NURBS Pipe Elbow Viewer

A browser-based 3D viewer for inspecting the inner and outer NURBS surfaces of a trivariate pipe-elbow control structure. The application embeds two 45-point radial control layers, their rational weights, three knot vectors, NURBS basis evaluation, tensor-product surface sampling, control-net rendering, interactive camera controls, and one-step circumferential knot refinement.

Main Features

- Interactive 3D pipe-elbow visualization
- Inner and outer NURBS surface layers
- Adjustable u/v mesh sampling resolution
- Control-net and control-point overlay
- Independent inner/outer surface visibility
- Circumferential knot insertion
- Homogeneous-coordinate refinement
- Control-point and knot-count diagnostics
- Mouse rotate, wheel zoom, and right-drag pan through OrbitControls

Embedded Control Data

The source embeds two arrays of control points:

- `P_layer1`: radial layer j=1
- `P_layer2`: radial layer j=2

Each layer contains 45 three-dimensional control points.

The code maps these into a trivariate control structure indexed as:

`P[i][j][k]`

with initial dimensions:

- circumferential controls: `n_u = 9`
- radial controls: `n_v = 2`
- axial controls: `n_w = 5`

Thus the initial control structure contains 9 x 2 x 5 positions.

Rational Weights

The application embeds a 45-value weight array for each radial layer.

The supplied values use combinations of:

- `1`
- `1 / sqrt(2)`
- `1 / 2`

The viewer evaluates the surfaces rationally by multiplying positions by their associated weights in homogeneous accumulation and dividing the accumulated xyz numerator by the accumulated weight.

Knot Vectors

The initial knot vectors are:

Circumferential U

`[0,0,0,1,1,2,2,3,3,4,4,4]`

Radial V

`[0,0,1,1]`

Axial W

`[0,0,0,1,1,2,2,2]`

The source normalizes all three knot vectors to the interval 0 through 1 before evaluation.

Degrees Used by the Source

The application sets:

- circumferential `p = 2`
- radial `q = 1`
- axial `r = 2`

The rendered inner and outer surface layers use the circumferential and axial directions. The radial layer is selected as layer 0 or layer 1 rather than continuously sampled between the two layers.

Control-Net Construction

`buildControlNet()` creates nested position and weight arrays for the 9 x 2 x 5 structure.

The 45 entries in each embedded layer are interpreted as nine circumferential blocks containing five axial entries each. Indexing follows:

```
index = i * 5 + k
```

Layer 1 is assigned to radial index 0 and Layer 2 to radial index 1.

NURBS Basis Functions

The application implements one-dimensional B-spline basis evaluation directly in JavaScript.

Span search

`findSpan()` locates the active knot span for a parameter value.

Cox-de Boor basis evaluation

`basisFuns()` computes the `p + 1` nonzero basis functions using left/right temporary arrays and the standard local recurrence structure.

The source describes this as a stable Cox-de Boor local algorithm.

Surface Evaluation

`evaluateSurfaceLayer()` samples one chosen radial layer as a tensor-product NURBS surface.

For every `(xi, zeta)` sample, it:

1. Finds the active circumferential span in U.
2. Evaluates circumferential basis values.
3. Finds the active axial span in W.
4. Evaluates axial basis values.
5. Multiplies the two basis contributions.
6. Retrieves the control point and rational weight.
7. Accumulates weighted homogeneous x, y, z, and weight.
8. Divides the weighted coordinates by the accumulated weight.

The resulting points are stored as mesh positions.

Mesh Resolution

Two sliders control sampled mesh density.

U resolution

- Minimum: 8
- Default: 28
- Maximum: 80

V resolution

- Minimum: 4
- Default: 20
- Maximum: 60

In the source, these represent circumferential and axial sampling counts for each displayed radial surface layer.

Changing either slider immediately rebuilds the rendered geometry.

Mesh Triangulation

Adjacent sampled points are joined into quadrilateral cells represented by two triangles.

For each mesh cell the source adds these triangle patterns:

- `(a, c, b)`
- `(b, c, d)`

The resulting index list is attached to a Three.js `BufferGeometry`.

Surface Normals

Surface normals are computed after sampling.

For every mesh triangle, the source:

1. Forms two triangle edge vectors.
2. Computes their cross product.
3. Adds that triangle normal to each of its three vertices.
4. Normalizes every accumulated vertex normal.

The normals are supplied to the surface material through a Three.js normal buffer attribute.

Inner Surface

The inner surface uses radial layer index 0.

Its material is configured as a light blue `MeshStandardMaterial` with:

- metalness 0.1
- roughness 0.5
- double-sided rendering
- opacity 0.95

The surface is visible by default.

Outer Surface

The outer surface uses radial layer index 1.

Its material is configured as an orange `MeshStandardMaterial` using the same metalness, roughness, sidedness, and opacity values as the inner surface.

The surface is visible by default.

Control Net

The Show control net checkbox is enabled initially.

When active, the application displays:

- Red spherical control points
- Circumferential connecting lines for each axial level in each radial layer
- Axial connecting lines for each circumferential index

Circumferential line color differs between the two radial layers, while axial control lines use a green-toned color.

The circumferential control-line construction closes each line back to its first point.

Surface Visibility

Two independent checkboxes control whether the application creates the inner and outer surface meshes:

- Show inner surface
- Show outer surface

Both are enabled on initial load.

Visibility changes trigger complete geometry rebuilding.

Geometry Rebuilding

Before generating new geometry, `buildMeshes()` removes prior meshes and control-net objects from the scene.

It also explicitly disposes of old Three.js geometries and materials before replacing them.

The method then rebuilds whichever surface layers and control-net elements are currently enabled and refreshes the diagnostic text.

Circumferential Knot Insertion

The interface provides one refinement action:

`Insert knot (split widest span)`

The corresponding `insertKnotU()` function inserts one knot in the circumferential U direction.

Choosing the Inserted Knot

The source scans consecutive U-knot values and locates the widest knot span.

The new knot value is selected as the midpoint of that span:

```
u = 0.5 * (U[k] + U[k+1])
```

The new knot is inserted immediately after the left side of the selected span.

Homogeneous Boehm-Style Refinement

The application converts each control point into four-dimensional homogeneous form:

```
[w*x, w*y, w*z, w]
```

For every radial layer and axial control position, it then builds a new circumferential control sequence.

Control points outside the affected refinement region are copied. Control points in the affected region are blended using:

```
alpha = (u - U[i]) / (U[i+p] - U[i])
```

and:

```
Q = alpha * H[i] + (1-alpha) * H[i-1]
```

The new homogeneous points are converted back to Euclidean coordinates by dividing xyz by the new weight.

After insertion:

- U contains one additional knot
- n_u increases by one
- the control-point/weight arrays are replaced by the refined versions
- the meshes and control net are rebuilt

The UI describes this operation as homogeneous-space Boehm insertion.

Diagnostics

The diagnostics line reports the current control structure and U-knot count in the form:

```
Control points: n_u x n_v x n_w | U knots: U.length
```

Initially this corresponds to the original 9 x 2 x 5 control structure and the initial U knot vector. Knot insertion increases the circumferential control count and knot count.

Three.js Scene

The viewer uses Three.js to create a WebGL scene.

The source initializes:

- `THREE.WebGLRenderer` with antialiasing
- `THREE.Scene`
- `THREE.PerspectiveCamera`
- `THREE.OrbitControls`
- one directional light
- one ambient light

The camera starts at:

`(12, 10, 18)`

with a 45-degree perspective field of view.

Lighting

The scene includes:

- Directional white light at `(10, 20, 10)` with intensity 1.0
- Ambient light with color `0x404050` and intensity 0.8

The renderer uses a dark blue-black clear color.

Camera Interaction

The application footer states:

- mouse drag: rotate
- scroll: zoom
- right-drag: pan

These interactions are provided through Three.js OrbitControls.

Render Loop and Responsive Canvas

The application continuously renders with `requestAnimationFrame`.

During each frame it compares the drawing-buffer dimensions with the canvas CSS client size. When they differ, it updates renderer size, camera aspect ratio, and projection matrix.

OrbitControls are updated before rendering the scene.

Error Banner

A global window error listener is registered before the interface and library imports.

If a script/runtime error occurs, the application creates a fixed red error banner and appends the error message or failed script URL information.

This is particularly relevant because the viewer depends on externally hosted Three.js resources.

External Dependencies

The HTML loads:

- Three.js 0.147.0 from jsDelivr
- Three.js OrbitControls for 0.147.0 from jsDelivr

The viewer therefore requires those external scripts to load successfully for its 3D rendering and camera controls.

Application Architecture

The application is delivered as a single HTML file containing:

- Embedded CSS
- Embedded control-point coordinates
- Embedded rational weights
- Embedded knot vectors
- Control-net construction
- Knot normalization
- B-spline span search
- Cox-de Boor basis evaluation
- Rational surface sampling
- Mesh topology construction
- Vertex-normal calculation
- Three.js scene and materials
- Control-net geometry
- Homogeneous knot insertion
- UI controls
- Continuous render loop
- Error reporting

Three.js and OrbitControls are the only external JavaScript dependencies referenced by the file.

Running the Application

Main file:

`NURBS_App_fixed (1).html`

Open the HTML file in a modern browser with JavaScript, WebGL, and access to the referenced jsDelivr resources.

Use the U and V resolution sliders to change surface sampling, toggle control-net and surface layers, manipulate the camera with the mouse, and use Insert knot to apply one circumferential refinement step at a time.

Implementation Note

The UI describes the geometry as a trivariate pipe elbow, and the source contains two radial control layers and a radial knot vector. The displayed surfaces are evaluated one radial layer at a time using the circumferential U direction and axial W direction. The source does not continuously sample the radial V direction to create a volume or side wall between the two layers.

Author of This Web File

Joyce Cunningham