Using Mathematica and Rhinoceros to produce 3D printed mathematical models - workshop notes

Henry Segerman

segerman@unimelb.edu.au

Abstract

This workshop will be a hands-on introduction to producing physical 3D printed mathematical models using computer software. 3D printing is rapidly becoming a very affordable way to produce physical objects, for use in outreach, teaching or research. In addition to being excellent visualisation aids, physical objects go further, allowing for a tactile understanding. Depending on the interests of the participants, we will use Mathematica, the 3D design program "Rhinoceros", and/or the Python scripting interface to Rhinoceros to produce 3D files ready to be sent to a 3D printer. The workshop will be based in part on this article: `http://www.ms.unimelb.edu.au/~segerman/papers/3d_printed_visualisation.pdf` see also `http://www.segerman.org/3d_printing_notes.html`.

1 3D printing file format

The most common format for sending a model to a 3D printer is the STL ("Standard Tessellation Language") format. This consists of a collection of triangles (a *mesh*), each triangle given by the coordinates in 3-dimensions of its three corners. Current 3D printers take this list of triangles and interpret their union as a closed orientable embedded surface in $\mathbb{R}^3$, from which they can determine which voxels (3-dimensional analogues of pixels) are inside the surface (and so should be solid, filled with plastic or whatever material the printer is using), and which are outside the surface (and so should be empty).

2 Mathematica

Open the example notebook "make_parametric_surface_grid_stl.nb". It has the following code:

```
f[u_, v_] := {u, v, u^2 - v^2};
scale = 40;
radius = 0.75;
numPoints = 24;
gridSteps = 10;
curvesU = Table[scale*f[u, i], {i, -1, 1, 2/gridSteps}];
curvesV = Table[scale*f[j, v], {j, -1, 1, 2/gridSteps}];
tubesU = ParametricPlot3D[curvesU, {u, -1, 1},
   PlotStyle -> Tube[radius, PlotPoints -> numPoints],
   PlotRange -> All];
tubesV = ParametricPlot3D[curvesV, {v, -1, 1},
   PlotStyle -> Tube[radius, PlotPoints -> numPoints],
```

```
   PlotRange -> All];
corners =
  Graphics3D[
   Table[Sphere[scale f[i, j], radius], {i, -1, 1, 2}, {j, -1, 1, 2}],
    PlotPoints -> numPoints];
output = Show[tubesU, tubesV, corners]
Export["MathematicaParametricSurface.stl", output]
```

- Left click on the code then Shift+Enter to run it.
- Left click on the output image then hold left click and drag to rotate.
- Line by line details on the code are given in "3d_printed_visualisation.pdf".
- Try changing the parametric formula in the first line.
- The code exports the file "MathematicaParametricSurface.stl" to the root directory.
- Some, but not all Mathematica 3D graphics objects export properly. In addition to ParametricPlot3D with PlotStyle → Tube and Sphere, Cylinder also works.
- The exported STL file is ready to upload to a 3D printing service as is, but if we want to check the file first we need to use a dedicated 3D program, for example Rhinoceros.

3 Rhinoceros

To get Rhinoceros, see `http://download.rhino3d.com/rhino/5.0/evaluationtimed/download/` for a trial version of Rhino 5. A Mac beta version is also available from `http://mac.rhino3d.com/`. The full Windows version is available at $195 for an educational license. Note that the Mac version is still under development, so some of the instructions listed below may be out of date, or the commands may be in a different place.

3.1 Navigating and viewing a file

- Open the Rhinoceros program and do File → Import, then choose the "MathematicaParametricSurface.stl" file.
- The object is shown from four views, three orthogonal views, the top, front and right views, and a perspective view.
- Shift+hold right click and drag on any of the windows to translate that view.
- Scroll wheel on any of the windows zooms the view.
- Hold right click and drag to rotate the perspective view.
- View → Wireframe, or Shaded, Rendered etc. changes the view style.
- Try changing a viewpoint to shaded mode, then zoom in to see the polygons.

3.2 Building an object: right angle tetrahedron

- File → New or Command+N to start a new file.
- We will be building the edges of a tetrahedron with vertices at $(0,0,0), (20,0,0), (0,20,0), (0,0,20)$.
- First turn on snap to grid, using the button "Snap" just above the top left viewpoint window.
- Build the three lines along the axes first, using the orthogonal viewpoints. Use Curve → Line → Single Line, then left click on one of the viewpoints at the start and endpoints to define the line. (Space bar repeats the last used command, which may be useful here.)
- Now turn on Object Snap, using the button "OSnap" at the top left of the screen, then check the "End" box in the "Object Snaps" window that appears.
- For the other three lines, use the perspective view. When drawing the new lines, the mouse will snap to the endpoints of the existing lines.
- To correct an error, either Edit → Undo, or Command+Z to undo, or alternatively left click to select an object and Backspace to delete. Left click and drag a box to select multiple objects. Shift+left click adds objects to the selection and Command+left click removes objects from the selection.
- Other things to try: Transform → Move, Transform → Rotate, Transform → Scale → Scale 3-D. Follow the instructions in the pop-up window that appears.

We now need to thicken these lines up so that we can print the tetrahedron.

- Edit → Select Objects → All Objects or Command+A to select all of the lines.
- Solid → Pipe then Enter to make pipes around the lines (of radius 1, this can be altered in the pop-up window).
- (Optional) use Solid → Sphere → Center, Radius to put a sphere at each of the vertices of the tetrahedron, to round off the corners.

Rhino uses *NURBS surfaces* as well as meshes to describe surfaces. ("NURBS" stands for "Non-uniform rational B-spline". NURBS surfaces are generalisations of Bézier curves.) The pipes and spheres we have at the moment are NURBS surfaces, but we need to convert them to meshes before sending the data to a 3D printer.

- Select all with Command+A.
- Mesh → From NURBS Object then Enter to convert the surfaces to meshes.
- The meshes and the NURBS surfaces now occupy the same space. To see only the meshes, you can either move them away from the NURBS surfaces, or put them on a different *layer*, using Window → Show Layers Panel and the layer tools accessed from the panel. For either strategy, use Edit → Select Objects → Polygon Meshes (or similar) to easily select one kind of object.
- To export the mesh as an STL, select only the polygon meshes, then File → Export Selected. Choose a filename, and then click Export.

4 Rhinoceros and Python

To run python scripts in the Mac version of Rhino, you will need to install a plugin, instructions here: `http://wiki.mcneel.com/rhino/mac/python`.

- Command+N to start a new file.
- To run a python script, type "RunPythonScript" (just start typing, there is no text box you are writing in; also the command should autocomplete after you type a few letters) then Enter. Choose "hyperbolic_paraboloid.py". This will draw a small hyperbolic paraboloid with lines.
- Open "hyperbolic_paraboloid.py" using a text editor to see the code.
- The function "rhino.AddLine" in the script is one of many functions used to interface with Rhinoceros, see `http://www.rhino3d.com/5/ironpython/index.html` for documentation on other functions.
- Select all the lines, then type "NetworkSrf" (again it will autocomplete after a few letters), then click Ok. This produces a NURBS surface from the grid of lines. My usual strategy is to build complicated shapes out of NURBS patches defined in this way. You may need to change the viewpoint from "Wireframe" to "Shaded" to see the surface properly.
- The python script "hyperbolic_paraboloid2.py" does the NetworkSrf function itself, using an imported utility script also in the folder, called "networksrf.py".

5 Joining

- Use Transform $\rightarrow$ Mirror to reflect the surface across the line $x = 1$, making a copy of it (you have to draw the mirror plane, use the top viewpoint to do this).
- Now select both NURBS surface patches and Edit $\rightarrow$ Join. This joins the two surfaces together along the edge they meet at.
- With the joined surface selected, do Surface $\rightarrow$ Edge Tools $\rightarrow$ Show Edges, then click the "Naked Edges" button in the pop-up window. This highlights the boundary ("naked") edges of the surface. To make a surface that we can turn into a mesh and then print, all edges or surface patches must be matched up properly to give a closed surface.
- With the joined surface selected, do Edit $\rightarrow$ Explode. This is the opposite of Join, and you should see that the edges between the two surface patches are now naked.
- When you Mesh a surface made from NURBS patches that are properly joined together, Rhino produces a mesh that is connected across the join. Without proper joins, the mesh surface will think it has non-empty boundary, and the 3D printer will not know what is inside and outside of the mesh. To check the status of a mesh, do Edit $\rightarrow$ Show Object Properties Panel, and then select the mesh. In the Details tab, under the Geometry heading, it will tell you if the mesh is open (with non-empty boundary) or closed. Surface $\rightarrow$ Edge Tools $\rightarrow$ Show Edges also works for meshes, to tell you where the mesh has boundary.

6 More information

The materials from this workshop are available at `http://www.ms.unimelb.edu.au/~segerman/EViMS_3d_printing_files.zip`. See `http://www.ms.unimelb.edu.au/~segerman/papers/3d_printed_visualisation.pdf` and `http://www.segerman.org/3d_printing_notes.html` for further information!