N7 PROGRAMMING LANGUAGE EXTENSION

Simple 3D Library (s3d) – API Reference

Introduction

The **Simple 3D library (`s3d`)** enables you to set up a 3D viewport and render triangles and quadrilaterals in N7. It is a pure software renderer computed entirely on the CPU (without DirectX or OpenGL), so keeping polygon counts genuinely low is essential for good performance.

Transformations & Coordinate System

A position in 3D space is defined by `(x, y, z)` coordinates:

- **X-axis:** Points **right**
- **Y-axis:** Points **down**
- **Z-axis:** Points **forward** (away from the camera)

Imagine standing at the origin `(0, 0, 0)` looking along the positive Z-axis. To place a box 4 units in front of you and rotate it 45 degrees around its Y-axis, you walk to the position (**translation**), turn (**rotation**), and place the object:

```
S3D_Push()
S3D_Translate(0, 0, 4)
S3D_RotateY(rad(45))
S3D_Mesh(myBox, 0)
S3D_Pop()
```

You can also resize objects using **scaling** (e.g., `S3D_Scale(1, 2, 1)` to double height). Instead of manually reversing every movement and rotation to return to `(0, 0, 0)`, use `S3D_Push()` to save your current transformation state and `S3D_Pop()` to restore it afterward.

View, Depth Buffer & Render Settings

`S3D_SetView(``(number)` **`targetImage,`** `(number)` **`fov,`** `(number)` **`zMin,`** `(number)` **`zMax)`**

Configures the 3D rendering viewport and projection.

- `targetImage` – Target image to render onto (usually `primary` for the main window).
- `fov` – Vertical field of view in **radians**.
- `zMin` / `zMax` – Near and far clipping planes. Geometry closer than `zMin` or farther than `zMax` is clipped or discarded.

`S3D_SetPerspectiveCorrection(number mode)`

Sets texture-mapping perspective correction quality.

Constant	Description
`S3D_NONE`	No perspective correction (not recommended).
`S3D_FAST` *(default)*	Low-quality perspective correction.
`S3D_NORMAL`	Medium-quality perspective correction.
`S3D_NICE`	High-quality perspective correction.

`S3D_SetDepthBuffer(number mode)`

Configures Z-buffer depth testing per pixel.

Constant	Description
`S3D_NONE`	Disables the Z-buffer completely.
`S3D_Z_BUFFER` *(default)*	Draws and updates the Z-buffer only if the new pixel is closer to the camera.
`S3D_Z_WRITE`	Draws and updates the Z-buffer unconditionally without comparing depth.
`S3D_Z_READ`	Draws only if the new pixel is closer, without updating the Z-buffer.

`S3D_ClearDepthBuffer()`

Resets all pixels in the depth buffer to the maximum possible distance.

`S3D_SetSorting(number mode)`

Configures automatic polygon depth sorting before rendering (useful as an alternative to a depth buffer in certain scenes, such as heightmaps). When sorting is enabled, polygons are queued until `S3D_Render()` is called.

Constant	Description
`S3D_NONE` *(default)*	No sorting; polygons render immediately.
`S3D_BACK_TO_FRONT`	Renders polygons from farthest to closest.

Constant	Description
`S3D_FRONT_TO_BACK`	Renders polygons from closest to farthest.

`S3D_Clear()`

Resets the current transformation and clears the transformation stack, clears the depth buffer, and resets the `s3d` draw color to opaque white `(255, 255, 255, 255)`.

Transformations & Matrix Stack

`S3D_ClearTransformation()`

Resets the current transformation to `(0, 0, 0)` looking down the positive Z-axis (X right, Y down) without clearing the stack or depth buffer.

`S3D_Translate(``(number)` **`x,`** `(number)` **`y,`** `(number)` **`z)`**

Moves the current transformation by `(x, y, z)` relative to its current position and orientation.

`S3D_RotateX(``(number)` **`a)`**

Rotates `a` radians around the X-axis relative to the current transformation.

`S3D_RotateY(``(number)` **`a)`**

Rotates `a` radians around the Y-axis relative to the current transformation.

`S3D_RotateZ(``(number)` **`a)`**

Rotates `a` radians around the Z-axis relative to the current transformation.

`S3D_Scale(``(number)` **`x,`** `(number)` **`y,`** `(number)` **`z)`**

Scales coordinates by `(x, y, z)` relative to the current transformation.

`S3D_Push()`

Pushes a copy of the current transformation onto the transformation stack.

`S3D_Pop()`

Pops the top transformation from the stack and restores it as the active transformation.

Immediate-Mode Drawing & Materials

> **S3D_Begin(**(number) **mode)**

Begins defining polygons of type `mode`. If polygon sorting is disabled (`S3D_NONE`), each triangle or quad renders immediately after every 3rd or 4th vertex call; otherwise, polygons are queued until `S3D_Render()`.

- `S3D_TRIANGLES` – Each triangle requires 3 vertex calls.
- `S3D_QUADS` – Each quadrilateral requires 4 vertex calls.

> **S3D_End()**

Ends the current polygon batch started with `S3D_Begin()`. Every `S3D_Begin()` must have a matching `S3D_End()`, and nesting is not allowed.

> **S3D_Vertex(**(number) **x,** (number) **y,** (number) **z,** (number) **u,** (number) **v)**
> **S3D_Vertex4(**(number) **x,** (number) **y,** (number) **z,** (number) **u,** (number) **v)**

Adds a vertex at `(x, y, z)` with normalized texture coordinates `(u, v)` in the range `[0..1]`, where `(0, 0)` is the top-left corner of the texture and `(1, 1)` is the bottom-right corner.

> **Winding Order & Backface Culling:** Polygons facing away from the camera are culled (not rendered). Front-facing vertices must be specified in **clockwise** order (given X right, Y down). For example, a 1×1 square centered at `z = 3` facing the camera:

```
S3D_Vertex(-0.5, -0.5, 3, 0, 0)
S3D_Vertex( 0.5, -0.5, 3, 1, 0)
S3D_Vertex( 0.5,  0.5, 3, 1, 1)
S3D_Vertex(-0.5,  0.5, 3, 0, 1)
```

> **S3D_Vertex3(**(number) **x,** (number) **y,** (number) **z)**

Adds a vertex at `(x, y, z)` with texture coordinates `(u, v)` defaulted to `(0, 0)`.

> **S3D_Color(**(number) **r,** (number) **g,** (number) **b)**
> **S3D_Color3(**(number) **r,** (number) **g,** (number) **b)**

Sets the active RGB color (`[0..255]`) and resets alpha to `255` for subsequent polygons. When applied to textured polygons, it tints the texture (note that tinted textured polygons are significantly slower to render than untinted ones). Use `S3D_Color(255, 255, 255)` to disable tinting.

```
S3D_Color4(number) r, (number) g, (number) b, (number) a)
```

Sets the active RGBA color and opacity (`[0..255]`). When using alpha transparency (`a < 255`), non-opaque polygons must be rendered from **back to front**.

```
S3D_Additive(number) value)
```

Enables additive color blending when `value` is `true` , or disables it when `false` .

```
S3D_Texture(number) img)
```

Sets the active texture for subsequent polygons using an N7 image ID (`img`).

```
S3D_Render()
```

Sorts and renders all queued polygons when polygon sorting is active.

```
S3D_RenderFog(number) r, (number) g, (number) b, (number) retro)
```

Applies a post-process depth fog effect across rendered pixels using the Z-buffer (requires `S3D_Z_BUFFER` or `S3D_Z_WRITE`). Pixels at `zMax` blend fully into `(r, g, b)` , while closer pixels are less affected.

- `r` , `g` , `b` – Fog color components in the range `[0..255]` .
- `retro` – Set to `true` for quantized, sharp-stepped retro fog bands; `false` for smooth fog.

Meshes & Vertex Animation

> Rendering pre-built or loaded meshes is substantially faster than submitting individual vertices every frame, as meshes allow early polygon rejection and internal optimizations.

```
(number) S3D_LoadMesh(string) filename, (number) scaleX, (number) scaleY, (number) scaleZ, (number) invertFaces)
```

Loads a 3D model from a Wavefront `.obj` file (including diffuse colors `Kd` and textures `map_Kd` from its `.mtl` file). Because `s3d` 's coordinate system differs from OpenGL's, `scaleY` typically needs to be negative. Keep polygon counts very low (classic Quake-era complexity).

- `filename` – Path to the `.obj` file.
- `scaleX` , `scaleY` , `scaleZ` – Static scale factors baked into the mesh upon loading.
- `invertFaces` – Set to `true` to flip face winding if the mesh renders inside-out.
- **Returns:** Mesh identifier on success, or `unset` on failure.

> **(number) S3D_LoadMeshFrame(**(number) **mesh,** (string) **filename,** (number) **scaleX,** (number) **scaleY,** (number) **scaleZ)**

Loads vertex positions from a separate `.obj` file and appends them as an animation keyframe to an existing `mesh`. The `.obj` file must represent the same model with the exact same vertex count and vertex order.

- `mesh` – Target mesh identifier.
- `filename` – Path to the keyframe `.obj` file.
- `scaleX`, `scaleY`, `scaleZ` – Scale factors matching those used when loading the base mesh.
- **Returns:** `true` on success.

> **S3D_Mesh(**(number) **mesh,** (number) **frame)**

Renders `mesh` using the active transformation.

- `mesh` – Mesh identifier.
- `frame` – Keyframe index (`0` for the base mesh, `1` for the first added frame, etc.).

> **S3D_BlendMesh(**(number) **mesh,** (number) **frame0,** (number) **frame1,** (number) **blend)**

Renders `mesh` by linearly interpolating vertices between two keyframes.

- `mesh` – Mesh identifier.
- `frame0` / `frame1` – Keyframe indices to blend between.
- `blend` – Interpolation factor in the range `[0..1]` (`0` = 100% `frame0`, `1` = 100% `frame1`, `0.75` = 25% `frame0` + 75% `frame1`).

> **(number) S3D_BeginMesh()**

Starts recording polygons into a reusable mesh object. Between `S3D_BeginMesh()` and `S3D_EndMesh()`, you can call `S3D_Begin()` / `S3D_End()`, `S3D_Vertex()`, `S3D_Color()`, transformations, and even draw other meshes:

```
squareMesh = S3D_BeginMesh()
S3D_Begin(S3D_QUADS)
    S3D_Vertex(-0.5, -0.5, 0, 0, 0)
    S3D_Vertex( 0.5, -0.5, 0, 1, 0)
    S3D_Vertex( 0.5,  0.5, 0, 1, 1)
    S3D_Vertex(-0.5,  0.5, 0, 0, 1)
S3D_End()
S3D_EndMesh()
```

- **Returns:** Mesh identifier for the recorded mesh.

```
S3D_EndMesh()
```

Stops mesh recording started with `S3D_BeginMesh()`.

```
(number) S3D_CreateMesh((array) vertexList, (array) uvList, (array) materialList, (array) faceList)
```

Low-level mesh constructor used internally by `S3D_LoadMesh()`.

- `vertexList` – Array of 3D coordinates `[[x0, y0, z0], [x1, y1, z1], ...]`. Must not be `unset` or empty.
- `uvList` – Array of UV pairs `[[u0, v0], [u1, v1], ...]`. Must not be `unset` or empty (use `[[0, 0]]` if unused).
- `materialList` – Array of materials `[[[r, g, b[, a]], texture], ...]`. May be `unset` or empty; individual color or texture entries may also be `unset`.
- `faceList` – Array of faces containing 3 (triangle) or 4 (quad) vertex indices, matching UV indices, and a material index: `[[v0, v1, v2[, v3], uv0, uv1, uv2[, uv3], matIdx], ...]` (`matIdx` may be `unset`; UV indices must be valid).
- **Returns:** Mesh identifier on success.

```
S3D_AddMeshFrame((number) mesh, (array) vertexList)
```

Low-level function used internally by `S3D_LoadMeshFrame()` to append a vertex list (`[[x0, y0, z0], ...]`) as a new animation frame to `mesh`. Must match the original mesh's vertex count.

```
S3D_FreeMesh((number) mesh)
```

Frees and disposes of `mesh` from memory.

Math, Projection & Visibility Helpers

```
S3D_TransformVector((array) dst, (array) src)
```

Multiplies vector `src` (`[x, y, z[, w=1]]`) by the current transformation matrix and stores the result in `dst`.

```
(number) S3D_ProjectVector((array) dst, (array) src)
```

Transforms and projects 3D vector `src` (`[x, y, z[, w=1]]`) into 2D screen coordinates in `dst` (`x` and `y` may fall outside `[0..width)`). Useful for visibility or UI marker checks.

- **Returns:** `true` on success, or `false` if the transformed `z` coordinate is behind the near clipping plane (`zMin`).

```
(number) S3D_ProjectFace((array) dst, (array) src)
```

Transforms, clips against the near plane, and projects a 3- or 4-vertex polygon `src` (`[x0, y0, z0, x1, y1, z1, ...]`) into 2D screen coordinates in `dst` . Useful for portal or frustum visibility tests.

- **Returns:** Number of vertices in the resulting clipped polygon (`0` if all vertices lie behind the near clipping plane).