Article

A browser-based interactive 3D visualizer for molecular and object structures

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 30 June 2026; *Accepted 12 July 2026*; *Published online 30 July 2026*; *Published 1 June 2028*

Abstract
This paper presents a lightweight, browser-based interactive 3D visualizer for molecular and geometric object structures. The system is implemented as a standalone HTML, CSS, and JavaScript application and requires no external libraries, server-side computation, or WebGL support. It supports two common structural data formats: XYZ files for molecular coordinates and OBJ files for polygonal object geometry. XYZ data are rendered as depth-sorted shaded atomic spheres with element-dependent colors, while OBJ data are rendered as grayscale depth-enhanced wireframe models. The visualization pipeline is based on a custom affine 3D transformation matrix that supports scaling, translation, rotation, zooming, panning, and orthographic projection onto an HTML5 canvas. Users can interact with the model through mouse-based rotation, click or wheel-based zooming, slider-based panning, view resetting, and PNG image export. The program demonstrates how fundamental 3D visualization functions can be implemented using only the 2D canvas API, making it suitable for education, rapid structural inspection, and lightweight scientific visualization. Although the current implementation does not include advanced features such as molecular bond detection, perspective projection, hidden-line removal, or material-based OBJ rendering, it provides a compact and extensible framework for interactive visualization of molecular and object structures in standard web environments.

Keywords 3D visualization; interactive browser-based tool; molecular structure; 3D object; XYZ file; OBJ file.

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

Scientific data visualization is a fundamental tool for uncovering the intrinsic structures and regularities of complex systems. In disciplines such as computational ecology, network biology, and cheminformatics, researchers routinely employ graphical interfaces to examine molecular conformations, three-dimensional object topologies, and network relationships. Over the past decade, Zhang has systematically advanced the

field of scientific visualization, establishing a clear technological progression from two-dimensional graph drawing to three-dimensional structure display, and from desktop applications to cross-platform deployment (Zhang, 2012; Zhang, 2021; Zhang, 2023; Zhang, 2024a; Zhang, 2024b; Zhang, 2024c; Zhang, 2024d; Zhang, 2024e).

Zhang's early contribution focused on two-dimensional graph visualization. In 2012, he developed a Java-based software for drawing directed, undirected, cyclic, and acyclic graphs, which ran as a standalone Java application on Windows platforms (Zhang, 2012). Subsequently, he transitioned to the Web environment and presented an online web tool for generating user-interface interactive networks, allowing users to mouse-press and drag nodes to interactively explore network topology and evaluate node centrality (Zhang, 2021). In 2023, he extended visualization from two dimensions to three dimensions by introducing visual3D 1.0, an integrative Java software that supports loading XYZ molecular coordinate files and OBJ object files for 3D rendering, with mouse-based zooming, rotati on, and scrollbar-based translation (Zhang, 2023). Building upon this foundation, Zhang further released multiple network visualization tools in 2024: netVisual, a Matlab software that generates HTML files for browser-based interactive network visualization (Zhang, 2024a); netVisual 2.0, a standalone executable for network visualization that runs on Windows without any runtime dependency (Zhang, 2024b); netGen, an executable Java software for visualizing both undirected and directed networks from text or CSV data files (Zhang, 2024c); netGen 3.0, a substantially improved version supporting weighted/unweighted and directed/undirected networks with customizable node size, color, link attributes, and automatic or manual layout (Zhang, 2024d); and objectVisual3D, a standalone executable software for 3D visualization of objects using OBJ files (Zhang, 2024e). Collectively, these contributions have provided a solid and diverse tool foundation for scientific data visualization.

Despite these advances, existing tools still exhibit limitations in deployment convenience and technical dependencies. Java-based desktop software such as visual3D 1.0 and objectVisual3D requires the Java Runtime Environment (JRE) to be installed, and cross-platform compatibility issues may hinder user experience (Zhang, 2023; Zhang, 2024e). Web tools, while eliminating local installation, often depend on external libraries or server-side computation (Zhang, 2021). Matlab-based tools, although functional, require a Matlab environment or compiled runtime (Zhang, 2024a). Standalone executables improve accessibility but are typically platform-specific (Zhang, 2024b; Zhang, 2024d). Moreover, some tools generate HTML files for browser display, yet their rendering capabilities are constrained by the underlying technologies employed. These limitations restrict the immediate usability of visualization tools in scenarios such as classroom education, rapid structural inspection, and lightweight scientific exploration.

To address these shortcomings, this paper, building upon the author's prior work in three-dimensional visualization (Zhang, 2023; Zhang, 2024e), presents a lightweight, browser-based interactive 3D visualizer for molecular and geometric object structures. The system is implemented as a self-contained HTML, CSS, and JavaScript application that requires no external libraries, server-side computation, or WebGL support. It leverages only the HTML5 2D canvas API to perform affine transformations, depth-sorted rendering, and orthographic projection of 3D objects. The program supports two common structural data formats, XYZ files for molecular coordinates and OBJ files for polygonal object geometry, and provides interactive features including mouse-based rotation, wheel-based zooming, slider-based panning, view resetting, and PNG image export. This work aims to offer an instantly accessible, cross-platform compatible solution for educational demonstrations, rapid structural inspection, and lightweight scientific visualization in standard web environments.

2 Interactive 3D Visualizer for Molecular and Object Structures

2.1 Overview

The attached HTML/JavaScript program implements a browser-based interactive 3D visualizer for two common structural data formats:

- **.xyz files**: rendered as molecular structures, where atoms are displayed as shaded colored spheres.
- **.obj files**: rendered as 3D object structures, displayed as wireframe models.

The program runs entirely in a web browser and uses the HTML5 <canvas> 2D drawing context for rendering. It does not require WebGL, external libraries, server-side processing, or installation.

The interface allows users to load a local structure file, adjust visualization parameters, interactively rotate and zoom the 3D model, pan the scene using sliders, reset the view, and export the current canvas as a PNG image.

2.2 Purpose of the Program

The main purpose of the program is to provide a lightweight, browser-based tool for visualizing simple molecular and geometric object structures.

It is especially useful for:

1. **Molecular visualization**
   - Reading atomic coordinates from .xyz files.
   - Displaying atoms as colored spheres.
   - Showing approximate depth and shading effects.
2. **Object visualization**
   - Reading vertex and face/line definitions from .obj files.
   - Displaying objects as wireframe structures.
3. **Interactive 3D exploration**
   - Rotating the structure using mouse drag.
   - Zooming in and out using mouse clicks or the mouse wheel.
   - Panning using horizontal and vertical sliders.
4. **Educational and demonstration purposes**
   - Explaining simple 3D transformations.
   - Demonstrating matrix-based rotation, scaling, translation, and projection.
   - Providing a compact example of browser-based scientific visualization.

2.3 Main Features

2.3.1 Automatic File-Type Detection

After the user selects a file, the program checks the file extension:

```
javascript
if(name.endsWith('.xyz')){
   fileTypeMsg.textContent = 'XYZ file detected → molecular structure (atoms shown as spheres).';
} else if(name.endsWith('.obj')){
   fileTypeMsg.textContent = 'OBJ file detected → object structure will be rendered (wireframe).';
}
```

The rendering mode is automatically selected:

File Type	Rendering Mode
.xyz	Molecular structure, atoms as spheres
.obj	Object structure, wireframe

2.3.2 Molecular Rendering

For .xyz files, each atom is drawn as a shaded circle representing a sphere. Atom colors are assigned according to chemical symbols.

The program defines the following color table:

javascript

```
const ATOM_COLORS = {
   c:[0,0,0],
   h:[210,210,210],
   n:[0,0,255],
   o:[255,0,0],
   p:[255,0,255],
   s:[255,255,0],
   hn:[150,255,150],
   default:[255,100,200]
};
```

Typical atom color meanings:

Atom	Color
Carbon C	Black
Hydrogen H	Light gray
Nitrogen N	Blue
Oxygen O	Red
Phosphorus P	Magenta
Sulfur S	Yellow
Other atoms	Pink

2.3.3 OBJ Wireframe Rendering

For .obj files, the program reads vertices and face/line definitions. It converts faces into edge pairs and draws them as lines.

Supported OBJ records include:

- v: vertex definition
- f: face definition
- fo: face/object face definition
- l: line definition

The wireframe is rendered with grayscale shading based on depth.

2.3.4 Interactive Mouse Control

The canvas supports the following operations:

Operation	Effect
Left mouse drag	Rotate the structure
Left click without dragging	Zoom in
Right click	Zoom out
Mouse wheel up	Zoom in
Mouse wheel down	Zoom out

The program prevents the default right-click context menu on the canvas:

javascript

```
canvas.addEventListener('contextmenu', e=>e.preventDefault());
```

2.3.5 Pan Control

Two range sliders are used to pan the scene:

- Horizontal Pan
- Vertical Pan

The slider values are converted into translation offsets in the internal transformation matrix.

2.3.6 Export Image

The current visualization can be saved as a PNG file:

javascript

```
canvas.toBlob(blob=>{
  const url = URL.createObjectURL(blob);
  const a = document.createElement('a');
  a.href = url;
  a.download = (state.mode==='xyz' ? 'molecule' : 'object') + '_view.png';
  a.click();
  URL.revokeObjectURL(url);
}, 'image/png');
```

The downloaded file is named either:

- molecule_view.png
- object_view.png

depending on the loaded file type.

2.4 Program Structure

The program consists of the following main components:

1. **HTML interface**
   - File selector
   - Parameter inputs
   - Canvas area
   - Sliders and buttons
2. **CSS styling**
   - Page layout
   - Settings panel
   - Viewer panel
   - Canvas appearance
   - Toolbar and legend
3. **JavaScript modules**
   - 3D matrix transformation class
   - File parsers for XYZ and OBJ
   - Bounding-box computation
   - View initialization
   - Rendering functions
   - Mouse and slider interaction
   - Save/reset/close controls

2.5 Algorithm Description

2.5.1 Data Loading Workflow

The overall data loading procedure is:

1. User selects a local .xyz or .obj file.
2. Program detects file type from extension.
3. User clicks **OK**.
4. Program reads file text using:

javascript

```
const text = await f.text();
```

5. Depending on file type:
   - .xyz file is parsed by parseXYZ(text).
   - .obj file is parsed by parseOBJ(text).
6. The vertex coordinate array is stored in a flattened format:

javascript

```
[x1, y1, z1, x2, y2, z2, ...]
```

7. The bounding box is calculated.
8. The viewing transformation is initialized.
9. The scene is rendered on the canvas.

2.5.2 XYZ Parsing Algorithm

The XYZ parser reads molecular coordinate data.

Typical XYZ format:

text

```
5
Example molecule
C      0.000      0.000      0.000
H      0.000      0.000      1.089
H      1.026      0.000     -0.363
H     -0.513     -0.889     -0.363
H     -0.513      0.889     -0.363
```

The parser:

1. Splits the file into lines.
2. Detects whether the first non-empty line is an atom count.
3. If an atom count is found, skips the first two lines.
4. Reads atom symbol and x/y/z coordinates.
5. Stores atom symbols in atoms.
6. Stores coordinates in verts.

Representative code:

javascript

```
function parseXYZ(text){
   const lines = text.split(/\r?\n/);
   const atoms=[], verts=[];
   let start = 0;

   const firstNonEmpty = lines.findIndex(l=>l.trim()!=="");
```

```
  if(firstNonEmpty>=0 && /^\d+$/.test(lines[firstNonEmpty].trim())){
    start = firstNonEmpty + 2;
  }

  for(let i=start;i<lines.length;i++){
    const line = lines[i].trim();
    if(!line) continue;

    const parts = line.split(/\s+/);
    if(parts.length<4) continue;

    const x=parseFloat(parts[1]);
    const y=parseFloat(parts[2]);
    const z=parseFloat(parts[3]);

    if(Number.isNaN(x)||Number.isNaN(y)||Number.isNaN(z)) continue;

    atoms.push(parts[0].toLowerCase());
    verts.push(x,y,z);
  }

  if(verts.length===0)
    throw new Error('No atomic coordinates could be parsed (XYZ format).');

  return {atoms, verts};
}
```

2.5.3 OBJ Parsing Algorithm

The OBJ parser reads vertices and edges.

Supported OBJ examples:

```
text
v 0 0 0
v 1 0 0
v 1 1 0
v 0 1 0
f 1 2 3 4
```

The parser:

1. Splits the file into lines.
2. Removes comments beginning with #.
3. Reads vertex records beginning with v.
4. Reads face or line records beginning with f, fo, or l.
5. Converts faces into edges.
6. Uses a Set to avoid duplicate edges.
7. Returns vertices and edge list.

Representative code:

```
javascript
function parseOBJ(text){
   const lines = text.split(/\r?\n/);
   const verts=[];
   const edgeSet = new Set();
   const edges=[];

   const addEdge=(a,b)=>{
      if(a<0||b<0||a===b) return;
      const lo=Math.min(a,b), hi=Math.max(a,b);
      const key=lo+'_'+hi;
      if(!edgeSet.has(key)){
         edgeSet.add(key);
         edges.push([lo,hi]);
      }
   };

   for(const raw of lines){
      const line = raw.split('#')[0].trim();
      if(!line) continue;

      const parts = line.split(/\s+/);
      const tag = parts[0];

      if(tag==='v'){
         const x=parseFloat(parts[1]);
         const y=parseFloat(parts[2]);
         const z=parseFloat(parts[3]);

         if(!Number.isNaN(x)&&!Number.isNaN(y)&&!Number.isNaN(z))
            verts.push(x,y,z);

      } else if(tag==='f' || tag==='fo' || tag==='l'){
         const idxs = parts.slice(1)
            .map(p=>parseInt(p.split('/')[0],10)-1)
            .filter(i=>!Number.isNaN(i) && i>=0);

         for(let i=0;i<idxs.length;i++)
            addEdge(idxs[i], idxs[(i+1)%idxs.length]);
      }
   }

   if(verts.length===0)
      throw new Error('No vertex data could be parsed (OBJ format).');
```

```
    return {verts, edges};
}
```

2.6 3D Transformation Algorithm

2.6.1 Matrix Class

The core of the visualization is the Mat3D class. It stores a 3D affine transformation matrix using twelve values:

javascript

```
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;
```

The matrix supports:

- Identity initialization
- Scaling
- Translation
- Rotation around the x-axis
- Rotation around the y-axis
- Matrix multiplication
- Transformation of vertex coordinates

2.6.2 Rotation

Rotation around the x-axis is implemented as:

javascript

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

    const yx=this.yx*c+this.zx*s;
    const yy=this.yy*c+this.zy*s;
    const yz=this.yz*c+this.zz*s;
    const yo=this.yo*c+this.zo*s;

    const zx=this.zx*c-this.yx*s;
    const zy=this.zy*c-this.yy*s;
    const zz=this.zz*c-this.yz*s;
    const 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;
}
```

Rotation around the y-axis is implemented similarly.

During mouse dragging, the program calculates rotation angles from the mouse displacement:

javascript

```
const fx = (lastY - y) * (360/canvas.height);
```

```
const fy = (x - lastX) * (360/canvas.width);
tmat.xrot(fx);
tmat.yrot(fy);
state.amat.mult(tmat);
```

Thus:

- Vertical mouse movement rotates the model around the x-axis.
- Horizontal mouse movement rotates the model around the y-axis.

2.6.3 Transformation Pipeline

For each rendering frame, the program performs the following transformation sequence:

1. Translate the model center to the origin.
2. Apply accumulated rotation and panning matrix.
3. Scale the object according to the canvas size and zoom factor.
4. Flip the y-axis to match canvas coordinates.
5. Map the object to the center of the canvas.
6. Convert the 3D vertices into 2D screen coordinates with depth.

Representative code:

```
javascript
function render(){
   ctx.fillStyle = '#ffffff';
   ctx.fillRect(0,0,canvas.width,canvas.height);

   if(!state.verts || !state.bb) return;

   const {xmin,xmax,ymin,ymax,zmin,zmax} = state.bb;

   const mat = new Mat3D();
   mat.unit();

   mat.translate(
      -(xmin+xmax)/2,
      -(ymin+ymax)/2,
      -(zmin+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 = state.verts.length/3;
  const tvert = mat.transform(state.verts, n);

  if(state.mode==='xyz')
    drawMolecule(tvert, n);
  else
    drawWire(tvert, state.edges);
}
```

This is an orthographic-style projection rather than a perspective projection. The x and y coordinates are mapped directly to canvas coordinates, while the transformed z coordinate is used mainly for depth sorting and shading.

2.7 Bounding Box and Initial Scaling

Before rendering, the program computes the minimum and maximum coordinates in x, y, and z directions.

javascript

```
function computeBB(){
  const v = state.verts;
  let xmin=v[0],xmax=v[0],
      ymin=v[1],ymax=v[1],
      zmin=v[2],zmax=v[2];

  for(let i=0;i<v.length;i+=3){
    const x=v[i], y=v[i+1], z=v[i+2];

    if(x<xmin)xmin=x;
    if(x>xmax)xmax=x;

    if(y<ymin)ymin=y;
    if(y>ymax)ymax=y;

    if(z<zmin)zmin=z;
    if(z>zmax)zmax=z;
  }

  state.bb={xmin,xmax,ymin,ymax,zmin,zmax};
}
```

The largest dimension of the model is used to calculate the initial display scale. This ensures that the entire model fits reasonably within the canvas.

javascript

```
let f = Math.max(xmax-xmin, ymax-ymin, zmax-zmin);
if(f<=0) f=1;

const f3 = canvas.width/f;
const f4 = canvas.height/f;
```

state.xfac = 0.7*Math.min(f3,f4)*state.scale;

The factor 0.7 leaves a margin around the displayed structure.

2.8 Rendering Methods

2.8.1 Molecule Rendering

Molecular rendering consists of three main steps:

1. Transform all atom coordinates.
2. Sort atoms by depth.
3. Draw each atom as a shaded ball.

The sorting step is important because the program uses the 2D canvas API and does not have a hardware depth buffer. Therefore, farther atoms are drawn first and nearer atoms are drawn later.

javascript

```
function drawMolecule(tvert, n){
   const order = Array.from({length:n}, (_,i)=>i);

   order.sort((a,b)=> tvert[a*3+2]-tvert[b*3+2]);

   for(const i of order){
      let z = tvert[i*3+2];
      if(z<0) z=0;
      if(z>15) z=15;

      const sym = state.atoms[i];
      const color = ATOM_COLORS[sym] || ATOM_COLORS.default;

      drawBall(tvert[i*3], tvert[i*3+1], z, color);
   }
}
```

Each ball is drawn using a radial gradient:

javascript

```
function drawBall(x,y,zLevel,colorArr){
   const radius = Math.max(2, (state.ballSize + zLevel)/2);

   const fogFrac = (zLevel/15) * 0.5;
   const base = colorArr.map(c=>Math.round(c*(1-fogFrac)+192*fogFrac));

   const grad = ctx.createRadialGradient(
      x-radius*0.35,
      y-radius*0.35,
      radius*0.05,
      x,
      y,
      radius
   );
```

```
    grad.addColorStop(0, 'rgb(255,255,255)');
    grad.addColorStop(0.35, `rgb(${base[0]},${base[1]},${base[2]})`);
    grad.addColorStop(1,
`rgb(${Math.round(base[0]*0.3)},${Math.round(base[1]*0.3)},${Math.round(base[2]*0.3)})`);

    ctx.beginPath();
    ctx.fillStyle = grad;
    ctx.arc(x,y,radius,0,Math.PI*2);
    ctx.fill();
}
```

This produces a simple 3D-like sphere effect without requiring real 3D rendering.

2.8.2 Wireframe Rendering

For OBJ objects, the program draws each edge as a line between two transformed vertices. Depth-dependent grayscale is used to enhance the 3D impression.

javascript

```
function drawWire(tvert, edges){
    for(const [a,b] of edges){
        let depth = tvert[a*3+2] + tvert[b*3+2];

        if(depth<0) depth=0;
        if(depth>15) depth=15;

        ctx.strokeStyle = GR_LEVELS[Math.round(depth)];

        ctx.beginPath();
        ctx.moveTo(tvert[a*3], tvert[a*3+1]);
        ctx.lineTo(tvert[b*3], tvert[b*3+1]);
        ctx.stroke();
    }
}
```

The grayscale table is generated by:

javascript

```
function computeGrLevels(){
    const arr=[];
    for(let i=0;i<16;i++){
        const k = Math.round(170*(1-Math.pow(i/15,2.3)));
        arr.push(`rgb(${k},${k},${k})`);
    }
    return arr;
}
```

2.9 User Interface Parameters

The settings panel contains several user-adjustable parameters.

Parameter	Meaning
Data file	Select a .xyz or .obj file

Parameter	Meaning
Zoom Scale	Initial magnification factor
Dilation Rate	Zoom increment/decrement ratio
Max Scroll	Maximum value of panning sliders
Canvas Width	Width of drawing canvas
Canvas Height	Height of drawing canvas
Ball Size	Base atom sphere size, for molecules only
Ball Color Levels	Intended molecular color-level parameter

Note: In the current implementation, ballCol is read from the interface but is not substantially used in the drawing algorithm. The visual depth effect is mainly controlled by zLevel, ballSize, and the radial gradient.

2.10 User Guide

2.10.1 Loading a File

1. Open the HTML file in a modern web browser.
2. In the settings panel, click the file input button.
3. Select a .xyz or .obj file.
4. The program will display a message indicating the detected file type.
5. Adjust parameters if necessary.
6. Click **OK**.

The viewer panel will appear and display the structure.

2.10.2 Recommended File Formats

XYZ Molecular File

Example:

text

```
3
Water molecule
O    0.0000    0.0000    0.0000
H    0.9572    0.0000    0.0000
H -0.2390    0.9270    0.0000
```

The first line may contain the number of atoms. The second line may contain a comment. Each following line should contain:

text

```
AtomSymbol    X    Y    Z
```

OBJ Object File

Example:

text

```
v 0 0 0
v 1 0 0
v 1 1 0
v 0 1 0
v 0 0 1
v 1 0 1
v 1 1 1
v 0 1 1
```

f 1 2 3 4
f 5 6 7 8
f 1 2 6 5
f 2 3 7 6
f 3 4 8 7
f 4 1 5 8

The program uses OBJ vertex indices starting from 1, as in standard OBJ files.

2.10.3 Rotating the Model

To rotate the model:

1. Move the mouse over the canvas.
2. Press and hold the left mouse button.
3. Drag the mouse.

Effects:

- Drag left/right: rotate around the vertical direction.
- Drag up/down: rotate around the horizontal direction.

2.10.4 Zooming

There are three ways to zoom:

1. **Left click without dragging**
   - Zoom in.
2. **Right click**
   - Zoom out.
3. **Mouse wheel**
   - Wheel up: zoom in.
   - Wheel down: zoom out.

The zoom ratio is controlled by the **Dilation Rate** parameter.

For example, if the dilation rate is 0.3, each zoom-in operation increases the scale by 30%.

2.10.5 Panning

Use the sliders below the canvas:

- **Horizontal Pan**
- **Vertical Pan**

Moving the sliders shifts the displayed model horizontally or vertically.

2.10.6 Saving the Image

To save the current view:

1. Rotate, zoom, or pan the model to the desired view.
2. Click **Save As PNG**.
3. A PNG image will be downloaded automatically.

2.10.7 Resetting the View

Click **Reset View** to restore:

- Initial rotation
- Initial zoom
- Centered position
- Slider positions

2.10.8 Returning to Settings

Click **Close / Back** to hide the viewer and return to the settings panel.

The previously loaded data remain in memory until a new file is loaded or the page is refreshed.

2.11 Important Implementation Details

2.11.1 Global State Object

The program stores visualization data and parameters in a single state object:

javascript

```
const state = {
  mode:null,
  verts:null,
  atoms:null,
  edges:null,
  bb:null,
  amat:null,
  xfac:1,
  xfac0:1,
  scale:2,
  dila:0.3,
  scrollmax:2000,
  canvasWidth:900,
  canvasHeight:680,
  ballSize:10,
  ballCol:20
};
```

This object controls:

- Current visualization mode
- Vertex coordinates
- Atom symbols
- OBJ edge list
- Bounding box
- Transformation matrix
- Zoom factor
- Canvas size
- Molecule rendering parameters

2.11.2 View Initialization

The function setupView() initializes the canvas and transformation state:

javascript

```
function setupView(){
  canvas.width = state.canvasWidth;
  canvas.height = state.canvasHeight;

  const {xmin,xmax,ymin,ymax,zmin,zmax} = state.bb;

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

```
    if(f<=0) f=1;

    const f3 = canvas.width/f;
    const f4 = canvas.height/f;

    state.xfac = 0.7*Math.min(f3,f4)*state.scale;
    state.xfac0 = state.xfac;

    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;
}
```

The initial view is rotated by 20 degrees around both the x-axis and y-axis, giving a clearer 3D impression.

2.12 Strengths of the Program

1. **No external dependencies**
   - Runs directly in the browser.
2. **Simple and portable**
   - Single HTML file containing HTML, CSS, and JavaScript.
3. **Automatic mode selection**
   - Detects molecular or object visualization from file extension.
4. **Interactive control**
   - Supports rotation, zoom, pan, reset, and image export.
5. **Educational value**
   - Clear example of matrix transformations and canvas-based rendering.
6. **Supports two useful formats**
   - XYZ for molecules.
   - OBJ for geometric objects.

2.13 Limitations and Possible Improvements

2.13.1 No Perspective Projection

The current renderer uses an orthographic-style projection. Adding perspective projection would improve depth realism.

2.13.2 No Bond Detection for Molecules

The molecular mode only displays atoms as spheres. It does not automatically detect or draw chemical bonds.

Possible improvement:

- Add distance-based bond detection.
- Draw bonds as cylinders or lines.

2.13.3 Limited OBJ Support

The OBJ parser supports vertices, faces, and lines, but ignores:

- Normals
- Texture coordinates
- Materials
- Groups
- Negative indices
- Smooth shading

2.13.4 No WebGL Acceleration

Rendering is performed using the 2D canvas API. Large molecules or high-complexity OBJ files may render slowly.

Possible improvement:

- Use WebGL or Three.js for hardware-accelerated 3D rendering.

2.13.5 Limited Depth Handling

The molecule renderer sorts atoms by depth, but the wireframe renderer does not perform hidden-line removal. All edges are visible.

2.13.6 ballCol Parameter Is Not Fully Used

The user interface includes “Ball Color Levels,” but the current rendering function does not fully apply this value. The color shading is instead hard-coded using a depth-based fog fraction.

2.14 Summary

This program is a compact and practical browser-based 3D visualization tool for molecular and object structures. It reads .xyz molecular files and .obj object files, automatically determines the rendering mode, and displays the structure interactively on an HTML5 canvas.

Its main algorithmic foundation is a custom 3D affine transformation class, Mat3D, which handles translation, scaling, rotation, and coordinate transformation. Molecular structures are rendered as depth-sorted shaded spheres, while object structures are rendered as grayscale depth-shaded wireframes.

The program is suitable for educational demonstrations, lightweight scientific visualization, and quick inspection of simple molecular or object data files.

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(1-2)/3D-Visualizer.htm; Fig. 1):

```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
<meta charset="UTF-8">
<title>Interactive 3D Visualizer for Molecular and Object Structures</title>
<style>
  body{font-family:"Microsoft YaHei",Arial,sans-serif;background:#f2f2f2;margin:0;padding:20px;color:#222}
  h2{margin-top:0}
```

```
  #settingsPanel{max-width:520px;background:#fff;border:1px solid #ccc;border-radius:6px;padding:20px}
  table{border-collapse:collapse;width:100%}
  td{padding:6px 8px}
  td.lbl{text-align:right;white-space:nowrap;width:45%}
  input[type=number], input[type=text]{width:120px}
  .hint{font-size:12px;color:#666;margin-top:4px}
  #fileTypeMsg{margin-top:8px;font-weight:bold}
  button{padding:6px 16px;margin-right:8px;cursor:pointer}
  #viewerPanel{margin-top:20px}
  #canvasWrap{position:relative;display:inline-block;border:1px solid #888;background:#fff}
  canvas{display:block;cursor:grab}
  canvas:active{cursor:grabbing}
  #hScrollWrap, #vScrollLabel{margin-top:6px}
  .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}
</style>
</head>
<body>

<h2>Interactive 3D Visualizer for Molecular and Object Structures</h2>
<p class="note"><a href="http://www.iaees.org/publications/journals/Selforganizology/articles/2028-15(1-2)/2-Zhang-Abstract.asp">Zhang WJ. 2028. A browser-based interactive 3D visualizer for molecular and object structures. Selforganizology, 15(1-2): 21-53. </a><br>
After selecting a data file, the program will automatically determine the rendering method based on the file extension: <b>.xyz → molecular structure (spheres)</b>, <b>.obj → object structure (wireframe)</b>(<a href="SampleDataFiles.zip">Download</a> the package of sample data files.<br> </p>

<div id="settingsPanel">
  <table>
    <tr>
      <td class="lbl">Data file (*.xyz, or *.obj)：</td>
      <td><input type="file" id="fileInput" accept=".xyz,.obj"></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="680"></td>
      </tr>
      <tr>
        <td class="lbl">Ball Size (molecules only):</td>
        <td><input type="number" id="ballSizeInput" value="10"></td>
      </tr>
      <tr>
        <td class="lbl">Ball Color Levels (molecules only):</td>
        <td><input type="number" id="ballColInput" value="20"></td>
      </tr>
    </table>
    <div class="toolbar">
      <button id="loadBtn">OK</button>
    </div>
</div>

<div id="viewerPanel" style="display:none">
    <div id="canvasWrap">
      <canvas id="glcanvas"></canvas>
    </div>
    <div id="hScrollWrap">
      Horizontal Pan: <input type="range" id="hScroll" style="width:400px">
    </div>
    <div>
      Vertical Pan: <input type="range" id="vScroll" style="width:400px">
    </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>
    <p class="note">Instructions: Left-click drag = rotate; left-click (without dragging) = zoom in, right-click = zoom out; scroll wheel = zoom; slider below = pan.</p>
</div>
```

```
<script>
/* ============================================================
    Mat3D — Revise and rewrite from: 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
    ============================================================ */
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;
    }
    // Convert a flattened array [x,y,z, x,y,z, ...] to screen coordinates (with depth z).
    transform(vertsFlat, 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;
            const vx=vertsFlat[j], vy=vertsFlat[j+1], vz=vertsFlat[j+2];
            out[j]     = vx*xx+vy*xy+vz*xz+xo;
            out[j+1] = vx*yx+vy*yy+vz*yz+yo;
            out[j+2] = vx*zx+vy*zy+vz*zz+zo;
        }
        return out;
    }
}

/* ============================================================
     Atom Color Table
     ============================================================ */
const ATOM_COLORS = {
    c:[0,0,0], h:[210,210,210], n:[0,0,255], o:[255,0,0],
    p:[255,0,255], s:[255,255,0], hn:[150,255,150],
    default:[255,100,200]
};

/* ============================================================
     File parsing: XYZ (molecules) / OBJ (wireframe objects)
     ============================================================ */
function parseXYZ(text){
    const lines = text.split(/\r?\n/);
    const atoms=[], verts=[];
    let start = 0;
    const firstNonEmpty = lines.findIndex(l=>l.trim()!=='');
    if(firstNonEmpty>=0 && /^\d+$/.test(lines[firstNonEmpty].trim())){
        start = firstNonEmpty + 2; // Skip atom count line and comment line.
    }
    for(let i=start;i<lines.length;i++){
        const line = lines[i].trim();
        if(!line) continue;
        const parts = line.split(/\s+/);
```

```
        if(parts.length<4) continue;
        const x=parseFloat(parts[1]), y=parseFloat(parts[2]), z=parseFloat(parts[3]);
        if(Number.isNaN(x)||Number.isNaN(y)||Number.isNaN(z)) continue;
        atoms.push(parts[0].toLowerCase());
        verts.push(x,y,z);
    }
    if(verts.length===0) throw new Error('No atomic coordinates could be parsed (XYZ format).');
    return {atoms, verts};
}

function parseOBJ(text){
    const lines = text.split(/\r?\n/);
    const verts=[];
    const edgeSet = new Set();
    const edges=[];
    const addEdge=(a,b)=>{
        if(a<0||b<0||a===b) return;
        const lo=Math.min(a,b), hi=Math.max(a,b);
        const key=lo+'_'+hi;
        if(!edgeSet.has(key)){ edgeSet.add(key); edges.push([lo,hi]); }
    };
    for(const raw of lines){
        const line = raw.split('#')[0].trim();
        if(!line) continue;
        const parts = line.split(/\s+/);
        const tag = parts[0];
        if(tag==='v'){
            const x=parseFloat(parts[1]), y=parseFloat(parts[2]), z=parseFloat(parts[3]);
            if(!Number.isNaN(x)&&!Number.isNaN(y)&&!Number.isNaN(z)) verts.push(x,y,z);
        } else if(tag==='f' || tag==='fo' || tag==='l'){
            const idxs = parts.slice(1)
                .map(p=>parseInt(p.split('/')[0],10)-1)
                .filter(i=>!Number.isNaN(i) && i>=0);
            for(let i=0;i<idxs.length;i++) addEdge(idxs[i], idxs[(i+1)%idxs.length]);
        }
    }
    if(verts.length===0) throw new Error('No vertex data could be parsed (OBJ format).');
    return {verts, edges};
}

/* ============================================================
   Initialize Global State and View
   ============================================================ */
const state = {
    mode:null, verts:null, atoms:null, edges:null, bb:null,
```

```
    amat:null, xfac:1, xfac0:1,
    scale:2, dila:0.3, scrollmax:2000,
    canvasWidth:900, canvasHeight:680,
    ballSize:10, ballCol:20
};

function computeBB(){
    const v = state.verts;
    let xmin=v[0],xmax=v[0],ymin=v[1],ymax=v[1],zmin=v[2],zmax=v[2];
    for(let i=0;i<v.length;i+=3){
        const x=v[i],y=v[i+1],z=v[i+2];
        if(x<xmin)xmin=x; if(x>xmax)xmax=x;
        if(y<ymin)ymin=y; if(y>ymax)ymax=y;
        if(z<zmin)zmin=z; if(z>zmax)zmax=z;
    }
    state.bb={xmin,xmax,ymin,ymax,zmin,zmax};
}

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.canvasWidth;
    canvas.height = state.canvasHeight;
    const {xmin,xmax,ymin,ymax,zmin,zmax} = state.bb;
    let f = Math.max(xmax-xmin, ymax-ymin, zmax-zmin);
    if(f<=0) f=1;
    const f3 = canvas.width/f, f4 = canvas.height/f;
    state.xfac = 0.7*Math.min(f3,f4)*state.scale;
    state.xfac0 = state.xfac;
    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;
}

/* ============================================================
     Rendering
     ============================================================ */
function computeGrLevels(){
```

```
    const arr=[];
    for(let i=0;i<16;i++){
        const k = Math.round(170*(1-Math.pow(i/15,2.3)));
        arr.push(`rgb(${k},${k},${k})`);
    }
    return arr;
}
const GR_LEVELS = computeGrLevels();

function drawBall(x,y,zLevel,colorArr){
    const radius = Math.max(2, (state.ballSize + zLevel)/2);
    const fogFrac = (zLevel/15) * 0.5; // Gray fog with distance, simulates ballCol shading.
    const base = colorArr.map(c=>Math.round(c*(1-fogFrac)+192*fogFrac));
    const grad = ctx.createRadialGradient(x-radius*0.35, y-radius*0.35, radius*0.05, x, y, radius);
    grad.addColorStop(0, 'rgb(255,255,255)');
    grad.addColorStop(0.35, `rgb(${base[0]},${base[1]},${base[2]})`);
    grad.addColorStop(1, `rgb(${Math.round(base[0]*0.3)},${Math.round(base[1]*0.3)},${Math.round(base[2]*0.3)})`);
    ctx.beginPath();
    ctx.fillStyle = grad;
    ctx.arc(x,y,radius,0,Math.PI*2);
    ctx.fill();
}

function drawMolecule(tvert, n){
    const order = Array.from({length:n}, (_,i)=>i);
    order.sort((a,b)=> tvert[a*3+2]-tvert[b*3+2]); // Back-to-front rendering: later draws overlay earlier ones.
    for(const i of order){
        let z = tvert[i*3+2];
        if(z<0) z=0; if(z>15) z=15;
        const sym = state.atoms[i];
        const color = ATOM_COLORS[sym] || ATOM_COLORS.default;
        drawBall(tvert[i*3], tvert[i*3+1], z, color);
    }
}

function drawWire(tvert, edges){
    for(const [a,b] of edges){
        let depth = tvert[a*3+2] + tvert[b*3+2];
        if(depth<0) depth=0; if(depth>15) depth=15;
        ctx.strokeStyle = GR_LEVELS[Math.round(depth)];
        ctx.beginPath();
        ctx.moveTo(tvert[a*3], tvert[a*3+1]);
        ctx.lineTo(tvert[b*3], tvert[b*3+1]);
        ctx.stroke();
    }
```

```
}

function render(){
   ctx.fillStyle = '#ffffff';
   ctx.fillRect(0,0,canvas.width,canvas.height);
   if(!state.verts || !state.bb) return;
   const {xmin,xmax,ymin,ymax,zmin,zmax} = state.bb;
   const mat = new Mat3D();
   mat.unit();
   mat.translate(-(xmin+xmax)/2, -(ymin+ymax)/2, -(zmin+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 = state.verts.length/3;
   const tvert = mat.transform(state.verts, n);
   if(state.mode==='xyz') drawMolecule(tvert, n);
   else drawWire(tvert, state.edges);
}

/* ============================================================
     Mouse Interaction: drag to rotate / click to zoom / scroll to zoom
     ============================================================ */
let dragging=false, lastX=0, lastY=0, downX=0, downY=0;

canvas.addEventListener('contextmenu', e=>e.preventDefault());

canvas.addEventListener('mousedown', e=>{
   dragging = true;
   const r = canvas.getBoundingClientRect();
   lastX = e.clientX-r.left; lastY = e.clientY-r.top;
   downX = lastX; downY = lastY;
});

window.addEventListener('mousemove', e=>{
   if(!dragging || !state.amat) return;
   const r = canvas.getBoundingClientRect();
   const x = e.clientX-r.left, y = e.clientY-r.top;
   const tmat = new Mat3D();
   const fx = (lastY - y) * (360/canvas.height);
   const fy = (x - lastX) * (360/canvas.width);
   tmat.xrot(fx);
   tmat.yrot(fy);
   state.amat.mult(tmat);
   lastX=x; lastY=y;
   render();
```

```
});

window.addEventListener('mouseup', e=>{
   if(!dragging) return;
   dragging=false;
   if(!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){
      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 += state.xfac*state.dila; render(); }
function zoomOut(){
   if(1-state.dila>0) state.xfac -= state.xfac*state.dila;
   else state.xfac = 0.1;
   render();
}

/* Pan Slider */
hScroll.addEventListener('input', ()=>{
   if(!state.amat) return;
   const pos = parseInt(hScroll.value,10);
   const dif = pos - prevH;
   prevH = pos;
   state.amat.translate(-dif, 0, 0);
   render();
});
vScroll.addEventListener('input', ()=>{
   if(!state.amat) return;
   const pos = parseInt(vScroll.value,10);
   const dif = pos - prevV;
   prevV = pos;
   state.amat.translate(0, dif, 0);
   render();
});

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

```
    UI：Auto-detect on selection / Load / Save / Close / Reset
    ========================================================== */
const fileInput = document.getElementById('fileInput');
const fileTypeMsg = document.getElementById('fileTypeMsg');

fileInput.addEventListener('change', ()=>{
  const f = fileInput.files[0];
  if(!f){ fileTypeMsg.textContent=''; return; }
  const name = f.name.toLowerCase();
  if(name.endsWith('.xyz')){
    fileTypeMsg.textContent = 'XYZ file detected → molecular structure (atoms shown as spheres).';
    fileTypeMsg.style.color = '#0a6';
  } else if(name.endsWith('.obj')){
    fileTypeMsg.textContent = 'OBJ file detected → object structure will be rendered (wireframe).';
    fileTypeMsg.style.color = '#06a';
  } else {
    fileTypeMsg.textContent = 'The file type is not supported. Please select a .xyz or .obj file. ';
    fileTypeMsg.style.color = '#c00';
  }
});

document.getElementById('loadBtn').addEventListener('click', async ()=>{
  const f = fileInput.files[0];
  if(!f){ alert('Please select a data file first.'); return; }
  const name = f.name.toLowerCase();
  let mode;
  if(name.endsWith('.xyz')) mode='xyz';
  else if(name.endsWith('.obj')) mode='obj';
  else { alert('The file type is not supported. Please select a .xyz or .obj file.'); return; }

  state.scale      = parseFloat(document.getElementById('scaleInput').value) || 2;
  state.dila       = parseFloat(document.getElementById('dilaInput').value) || 0.3;
  state.scrollmax = parseInt(document.getElementById('scrollMaxInput').value) || 2000;
  state.canvasWidth  = parseInt(document.getElementById('canvasWInput').value) || 900;
  state.canvasHeight = parseInt(document.getElementById('canvasHInput').value) || 680;
  state.ballSize  = parseFloat(document.getElementById('ballSizeInput').value) || 10;
  state.ballCol   = parseInt(document.getElementById('ballColInput').value) || 20;

  try{
    const text = await f.text();
    if(mode==='xyz'){
      const {atoms, verts} = parseXYZ(text);
      state.mode='xyz'; state.atoms=atoms; state.edges=null;
      state.verts = new Float32Array(verts);
    } else {
```

```
            const {verts, edges} = parseOBJ(text);
            state.mode='obj'; state.edges=edges; state.atoms=null;
            state.verts = new Float32Array(verts);
        }
        computeBB();
        setupView();
        document.getElementById('settingsPanel').style.display='none';
        document.getElementById('viewerPanel').style.display='block';
        updateLegend();
        render();
    }catch(err){
        alert('File parsing error:' + err.message);
    }
});

function updateLegend(){
    const legend = document.getElementById('legend');
    if(state.mode!=='xyz'){ legend.innerHTML=''; return; }
    const names = {c:'Carbon C', h:'Hydrogen H', n:'Nitrogen N', o:'Oxygen O', p:'Phosphorus P', s:'Sulfur S', hn:'HN', default:'Other'};
    let html = 'Atom Color Legend:';
    for(const k in ATOM_COLORS){
        const c = ATOM_COLORS[k];
        html += `<span style="background:rgb(${c[0]},${c[1]},${c[2]})"></span>${names[k]}  `;
    }
    legend.innerHTML = html;
}

document.getElementById('saveBtn').addEventListener('click', ()=>{
    canvas.toBlob(blob=>{
        const url = URL.createObjectURL(blob);
        const a = document.createElement('a');
        a.href = url;
        a.download = (state.mode==='xyz' ? 'molecule' : 'object') + '_view.png';
        document.body.appendChild(a);
        a.click();
        a.remove();
        URL.revokeObjectURL(url);
    }, 'image/png');
});

document.getElementById('resetBtn').addEventListener('click', ()=>{
    if(!state.bb) return;
    setupView();
    render();
```

```
});

document.getElementById('closeBtn').addEventListener('click', ()=>{
   document.getElementById('viewerPanel').style.display='none';
   document.getElementById('settingsPanel').style.display='block';
});
</script>
</body>
</html>
```

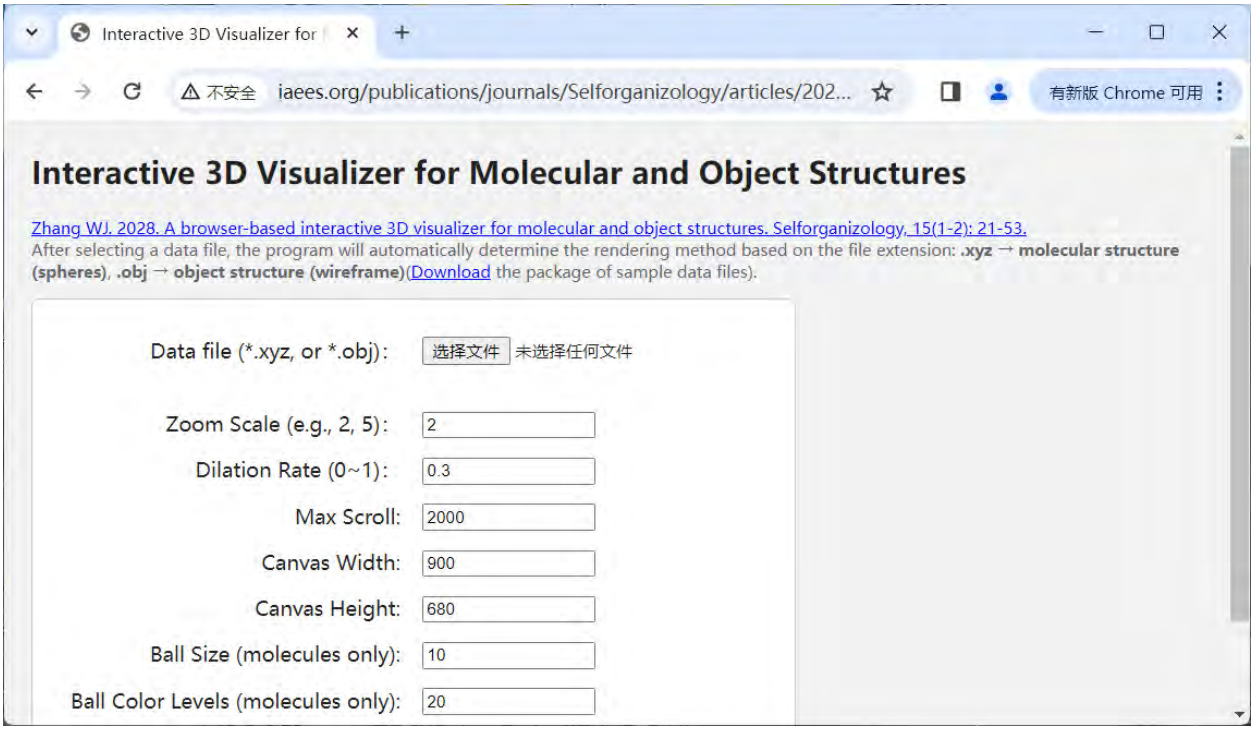
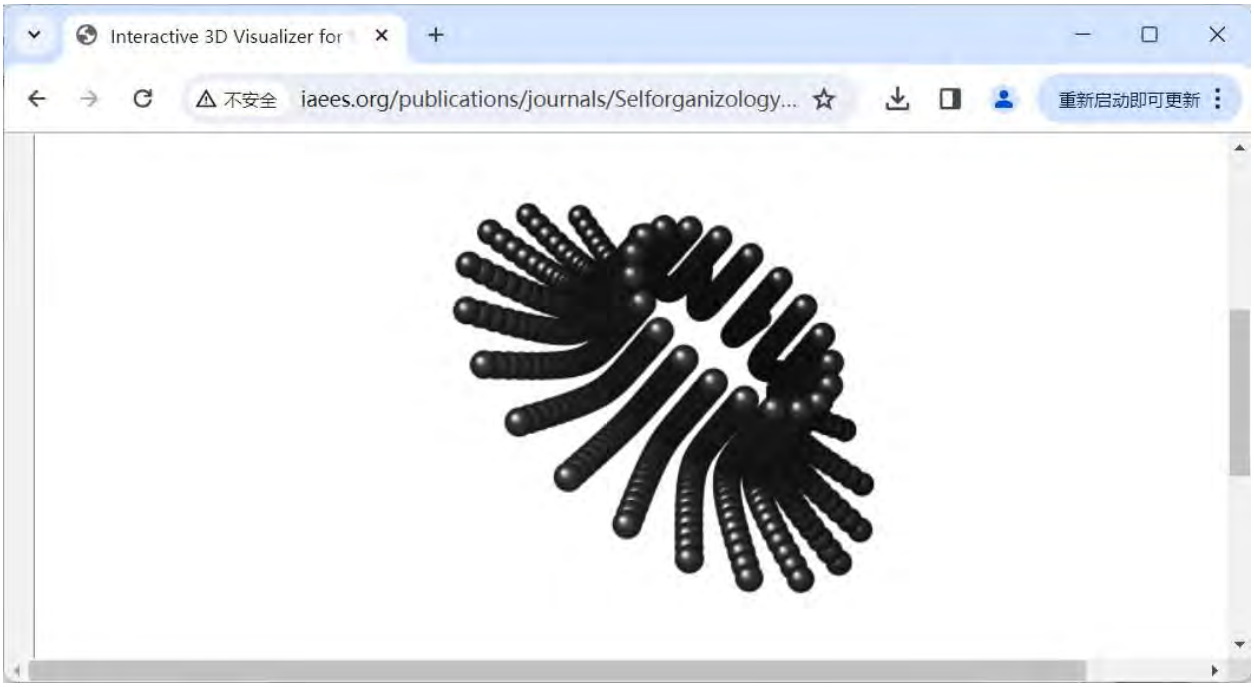
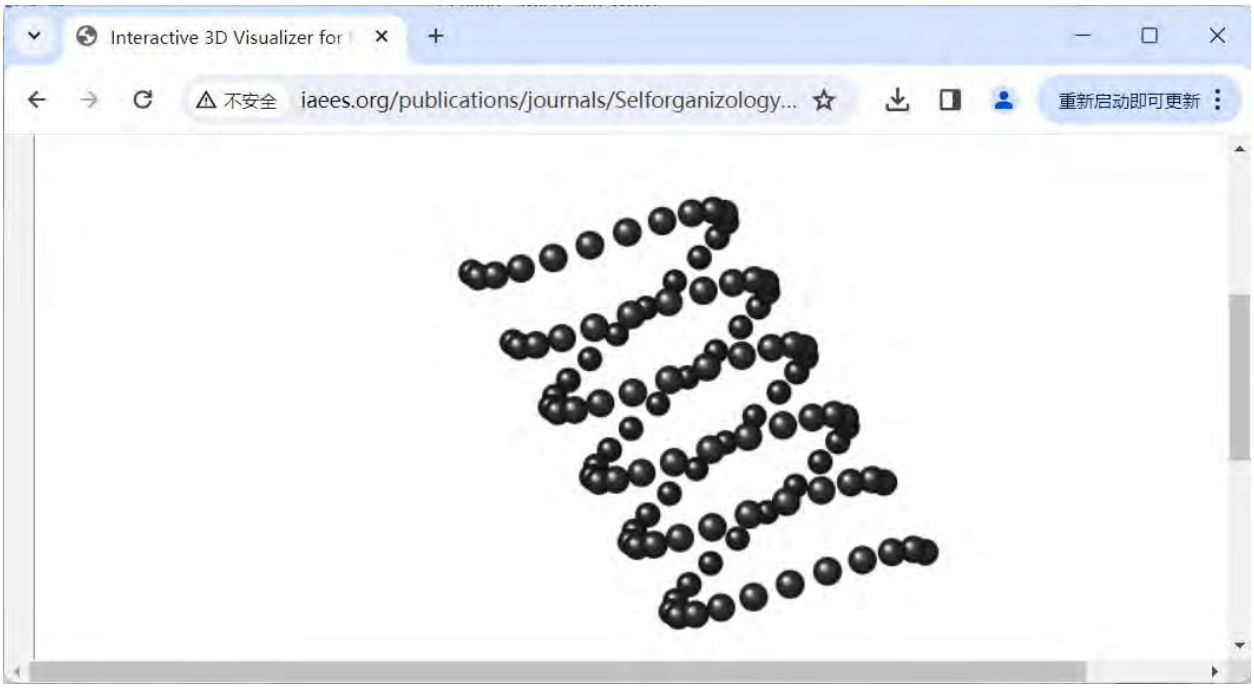


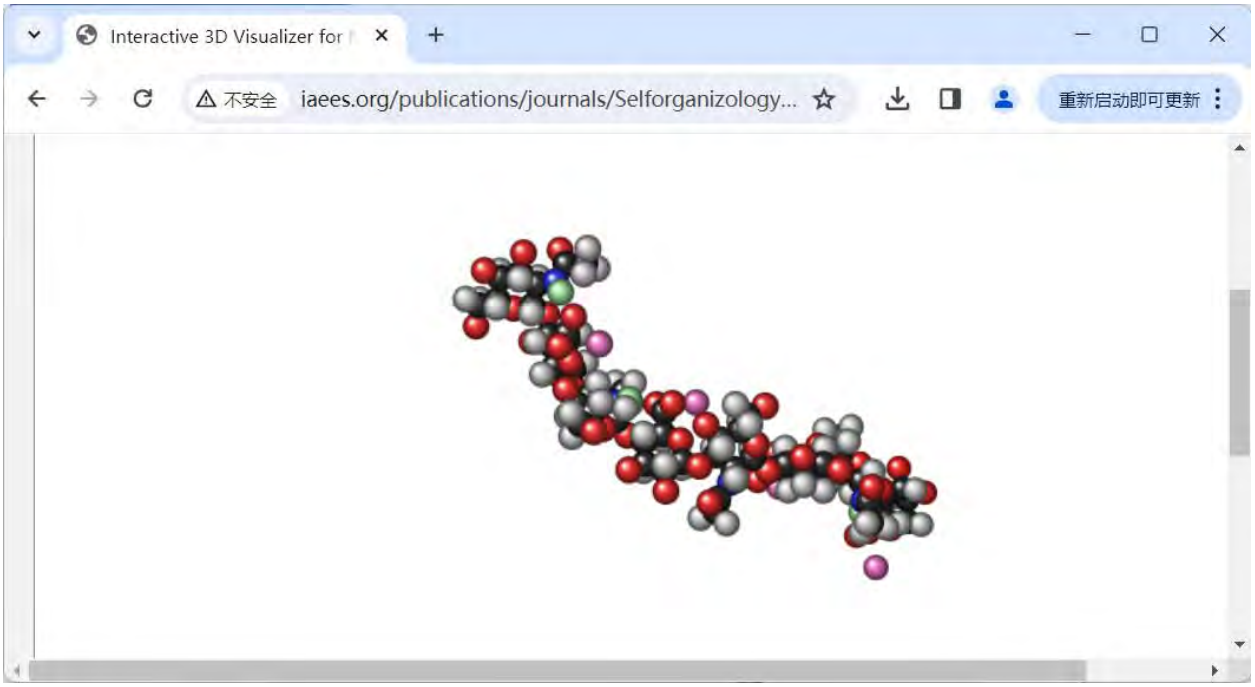
Fig. 1 The browser-based tool pages.

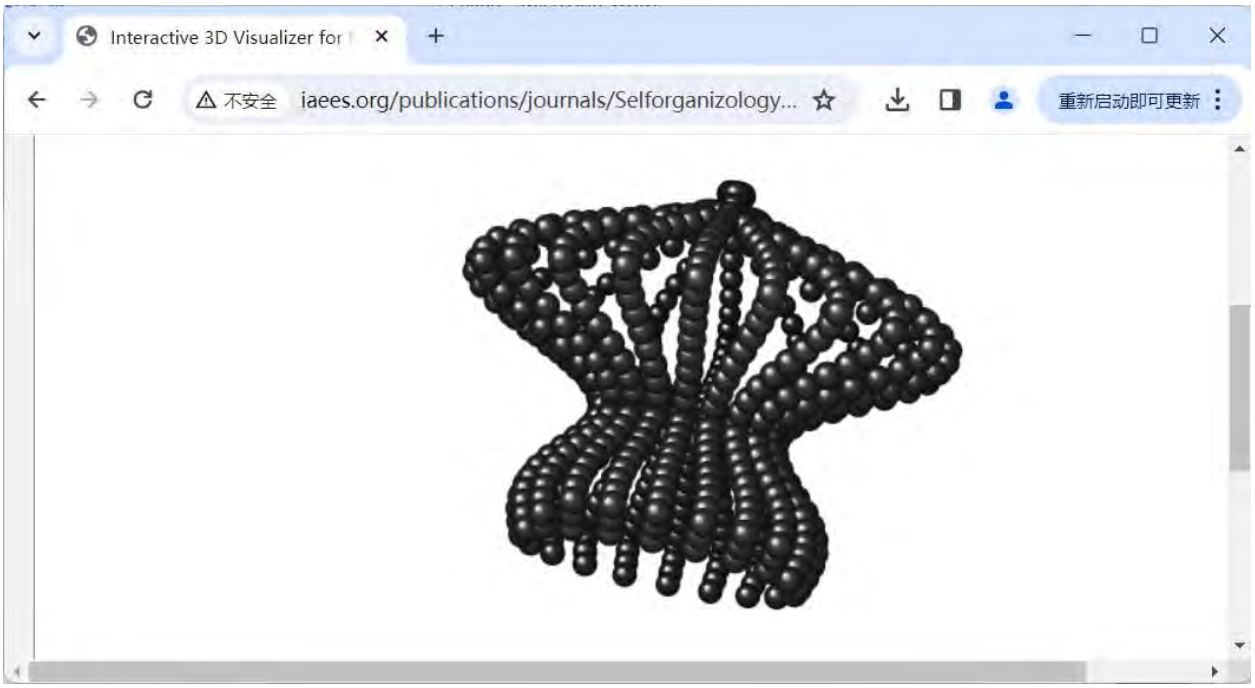
4 Demo

Using the downloaded sample data files (http://www.iaees.org/publications/journals/Selforganizology/articles/2028-15(1-2)/SampleDataFiles.zip), the program can be used to generate various structures (Fig. 2).


Interactive 3D Visualizer for
不安全
iaees.org/publications/journals/Selforganizology...
重新启动即可更新


Interactive 3D Visualizer for
不安全
iaees.org/publications/journals/Selforganizology...
重新启动即可更新


Interactive 3D Visualizer for
不安全
iaees.org/publications/journals/Selforganizology...
重新启动即可更新


Interactive 3D Visualizer for
不安全
iaees.org/publications/journals/Selforganizology...
重新启动即可更新

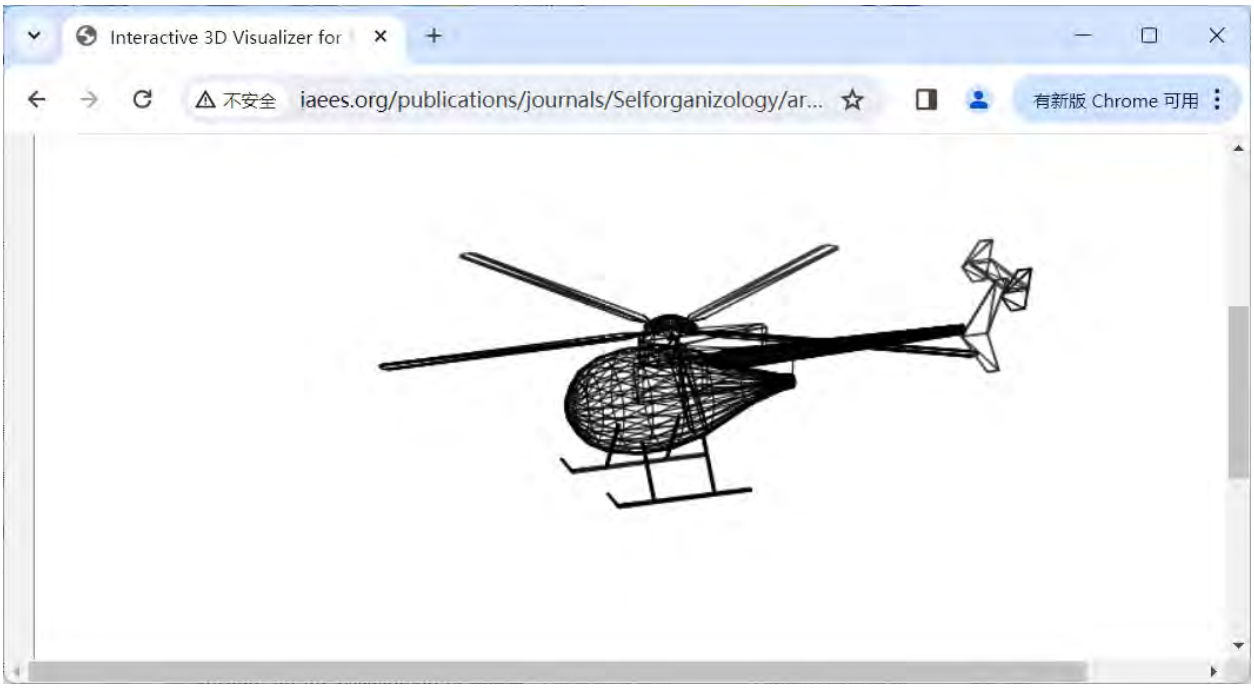


Fig. 2 Some structures generated by the tool.

References

Zhang WJ. 2012. A Java software for drawing graphs. Network Biology, 2(1): 38-44. http://www.iaees.org/publications/journals/nb/articles/2012-2(1)/a-Java-software-for-drawing-graphs.pdf

Zhang WJ. 2021. A web tool for generating user-interface interactive networks. Network Biology, 11(4): 247-262. http://www.iaees.org/publications/journals/nb/articles/2021-11(4)/web-tool-for-generating-user-interface-interactive-networks.pdf

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

Zhang WJ. 2024a. A Matlab software for visualizing user-interface interactive networks. Network Biology, 14(1): 13-19. http://www.iaees.org/publications/journals/nb/articles/2024-14(1)/2-Zhang-Abstract.asp

Zhang WJ. 2024b. A standalone executable software for network visualization. Network Pharmacology, 9(1-2): 1-10. http://www.iaees.org/publications/journals/np/articles/2024-9(1-2)/1-Zhang-Abstract.asp

Zhang WJ. 2024c. An executable Java software for visualizing networks. Network Biology, 14(1): 1-12. http://www.iaees.org/publications/journals/nb/articles/2024-14(1)/1-Zhang-Abstract.asp

Zhang WJ. 2024d. netGen 3.0: The executable Java software for network visualization. Selforganizology, 11(1-2): 1-27. http://www.iaees.org/publications/journals/selforganizology/articles/2024-11(1-2)/1-Zhang-Abstract.asp

Zhang WJ. 2024e. 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