Frederick Doering

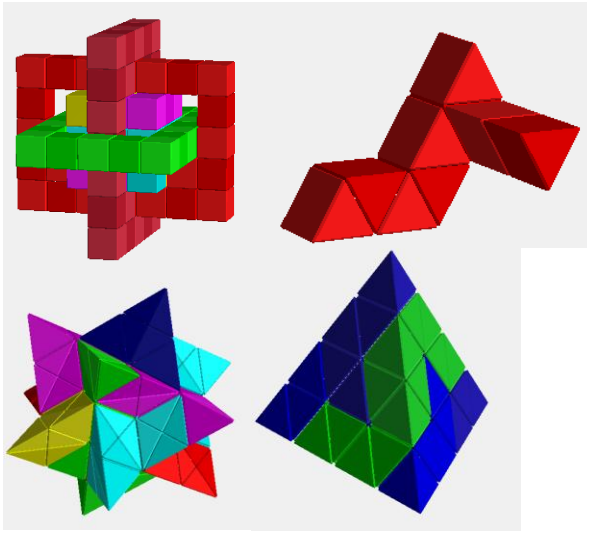
CS 285: Solid Modeling

Final Project Write Up: Burr Puzzle Assistant

15 December 2011

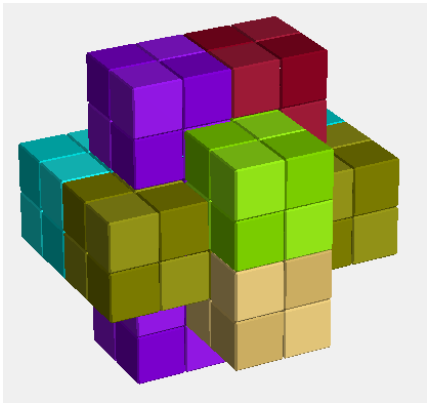
Burr Puzzle Assistant

Introduction:



Burr puzzles consist of a set of interlocking pieces. The pieces slide linearly along a set of axes. To facilitate these sliding moves the pieces are composed of voxels, which are defined by sets of equally spaced planes that are parallel to each pair of these axes. In the case of three orthogonal axes, these voxels are cubes. There are three other common axes. First there are three axes offset sixty degrees from each other and perpendicular to a fourth, producing equilateral triangular prism voxels. The other two types are rhombic-tetrahedra, and tetrahedral-octohedra, pictured above on the bottom left and right respectively. Burr puzzles are interesting for the casual puzzler, puzzle enthusiasts, and mathematicians.

Previous Work:



Mathematician Bill Cutler intensively studied what is commonly referred to as “six-piece burrs.” In a “six-piece burr” the six pieces each are two by two by eight in size and the always assemble to form the shape shown on the right.

Mr. Cutler analyzed all 35 billion possible puzzles using a computer program he helped develop. In doing this he discovered the most complex puzzles which are now sold to puzzle enthusiasts. His experience shows that computer programs are both useful and necessary for designing leading edge burr puzzles.

Project Goals:

For this project, my goals were fourfold. First, I wanted to create a program that could disassemble burrs. I also wanted the program to have a nice graphical user interface (GUI), so that it would be more appealing to end users. Third, I needed the program to rate burr puzzles, so I wanted to design an algorithm to do so. Lastly, the program would need to automatically generate piece geometry to determine disassembly and rate, so it could automatically generate burr puzzles, in addition to solving them.

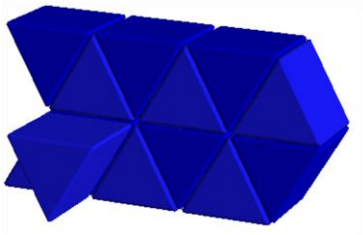
In order to achieve these goals within the semester, I found open source project called BurrTools that accomplished the first two goals. I was able to achieve my third goal by rating puzzle difficulty by the minimum number of moves necessary to fully disassemble the puzzle. This left the fourth goal: piece generation. This became my focus for the project.

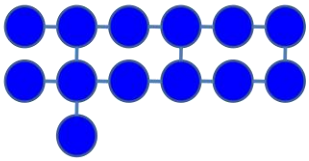
Burr Puzzles:

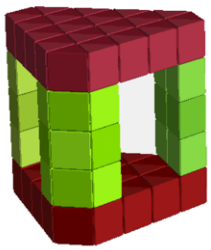
There are two types of burr puzzles. One type consists of pieces that are all of the same dimensions, but that may have different internal structures. Additionally, this piece size is much smaller than the entirety of the assembled structure. The second type, which I have designated as "dissection" puzzles, differ in that the pieces are of seemingly random shape and may span the size of the main object. Since the two puzzles differ greatly in how they are designed, I chose to focus on dissection puzzles.

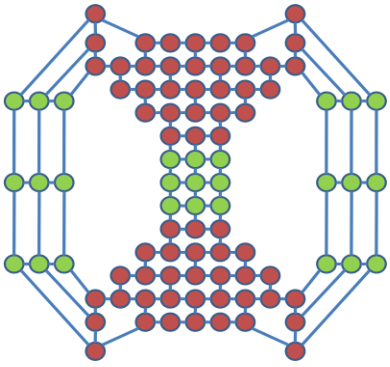
There are several requirements for automated piece generation. The obvious first two, are that each piece must consist of a set of connected voxels, and that these pieces must be able to assemble and disassemble. I also posed some design requirements. After the user specifies the final shape, the pieces must assemble to fill the voxels specified as required, and only fill the voxels specified as allowed. I also let the user specify how many pieces the final shape should be dissected into, and what the minimum and maximum number of voxels in each piece should be. The puzzle difficulty requirement was met by only keeping one puzzle of each difficulty level. This would allow the user to select the difficulty of their choice without having to know beforehand what difficulty levels were possible, and without storing too much data. Finally the last requirement is that the program needs to run in a reasonable amount of time. This last requirement means that an intelligent approach to piece generation needs to be implemented. For example, in the case of a four by four by four cube, with four pieces, if a brute force method of assigning each voxel to a piece and checking the requirements was used, then the requirements would need to be checked over 10^{39} times. This is an infeasible task with current or projected future computational power. Thus I designed a piece generation routine to actively consider these requirements.

The requirement that pieces consist of connected voxels is achieved by generating pieces one voxel at a time and only adding neighboring voxels. First, a method of quickly finding neighbors is created. I did this by creating a graph, where each node represented a voxel that was allowed in the final shape. Then, each







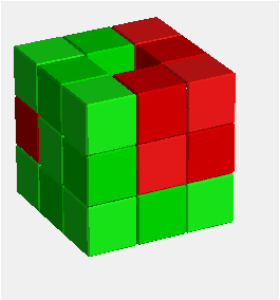


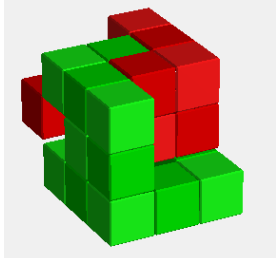
node had a list of pointers to neighboring nodes. Two shapes with their respective neighbor graphs are shown on the right. These graphs allowed for a generalized algorithm that would work on any voxel type. Additionally, by constructing these graphs only for the voxels that the user specified as allowed to be in the final assembly, that requirement is taken care of. The piece generation by adding a single voxel at a time, naturally leads to a recursive depth first search algorithm for generating the puzzles.

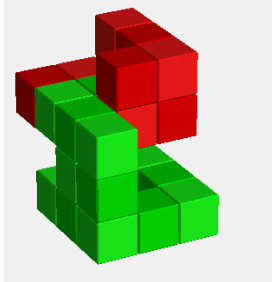
The requirement that the pieces be able to disassemble allows the depth first tree search to backtrack as soon as the puzzle becomes impossible to disassemble. Unfortunately, the computational cost of determining if a puzzle can disassemble is quite large. Thus in the algorithm I implemented, it only starts checking for disassembly once the pieces have the minimum required number of voxels. The frequency with which the puzzle would not disassemble before that point would not be high enough to make checking worthwhile.

Results:

After many efficiency improvements to my code, such as keeping a shared linked list of the voxels whose neighbors were untried, I was able to run the program on the disassembly of a three by three by three cube. I spent several days trying to design a difficult of a two piece direction by hand. The most difficult puzzle I came up with only took two moves to disassemble. The burr puzzle assistant was able to find several puzzles that took three moves to disassemble. Shown below are the steps to disassemble one of these puzzles. This shows that burr puzzle assistant is successful in assisting in the design







of burr puzzles. Unfortunately, it took three days to find this puzzle. There are still many improvements that could be incorporated to help make this program more practical. The algorithm to disassemble could use bitwise operations for integers on arrays representing the pieces to help parallelize the collision detection. Another possibility is to interactively show solution paths as a designer is editing puzzle pieces. This would allow a user, with great intuition, to quickly hone in on good puzzles. Additionally, with enough study of puzzle design there may be iterative improvement techniques that could be applied to hone in on good puzzles programmatically. Overall, I think this project was successful in that the program met the goals I set out to meet, and I learned a lot about programming, graph theory, search optimizations, and the math behind burr construction while working on it.