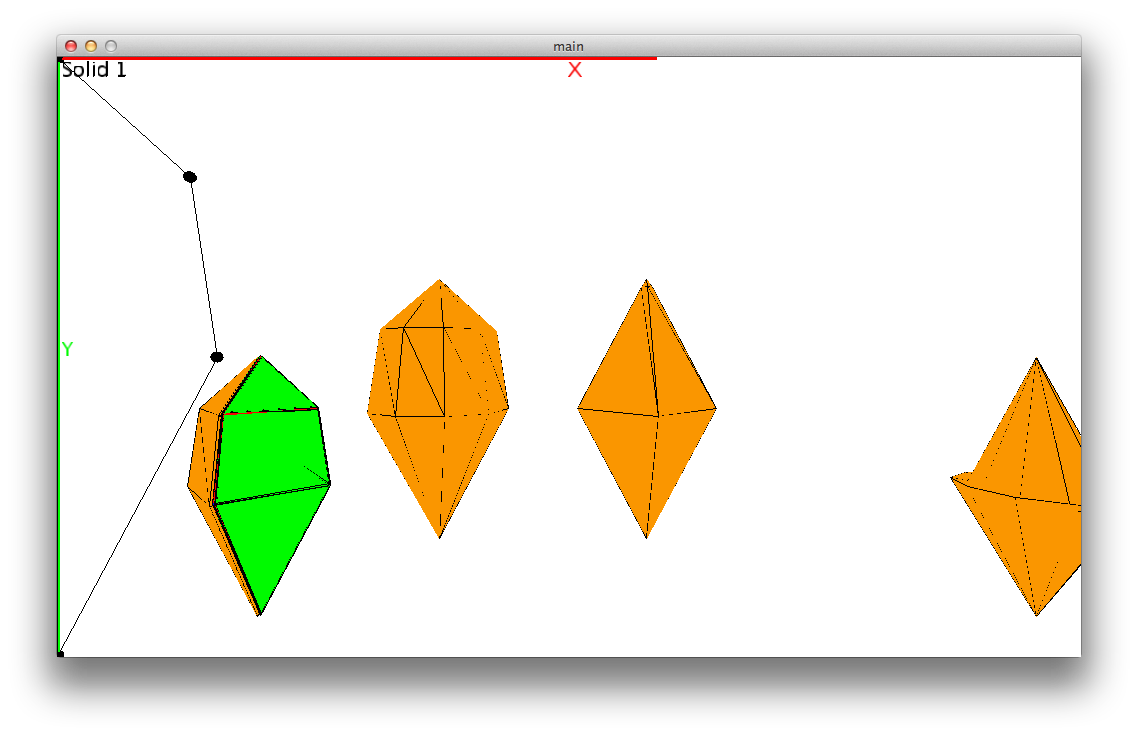
Names: Joon Ki Hong / Krista Patel

Project 5 – CS3451 Fall 2012

Instructor: Jarek Rossignac

How to use:

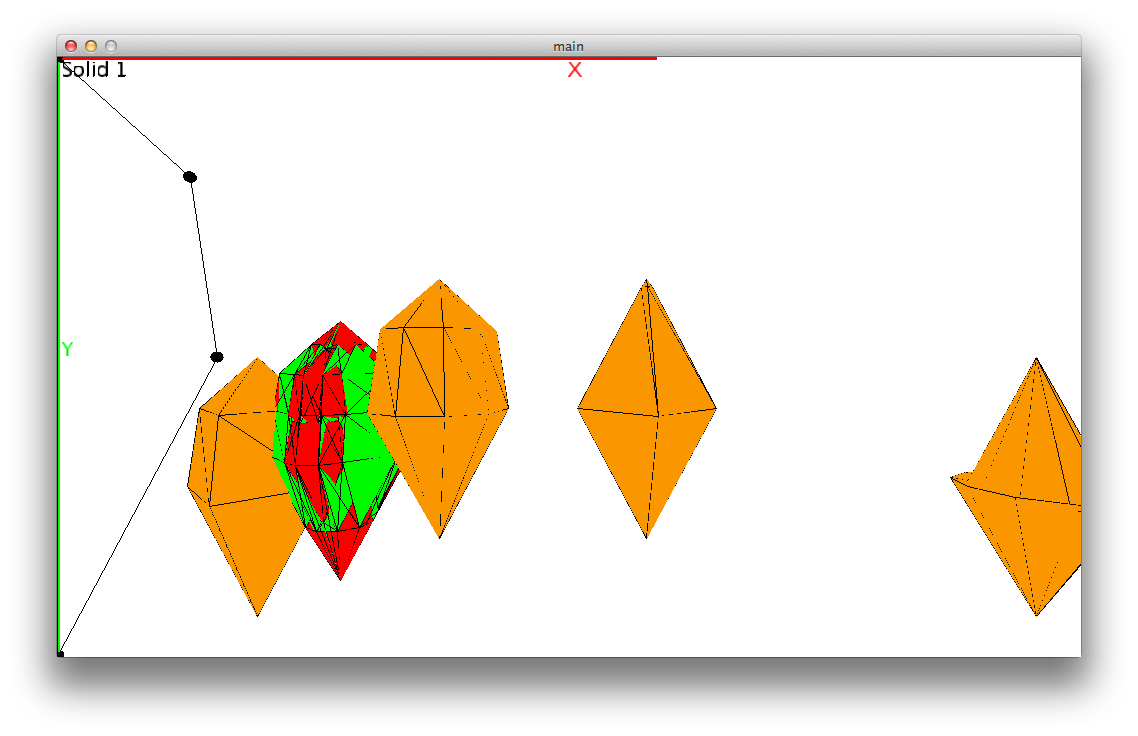
1. Drag mouse / press key:
   1. ‘a’ = Drag/edit point on profile
   2. ‘o’ = Increment x comp of direction vector of the current solid
   3. ‘p’ = Increment y comp of direction vector of the current solid
2. On mouse release:
   1. ‘d’ = Deletes point closest to mouse on profile
3. On key release:
   1. ‘i’ = Insert point next to the nearest point on profile
   2. ‘1-4’ = Select between solids 1-4
   3. ‘C’ = Makes the profile convex if it isn’t already
   4. ‘,’ = Decreases the K value for the current solid
   5. ‘.’ = Increase the K value for the current solid
   6. ‘g’ = toggle to show triangle normals or vertex normal
   7. ‘y’ = toggle between manual or automatic time steps
   8. ‘u’ = increment t when ‘a’ is toggled (reset if t>1)
   9. ‘W’ = Save the states of all four solids
   10. ‘L’ = Load the states of all four solids
   11. ‘m’ = switch between mesh solid and sweep solid



Implementation details:

Part A:

Our implementation used professor Rossignacs’ Curve, PV, col, help, Mesh, and pick files. We re-used our project 4 implementation of a curve to use its methods relevant to editing. We then made it so that the curve or profile stretched the screen height to give the user more flexibility and ease. The vertices are clearly marked on the profile and the user may insert and delete points. We also made sure that at least one point excluding the first and last existed to keep the integrity of the solid. Once we had an editable profile, we went ahead and created our own Solid class which contained our Direction axis, K value, solid ID for our own reference, original un-scaled profile, a scaled profile, the mesh for the solid, and the solids’ position. We added various methods to draw the profile sweep using the sweep algorithm, as well as methods to draw and calculate our mesh. The orientation of the solid is originally defined by the direction of the axis. In order to do this, a plane that is normal to the vector is calculated from the scaled profile. In order to find the normal plane, take the vector from the starting point of the profile to the current point and subtract the projection. This will give the j vector in which to align the solid. Then, take the cross product of the j vector with the vector from the starting point to the ending point of the profile. This gives the i vector that forms a 90 degree angle, which with the j vector forms the normal plane. Then, calculate where point in which the point of the profile will be rotated around. To create the mesh, we first calculated the sweep and stored all the sample profiles in an array called profiles in the Solid class. We then looped through the profiles and its vertices and added triangles to the mesh. When we encountered the second to last vertex and first vertex we knew we immediately had triangles and added them to our mesh. Everything in between formed quads that we had to split into two coplanar triangles. We had to be careful in creating our mesh so that we don’t add repeat vertices. We used several data-structures to keep track of vertex id’s of the previous profile. After creating our mesh, we simply drew them onto the screen. For the Minkowski morph, we matched the triangles of s1 to the vertices of s2 and vice versa. We then matched the edges of s1 to the edges of s2. We then interpolated these constructs to create our morph from t=0 to t=1. To help maintain the readability of our code and main file, we tried to abstract our code as much as we could and keep our logic within our range of classes. We used the PeasyCam library to help use debug and develop our code for part A from the beginning. It allowed us pan around and rotate our view of our solid/profile to verify that our code was working and allowed us to speed up our development process. \*The download link to this library can be found in the references section below in this document. To install, unzip and add the peasycam folder into your Processing/libraries folder.

* Eliminating Concavity
  + Our algorithm loops through the points of the profile. For each point, we compute the edge between the current point and the next point as well as the edge between the next point and the point after that. Then we compute the cross product of the two edges to see if it’s a ‘left turn’. This is the basis of our algorithm. However to consider the case in which we have 2 or more consecutive vertices that contribute to the concavity of the profile, we basically loop with the above code separately. So two separate loops.
* Mid course situation 2 solid morph w/ missing quads
  + 
* Mid course situation 2 solid morph showing only blue quads
  + 