

Article

3D-Visualizer 2.0: A universal 3D structure visualizer and data converter

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 31 July 2026; Accepted 10 August 2026; Published online 25 August 2026; Published 1 December 2028



Abstract

3D-Visualizer 2.0, developed on the basis of the earlier 3D-Visualizer, is a universal browser-based tool for viewing, exploring, and converting many kinds of 3D structure data without installing specialized software. Built as a self-contained HTML/CSS/JavaScript application, it runs entirely on the client side and can be used offline after loading. The tool is designed to provide a lightweight solution for rapid inspection of scientific, molecular, geometrical, and volumetric datasets. The program supports a wide range of input formats. These include common 3D geometry and mesh formats such as OBJ, STL, PLY, OFF, VTK, 3DS, FBX, glTF/GLB, DAE, X3D, 3MF, and DXF; molecular formats such as XYZ, PDB, MOL, SDF, MOL2, CML, and CUBE; and volumetric or CAD-related formats such as MRC/CCP4/MAP, STEP/STP, and IGES/IGS. According to the input type, structures can be displayed as wireframes, flat-shaded surfaces, point clouds, or ball-and-stick molecular models. The system also supports export to multiple widely used formats, enabling practical cross-format conversion for reuse in other software environments. 3D-Visualizer 2.0 combines file parsers, format writers, a custom 3D affine transformation engine, and a 2D Canvas renderer in a single compact platform. Molecular visualization includes automatic bond detection based on atomic distances and covalent radii, together with standard element coloring. Volumetric datasets can be previewed through point-cloud extraction at a selected iso level, allowing quick examination of density distributions. Interactive operations include rotation, zooming, panning, resetting the view, saving PNG images, and adjusting display parameters such as canvas size, zoom scale, bond tolerance, ball size, iso threshold, and point limits. Because it is portable, easy to use, and independent of operating system and installation environment, 3D-Visualizer 2.0 is suitable for education, preliminary data inspection, lightweight scientific visualization, and convenient 3D format conversion in both online and offline settings.

Keywords 3D visualization; browser-based software; data conversion; molecular visualization; mesh rendering; volumetric data; CAD data; JavaScript; client-side computing; scientific visualization.

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 (3D) structural visualization has become an indispensable methodology across a wide spectrum of scientific and engineering disciplines, ranging from structural biology and drug discovery to computer-aided design, 3D printing, and digital manufacturing. The ability to intuitively inspect molecular conformations, protein assemblies, geometric models, and volumetric data not only facilitates hypothesis generation and experimental planning but also serves as a critical step in data interpretation and communication (O'Donoghue et al., 2010). In cheminformatics and bioinformatics, 3D representations of macromolecules are essential for understanding structure–function relationships, binding modes, and allosteric mechanisms, while in engineering and architecture, accurate visual inspection of CAD models is fundamental to design validation and quality assurance.

The landscape of 3D structure data is remarkably heterogeneous, encompassing a multitude of file formats tailored to specific domains. In computer graphics, game development, and 3D printing, common geometry and mesh formats include OBJ, a versatile static geometry exchange format storing vertices, texture coordinates, and normals, and STL, the de facto standard for 3D printing that represents surfaces as triangular meshes without color or texture support. FBX, developed by Autodesk, offers comprehensive scene, animation, skeleton, and material data and is widely adopted in gaming and animation industries. With the rise of web and AR/VR technologies, glTF has emerged as an open standard, and its binary form GLB provides high compactness and efficiency. PLY is frequently used for point clouds and polygon meshes from 3D scanning, while formats such as 3DS, COLLADA, X3D, OFF, and 3MF serve various specialized roles, with 3MF addressing STL's limitations by supporting color and material information. In chemistry, biology, and physics, molecular and scientific computing formats include XYZ, the simplest structure format with only atomic symbols and Cartesian coordinates, widely used in quantum chemistry, and PDB, the standard for biomacromolecules, which encodes atomic coordinates, occupancy, and temperature factors. MDL Molfile explicitly stores bonding information and bond orders, while MOL2, CML, Gaussian Cube, CCP4/MRC, and VTK serve specialized purposes in molecular property storage, volumetric data representation, and scientific visualization. In CAD and engineering, STEP serves as the international standard for precise representation of solids, surfaces, and assemblies, while IGES, DXF, and DWG facilitate data exchange among CAD systems. The choice of format depends entirely on the application: STL or 3MF for 3D printing, FBX or glTF for animation, glTF for web deployment, OBJ for general geometry exchange, XYZ/PDB/MOL for molecular chemistry, and STEP/IGES for engineering. Notably, many formats have multiple variants, and specialized software is often required to handle each format properly.

Despite the abundance of formats and the availability of numerous visualization tools, several significant challenges remain. Standalone desktop applications such as PyMOL and ChimeraX (Goddard et al., 2018) offer powerful molecular visualization but require local installation, may involve licensing costs, and are not always cross-platform friendly. Web-based solutions like 3Dmol.js (Rego and Koes, 2015), NGL Viewer (Rose et al., 2018), and Mol* Viewer (Sehna et al., 2021) provide accessible browser-based visualization, yet they primarily target molecular structures and rely on WebGL, which may not be available in all environments or may impose performance overhead. Other online tools such as the Online 3D Viewer (Kovács) support multiple formats but often depend on external libraries or server-side processing. Furthermore, many existing tools are designed for specific data types and lack comprehensive support for the full spectrum of formats, from molecular coordinates to CAD assemblies to volumetric cryo-EM maps (Wang et al., 2016). Even when format support is broad, bidirectional conversion between formats is rarely offered, forcing users to rely on separate conversion utilities or scripting libraries such as those in the SciPy ecosystem (Virtanen et al., 2020). Earlier contributions by Zhang (2023) developed visual3D 1.0, an integrative Java software for 3D

visualization of objects and molecules using XYZ and OBJ files, and Zhang (2024) introduced objectVisual3D, a standalone executable for object visualization from OBJ. A recent browser-based tool by Zhang (2028a) implemented fundamental 3D visualization using only the 2D Canvas API (without WebGL) for XYZ and OBJ formats, and Zhang (2028b) extended browser-based capabilities to silhouette-based 3D reconstruction. However, these earlier tools, while innovative, are limited in format coverage, lack conversion features, or require Java runtime or separate executables. There remains a clear need for a lightweight, dependency-free, client-side solution that can handle diverse 3D structure data types, provide flexible rendering modes, and enable cross-format conversion in a single unified platform.

To address these gaps, the present work introduces 3D-Visualizer 2.0, a universal browser-based tool built on the foundation of the earlier 3DVisualizer (Zhang, 2028a). This new version substantially extends format support to encompass common geometry and mesh formats (OBJ, STL, PLY, OFF, VTK, 3DS, FBX, glTF/GLB, DAE, X3D, 3MF, DXF), molecular formats (XYZ, PDB, MOL, SDF, MOL2, CML, CUBE), and volumetric or CAD-related formats (MRC/CCP4/MAP, STEP/STP, IGES/IGS). Implemented as a self-contained HTML/CSS/JavaScript application that runs entirely on the client side, it can be used offline after loading, requiring no installation or external dependencies. The system supports multiple rendering modes, wireframes, flat-shaded surfaces, point clouds, and ball-and-stick molecular models, with automatic bond detection based on atomic distances and covalent radii, together with standard element coloring. Volumetric datasets can be previewed through pointcloud extraction at user-selected iso levels. Interactive operations include rotation, zooming, panning, view resetting, PNG image export, and adjustable display parameters such as canvas size, zoom scale, bond tolerance, ball size, iso threshold, and point limits. Importantly, the tool also provides export to multiple widely used formats, enabling practical cross-format data conversion for reuse in other software environments. By combining comprehensive format support, client-side independence, and bidirectional conversion capabilities in a single lightweight platform, 3D-Visualizer 2.0 aims to serve as a practical solution for education, preliminary data inspection, lightweight scientific visualization, and convenient 3D data conversion in both online and offline settings.

2 Universal 3D Structure Visualizer and Data Converter: 3D-Visualizer 2.0

2.1 Overview

The tool is a single-page, browser-based 3D structure viewer and data converter implemented in plain HTML/CSS/JavaScript. It supports three broad classes of 3D data:

1. **Mesh / solid geometry**
 - o Examples: .obj, .stl, .ply, .off, .3ds, .fbx, .gltf, .glb, .dae, .x3d, .3mf, .dxf, .vtk
 - o Displayed as: **wireframe** or **flat-shaded surfaces**
2. **Molecular / chemical structures**
 - o Examples: .xyz, .pdb, .mol, .sdf, .mol2, .cml, .cube
 - o Displayed as: **ball-and-stick**
3. **Volumetric data / CAD exchange point-like sampling**
 - o Examples: .mrc, .ccp4, .map, .step, .stp, .igs, .iges
 - o Displayed as: **point cloud**

The application works **entirely in the browser**:

- no server-side rendering,
- no external library dependency,
- direct file upload,
- direct conversion and download,

- PNG export of current view.

Its architecture combines:

- a **custom 3D transformation matrix class**,
- many **format parsers**,
- many **export writers**,
- a **2D canvas software renderer**,
- and a **self-test harness**.

2.2 Primary Purpose

The application is designed as a universal offline visualizer and converter for scientific and geometric 3D data.

Its purposes are:

- **Visual inspection** of imported 3D structures
- **Cross-format data conversion**
- **Previewing unsupported/legacy formats** without dedicated CAD or molecular software
- **Educational demonstration** of 3D rendering, file parsing, and geometry processing in JavaScript
- **Portable deployment** as a single HTML file

This tool is especially useful when a user needs:

- quick visualization of a 3D mesh,
- molecular structure display without installing chemistry software,
- basic iso-threshold extraction from electron density maps,
- rough point-based preview of STEP/IGES files,
- browser-based conversion into common interchange formats.

2.3 Main Features

2.3.1 Supported Import Formats

Mesh / solid

- OBJ
- STL (ASCII + binary)
- PLY (ASCII + binary)
- OFF
- VTK (legacy ASCII)
- 3DS
- FBX (ASCII + binary, partial support)
- glTF / GLB
- DAE (COLLADA)
- X3D
- 3MF
- DXF (ASCII)

Molecular

- XYZ
- PDB / ENT
- MOL / SDF (V2000)
- MOL2
- CML
- Gaussian CUBE

Volumetric / CAD exchange

- MRC / CCP4 / MAP
- STEP / STP
- IGES / IGS

2.3.2 Supported Export Formats

Depending on model type:

- OBJ
- STL (binary and internally also ASCII option)
- PLY
- OFF
- VTK
- 3MF
- GLB
- X3D
- XYZ
- PDB
- MOL
- SDF
- MOL2
- CML

2.3.3 Rendering Styles

The display style is selected automatically but can be changed manually.

- wire — wireframe
- flat — flat-shaded triangles
- points — point cloud
- ballstick — ball-and-stick molecular style
- auto — chosen by model category

Automatic mapping:

- molecular → ball-and-stick
- point set → points
- mesh with triangles → flat
- mesh without triangles → wire

2.3.4 Interaction Features

- **Drag** → rotate
- **Left click** → zoom in
- **Right click** → zoom out
- **Mouse wheel** → zoom
- **Horizontal / vertical sliders** → pan
- **Save as PNG**
- **Reset view**
- **Back / close**
- **Run self-test**

2.4 High-Level Architecture

The application can be divided into these modules:

1. **Utility functions**
2. **3D matrix / transform engine (Mat3D)**
3. **Model container (Model)**
4. **Chemistry tables and bond perception**
5. **ZIP reading/writing**
6. **Format parsers**
7. **Volumetric point-cloud generation**
8. **Format dispatch**
9. **Format writers**
10. **Canvas renderer**
11. **UI wiring**
12. **Self-test framework**

2.5 Data Model

The central structure is the Model class.

2.5.1 Model fields

js

```
class Model {
  constructor(kind){
    this.kind=kind; this.verts=null; this.tris=null; this.edges=null;
    this.atoms=null; this.bonds=null; this.colors=null; this.volume=null;
    this.source="";
  }
}
```

Semantic meaning

- kind: 'mesh' | 'mol' | 'points'
- verts: Float32Array, length 3N
- tris: Int32Array, length 3T
- edges: explicit edge list
- atoms: chemical element symbols
- bonds: atom index pairs
- colors: per-vertex RGB
- volume: volumetric metadata and scalar field
- source: textual note about parsing or sampling

2.5.2 Bounding box

The model computes axis-aligned bounds:

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

These are used for:

- centering,
- initial scaling,
- informational output,
- export checks.

2.6 Mathematical Core: Mat3D

The file uses a custom affine 3D matrix-like object called Mat3D.

2.6.1 Representation

Instead of a conventional 4×4 matrix array, the transform is stored as twelve scalars:

$$x' = x \cdot x + y \cdot y + z \cdot z + x_0$$

$$y' = y \cdot x + y \cdot y + y \cdot z + y_0$$

$$z' = z \cdot x + z \cdot y + z \cdot z + z_0$$

This is effectively a 3×4 affine transform.

2.6.2 Identity transform

The unit() method initializes:

$$1 \quad 0 \quad 0 \quad x_0=0$$

$$0 \quad 1 \quad 0 \quad y_0=0$$

$$0 \quad 0 \quad 1 \quad z_0=0$$

2.6.3 Scaling

Uniform or anisotropic scale:

$$x' = f_x x, y' = f_y y, z' = f_z z$$

If only one parameter is supplied, all axes use the same scale.

2.6.4 Translation

$$x_0 \leftarrow x_0 + t_x, y_0 \leftarrow y_0 + t_y, z_0 \leftarrow z_0 + t_z$$

2.6.5 Rotation about X axis

For angle θ :

$$y' = y \cos \theta + z \sin \theta$$

$$z' = z \cos \theta - y \sin \theta$$

This corresponds to the code:

```
js
xrot(deg){
  const d=deg*Math.PI/180, c=Math.cos(d), s=Math.sin(d);
  ...
}
```

2.6.6 Rotation about Y axis

For angle θ :

$$x' = x \cos \theta + z \sin \theta$$

$$z' = z \cos \theta - x \sin \theta$$

2.6.7 Matrix composition

mult(m) post-composes the current transform with another affine transform.

If current transform is A and new transform is B, then the result is:

$$A \leftarrow A \cdot B$$

implemented explicitly with scalar multiplication and addition.

2.6.8 Vector transformation

For n vertices:

$$\mathbf{v}_i = (x_i, y_i, z_i)$$

the transformed point is:

$$x'_i = x x x_i + x y y_i + x z z_i + x o$$

$$y'_i = y x x_i + y y y_i + y z z_i + y o$$

$$z'_i = z x x_i + z y y_i + z z z_i + z o$$

Output is a new Float32Array(3n).

2.7 Rendering Pipeline

The renderer is a software renderer on 2D Canvas, not WebGL.

2.7.1 Pipeline stages

1. Compute model bounding box
2. Translate model so its center is at origin
3. Apply user rotation/panning matrix
4. Scale to canvas
5. Flip Y axis for screen coordinates
6. Apply pseudo-depth scaling in Z
7. Translate to canvas center
8. Render according to selected style

2.7.2 Initial view setup

In setupView():

js

```
let f=Math.max(bb.xmax-bb.xmin, bb.ymax-bb.ymin, bb.zmax-bb.zmin);
```

```
state.xfac=0.7*Math.min(canvas.width/f, canvas.height/f)*state.scale;
```

Meaning

Let the largest dimension of the model be:

$$f = \max(\Delta x, \Delta y, \Delta z)$$

Then the screen scaling factor is:

$$x_{fac}=0.7 \cdot \min(W/f, H/f) \cdot \text{userScale}$$

This ensures the object fits within the canvas with margin.

2.7.3 Composite display transform

The display matrix is built as:

1. Center object

$$(x, y, z) \rightarrow (x - x_c, y - y_c, z - z_c)$$

2. Apply rotation/pan matrix state.amat

3. Scale

$$x \rightarrow x \cdot x_{fac}, y \rightarrow y \cdot x_{fac}, z \rightarrow z \cdot 16 \cdot x_{fac} / W$$

4. Translate to canvas center

$$x \rightarrow x + W/2, y \rightarrow y + H/2, z \rightarrow z + 8$$

2.8 Rendering Modes

2.8.1 Wireframe

Draw each edge as a line segment.

```
js
for (const [a, b] of edges) {
  ctx.beginPath();
  ctx.moveTo(tv[a * 3], tv[a * 3 + 1]);
  ctx.lineTo(tv[b * 3], tv[b * 3 + 1]);
  ctx.stroke();
}
```

Edge source

- explicit edges, or
- derived uniquely from triangle mesh using wireEdges().

2.8.2 Flat Shading

Triangles are sorted by average depth:

$$d_i = (z_a + z_b + z_c) / 3$$

Then rendered from far to near.

Face normal

For triangle with points A,B,C:

$$\mathbf{u} = \mathbf{B} - \mathbf{A}, \mathbf{w} = \mathbf{C} - \mathbf{A}$$

$$\mathbf{n} = \mathbf{u} \times \mathbf{w}$$

Expanded:

$$n_x = u_y w_z - u_z w_y$$

$$n_y = u_z w_x - u_x w_z$$

$$n_z = u_x w_y - u_y w_x$$

Normalize:

$$\hat{\mathbf{n}} = \mathbf{n} / \|\mathbf{n}\|$$

View-space normal depth component

The code computes:

$$r_z = n_x A_{zx} + n_y A_{zy} + n_z A_{zz}$$

where A is the current orientation matrix.

Lambert-like intensity

$$\lambda = \min(1, 0.25 + 0.75 |r_z|)$$

So brightness increases when the face is more front-facing or back-facing in view Z .

Color application

If vertex colors exist, triangle color is the average of its vertices:

$$c_r = (r_1 + r_2 + r_3) / 3, c_g = (g_1 + g_2 + g_3) / 3, c_b = (b_1 + b_2 + b_3) / 3$$

Then shaded color:

$$(c'_r, c'_g, c'_b) = \lambda (c_r, c_g, c_b)$$

2.8.3 Point Rendering

Points are sorted by depth and rendered as small squares.

For each point, depth-based fog:

$$fog = 1 - 0.55 \cdot d / 15$$

with clamping of $d \in [0, 15]$.

Final color mixes the original point color with a light gray background (235):

$$c' = c \cdot fog + 235 \cdot (1 - fog)$$

This simulates distance fading.

2.8.4 Ball-and-Stick Rendering

Bonds

Each bond is drawn as a thick rounded gray line. Bond color varies with average depth.

Atoms

Atoms are drawn back-to-front as circles with radial gradients.

Radius formula

js

```
const rad=Math.max(2, state.ballSize*0.5*(0.5+elemRad(sym)) * (state.xfac/60+0.4));
```

Approximate formula:

$$r=\max(2, ballSize(0.5+\rho_e)(xfac/60+0.4)/2)$$

where ρ_e is the covalent radius for the element.

Fog blending

For depth $d \in [0,15]$:

$$fog=0.45 \cdot d/15$$

Final atom color:

$$c'=c(1-fog)+215fog$$

Radial gradient

- center highlight = white
- mid color = blended element color
- edge = darkened version (~32%)

This creates a pseudo-3D sphere appearance.

2.9 Chemistry-Specific Algorithms**2.9.1 Element normalization**

Input element strings are cleaned:

- non-letters removed
- lowercase applied
- limited to one or two letters
- fallback to carbon if unknown

2.9.2 Color mapping

Uses a simplified **CPK color scheme**.

Examples:

- H → white
- C → dark gray
- N → blue
- O → red
- S → yellow
- Cl → green

2.9.3 Covalent radii

Used for bond perception and sphere size.

2.9.4 Bond perception algorithm

If explicit bonds are missing, the tool computes them by distance.

Basic rule

For atoms i and j , define:

$$cut_{ij} = r_i + r_j + tol$$

where:

- r_i, r_j are covalent radii
- tol is user bond tolerance.

A bond exists if:

$$0 < \|x_j - x_i\|^2 < cut_{ij}^2$$

2.9.5 Spatial hash acceleration

Naive all-pairs search is $O(n^2)$.

This code uses a **uniform spatial hash**.

Cell size

$$cell = \max(10^{-6}, 2r_{\max} + tol)$$

Each atom is assigned to a cell:

$$c_x = \lfloor x/cell \rfloor, c_y = \lfloor y/cell \rfloor, c_z = \lfloor z/cell \rfloor$$

Only neighboring $3 \times 3 \times 3$ cells are checked.

Complexity

This often reduces practical cost close to linear for spatially distributed molecules.

Representative code:

```
js
function perceiveBonds(verts, atoms, tol){
  ...
  for(let i=0; i<n; i++){
    const ci=Math.floor(verts[i*3]/cell), cj=Math.floor(verts[i*3+1]/cell), ck=Math.floor(verts[i*3+2]/cell);
    ...
  }
  ...
}
```

2.10 Volumetric Data Algorithm**2.10.1 MRC/CCP4 parsing**

The code reads the 1024-byte header and determines:

- dimensions nx, ny, nz
- mode
- cell lengths
- origin
- voxel spacing

- optional symmetry block length nsymbt

Supported voxel types:

- mode 0: int8
- mode 1: int16
- mode 2: float32
- mode 6: uint16

2.10.2 Iso-threshold point cloud extraction

Instead of full isosurface extraction (e.g., Marching Cubes), the tool creates a point cloud of voxels above threshold.

Automatic threshold

If no iso level is supplied:

$$iso = \mu + 2\sigma$$

where:

$$\mu = \sum_{i=1}^N v_i / N$$

$$\sigma = \sqrt{\sum_{i=1}^N (v_i - \mu)^2 / N}$$

Selection rule

A voxel is included if:

$$v_i \geq iso$$

Downsampling

If number of selected voxels is count, and user maximum is maxPts:

$$stride = \max(1, \lceil count / maxPts \rceil)$$

Only every stride-th selected voxel is emitted.

Coordinate mapping

For voxel indices (i, j, k) :

$$x = o_x + id_x, y = o_y + jd_y, z = o_z + kd_z$$

Color mapping by scalar intensity

Let:

$$t = \text{clamp}((v - iso) / (hi - lo), 0, 1)$$

with:

- $lo = iso$
- $hi = \max(iso + \epsilon, \mu + 4\sigma)$

Then RGB is set as:

$$R=40+215t$$

$$G=120+80(1-t)$$

$$B=230-160t$$

This creates a blue-to-red style gradient.

Representative code:

```
js
const iso = Number.isFinite(opt.iso) ? opt.iso : mean+2*sd;
...
V.push(vol.ox+i*vol.dx, vol.oy+j*vol.dy, vol.oz+k*vol.dz);
```

2.11 CAD Exchange Point Sampling

2.11.1 STEP

The parser extracts CARTESIAN_POINT(...) entities only.

So it does not tessellate B-rep geometry.

It simply produces a point set from discrete point coordinates.

Regex used:

```
js
const re=/CARTESIAN_POINT\s*(\s*'[^']*'\s*,\s*(((\^[^)]*)))/gi;
```

2.11.2 IGES

The parser extracts coordinates from a few point-bearing entity types:

- 116: POINT
- 110: LINE endpoints
- 126: rational B-spline curve control points
- 128: rational B-spline surface control net

Again, this is **discrete sampling**, not full surface reconstruction.

2.12 File Parsing Strategy by Format

This section summarizes the main logic.

2.12.1 OBJ

- parse v
- parse f
- parse l
- polygon faces triangulated by fan

For polygon indices $[i_0, i_1, \dots, i_n]$, triangulation is:

$(i_0, i_1, i_2), (i_0, i_2, i_3), \dots$

Representative code:

```
js
function fanTriangles(idx, out){
  for(let i=1;i+1<idx.length;i++){ out.push(idx[0],idx[i],idx[i+1]); }
}
```

2.12.2 STL

Supports:

- ASCII STL by regex over vertex

- binary STL by fixed 50-byte face records

Each binary face:

- 12 bytes normal
- 36 bytes vertices
- 2 bytes attribute count

The code ignores stored face normals and recomputes at render time.

2.12.3 PLY

Supports:

- ASCII
- binary little-endian / big-endian

Reads header:

- format
- elements
- properties
- list/scalar declarations

Can read:

- vertex coordinates
- per-vertex RGB
- face indices

If no faces exist, model kind becomes points.

2.12.4 OFF

Reads:

- counts line
- vertices
- polygon faces
- fan triangulation

2.12.5 VTK legacy ASCII

Supports only legacy ASCII datasets, mainly POLYDATA-like point and primitive sections:

- POINTS
- POLYGONS
- TRIANGLE_STRIP
- LINES
- VERTICES

No binary VTK support.

2.12.6 3DS

Walks nested chunk structure recursively.

Relevant chunks:

- 0x4110: vertex list
- 0x4120: face list

2.12.7 glTF / GLB

Parses:

- JSON
- buffer views
- accessors

- scene graph transforms
- mesh primitives in TRIANGLES mode

Transform handling

Node transforms are accumulated through scene traversal.

If node uses TRS:

- translation
- quaternion rotation
- scale

the code converts TRS to a 4×4 matrix.

2.12.8 3MF

Reads ZIP container, then XML .model part.

Extracts:

- vertices
- triangles
- build items
- optional item transform

2.12.9 DAE / COLLADA

Parses XML:

- <source> arrays
- <vertices> POSITION indirection
- <triangles>, <polylist>, <polygons>

Supports unit scaling from <unit meter="...">.

2.12.10 X3D

Parses:

- IndexedFaceSet
- IndexedTriangleSet
- IndexedLineSet
- PointSet

2.12.11 DXF

ASCII only.

Recognized entity types:

- 3DFACE
- LINE
- POINT
- LWPOLYLINE
- POLYLINE + VERTEX + SEQEND

2.12.12 FBX

Supports:

- binary FBX
- ASCII FBX

Extracts mainly:

- Vertices
- PolygonVertexIndex

This is a partial FBX parser, focused on geometry.

2.12.13 XYZ

Reads atom label + x y z.

If a first line is atom count, it skips the comment line.

2.12.14 PDB

Reads:

- ATOM
- HETATM
- optional CONECT

Uses only the first model before ENDMDL.

2.12.15 MOL/SDF

Supports V2000 only, not V3000.

Reads:

- counts line
- atom block
- bond block

2.12.16 MOL2

Reads @<TRIPOS>ATOM and @<TRIPOS>BOND.

2.12.17 CML

Parses XML atoms and bonds.

Supports 3D and fallback 2D coordinates.

2.12.18 CUBE

Reads:

- atom list
- volumetric grid
- optional scalar field
- atom positions converted from Bohr to Å if needed

2.13 ZIP Support

The code contains a mini ZIP subsystem.

2.13.1 Reading ZIP

Used for:

- 3MF import

Supports:

- method 0 = stored
- method 8 = deflate

Uses browser DecompressionStream("deflate-raw").

2.13.2 Writing ZIP

Used for:

- 3MF export

Writes STORED entries only.

2.13.3 CRC32

A standard lookup-table-based CRC32 is implemented.

2.14 Export Algorithms

2.14.1 OBJ writer

js

```
function writeOBJ(m){
    ...
    L.push("v ...");
    L.push("f ...");
    L.push("l ...");
}
```

Outputs:

- vertices
- triangle faces
- line entities if no triangles

2.14.2 STL writer

Two versions:

- ASCII
- binary

Face normals are recomputed using cross products.

2.14.3 PLY writer

ASCII only.

Outputs:

- vertex positions
- optional vertex colors
- faces

2.14.4 OFF writer

Simple vertices + triangular faces.

2.14.5 VTK writer

Legacy ASCII POLYDATA.

Depending on available topology:

- POLYGONS
- LINES
- or VERTICES

2.14.6 3MF writer

Builds:

- [Content_Types].xml
- _rels/.rels
- 3D/3dmodel.model

and packages them into ZIP.

2.14.7 GLB writer

Exports a minimal glTF 2.0 binary package with:

- one scene
- one node
- one mesh
- one POSITION accessor
- one indices accessor

Uses:

- Float32Array for positions

- Uint32Array for indices

2.14.8 Molecular writers

Available:

- XYZ
- PDB
- MOL
- SDF
- MOL2
- CML

These use atom list if present; otherwise generic element placeholders.

2.15 User Interface Guide

2.15.1 Loading a file

1. Open the HTML page in a browser.
2. Click **Structure data file**
3. Select a supported file.
4. The page automatically detects its category.
5. Set visualization parameters if desired.
6. Click **OK**

2.15.2 Parameter settings

General

- **Zoom scale:** initial object scale
- **Dilation rate:** zoom in/out multiplier
- **Max scroll:** pan slider range
- **Canvas width / height:** viewport size

Molecular

- **Ball size:** atom sphere size basis
- **Bond tolerance:** distance tolerance for inferred bonds

Volumetric / point-cloud

- **Iso level:** threshold; blank means $\mu+2\sigma$
- **Max sampled points:** point cloud cap

2.15.3 View controls

- **Drag mouse** to rotate
- **Left click without dragging** to zoom in
- **Right click without dragging** to zoom out
- **Mouse wheel** to zoom
- **Scrollbars** to pan horizontally/vertically

2.15.4 Display style

Use the **Display** dropdown:

- Auto
- Wireframe
- Flat-shaded
- Points
- Ball & stick

2.15.5 Conversion

1. Choose output format in **Convert to**
2. Click **Convert & Save**
3. Download starts automatically

Note:

- some formats require triangles,
- some are meaningful only for molecular data.

2.15.6 Save image

Click **Save as PNG** to export the current canvas view.

2.15.7 Reset / back

- **Reset view** restores default orientation and zoom
- **Close / Back** returns to settings panel

2.15.8 Self-test

Click **Run Self-Test** to execute built-in parser and conversion checks.

The test output appears in a console-like box.

2.16 Representative JavaScript Code Snippets

2.16.1 File extension dispatch

js

```
async function parseAny(name, buf, opt){
  const e=ext(name);
  let m;
  switch(e){
    case "obj":  m=parseOBJ(buf); break;
    case "stl":  m=parseSTL(buf); break;
    case "ply":  m=parsePLY(buf); break;
    ...
    default: fail("Unsupported extension '"+e+"'");
  }
  ...
  return m;
}
```

This is the central import router.

2.16.2 Transformation of vertices

js

```
transform(v,n){
  const out=new Float32Array(n*3);
  const {xx,xy,xz,xo,yx,yy,yz,yo,zx,zy,zz,zo}=this;
  for(let i=0;i<n;i++){
    const j=i*3, a=v[j], b=v[j+1], c=v[j+2];
    out[j]=a*xx+b*xy+c*xz+xo;
    out[j+1]=a*yx+b*yy+c*yz+yo;
    out[j+2]=a*zx+b*zy+c*zz+zo;
  }
  return out;
}
```

This is the geometric core of rendering.

2.16.3 Fan triangulation

```
js
function fanTriangles(idx, out){
  for(let i=1;i+1<idx.length;i++){ out.push(idx[0],idx[i],idx[i+1]); }
}
```

Used across OBJ, OFF, PLY, DAE, X3D, FBX, etc.

2.16.4 Bond perception

```
js
const cut=elemRad(atoms[i])+elemRad(atoms[j])+tol;
if(d2>1e-8 && d2<cut*cut) bonds.push([i,j]);
```

Chemical adjacency is inferred from geometric distance.

2.16.5 Flat shading

```
js
const ux=V[i1*3]-V[i0*3], uy=V[i1*3+1]-V[i0*3+1], uz=V[i1*3+2]-V[i0*3+2];
const wx=V[i2*3]-V[i0*3], wy=V[i2*3+1]-V[i0*3+1], wz=V[i2*3+2]-V[i0*3+2];
let nx=uy*wz-uz*wy, ny=uz*wx-ux*wz, nz=ux*wy-uy*wx;
```

Cross-product normal calculation for triangle lighting.

2.16.6 Auto iso threshold

```
js
const iso = Number.isFinite(opt.iso) ? opt.iso : mean+2*sd;
```

This makes volumetric preview usable without manual tuning.

2.17 Self-Test System

The self-test is a major strength of the code.

2.17.1 What it does

It generates internal sample files and then:

1. **Parses each sample**
2. **Checks expected counts**
3. **Converts parsed models to all valid export formats**
4. **Re-parses converted outputs**
5. **Compares bounding boxes and counts**
6. **Runs negative tests for invalid files**

2.17.2 Round-trip criteria

For each round-trip:

- relative bounding-box drift must be small:

$$rel = \Delta_{bbox} / scale$$

with failure if:

$$rel > 10^{-4}$$

- triangle count must survive mesh-preserving formats
- atom count and element sequence must survive molecular formats

2.17.3 Why this matters

This substantially improves trustworthiness:

- parser bugs are easier to catch,
- export-import consistency can be monitored,
- regressions can be detected manually.

2.18 Strengths of the Implementation

2.18.1 Fully self-contained

No external JS libraries, no CSS framework, no server.

2.18.2 Very broad format support

It supports a remarkable set of geometry, chemistry, and volumetric formats in one page.

2.18.3 Robust error messages

Many parsers use explicit validation and descriptive fail(...) messages.

2.18.4 Good fallback strategy

Examples:

- missing molecular bonds → infer them
- missing iso level → compute from mean + 2σ
- no mesh faces → display as points or lines
- unsupported glTF external buffers → clear message

2.18.5 Practical conversion pipeline

The tool is not just a viewer; it also exports to many useful interchange formats.

2.18.6 Useful for education

The code demonstrates:

- affine transforms,
- parsing text and binary data,
- XML/ZIP handling,
- software 3D rendering,
- scientific data processing.

2.19 Limitations and Boundaries

The code is powerful, but intentionally simplified in some areas.

2.19.1 Rendering limitations

- no perspective projection; display is effectively orthographic / pseudo-depth
- no hidden-line removal in wireframe
- no z-buffer
- flat shading only
- no textures
- no transparency compositing beyond simple overdraw order

2.19.2 Parser limitations

- VTK: ASCII legacy only
- DXF: ASCII only
- MOL/SDF: V2000 only
- STEP/IGES: point sampling only, no true tessellation
- glTF: embedded/data-URI/GLB buffers only, no external linked files
- FBX: geometry-focused partial support
- no support for complex material/scene hierarchies in many formats

2.19.3 Performance limitations

- all rendering is CPU-based on 2D canvas
- very large models may render slowly
- sorting triangles or points can become expensive
- JavaScript memory limits may matter for huge volumetric maps

2.19.4 Conversion semantics

Not all conversions preserve all information.

Examples:

- mesh $\rightarrow$ XYZ loses topology semantics
- molecular $\rightarrow$ OBJ loses chemistry semantics except as geometry/lines
- volumetric $\rightarrow$ point cloud is not the original scalar field

2.20 Typical Use Cases

2.20.1 Mesh preview and conversion

Load .stl, preview as flat-shaded, export as .obj or .glb.

2.20.2 Molecular quick look

Load .pdb or .xyz, inspect atom types and bonds, export to .mol2 or .cml.

2.20.3 Density map thresholding

Load .mrc, let the program compute mean+2 σ threshold, inspect high-density voxels as points.

2.20.4 CAD data sampling

Load .step or .iges and get a quick point-based spatial overview.

2.20.5 Format sanity checking

Use the parser and self-test behavior to see whether a file contains recognizable coordinates.

2.21 Internal Workflow Summary

When a user loads a file, the sequence is:

1. Read file extension
2. Determine category
3. Read user options
4. Load file as ArrayBuffer
5. Dispatch to parser
6. Validate produced coordinates and indices
7. Store Model
8. Compute default view
9. Render with chosen style
10. Allow export to formats compatible with model contents

2.22 Conversion Logic Summary

The conversion targets depend on model properties.

2.22.1 If triangles exist

Allowed mesh-like exports:

- obj
- stl
- ply
- off
- vtk
- 3mf

- glb
- x3d

2.22.2 If no triangles

Allowed:

- obj
- ply
- vtk
- x3d

2.22.3 If atoms exist

Allowed chemistry exports:

- xyz
- pdb
- mol
- sdf
- mol2
- cml

2.22.4 Otherwise

Fallback exports:

- xyz
- pdb

2.23 Practical Notes for Users

Recommended browsers

A modern Chromium/Firefox/Safari-class browser is needed.

Especially important:

- DecompressionStream for ZIP/deflate support

File size caution

For very large files:

- increase browser memory allowance if possible,
- reduce volumetric Max sampled points,
- prefer point or wireframe display over dense flat-shading.

For molecular files without bonds

Adjust **Bond tolerance** if too many or too few bonds appear.

For volumetric maps

If no points appear, lower **Iso level** manually.

2.24 Practical Notes for Developers

If extending the program, the easiest areas are:

- adding new parser functions,
- adding new export writers,
- improving rendering styles,
- adding perspective projection,
- adding surface extraction for volumes,
- improving CAD tessellation.

A typical extension pattern is:

1. write parseFORMAT(buf,opt)

2. add extension to category arrays
3. add dispatch case in parseAny
4. optionally add writer and conversion target

2.25 Conclusion

The tool implements a surprisingly comprehensive universal 3D structure viewer and data converter as a single browser page.

Its major achievements are:

- broad support for geometry, molecular, and volumetric formats,
- no dependency on external libraries,
- custom affine transform and software renderer,
- many import/export pathways,
- scientific features such as bond perception and iso-thresholding,
- built-in self-testing.

From a software design perspective, it is a compact but capable example of:

- client-side scientific visualization,
- geometry parsing and transformation,
- format interoperability,
- and Canvas-based 3D rendering.

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\(3-4\)/3D-Visualizer2.0.htm](http://www.iaees.org/publications/journals/Selforganizology/articles/2028-15(3-4)/3D-Visualizer2.0.htm); Fig. 1):

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>3D-Visualizer 2.0: Universal 3D Structure Visualizer and Data Converter</title>
<style>
  body{font-family:"Segoe UI",Arial,sans-serif;background:#f2f2f2;margin:0;padding:20px;color:#222}
  h2{margin-top:0}
  #settingsPanel{max-width:640px;background:#fff;border:1px solid #ccc;border-radius:6px;padding:20px}
  table{border-collapse:collapse;width:100%}
  td{padding:5px 8px;vertical-align:middle}
  td.lbl{text-align:right;white-space:nowrap;width:48%}
  input[type=number],input[type=text],select{width:150px}
  .hint{font-size:12px;color:#666;margin-top:4px}
  #fileTypeMsg{margin-top:6px;font-weight:bold}
  button{padding:6px 14px;margin-right:8px;cursor:pointer}
  #viewerPanel{margin-top:20px;display:none}
  #stage{display:grid;grid-template-columns:auto 24px;grid-template-rows:auto 24px;gap:4px;width:max-content}
  #canvasWrap{border:1px solid #888;background:#fff}
  canvas{display:block;cursor:grab}
  canvas:active{cursor:grabbing}
```

```

#vScroll{-webkit-appearance:slider-vertical;writing-mode:vertical-lr;direction:rtl;width:22px;height:100%}
#hScroll{width:100%}
.legend{margin-top:10px;font-size:13px}
.legend span{display:inline-block;width:12px;height:12px;border:1px solid #999;margin-right:4px;vertical-align:middle}
.toolbar{margin-top:10px}
.note{font-size:12px;color:#666}
#status{margin-top:8px;font-size:12px;color:#444;white-space:pre-wrap}
#testOut{margin-top:10px;background:#111;color:#ddd;padding:10px;border-radius:4px;
font:11px/1.45 Consolas,monospace;max-height:340px;overflow:auto;display:none;white-space:pre}
fieldset{border:1px solid #ddd;margin:12px 0;padding:8px 12px}
legend{font-weight:bold;font-size:12px;color:#555}
</style>
</head>
<body>

<h2>3D-Visualizer 2.0: Universal 3D Structure Visualizer and Data Converter</h2>
<a href="http://www.iaees.org/publications/journals/Selforganizology/articles/2028-15(3-4)/2-Zhang-Abstract.asp" style="color:
brown; font-size: 12px ">Zhang WJ. 2028. 3D-Visualizer 2.0: A universal 3D structure visualizer and data converter.
Selforganizology, 15(3-4): 113-195. </a><br>

<p class="note">
The renderer is chosen automatically from the file extension.<br>
<b>Mesh / solid</b> (.obj .stl .ply .off .3ds .fbx .gltf .glb .dae .x3d .3mf .dxf .vtk) &rarr; wireframe / flat-shaded surface<br>
<b>Molecular / chemical</b> (.xyz .pdb .mol .sdf .mol2 .cml .cube) &rarr; ball-and-stick model<br>
<b>Volumetric & CAD exchange</b> (.mrc .ccp4 .map .step .stp .igs .iges) &rarr; point cloud (iso-level / discrete
sampling)
</p>

<div id="settingsPanel">
  <table>
    <tr>
      <td class="lbl">Structure data file:</td>
      <td><input type="file" id="fileInput"
accept=".obj,.stl,.ply,.off,.3ds,.fbx,.gltf,.glb,.dae,.x3d,.3mf,.dxf,.vtk,.xyz,.pdb,.mol,.sdf,.mol2,.cml,.cube,.mrc,.ccp4,.map,.step,.st
p,.igs,.iges"></td>
    </tr>
    <tr><td colspan="2"><div id="fileTypeMsg"></div></td></tr>
    <tr><td class="lbl">Zoom scale (e.g. 2, 5):</td><td><input type="number" id="scaleInput" value="2"
step="0.1"></td></tr>
    <tr><td class="lbl">Dilation rate (0~1):</td><td><input type="number" id="dilaInput" value="0.3" step="0.05" min="0"
max="0.99"></td></tr>
    <tr><td class="lbl">Max scroll:</td><td><input type="number" id="scrollMaxInput" value="2000"></td></tr>
    <tr><td class="lbl">Canvas width:</td><td><input type="number" id="canvasWInput" value="900"></td></tr>
    <tr><td class="lbl">Canvas height:</td><td><input type="number" id="canvasHInput" value="640"></td></tr>
  </table>

```

```

<fieldset><legend>Molecular options</legend>
  <table>
    <tr><td class="lbl">Ball size (px at unit zoom):</td><td><input type="number" id="ballSizeInput"
value="10"></td></tr>
    <tr><td class="lbl">Bond tolerance (&Aring;):</td><td><input type="number" id="bondTollInput" value="0.45"
step="0.05"></td></tr>
  </table>
</fieldset>
<fieldset><legend>Volumetric / point-cloud options</legend>
  <table>
    <tr><td class="lbl">Iso level (blank = mean+2&sigma;):</td><td><input type="text" id="isoInput"
placeholder="auto"></td></tr>
    <tr><td class="lbl">Max sampled points:</td><td><input type="number" id="maxPtsInput" value="120000"></td></tr>
  </table>
</fieldset>
<div class="toolbar">
  <button id="loadBtn">OK</button>
  <button id="selfTestBtn">Run Self-Test</button>
</div>
<div id="status"></div>
<div id="testOut"></div>
</div>

<div id="viewerPanel">
  <div id="stage">
    <div id="canvasWrap"><canvas id="glcanvas"></canvas></div>
    <input type="range" id="vScroll" orient="vertical">
    <input type="range" id="hScroll">
    <div></div>
  </div>

  <div class="toolbar">
    <label>Display:
    <select id="styleSel">
      <option value="auto">Auto</option>
      <option value="wire">Wireframe</option>
      <option value="flat">Flat-shaded</option>
      <option value="points">Points</option>
      <option value="ballstick">Ball & stick</option>
    </select>
    </label>
    &nbsp;
    <label>Convert to:
    <select id="convSel"></select>
    </label>
  </div>

```

```

    <button id="convBtn">Convert & Save</button>
</div>

<div class="toolbar">
    <button id="saveBtn">Save as PNG</button>
    <button id="resetBtn">Reset view</button>
    <button id="closeBtn">Close / Back</button>
</div>
<div class="legend" id="legend"></div>
<div id="info" class="note"></div>
<p class="note">Drag = rotate; left-click without dragging = zoom in; right-click = zoom out;
    wheel = zoom; the scrollbars beside the canvas pan the view.</p>
</div>

<script>
"use strict";
/* =====
    0. Small utilities
    ===== */
const TD = new TextDecoder("utf-8");
const asText = buf => TD.decode(buf instanceof Uint8Array ? buf : new Uint8Array(buf));
const ext = name => (name.toLowerCase().match(/\.[a-z0-9+]+$/ || [,""])[1]);
const num = s => { const v = parseFloat(s); return Number.isFinite(v) ? v : NaN; };
function fail(msg){ throw new Error(msg); }

/* =====
    1. Mat3D (rotation/scale/translate stack, after Zhang 2023)
    ===== */
class Mat3D {
    constructor(){ this.unit(); }
    unit(){
        this.xo=0; this.xx=1; this.xy=0; this.xz=0;
        this.yo=0; this.yx=0; this.yy=1; this.yz=0;
        this.zo=0; this.zx=0; this.zy=0; this.zz=1;
    }
    scale(f,f1,f2){
        if(f1===undefined){ f1=f; f2=f; }
        this.xx*=f; this.xy*=f; this.xz*=f; this.xo*=f;
        this.yx*=f1; this.yy*=f1; this.yz*=f1; this.yo*=f1;
        this.zx*=f2; this.zy*=f2; this.zz*=f2; this.zo*=f2;
    }
    translate(f,f1,f2){ this.xo+=f; this.yo+=f1; this.zo+=f2; }
    xrot(deg){
        const d=deg*Math.PI/180, c=Math.cos(d), s=Math.sin(d);
        const yx=this.yx*c+this.zx*s, yy=this.yy*c+this.zy*s, yz=this.yz*c+this.zz*s, yo=this.yo*c+this.zo*s;

```

```

    const zx=this.zx*c-this.yx*s, zy=this.zy*c-this.yy*s, zz=this.zz*c-this.yz*s, zo=this.zo*c-this.yo*s;
    this.yx=yx; this.yy=yy; this.yz=yz; this.yo=yo;
    this.zx=zx; this.zy=zy; this.zz=zz; this.zo=zo;
  }
  yrot(deg){
    const d=deg*Math.PI/180, c=Math.cos(d), s=Math.sin(d);
    const xx=this.xx*c+this.zx*s, xy=this.xy*c+this.zy*s, xz=this.xz*c+this.zz*s, xo=this.xo*c+this.zo*s;
    const zx=this.zx*c-this.xx*s, zy=this.zy*c-this.xy*s, zz=this.zz*c-this.xz*s, zo=this.zo*c-this.xo*s;
    this.xx=xx; this.xy=xy; this.xz=xz; this.xo=xo;
    this.zx=zx; this.zy=zy; this.zz=zz; this.zo=zo;
  }
  mult(m){
    const f  = this.xx*m.xx+this.yx*m.xy+this.zx*m.xz;
    const f1 = this.xy*m.xx+this.yy*m.xy+this.zy*m.xz;
    const f2 = this.xz*m.xx+this.yz*m.xy+this.zz*m.xz;
    const f3 = this.xo*m.xx+this.yo*m.xy+this.zo*m.xz+m.xo;
    const f4 = this.xx*m.yx+this.yx*m.yy+this.zx*m.yz;
    const f5 = this.xy*m.yx+this.yy*m.yy+this.zy*m.yz;
    const f6 = this.xz*m.yx+this.yz*m.yy+this.zz*m.yz;
    const f7 = this.xo*m.yx+this.yo*m.yy+this.zo*m.yz+m.yo;
    const f8 = this.xx*m.zx+this.yx*m.zy+this.zx*m.zz;
    const f9 = this.xy*m.zx+this.yy*m.zy+this.zy*m.zz;
    const f10= this.xz*m.zx+this.yz*m.zy+this.zz*m.zz;
    const f11= this.xo*m.zx+this.yo*m.zy+this.zo*m.zz+m.zo;
    this.xx=f;  this.xy=f1; this.xz=f2;  this.xo=f3;
    this.yx=f4; this.yy=f5; this.yz=f6;  this.yo=f7;
    this.zx=f8; this.zy=f9; this.zz=f10; this.zo=f11;
  }
  transform(v,n){
    const out=new Float32Array(n*3);
    const {xx,xy,xz,xo,yx,yy,yz,yo,zx,zy,zz,zo}=this;
    for(let i=0;i<n;i++){
      const j=i*3, a=v[j], b=v[j+1], c=v[j+2];
      out[j]=a*xx+b*xy+c*xz+xo;
      out[j+1]=a*yx+b*yy+c*yz+yo;
      out[j+2]=a*zx+b*zy+c*zz+zo;
    }
    return out;
  }
}

/* =====
2. Model container
kind: 'mesh' | 'mol' | 'points'
verts : Float32Array (3N)

```

```

    tris : Int32Array (3T)  – triangles, mesh only
    edges : Array<[a,b]>    – explicit line entities
    atoms : Array<string>   – element symbols, molecular only
    bonds : Array<[a,b]>
    colors: Uint8Array (3N) – optional per-vertex colour
    volume: {data,nx,ny,nz,ox,oy,oz,dx,dy,dz}

    ===== */
class Model {
  constructor(kind){
    this.kind=kind; this.verts=null; this.tris=null; this.edges=null;
    this.atoms=null; this.bonds=null; this.colors=null; this.volume=null;
    this.source="";
  }
  get nVerts(){ return this.verts ? this.verts.length/3 : 0; }
  get nTris(){ return this.tris ? this.tris.length/3 : 0; }
  bbox(){
    const v=this.verts;
    if(!v || !v.length) return {xmin:0,xmax:1,ymin:0,ymax:1,zmin:0,zmax:1};
    let a=v[0],b=v[0],c=v[1],d=v[1],e=v[2],f=v[2];
    for(let i=3;i<v.length;i+=3){
      const x=v[i],y=v[i+1],z=v[i+2];
      if(x<a)a=x; if(x>b)b=x;
      if(y<c)c=y; if(y>d)d=y;
      if(z<e)e=z; if(z>f)f=z;
    }
    return {xmin:a,xmax:b,ymin:c,ymax:d,zmin:e,zmax:f};
  }
  /* unique undirected edges of the triangle set, for wireframe display */
  wireEdges(limit=400000){
    if(this.edges && this.edges.length && !this.tris) return this.edges;
    const out=[], seen=new Set();
    const push=(a,b)=>{
      if(a===b) return;
      const k = a<b ? a*4294967296+b : b*4294967296+a;
      if(!seen.has(k)){ seen.add(k); out.push([a,b]); }
    };
    if(this.tris) for(let i=0;i<this.tris.length && out.length<limit;i+=3){
      push(this.tris[i],this.tris[i+1]);
      push(this.tris[i+1],this.tris[i+2]);
      push(this.tris[i+2],this.tris[i]);
    }
    if(this.edges) for(const e of this.edges) push(e[0],e[1]);
    return out;
  }
}

```

```

/* fan-triangulate a polygon given as an index array */
function fanTriangles(idx, out){
  for(let i=1;i+1<idx.length;i++){ out.push(idx[0],idx[i],idx[i+1]); }
}

/* =====
3. Chemistry tables
===== */

const CPK = {
  h:[255,255,255], he:[217,255,255], li:[204,128,255], be:[194,255,0],
  b:[255,181,181], c:[80,80,80], n:[48,80,248], o:[255,13,13],
  f:[144,224,80], ne:[179,227,245], na:[171,92,242], mg:[138,255,0],
  al:[191,166,166], si:[240,200,160], p:[255,128,0], s:[255,255,48],
  cl:[31,240,31], ar:[128,209,227], k:[143,64,212], ca:[61,255,0],
  fe:[224,102,51], cu:[200,128,51], zn:[125,128,176], br:[166,41,41],
  i:[148,0,148], au:[255,209,35], hg:[184,184,208],
  default:[255,100,200]
};

const COVR = {
  h:0.31, he:0.28, li:1.28, be:0.96, b:0.84, c:0.76, n:0.71, o:0.66,
  f:0.57, ne:0.58, na:1.66, mg:1.41, al:1.21, si:1.11, p:1.07, s:1.05,
  cl:1.02, ar:1.06, k:2.03, ca:1.76, fe:1.32, cu:1.32, zn:1.22,
  br:1.20, i:1.39, au:1.36, hg:1.32, default:1.20
};

const elemColor = s => CPK[s] || CPK.default;
const elemRad = s => COVR[s] || COVR.default;

function normElement(sym){
  if(!sym) return "c";
  let s = String(sym).trim().replace(/^[A-Za-z]/g,"");
  if(!s) return "c";
  s = s.toLowerCase();
  if(s.length>2) s=s.slice(0,2);
  if(s.length===2 && !(s in COVR)) s=s[0];
  return s;
}

/* distance-based bond perception with a uniform spatial hash */
function perceiveBonds(verts, atoms, tol){
  const n = atoms.length, bonds=[];
  if(n<2) return bonds;
  let maxR=0;
  for(const a of atoms) maxR=Math.max(maxR, elemRad(a));
  const cell = Math.max(1e-6, 2*maxR + tol);

```

```

const map = new Map();
const key=(i,j,k)=>i+","+j+","+k;
for(let i=0;i<n;i++){
  const ci=Math.floor(verts[i*3]/cell), cj=Math.floor(verts[i*3+1]/cell), ck=Math.floor(verts[i*3+2]/cell);
  const k=key(ci,cj,ck);
  let arr=map.get(k); if(!arr){ arr=[]; map.set(k,arr); }
  arr.push(i);
}
for(let i=0;i<n;i++){
  const x=verts[i*3], y=verts[i*3+1], z=verts[i*3+2];
  const ci=Math.floor(x/cell), cj=Math.floor(y/cell), ck=Math.floor(z/cell);
  for(let a=-1;a<=1;a++)for(let b=-1;b<=1;b++)for(let c=-1;c<=1;c++){
    const arr=map.get(key(ci+a,cj+b,ck+c));
    if(!arr) continue;
    for(const j of arr){
      if(j<=i) continue;
      const dx=verts[j*3]-x, dy=verts[j*3+1]-y, dz=verts[j*3+2]-z;
      const d2=dx*dx+dy*dy+dz*dz;
      const cut=elemRad(atoms[i])+elemRad(atoms[j])+tol;
      if(d2>1e-8 && d2<cut*cut) bonds.push([i,j]);
    }
  }
}
return bonds;
}

/* =====
4. ZIP (read: stored + deflate; write: stored)
===== */

const CRC_TABLE = (()=>{
  const t=new Uint32Array(256);
  for(let i=0;i<256;i++){ let c=i; for(let k=0;k<8;k++) c = c&1 ? 0xEDB88320^(c>>>1) : c>>>1; t[i]=c>>>0; }
  return t;
})();

function crc32(u8){
  let c=0xFFFFFFFF;
  for(let i=0;i<u8.length;i++) c = CRC_TABLE[(c ^ u8[i]) & 0xFF] ^ (c>>>8);
  return (c ^ 0xFFFFFFFF)>>>0;
}

async function inflateRaw(u8){
  if(typeof DecompressionStream === "undefined")
    fail("This browser cannot decompress deflate data (DecompressionStream missing).");
  const ds=new DecompressionStream("deflate-raw");
  const w=ds.writable.getWriter(); w.write(u8); w.close();
  const chunks=[]; const r=ds.readable.getReader();

```



```

for(;;){ const {done,value}=await r.read(); if(done) break; chunks.push(value); }
let len=0; for(const c of chunks) len+=c.length;
const out=new Uint8Array(len); let o=0;
for(const c of chunks){ out.set(c,o); o+=c.length; }
return out;
}
async function zipRead(buf){
  const u8=new Uint8Array(buf), dv=new DataView(buf);
  let eocd=-1;
  for(let i=u8.length-22;i>=0 && i>u8.length-66000;i--){
    if(dv.getUint32(i,true)==0x06054b50){ eocd=i; break; }
  }
  if(eocd<0) fail("Not a ZIP container (end-of-central-directory not found).");
  const count=dv.getUint16(eocd+10,true);
  let p=dv.getUint32(eocd+16,true);
  const files={};
  for(let i=0;i<count;i++){
    if(dv.getUint32(p,true)!=0x02014b50) fail("Corrupt ZIP central directory.");
    const method=dv.getUint16(p+10,true);
    const csize=dv.getUint32(p+20,true);
    const nlen=dv.getUint16(p+28,true), elen=dv.getUint16(p+30,true), clen=dv.getUint16(p+32,true);
    const lho=dv.getUint32(p+42,true);
    const name=asText(u8.subarray(p+46,p+46+nlen));
    const lnlen=dv.getUint16(lho+26,true), lelen=dv.getUint16(lho+28,true);
    const dstart=lho+30+lnlen+lelen;
    const raw=u8.subarray(dstart,dstart+csize);
    files[name]={method,raw};
    p+=46+nlen+elen+clen;
  }
  return {
    names:()=>Object.keys(files),
    async get(name){
      const f=files[name]; if(!f) return null;
      if(f.method===0) return f.raw;
      if(f.method===8) return await inflateRaw(f.raw);
      fail("Unsupported ZIP compression method "+f.method+".");
    }
  };
}
function zipWrite(entries){ /* entries: [{name, data:Uint8Array}] – STORED */
  const enc=new TextEncoder();
  const locals=[], centrals=[];
  let offset=0;
  for(const e of entries){
    const nm=enc.encode(e.name), crc=crc32(e.data), sz=e.data.length;

```

```

const lh=new Uint8Array(30+nm.length), lv=new DataView(lh.buffer);
lv.setUint32(0,0x04034b50,true); lv.setUint16(4,20,true); lv.setUint16(6,0,true);
lv.setUint16(8,0,true); lv.setUint16(10,0,true); lv.setUint16(12,0,true);
lv.setUint32(14,crc,true); lv.setUint32(18,sz,true); lv.setUint32(22,sz,true);
lv.setUint16(26,nm.length,true); lv.setUint16(28,0,true);
lh.set(nm,30);
locals.push(lh,e.data);
const ch=new Uint8Array(46+nm.length), cv=new DataView(ch.buffer);
cv.setUint32(0,0x02014b50,true); cv.setUint16(4,20,true); cv.setUint16(6,20,true);
cv.setUint16(8,0,true); cv.setUint16(10,0,true);
cv.setUint32(16,crc,true); cv.setUint32(20,sz,true); cv.setUint32(24,sz,true);
cv.setUint16(28,nm.length,true);
cv.setUint32(42,offset,true);
ch.set(nm,46);
centrals.push(ch);
offset += lh.length + sz;
}
let cdSize=0; for(const c of centrals) cdSize+=c.length;
const eocd=new Uint8Array(22), ev=new DataView(eocd.buffer);
ev.setUint32(0,0x06054b50,true);
ev.setUint16(8,entries.length,true); ev.setUint16(10,entries.length,true);
ev.setUint32(12,cdSize,true); ev.setUint32(16,offset,true);
const parts=[...locals,...centrals,eocd];
let total=0; for(const p of parts) total+=p.length;
const out=new Uint8Array(total); let o=0;
for(const p of parts){ out.set(p,o); o+=p.length; }
return out;
}

```

```
/* =====
```

5. MESH PARSERS

```
===== */
```

```

function parseOBJ(buf){
const m=new Model("mesh"), V=[], T=[], E=[];
for(const raw of asText(buf).split(/\r?\n/)){
const line=raw.split("#")[0].trim();
if(!line) continue;
const p=line.split(/\s+/);
if(p[0]=== "v"){
V.push(num(p[1]),num(p[2]),num(p[3]));
} else if(p[0]=== "f"){
const idx=[];
for(let i=1;i<p.length;i++){
let k=parseInt(p[i].split("/")[0],10);
if(!Number.isFinite(k)) continue;

```

```

        idx.push(k>0 ? k-1 : V.length/3 + k);
    }
    if(idx.length>=3) fanTriangles(idx,T);
    else if(idx.length===2) E.push([idx[0],idx[1]]);
} else if(p[0]=== "I"){
    const idx=[];
    for(let i=1;i<p.length;i++){
        let k=parseInt(p[i],10);
        if(Number.isFinite(k)) idx.push(k>0?k-1:V.length/3+k);
    }
    for(let i=0;i+1<idx.length;i++) E.push([idx[i],idx[i+1]]);
}
}
if(!V.length) fail("OBJ: no vertices found.");
m.verts=new Float32Array(V);
if(T.length) m.tris=new Int32Array(T);
if(E.length) m.edges=E;
return m;
}

function parseSTL(buf){
    const u8=new Uint8Array(buf), dv=new DataView(buf);
    const binaryPlausible = u8.length>=84 && (84 + 50*dv.getUint32(80,true) === u8.length);
    const head = asText(u8.subarray(0,Math.min(256,u8.length))).trim().toLowerCase();
    if(!binaryPlausible && head.startsWith("solid") && /facet|endsolid/.test(head+asText(u8.subarray(0,2048)).toLowerCase()))
        return parseSTLAscii(asText(buf));
    if(binaryPlausible) return parseSTLBinary(buf);
    if(head.startsWith("solid")) return parseSTLAscii(asText(buf));
    fail("STL: file matches neither the ASCII nor the binary layout.");
}

function parseSTLAscii(text){
    const m=new Model("mesh"), V=[], T=[];
    const re=/vertex\s+(\S+)\s+(\S+)\s+(\S+)/g;
    let mt;
    while((mt=re.exec(text))){ V.push(num(mt[1]),num(mt[2]),num(mt[3])); }
    if(!V.length) fail("STL (ASCII): no vertex records.");
    if((V.length/3)%3!==0) fail("STL (ASCII): vertex count is not a multiple of three.");
    for(let i=0;i<V.length/3;i++) T.push(i);
    m.verts=new Float32Array(V); m.tris=new Int32Array(T);
    return m;
}

function parseSTLBinary(buf){
    const dv=new DataView(buf), n=dv.getUint32(80,true);
    const m=new Model("mesh");
    const V=new Float32Array(n*9), T=new Int32Array(n*3);

```

```

let o=84;
for(let f=0;f<n;f++){
  o+=12; // face normal, recomputed at draw time
  for(let k=0;k<3;k++){
    V[f*9+k*3] =dv.getFloat32(o,true);
    V[f*9+k*3+1]=dv.getFloat32(o+4,true);
    V[f*9+k*3+2]=dv.getFloat32(o+8,true);
    o+=12;
  }
  T[f*3]=f*3; T[f*3+1]=f*3+1; T[f*3+2]=f*3+2;
  o+=2; // attribute byte count
}
m.verts=V; m.tris=T;
return m;
}

function parsePLY(buf){
  const u8=new Uint8Array(buf);
  // locate end_header
  let hEnd=-1;
  const probe=asText(u8.subarray(0,Math.min(u8.length,65536)));
  const mm=probe.match(/end_header\s*?\r?\n/);
  if(!mm) fail("PLY: end_header not found.");
  hEnd=mm.index+mm[0].length;
  const header=probe.slice(0,mm.index);
  const lines=header.split(/\r?\n/).map(s=>s.trim()).filter(Boolean);
  if(!/^ply$/i.test(lines[0])) fail("PLY: missing magic 'ply'.");
  let format=null, elements=[], cur=null;
  for(const L of lines){
    const p=L.split(/\s+/);
    if(p[0]=== "format") format=p[1];
    else if(p[0]=== "element"){ cur={ name:p[1],count:parseInt(p[2],10),props:[]; elements.push(cur); }
    else if(p[0]=== "property" && cur){
      if(p[1]=== "list") cur.props.push({list:true,ctype:p[2],etype:p[3],name:p[4]});
      else cur.props.push({list:false,type:p[1],name:p[2]});
    }
  }
  if(!format) fail("PLY: no format line.");
  const m=new Model("mesh");
  const V=[], T=[], C=[];
  let hasColor=false;

  if(format=== "ascii"){
    const body=asText(u8.subarray(hEnd)).split(/\r?\n/);
    let li=0;

```

```

const nextTokens=()=>=>{
  while(li<body.length){
    const t=body[li++].trim();
    if(t) return t.split(/\s+/);
  }
  return null;
};
for(const el of elements){
  for(let i=0;i<el.count;i++){
    const tk=nextTokens();
    if(!tk) fail("PLY (ASCII): file ends inside element '"+el.name+"'");
    if(el.name==="vertex"){
      let k=0; const rec={};
      for(const pr of el.props){
        if(pr.list){ const c=parseInt(tk[k++],10); k+=c; }
        else rec[pr.name]=num(tk[k++]);
      }
      V.push(rec.x,rec.y,rec.z);
      if(rec.red!==undefined){ hasColor=true; C.push(rec.red|0,rec.green|0,rec.blue|0); }
      else C.push(200,200,200);
    } else if(el.name==="face"){
      let k=0, idx=null;
      for(const pr of el.props){
        if(pr.list){
          const c=parseInt(tk[k++],10);
          const a=[]; for(let j=0;j<c;j++) a.push(parseInt(tk[k++],10));
          if(!idx) idx=a;
        } else k++;
      }
      if(idx && idx.length>=3) fanTriangles(idx,T);
    }
  }
}
} else {
  const little = /little/i.test(format);
  const dv=new DataView(buf);
  let o=hEnd;
  const SZ={char:1,uchar:1,int8:1,uint8:1,short:2,ushort:2,int16:2,uint16:2,
    int4:4,uint4:4,int32:4,uint32:4,float:4,float32:4,double:8,float64:8};
  const rd=t=>{
    const s=SZ[t]; if(!s) fail("PLY: unknown scalar type '"+t+"'");
    let v;
    switch(t){
      case "char": case "int8": v=dv.getInt8(o); break;
      case "uchar": case "uint8": v=dv.getUint8(o); break;

```

```

    case "short": case "int16": v=dv.getInt16(o,little); break;
    case "ushort": case "uint16": v=dv.getUint16(o,little); break;
    case "int": case "int32": v=dv.getInt32(o,little); break;
    case "uint": case "uint32": v=dv.getUint32(o,little); break;
    case "float": case "float32": v=dv.getFloat32(o,little); break;
    default: v=dv.getFloat64(o,little);
  }
  o+=s; return v;
};

for(const el of elements){
  for(let i=0;i<el.count;i++){
    if(el.name==="vertex"){
      const rec={};
      for(const pr of el.props){
        if(pr.list){ const c=rd(pr.ctype); for(let j=0;j<c;j++) rd(pr.etype); }
        else rec[pr.name]=rd(pr.type);
      }
      V.push(rec.x,rec.y,rec.z);
      if(rec.red!==undefined){ hasColor=true; C.push(rec.red|0,rec.green|0,rec.blue|0); }
      else C.push(200,200,200);
    } else if(el.name==="face"){
      let idx=null;
      for(const pr of el.props){
        if(pr.list){
          const c=rd(pr.ctype); const a=[];
          for(let j=0;j<c;j++) a.push(rd(pr.etype));
          if(!idx) idx=a;
        } else rd(pr.type);
      }
      if(idx && idx.length>=3) fanTriangles(idx,T);
    } else {
      for(const pr of el.props){
        if(pr.list){ const c=rd(pr.ctype); for(let j=0;j<c;j++) rd(pr.etype); }
        else rd(pr.type);
      }
    }
  }
}

if(!V.length) fail("PLY: no vertices.");
m.verts=new Float32Array(V);
if(T.length) m.tris=new Int32Array(T);
if(hasColor) m.colors=new Uint8Array(C);
if(!T.length) m.kind="points";
return m;

```

```

}

function parseOFF(buf){
  const toks=[];
  for(const raw of asText(buf).split(/\r?\n/)){
    const line=raw.split("#")[0].trim();
    if(line) toks.push(...line.split(/\s+/));
  }
  if(!toks.length) fail("OFF: empty file.");
  let i=0;
  if(/^([CN]*OFF$/i.test(toks[0])) i=1;
  else if(/^OFF/i.test(toks[0])) i=1;
  const nv=parseInt(toks[i++],10), nf=parseInt(toks[i++],10);
  i++; // edge count, ignored
  if(!Number.isFinite(nv)||!Number.isFinite(nf)) fail("OFF: malformed counts line.");
  const V=new Float32Array(nv*3), T=[];
  for(let k=0;k<nv;k++){
    V[k*3]=num(toks[i++]); V[k*3+1]=num(toks[i++]); V[k*3+2]=num(toks[i++]);
  }
  for(let k=0;k<nf;k++){
    const c=parseInt(toks[i++],10);
    if(!Number.isFinite(c)) fail("OFF: malformed face record "+k+");");
    const idx=[];
    for(let j=0;j<c;j++) idx.push(parseInt(toks[i++],10));
    // an optional per-face colour may follow; skip trailing floats
    while(i<toks.length && /^[+-]\d/.test(toks[i]) && toks[i].includes(".")) i++;
    if(idx.length>=3) fanTriangles(idx,T);
  }
  const m=new Model("mesh"); m.verts=V;
  if(T.length) m.tris=new Int32Array(T); else m.kind="points";
  return m;
}

function parseVTK(buf){
  const text=asText(buf);
  if(!/^#\s*vtk\s+DataFile$/i.test(text.trim()))
    fail("VTK: only the legacy ASCII '# vtk DataFile' format is supported.");
  if(/^#\s*BINARY\s*$/mi.test(text)) fail("VTK: binary legacy files are not supported (ASCII only).");
  const lines=text.split(/\r?\n/);
  const m=new Model("mesh");
  let V=null, T=[], E=[];
  for(let i=0;i<lines.length;i++){
    const p=lines[i].trim().split(/\s+/);
    const kw=(p[0]||"").toUpperCase();
    if(kw=="POINTS"){

```

```

const n=parseInt(p[1],10);
const vals=[];
i++;
while(i<lines.length && vals.length<n*3){
    const t=lines[i].trim();
    if(t && /^[A-Za-z]/.test(t)) vals.push(...t.split(/\s+/).map(num));
    else if(t && /^[A-Za-z]/.test(t)) break;
    i++;
}
i--;
if(vals.length<n*3) fail("VTK: POINTS declares "+n+" points but only "+(vals.length/3|0)+" were read.");
V=new Float32Array(vals.slice(0,n*3));
} else if(kw==="POLYGONS"||kw==="TRIANGLE_STRIP"||kw==="LINES"||kw==="CELLS"||kw==="VERTICES"){
    const n=parseInt(p[1],10);
    const ints=[];
    i++;
    while(i<lines.length && ints.length<parseInt(p[2],10)){
        const t=lines[i].trim();
        if(t && /^[A-Za-z]/.test(t)) ints.push(...t.split(/\s+/).map(s=>parseInt(s,10)));
        else if(t && /^[A-Za-z]/.test(t)) break;
        i++;
    }
    i--;
    let k=0;
    for(let c=0;c<n && k<ints.length;c++){
        const cnt=ints[k++]; const idx=ints.slice(k,k+cnt); k+=cnt;
        if(kw==="LINES"){ for(let j=0;j+1<idx.length;j++) E.push([idx[j],idx[j+1]]); }
        else if(kw==="TRIANGLE_STRIP"){ for(let j=0;j+2<idx.length;j++) T.push(idx[j],idx[j+1],idx[j+2]); }
        else if(kw==="VERTICES"){ /* points only */ }
        else if(idx.length>=3) fanTriangles(idx,T);
    }
}
}
if(!V) fail("VTK: no POINTS section.");
m.verts=V;
if(T.length) m.tris=new Int32Array(T);
if(E.length) m.edges=E;
if(!T.length && !E.length) m.kind="points";
return m;
}

function parse3DS(buf){
    const dv=new DataView(buf), end=buf.byteLength;
    const V=[], T=[];
    let base=0;

```



```

function walk(start,stop){
  let p=start;
  while(p+6<=stop){
    const id=dv.getUint16(p,true), len=dv.getUint32(p+2,true);
    if(len<6 || p+len>stop) break;
    const body=p+6;
    switch(id){
      case 0x4D4D: case 0x3D3D: case 0x4100:
        walk(body,p+len); break;
      case 0x4000: { // named object: skip the ASCIIZ name, then recurse
        let q=body; while(q<p+len && dv.getUint8(q)!=0) q++;
        walk(q+1,p+len); break;
      }
      case 0x4110: { // vertex list
        const n=dv.getUint16(body,true);
        base=V.length/3;
        for(let i=0;i<n;i++){
          const o=body+2+i*12;
          V.push(dv.getFloat32(o,true),dv.getFloat32(o+4,true),dv.getFloat32(o+8,true));
        }
        break;
      }
      case 0x4120: { // face list
        const n=dv.getUint16(body,true);
        for(let i=0;i<n;i++){
          const o=body+2+i*8;
          T.push(base+dv.getUint16(o,true), base+dv.getUint16(o+2,true), base+dv.getUint16(o+4,true));
        }
        break;
      }
    }
    p+=len;
  }
}

if(end<6 || dv.getUint16(0,true)!=0x4D4D) fail("3DS: primary chunk 0x4D4D not found.");
walk(0,end);
if(!V.length) fail("3DS: no vertex data found.");
const m=new Model("mesh"); m.verts=new Float32Array(V);
if(T.length) m.tris=new Int32Array(T);
return m;
}

/* --- glTF / GLB ----- */
function b64ToU8(b64){
  const bin=atob(b64), u=new Uint8Array(bin.length);

```

```

    for(let i=0;i<bin.length;i++) u[i]=bin.charCodeAtAt(i);
    return u;
}
function mat4mul(a,b){ // column-major, as glTF stores them
    const o=new Float64Array(16);
    for(let c=0;c<4;c++)for(let r=0;r<4;r++){
        let s=0; for(let k=0;k<4;k++) s+=a[k*4+r]*b[c*4+k];
        o[c*4+r]=s;
    }
    return o;
}
function trsToMat(n){
    const t=n.translation||[0,0,0], r=n.rotation||[0,0,0,1], s=n.scale||[1,1,1];
    const [x,y,z,w]=r;
    const R=[
        1-2*(y*y+z*z), 2*(x*y+z*w), 2*(x*z-y*w), 0,
        2*(x*y-z*w), 1-2*(x*x+z*z), 2*(y*z+x*w), 0,
        2*(x*z+y*w), 2*(y*z-x*w), 1-2*(x*x+y*y), 0,
        0,0,0,1];
    for(let c=0;c<3;c++)for(let r2=0;r2<3;r2++) R[c*4+r2]*=s[c];
    R[12]=t[0]; R[13]=t[1]; R[14]=t[2];
    return new Float64Array(R);
}
function parseGLTF(buf, isGLB){
    let json, binChunk=null;
    if(isGLB){
        const dv=new DataView(buf);
        if(dv.getUint32(0,true)!=0x46546C67) fail("GLB: bad magic (expected 'glTF').");
        const total=dv.getUint32(8,true);
        let o=12;
        while(o+8<=Math.min(total,buf.byteLength)){
            const len=dv.getUint32(o,true), type=dv.getUint32(o+4,true);
            const data=new Uint8Array(buf,o+8,len);
            if(type===0x4E4F534A) json=JSON.parse(asText(data));
            else if(type===0x004E4942) binChunk=data;
            o+=8+len+((4-len%4)%4===4?0:(len%4?4-len%4:0));
            if(len%4) o+= 0; // padding already handled by the chunk length alignment
        }
        if(!json) fail("GLB: JSON chunk missing.");
    } else {
        json=JSON.parse(asText(buf));
    }
    const buffers=(json.buffers||[]).map((b,i)=>{
        if(b.uri===undefined) {
            if(!binChunk) fail("glTF: buffer "+i+" has no URI and no GLB binary chunk is present.");

```

```

    return binChunk;
}
const dm=/^data:[^;]*;base64,(.*)$/exec(b.uri);
if(dm) return b64ToU8(dm[1]);
fail("glTF: buffer "+i+" references the external file '"+b.uri+"'. Load the .glb, or embed the buffer as a data URI.");
});
const NCOMP={SCALAR:1,VEC2:2,VEC3:3,VEC4:4,MAT2:4,MAT3:9,MAT4:16};
const CSIZE={5120:1,5121:1,5122:2,5123:2,5125:4,5126:4};
function readAccessor(ai){
    const acc=json.accessors[ai];
    if(!acc) fail("glTF: accessor "+ai+" is missing.");
    const nc=NCOMP[acc.type], cs=CSIZE[acc.componentType];
    if(!nc||!cs) fail("glTF: accessor "+ai+" uses an unsupported type.");
    const out=new Float64Array(acc.count*nc);
    if(acc.bufferView===undefined) return out; // sparse-only accessor: treated as zeros
    const bv=json.bufferViews[acc.bufferView];
    const src=buffers[bv.buffer];
    const dv=new DataView(src.buffer, src.byteOffset, src.byteLength);
    const base=(bv.byteOffset||0)+(acc.byteOffset||0);
    const stride=bv.byteStride || nc*cs;
    for(let i=0;i<acc.count;i++){for(let c=0;c<nc;c++){
        const o=base+i*stride+c*cs;
        let v;
        switch(acc.componentType){
            case 5120: v=dv.getInt8(o); break;
            case 5121: v=dv.getUint8(o); break;
            case 5122: v=dv.getInt16(o,true); break;
            case 5123: v=dv.getUint16(o,true); break;
            case 5125: v=dv.getUint32(o,true); break;
            default: v=dv.getFloat32(o,true);
        }
        out[i*nc+c]=v;
    }
    return out;
}
const V=[], T=[];
let skipped=0;
function emitMesh(mi, M){
    const mesh=json.meshes[mi]; if(!mesh) return;
    for(const prim of mesh.primitives||[]){
        const mode=prim.mode===undefined?4:prim.mode;
        if(mode!==4){ skipped++; continue; }
        if(prim.attributes.POSITION===undefined){ skipped++; continue; }
        const pos=readAccessor(prim.attributes.POSITION);
        const base=V.length/3, n=pos.length/3;

```

```

    for(let i=0;i<n;i++){
        const x=pos[i*3], y=pos[i*3+1], z=pos[i*3+2];
        V.push(M[0]*x+M[4]*y+M[8]*z+M[12],
            M[1]*x+M[5]*y+M[9]*z+M[13],
            M[2]*x+M[6]*y+M[10]*z+M[14]);
    }
    if(prim.indices!==undefined){
        const idx=readAccessor(prim.indices);
        for(let i=0;i+2<idx.length;i+=3) T.push(base+idx[i],base+idx[i+1],base+idx[i+2]);
    } else {
        for(let i=0;i+2<n;i+=3) T.push(base+i,base+i+1,base+i+2);
    }
}
}

function walkNode(ni,parent){
    const nd=json.nodes[ni]; if(!nd) return;
    const local=nd.matrix?new Float64Array(nd.matrix):trsToMat(nd);
    const M=mat4mul(parent,local);
    if(nd.mesh!==undefined) emitMesh(nd.mesh,M);
    for(const c of nd.children||[]) walkNode(c,M);
}

const I=new Float64Array([1,0,0,0, 0,1,0,0, 0,0,1,0, 0,0,0,1]);
const scenes=json.scenes||[];
const sc=scenes[json.scene===undefined?0:json.scene];
if(sc && sc.nodes) for(const n of sc.nodes) walkNode(n,I);
else if(json.nodes) json.nodes.forEach((nd,i)=>{ if(nd.mesh!==undefined) emitMesh(nd.mesh,I); });
else if(json.meshes) json.meshes.forEach((_,i)=>emitMesh(i,I));
if(!V.length) fail("glTF: no TRIANGLES primitive with POSITION data was found"+(skipped?" (" +skipped+" primitive(s)
skipped).":"."));
const m=new Model("mesh"); m.verts=new Float32Array(V); m.tris=new Int32Array(T);
if(skipped) m.source="note: "+skipped+" non-triangle primitive(s) skipped";
return m;
}

async function parse3MF(buf){
    const zip=await zipRead(buf);
    let name=zip.names().find(n=>/^3dV.*\.model$/i.test(n)) ||
        zip.names().find(n=>/\.model$/i.test(n));
    if(!name) fail("3MF: no *.model part inside the container.");
    const xml=asText(await zip.get(name));
    const doc=new DOMParser().parseFromString(xml,"application/xml");
    if(doc.querySelector("parsererror")) fail("3MF: the model part is not well-formed XML.");
    const objs={};
    for(const ob of doc.getElementsByTagName("object")){
        const id=ob.getAttribute("id");

```

```

const mesh=ob.getElementsByTagName("mesh")[0];
if(!mesh) continue;
const V=[], T=[];
for(const v of mesh.getElementsByTagName("vertex"))
    V.push(num(v.getAttribute("x")),num(v.getAttribute("y")),num(v.getAttribute("z")));
for(const t of mesh.getElementsByTagName("triangle"))
    T.push(parseInt(t.getAttribute("v1"),10),parseInt(t.getAttribute("v2"),10),parseInt(t.getAttribute("v3"),10));
objs[id]={V,T};
}
const items=[...doc.getElementsByTagName("item")];
const V=[], T=[];
const addObj=(o,tr)=>{
    const base=V.length/3;
    for(let i=0;i<o.V.length;i+=3){
        let x=o.V[i],y=o.V[i+1],z=o.V[i+2];
        if(tr){
            const t=tr;
            const nx=t[0]*x+t[3]*y+t[6]*z+t[9];
            const ny=t[1]*x+t[4]*y+t[7]*z+t[10];
            const nz=t[2]*x+t[5]*y+t[8]*z+t[11];
            x=nx; y=ny; z=nz;
        }
        V.push(x,y,z);
    }
    for(const k of o.T) T.push(base+k);
};
if(items.length){
    for(const it of items){
        const o=objs[it.getAttribute("objectid")];
        if(!o) continue;
        const ts=it.getAttribute("transform");
        const tr=ts ? ts.trim().split(/\s+/).map(num) : null;
        addObj(o, tr && tr.length===12 ? tr : null);
    }
} else {
    for(const id in objs) addObj(objs[id],null);
}
if(!V.length) fail("3MF: the model contains no mesh vertices.");
const m=new Model("mesh"); m.verts=new Float32Array(V);
if(T.length) m.tris=new Int32Array(T);
return m;
}

```

```

function parseDAE(buf){
    const doc=new DOMParser().parseFromString(asText(buf),"application/xml");

```

```

if(doc.querySelector("parsererror")) fail("DAE: not well-formed XML.");
const tag=(el,n)=>[...el.getElementsByTagName("*")].filter(x=>x.localName===n);
const V=[], T=[];
let unit=1;
const au=tag(doc.documentElement,"unit")[0];
if(au && au.getAttribute("meter")) unit=num(au.getAttribute("meter"))||1;
for(const mesh of tag(doc.documentElement,"mesh")){
  // gather sources
  const src={};
  for(const s of tag(mesh,"source")){
    const fa=tag(s,"float_array")[0];
    if(!fa) continue;
    const arr=fa.textContent.trim().split(/\s+/).map(num);
    const acc=tag(s,"accessor")[0];
    const stride=acc && acc.getAttribute("stride") ? parseInt(acc.getAttribute("stride"),10) : 3;
    src["#" + s.getAttribute("id")]={arr,stride};
  }
  // <vertices> indirection
  const vertsEl=tag(mesh,"vertices")[0];
  let posSrc=null;
  const vertsId = vertsEl ? "#" + vertsEl.getAttribute("id") : null;
  if(vertsEl){
    for(const inp of tag(vertsEl,"input"))
      if(inp.getAttribute("semantic")==="POSITION") posSrc=src[inp.getAttribute("source")];
  }
  const base=V.length/3;
  if(posSrc){
    const st=posSrc.stride;
    for(let i=0;i+2<posSrc.arr.length;i+=st)
      V.push(posSrc.arr[i]*unit,posSrc.arr[i+1]*unit,posSrc.arr[i+2]*unit);
  }
  const prims=[...tag(mesh,"triangles"),...tag(mesh,"polylist"),...tag(mesh,"polygons")];
  for(const pr of prims){
    const inputs=tag(pr,"input");
    let vOffset=-1, maxOff=0;
    for(const inp of inputs){
      const off=parseInt(inp.getAttribute("offset"))||"0",10);
      maxOff=Math.max(maxOff,off);
      const sem=inp.getAttribute("semantic");
      if(sem==="VERTEX" || (sem==="POSITION" && vOffset<0)) vOffset=off;
    }
    if(vOffset<0) vOffset=0;
    const stride=maxOff+1;
    const pEls=tag(pr,"p");
    const vcountEl=tag(pr,"vcount")[0];

```

```

const vcounts = vcountEl ? vcountEl.textContent.trim().split(/\s+/).map(x=>parseInt(x,10)) : null;
let allP=[];
if(pr.localName==="polygons"){
  for(const p of pEls){
    const ints=p.textContent.trim().split(/\s+/).map(x=>parseInt(x,10));
    const idx=[]; for(let i=vOffset;i<ints.length;i+=stride) idx.push(base+ints[i]);
    if(idx.length>=3) fanTriangles(idx,T);
  }
  continue;
}
if(pEls[0]) allP=pEls[0].textContent.trim().split(/\s+/).map(x=>parseInt(x,10));
if(!allP.length) continue;
if(vcounts){
  let k=0;
  for(const c of vcounts){
    const idx=[];
    for(let j=0;j<c;j++) idx.push(base+allP[(k+j)*stride+vOffset]);
    k+=c;
    if(idx.length>=3) fanTriangles(idx,T);
  }
} else {
  const flat=[];
  for(let i=vOffset;i<allP.length;i+=stride) flat.push(base+allP[i]);
  for(let i=0;i+2<flat.length;i+=3) T.push(flat[i],flat[i+1],flat[i+2]);
}
}
}
if(!V.length) fail("DAE: no POSITION source found in any <mesh>.");
const m=new Model("mesh"); m.verts=new Float32Array(V);
if(T.length) m.tris=new Int32Array(T);
return m;
}

```

```

function parseX3D(buf){
  const doc=new DOMParser().parseFromString(asText(buf),"application/xml");
  if(doc.querySelector("parsererror")) fail("X3D: not well-formed XML.");
  const all=[...doc.getElementsByTagName("*")];
  const V=[], T=[], E=[];
  const coordOf=el=>{
    const c=[...el.getElementsByTagName("*")].find(x=>x.localName==="Coordinate"||x.localName==="CoordinateDouble");
    if(!c) return null;
    const p=c.getAttribute("point");
    if(!p) return null;
    return p.trim().split(/[ \,]+/).map(num);
  };
}

```

```

const idxOf=(el,attr)=>{
  const s=el.getAttribute(attr);
  return s ? s.trim().split(/[^\s,]+/).map(x=>parseInt(x,10)).filter(Number.isFinite) : [];
};

for(const el of all){
  if(el.localName==="IndexedFaceSet"||el.localName==="IndexedTriangleSet"||
    el.localName==="IndexedLineSet"||el.localName==="PointSet"){
    const pts=coordOf(el);
    if(!pts) continue;
    const base=V.length/3;
    for(let i=0;i+2<pts.length;i+=3) V.push(pts[i],pts[i+1],pts[i+2]);
    if(el.localName==="IndexedFaceSet"){
      const ci=idxOf(el,"coordIndex"); let poly=[];
      for(const k of ci){
        if(k<0){ if(poly.length>=3) fanTriangles(poly.map(q=>q+base),T); poly=[]; }
        else poly.push(k);
      }
      if(poly.length>=3) fanTriangles(poly.map(q=>q+base),T);
    } else if(el.localName==="IndexedTriangleSet"){
      const ci=idxOf(el,"index");
      for(let i=0;i+2<ci.length;i+=3) T.push(base+ci[i],base+ci[i+1],base+ci[i+2]);
    } else if(el.localName==="IndexedLineSet"){
      const ci=idxOf(el,"coordIndex"); let prev=-1;
      for(const k of ci){
        if(k<0){ prev=-1; continue; }
        if(prev>=0) E.push([base+prev,base+k]);
        prev=k;
      }
    }
  }
}

if(!V.length) fail("X3D: no Coordinate 'point' data found.");
const m=new Model("mesh"); m.verts=new Float32Array(V);
if(T.length) m.tris=new Int32Array(T);
if(E.length) m.edges=E;
if(!T.length && !E.length) m.kind="points";
return m;
}

```

```

function parseDXF(buf){
  const lines=asText(buf).split(/\r?\n/);
  const pairs=[];
  for(let i=0;i+1<lines.length;i+=2){
    const code=parseInt(lines[i].trim(),10);
    if(!Number.isFinite(code)) continue;

```



```

    pairs.push([code,lines[i+1].trim()]);
  }
  if(!pairs.length) fail("DXF: no group code/value pairs (ASCII DXF expected; binary DXF is not supported).");
  const V=[], T=[], E=[];
  const addV=(x,y,z)=>{ V.push(x,y,z||0); return V.length/3-1; };
  let i=0, inEnt=false;
  let ent=null, g={}, polyVerts=null, polyClosed=false;
  const flush()=>{
    if(!ent) return;
    if(ent==="3DFACE"){
      const p=[];
      for(const k of [0,1,2,3]){
        const x=g[10+k], y=g[20+k], z=g[30+k];
        if(x===undefined||y===undefined) continue;
        p.push(addV(x,y,z));
      }
      if(p.length>=3){
        // a degenerate 4th corner equal to the 3rd marks a triangle
        if(p.length===4 && g[13]===g[12] && g[23]===g[22] && g[33]===g[32]) p.pop();
        fanTriangles(p,T);
      }
    }
    else if(ent==="LINE"){
      if(g[10]!==undefined && g[11]!==undefined){
        const a=addV(g[10],g[20],g[30]), b=addV(g[11],g[21],g[31]);
        E.push([a,b]);
      }
    }
    else if(ent==="POINT"){
      if(g[10]!==undefined) addV(g[10],g[20],g[30]);
    }
    else if(ent==="LWPOLYLINE"){
      if(g.__pts && g.__pts.length>1){
        const ids=g.__pts.map(p=>addV(p[0],p[1],g[38]||0));
        for(let k=0;k+1<ids.length;k++) E.push([ids[k],ids[k+1]]);
        if(g[70]&1) E.push([ids[ids.length-1],ids[0]]);
      }
    }
  }
  ent=null; g={};
};
for(i=0;i<pairs.length;i++){
  const [c,v]=pairs[i];
  if(c===0){
    flush();
    if(v==="SECTION"||v==="ENDSEC"||v==="EOF"){ continue; }
    if(v==="POLYLINE"){ polyVerts=[]; polyClosed=false; continue; }
    if(v==="VERTEX"){ ent="VERTEX"; g={}; continue; }
    if(v==="SEQEND"){

```

```

    if(polyVerts && polyVerts.length>1){
        const ids=polyVerts.map(p=>addV(p[0],p[1],p[2]));
        for(let k=0;k+1<ids.length;k++) E.push([ids[k],ids[k+1]]);
        if(polyClosed) E.push([ids[ids.length-1],ids[0]]);
    }
    polyVerts=null; continue;
}
ent=v; g={};
if(v==="LWPOLYLINE") g.__pts=[];
continue;
}
if(ent==="VERTEX"){
    if(c===10) g[10]=num(v);
    if(c===20) g[20]=num(v);
    if(c===30) g[30]=num(v);
    // vertex closes at the next 0 group; capture then
    if(polyVerts && c===30) polyVerts.push([g[10]||0,g[20]||0,g[30]||0]);
    continue;
}
if(ent==="LWPOLYLINE"){
    if(c===10) g.__pending=num(v);
    else if(c===20 && g.__pending!==undefined){ g.__pts.push([g.__pending,num(v)]); g.__pending=undefined; }
    else if(c===70) g[70]=parseInt(v,10);
    else if(c===38) g[38]=num(v);
    continue;
}
if(ent) g[c]=num(v);
}
flush();
if(!V.length) fail("DXF: no 3DFACE, LINE, POLYLINE, LWPOLYLINE or POINT entity produced geometry.");
const m=new Model(T.length?"mesh":(E.length?"mesh":"points"));
m.verts=new Float32Array(V);
if(T.length) m.tris=new Int32Array(T);
if(E.length) m.edges=E;
return m;
}

/* --- FBX (binary 6.x/7.x and ASCII) ----- */
async function parseFBX(buf){
    const u8=new Uint8Array(buf);
    const magic="Kaydara FBX Binary";
    if(u8.length>20 && asText(u8.subarray(0,18))===magic) return await parseFBXBinary(buf);
    const text=asText(buf);
    if(/Vertices\s*:.test(text)) return parseFBXAscii(text);
    fail("FBX: neither the binary signature nor an ASCII 'Vertices:' block was found.");
}

```

```

}
async function parseFBXBinary(buf){
  const dv=new DataView(buf), u8=new Uint8Array(buf);
  const version=dv.getUint32(23,true);
  const wide=version>=7500;
  const pend=[];
  let pos=wide?27:27;
  function readArray(type){
    const len=dv.getUint32(pos,true), enc=dv.getUint32(pos+4,true), clen=dv.getUint32(pos+8,true);
    pos+=12;
    const raw=u8.subarray(pos,pos+ (enc? clen : len*({f:4,d:8,l:8,i:4,b:1}[type])));
    pos += enc ? clen : len*({f:4,d:8,l:8,i:4,b:1}[type]);
    const holder={__array:true,type,type,len,enc,raw,values:null};
    if(enc) pend.push(holder); else holder.values=decodeArr(type,raw,len);
    return holder;
  }
  function decodeArr(type,bytes,len){
    const d=new DataView(bytes.buffer,bytes.byteOffset,bytes.byteLength);
    const out = type==="d"||type==="f" ? new Float64Array(len) : new Int32Array(len);
    for(let i=0;i<len;i++){
      switch(type){
        case "f": out[i]=d.getFloat32(i*4,true); break;
        case "d": out[i]=d.getFloat64(i*8,true); break;
        case "i": out[i]=d.getInt32(i*4,true); break;
        case "l": out[i]=Number(d.getBigInt64(i*8,true)); break;
        case "b": out[i]=d.getUint8(i); break;
      }
    }
    return out;
  }
  function readProp(){
    const t=String.fromCharCode(dv.getUint8(pos++));
    switch(t){
      case "Y": { const v=dv.getInt16(pos,true); pos+=2; return v; }
      case "C": { const v=dv.getUint8(pos); pos+=1; return !!v; }
      case "I": { const v=dv.getInt32(pos,true); pos+=4; return v; }
      case "F": { const v=dv.getFloat32(pos,true); pos+=4; return v; }
      case "D": { const v=dv.getFloat64(pos,true); pos+=8; return v; }
      case "L": { const v=Number(dv.getBigInt64(pos,true)); pos+=8; return v; }
      case "f": case "d": case "l": case "i": case "b": return readArray(t);
      case "S": case "R": {
        const n=dv.getUint32(pos,true); pos+=4;
        const s=asText(u8.subarray(pos,pos+n)); pos+=n; return s;
      }
      default: fail("FBX: unknown property type code '"+t+"'");
    }
  }
}

```

```

    }
  }
function readNode(){
  const endOff = wide ? Number(dv.getBigUint64(pos,true)) : dv.getUint32(pos,true);
  const nProps = wide ? Number(dv.getBigUint64(pos+8,true)) : dv.getUint32(pos+4,true);
  pos += wide ? 24 : 12;
  const nameLen=dv.getUint8(pos); pos+=1;
  if(endOff===0 && nProps===0 && nameLen===0) return null;
  const name=asText(u8.subarray(pos,pos+nameLen)); pos+=nameLen;
  const props=[];
  for(let i=0;i<nProps;i++) props.push(readProp());
  const kids=[];
  const sentinel = wide?25:13;
  while(pos < endOff - sentinel){
    const k=readNode();
    if(!k) break;
    kids.push(k);
  }
  pos = endOff;
  return {name,props,kids};
}
const roots=[];
while(pos < u8.length - (wide?25:13)){
  const n=readNode();
  if(!n) break;
  roots.push(n);
}
for(const h of pend){
  const inf=await inflateRaw(h.raw);
  h.values=decodeArr(h.type,inf,h.len);
}
const V=[], T=[];
function visit(n){
  if(n.name==="Geometry"||n.name==="Mesh"){
    const vn=n.kids.find(k=>k.name==="Vertices");
    const pn=n.kids.find(k=>k.name==="PolygonVertexIndex");
    if(vn && vn.props[0] && vn.props[0].__array){
      const vs=vn.props[0].values, base=V.length/3;
      for(let i=0;i+2<vs.length;i+=3) V.push(vs[i],vs[i+1],vs[i+2]);
      if(pn && pn.props[0] && pn.props[0].__array){
        const pi=pn.props[0].values; let poly=[];
        for(let i=0;i<pi.length;i++){
          let v=pi[i], end=false;
          if(v<0){ v=~v; end=true; }
          poly.push(base+v);
        }
      }
    }
  }
}

```

/*=====

6. MOLECULAR PARSERS

```

===== */
function finishMol(V,atoms,bonds,tol){
  const m=new Model("mol");
  m.verts=new Float32Array(V);
  m.atoms=atoms;
  m.bonds=(bonds && bonds.length) ? bonds : perceiveBonds(m.verts,atoms,tol);
  m.colors=new Uint8Array(atoms.length*3);
  atoms.forEach((a,i)=>{ const c=elemColor(a); m.colors[i*3]=c[0]; m.colors[i*3+1]=c[1]; m.colors[i*3+2]=c[2]; });
  return m;
}

function parseXYZ(buf,opt){
  const lines=asText(buf).split(/\r?\n/);
  const V=[], atoms=[];
  let start=0;
  const fne=lines.findIndex(l=>l.trim()!="");
  if(fne>=0 && /^d+$/ .test(lines[fne].trim())) start=fne+2;
  for(let i=start;i<lines.length;i++){
    const L=lines[i].trim();
    if(!L) continue;
    if(/^d+$/ .test(L) && i+1<lines.length){ i++; continue; } // next frame header
    const p=L.split(/\s+/);
    if(p.length<4) continue;
    const x=num(p[1]),y=num(p[2]),z=num(p[3]);
    if(!Number.isFinite(x)||!Number.isFinite(y)||!Number.isFinite(z)) continue;
    atoms.push(normElement(p[0])); V.push(x,y,z);
  }
  if(!atoms.length) fail("XYZ: no atomic coordinates parsed.");
  return finishMol(V,atoms,null,opt.bondTol);
}

function parsePDB(buf,opt){
  const lines=asText(buf).split(/\r?\n/);
  const V=[], atoms=[], serial=new Map(), bonds=[];
  for(const L of lines){
    const rec=L.slice(0,6);
    if(rec=="ATOM" ||rec=="HETATM"){
      const x=num(L.slice(30,38)), y=num(L.slice(38,46)), z=num(L.slice(46,54));
      if(!Number.isFinite(x)||!Number.isFinite(y)||!Number.isFinite(z)) continue;
      let el=L.slice(76,78).trim();
      if(!el) el=L.slice(12,16).trim().replace(/^[A-Za-z]/g,"").slice(0,1);
      const id=parseInt(L.slice(6,11),10);
      serial.set(id, atoms.length);
      atoms.push(normElement(el)); V.push(x,y,z);
    } else if(rec=="ENDMDL") break;
  }
}

```

```

if(!atoms.length) fail("PDB: no ATOM/HETATM records with parsable coordinates.");
for(const L of lines){
  if(L.slice(0,6)!="CONNECT") continue;
  const a=serial.get(parseInt(L.slice(6,11),10));
  if(a===undefined) continue;
  for(const cs of [11,16,21,26]){
    const b=serial.get(parseInt(L.slice(cs,cs+5),10));
    if(b!==undefined && b>a) bonds.push([a,b]);
  }
}
return finishMol(V,atoms,bonds,opt.bondTol);
}

function parseMOL(buf,opt){
  const lines=asText(buf).split(/\r?\n/);
  if(lines.length<4) fail("MOL/SDF: file is too short to contain a connection table.");
  const counts=lines[3];
  let na=parseInt(counts.slice(0,3),10), nb=parseInt(counts.slice(3,6),10);
  if(!Number.isFinite(na)){
    const p=counts.trim().split(/\s+/);
    na=parseInt(p[0],10); nb=parseInt(p[1],10);
  }
  if(!Number.isFinite(na)||na<=0) fail("MOL/SDF: unreadable atom count on line 4.");
  if(/V3000/i.test(counts)) fail("MOL/SDF: V3000 blocks are not supported (V2000 only).");
  const V=[], atoms=[], bonds=[];
  for(let i=0;i<na;i++){
    const L=lines[4+i]||"";
    let x,y,z,sym;
    if(L.length>=34){
      x=num(L.slice(0,10)); y=num(L.slice(10,20)); z=num(L.slice(20,30)); sym=L.slice(31,34).trim();
    }
    if(!Number.isFinite(x)){
      const p=L.trim().split(/\s+/);
      x=num(p[0]); y=num(p[1]); z=num(p[2]); sym=p[3];
    }
    if(!Number.isFinite(x)) fail("MOL/SDF: bad atom line "+(4+i+1)+".");
    atoms.push(normElement(sym)); V.push(x,y,z);
  }
  for(let i=0;i<(nb||0);i++){
    const L=lines[4+na+i]||"";
    let a=parseInt(L.slice(0,3),10), b=parseInt(L.slice(3,6),10);
    if(!Number.isFinite(a)){ const p=L.trim().split(/\s+/); a=parseInt(p[0],10); b=parseInt(p[1],10); }
    if(Number.isFinite(a)&&Number.isFinite(b)&&a>=1&&b>=1&&a<=na&&b<=na) bonds.push([a-1,b-1]);
  }
  return finishMol(V,atoms,bonds,opt.bondTol);
}

```

```

function parseMOL2(buf,opt){
  const lines=asText(buf).split(/\r?\n/);
  const V=[], atoms=[], bonds=[];
  let sec=null;
  for(const raw of lines){
    const L=raw.trim();
    if(L.startsWith("@<TRIPOS>")){ sec=L.slice(9).toUpperCase(); continue; }
    if(!L) continue;
    if(sec=="ATOM"){
      const p=L.split(/\s+/);
      if(p.length<6) continue;
      const x=num(p[2]),y=num(p[3]),z=num(p[4]);
      if(!Number.isFinite(x)) continue;
      atoms.push(normElement(p[5].split(".")[0])); V.push(x,y,z);
    } else if(sec=="BOND"){
      const p=L.split(/\s+/);
      const a=parseInt(p[1],10), b=parseInt(p[2],10);
      if(Number.isFinite(a)&&Number.isFinite(b)) bonds.push([a-1,b-1]);
    }
  }
  if(!atoms.length) fail("MOL2: no @<TRIPOS>ATOM records parsed.");
  return finishMol(V,atoms,bonds.filter(b=>b[0]>=0&&b[1]>=0&&b[0]<atoms.length&&b[1]<atoms.length),opt.bondTol);
}

function parseCML(buf,opt){
  const doc=new DOMParser().parseFromString(asText(buf),"application/xml");
  if(doc.querySelector("parsererror")) fail("CML: not well-formed XML.");
  const els=[...doc.getElementsByTagName("*")];
  const V=[], atoms=[], idMap=new Map(), bonds=[];
  for(const a of els.filter(e=>e.localName==="atom")){
    let x=a.getAttribute("x3"), y=a.getAttribute("y3"), z=a.getAttribute("z3");
    if(x===null){ x=a.getAttribute("x2"); y=a.getAttribute("y2"); z="0"; }
    if(x===null) continue;
    const el=a.getAttribute("elementType")||a.getAttribute("element")||"C";
    idMap.set(a.getAttribute("id"),atoms.length);
    atoms.push(normElement(el)); V.push(num(x),num(y),num(z||0));
  }
  if(!atoms.length) fail("CML: no <atom> with 3-D (x3/y3/z3) or 2-D coordinates.");
  for(const b of els.filter(e=>e.localName==="bond")){
    const ar=(b.getAttribute("atomRefs2")||b.getAttribute("atomRefs")||"").trim().split(/\s+/);
    if(ar.length>=2){
      const a=idMap.get(ar[0]), c=idMap.get(ar[1]);
      if(a!==undefined && c!==undefined) bonds.push([a,c]);
    }
  }
  return finishMol(V,atoms,bonds,opt.bondTol);
}

```



```

}
function parseCUBE(buf,opt){
  const lines=asText(buf).split(/\r?\n/);
  if(lines.length<7) fail("CUBE: header is incomplete.");
  const B=0.529177210903; // bohr -> angstrom
  const h=i=>lines[i].trim().split(/\s+/).map(num);
  let l3=h(2);
  let na=Math.round(l3[0]);
  const negNA = na<0;          // negative atom count = orbital file
  na=Math.abs(na);
  const origin=[l3[1],l3[2],l3[3]];
  const a1=h(3), a2=h(4), a3=h(5);
  const nx=Math.abs(Math.round(a1[0])), ny=Math.abs(Math.round(a2[0])), nz=Math.abs(Math.round(a3[0]));
  // A negative grid count means the units are already angstrom.
  const unit = a1[0]<0 ? 1 : B;
  const V=[], atoms=[];
  for(let i=0;i<na;i++){
    const p=h(6+i);
    if(p.length<5) fail("CUBE: bad atom line "+(7+i)+".");
    atoms.push(normElement(elementFromZ(Math.round(p[0]))));
    V.push(p[2]*unit,p[3]*unit,p[4]*unit);
  }
  const m=finishMol(V,atoms,null,opt.bondTol);
  // volumetric block
  let start=6+na;
  if(negNA) start++; // orbital count line
  const vals=[];
  const need=nx*ny*nz;
  for(let i=start;i<lines.length && vals.length<need;i++){
    const t=lines[i].trim();
    if(!t) continue;
    for(const s of t.split(/\s+/)){ const v=num(s); if(Number.isFinite(v)) vals.push(v); }
  }
  if(vals.length>=need && need>0){
    m.volume={
      data:Float32Array.from(vals.slice(0,need)),
      nx,ny,nz,
      ox:origin[0]*unit, oy:origin[1]*unit, oz:origin[2]*unit,
      dx:a1[1]*unit, dy:a2[2]*unit, dz:a3[3]*unit,
      order:"xyz"
    };
  }
  return m;
}
const Z_SYM=["","H","He","Li","Be","B","C","N","O","F","Ne","Na","Mg","Al","Si","P","S","Cl","Ar","K","Ca",

```

```

"Sc","Ti","V","Cr","Mn","Fe","Co","Ni","Cu","Zn","Ga","Ge","As","Se","Br","Kr","Rb","Sr","Y","Zr","Nb","Mo",
"Tc","Ru","Rh","Pd","Ag","Cd","In","Sn","Sb","Te","I","Xe","Cs","Ba"];
const elementFromZ = z => Z_SYM[z] || "C";
const zFromElement = s => { const i=Z_SYM.findIndex(x=>x.toLowerCase()===s.toLowerCase()); return i>0?i:6; };

/* =====
7.  VOLUMETRIC + CAD-EXCHANGE PARSERS (point clouds)
===== */
function parseMRC(buf,opt){
  if(buf.byteLength<1024) fail("MRC/CCP4: file is shorter than the 1024-byte header.");
  let dv=new DataView(buf), little=true;
  const plausible=(le)=>{
    const nx=dv.getInt32(0,le), ny=dv.getInt32(4,le), nz=dv.getInt32(8,le), mode=dv.getInt32(12,le);
    return nx>0&&ny>0&&nz>0&&nx<20000&&ny<20000&&nz<20000&&mode>=0&&mode<=6;
  };
  if(!plausible(true)){
    if(plausible(false)) little=false;
    else fail("MRC/CCP4: header does not describe a supported map (nx/ny/nz/mode out of range).");
  }
  const gi=o=>dv.getInt32(o,little), gf=o=>dv.getFloat32(o,little);
  const nx=gi(0), ny=gi(4), nz=gi(8), mode=gi(12);
  const mx=gi(28)||nx, my=gi(32)||ny, mz=gi(36)||nz;
  const xlen=gf(40), ylen=gf(44), zlen=gf(48);
  const nsymbt=gi(92);
  let ox=gf(196), oy=gf(200), oz=gf(204);
  if(!(Number.isFinite(ox)&&Number.isFinite(oy)&&Number.isFinite(oz))){ ox=oy=oz=0; }
  if(ox===0&&oy===0&&oz===0){
    const nxs=gi(16), nys=gi(20), nzs=gi(24);
    ox=nxs*(xlen/mx||1); oy=nys*(ylen/my||1); oz=nzs*(zlen/mz||1);
  }
  const dxs=(xlen/mx)||1, dys=(ylen/my)||1, dzs=(zlen/mz)||1;
  const off=1024+(nsymbt>0?nsymbt:0);
  const n=nx*ny*nz;
  const data=new Float32Array(n);
  const need={0:1,1:2,2:4,6:2}[mode];
  if(need===undefined) fail("MRC/CCP4: data mode "+mode+" is not supported (modes 0, 1, 2 and 6 are).");
  if(off+n*need>buf.byteLength) fail("MRC/CCP4: truncated density block.");
  for(let i=0;i<n;i++){
    const o=off+i*need;
    data[i] = mode===0 ? dv.getInt8(o)
      : mode===1 ? dv.getInt16(o,little)
      : mode===6 ? dv.getUint16(o,little)
      : dv.getFloat32(o,little);
  }
  const vol={data,nx,ny,nz,ox,oy,oz,dx:dxs,dy:dys,dz:dzs};

```

www.iaees.org

```

    if(p.length>=3 && p.every(Number.isFinite)) V.push(p[0],p[1],p[2]);
    else if(p.length===2 && p.every(Number.isFinite)) V.push(p[0],p[1],0);
    else dropped++;
  }
  if(!V.length) fail("STEP: no CARTESIAN_POINT coordinates could be read. "+
    "Note that only the discrete point sampling of a STEP file is supported; B-rep surfaces are not tessellated.");
  const m=new Model("points");
  m.verts=decimate(V,opt.maxPts);
  m.source="STEP discrete sampling: "+(V.length/3)+" CARTESIAN_POINTS"+(dropped?" ("+"dropped"+" malformed)": "");
  return m;
}

function parseIGES(buf,opt){
  const text=asText(buf);
  const lines=text.split(/\r?\n/).filter(l=>l.length>=73);
  if(!lines.length) fail("IGES: no 80-column records found.");
  let pd="";
  for(const L of lines){
    const sec=L[72];
    if(sec==="P") pd += L.slice(0,72);
  }
  if(!pd) fail("IGES: parameter-data (column 73 = 'P') section is empty.");
  const records=pd.split(";").map(s=>s.trim()).filter(Boolean);
  const V=[];
  for(const rec of records){
    const f=rec.split(",").map(s=>s.trim());
    const type=parseInt(f[0],10);
    const g=i=>num(f[i]);
    if(type===116){ // POINT
      V.push(g(1),g(2),g(3));
    } else if(type===110){ // LINE
      V.push(g(1),g(2),g(3), g(4),g(5),g(6));
    } else if(type===126){ // rational B-spline curve: control points
      const K=parseInt(f[1],10), M=parseInt(f[2],10);
      if(Number.isFinite(K)){
        const A=K+M+1, base=7+(A+1)+(K+1);
        for(let i=0;i<=K;i++){
          const o=base+i*3;
          const x=g(o),y=g(o+1),z=g(o+2);
          if([x,y,z].every(Number.isFinite)) V.push(x,y,z);
        }
      }
    } else if(type===128){ // rational B-spline surface: control net
      const K1=parseInt(f[1],10), K2=parseInt(f[2],10),
        M1=parseInt(f[3],10), M2=parseInt(f[4],10);
      if([K1,K2,M1,M2].every(Number.isFinite)){

```

```

    const N1=1+K1-M1, N2=1+K2-M2;
    const A=N1+2*M1, B=N2+2*M2;
    const nCP=(K1+1)*(K2+1);
    const base=10+(A+1)+(B+1)+nCP;    // skip knots and weights
    for(let i=0;i<nCP;i++){
        const o=base+i*3;
        const x=g(o),y=g(o+1),z=g(o+2);
        if([x,y,z].every(Number.isFinite)) V.push(x,y,z);
    }
}
}
const clean=[];
for(let i=0;i+2<V.length;i+=3){
    if(Number.isFinite(V[i])&&Number.isFinite(V[i+1])&&Number.isFinite(V[i+2]))
        clean.push(V[i],V[i+1],V[i+2]);
}
if(!clean.length) fail("IGES: no point-bearing entity (types 110, 116, 126, 128) yielded coordinates.");
const m=new Model("points");
m.verts=decimate(clean,opt.maxPts);
m.source="IGES discrete sampling: "+(clean.length/3)+" points";
return m;
}
function decimate(arr,maxPts){
    const n=arr.length/3;
    if(!maxPts || n<=maxPts) return new Float32Array(arr);
    const stride=Math.ceil(n/maxPts);
    const out=[];
    for(let i=0;i<n;i+=stride) out.push(arr[i*3],arr[i*3+1],arr[i*3+2]);
    return new Float32Array(out);
}

/* =====
8.  Parser dispatch
===== */
const MESH_EXT = ["obj","stl","ply","off","3ds","fbx","gltf","glb","dae","x3d","3mf","dxf","vtk"];
const MOL_EXT  = ["xyz","pdb","ent","mol","sdf","mol2","cml","cube","cub"];
const VOL_EXT   = ["mrc","ccp4","map","step","stp","igs","iges"];

function categoryOf(e){
    if(MESH_EXT.includes(e)) return "mesh";
    if(MOL_EXT.includes(e))  return "mol";
    if(VOL_EXT.includes(e))  return "vol";
    return null;
}

```

```

async function parseAny(name, buf, opt){
  const e=ext(name);
  let m;
  switch(e){
    case "obj": m=parseOBJ(buf); break;
    case "stl": m=parseSTL(buf); break;
    case "ply": m=parsePLY(buf); break;
    case "off": m=parseOFF(buf); break;
    case "vtk": m=parseVTK(buf); break;
    case "3ds": m=parse3DS(buf); break;
    case "fbx": m=await parseFBX(buf); break;
    case "gltf": m=parseGLTF(buf,false); break;
    case "glb": m=parseGLTF(buf,true); break;
    case "dae": m=parseDAE(buf); break;
    case "x3d": m=parseX3D(buf); break;
    case "3mf": m=await parse3MF(buf); break;
    case "dxf": m=parseDXF(buf); break;
    case "xyz": m=parseXYZ(buf,opt); break;
    case "pdb": case "ent": m=parsePDB(buf,opt); break;
    case "mol": case "sdf": m=parseMOL(buf,opt); break;
    case "mol2": m=parseMOL2(buf,opt); break;
    case "cml": m=parseCML(buf,opt); break;
    case "cube": case "cub": m=parseCUBE(buf,opt); break;
    case "mrc": case "ccp4": case "map": m=parseMRC(buf,opt); break;
    case "step": case "stp": m=parseSTEP(buf,opt); break;
    case "igs": case "iges": m=parseIGES(buf,opt); break;
    default: fail("Unsupported extension '"+e+"'");
  }
  if(!m.verts || !m.verts.length) fail("'+e+'": parsing produced no coordinates.");
  for(let i=0;i<m.verts.length;i++){
    if(!Number.isFinite(m.verts[i])) fail("'+e+'": coordinate "+i+" is not a finite number.");
  }
  if(m.tris) for(let i=0;i<m.tris.length;i++){
    const k=m.tris[i];
    if(!(k>=0 && k<m.nVerts)) fail("'+e+'": triangle index "+k+" is out of range (0.."+(m.nVerts-1)+").");
  }
  return m;
}

/* =====
9. WRITERS
===== */

const f6 = v => (Math.abs(v)<1e-12?0:v).toPrecision(7).replace(/\.?0+e/,"e");
const fx = (v,d) => (Math.abs(v)<1e-12?0:v).toFixed(d);

function writeOBJ(m){

```

```

const L=["# Universal 3D Structure Visualizer export"];
for(let i=0;i<m.nVerts;i++)
  L.push("v "+f6(m.verts[i*3])+" "+f6(m.verts[i*3+1])+" "+f6(m.verts[i*3+2]));
if(m.tris) for(let i=0;i<m.tris.length;i+=3)
  L.push("f "+(m.tris[i]+1)+" "+(m.tris[i+1]+1)+" "+(m.tris[i+2]+1));
const ed = m.bonds || m.edges;
if(ed && !m.tris) for(const [a,b] of ed) L.push("l "+(a+1)+" "+(b+1));
return L.join("\n")+"\n";
}

function writeSTLAscii(m){
  if(!m.tris || !m.tris.length) fail("STL requires triangles; this model has none.");
  const L=["solid model"];
  for(let i=0;i<m.tris.length;i+=3){
    const a=m.tris[i]*3,b=m.tris[i+1]*3,c=m.tris[i+2]*3, v=m.verts;
    const ux=v[b]-v[a], uy=v[b+1]-v[a+1], uz=v[b+2]-v[a+2];
    const wx=v[c]-v[a], wy=v[c+1]-v[a+1], wz=v[c+2]-v[a+2];
    let nx=uy*wz-uz*wy, ny=uz*wx-ux*wz, nz=ux*wy-uy*wx;
    const len=Math.hypot(nx,ny,nz)||1; nx/=len; ny/=len; nz/=len;
    L.push("  facet normal "+f6(nx)+" "+f6(ny)+" "+f6(nz));
    L.push("    outer loop");
    for(const o of [a,b,c]) L.push("      vertex "+f6(v[o])+" "+f6(v[o+1])+" "+f6(v[o+2]));
    L.push("    endloop");
    L.push("  endfacet");
  }
  L.push("endsolid model");
  return L.join("\n")+"\n";
}

function writeSTLBinary(m){
  if(!m.tris || !m.tris.length) fail("STL requires triangles; this model has none.");
  const n=m.tris.length/3;
  const out=new Uint8Array(84+50*n), dv=new DataView(out.buffer);
  dv.setUint32(80,n,true);
  let o=84;
  for(let i=0;i<n;i++){
    const a=m.tris[i*3]*3,b=m.tris[i*3+1]*3,c=m.tris[i*3+2]*3, v=m.verts;
    const ux=v[b]-v[a], uy=v[b+1]-v[a+1], uz=v[b+2]-v[a+2];
    const wx=v[c]-v[a], wy=v[c+1]-v[a+1], wz=v[c+2]-v[a+2];
    let nx=uy*wz-uz*wy, ny=uz*wx-ux*wz, nz=ux*wy-uy*wx;
    const len=Math.hypot(nx,ny,nz)||1;
    dv.setFloat32(o,nx/len,true); dv.setFloat32(o+4,ny/len,true); dv.setFloat32(o+8,nz/len,true); o+=12;
    for(const p of [a,b,c]){
      dv.setFloat32(o,v[p],true); dv.setFloat32(o+4,v[p+1],true); dv.setFloat32(o+8,v[p+2],true); o+=12;
    }
    dv.setUint16(o,0,true); o+=2;
  }
}

```

```

    return out;
}
function writePLY(m){
    const n=m.nVerts, t=m.tris?m.tris.length/3:0, hasC=!!m.colors;
    const L=["ply", "format ascii 1.0", "comment Universal 3D Structure Visualizer",
        "element vertex "+n, "property float x", "property float y", "property float z"];
    if(hasC) L.push("property uchar red", "property uchar green", "property uchar blue");
    if(t) L.push("element face "+t, "property list uchar int vertex_indices");
    L.push("end_header");
    for(let i=0; i<n; i++){
        let s=f6(m.verts[i*3])+" "+f6(m.verts[i*3+1])+" "+f6(m.verts[i*3+2]);
        if(hasC) s+=" "+m.colors[i*3]+" "+m.colors[i*3+1]+" "+m.colors[i*3+2];
        L.push(s);
    }
    for(let i=0; i<t; i++) L.push("3 "+m.tris[i*3]+" "+m.tris[i*3+1]+" "+m.tris[i*3+2]);
    return L.join("\n")+"\n";
}
function writeOFF(m){
    const n=m.nVerts, t=m.tris?m.tris.length/3:0;
    const L=["OFF", n+" "+t+" 0"];
    for(let i=0; i<n; i++) L.push(f6(m.verts[i*3])+" "+f6(m.verts[i*3+1])+" "+f6(m.verts[i*3+2]));
    for(let i=0; i<t; i++) L.push("3 "+m.tris[i*3]+" "+m.tris[i*3+1]+" "+m.tris[i*3+2]);
    return L.join("\n")+"\n";
}
function writeVTK(m){
    const n=m.nVerts;
    const L=["# vtk DataFile Version 3.0", "Universal 3D Structure Visualizer", "ASCII", "DATASET POLYDATA",
        "POINTS "+n+" float"];
    for(let i=0; i<n; i++) L.push(f6(m.verts[i*3])+" "+f6(m.verts[i*3+1])+" "+f6(m.verts[i*3+2]));
    if(m.tris && m.tris.length){
        const t=m.tris.length/3;
        L.push("POLYGONS "+t+" "+(4*t));
        for(let i=0; i<t; i++) L.push("3 "+m.tris[i*3]+" "+m.tris[i*3+1]+" "+m.tris[i*3+2]);
    } else {
        const ed=m.bonds||m.edges;
        if(ed && ed.length){
            L.push("LINES "+ed.length+" "+(3*ed.length));
            for(const [a,b] of ed) L.push("2 "+a+" "+b);
        } else {
            L.push("VERTICES 1 "+(n+1));
            L.push(n+" "+Array.from({length:n}, (_,i)=>i).join(" "));
        }
    }
    return L.join("\n")+"\n";
}

```



```

function write3MF(m){
  if(!m.tris || !m.tris.length) fail("3MF requires triangles; this model has none.");
  const enc=new TextEncoder();
  let x='<?xml version="1.0" encoding="UTF-8"?>\n'+
    '<model unit="millimeter" xml:lang="en-US" xmlns="http://schemas.microsoft.com/3dmanufacturing/core/2015/02">\n'+
    '  <resources>\n    <object id="1" type="model">\n      <mesh>\n        <vertices>\n';
  for(let i=0;i<m.nVerts;i++)
    x+='          <vertex x="'+f6(m.verts[i*3])+'" y="'+f6(m.verts[i*3+1])+'" z="'+f6(m.verts[i*3+2])+'" />\n';
  x+='        </vertices>\n      <triangles>\n';
  for(let i=0;i<m.tris.length;i+=3)
    x+='          <triangle v1="'+m.tris[i]+" v2="'+m.tris[i+1]+" v3="'+m.tris[i+2]+" />\n';
  x+='        </triangles>\n      </mesh>\n    </object>\n  </resources>\n'+
    '  <build>\n    <item objectid="1" />\n  </build>\n</model>\n';
  const ct='<?xml version="1.0" encoding="UTF-8"?>\n'+
    '<Types xmlns="http://schemas.openxmlformats.org/package/2006/content-types">'+
    '<Default Extension="rels" ContentType="application/vnd.openxmlformats-package.relationships+xml"/>'+
    '<Default Extension="model" ContentType="application/vnd.ms-package.3dmanufacturing-3dmodel+xml"/></Types>\n';
  const rels='<?xml version="1.0" encoding="UTF-8"?>\n'+
    '<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">'+
    '<Relationship Target="/3D/3dmodel.model" Id="rel0" '+
    'Type="http://schemas.microsoft.com/3dmanufacturing/2013/01/3dmodel"/></Relationships>\n';
  return zipWrite([
    {name:"[Content_Types].xml", data:enc.encode(ct)},
    {name:"_rels/.rels", data:enc.encode(rels)},
    {name:"3D/3dmodel.model", data:enc.encode(x)}
  ]);
}

function writeGLB(m){
  if(!m.tris || !m.tris.length) fail("glTF export requires triangles; this model has none.");
  const nv=m.nVerts, ni=m.tris.length;
  const pos=new Float32Array(m.verts);
  const idx=new Uint32Array(m.tris);
  const pad4=n=>(4-(n%4))%4;
  const posBytes=pos.byteLength, idxBytes=idx.byteLength;
  const posPad=pad4(posBytes);
  const bin=new Uint8Array(posBytes+posPad+idxBytes+pad4(idxBytes));
  bin.set(new Uint8Array(pos.buffer),0);
  bin.set(new Uint8Array(idx.buffer),posBytes+posPad);
  const bb=m.bbox();
  const json={
    asset:{version:"2.0",generator:"Universal 3D Structure Visualizer"},
    scene:0, scenes:[{nodes:[0]}], nodes:[{mesh:0}],
    meshes:[{primitives:[{attributes:{POSITION:0},indices:1,mode:4}]}],
    buffers:[{byteLength:bin.length}],
    bufferViews:[

```

```

    {buffer:0,byteOffset:0,byteLength:posBytes,target:34962},
    {buffer:0,byteOffset:posBytes+posPad,byteLength:idxBytes,target:34963}
  ],
  accessors:[
    {bufferView:0,componentType:5126,count:nv,type:"VEC3",
      min:[bb.xmin,bb.ymin,bb.zmin],max:[bb.xmax,bb.ymax,bb.zmax]},
    {bufferView:1,componentType:5125,count:ni,type:"SCALAR"}
  ]
};
const enc=new TextEncoder();
let jb=enc.encode(JSON.stringify(json));
const jPad=pad4(jb.length);
if(jPad){ const t=new Uint8Array(jb.length+jPad); t.set(jb); t.fill(0x20,jb.length); jb=t; }
const total=12+8+jb.length+8+bin.length;
const out=new Uint8Array(total), dv=new DataView(out.buffer);
dv.setUint32(0,0x46546C67,true); dv.setUint32(4,2,true); dv.setUint32(8,total,true);
dv.setUint32(12,jb.length,true); dv.setUint32(16,0x4E4F534A,true);
out.set(jb,20);
const bo=20+jb.length;
dv.setUint32(bo,bin.length,true); dv.setUint32(bo+4,0x004E4942,true);
out.set(bin,bo+8);
return out;
}
function writeXYZ(m){
  const n=m.nVerts;
  const L=[String(n),"Universal 3D Structure Visualizer export"];
  for(let i=0;i<n;i++){
    const sym=m.atoms ? capital(m.atoms[i]) : "X";
    L.push(sym.padEnd(3)+"
"+fx(m.verts[i*3],6).padStart(14)+fx(m.verts[i*3+1],6).padStart(14)+fx(m.verts[i*3+2],6).padStart(14));
  }
  return L.join("\n")+"\n";
}
const capital = s => s ? s[0].toUpperCase()+s.slice(1) : "X";
function writePDB(m){
  const n=m.nVerts, L=[];
  for(let i=0;i<n;i++){
    const el=capital(m.atoms?m.atoms[i]:"C");
    const nm=(el.length<4?" "+el:el).padEnd(4);
    L.push("HETATM"+String(i+1).padStart(5)+" "+nm+" MOL A    1 "+
fx(m.verts[i*3],3).padStart(8)+fx(m.verts[i*3+1],3).padStart(8)+fx(m.verts[i*3+2],3).padStart(8)+
"  1.00  0.00          "+el.toUpperCase().padStart(2));
  }
  const ed=m.bonds||m.edges;
  if(ed) for(const [a,b] of ed)

```

```

    L.push("CONNECT"+String(a+1).padStart(5)+String(b+1).padStart(5));
    L.push("END");
    return L.join("\n")+"\n";
}

function writeMOL(m){
    const n=m.nVerts, bonds=(m.bonds||m.edges||[]);
    if(n>999 || bonds.length>999) fail("MOL V2000 cannot hold more than 999 atoms or bonds (this model has "+n+" /
"+bonds.length+".");
    const L=["Universal3DVisualizer", " V3D      ", ""];
    L.push(String(n).padStart(3)+String(bonds.length).padStart(3)+" 0 0 0 0 0 0 0 0 0999 V2000");
    for(let i=0;i<n;i++){
        L.push(fx(m.verts[i*3],4).padStart(10)+fx(m.verts[i*3+1],4).padStart(10)+fx(m.verts[i*3+2],4).padStart(10)+
            " "+capital(m.atoms?m.atoms[i]:"C").padEnd(3)+" 0 0 0 0 0 0 0 0 0 0 0 0 0");
    }
    for(const [a,b] of bonds)
        L.push(String(a+1).padStart(3)+String(b+1).padStart(3)+" 1 0 0 0 0");
    L.push("M END");
    return L.join("\n")+"\n";
}

function writeSDF(m){ return writeMOL(m)+"$$$$\n"; }

function writeMOL2(m){
    const n=m.nVerts, bonds=(m.bonds||m.edges||[]);
    const L["@<TRIPOS>MOLECULE", "MODEL", n+" "+bonds.length+" 0 0
0", "SMALL", "NO_CHARGES", "", "@<TRIPOS>ATOM"];
    for(let i=0;i<n;i++){
        const el=capital(m.atoms?m.atoms[i]:"C");
        L.push(String(i+1).padStart(7)+" "+(el+(i+1)).padEnd(8)+
            fx(m.verts[i*3],4).padStart(10)+fx(m.verts[i*3+1],4).padStart(10)+fx(m.verts[i*3+2],4).padStart(10)+
            " "+el.padEnd(6)+" 1 MOL1      0.0000");
    }
    L.push("@<TRIPOS>BOND");
    bonds.forEach((b,i)=>L.push(String(i+1).padStart(6)+String(b[0]+1).padStart(6)+String(b[1]+1).padStart(6)+" 1"));
    return L.join("\n")+"\n";
}

function writeCML(m){
    const n=m.nVerts, bonds=(m.bonds||m.edges||[]);
    let x='<?xml version="1.0"?>\n<molecule xmlns="http://www.xml-cml.org/schema" id="m1">\n <atomArray>\n';
    for(let i=0;i<n;i++){
        x+='  <atom id="a'+(i+1)+'" elementType="'+capital(m.atoms?m.atoms[i]:"C")+'
        "' x3="'+f6(m.verts[i*3])+'"' y3="'+f6(m.verts[i*3+1])+'"' z3="'+f6(m.verts[i*3+2])+'"/>\n';
    }
    x+=' </atomArray>\n';
    if(bonds.length){
        x+=' <bondArray>\n';
        for(const [a,b] of bonds) x+='  <bond atomRefs2="a'+(a+1)+' a'+(b+1)+'" order="1"/>\n';
        x+=' </bondArray>\n';
    }
}

```

```

    }
    return x+'</molecule>\n';
}
function writeX3D(m){
    const n=m.nVerts;
    let pts=[];
    for(let i=0;i<n;i++) pts.push(f6(m.verts[i*3])+ " "+f6(m.verts[i*3+1])+ " "+f6(m.verts[i*3+2]));
    let geo;
    if(m.tris && m.tris.length){
        const ci=[];
        for(let i=0;i<m.tris.length;i+=3) ci.push(m.tris[i]+" "+m.tris[i+1]+" "+m.tris[i+2]+" -1");
        geo='<IndexedFaceSet solid="false" coordIndex="'+ci.join(" ")+" "><Coordinate point="'+pts.join("
")+'"/></IndexedFaceSet>';
    } else {
        geo='<PointSet><Coordinate point="'+pts.join(" ")+'"/></PointSet>';
    }
    return '<?xml version="1.0" encoding="UTF-8"?>\n<X3D profile="Interchange" version="3.3">\n <Scene>\n'+
        ' <Shape>\n    '+geo+'\n </Shape>\n </Scene>\n</X3D>\n';
}

/* which conversions are legal for a given model */
function conversionTargets(m){
    const t=[];
    const hasTris=!!(m.tris && m.tris.length);
    if(hasTris) t.push("obj", "stl", "ply", "off", "vtk", "3mf", "glb", "x3d");
    else t.push("obj", "ply", "vtk", "x3d");
    if(m.atoms) t.push("xyz", "pdb", "mol", "sdf", "mol2", "cml");
    else t.push("xyz", "pdb");
    return [...new Set(t)];
}

function convert(m,target){
    switch(target){
        case "obj": return {data:writeOBJ(m), mime:"text/plain", ext:"obj"};
        case "stl": return {data:writeSTLBinary(m), mime:"model/stl", ext:"stl"};
        case "stla": return {data:writeSTLAscii(m), mime:"model/stl", ext:"stl"};
        case "ply": return {data:writePLY(m), mime:"text/plain", ext:"ply"};
        case "off": return {data:writeOFF(m), mime:"text/plain", ext:"off"};
        case "vtk": return {data:writeVTK(m), mime:"text/plain", ext:"vtk"};
        case "3mf": return {data:write3MF(m), mime:"model/3mf", ext:"3mf"};
        case "glb": return {data:writeGLB(m), mime:"model/gltf-binary", ext:"glb"};
        case "x3d": return {data:writeX3D(m), mime:"model/x3d+xml", ext:"x3d"};
        case "xyz": return {data:writeXYZ(m), mime:"text/plain", ext:"xyz"};
        case "pdb": return {data:writePDB(m), mime:"chemical/x-pdb", ext:"pdb"};
        case "mol": return {data:writeMOL(m), mime:"chemical/x-mdl-molfile", ext:"mol"};
        case "sdf": return {data:writeSDF(m), mime:"chemical/x-mdl-sdfile", ext:"sdf"};
    }
}

```

```

    case "mol2": return {data:writeMOL2(m), mime:"chemical/x-mol2", ext:"mol2"};
    case "cml":  return {data:writeCML(m), mime:"chemical/x-cml", ext:"cml"};
    default: fail("No writer for target '"+target+"'.");
  }
}

const TARGET_LABEL={obj:"Wavefront OBJ",stl:"STL (binary)",stla:"STL (ASCII)",ply:"PLY (ASCII)",
  off:"OFF",vtk:"VTK legacy (ASCII)","3mf":"3MF",glb:"glTF binary (.glb)",x3d:"X3D",
  xyz:"XYZ",pdb:"PDB",mol:"MDL MOL (V2000)",sdf:"SDF",mol2:"Tripos MOL2",cml:"CML"};

/* =====
10.  Viewer state and rendering
===== */

const state={
  model:null, name:"", amat:null, xfac:1,
  scale:2, dila:0.3, scrollmax:2000, cw:900, ch:640,
  ballSize:10, bondTol:0.45, iso:NaN, maxPts:120000, style:"auto"
};

const canvas=document.getElementById("glcanvas");
const ctx=canvas.getContext("2d");
const hScroll=document.getElementById("hScroll");
const vScroll=document.getElementById("vScroll");
let prevH=0, prevV=0;

function setupView(){
  canvas.width=state.cw; canvas.height=state.ch;
  const bb=state.model.bbox();
  let f=Math.max(bb.xmax-bb.xmin, bb.ymax-bb.ymin, bb.zmax-bb.zmin);
  if(!(f>0)) f=1;
  state.xfac=0.7*Math.min(canvas.width/f, canvas.height/f)*state.scale;
  state.amat=new Mat3D();
  state.amat.xrot(20); state.amat.yrot(20);
  const mid=Math.round(state.scrollmax/2);
  hScroll.min=0; hScroll.max=state.scrollmax; hScroll.value=mid; prevH=mid;
  vScroll.min=0; vScroll.max=state.scrollmax; vScroll.value=mid; prevV=mid;
  document.getElementById("stage").style.gridTemplateRows=canvas.height+"px 24px";
}

function effectiveStyle(){
  if(state.style!=="auto") return state.style;
  const m=state.model;
  if(m.kind==="mol") return "ballstick";
  if(m.kind==="points") return "points";
  return (m.tris && m.tris.length) ? "flat" : "wire";
}

const GRAY=(()=>{
  const a=[];

```

```

    for(let i=0;i<16;i++){ const k=Math.round(170*(1-Math.pow(i/15,2.3))); a.push("rgb("+k+","+k+","+k+")"); }
    return a;
  })();

function render(){
  ctx.fillStyle="#ffffff";
  ctx.fillRect(0,0,canvas.width,canvas.height);
  const m=state.model;
  if(!m) return;
  const bb=m.bbox();
  const mat=new Mat3D();
  mat.translate(-(bb.xmin+bb.xmax)/2, -(bb.ymin+bb.ymax)/2, -(bb.zmin+bb.zmax)/2);
  mat.mult(state.amat);
  mat.scale(state.xfac, -state.xfac, 16*state.xfac/canvas.width);
  mat.translate(canvas.width/2, canvas.height/2, 8);
  const n=m.nVerts;
  const tv=mat.transform(m.verts,n);
  const st=effectiveStyle();
  if(st=="flat" && m.tris && m.tris.length) drawFlat(m,tv);
  else if(st=="ballstick") drawBallStick(m,tv);
  else if(st=="points") drawPoints(m,tv);
  else drawWire(m,tv);
}

function drawWire(m, tv){
  const edges = m.wireEdges();
  ctx.lineWidth = 1;
  for (const [a, b] of edges) {
    ctx.strokeStyle = "#000000";
    ctx.beginPath();
    ctx.moveTo(tv[a * 3], tv[a * 3 + 1]);
    ctx.lineTo(tv[b * 3], tv[b * 3 + 1]);
    ctx.stroke();
  }
}

function drawFlat(m,tv){
  const T=m.tris, nt=T.length/3;
  const order=new Int32Array(nt);
  const depth=new Float32Array(nt);
  for(let i=0;i<nt;i++){
    order[i]=i;
    depth[i]=(tv[T[i*3]*3+2]+tv[T[i*3+1]*3+2]+tv[T[i*3+2]*3+2])/3;
  }
  const ord=Array.from(order).sort((a,b)=>depth[b]-depth[a]); // far first
  const A=state.amat;
  const V=m.verts;

```

```

for(const i of ord){
  const i0=T[i*3], i1=T[i*3+1], i2=T[i*3+2];
  // object-space normal rotated by the (rotation-only) view matrix
  const ux=V[i1*3]-V[i0*3], uy=V[i1*3+1]-V[i0*3+1], uz=V[i1*3+2]-V[i0*3+2];
  const wx=V[i2*3]-V[i0*3], wy=V[i2*3+1]-V[i0*3+1], wz=V[i2*3+2]-V[i0*3+2];
  let nx=uy*wz-uz*wy, ny=uz*wx-ux*wz, nz=ux*wy-uy*wx;
  const ln=Math.hypot(nx,ny,nz);
  if(ln<1e-20) continue;
  nx/=ln; ny/=ln; nz/=ln;
  const rz = nx*A.zx+ny*A.zy+nz*A.zz;    // view-space z of the normal
  const lam = Math.min(1, 0.25+0.75*Math.abs(rz));
  let cr=200,cg=205,cb=215;
  if(m.colors){
    cr=(m.colors[i0*3]+m.colors[i1*3]+m.colors[i2*3])/3;
    cg=(m.colors[i0*3+1]+m.colors[i1*3+1]+m.colors[i2*3+1])/3;
    cb=(m.colors[i0*3+2]+m.colors[i1*3+2]+m.colors[i2*3+2])/3;
  }
  ctx.fillStyle="rgb("+Math.round(cr*lam)+","+Math.round(cg*lam)+","+Math.round(cb*lam)+")";
  ctx.beginPath();
  ctx.moveTo(tv[i0*3],tv[i0*3+1]);
  ctx.lineTo(tv[i1*3],tv[i1*3+1]);
  ctx.lineTo(tv[i2*3],tv[i2*3+1]);
  ctx.closePath();
  ctx.fill();
  ctx.strokeStyle=ctx.fillStyle;
  ctx.lineWidth=0.6;
  ctx.stroke();
}
}

function drawPoints(m,tv){
  const n=m.nVerts;
  const idx=Array.from({length:n},(_i)=>i).sort((a,b)=>tv[b*3+2]-tv[a*3+2]);
  const r=Math.max(1,Math.min(3,Math.round(state.xfac/40)));
  for(const i of idx){
    let d=tv[i*3+2]; d=d<0?0:(d>15?15:d);
    const fog=1-0.55*(d/15);
    let cr=60,cg=110,cb=200;
    if(m.colors){ cr=m.colors[i*3]; cg=m.colors[i*3+1]; cb=m.colors[i*3+2]; }
    ctx.fillStyle="rgb("+Math.round(cr*fog+235*(1-fog))+","+
      Math.round(cg*fog+235*(1-fog))+","+
      Math.round(cb*fog+235*(1-fog))+")";
    ctx.fillRect(tv[i*3]-r/2, tv[i*3+1]-r/2, r, r);
  }
}

function drawBallStick(m,tv){

```

```

const n=m.nVerts;
// sticks first, then depth-sorted balls
const bonds=m.bonds||[];
ctx.lineCap="round";
for(const [a,b] of bonds){
  if(a>=n||b>=n) continue;
  let d=(tv[a*3+2]+tv[b*3+2])/2; d=d<0?0:(d>15?15:d);
  const g=Math.round(90+90*(d/15));
  ctx.strokeStyle="rgb("+g+", "+g+", "+g+")";
  ctx.lineWidth=Math.max(1.2, state.xfac*0.035);
  ctx.beginPath();
  ctx.moveTo(tv[a*3],tv[a*3+1]);
  ctx.lineTo(tv[b*3],tv[b*3+1]);
  ctx.stroke();
}
const idx=Array.from({length:n},(_i,i)=>i).sort((a,b)=>tv[b*3+2]-tv[a*3+2]);
for(const i of idx){
  let d=tv[i*3+2]; d=d<0?0:(d>15?15:d);
  const sym=m.atoms?m.atoms[i]:"c";
  const base=elemColor(sym);
  const rad=Math.max(2, state.ballSize*0.5*(0.5+elemRad(sym))*(state.xfac/(state.xfac||1)) * (state.xfac/60+0.4));
  const fog=(d/15)*0.45;
  const c=base.map(v=>Math.round(v*(1-fog)+215*fog));
  const x=tv[i*3], y=tv[i*3+1];
  const g=ctx.createRadialGradient(x-rad*0.35,y-rad*0.35,rad*0.05,x,y,rad);
  g.addColorStop(0,"rgb(255,255,255)");
  g.addColorStop(0.35,"rgb("+c[0]+", "+c[1]+", "+c[2]+")");
  g.addColorStop(1,"rgb("+Math.round(c[0]*0.32)+", "+Math.round(c[1]*0.32)+", "+Math.round(c[2]*0.32)+")");
  ctx.fillStyle=g;
  ctx.beginPath();
  ctx.arc(x,y,rad,0,Math.PI*2);
  ctx.fill();
}
}

/* ---- interaction ---- */
let dragging=false,lastX=0,lastY=0,downX=0,downY=0,moved=false;
canvas.addEventListener("contextmenu",e=>e.preventDefault());
canvas.addEventListener("mousedown",e=>{
  const r=canvas.getBoundingClientRect();
  dragging=true; moved=false;
  lastX=downX=e.clientX-r.left; lastY=downY=e.clientY-r.top;
});
window.addEventListener("mousemove",e=>{
  if(!dragging||!state.amat) return;

```



```

const r=canvas.getBoundingClientRect();
const x=e.clientX-r.left, y=e.clientY-r.top;
if(Math.abs(x-downX)+Math.abs(y-downY)>3) moved=true;
const t=new Mat3D();
t.xrot((lastY-y)*(360/canvas.height));
t.yrot((x-lastX)*(360/canvas.width));
state.amat.mult(t);
lastX=x; lastY=y;
render();
});
window.addEventListener("mouseup",e=>{
  if(!dragging) return;
  dragging=false;
  if(!state.amat||moved) return;
  if(e.button===0) zoomIn(); else zoomOut();
});
canvas.addEventListener("wheel",e=>{
  if(!state.amat) return;
  e.preventDefault();
  if(e.deltaY<0) zoomIn(); else zoomOut();
},{passive:false});
function zoomIn(){ state.xfac*=1+state.dila; render(); }
function zoomOut(){ state.xfac=Math.max(0.01,state.xfac*(1-state.dila)); render(); }
hScroll.addEventListener("input",()=>{
  if(!state.amat) return;
  const p+=hScroll.value, d=p-prevH; prevH=p;
  state.amat.translate(-d,0,0); render();
});
vScroll.addEventListener("input",()=>{
  if(!state.amat) return;
  const p+=vScroll.value, d=p-prevV; prevV=p;
  state.amat.translate(0,d,0); render();
});

/* =====
11.  UI wiring
===== */

const fileInput=document.getElementById("fileInput");
const fileTypeMsg=document.getElementById("fileTypeMsg");
const statusEl=document.getElementById("status");
const testOut=document.getElementById("testOut");

fileInput.addEventListener("change",()=>{
  const f=fileInput.files[0];
  if(!f){ fileTypeMsg.textContent=""; return; }

```

```

const e=ext(f.name), cat=categoryOf(e);
const msg={ mesh:"mesh / solid file → wireframe or flat-shaded surface",
            mol:"molecular file → ball-and-stick model",
            vol:"volumetric / CAD-exchange file → point cloud"}[cat];
if(cat){ fileTypeMsg.textContent="."+e+" detected: "+msg+"."; fileTypeMsg.style.color="#0a6"; }
else{ fileTypeMsg.textContent="Unsupported extension '."+e+"'."; fileTypeMsg.style.color="#c00"; }
});

```

```

function readOpts(){
  state.scale=parseFloat(document.getElementById("scaleInput").value)||2;
  state.dila=parseFloat(document.getElementById("dilaInput").value)||0.3;
  if(state.dila<=0||state.dila>=1) state.dila=0.3;
  state.scrollmax=parseInt(document.getElementById("scrollMaxInput").value,10)||2000;
  state.cw=parseInt(document.getElementById("canvasWInput").value,10)||900;
  state.ch=parseInt(document.getElementById("canvasHInput").value,10)||640;
  state.ballSize=parseFloat(document.getElementById("ballSizeInput").value)||10;
  state.bondTol=parseFloat(document.getElementById("bondTolInput").value);
  if(!Number.isFinite(state.bondTol)) state.bondTol=0.45;
  const isoRaw=document.getElementById("isoInput").value.trim();
  state.iso=isoRaw===""?NaN:parseFloat(isoRaw);
  state.maxPts=parseInt(document.getElementById("maxPtsInput").value,10)||120000;
  return {bondTol:state.bondTol, iso:state.iso, maxPts:state.maxPts};
}

```

```

document.getElementById("loadBtn").addEventListener("click",async ()=>{
  const f=fileInput.files[0];
  if(!f){ alert("Please choose a structure data file first."); return; }
  if(!categoryOf(ext(f.name))){ alert("Unsupported extension '."+ext(f.name)+"'."); return; }
  const opt=readOpts();
  statusEl.textContent="Parsing "+f.name+" ...";
  try{
    const buf=await f.arrayBuffer();
    const m=await parseAny(f.name,buf,opt);
    state.model=m; state.name=f.name;
    state.style="auto";
    document.getElementById("styleSel").value="auto";
    fillConvSelect();
    setupView();
    document.getElementById("settingsPanel").style.display="none";
    document.getElementById("viewerPanel").style.display="block";
    updateLegend(); updateInfo();
    render();
    statusEl.textContent="";
  } catch(err){
    statusEl.textContent="";
  }
}

```

```

        alert("Could not read "+f.name+":\n"+err.message);
    }
});
function fillConvSelect(){
    const sel=document.getElementById("convSel");
    sel.innerHTML="";
    const t=conversionTargets(state.model);
    if(state.model.tris && state.model.tris.length) t.splice(t.indexOf("stl")+1,0,"stla");
    for(const k of t){
        const o=document.createElement("option");
        o.value=k; o.textContent=TARGET_LABEL[k]||k;
        sel.appendChild(o);
    }
}
document.getElementById("styleSel").addEventListener("change",e=>{
    state.style=e.target.value; render();
});
document.getElementById("convBtn").addEventListener("click",()=>{
    const target=document.getElementById("convSel").value;
    try{
        const {data,mime,ext:oe}=convert(state.model,target);
        const blob=new Blob([data],{type:mime});
        const base=state.name.replace(/\.[^.]*/,"")||"model";
        downloadBlob(blob, base+"_converted."+oe);
    }catch(err){
        alert("Conversion failed:\n"+err.message);
    }
});
function downloadBlob(blob,name){
    const url=URL.createObjectURL(blob);
    const a=document.createElement("a");
    a.href=url; a.download=name;
    document.body.appendChild(a); a.click(); a.remove();
    setTimeout(()=>URL.revokeObjectURL(url),1000);
}
document.getElementById("saveBtn").addEventListener("click",()=>{
    canvas.toBlob(b=>downloadBlob(b,(state.name.replace(/\.[^.]*/,"")||"view")+".png"),"image/png");
});
document.getElementById("resetBtn").addEventListener("click",()=>{ if(state.model){ setupView(); render(); } });
document.getElementById("closeBtn").addEventListener("click",()=>{
    document.getElementById("viewerPanel").style.display="none";
    document.getElementById("settingsPanel").style.display="block";
});
function updateLegend(){
    const el=document.getElementById("legend");

```

```

const m=state.model;
if(!m || m.kind!=="mol"){ el.innerHTML=""; return; }
const present=[...new Set(m.atoms)].sort();
let html="Element colours: ";
for(const s of present){
  const c=elemColor(s);
  html+="

```

```

const out=new Uint8Array(1024+n*4), dv=new DataView(out.buffer);
dv.setInt32(0,nx,true); dv.setInt32(4,ny,true); dv.setInt32(8,nz,true); dv.setInt32(12,2,true);
dv.setInt32(28,nx,true); dv.setInt32(32,ny,true); dv.setInt32(36,nz,true);
dv.setFloat32(40,nx,true); dv.setFloat32(44,ny,true); dv.setFloat32(48,nz,true);
dv.setInt32(92,0,true);
for(let k=0;k<nz;k++)for(let j=0;j<ny;j++)for(let i=0;i<nx;i++){
    const dx=i-nx/2, dy=j-ny/2, dz=k-nz/2;
    const v=Math.exp(-(dx*dx+dy*dy+dz*dz)/8);
    dv.setFloat32(1024+((k*ny+j)*nx+i)*4, v, true);
}
return out;
}
function sampleSTEP(){
    return "ISO-10303-21;\nHEADER;\nENDSEC;\nDATA;\n"+
        "#1=CARTESIAN_POINT(",(0,0,0));\n#2=CARTESIAN_POINT(",(1,0,0));\n"+
        "#3=CARTESIAN_POINT(",(1,1,0));\n#4=CARTESIAN_POINT(",(0,1,2.5));\n"+
        "ENDSEC;\nEND-ISO-10303-21;\n";
}
function sampleIGES(){
    const pad=(s,n)=>s.length>=n?s.slice(0,n):s+" ".repeat(n-s.length);
    const rec=(body,sec,i)=>pad(body,72)+sec+String(i).padStart(7);
    const L=[];
    L.push(rec("Self-test IGES", "S",1));
    L.push(rec("1H,,1H,,4Htest,,,,,,,,", "G",1));
    L.push(rec("    116    1", "D",1));
    L.push(rec("    116    0", "D",2));
    L.push(rec("116,0.0,0.0,0.0,0;", "P",1));
    L.push(rec("116,3.0,1.0,-2.0,0;", "P",2));
    L.push(rec("110,0.,0.,0.,1.,2.,3.,", "P",3));
    L.push(rec("S    1G    1D    2P    3", "T",1));
    return L.join("\n")+"\n";
}
function sampleDXF(){
    const p=[];
    const g=(c,v)=>{ p.push(String(c)); p.push(String(v)); };
    g(0,"SECTION"); g(2,"ENTITIES");
    g(0,"3DFACE"); g(10,0); g(20,0); g(30,0); g(11,1); g(21,0); g(31,0);
        g(12,1); g(22,1); g(32,0); g(13,1); g(23,1); g(33,0);
    g(0,"LINE"); g(10,0); g(20,0); g(30,0); g(11,0); g(21,0); g(31,2);
    g(0,"ENDSEC"); g(0,"EOF");
    return p.join("\n")+"\n";
}
function sampleDAE(){
    return '<?xml version="1.0"?>\n<COLLADA xmlns="http://www.collada.org/2005/11/COLLADASchema"
version="1.4.1">'+

```

```

'<library_geometries><geometry id="g"><mesh>'+
'<source id="pos"><float_array id="pa" count="12">0 0 0 1 0 0 1 1 0 0 1 0</float_array>'+
'<technique_common><accessor source="#pa" count="4" stride="3">'+
'<param name="X" type="float"/><param name="Y" type="float"/><param name="Z" type="float"/>'+
'</accessor></technique_common></source>'+
'<vertices id="v"><input semantic="POSITION" source="#pos"/></vertices>'+
'<triangles count="2"><input semantic="VERTEX" source="#v" offset="0"/>'+
'<p>0 1 2 0 2 3</p></triangles></mesh></geometry></library_geometries></COLLADA>';
}

function sample3DS(){
  const nv=4, nf=2;
  const vLen=6+2+nv*12, fLen=6+2+nf*8;
  const objName="cube\0";
  const triLen=6+vLen+fLen;
  const objLen=6+objName.length+triLen;
  const editLen=6+objLen;
  const total=6+editLen;
  const out=new Uint8Array(total), dv=new DataView(out.buffer);
  let o=0;
  const chunk=(id,len)=>{ dv.setUint16(o,id,true); dv.setUint32(o+2,len,true); o+=6; };
  chunk(0x4D4D,total);
  chunk(0x3D3D,editLen);
  chunk(0x4000,objLen);
  for(const ch of objName) out[o++]=ch.charCodeAt(0);
  chunk(0x4100,triLen);
  chunk(0x4110,vLen);
  dv.setUint16(o,nv,true); o+=2;
  const V=[0,0,0, 1,0,0, 1,1,0, 0,1,0];
  for(let i=0;i<V.length;i++){ dv.setFloat32(o,V[i],true); o+=4; }
  chunk(0x4120,fLen);
  dv.setUint16(o,nf,true); o+=2;
  const F=[[0,1,2],[0,2,3]];
  for(const f of F){
    dv.setUint16(o,f[0],true); dv.setUint16(o+2,f[1],true); dv.setUint16(o+4,f[2],true);
    dv.setUint16(o+6,7,true); o+=8;
  }
  return out;
}

const enc=new TextEncoder();
const toBuf = d => (typeof d=="string" ? enc.encode(d) : d);

async function runSelfTest(){
  const lines=[];
  const log=s=>lines.push(s);
  let pass=0, fail_=0;

```

```

const opt={bondTol:0.45, iso:NaN, maxPts:120000};
const cube=sampleCube(), mol=sampleMol();

const cases=[
  ["cube.obj", writeOBJ(cube),      {tris:12,verts:8}],
  ["cube.stl", writeSTLBinary(cube), {tris:12}],
  ["cube.stl", writeSTLAscii(cube),  {tris:12, "stl-ascii"}],
  ["cube.ply", writePLY(cube),      {tris:12,verts:8}],
  ["cube.off", writeOFF(cube),      {tris:12,verts:8}],
  ["cube.vtk", writeVTK(cube),      {tris:12,verts:8}],
  ["cube.3mf", write3MF(cube),      {tris:12,verts:8}],
  ["cube.glb", writeGLB(cube),      {tris:12,verts:8}],
  ["cube.x3d", writeX3D(cube),      {tris:12,verts:8}],
  ["cube.3ds", sample3DS(),         {tris:2,verts:4}],
  ["quad.dae", sampleDAE(),         {tris:2,verts:4}],
  ["shapes.dxf",sampleDXF(),         {tris:1}],
  ["ch4.xyz",  writeXYZ(mol),       {verts:5}],
  ["ch4.pdb",  writePDB(mol),       {verts:5}],
  ["ch4.mol",  writeMOL(mol),       {verts:5}],
  ["ch4.sdf",  writeSDF(mol),       {verts:5}],
  ["ch4.mol2", writeMOL2(mol),      {verts:5}],
  ["ch4.cml",  writeCML(mol),       {verts:5}],
  ["map.mrc",  sampleVolumeMRC(),   {min:1}],
  ["part.step", sampleSTEP(),       {verts:4}],
  ["part.iges", sampleIGES(),       {min:2}]
];

log("=== PARSE TESTS ===");
const parsed=[];
for(const [name,data,expect,label] of cases){
  const tag=(label||name).padEnd(12);
  try{
    const buf=toBuf(data);
    const m=await parseAny(name, buf.buffer ? buf.buffer.slice(buf.byteOffset,buf.byteOffset+buf.byteLength) : buf, opt);
    let ok=true, why=[];
    if(expect.verts!==undefined && m.nVerts!==expect.verts){ ok=false; why.push("verts "+m.nVerts+" != "+expect.verts); }
    if(expect.tris!==undefined && m.nTris!==expect.tris){ ok=false; why.push("tris "+m.nTris+" != "+expect.tris); }
    if(expect.min!==undefined && m.nVerts<expect.min){ ok=false; why.push("verts "+m.nVerts+" < "+expect.min); }
    if(ok){ pass++; log(" PASS "+tag+" kind="+m.kind.padEnd(6)+" v="+m.nVerts+" t="+m.nTris);
    parsed.push([name,m]); }
    else { fail_++; log(" FAIL "+tag+" "+why.join(" "); }
  }catch(e){
    fail_++; log(" FAIL "+tag+" threw: "+e.message);
  }
}

```

```

}

log("");
log("=== ROUND-TRIP CONVERSION TESTS (bbox delta and counts) ===");
for(const [name,m] of parsed){
  const targets=conversionTargets(m);
  for(const t of targets){
    const tag=(ext(name)+" -> "+t).padEnd(18);
    try{
      const {data,ext:oe}=convert(m,t);
      const buf=toBuf(data);
      const ab=buf.buffer ? buf.buffer.slice(buf.byteOffset,buf.byteOffset+buf.byteLength) : buf;
      const back=await parseAny("rt."+oe, ab, opt);
      const A=m.bbox(), B=back.bbox();
      const d=Math.max(Math.abs(A.xmin-B.xmin),Math.abs(A.xmax-B.xmax),
        Math.abs(A.ymin-B.ymin),Math.abs(A.ymax-B.ymax),
        Math.abs(A.zmin-B.zmin),Math.abs(A.zmax-B.zmax));
      const scale=Math.max(1e-9,
Math.max(Math.abs(A.xmax-A.xmin),Math.abs(A.ymax-A.ymin),Math.abs(A.zmax-A.zmin)));
      const rel=d/scale;
      const problems=[];
      if(rel>1e-4) problems.push("bbox drift "+rel.toExponential(2));
      // triangle count must survive any mesh-preserving target
      const meshTarget=["obj","stl","stla","ply","off","vtk","3mf","glb","x3d"].includes(t);
      if(meshTarget && m.nTris && back.nTris!==m.nTris)
        problems.push("tris "+back.nTris+" != "+m.nTris);
      const molTarget=["xyz","pdb","mol","sdf","mol2","cml"].includes(t);
      if(molTarget && back.nVerts!==m.nVerts)
        problems.push("atoms "+back.nVerts+" != "+m.nVerts);
      if(molTarget && m.atoms && back.atoms){
        const bad=m.atoms.findIndex((s,i)=>s!==back.atoms[i]);
        if(bad>=0) problems.push("element["+bad+"] "+m.atoms[bad]+" != "+back.atoms[bad]);
      }
      if(problems.length){ fail_++; log("  FAIL  "+tag+" "+problems.join("; ")); }
      else { pass++; log("  PASS  "+tag+" v="+back.nVerts+" t="+back.nTris+" maxΔ="+d.toExponential(2)); }
    }catch(e){
      fail_++; log("  FAIL  "+tag+" threw: "+e.message);
    }
  }
}

log("");
log("=== NEGATIVE TESTS (must raise a clear error) ===");
const neg=[
  ["broken.obj","this file has no vertices\n"],

```



```

["broken.off","OFF\nnot numbers here\n"],
["broken.xyz","hello world\n"],
["broken.3mf","not a zip at all"],
["broken.glb","not a glb"]
];
for(const [n,d] of neg){
  const buf=toBuf(d);
  try{
    await parseAny(n, buf.buffer.slice(buf.byteOffset,buf.byteOffset+buf.byteLength), opt);
    fail_++; log("  FAIL  "+n.padEnd(12)+" accepted invalid input");
  }catch(e){
    pass++; log("  PASS  "+n.padEnd(12)+" rejected: "+e.message.slice(0,72));
  }
}

log("");
log("=== SUMMARY: "+pass+" passed, "+fail_+" failed ===");
if(!fail_) log("All checks passed.");
return lines.join("\n");
}

document.getElementById("selfTestBtn").addEventListener("click",async ()=>{
  testOut.style.display="block";
  testOut.textContent="Running ...";
  try{ testOut.textContent=await runSelfTest(); }
  catch(e){ testOut.textContent="Self-test harness error: "+e.message+"\n"+e.stack; }
});
</script>
</body>
</html>

```

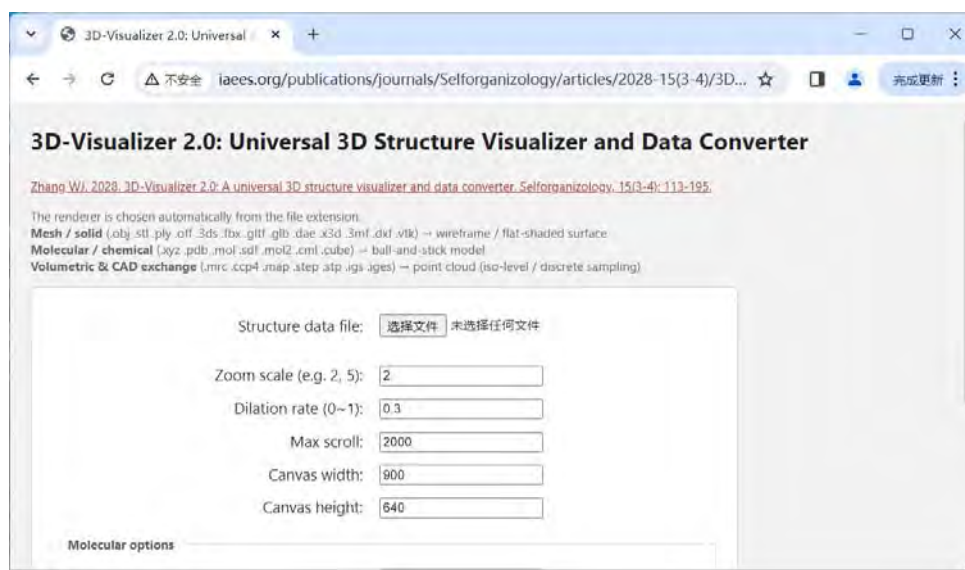


Fig. 1 3D-Visualizer 2.0.

4 Demo

Fig.2 shows the 3D visualization of an OBJ structure.

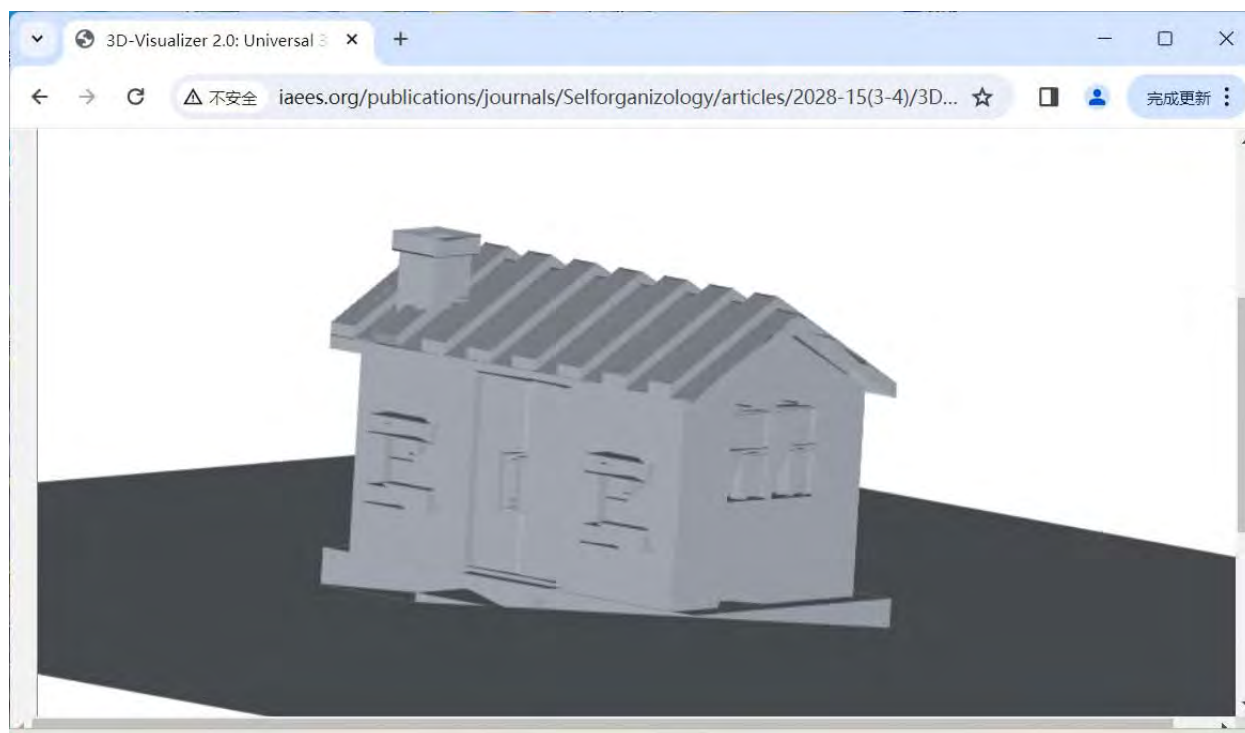


Fig. 2 A space structure visualized by 3D-Visualizer 2.0.

References

- Goddard TD, Huang CC, Meng EC, Pettersen EF, Couch GS, Morris JH, Ferrin TE. 2018. UCSF ChimeraX: Meeting modern challenges in visualization and analysis. *Protein Science*, 27(1): 14-25. <https://onlinelibrary.wiley.com/doi/10.1002/pro.3235>
- Kovács G. Online 3D Viewer: A free and open source web solution to visualize and explore 3D models in your browser. <https://3dviewer.net>
- O'Donoghue SI, Goodsell DS, Frangakis AS, Jossinet F, Laskowski RA, Nilges M, Saibil HR, Schafferhans A, Wade RC, Westhof E, Olson AJ. 2010. Visualization of macromolecular structures. *Nature Methods*, 7(3 Suppl): S42-S55. <https://www.nature.com/articles/nmeth.1427>
- Rego N, Koes D. 2015. 3Dmol.js: Molecular visualization with WebGL. *Bioinformatics*, 31(8): 1322-1324. <https://academic.oup.com/bioinformatics/article/31/8/1322/213186>
- Rose AS, Bradley AR, Valasatava Y, Duarte JM, Prlić A, Rose PW. 2018. NGL viewer: Web-based molecular graphics for large complexes. *Bioinformatics*, 34(21): 3755-3758. <https://academic.oup.com/bioinformatics/article/34/21/3755/5021685>
- Sehnal D, Bittrich S, Deshpande M, Svobodová R, Berka K, Bazgier V, Velankar SK, Burley SK, Koča J, Rose AS. 2021. Mol* Viewer: Modern web app for 3D visualization and analysis of large biomolecular structures. *Nucleic Acids Research*, 49(W1): W431-W437. <https://academic.oup.com/nar/article/49/W1/W431/6270780>
- Virtanen P, Gommers R, Oliphant TE, et al. 2020. SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, 17: 261-272. <https://www.nature.com/articles/s41592-019-0686-2>

- Wang RY, Song YF, Barad BA, Cheng YF, Fraser JS, DiMaio F. 2016. Automated structure refinement of macromolecular assemblies from cryo-EM maps using Rosetta. *eLife*, 5: e17219. <https://elifesciences.org/articles/17219>
- Zhang WJ. 2023. 3D visualization of objects and molecules: An integrative Java software. *Computational Ecology and Software*, 13(4): 81-108. [http://www.iaees.org/publications/journals/ces/articles/2023-13\(4\)/1-Zhang-Abstract.asp](http://www.iaees.org/publications/journals/ces/articles/2023-13(4)/1-Zhang-Abstract.asp)
- 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](http://www.iaees.org/publications/journals/selforganizology/articles/2024-11(3-4)/1-Zhang-Abstract.asp)
- Zhang WJ. 2028a. 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](http://www.iaees.org/publications/journals/selforganizology/articles/2028-15(1-2)/2-Zhang-Abstract.asp)
- Zhang WJ. 2028b. SpaceCarvingJS: Silhouette-based 3D reconstruction using space carving in the browser. *Selforganizology*, 15(3-4): 54-112. [http://www.iaees.org/publications/journals/selforganizology/articles/2028-15\(3-4\)/1-Zhang-Abstract.asp](http://www.iaees.org/publications/journals/selforganizology/articles/2028-15(3-4)/1-Zhang-Abstract.asp)