determines four polygonal chains bounding the sub triangulation: left, right, bottom, top. For each sub, we consider the chains in (α, θ) space. In this space, the chains are straight sides of the trapezoid. We triangulate the interior in a standard manner respecting the boundary vertices and then transform back into 3D. There is no duplications of vertex points: triangulation vertices that are shared by two subs are actually indices into a common point array. This way, the set of triangles from all subs forms a triangulation of the 3D free space boundary with no gaps.

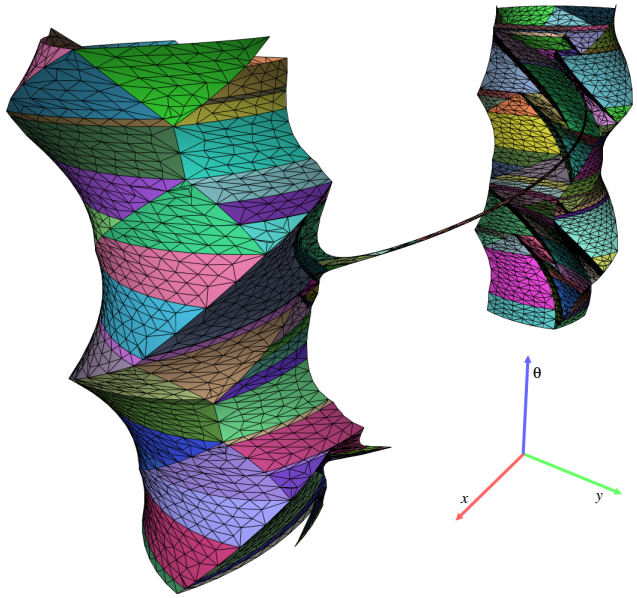

Figure 1: A triangulated approximation to a free space boundary viewed from outside. Each sub is drawn in a unique color, and a black wireframe is overlaid to show the triangles. The two vertical shafts indicate larger spaces the robot can translate and rotate within; the narrow tube connecting the shafts illustrates that the robot can move between these areas with restricted translation and rotation.

3. VIDEO OVERVIEW

Our video begins by introducing the 2D scene with the robot and obstacle, and demonstrates how the robot can be translated and rotated by the user. We then describe the parts that make up the complete representation used to model the configuration space. The 3D visualization follows, and a side-by-side comparison illustrates the relation of the 2D and 3D geometry.

We highlight how the user can configure the robot and see its corresponding 3D point in the configuration space. A brief summary of the triangulation algorithm is provided, and we conclude by showing path planning in the configuration space. Some additional scenes demonstrate the robustness of the software with narrow configuration space channels. We also show how linear robot and obstacle edges can be approximated in practice using arcs with very large radii (10^6).

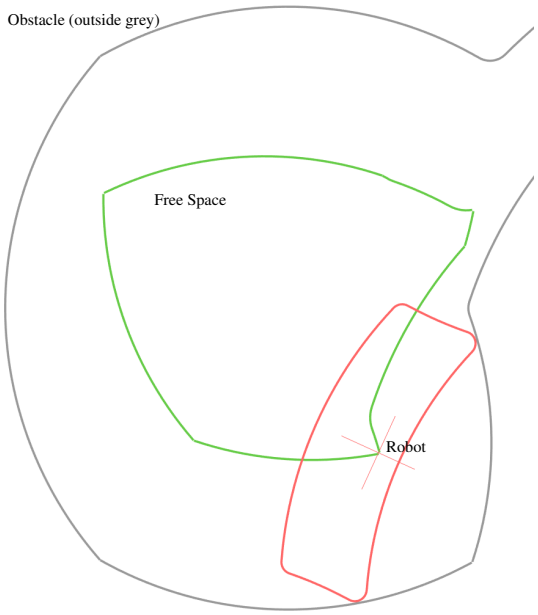

Figure 2: Visualization of the cross section of the free space for a fixed orientation of the robot. Since the robot (red) is at the junction of two sub edges (green), it has two contacts with the obstacle (gray).

4. SOFTWARE DETAILS

The software in this video was written in Java using the OpenGL graphics API through JOGL[3]. GLSL shaders are used for rendering effects, and YAML is used for configuration files. The visualization software[4] can be experimented with and is bundled with some sample scenes; the software that produces the input, described in [2], can be downloaded[5] as well.

5. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 0904707. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

6. REFERENCES

[1] D. Atariah and G. Rote. Configuration space visualization. In *Proceedings of the 2012 Symposium on Computational Geometry*, pages 415–416, 2012.

[2] Victor Milenkovic, Elisha Sacks, and Steven Trac. Robust complete path planning in the plane. In *Proceedings of the Workshop on the Algorithmic Foundations of Robotics*. WAFR, 2012. Invited version to appear in IEEE T-ASE.

[3] `http://jogamp.org/jogl/www/`
[4] `http://jstoecker.github.com/cspace`
[5] `http://www.cs.miami.edu/~vjm/robust/`