# How Can We convert 2D Pixel Art into a 3D Voxel Representation

## Exploring different 3D reconstruction algorithms on 2D pixel input

**Tihana Krajtmajer** [1]

**Supervisors: Elmar Eisemann, Petr Kellnhofer, Mathijs Molenaar** [1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
Jun 22, 2025

Name of the student: Tihana Krajtmajer
Final project course: CSE3000 Research Project
Thesis committee: Petr Kellnhofer, Mathijs Molenaar, Joana de Pinho Gonçalves

## Abstract

**This paper investigates how standard 3D reconstruction techniques can be adapted to work with orthographic projections of pixel art images. Reconstructing 3D models from 2D images is typically done with real world objects. However, little work has explored this problem in the context of pixel art, which has a lower resolution, and uses stylistic colors and shading, making it a problem that requires different or adapted techniques. We implement three reconstruction algorithms based on some common practices in the field: silhouette-based intersection, spatial carving using color consistency, and gradient-based depth estimation. Results show that silhouette intersection is an effective method for simple models, but fails to capture concave regions. Spatial carving addresses this limitation, although it also fails in some use cases. An adapted depth estimation technique is also used, which most accurately captures these regions, although it does not fill in the model as well. We implement a custom Blender plugin to support user annotations and improve accuracy in some ambiguous areas. We conclude that a hybrid approach, where silhouette intersection is combined with depth estimation, gives the most accurate results, and we suggest future work that can be done on the topic, including better color merging principles, adding custom viewpoint orientations, adding support for multiple object reconstruction and using perspective projections.**


## I. Introduction

The topic of reconstructing 3D models from a set of 2D images is a well-known and well-defined problem in Computer Graphics, and it has been relevant in the field of Computer Vision for the last thirty years. The topic is especially relevant in the fields of medicine, robotics, architecture and game media [1]. One area that has not been explored in formal research is the reconstruction of models using 2D pixel art images.

Pixel art is a digital art form that uses simplified, low-resolution visuals. It is often used in stylized digital art and video game development [2]. Unlike 2D images of real life models, pixel art has flat colors, and shading is represented artistically, rather than in a realistic manner.

There have been many different algorithms developed for approximating a 3D object from its 2D projections that use vastly different approaches. They can be categorized in terms of the types of image cues that are used in the reconstruction process [3], i.e. the information that can be derived from each 2D image. Although many different reconstruction methods exist, there has, to our knowledge, not been any attempt made to examine how they perform when using pixel art. Pixel art has a considerably lower amount of information compared to real-life objects, and it lacks features that traditional 3D reconstruction techniques rely on, such as lighting, texture gradients, or perspective distortion.

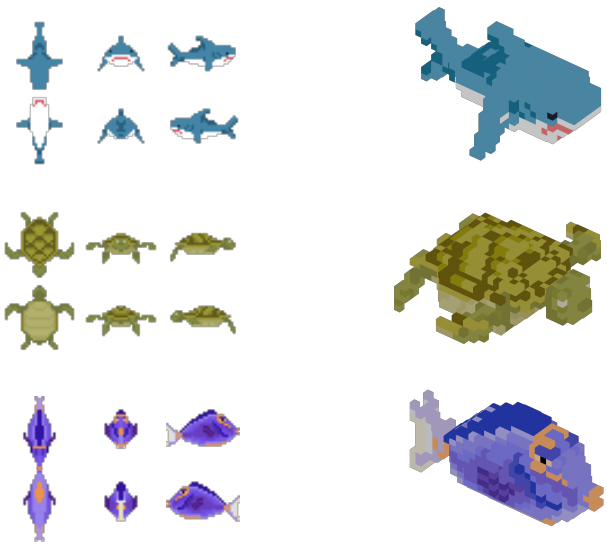

Fig. 1: Six input images (on the left), and the resulting 3D model (on the right).

The aim of this paper is to fill in this knowledge gap, by examining how three different algorithms that rely on different image cues perform on 2D pixel art inputs. The algorithms are: silhouette-based intersection, spatial carving using color consistency, and gradient-based depth estimation. These methods use three types of image information: shape or silhouette boundaries, surface color consistency, and intensity changes, respectively.

The methods are adapted to work with orthographic projections of the pixel art images taken from multiple viewpoints. An example of our results is illustrated in Figure 1. The final results are analyzed using a wide collection of images, representing both concave and convex shapes. Additionally, in some cases fully automatic reconstruction is not possible. To address this, the proposed solution is implemented using a simple voxel editor, which supports user annotations for correcting the final shapes.

The main question this paper answers is how multiple orthographic 2D pixel art images from different viewpoints can be used to construct a 3D voxel representation using various fill methods. The problem of reconstruction accuracy will be further examined, as what determines the quality of the resulting model needs to be defined as a basis of comparison between different methods. Another aspect that is worth examining is whether the fill methods are influenced by different independent variables, such as the number of 2D images used in the final reconstruction, or the angles of the projections. Additionally, due to well-known limitations in the proposed approaches, certain characteristics of 3D shapes cannot possibly be reconstructed without some user input. The limitation of automatic reconstruction will be

examined in further detail, as well as the benefits of using user input for a complete reconstruction.

To assess the quality of all three methods, two evaluation strategies are used. The first method uses existing voxel models to test reconstruction accuracy. The second method uses 2D input images to test how well the algorithms generalize unknown viewpoints. We conclude by analyzing the results and their implications, where combining silhouette intersection with depth map estimation is chosen as the most accurate reconstruction method.

## II. Related Work

Early work in 3D reconstruction was focused mostly on using stereo correspondence [4]–[6] and structure-from-motion [7], [8], both matching corresponding pixels in two images from slightly different viewpoints to estimate their depth. Over time, different image cues were introduced as a basis for new reconstruction algorithms, such as silhouettes, shading and texture consistency.

One early approach was based on object silhouettes. This involves extracting the outer contours of an object from different views and intersecting the resulting silhouette cones to approximate the object's shape. Intersecting the volumes obtained from different viewpoints based on orthographic projections was first described by Martin and Aggarwal [9]. Subsequently, some attempts were made to generalize the approach of using silhouette information and examining the limitations in the reconstructed results. Attempts were made using both orthographic and perspective projection [10], [11]. However, Laurentini [12] was the first to introduce the concept of a visual hull and to fully formalize the extent to which objects can be reproduced from silhouettes alone. The visual hull, as defined in the work, is the maximum volume that can be reconstructed from a set of images, such that adding new viewpoints does not add any more information. While it is effective at representing the outer shape, the visual hull cannot reconstruct concavities or internal features that are not visible in any silhouette. This limits the results of this approach, as it is expected to not perform well at addressing concavities and overlaps in shapes from the silhouette. A silhouette intersection method will be further examined in this work based on the approaches taken by Martin and Aggarwal [9] and Laurentini [12].

To address some of the limitations of silhouette-based methods, space carving was introduced by Kutulakos and Seitz [13]. In contrast to purely geometric reconstruction, space carving evaluates the photo-consistency of voxels across multiple images. A point is retained only if it is consistent in color across all viewpoints. Due to this, features like concavities and other internal features are also reconstructed. Their approach introduced the concept of a photo hull - an extension of the visual hull that also takes into account photometric constraints to better approximate the 3D shape. Broadhurst et al. [14] improved on this idea by introducing a probabilistic model for space carving, where each point is assigned an occupancy probability. This was an improvement because the algorithm could handle uncertainty from image noise and occlusions. Further advancements on the original idea have since been proposed that improve performance on noisy or texture-rich scenes [15], [16]. A method for spatial carving based on the approach by Kutulakos and Seitz [13] is used in this work to create a photo hull and improve upon the limitations of silhouette intersection.

Techniques based on estimating depth are also quite common. They are typically applied to single input images in methods such as shape-from-shading [17], where the depth is calculated from changes in illumination across the surface of the image. However, because this project focuses on pixel art images that have flat shading and have no light source affecting the image, such approaches are adapted. In this work, instead of relying on a full shading model, a simplified grayscale-based approach is used. Input images are converted to grayscale, and local intensity gradients are computed to identify changes in intensity. Similar approaches using gradient or edge information to infer structure have been proposed in prior work [18], [19]. Although this method is not standard in 3D reconstruction, it is an adjusted approach that works on pixel art image input specifically, while also addressing some of the limitations that the other approaches have.

Recent research focuses on using deep learning for reconstructing models from both single or multiple input images. These methods typically learn 3D representations by training on large datasets of image-shape pairs and learning view-consistent priors for shape reconstruction [20], [21]. Such approaches were, however, deemed to be too complex for our work due to requiring a high amount of training data and computational resources, and are hence out of the scope for our research.

## III. Methodology

The goal of the research is to reconstruct 3D voxel models using a set of 2D pixel art images. The images are sampled at every pixel and projected orthographically onto the grid as shown in Figure 2. Different standard approaches in 3D reconstruction algorithms are used in order to examine the limitations of automatic reconstruction. Additionally, we examine the validity of relying on user input where automatic generation fails.

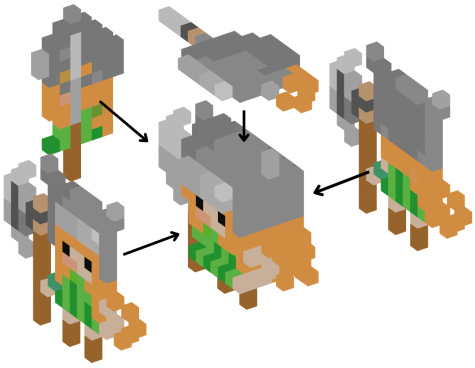

Fig. 2: Example of how input images are projected into the final shape.

A set of 2D images with their corresponding viewpoint directions are given as input to all three algorithms. Furthermore, only images containing single objects are considered, therefore no attempt is made to address object occlusions. At least two images are required as input to be able to reconstruct a valid shape. Each algorithm also has a set of parameters that have to be specified manually, and have an effect on the final result. The resulting output is a 3D grid containing the color information of each voxel. Colors are determined using two different merging techniques based on overlapping points in the 3D space.

## A. Reconstruction Methods

Using different information from the input images allows for generating various results that may capture different aspects of the 3D shape. Three algorithms that rely on different image cues are implemented: silhouette intersection, spatial carving, and depth estimation. In Figure 3 we show the three algorithms generates different outputs.

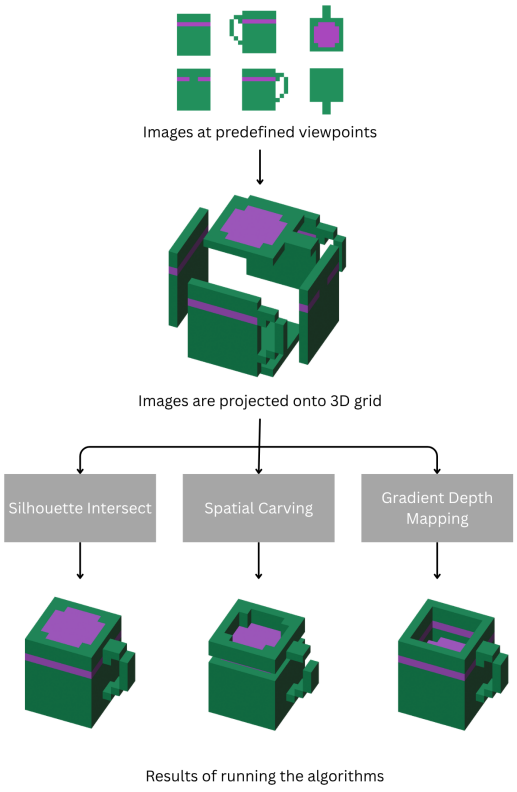


Fig. 3: Results for running the three algorithms using the same images. The object is expected to have a concavity in the top view.

### Silhouette Intersection

Silhouette intersection is a technique where the overlap of the silhouettes of different 2D projections is taken as a basis for filling in the model. A silhouette of an image encapsulates all the visible, i.e. non-transparent pixels that the image contains. Typically, the silhouette of the object in a scene is determined using a mask to split the captured image into foreground and background components, however, because we are using single objects this step is not necessary.

In order for a voxel to be filled in the grid at any position in the 3D space it needs to have at least a certain amount of overlapping projections. At each voxel $v$, let there be a set of overlapping projections $V$, and let $P_v$ denote the subset of visible views. Then $v$ is filled if:

$$\frac{|P_v|}{|V|} \geq T_s$$

where $T_s \in [0, 1]$ is the user-defined overlap threshold. Table I lists all the user-specified parameters for silhouette intersection.

| Name | Description | Range of Values |
|---|---|---|
| Color Merging Technique | Determines the color merging technique, as specified in section III-B | Nearest projection, Majority vote |
| Threshold | Determines the number of overlapping projections that are considered enough to fill in a point. | [0.0 - 1.0] |

TABLE I: Silhouette intersection input parameters.

### Spatial Carving

Spatial carving uses a photometric consistency method, whereby the color information of the input images is used as a basis for reconstruction. Carving refers to a process where the grid is initially completely filled in with a solid color. By looking at the 2D projections, the areas that are determined to be empty are 'carved out' or removed from the grid in a step-by-step procedure. The color information is used to further expand upon this initial idea. Rather than looking at the silhouettes alone, which would generate the same shape as silhouette intersection, and likewise omit any information contained within the shape, the colors of overlapping projections are further examined based on their variance. If the variance of overlapping colors at a point is too high (above the defined threshold), the point is considered inconsistent and is carved out. For a voxel $v$, the color variance is defined as:

$$\sigma_v^2 = \frac{1}{n} \sum_{i=1}^{n} \|c_i - \mu_v\|^2$$

where $\{c_1, c_2, \ldots, c_n\}$ is the set of all projected colors onto the voxel for a total of $n$ projections. Then the voxel is carved out if:

$$\sigma_v^2 > T_c$$

where $T_c \in [0, 1]$ is the user-defined variance threshold. The resulting method is a further expansion upon the simple silhouette-based one because it is able to successfully carve out concave shapes. However, the results do depend on

setting the correct variance threshold, as well as using input images that have some shading cues within them, such as using darker colors than the base ones to denote shadows, where the concavities may fall. The user-specific parameters that influence the final output are listed in Table II.

| Name | Description | Range of Values |
|---|---|---|
| Color Merging Technique | Determines the color merging technique, as specified in section III-B | Nearest projection, Majority vote |
| Variance Threshold | Determines the amount of color variance that is allowed at a point. | [0.0 - 1.0] |
| Concavity Depth Threshold | Influences how much the concavity is carved out along its depth axis. | [0.0 - 1.0] |

TABLE II: Spatial carving input parameters.

### Gradient Depth Estimation

Gradient depth estimation is a custom third technique that is implemented for the reconstruction process. What makes it distinct from silhouette intersection or spatial carving is that it relies on shading cues directly, and has more flexibility in handling concavities. It was inspired by techniques from shape-from-shading, but adapted to work better for pixel art, as pixel art has flat colors and stylized shading (no lighting cues).

This method is implemented by analyzing the input images and creating their corresponding depth maps. A depth map is a 2D array, where each entry contains the depth of the pixel at position (x, y) along its corresponding projection axis. For example, in a top-view image that has coordinates (x, y), the projected points influence coordinates (x, y, z) in the 3D space, where z corresponds to the height of the model. Here, the depth map would store only its z-coordinates explicitly, as the other two can be directly inferred from the 2-dimensional image. The depth maps are intersected to create the final model.

Depth estimation is performed by calculating the image gradients, which highlight the areas of intensity changes, and which we interpret as areas of interest where concavities may appear. Input images are first converted to grayscale. The gradients are calculated using the Sobel operator, which is applied in the horizontal $G_x$ and vertical $G_y$ directions:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}, \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

The operator is convolved over the grayscale image. The magnitude of the gradient at each pixel is then computed as:

$$\|\nabla I(x,y)\| = \sqrt{G_x(x,y)^2 + G_y(x,y)^2}$$

The result is a 2D grayscale image of gradient magnitudes, where the highest values correspond to areas of high intensity change, such as edges or concavities. The image is eroded with a 3x3 kernel that removes edge details. Then a binary mask is generated based on regions of interest (where the gradient magnitudes are higher above the specified threshold). The mask is filtered a final time, in order to remove regions where their area is too small. This step is done in order to prevent noise from being treated as a concave region. For each valid identified concave region, the area is 'carved' into the depth map, using the projection direction and the pixel intensity, as well as the parameters specified in more detail in Table III.

| Name | Description | Range of Values |
|---|---|---|
| Intensity Threshold | Determines the intensity value that is filtered above in the algorithm. | [0.0 - 1.0] |
| Concavity Depth Threshold | Influences how much the concavity is carved out along its depth axis. | [0.0 - 1.0] |
| Depth Factor | Multiplier that affects offset between concave regions. Because concavities are generated based on image intensity, | [-5.0 - 5.0] |
| Minimum Region Size | Filter for determining the minimum size a region has to be to be carved out. | [0.0 - unspecified] |

TABLE III: Depth estimation input parameters.

### The Hybrid Approach

Silhouette intersection can be combined with depth estimation to create a shape that is fully filled in with no gaps, and that is able to reflect finer details reflected as changes in the image shading. This method was implemented by combining the two: depth maps are precomputed, after which silhouette intersection is performed as described in subsection III-A. If the voxel falls out of predicted range, i.e. if it is 'closer' to the projection image than what was calculated in the depth map, it is not filled in. The parameters for this method include all of the parameters for silhouette intersection and depth mapping.

## B. Color Merging Techniques

Different color merging approaches are used to determine the final color at each position in the voxel grid. Two different methods are used: calculating the color based on the nearest projection plane to the voxel point and using a voting system to determine the most frequent color at a point. The merge strategies are implemented as options in the user editor to allow further model customization and to observe their effects on color accuracy for the reconstruction process.

Choosing colors from the nearest projection is done by finding the minimum distance from the overlapping point

to each face of the bounding volume, i.e. the sides of the enclosing cube. The color is then taken from the projection corresponding to the closest face. In the majority voting approach, the system selects the color that appears most frequently among the overlapping values. Ties are broken arbitrarily, by always selecting the first color that has the maximum amount of overlap.

### C. User Input

User input is used to further extend the editor and allow the user to change parts of the model that are not consistent with expected results. The user is able to draw a selection over any input image, effectively a mask to select which pixels in the input should be used to construct the final model. The mask acts as a filter for the reconstruction methods, allowing the user to prevent unwanted artifacts. Only the selected pixels are used further by the four algorithms, meaning calculations such as gradient estimation may be affected by the mask. Additionally, users can select specific projection images to exclude from processing or manually adjust voxel colors after reconstruction. This allows the user to address the limitations of the three algorithms.

## IV. Implementation

The solution is implemented using Python 3.0. A Blender plugin is added in order to facilitate easier user input for testing purposes. It supports loading 2D pixel images, selecting their viewpoints (from a predefined list), selecting the desired reconstruction algorithm and color merging technique, as well as support for adjusting some parameters that are given as inputs to the algorithms. An overview of what the plugin looks like is shown in Figure 4.

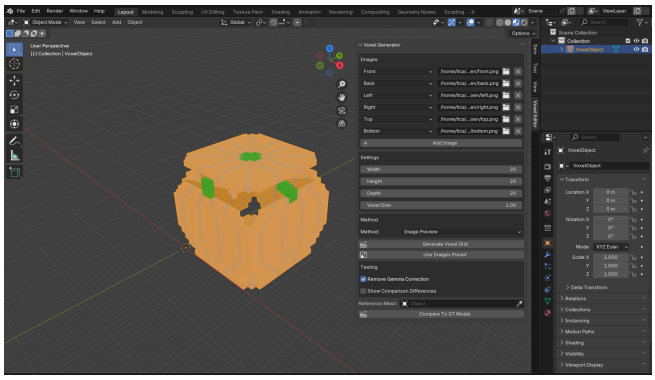

Fig. 4: Final Blender plugin overview

The models are represented using a simple dense 3D grid that stores color information at each point. Voxel cubes are then generated at the points where the color values are non-transparent. Additionally, we hollow out the final grid to improve performance and ensure a more fair comparison between the methods. This is done by removing any voxel that is not visible from any of the six orthogonal directions, i.e. it is fully enclosed by other occupied voxels.

## V. Experimental Setup and Results

We evaluate three aspects of the algorithms: accuracy, generalization ability and effectiveness of user input to disambiguate shapes with excess geometry. To test accuracy, we compare our generated models with existing ones. Generalization is tested by using a subset of 2D input images, then comparing how well unseen views correspond to our images from the opposite viewpoints. We also give a brief assessment of how using user annotations improves the overall shape quality in some cases.

For the first evaluation method we use a set of existing voxel models that are permitted for free use under the Creative Commons license. For the second, we use a custom set of 2D images created specifically for this study. All evaluation methods are conducted using RGBA pixel images with maximum size 32x32, which we choose for performance reasons. However, the conclusions we make can be generalized to larger pixel images as well.

### A. Comparison with Existing Voxel Models

In order to compare the results of our algorithms with existing voxel models, we calculate reconstruction accuracy. Point-based and color-based accuracy are measured. The reference models are sampled at six viewpoints corresponding to six faces of a cube, which gives a comprehensive overview of the model's shape. The samples are rendered and used as input images for the four algorithms. The models were created by artists Sona-Sar [22] and Kyrise [23] on Itch.io. Some are also taken from the free package in the MagicaVoxel [24] editor.

Point-based accuracy is measured using intersection-over-union, where:

$$\text{IoU} = \frac{|\text{Prediction} \cap \text{Ground Truth}|}{|\text{Prediction} \cup \text{Ground Truth}|}$$

Results are shown in Table IV. When running the algorithms, we vary the parameters using sets of possible values, and the ones that produce the best IoU are selected. The full list of parameters was too extensive to be included in the report, however it can be found in the Github repository [25] of this project.

For color-based accuracy, only the voxels occupying the same positions in both grids are considered, and their color similarity is computed. To account for minor variations in the color values due to rendering artifacts, a small tolerance threshold is used when comparing color values. We measure color similarity using Mean Squared Error as:

$$\text{MSE} = \frac{1}{|V|} \sum_{i \in V} \left\| \mathbf{c}_{\text{pred}}^{(i)} - \mathbf{c}_{\text{gt}}^{(i)} \right\|_2^2$$

$$\text{where} \quad V = \left\{ i \,:\, \alpha_{\text{pred}}^{(i)} > T_{\text{tol}} \quad \text{or} \quad \alpha_{\text{gt}}^{(i)} > T_{\text{tol}} \right\}$$

Here, $\mathbf{c}_{\text{pred}}^{(i)}$ is the predicted color at voxel $i$, $\mathbf{c}_{\text{gt}}^{(i)}$ is the ground truth color at voxel $i$, $\alpha_{\text{pred}}^{(i)}$ and $\alpha_{\text{gt}}^{(i)}$ are the predicted and ground truth alpha values at voxel $i$, $T_{\text{tol}}$ is the transparency threshold below which a voxel is considered fully transparent and $V$ is the set of voxels where at least one of the alpha

| Model | Silhouette | Carve | Depth | Hybrid |
|---|---|---|---|---|
| bone | **0.94** | **0.94** | 0.56 | **0.94** |
| box | 0.23 | 0.23 | 0.42 | **0.43** |
| box2 | 0.20 | 0.20 | 0.40 | **0.41** |
| box3 | **1.00** | **1.00** | 0.91 | **1.00** |
| box4 | **0.92** | **0.92** | 0.82 | **0.92** |
| bush | **0.90** | **0.90** | 0.79 | **0.90** |
| cactuss | **0.56** | **0.56** | 0.44 | **0.56** |
| castle | 0.72 | **0.78** | 0.45 | **0.78** |
| chr_knight | 0.60 | **0.61** | 0.47 | 0.60 |
| chr_sword | 0.68 | **0.70** | 0.48 | 0.68 |
| coin | 0.41 | 0.41 | 0.42 | **0.45** |
| covjek | **0.66** | **0.66** | 0.55 | **0.66** |
| fence | **1.00** | **1.00** | 0.79 | **1.00** |
| rock | **0.81** | **0.81** | 0.72 | **0.81** |
| skull | 0.87 | **1.0** | 0.81 | 0.91 |
| smallguy | **0.92** | **0.92** | 0.70 | **0.92** |
| **Average** | 0.714 | 0.740 | 0.608 | 0.750 |

TABLE IV: IoU scores for the first experiment. The best results are highlighted.

values is greater than $T_{\text{tol}}$. Both color merging techniques are considered separately. Table V shows the results for each algorithm and color merging combination.

| Model | SilNP | SilMV | CarvNP | CarvMV | Depth | HybridNP | HybridMV |
|---|---|---|---|---|---|---|---|
| bone | 0.037 | **0.012** | 0.037 | **0.012** | 0.260 | 0.037 | **0.012** |
| box | 0.201 | 0.194 | 0.203 | 0.194 | 0.182 | **0.162** | 0.165 |
| box2 | 0.343 | 0.315 | 0.354 | 0.315 | 0.282 | **0.270** | 0.283 |
| box3 | 0.070 | 0.075 | **0.067** | 0.075 | 0.071 | 0.070 | 0.075 |
| box4 | 0.066 | **0.063** | **0.063** | **0.063** | 0.072 | 0.067 | 0.064 |
| bush | 0.077 | 0.091 | **0.072** | 0.091 | 0.078 | 0.089 | 0.091 |
| cactuss | **0.159** | 0.170 | 0.167 | 0.170 | 0.184 | 0.160 | 0.171 |
| castle | 0.179 | 0.181 | **0.145** | **0.145** | 0.359 | 0.147 | **0.145** |
| chr_knight | 0.175 | **0.173** | **0.173** | **0.173** | 0.180 | 0.175 | **0.173** |
| chr_sword | **0.187** | 0.191 | 0.215 | 0.212 | 0.302 | **0.187** | 0.191 |
| coin | 0.265 | **0.258** | 0.262 | **0.258** | 0.270 | 0.262 | **0.258** |
| covjek | 0.211 | 0.210 | **0.198** | 0.210 | 0.235 | 0.211 | 0.210 |
| fence | **0.079** | 0.086 | 0.080 | 0.086 | 0.104 | **0.079** | 0.086 |
| rock | 0.090 | 0.098 | **0.072** | 0.098 | 0.073 | 0.090 | 0.098 |
| skull | 0.063 | 0.072 | **0.021** | 0.040 | 0.155 | 0.066 | 0.074 |
| smallguy | 0.146 | 0.133 | **0.131** | 0.133 | 0.174 | 0.146 | 0.133 |
| **Average** | 0.147 | 0.145 | 0.141 | 0.142 | 0.186 | 0.139 | 0.139 |

TABLE V: MSE scores for the first experiment. The best results are highlighted. MV = Majority Vote, NP = Nearest Projection

The results indicate that the hybrid technique is the most accurate in terms of voxel overlap. Spatial carving produces results that are almost as accurate, and in the majority of cases, both these methods are selected as the most optimal ones. Depth mapping gives the worst results, indicating that this technique fails to fill in the shape exactly. Figure 3 highlights some of the results of running this experiment which highlight some of the key differences between the four algorithms. In terms of color accuracy, the methods overall have a low MSE score for both color merging techniques, indicating a higher reconstruction quality. The hybrid methods perform the best, however not by a large margin. Depth mapping by itself gives the worst results, as is never chosen as the best technique for color mapping. Additionally, the choice of merge technique is highly dependent on the object itself, as there is no clear distinction between why nearest projection might be chosen over majority voting.

## B. Varying the Amount of Input Images

To examine the effect of reducing the number of input views, reconstructions are also performed using a subset of the available input images. The results are re-projected into known 2D viewpoints, and the amount of overlap with the ground-truth projections is evaluated. This helps assess how the algorithms extrapolate missing information, and how consistent the results are for unseen viewpoints. Parameters are kept the same for each algorithm, and were chosen based on the values that were most repeated in experiment one. We measure IoU and MSE as was already outlined, the results of which are presented in Table VI and VII respectively. A full overview of intermediate values is given in Appendix A. Figure 5 shows an example of outputs for this experiment.

| Average | Silhouette | Carving | Depth | Hybrid |
|---|---|---|---|---|
| Two views | **0.995** | **0.995** | 0.988 | 0.979 |
| Three views | **0.993** | **0.993** | 0.974 | 0.888 |

TABLE VI: Averaged IoU scores for the second experiment. The best results are highlighted.

| | Silhouette | | Carving | | Depth | Hybrid | |
|---|---|---|---|---|---|---|---|
| Average | NP | MV | NP | MV | - | NP | MV |
| Two views | 0.0283 | 0.0362 | 0.0283 | 0.0362 | **0.0280** | 0.0284 | 0.0363 |
| Three views | **0.0174** | 0.0193 | **0.0174** | 0.0193 | 0.0243 | 0.0178 | 0.0204 |

TABLE VII: Averaged MSE scores for the second experiment. The best results are highlighted. MV = Majority Vote, NP = Nearest Projection

The results show that both silhouette intersection and spatial carving achieve high voxel overlap, both when using two and three input images, suggesting that these methods work really well when only a subset of images is available for reconstruction. In contrast, the hybrid approach performs slightly worse, especially when three views are used. In terms of color, MSE is overall reduced when using more inputs, which is to be expected. Using nearest projections is also slightly favored by all algorithms.

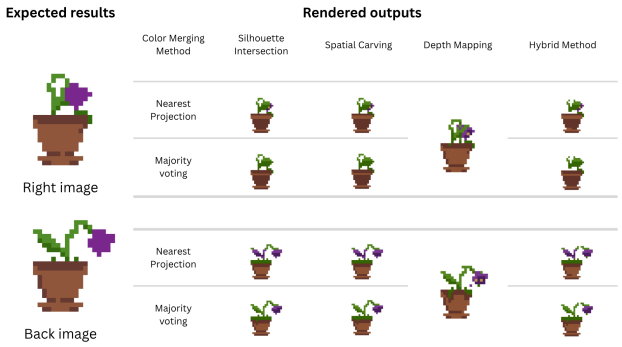


Fig. 5: Example results of running the experiment, with expected outputs (on the left) and the full set of outputs that were generated for each algorithm and merge technique (right).

### C. Adding User Annotations

Some artifacts in the reconstruction process could not be resolved using automatic methods alone, particularly in cases where a part of the object might be adding excess information. To address this, select input images are masked to exclude the unwanted regions during reconstruction. Figure 6 shows such a case for the models of a chair and a bear. Using unmasked images creates extra geometry and the shapes get overfilled. While we do not evaluate this method quantitatively, visual results show that applying the mask significantly improves the clarity of the shapes.

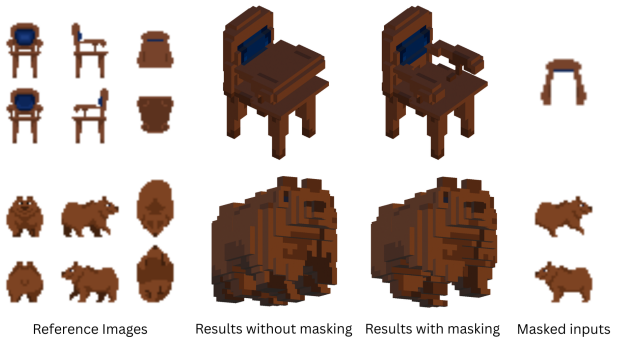


Fig. 6: Comparison of using unmasked and masked input images on two objects to refine the shape. The created masks are shown on the left.

## VI. Responsible Research

Ethical concerns are fully taken into account. We acknowledge that this tool can be used to automate a part of the artistic process, which may not be in line with what artists would do to create their art. However, we emphasize that our intention is to support rather than replace artistic input. The tool is designed such that the created models can be used as a baseline for making original voxel models, especially because the tool itself requires 2D pixel images that artists can create themselves.

To make the methods and their results accessible and reproducible, the entire code-base is made public in a Github repository [25]. The code is well-documented, and includes a Read Me entry that details how to set up the Blender plugin, as well as some input examples to get started. In the repository, we also include the intermediary results of our experiments, as well as the full outputs and all parameters that were selected. Additionally, the voxel models as well as some 2D input images that were not created by us are fully attributed to their authors, and only open-source material was used in the experimental process. The 2D art was created partly by us, and partly by Demetra Carata Dejoianu [26], solely for the purposes of the project.

## VII. Discussion

Results show that silhouette intersection is very effective at capturing simple convex shapes, without necessarily being able to reproduce the internal features contained in the silhouettes of the projection images. This idea is in line with previous research on silhouette-based methods and the visual hull.

Spatial carving addresses this somewhat successfully. However because a strict variance threshold is defined, there are cases where the model 'over-carves', and removes details that happen to meet the variance threshold criteria, even when they are not actual concavities.

The gradient-based depth estimation technique was developed throughout the research by observing some early results from the other two methods, and attempting to address their fail-cases. As such, it most accurately captures details on concavities, although it does not completely fill in the shape and produces some gaps as a result. This is because each depth map holds only one layer of data, meaning if a shape is extruded from the surface only the nearest points are captured.

The hybrid method of combining silhouette intersection with estimated depth maps produces the best results, as concavities are identified efficiently without also misclassifying smaller details, and the shape is filled in correctly, resulting in no gaps in the model.

These observations are consistent with the results of the experiments. When measuring accuracy, the hybrid method outperforms all others, while spatial carving is also almost as effective. This is because the hybrid method is able to capture both the overall shape structure, but also the finer details provided by depth estimation. Spatial carving further refines the visual hull produced by silhouette intersection by creating the photo hull of the object, and removes further ambiguities, particularly for simpler or more uniformly colored objects. The results of assessing color-based accuracy were somewhat inconclusive, as they seemed to depend more on the object itself, and both color merge techniques produced results that were reasonably good, with no single method consistently outperforming the other.

When considering how using a subset of views affects reconstruction, silhouette intersection and spatial carving outperform the other methods in terms of both point-based and color-based accuracy. This is because both methods rely primarily on the shapes of the silhouettes themselves, rather than attempting to estimate the shape structure. Silhouette intersection is especially reliable in this use-case because it does not make assumptions about the shape beyond the silhouette itself, and is able to give a good conservative estimate of the shape even when fewer views are available. Spatial carving also fundamentally relies on the silhouette information to carve out the general shape.

## VIII. Conclusions and Future Work

In this paper, we presented a study on reconstructing 3D voxel models form 2D pixel art images using pre-existing 3D reconstruction techniques that were adapted to our pixel inputs. Three algorithms: silhouette-based intersection, spatial carving using color consistency, and gradient-based depth estimation, were implemented and evaluated in different ways to test their accuracy in reconstructing existing models


Fig. 7: Results of running the four reconstruction algorithms for various input images. The images on the left show the input that was used (the existing models were sampled at the same six predefined viewpoints). The second column shows the rendered reference models.

and extrapolating information when missing some of the input images.

A major contribution of this work is demonstrating that silhouette intersection can be combined together with depth estimation, in order to both produce a full shape and correctly identify regions with varying depths. This hybrid approach is not considered in existing studies because they focus on real-world objects and use different, more complex shading models. However, our slightly simplified approach was proven to be meaningful in the context of stylized digital art using flat colors.

As part of our work, we also developed a custom Blender plugin that uses our reconstruction methods. The plugin provides an interface for users to load 2D pixel images, select viewpoints, apply reconstruction algorithms, and make manual annotations to improve the final model quality. This also provides artists with a starting point for converting their 2D pixel art into editable 3D models.

We have identified a few possible areas of future work that our implementations can be extended with. First, in this paper we make the assumption that the inputs should be projected onto the grid orthographically, however pixel

art is also drawn with perspective depending on the use case. The system should be extended to allow the user to specify which camera position and angle they want to use, and allow them to generate models from perspective 2D projections. Additionally, the system currently supports only a predefined set of viewpoints, however this can be further expanded upon to allow the user to specify the input image orientations manually. Currently, our solution handles only single object reconstruction. Support for reconstructing multiple objects can be added, and visibility maps can be added as a parameter that determines which objects occlude which others. Other than this, our suggested color merging strategies can be further expanded upon. Currently, we only consider the color overlap at a single point. However, the surrounding colors around the point could be used to, for example, break ties more meaningfully in the majority vote color merging scheme. Although we are able to make some conclusions about our methods, we acknowledge that our datasets are quite small, and that future studies should focus on testing these methods on larger amounts of data. We also note that, although our methods were able to produce results that are reasonably accurate, further work needs to be done to produce methods where the accuracy is even higher.

# Appendix

## A. Experiment 2: Full Overview

Here, we give the intermediate values that were used to compute the final average IoU and MSE scores in the second experiment.

| input views | view | silhouette | carving | depth | hybrid |
|---|---|---|---|---|---|
| front, left | back | 0.995 | 0.995 | 0.993 | 0.988 |
| | right | 0.998 | 0.998 | 0.991 | 0.991 |
| front, top | back | 0.996 | 0.996 | 0.985 | 0.972 |
| | bottom | 0.992 | 0.992 | 0.990 | 0.985 |
| top, left | bottom | 0.992 | 0.992 | 0.986 | 0.971 |
| | right | 0.998 | 0.998 | 0.984 | 0.966 |
| Average using two views | | 0.995 | 0.995 | 0.988 | 0.979 |
| front, top, left | back | 0.991 | 0.991 | 0.970 | 0.897 |
| | bottom | 0.992 | 0.992 | 0.981 | 0.904 |
| | right | 0.997 | 0.997 | 0.972 | 0.863 |
| Average using three views | | 0.993 | 0.993 | 0.974 | 0.888 |

TABLE VIII: IoU calculations for each set of input views, averaged over all the objects.

| | | Silhouette Intersection | | Spatial Carving | | Depth Mapping | Hybrid method | |
|---|---|---|---|---|---|---|---|---|
| input views | view | NP | MV | NP | MV | - | NP | MV |
| front, left | back | 0.0388 | 0.0348 | 0.0388 | 0.0348 | 0.0187 | 0.0390 | 0.0349 |
| | right | 0.0404 | 0.0380 | 0.0404 | 0.0380 | 0.0097 | 0.0404 | 0.0381 |
| front, top | back | 0.0203 | 0.0197 | 0.0203 | 0.0197 | 0.0220 | 0.0205 | 0.0198 |
| | bottom | 0.0239 | 0.0517 | 0.0239 | 0.0517 | 0.0508 | 0.0238 | 0.0517 |
| top, left | bottom | 0.0251 | 0.0505 | 0.0251 | 0.0505 | 0.0522 | 0.0249 | 0.0505 |
| | right | 0.0216 | 0.0224 | 0.0216 | 0.0224 | 0.0105 | 0.0218 | 0.0228 |
| Average using two views | | 0.02835 | 0.03618 | 0.02835 | 0.03618 | 0.02798 | 0.0284 | 0.0363 |
| front, top, left | back | 0.0161 | 0.0192 | 0.0161 | 0.0192 | 0.0159 | 0.0163 | 0.0193 |
| | bottom | 0.0182 | 0.0234 | 0.0182 | 0.0234 | 0.0447 | 0.0198 | 0.0266 |
| | right | 0.0179 | 0.0152 | 0.0179 | 0.0152 | 0.0124 | 0.0173 | 0.0153 |
| Average using three views | | 0.0174 | 0.0193 | 0.0174 | 0.0193 | 0.0243 | 0.0178 | 0.0204 |

TABLE IX: MSE calculations for each set of input views, averaged over all the objects. MV = Majority Vote, NP = Nearest Projection

## B. Use of Generative AI Models

ChatGPT was used to generate code to run both experiments. For the first experiment, the prompt that was used is:

```
i want to write a script to automate my testing process;
     i have a set of reference photos for each object
     from 6 view directions
     (top,bottom,left,right,back,front) and a txt file
     that is my ground truth; i want to run three
     algorithms on the reference photos and get the best
     iou values for all of them; the algorithms all have
     different parameters, and i guess i wanna predefine
     possible values for them, then run my comparison
     script multiple times on each one and save the best
     iou and the corresponding parameter values; can you
     give me a simple python script as follows: assume i
     have a subfolder with all of my txt files that
     represent the ground truth; assume i have subfolders
     with all of my model references for each viewpoint;
     also assume for each object that the name of the txt
     file is the same as the name of the subfolder
     containing the reference images; give me a python
     script that, for each matching pair (txt and
     subfolder name) uses the method i already have def
     generate_comp(ref_file, gen_model) which outputs iuo
     and mse, and runs it on a predefined set of
     parameter values, and in the end, for each model
     prints the best iuo and its corresponding
     parameters; additionally, export the final(best)
     results to a csv file
```

The corresponding output:

```
import os
import csv
from itertools import product

# Import your methods
# from your_module import generate_comp, run_algorithm_1,
     run_algorithm_2, run_algorithm_3

# === Mock functions to replace ===
def run_algorithm_1(ref_images, param1, param2):
    return DummyModel()  # replace with actual logic

def run_algorithm_2(ref_images, paramA):
    return DummyModel()

def run_algorithm_3(ref_images, flag):
    return DummyModel()

class DummyModel:
    def __init__(self):
        self.colors = None  # Should be the predicted
             voxel RGBA array

# Paths
ground_truth_dir = "ground_truth"
references_dir = "references"

# Parameter grids
algorithm_configs = {
    'algorithm_1': {
        'function': run_algorithm_1,
        'params': {
            'param1': [0.1, 0.2],
            'param2': [5, 10]
        }
    },
    'algorithm_2': {
        'function': run_algorithm_2,
        'params': {
            'paramA': ['low', 'medium', 'high']
        }
    },
    'algorithm_3': {
        'function': run_algorithm_3,
        'params': {
            'flag': [True, False]
        }
```

```
    }
}

results = []

for gt_filename in os.listdir(ground_truth_dir):
    if not gt_filename.endswith('.txt'):
        continue

    model_name = gt_filename[:-4]  # Strip .txt
    ref_folder = os.path.join(references_dir, model_name)
    gt_path = os.path.join(ground_truth_dir, gt_filename)

    if not os.path.isdir(ref_folder):
        print(f"Skipping {model_name}: reference folder
            missing")
        continue

    # You may want to actually load and pass the images
        here
    ref_images = os.listdir(ref_folder)  # Placeholder

    for algo_name, config in algorithm_configs.items():
        func = config['function']
        param_keys = config['params'].keys()
        param_values =
            list(product(*config['params'].values()))

        best_iou = -1.0
        best_mse = None
        best_params = None

        for values in param_values:
            param_dict = dict(zip(param_keys, values))
            gen_model = func(ref_images, **param_dict)

            try:
                iou, mse = generate_comp(gt_path,
                    gen_model)
            except Exception as e:
                print(f"[{model_name}][{algo_name}] Error
                    with params {param_dict}: {e}")
                continue

            if iou > best_iou:
                best_iou = iou
                best_mse = mse
                best_params = param_dict

        results.append({
            'model': model_name,
            'algorithm': algo_name,
            'best_iou': best_iou,
            'mse': best_mse,
            'params': best_params
        })

        print(f"{model_name} | {algo_name} | Best IOU:
            {best_iou:.4f} | Params: {best_params}")

# Save results
csv_path = "best_results.csv"
with open(csv_path, 'w', newline='') as f:
    writer = csv.DictWriter(f, fieldnames=['model',
        'algorithm', 'best_iou', 'mse', 'params'])
    writer.writeheader()
    for row in results:
        row_copy = row.copy()
        row_copy['params'] = str(row_copy['params'])  #
            convert dict to string
        writer.writerow(row_copy)

print(f"Results saved to {csv_path}")
```

For the second experiment the prompt was:

```
i want to do another experiment; i have a set of 13
    objects, all are represented by 2D pixel art from
    the same 6 predefined viewpoints; i wanna run my
    algorithms on a subset of the pixel images, render
    the resulting model, then compare the 2d render to
    the reference for the same viewpoint to see how well
    the algorithms estimate missing information and how
    well the color merging techniques work; i first need
    a pipeline to run the algorithms on a subset of
    images and render them using an orthographic camera
    in blender; imagine i have a subfolder for each
    object in the ./exp2 directory; for each one i want
    to import a subset of images (2: top, left; top,
    front; front, left, 3: top, front, left) then render
    the results from the opposite views (i have to
    rotate the model, but assume i can have a reference
    to the camera i want to use and i might wanna adjust
    the camera's orthographic scale here), i wanna save
    the renders from each object in
    ./exp2/subfoldername_extrapolated or sth like this
```

The corresponding output is:

```
subset_to_render_views = {
    ("TOP", "LEFT"): ["BOTTOM", "RIGHT"],
    ("TOP", "FRONT"): ["BOTTOM", "BACK"],
    ("FRONT", "LEFT"): ["BACK", "RIGHT"],
    ("TOP", "FRONT", "LEFT"): ["BOTTOM", "BACK", "RIGHT"],
}

def load_subset_images(obj_folder, views):
    images = {}
    for view in views:
        img_path = os.path.join(obj_folder, view.lower()
            + ".png")
        image = bpy.data.images.load(img_path)
        w, h = image.size
        pixels_np = np.array(image.pixels[:]).reshape((h,
            w, 4))
        pixels_np = np.round(pixels_np, 3)
        images[view] = pixels_np
    return images

def voxel_grid_to_mesh(voxels, name="voxel_model"):
    import bmesh
    import mathutils

    mesh = bpy.data.meshes.new(name)
    obj = bpy.data.objects.new(name, mesh)
    bpy.context.collection.objects.link(obj)

    bm = bmesh.new()
    for x in range(voxels.shape[0]):
        for y in range(voxels.shape[1]):
            for z in range(voxels.shape[2]):
                if voxels[x, y, z]:
                    bmesh.ops.create_cube(
                        bm,
                        size=1.0,
                        matrix=mathutils.Matrix.Translation((x,
                            y, z))
                    )
    bm.to_mesh(mesh)
    bm.free()
    return obj

def render_from_view(obj, view_name, save_path,
    ortho_scale=30):
    cam = bpy.data.objects.get("OrthoCamera")
    if cam is None:
        cam_data = bpy.data.cameras.new("OrthoCamera")
        cam_data.type = 'ORTHO'
        cam = bpy.data.objects.new("OrthoCamera",
            cam_data)
        bpy.context.collection.objects.link(cam)

    cam.data.ortho_scale = ortho_scale

    # Set rotation based on view
    view_rotations = {
        "FRONT": (0, 0, 0),
        "BACK": (0, 0, math.radians(180)),
        "LEFT": (0, 0, math.radians(90)),
        "RIGHT": (0, 0, math.radians(-90)),
        "TOP": (math.radians(-90), 0, 0),
        "BOTTOM": (math.radians(90), 0, 0),
```

```
    }
    cam.rotation_euler = view_rotations[view_name]
    cam.location = (0, 0, 100)  # You can refine based on
         grid size

    bpy.context.scene.camera = cam
    bpy.context.view_layer.update()

    bpy.context.scene.render.filepath = save_path
    bpy.ops.render.render(write_still=True)

def run_experiment2(base_dir, subset_views):
    for obj_name in os.listdir(base_dir):
        obj_dir = os.path.join(base_dir, obj_name)
        if not os.path.isdir(obj_dir): continue

        subset_key = tuple(sorted(subset_views))
        render_views = subset_to_render_views[subset_key]

        images = load_subset_images(obj_dir, subset_views)
        size = size_map.get(obj_name, 20)

        # Choose algorithm here:
        voxel_grid = your_voxel_method(
            images, size, size, size,  # w, h, d
            concavity_depth=0.2,
            factor=0.5,
            min_region_size=2,
            keep_concave_regions=True
        )

        mesh_obj = voxel_grid_to_mesh(voxel_grid)
        output_dir = os.path.join(base_dir,
             f"{obj_name}_extrapolated")
        os.makedirs(output_dir, exist_ok=True)

        for view in render_views:
            render_path = os.path.join(output_dir,
                 f"{view.lower()}.png")
            render_from_view(mesh_obj, view, render_path,
                 ortho_scale=size)
```

## References


[1] U. M. Sharma and R. Chadha, "Exploring the applications and advances in 3d imaging and 3d reconstruction," in *2023 International Conference on IoT, Communication and Automation Technology (ICICAT)*, 2023, pp. 1–6. DOI: 10.1109/ICICAT57735.2023.10263701.

[2] D. Silber, *Pixel art for game developers*. Boca Raton: CRC Press, 2016, 238 pp., ISBN: 978-1-4822-5230-9.

[3] S. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski, "A comparison and evaluation of multi-view stereo reconstruction algorithms," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 1, 2006, pp. 519–528. DOI: 10.1109/CVPR.2006.19.

[4] R. O. Duda and P. E. Hart, "Pattern classification and scene analysis," in *A Wiley-Interscience publication*, 1974. [Online]. Available: https://api.semanticscholar.org/CorpusID:12946615.

[5] D. H. Ballard and C. M. Brown, *Computer Vision*. Englewood Cliffs, NJ: Prentice-Hall, 1982.

[6] A. Barr and E. A. Feigenbaum, "The handbook of artificial intelligence," in *The Handbook of Artificial Intelligence*, vol. 3, New York: Pitman, 1983, pp. 125–323.

[7] S. Ullman, "The interpretation of structure from motion," *Cognitive Science*, vol. 1, no. 1, pp. 97–122, 1979. DOI: 10.1207/s15516709cog0101_6.

[8] C. Tomasi and T. Kanade, "Shape and motion from image streams under orthography: A factorization method," *International Journal of Computer Vision*, vol. 9, no. 2, pp. 137–154, 1992. DOI: 10.1007/BF00129684.

[9] W. N. Martin and J. K. Aggarwal, "Volumetric descriptions of objects from multiple views," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-5, no. 2, pp. 150–158, 1983. DOI: 10.1109/TPAMI.1983.4767367.

[10] V. Cappellini, R. Casini, M. Pareschi, and C. Raspollini, "From multiple views to object recognition," *IEEE Transactions on Circuits and Systems*, vol. 34, no. 11, pp. 1344–1350, 1987. DOI: 10.1109/TCS.1987.1086069.

[11] N. Ahuja and J. Veenstra, "Generating octrees from object silhouettes in orthographic views," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 2, pp. 137–149, 1989. DOI: 10.1109/34.16710.

[12] A. Laurentini, "The visual hull concept for silhouette-based image understanding," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 2, pp. 150–162, 1994. DOI: 10.1109/34.273735.

[13] K. N. Kutulakos and S. M. Seitz, "A theory of shape by space carving," *International Journal of Computer Vision*, vol. 38, no. 3, pp. 199–218, 2000. DOI: 10.1023/A:1008191222954.

[14] A. Broadhurst, T. Drummond, and R. Cipolla, "A probabilistic framework for space carving," in *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, vol. 1, 2001, 388–393 vol.1. DOI: 10.1109/ICCV.2001.937544.

[15] G. Slabaugh, R. Schafer, and M. Hans, "Multi-resolution space carving using level set methods," in *Proceedings. International Conference on Image Processing*, vol. 2, 2002, pp. II–II. DOI: 10.1109/ICIP.2002.1040008.

[16] A. Yezzi, G. Slabaugh, R. Cipolla, and R. Schafer, "A surface evolution approach of probabilistic space carving," in *Proceedings. First International Symposium on 3D Data Processing Visualization and Transmission*, 2002, pp. 618–621. DOI: 10.1109/TDPVT.2002.1024126.

[17] B. K. Horn, "Shape from shading: A method for obtaining the shape of a smooth opaque object from one view," *Massachusetts Institute of Technology, Artificial Intelligence Laboratory*, 1970.

[18] Y. Fang, Z. Zhao, and C. Kambhamettu, "Depth estimation from single images using gradient scale networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 927–934.

[19] X. Li and J. Sun, "Estimating depth from single monocular images using gradient domain learning," in *Asian Conference on Computer Vision (ACCV)*, 2006, pp. 408–418.

[20] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger, "Occupancy networks: Learning 3d reconstruction in function space," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 4460–4470.

[21] M. Tatarchenko, J. J. Park, V. Koltun, and T. Brox, "Single-view to multi-view: Reconstructing unseen views with a convolutional multi-view network," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[22] Sona-Sar, *Voxel platformer pack*, https://sona-sar.itch.io/voxelplatformerpack, Accessed: 22-06-2025.

[23] Kyrise, *Voxel graveyard environment pack*, https://kyrise.itch.io/kyrises-voxel-graveyard-environment-pack, Accessed: 22-06-2025.

[24] Ephtracy, *Magicavoxel*, https://ephtracy.github.io/, Accessed: 22-06-2025.

[25] T. Krajtmajer, *Voxel blender plugin*, https://github.com/tkrajtmajer/voxel_blender_plugin, GitHub repository, Accessed: 22-06-2025.

[26] D. Carata, *Github profile*, https://github.com/demicarata, Accessed: 22-06-2025.