Article

SpaceCarvingJS: Silhouette-based 3D reconstruction using space carving in the browser

WenJun Zhang
School of Life Sciences, Sun Yat-sen University, Guangzhou 510275, China; International Academy of Ecology and Environmental Sciences, Hong Kong
E-mail: wjzhang@iaees.org, zhwj@mail.sysu.edu.cn

Received 13 July 2026; *Accepted 26 July 2026*; *Published online 10 August 2026*; *Published 1 December 2028*

Abstract
SpaceCarvingJS is a fully client-side, browser-based 3D reconstruction tool that recovers a visual-hull model from turntable rotation videos or ordered image sequences using silhouette segmentation and space carving. Entirely self-contained, it requires no external libraries, servers, WebGL, or cloud services; all computation runs locally in the user's browser. The pipeline consists of frame sampling with adaptive letterboxing, binary silhouette extraction via background-colour distance or luminance thresholding (with Otsu auto-threshold, morphological opening/closing, largest-component filtering, and hole filling), estimation of turntable geometry (rotation axis, radius, and vertical extent), voxel-based space carving under orthographic or perspective projection with a user-adjustable consistency threshold, isosurface extraction using a table-free Surface Nets algorithm, and Taubin mesh smoothing. The tool supports video (MP4, WebM, MOV, etc.) and image (PNG, JPEG, etc.) inputs, as well as a built-in synthetic demo for testing. Users can interactively tune segmentation parameters, voxel resolution (typically 112–224 per axis), consistency thresholds, smoothing iterations, and projection mode. A software rasterizer provides real-time 3D preview without WebGL, and the final mesh can be exported as OBJ or PLY with optional vertex normals and normalized height. The method reconstructs the visual hull, the intersection of all silhouette cones, which faithfully represents exterior surfaces visible from the input views but cannot recover concavities that remain hidden in every silhouette. It does not estimate texture, material properties, or precise camera calibration. With a 112^3 grid, memory usage is approximately 45 MB for the signed volume, and computational cost scales with voxel count and number of views. SpaceCarvingJS is particularly suited for teaching, rapid prototyping, and rough modeling of objects on a turntable with uniform background and stable lighting. It offers a transparent, dependency-free, and fully interactive implementation of the classical space-carving pipeline, making it a valuable educational and research tool for visual-hull reconstruction.

Keywords 3D reconstruction; visual hull; space carving; silhouette segmentation; voxel; Surface Nets; turntable; browser-based; dependency-free; OBJ/PLY export.

Selforganizology
ISSN 2410-0080
URL: http://www.iaees.org/publications/journals/selforganizology/online-version.asp
RSS: http://www.iaees.org/publications/journals/selforganizology /rss.xml
E-mail: selforganizology@iaees.org
Editor-in-Chief: WenJun Zhang
Publisher: International Academy of Ecology and Environmental Sciences

1 Introduction

Three-dimensional reconstruction from two-dimensional imagery is a fundamental and long-standing challenge in computer vision, with critical applications spanning robotics, cultural heritage preservation, medical imaging, industrial inspection, and virtual reality. Among the numerous approaches, silhouette-based methods have maintained a distinctive position owing to their robustness and minimal requirements: they rely solely on object-background segmentation, bypassing the need for texture features, correspondence matching, or complex calibration patterns. The theoretical foundation of this family is the visual hull, the maximal shape consistent with the silhouettes observed from all viewpoints, which is obtained by intersecting the back-projected silhouette cones. This concept was first rigorously defined by Laurentini (1994). The subsequent space carving algorithm, formalized by Kutulakos and Seitz (2000), operates on a volumetric voxel grid and iteratively eliminates (carves) voxels that are not photo-consistent with the input silhouettes, thereby producing the photo hull, which is generally tighter than the visual hull. Efficient variants, such as the image-based visual hull introduced by Matusik et al. (2000), have enabled real-time performance for certain configurations.

Nevertheless, silhouette-based reconstruction, including space carving, suffers from inherent limitations. The quality of reconstruction is highly sensitive to the accuracy of silhouette extraction and camera calibration; any segmentation errors propagate directly into the volumetric model. More fundamentally, concave surface regions that remain hidden from all viewing directions cannot be recovered, because the silhouette information alone is insufficient to infer depth variations in occluded areas. Additionally, the placement and number of cameras can introduce synthetic bulges or ghost volumes. To mitigate these issues, researchers have proposed various enhancements, such as enforcing exact silhouette constraints alongside photo-consistency (Sinha & Pollefeys, 2005) and adopting hierarchical volumetric representations to improve efficiency and robustness (Hornung & Kobbelt, 2006). Despite these advances, most existing implementations rely on specialised hardware, external libraries, or server-side processing, which limits accessibility and reproducibility.

Concurrently, the field of web-based 3D visualisation has seen substantial progress. Zhang (2023) developed an integrative Java software for interactive 3D visualisation of objects and molecules, providing a flexible platform for scientific exploration. Zhang (2024) later introduced a standalone executable tool for 3D object rendering with support for standard formats. Most recently, Zhang (2028) presented a browser-based interactive visualiser that operates without external libraries or WebGL, leveraging pure HTML/CSS/JavaScript. However, these tools are designed primarily for viewing pre-existing 3D models, not for reconstructing them from image sequences. There remains a notable gap: a fully self-contained, client-side implementation of the complete space carving pipeline that runs entirely in the user's browser, requiring no server, cloud service, or graphics hardware acceleration.

For this reason, this paper develops a browser-based tool, SpaceCarvingJS. It implements the entire visual-hull reconstruction workflow: frame sampling with adaptive letterboxing, binary silhouette extraction via background-colour distance or luminance thresholding (with Otsu auto-threshold, morphological opening/closing, largest-component filtering, and hole filling), automatic estimation of turntable geometry (rotation axis, radius, and height), voxel-based space carving under either orthographic or perspective projection with a user-adjustable consistency threshold, isosurface extraction using a table-free Surface Nets algorithm, and Taubin mesh smoothing. The tool accepts video (MP4, WebM, MOV) and image (PNG, JPEG) inputs, and includes a built-in synthetic demo for testing. Users can interactively tune segmentation parameters, voxel resolution (typically 112–224 per axis), consistency thresholds, smoothing iterations, and projection modes. By offering a transparent, dependency-free, and fully interactive implementation of the classical space-carving pipeline, SpaceCarvingJS is particularly suited for teaching, rapid prototyping, and rough

modelling of turntable objects with uniform backgrounds and stable lighting. It serves as both an educational resource and a practical platform for visual-hull reconstruction research.

2 Technical Report: Silhouette → 3D Reconstruction → OBJ

2.1 Overview

It is a complete standalone HTML application for reconstructing a 3D object from:

1. A turntable rotation video, or
2. A sequence of silhouette images, or
3. A built-in synthetic demonstration object.

The application implements a simplified visual hull reconstruction pipeline entirely in the browser:

Input frames→silhouette segmentation→turntable geometry estimation→voxel space carving→surface extract ion→mesh smoothing→OBJ/PLY export

It does not use external JavaScript libraries, servers, WebGL, machine-learning models, or cloud services. All processing occurs locally in the browser.

The reconstructed model is a visual hull, not a complete photogrammetric or volumetric recovery. Consequently, it can represent the exterior shape visible in silhouettes, but it cannot recover concave regions that are invisible from every supplied view.

2.2 Application Purpose

The application is intended for quick, dependency-free reconstruction of an object placed on a turntable.

Typical use cases include:

- Creating a rough 3D model from a rotating object video.
- Converting a collection of binary silhouette images into a mesh.
- Demonstrating voxel-based shape reconstruction.
- Exporting a visual-hull model to .OBJ or .PLY.
- Teaching image segmentation, space carving, surface nets, and mesh processing.

The application is especially suitable for objects with:

- A static camera.
- A roughly vertical rotation axis.
- A plain background.
- Strong contrast between object and background.
- Approximately uniform lighting.
- One complete or known partial rotation.

It is not designed to recover:

- Texture.
- Material properties.
- Fine hidden details.
- Deep concavities.
- Accurate camera calibration.
- Exact physical dimensions without scaling.

2.3 Main Features

2.3.1 Input Support

The file input accepts:

- MP4, WebM, MOV, M4V, OGV, AVI, and MKV-like video filenames.
- PNG, JPEG, BMP, GIF, and WebP images.

The input interface supports:

- File selection.
- Drag-and-drop.
- Loading multiple images.
- Loading a synthetic demo.
- Clearing the current project.

For videos, frames are sampled uniformly between the selected trim percentages.

For image sequences, files are naturally sorted by filename:

javascript

```
files=[...files].sort((a,b)=>
  a.name.localeCompare(b.name,undefined,{numeric:true})
);
```

This is important because image filenames are assumed to encode the rotation order, for example:

text

```
object_000.png
object_030.png
object_060.png
...
```

2.3.2 Adjustable Processing Resolution

The working resolution controls the common canvas size used for all frames.

The application scales the input so that its largest dimension does not exceed the requested processing size:

$$s=\min(1, P/\max(w,h))$$

where:

- w, h are the original image dimensions.
- P is the selected processing size.
- s is the scale factor.

The scaled dimensions are:

$$W=\text{round}(ws),\ H=\text{round}(hs)$$

All frames are then letterboxed into a common $W\times H$ canvas.

2.3.3 Segmentation Methods

Three segmentation modes are available:

1. **Auto: distance to border color**
2. **Bright object / dark background**
3. **Dark object / bright background**

Additional segmentation controls include:

- Otsu automatic thresholding.
- Manual threshold.
- Threshold bias.
- Pre-blur.
- Morphological opening.

- Morphological closing.
- Largest-component filtering.
- Hole filling.
- Mask inversion.

2.3.4 Turntable Geometry Controls

The reconstruction geometry can be controlled by:

- Total rotation angle.
- Closed 360-degree loop.
- Rotation direction.
- Custom angle list.
- Rotation-axis horizontal shift.
- Orthographic or perspective projection.
- Camera distance for perspective projection.

2.3.5 Volumetric Reconstruction

The volumetric stage supports:

- Adjustable grid resolution.
- Silhouette consistency threshold.
- 3D volume smoothing.
- Taubin mesh smoothing.
- Boundary sealing.
- Horizontal mirroring.

The default volumetric grid resolution is $112\times n_y\times 112$, where n_y is automatically computed from the object's vertical extent and voxel size.

2.3.6 Export

The generated model can be exported as:

- Wavefront OBJ.
- ASCII PLY.

The OBJ exporter can optionally write vertex normals.

The output model is normalized to a selected height, defaulting to:

$H_{\text{out}}=1$

The model is translated so that its lowest point is at $Y=0$.

2.4 Application Architecture

The program is organized into the following logical stages.

2.4.1 State Object

The global state is stored in object S:

javascript

```
const S = {
  mode:null,
  frames:[],
  views:[],
  W:0,
  H:0,
  geom:null,
  mesh:null,
```

```
  sel:0
};
```

Its major fields are:

Field	Meaning
mode	'video', 'images', or 'demo'
frames	Preprocessed image canvases
views	Segmented silhouettes and statistics
W, H	Common processing dimensions
geom	Estimated turntable geometry
mesh	Final reconstructed mesh
sel	Currently selected inspection frame

Each view has approximately the following structure:

```
javascript
{
  mask: Uint8Array,
  area,
  cx,
  cy,
  x0,
  x1,
  y0,
  y1,
  angle,
  thr
}
```

2.5 Input Processing

2.5.1 Canvas Fitting and Letterboxing

The helper function fitCanvas() converts each input frame into a common canvas:

```
javascript
function fitCanvas(src,sw,sh,W,H){
  const cv=document.createElement('canvas');
  cv.width=W;
  cv.height=H;

  const g=cv.getContext('2d',{willReadFrequently:true});
  const s=Math.min(W/sw,H/sh);
  const dw=sw*s;
  const dh=sh*s;

  g.fillStyle='#000';
  g.fillRect(0,0,W,H);
  g.drawImage(src,(W-dw)/2,(H-dh)/2,dw,dh);
  return cv;
```

}

The object is centered inside a black letterboxed image. This ensures that all frames share identical pixel dimensions.

2.5.2 Video Sampling

For a video of duration T, with trim fractions p_0, p_1 , the temporal interval is:

$$a=Tp_0,b=Tp_1$$

If a closed loop is selected, the sampled frame time for frame i is:

$$t_i=a+i(b-a)/N$$

where N is the number of frames.

The endpoint $i=N$ is intentionally excluded in the loop case because it duplicates the first frame.

For a non-loop sequence:

$$t_i=a+ i(b-a)/(N-1)$$

for $N>1$.

Representative code:

javascript

```
const t=clamp(
   a+(b-a)*(loop ? i/count : (count>1 ? i/(count-1) : 0)),
   0,
   Math.max(0,dur-1e-3)
);
```

The final 1 ms1\text{ ms}1 ms safety margin avoids seeking exactly to the video duration.

2.5.3 Image Loading

Images are decoded with createImageBitmap() and resized into common canvases.

The image order is determined by natural filename sorting. Therefore, users should name images consistently.

2.5.4 Synthetic Demo

The demo creates silhouettes by ray marching through an implicit object.

The object is defined using several primitives:

Ellipsoidal body

$$(x/0.55)^2+(y/0.80)^2+(z/0.42)^2<1$$

Spherical head

$$x^2+(y-0.95)^2+z^2<0.34^2$$

Small offset nose

$$x^2+(y-0.98)^2+(z-0.40)^2<0.13^2$$

Arm-like rectangular region

$|x|<0.95,\ |y-0.10|<0.09,\ |z|<0.10$

Legs

For each leg:

$|x\mp 0.15|<0.13,\ -1.25<y<-0.60,\ |z|<0.13$

Each pixel is tested along a depth ray. If any sampled depth position is inside the implicit object, the pixel becomes foreground.

The demo therefore provides a complete test input without requiring external files.

2.6 Silhouette Segmentation

2.6.1 Image Data Representation

Each frame is read with:

javascript

const data=g.getImageData(0,0,W,H).data;

The array contains RGBA values:

(R,G,B,A)

with each color channel in the interval [0,255][0,255][0,255].

The segmentation procedure creates a scalar field:

$f(x,y)\in[0,1]$

and thresholds it to obtain a binary mask:

$M(x,y)=1, \quad f(x,y)>\tau$

$M(x,y)=0, \quad \text{otherwise}$

2.6.2 Background-Distance Segmentation

In the automatic background mode, the program estimates the background color from the image border.

A border width is selected as:

$b=\max(2,\text{round}(0.02\min(W,H)))$

Pixels outside the inner rectangle are collected. The median of each channel is used:

$B=(B_R,B_G,B_B)$

For a pixel with color $C=(R,G,B)$, the normalized Manhattan color distance is:

$f(x,y)=(|R-B_R|+|G-B_G|+|B-B_B|)/765$

because $3 \cdot 255=765$
This makes pixels different from the border color brighter in the scalar field.
Representative code:
javascript

```
const [br,bg_,bb]=medianBorderColor(data,W,H);

for(let i=0,o=0;i<n;i++,o+=4){
  f[i]=(
    Math.abs(data[o]-br)+
    Math.abs(data[o+1]-bg_)+
    Math.abs(data[o+2]-bb)
  )/765;
}
```

This method is effective when the background is relatively uniform and touches the image boundaries.

2.6.3 Bright and Dark Segmentation

For bright-object mode, the luminance is:

$$L=0.299R+0.587G+0.114B$$

normalized to:

$$\ell=L/255$$

For bright objects:

$$f(x,y)=\ell$$

For dark objects:

$$f(x,y)=1-\ell$$

The code uses:
javascript

```
const L=(0.299*data[o]+0.587*data[o+1]+0.114*data[o+2])/255;
f[i]=P.segMode==='bright' ? L : 1-L;
```

2.6.4 Pre-Blur

The scalar field can be smoothed with a separable box blur.
For radius rrr, the one-dimensional horizontal pass computes:

$$g(x,y)=\sum_{k=-r}^{r} f(x+k,y)/N_x$$

where only valid image coordinates are included.
The vertical pass then computes:

$h(x,y)=\sum_{k=-r}^{r} g(x,y+k)/N_y$

This is implemented in two passes for lower computational complexity than a full two-dimensional kernel.

2.6.5 Otsu Thresholding

When automatic thresholding is selected, the normalized scalar field is quantized into 256 histogram bins.

Let $h(t)$ be the number of pixels in bin ttt, with $t \in \{0,\ldots,255\}$

For a candidate threshold t:

- w_B : number of background-class pixels.
- w_F: number of foreground-class pixels.
- μ_B: mean intensity of the background class.
- μ_F: mean intensity of the foreground class.

The between-class variance is:

$\sigma_B^2(t)= w_B w_F\ (\mu_B-\mu_F)^2$

Otsu's method chooses:

$t^*=\arg\ \max_t\ \sigma_B^2(t)$

The normalized threshold is then multiplied by the user-selected bias:

$\tau=\text{clamp}(\beta\ t^*/255,0.004,0.996)$

where β is thrBias.

For manual thresholding, the program maps the bias control to:

$\tau=\text{clamp}(\beta/2.5,0.004,0.996)$

2.6.6 Mask Inversion

After thresholding, the mask can be inverted:

$M'(x,y)=1-M(x,y)$

Code:

javascript

```
if(P.invert)
   for(let i=0;i<n;i++) m[i]^=1;
```

2.6.7 Morphological Opening and Closing

The morph() function performs binary dilation or erosion using a 3×3 neighborhood.

Dilation

A pixel becomes foreground if at least one neighborhood pixel is foreground:

$(M \oplus K)(x,y)=\max_{(u,v)\in K} M(x-u,y-v)$

Erosion

A pixel remains foreground only if all neighborhood pixels are foreground:

$$(M \ominus K)(x,y)=\max_{(u,v)\in K} M(x-u,y-v)$$

The structuring element is the 3×3 square:

$$K=\{-1,0,1\}\times\{-1,0,1\}$$

Opening is:

$$M \circ K=(M \ominus K) \oplus K$$

It removes small isolated foreground regions.

Closing is:

$$M \cdot K=(M \oplus K) \ominus K$$

It fills small gaps and seals narrow breaks.

2.6.8 Largest Connected Component

The application optionally retains only the largest 4-connected foreground component.

For each foreground pixel, neighbors are:

$$(x+1,y),(x-1,y),(x,y+1),(x,y-1)$$

A stack-based flood-fill labels each connected component. The component with the greatest number of pixels is retained.

This is useful when shadows, reflections, or noise create additional foreground blobs.

2.6.9 Hole Filling

Hole filling identifies background pixels connected to the image boundary. Any background pixel not connected to the boundary is considered an interior hole and converted to foreground.

Mathematically, if B is the set of background pixels and B_{∂} is the boundary-connected subset, then filled foreground is:

$$M_{\text{filled}}=M \cup (B \setminus B_{\partial})$$

This assumes that enclosed background regions are segmentation holes rather than intended object cavities.

2.6.10 Silhouette Statistics

For each final mask, the program computes:

- Area.
- Centroid.
- Bounding box.

Area:

$A=\sum_{x,y} M(x,y)$

Centroid:

$c_x=\sum_{x,y} xM(x,y)/A$
$c_y=\sum_{x,y} yM(x,y)/A$

Bounding coordinates are:

$x_0=\min\{x:M(x,y)=1\}$
$x1=\max\{x:M(x,y)=1\}$
$y0=\min\{y:M(x,y)=1\}$
$y1=\max\{y:M(x,y)=1\}$

These values are later used to estimate the rotation axis, object radius, and vertical extent.

2.7 Turntable Geometry Estimation

2.7.1 Rotation Axis

The rotation axis is assumed to be vertical in the image. Its horizontal coordinate is estimated from the mean silhouette centroid:

$x_{\text{axis}}=\sum_{i=1}^{K} c_{x,i}/K \ +\Delta_x$

where:

- K is the number of nonempty views.
- $c_{x,i}$ is the centroid x-coordinate of view i.
- Δ_x is the user-controlled axis shift.

Code:

```
javascript
let sx=0,k=0;
for(const v of V)
  if(v.area>0){ sx+=v.cx; k++; }

let axisX=(k?sx/k:W/2)+P.axisShift;
```

This works best when the object rotates around a fixed axis and the camera is centered.

2.7.2 Rotation Radius

The radius is estimated from the maximum horizontal silhouette extent relative to the axis:

$R=1.04\cdot\max_i\left(|x_{0,i}-x_{\text{axis}}|,|x_{1,i}-x_{\text{axis}}|\right)+2$

The additional factor and constant provide a small safety margin.

2.7.3 Vertical Extent

The global object vertical extent is:

$y_{\text{top}}=\min_i y_{0,i}-2$

$y_{\text{bottom}}=\max_i y_{1,i}+2$

The values are clamped to the image boundaries.

The algorithm assumes that the object remains vertically aligned while rotating. The vertical axis is not estimated from perspective distortion or camera calibration.

2.8 Assigning View Angles

If the user supplies a valid comma-separated angle list, its length must equal the number of views.

Otherwise, angles are assigned uniformly.

For a closed loop:

$\theta_i=i\Theta/N$

For an open sequence:

$\theta_i=i\Theta/(N\text{-}1)$

where:

- Θ is total rotation in degrees.
- N is the number of views.
- $i\in[0,N-1]$.

The direction control is applied later during carving:

$\theta_i^{\text{effective}}=d\theta_i$

where $d\in\{+1,-1\}$

2.9 Coordinate System

The voxel volume is defined in a local coordinate system:

- x: horizontal distance from the rotation axis.
- y: vertical image direction.
- z: depth relative to the object's rotation axis.

The voxel grid is centered around the estimated axis:

$x\in[-R,R]$
$z\in[-R,R]$

The image y-coordinate is initially downward, so the final world-space y-coordinate is inverted:

$Y=-(y_{\text{top}}+j\Delta y)$

The model is later translated and scaled so that its base is at $Y=0$.

2.10 Voxel Space Carving

2.10.1 Grid Dimensions

The horizontal and depth resolutions are both selected by the user:

$n_x=n_z=N_{\text{res}}$

The horizontal and depth voxel spacing is:

$\Delta x=\Delta z=2R/(n_x-1)$

The vertical resolution is determined from the object height:

$n_y=\text{clamp}(\text{round}((y_{\text{bottom}}-y_{\text{top}})/\Delta x)+1,4,512)$

The vertical spacing is:

$\Delta y=(y_{\text{bottom}}-y_{\text{top}})/(n_y\text{-}1)$

The total number of voxels is:

$N_{\text{voxels}}=n_x n_y n_z$

2.10.2 Voxel Coordinates

The coordinate arrays are initialized as:

$x_i=-R+i\Delta x$
$z_k=-R+k\Delta z$
$y_j=y_{\text{top}}+j\Delta y$

Every voxel stores a count indicating how many silhouettes contain its projection.

2.10.3 Rotation Transformation

For view angle θ\thetaθ, the voxel coordinates are transformed into the camera-view coordinate system.
The code uses:
javascript
const xr=x*c+z*s;
const zr=-x*s+z*c;
where:

$c=\cos\theta, s=\sin\theta$

Thus:

$x_r=x\cos\theta+z\sin\theta$
$z_r=-x\sin\theta+z\cos\theta$

This is a rotation around the vertical y-axis.
The transformed horizontal coordinate is projected into image coordinates using the estimated axis:

$u=x_{\text{axis}}+x_r$

under orthographic projection.

2.10.4 Orthographic Projection

For orthographic projection:

$u=x_{\text{axis}}+x_r$
$v=y$

Since all points along a depth ray project to the same image position, the silhouette test is independent of z_r.
The corresponding pixel coordinates are rounded:

$u_{\text{pix}}=\text{round}(u)$
$v_{\text{pix}}=\text{round}(y_j)$

A voxel receives support from the view if:

$M_i(v_{\text{pix}}, u_{\text{pix}})=1$

2.10.5 Perspective Projection

For perspective projection, the camera distance is:

$D=\max(1.6,P_{\text{camDist}})R$

The scale factor is:

$s=D/(D-z_r)$

The projected image coordinates are:

$u=x_{\text{axis}}+x_r s$
$v=v_c+(y-v_c)s$

where:

$v_c=(y_{\text{top}}+y_{\text{bottom}})/2$

The expression $D-z_r$ represents the remaining distance from the point to the camera under the chosen coordinate convention.
The perspective test is therefore:

$M_i(\text{round}(v),\text{round}(u))=1$

2.10.6 Consistency Counting

For every voxel and every usable view, the program increments a count if that view's silhouette contains the projected voxel.
Let:

$C(x,y,z)=\sum_{i=1}^{K} M_i\ (\prod_i(x,y,z))$

where:

- K is the number of nonempty views.
- Π_i is the projection function for view i.

The normalized occupancy field is:

$F(x,y,z)= C(x,y,z)/K$

Thus $F \in [0,1]$.
A point that projects inside every silhouette has $F=1$. A point inconsistent with many silhouettes has a smaller value.

2.10.7 Consistency Threshold

The user-selected consistency threshold qqq determines occupancy:

$\text{inside}(x,y,z) \iff F(x,y,z) \geq q$

The signed scalar field used for surface extraction is:

$\phi(x,y,z)=q-F(x,y,z)$

Therefore:

- $\phi<0$: inside.
- $\phi =0$: reconstructed surface.
- $\phi >0$: outside.

The code implements:

```
javascript
for(let i=0;i<N;i++){
   data[i]=P.cons-field[i];
   if(data[i]<0) solid++;
}
```

A high threshold gives a stricter visual hull and usually produces a smaller model. A lower threshold tolerates segmentation inconsistencies but can inflate the model or create artifacts.

2.10.8 Boundary Sealing

If boundary sealing is enabled, all voxels on the volume boundary are forced outside:

```
javascript
if(
   i===0 || j===0 || k===0 ||
   i===nx-1 || j===ny-1 || k===nz-1
```

```
){
   field[...] = 0;
}
```

Because the signed field is:

$\phi=q-F$

setting $F=0$ yields:

$\phi=q>0$

which prevents the extracted surface from reaching the outer grid boundary.

2.11 Volume Smoothing

The occupancy field can be smoothed using a separable 3D 1-2-1 filter.

Along one dimension, the kernel is:

$K=[1/4,1/2,1/4]$

For the x-direction:

$F_x(i,j,k)=F(i-1,j,k)/4+F(i,j,k)/2+F(i+1,j,k)/4$

The same operation is applied along y and z:

$F_{\text{smooth}}=K_z*K_y*K_x*F$

Each pass is separable and therefore more efficient than directly applying a 3×3×3 kernel.

The smoothing is repeated according to volSm.

At volume boundaries, out-of-range samples are treated as zero. This slightly biases the field toward the exterior near the boundary.

2.12 Surface Extraction: Surface Nets

2.12.1 Principle

The application converts the signed voxel field into a triangle mesh using a table-free variant of **Surface Nets**.

Each grid cell consists of eight corner samples:

$\phi_0,\phi_1,\ldots,\phi_7$

A cell is ignored if all eight values have the same sign:

mask=0 or mask=255

Otherwise, the isosurface crosses the cell.

2.12.2 Cell Sign Mask

The sign mask is constructed as:

$\text{mask}=\sum_{g=0}^{7}\mathbf{I}[\phi_g<0]\,2^g$

where $\mathbf{I}$ is the indicator function.

The code uses:

javascript

mask |= (p<0) ? (1<<g) : 0;

2.12.3 Edge Interpolation

For each cell edge whose endpoints have different signs, the zero crossing is linearly interpolated.

Given edge endpoint values g_0 and g_1, the interpolation parameter is:

$t=g_0/(g_0 - g_1)$

The edge crossing position is:

$p(t)=(1-t)p_0+tp_1$

The code uses:

javascript

let t=g0-g1;

if(Math.abs(t)>1e-8) t=g0/t;

Because the local edge length is one grid unit, the resulting crossing coordinates are in cell-local grid coordinates.

2.12.4 Surface-Net Vertex

Unlike marching cubes, which generally produces several vertices per cell, Surface Nets places one representative vertex inside each intersected cell.

If the cell has E intersected edges with crossing positions p_e, the vertex is:

$v_{\text{cell}}=\sum_{e=1}^{E} p_e/E$

This produces a compact mesh and avoids a large lookup table.

2.12.5 Quad-to-Triangle Construction

Adjacent surface-net vertices are connected around sign-changing grid edges. Each quadrilateral is split into two triangles.

Depending on the sign configuration, the winding order is chosen in one of two ways:

javascript

faces.push([buffer[m],buffer[m-du-dv],buffer[m-du]]);

faces.push([buffer[m],buffer[m-dv],buffer[m-du-dv]]);

or:

javascript

faces.push([buffer[m],buffer[m-du-dv],buffer[m-dv]]);

faces.push([buffer[m],buffer[m-du],buffer[m-du-dv]]);

The code also computes a field-gradient-based orientation vote to determine whether the entire mesh needs a winding reversal.

2.13 Conversion from Grid Coordinates to Model Coordinates

Surface-net vertices initially exist in grid coordinates:

(i,j,k)

They are converted into pixel-space coordinates:

$x_{px}=-R+I\,\Delta x$
$y_{px}=y_{top}+j\,\Delta y$
$z_{px}=-R+k\,\Delta x$

The final pre-normalized world coordinates are:

$X=s_x x_{px}$
$Y=-y_{px}$
$Z=z_{px}$

where:
$s_x=-1$,if horizontal mirroring is enabled,otherwises $s_x=1$
The y-coordinate is negated because screen coordinates increase downward while 3D coordinates conventionally increase upward.

2.14 Mesh Orientation

The signed field increases from inside to outside:

$\phi=q-F$

Therefore, the outward direction approximately follows the gradient:

$\nabla\phi=(\partial\phi/\partial x,\ \partial\phi/\partial y,\ \partial\phi/\partial z)$

The implementation estimates the gradient using central finite differences:

$\partial\phi/\partial x\approx\phi(x+1,y,z)-\phi(x-1,y,z)$
$\partial\phi/\partial y\approx\phi(x,y+1,z)-\phi(x,y-1,z)$
$\partial\phi/\partial z\approx\phi(x,y,z+1)-\phi(x,y,z-1)$

For each sampled triangle, the geometric normal is:

$n=(B-A)\times(C-A)$

The program checks:

$n\ \nabla\phi$

If the majority of sampled triangles disagree with the field gradient, it reverses triangle winding.
The flipX option changes handedness by reflecting the x-coordinate and compensating for the expected winding change.

2.15 Taubin Mesh Smoothing

The mesh smoother uses a two-step Taubin-style filter.
For each vertex i, let $N(i)$ be its neighboring vertices. The local average is:

$\bar{p}_i = \sum_{j \in N(i)} p_j / |N(i)|$

The displacement is:

$d_i = \bar{p}_i - p_i$

The first smoothing step is:

$p_i' = p_i + \lambda d_i$

with λ=0.55.
The second reverse step is:

$p_i'' = p_i' + \mu(\bar{p}_i' - p_i')$

with μ=−0.58.
This two-pass method reduces high-frequency noise while limiting the strong volume shrinkage associated with ordinary Laplacian smoothing.
The process is repeated meshSm times.

2.16 Vertex Normal Calculation

For each triangle (a,b,c)

$u = p_b - p_a$
$v = p_c - p_a$
$n_f = u \times v$

The face normal is accumulated into all three vertex normals:

$n_a += n_f, n_b += n_f, n_c += n_f$

Each accumulated normal is normalized:

$n_i \leftarrow n_i / \| n_i \|$

The implementation uses:
javascript
const nx=uy*vz-uz*vy;
const ny=uz*vx-ux*vz;
const nz=ux*vy-uy*vx;
and then normalizes each vertex vector.

2.17 Model Normalization and Scaling

After mesh construction, the program computes the model bounds:

$x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max}$

The raw height is:

$$h_{raw} = y_{max} - y_{min}$$

The scale factor is:

$$k = H_{requested} / h_{raw}$$

The model center in x and z is:

$$c_x = (x_{min} + x_{max})/2$$
$$c_z = (z_{min} + z_{max})/2$$

Each vertex is transformed to:

$$X' = k(X - c_x)$$
$$Y' = k(Y - y_{min})$$
$$Z' = k(Z - c_z)$$

As a result:

- The model has the requested height.
- The base lies at $Y=0$.
- The horizontal center is approximately at the origin.

2.18. 3D Preview Renderer

The preview does not use WebGL. It is a software rasterizer implemented with a 2D canvas.

2.18.1 View Rotation

The current view uses yaw and pitch.

For yaw α:

$$x_1 = x\cos\alpha + z\sin\alpha$$
$$z_1 = -x\sin\alpha + z\cos\alpha$$

For pitch β:

$$y_2 = y\cos\beta - z_1\sin\beta$$
$$z_2 = y\sin\beta + z_1\cos\beta$$

2.18.2 Perspective Screen Projection

The preview uses a perspective-like projection:

$$w = D - z_2$$

$x_{screen}=f\,x_1/w+W/2+pan_x$
$y_{screen}=-f\,y_2/w+H/2+pan_y$

where:

- D is the preview camera distance.
- f is a focal scale based on canvas size.
- W,H are preview dimensions.

2.18.3 Z-Buffering

A depth buffer stores the closest projected triangle depth for each pixel.

For a triangle with barycentric coordinates $(\lambda_0,\lambda_1,\lambda_2)$:

$z=\lambda_0 z_a+\lambda_1 z_b+\lambda_2 z_c$

A pixel is written only when the current triangle is closer than the stored depth.

2.18.4 Flat Lighting

A triangle normal is computed in camera space and normalized. Lighting uses a fixed directional vector:

$L=(0.38,0.55,0.74)$

The diffuse term is:

$d=|n\,L|$

The displayed intensity is approximately:

$I=0.16+0.84d$

A specular-like term is added:

$s=0.35d^{18}$

The resulting color channels are generated from weighted variants of the lighting value.

2.19 Export Formats

2.19.1 OBJ

The OBJ exporter writes:

- Vertex positions using v.
- Vertex normals using vn, if enabled.
- Triangle faces using f.

With normals enabled, a face is written as:

text

f v1//n1 v2//n2 v3//n3

The vertex and normal indices are one-based, as required by OBJ.

The file also includes metadata comments such as:

text

views=36 grid=112 consistency=0.97

vertices=... triangles=...

2.19.2 PLY

The PLY exporter writes ASCII PLY with:

- Vertex positions.
- Vertex normals.
- Triangle vertex indices.

The header contains the vertex and face counts:

text

element vertex N

...

element face M

property list uchar int vertex_indices

PLY normals are always written by this implementation.

2.20 Representative JavaScript Code

2.20.1 Complete Segmentation Core

javascript

```
function segment(cv,W,H,P){
  const g=cv.getContext('2d',{willReadFrequently:true});
  const data=g.getImageData(0,0,W,H).data;
  const n=W*H;
  let f=new Float32Array(n);

  if(P.segMode==='bg'){
    const [br,bg_,bb]=medianBorderColor(data,W,H);
    for(let i=0,o=0;i<n;i++,o+=4){
      f[i]=(
        Math.abs(data[o]-br)+
        Math.abs(data[o+1]-bg_)+
        Math.abs(data[o+2]-bb)
      )/765;
    }
  } else {
    for(let i=0,o=0;i<n;i++,o+=4){
      const L=(
        0.299*data[o]+
        0.587*data[o+1]+
        0.114*data[o+2]
      )/255;

      f[i]=P.segMode==='bright' ? L : 1-L;
    }
  }

  f=boxBlur(f,W,H,P.blur);
```

```
const thr=P.thrMode==='otsu'
   ? clamp(otsu(f,n)*P.thrBias,0.004,0.996)
   : clamp(P.thrBias/2.5,0.004,0.996);

let m=new Uint8Array(n);
for(let i=0;i<n;i++) m[i]=f[i]>thr ? 1 : 0;

if(P.invert)
   for(let i=0;i<n;i++) m[i]^=1;

if(P.mOpen){
   m=morph(m,W,H,P.mOpen,-1);
   m=morph(m,W,H,P.mOpen,1);
}

if(P.mClose){
   m=morph(m,W,H,P.mClose,1);
   m=morph(m,W,H,P.mClose,-1);
}

if(P.largest) m=keepLargest(m,W,H);
if(P.fill) m=fillHoles(m,W,H);

let area=0,sx=0,sy=0;
let x0=W,x1=-1,y0=H,y1=-1;

for(let y=0;y<H;y++){
   for(let x=0;x<W;x++){
      if(!m[y*W+x]) continue;
      area++;
      sx+=x;
      sy+=y;
      x0=Math.min(x0,x);
      x1=Math.max(x1,x);
      y0=Math.min(y0,y);
      y1=Math.max(y1,y);
   }
}

return {
   mask:m,
   area,
   cx:area ? sx/area : W/2,
   cy:area ? sy/area : H/2,
```

```
      x0,x1,y0,y1,thr
   };
}
```

2.20.2 Angle Assignment

```
javascript
function assignAngles(P,n){
   if(P.anglesTxt.trim()){
      const a=P.anglesTxt
         .split(/[,;\s]+/)
         .filter(s=>s.length)
         .map(Number);

      if(a.length===n && a.every(v=>isFinite(v))){
         return a;
      }
   }

   const out=[];
   for(let i=0;i<n;i++){
      out.push(
         P.loop
            ? P.totalRot*i/n
            : P.totalRot*(n>1 ? i/(n-1) : 0)
      );
   }
   return out;
}
```

2.20.3 Core Voxel Test

```
javascript
const th=P.dir*v.angle*Math.PI/180;
const c=Math.cos(th), s=Math.sin(th);

const xr=x*c+z*s;
const zr=-x*s+z*c;

if(P.proj==='persp'){
   const wd=camD-zr;
   if(wd<=1e-3) continue;

   const sc=camD/wd;
   u=axisX+xr*sc;
   vv=Math.round(vc+(ys[j]-vc)*sc);
} else {
   u=axisX+xr;
```

```
   vv=Math.round(ys[j]);
}

const ui=Math.round(u);

if(ui>=0 && ui<W && vv>=0 && vv<H && mask[vv*W+ui]){
   count++;
}
```

2.20.4 Signed Volume Field

javascript

```
for(let i=0;i<N;i++){
   field[i]=count[i]/numberOfValidViews;
   data[i]=P.cons-field[i];
}
```

This compactly represents the visual hull criterion.

2.21 User Guide

2.21.1 Preparing Input

For best results:

1. Place the object on a turntable.
2. Keep the camera fixed.
3. Rotate the object around one vertical axis.
4. Use one complete 360-degree rotation.
5. Use an uncluttered background.
6. Avoid shadows touching the object.
7. Keep the complete object visible in every frame.
8. Do not zoom or move the camera.
9. Use 24–72 views for typical objects.
10. Ensure image filenames are ordered by rotation angle.

A suitable video should show the object rotating, not the camera moving around the object.

2.21.2 Loading Data

Either:

- Click **Click or drop files** and select a video.
- Select multiple silhouette images.
- Drag files onto the input area.
- Click **Load synthetic demo** to test the program.

The synthetic demo is recommended for confirming that the browser and application are functioning correctly.

2.21.3 Choosing Segmentation Settings

Uniform background

Use:

text

Auto: distance to border color

This is the preferred mode for a plain background.

Bright object

Use:

text

Bright object / dark background

Dark object

Use:

text

Dark object / bright background

Threshold adjustment

Start with:

text

Otsu (auto)

If the mask is too small or fragmented:

- Increase the threshold bias for some segmentation modes.
- Reduce it if too much background is classified as object.
- Use manual threshold for predictable lighting.

Morphological cleanup

Increase:

- **Open** to remove isolated specks.
- **Close** to seal small gaps.

Use these carefully because excessive morphology can remove thin parts or merge nearby structures.

Largest blob

Keep enabled when the object is the dominant foreground component.

Disable it if the object consists of multiple separated pieces that must all be retained.

Fill interior holes

Keep enabled for solid objects.

Disable it when an actual through-hole is an important part of the object shape.

Invert mask

Enable it if the object and background are reversed.

2.21.4 Checking the Silhouette Inspector

The silhouette strip shows each processed view.

The selected view is outlined in blue. The inspector displays:

- Original frame.
- Segmentation overlay.
- Red silhouette boundary.
- Blue estimated rotation axis.
- Green reconstruction bounds.

Before running reconstruction, inspect several views. A good mask should:

- Cover the complete object.
- Exclude the background.
- Be relatively stable across all frames.
- Not include shadows or turntable artifacts.

2.21.5 Geometry Settings

Total rotation

Use 360 for a complete turntable rotation.

For a half-turn sequence, use 180, provided the object has suitable symmetry or the views are interpreted accordingly.

Closed loop

Enable this for a sequence representing a cyclic turntable rotation. The last sampled frame will not duplicate the first.

Disable it for an open sequence where the first and last frames are separate endpoints.

Direction

Use the direction that corresponds to the actual object rotation. If the reconstruction appears twisted or inconsistent, try the opposite direction.

Custom angles

Provide one angle per view:

text

0,30,60,90,120,150,180,210,240,270,300,330

The number of values must exactly match the number of views.

Axis shift

Adjust this if the vertical blue axis does not align with the physical turntable axis.

A wrong axis is one of the most common causes of an empty or distorted reconstruction.

Projection

Use:

- **Orthographic** when the camera is far away and perspective is negligible.
- **Perspective** when the object visibly changes size with depth.

For most ordinary turntable setups, orthographic projection is a good starting point.

2.21.6 Volumetric Settings

Grid resolution

Higher values provide greater detail but consume more memory and CPU time.

Approximate voxel count is:

$$N_{\text{voxels}}=n_x n_y n_z$$

Because the vertical resolution is computed automatically, increasing the horizontal/depth resolution can increase the total memory substantially.

Suggested values:

Situation	Resolution
Quick preview	64–96
Normal reconstruction	112–160
High detail	176–224

Consistency threshold

Recommended starting point:

text

0.90–0.97

- Higher values produce stricter carving.
- Lower values tolerate noisy or inconsistent masks.
- Very high values can make the volume disappear if even a few masks are imperfect.

Volume smoothing

Use low values, typically 1–2, to reduce voxel stair-stepping.

Excessive smoothing may remove thin features.

Mesh smoothing

Taubin smoothing can improve the final surface.

Use approximately:

text

4–10 iterations

Too much smoothing can erase sharp details and narrow appendages.

Seal grid boundary

Usually keep enabled. It ensures that the isosurface does not run into the outer reconstruction grid.

Mirror model

Use this if the exported model has the opposite horizontal handedness from the desired orientation.

2.21.7 Reconstruction

Click:

text

Reconstruct 3D

The log reports:

- Grid size.
- Number of voxels.
- Occupied voxel count.
- Surface-net vertex count.
- Triangle count.
- Mesh smoothing status.

The preview can be controlled with:

- Drag: rotate.
- Mouse wheel: zoom.
- Shift + drag: pan.

2.21.8 Exporting

Choose a model height, for example:

text

1.0

Then click:

- **Download .OBJ**
- **Download .PLY**

OBJ is convenient for most modeling software and game engines.

PLY is useful for geometry-processing tools and research workflows.

2.22 Error Messages and Troubleshooting

2.22.1 “No decodable images found”

Possible causes:

- Unsupported image format.
- Corrupted file.
- Browser decoding limitation.

Try converting the images to PNG or JPEG.

2.22.2 “Cannot decode this video”

The browser may not support the video codec.

Recommended formats:

- MP4 with H.264 video.
- WebM with VP8 or VP9.

2.22.3 “Empty silhouettes”

Possible causes:

- Wrong segmentation method.
- Incorrect threshold.
- Object is too dark or too bright for the selected mode.
- Mask inversion is required.
- The object does not contrast with the background.

Inspect the silhouette inspector and adjust threshold-related settings.

2.22.4 “Carved volume is empty”

Possible causes:

- Incorrect rotation direction.
- Wrong rotation angles.
- Incorrect axis position.
- Excessively high consistency threshold.
- One or more empty silhouettes.
- Object position changes between frames.

Suggested actions:

1. Lower consistency threshold.
2. Check the blue axis line.
3. Try the opposite rotation direction.
4. Confirm the number and order of views.
5. Improve the segmentation masks.
6. Disable perspective unless perspective is clearly present.

2.22.5 Distorted or Twisted Model

Likely causes:

- Image order is incorrect.
- Frame angles are incorrect.
- The object does not rotate around a fixed vertical axis.
- The camera moves.
- The rotation direction is reversed.
- The axis is shifted.

2.23 Computational Complexity

Let:

- V be the number of views.
- $N=n_x n_y n_z$ be the number of voxels.
- W,H be image dimensions.
- r be a morphology or blur radius.

The main carving cost is approximately:

$O(VN)$

This is the dominant stage.

Segmentation is approximately:

$O(VWH)$

The naive box blur has approximately:

$O(VWH_r)$

complexity per pass.

The Surface Nets traversal is approximately:

$O(N)$

Taubin smoothing is approximately:

$O(I(V_m+E_m))$

where:

- I is the number of smoothing iterations.
- V_m is the number of mesh vertices.
- E_m is the number of adjacency entries.

The software preview rasterizer can be expensive for large triangles and high-resolution canvases. During interaction, it decimates triangles when the mesh is very large:

javascript

```
const step=fast&&idx.length>300000 ? 2 : 1;
```

2.24 Memory Considerations

The occupancy count uses:

javascript

```
const cnt=new Uint16Array(N);
```

The maximum count is the number of views, limited by the UI to 240, which fits safely inside a 16-bit unsigned integer.

The signed volume uses:

javascript

```
const data=new Float32Array(N);
```

A grid of 224^3 voxels contains approximately:

11.2 million voxels

A single Float32 volume of this size requires roughly:

$11.2\times10^6\times4\approx44.8$ MB

Additional arrays are allocated during smoothing and mesh generation, so high resolutions may require considerably more memory.

2.25 Important Algorithmic Limitations

25.1 Visual Hull Limitation

The reconstructed object is approximately:

$$V_{\text{visual hull}}=\cap_{i=1}^{K}\prod_i^{-1}S_i$$

where:

- S_i is the silhouette in view i.
- $\prod_i^{-1}S_i$ is the generalized cone or extrusion of that silhouette into 3D.
- The intersection is the volume consistent with all silhouettes.

This is not necessarily the true object volume.

A concavity that is never visible in any silhouette is filled in by the visual hull.

2.25.2 Approximate Camera Model

The program does not estimate:

- Lens focal length.
- Principal point.
- Radial distortion.
- Exact extrinsic camera parameters.
- Turntable center through calibration.

Instead, it estimates a single vertical image axis and uses simplified orthographic or perspective equations.

2.25.3 Axis Estimation Assumptions

The axis is estimated from the average silhouette centroid. This can be inaccurate when:

- The object is strongly asymmetric.
- The object's center of mass is not aligned with the rotation axis.
- The camera is not centered.
- The object shifts on the turntable.
- The silhouettes are poorly segmented.

The manual axis shift exists to compensate for such problems.

2.25.4 Shadows and Reflections

Automatic segmentation may classify shadows, reflections, turntable surfaces, or background gradients as part of the object.

This is why the input recommendations emphasize:

- Uniform lighting.
- Contrast.
- A clean background.
- Shadows separated from the object.

2.25.5 Thin Structures

Thin features may disappear because of:

- Insufficient input resolution.
- Voxel discretization.
- High consistency threshold.
- Morphological opening.
- Volume smoothing.
- Mesh smoothing.

For thin parts, use:

- More views.
- Higher processing resolution.
- Higher voxel resolution.
- Lower opening and smoothing values.
- A slightly lower consistency threshold if masks are noisy.

2.25.6 Image Sequence Angle Assumption

If no custom angle list is provided, image views are assumed to be evenly spaced. Irregularly sampled images require an explicit custom angle list.

2.26 Recommended Default Workflow

A practical workflow is:

1. Click **Load synthetic demo** to verify the application.
2. Load a turntable video or ordered image sequence.
3. Set processing size to 440–640 pixels.
4. Use 36–72 sampled frames.
5. Start with automatic border-color segmentation.
6. Inspect several silhouette masks.
7. Enable largest-component filtering.
8. Enable hole filling for solid objects.
9. Estimate geometry and verify the axis overlay.
10. Use orthographic projection initially.
11. Set grid resolution around 112–160.
12. Set consistency to approximately 0.95.
13. Use volume smoothing of 1.
14. Use mesh smoothing of 4–8.
15. Run reconstruction.
16. Rotate and inspect the model.
17. Adjust axis, threshold, direction, or consistency if needed.
18. Set the desired model height.
19. Export OBJ or PLY.

2.27 Summary of the Complete Mathematical Pipeline

Given segmented silhouettes M_i, view angles θ_i, axis coordinate x_a, radius R, and grid coordinates (x,y,z):

Step 1: Rotate the voxel point

$$x_r = x\cos\theta_i + z\sin\theta_i$$
$$z_r = -x\sin\theta_i + z\cos\theta_i$$

Step 2: Project into view *i*

Orthographic:

$$u_i = x_a + x_r$$
$$v_i = y$$

Perspective:

$$s_i = D/(D - z_r)$$
$$u_i = x_a + x_r s_i$$
$$v_i = v_c + (y - v_c)s_i$$

Step 3: Query the silhouette

$$q_i(x,y,z) = M_i(\text{round}(u_i),\text{round}(v_i))$$

Step 4: Compute consistency

$F(x,y,z)=\sum_{i=1}^{K} q_i\ (x,y,z))/K$

Step 5: Build the signed field

$\phi(x,y,z)=q- F(x,y,z)$

Step 6: Define the solid

$\phi(x,y,z)<0$

Step 7: Extract the zero isosurface

$\phi(x,y,z)=0$

using Surface Nets.

Step 8: Smooth the mesh

Apply repeated Taubin updates:

$p_i=p_i+\lambda(\bar{p}_i-p_i)$
$p_i=p_i+\mu(\bar{p}_i-p_i)$

Step 9: Normalize height

$k= H_{out}/(y_{max}-y_{min})$
$p_i^{'}=(k(x_i-c_x),k(y_i-y_{min}),k(z_i-c_z))$

Step 10: Export

Write the normalized vertices, normals, and triangle indices to OBJ or PLY.

2.28 Final Assessment

This file is a compact but technically substantial browser implementation of a silhouette-based 3D reconstruction system. Its main strengths are:

- Fully client-side operation.
- No external dependencies.
- Support for video and image sequences.
- Adjustable segmentation and reconstruction parameters.
- Explicit visualization of intermediate masks.
- Voxel-based visual-hull reconstruction.
- Table-free Surface Nets extraction.
- Taubin mesh smoothing.
- OBJ and PLY export.
- A software-rendered 3D preview that does not require WebGL.

Its reconstruction quality depends primarily on:

1. Silhouette accuracy.
2. Correct view ordering and angular calibration.

3. Accurate rotation-axis placement.
4. Sufficient view count.
5. Adequate voxel resolution.
6. Appropriate consistency and smoothing settings.

The application should be understood as a practical visual-hull demonstrator and rough modeling tool. It produces a useful exterior approximation from turntable silhouettes, but it is not a replacement for calibrated multi-view stereo, photogrammetry, volumetric fusion, or neural implicit reconstruction when accurate concave geometry and surface detail are required.

3 JS/HTML Codes

The following are the JavaScript/HTML codes of the browser-based tool (http://www.iaees.org/publications/journals/Selforganizology/articles/2028-15(3-4)/SpaceCarvingJS.htm; Fig. 1):

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8"/>
<meta name="viewport" content="width=device-width,initial-scale=1"/>
<title>SpaceCarvingJS: Silhouette-based 3D reconstruction using space carving in the browser</title>
<style>
  :root{
    --bg:#12151b; --panel:#1a1f28; --panel2:#222834; --line:#2e3746;
    --txt:#dfe6f0; --dim:#93a1b5; --acc:#4da3ff; --ok:#43c98b; --warn:#ffb454; --err:#ff6b6b;
  }
  *{box-sizing:border-box}
  html,body{margin:0;height:100%;background:var(--bg);color:var(--txt);
    font:13px/1.45 system-ui,Segoe UI,Roboto,Helvetica,Arial,sans-serif}
  h1{font-size:15px;margin:0;font-weight:600;letter-spacing:.3px}
  h2{font-size:12px;text-transform:uppercase;letter-spacing:.8px;color:var(--dim);
     margin:14px 0 6px;border-bottom:1px solid var(--line);padding-bottom:4px}
  header{display:flex;align-items:center;gap:12px;padding:10px 14px;background:var(--panel);
    border-bottom:1px solid var(--line)}
  header .sub{color:var(--dim);font-size:11.5px}
  #app{display:grid;grid-template-columns:322px 1fr;height:calc(100% - 45px)}
  #panel{overflow:auto;padding:10px 14px 40px;background:var(--panel);border-right:1px solid var(--line)}
  #main{display:grid;grid-template-rows:auto 1fr;overflow:hidden}
  .row{display:flex;align-items:center;gap:8px;margin:6px 0}
  .row label{flex:1 1 auto;color:var(--dim)}
  .row .val{min-width:44px;text-align:right;font-variant-numeric:tabular-nums;color:var(--txt)}
  input[type=range]{flex:0 0 128px;accent-color:var(--acc)}
  input[type=number],select,input[type=text]{background:var(--panel2);color:var(--txt);
    border:1px solid var(--line);border-radius:5px;padding:3px 6px;font:inherit;width:96px}
  input[type=checkbox]{accent-color:var(--acc)}
  button{background:var(--panel2);color:var(--txt);border:1px solid var(--line);border-radius:6px;
```

```
    padding:7px 10px;font:inherit;cursor:pointer}
  button:hover:not(:disabled){border-color:var(--acc);color:#fff}
  button:disabled{opacity:.45;cursor:not-allowed}
  button.primary{background:linear-gradient(#3d8ee0,#2f6fb8);border-color:#2f6fb8;color:#fff;font-weight:600}
  button.go{background:linear-gradient(#33b97e,#249a66);border-color:#249a66;color:#fff;font-weight:600}
  .btns{display:flex;gap:8px;flex-wrap:wrap;margin:10px 0}
  #drop{border:2px dashed var(--line);border-radius:10px;padding:14px;text-align:center;color:var(--dim);
    cursor:pointer;transition:.15s}
  #drop.hot{border-color:var(--acc);color:var(--txt);background:#1e2836}
  #modeBadge{display:inline-block;padding:2px 8px;border-radius:20px;background:var(--panel2);
    border:1px solid var(--line);font-size:11px;color:var(--dim)}
  #strip{display:flex;gap:6px;overflow-x:auto;padding:8px 10px;background:#0e1116;
    border-bottom:1px solid var(--line);min-height:118px;align-items:center}
  #strip canvas{border:1px solid var(--line);border-radius:4px;cursor:pointer;background:#000}
  #strip canvas.sel{border-color:var(--acc);box-shadow:0 0 0 2px rgba(77,163,255,.35)}
  #stage{display:grid;grid-template-columns:1fr 1fr;gap:10px;padding:10px;overflow:hidden}
  .card{background:var(--panel);border:1px solid var(--line);border-radius:10px;display:flex;
    flex-direction:column;overflow:hidden;min-height:0}
  .card .hd{padding:6px 10px;border-bottom:1px solid var(--line);color:var(--dim);font-size:11.5px;
    display:flex;justify-content:space-between;align-items:center}
  .card .bd{flex:1 1 auto;display:flex;align-items:center;justify-content:center;overflow:hidden;background:#0b0e13}
  canvas{max-width:100%;max-height:100%;display:block}
  #log{height:112px;overflow:auto;background:#0b0e13;border:1px solid var(--line);border-radius:8px;
    padding:6px 8px;font:11.5px/1.5 ui-monospace,Consolas,monospace;color:var(--dim);white-space:pre-wrap}
  #bar{height:6px;background:var(--panel2);border-radius:4px;overflow:hidden;margin:8px 0}
  #bar>i{display:block;height:100%;width:0;background:linear-gradient(90deg,#4da3ff,#43c98b);transition:.1s}
  .hint{color:var(--dim);font-size:11.5px;margin:6px 0}
  .kv{display:grid;grid-template-columns:auto 1fr;gap:2px 10px;font-size:11.5px;color:var(--dim)}
  .kv b{color:var(--txt);font-weight:600;font-variant-numeric:tabular-nums}
  details{margin:8px 0}summary{cursor:pointer;color:var(--dim)}
</style>
</head>
<body>
<header>
  <h1>SpaceCarvingJS: Silhouette-based 3D reconstruction using space carving in the browser</h1>
  <span class="sub">Silhouette → 3D → OBJ · pure client-side · no dependencies</span>
  <span style="flex:1"></span>
  <span id="modeBadge">no input</span>
  </header>
  <a href="http://www.iaees.org/publications/journals/Selforganizology/articles/2028-15(3-4)/1-Zhang-Abstract.asp"
style="color: brown;">Zhang WJ. 2028. SpaceCarvingJS: Silhouette-based 3D reconstruction using space carving in the browser.
Selforganizology, 15(3-4): 54-112. </a><br>


<div id="app">
```

```
<!-- ──────────────────────────────────── CONTROL PANEL ──────────────────────────────────── -->
<aside id="panel">
  <h2>1 · Input</h2>
  <div id="drop">
    <b>Click or drop files</b><br>
    one <b>rotation video</b> (mp4/webm/mov)<br>or <b>many silhouette images</b> (png/jpg)
    <input id="file" type="file" multiple accept="video/*,image/*" style="display:none">
  </div>
  <div class="btns">
    <button id="btnDemo">Load synthetic demo</button>
    <button id="btnClear">Clear</button>
  </div>
  <div class="row"><label>Working resolution (px)</label>
    <input id="procSize" type="number" min="128" max="1024" step="32" value="440"></div>
  <div class="row"><label>Frames to sample (video)</label>
    <input id="nFrames" type="number" min="6" max="240" step="1" value="8"></div>
  <div class="row"><label>Video trim start / end (%)</label>
    <input id="tStart" type="number" min="0" max="99" step="1" value="0" style="width:44px">
    <input id="tEnd" type="number" min="1" max="100" step="1" value="100" style="width:44px"></div>

  <h2>2 · Silhouette segmentation</h2>
  <div class="row"><label>Method</label>
    <select id="segMode">
      <option value="bg">Auto: distance to border color</option>
      <option value="bright">Bright object / dark background</option>
      <option value="dark">Dark object / bright background</option>
    </select></div>
  <div class="row"><label>Threshold</label>
    <select id="thrMode" style="width:104px">
      <option value="otsu">Otsu (auto)</option>
      <option value="manual">Manual</option>
    </select></div>
  <div class="row"><label>Threshold bias / value</label>
    <input id="thrBias" type="range" min="0.2" max="2.5" step="0.02" value="1"><span class="val"
id="thrBiasV">1.00</span></div>
  <div class="row"><label>Pre-blur (px)</label>
    <input id="blur" type="range" min="0" max="6" step="1" value="1"><span class="val" id="blurV">1</span></div>
  <div class="row"><label>Open (remove specks)</label>
    <input id="mOpen" type="range" min="0" max="6" step="1" value="1"><span class="val"
id="mOpenV">1</span></div>
  <div class="row"><label>Close (seal gaps)</label>
    <input id="mClose" type="range" min="0" max="6" step="1" value="1"><span class="val"
id="mCloseV">1</span></div>
  <div class="row"><label>Keep largest blob</label><input id="largest" type="checkbox" checked></div>
  <div class="row"><label>Fill interior holes</label><input id="fill" type="checkbox" checked></div>
```

```
<div class="row"><label>Invert mask</label><input id="invert" type="checkbox"></div>
<div class="btns"><button id="btnSeg" class="primary" disabled>Re-run segmentation</button></div>

<h2>3 · Turntable geometry</h2>
<div class="row"><label>Total rotation (deg)</label><input id="totalRot" type="number" value="360" step="1"></div>
<div class="row"><label>Closed 360° loop</label><input id="loop" type="checkbox" checked></div>
<div class="row"><label>Direction</label>
  <select id="dir"><option value="1">CCW (+)</option><option value="-1">CW (−)</option></select></div>
<div class="row"><label>Custom angles (deg, csv)</label></div>
<div class="row"><input id="angles" type="text" placeholder="e.g. 0,30,60,90,..." style="width:100%"></div>
<div class="row"><label>Rotation-axis X shift (px)</label>
  <input id="axisShift" type="range" min="-60" max="60" step="0.5" value="0"><span class="val" id="axisShiftV">0</span></div>
<div class="row"><label>Projection</label>
  <select id="proj"><option value="ortho">Orthographic</option><option value="persp">Perspective</option></select></div>
<div class="row"><label>Camera distance (× radius)</label>
  <input id="camDist" type="range" min="2" max="30" step="0.5" value="8"><span class="val" id="camDistV">8</span></div>

<h2>4 · Volumetric carving</h2>
<div class="row"><label>Grid resolution</label>
  <input id="res" type="range" min="32" max="224" step="8" value="112"><span class="val" id="resV">112</span></div>
<div class="row"><label>Consistency threshold</label>
  <input id="cons" type="range" min="0.5" max="1" step="0.01" value="0.97"><span class="val" id="consV">0.97</span></div>
<div class="row"><label>Volume smoothing</label>
  <input id="volSm" type="range" min="0" max="6" step="1" value="1"><span class="val" id="volSmV">1</span></div>
<div class="row"><label>Mesh smoothing (Taubin)</label>
  <input id="meshSm" type="range" min="0" max="30" step="1" value="6"><span class="val" id="meshSmV">6</span></div>
<div class="row"><label>Seal grid boundary</label><input id="seal" type="checkbox" checked></div>
<div class="row"><label>Mirror model (handedness)</label><input id="flipX" type="checkbox"></div>
<div class="btns"><button id="btnRun" class="go" disabled>Reconstruct 3D</button></div>
<div id="bar"><i></i></div>

<h2>5 · Export</h2>
<div class="row"><label>Model height (units)</label><input id="outH" type="number" value="1" step="0.1"></div>
<div class="row"><label>Write vertex normals</label><input id="wNorm" type="checkbox" checked></div>
<div class="btns">
  <button id="btnObj" class="primary" disabled>Download .OBJ</button>
  <button id="btnPly" disabled>Download .PLY</button>
</div>
```

```
    <div class="kv" id="stats"></div>

    <details>
      <summary>How to capture good input</summary>
      <div class="hint">
        • Put the object on a turntable, keep the <b>camera fixed</b>; rotate exactly one full turn.<br>
        • Use a <b>plain, contrasting background</b> and even lighting; avoid shadows touching the object.<br>
        • Keep the object fully inside the frame at every angle; do not zoom.<br>
        • The rotation axis must be roughly <b>vertical in the image</b>.<br>
        • 24–72 views is plenty; more views ≠ better if segmentation is noisy.<br>
        • Space carving reconstructs the <b>visual hull</b>: deep concavities that never appear on any
          silhouette cannot be recovered (that is a mathematical limit, not a bug).
      </div>
    </details>
    <h2>Log</h2>
    <div id="log"></div>
  </aside>

  <!-- ──────────────────────────────── MAIN ──────────────────────────────── -->
  <div id="main">
    <div id="strip"><span class="hint" style="padding:0 8px">Frame / silhouette strip appears here…</span></div>
    <div id="stage">
      <div class="card">
        <div class="hd"><span>Silhouette inspector</span><span id="insLbl"></span></div>
        <div class="bd"><canvas id="ins" width="520" height="420"></canvas></div>
      </div>
      <div class="card">
        <div class="hd"><span>3D preview — drag: rotate · wheel: zoom · shift+drag: pan</span>
          <span id="viewLbl"></span></div>
        <div class="bd"><canvas id="view3d" width="560" height="470"></canvas></div>
      </div>
    </div>
  </div>
</div>

<script>
"use strict";
/* ============================================================================
   Silhouette-based 3D reconstruction (turntable visual hull)
   Pipeline:  frames → binary silhouettes → axis/scale estimation →
              voxel space carving → Surface Nets iso-surface →
              Taubin smoothing → OBJ / PLY export
   Everything runs locally in the browser; no external libraries.
   ============================================================================ */
```

```
const $ = id => document.getElementById(id);
const clamp = (v,a,b)=>v<a?a:v>b?b:v;

/* ---------------------------------- state ---------------------------------- */
const S = {
  mode:null,                          // 'video' | 'images' | 'demo'
  frames:[],                          // {cv:HTMLCanvasElement, label:string}
  views:[],                           // {mask:Uint8Array, area, cx, cy, x0,x1,y0,y1, angle}
  W:0, H:0,
  geom:null,                          // {axisX, yTop, yBottom, R}
  mesh:null,                          // {pos:Float32Array, idx:Uint32Array, nrm:Float32Array}
  sel:0
};

/* ---------------------------------- log ---------------------------------- */
function log(msg,cls){
  const el=$('log'), c={ok:'--ok',warn:'--warn',err:'--err'}[cls];
  const line=document.createElement('div');
  if(c) line.style.color=`var(${c})`;
  line.textContent=msg; el.appendChild(line); el.scrollTop=el.scrollHeight;
}
function prog(f){ $('bar').firstElementChild.style.width=(clamp(f,0,1)*100).toFixed(1)+'%'; }
const nextTick = ()=>new Promise(r=>setTimeout(r,0));

/* ========================================================================== */
/*    1. INPUT — video frames or image files                                  */
/* ========================================================================== */
function fitCanvas(src,sw,sh,W,H){                    // letterbox-draw into common canvas
  const cv=document.createElement('canvas'); cv.width=W; cv.height=H;
  const g=cv.getContext('2d',{willReadFrequently:true});
  const s=Math.min(W/sw,H/sh), dw=sw*s, dh=sh*s;
  g.fillStyle='#000'; g.fillRect(0,0,W,H);
  g.drawImage(src,(W-dw)/2,(H-dh)/2,dw,dh);
  return cv;
}
function seek(v,t){
  return new Promise(res=>{
    let done=false;
    const h=()=>{ if(done)return; done=true; v.removeEventListener('seeked',h); res(); };
    v.addEventListener('seeked',h);
    v.currentTime=t;
    setTimeout(h,2500);                               // safety timeout
  });
}
async function loadVideo(file,count,p0,p1,proc){
```

```
  const v=document.createElement('video');
  v.muted=true; v.playsInline=true; v.preload='auto'; v.crossOrigin='anonymous';
  v.src=URL.createObjectURL(file);
  await new Promise((res,rej)=>{ v.onloadedmetadata=()=>res(); v.onerror=()=>rej(new Error('Cannot decode this video (codec unsupported by the browser).')); });
  const dur=v.duration;
  if(!isFinite(dur)||dur<=0) throw new Error('Video duration unknown — try converting to MP4/H.264 or WebM.');
  const vw=v.videoWidth, vh=v.videoHeight;
  const s=Math.min(1,proc/Math.max(vw,vh));
  const W=Math.max(8,Math.round(vw*s)), H=Math.max(8,Math.round(vh*s));
  log(`Video: ${vw}×${vh}, ${dur.toFixed(2)} s → sampling ${count} frames at ${W}×${H}`);
  const a=dur*p0, b=dur*p1;
  const loop=$('loop').checked;
  const frames=[];
  await seek(v,Math.min(a+1e-3,dur-1e-3));
  for(let i=0;i<count;i++){
    const t=clamp(a+(b-a)*(loop?i/count:(count>1?i/(count-1):0)),0,Math.max(0,dur-1e-3));
    await seek(v,t);
    frames.push({cv:fitCanvas(v,vw,vh,W,H),label:t.toFixed(2)+'s'});
    prog((i+1)/count); if(i%4===0) await nextTick();
  }
  URL.revokeObjectURL(v.src);
  return {frames,W,H};
}
async function loadImages(files,proc){
  files=[...files].sort((a,b)=>a.name.localeCompare(b.name,undefined,{numeric:true}));
  const bmps=[];
  for(const f of files){
    try{ bmps.push({bmp:await createImageBitmap(f),name:f.name}); }
    catch(e){ log('Skipped unreadable image: '+f.name,'warn'); }
  }
  if(!bmps.length) throw new Error('No decodable images found.');
  let mw=0,mh=0; for(const b of bmps){ mw=Math.max(mw,b.bmp.width); mh=Math.max(mh,b.bmp.height); }
  const s=Math.min(1,proc/Math.max(mw,mh));
  const W=Math.max(8,Math.round(mw*s)), H=Math.max(8,Math.round(mh*s));
  log(`Images: ${bmps.length} files, canvas ${W}×${H} (largest input ${mw}×${mh})`);
  const frames=bmps.map(b=>({cv:fitCanvas(b.bmp,b.bmp.width,b.bmp.height,W,H),label:b.name}));
  return {frames,W,H};
}

/* --- synthetic demo: renders silhouettes of an implicit 3-D object -------- */
function demoData(nv=32,size=300){
  const inside=(x,y,z)=>{
    if((x/0.55)**2+(y/0.80)**2+(z/0.42)**2<1) return true;                    // body
    if(x*x+(y-0.95)**2+z*z<0.34*0.34)              return true;                    // head
```

```
    if(x*x+(y-0.98)**2+(z-0.40)**2<0.13*0.13) return true;                    // nose (breaks symmetry)
    if(Math.abs(x)<0.95&&Math.abs(y-0.10)<0.09&&Math.abs(z)<0.10) return true; // arms
    if(Math.abs(x-0.15)<0.13&&y>-1.25&&y<-0.60&&Math.abs(z)<0.13) return true; // leg
    if(Math.abs(x+0.15)<0.13&&y>-1.25&&y<-0.60&&Math.abs(z)<0.13) return true;
    return false;
  };
  const W=size,H=size, sc=size/3.0, cx=W/2, cy=H/2, frames=[];
  for(let f=0;f<nv;f++){
    const th=2*Math.PI*f/nv, c=Math.cos(th), s=Math.sin(th);
    const cv=document.createElement('canvas'); cv.width=W; cv.height=H;
    const g=cv.getContext('2d',{willReadFrequently:true});
    const im=g.createImageData(W,H), d=im.data;
    for(let py=0;py<H;py++){
      const wy=-(py-cy)/sc;
      for(let px=0;px<W;px++){
        const wx=(px-cx)/sc; let hit=0;
        for(let zi=-40;zi<=40;zi++){                                          // ray-march along depth
          const wz=zi/25;
          // camera ray point → object space (inverse turntable rotation)
          const ox= wx*c - wz*s, oz= wx*s + wz*c;
          if(inside(ox,wy,oz)){hit=1;break;}
        }
        const o=(py*W+px)*4, v=hit?235:18;
        d[o]=d[o+1]=d[o+2]=v; d[o+3]=255;
      }
    }
    g.putImageData(im,0,0);
    frames.push({cv,label:'demo '+(f*360/nv).toFixed(0)+'°'});
  }
  return {frames,W,H};
}

/* ========================================================================== */
/*    2. SILHOUETTE SEGMENTATION                                              */
/* ========================================================================== */
function medianBorderColor(data,W,H){
  const b=Math.max(2,Math.round(Math.min(W,H)*0.02)), R=[],G=[],B=[];
  for(let y=0;y<H;y+=2)for(let x=0;x<W;x+=2){
    if(x>=b&&x<W-b&&y>=b&&y<H-b) continue;
    const o=(y*W+x)*4; R.push(data[o]);G.push(data[o+1]);B.push(data[o+2]);
  }
  const med=a=>{a.sort((p,q)=>p-q);return a[a.length>>1]||0;};
  return [med(R),med(G),med(B)];
}
function otsu(f,n){                                          // f: Float32Array in [0,1]
```

```
  const hist=new Float64Array(256);
  for(let i=0;i<n;i++) hist[clamp(f[i]*255|0,0,255)]++;
  let sum=0; for(let i=0;i<256;i++) sum+=i*hist[i];
  let sB=0,wB=0,best=-1,thr=128;
  for(let t=0;t<256;t++){
    wB+=hist[t]; if(!wB)continue; const wF=n-wB; if(!wF)break;
    sB+=t*hist[t];
    const mB=sB/wB, mF=(sum-sB)/wF, v=wB*wF*(mB-mF)*(mB-mF);
    if(v>best){best=v;thr=t;}
  }
  return thr/255;
}
function boxBlur(f,W,H,r){
  if(r<1) return f;
  const t=new Float32Array(f.length), o=new Float32Array(f.length);
  for(let y=0;y<H;y++){ const b=y*W;
    for(let x=0;x<W;x++){ let s=0,c=0;
      for(let k=-r;k<=r;k++){const xx=x+k; if(xx>=0&&xx<W){s+=f[b+xx];c++;}}
      t[b+x]=s/c; } }
  for(let x=0;x<W;x++)for(let y=0;y<H;y++){ let s=0,c=0;
    for(let k=-r;k<=r;k++){const yy=y+k; if(yy>=0&&yy<H){s+=t[yy*W+x];c++;}}
    o[y*W+x]=s/c; }
  return o;
}
function morph(m,W,H,it,grow){                          // grow>0 dilate, <0 erode
  let a=m;
  for(let n=0;n<it;n++){
    const b=new Uint8Array(a.length);
    for(let y=0;y<H;y++)for(let x=0;x<W;x++){
      let v=grow>0?0:1;
      for(let dy=-1;dy<=1;dy++)for(let dx=-1;dx<=1;dx++){
        const xx=x+dx,yy=y+dy;
        const s=(xx<0||yy<0||xx>=W||yy>=H)?0:a[yy*W+xx];
        if(grow>0){ if(s) v=1; } else { if(!s) v=0; }
      }
      b[y*W+x]=v;
    }
    a=b;
  }
  return a;
}
function keepLargest(m,W,H){
  const lab=new Int32Array(m.length).fill(-1), st=new Int32Array(m.length);
  let best=-1,bestN=0;
  for(let i=0;i<m.length;i++){
```

```
      if(!m[i]||lab[i]>=0) continue;
      let sp=0,n=0; st[sp++]=i; lab[i]=i;
      while(sp){ const p=st[--sp]; n++;
        const x=p%W,y=(p-x)/W;
        for(let k=0;k<4;k++){
          const xx=x+(k===0?1:k===1?-1:0), yy=y+(k===2?1:k===3?-1:0);
          if(xx<0||yy<0||xx>=W||yy>=H) continue;
          const q=yy*W+xx; if(m[q]&&lab[q]<0){lab[q]=i;st[sp++]=q;}
        } }
      if(n>bestN){bestN=n;best=i;}
    }
    if(best<0) return m;
    const out=new Uint8Array(m.length);
    for(let i=0;i<m.length;i++) if(lab[i]===best) out[i]=1;
    return out;
}
function fillHoles(m,W,H){
    const vis=new Uint8Array(m.length), st=new Int32Array(m.length); let sp=0;
    const push=i=>{ if(!m[i]&&!vis[i]){vis[i]=1;st[sp++]=i;} };
    for(let x=0;x<W;x++){push(x);push((H-1)*W+x);}
    for(let y=0;y<H;y++){push(y*W);push(y*W+W-1);}
    while(sp){ const p=st[--sp], x=p%W, y=(p-x)/W;
      for(let k=0;k<4;k++){
        const xx=x+(k===0?1:k===1?-1:0), yy=y+(k===2?1:k===3?-1:0);
        if(xx<0||yy<0||xx>=W||yy>=H) continue; push(yy*W+xx);
      } }
    const out=Uint8Array.from(m);
    for(let i=0;i<m.length;i++) if(!m[i]&&!vis[i]) out[i]=1;
    return out;
}
function segment(cv,W,H,P){
    const g=cv.getContext('2d',{willReadFrequently:true});
    const data=g.getImageData(0,0,W,H).data, n=W*H;
    let f=new Float32Array(n);
    if(P.segMode==='bg'){
      const [br,bg_,bb]=medianBorderColor(data,W,H);
      for(let i=0,o=0;i<n;i++,o+=4)
        f[i]=(Math.abs(data[o]-br)+Math.abs(data[o+1]-bg_)+Math.abs(data[o+2]-bb))/765;
    }else{
      for(let i=0,o=0;i<n;i++,o+=4){
        const L=(0.299*data[o]+0.587*data[o+1]+0.114*data[o+2])/255;
        f[i]=P.segMode==='bright'?L:1-L;
      }
    }
    f=boxBlur(f,W,H,P.blur);
```

```
  let thr = P.thrMode==='otsu' ? clamp(otsu(f,n)*P.thrBias,0.004,0.996)
                               : clamp(P.thrBias/2.5,0.004,0.996);
  let m=new Uint8Array(n);
  for(let i=0;i<n;i++) m[i]=(f[i]>thr)?1:0;
  if(P.invert) for(let i=0;i<n;i++) m[i]^=1;
  if(P.mOpen){ m=morph(m,W,H,P.mOpen,-1); m=morph(m,W,H,P.mOpen,1); }
  if(P.mClose){ m=morph(m,W,H,P.mClose,1); m=morph(m,W,H,P.mClose,-1); }
  if(P.largest) m=keepLargest(m,W,H);
  if(P.fill)    m=fillHoles(m,W,H);
  // statistics
  let area=0,sx=0,sy=0,x0=W,x1=-1,y0=H,y1=-1;
  for(let y=0;y<H;y++)for(let x=0;x<W;x++) if(m[y*W+x]){
    area++;sx+=x;sy+=y;
    if(x<x0)x0=x; if(x>x1)x1=x; if(y<y0)y0=y; if(y>y1)y1=y;
  }
  return {mask:m,area,cx:area?sx/area:W/2,cy:area?sy/area:H/2,x0,x1,y0,y1,thr};
}

/* ========================================================================== */
/*   3. TURNTABLE GEOMETRY (rotation axis, vertical extent, radius)           */
/* ========================================================================== */
function assignAngles(P,n){
  if(P.anglesTxt.trim()){
    const a=P.anglesTxt.split(/[,;\s]+/).filter(s=>s.length).map(Number);
    if(a.length===n && a.every(v=>isFinite(v))) { log('Using custom angle list.'); return a; }
    log(`Custom angle list has ${a.length} values but ${n} views — falling back to uniform spacing.`,'warn');
  }
  const tot=P.totalRot, out=[];
  for(let i=0;i<n;i++) out.push(P.loop ? tot*i/n : (n>1?tot*i/(n-1):0));
  return out;
}
function estimateGeometry(P){
  const V=S.views, W=S.W, H=S.H;
  let sx=0,k=0;                                          // axis X ≈ mean centroid over a full turn
  for(const v of V) if(v.area>0){ sx+=v.cx; k++; }
  let axisX=(k?sx/k:W/2)+P.axisShift;
  let R=0,yTop=H,yBot=-1;
  for(const v of V){
    if(v.area===0) continue;
    R=Math.max(R,Math.abs(v.x0-axisX),Math.abs(v.x1-axisX));
    yTop=Math.min(yTop,v.y0); yBot=Math.max(yBot,v.y1);
  }
  R=Math.max(4,R*1.04+2);
  yTop=Math.max(0,yTop-2); yBot=Math.min(H-1,yBot+2);
  if(yBot<=yTop) throw new Error('Empty silhouettes — adjust the segmentation settings.');
```

```
  return {axisX,yTop,yBottom:yBot,R};
}

/* ========================================================================== */
/*  4. VOXEL SPACE CARVING                                                    */
/* ========================================================================== */
async function carve(P){
  const {axisX,yTop,yBottom,R}=S.geom, W=S.W, H=S.H, V=S.views;
  const nx=P.res, nz=P.res, dx=2*R/(nx-1);
  const ny=clamp(Math.round((yBottom-yTop)/dx)+1,4,512), dy=(yBottom-yTop)/(ny-1);
  const N=nx*ny*nz;
  log(`Grid ${nx}×${ny}×${nz} = ${(N/1e6).toFixed(2)} M voxels · voxel ≈ ${dx.toFixed(2)} px`);
  const cnt=new Uint16Array(N);
  const xs=new Float32Array(nx), zs=new Float32Array(nz), ys=new Float32Array(ny);
  for(let i=0;i<nx;i++) xs[i]=-R+i*dx;
  for(let k=0;k<nz;k++) zs[k]=-R+k*dx;
  for(let j=0;j<ny;j++) ys[j]=yTop+j*dy;
  const persp=P.proj==='persp', camD=Math.max(1.6,P.camDist)*R, vc=(yTop+yBottom)/2;
  const vi=new Int32Array(ny);

  for(let vI=0;vI<V.length;vI++){
    const v=V[vI]; if(v.area===0) continue;
    const th=P.dir*v.angle*Math.PI/180, c=Math.cos(th), s=Math.sin(th), m=v.mask;
    if(!persp){ for(let j=0;j<ny;j++) vi[j]=Math.round(ys[j]); }
    for(let k=0;k<nz;k++){
      const z=zs[k], kb=k*nx*ny;
      for(let i=0;i<nx;i++){
        const x=xs[i];
        const xr=x*c+z*s, zr=-x*s+z*c;
        let u,sc=1;
        if(persp){ const wd=camD-zr; if(wd<=1e-3) continue; sc=camD/wd; u=axisX+xr*sc; }
        else u=axisX+xr;
        const ui=Math.round(u); if(ui<0||ui>=W) continue;
        let idx=kb+i;
        if(persp){
          for(let j=0;j<ny;j++,idx+=nx){
            const vv=Math.round(vc+(ys[j]-vc)*sc);
            if(vv>=0&&vv<H&&m[vv*W+ui]) cnt[idx]++;
          }
        }else{
          for(let j=0;j<ny;j++,idx+=nx){
            const vv=vi[j];
            if(vv>=0&&vv<H&&m[vv*W+ui]) cnt[idx]++;
          }
        }
```

```
        }
      }
      prog(0.05+0.55*(vI+1)/V.length);
      if(vI%3===0) await nextTick();
    }
    // occupancy ratio field
    const nV=V.reduce((a,v)=>a+(v.area>0?1:0),0);
    let field=new Float32Array(N);
    for(let i=0;i<N;i++) field[i]=cnt[i]/nV;
    // separable 1-2-1 volume smoothing
    for(let it=0;it<P.volSm;it++) field=smooth3D(field,nx,ny,nz);
    if(P.seal){
      for(let k=0;k<nz;k++)for(let j=0;j<ny;j++)for(let i=0;i<nx;i++)
        if(i===0||j===0||k===0||i===nx-1||j===ny-1||k===nz-1) field[k*nx*ny+j*nx+i]=0;
    }
    // signed field: negative = inside
    const data=new Float32Array(N);
    let solid=0;
    for(let i=0;i<N;i++){ data[i]=P.cons-field[i]; if(data[i]<0) solid++; }
    log(`Occupied voxels: ${solid} (${(100*solid/N).toFixed(2)} % of grid)`);
    if(solid<8) throw new Error('Carved volume is empty. Lower the consistency threshold, check the axis shift, the rotation direction, or the silhouettes.');
    return {data,dims:[nx,ny,nz],dx,dy,R,yTop};
}
function smooth3D(f,nx,ny,nz){
    const a=new Float32Array(f.length), b=new Float32Array(f.length), sxy=nx*ny;
    for(let k=0;k<nz;k++)for(let j=0;j<ny;j++){ const base=k*sxy+j*nx;
      for(let i=0;i<nx;i++){
        const l=i>0?f[base+i-1]:0, r=i<nx-1?f[base+i+1]:0;
        a[base+i]=0.25*l+0.5*f[base+i]+0.25*r; } }
    for(let k=0;k<nz;k++)for(let i=0;i<nx;i++){ const base=k*sxy+i;
      for(let j=0;j<ny;j++){
        const l=j>0?a[base+(j-1)*nx]:0, r=j<ny-1?a[base+(j+1)*nx]:0;
        b[base+j*nx]=0.25*l+0.5*a[base+j*nx]+0.25*r; } }
    for(let j=0;j<ny;j++)for(let i=0;i<nx;i++){ const base=j*nx+i;
      for(let k=0;k<nz;k++){
        const l=k>0?b[base+(k-1)*sxy]:0, r=k<nz-1?b[base+(k+1)*sxy]:0;
        a[base+k*sxy]=0.25*l+0.5*b[base+k*sxy]+0.25*r; } }
    return a;
}

/* ========================================================================== */
/*    5. ISO-SURFACE — Naive Surface Nets (dual contouring, table-free)       */
/* ========================================================================== */
const CUBE_EDGES=new Int32Array(24), EDGE_TABLE=new Int32Array(256);
```

```
(function(){
  let k=0;
  for(let i=0;i<8;i++)for(let j=1;j<=4;j<<=1){ const p=i^j; if(i<=p){CUBE_EDGES[k++]=i;CUBE_EDGES[k++]=p;} }
  for(let i=0;i<256;i++){ let em=0;
    for(let j=0;j<24;j+=2){
      const a=!!(i&(1<<CUBE_EDGES[j])), b=!!(i&(1<<CUBE_EDGES[j+1]));
      em|=(a!==b)?(1<<(j>>1)):0; }
    EDGE_TABLE[i]=em; }
})();
function surfaceNets(data,dims){
  const vertices=[], faces=[];
  let n=0, buf_no=1;
  const x=new Int32Array(3);
  const Rr=new Int32Array([1,dims[0]+1,(dims[0]+1)*(dims[1]+1)]);
  const grid=new Float32Array(8);
  const buffer=new Int32Array(Rr[2]*2);
  for(x[2]=0;x[2]<dims[2]-1;++x[2],n+=dims[0],buf_no^=1,Rr[2]=-Rr[2]){
    let m=1+(dims[0]+1)*(1+buf_no*(dims[1]+1));
    for(x[1]=0;x[1]<dims[1]-1;++x[1],++n,m+=2)
    for(x[0]=0;x[0]<dims[0]-1;++x[0],++n,++m){
      let mask=0,g=0,idx=n;
      for(let k=0;k<2;++k,idx+=dims[0]*(dims[1]-2))
      for(let j=0;j<2;++j,idx+=dims[0]-2)
      for(let i=0;i<2;++i,++g,++idx){ const p=data[idx]; grid[g]=p; mask|=(p<0)?(1<<g):0; }
      if(mask===0||mask===0xff) continue;
      const em=EDGE_TABLE[mask], v=[0,0,0]; let ec=0;
      for(let i=0;i<12;++i){
        if(!(em&(1<<i))) continue;
        ++ec;
        const e0=CUBE_EDGES[i<<1], e1=CUBE_EDGES[(i<<1)+1];
        const g0=grid[e0], g1=grid[e1]; let t=g0-g1;
        if(Math.abs(t)>1e-8) t=g0/t; else continue;
        for(let j=0,k=1;j<3;++j,k<<=1){
          const a=e0&k, b=e1&k;
          if(a!==b) v[j]+=a?1-t:t; else v[j]+=a?1:0;
        }
      }
      if(!ec) continue;
      const s=1/ec;
      for(let i=0;i<3;++i) v[i]=x[i]+s*v[i];
      buffer[m]=vertices.length; vertices.push(v);
      for(let i=0;i<3;++i){
        if(!(em&(1<<i))) continue;
        const iu=(i+1)%3, iv=(i+2)%3;
        if(x[iu]===0||x[iv]===0) continue;
```

```
            const du=Rr[iu], dv=Rr[iv];
            if(mask&1){
              faces.push([buffer[m],buffer[m-du-dv],buffer[m-du]]);
              faces.push([buffer[m],buffer[m-dv],buffer[m-du-dv]]);
            }else{
              faces.push([buffer[m],buffer[m-du-dv],buffer[m-dv]]);
              faces.push([buffer[m],buffer[m-du],buffer[m-du-dv]]);
            }
          }
        }
      }
    return {vertices,faces};
}
/* ----------------------- mesh post-processing ---------------------------- */
function taubin(pos,idx,iters,lam=0.55,mu=-0.58){
    const nv=pos.length/3;
    const cnt=new Uint16Array(nv);
    for(let t=0;t<idx.length;t+=3){ cnt[idx[t]]+=2;cnt[idx[t+1]]+=2;cnt[idx[t+2]]+=2; }
    const off=new Uint32Array(nv+1);
    for(let i=0;i<nv;i++) off[i+1]=off[i]+cnt[i];
    const adj=new Uint32Array(off[nv]), fill=new Uint32Array(nv);
    const add=(a,b)=>{ adj[off[a]+fill[a]++]=b; };
    for(let t=0;t<idx.length;t+=3){
      const a=idx[t],b=idx[t+1],c=idx[t+2];
      add(a,b);add(a,c);add(b,a);add(b,c);add(c,a);add(c,b);
    }
    let p=pos, q=new Float32Array(pos.length);
    const step=(src,dst,f)=>{
      for(let i=0;i<nv;i++){
        let sx=0,sy=0,sz=0; const s=off[i],e=off[i]+fill[i];
        for(let k=s;k<e;k++){ const j=adj[k]*3; sx+=src[j];sy+=src[j+1];sz+=src[j+2]; }
        const c=e-s, i3=i*3;
        if(!c){dst[i3]=src[i3];dst[i3+1]=src[i3+1];dst[i3+2]=src[i3+2];continue;}
        dst[i3  ]=src[i3  ]+f*(sx/c-src[i3  ]);
        dst[i3+1]=src[i3+1]+f*(sy/c-src[i3+1]);
        dst[i3+2]=src[i3+2]+f*(sz/c-src[i3+2]);
      }
    };
    for(let it=0;it<iters;it++){ step(p,q,lam); step(q,p,mu); }
    return p;
}
function vertexNormals(pos,idx){
    const nrm=new Float32Array(pos.length);
    for(let t=0;t<idx.length;t+=3){
      const a=idx[t]*3,b=idx[t+1]*3,c=idx[t+2]*3;
```

```
      const ux=pos[b]-pos[a],uy=pos[b+1]-pos[a+1],uz=pos[b+2]-pos[a+2];
      const vx=pos[c]-pos[a],vy=pos[c+1]-pos[a+1],vz=pos[c+2]-pos[a+2];
      const nx=uy*vz-uz*vy, ny=uz*vx-ux*vz, nz=ux*vy-uy*vx;
      nrm[a]+=nx;nrm[a+1]+=ny;nrm[a+2]+=nz;
      nrm[b]+=nx;nrm[b+1]+=ny;nrm[b+2]+=nz;
      nrm[c]+=nx;nrm[c+1]+=ny;nrm[c+2]+=nz;
    }
    for(let i=0;i<nrm.length;i+=3){
      const l=Math.hypot(nrm[i],nrm[i+1],nrm[i+2])||1;
      nrm[i]/=l;nrm[i+1]/=l;nrm[i+2]/=l;
    }
    return nrm;
}
function buildMesh(vol,P){
    const {data,dims,dx,dy,R,yTop}=vol;
    const t0=performance.now();
    const {vertices,faces}=surfaceNets(data,dims);
    log(`Surface Nets: ${vertices.length} vertices, ${faces.length} triangles (${(performance.now()-t0|0)} ms)`);
    if(!vertices.length) throw new Error('Iso-surface is empty.');
    const [nx,ny,nz]=dims, sxy=nx*ny;
    const gAt=(i,j,k)=>data[clamp(k,0,nz-1)*sxy+clamp(j,0,ny-1)*nx+clamp(i,0,nx-1)];
    // orientation vote using the field gradient (outward = increasing field)
    let agree=0,dis=0;
    const stride=Math.max(1,Math.floor(faces.length/2000));
    for(let t=0;t<faces.length;t+=stride){
      const [a,b,c]=faces[t], A=vertices[a],B=vertices[b],C=vertices[c];
      const ux=B[0]-A[0],uy=B[1]-A[1],uz=B[2]-A[2];
      const vx=C[0]-A[0],vy=C[1]-A[1],vz=C[2]-A[2];
      const nx_=uy*vz-uz*vy, ny_=uz*vx-ux*vz, nz_=ux*vy-uy*vx;
      const i=Math.round((A[0]+B[0]+C[0])/3),j=Math.round((A[1]+B[1]+C[1])/3),k=Math.round((A[2]+B[2]+C[2])/3);
      const gx=gAt(i+1,j,k)-gAt(i-1,j,k), gy=gAt(i,j+1,k)-gAt(i,j-1,k), gz=gAt(i,j,k+1)-gAt(i,j,k-1);
      (nx_*gx+ny_*gy+nz_*gz>0)?agree++:dis++;
    }
    const sx=P.flipX?-1:1;
    const reflections=1+(P.flipX?1:0);                    // Y is mirrored (image rows go down)
    const flipWinding=(dis>agree)!==(reflections%2===1);
    // grid → world (pixel units; Y up)
    const nv=vertices.length;
    const pos=new Float32Array(nv*3);
    for(let i=0;i<nv;i++){
      const v=vertices[i];
      pos[i*3  ]=sx*(-R+v[0]*dx);
      pos[i*3+1]=-(yTop+v[1]*dy);
      pos[i*3+2]=-R+v[2]*dx;
    }
```

```
  const idx=new Uint32Array(faces.length*3);
  for(let t=0;t<faces.length;t++){
    const f=faces[t];
    if(flipWinding){ idx[t*3]=f[0];idx[t*3+1]=f[2];idx[t*3+2]=f[1]; }
    else{ idx[t*3]=f[0];idx[t*3+1]=f[1];idx[t*3+2]=f[2]; }
  }
  let p=pos;
  if(P.meshSm>0){ p=taubin(pos,idx,P.meshSm); log(`Taubin smoothing: ${P.meshSm} iteration pairs`); }
  // normalise: axis at origin in X/Z, base at Y=0, height = requested
  let miny=Infinity,maxy=-Infinity,minx=Infinity,maxx=-Infinity,minz=Infinity,maxz=-Infinity;
  for(let i=0;i<p.length;i+=3){
    if(p[i]<minx)minx=p[i]; if(p[i]>maxx)maxx=p[i];
    if(p[i+1]<miny)miny=p[i+1]; if(p[i+1]>maxy)maxy=p[i+1];
    if(p[i+2]<minz)minz=p[i+2]; if(p[i+2]>maxz)maxz=p[i+2];
  }
  const hpx=Math.max(1e-6,maxy-miny), k=(P.outH>0?P.outH:1)/hpx;
  const cx=(minx+maxx)/2, cz=(minz+maxz)/2;
  for(let i=0;i<p.length;i+=3){
    p[i  ]=(p[i]-cx)*k;
    p[i+1]=(p[i+1]-miny)*k;
    p[i+2]=(p[i+2]-cz)*k;
  }
  const nrm=vertexNormals(p,idx);
  const dimsOut={x:(maxx-minx)*k,y:hpx*k,z:(maxz-minz)*k};
  return {pos:p,idx,nrm,dimsOut,triangles:faces.length,verts:nv};
}

/* ========================================================================== */
/*   6. EXPORT                                                                */
/* ========================================================================== */
function download(name,text,type){
  const blob=new Blob([text],{type:type||'text/plain'});
  const a=document.createElement('a');
  a.href=URL.createObjectURL(blob); a.download=name;
  document.body.appendChild(a); a.click(); a.remove();
  setTimeout(()=>URL.revokeObjectURL(a.href),4000);
}
function toOBJ(m,P){
  const L=[];
  L.push('# OBJ generated by Silhouette-to-3D (turntable space carving / visual hull)');
  L.push('# '+new Date().toISOString());
  L.push(`# views=${S.views.length} grid=${P.res} consistency=${P.cons} volSmooth=${P.volSm}
meshSmooth=${P.meshSm}`);
  L.push(`# projection=${P.proj} rotation=${P.totalRot}deg direction=${P.dir>0?'CCW':'CW'} mirrored=${P.flipX}`);
  L.push(`# vertices=${m.verts} triangles=${m.triangles}`);
```

```
  L.push('o VisualHull');
  const p=m.pos,n=m.nrm,f=m.idx;
  for(let i=0;i<p.length;i+=3) L.push(`v ${p[i].toFixed(6)} ${p[i+1].toFixed(6)} ${p[i+2].toFixed(6)}`);
  if(P.wNorm) for(let i=0;i<n.length;i+=3) L.push(`vn ${n[i].toFixed(4)} ${n[i+1].toFixed(4)} ${n[i+2].toFixed(4)}`);
  for(let t=0;t<f.length;t+=3){
    const a=f[t]+1,b=f[t+1]+1,c=f[t+2]+1;
    L.push(P.wNorm?`f ${a}//${a} ${b}//${b} ${c}//${c}`:`f ${a} ${b} ${c}`);
  }
  return L.join('\n')+'\n';
}
function toPLY(m){
  const p=m.pos,n=m.nrm,f=m.idx,nv=m.verts,nf=m.triangles,L=[];
  L.push('ply','format ascii 1.0','comment Silhouette-to-3D visual hull',
    `element vertex ${nv}`,'property float x','property float y','property float z',
    'property float nx','property float ny','property float nz',
    `element face ${nf}`,'property list uchar int vertex_indices','end_header');
  for(let i=0;i<nv;i++) L.push(`${p[i*3].toFixed(6)} ${p[i*3+1].toFixed(6)} ${p[i*3+2].toFixed(6)} ${n[i*3].toFixed(4)}
${n[i*3+1].toFixed(4)} ${n[i*3+2].toFixed(4)}`);
  for(let t=0;t<f.length;t+=3) L.push(`3 ${f[t]} ${f[t+1]} ${f[t+2]}`);
  return L.join('\n')+'\n';
}

/* ============================================================================ */
/*  7. VIEWS — silhouette strip / inspector / software-rasterised 3D preview */
/* ============================================================================ */
function drawStrip(){
  const strip=$('strip'); strip.innerHTML='';
  const tw=88, th=Math.max(20,Math.round(tw*S.H/S.W));
  S.views.forEach((v,i)=>{
    const c=document.createElement('canvas'); c.width=tw; c.height=th;
    if(i===S.sel) c.className='sel';
    const g=c.getContext('2d');
    g.drawImage(S.frames[i].cv,0,0,tw,th);
    const im=g.getImageData(0,0,tw,th), d=im.data;
    for(let y=0;y<th;y++)for(let x=0;x<tw;x++){
      const sx=Math.min(S.W-1,Math.round(x*S.W/tw)), sy=Math.min(S.H-1,Math.round(y*S.H/th));
      const o=(y*tw+x)*4;
      if(v.mask[sy*S.W+sx]){ d[o]=Math.min(255,d[o]*.35+180);d[o+1]=d[o+1]*.35;d[o+2]=d[o+2]*.35+40; }
      else { const l=(d[o]+d[o+1]+d[o+2])/3*0.45; d[o]=d[o+1]=d[o+2]=l; }
    }
    g.putImageData(im,0,0);
    g.strokeStyle='#4da3ff'; g.beginPath();
    const ax=(S.geom?S.geom.axisX:S.W/2)*tw/S.W; g.moveTo(ax,0);g.lineTo(ax,th);g.stroke();
    g.fillStyle='#9fb'; g.font='9px monospace';
    g.fillText((v.angle).toFixed(0)+'°',3,th-3);
```

```
    c.title=`#${i} ${S.frames[i].label} · ${v.angle.toFixed(1)}° · area ${v.area}px`;
    c.onclick=()=>{S.sel=i;drawStrip();drawInspector();};
    strip.appendChild(c);
  });
}
function drawInspector(){
  const cv=$('ins'), g=cv.getContext('2d');
  if(!S.views.length){ g.clearRect(0,0,cv.width,cv.height); return; }
  const v=S.views[S.sel];
  const scale=Math.min(520/S.W,420/S.H);
  cv.width=Math.round(S.W*scale); cv.height=Math.round(S.H*scale);
  g.imageSmoothingEnabled=false;
  g.drawImage(S.frames[S.sel].cv,0,0,cv.width,cv.height);
  // mask outline
  const tmp=document.createElement('canvas'); tmp.width=S.W; tmp.height=S.H;
  const tg=tmp.getContext('2d'); const im=tg.createImageData(S.W,S.H), d=im.data;
  for(let y=0;y<S.H;y++)for(let x=0;x<S.W;x++){
    const i=y*S.W+x; if(!v.mask[i]) continue;
    const edge=(x===0||y===0||x===S.W-1||y===S.H-1)||!v.mask[i-1]||!v.mask[i+1]||!v.mask[i-S.W]||!v.mask[i+S.W];
    const o=i*4;
    if(edge){ d[o]=255;d[o+1]=80;d[o+2]=80;d[o+3]=255; } else { d[o]=90;d[o+1]=190;d[o+2]=255;d[o+3]=70; }
  }
  tg.putImageData(im,0,0);
  g.drawImage(tmp,0,0,cv.width,cv.height);
  if(S.geom){
    const {axisX,yTop,yBottom,R}=S.geom;
    g.strokeStyle='#4da3ff'; g.lineWidth=1; g.setLineDash([5,4]);
    g.beginPath(); g.moveTo(axisX*scale,0); g.lineTo(axisX*scale,cv.height); g.stroke();
    g.setLineDash([]); g.strokeStyle='#43c98b';
    g.strokeRect((axisX-R)*scale,yTop*scale,2*R*scale,(yBottom-yTop)*scale);
  }
  $('insLbl').textContent=`#${S.sel} ${S.frames[S.sel].label} · ${v.angle.toFixed(1)}° · thr ${v.thr.toFixed(3)}`;
}

/* --------- minimal z-buffered software renderer (no WebGL needed) -------- */
const VIEW={yaw:0.6,pitch:-0.25,zoom:1,panx:0,pany:0,drag:null};
function drawMesh(fast){
  const cv=$('view3d'), ctx=cv.getContext('2d');
  const W=cv.width,H=cv.height;
  const img=ctx.createImageData(W,H), D=img.data;
  for(let i=0;i<D.length;i+=4){ D[i]=14;D[i+1]=17;D[i+2]=22;D[i+3]=255; }
  const m=S.mesh;
  if(!m){ ctx.putImageData(img,0,0);
    ctx.fillStyle='#93a1b5';ctx.font='13px system-ui';
    ctx.fillText('Reconstruct to see the 3D model here.',18,28); return; }
```

```
// normalise for viewing
if(!m._vpos){
  let minx=1e9,maxx=-1e9,miny=1e9,maxy=-1e9,minz=1e9,maxz=-1e9,p=m.pos;
  for(let i=0;i<p.length;i+=3){
    if(p[i]<minx)minx=p[i];if(p[i]>maxx)maxx=p[i];
    if(p[i+1]<miny)miny=p[i+1];if(p[i+1]>maxy)maxy=p[i+1];
    if(p[i+2]<minz)minz=p[i+2];if(p[i+2]>maxz)maxz=p[i+2];
  }
  const cx=(minx+maxx)/2,cy=(miny+maxy)/2,cz=(minz+maxz)/2;
  const s=2/Math.max(maxx-minx,maxy-miny,maxz-minz,1e-6);
  const q=new Float32Array(p.length);
  for(let i=0;i<p.length;i+=3){ q[i]=(p[i]-cx)*s;q[i+1]=(p[i+1]-cy)*s;q[i+2]=(p[i+2]-cz)*s; }
  m._vpos=q;
}
const p=m._vpos, idx=m.idx, nv=p.length/3;
const cy=Math.cos(VIEW.yaw),sy=Math.sin(VIEW.yaw),cp=Math.cos(VIEW.pitch),sp=Math.sin(VIEW.pitch);
const tx=new Float32Array(nv),ty=new Float32Array(nv),tz=new Float32Array(nv);
const sxA=new Float32Array(nv),syA=new Float32Array(nv);
const dist=4.2/VIEW.zoom, f=Math.min(W,H)*0.95;
for(let i=0;i<nv;i++){
  const x=p[i*3],y=p[i*3+1],z=p[i*3+2];
  const x1=x*cy+z*sy, z1=-x*sy+z*cy;
  const y2=y*cp-z1*sp, z2=y*sp+z1*cp;
  tx[i]=x1;ty[i]=y2;tz[i]=z2;
  const w=dist-z2; const iw=w>0.05?1/w:1/0.05;
  sxA[i]=W/2+f*x1*iw+VIEW.panx; syA[i]=H/2-f*y2*iw+VIEW.pany;
}
const zb=new Float32Array(W*H).fill(-1e30);
const step=fast&&idx.length>300000?2:1;      // decimate while dragging huge meshes
const Lx=0.38,Ly=0.55,Lz=0.74;
for(let t=0;t<idx.length;t+=3*step){
  const a=idx[t],b=idx[t+1],c=idx[t+2];
  const ax=sxA[a],ay=syA[a],bx=sxA[b],by=syA[b],cx2=sxA[c],cy2=syA[c];
  const area=(bx-ax)*(cy2-ay)-(by-ay)*(cx2-ax);
  if(area===0) continue;
  let x0=Math.max(0,Math.floor(Math.min(ax,bx,cx2))), x1=Math.min(W-1,Math.ceil(Math.max(ax,bx,cx2)));
  let y0=Math.max(0,Math.floor(Math.min(ay,by,cy2))), y1=Math.min(H-1,Math.ceil(Math.max(ay,by,cy2)));
  if(x1<x0||y1<y0) continue;
  // camera-space face normal → flat shading (two-sided)
  const ux=tx[b]-tx[a],uy=ty[b]-ty[a],uz=tz[b]-tz[a];
  const vx=tx[c]-tx[a],vy=ty[c]-ty[a],vz=tz[c]-tz[a];
  let nx=uy*vz-uz*vy, ny=uz*vx-ux*vz, nz=ux*vy-uy*vx;
  const nl=Math.hypot(nx,ny,nz)||1; nx/=nl;ny/=nl;nz/=nl;
  const diff=Math.abs(nx*Lx+ny*Ly+nz*Lz);
  const sh=0.16+0.84*diff, spec=Math.pow(diff,18)*0.35;
```

```
      const cr=clamp((0.62*sh+spec)*255,0,255)|0,
            cg=clamp((0.72*sh+spec)*255,0,255)|0,
            cb=clamp((0.86*sh+spec)*255,0,255)|0;
      const inv=1/area;
      for(let y=y0;y<=y1;y++){
        const py=y+0.5;
        for(let x=x0;x<=x1;x++){
          const px=x+0.5;
          let w0=((bx-ax)*(py-ay)-(by-ay)*(px-ax))*inv;
          let w1=((px-ax)*(cy2-ay)-(py-ay)*(cx2-ax))*inv;
          // barycentric: w1->b? use standard formulation
          const l2=w0, l1=w1, l0=1-l1-l2;
          if(l0<0||l1<0||l2<0) continue;
          const z=l0*tz[a]+l1*tz[b]+l2*tz[c];
          const o=y*W+x;
          if(z<=zb[o]) continue;
          zb[o]=z;
          const q=o*4; D[q]=cr;D[q+1]=cg;D[q+2]=cb;
        }
      }
    }
    ctx.putImageData(img,0,0);
    ctx.fillStyle='#93a1b5'; ctx.font='11px ui-monospace,monospace';
    ctx.fillText(`${m.verts} verts · ${m.triangles} tris`,10,H-10);
}
(function bindViewer(){
    const cv=$('view3d');
    cv.addEventListener('mousedown',e=>{VIEW.drag={x:e.clientX,y:e.clientY,shift:e.shiftKey};});
    window.addEventListener('mouseup',()=>{ if(VIEW.drag){VIEW.drag=null;drawMesh(false);} });
    window.addEventListener('mousemove',e=>{
      if(!VIEW.drag) return;
      const dx=e.clientX-VIEW.drag.x, dy=e.clientY-VIEW.drag.y;
      VIEW.drag.x=e.clientX; VIEW.drag.y=e.clientY;
      if(VIEW.drag.shift){ VIEW.panx+=dx; VIEW.pany+=dy; }
      else{ VIEW.yaw+=dx*0.01; VIEW.pitch=clamp(VIEW.pitch+dy*0.01,-1.5,1.5); }
      drawMesh(true);
    });
    cv.addEventListener('wheel',e=>{e.preventDefault();
      VIEW.zoom=clamp(VIEW.zoom*(e.deltaY<0?1.12:0.9),0.15,14); drawMesh(true);
      clearTimeout(cv._t); cv._t=setTimeout(()=>drawMesh(false),160);
    },{passive:false});
})();

/* ========================================================================== */
/*    8. UI GLUE                                                              */
```

```
/* ========================================================================== */
function params(){
  return {
    procSize:+$('procSize').value, nFrames:+$('nFrames').value,
    t0:clamp(+$('tStart').value/100,0,0.99), t1:clamp(+$('tEnd').value/100,0.01,1),
    segMode:$('segMode').value, thrMode:$('thrMode').value, thrBias:+$('thrBias').value,
    blur:+$('blur').value, mOpen:+$('mOpen').value, mClose:+$('mClose').value,
    largest:$('largest').checked, fill:$('fill').checked, invert:$('invert').checked,
    totalRot:+$('totalRot').value, loop:$('loop').checked, dir:+$('dir').value,
    anglesTxt:$('angles').value, axisShift:+$('axisShift').value,
    proj:$('proj').value, camDist:+$('camDist').value,
    res:+$('res').value, cons:+$('cons').value, volSm:+$('volSm').value,
    meshSm:+$('meshSm').value, seal:$('seal').checked, flipX:$('flipX').checked,
    outH:+$('outH').value, wNorm:$('wNorm').checked
  };
}
[['thrBias','thrBiasV',v=>(+v).toFixed(2)],['blur','blurV'],['mOpen','mOpenV'],['mClose','mCloseV'],
 ['axisShift','axisShiftV'],['camDist','camDistV'],['res','resV'],['cons','consV',v=>(+v).toFixed(2)],
 ['volSm','volSmV'],['meshSm','meshSmV']].forEach(([a,b,fmt])=>{
  const i=$(a),o=$(b); const upd=()=>o.textContent=fmt?fmt(i.value):i.value; i.addEventListener('input',upd); upd();
});
$('axisShift').addEventListener('change',()=>{ if(S.views.length){ try{S.geom=estimateGeometry(params()); drawStrip();
drawInspector();}catch(e){} } });

$('drop').onclick=()=>$('file').click();
$('drop').addEventListener('dragover',e=>{e.preventDefault();$('drop').classList.add('hot');});
$('drop').addEventListener('dragleave',()=>$('drop').classList.remove('hot'));
$('drop').addEventListener('drop',e=>{e.preventDefault();$('drop').classList.remove('hot');handleFiles(e.dataTransfer.files);});
$('file').addEventListener('change',e=>handleFiles(e.target.files));
$('btnDemo').onclick=async()=>{
  $('modeBadge').textContent='synthetic demo';
  log('Generating synthetic turntable silhouettes (32 views)…');
  await nextTick();
  const {frames,W,H}=demoData(32,300);
  S.mode='demo'; S.frames=frames; S.W=W; S.H=H;
  $('segMode').value='bright';
  await runSegmentation();
};
$('btnClear').onclick=()=>{
  S.mode=null;S.frames=[];S.views=[];S.geom=null;S.mesh=null;S.sel=0;
  $('modeBadge').textContent='no input'; $('strip').innerHTML='';
  $('stats').innerHTML=''; $('btnSeg').disabled=$('btnRun').disabled=$('btnObj').disabled=$('btnPly').disabled=true;
  $('ins').getContext('2d').clearRect(0,0,999,999); drawMesh(false); prog(0);
  log('Cleared.');
};
```

```
async function handleFiles(list){
  const files=[...list]; if(!files.length) return;
  const P=params();
  const vid=files.find(f=>f.type.startsWith('video/')||/\.(mp4|webm|mov|m4v|ogv|avi|mkv)$/i.test(f.name));
  try{
    prog(0);
    if(vid){
      S.mode='video'; $('modeBadge').textContent='video · turntable';
      log(`Detected ROTATION VIDEO: ${vid.name}`,'ok');
      const r=await loadVideo(vid,clamp(P.nFrames,3,240),P.t0,P.t1,P.procSize);
      S.frames=r.frames;S.W=r.W;S.H=r.H;
    }else{
      const imgs=files.filter(f=>f.type.startsWith('image/')||/\.(png|jpe?g|bmp|gif|webp)$/i.test(f.name));
      if(imgs.length<2) throw new Error('Provide a rotation video, or at least 2 (ideally 12–72) silhouette images.');
      S.mode='images'; $('modeBadge').textContent=`${imgs.length} images · turntable`;
      log(`Detected ${imgs.length} SILHOUETTE IMAGES (sorted naturally by filename)`,'ok');
      const r=await loadImages(imgs,P.procSize);
      S.frames=r.frames;S.W=r.W;S.H=r.H;
    }
    await runSegmentation();
  }catch(err){ log('ERROR: '+err.message,'err'); prog(0); }
}
async function runSegmentation(){
  const P=params();
  if(!S.frames.length){ log('No frames loaded.','warn'); return; }
  log(`Segmenting ${S.frames.length} frames (method=${P.segMode}, ${P.thrMode})…`);
  await nextTick();
  S.views=[];
  const angles=assignAngles(P,S.frames.length);
  for(let i=0;i<S.frames.length;i++){
    const r=segment(S.frames[i].cv,S.W,S.H,P);
    r.angle=angles[i]; S.views.push(r);
    prog((i+1)/S.frames.length);
    if(i%5===0) await nextTick();
  }
  const areas=S.views.map(v=>v.area), tot=S.W*S.H;
  const empty=areas.filter(a=>a<20).length;
  log(`Silhouette coverage: min ${(100*Math.min(...areas)/tot).toFixed(1)}% · max
${(100*Math.max(...areas)/tot).toFixed(1)}%`);
  if(empty) log(`${empty} frame(s) look empty — tune the threshold / try "Invert mask".`,'warn');
  try{ S.geom=estimateGeometry(P);
    log(`Axis X = ${S.geom.axisX.toFixed(1)} px · radius = ${S.geom.R.toFixed(1)} px · rows
${S.geom.yTop}–${S.geom.yBottom}`,'ok');
  }catch(e){ log('ERROR: '+e.message,'err'); }
  S.sel=0; drawStrip(); drawInspector();
```

```
  $('btnSeg').disabled=false; $('btnRun').disabled=!S.geom;
  prog(1); setTimeout(()=>prog(0),400);
}
$('btnSeg').onclick=()=>runSegmentation();

$('btnRun').onclick=async()=>{
  const P=params();
  $('btnRun').disabled=true;
  try{
    if(!S.geom) S.geom=estimateGeometry(P);
    log('— Space carving —');
    prog(0.02); await nextTick();
    const vol=await carve(P);
    prog(0.65); await nextTick();
    S.mesh=buildMesh(vol,P);
    S.mesh._vpos=null;
    const d=S.mesh.dimsOut;
    $('stats').innerHTML=`
      <span>Vertices</span><b>${S.mesh.verts}</b>
      <span>Triangles</span><b>${S.mesh.triangles}</b>
      <span>Views used</span><b>${S.views.filter(v=>v.area>0).length}</b>
      <span>Size X×Y×Z</span><b>${d.x.toFixed(3)} × ${d.y.toFixed(3)} × ${d.z.toFixed(3)}</b>`;
    log(`Mesh ready: ${S.mesh.verts} vertices / ${S.mesh.triangles} triangles`,'ok');
    $('btnObj').disabled=$('btnPly').disabled=false;
    prog(1); drawMesh(false);
    setTimeout(()=>prog(0),600);
  }catch(err){ log('ERROR: '+err.message,'err'); prog(0); }
  $('btnRun').disabled=false;
};
$('btnObj').onclick=()=>{ const t=toOBJ(S.mesh,params());
  download('reconstruction.obj',t,'model/obj'); log(`Exported reconstruction.obj (${(t.length/1048576).toFixed(2)} MB)`,'ok'); };
$('btnPly').onclick=()=>{ const t=toPLY(S.mesh);
  download('reconstruction.ply',t); log('Exported reconstruction.ply','ok'); };

/* boot */
log('Ready. Load a rotation video or a set of silhouette images — or press "Load synthetic demo".');
drawMesh(false);
</script>
</body>
</html>
```

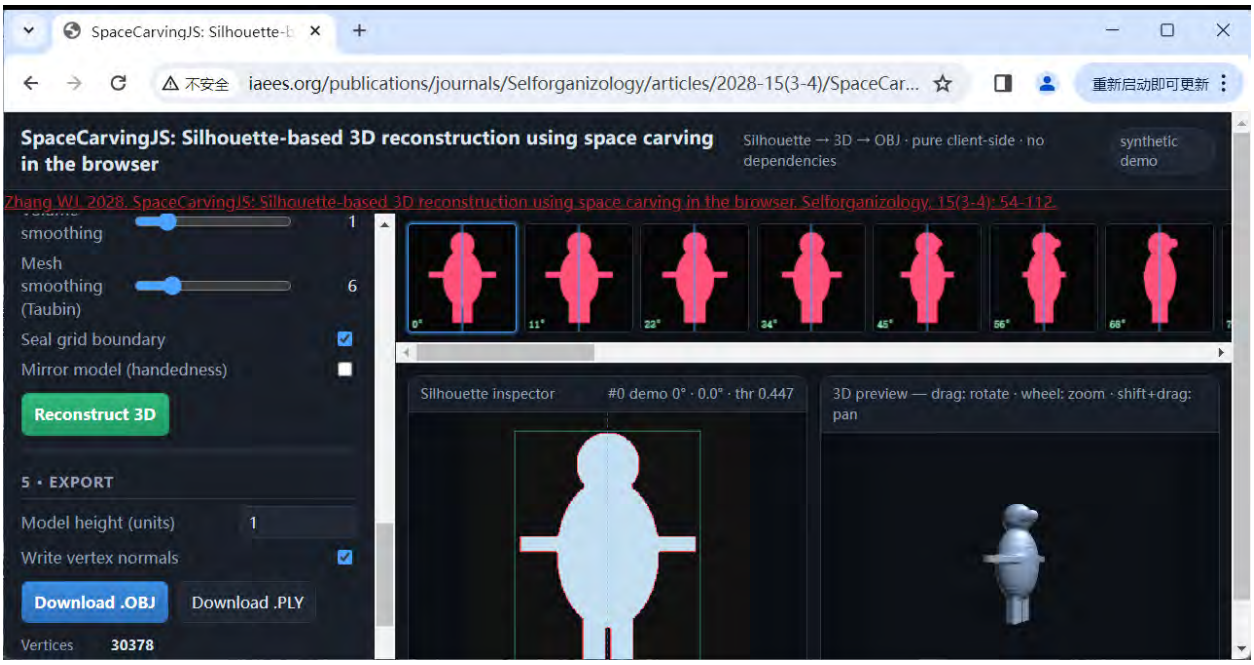

Fig. 1 The browser-based tool page.

References

Hornung A, Kobbelt L. 2006. Hierarchical volumetric multi-view stereo reconstruction of manifold surfaces based on dual graph embedding. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR). https://ieeexplore.ieee.org/document/1640798

Kutulakos KN, Seitz SM. 2000. A theory of shape by space carving. International Journal of Computer Vision, 38(3): 199-218. https://link.springer.com/article/10.1023/A:1008191222954

Laurentini A. 1994. The visual hull concept for silhouette-based image understanding. IEEE Transactions on Pattern Analysis and Machine Intelligence, 16(2): 150-162. https://ieeexplore.ieee.org/document/273735

Matusik W, Buehler C, Raskar R, Gortler SJ, McMillan L. 2000. Image-based visual hulls. Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH), 369-374. https://dl.acm.org/doi/10.1145/344779.344951

Sinha SN, Pollefeys M. 2005. Multi-view reconstruction using photo-consistency and exact silhouette constraints: A maximum-flow formulation Proceedings of the IEEE International Conference on Computer Vision (ICCV), 349-356. https://mlanthology.org/iccv/2005/sinha2005iccv-multi/

Zhang WJ. 2024. objectVisual3D: A standalone executable software for 3D visualization of objects. Selforganizology, 11(3-4): 28-53. http://www.iaees.org/publications/journals/selforganizology/articles/2024-11(3-4)/ 1-Zhang-Abstract.asp

Zhang WJ. 2027. NetGen 4.0: The browser-based tool for network visualization. Network Biology, 17(2): 468-537. http://www.iaees.org/publications/journals/nb/articles/2027-17(2)/2-Zhang-Abstract.asp

Zhang WJ. 2028. A browser-based interactive 3D visualizer for molecular and object structures. Selforganizology, 15(1-2): 21-53. http://www.iaees.org/publications/journals/selforganizology/articles/2028-15(1-2)/2-Zhang-Abstract.asp