CS 770/870
Assignment 5 v. 2: Translational sweep surfaces

November 26, 2014

Due Dates: **Alpha submission:** 12/7 midnight; **no late days without prior approval.** Style counts
Beta submission: 12/14 midnight; **no late days without prior approval**; 90%; no style
Final submission: 12/18 midnight; **no late days without prior approval**; last 10%; style counts

Main goals: Implement a "sweep" surface child of *Object3D* that supports parametric texture mapping.

Sweep surface object

You should implement a *SweepSurface* class that inherits from *Object3D.* The constructor(s) for this class and/or setter methods should support the definition of the following parameters:

1. an array of 2D vertices defining a planar polygon whose interior should include the origin;
2. an array of 3D vertices defining a path in space through which the polygon is "swept" in order to create a solid object;
3. an array whose length matches the path length and whose elements are *pairs* of *float* values that define *x* and *y* scale factors that are applied to the polygon defined at that step of the path;
4. an array of *float* values whose length matches that of the path array; each value specifies a rotation in degrees to be applied to the original polygon to orient it in the plane defined at that step in the path.

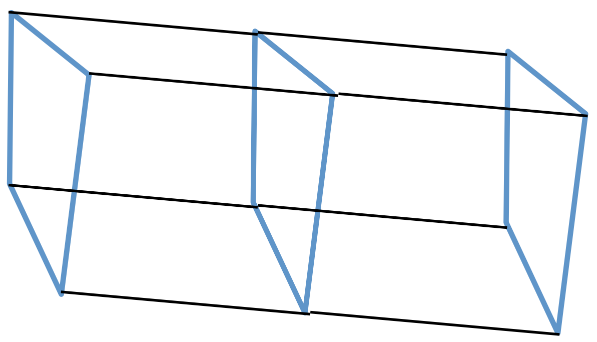
These parameters define a series of planes on which you will place the vertices of the polygon shape. You then need to connect the corresponding vertices of each consecutive pair of polygons (now in 3D) with a line and then define two triangles between each *pair* of neighboring points and their connecting lines.

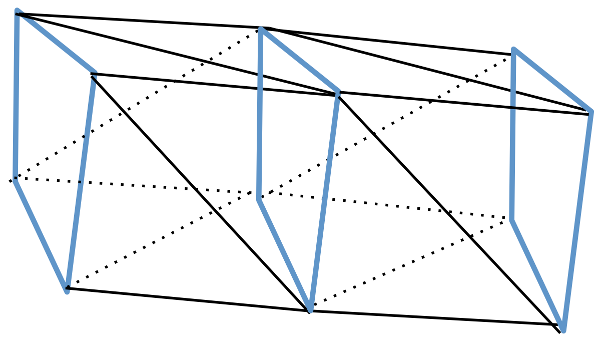
Placement of polygons

Let V_k represent the k-th vertex on the sweep path where k=0, …, path.length
Let P_k represent the k-th polygon where k=0, …, path.length.

- The initial polygon (P_0) is in the plane parallel to z=0 that contains the point V_0. The "center" of the polygon should be at V_0. This will be simple translation of the points that define the polygon by (V_{0x},V_{0y},V_{0z}).
- Polygon, P_k has its center at V_k and it lies in the plane whose normal is V_{k-1}-V_k. Hence the outer normal to plane *k* points towards V_{k-1}.
- The vertices of the original polygon get scaled in 2D by the values in the kth entry of the scale vector.
- The *up* direction of P_k is determined by the rotation angle specified in the rotation vector.
- These transformations can be defined as a *coordinate frame*. That is the description of a coordinate system for step k defined in the coordinate system of the original polygon.

Triangulation





With just a little care (and reading about them), you can make a single triangle strip for all the triangles between 2 neighboring polygons. A triangle strip let's you specify each vertex in the strip only once since each new vertex (after the first 2) defines a new triangle along with the previous 2 vertices. Without special treatment the simple calculation of the normal to the triangle would alternate outward normals. However, OpenGL knows it is a triangle strip so it reverses every other normal.

Transformations

You must support the TRS transformations common to all *Object3D* classes as well as the color specs – although one color value is enough for the entire surface.

Texture support/parametric coordinates for surface

Your sweep object should support textures and must provide an interface that allows the application programmer user to specify the mapping of the texture parametric coordinated (*r,s*) to parametric coordinates defined by the sweep surface (*u,v*). You can create a parametric coordinate system for the sweep surface by defining the ***u* coordinate** based on the points in the path array (the centers of the polygons). V_0 is u=0, V_n is u=1 and the other vertexes are mapped to values in (0,1) that are proportional to their positions along the center line. In other words, let d($V_{n-1,}V_n$ = distance($V_{i-1,}V_i$) and let L be the sum(d_i) for i=1…n. Then the *u* parameter value for V_k =[sum(d_i) for i=1…k] / L. By this definition, every vertex in polygon k has this same *u* value. You can do a similar thing for the **u** coordinate, but do that around the perimeter of the polygon. In this case the *j*-th vertex of all polygons will have the same *u* value.

Backward compatibility

The user must be able to step through a **scene sequence** that clearly shows the different features of your program that are working and meaningful **scene labels** displayed for each scene. You must also support the interactive global **scene transformation**. You must have enough **light functionality** that the shapes of your sweep objects can be readily seen. You do not need to support interactive light changes.

Last 10 points (for beta/final submissions)

As a small concession to the idea of a "project", you may choose how to earn the last 10 points for the assignment. These points will be awarded based on the complexity, quantity, and implementation correctness of the feature(s) you choose to add. The points will be awarded on an exponential scale: the first 3 will be relatively easy to earn; the next 4 will be harder; the final 3 will be very hard. An obvious candidate is to implement your code using GLSL. I will consider this to be relatively easy, which it will be if you get P4 working. Other candidate features include particularly interesting objects, particularly good scenes, additional interaction features, scene/object input from a file, interactive definition of the base polygon, etc. If you want feedback on ideas you have, just ask, but I won't associate a specific number of points with a specific feature.

Notes

1. You should create some instances of your sweep object that have textures and some without. It's easier to see the 3D shapes of these objects with just lighting on simple colored surfaces.
2. As usual, your scenes should show us that all features of your program work (or at least show us clearly which ones **do** work). Please set up your scenes to explicitly address the rubrics.
3. You should have a check box that allows the user to interactively see the objects drawn only with lines or with lighting/textures, but you also should be able to override the current check box value in each specific object in order to build scenes that demonstrate your functionality. In other words, you should be able to create an instance of your sweep class that is **always** drawn as wire frame and have a scene that shows two instances next to each other that only vary by their position; one of those would have its wire-frame only flag set. The alpha submission is almost exclusively based on these features.
4. Your object implementation should allow you to vary the different inputs such as the number of vertices in the base polygon, the rotation and scale parameters, etc. You do not need to perform validity tests on the specifications (such as convexity, origin in interior, etc.)

Alpha submission point allocation

20	Overall framework with scene sequences with meaningful labeling; scene transformation.
30	***Wire frame*** objects with path defined as a series of steps along the *z-axis*. All planes have same normal, no scale, no rotate.
10	Scale feature added
10	Rotation feature added
30	Path has arbitrary point locations. (Of course, not all paths will make useful objects!)

Programming style **will** be evaluated. You do not earn points for good style, you lose them for poor style.

Alpha submission: Submit as *A5* including ***Makefile***, with either MAIN or PROG set to your executable.

Beta submission point allocation:

20 All components of alpha submission completed.

20 "Skin" of the swept object is rendered and clearly visible as flat-shaded triangles. Basic lighting needs to work for this to be clearly visible. You do not need to have interactive lighting changes, but you might want to have lighting changes built into your Scene object so that you can still show the same set of objects in 2 different scenes with different lighting. This option uses the normal to the triangle as the normals to the vertices of the triangle.

20 Skin of the swept objects uses shared normals so that smooth shading across the shared edges makes the surface looked curved without the edge being visible (or at least not prominent). This requires that each vertex have a normal that is the average of the triangles/polygons that share that same vertex.

30 Texture mapping: scenes should show a variety of mappings to your swept objects using different mapping parameters. This does not need to be so extensive as was required for P3; the goal is to show that you have implemented the surface parameterization effectively.

Programming style will **not** be evaluated.

Submit as B5: Be sure to include a working ***Makefile***.

Final submission point allocation:

10 The last 10 points of the Beta. Your choice. As mentioned in class, a good implementation of triangle strips for your skin will be worth up to 3 points. Covering your ends will also be a candidate for the last 10. Just remember, that the 10 points get progressively harder to earn, so these 2 relatively straightforward options are worth less as a second feature than as the first or only. In order to get points for this you must submit a *readme.txt* that describes what you have implemented that you would like us to consider for the last 10 points.

Programming style **will** be evaluated. You do not earn points for good style; you lose them for poor style.

Submit as F5: Be sure to include a working ***Makefile*** and a ***readme.txt*** (no other variations) that describes what you want considered for the last 10 points.