3D Shape Creator

Dori Rosenberg, Guy Kornblit, Hadas Aizik, Yaakov Sanders
Hebrew University in Jerusalem, Israel

In this paper, we present the research of different techniques for constructing a three-dimensional object, so that given n-images, each image corresponds to a different projection of the resulting object. The challenge in the task is to bring the object's side projections closer to the input images, and the difficulty of representing the images increases as they differ from each other.

We approached the problem as a search problem and examined various algorithms to find an optimal one. The inspiration for the project came from the American artist Michael Murphy, who assembles three-dimensional objects hanging from different objects and shapes to display different images from different angles. [1]

1. INTRODUCTION

Problem definition: Given k binary images, a novel algorithm should create a three-dimensional object, so that each input image is viewed at a different angle of the object.

Our program receives an input of 3 or 4 images sized n^2 pixels. Following image pre-processing, A 3d grid of n^3 voxels is created, and our program assigns each voxel a binary assignment (full \ empty). Finally, the result voxels are transformed to a triangular mesh, that can be later smoothed and subdivided.

During our research, we discovered that the solution space is rich in various aspects of the resulting 3D object.

Figure 1: 3 images and 4 images input, as side projection of a 3d grid.

We discovered that for different loss functions, and different progressions throughout search space, we arrived at objects whose projections faithfully represent the input images, but whose topological properties are different (single watertight volume, hollow shell, or detached shapes). In terms of our solution space, our search led us to multiple local minimum points.

2. GEOMETRIC PROPERTIES

2.1. Result topology

During the work, we discovered that the solution space is rich in various properties of the 3D objects. We observed that in the variety of algorithms we used, the loss function that was selected and the successor function (or mutations, in the genetic case) had a decisive influence on the results of the algorithm.

The resulting 3d meshes, could be separated into the following division:

• Split detached points in space.

• Single watertight 3D object.

• Failure - A certain combination of images can result in a relatively random scramble of points in space.

The following table summarizes the relationship between the loss function, the successor function, and the nature of the result obtained:

Table 1: The effect of successor function and loss function over result

Loss Function \ Successor function	**Penalize Black Pixels**	**L1 / L2**	**Penalize Empty Pixels**
Random	Multiple shredded meshes	Multiple shredded meshes	Single full object
Neighbour Successor	Multiple shredded meshes	Single full object	Single full object

2.2. Loss Functions

Our loss functions all behaved similarly. Each loss function counted the cumulative difference between each projection of the 3d grid, and it's matching input image, in an $L_1 \backslash L_2$ loss metric.

In addition, the "**Penalize Black Pixels"**, and "**Penalize Black Pixels"** loss functions penalized the value by the assignment or lack of assigned voxels, respectively. The loss value was then normalized relative to size n. By significantly penalizing assigned \ un-assigned voxels resulted in prioritizing full or empty assignments of the 3d grid, over perfect accuracy according to each projection.

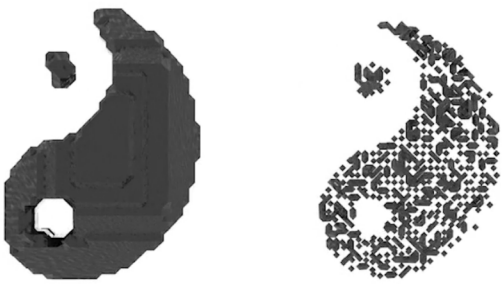

Fig 2: Single full object and Multiple Shredded meshes, for the same input images

2.3. Successor Functions

The different methods of exploring the solutions space had a large effect on the result of our program, and the time required for convergence. Each method of progressing through solution space used a different combination of exploration and exploitation.

- **Naïve successor** explores the search space without much exploitation. Randomly selects a few voxels and changes their previous assignment. Results in multiple shredded mesh objects, since physical adjacency of voxels does not effect their assignment.

- **Lines successor** explores and exploits the search space. Assigns voxels that are adjacent to the previously assigned voxel, but with no prior knowledge of previous assignment. Results in a few mesh objects, that seem to be made from pipes.

- **Neighboring successor** exploits the search space and it's known assignment. Randomly selects an assigned voxel, and then assigns simultaneously it's 26 neighbors. At a low probability, might also use binary operators to clean the 3d grid from voxels that are assigned differently from their surroundings.

We learned that the successor functions can be improved by adding randomness to each of them, at a small probability. After enough iterations that randomness assists the algorithm of avoiding local minimum points.

2.4. Initial State

The initial assignment of the 3d grid had a large impact on the final output. As can be seen in the graph and the examples that follow, a complete black slate (all of the voxels are assigned) caused the algorithm to fail. The less voxels originally assigned, the better the algorithm performed. This might be caused since the loss function calculates each projection separately, and when the 3d grid is too rich with assignments, a small change of assignments won't affect the loss value enough to make the algorithm continue exploring that path.

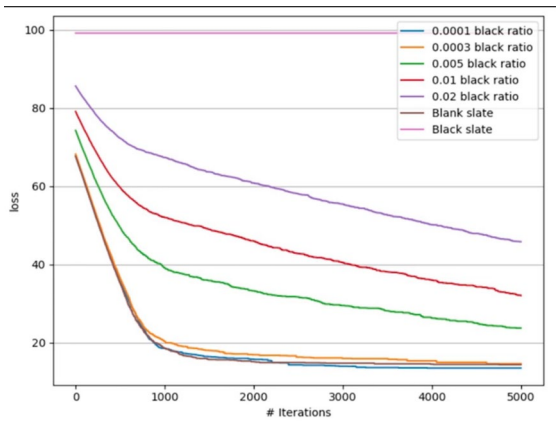

Fig 3: Graph showing amount of iterations required for convergence, based on initial state.

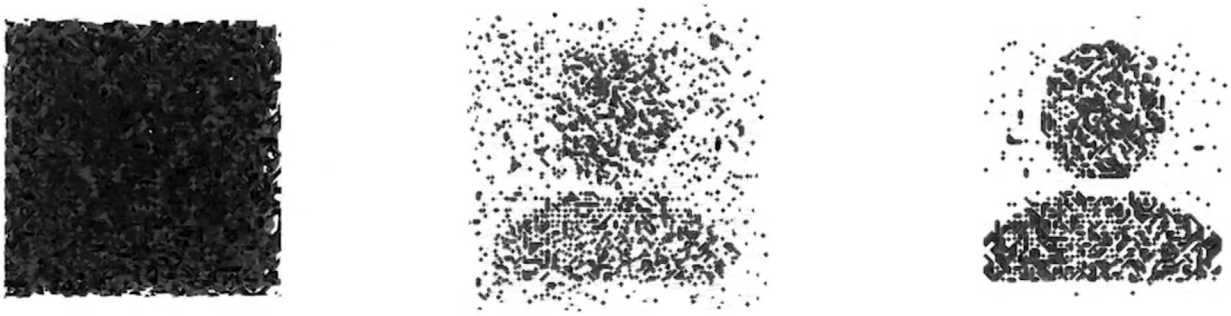

Fig 4: Different initial states and their effect on resulting output. Left – Black slate, mid – 0.01 black ratio, right – 0.003 black ratio.

3. ALGORITHMIC APPROACHES

3.1. CSP

Modeling the problem as a CSP problem requires translating the projection similarity to constraint. Each voxel has can either be full or empty, and belongs to 3 different axes. The resulting CSP problem is as follows:

$$variables = \{x_0, x_{dim*3-1}\}$$

$$domain = \{0,1\}$$

For each image we chose a matching edge of the 3d grid, then went through each voxel on that edge:

If the voxel is empty, then the sum of the entire line in the normal axis behind that voxel should be 0 (which is equivalent to the whole line being full of zeros, where no voxel is assigned). If the voxel is assigned, then the sum of the entire normal axis behind the voxel should be more than 0 (which means that there is at least one assigned voxel in the line so the assignment will be visible from projection).

Using CSP, only optimal solutions can be found, for image combinations that would not conflict. Depending on the specific CSP method, results differ. This limitation, along with the ability to solve for a small scale only (since a $\mathbf{100^3}$ 3d grid is requires 1,000,000 variables), resulted in limited use of this method in our program. As seen in the following graph, as the dimension of the problem increased, solving with Minimum Conflict required exponential runtime, whereas solving with Minimum Remaining Value was more efficient.

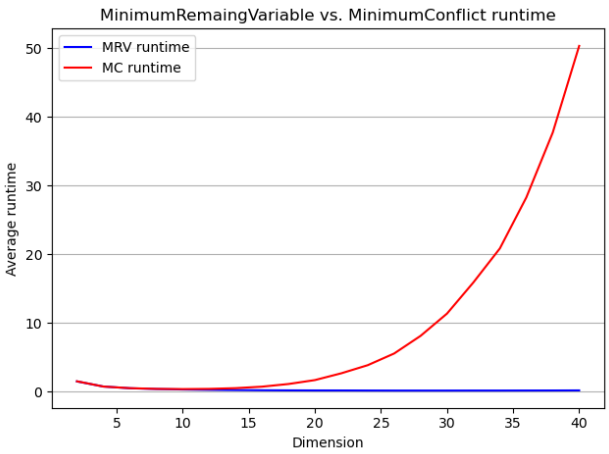

Fig 5: Graph showing amount of average runtime of CSP algorithms, in relation to image dimensions.

3.2. Local Search

Our most successful method for finding optimal solution was using various search method through the large solution space of $\boldsymbol{n^3 * 2}$ possible binary assignments. The problem is best described as a Large Neighborhood Search problem, which usually requires an element of randomness in the successor function. As mentioned earlier, different successor functions had impact on both output topology,

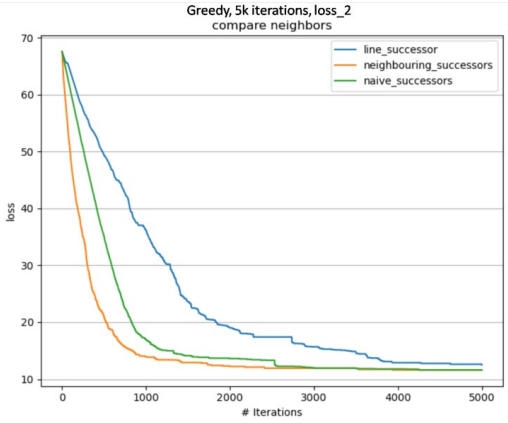

Fig 6: Graph showing amount of iterations required for convergence, based on successor function.

and convergence time.

"**Naïve successor**" explores the solution space in a complete random method, randomizing several voxels and flips their assignment. **"Neighboring successor"** exploits the solution space more than it explores it. The function randomly selects an already assigned voxel and performs random flips to the assignments of its neighboring voxels. It also performs binary operators on the 3d grid (at a low probability), causing the reduction of "noise" (voxels that their assignment is different from their whole area).

3.3. Genetic Algorithms

Genetic algorithms proved to be a difficult challenge regarding this task. Modeling the 3d voxel grid to chromosomes and genes, with successor functions as crossover and mutation, required an efficient method of splitting the 3d grid to various local components. During our research we noticed that the gene representation significantly effected the speed of convergence. The most common 1-dimentional gene representation proved to be less valuable then a 3d representation, which was able to express the geometric properties of our problem better.

4. CONCLUSIONS

From the different methods we researched, local search algorithms proved to be the best method for creating 3d objects from k-image projections. The large search space resulted in a variety of solutions, achievable using different successor functions and loss functions. We suggest our program as a tool for modeling 3d objects, and a simulator for artworks in the like of Michael Murphy.

Acknowledgments

This project was made as part of the Introduction to artificial intelligence course (67842) at HUJI.

References

[1] Michael Murphy official website.
[2] "Three-Dimensional Genetic Algorithm For The Kirkman's Schoolgirl Problem" by Jan Merta, Tomáš Brandejský, University of Pardubice, Czech Republic.