether two edges match were discussed in class. Then involve computing the cross-product of the two edge tangents and testing whether

all vectors from their vertices to neighboring vertices either all have a positive dot product or all have a negative dot product with that normal. Once you have debugged the matching, show the animation of the blue quad that morphs between the edge of S1 and the corresponding edges of S2. Discuss in your write up whether each edge of S1 must have at least one matching edge on S2 and also discuss the maximum number (computational complexity) of the matching edges.

7) Once the above is debugged, put it in a loop over all corners c of S1 for which c<c.o (to avoid duplication) and animate all quads. Capture a mid-course image of that animation (showing only the blue quads) and include in your write up.
8) Now, put the quads together with the green and red faces. Try this on a variety of convex and non-convex models. There should be no holes in the animated surface, but for non-convex models, it may self-intersect.
9) Make a video showing the editing of S1 and S2, including insertion and moving of vertices and rotation of the axis and also the animations of the green faces, of the red faces, of the green and red faces, of the blue quads, and of all 3 sets together. Show an animation for convex solids, then for a case when one solid is convex (sphere like or cylinder like) and the other not, and finally when both of them are not convex.

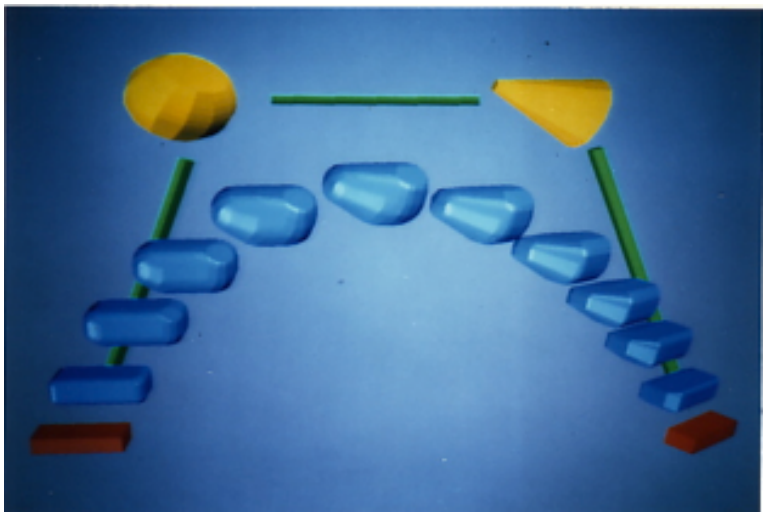


PART C (Morphing between four convex solids): Should be done by Nov 27

1) Here you want to extend the approach to 4 solids. A paper was provided that explains a Minkowski morph between four polyhedral using a Bezier formulation (which is not interpolating), while your project should use an

interpolating Neville's formulation. In that sense, your solution will be a novel contribution, never done before! Congratulations! But, there is a risk that it may not work. If it does not, then implement the cubic Bezier morph as defined in the paper provided.

2) Suggested approach: The Neville formulation for curves expresses a cubic interpolant as a cascade of linear interplants (extrapolating past the control points as needed). These linear interpolants can be written as (1-t)A+tB. Simply replace A and B by solids and + by a Minkowski sum. The difficulty of this approach is that it assumes that you can cascade Minkowski sums (to a Minkowski sum of Minkowski sums). Hence, I suggest that you compute the corner table of the triangle mesh of the Minkowski sum of two triangle meshes. For simplicity, assume that the two solids are convex. The difficulty there is not to split the quads into triangle pairs (which is trivial) but to identify coincident vertices. The procedure outlined in PART B identifies the path of each vertex-use of a face or edge. In other words, it produces a set of triangles for the morphing mesh and for each vertex of each triangle, it produces a path, which is defined in terms of 4 vertex IDs (one in S1, one in S2…). Vertices that have the same path must be identified as identical and their triangle description (V[] table) must be updated accordingly. Then, you can compute the O[] table as usual and the connectivity of the morphing mesh will remain constant, so you are done.
3) The debugging here is to verify that the morphing mesh has proper connectivity. Use the Gouraud (smooth) shading discussed above to check. Also, check that each corner has an opposite and that c.o.o==c.
4) This process may be more expensive, so you may want to precompute the connectivity and use it during animation (see paper).
5) Produce some animations showing various combinations of simple convex shapes: cube, 6 sided cylinder, tetrahedron, pyramid and capture them on tape. Here, you may use the same color for each face, but show the 4 solids around the animated one, so that in the video you see what is going on.



PART D (Video and report):

Put together a short video and post it on You Tube. It should contain the following:

1) It should start with a title image (3 secs) with the pictures and names of the group members, the tile ("Nevill-Minkowski morph") and the subtitle: "Project 5 for CS3451 Fall 2012. Georgia Tech. Instructor: Jarek Rossignac."
2) Then, it should show 4 interesting solids (1 second each) and then the morphing animation played twice, once with a static camera and once when the camera is moving around the solid.
3) Then, show a screen capture where the user modifies one of the profiles and its direction.
4) Then, show the resulting animation.
5) Then capture a brief explanation of how the animation was computed, making references to prior art.
6) In your explanation, first show and explain Minkoswki sum, and then explain how you compute the Minkowski morph between two solids and use the green, red, blue animations produced in phase B. Explain briefly how the matching is defined and computed.
7) To explain the final part, show the Bezier and Neville formulae and how they work on a curve. Then explain how I have extended the Bezier formulation to Minkowski sums. Finally explain how you have extended the Neville formulation to Minkowski sums of polyhedral.
8) These explanations may use screen capture and voice over or a video of you explaining things on paper or on a white board.

REFERENCES:

http://en.wikipedia.org/wiki/Minkowski_sum

"Solid-Interpolating Deformations: Construction and Animation of PIPs", A. Kaul and J. Rossignac, Computers&Graphics, Vol. 16, No. 1, pp. 107-115, 1992.

"AGRELs and BIPs: Metamorphosis as a Bezier curve in the space of polyhedra", J. Rossignac and A. Kaul, Proc. Eurographics, Oslo, Norway, Computer Graphics Forum, Vol 13, No 3, pp. C179-C184, Sept 1994.

Solid and Physical Modeling, J. Rossignac. 2007. PDF. Chapter in the Wiley Encyclopedia of Electrical and Electronics Engineerig. Ed. J. Webster.
http://www.gvu.gatech.edu/people/official/jarek/papers/SPM.pdf