CS285 Final Project: A-Mazing Cube

Ayden Ye*
aydenye@berkeley.edu

Jonathan Ko*
jonathan.ko@berkeley.edu

Dec 15, 2011

1 Introduction and Motivation

Our CS285 final course project is called the 'A-Mazing Cube.' As its name indicates, the idea is to procedurally design and model a cube with mazes covering its six faces.

Our motivation for this project is to implement several important concepts that we learned from this course, including image-based mazes, procedural modeling in 3D, and creation of geometry from images. We aim to design an end product that is aesthetically pleasing and enjoyable and challenging to solve.

Figure 1: Final model of the A-Mazing Cube

*University of California, Berkeley

2 Design

Our inspiration for the A-Mazing Cube originates from several concepts. The first is the image-based maze, which can be programmatically generated from digital images, or manually generated in cornfields and gardens and viewed from a bird's-eye view. The second idea is based on the concept of the wormhole, a hypothesized topological feature of space-time fundamentally equivalent to a shortcut between two points that is popularized in science fiction. The third inspiration is to create a fun toy with which people can directly interact. We chose to incorporate the concept of the ball-in-labyrinth puzzle, a two-dimensional game you may have seen in toy stores or as a mobile application.

2.1 Image-Based Mazes

Constructing paths based on the patterns in a given image allow greater aesthetic control over the large-scale appearance of a maze. Figure 2 shows examples of this approach taken in real life in cornfields and gardens, and in the digital world via previously developed algorithms that procedurally generate mazes from an input image, such as the work of Xu, et al. and Wan, et al. [4, 6]. These mazes bear unique artistic features and their routes can also be excitingly tricky to navigate. Since we want our final work to have both aesthetic and entertainment value, we begin with already-designed image-based mazes and extend with more complicated structures.

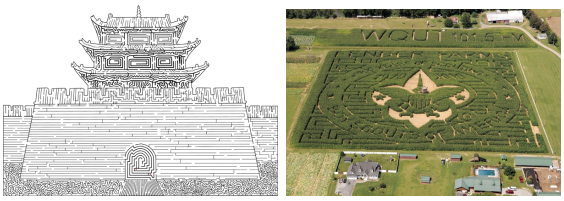

Figure 2: Left: Image-based maze of the Jia Yu Guan pass of the Great Wall of China[6]. Right: Bird's-eye view of a cornfield maze in Fender's Farm, Tennessee.

2.2 Wormholes, Tunnels, and the 2.5D Maze

Our second idea is inspired by the concept of the wormhole, common in science fiction and proposed in theoretical physics. Wormholes act as a connection between two points in space, and enable an object to move directly through a shorter path to reach another seemingly further target (Figure 3). Adding wormhole-like structures to our maze provides us with two distinct advantages.

First, we do not have to worry about how the player's ball will cross from one cube face to another. Wormholes will originate at arbitrary locations within each face, and jump between such locations on pairs of faces, creating a graph-like structure of face-nodes and wormhole-edges. Then each face may, like its source maze image, be self-contained with a closed border, and placed independently on any cube face; without such a mechanism, we would need to carefully construct and align wall openings across cube edges, and stick to a single face layout. The final product will be a connected graph, with each face a 2.5D maze (two-dimensional with entrance/exit locations).

Second, this approach adds to the entertainment value of the final product. Where conventional mazes require traversal of the entire face at once, our construction requires the player to make the jump after only partial navigation of a face.

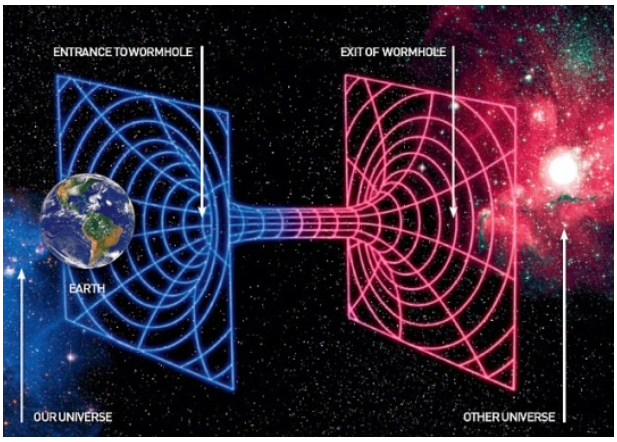

Figure 3: Conceptual structure of a wormhole between two distant galaxies

2.3 Fabrication

In addition to constructing a CAD file of our maze, one of our chief goals is to produce a tangible toy for people to physically manipulate. As previously discussed, our design is to localize six different two-dimensional mazes onto the faces of a cube, and install a number of inner tunnels through the cube body connecting various pairs of faces. We plan to print the maze floor and walls on the FDM machine in Etcheverry Hall, connect them via plastic tubing from a local hardware shop, and encase them in acrylic from Tap Plastics. By doing this we have a ceiling and floor constraining a small metal ball to the surface of the cube, and players may orient the cube to try to lead it through a designated opening in one acrylic outer face of the cube. Our design results in a playable toy with which people can compete or set self-challenges, and will be a piece of artwork with both attractive image-based qualities and a certain degree of trickiness and complexity. The parts, quantities, and prices needed to produce our final product are listed in Table 1, and a preliminary design sketch is shown in Figure 4.

Part	Quantity	Price
Acrylic Squares	6 sheets	$25
Plastic Tubing	8 feet	$1.52
Maze Fabrication	6 faces	—

Table 1: Parts and costs for the A-Mazing Cube

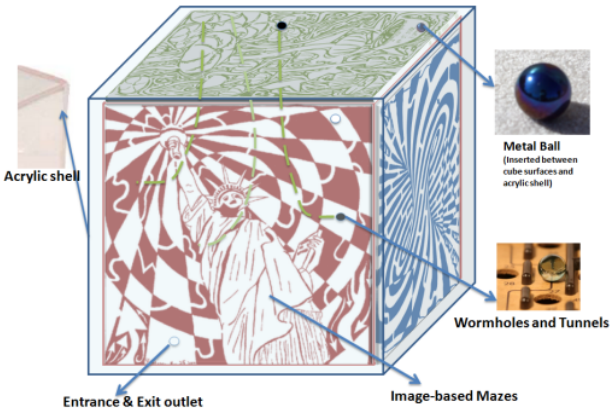

Figure 4: Design sketch for the A-Mazing Cube

3 Implementation

We use several existing software tools and one in-house program to generate our CAD files. The process is summarized as follows:

- Select and obtain six maze patterns, and refine them for suitable navigation by the puzzle's metal ball.
- Convert the maze image into a polygon description to be imported into a CAD tool.
- Extrude walls into 3D, refine geometry to create watertight closed two-manifold b-rep.
- Process and prepare model for production in fused deposition modelling (FDM) machine.

3.1 Design of 2.5D Maze Topology

Before designing the actual maze, we take time to design the topology of the final maze, consisting of six interconnected 2D mazes (face mazes). We want to equip the resulting 2.5D maze (cube maze) with enough complexity that the player must travel back and forth from each surface multiple times before reaching the exit. To achieve this, we divide each face maze into four regions, where each region has one entrance and one exit. The first opening of a region is connected via wormhole to another same-colored region on a different face, and the second opening is connected to an adjacent region on the same face. We then iteratively design a route that performs a tour of all 24 regions, where the initial entrance and final exit will be not across regions, but through the acrylic outer surface (Figure 5).

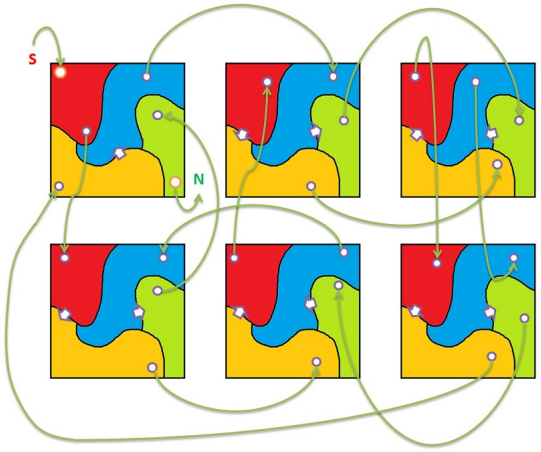

Figure 5: Design of 2.5D maze topology

3.2 2D Maze Generation

We can either generate the 2D mazes we need using algorithms described in previous works [4, 6], or directly start from existing image-based maze patterns and modify them to suit our needs. Since we want to focus more on the procedural generation and image-to-product process, we choose the latter source. Thanks to the Yonatan Frimer[1], we were

[1]Yonatan is the artist of the original mazes we chose for our cube. We would like to thank Yonatan for his great work, and for his express permission to use his images in our project

able to review hundreds of well-designed image-based mazes incorporating many different styles and topics, of which we have selected the six we consider to be most suitable for this project (Figure 6).

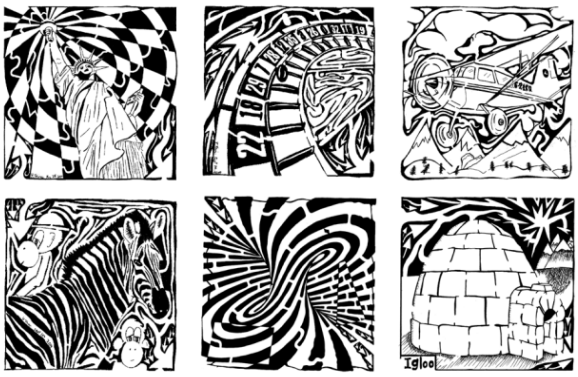

Figure 6: Six chosen maze patterns, originally designed by Yonatan Frimer.

Since these mazes were designed to have a single entrance and exit, and do not necessarily optimize for maximal path coverage, we need to hand-modify, using Adobe Photoshop, the six mazes to suit our particular puzzle design.

The first step is to divide each maze into four distinct regions. Every pathway (white areas, in all subsequent figures) should be covered by one and only one region, and every region should be completely enclosed by walls (black areas). We can easily test regions for such criteria by using the Paint Bucket Tool in Photoshop, and add additional enclosing walls using the Pencil tool.

The second step is to refine the maze path within each region. We designate one wormhole entry point and one wall segment as an inter-face connection, and refine the intermediate path, preferring constant and maximal path-coverage-to-space density and adding new or carving existing walls as necessary to achieve this effect. We also prefer a labyrinth structure (single path) over a maze (branching path), though occasionally it is aesthetically necessary to branch from the main path, so we still refer to the path as a maze; in such branching scenarios, the alternate pathways

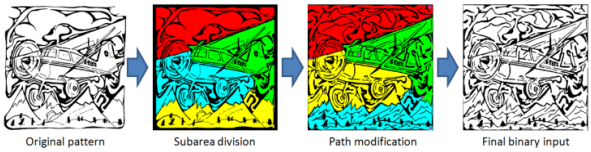

Figure 7: Four-step process to prepare arbitrary source maze for input to our pipeline.

usually regroup shortly after diverting from the main branch, or quickly meet a dead-end. After modification, we have six well-designed mazes with marked interconnections for the topology described in Section 3.1.

The third step is to take the physical dimensions into account. We rescale all images to the same pixel dimensions (1000px square), and calculate the necessary path width for the puzzle ball to comfortably navigate (which translated to about 15px in our particular construction).

The final step is to translate the designed image back to grayscale, for input into the next stage of our pipeline. The actual image format is not important, as the program will automatically convert an RGB image into grayscale, but because of the differing luminance values of the four region colors used, we decide to simply remove the colors from each region to produce the final maze image. It may also be useful to apply a Threshold operation to the image at this point, to mitigate the effects of any Brush operations that may mismatch in softness or opacity with the original maze.

The entire four-step process is detailed in Figure 7.

3.3 Pixels to Polygons

When converting our image files into 3D geometry, we came across a challenging but interesting problem: When the image is used directly as a displacement map on a regularly tesselated plane, the resulting mesh exhibits topological discontinuities with an effect that resembles jagged mountain ranges (Figure 8). This occurs only along non-axis-aligned curves, where the topology of the underlying mesh does not

match the topology of the source image.

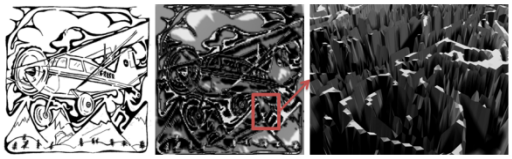

Figure 8: Topological discontinuities and the 'jagged mountain' effect.

To alleviate this disparity, we turn to the marching squares algorithm, which is the two-dimensional version of the well-known marching cubes algorithm [5]. By binarizing and examining the local layouts of pixel neighborhoods, we can construct polygons with a topology that matches the contours of the image.

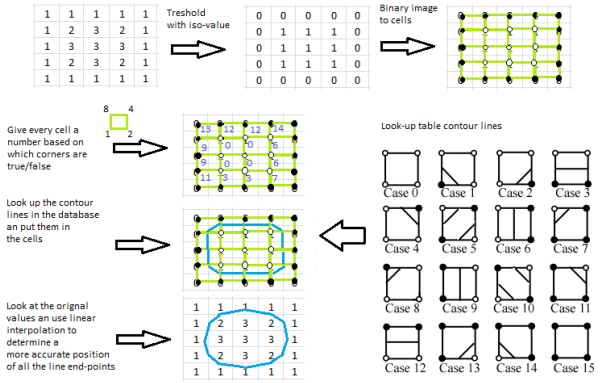

Figure 9: Overview of the marching squares algorithm. [5]

We implement the standard algorithm, as described in its Wikipedia article [5], with the additional features of linear interpolation and STL output [1]. This basic program takes an image as input, allows the user to view the generated polygons with or without linear interpolation, and export to STL. Refer to the source code for additional details.

In order to achieve smooth, aesthetically pleasing results, we apply a standard linear interpolation between endpoints for each processed pixel. Along a contour line, determined via the pixel value threshold T, the algorithm slices cells in two, where we designate the first endpoint of the slicing line as v_1 and the second as v_2. Vertex v_i is a vertex along the cell edge between vertices A_i and B_i, which are derived from pixels whose semantic values are α_i and β_i, respectively[2]. We then calculate the interpolation parameter x and resulting vertex position v_i as:

$$x = \frac{\alpha_1 - T}{\beta_1 - \alpha_1} \tag{1}$$

$$v_i = xA_i + (1 - x)B_i \tag{2}$$

This interpolation process produces polygon contour lines that better fit the isocurve at the desired threshold T.

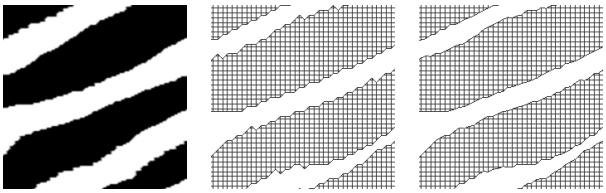

Figure 10: From left to right: Original image, polygon output without interpolation, with interpolation.

Figure 11: Left: Blur = 0. Right: Blur = 1.5.

Additionally, the user may specify a blur value applied as a prefilter before running the algorithm. This has the effect of smoothing sharp 'stairstep' edges in the output geometry, at the expense of losing some

[2]We calculate the value of a pixel to be its luminance in the source image, determined by $l = 0.30r + 0.59b + 0.11g$.

corner integrity, as is expected from blurring an image (Figure 11). The specific filter used is implemented in the CImg library, which at the time of writing is an anisotropic Canny-Deriche filter[2]. In practice, we found a blur value of 0.5 – 1.5 to be acceptable for 1000px-square images.

3.4 Wall Generation

After obtaining our planar mesh output from the marching squares program, we import into Autodesk 3ds Max 2011 for the remainder of the design process. The built-in STL import plugin provides options to Quick Weld the vertices, as well as unify normals; both of these options should be selected[3]. We apply the following series of modifiers, in order, to the mesh in order to obtain the extruded wall segment of each plate. For brevity, we will not elaborate on details, as a basic knowledge of 3ds Max should suffice to construct the model.

1. Optimize
2. ProOptimizer
3. Normal (flip)
4. Edit Mesh (manually adjust vertices)
5. Face Extrude
6. Cap Holes
7. Optimize
8. STL Check

This process can be divided into three phases. First, we optimize the input mesh to reduce its polygon count to a reasonable and workable level. Second, we make any necessary adjustments, including altering vertices, expanding walls, bridging gaps, and removing spurious small-area faces, before applying the extrusion. Last, we ensure that the mesh is closed and watertight, and verify the validity of an STL using the built-in STL Check modifier.

[3]We used a weld tolerance of 0, so as not to lose any detail. It appears to work correctly, with no visible adverse effects of roundoff error.

After obtaining the walls, we then construct a floor underneath the walls, and attach borders to the edges of the walls. The floor should be designed at a 45 deg angle to accommodate adjacent plates on all four sides, as shown in Figure 12. The twelve edge walls (one per cube edge) should form a uniform-width border around all faces, with identical thickness to the maze, and can be distributed among the six faces in any appropriate manner.

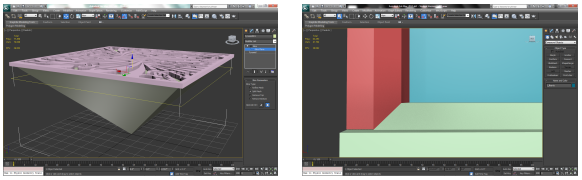

Figure 12: Left: Creation of pyramidal floor with slice plane. Right: Juxtaposition of adjacent plates.

The wormhole openings were implemented using a Tube primitive with a slight Taper effect. Instances of this opening can be applied to the plate using a Boolean or ProBoolean subtract operation, or by inverting the normals when exporting the STL to QuickSlice (which will perform its own pseudo-boolean operations on slices).

Care must be taken to ensure that the exported geometry properly conveys the designer's intentions to QuickSlice. We noted that a union effect between the maze walls and cube walls could be produced in QuickSlice by simply intersecting the two meshes, with an overlap of at least one QuickSlice road width. Additionally, wormhole openings must not intersect with any maze walls, or they will be incorrectly filled with support material.

3.5 Processing Model in QuickSlice

This stage targets the StrataSys 1650 FDM machine for actual production of the six designed plates. Once we are satisfied with the 3ds Max design of a plate, we export to STL and import into QuickSlice, following standard procedures to scale the model and generate slices, supports, base, and roads. Before finalizing, we should carefully inspect the roads to ensure that there are no missing walls (due to very thin walls

in the STL). In such cases, it is necessary to fix the problematic areas in the Edit Mesh modifier, from Section 3.4, and redo the process until a satisfying QuickSlice file is produced.

The most important concern at this stage is to target a reasonable build time per plate. Our first attempt to build a plate with a wall thickness of $^1/_8$" and floor thickness of $^1/_8$" estimated an average build time of 22.7 hours, which is not acceptable for our project. To reduce build times, we decided to experiment with thinner wall and floor dimensions, eventually settling on a wall thickness of $^5/_{96}$" (5 slices) and floor thickness of $^3/_{96}$" (3 slices). With eight slices per plate, our average build time per plate is reduced to approximately nine hours.

The FDM allows us to change filament colors midway through the build process, so we construct our plates with different colors for the floor and walls, purely for visual effect.

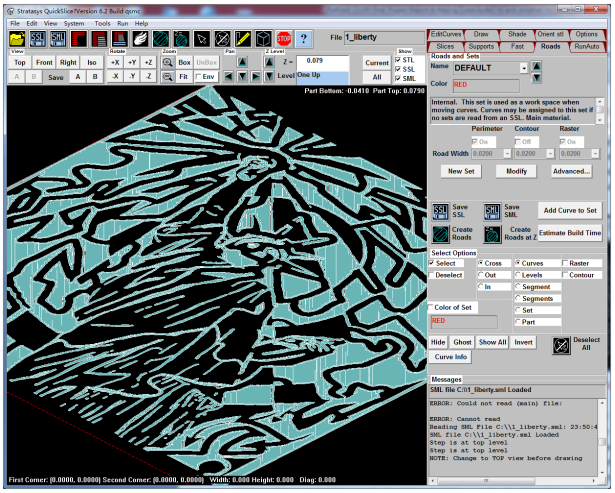

Figure 13: Liberty maze plate with roads in QuickSlice.

3.6 Cube Construction

After producing individual plates, we are able to construct the final cube. Each plate is glued to an outer acrylic plate, plastic tubing is connected between pairs of plates, and the six assemblies are glued together to form a cube. We obtained six colored acrylic sheets from Tap Plastics in El Cerrito and plastic tubing from Berkeley Ace Hardware for this purpose [3].

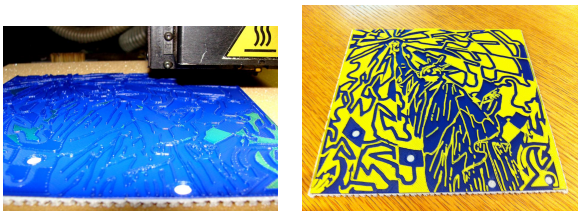

Figure 14: Left: Changing filaments during build. Right: Completed Liberty plate.

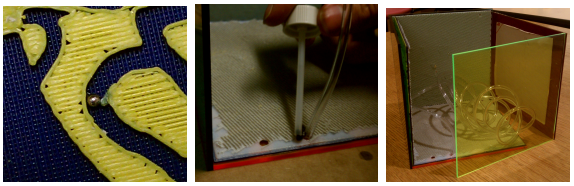

Figure 15: Left: Extra wall deposited by FDM machine. Middle: Gluing wormhole. Right: Assembling cube.

Before affixing the printed plates to the acrylic shell, it is advisable to perform a test traversal of the maze to ensure the ball has a clear and correct path. One problem we encountered was spurious wall deposition, shown in Figure 15, which required manual work with a small blade.

Construction of the complete cube is fairly intuitive; maze plates are glued to the outer acrylic sheets using all-purpose plastic solvent, then wormholes are inserted and glued into appropriate openings in the maze plates, and the outer acrylic sheets are sealed in place with acrylic cement.

4 Concluding Remarks

This project provided us with plenty of opportunities and insights into the manufacturing process, from conceptual design to CAD to iterative refinement to

Figure 16: Final cube.

fabrication, that we feel fall in line with the educational goals of CS285. First, we have developed a smooth pipeline for processing and integrating two-dimensional images into a three-dimensional cube, and enjoyed the challenge of creating a tricky and attractive product. Second, we have gained practical experience in dealing with graphics problems such as image-to-geometry conversion, and explored methods of improving on the basic marching squares algorithm. Third, we exercised modularity in our design of the A-Mazing cube, identifying the basic components needed to manufacture each face, and the minimal pathways by which to make efficient adjustments. Last, we encountered and overcame several manufacturing challenges, including balancing quality and cost, shortening turnaround and build time by optimizing the iteration and manufacturing process, and scaling product parameters to a reasonable extent.

We would like to thank Professor Séquin and the class of CS285 Fall 2011 for a great semester.

References

[1] Jonathan Ko, *Marching squares implementation for CS285 final project*, 2011, https://bitbucket.org/jonathank/cs285-final.

[2] The CImg Library, *The CImg Library - C++ Template Image Processing Toolkit.*

[3] TAP Plastics, http://tapplastics.com/.

[4] Liang Wan, Xiaopei Liu, Tien-Tsin Wong, and Chi-Sing Leung, *Evolving mazes from images*, Visualization and Computer Graphics, IEEE Transactions on **16** (2010), no. 2, 287 –297.

[5] Wikipedia, *Marching squares — Wikipedia, the free encyclopedia*, 2011, [Online; accessed 13-Dec-2011].

[6] Jie Xu and Craig S. Kaplan, *Image-guided maze construction*, ACM Trans. Graph. **26** (2007).