Contents

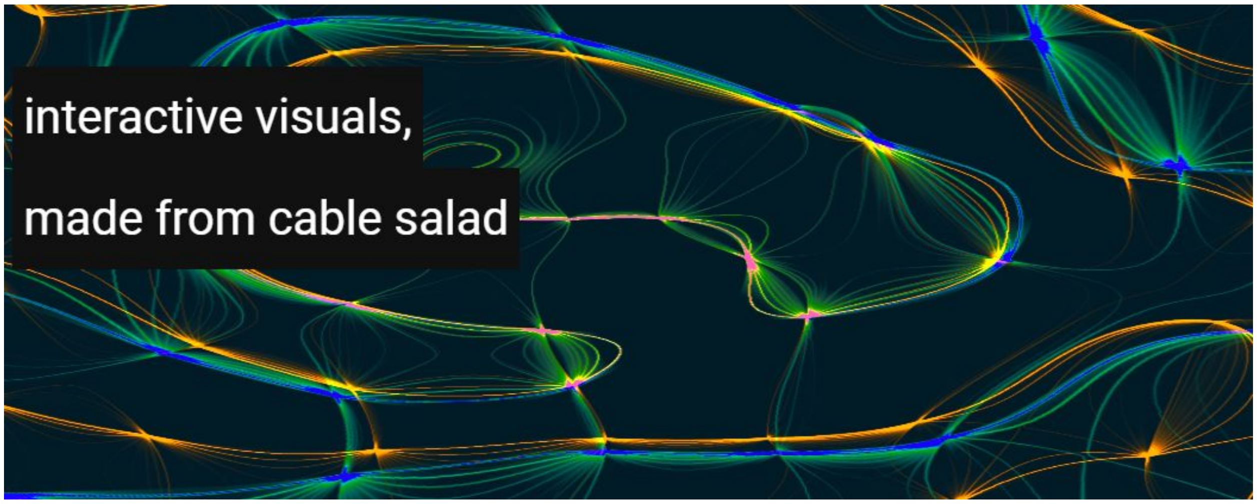
1 The Cables.gl Book

Alexandre Rangel, 2025

---

2 Introduction to Cables.gl

2.1 What is Cables.gl?



Cables.gl is a powerful, browser-based visual programming environment for creating interactive 2D and 3D graphics using WebGL. It was created by undev in Berlin and has become a popular tool for creative coding, interactive installations, data visualization, and web-based visual experiences.

Unlike traditional coding environments, cables.gl uses a **node-based** (or "patch-based") approach where you connect visual operators (ops) together to create your projects. This makes it accessible to artists and designers while still being powerful enough for developers.

2.2 A (Brief) History of cables.gl

cables.gl was created by **undev** (Berlin) with the goal of making **real-time WebGL** creation approachable through a node-based workflow—similar in spirit to visual programming environments used in motion design and interactive installations, but built for the browser.

Over time, cables.gl grew from a tool for quick experiments into a full ecosystem:

- **Early days**: a strong focus on rapid prototyping and sharing patches online.
- **Maturing platform**: a steadily growing op library for 2D, 3D, textures, audio, and interaction, plus better tooling (timeline, profiling/debugging utilities, export options).
- **Community-driven growth**: more public patches, tutorials, Discord knowledge-sharing, and reusable patterns (e.g., render-to-texture workflows, post-processing chains, audio-

reactive setups).
- **Production use**: cables.gl exports make it viable for deployment in websites, installations, and client work—where performance, asset management, and reliable runtime behavior matter.

If you're coming from traditional code, it helps to think of cables.gl as a **visual runtime graph**: triggers define *when* things run; value connections define *what data* flows; and the patch as a whole becomes a web-ready app.

2.3 Why Use Cables.gl?

2.3.1 Visual Programming

- No coding required to get started
- Drag-and-drop interface
- See results in real-time as you build

2.3.2 Browser-Based

- No installation needed
- Works on any modern browser
- Collaborate and share easily

2.3.3 High Performance

- Built on WebGL for GPU-accelerated graphics
- Optimized for real-time rendering
- Handles complex 3D scenes smoothly

2.3.4 Export Options

- Standalone HTML/JS builds
- Embed in websites
- Create offline applications

2.3.5 Extensible

- Write custom operators (ops) in JavaScript

- GLSL shader support
- Import external libraries

2.4 Key Concepts

2.4.1 Operators (Ops)

The building blocks of cables.gl. Each op performs a specific function - from drawing shapes to processing audio to handling user input.

2.4.2 Patches

A patch is your complete project - a collection of ops connected together to create your visual experience.

2.4.3 Ports

Ops have input and output ports. You connect ports together with "cables" (hence the name!) to pass data between ops.

2.4.4 Types of Ports

- **Trigger** (grey) - Execution flow, like "when to do something"
- **Number** (green) - Numerical values
- **String** (yellow) - Text values
- **Object** (blue) - Complex data like meshes, textures, arrays
- **Array** (cyan) - Collections of data

2.5 Featured Videos

2.5.1 Overview and Getting Started

```
https://youtu.be/hVxrxXhH7vQ
Title: Cables.gl Standalone (Offline) Build: Create Without Limits!
Author: Decode GL
Thumbnail: https://i.ytimg.com/vi/hVxrxXhH7vQ/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@Decode_gl
```

```
https://youtu.be/goO3PhuenBI
Title: First Steps in Cables.gl - Tutorial
Author: The Interactive & Immersive HQ
Thumbnail: https://i.ytimg.com/vi/goO3PhuenBI/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@TheInteractiveImmersiveHQ
```

```
https://youtu.be/xnObNRv8n9I
Title: Introduction to cables.gl - Data-Driven Gradient from Geo-Located Weather -
Part 0
Author: Kirell Benzi
Thumbnail: https://i.ytimg.com/vi/xnObNRv8n9I/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@kirellbenzi
```

2.5.2 More Resources

Note: There are limited intro-specific YouTube videos for cables.gl, but the platform has excellent resources: - Browse the cables.gl Public Patches to see examples - Check the official cables.gl YouTube channel for official tutorials - The Decode GL channel has multiple cables.gl tutorials - Search for "cables.gl" on YouTube for the latest community content - Many cables.gl creators share their work on social media and personal channels

2.6 Getting Help

- **Official Documentation**: cables.gl/docs
- **Example Patches**: Browse public patches for inspiration

---

3 Getting Started with Cables.gl

3.1 Creating Your Account

1. Go to cables.gl
2. Click "Sign Up" to create a free account
3. Verify your email
4. You're ready to start creating!

3.2 The Interface

3.2.1 Main Areas

```
+-------------------------------------------------------------+
|                       TOOLBAR                               |
+----------------------+--------------------------------------+
|                      |                                      |
|      OP LIBRARY      |          CANVAS (Preview)           |
|                      |                                      |
+----------------------+                                      |
|                      |                                      |
|    PATCH EDITOR      |                                      |
|   (Node workspace)   |                                      |
|                      |                                      |
+----------------------+--------------------------------------+
```

3.2.2 Key Interface Elements

1. **Canvas** - Live preview of your creation
2. **Patch Editor** - Where you place and connect ops
3. **Op Library** - Browse and search for operators
4. **Parameters Panel** - Adjust settings for selected ops
5. **Timeline** - For animation keyframes

3.3 Navigation Controls

Action	Control
Pan the view	Middle mouse drag or Space + drag
Zoom in/out	Mouse scroll wheel
Select op	Left click
Multi-select	Shift + click or drag box
Delete selected	Delete or Backspace
Add new op	Double-click or Tab
Connect ports	Drag from output to input

3.4 Keyboard Shortcuts

Mastering keyboard shortcuts will significantly speed up your workflow in cables.gl.

3.4.1 Essential Shortcuts

Shortcut	Action
Tab or **Double-click**	Add new op (opens search)
Delete or **Backspace**	Delete selected op(s)
Ctrl + C / **Cmd + C**	Copy selected op(s)
Ctrl + V / **Cmd + V**	Paste op(s)
Ctrl + X / **Cmd + X**	Cut selected op(s)
Ctrl + D / **Cmd + D**	Duplicate selected op(s)
Ctrl + Z / **Cmd + Z**	Undo
Ctrl + Shift + Z / **Cmd + Shift + Z**	Redo

3.4.2 Selection & Navigation

Shortcut	Action
Ctrl + A / **Cmd + A**	Select all ops
Shift + Click	Add to selection
Ctrl + Click / **Cmd + Click**	Toggle selection
Escape	Deselect all
Space + Drag	Pan the patch view

Shortcut	Action
Mouse Wheel	Zoom in/out
Ctrl + 0 / **Cmd + 0**	Zoom to fit all ops
F	Focus/frame selected op(s)

3.4.3 Organizing & Aligning

Shortcut	Action
Ctrl + Shift + A / **Cmd + Shift + A**	Align selected ops horizontally
Ctrl + Shift + D / **Cmd + Shift + D**	Distribute selected ops evenly
Ctrl + G / **Cmd + G**	Group selected ops
Arrow Keys	Nudge selected op(s) by small amount
Shift + Arrow Keys	Nudge selected op(s) by larger amount

3.4.4 Working with Ops

Shortcut	Action
Enter	Open/edit selected op's parameters
Ctrl + E / **Cmd + E**	Enable/disable selected op
Ctrl + M / **Cmd + M**	Mute selected op
R	Rename selected op
C	Add comment node
Ctrl + F / **Cmd + F**	Find/search ops in patch

3.4.5 Cables & Connections

Shortcut	Action
Drag from port	Create connection
Click connection	Select cable
Alt + Click connection	Delete cable
Shift + Drag port	Create cable with search

3.4.6 View & Interface

Shortcut	Action
T	Toggle timeline
Ctrl + / / **Cmd + /**	Toggle op library
H	Toggle patch editor visibility
P	Toggle parameters panel
Ctrl + S / **Cmd + S**	Save patch
Ctrl + Shift + S / **Cmd + Shift + S**	Save as…

3.4.7 Performance & Debugging

Shortcut	Action
Ctrl + Shift + P / **Cmd + Shift + P**	Performance monitor
Ctrl + Shift + L / **Cmd + Shift + L**	Show patch loading info
Alt + Click op	View op documentation

3.4.8 Pro Tips

- **Hold Shift while connecting**: Automatically opens op search to insert an op in the connection
- **Hold Alt while dragging**: Duplicate op while moving
- **Double-click a connection**: Insert a new op in that cable
- **Right-click an op**: Quick access to op menu (rename, mute, group, etc.)
- **Click and drag in empty space**: Selection box for multiple ops

3.5 Your First Patch

Let's create a simple animated shape!

3.5.1 Step 1: Create the Render Pipeline

1. Double-click in the patch editor to open the op search
2. Search for `MainLoop` and add it
3. The MainLoop is the heartbeat of your patch - it triggers every frame

3.5.2 Step 2: Add a BasicMaterial

1. Add a `BasicMaterial` op
2. Connect `MainLoop`'s trigger output to `BasicMaterial`'s trigger input
3. You should see a black canvas appear

3.5.3 Step 3: Draw a Circle

1. Add a `Circle` op
2. Connect `BasicMaterial` -> `Circle`
3. A white circle appears!

Here's what your patch should look like:

Visualization Options

Option 1: Screenshot from Real Cables.gl (Most Authentic)

See Screenshot Guide for instructions on capturing real cables.gl patches.

Option 3: HTML/CSS Interactive

Open HTML Version in your browser for an interactive view.

Option 4: Mermaid Diagram (Simple Flow)

```
graph TD
    MainLoop[MainLoop] -->|trigger| BasicMaterial[BasicMaterial]
    BasicMaterial -->|trigger| Circle[Circle]

    style MainLoop fill:#2d2d2d,stroke:#4a4a4a,color:#e0e0e0
    style BasicMaterial fill:#2d2d2d,stroke:#4a4a4a,color:#e0e0e0
    style Circle fill:#2d2d2d,stroke:#4a4a4a,color:#e0e0e0
```

The basic render chain: MainLoop triggers the BasicMaterial, which then draws the Circle

3.5.4 Step 4: Add Color

1. Select the `BasicMaterial` op
2. Adjust the color values (r, g, b) in the parameters panel
3. Or connect a `SetColor` op's output to `BasicMaterial`'s color input ports

4. The circle will display with your chosen color

3.5.5 Step 5: Animate It

1. Add a `Time` op (outputs current time)
2. Add a `Math` op (for calculations)
3. Add a `Sin` op (sine wave)
4. Connect: `Time` -> `Sin` -> Circle's `Scale` input
5. Watch your circle pulse!

3.6 Understanding the Flow

Data flows from **top to bottom** and **left to right**:

```
MainLoop (starts the frame)
    |
BasicMaterial (defines appearance and color)
    |
Circle (draws the shape)
```

The **trigger** connection (grey) determines WHEN things happen. The **value** connections (colored) determine WHAT values are used.

3.7 Saving Your Work

- Patches auto-save regularly
- Click the save icon to force a save
- Use "Save As" to create copies
- Export for standalone deployment

3.8 Tips for Beginners

1. **Start Simple** - Begin with basic shapes before complex 3D
2. **Explore Examples** - Study public patches to learn patterns
3. **Use Comments** - Add comment ops to document your work
4. **Name Your Ops** - Rename ops for clarity in complex patches
5. **Save Often** - And use versioning for major changes

3.9 Featured Videos

3.10 Common First-Patch Issues

3.10.1 "I don't see anything!"

- Make sure MainLoop is connected to BasicMaterial
- Check that your shape ops are connected in the chain
- Verify the canvas is visible (not minimized)

3.10.2 "Colors aren't changing!"

- Check BasicMaterial's color values (r, g, b) in the parameters panel
- Make sure RGB values aren't all 0 (black)
- If using SetColor, connect it to BasicMaterial's color input ports

3.10.3 "Animation isn't working!"

- Ensure Time op is connected
- Check that the animated value is actually changing (view the port value)

---

4 2D Graphics in Cables.gl

4.1 Introduction to 2D Drawing

Cables.gl excels at creating stunning 2D graphics and animations, from simple shapes to complex generative art. This comprehensive chapter covers fundamental 2D drawing operations, advanced transformations, interactive elements, feedback loops, post-processing effects, and professional techniques for creating production-ready 2D visuals.

Whether you're creating data visualizations, interactive installations, or generative art, this chapter will give you the tools and knowledge to master 2D graphics in cables.gl.

4.2 Basic Shapes

4.2.1 Circle

The `Circle` op is one of the most common 2D primitives.

Key Parameters: - `Radius` - Size of the circle - `Segments` - Smoothness (more segments = smoother circle) - `Inner Radius` - Creates a ring when > 0

4.2.2 Rectangle

The `Rectangle` op draws rectangular shapes.

Key Parameters: - `Width` - Horizontal size - `Height` - Vertical size - `Pivot` - Origin point for positioning

4.2.3 RoundedRectangle

A rectangle with smooth corners.

Key Parameters: - `Width` / `Height` - Dimensions - `Corner Radius` - How rounded the corners are

4.2.4 Polygon

Create regular polygons (triangles, pentagons, etc.)

Key Parameters: - `Sides` - Number of sides (3 = triangle, 5 = pentagon, etc.) - `Radius` - Size of the polygon

4.2.5 Line / Lines

Draw single or multiple lines.

Key Parameters: - Start and End coordinates - Line width - Line style (solid, dashed)

4.3 Color and Appearance

4.3.1 SetColor

Changes the drawing color for subsequent shapes.

```
MainLoop -> BasicMaterial -> Circle
```

Connect `SetColor` output to `BasicMaterial`'s color input ports (r, g, b, a) to set the color.

Color Modes: - RGB (Red, Green, Blue) - HSB (Hue, Saturation, Brightness) - Hex values

4.3.2 SetAlpha

Controls transparency.

```
MainLoop -> BasicMaterial -> Shape
```

Connect `SetAlpha` output to `BasicMaterial`'s alpha (a) input port to control transparency.

Values range from 0 (invisible) to 1 (fully opaque).

4.3.3 Gradients

Use texture-based gradients or shader-generated gradients for smooth color transitions.

4.4 Transformations

4.4.1 Transform

The `Transform` op modifies position, rotation, and scale of all following shapes.

```
+--------------------------------------------------------------+
|                    TRANSFORM PIPELINE                        |
+--------------------------------------------------------------+
|                                                              |
|  MainLoop                                                    |
|    |                                                         |
|  BasicMaterial                                               |
|    |                                                         |
|  Transform                                                   |
|    +-> Position (X, Y, Z)                                    |
|    +-> Rotation (X, Y, Z)                                    |
|    +-> Scale                                                 |
|    |                                                         |
|  Shape (Circle, Rectangle, etc.)                             |
|                                                              |
+--------------------------------------------------------------+
```

Parameters: - `TranslateX`, `TranslateY`, `TranslateZ` - Position - `RotateX`, `RotateY`, `RotateZ` - Rotation (degrees) - `Scale` - Uniform scaling

4.4.2 Transformation Order Matters!

Transformations are applied in order. These produce different results:

Rotate then Translate:

```
+--------------------------------------------------------------+
|              ROTATE THEN TRANSLATE                           |
+--------------------------------------------------------------+
|                                                              |
|  Original Position                                           |
|      o                                                       |
|                                                              |
|  Step 1: Rotate 45 degrees                                   |
|      o                                                       |
|      (rotate)                                                |
|                                                              |
|  Step 2: Translate Right                                    |
|            -> o                                               |
|                                                             |
|  Result: Object rotates around origin, then moves           |
|                                                             |
+-----------------------------------------------------------+
```

```
Transform (rotate) -> Transform (translate) -> Shape
```

Translate then Rotate:

```
+-----------------------------------------------------------+
|              TRANSLATE THEN ROTATE                        |
+-----------------------------------------------------------+
|                                                             |
|  Original Position                                          |
|      o                                                      |
|                                                             |
|  Step 1: Translate Right                                    |
|            -> o                                              |
|                                                             |
|  Step 2: Rotate 45 degrees (around new position)            |
|            (rotate) o                                       |
|                                                             |
|  Result: Object moves first, then rotates around new origin  |
|                                                             |
+-----------------------------------------------------------+
```

```
Transform (translate) -> Transform (rotate) -> Shape
```

4.4.3 Nested Transforms

Create hierarchies by chaining transforms:

```
Transform (parent)
    |
Transform (child)
    |
Shape
```

The child inherits and adds to the parent's transformations.

4.5 Blending Modes

4.5.1 SetBlending

Controls how colors combine when shapes overlap.

Common Modes: - `Normal` - Standard opacity blending - `Add` - Colors add together (great for glow effects) - `Multiply` - Colors multiply (darkening effect)

4.5.2 Depth Testing

For 2D, you often want to disable depth testing:

```
MainLoop -> BasicMaterial -> DepthTest (disabled) -> Your 2D Content
```

This ensures draw order matches your connection order.

4.6 Patterns and Repetition

4.6.1 IteratorLoop

Create patterns by repeating shapes:

```
MainLoop -> IteratorLoop -> [Your Shape Setup]
```

Use the iterator index to offset position, color, or other properties.

4.6.2 ArrayIterator

Iterate over data arrays to position multiple shapes.

4.7 Text Rendering

4.7.1 DrawText

Display text in your patches.

Key Parameters: - `Text` - The string to display - `Font` - Font family - `Size` - Text size - `Alignment` - Left, center, right

4.7.2 TextTexture

Create textures from text for more advanced effects.

4.8 Advanced Transformation Techniques

4.8.1 Matrix Transformations

For precise control, work directly with transformation matrices:

```
MatrixMultiply -> Combine multiple transformations
MatrixInvert -> Reverse a transformation
```

4.8.2 Pivot Points

Control the center of rotation and scaling:

```
Transform (set pivot) -> Transform (rotate) -> Shape
```

Common Pivot Values: - `0, 0` - Bottom left corner - `0.5, 0.5` - Center (default) - `1, 1` - Top right corner

4.8.3 Compound Transformations

Build complex motion by layering transforms:

Example: Orbital Motion

```
Transform (parent orbit)
    |
Transform (child rotation)
```

```
    |
Transform (child offset)
    |
Shape
```

This creates a shape that orbits while rotating on its own axis.

4.9 Interactive 2D Elements

4.9.1 InteractiveRectangle

Create draggable, clickable UI elements:

```
InteractiveRectangle
    | (outputs X, Y, Width, Height on interaction)
Control other ops with mouse input
```

Use Cases: - On-screen sliders - Draggable controllers - Interactive buttons - Touch-enabled interfaces

4.9.2 Mouse Input

Capture and use mouse position:

```
Mouse -> Map (screen to world coords) -> Visual property
```

Mouse Ops: - `MouseX` / `MouseY` - Cursor position - `MouseButton` - Click detection - `MouseWheel` - Scroll input

4.9.3 Example: Interactive Color Picker

```
MainLoop
    |
MouseX -> Map (0 to 1) -> Hue
MouseY -> Map (0 to 1) -> Brightness
    |
HSBtoRGB -> BasicMaterial (color input)
    |
FullscreenRectangle
```

4.10 Generative Art Techniques

4.10.1 Feedback Loops

Create evolving, self-referential visuals by feeding output back as input:

Basic Feedback Setup:

```
MainLoop
    |
RenderToTexture (previous frame)
    |
ImageCompose (blend with new content)
    |
Transform (slight scale/rotate)
    |
TextureEffects (blur, fade)
    |
Draw new shapes
    |
Output (becomes next frame's input)
```

Parameters to Experiment With: - Feedback decay (fade amount) - Transformation amount (scale, rotation) - Blend mode (add, multiply, screen) - Blur intensity

Result: Trails, echoes, and organic growth patterns

4.10.2 Op Art and Moiré Patterns

Create optical illusions with overlapping patterns:

```
IteratorLoop (creates grid)
    |
Time -> Sin -> Rotation angle
    |
IteratorLoop (nested for lines)
```

```
    |
Rectangle (thin line)
```

Vary parameters like: - Line spacing - Rotation speed - Line thickness - Pattern density

4.10.3 Procedural Pattern Generation

Use noise and math to create endless variations:

Perlin Noise-Based Patterns:

```
IteratorLoop
    |
Position -> NoiseTexture sample
    |
Noise value -> Circle size
    |
Noise value -> Color
```

Grid Distortion:

```
IteratorLoop (grid)
    |
Position + (Noise * distortion amount)
    |
Shape
```

4.11 Post-Processing Effects

4.11.1 Image Composition

Layer multiple render passes for rich effects:

```
RenderToTexture (Pass 1: Shapes)
RenderToTexture (Pass 2: Glow)
RenderToTexture (Pass 3: Noise)
    |
ImageCompose (blend all layers)
```

```
    |
Final Output
```

4.11.2 TextureEffects for 2D

Apply effects to your rendered 2D scene:

Blur:

```
RenderToTexture -> TextureEffects (Blur) -> Output
```

Color Grading:

```
RenderToTexture -> ColorCorrection
    | (adjust hue, saturation, brightness, contrast)
Output
```

Glow Effect:

```
Original scene
    |
RenderToTexture (bright pass)
    |
Blur (large radius)
    |
ImageCompose (add to original)
```

4.11.3 Displacement Mapping

Distort shapes using textures:

```
NoiseTexture -> DisplacementMap -> Shape rendering
```

Creates wavy, distorted effects on 2D graphics.

4.12 Advanced Pattern Techniques

4.12.1 Recursive Subdivision

Create fractal-like patterns:

```
// Custom op: Recursive shape division
for (depth = 0; depth < maxDepth; depth++) {
    // Draw shape
    // Divide into smaller shapes
    // Recursively apply
}
```

4.12.2 Particle Systems in 2D

Simple particle engine structure:

```
ArrayLoop (particle count)
    |
Particle data (position, velocity, life)
    |
Physics update (gravity, friction)
    |
Transform -> Circle (particle visual)
```

4.12.3 Grid-Based Automata

Cellular automata and Game of Life patterns:

```
ArrayIterator (grid cells)
    |
Cell state + neighbor count
    |
Update rules (Conway's rules, etc.)
    |
Visual representation
```

4.13 Data Visualization

4.13.1 Chart Generation

Create custom charts and graphs:

Bar Chart:

```
ArrayIterator (data values)
    |
Index -> X position
Value -> Rectangle height
    |
Rectangle (bar)
```

Line Chart:

```
ArrayIterator (data points)
    |
Connect points with Lines op
    |
Add circles for data points
```

4.13.2 Integration with ECharts

Combine cables.gl with Apache ECharts for advanced charts:

1. Load ECharts library
2. Generate chart in HTML/Canvas
3. Capture as texture in cables.gl
4. Apply 3D transforms or effects

4.13.3 Real-Time Data

Visualize live data streams:

```
WebSocket/API -> Parse data
    |
ArrayIterator -> Visualize each value
    |
Smooth/Interpolate for fluid animation
```

4.14 Complex Example Projects

4.14.1 Example 4: Kaleidoscope Effect

```
MainLoop
    |
BasicMaterial
    |
IteratorLoop (6 segments)
    |
Transform (rotate by index * 60°)
    |
Transform (mirror flip alternating)
    |
Your content (shapes, webcam, etc.)
```

4.14.2 Example 5: Audio-Reactive Loading Animation

```
AudioAnalyzer (beat detection)
    |
IteratorLoop (circle of dots)
    |
Index + Time -> Rotation
Beat amplitude -> Scale pulse
    |
SetColor (beat changes color)
    |
Circle (dot)
```

4.14.3 Example 6: Data-Driven Weather Visualization

```
API -> Fetch weather data
    |
Parse JSON -> Extract values
    |
Temperature -> Background color
Humidity -> Particle density
Wind -> Animation speed
    |
Animated scene reflecting weather
```

4.14.4 Example 7: Feedback Tunnel Effect

```
RenderToTexture (previous frame)
    |
Transform (scale 1.05, center pivot)
    |
SetAlpha (0.98 for fade)
    |
Draw to screen
    |
Add new circles at edges
    |
Feed back into texture
```

Creates an infinite tunnel effect.

4.14.5 Example 8: Mouse Trail with Fade

```
MousePosition
    |
RenderToTexture (with feedback)
    |
ColorCorrection (reduce brightness)
    |
Draw circle at mouse position
    |
Blend with previous frame
```

Creates smooth, fading trails following the cursor.

4.15 Performance Optimization

4.15.1 Culling and Clipping

Only draw what's visible:

```
If (shape position in viewport bounds)
    -> Draw shape
Else
    -> Skip
```

4.15.2 Object Pooling

Reuse shape instances instead of creating new ones:

```
// Maintain pool of inactive shapes
// Activate/deactivate as needed
// Prevents GC thrashing
```

4.15.3 Level of Detail (LOD)

Simplify distant or small shapes:

```
If (shape size < threshold)
    -> Use simple circle
Else
    -> Use detailed polygon
```

4.15.4 Batching Draw Calls

Group similar operations:

```
SetColor once
    |
Draw all shapes of same color
    |
SetColor again
    |
Draw next batch
```

Reduces state changes and improves performance.

4.16 Masking and Clipping

4.16.1 Stencil Buffer Masking

Use shapes as masks for other shapes:

```
EnableStencil
    |
Draw mask shape (Circle)
    |
SetStencilMode (draw only inside)
    |
Draw content (Rectangle)
    |
DisableStencil
```

4.16.2 Alpha Mask Technique

Use texture alpha for complex masks:

```
MaskTexture -> AlphaMask
    |
Your content (masked by texture)
```

4.17 Color Theory in Practice

4.17.1 Color Harmonies

Generate pleasing color palettes:

Complementary:

```
BaseHue -> SetColor (shape 1)
BaseHue + 180° -> SetColor (shape 2)
```

Triadic:

```
BaseHue -> Color 1
BaseHue + 120° -> Color 2
BaseHue + 240° -> Color 3
```

Analogous:

```
BaseHue -> Color 1
BaseHue + 30° -> Color 2
BaseHue - 30° -> Color 3
```

4.17.2 Gradient Creation

Smooth color transitions:

Linear Gradient:

```
IteratorLoop (steps)
    |
Index / TotalSteps -> Mix (Color1, Color2, t)
    |
SetColor -> Rectangle strip
```

Radial Gradient:

```
Distance from center -> Mix (Inner, Outer, t)
```

4.18 Typography and Text Effects

4.18.1 Dynamic Text

Animate text properties:

```
Time -> Character spacing
MouseX -> Font size
AudioLevel -> Text opacity
```

4.18.2 Text as Texture

Use text rendering for effects:

```
TextTexture (render text to texture)
    |
Apply shader effects
    |
Use as sprite or background
```

4.18.3 Kinetic Typography

Animate individual letters:

```
TextArray (split into chars)
    |
ArrayIterator
    |
Transform (unique per character)
    |
DrawText (single char)
```

4.19 Practical Examples

4.19.1 Example 1: Pulsing Circle

```
MainLoop
    |
BasicMaterial (set your color)
    |
Time -> Sin -> Scale input
```

```
    |
Circle
```

4.19.2 Example 2: Rotating Grid

```
MainLoop
    |
BasicMaterial
    |
IteratorLoop (10x10)
    |
Transform (position from iterator)
    |
Transform (rotation from Time)
    |
Rectangle
```

4.19.3 Example 3: Color Gradient Circle

```
MainLoop
    |
IteratorLoop (for each ring)
    |
IteratorIndex -> Map to Hue -> HSBtoRGB -> BasicMaterial (color input)
    |
BasicMaterial
    |
Circle (radius from iterator index)
```

4.20 Debugging and Workflow Tips

4.20.1 Visualizing Values

See what your ops are outputting:

```
Value -> NumberDisplay
Value -> DrawNumber (on screen)
```

4.20.2 Color Coding

Use consistent colors to identify different element types: - Structural elements: Blue - Interactive elements: Green
- Data elements: Yellow - Background: Dark grey

4.20.3 Naming Convention

Name ops descriptively: - `TransformRotation_MainShape` - `Color_Background` - `Iterator_ParticleGrid`

4.20.4 Comment Ops

Document complex sections:

```
Comment ("This section creates the feedback loop")
    |
Your complex patch area
```

4.21 Common Patterns and Recipes

4.21.1 Pattern: Circular Array

Arrange shapes in a circle:

```
IteratorLoop (count)
    |
Index * (360 / count) -> Angle
Angle -> Cos -> X position
Angle -> Sin -> Y position
    |
Transform -> Shape
```

4.21.2 Pattern: Wave Grid

Create wave motion across a grid:

```
IteratorLoop (rows)
    |
IteratorLoop (columns)
    |
(X + Time) -> Sin -> Y offset
    |
Transform -> Shape
```

4.21.3 Pattern: Spiral

Generate spiral patterns:

```
IteratorLoop
    |
Index -> Angle (index * goldenAngle)
Index -> Radius (sqrt(index) * spacing)
    |
Polar to Cartesian
    |
Transform -> Shape
```

4.21.4 Pattern: Responsive Grid

Grid that adapts to screen size:

```
ViewportWidth / CellSize -> Columns
ViewportHeight / CellSize -> Rows
    |
IteratorLoop (columns * rows)
    |
Grid positioning logic
```

4.22 Troubleshooting Common Issues

4.22.1 “Shapes not appearing”

- Check trigger connections (grey ports)
- Verify MainLoop is connected to BasicMaterial
- Check BasicMaterial alpha isn’t 0
- Verify camera/viewport settings

4.22.2 “Performance is slow”

- Reduce segment count on circles
- Lower particle/iterator counts
- Disable antialiasing if not needed
- Use simpler blend modes
- Check for unnecessary texture reads

4.22.3 “Colors look wrong”

- Verify color space (RGB vs HSB)
- Check SetColor is before shapes
- Verify alpha values
- Check blend modes

4.22.4 “Animation is jerky”

- Use Smooth op for value transitions
- Check frame rate in performance monitor
- Reduce complexity during motion
- Pre-calculate expensive operations

4.23 Performance Tips

1. **Reduce Segments** - Circles don’t need 100 segments if they’re small
2. **Batch Similar Shapes** - Group similar operations together
3. **Use Instancing** - For many identical shapes, use instanced drawing
4. **Limit Transparency** - Overlapping transparent shapes are expensive

5. **Cache Calculations** - Don’t recalculate same values each frame
6. **Cull Off-Screen** - Don’t draw what’s not visible
7. **Simplify Blending** - Complex blend modes are expensive
8. **Optimize Textures** - Use appropriate texture sizes
9. **Limit Feedback Depth** - Don’t keep too many feedback history frames
10. **Profile Regularly** - Use performance monitor to identify bottlenecks

4.24 Featured Videos

4.24.1 Official Tutorials

```
https://youtu.be/goO3PhuenBI
Title: First Steps in Cables.gl - Kaleidoscope Webcam Effect
Author: The Interactive & Immersive HQ
Thumbnail: https://i.ytimg.com/vi/goO3PhuenBI/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@TheInteractiveImmersiveHQ
```

```
https://youtu.be/xnObNRv8n9I
Title: Introduction to cables.gl - Data-Driven Gradient from Geo-Located Weather
Author: Kirell Benzi
Thumbnail: https://i.ytimg.com/vi/xnObNRv8n9I/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@kirellbenzi
```

4.24.2 Additional Resources

- **Generative Op Art Tutorial**: Class Central Course - Learn feedback loops and Op Art
- **Interactive Rectangle Tutorial**: Blog Post - Create on-screen sliders
- **Post-Processing Guide**: Official Docs - Apply effects to scenes
- **Data Visualization**: Apache ECharts Integration - Combine with charting libraries
- **Cables.gl Examples**: Official Examples - Browse community creations
- **Coding with Cables**: GitHub Repo - Code examples and custom ops

4.25 Exercises

4.25.1 Beginner

1. Create a colorful loading spinner using rotating circles
2. Build a grid of squares that change color based on mouse position
3. Make a simple particle system with random positions and sizes

4.25.2 Intermediate

4. Create a kaleidoscope effect with 8 mirrored segments
5. Build an interactive color picker using mouse position
6. Implement a feedback tunnel with infinite zoom effect
7. Create a data visualization showing time-series data as animated bars

4.25.3 Advanced

8. Build a generative Op Art piece using feedback loops
9. Create a particle system with physics (gravity, collision)
10. Implement a cellular automaton (Game of Life or similar)
11. Create an audio-reactive geometric pattern generator
12. Build a real-time weather visualization using API data

4.26 Project Ideas

1. **Abstract Clock** - Time visualization with geometric shapes
2. **Music Visualizer** - Frequency bands displayed as 2D patterns
3. **Generative Logo** - Company logo with parametric variations
4. **Loading Animations** - Collection of animated loaders
5. **Data Dashboard** - Real-time data display with charts
6. **Interactive Art Installation** - Touch/camera-driven visuals
7. **Typography Animation** - Kinetic text effects
8. **Pattern Generator** - Infinite procedural pattern variations
9. **Mouse-Driven Drawing Tool** - Paint with code
10. **Meditation Visual** - Calming, slowly evolving patterns

---

5 3D Graphics in Cables.gl

5.1 Introduction to 3D

Cables.gl provides powerful tools for creating real-time 3D graphics using WebGL. This chapter covers everything from basic 3D concepts to advanced rendering techniques, scene management, and performance optimization. Whether you're creating simple 3D visualizations or complex interactive experiences, this guide will give you the knowledge to master 3D graphics in cables.gl.

5.2 The 3D Pipeline

A basic 3D setup requires:

```
+-------------------------------------------------------------+
|                    3D RENDERING PIPELINE                     |
+-------------------------------------------------------------+
|                                                               |
|  MainLoop (Frame Start)                                      |
|    |                                                          |
|  Camera (View Setup)                                          |
|    +-> Position (X, Y, Z)                                     |
|    +-> Rotation / LookAt                                      |
|    +-> Projection (Perspective/Orthographic)                  |
|    |                                                          |
|  Lighting (Optional)                                          |
|    +-> Directional Light                                      |
|    +-> Point Light                                             |
|    +-> Ambient Light                                           |
|    |                                                          |
|  Materials (Surface Properties)                               |
|    +-> BasicMaterial / StandardMaterial                        |
|    +-> Color / Texture                                         |
|    +-> Shading Properties                                      |
|    |                                                          |
|  Mesh/Geometry (3D Shapes)                              |
|    +-> Box, Sphere, Plane, etc.                           |
|    +-> Custom Models                                      |
|    |                                                    |
|  Rendered Output (Canvas)                               |
|                                                         |
+--------------------------------------------------------+
```

5.3 Cameras

Cameras define how we view the 3D scene.

5.3.1 PerspectiveCamera

The most common camera type - mimics human vision with perspective distortion.

Key Parameters: - `FOV` (Field of View) - How wide the view is (typically 45-90 degrees) - `Near` / `Far` - Clipping planes (objects outside this range aren't rendered) - `Position X/Y/Z` - Camera location

5.3.2 OrthographicCamera

No perspective distortion - useful for UI, 2D-style 3D, or technical views.

Key Parameters: - `Zoom` - Scale of the view - `Near` / `Far` - Clipping planes

5.3.3 Orbit Controls

Add interactive camera controls:

```
Camera -> OrbitControls
```

Allows users to rotate, zoom, and pan the view.

5.3.4 LookAt

Point the camera at a specific location or object.

```
Camera -> LookAt (target position)
```

Use Cases: - Follow a moving object - Create cinematic camera movements - Focus on specific scene elements

5.3.5 Camera Animation

Animate camera movement for cinematic effects:

```
Time -> Sin -> Camera Position X
Time -> Cos -> Camera Position Z
Time -> Camera Rotation Y (orbit)
```

5.3.6 Camera Shake Effect

Add dynamic camera shake:

```
Random -> Multiply (shake intensity) -> Add to Camera Position
```

5.3.7 First-Person Camera

Create FPS-style camera controls:

```
MouseX -> Camera Rotation Y
MouseY -> Camera Rotation X
WASD Keys -> Camera Position
```

5.3.8 Camera Path Following

Follow a predefined path:

```
ArrayIterator (path points)
    |
Smooth interpolation between points
    |
Camera Position
```

5.3.9 Camera Constraints

Limit camera movement:

```
Camera Position -> Clamp (min, max) -> Constrained Position
```

5.4 Lighting

Lighting brings depth and realism to 3D scenes.

5.4.1 AmbientLight

Uniform light that illuminates everything equally.

```
MainLoop -> Camera -> AmbientLight -> [Rest of scene]
```

Tip: Use subtle ambient light to prevent completely black shadows.

5.4.2 DirectionalLight

Light from a specific direction (like the sun).

Key Parameters: - Direction (X, Y, Z) - Color - Intensity

5.4.3 PointLight

Light emanating from a point in space (like a light bulb).

Key Parameters: - Position (X, Y, Z) - Color - Intensity - Falloff radius

5.4.4 SpotLight

Focused beam of light (like a flashlight or stage light).

Key Parameters: - Position and direction - Cone angle - Falloff

5.4.5 Shadow Mapping

Enable shadows for more realism:

```
DirectionalLight (shadows enabled) -> ShadowMap -> Scene
```

Shadow Parameters: - `Shadow Map Size` - Resolution (higher = sharper, slower) - `Shadow Bias` - Prevents shadow acne - `Shadow Radius` - Softness of shadow edges

Tip: Use lower shadow map sizes for better performance. 1024x1024 is usually sufficient.

5.4.6 Three-Point Lighting Setup

Professional lighting arrangement:

```
MainLoop -> Camera
    |
AmbientLight (subtle, 0.2 intensity) - Fill light
    |
DirectionalLight (main, from top-left) - Key light
    |
PointLight (weaker, opposite side) - Rim light
    |
[Your scene]
```

Key Light: Main illumination (brightest) **Fill Light:** Reduces harsh shadows (ambient or weak directional) **Rim Light:** Creates edge highlights (back/side lighting)

5.4.7 Image-Based Lighting (IBL)

Use environment maps for realistic lighting:

```
HDRITexture -> Environment Map -> PBRMaterial
```

Creates reflections and lighting based on real-world environments.

5.4.8 Light Probes

Place light probes in your scene for accurate local lighting:

```
LightProbe -> Sample nearby lights -> Apply to objects
```

5.4.9 Volumetric Lighting

Create god rays and atmospheric lighting:

```
DirectionalLight -> VolumetricScattering -> [Scene]
```

5.4.10 Light Animation

Animate lights for dynamic scenes:

```
Time -> Sin -> Light Intensity (pulsing)
Time -> Rotate -> Light Direction (rotating sun)
AudioAnalyzer -> Light Color (audio-reactive)
```

5.5 Geometry and Meshes

5.5.1 Primitive Shapes

Cube - Basic box shape

```
Parameters: Width, Height, Depth
```

Sphere - Perfect sphere

```
Parameters: Radius, Segments (horizontal/vertical)
```

Cylinder - Tube shape

```
Parameters: Radius Top/Bottom, Height, Segments
```

Plane - Flat surface

```
Parameters: Width, Height
```

Torus - Donut shape

```
Parameters: Radius, Tube Radius, Segments
```

5.5.2 Loading 3D Models

OBJLoader - Load .obj format models

```
OBJLoader -> Mesh
```

GLTFLoader - Load .gltf/.glb models (recommended)

```
GLTFLoader -> Scene/Mesh
```

FBXLoader - Load .fbx models

5.5.3 Creating Custom Geometry

Use **PointCloud** or **CustomGeometry** ops to build meshes from data.

PointCloud:

```
ArrayIterator (positions) -> PointCloud
```

CustomGeometry:

```
Vertices Array -> Normals Array -> UVs Array -> CustomGeometry
```

5.5.4 Procedural Geometry Generation

Create geometry programmatically:

Example: Procedural Terrain

```
IteratorLoop (grid)
    |
NoiseTexture (sample at position) -> Height
    |
Calculate vertex positions
    |
Generate normals
    |
CustomGeometry
```

Example: Parametric Surfaces

```
U/V parameters -> Math functions -> Vertex positions
    |
CustomGeometry
```

5.5.5 Geometry Instancing

Render many copies efficiently:

```
Mesh -> InstanceTransform (array of transforms) -> InstancedMesh
```

Use Cases: - Forests of trees - Crowds of characters - Particle systems - Repeating architectural elements

5.5.6 Geometry Modifiers

Modify existing geometry:

Subdivision:

```
Mesh -> Subdivide -> Smoother surface
```

Displacement:

```
Mesh -> DisplacementMap -> Deformed geometry
```

Morphing:

```
Mesh1 -> Morph -> Mesh2 (blend between shapes)
```

5.5.7 Boolean Operations

Combine geometries:

```
Mesh1 -> BooleanUnion -> Mesh2
Mesh1 -> BooleanSubtract -> Mesh2
Mesh1 -> BooleanIntersect -> Mesh2
```

5.6 Real-Time Mesh Distortion

Real-time mesh distortion allows you to dynamically modify geometry vertices during rendering, creating effects like bending walls, scaling surfaces, and warping shapes. This is essential for architectural visualization, interactive installations, and dynamic 3D effects.

5.6.1 Understanding Vertex Manipulation

Mesh distortion works by modifying vertex positions in real-time. Each vertex has: - **Position** (X, Y, Z) - Where the vertex is located - **Normal** (NX, NY, NZ) - Which direction the surface faces - **UV Coordinates** (U, V) - Texture mapping coordinates

```
+-------------------------------------------------------------+
|              MESH DISTORTION PIPELINE                        |
+-------------------------------------------------------------+
|                                                              |
|  Base Mesh (Plane, Box, etc.)                                |
|    |                                                         |
|  Vertex Data Extraction                                     |
|    +-> Positions Array                                      |
|    +-> Normals Array                                        |
|    +-> UVs Array                                            |
|    |                                                         |
|  Distortion Function                                         |
|    +-> Calculate new positions                              |
|    +-> Update normals (if needed)                           |
|    +-> Preserve UVs                                         |
|    |                                                         |
|  CustomGeometry (with distorted vertices)                    |
|    |                                                         |
|  Material -> Render                                          |
|                                                              |
+-------------------------------------------------------------+
```

5.6.2 Method 1: Node-Based Distortion

Using built-in cables.gl ops to distort meshes.

Example 1: Scaling a Wall (Size Transformation)

Transform a plain wall into different sizes using procedural scaling:

```
+-------------------------------------------------------------+
|              WALL SCALING SETUP                             |
+-------------------------------------------------------------+
|                                                             |
|  Plane (Base Wall)                                          |
|    |                                                        |
|  GetVertices -> Positions Array                            |
|    |                                                        |
|  Scale Factor (X, Y, Z)                                     |
|    |                                                        |
|  ArrayMap (multiply each vertex by scale)                  |
|    |                                                        |
|  GetNormals -> Normals Array                                |
|  GetUVs -> UVs Array                                        |
|    |                                                        |
|  CustomGeometry (new positions, normals, UVs)              |
|    |                                                        |
|  Material -> Render                                         |
|                                                             |
+-------------------------------------------------------------+
```

Step-by-Step Node Setup:

1. **Create Base Plane:**
   - Add `Plane` op
   - Set Width: 10, Height: 5
   - Set Segments Width: 20, Segments Height: 10 (for smooth distortion)
2. **Extract Vertex Data:**

    - Add `GetVertices` op
    - Connect Plane -> GetVertices
    - Output: Array of vertex positions
3. **Create Scale Controls:**
    - Add `Slider` ops for X, Y, Z scale
    - Or use `Number` ops with values
4. **Apply Scaling:**
    - Use `ArrayMap` or `ArrayIterator` to multiply each vertex
    - For each vertex: `[x, y, z] * [scaleX, scaleY, scaleZ]`
5. **Rebuild Geometry:**
    - Get original normals and UVs from Plane
    - Add `CustomGeometry` op
    - Connect: Scaled Positions -> CustomGeometry
    - Connect: Original Normals -> CustomGeometry
    - Connect: Original UVs -> CustomGeometry

Example 2: Bending a Wall (Curved Distortion)

Bend a plain wall into a curved wall with controllable angle:

```
+-------------------------------------------------------------+
|              WALL BENDING SETUP                             |
+-------------------------------------------------------------+
|                                                             |
|  Plane (Base Wall)                                          |
|    |                                                        |
|  GetVertices -> Positions Array                            |
|    |                                                        |
|  Bend Angle (Slider/Number)                                 |
|  Bend Center (X position where bend occurs)                 |
|    |                                                        |
|  ArrayMap (apply bend transformation)                      |
|    |                                                        |
|  Calculate New Normals (for proper lighting)               |
|    |                                                        |
|  CustomGeometry -> Material -> Render                      |
|                                                               |
+-------------------------------------------------------------+
```

Bending Algorithm (Node-Based):

For each vertex: 1. Calculate distance from bend center 2. Calculate angle based on distance and bend amount 3. Rotate vertex around bend axis 4. Update position

Node Setup for Bending:

```
Plane
  |
GetVertices -> ArrayIterator
  |
For each vertex:
  |
  Vertex X -> Subtract (Bend Center) -> Distance from center
  |
  Distance -> Multiply (Bend Angle) -> Rotation angle
  |
  Vertex Y -> Sin(Rotation) -> New Y position
  Vertex Z -> Cos(Rotation) -> New Z position
  Vertex X -> Keep original
  |
  Combine -> New Vertex Position
  |
ArrayCollect -> All Distorted Vertices
  |
CustomGeometry
```

5.6.3 Method 2: JavaScript Custom Op for Mesh Distortion

For more control and performance, use a JavaScript custom op to handle distortion.

Custom Op: Wall Distorter

Create a custom op that handles both scaling and bending:

```
// Custom Op: WallDistorter
// Distorts a plane mesh with scaling and bending

const inVertices = op.inArray("Input Vertices");
const inNormals = op.inArray("Input Normals");
const inUVs = op.inArray("Input UVs");

// Scale parameters
const inScaleX = op.inFloat("Scale X", 1.0);
const inScaleY = op.inFloat("Scale Y", 1.0);
const inScaleZ = op.inFloat("Scale Z", 1.0);

// Bend parameters
const inBendAngle = op.inFloat("Bend Angle", 0.0); // in radians
const inBendCenter = op.inFloat("Bend Center X", 0.0); // X position of bend
const inBendAxis = op.inSwitch("Bend Axis", ["X", "Y", "Z"], "X");

// Outputs
const outVertices = op.outArray("Distorted Vertices");
const outNormals = op.outArray("Distorted Normals");
const outUVs = op.outArray("Output UVs");

function distortVertices() {
    const vertices = inVertices.get();
    const normals = inNormals.get();
    const uvs = inUVs.get();

    if (!vertices || vertices.length === 0) {
        outVertices.set([]);
        outNormals.set([]);
        outUVs.set([]);
        return;
    }

    const scaleX = inScaleX.get();
    const scaleY = inScaleY.get();
    const scaleZ = inScaleZ.get();
    const bendAngle = inBendAngle.get();
    const bendCenter = inBendCenter.get();
    const bendAxis = inBendAxis.get();

    const distortedVertices = [];
    const distortedNormals = [];

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

        // Apply scaling first
        x *= scaleX;
        y *= scaleY;
        z *= scaleZ;

        // Apply bending
        if (Math.abs(bendAngle) > 0.001) {
            if (bendAxis === "X") {
                // Bend along X axis (curves in Y-Z plane)
                const distanceFromCenter = x - bendCenter;
                const angle = distanceFromCenter * bendAngle;

                // Rotate around X axis
                const cosA = Math.cos(angle);
                const sinA = Math.sin(angle);
                const newY = y * cosA - z * sinA;
                const newZ = y * sinA + z * cosA;
                y = newY;
                z = newZ;

                // Update normals
                if (normals && normals.length > i + 2) {
                    const nx = normals[i];
                    const ny = normals[i + 1];
                    const nz = normals[i + 2];
                    distortedNormals.push(
                        nx,
                        ny * cosA - nz * sinA,
                        ny * sinA + nz * cosA
                    );
                }
            } else if (bendAxis === "Y") {
                // Bend along Y axis (curves in X-Z plane)
                const distanceFromCenter = y - bendCenter;
                const angle = distanceFromCenter * bendAngle;

                const cosA = Math.cos(angle);
                const sinA = Math.sin(angle);
                const newX = x * cosA - z * sinA;
                const newZ = x * sinA + z * cosA;
                x = newX;
                z = newZ;

                if (normals && normals.length > i + 2) {
                    const nx = normals[i];
                    const ny = normals[i + 1];
                    const nz = normals[i + 2];
                    distortedNormals.push(
                        nx * cosA - nz * sinA,
                        ny,
                        nx * sinA + nz * cosA
                    );
                }
            } else if (bendAxis === "Z") {
                // Bend along Z axis (curves in X-Y plane)
                const distanceFromCenter = z - bendCenter;
                const angle = distanceFromCenter * bendAngle;

                const cosA = Math.cos(angle);
```

```
            const sinA = Math.sin(angle);
            const newX = x * cosA - y * sinA;
            const newY = x * sinA + y * cosA;
            x = newX;
            y = newY;

            if (normals && normals.length > i + 2) {
                const nx = normals[i];
                const ny = normals[i + 1];
                const nz = normals[i + 2];
                distortedNormals.push(
                    nx * cosA - ny * sinA,
                    nx * sinA + ny * cosA,
                    nz
                );
            }
        }
    } else {
        // No bending, just copy normals
        if (normals && normals.length > i + 2) {
            distortedNormals.push(
                normals[i],
                normals[i + 1],
                normals[i + 2]
            );
        }
    }

    distortedVertices.push(x, y, z);
}

outVertices.set(distortedVertices);
if (distortedNormals.length > 0) {
    outNormals.set(distortedNormals);
} else if (normals) {
    outNormals.set(normals);
    }
    if (uvs) {
        outUVs.set(uvs);
    }
}

// Update when inputs change
inVertices.onChange = distortVertices;
inNormals.onChange = distortVertices;
inUVs.onChange = distortVertices;
inScaleX.onChange = distortVertices;
inScaleY.onChange = distortVertices;
inScaleZ.onChange = distortVertices;
inBendAngle.onChange = distortVertices;
inBendCenter.onChange = distortVertices;
inBendAxis.onChange = distortVertices;
```

Using the Wall Distorter Op

Setup:

```
Plane (Base Wall)
  |
GetVertices -> WallDistorter (Input Vertices)
GetNormals -> WallDistorter (Input Normals)
GetUVs -> WallDistorter (Input UVs)
  |
WallDistorter (Distorted Vertices) -> CustomGeometry
WallDistorter (Distorted Normals) -> CustomGeometry
WallDistorter (Output UVs) -> CustomGeometry
  |
Material -> Render
```

Controls: - **Scale X/Y/Z**: Resize the wall - **Bend Angle**: Curvature amount (in radians, use Math.PI/4 for 45°) - **Bend Center X**: Where the bend occurs along the wall - **Bend Axis**: Which axis to bend around

5.6.4 Advanced: Animated Wall Distortion

Combine distortion with animation for dynamic effects:

```
// Custom Op: AnimatedWallDistorter
// Adds time-based animation to distortion

const inVertices = op.inArray("Input Vertices");
const inNormals = op.inArray("Input Normals");
const inUVs = op.inArray("Input UVs");

// Animation parameters
const inTime = op.inFloat("Time", 0.0);
const inAnimationSpeed = op.inFloat("Animation Speed", 1.0);
const inAnimationType = op.inSwitch("Animation Type",
    ["None", "Pulse", "Wave", "Oscillate"], "None");

// Distortion parameters (same as before)
const inScaleX = op.inFloat("Scale X", 1.0);
const inScaleY = op.inFloat("Scale Y", 1.0);
const inScaleZ = op.inFloat("Scale Z", 1.0);
const inBendAngle = op.inFloat("Bend Angle", 0.0);
const inBendCenter = op.inFloat("Bend Center X", 0.0);

// Outputs
const outVertices = op.outArray("Distorted Vertices");
const outNormals = op.outArray("Distorted Normals");
const outUVs = op.outArray("Output UVs");

function getAnimatedBendAngle() {
    const baseAngle = inBendAngle.get();
    const time = inTime.get();
    const speed = inAnimationSpeed.get();
    const type = inAnimationType.get();

    if (type === "None") {
        return baseAngle;
    } else if (type === "Pulse") {
        // Pulse between 0 and baseAngle
        const pulse = (Math.sin(time * speed) + 1) / 2; // 0 to 1
        return baseAngle * pulse;
    } else if (type === "Wave") {
        // Wave effect
        return baseAngle * Math.sin(time * speed);
    } else if (type === "Oscillate") {
        // Oscillate around baseAngle
        return baseAngle + Math.sin(time * speed) * (baseAngle * 0.5);
    }

    return baseAngle;
}

function distortVertices() {
    const vertices = inVertices.get();
    const normals = inNormals.get();
    const uvs = inUVs.get();

    if (!vertices || vertices.length === 0) {
        outVertices.set([]);
        outNormals.set([]);
        outUVs.set([]);
        return;
    }

    const scaleX = inScaleX.get();
    const scaleY = inScaleY.get();
    const scaleZ = inScaleZ.get();
    const bendAngle = getAnimatedBendAngle();
    const bendCenter = inBendCenter.get();

    const distortedVertices = [];
    const distortedNormals = [];
```

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

    // Apply scaling
    x *= scaleX;
    y *= scaleY;
    z *= scaleZ;

    // Apply animated bending
    if (Math.abs(bendAngle) > 0.001) {
        const distanceFromCenter = x - bendCenter;
        const angle = distanceFromCenter * bendAngle;

        const cosA = Math.cos(angle);
        const sinA = Math.sin(angle);
        const newY = y * cosA - z * sinA;
        const newZ = y * sinA + z * cosA;
        y = newY;
        z = newZ;

        // Update normals
        if (normals && normals.length > i + 2) {
            const nx = normals[i];
            const ny = normals[i + 1];
            const nz = normals[i + 2];
            distortedNormals.push(
                nx,
                ny * cosA - nz * sinA,
                ny * sinA + z * cosA
            );
        }
    } else {
        if (normals && normals.length > i + 2) {
                distortedNormals.push(
                    normals[i],
                    normals[i + 1],
                    normals[i + 2]
                );
            }
        }

        distortedVertices.push(x, y, z);
    }

    outVertices.set(distortedVertices);
    if (distortedNormals.length > 0) {
        outNormals.set(distortedNormals);
    } else if (normals) {
        outNormals.set(normals);
    }
    if (uvs) {
        outUVs.set(uvs);
    }
}

// Update on input changes
inVertices.onChange = distortVertices;
inNormals.onChange = distortVertices;
inUVs.onChange = distortVertices;
inTime.onChange = distortVertices;
inAnimationSpeed.onChange = distortVertices;
inAnimationType.onChange = distortVertices;
inScaleX.onChange = distortVertices;
inScaleY.onChange = distortVertices;
inScaleZ.onChange = distortVertices;
inBendAngle.onChange = distortVertices;
inBendCenter.onChange = distortVertices;
```

5.6.5 Practical Example: Interactive Curved Wall

Complete setup for an interactive curved wall with real-time controls:

```
+-------------------------------------------------------------+
|              INTERACTIVE CURVED WALL SETUP                  |
+-------------------------------------------------------------+
|                                                             |
|  MainLoop                                                   |
|    |                                                        |
|  Plane (Base Wall)                                          |
|    Width: 10, Height: 5                                     |
|    Segments: 30x15 (for smooth curves)                     |
|    |                                                        |
|  GetVertices -> WallDistorter                               |
|  GetNormals -> WallDistorter                                |
|  GetUVs -> WallDistorter                                    |
|    |                                                        |
|  Slider (Bend Angle: 0 to PI/2) -> WallDistorter          |
|  Slider (Bend Center: -5 to 5) -> WallDistorter           |
|  Slider (Scale X: 0.5 to 2.0) -> WallDistorter            |
|  Slider (Scale Y: 0.5 to 2.0) -> WallDistorter            |
|    |                                                        |
|  WallDistorter -> CustomGeometry                            |
|    |                                                        |
|  StandardMaterial -> Render                                 |
|    |                                                        |
|  Camera -> OrbitControls                                    |
|                                                             |
+-------------------------------------------------------------+
```

5.6.6 Performance Optimization

For real-time distortion, optimize your setup:

1. **Reduce Vertex Count When Possible:**
    - Use fewer segments for static walls

- Increase segments only where distortion is visible

2. **Cache Calculations:**

```
let cachedVertices = null;
let cachedBendAngle = null;
let cachedScale = null;

function distortVertices() {
    const bendAngle = inBendAngle.get();
    const scale = inScaleX.get();

    // Only recalculate if inputs changed
    if (cachedVertices &&
        cachedBendAngle === bendAngle &&
        cachedScale === scale) {
        return; // Use cached result
    }

    // Recalculate...
    cachedVertices = distortedVertices;
    cachedBendAngle = bendAngle;
    cachedScale = scale;
}
```

3. **Use Instancing for Multiple Walls:**
   - Create one distorted wall
   - Use `InstancedMesh` to duplicate it
   - Much faster than distorting each wall separately

4. **Update Only When Needed:**

```
// Only update on frame if animation is active
const inRender = op.inTrigger("Render");
inRender.onTriggered = function() {
    if (inAnimationType.get() !== "None") {
        distortVertices();
    }
};
```

5.6.7 Advanced Techniques

Multi-Axis Bending

Bend along multiple axes simultaneously:

```
// Bend along both X and Y axes
const bendX = distanceFromCenterX * bendAngleX;
const bendY = distanceFromCenterY * bendAngleY;

// Apply rotations in sequence
// First rotate around X, then around Y
```

Non-Linear Distortion

Use easing functions for smooth transitions:

```
function easeInOutCubic(t) {
    return t < 0.5
        ? 4 * t * t * t
        : 1 - Math.pow(-2 * t + 2, 3) / 2;
}

const easedAngle = baseAngle * easeInOutCubic(progress);
```

Texture Coordinate Preservation

When distorting, UVs should remain unchanged for proper texturing:

```
// Always preserve original UVs
outUVs.set(inUVs.get()); // Don't modify UVs during distortion
```

5.6.8 Common Use Cases

1. **Architectural Visualization:**
   - Bend walls to show different room layouts
   - Scale walls to demonstrate space variations
2. **Interactive Installations:**
   - User-controlled wall distortion
   - Audio-reactive bending
3. **Animation:**
   - Morphing between straight and curved walls
   - Dynamic space transformations
4. **Game Mechanics:**
   - Procedural level generation
   - Dynamic environment changes

5.6.9 Troubleshooting

Problem: Normals look wrong after distortion - Solution: Recalculate normals after distortion - Use `CalculateNormals` op or compute in JavaScript

Problem: Texture stretches or distorts - Solution: Don't modify UV coordinates - Keep original UVs from the base mesh

Problem: Performance is slow - Solution: Reduce vertex count - Cache calculations - Only update when parameters change

Problem: Bending looks jagged - Solution: Increase mesh segments - Use smoother interpolation

5.7 Materials

Materials define how surfaces appear when lit.

5.7.1 BasicMaterial

Simple colored material, not affected by lighting.

5.7.2 LambertMaterial

Matte material with diffuse lighting.

5.7.3 PhongMaterial

Shiny material with specular highlights.

Key Parameters: - `Diffuse Color` - Base color - `Specular Color` - Highlight color - `Shininess` - How sharp the highlights are

5.7.4 PBRMaterial (Physically Based Rendering)

Most realistic material option.

Key Parameters: - `Albedo` - Base color - `Metalness` - How metallic (0 = plastic, 1 = metal) - `Roughness` - Surface smoothness (0 = mirror, 1 = rough) - `Normal Map` - Surface detail - `Ambient Occlusion` - Crevice shadows - `Emissive` - Self-illumination - `Clearcoat` - Additional glossy layer (for car paint, etc.)

PBR Workflow Tips: - Use real-world material values for best results - Metalness and Roughness are inverse - metals are usually smooth (low roughness) - Combine texture maps for realistic surfaces - Use HDR environment maps for accurate reflections

5.7.5 Material Blending

Blend between materials:

```
Material1 -> Mix -> Material2 (blend factor) -> BlendedMaterial
```

5.7.6 Animated Materials

Animate material properties:

```
Time -> Sin -> Material Color (pulsing)
Time -> Material Roughness (shimmer effect)
MouseX -> Material Metalness (interactive)
```

5.7.7 Material Variants

Create material variations:

```
BaseMaterial -> Multiply Color -> Variant1
BaseMaterial -> Multiply Color -> Variant2
```

5.7.8 Custom Shader Materials

Use custom GLSL shaders (see Shaders chapter):

```
ShaderMaterial (custom GLSL) -> Mesh
```

5.7.9 Material Instancing

Apply same material to multiple objects efficiently:

```
Material -> Apply to multiple meshes
```

5.8 Transformations in 3D

5.8.1 Transform

Same as 2D but with full 3D control:

```
Transform
+-- TranslateX, TranslateY, TranslateZ
+-- RotateX, RotateY, RotateZ
+-- ScaleX, ScaleY, ScaleZ (or uniform Scale)
```

5.8.2 Matrix Operations

For advanced control, use matrix ops: - `MatrixMultiply` - Combine transformations - `LookAt` - Point object at target - `Billboard` - Always face camera - `MatrixInvert` - Reverse transformation - `MatrixDecompose` - Extract position/rotation/scale

5.8.3 Hierarchical Transforms

Create parent-child relationships:

```
Transform (parent)
    |
Transform (child) - inherits parent's transform
    |
Mesh
```

Use Cases: - Character rigging (body -> arm -> hand) - Vehicle systems (car -> wheel -> tire) - Solar systems (sun -> planet -> moon)

5.8.4 Constraint Systems

Constrain object movement:

Distance Constraint:

```
Object1 Position -> Distance -> Object2 Position (maintain distance)
```

Look-At Constraint:

```
Object -> LookAt -> Target (always face target)
```

Path Constraint:

```
Object -> Follow Path -> Constrained movement
```

5.8.5 IK (Inverse Kinematics)

Control chains of objects:

```
End Effector Position -> IK Solver -> Joint Angles
    |
Transform chain
```

5.8.6 Physics-Based Transforms

Use physics for natural movement:

```
PhysicsBody -> Transform (position/rotation from physics)
```

5.8.7 Transform Caching

Cache expensive transformations:

```
Transform -> Cache -> Reuse for multiple objects
```

5.9 Rendering Techniques

5.9.1 Rendering Order

Opaque objects should render before transparent ones:

```
MainLoop -> Camera
    |
[Opaque objects]
    |
EnableBlending
    |
[Transparent objects]
```

5.9.2 Multiple Render Passes

Create effects like glow, depth of field, or reflections:

```
MainLoop -> Camera -> RenderToTexture -> [Scene]
                |
            TextureEffect
                |
            RenderToScreen
```

5.9.3 Fog

Add atmospheric depth:

```
MainLoop -> Camera -> Fog -> [Scene]
```

Types: - Linear fog - Constant density - Exponential fog - Density increases with distance - Height fog - Fog based on Y position

5.9.4 Screen-Space Ambient Occlusion (SSAO)

Add depth and realism:

```
MainLoop -> Camera -> RenderToTexture (depth)
    |
SSAO Effect
    |
Apply to scene
```

5.9.5 Screen-Space Reflections (SSR)

Realistic reflections without reflection probes:

```
Scene -> RenderToTexture -> SSR Effect -> Reflections
```

5.9.6 Depth of Field

Focus blur effect:

```
Camera -> DepthOfField -> Focus distance -> Blur amount
```

5.9.7 Bloom

Glowing highlights:

```
Scene -> Brightness threshold -> Blur -> Add back -> Bloom
```

5.9.8 Motion Blur

Blur moving objects:

```
Previous frame -> Current frame -> Blend -> Motion blur
```

5.9.9 Color Grading

Post-process color adjustments:

```
Scene -> ColorCorrection
    +-- Exposure
    +-- Contrast
    +-- Saturation
    +-- Color temperature
    +-- Tint
```

5.9.10 Chromatic Aberration

Color separation effect:

```
Scene -> ChromaticAberration -> Distorted colors
```

5.9.11 Vignette

Darken edges:

```
Scene -> Vignette -> Darkened corners
```

5.9.12 Post-Processing Chain

Combine multiple effects:

```
Scene
    |
RenderToTexture
```

```
    |
SSAO
    |
Bloom
    |
ColorGrading
    |
ChromaticAberration
    |
Vignette
    |
Final Output
```

5.10 Scene Management

5.10.1 Scene Hierarchy

Organize complex scenes:

```
MainLoop -> Camera
    |
Scene (root)
    +-- Environment
    |   +-- Skybox
    |   +-- Fog
    +-- Lighting
    |   +-- AmbientLight
    |   +-- DirectionalLight
    |   +-- PointLights (array)
    +-- Static Objects
    |   +-- [Buildings, terrain, etc.]
    +-- Dynamic Objects
    |   +-- [Characters, vehicles, etc.]
    +-- Effects
        +-- Particles
        +-- Post-processing
```

5.10.2 Object Grouping

Group related objects:

```
Group (name: "Characters")
    +-- Character1
    +-- Character2
    +-- Character3
```

5.10.3 Layer System

Use layers for organization:

```
Layer 0: Background
Layer 1: Environment
Layer 2: Characters
Layer 3: Effects
Layer 4: UI
```

5.10.4 Culling and Optimization

Hide objects outside view:

```
Object Position -> FrustumCull -> Only render if visible
```

5.10.5 LOD (Level of Detail) System

Use simpler models at distance:

```
Distance from camera -> If > threshold -> Use LOD model
```

5.11 Practical Examples

5.11.1 Example 1: Rotating Cube

```
MainLoop
    |
PerspectiveCamera
    |
DirectionalLight
    |
Time -> RotateY input
    |
PhongMaterial
    |
Cube
```

5.11.2 Example 2: Lit Sphere with Orbit Controls

```
MainLoop
    |
PerspectiveCamera -> OrbitControls
    |
AmbientLight (subtle)
    |
PointLight
    |
PBRMaterial (metalness: 1, roughness: 0.2)
    |
Sphere
```

5.11.3 Example 3: Loading a 3D Model

```
MainLoop
    |
PerspectiveCamera
    |
DirectionalLight
    |
GLTFLoader (your model.glb)
    |
Transform (scale/position)
```

5.11.4 Example 4: Solar System

```
MainLoop
    |
PerspectiveCamera -> OrbitControls
    |
AmbientLight (space ambient)
    |
DirectionalLight (sun)
    |
[Sun] - Static sphere with emissive material
    |
[Planet1] - Transform (orbit around sun)
    |   +-- Time -> RotateY (orbit)
    |   +-- Time -> RotateY (self-rotation)
    |       +-- Sphere
    |
[Planet2] - Different orbit speed
    +-- [Moon] - Orbits planet
```

5.11.5 Example 5: Procedural Terrain

```
MainLoop
    |
PerspectiveCamera -> OrbitControls
    |
DirectionalLight
    |
IteratorLoop (grid: 100x100)
    |
Position -> NoiseTexture (3D noise) -> Height
    |
Calculate vertex (X, height, Z)
    |
Calculate normal from neighbors
    |
CustomGeometry
    |
PBRMaterial (terrain textures)
```

5.11.6 Example 6: Instanced Forest

```
MainLoop
    |
PerspectiveCamera
    |
DirectionalLight
    |
TreeModel (loaded GLTF)
    |
ArrayIterator (1000 positions)
    |
Random -> Scale variation
Random -> Rotation variation
    |
InstanceTransform
    |
InstancedMesh
```

5.11.7 Example 7: Interactive 3D Scene

```
MainLoop
    |
PerspectiveCamera -> OrbitControls
    |
MouseX -> Map -> Light Direction X
MouseY -> Map -> Light Direction Y
    |
DirectionalLight
    |
MouseClick -> Toggle -> Object visibility
```

```
    |
[Scene objects]
```

5.11.8 Example 8: Animated Character

```
MainLoop
    |
PerspectiveCamera
    |
DirectionalLight
    |
CharacterModel
    |
Timeline
    +-- Frame 0: Idle pose
    +-- Frame 30: Walk cycle start
    +-- Frame 60: Walk cycle end
    +-- [Loop]
    |
Apply to skeleton
    |
AnimatedMesh
```

5.11.9 Example 9: Particle System

```
MainLoop
    |
PerspectiveCamera
    |
ArrayIterator (particles)
    |
Particle Data
    +-- Position (update with velocity)
    +-- Velocity (update with forces)
    +-- Life (decrease over time)
    +-- Size (scale with life)
    |
Transform (position, scale)
    |
BasicMaterial (color from life)
    |
Sphere (small)
```

5.11.10 Example 10: Reflective Surface

```
MainLoop
    |
PerspectiveCamera
    |
[Scene to reflect]
    |
RenderToTexture (reflection view)
    |
CubemapTexture
    |
PBRMaterial (reflection map)
    |
Plane (mirror surface)
```

5.11.11 Example 11: Volumetric Fog

```
MainLoop
    |
PerspectiveCamera
    |
Scene
    |
RenderToTexture (depth)
    |
VolumetricFog
    +-- Depth texture
    +-- Noise texture (for variation)
    +-- Light direction
    |
Blend with scene
```

5.11.12 Example 12: Dynamic Lighting Setup

```
MainLoop
    |
PerspectiveCamera
    |
Time -> Sin -> Sun angle
    |
Sun angle -> Calculate direction
    |
DirectionalLight (sun)
    +-- Color (warm -> cool based on angle)
    +-- Intensity (day -> night)
    |
AmbientLight
    +-- Intensity (complement sun)
    |
[Scene]
```

5.11.13 Example 13: Morphing Objects

```
MainLoop
    |
PerspectiveCamera
    |
Time -> Sin -> Morph factor (0 to 1)
    |
Mesh1 -> Morph -> Mesh2
    |
Material
```

5.11.14 Example 14: Physics Simulation

```
MainLoop
    |
PerspectiveCamera
    |
PhysicsWorld
    +-- Gravity
    +-- Colliders
    |
PhysicsBody (rigid body)
    +-- Mass
    +-- Forces
    +-- Collisions
    |
Transform (from physics)
    |
Mesh
```

5.11.15 Example 15: Post-Processing Pipeline

```
MainLoop
    |
PerspectiveCamera
    |
[Render scene]
    |
RenderToTexture
    |
SSAO
    |
Bloom (extract bright areas)
    |
Blur (bloom)
    |
Add bloom back
    |
ColorGrading
```

```
    +-- Exposure
    +-- Contrast
    +-- Saturation
    |
ChromaticAberration
    |
Vignette
    |
Final output
```

5.11.16 Example 16: Audio-Reactive 3D

```
MainLoop
    |
PerspectiveCamera
    |
AudioAnalyzer -> FFTArray
    |
ArrayIterator (frequency bands)
    |
FFT Value -> Scale Y
    |
Transform (position from index, scale from FFT)
    |
Cube (bar visualization)
```

5.11.17 Example 17: Procedural City

```
MainLoop
    |
PerspectiveCamera -> OrbitControls
    |
DirectionalLight
    |
IteratorLoop (grid: city blocks)
    |
NoiseTexture -> Building height
Random -> Building type
    |
Transform (position, height)
    |
Cube (building)
    |
PBRMaterial (building texture)
```

5.11.18 Example 18: Water Surface

```
MainLoop
    |
PerspectiveCamera
    |
Time -> Sin -> Wave offset
    |
Plane (subdivided)
    |
Vertex shader (displace vertices)
    |
WaterMaterial
    +-- Normal map (animated)
    +-- Reflection (scene)
    +-- Refraction
    +-- Foam (at edges)
```

5.11.19 Example 19: Portal Effect

```
MainLoop
    |
PerspectiveCamera
    |
[Main scene]
    |
PortalCamera (different view)
```

```
    |
RenderToTexture (portal view)
    |
Plane (portal frame)
    |
Material (portal texture)
    |
Stencil buffer (mask to portal shape)
```

5.11.20 Example 20: Multi-Pass Rendering

```
MainLoop
    |
PerspectiveCamera
    |
[Pass 1: Opaque objects]
    |
RenderToTexture
    |
[Pass 2: Transparent objects]
    |
Blend with Pass 1
    |
[Pass 3: Effects]
    |
Blend all passes
    |
Post-processing
```

5.12 Advanced Animation Techniques

5.12.1 Skeletal Animation

Animate characters with bones:

```
Skeleton (bone hierarchy)
    |
Animation data (keyframes)
    |
Skin weights (vertex -> bone influence)
    |
AnimatedMesh
```

5.12.2 Morph Targets

Blend between shape variations:

```
BaseMesh -> MorphTarget1 (blend factor) -> MorphTarget2
```

Use Cases: - Facial expressions - Shape variations - Smooth transitions

5.12.3 Procedural Animation

Generate animation with code:

```
Time -> Math functions -> Transform values
    |
Apply to objects
```

5.12.4 Physics Animation

Use physics for natural movement:

```
PhysicsBody -> Forces -> Motion -> Transform
```

5.12.5 Animation Blending

Smoothly transition between animations:

```
Animation1 -> Blend -> Animation2 (blend factor)
```

5.13 Performance Optimization

5.13.1 General Tips

1. **Reduce Polygon Count** - Use lower-poly models when possible
2. **Texture Atlas** - Combine textures to reduce draw calls
3. **Level of Detail (LOD)** - Use simpler models for distant objects
4. **Frustum Culling** - Built-in, but organize scenes efficiently
5. **Bake Lighting** - Pre-calculate lighting for static scenes

5.13.2 Advanced Optimization

Occlusion Culling:

```
Object -> Check if occluded -> Skip rendering
```

Batching:

```
Similar objects -> Batch -> Single draw call
```

Texture Compression: - Use compressed texture formats (DXT, ETC) - Reduce texture resolution when possible - Use mipmaps for distant objects

Geometry Optimization: - Remove unnecessary vertices - Use indexed geometry - Optimize UV mapping

Shader Optimization: - Minimize texture samples - Use simpler shaders when possible - Avoid branching in shaders

Render Target Optimization: - Use appropriate render target sizes - Don't render at higher resolution than display - Use half-precision floats when possible

5.13.3 Performance Monitoring

Track performance metrics:

```
PerformanceMonitor
    +-- FPS
    +-- Draw calls
    +-- Triangle count
    +-- Texture memory
    +-- Shader compilation time
```

5.13.4 Adaptive Quality

Adjust quality based on performance:

```
FPS -> If < 30 -> Reduce quality
    +-- Lower LOD
    +-- Disable effects
    +-- Reduce particle count
```

5.14 Common Patterns and Workflows

5.14.1 Pattern: Object Pooling

Reuse objects instead of creating/destroying:

```
Pool of inactive objects
    |
Activate when needed
    |
Deactivate when done
    |
Return to pool
```

5.14.2 Pattern: Component System

Organize object behavior:

```
GameObject
    +-- Transform component
    +-- Render component
    +-- Physics component
    +-- Script component
```

5.14.3 Pattern: Event System

Decouple object interactions:

```
EventEmitter
    +-- Subscribe (listener)
    +-- Emit (event)
    |
Objects react to events
```

5.14.4 Pattern: State Machine

Manage object states:

```
StateMachine
    +-- Idle state
    +-- Active state
    +-- Transition conditions
```

5.15 Debugging 3D Scenes

5.15.1 Visual Debugging

Show Normals:

```
Mesh -> DebugNormals -> Visualize normals
```

Show Bounding Boxes:

```
Mesh -> DebugBounds -> Show bounding boxes
```

Show Wireframe:

```
Material -> Wireframe mode -> See geometry
```

Show Grid:

```
GridHelper -> Visual reference
```

5.15.2 Common Issues

"Objects not visible" - Check camera position and direction - Verify objects are within near/far planes - Check material alpha values - Verify lighting setup

"Shadows look wrong" - Adjust shadow bias - Increase shadow map resolution - Check light shadow settings - Verify shadow receiving objects

"Performance is slow" - Reduce polygon count - Lower texture resolutions - Disable expensive effects - Use LOD system - Optimize shaders

"Materials look incorrect" - Verify texture UV mapping - Check normal map orientation - Verify PBR material values - Check lighting setup

5.16 Best Practices

1. **Start Simple** - Build complexity gradually
2. **Optimize Early** - Consider performance from the start
3. **Use Instancing** - For repeated objects
4. **Organize Scenes** - Use hierarchies and groups
5. **Test on Target Hardware** - Performance varies by device
6. **Use Appropriate Formats** - GLTF for models, compressed textures
7. **Profile Regularly** - Use performance tools
8. **Document Complex Setups** - Add comments to patches
9. **Version Control** - Save iterations of complex scenes
10. **Reuse Assets** - Don't duplicate unnecessarily

5.17 Featured Videos

5.18 Exercises

5.18.1 Beginner

1. Create a solar system with orbiting planets
2. Build a simple room with multiple light sources
3. Load a 3D model and add interactive rotation controls

4. Create a rotating cube with different materials
5. Build a simple scene with fog

5.18.2 Intermediate

6. Create a procedural terrain with noise
7. Build an instanced forest with 100+ trees
8. Implement a three-point lighting setup
9. Create a water surface with animated waves
10. Build a particle system with physics
11. Create a portal effect with dual cameras
12. Implement post-processing effects (bloom, SSAO)
13. Build an audio-reactive 3D visualization
14. Create a morphing object animation
15. Implement a character with skeletal animation

5.18.3 Advanced

16. Build a complete scene with LOD system
17. Create a volumetric fog effect
18. Implement screen-space reflections
19. Build a physics-based simulation
20. Create a procedural city generator
21. Implement a multi-pass rendering pipeline
22. Build an interactive 3D game scene
23. Create advanced post-processing chain
24. Implement custom shader materials
25. Build a complex scene with optimization techniques

5.19 Project Ideas

1. **3D Product Viewer** - Interactive product showcase
2. **Architectural Visualization** - Building walkthrough
3. **Game Prototype** - Simple 3D game mechanics
4. **Data Visualization** - 3D charts and graphs
5. **Virtual Gallery** - 3D art exhibition
6. **Interactive Installation** - Museum or event display

7. **Music Visualizer** - 3D audio-reactive visuals
8. **Procedural World** - Generated landscape exploration
9. **Character Animation** - Animated character showcase
10. **Physics Sandbox** - Interactive physics playground

---

6 Texturing in Cables.gl

6.1 Introduction to Textures

Textures add detail, color, and realism to your visuals. In cables.gl, textures can come from images, videos, webcams, or be generated procedurally.

6.2 Loading Textures

6.2.1 ImageTexture

Load images from files or URLs:

```
ImageTexture -> Material (texture input)
```

Supported Formats: - PNG (with transparency) - JPG - WebP - GIF (first frame or animated)

Key Parameters: - `URL` - Path to image - `Filter` - Nearest (pixelated) or Linear (smooth) - `Wrap` - Repeat, Clamp, Mirror

6.2.2 VideoTexture

Use video as a texture:

```
VideoTexture -> Material (texture input)
```

Key Parameters: - `URL` - Path to video file - `Loop` - Whether to loop playback - `Playback Rate` - Speed control - `Volume` - Audio volume

Supported Formats: - MP4 (H.264) - WebM

6.2.3 WebcamTexture

Live webcam input as a texture:

```
WebcamTexture -> Material (texture input)
```

Tip: Great for interactive installations!

6.3 Texture Mapping

6.3.1 UV Coordinates

UV coordinates define how textures wrap onto geometry:

```
+-------------------------------------------------------------+
|                    UV COORDINATE SYSTEM                      |
+-------------------------------------------------------------+
|                                                               |
|  Texture (Image)                                              |
|  +--------------------------+                               |
|  | U=0,V=0        U=1,V=0  |  <- Top edge                    |
|  |                         |                               |
|  |                         |                               |
|  |                         |                               |
|  | U=0,V=1        U=1,V=1  |  <- Bottom edge               |
|  +--------------------------+                               |
|                                                               |
|  Mapped to 3D Geometry:                                      |
|                                                               |
|        U=0,V=0 ------------ U=1,V=0                         |
|           |                      |                           |
|           |     3D Surface      |                           |
|           |                      |                           |
|        U=0,V=1 ------------ U=1,V=1                         |
|                                                               |
|  - U = Horizontal (0 to 1, left to right)                   |
|  - V = Vertical (0 to 1, top to bottom)                    |
|                                                               |
+-------------------------------------------------------------+
```

- **U** = Horizontal position (0 to 1)
- **V** = Vertical position (0 to 1)

Most primitive shapes have automatic UV mapping.

6.3.2 UV Transform

Modify texture coordinates:

```
+-------------------------------------------------------------+
|                   UV TRANSFORM OPERATIONS                    |
+-------------------------------------------------------------+
|                                                               |
|  Original Texture                                            |
|  +---------+                                                 |
|  |         |                                                 |
|  | Texture |                                                 |
|  |         |                                                 |
|  +---------+                                                 |
|                                                               |
|  Offset U/V: Shift texture position                          |
|  +---------+                                                 |
|  |    +---------+                                           |
|  |    | Texture |  <- Moved right/up                        |
|  |    +---------+                                           |
|  +---------+                                                 |
|                                                               |
|  Scale U/V: Tile or shrink texture                           |
|  +-----+-----+-----+                                        |
|  |Tex  |Tex  |Tex  |  <- Tiled horizontally                |
|  +-----+-----+-----+                                        |
|                                                               |
|  Rotate: Rotate texture around center                       |
|      +-----+                                                 |
|     /       \                                                |
|    / Texture \  <- Rotated                                     |
|     \       /                                                |
|      +-----+                                                 |
|                                                               |
+-------------------------------------------------------------+
TextureTransform -> Before texture application
```

Parameters: - `Offset U/V` - Shift the texture - `Scale U/V` - Tile or shrink - `Rotate` - Rotate the texture

6.3.3 Tiling Textures

For seamless repeating:

1. Set wrap mode to `Repeat`
2. Scale UV coordinates > 1

6.4 Advanced Texture Workflow (Production Mindset)

Texturing is where many cables.gl projects move from "cool prototype" to "polished piece". The two recurring themes are:

- **Correctness**: color space, alpha handling, UVs, aspect ratios, and predictable sampling.
- **Performance**: texture sizes, filtering, mipmaps, compression, and "how many textures are you sampling per pixel".

6.4.1 Color Space: sRGB vs Linear (Why Your Colors Look "Off")

Most images you download (JPG/PNG/WebP) are authored in **sRGB** (gamma corrected). Most lighting and shading math expects **linear** values. If your project mixes lit materials (e.g., PBR) with UI-like textures, you can run into:

- washed-out or too-dark textures
- incorrect blending
- "metal looks wrong" in PBR

Practical rule of thumb:

- **Color/albedo** textures are usually **sRGB**.
- **Data** textures (normal maps, roughness/metalness/AO, masks) are usually **linear**.

If a texture looks wrong, verify you're not treating a data map like a color map (or vice versa).

6.4.2 Alpha (Transparency) Pitfalls

If you see dark/bright halos around transparent textures (logos, sprites), you're likely looking at one of these issues:

- The texture was exported with a bad matte color (common in PNGs).
- The pipeline expects **premultiplied alpha** but you provided straight alpha (or the other way around).
- Filtering/mipmaps sample transparent pixels and "bleed" colors into edges.

Fix strategies:

- Add padding/bleed around sprites in your source image.
- Prefer power-of-two textures with mipmaps for distant rendering.
- If you have control over asset export, re-export with correct alpha handling.

6.4.3 Filtering, Mipmaps, and Why Textures "Shimmer"

When a textured surface gets small on screen, the GPU needs mipmaps to avoid shimmer and crawling.

- **Nearest** filtering: crisp pixels, great for pixel-art, terrible for most 3D.
- **Linear** filtering: smoother sampling, better for general use.
- **Mipmaps**: essential for 3D surfaces viewed at varying distances.

If a ground texture "crawls" when the camera moves, you typically need mipmaps and (if available) anisotropic filtering.

6.4.4 Power-of-Two Sizes (and When It Matters)

Power-of-two textures (256/512/1024/2048/4096) generally behave better for:

- mipmaps
- repeating wrap modes
- GPU compatibility/performance

Non-power-of-two often still works in modern WebGL, but when things behave oddly, returning to power-of-two sizes is a reliable fix.

6.4.5 Aspect Ratio Correctness (Especially for Video)

Video textures are a frequent source of "why is it stretched?" issues.

- Match the **Plane** aspect ratio to the video's aspect ratio.
- If you use Fullscreen rectangles, make sure you're compensating for screen aspect.

6.5 Advanced Techniques and Patch Recipes

These are "building block" patterns you can reuse across many projects.

6.5.1 Recipe: Masked Texture Blend (Two Textures + a Mask)

Use a mask texture (black/white) to blend between two images.

Conceptual chain:

```
ImageTexture (A) -+
                  +-> (blend using mask) -> Material -> Mesh
ImageTexture (B) -+
ImageTexture (Mask)
```

Notes: - The mask should be treated as a data texture (linear). - Great for dirt overlays, decals, and transitions.

6.5.2 Recipe: Animated UVs (Scrolling / Parallax)

Scrolling textures are perfect for conveyor belts, moving backgrounds, water normals, etc.

```
Time -> (speed multiply) -> TextureTransform (Offset U/V)
ImageTexture -> Material (texture input)
Material -> Mesh
```

6.5.3 Recipe: Render-to-Texture for Post-Processing

Render your scene to a texture, apply effects, then output.

```
MainLoop -> Camera -> RenderToTexture
              |
           [Scene]
              |
TextureEffects -> Output
```

Use Cases: - blur/glow chains - color grading - stylized distortion - feedback trails (see next recipe)

6.5.4 Recipe: Feedback / Trails (Texture Feedback Loop)

Feedback is a signature look in real-time visuals.

High-level structure:

```
Previous Frame Texture
        |
TextureEffects (fade/blur)
        |
Combine with New Frame Content
        |
RenderToTexture (becomes “previous frame” next tick)
```

Tip: Keep feedback subtle (small fade each frame). Large blur + high persistence can become very expensive.

6.5.5 Recipe: Planar “Mirror” Reflection (Render-to-Texture)

To fake a mirror floor:

- Render the scene from a reflected camera to a texture.
- Apply that texture onto a plane.

```
MainLoop
  +-> Camera (main) -> [Scene]
  +-> Camera (reflected) -> RenderToTexture -> Plane Material -> Mirror Plane
```

6.5.6 Recipe: Environment Reflections (Cubemap/HDRI)

Use an environment texture for reflections and more believable PBR materials.

```
HDRITexture or CubemapTexture -> (environment input) -> PBRMaterial -> Mesh
```

Tip: Even simple objects look dramatically better with good environment lighting.

6.5.7 Recipe: Video Texture "Billboard" (Reliable Playback)

```
VideoTexture -> BasicMaterial -> Plane
```

Checklist: - Use a browser-served URL (avoid `file://` in production). - Make sure autoplay policies are satisfied (user interaction may be required). - Use a fallback poster image if video takes time to load.

6.5.8 Recipe: Webcam Texture (Permissions + UX)

```
WebcamTexture -> BasicMaterial -> Plane
```

Checklist: - Provide a UI prompt ("Click to enable camera"). - Handle denied permissions gracefully (fallback texture). - Keep resolution reasonable for performance.

6.6 Texture Types for PBR Materials

6.6.1 Albedo/Diffuse Map

The base color of the surface.

6.6.2 Normal Map

Adds surface detail without extra geometry.

```
NormalMap -> PBRMaterial (normal input)
```

Tip: Use tangent-space normal maps (blue-purple appearance).

6.6.3 Roughness Map

Controls surface smoothness per-pixel.

- White = rough
- Black = smooth/shiny

6.6.4 Metalness Map

Defines metallic vs. non-metallic regions.

- White = metal
- Black = non-metal (dielectric)

6.6.5 Ambient Occlusion Map

Pre-baked shadow information for crevices.

6.6.6 Height/Displacement Map

Actual geometry displacement (more expensive).

6.6.7 Emissive Map

Self-illuminating regions of the surface.

6.7 Procedural Textures

Generate textures with code/nodes:

6.7.1 Noise Textures

```
NoiseTexture -> Creates Perlin/Simplex noise
```

Types: - Perlin noise - Simplex noise - Voronoi - Fractal/FBM

6.7.2 Gradient Textures

```
GradientTexture -> Creates color gradients
```

6.7.3 Pattern Generators

- Checkerboard
- Stripes
- Dots
- Custom math-based patterns

6.8 Render to Texture

Capture your scene as a texture for post-processing or effects:

```
MainLoop -> Camera -> RenderToTexture
                 |
           [Scene to capture]
                 |
           TextureOutput -> Use elsewhere
```

6.8.1 Common Uses:

1. **Post-processing effects** - Apply shaders to the entire scene
2. **Mirrors/Reflections** - Render from reflection viewpoint
3. **Dynamic textures** - Use one patch's output in another
4. **Feedback effects** - Feed output back as input

6.9 Texture Effects

6.9.1 TextureEffects Op

Chain of image processing effects:

```
ImageTexture -> TextureEffects -> Output
```

Available Effects: - Blur - Sharpen - Color correction - Distortion - Edge detection - Pixelation

6.9.2 Custom Shader Effects

Write GLSL for custom texture processing (see Shaders chapter).

6.10 Cubemaps and Environment Maps

6.10.1 CubemapTexture

Six images forming a surrounding environment:

```
CubemapTexture -> Environment lighting
```

Uses: - Skyboxes - Reflections - Image-based lighting (IBL)

6.10.2 HDRITexture

High Dynamic Range images for realistic lighting:

```
HDRITexture -> IBL/Environment
```

6.11 Texture Compression and Optimization

6.11.1 File Size Tips:

1. **Use appropriate formats:**
   - PNG for transparency
   - JPG for photos (no transparency)
   - WebP for best compression
2. **Power of 2 sizes:** 256, 512, 1024, 2048, 4096 pixels
3. **Mipmaps:** Enable for textures viewed at varying distances
4. **Compress textures:** Use tools like TinyPNG, Squoosh

6.11.2 Memory Considerations:

Size	Approximate Memory
512x512	~1 MB
1024x1024	~4 MB
2048x2048	~16 MB
4096x4096	~64 MB

6.12 Practical Examples

6.12.1 Example 1: Textured Rotating Cube

```
MainLoop
    |
PerspectiveCamera
    |
DirectionalLight
    |
Time -> RotateY
    |
ImageTexture -> PhongMaterial (texture input)
    |
Cube
```

6.12.2 Example 2: Video on a Plane

```
MainLoop
    |
VideoTexture -> BasicMaterial
    |
Plane (aspect ratio matching video)
```

6.12.3 Example 3: Animated Noise Background

```
MainLoop
    |
Time -> NoiseTexture (animate offset)
    |
BasicMaterial
    |
FullscreenRectangle
```

6.12.4 Example 4: PBR Textured Material

```
ImageTexture (albedo)
ImageTexture (normal)
ImageTexture (roughness)
ImageTexture (metalness)
```

```
    | (all connected to PBRMaterial)
PBRMaterial
    |
Mesh
```

6.13 Featured Videos

6.14 Exercises

1. Create a textured cube that rotates and displays different images on each face
2. Build a video wall with multiple video textures
3. Create a procedural noise-based animated background
4. Apply PBR textures to a loaded 3D model

---

7 Shaders & GLSL in Cables.gl

7.1 Introduction to Shaders

Shaders are programs that run on the GPU, enabling custom visual effects and rendering techniques. Cables.gl provides powerful tools for writing and using GLSL (OpenGL Shading Language) shaders.

7.2 What Are Shaders?

Shaders are small programs that determine how graphics are rendered:

- **Vertex Shaders** - Transform vertex positions
- **Fragment Shaders** - Determine pixel colors

Together, they control everything you see on screen.

7.3 Why Use Custom Shaders?

- Create unique visual effects
- Achieve effects impossible with built-in ops
- Optimize performance for specific use cases
- Learn the fundamentals of graphics programming

7.4 Shader Ops in Cables.gl

7.4.1 ShaderMaterial

Apply custom GLSL code as a material:

```
ShaderMaterial -> Mesh
```

7.4.2 TextureEffect (Shader-based)

Process textures with custom fragment shaders.

7.4.3 CustomShader

Full control over vertex and fragment shaders.

7.5 GLSL Basics

7.5.1 Data Types

```
// Scalars
float a = 1.0;
int b = 5;
bool c = true;

// Vectors
vec2 uv = vec2(0.5, 0.5);
vec3 color = vec3(1.0, 0.0, 0.0);  // RGB
vec4 rgba = vec4(1.0, 1.0, 1.0, 1.0);

// Matrices
mat4 transform;

// Samplers (textures)
sampler2D myTexture;
```

7.5.2 Swizzling

Access vector components in any order:

```
vec4 color = vec4(1.0, 0.5, 0.25, 1.0);
vec3 rgb = color.rgb;    // (1.0, 0.5, 0.25)
vec2 rg = color.rg;      // (1.0, 0.5)
float r = color.r;       // 1.0
vec3 bgr = color.bgr;    // (0.25, 0.5, 1.0) - reversed!
```

7.5.3 Built-in Functions

```
// Math
sin(x), cos(x), tan(x)
pow(x, y)
sqrt(x)
abs(x)
min(a, b), max(a, b)
clamp(x, min, max)

// Interpolation
mix(a, b, t)           // Linear interpolation
smoothstep(edge0, edge1, x)

// Vector operations
length(v)
normalize(v)
dot(a, b)
cross(a, b)
reflect(incident, normal)

// Texture sampling
texture(sampler, uv)
```

7.6 Your First Fragment Shader

A simple color gradient:

```
// Fragment Shader
precision mediump float;

varying vec2 vUV;  // UV coordinates from vertex shader

void main() {
    // Create gradient based on UV
    vec3 color = vec3(vUV.x, vUV.y, 0.5);

    gl_FragColor = vec4(color, 1.0);
}
```

7.7 Common Shader Patterns

7.7.1 Solid Color

```
void main() {
    gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);  // Red
}
```

7.7.2 UV Gradient

```
void main() {
    gl_FragColor = vec4(vUV, 0.0, 1.0);
}
```

7.7.3 Circle (SDF)

```
void main() {
    vec2 center = vec2(0.5, 0.5);
    float dist = length(vUV - center);
    float circle = step(dist, 0.3);

    gl_FragColor = vec4(vec3(circle), 1.0);
}
```

7.7.4 Smooth Circle

```
void main() {
    vec2 center = vec2(0.5, 0.5);
    float dist = length(vUV - center);
    float circle = smoothstep(0.3, 0.28, dist);

    gl_FragColor = vec4(vec3(circle), 1.0);
}
```

7.7.5 Animated Pattern

```
uniform float time;

void main() {
    float wave = sin(vUV.x * 10.0 + time) * 0.5 + 0.5;
    gl_FragColor = vec4(vec3(wave), 1.0);
}
```

7.8 Uniforms

Uniforms are values passed from cables.gl to your shader:

```
uniform float time;         // Current time
uniform vec2 resolution;    // Canvas size
uniform sampler2D tex;      // Texture
uniform vec3 color;         // Custom color
```

In cables.gl, connect ops to shader uniform inputs.

7.9 Advanced Shader Workflows in cables.gl

The biggest jump in quality comes from treating shaders like reusable “modules”:

- a **clear input contract** (uniforms you expect: time, resolution, textures, parameters)
- predictable **coordinate conventions** (UV vs screen space vs world space)
- a **debug strategy** (visualize intermediate values)
- performance awareness (texture samples, loops, precision)

7.9.1 A Practical Uniform “Contract”

In most patches you’ll end up with a small set of recurring uniforms:

- `time` (float): animation driver
- `resolution` (vec2): coordinate normalization
- `tex` / `tex0` / `tex1` (sampler2D): one or more textures
- `amount` / `strength` (float): effect intensity
- `colorA` / `colorB` (vec3): palette endpoints

Tip: name your uniforms consistently so you can reuse the same patch wiring across multiple shader materials/effects.

7.9.2 Coordinate Spaces: UV vs Screen Space

- **UV space** (`vUV`) is normalized 0..1 per surface.
- **Screen space** is often derived from UV + resolution when you need pixel-sized offsets.

Example helper:

```
vec2 pixel(vec2 uv, vec2 resolution) {
    return 1.0 / resolution;
}
```

7.9.3 Anti-Aliasing SDFs (Clean Edges)

Hard `step()` edges often look jagged. A common pattern is to use `smoothstep()` with a small "feather":

```
float aa(float dist, float radius) {
    float edge = 0.002; // tweak for your resolution / style
    return 1.0 - smoothstep(radius - edge, radius + edge, dist);
}
```

When available, `fwidth()` can provide adaptive edge widths, but keep in mind WebGL precision/derivative constraints in some contexts.

7.9.4 Palette Mapping (Better Color Fast)

Instead of picking random RGB values, map a scalar to a palette:

```
vec3 palette(float t, vec3 a, vec3 b, vec3 c, vec3 d) {
    return a + b * cos(6.28318 * (c * t + d));
}
```

This gives you rich gradients with a tiny amount of code.

7.10 Advanced Examples (Copy-and-Adapt)

These examples are written so you can drop them into a ShaderMaterial/TextureEffect-style fragment shader and then wire the uniforms from your patch.

7.10.1 Example: Texture Distortion (UV Warp)

```
precision mediump float;
varying vec2 vUV;
uniform sampler2D tex;
uniform float time;
uniform float amount;

void main() {
    vec2 uv = vUV;
    uv.x += sin(uv.y * 10.0 + time) * amount;
    uv.y += cos(uv.x * 10.0 + time) * amount;
    gl_FragColor = texture2D(tex, uv);
}
```

Patch wiring idea: - `Time` -> `time` - a slider (0..0.05) -> `amount` - input texture -> `tex`

7.10.2 Example: Simple Bloom-ish Glow (Threshold + Blur-ish)

This isn't a full separable blur, but it demonstrates the "sample neighbors" pattern.

```
precision mediump float;
varying vec2 vUV;
uniform sampler2D tex;
uniform vec2 resolution;
uniform float threshold;
uniform float strength;

void main() {
    vec2 px = 1.0 / resolution;
    vec3 c = texture2D(tex, vUV).rgb;
    // crude 5-tap blur
    vec3 b = vec3(0.0);
    b += texture2D(tex, vUV + vec2( 1.0, 0.0) * px).rgb;
    b += texture2D(tex, vUV + vec2(-1.0, 0.0) * px).rgb;
    b += texture2D(tex, vUV + vec2( 0.0, 1.0) * px).rgb;
    b += texture2D(tex, vUV + vec2( 0.0,-1.0) * px).rgb;
    b *= 0.25;

    float luma = dot(c, vec3(0.299, 0.587, 0.114));
    vec3 glow = (luma > threshold) ? b : vec3(0.0);

    gl_FragColor = vec4(c + glow * strength, 1.0);
}
```

7.10.3 Example: Domain Warping (More Organic Noise)

Domain warping is a standard “make it look expensive” trick: distort the coordinates before sampling noise.

```
precision mediump float;
varying vec2 vUV;
uniform float time;

float hash(vec2 p) {
    return fract(sin(dot(p, vec2(127.1, 311.7))) * 43758.5453);
}

float noise(vec2 p) {
    vec2 i = floor(p);
    vec2 f = fract(p);
    float a = hash(i);
    float b = hash(i + vec2(1.0, 0.0));
    float c = hash(i + vec2(0.0, 1.0));
    float d = hash(i + vec2(1.0, 1.0));
    vec2 u = f * f * (3.0 - 2.0 * f);
    return mix(a, b, u.x) + (c - a) * u.y * (1.0 - u.x) + (d - b) * u.x * u.y;
}

void main() {
    vec2 uv = vUV * 4.0;
    vec2 warp = vec2(
        noise(uv + time * 0.2),
        noise(uv + vec2(5.2, 1.3) - time * 0.2)
    );
    float n = noise(uv + warp * 2.0);
    gl_FragColor = vec4(vec3(n), 1.0);
}
```

7.11 Debugging Shaders (In Practice)

When something is wrong, render the intermediate value:

- visualize UVs: `gl_FragColor = vec4(vUV, 0.0, 1.0);`
- visualize a scalar: `gl_FragColor = vec4(vec3(val), 1.0);`
- isolate channels: `gl_FragColor = vec4(texture2D(tex, vUV).rrr, 1.0);`

7.11.1 Common Gotchas

- **Black output**: your shader compiles but outputs 0 (check uniform wiring; check ranges).
- **Solid color**: UVs are constant or your sampling coord is wrong.
- **Stretching**: you're using UVs but expect square pixels; incorporate `resolution`.
- **Banding**: precision too low; consider `highp` where supported, or dither slightly.

7.12 Performance Guidelines (Real-Time Friendly)

- **Texture samples are expensive**: keep them minimal and reuse results.
- **Avoid nested loops**: especially dynamic loops in fragment shaders.
- **Prefer simple math over heavy branching**: GPUs dislike divergent branches.
- **Keep effects modular**: multiple simpler passes can be easier to tune than one huge shader.

7.13 Signed Distance Functions (SDFs)

SDFs define shapes mathematically:

7.13.1 SDF Primitives

```
// Circle
float sdCircle(vec2 p, float r) {
    return length(p) - r;
}

// Box
float sdBox(vec2 p, vec2 b) {
    vec2 d = abs(p) - b;
    return length(max(d, 0.0)) + min(max(d.x, d.y), 0.0);
}

// Line segment
float sdSegment(vec2 p, vec2 a, vec2 b) {
    vec2 pa = p - a, ba = b - a;
    float h = clamp(dot(pa, ba) / dot(ba, ba), 0.0, 1.0);
    return length(pa - ba * h);
}
```

7.13.2 SDF Operations

```
// Union (combine shapes)
float opUnion(float d1, float d2) {
    return min(d1, d2);
}

// Subtraction (cut one from another)
float opSubtract(float d1, float d2) {
    return max(-d1, d2);
}

// Intersection (overlap only)
float opIntersect(float d1, float d2) {
    return max(d1, d2);
}
```

```
// Smooth union
float opSmoothUnion(float d1, float d2, float k) {
    float h = clamp(0.5 + 0.5 * (d2 - d1) / k, 0.0, 1.0);
    return mix(d2, d1, h) - k * h * (1.0 - h);
}
```

7.14 Noise Functions

7.14.1 Simple Value Noise

```
float random(vec2 st) {
    return fract(sin(dot(st.xy, vec2(12.9898, 78.233))) * 43758.5453);
}

float noise(vec2 st) {
    vec2 i = floor(st);
    vec2 f = fract(st);

    float a = random(i);
    float b = random(i + vec2(1.0, 0.0));
    float c = random(i + vec2(0.0, 1.0));
    float d = random(i + vec2(1.0, 1.0));

    vec2 u = f * f * (3.0 - 2.0 * f);

    return mix(a, b, u.x) + (c - a) * u.y * (1.0 - u.x) + (d - b) * u.x * u.y;
}
```

7.14.2 Fractal Brownian Motion (FBM)

```
float fbm(vec2 st) {
    float value = 0.0;
    float amplitude = 0.5;

    for (int i = 0; i < 5; i++) {
        value += amplitude * noise(st);
        st *= 2.0;
        amplitude *= 0.5;
    }

    return value;
}
```

7.15 Post-Processing Effects

7.15.1 Vignette

```
float vignette = 1.0 - length(vUV - 0.5) * 1.5;
color *= vignette;
```

7.15.2 Chromatic Aberration

```
precision mediump float;
varying vec2 vUV;
uniform sampler2D tex;

void main() {
    vec2 offset = (vUV - 0.5) * 0.01;
    float r = texture2D(tex, vUV + offset).r;
    float g = texture2D(tex, vUV).g;
    float b = texture2D(tex, vUV - offset).b;
    vec3 color = vec3(r, g, b);
    gl_FragColor = vec4(color, 1.0);
}
```

7.15.3 Blur (Box Blur)

```
precision mediump float;
varying vec2 vUV;
uniform sampler2D tex;

```

```
void main() {
    vec3 blur = vec3(0.0);
    float samples = 9.0;
    float offset = 0.005;

    for (float x = -1.0; x <= 1.0; x++) {
        for (float y = -1.0; y <= 1.0; y++) {
            blur += texture2D(tex, vUV + vec2(x, y) * offset).rgb;
        }
    }
    blur /= samples;
    gl_FragColor = vec4(blur, 1.0);
}
```

7.15.4 Pixelation

```
precision mediump float;
varying vec2 vUV;
uniform sampler2D tex;

void main() {
    float pixels = 100.0;
    vec2 pixelUV = floor(vUV * pixels) / pixels;
    vec3 color = texture2D(tex, pixelUV).rgb;
    gl_FragColor = vec4(color, 1.0);
}
```

7.16 Vertex Shader Basics

Modify geometry positions:

```
// Vertex Shader
attribute vec3 position;
attribute vec2 uv;

uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;
uniform float time;

varying vec2 vUV;

void main() {
    vUV = uv;

    vec3 pos = position;
    // Wave deformation
    pos.z += sin(pos.x * 5.0 + time) * 0.2;

    gl_Position = projectionMatrix * modelViewMatrix * vec4(pos, 1.0);
}
```

7.17 Debugging Shaders

7.17.1 Visualize Values

```
// Show UV coordinates
gl_FragColor = vec4(vUV, 0.0, 1.0);

// Show a value as grayscale
gl_FragColor = vec4(vec3(someValue), 1.0);

// Show negative values in red
float val = someCalculation;
if (val < 0.0) {
    gl_FragColor = vec4(-val, 0.0, 0.0, 1.0);
} else {
    gl_FragColor = vec4(0.0, val, 0.0, 1.0);
}
```

7.18 Performance Tips

1. **Avoid branching -** GPUs don't like if/else

2. **Use built-in functions** - They're optimized
3. **Minimize texture samples** - Each sample has cost
4. **Precision matters** - Use `mediump` when possible
5. **Precompute values** - Do math in JavaScript when possible

7.19 Professional Video Projection Mapping in Cables.gl

Projection mapping (also called video mapping or spatial augmented reality) involves projecting images onto real-world surfaces, often requiring geometric correction, multi-projector blending, and specialized color correction. This section provides professional-grade shaders for simulating and preparing projection mapping content within cables.gl.

All shaders in this section are designed for use with cables.gl's built-in `TextureEffect` or `ShaderMaterial` ops - simply paste the shader code into the fragment shader field and connect your inputs. For JavaScript custom op implementations, see the "JavaScript Custom Op Examples" section below.

7.19.1 Understanding Cables.gl Shader Context

Critical Notes for Cables.gl Shaders:

1. **Resolution Handling**: In cables.gl, `resolution` uniform is typically `vec2(width, height)` in pixels. When working with UV coordinates (`vUV`), remember:

   - `vUV` ranges from `0.0` to `1.0`
   - Screen space = `vUV * resolution`
   - Pixel size = `1.0 / resolution`
   - **Important:** `resolution` is NOT automatically provided - you must connect a `CanvasInfo` or `GetResolution` op to the `resolution` port

2. **Texture Sampling**: Always use `texture2D()` (WebGL 1.0 style) in cables.gl, not `texture()`.

3. **Coordinate Systems**:

   - UV space: `vUV` (0.0 to 1.0) - automatically provided
   - Screen space: `vUV * resolution`
   - Normalized screen space: `(vUV - 0.5) * 2.0` (ranges -1.0 to 1.0)

4. **Shader Headers**: Always include precision declaration at the top:

```
precision mediump float;
```

5. **Uniform Types**:
   - `float`, `vec2`, `vec3`, `vec4` - Fully supported, become Number/Vector ports
   - `sampler2D` - Fully supported, becomes Texture port
   - `mat3`, `mat4` - Supported, but verify with Matrix ops in your cables.gl version
   - `int` - **Not recommended** - Use `float` instead and compare with `< 0.5` patterns
6. **Auto-Provided Variables**:
   - `varying vec2 vUV` - Always available (no need to declare in vertex shader for Texture-Effect)
   - `uniform float time` - Available if you connect a Time op
   - `uniform vec2 resolution` - **NOT auto-provided** - must connect manually

7.19.2 Cables.gl Shader Compliance Checklist

Before using any shader in cables.gl, verify:

- ☐ Shader starts with `precision mediump float;`
- ☐ Uses `texture2D()` not `texture()` for sampling
- ☐ Uses `varying vec2 vUV` (auto-provided, don't declare in vertex shader for TextureEffect)
- ☐ No `uniform int` - converted to `uniform float` with float comparisons
- ☐ All uniforms are properly typed (float, vec2, vec3, vec4, sampler2D)
- ☐ Resolution uniform is documented as requiring manual connection
- ☐ Shader compiles without errors
- ☐ All texture samples are within 0.0-1.0 UV bounds (or clamped)
- ☐ No WebGL 2.0 specific features (use WebGL 1.0 compatible code)

7.19.3 Troubleshooting Common Issues

Issue: "Shader won't compile" - Check for `precision mediump float;` at the top - Verify all `texture()` calls are `texture2D()` - Ensure no WebGL 2.0 features are used - Check for syntax errors (missing semicolons, etc.)

Issue: "Black screen or no output" - Verify texture is connected to `tex` (or appropriate sampler2D) port - Check UV coordinates are in 0.0-1.0 range - Ensure resolution is connected if shader uses it - Check if shader is sampling outside texture bounds

Issue: "Resolution uniform not working" - `resolution` is NOT automatically provided - Connect `CanvasInfo` op or `GetResolution` op to `resolution` port - Verify resolution values are correct (width, height in pixels)

Issue: "Integer uniforms not working" - Cables.gl may not support `uniform int` reliably - Convert to `uniform float` and use float comparisons: - `if (direction == 0)` -> `if (direction < 0.5)` - `if (direction == 1)` -> `if (direction > 0.5 && direction < 1.5)`

Issue: "Matrix uniforms not working" - Verify your cables.gl version supports `mat3`/`mat4` - Use Matrix ops to create matrix values - Consider using `vec4` arrays or separate `vec2`/`vec3` values if matrices aren't supported

Issue: "Performance is poor" - Reduce texture samples per pixel - Use `mediump` precision (already done) - Avoid branching in shaders when possible - Consider breaking into multiple passes - Check if using custom JavaScript ops (adds overhead)

Issue: "Ports not appearing" - Ensure uniform declarations match exactly (case-sensitive) - Check uniform types are supported - Verify shader compiles successfully - Try recompiling the shader in TextureEffect

7.19.4 Using Shaders in Cables.gl: Two Approaches

Cables.gl offers two ways to use custom shaders:

Approach 1: Built-in Shader Ops (Recommended for Most Cases)

ShaderMaterial and **TextureEffect** ops automatically: - Create input ports for each `uniform` declaration - Provide `varying vec2 vUV` automatically - Handle shader compilation and execution on GPU - Require no JavaScript wrapper code

How to Use: 1. Add a `TextureEffect` op to your patch 2. Paste the shader code into the "Fragment Shader" field 3. Connect your textures and values to the automatically created ports 4. The shader runs directly on the GPU

Auto-Provided Uniforms: - `varying vec2 vUV` - Always available (0.0 to 1.0) - `uniform float time` - Available if you connect a Time op - `uniform vec2 resolution` - Available if you connect a Resolution/CanvasInfo op

Manual Uniforms: - All other `uniform` declarations become input ports automatically - Connect Texture ops for `sampler2D` uniforms - Connect Number/Vector ops for `float`, `vec2`, `vec3`, `vec4` uniforms - Connect Matrix ops for `mat3`, `mat4` uniforms (if supported)

Example Patch Wiring for Keystone Correction:

```
ImageTexture -> TextureEffect (tex port)
CanvasInfo -> TextureEffect (resolution port)
Vector2 (topLeft) -> TextureEffect (topLeft port)
Vector2 (topRight) -> TextureEffect (topRight port)
Vector2 (bottomLeft) -> TextureEffect (bottomLeft port)
Vector2 (bottomRight) -> TextureEffect (bottomRight port)
```

Approach 2: Custom JavaScript Ops (For Advanced Control)

JavaScript custom ops allow you to: - Wrap shader code with additional logic - Dynamically modify shader uniforms - Create reusable, parameterized shader ops - Add custom UI and port organization - Handle complex texture management

Trade-offs: - More setup required (JavaScript wrapper code) - Potential JavaScript overhead - More control over execution flow - Better for reusable, packaged ops

See the "JavaScript Custom Op Examples" section below for implementation details.

7.19.5 Geometric Distortion Correction

Geometric distortion occurs when projectors are not perpendicular to the projection surface. Common types include keystone distortion, barrel distortion, and pincushion distortion.

Keystone Correction (Perspective Distortion)

Built-in Shader Op Ready - Paste into TextureEffect

Keystone distortion creates a trapezoidal shape. This shader corrects it by applying inverse perspective transformation:

```
precision mediump float;

varying vec2 vUV;
uniform sampler2D tex;
uniform vec2 resolution;

```

```
// Keystone correction parameters
// topLeft, topRight, bottomLeft, bottomRight corners in UV space (0-1)
uniform vec2 topLeft;
uniform vec2 topRight;
uniform vec2 bottomLeft;
uniform vec2 bottomRight;

// Helper function: bilinear interpolation for perspective correction
vec2 perspectiveTransform(vec2 uv, vec2 tl, vec2 tr, vec2 bl, vec2 br) {
    // Convert UV to normalized coordinates (-1 to 1)
    vec2 nuv = (uv - 0.5) * 2.0;

    // Perspective correction using bilinear interpolation
    vec2 top = mix(tl, tr, uv.x);
    vec2 bottom = mix(bl, br, uv.x);
    vec2 corrected = mix(bottom, top, uv.y);

    return corrected;
}

void main() {
   vec2 correctedUV = perspectiveTransform(vUV, topLeft, topRight, bottomLeft, bottomRight);

    // Clamp to prevent sampling outside texture
    correctedUV = clamp(correctedUV, 0.0, 1.0);

    vec3 color = texture2D(tex, correctedUV).rgb;
    gl_FragColor = vec4(color, 1.0);
}
```

Usage with TextureEffect (Built-in Shader Op):

1. Add a `TextureEffect` op to your patch
2. Paste the shader code above into the "Fragment Shader" field
3. Connect your inputs:
    - Input texture -> `tex` port (automatically created)
    - `CanvasInfo` op -> `resolution` port (or use `GetResolution` op)
    - Four `Vector2` ops for corners -> `topLeft`, `topRight`, `bottomLeft`, `bottomRight` ports

4. The output texture will have keystone correction applied

Note: The `resolution` uniform is not automatically provided. You must connect a Resolution or CanvasInfo op to the `resolution` port.

Advanced Keystone with Homography Matrix

For more precise control, use a 3x3 homography matrix:

Note: `mat3` support may vary in cables.gl versions. Verify with Matrix ops or use the corner-based approach above if matrices aren't supported.

```
precision mediump float;

varying vec2 vUV;
uniform sampler2D tex;
uniform mat3 homographyMatrix; // 3x3 transformation matrix - verify Matrix op support in your cables.

vec2 applyHomography(mat3 H, vec2 uv) {
    vec3 p = vec3(uv, 1.0);
    vec3 result = H * p;
    return result.xy / result.z;
}

void main() {
    vec2 correctedUV = applyHomography(homographyMatrix, vUV);

    // Check if point is within bounds
    if (correctedUV.x < 0.0 || correctedUV.x > 1.0 ||
        correctedUV.y < 0.0 || correctedUV.y > 1.0) {
        gl_FragColor = vec4(0.0, 0.0, 0.0, 1.0); // Black outside bounds
    } else {
        vec3 color = texture2D(tex, correctedUV).rgb;
        gl_FragColor = vec4(color, 1.0);
    }
}
```

Barrel Distortion Correction

Built-in Shader Op Ready - Paste into TextureEffect

Barrel distortion creates a “bulging” effect. This shader corrects it:

```
precision mediump float;

varying vec2 vUV;
uniform sampler2D tex;
uniform vec2 resolution;
uniform float barrelStrength; // Typically -0.1 to -0.3 for correction

vec2 barrelDistortion(vec2 uv, float strength) {
    vec2 center = vec2(0.5, 0.5);
    vec2 coord = uv - center;
    float dist = length(coord);

    // Barrel distortion formula
    float factor = 1.0 + strength * dist * dist;
    vec2 corrected = center + coord * factor;

    return corrected;
}

void main() {
    vec2 correctedUV = barrelDistortion(vUV, barrelStrength);

    // Only sample if within bounds
    if (correctedUV.x < 0.0 || correctedUV.x > 1.0 ||
        correctedUV.y < 0.0 || correctedUV.y > 1.0) {
        gl_FragColor = vec4(0.0, 0.0, 0.0, 1.0);
    } else {
        vec3 color = texture2D(tex, correctedUV).rgb;
        gl_FragColor = vec4(color, 1.0);
    }
}
```

Pincushion Distortion Correction

Built-in Shader Op Ready - Paste into TextureEffect

Pincushion distortion creates a “pinched” effect. This shader corrects it:

```
precision mediump float;

varying vec2 vUV;
uniform sampler2D tex;
uniform vec2 resolution;
uniform float pincushionStrength; // Typically 0.1 to 0.3 for correction

vec2 pincushionDistortion(vec2 uv, float strength) {
    vec2 center = vec2(0.5, 0.5);
    vec2 coord = uv - center;
    float dist = length(coord);

    // Pincushion distortion formula (opposite of barrel)
    float factor = 1.0 - strength * dist * dist;
    vec2 corrected = center + coord * factor;

    return corrected;
}

void main() {
    vec2 correctedUV = pincushionDistortion(vUV, pincushionStrength);

    if (correctedUV.x < 0.0 || correctedUV.x > 1.0 ||
        correctedUV.y < 0.0 || correctedUV.y > 1.0) {
        gl_FragColor = vec4(0.0, 0.0, 0.0, 1.0);
    } else {
        vec3 color = texture2D(tex, correctedUV).rgb;
        gl_FragColor = vec4(color, 1.0);
    }
}
```

Combined Geometric Correction

A comprehensive shader combining multiple distortion types:

```
precision mediump float;

varying vec2 vUV;
uniform sampler2D tex;
uniform vec2 resolution;

// Keystone corners
uniform vec2 topLeft;
uniform vec2 topRight;
uniform vec2 bottomLeft;
uniform vec2 bottomRight;

// Distortion parameters
uniform float barrelAmount;
uniform float pincushionAmount;
uniform float rotation; // Rotation in radians

vec2 rotateUV(vec2 uv, float angle) {
    vec2 center = vec2(0.5, 0.5);
    vec2 coord = uv - center;
    float c = cos(angle);
    float s = sin(angle);
    mat2 rot = mat2(c, -s, s, c);
    return center + rot * coord;
}

vec2 applyDistortion(vec2 uv, float barrel, float pincushion) {
    vec2 center = vec2(0.5, 0.5);
    vec2 coord = uv - center;
    float dist = length(coord);

    float factor = 1.0 + (barrel + pincushion) * dist * dist;
    return center + coord * factor;
}

vec2 perspectiveTransform(vec2 uv, vec2 tl, vec2 tr, vec2 bl, vec2 br) {
    vec2 top = mix(tl, tr, uv.x);
    vec2 bottom = mix(bl, br, uv.x);
    return mix(bottom, top, uv.y);
}

void main() {
    vec2 uv = vUV;

    // Apply transformations in order: rotation -> distortion -> keystone
    uv = rotateUV(uv, rotation);
    uv = applyDistortion(uv, barrelAmount, pincushionAmount);
    uv = perspectiveTransform(uv, topLeft, topRight, bottomLeft, bottomRight);

    if (uv.x < 0.0 || uv.x > 1.0 || uv.y < 0.0 || uv.y > 1.0) {
        gl_FragColor = vec4(0.0, 0.0, 0.0, 1.0);
    } else {
        vec3 color = texture2D(tex, uv).rgb;
        gl_FragColor = vec4(color, 1.0);
    }
}
```

7.19.6 Multi-Projector Setups

When using multiple projectors, you need to define projection zones and blend overlapping areas.

Projection Zone Mask

Define which projector covers which area:

```
precision mediump float;

varying vec2 vUV;
```

```
uniform sampler2D tex;
uniform vec2 resolution;

// Projection zone definition (in UV space, 0-1)
uniform vec4 zoneRect; // x, y, width, height of this projector's zone
uniform float feather; // Edge feathering amount

float getZoneMask(vec2 uv, vec4 zone) {
    vec2 zoneMin = zone.xy;
    vec2 zoneMax = zone.xy + zone.zw;

    // Distance to zone edges
    vec2 distToMin = uv - zoneMin;
    vec2 distToMax = zoneMax - uv;
    vec2 distToEdge = min(distToMin, distToMax);

    // Create mask with feathering
    float mask = 1.0;
    if (distToEdge.x < feather) {
        mask *= smoothstep(0.0, feather, distToEdge.x);
    }
    if (distToEdge.y < feather) {
        mask *= smoothstep(0.0, feather, distToEdge.y);
    }

    // Check if outside zone
    if (uv.x < zoneMin.x || uv.x > zoneMax.x ||
        uv.y < zoneMin.y || uv.y > zoneMax.y) {
        mask = 0.0;
    }

    return mask;
}

void main() {
    float mask = getZoneMask(vUV, zoneRect);
    vec3 color = texture2D(tex, vUV).rgb;

    gl_FragColor = vec4(color * mask, mask);
}
```

Multi-Projector Blending

Blend multiple projector outputs with smooth transitions:

```
precision mediump float;

varying vec2 vUV;
uniform sampler2D tex;
uniform vec2 resolution;

// Blend zone definition
uniform vec4 blendZone; // x, y, width, height of blend area
uniform float blendWidth; // Width of blend transition
uniform float blendDirection; // 0.0=horizontal, 1.0=vertical, 2.0=both (use float instead of int for

float getBlendMask(vec2 uv, vec4 zone, float width, float direction) {
    vec2 zoneMin = zone.xy;
    vec2 zoneMax = zone.xy + zone.zw;
    vec2 zoneCenter = (zoneMin + zoneMax) * 0.5;

    float mask = 1.0;

    // Use float comparisons instead of int (cables.gl compatibility)
    if (direction < 0.5 || direction > 1.5) {
        // Horizontal blend (direction == 0.0 or 2.0)
        float distToCenter = abs(uv.x - zoneCenter.x);
        float zoneWidth = zone.z;
        if (distToCenter < zoneWidth * 0.5) {
            float blendDist = (zoneWidth * 0.5 - distToCenter) / width;
            mask *= smoothstep(0.0, 1.0, blendDist);
        }
    }

    if (direction > 0.5 && direction < 1.5 || direction > 1.5) {
        // Vertical blend (direction == 1.0 or 2.0)
        float distToCenter = abs(uv.y - zoneCenter.y);
        float zoneHeight = zone.w;
        if (distToCenter < zoneHeight * 0.5) {
            float blendDist = (zoneHeight * 0.5 - distToCenter) / width;
            mask *= smoothstep(0.0, 1.0, blendDist);
        }
    }

    return clamp(mask, 0.0, 1.0);
}

void main() {
    float blendMask = getBlendMask(vUV, blendZone, blendWidth, blendDirection);
    vec3 color = texture2D(tex, vUV).rgb;

    gl_FragColor = vec4(color * blendMask, blendMask);
}
```

7.19.7 Projector Stacking

Projector stacking involves overlapping multiple projectors to increase brightness and redundancy. This shader combines multiple inputs:

```
precision mediump float;

varying vec2 vUV;
uniform sampler2D tex1; // First projector
uniform sampler2D tex2; // Second projector
uniform sampler2D tex3; // Optional third projector
uniform sampler2D tex4; // Optional fourth projector

uniform float stackCount; // Number of active projectors (1-4)
uniform float blendMode; // 0=additive, 1=average, 2=max

vec3 blendStacked(vec3 c1, vec3 c2, vec3 c3, vec3 c4, float count, float mode) {
    vec3 result = vec3(0.0);

    if (mode < 0.5) {
        // Additive blending (brightest, but can clip)
        if (count > 0.5) result += c1;
        if (count > 1.5) result += c2;
        if (count > 2.5) result += c3;
        if (count > 3.5) result += c4;
        result = clamp(result, 0.0, 1.0);
    } else if (mode < 1.5) {
        // Average blending (natural, reduces brightness)
        float sum = 0.0;
        if (count > 0.5) { result += c1; sum += 1.0; }
        if (count > 1.5) { result += c2; sum += 1.0; }
        if (count > 2.5) { result += c3; sum += 1.0; }
        if (count > 3.5) { result += c4; sum += 1.0; }
        result /= max(sum, 1.0);
    } else {
        // Maximum blending (preserves highlights)
        result = c1;
        if (count > 1.5) result = max(result, c2);
        if (count > 2.5) result = max(result, c3);
        if (count > 3.5) result = max(result, c4);
    }

    return result;
}

void main() {
    vec3 c1 = texture2D(tex1, vUV).rgb;
    vec3 c2 = texture2D(tex2, vUV).rgb;
    vec3 c3 = texture2D(tex3, vUV).rgb;
    vec3 c4 = texture2D(tex4, vUV).rgb;
    vec3 result = blendStacked(c1, c2, c3, c4, stackCount, blendMode);

    gl_FragColor = vec4(result, 1.0);
}
```

7.19.8 Gradient Blend Composition

Gradient blends create smooth transitions between overlapping projectors. This is essential for seamless multi-projector setups.

Linear Gradient Blend

```
precision mediump float;

varying vec2 vUV;
uniform sampler2D tex;
uniform vec2 resolution;

// Blend parameters
uniform float blendStart; // Where blend starts (0-1)
uniform float blendEnd; // Where blend ends (0-1)
uniform float blendAxis; // 0.0=horizontal, 1.0=vertical (use float instead of int for cables.gl compa
uniform float blendPower; // Blend curve (1.0=linear, 2.0=smooth)

float getLinearBlend(vec2 uv, float start, float end, float axis, float power) {
    float pos = axis < 0.5 ? uv.x : uv.y; // Use float comparison

    // Calculate blend factor
    float blendFactor = 0.0;
    if (pos < start) {
        blendFactor = 0.0;
    } else if (pos > end) {
        blendFactor = 1.0;
    } else {
        // Normalize to 0-1 range
        float t = (pos - start) / (end - start);
        // Apply power curve
        blendFactor = pow(t, power);
    }

    return blendFactor;
}

void main() {
    float blend = getLinearBlend(vUV, blendStart, blendEnd, blendAxis, blendPower);
    vec3 color = texture2D(tex, vUV).rgb;

    gl_FragColor = vec4(color * blend, blend);
}
```

Radial Gradient Blend

For circular or elliptical blend zones:

```
precision mediump float;

varying vec2 vUV;
uniform sampler2D tex;
uniform vec2 resolution;

// Radial blend parameters
uniform vec2 center; // Blend center in UV space
uniform float innerRadius; // Inner radius (full opacity)
uniform float outerRadius; // Outer radius (zero opacity)
uniform float aspectRatio; // Aspect ratio correction
uniform float blendPower; // Blend curve

float getRadialBlend(vec2 uv, vec2 center, float innerR, float outerR, float aspect, float power) {
    vec2 offset = (uv - center) * vec2(aspect, 1.0);
    float dist = length(offset);

    float blendFactor = 0.0;
    if (dist < innerR) {
        blendFactor = 1.0;
    } else if (dist > outerR) {
        blendFactor = 0.0;
    } else {
        float t = (dist - innerR) / (outerR - innerR);
        blendFactor = 1.0 - pow(t, power);
    }

    return blendFactor;
}

void main() {
   float blend = getRadialBlend(vUV, center, innerRadius, outerRadius, aspectRatio, blendPower);
    vec3 color = texture2D(tex, vUV).rgb;

    gl_FragColor = vec4(color * blend, blend);
}
```

Advanced Feather Blend with Soft Edges

Professional-grade blend with multiple falloff curves:

```
precision mediump float;

varying vec2 vUV;
uniform sampler2D tex;
uniform vec2 resolution;

uniform vec4 blendRect; // x, y, width, height
uniform float featherSize; // Feather size in UV units
uniform float featherCurve; // 0.0=linear, 1.0=smooth, 2.0=very smooth

float getFeatherBlend(vec2 uv, vec4 rect, float feather, float curve) {
    vec2 rectMin = rect.xy;
    vec2 rectMax = rect.xy + rect.zw;
    // Calculate distance to each edge
    float distLeft = uv.x - rectMin.x;
    float distRight = rectMax.x - uv.x;
    float distBottom = uv.y - rectMin.y;
    float distTop = rectMax.y - uv.y;

    // Find minimum distance to any edge
    float minDist = min(min(distLeft, distRight), min(distBottom, distTop));

    // Create feather mask
    float mask = 1.0;
    if (minDist < feather) {
        float t = minDist / feather;
        // Apply curve
        if (curve < 0.5) {
            // Linear
            mask = t;
        } else if (curve < 1.5) {
            // Smoothstep
            mask = smoothstep(0.0, 1.0, t);
        } else {
            // Custom smooth curve
            mask = t * t * (3.0 - 2.0 * t);
            mask = pow(mask, 1.0 / (curve - 0.5));
        }
    }

    // Check if outside rectangle
    if (uv.x < rectMin.x || uv.x > rectMax.x ||
        uv.y < rectMin.y || uv.y > rectMax.y) {
        mask = 0.0;
    }

    return mask;
}
```

```
void main() {
    float blend = getFeatherBlend(vUV, blendRect, featherSize, featherCurve);
    vec3 color = texture2D(tex, vUV).rgb;

    gl_FragColor = vec4(color * blend, blend);
}
```

7.19.9 Color Correction for Projection Mapping

Projection mapping requires specialized color correction to account for surface colors, ambient light, and projector characteristics.

Basic Color Correction

Built-in Shader Op Ready - Paste into TextureEffect

```
precision mediump float;

varying vec2 vUV;
uniform sampler2D tex;
uniform vec2 resolution;

// Color correction parameters
uniform float brightness; // -1.0 to 1.0
uniform float contrast; // -1.0 to 1.0
uniform float saturation; // -1.0 to 1.0
uniform float gamma; // Typically 0.5 to 3.0

vec3 applyColorCorrection(vec3 color, float bright, float cont, float sat, float gam) {
    // Brightness
    color += bright;

    // Contrast
    color = (color - 0.5) * (1.0 + cont) + 0.5;

    // Saturation
    float luma = dot(color, vec3(0.299, 0.587, 0.114));
    color = mix(vec3(luma), color, 1.0 + sat);

    // Gamma
    color = pow(max(color, 0.0), vec3(1.0 / max(gam, 0.01)));

    return clamp(color, 0.0, 1.0);
}

void main() {
    vec3 color = texture2D(tex, vUV).rgb;
    color = applyColorCorrection(color, brightness, contrast, saturation, gamma);

    gl_FragColor = vec4(color, 1.0);
}
```

Advanced Color Correction with Color Temperature

```
precision mediump float;

varying vec2 vUV;
uniform sampler2D tex;
uniform vec2 resolution;

uniform float brightness;
uniform float contrast;
uniform float saturation;
uniform float gamma;
uniform float colorTemperature; // -1.0 (cool/blue) to 1.0 (warm/orange)

// Color temperature adjustment
vec3 adjustColorTemperature(vec3 color, float temp) {
    // Convert to warmer (orange) or cooler (blue)
    if (temp > 0.0) {
        // Warmer: increase red/orange, decrease blue
        color.r += temp * 0.2;
        color.b -= temp * 0.1;
    } else {
        // Cooler: increase blue, decrease red
        color.r += temp * 0.1;
        color.b -= temp * 0.2;
    }
    return color;
}

vec3 applyColorCorrection(vec3 color, float bright, float cont, float sat, float gam, float temp) {
    // Brightness
    color += bright;

    // Contrast
    color = (color - 0.5) * (1.0 + cont) + 0.5;

    // Saturation
    float luma = dot(color, vec3(0.299, 0.587, 0.114));
    color = mix(vec3(luma), color, 1.0 + sat);

    // Color temperature
    color = adjustColorTemperature(color, temp);

    // Gamma
    color = pow(max(color, 0.0), vec3(1.0 / max(gam, 0.01)));

    return clamp(color, 0.0, 1.0);
}

void main() {
    vec3 color = texture2D(tex, vUV).rgb;
   color = applyColorCorrection(color, brightness, contrast, saturation, gamma, colorTemperature);

    gl_FragColor = vec4(color, 1.0);
}
```

Per-Channel Color Correction

Independent control over RGB channels:

```
precision mediump float;

varying vec2 vUV;
uniform sampler2D tex;
uniform vec2 resolution;

// Per-channel brightness and contrast
uniform vec3 channelBrightness; // R, G, B
uniform vec3 channelContrast; // R, G, B
uniform vec3 channelGamma; // R, G, B

vec3 applyPerChannelCorrection(vec3 color, vec3 bright, vec3 cont, vec3 gam) {
    // Apply per-channel brightness
    color += bright;

    // Apply per-channel contrast
    color = (color - 0.5) * (1.0 + cont) + 0.5;

    // Apply per-channel gamma
    color = pow(max(color, 0.0), vec3(1.0 / max(gam, vec3(0.01))));

    return clamp(color, 0.0, 1.0);
}

void main() {
    vec3 color = texture2D(tex, vUV).rgb;
   color = applyPerChannelCorrection(color, channelBrightness, channelContrast, channelGamma);

    gl_FragColor = vec4(color, 1.0);
}
```

Surface Color Compensation

Compensate for colored projection surfaces (e.g., projecting on a red wall):

```glsl
precision mediump float;

varying vec2 vUV;
uniform sampler2D tex;
uniform vec2 resolution;

// Surface color (what color the surface appears)
uniform vec3 surfaceColor;
uniform float compensationStrength; // 0.0 to 1.0

vec3 compensateSurfaceColor(vec3 color, vec3 surface, float strength) {
    // Calculate inverse of surface color
    vec3 inverseSurface = vec3(1.0) - surface;

    // Blend between original and compensated
    vec3 compensated = color / max(surface, vec3(0.01)); // Prevent division by zero
    compensated = clamp(compensated, 0.0, 1.0);

    return mix(color, compensated, strength);
}

void main() {
    vec3 color = texture2D(tex, vUV).rgb;
    color = compensateSurfaceColor(color, surfaceColor, compensationStrength);

    gl_FragColor = vec4(color, 1.0);
}
```

Advanced LUT-Based Color Correction

Use a 3D Look-Up Table (LUT) for professional color grading:

```
precision mediump float;

varying vec2 vUV;
uniform sampler2D tex;
uniform sampler2D lutTexture; // 3D LUT as 2D texture (typically 64x64 or 32x32)
uniform vec2 resolution;
uniform float lutStrength; // 0.0 to 1.0

// Sample 3D LUT (stored as 2D texture)
vec3 sampleLUT(sampler2D lut, vec3 color, float lutSize) {
    // Assume LUT is organized as a grid
    // For a 64x64 LUT, we have 8x8 grid of 8x8 color cubes

    float cellSize = 1.0 / 8.0; // 8x8 grid
    float cellPixelSize = 1.0 / 64.0; // 64 pixels per cell

    // Find which cell we're in
    vec3 cell = floor(color * 7.0);
    vec3 cellPos = fract(color * 7.0);

    // Calculate UV coordinates in LUT texture
    float cellIndex = cell.b * 8.0 + cell.r;
    vec2 lutUV = vec2(
        (cellIndex * cellSize) + (cellPos.r * cellPixelSize * 8.0),
        cell.g * cellSize + cellPos.g * cellPixelSize * 8.0
    );

    // Sample LUT
    vec3 lutColor = texture2D(lut, lutUV).rgb;

    return lutColor;
}

void main() {
    vec3 originalColor = texture2D(tex, vUV).rgb;
    vec3 lutColor = sampleLUT(lutTexture, originalColor, 64.0);
    vec3 finalColor = mix(originalColor, lutColor, lutStrength);

    gl_FragColor = vec4(finalColor, 1.0);
}
```

Note: For LUT textures, you'll need to create or load a 3D LUT texture. Common formats include 64x64 (8x8 grid) or 32x32 (4x4 grid) textures.

Shadow and Highlight Recovery

Recover details in shadows and highlights:

```
precision mediump float;

varying vec2 vUV;
uniform sampler2D tex;
uniform vec2 resolution;

uniform float shadowRecovery; // 0.0 to 1.0
uniform float highlightRecovery; // 0.0 to 1.0
uniform float shadowPoint; // Where shadows start (0.0 to 1.0)
uniform float highlightPoint; // Where highlights start (0.0 to 1.0)

vec3 recoverShadowsHighlights(vec3 color, float shadowRec, float highlightRec, float shadowPt, float
    float luma = dot(color, vec3(0.299, 0.587, 0.114));

    // Shadow recovery
    float shadowMask = smoothstep(shadowPt - 0.1, shadowPt, luma);
    color += shadowMask * shadowRec * (1.0 - luma) * 0.5;

    // Highlight recovery (compress highlights)
    float highlightMask = smoothstep(highlightPt, highlightPt + 0.1, luma);
    color = mix(color, vec3(1.0) - (vec3(1.0) - color) * (1.0 - highlightRec), highlightMask);

    return clamp(color, 0.0, 1.0);
}
```

```
void main() {
    vec3 color = texture2D(tex, vUV).rgb;
   color = recoverShadowsHighlights(color, shadowRecovery, highlightRecovery, shadowPoint, highlight

    gl_FragColor = vec4(color, 1.0);
}
```

7.19.10 Complete Projection Mapping Pipeline

Built-in Shader Op Ready - Paste into TextureEffect (Note: This is a complex shader with many uniforms - consider breaking into multiple passes for easier management)

A comprehensive shader combining all projection mapping features:

```
precision mediump float;

varying vec2 vUV;
uniform sampler2D tex;
uniform vec2 resolution;

// Geometric correction
uniform vec2 topLeft;
uniform vec2 topRight;
uniform vec2 bottomLeft;
uniform vec2 bottomRight;
uniform float barrelAmount;
uniform float rotation;

// Blend parameters
uniform vec4 blendZone;
uniform float blendWidth;
uniform float blendPower;

// Color correction
uniform float brightness;
uniform float contrast;
uniform float saturation;
uniform float gamma;
uniform float colorTemperature;
uniform vec3 surfaceColor;
uniform float surfaceCompensation;

// Helper functions (include all from above)
vec2 perspectiveTransform(vec2 uv, vec2 tl, vec2 tr, vec2 bl, vec2 br) {
    vec2 top = mix(tl, tr, uv.x);
    vec2 bottom = mix(bl, br, uv.x);
    return mix(bottom, top, uv.y);
}

vec2 applyDistortion(vec2 uv, float barrel) {
    vec2 center = vec2(0.5, 0.5);
    vec2 coord = uv - center;
    float dist = length(coord);
    float factor = 1.0 + barrel * dist * dist;
    return center + coord * factor;
}

vec2 rotateUV(vec2 uv, float angle) {
    vec2 center = vec2(0.5, 0.5);
    vec2 coord = uv - center;
    float c = cos(angle);
    float s = sin(angle);
    mat2 rot = mat2(c, -s, s, c);
    return center + rot * coord;
}

float getBlendMask(vec2 uv, vec4 zone, float width, float power) {
    vec2 zoneMin = zone.xy;
    vec2 zoneMax = zone.xy + zone.zw;
    vec2 zoneCenter = (zoneMin + zoneMax) * 0.5;

    float distToCenter = length(uv - zoneCenter);
    float maxDist = length(zoneMax - zoneCenter);

    if (distToCenter > maxDist) return 0.0;

    float blendDist = (maxDist - distToCenter) / width;
    return pow(clamp(blendDist, 0.0, 1.0), power);
}

vec3 applyColorCorrection(vec3 color, float bright, float cont, float sat, float gam, float temp, vec3
    color += bright;
    color = (color - 0.5) * (1.0 + cont) + 0.5;

    float luma = dot(color, vec3(0.299, 0.587, 0.114));
    color = mix(vec3(luma), color, 1.0 + sat);

    if (temp > 0.0) {
        color.r += temp * 0.2;
        color.b -= temp * 0.1;
    } else {
        color.r += temp * 0.1;
        color.b -= temp * 0.2;
    }

    vec3 compensated = color / max(surface, vec3(0.01));
    color = mix(color, clamp(compensated, 0.0, 1.0), comp);

    color = pow(max(color, 0.0), vec3(1.0 / max(gam, 0.01)));

    return clamp(color, 0.0, 1.0);
}

void main() {
    // Step 1: Geometric correction
    vec2 uv = vUV;
    uv = rotateUV(uv, rotation);
    uv = applyDistortion(uv, barrelAmount);
    uv = perspectiveTransform(uv, topLeft, topRight, bottomLeft, bottomRight);

    // Step 2: Sample texture
    if (uv.x < 0.0 || uv.x > 1.0 || uv.y < 0.0 || uv.y > 1.0) {
        gl_FragColor = vec4(0.0, 0.0, 0.0, 0.0);
        return;
    }

    vec3 color = texture2D(tex, uv).rgb;

    // Step 3: Color correction
    color = applyColorCorrection(color, brightness, contrast, saturation, gamma, colorTemperature, sur

    // Step 4: Apply blend mask
    float blend = getBlendMask(vUV, blendZone, blendWidth, blendPower);
    color *= blend;

    gl_FragColor = vec4(color, blend);
}
```

7.19.11 JavaScript Custom Op Examples

For cases where you need more control, reusable components, or dynamic shader management, you can wrap shaders in JavaScript custom ops. Here are examples for key projection mapping features:

Keystone Correction Custom Op

```
// Custom Op: KeystoneCorrection
// Name: Ops.User.ProjectionMapping.KeystoneCorrection

const inTexture = op.inTexture("Input Texture");
const inTopLeft = op.inVec2("Top Left", [0.0, 1.0]);
const inTopRight = op.inVec2("Top Right", [1.0, 1.0]);
const inBottomLeft = op.inVec2("Bottom Left", [0.0, 0.0]);
const inBottomRight = op.inVec2("Bottom Right", [1.0, 0.0]);
const inResolution = op.inVec2("Resolution", [1920.0, 1080.0]);
const outTexture = op.outTexture("Output");

// Shader code as string
const shaderCode = `
precision mediump float;
varying vec2 vUV;
uniform sampler2D tex;
uniform vec2 resolution;
uniform vec2 topLeft;
uniform vec2 topRight;
uniform vec2 bottomLeft;
uniform vec2 bottomRight;

vec2 perspectiveTransform(vec2 uv, vec2 tl, vec2 tr, vec2 bl, vec2 br) {
    vec2 top = mix(tl, tr, uv.x);
    vec2 bottom = mix(bl, br, uv.x);
    return mix(bottom, top, uv.y);
}

void main() {
   vec2 correctedUV = perspectiveTransform(vUV, topLeft, topRight, bottomLeft, bottomRight);
    correctedUV = clamp(correctedUV, 0.0, 1.0);
    vec3 color = texture2D(tex, correctedUV).rgb;
    gl_FragColor = vec4(color, 1.0);
}
`;

let shaderMaterial = null;

function updateShader() {
    const tex = inTexture.get();
    if (!tex) return;

    // Create or update shader material
    if (!shaderMaterial) {
        shaderMaterial = new op.patch.cgl.ShaderMaterial({
            fragmentShader: shaderCode,
            uniforms: {}
        });
    }

    // Update uniforms
    shaderMaterial.uniforms.tex = { value: tex };
    shaderMaterial.uniforms.resolution = { value: inResolution.get() };
    shaderMaterial.uniforms.topLeft = { value: inTopLeft.get() };
    shaderMaterial.uniforms.topRight = { value: inTopRight.get() };
    shaderMaterial.uniforms.bottomLeft = { value: inBottomLeft.get() };
    shaderMaterial.uniforms.bottomRight = { value: inBottomRight.get() };

    // Render to texture
    const renderTarget = op.patch.cgl.createRenderTarget(
        inResolution.get()[0],
        inResolution.get()[1]
    );

    // Apply shader and render
    op.patch.cgl.render(renderTarget, shaderMaterial);

    outTexture.set(renderTarget.texture);
}

inTexture.onChange = updateShader;
inTopLeft.onChange = updateShader;
inTopRight.onChange = updateShader;
inBottomLeft.onChange = updateShader;
inBottomRight.onChange = updateShader;
inResolution.onChange = updateShader;
```

Note: The above example shows the concept, but cables.gl's actual API may differ. In practice, you might use `TextureEffect` programmatically or create a render pass.

Color Correction Custom Op

```
// Custom Op: ColorCorrection
// Name: Ops.User.ProjectionMapping.ColorCorrection

const inTexture = op.inTexture("Input Texture");
const inBrightness = op.inFloat("Brightness", 0.0);
const inContrast = op.inFloat("Contrast", 0.0);
const inSaturation = op.inFloat("Saturation", 0.0);
const inGamma = op.inFloat("Gamma", 1.0);
const inColorTemperature = op.inFloat("Color Temperature", 0.0);
const outTexture = op.outTexture("Output");

const shaderCode = `
precision mediump float;
varying vec2 vUV;
uniform sampler2D tex;
uniform float brightness;
uniform float contrast;
uniform float saturation;
uniform float gamma;
uniform float colorTemperature;

vec3 adjustColorTemperature(vec3 color, float temp) {
    if (temp > 0.0) {
        color.r += temp * 0.2;
        color.b -= temp * 0.1;
    } else {
        color.r += temp * 0.1;
        color.b -= temp * 0.2;
    }
    return color;
}

vec3 applyColorCorrection(vec3 color, float bright, float cont, float sat, float gam, float temp) {
    color += bright;
    color = (color - 0.5) * (1.0 + cont) + 0.5;
    float luma = dot(color, vec3(0.299, 0.587, 0.114));
    color = mix(vec3(luma), color, 1.0 + sat);

    color = adjustColorTemperature(color, temp);
    color = pow(max(color, 0.0), vec3(1.0 / max(gam, 0.01)));

    return clamp(color, 0.0, 1.0);
}

void main() {
    vec3 color = texture2D(tex, vUV).rgb;
    color = applyColorCorrection(color, brightness, contrast, saturation, gamma, colorTemperature);
    gl_FragColor = vec4(color, 1.0);
}
`;

// Implementation similar to keystone op above
// (Actual implementation depends on cables.gl's rendering API)
```

Blend Composition Custom Op

```
// Custom Op: BlendComposition
// Name: Ops.User.ProjectionMapping.BlendComposition

const inTexture = op.inTexture("Input Texture");
const inBlendStart = op.inFloat("Blend Start", 0.0);
const inBlendEnd = op.inFloat("Blend End", 1.0);
const inBlendAxis = op.inFloat("Blend Axis", 0.0); // 0.0=horizontal, 1.0=vertical
const inBlendPower = op.inFloat("Blend Power", 1.0);
const inResolution = op.inVec2("Resolution", [1920.0, 1080.0]);
const outTexture = op.outTexture("Output");
const outAlpha = op.outNumber("Alpha Mask"); // For compositing

const shaderCode = `
precision mediump float;
varying vec2 vUV;
```

```
uniform sampler2D tex;
uniform vec2 resolution;
uniform float blendStart;
uniform float blendEnd;
uniform float blendAxis;
uniform float blendPower;

float getLinearBlend(vec2 uv, float start, float end, float axis, float power) {
    float pos = axis < 0.5 ? uv.x : uv.y;
    float blendFactor = 0.0;

    if (pos < start) {
        blendFactor = 0.0;
    } else if (pos > end) {
        blendFactor = 1.0;
    } else {
        float t = (pos - start) / (end - start);
        blendFactor = pow(t, power);
    }

    return blendFactor;
}

void main() {
    float blend = getLinearBlend(vUV, blendStart, blendEnd, blendAxis, blendPower);
    vec3 color = texture2D(tex, vUV).rgb;
    gl_FragColor = vec4(color * blend, blend);
}
`;

// Implementation with uniform updates
// Note: This is a conceptual example - actual cables.gl API may vary
```

Important Notes for JavaScript Custom Ops:

1. **Texture Handling**: You need to manage texture creation, rendering, and cleanup
2. **Render Targets**: May need to create render targets for shader output
3. **Performance**: JavaScript overhead can impact real-time performance

4. **API Differences**: Cables.gl's internal API may differ from these examples
5. **Best Practice**: Use built-in `TextureEffect` when possible; use custom ops for complex logic or reusable components

7.19.12 Comparison: Built-in Shader Ops vs Custom JavaScript Ops

Code Cleanliness

Built-in Shader Ops (TextureEffect/ShaderMaterial): - Pure GLSL code - no wrapper needed - Minimal boilerplate - Easy to read and maintain - Direct shader editing in cables.gl UI - No JavaScript knowledge required

Custom JavaScript Ops: - [!] Requires JavaScript wrapper code - [!] Shader code stored as string (less readable) - [!] More complex file structure - [!] Requires understanding of both GLSL and JavaScript - Can organize shader code in separate files - Can add pre/post processing logic

Winner: Built-in Shader Ops - cleaner, more maintainable for pure shader effects

Integration Ease

Built-in Shader Ops: - Paste shader code directly into TextureEffect - Ports created automatically from uniforms - Immediate visual feedback - No compilation step - Works out of the box - [!] Limited customization of port UI - [!] Can't add custom logic around shader

Custom JavaScript Ops: - [!] Must create op, write wrapper code - [!] Must manually create and configure ports - [!] More setup time - [!] Requires testing and debugging - Full control over port organization - Can add port groups, custom UI - Can add validation, error handling - Reusable across patches

Winner: Built-in Shader Ops - significantly easier to get started

Performance

Built-in Shader Ops: - Direct GPU execution - Minimal overhead - Optimized by cables.gl - No JavaScript execution per frame - Efficient texture passing - Automatic shader compilation caching

Custom JavaScript Ops: - [!] Potential JavaScript overhead per frame - [!] Texture copying may be required - [!] Render target management overhead - [!] Uniform updates in JavaScript (CPU work) - Can optimize with dirty flags - Can batch operations - Can cache render targets

Performance Comparison: - Built-in: ~0.1-0.5ms overhead (shader execution only) - Custom: ~1-5ms overhead (JavaScript + shader execution) - **Winner:** Built-in Shader Ops - better performance for real-time applications

When to Use Each Approach

Use Built-in Shader Ops (TextureEffect/ShaderMaterial) when: - You have pure shader effects (no complex logic) - You want quick prototyping - Performance is critical - You're learning shaders - You need immediate visual feedback - You don't need custom port organization

Use Custom JavaScript Ops when: - You need reusable, packaged shader components - You need complex pre/post processing logic - You need dynamic shader generation - You want custom port UI and organization - You're building a library of shader ops - You need conditional shader selection - You need to manage multiple render passes

Hybrid Approach: - Use built-in shader ops for individual effects - Use custom JavaScript ops to orchestrate multiple shader passes - Use custom ops for complex parameter management - Use built-in ops for simple, one-off effects

7.19.13 Quick Reference: Using These Shaders

Step-by-Step Guide:

1. **Add TextureEffect Op:**
   - Click "+" in your patch
   - Search for "TextureEffect"
   - Add it to your patch
2. **Paste Shader Code:**
   - Click on the TextureEffect op
   - Find the "Fragment Shader" field
   - Paste the shader code (including `precision mediump float;` and `varying vec2 vUV;`)
3. **Connect Inputs:**
   - Input texture -> `tex` port (or `tex0`, `tex1`, etc. for multi-texture shaders)
   - `CanvasInfo` or `GetResolution` -> `resolution` port (if shader uses it)
   - Number/Vector ops -> parameter ports (brightness, contrast, corners, etc.)
4. **Get Output:**
   - Connect TextureEffect output to your render target or next effect

Common Port Types: - `sampler2D tex` -> Texture port (connect ImageTexture, VideoTexture, etc.) - `vec2 resolution` -> Vec2 port (connect CanvasInfo or GetResolution) - `float brightness`

-> Number port (connect Number op or slider) - `vec2 topLeft` -> Vec2 port (connect Vector2 op) - `vec3 color` -> Vec3 port (connect Vector3 op or Color op)

7.19.14 Best Practices for Projection Mapping in Cables.gl

1. **Resolution Handling**: Always use `resolution` uniform for pixel-perfect calculations. Convert between UV space and screen space as needed. **Remember:** resolution is NOT auto-provided - connect it manually.
2. **Performance**: Projection mapping shaders can be expensive. Consider:
   - Using lower precision where possible (`mediump` instead of `highp`)
   - Minimizing texture samples
   - Pre-computing values in JavaScript ops when possible
3. **Modular Approach**: Break complex setups into multiple shader passes:
   - First pass: Geometric correction
   - Second pass: Color correction
   - Third pass: Blending
4. **Testing**: Always test with actual projection surfaces when possible. Screen simulation can differ from real-world results.
5. **Calibration**: Use test patterns (grids, color bars) to calibrate geometric and color corrections.
6. **Masking**: Use alpha channel output for blend masks to composite multiple projectors correctly.

7.19.15 Debug Visualization Shaders

Helpful shaders for debugging projection mapping setups:

Grid Overlay

```
precision mediump float;

varying vec2 vUV;
uniform vec2 resolution;
uniform float gridSize; // Grid divisions
uniform vec3 gridColor;
uniform float gridOpacity;

void main() {
    vec2 gridUV = vUV * gridSize;
   // Use manual derivative calculation instead of fwidth() for better WebGL 1.0 compatibility
    vec2 grid = abs(fract(gridUV - 0.5) - 0.5);
    // Approximate derivative using step function
   float line = min(grid.x, grid.y) * gridSize * 100.0; // Scale factor for visibility
    float gridMask = 1.0 - min(line, 1.0);

    vec3 color = mix(vec3(0.0), gridColor, gridMask * gridOpacity);
    gl_FragColor = vec4(color, 1.0);
}
```

Corner Pin Visualization

```
precision mediump float;

varying vec2 vUV;
uniform vec2 resolution;
uniform vec2 topLeft;
uniform vec2 topRight;
uniform vec2 bottomLeft;
uniform vec2 bottomRight;
uniform vec3 cornerColor;

void main() {
    vec3 color = vec3(0.0);

    // Draw corner points
    float cornerSize = 0.02;
    float dist1 = length(vUV - topLeft);
    float dist2 = length(vUV - topRight);
    float dist3 = length(vUV - bottomLeft);
    float dist4 = length(vUV - bottomRight);

    float minDist = min(min(dist1, dist2), min(dist3, dist4));
    if (minDist < cornerSize) {
        color = cornerColor;
    }

    // Draw lines between corners
    // (Simplified - you'd use line SDF for proper lines)

    gl_FragColor = vec4(color, 1.0);
}
```

7.19.16 Summary: Shader Compliance and Usage

All shaders in this projection mapping section are:

Compliant with cables.gl's built-in shader ops (TextureEffect/ShaderMaterial) **Ready to paste directly** into the fragment shader field **WebGL 1.0 compatible** (using `texture2D()`, `mediump` precision) **Properly formatted** with required headers and declarations **Uniform types verified** (float instead of int, proper vector types)

Key Compliance Features: - All shaders start with `precision mediump float;` - All use `texture2D()` for texture sampling - All use `varying vec2 vUV` (auto-provided by cables.gl) - Integer uniforms converted to float with float comparisons - Resolution handling documented (requires manual connection) - Matrix uniforms noted with version compatibility warnings

Usage Pattern: 1. Copy shader code 2. Paste into TextureEffect op's fragment shader field 3. Connect inputs to automatically created ports 4. Get output texture

For Advanced Use Cases: - See "JavaScript Custom Op Examples" section for wrapper implementations - See "Comparison" section for when to use each approach - See "Troubleshooting" section for common issues

7.20 Featured Videos

```
https://youtu.be/Zfhn8xSM0SE
Title: Coding with cables - custom shader op
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/Zfhn8xSM0SE/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/j_ins4RW0c8
Title: Shadertoy to cables - part  01
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/j_ins4RW0c8/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/nil-HkZgNZ8
Title: Programmation d'un shadertoy avec Cables.gl Partie 8.
Author: Meletou1
Thumbnail: https://i.ytimg.com/vi/nil-HkZgNZ8/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@Meletou1
```

7.21 Resources

- The Book of Shaders - Excellent GLSL learning resource
- Shadertoy - Shader examples and inspiration
- GLSL Sandbox - More shader experiments

7.22 Exercises

1. Create a animated gradient that shifts colors over time
2. Build a kaleidoscope effect using UV manipulation
3. Write an SDF shader that draws a morphing shape
4. Create a post-processing glow effect
5. **Projection Mapping**: Implement keystone correction for a trapezoidal projection
6. **Projection Mapping**: Create a multi-projector blend setup with gradient transitions
7. **Projection Mapping**: Build a color correction shader that compensates for a colored projection surface
8. **Projection Mapping**: Combine geometric correction, color correction, and blending in a single shader pipeline
9. **Projection Mapping**: Create a debug visualization shader showing projection zones and blend areas
10. **Projection Mapping**: Implement projector stacking with additive and average blend modes

8 JavaScript & Custom Ops in Cables.gl

8.1 Introduction

While cables.gl's visual node system is powerful, sometimes you need custom functionality. JavaScript allows you to create your own operators (ops) and extend cables.gl's capabilities.

8.2 When to Use Custom Ops

- Processing data in ways built-in ops don't support
- Integrating external APIs or libraries
- Creating reusable custom functionality
- Performance optimization for specific tasks
- Complex mathematical operations

8.3 Creating Your First Op

8.3.1 Step 1: Open the Op Editor

1. In your patch, click the "+" button
2. Select "Create Op"
3. Choose a name (e.g., `Ops.User.YourName.MyFirstOp`)

8.3.2 Step 2: Understanding the Structure

```
// Ports (inputs and outputs)
const inValue = op.inFloat("Input Value", 0);
const outResult = op.outNumber("Result");

// When input changes, recalculate
inValue.onChange = function() {
    outResult.set(inValue.get() * 2);
};
```

8.4 Port Types

8.4.1 Input Ports

```
// Trigger (execution flow)
const inTrigger = op.inTrigger("Trigger");

// Numbers
const inFloat = op.inFloat("Float Value", 0.0);
const inInt = op.inInt("Integer", 0);
const inValue = op.inValue("Value", 0);

// Boolean
const inBool = op.inBool("Enabled", true);

// String
const inString = op.inString("Text", "default");

// Objects (textures, arrays, etc.)
const inObject = op.inObject("Object");
const inTexture = op.inTexture("Texture");
const inArray = op.inArray("Array");
```

8.4.2 Output Ports

```
// Trigger
const outTrigger = op.outTrigger("Trigger Out");

// Numbers
const outNumber = op.outNumber("Number Out");
const outValue = op.outValue("Value Out");

// Boolean
const outBool = op.outBool("Bool Out");

// String
```

```
const outString = op.outString("String Out");

// Objects
const outObject = op.outObject("Object Out");
const outTexture = op.outTexture("Texture Out");
const outArray = op.outArray("Array Out");
```

8.5 Handling Events

8.5.1 Trigger Execution

```
const inTrigger = op.inTrigger("Execute");
const outNext = op.outTrigger("Next");

inTrigger.onTriggered = function() {
    // Do something when triggered
    console.log("Op was triggered!");

    // Continue the chain
    outNext.trigger();
};
```

8.5.2 Value Changes

```
const inValue = op.inFloat("Value", 0);
const outDouble = op.outNumber("Double");

inValue.onChange = function() {
    const val = inValue.get();
    outDouble.set(val * 2);
};
```

8.5.3 Linking Ports

```
// Automatically update output when input changes
const inValue = op.inFloat("Value", 0);
const outValue = op.outNumber("Value Out");

inValue.onChange = outValue.setRef.bind(outValue, inValue);
// or simply:
// inValue.onChange = () => outValue.set(inValue.get());
```

8.6 Working with Arrays

```
const inArray = op.inArray("Input Array");
const outArray = op.outArray("Output Array");

inArray.onChange = function() {
    const arr = inArray.get();
    if (!arr) return;

    // Process array
    const result = arr.map(x => x * 2);

    outArray.set(result);
};
```

8.7 Working with Objects

```
const inObject = op.inObject("Input");
const outObject = op.outObject("Output");

inObject.onChange = function() {
    const obj = inObject.get();
    if (!obj) return;

    // Process or wrap the object
    const processed = {
        ...obj,
        processed: true
    };
    outObject.set(processed);
};
```

8.8 Render Loop Integration

For ops that need to run every frame:

```
const inTrigger = op.inTrigger("Render");
const outNext = op.outTrigger("Next");

let time = 0;

inTrigger.onTriggered = function() {
    time += op.patch.timer.getDelta();

    // Do per-frame calculations

    outNext.trigger();
};
```

8.9 UI Port Groups

Organize your ports into collapsible groups:

```
// Create ports
const inX = op.inFloat("X", 0);
const inY = op.inFloat("Y", 0);
const inZ = op.inFloat("Z", 0);

// Group them
op.setPortGroup("Position", [inX, inY, inZ]);
```

8.10 Port UI Types

Change how ports appear in the UI:

```
// Slider
const inValue = op.inFloat("Value", 0.5);
op.setUiAttrib({ "type": "slider", "min": 0, "max": 1 });

// Color picker
const inR = op.inFloat("R", 1);
const inG = op.inFloat("G", 1);
const inB = op.inFloat("B", 1);
op.setPortGroup("Color", [inR, inG, inB]);
inR.setUiAttribs({ colorPick: true });

// Dropdown
const inMode = op.inSwitch("Mode", ["Option1", "Option2", "Option3"], "Option1");
```

8.11 Accessing Patch Resources

8.11.1 Timer and Time

```
// Current time
const time = op.patch.timer.getTime();

// Delta time (time since last frame)
const delta = op.patch.timer.getDelta();

// FPS
const fps = op.patch.timer.getFPS();
```

8.11.2 Canvas and Context

```
// Canvas element
const canvas = op.patch.cgl.canvas;

// WebGL context
const gl = op.patch.cgl.gl;
```

8.11.3 Loading External Resources

```
const inUrl = op.inString("URL", "");
const outData = op.outObject("Data");

inUrl.onChange = function() {
    const url = inUrl.get();
    if (!url) return;

    fetch(url)
        .then(response => response.json())
        .then(data => {
            outData.set(data);
        })
        .catch(error => {
            op.logError("Failed to load:", error);
        });
};
```

8.12 Using External Libraries

8.12.1 Including Libraries

```
// In op's code, load an external script
const script = document.createElement("script");
script.src = "https://cdn.example.com/library.js";
script.onload = function() {
    // Library is ready
    initLibrary();
};
document.head.appendChild(script);
```

8.12.2 Or use op.patch.loading for proper load tracking:

```
op.patch.loading.start();
```

```
const script = document.createElement("script");
script.src = "https://cdn.example.com/library.js";
script.onload = function() {
    op.patch.loading.finished();
    initLibrary();
};
script.onerror = function() {
    op.patch.loading.finished();
    op.logError("Failed to load library");
};
document.head.appendChild(script);
```

8.13 Error Handling

```
try {
    // Risky operation
    const result = riskyFunction();
    outResult.set(result);
} catch (error) {
    op.logError("Operation failed:", error);
    op.setUiError("error", error.message);
}

// Clear error when fixed
op.setUiError("error", null);
```

8.14 Example: Custom Math Op

```
// Custom clamp with smoothing

const inValue = op.inFloat("Value", 0);
const inMin = op.inFloat("Min", 0);
const inMax = op.inFloat("Max", 1);
const inSmoothing = op.inFloat("Smoothing", 0);
const outValue = op.outNumber("Result");

```

```
let currentValue = 0;

function update() {
    let val = inValue.get();
    const min = inMin.get();
    const max = inMax.get();
    const smooth = inSmoothing.get();

    // Clamp
    val = Math.max(min, Math.min(max, val));

    // Smooth
    if (smooth > 0) {
        currentValue += (val - currentValue) * (1 - smooth);
    } else {
        currentValue = val;
    }

    outValue.set(currentValue);
}

inValue.onChange = update;
inMin.onChange = update;
inMax.onChange = update;
inSmoothing.onChange = update;
```

8.15 Example: Array Processor

```
// Sum all values in an array

const inArray = op.inArray("Values");
const outSum = op.outNumber("Sum");
const outAverage = op.outNumber("Average");
const outCount = op.outNumber("Count");

inArray.onChange = function() {
    const arr = inArray.get();

    if (!arr || arr.length === 0) {
        outSum.set(0);
        outAverage.set(0);
        outCount.set(0);
        return;
    }

    const sum = arr.reduce((a, b) => a + b, 0);
    const count = arr.length;
    const average = sum / count;

    outSum.set(sum);
    outAverage.set(average);
    outCount.set(count);
};
```

8.16 Example: API Fetcher

```
// Fetch data from an API

const inUrl = op.inString("API URL", "");
const inFetch = op.inTriggerButton("Fetch");
const outData = op.outObject("Data");
const outLoading = op.outBool("Loading");
const outError = op.outString("Error");

inFetch.onTriggered = async function() {
    const url = inUrl.get();
    if (!url) return;

    outLoading.set(true);
    outError.set("");

    try {
        const response = await fetch(url);
        const data = await response.json();
        outData.set(data);
    } catch (error) {
        outError.set(error.message);
        outData.set(null);
    } finally {
        outLoading.set(false);
    }
};
```

8.17 Debugging Tips

```
// Log to console
console.log("Value:", inValue.get());

// Op-specific logging (shows in cables UI)
op.log("This is a log message");
op.logWarn("This is a warning");
op.logError("This is an error");

// Visual debugging
op.setUiAttrib({ "error": "Something went wrong" });
```

8.18 Advanced Patterns (How to Build “Good” Ops)

Once you start writing more than a couple custom ops, quality becomes less about JavaScript syntax and more about **behavior**:

- **Determinism**: given the same inputs, the op produces the same outputs.
- **Clear execution model**: value changes vs trigger-based evaluation are intentional.
- **Performance**: avoid unnecessary allocations and expensive work per frame.
- **Good UI/UX**: errors are visible, defaults are sane, ports are grouped and labeled.

8.18.1 Pattern: Separate “Compute” from “Trigger”

A clean approach is:

- collect values in `onChange`
- do the heavy compute in one `update()` function
- call `update()` from whichever events are relevant

```
const inTrigger = op.inTrigger("Update");
const inA = op.inFloat("A", 0);
const inB = op.inFloat("B", 0);
const outResult = op.outNumber("Result");
const outNext = op.outTrigger("Next");

function update() {
  outResult.set(inA.get() + inB.get());
}

inA.onChange = update;
inB.onChange = update;

inTrigger.onTriggered = function () {
  update();
  outNext.trigger();
};
```

8.18.2 Pattern: "Only Recompute When Dirty"

If an op gets triggered every frame but its inputs rarely change, cache the result:

```
const inTrigger = op.inTrigger("Render");
const outNext = op.outTrigger("Next");

const inValue = op.inFloat("Value", 0);
const outProcessed = op.outNumber("Processed");

let dirty = true;
let cached = 0;

function recompute() {
  const v = inValue.get();
  // pretend this is expensive:
  cached = Math.sin(v) * Math.cos(v) * 1000;
  outProcessed.set(cached);
  dirty = false;
}

inValue.onChange = function () {
  dirty = true;
};

inTrigger.onTriggered = function () {
  if (dirty) recompute();
  outNext.trigger();
};
```

8.18.3 Pattern: Debounce (Stabilize Noisy Inputs)

Useful for sliders, mouse input, or network-driven values.

```
const inValue = op.inFloat("Value", 0);
const inDelayMs = op.inInt("Delay (ms)", 200);
const outValue = op.outNumber("Debounced");

let t = null;

inValue.onChange = function () {
  if (t) clearTimeout(t);
  const v = inValue.get();
  t = setTimeout(() => outValue.set(v), inDelayMs.get());
};
```

8.18.4 Pattern: Rate-Limit (Prevent Flooding Downstream)

Useful when sending values to other systems (e.g., API calls, heavy compute, UI).

```
const inTrigger = op.inTrigger("Trigger");
const inMinIntervalMs = op.inInt("Min Interval (ms)", 100);
const outNext = op.outTrigger("Next");

let last = 0;

inTrigger.onTriggered = function () {
  const now = performance.now();
  if (now - last >= inMinIntervalMs.get()) {
    last = now;
    outNext.trigger();
  }
};
```

8.18.5 Pattern: Stateful Ops (Resettable Systems)

Any op that accumulates state should expose a reset trigger.

```
const inAdd = op.inTrigger("Add");
const inReset = op.inTrigger("Reset");
const inValue = op.inFloat("Value", 1);
const outSum = op.outNumber("Sum");

let sum = 0;

function emit() {
  outSum.set(sum);
}

inAdd.onTriggered = function () {
  sum += inValue.get();
  emit();
};

inReset.onTriggered = function () {
  sum = 0;
  emit();
};
```

8.19 Async Ops (Fetching Data Safely)

When you talk to the network, the two most important qualities are:

- **cancellation**: don’t keep old requests alive if the user changes the URL
- **loading/error UX**: surface the state to the patch (and optionally the UI)

8.19.1 Example: Fetch JSON with Cancellation

```
const inUrl = op.inString("URL", "");
const inFetch = op.inTriggerButton("Fetch");

const outData = op.outObject("Data");
const outLoading = op.outBool("Loading");
const outError = op.outString("Error");

let controller = null;

inFetch.onTriggered = async function () {
  const url = inUrl.get();
  if (!url) return;

  // cancel previous request
  if (controller) controller.abort();
  controller = new AbortController();

  outLoading.set(true);
  outError.set("");

  try {
    const res = await fetch(url, { signal: controller.signal });
    if (!res.ok) throw new Error(`HTTP ${res.status}`);
    const json = await res.json();
    outData.set(json);
  } catch (e) {
    // ignore abort errors as “expected”
    if (e && e.name === "AbortError") return;
    outError.set(String(e && e.message ? e.message : e));
    outData.set(null);
  } finally {
    outLoading.set(false);
  }
};
```

8.19.2 Loading Semantics (Patch-Friendly)

If an op blocks the patch from being “ready” until something loads, use the patch loading tracking mechanism shown earlier (`op.patch.loading.start()` / `finished()`), and keep those calls paired even on error paths.

8.20 Performance Tips for Custom Ops

- **Avoid allocations in per-frame triggers**: reuse arrays/objects when possible.
- **Minimize DOM work**: avoid creating elements repeatedly; cache references.
- **Don’t spam logs**: logging inside every-frame triggers will kill performance.
- **Prefer simple math**: it’s easy to do too much in JS when the GPU could do it (shader).

8.21 Featured Videos

8.22 Exercises

1. Create a custom op that formats a number with a prefix and suffix
2. Build an array shuffler op
3. Create a simple state machine op
4. Build an op that fetches and parses CSV data

---

9 Audio & Sound in Cables.gl

9.1 Introduction

Cables.gl has powerful audio capabilities, enabling you to create audio-reactive visuals, music visualizations, and interactive sound experiences.

9.2 Audio Sources

9.2.1 AudioFile

Load and play audio files:

```
AudioFile -> AudioAnalyzer -> Visual ops
```

Supported Formats: - MP3 - WAV - OGG

Key Parameters: - `URL` - Path to audio file - `Loop` - Repeat playback - `Volume` - Playback volume - `Playback Rate` - Speed control

9.2.2 Microphone

Capture live audio input:

```
Microphone -> AudioAnalyzer -> Visual ops
```

Note: Requires user permission in browser.

9.2.3 AudioBuffer

Load audio into memory for precise control.

9.2.4 WebAudio Oscillator

Generate synthetic sounds:

```
Oscillator -> Audio output
```

Types: - Sine - Square - Sawtooth - Triangle

9.3 Audio Analysis

9.3.1 AudioAnalyzer

The core op for audio-reactive visuals:

```
AudioSource -> AudioAnalyzer
                   |
    Outputs: FFT, Volume, Bass, Mid, High
```

Key Outputs: - `FFT Array` - Frequency spectrum data - `Volume` - Overall loudness - `Bass` - Low frequency level - `Mid` - Middle frequency level - `High` - High frequency level

9.3.2 FFT (Fast Fourier Transform)

Breaks audio into frequency bands:

```
AudioAnalyzer -> FFTArray -> ArrayIterator
                                |
                          Visualize each band
```

FFT Size Options: - 32, 64, 128, 256, 512, 1024, 2048, 4096 - Larger = more detail, but slower

9.3.3 Smoothing

Apply smoothing to prevent jittery visuals:

```
AudioValue -> Smooth -> Visual parameter
```

9.4 Common Audio-Reactive Patterns

9.4.1 Volume-Based Scaling

```
AudioAnalyzer (volume) -> Scale input of shape
```

9.4.2 Frequency Band Visualization

```
MainLoop
    |
BasicMaterial
    |
AudioAnalyzer -> FFTArray
    |
ArrayIterator
    |
Transform (X position from index)
    |
Transform (Y scale from FFT value)
    |
Rectangle
```

9.4.3 Color from Audio

```
AudioAnalyzer (bass) -> Hue input of HSBtoRGB
HSBtoRGB -> BasicMaterial (color input)
```

9.4.4 Beat Detection

```
AudioAnalyzer (volume) -> Threshold -> Trigger
                             |
                    (triggers on beat)
```

9.5 Audio Effects

9.5.1 Gain

Control volume:

```
AudioSource -> Gain -> Output
```

9.5.2 Filter

Shape the frequency content:

```
AudioSource -> Filter -> Output
```

Filter Types: - Lowpass - Removes high frequencies - Highpass - Removes low frequencies - Bandpass - Keeps only middle frequencies - Notch - Removes specific frequency

9.5.3 Delay

Add echo effect:

```
AudioSource -> Delay -> Output
```

9.5.4 Reverb

Add space/ambience:

```
AudioSource -> Reverb -> Output
```

9.5.5 Compressor

Even out dynamics:

```
AudioSource -> Compressor -> Output
```

9.6 Building a Visualizer

9.6.1 Step 1: Set Up Audio

```
AudioFile (your music)
    |
AudioAnalyzer
```

9.6.2 Step 2: Create Base Render

```
MainLoop
    |
Camera (for 3D) or BasicMaterial (for 2D)
```

9.6.3 Step 3: Add Audio-Reactive Elements

Example: Pulsing Circle

```
MainLoop -> BasicMaterial
                |
AudioAnalyzer (volume)
    |
Smooth (for smoother animation)
    |
Math (multiply by desired scale)
    |
Circle (size input)
```

9.6.4 Step 4: Add Frequency Visualization

```
AudioAnalyzer -> FFTArray
    |
ArrayIterator (iterate through frequencies)
    |
Index -> Calculate X position
    |
FFT Value -> Calculate height/color
    |
Rectangle (bar for each frequency)
```

9.7 Synchronizing to Music

9.7.1 BPM and Beat Sync

```
AudioFile
    |
BPMSync (set your song's BPM)
    |
Beat triggers for animations
```

9.7.2 Timeline with Audio

1. Load audio file
2. Add to timeline
3. Use timeline markers for sync points
4. Keyframe animations to match audio

9.8 Advanced Audio Techniques (Make It Feel "Musical")

Audio-reactive visuals often fail in the same way: they're *too jittery* and *too literal*. The goal is usually:

- stable motion with **musical** response (not "random noise" response)
- clear separation between **slow energy** (overall level) and **fast transients** (kicks/snare hits)
- mappings that feel good: log frequency, clamped ranges, smoothing that doesn't lag

9.8.1 Technique: Energy vs Transient (Two-Signal Approach)

Treat audio as two complementary control signals:

- **Energy**: smoothed volume/bass/mid/high (drives slow changes: camera drift, fog density, palette)
- **Transients**: thresholded + debounced triggers (drives discrete events: flashes, spawns, scene cuts)

Typical building blocks:

```
AudioAnalyzer (volume/bass/mid/high)
  +-> Smooth (slow) -> Energy signal
  +-> Threshold -> (optional Delay/Interval gating) -> Transient trigger
```

9.8.2 Technique: Log Frequency Mapping (Better Spectra)

FFT bins are linear in frequency, but our hearing is closer to logarithmic. If your spectrum visualization looks "all action on the left", try mapping indices in a non-linear way:

- compress the low bins less (give bass more space)
- compress high bins more (reduce over-detail)

Conceptually:

```
Index -> Normalize (0..1) -> Pow (curve) -> Sample FFT
```

9.8.3 Technique: Peak Hold (Readable Visuals)

Human-friendly meters often have a "peak hold" that decays slowly. You can build this by:

- capturing the max value over a short window
- then decaying it over time

Conceptually:

```
AudioValue -> Max (with previous peak) -> Decay over time -> Peak output
```

9.8.4 Technique: Band-Specific Control (Bass Drives Scale, High Drives Detail)

Instead of driving everything from overall volume:

- **bass** -> big scale/position changes
- **mid** -> color shifts or mid-size motion
- **high** -> small jitter/detail/particles

This makes visuals feel much more "mixed".

9.8.5 Technique: Audio -> Shader (The "Pro" Move)

Shading is where audio-reactive projects often become cinematic.

High-level pattern:

```
AudioAnalyzer (energy) -> Smooth -> Shader uniform (e.g., amount)
FFTArray -> (reduce / select bands) -> Shader uniform(s)
Time -> Shader uniform (time)
```

Then, in the shader, use audio as **a modulation source**, not as the final value. (Example: warp UVs slightly, not wildly.)

9.9 Advanced Patch Recipes

9.9.1 Recipe: Stable Beat Trigger (Avoid Double-Triggers)

The simplest fix for "machine-gun" beats is gating:

```
AudioAnalyzer (volume or bass)
  |
Threshold (set just above noise floor)
  |
(Gate / minimum time between triggers)
  |
Trigger (spawn / flash / step timeline)
```

9.9.2 Recipe: Audio-Reactive Post-Processing

Drive a texture effect strength from music:

```
MainLoop -> Camera -> RenderToTexture -> TextureEffect -> Output
                         ^
AudioAnalyzer (volume) -> Smooth -> Map -> effect strength
```

9.9.3 Recipe: Audio-Reactive 3D Equalizer (Optimized)

If you build an equalizer with many bars:

- keep geometry simple
- reduce FFT size to what you need
- avoid doing heavy work per bar per frame

Conceptually:

```
AudioAnalyzer -> FFTArray
  |
ArrayIterator (N bands)
  |
Transform (X from index, Y scale from FFT)
  |
Cube (bar)
```

9.9.4 Recipe: Audio-Driven Palette

Map energy to hue/saturation to get coherent color shifts:

```
AudioAnalyzer (mid) -> Smooth -> Map -> Hue
AudioAnalyzer (bass) -> Smooth -> Map -> Saturation
HSBtoRGB -> BasicMaterial (color)
```

9.10 Practical Examples

9.10.1 Example 1: Bass-Reactive Background

```
MainLoop
    |
AudioFile -> AudioAnalyzer (bass)
    |
Smooth (0.9)
    |
Map (0-1 to desired range)
    |
HSBtoRGB (bass controls saturation) -> BasicMaterial (color input)
    |
BasicMaterial
    |
FullscreenRectangle
```

9.10.2 Example 2: Circular Spectrum

```
MainLoop
    |
BasicMaterial
    |
AudioAnalyzer -> FFTArray
    |
ArrayIterator
    |
Transform (rotate based on index)
    |
Transform (translate by FFT value)
    |
Circle (small)
```

9.10.3 Example 3: Waveform Display

```
MainLoop
    |
BasicMaterial
    |
AudioAnalyzer -> WaveformArray
    |
PointCloud or LineStrip
```

9.10.4 Example 4: 3D Audio Visualization

```
MainLoop
    |
Camera -> OrbitControls
    |
AudioAnalyzer -> FFTArray
    |
ArrayIterator (creates ring)
    |
Transform (position in circle)
    |
Transform (scale Y by FFT)
    |
Cube
```

9.11 Performance Considerations

1. **FFT Size** - Use smallest size that gives needed detail
2. **Smoothing** - Higher smoothing = less CPU for animations
3. **Update Rate** - Don't need 60fps for all audio analysis
4. **Visualizer Complexity** - Balance detail with performance

9.12 Browser Audio Policies

Modern browsers require user interaction before playing audio:

1. Add a "Start" button
2. Start audio on button click
3. Or use `AudioContext.resume()` on first interaction

```
// In custom op or patch
document.addEventListener('click', () => {
    if (audioContext.state === 'suspended') {
        audioContext.resume();
    }
}, { once: true });
```

9.13 Featured Videos

9.14 Exercises

1. Create a simple volume meter with animated bars
2. Build a circular frequency spectrum visualizer
3. Make a 3D landscape that morphs to music
4. Create a beat-triggered strobe effect

---

10 Animation & Timeline in Cables.gl

10.1 Introduction

Cables.gl provides multiple ways to create animations, from simple time-based movements to complex keyframed sequences using the timeline.

10.2 Types of Animation

10.2.1 1. Procedural Animation

Using math and time to create continuous motion.

10.2.2 2. Keyframe Animation

Defining specific values at specific times.

10.2.3 3. Physics-Based Animation

Simulating natural motion with springs, gravity, etc.

10.2.4 4. Data-Driven Animation

Animating based on input data or user interaction.

10.3 Procedural Animation

10.3.1 The Time Op

The foundation of procedural animation:

```
Time -> Outputs current time in seconds
```

Uses: - Input for trigonometric functions - Driving continuous rotation - Creating loops and cycles

10.3.2 Basic Movement Patterns

Linear Movement:

```
Time -> Modulo (loop duration) -> Position
```

Oscillation (Sine Wave):

```
Time -> Sin -> Scale/Position
```

Bounce:

```
Time -> Sin -> Abs -> Position
```

Circular Motion:

```
Time -> Cos -> X position
Time -> Sin -> Y position
```

10.3.3 Easing Functions

Transform linear time into smooth curves:

Ease In (slow start):

```
t * t  // Quadratic
t * t * t  // Cubic
```

Ease Out (slow end):

```
1 - (1 - t) * (1 - t)
```

Ease In-Out (smooth both):

```
t < 0.5 ? 2 * t * t : 1 - pow(-2 * t + 2, 2) / 2
```

10.3.4 The Smooth Op

Smoothly interpolate towards target values:

```
TargetValue -> Smooth -> AnimatedValue
```

Parameter: - `Smoothing` - Higher = slower, smoother transitions

10.3.5 Spring Animation

Create bouncy, natural motion:

```
TargetValue -> Spring -> AnimatedValue
```

Parameters: - `Stiffness` - How quickly it moves - `Damping` - How quickly it settles

10.4 Timeline Animation

10.4.1 Opening the Timeline

1. Click the timeline icon in the toolbar
2. Or press `T` to toggle timeline visibility

10.4.2 Timeline Interface

```
+-------------------------------------------------------+
| [Play][Pause][Stop] [Time: 00:00:00]  [BPM: 120]   |
+-------------------------------------------------------+
| Property Name  | o----o--------o----o---------     |
| Property 2     | o---------o------o--------        |
| Property 3     | o---------------o----------       |
+-------------------------------------------------------+
| <- 0s        5s        10s       15s       20s ->    |
+-------------------------------------------------------+
```

10.4.3 Adding Keyframes

1. Select the op with the property to animate
2. Move the timeline playhead to the desired time
3. Set the value
4. Click the keyframe button (or right-click the property)

10.4.4 Keyframe Types

- **Linear** - Straight line between keyframes
- **Step** - Instant change at keyframe
- **Ease In** - Slow start
- **Ease Out** - Slow end
- **Ease In-Out** - Smooth start and end
- **Bezier** - Custom curve with handles

10.4.5 Editing Keyframes

- **Move:** Drag keyframe left/right (time) or up/down (value)
- **Delete:** Select and press Delete
- **Copy/Paste:** Ctrl+C, Ctrl+V
- **Multi-select:** Shift+click or drag box

10.4.6 Timeline Tracks

Organize animations into tracks:

- **Property tracks** - Individual values
- **Trigger tracks** - Fire events at specific times
- **Audio tracks** - Sync with music

10.5 Non-Linear Animation Clips (New Animation System - November 2025)

The new animation system in Cables.gl introduces powerful non-linear animation capabilities through **animation clips**. Clips are reusable, addable, and mixable animation sequences that can be layered and blended to create complex motion.

10.5.1 What Are Animation Clips?

Animation clips are self-contained animation sequences that can be: - **Reusable** - Create once, apply to multiple parameters - **Addable** - Layer multiple clips together (additive blending) - **Mixable** - Blend between clips with different weights - **Non-linear** - Don't require strict sequential playback

10.5.2 Creating Animation Clips

Step 1: Enable Clip Mode

1. Add an **Anim** operator to your patch
2. Connect it to the parameter you want to animate
3. Open the Anim operator's properties
4. Enable the **Clip** option
5. Assign a **Clip Name** (e.g., "bounce", "fadeIn", "rotate360")

```
Parameter -> Anim (Clip enabled, Name: "myClip") -> Animated Value
```

Step 2: Define Keyframes

1. With the Anim operator selected, open the Timeline
2. Set keyframes for your animation sequence
3. Adjust easing curves and timing
4. The animation is now stored as a named clip

Step 3: Apply Clips to Other Parameters

Once created, clips can be applied to any other Anim operator:

1. Add another Anim operator
2. In the Timeline, right-click on a keyframe
3. Select "Apply Clip" and choose your clip name
4. The clip's animation will be applied at that keyframe

10.5.3 Clip Properties and Options

Looping Modes

Clips support different looping behaviors:

- **None** - Play once and stop
- **Repeat** - Loop from start to end
- **Mirror** - Play forward, then backward
- **Offset** - Continue from end value

Interpolation Methods

- **Linear** - Straight interpolation
- **Ease In/Out** - Smooth acceleration/deceleration
- **Bezier** - Custom curve control
- **Step** - Instant value changes

10.5.4 Additive Animation (Layering Clips)

Multiple clips can be **added together** to create combined effects:

```
Base Value
    |
Anim Clip 1 ("bounce") -> Add
    |
Anim Clip 2 ("rotate") -> Add
    |
Anim Clip 3 ("scale") -> Add
    |
Final Animated Value
```

Use Cases: - Base idle animation + triggered bounce effect - Procedural motion + keyframed structure - Multiple independent motion layers

Example: Character Animation

```
Idle Clip (continuous breathing)
    |
Walk Clip (additive, triggered on movement)
    |
Jump Clip (additive, triggered on jump)
    |
Final Position
```

10.5.5 Mixable Animation (Blending Clips)

Clips can be **mixed** with different weights to blend between animations:

```
Clip A ("walk") -> Mix (weight: 0.7)
Clip B ("run")  -> Mix (weight: 0.3)
    |
Blended Animation
```

Blending Modes: - **Linear Blend** - Simple weighted average - **Smooth Blend** - Eased transition between clips - **Additive Blend** - Add clips together with weights

Example: Walk-to-Run Transition

```
Walk Clip -> Mix (weight: 1.0 - runProgress)
Run Clip  -> Mix (weight: runProgress)
    |
Smooth transition from walk to run
```

10.5.6 Clip Management

Organizing Clips

Clips are stored within your project and can be: - **Renamed** - Right-click clip in timeline -> Rename - **Duplicated** - Copy clip to create variations - **Deleted** - Remove unused clips - **Exported/Imported** - Share clips between projects

Clip Library

Access all clips in your project: 1. Open Timeline 2. Click “Clips” tab 3. View all available clips 4. Drag clips onto timeline tracks

10.5.7 Advanced Clip Techniques

Clip Offsets and Time Remapping

Apply clips at different time offsets:

```
Clip "bounce" (duration: 2s)
    |
Apply at t=0s: Full clip
Apply at t=5s: Clip starts here
Apply at t=10s: Clip with 0.5x speed (time remap)
```

Clip Masking

Use clips to mask or modulate other animations:

```
Base Animation -> Multiply
Clip "mask" (0 to 1) -> Multiply
    |
Masked Animation (only active where mask = 1)
```

Conditional Clip Playback

Control clip playback based on conditions:

```
Condition -> If
    +-> True: Play Clip A
    +-> False: Play Clip B
```

10.6 JavaScript Custom Op Integration with Animation System

The new animation system integrates seamlessly with JavaScript custom operators, allowing programmatic control and extension of animation capabilities.

10.6.1 Accessing Animation Data from Custom Ops

Reading Animation Values

```
// Get current animation value from an Anim op
const animOp = op.patch.findOpByName("MyAnimOp");
if (animOp) {
    const currentValue = animOp.outValue.get();
    // Use the animated value
}
```

Monitoring Animation State

```
const inTrigger = op.inTrigger("Render");
const outAnimValue = op.outNumber("Animation Value");
const outIsPlaying = op.outBool("Is Playing");

let animOp = null;

// Find the Anim op (call once on init)
op.onInit = function() {
    animOp = op.patch.findOpByName("MyAnimOp");
};

inTrigger.onTriggered = function() {
    if (animOp) {
        // Get current animated value
        outAnimValue.set(animOp.outValue.get());

        // Check if timeline is playing
        const timeline = op.patch.timeline;
        if (timeline) {
            outIsPlaying.set(timeline.isPlaying());
        }
    }
};
```

10.6.2 Controlling Timeline from Custom Ops

Playback Control

```
const inPlay = op.inTriggerButton("Play");
const inPause = op.inTriggerButton("Pause");
const inStop = op.inTriggerButton("Stop");
const inSeek = op.inFloat("Seek Time", 0);
const inSeekTrigger = op.inTrigger("Seek");

inPlay.onTriggered = function() {
    const timeline = op.patch.timeline;
    if (timeline) timeline.play();
};

inPause.onTriggered = function() {
    const timeline = op.patch.timeline;
    if (timeline) timeline.pause();
};

inStop.onTriggered = function() {
    const timeline = op.patch.timeline;
    if (timeline) timeline.stop();
};

inSeekTrigger.onTriggered = function() {
    const timeline = op.patch.timeline;
    if (timeline) {
        timeline.seek(inSeek.get());
    }
};
```

Timeline Time and Progress

```
const inTrigger = op.inTrigger("Render");
const outTime = op.outNumber("Current Time");
const outProgress = op.outNumber("Progress (0-1)");
const outDuration = op.outNumber("Total Duration");

inTrigger.onTriggered = function() {
    const timeline = op.patch.timeline;
    if (timeline) {
        const currentTime = timeline.getTime();
        const duration = timeline.getDuration();

        outTime.set(currentTime);
        outDuration.set(duration);
        outProgress.set(duration > 0 ? currentTime / duration : 0);
    }
};
```

10.6.3 Creating Animation Clips Programmatically

Generating Clip Data

```
// Custom op that generates animation clip data
const inDuration = op.inFloat("Duration", 2.0);
const inAmplitude = op.inFloat("Amplitude", 1.0);
const inFrequency = op.inFloat("Frequency", 1.0);
const inGenerate = op.inTriggerButton("Generate Clip");
const outClipData = op.outObject("Clip Data");

inGenerate.onTriggered = function() {
    const duration = inDuration.get();
    const amplitude = inAmplitude.get();
    const freq = inFrequency.get();
    const sampleRate = 60; // samples per second
    const numSamples = Math.floor(duration * sampleRate);

    const keyframes = [];
    for (let i = 0; i <= numSamples; i++) {
        const t = i / numSamples;
        const time = t * duration;
        // Generate sine wave animation
        const value = Math.sin(time * freq * Math.PI * 2) * amplitude;
        keyframes.push({
            time: time,
            value: value,
            easing: "easeInOut"
        });
    }

    outClipData.set({
        name: "generatedSine",
        duration: duration,
        keyframes: keyframes,
        loop: "repeat"
    });
};
```

10.6.4 Manipulating Animation Clips

Blending Multiple Clips

```
// Custom op that blends multiple animation clips
const inClipA = op.inObject("Clip A Data");
const inClipB = op.inObject("Clip B Data");
const inBlendFactor = op.inFloat("Blend Factor", 0.5); // 0 = A, 1 = B
const inTime = op.inFloat("Time", 0);
const outBlendedValue = op.outNumber("Blended Value");

inTime.onChange = function() {
    const clipA = inClipA.get();
    const clipB = inClipB.get();
    const blend = inBlendFactor.get();
    const t = inTime.get();

    if (!clipA || !clipB) return;

    // Sample both clips at time t
    const valueA = sampleClip(clipA, t);
    const valueB = sampleClip(clipB, t);

    // Blend
    const blended = valueA * (1 - blend) + valueB * blend;
    outBlendedValue.set(blended);
};

function sampleClip(clip, time) {
    const keyframes = clip.keyframes;
    if (!keyframes || keyframes.length === 0) return 0;
    // Clamp time to clip duration
    time = time % clip.duration;

    // Find surrounding keyframes
    for (let i = 0; i < keyframes.length - 1; i++) {
        if (time >= keyframes[i].time && time <= keyframes[i + 1].time) {
            // Interpolate
            const t0 = keyframes[i].time;
            const t1 = keyframes[i + 1].time;
            const v0 = keyframes[i].value;
            const v1 = keyframes[i + 1].value;

            const t = (time - t0) / (t1 - t0);
            return v0 + (v1 - v0) * t;
        }
    }

    return keyframes[keyframes.length - 1].value;
}
```

Additive Clip Combination

```
// Custom op that adds multiple clips together
const inClips = op.inArray("Clips Array");
const inTime = op.inFloat("Time", 0);
const outCombinedValue = op.outNumber("Combined Value");

inTime.onChange = function() {
    const clips = inClips.get();
    const t = inTime.get();

    if (!clips || clips.length === 0) {
        outCombinedValue.set(0);
        return;
    }
    let sum = 0;
    for (let i = 0; i < clips.length; i++) {
        const clip = clips[i];
        if (clip && clip.keyframes) {
            sum += sampleClip(clip, t);
        }
    }

    outCombinedValue.set(sum);
};
```

10.6.5 Advanced: Custom Easing Functions

```
// Custom op with advanced easing functions
const inValue = op.inFloat("Input (0-1)", 0);
const inEasingType = op.inSwitch("Easing",
    ["linear", "easeInQuad", "easeOutQuad", "easeInOutQuad",
     "easeInCubic", "easeOutCubic", "easeInOutCubic",
     "easeInElastic", "easeOutBounce"],
    "easeInOutQuad");
const outEased = op.outNumber("Eased Value");

inValue.onChange = function() {
    const t = Math.max(0, Math.min(1, inValue.get()));
    const type = inEasingType.get();
    let eased = 0;

    switch(type) {
        case "linear":
            eased = t;
            break;
        case "easeInQuad":
            eased = t * t;
            break;
        case "easeOutQuad":
            eased = 1 - (1 - t) * (1 - t);
            break;
        case "easeInOutQuad":
            eased = t < 0.5
                ? 2 * t * t
                : 1 - Math.pow(-2 * t + 2, 2) / 2;
            break;
        case "easeInCubic":
            eased = t * t * t;
            break;
        case "easeOutCubic":
            eased = 1 - Math.pow(1 - t, 3);
            break;
        case "easeInOutCubic":
            eased = t < 0.5
                ? 4 * t * t * t
                : 1 - Math.pow(-2 * t + 2, 3) / 2;
            break;
        case "easeInElastic":
            const c4 = (2 * Math.PI) / 3;
            eased = t === 0 ? 0 : t === 1 ? 1
                : -Math.pow(2, 10 * t - 10) * Math.sin((t * 10 - 10.75) * c4);
            break;
        case "easeOutBounce":
            const n1 = 7.5625;
            const d1 = 2.75;
            if (t < 1 / d1) {
                eased = n1 * t * t;
            } else if (t < 2 / d1) {
                eased = n1 * (t -= 1.5 / d1) * t + 0.75;
            } else if (t < 2.5 / d1) {
                eased = n1 * (t -= 2.25 / d1) * t + 0.9375;
            } else {
                eased = n1 * (t -= 2.625 / d1) * t + 0.984375;
            }
            break;
    }
```

```
    outEased.set(eased);
};
```

10.6.6 Real-Time Animation Modification

```
// Custom op that modifies animation in real-time based on input
const inBaseAnim = op.inObject("Base Animation Clip");
const inModifier = op.inFloat("Modifier", 1.0);
const inTime = op.inFloat("Time", 0);
const outModifiedValue = op.outNumber("Modified Value");

inTime.onChange = function() {
    const clip = inBaseAnim.get();
    const mod = inModifier.get();
    const t = inTime.get();

    if (!clip) return;

    // Sample base animation
    let value = sampleClip(clip, t);

    // Apply modifier (could be scale, offset, etc.)
    value *= mod;

    outModifiedValue.set(value);
};
```

10.6.7 Integration Example: Physics-Driven Animation

```
// Custom op that combines physics simulation with animation clips
const inAnimClip = op.inObject("Animation Clip");
const inPhysicsForce = op.inFloat("Physics Force", 0);
const inDamping = op.inFloat("Damping", 0.9);
const inTime = op.inFloat("Time", 0);
const outCombinedValue = op.outNumber("Combined Value");
```

```
let velocity = 0;
let position = 0;

inTime.onChange = function() {
    const clip = inAnimClip.get();
    const force = inPhysicsForce.get();
    const damp = inDamping.get();
    const t = inTime.get();

    // Get base animation value
    const animValue = clip ? sampleClip(clip, t) : 0;

    // Apply physics
    velocity += force;
    velocity *= damp;
    position += velocity;

    // Combine animation + physics
    const combined = animValue + position;
    outCombinedValue.set(combined);
};
```

10.6.8 Best Practices for Animation + Custom Ops

1. **Cache Clip Sampling** - If sampling clips every frame, cache results when time hasn't changed
2. **Batch Operations** - Process multiple clips in one op rather than multiple ops
3. **Use Native Anim Op When Possible** - Only use custom ops when you need functionality beyond built-in features
4. **Optimize Keyframe Lookups** - Use binary search for large clip keyframe arrays
5. **Handle Edge Cases** - Always check for null/undefined clips and handle time out of bounds

10.6.9 Example: Complete Animation Controller Op

```
// Comprehensive animation controller custom op
const inPlay = op.inTriggerButton("Play");
const inPause = op.inTriggerButton("Pause");
const inStop = op.inTriggerButton("Stop");
const inSeek = op.inFloat("Seek", 0);
const inSpeed = op.inFloat("Speed", 1.0);
const inLoop = op.inBool("Loop", true);

const outTime = op.outNumber("Current Time");
const outProgress = op.outNumber("Progress");
const outIsPlaying = op.outBool("Is Playing");

let currentTime = 0;
let isPlaying = false;
let lastFrameTime = 0;

op.onInit = function() {
    lastFrameTime = op.patch.timer.getTime();
};

const inRender = op.inTrigger("Render");
inRender.onTriggered = function() {
    const now = op.patch.timer.getTime();
    const delta = now - lastFrameTime;
    lastFrameTime = now;

    if (isPlaying) {
        currentTime += delta * inSpeed.get();

        const timeline = op.patch.timeline;
        if (timeline) {
            const duration = timeline.getDuration();
            if (currentTime >= duration) {
                if (inLoop.get()) {
                    currentTime = currentTime % duration;
                } else {
                    currentTime = duration;
                    isPlaying = false;
                }
            }

            timeline.seek(currentTime);
        }
    }

    outTime.set(currentTime);
    const timeline = op.patch.timeline;
    if (timeline) {
        const duration = timeline.getDuration();
        outProgress.set(duration > 0 ? currentTime / duration : 0);
    }
    outIsPlaying.set(isPlaying);
};

inPlay.onTriggered = function() {
    isPlaying = true;
    const timeline = op.patch.timeline;
    if (timeline) timeline.play();
};

inPause.onTriggered = function() {
    isPlaying = false;
    const timeline = op.patch.timeline;
    if (timeline) timeline.pause();
};

inStop.onTriggered = function() {
    isPlaying = false;
    currentTime = 0;
    const timeline = op.patch.timeline;
    if (timeline) {
        timeline.stop();
        timeline.seek(0);
    }
```

```
};

inSeek.onChange = function() {
    currentTime = inSeek.get();
    const timeline = op.patch.timeline;
    if (timeline) timeline.seek(currentTime);
};
```

10.7 Sequence and Timing Ops

10.7.1 Sequence

Chain multiple actions in order:

```
Trigger -> Sequence
             +-> Action 1
             +-> Action 2 (after delay)
             +-> Action 3 (after delay)
```

10.7.2 Delay

Pause before triggering:

```
Trigger -> Delay (seconds) -> DelayedAction
```

10.7.3 Timer

Count down or up:

```
StartTrigger -> Timer -> TimeValue
```

10.7.4 Interval

Trigger repeatedly:

```
Interval (every X seconds) -> RepeatedAction
```

10.8 Animation Patterns

10.8.1 Staggered Animation

Animate multiple items with offset timing:

```
ArrayIterator
    |
Index -> Delay offset
    |
AnimatedProperty
```

10.8.2 Loop with Pause

```
Time -> Modulo (total duration)
     -> If < activeTime: animate
     -> Else: hold at end value
```

10.8.3 Ping-Pong (Back and Forth)

```
Time -> Sin -> Map to range -> Property
```

Or with timeline: set keyframes to go forward then backward.

10.8.4 One-Shot Animation

```
Trigger -> SetValue (start)
       -> Smooth -> AnimatedValue
```

10.9 State Machines

Create complex animation logic:

10.9.1 Simple States

```
// In custom op
let state = "idle";

function setState(newState) {
    state = newState;
    switch(state) {
        case "idle":
            // Set idle animation params
            break;
        case "active":
            // Set active animation params
            break;
        case "exit":
            // Set exit animation params
            break;
    }
}
```

10.9.2 Transition Between States

Use Smooth or Spring ops to blend between state values.

10.10 Interactive Animation

10.10.1 Mouse-Based

```
MouseX -> Map to range -> Target value -> Smooth -> Property
```

10.10.2 Scroll-Based

```
ScrollPosition -> Map (0 to page height) -> (0 to 1) -> Animation progress
```

10.10.3 Click-Triggered

```
MouseClick -> Toggle state -> Smooth -> Animated property
```

10.11 Advanced Animation Systems (How to Build "Scenes")

As patches grow, animation becomes less about a single value moving and more about **systems**:

- multiple objects animated together ("shots" / "scenes")
- blending procedural motion with keyframed structure
- sequencing events reliably (no double-triggers, no race conditions)
- keeping things readable and maintainable

10.11.1 Layering: Timeline for Structure, Procedural for Life

A reliable pattern is:

- **Timeline**: controls the big structure (when things appear, when the camera moves, when a section starts/ends)
- **Procedural**: adds micro-motion (subtle noise, breathing, idle motion, wobble)

Example idea:

```
Timeline -> Base position
Time -> Sin (small) -> Add
Result -> Transform position
```

10.11.2 Shot-Based Timelines (Cinematic Organization)

Instead of one giant timeline track list, treat the timeline as a set of "shots":

- Shot 1: intro framing
- Shot 2: reveal
- Shot 3: close-up detail
- Shot 4: outro / logo

Each shot has:

- a start time, end time
- a camera pose
- a set of object visibility/alpha states

10.11.3 Animation Curves: Clamp Early, Map Late

If you see overshoot or sudden jumps, it's usually a range mismatch.

Good practice:

- normalize to 0..1 early
- clamp to 0..1 before sensitive operations
- map to target range at the end

Conceptually:

```
t (0..1) -> Clamp -> Ease -> Map (min..max)
```

10.11.4 Reusable "Rig" Pattern

For any object you animate often, create a mini rig:

- one Transform for position
- one Transform for rotation
- one Transform for scale
- optional "wobble" layer

This makes it easy to swap animation sources later without rewiring the whole patch.

10.11.5 Avoiding Jitter in Interactive Animation

If input is noisy (mouse, audio, sensors):

- map input into a safe range
- apply Smooth/Spring
- optionally add dead zones

```
Input -> Map -> Clamp -> Smooth -> Property
```

10.11.6 Choreographing Triggers Reliably

For sequences of actions:

- use `Sequence` for deterministic ordering
- use `Delay` for spacing

- use `Interval` for periodic triggers

The key is to avoid “implicit timing” where the order depends on frame timing.

10.12 Advanced Recipes

10.12.1 Recipe: Scroll-Driven Scene (Interactive Storytelling)

Use scroll position as a normalized progress value:

```
ScrollPosition -> Map (0..pageHeight -> 0..1) -> Clamp -> progress
progress -> Ease -> Drive camera/object parameters
```

Then you can tie multiple properties to the same progress signal for a coherent experience.

10.12.2 Recipe: Beat-Synced Timeline Sections

Use BPM sync to trigger timeline jumps or section changes:

```
AudioFile -> BPMSync -> Beat trigger
Beat trigger -> Sequence -> (advance state) -> set target animation values
```

10.12.3 Recipe: One-Shot “Punch” Animation (No Keyframes)

Great for UI hits, impacts, kick drums:

```
Trigger -> SetValue (1)
       -> Smooth (fast decay) -> scale/brightness
```

You can combine a fast rise + slower decay by chaining two Smooth ops with different parameters.

10.12.4 Recipe: Camera Rig (Orbit + Handheld Micro Motion)

```
Time -> Sin/Cos -> Orbit position
Random (small) -> Smooth -> micro offset
Add (orbit + micro) -> Camera position
LookAt -> Camera aim
```

This produces camera movement that feels “alive” but still controlled.

10.13 Practical Examples

10.13.1 Example 1: Bouncing Ball

```
MainLoop
    |
BasicMaterial
    |
Time -> Sin -> Abs -> Y position
    |
Transform
    |
Circle
```

10.13.2 Example 2: Rotating Carousel

```
MainLoop
    |
Camera
    |
ArrayIterator (items)
    |
Time + (Index * offset) -> Cos -> X position
Time + (Index * offset) -> Sin -> Z position
    |
Transform
    |
Item
```

10.13.3 Example 3: Fade In Sequence

```
MainLoop
    |
BasicMaterial
```

```
    |
ArrayIterator
    |
Time - (Index * staggerDelay) -> Clamp (0, 1) -> BasicMaterial (alpha input)
    |
Shape
```

10.13.4 Example 4: Timeline-Based Scene

```
Timeline
+-- 0s: Camera position keyframe
+-- 2s: Object appears (alpha 0->1)
+-- 4s: Object rotates
+-- 6s: Color change
+-- 8s: Fade out
```

10.13.5 Example 5: Layered Animation Clips (Additive)

Create a character with multiple animation layers:

```
Base Position (0, 0, 0)
    |
Anim Clip "idleBreath" (vertical oscillation) -> Add
    |
Anim Clip "walkCycle" (horizontal movement, triggered) -> Add
    |
Anim Clip "jump" (vertical boost, triggered) -> Add
    |
Final Position -> Transform
```

Setup: 1. Create "idleBreath" clip: 2-second vertical sine wave (amplitude: 0.1) 2. Create "walkCycle" clip: 1-second horizontal movement (0 to 1, repeat) 3. Create "jump" clip: 0.5-second vertical boost (0 to 2, one-shot) 4. Connect all three Anim ops to Add ops in sequence 5. Trigger walkCycle and jump clips via user input

10.13.6 Example 6: Blended Animation Clips (Mixable)

Smooth transition between walk and run:

```
Walk Clip -> Anim (weight: 1.0 - runBlend)
Run Clip  -> Anim (weight: runBlend)
    |
Mix -> Final Position
```

Setup: 1. Create "walk" clip: slow horizontal movement 2. Create "run" clip: fast horizontal movement 3. Use a Smooth op to blend between 0 (walk) and 1 (run) 4. Connect both clips to Mix op with blend factor

10.13.7 Example 7: Reusable Clip System

Create a library of reusable animation clips:

```
Clip Library:
- "fadeIn" (alpha 0->1, 1s, easeOut)
- "fadeOut" (alpha 1->0, 1s, easeIn)
- "bounce" (scale 1->1.2->1, 0.5s, easeOut)
- "slideInLeft" (x: -100->0, 1s, easeOut)
- "rotate360" (rotation 0->360, 2s, linear)

Apply to multiple objects:
Object 1: fadeIn at t=0s, bounce at t=2s
Object 2: slideInLeft at t=1s, fadeOut at t=5s
Object 3: rotate360 at t=3s (looping)
```

10.13.8 Example 8: JavaScript-Controlled Animation

Custom op that controls animation based on game state:

```
Game State -> Custom Op
    +-> State = "idle": Play "idle" clip
    +-> State = "walk": Play "walk" clip
    +-> State = "run": Play "run" clip
    +-> State = "jump": Play "jump" clip (one-shot)
    |
Selected Clip -> Anim -> Position
```

Custom Op Code:

```
const inState = op.inString("State", "idle");
const inTime = op.inFloat("Time", 0);
const outClipName = op.outString("Clip Name");
const outValue = op.outNumber("Animation Value");

let currentClip = null;

inState.onChange = function() {
    const state = inState.get();
    switch(state) {
        case "idle":
            currentClip = "idle";
            break;
        case "walk":
            currentClip = "walk";
            break;
        case "run":
            currentClip = "run";
            break;
        case "jump":
            currentClip = "jump";
            break;
    }
    outClipName.set(currentClip);
};

inTime.onChange = function() {
    // Sample the current clip
    if (currentClip) {
        const animOp = op.patch.findOpByName("Anim_" + currentClip);
        if (animOp) {
            outValue.set(animOp.outValue.get());
        }
    }
};
```

10.13.9 Example 9: Physics + Animation Clip Hybrid

Combine procedural physics with keyframed animation:

```
Anim Clip "baseMotion" (keyframed path)
    |
Add
Physics Force (gravity, wind) -> Integrate -> Add
    |
Final Position
```

Custom Op for Physics Integration:

```
const inAnimValue = op.inFloat("Animation Value", 0);
const inPhysicsForce = op.inFloat("Physics Force", 0);
const inDamping = op.inFloat("Damping", 0.95);
const inRender = op.inTrigger("Render");
const outCombined = op.outNumber("Combined Value");

let velocity = 0;
let position = 0;

inRender.onTriggered = function() {
    const delta = op.patch.timer.getDelta();
    const anim = inAnimValue.get();
    const force = inPhysicsForce.get();
    const damp = inDamping.get();

    // Update physics
    velocity += force * delta;
    velocity *= damp;
    position += velocity * delta;

    // Combine with animation
    outCombined.set(anim + position);
};
```

10.13.10 Example 10: Conditional Clip Playback

Play different clips based on conditions:

```
Condition A -> If (True: Clip A, False: Clip B)
Condition B -> If (True: Clip C, False: Clip D)
    |
Mix (blend between conditional results)
    |
Final Animation
```

10.14 Performance Tips

1. **Limit active animations** - Don't animate everything
2. **Use requestAnimationFrame** - Built into cables.gl
3. **Cache calculations** - Don't recalculate every frame
4. **Simplify when far** - Reduce animation complexity for distant objects
5. **Use GPU** - Animate in shaders when possible

10.15 Debugging Animation

10.15.1 Slow Motion

```
Time -> Multiply (0.1) -> SlowTime
```

10.15.2 Visualize Values

Add a `DrawNumber` op to see animated values in real-time.

10.15.3 Pause at Problem

Use timeline pause to inspect a specific frame.

10.16 Featured Videos

10.17 Exercises

10.17.1 Basic Animation

1. Create a loading animation with staggered dots
2. Build an interactive hover animation
3. Design a full intro sequence with timeline
4. Create a physics-based pendulum

10.17.2 Animation Clips

5. Create a reusable “bounce” clip and apply it to 5 different objects
6. Build a character animation system with 3 additive clips (idle, walk, jump)
7. Create a smooth walk-to-run transition using clip blending
8. Design a clip library with 5 common animations (fade, slide, scale, rotate, bounce)

10.17.3 JavaScript Integration

9. Build a custom op that generates a sine wave animation clip programmatically
10. Create an animation controller op with play/pause/stop/seek functionality
11. Design a custom op that blends two animation clips with a configurable blend factor
12. Build a state machine op that switches between different animation clips based on input

10.17.4 Advanced

13. Combine procedural animation (Time -> Sin) with a keyframed clip using additive blending
14. Create a custom easing function op and apply it to an animation clip
15. Build a system that plays different animation clips based on user interaction (mouse, keyboard, touch)
16. Design a complex scene with multiple objects, each using a combination of clips and procedural motion

---

11 Interfaces in Cables.gl

11.1 Introduction

Cables.gl provides multiple ways to create user interfaces for your patches. You can build interfaces using HTML and CSS for full customization, or use native Cables sidebar interface operators for quick, integrated controls. This chapter covers both approaches in detail.

11.2 Interface Approaches Overview

```
+--------------------------------------------------------------+
|                     INTERFACE OPTIONS                        |
+--------------------------------------------------------------+
|                                                              |
|  1. HTML/CSS Interfaces                                      |
|     +-------------------------------------+                  |
|     | Full DOM control                    |                  |
|     | Custom styling                      |                  |
|     | Overlay on canvas                   |                  |
|     | Complete flexibility                |                  |
|     +-------------------------------------+                  |
|                                                              |
|  2. Native Sidebar Interface Ops                             |
|     +-------------------------------------+                  |
|     | Built-in UI elements                |                  |
|     | Integrated with patch               |                  |
|     | CSS-stylable                        |                  |
|     | Quick to implement                  |                  |
|     +-------------------------------------+                  |
|                                                              |
+--------------------------------------------------------------+
```

11.3 HTML/CSS Interfaces

11.3.1 Overview

HTML/CSS interfaces give you complete control over the user interface. You can create custom overlays, forms, buttons, and any HTML element positioned over or alongside your canvas.

11.3.2 The HTML Op

The HTML op allows you to create and manipulate DOM elements directly within your patch.

Basic HTML Op Setup

```
+--------------------------------------------------------------+
|                      HTML OP FLOW                              |
+--------------------------------------------------------------+
|                                                                |
|  MainLoop                                                      |
|    |                                                           |
|  HTML Op                                                       |
|    +-> HTML Content (string)                                   |
|    +-> CSS Styles (string)                                     |
|    +-> Position (x, y)                                         |
|    +-> Size (width, height)                                    |
|    +-> Visibility (bool)                                         |
|    |                                                           |
|  DOM Element (rendered on page)                              |
|                                                                |
+--------------------------------------------------------------+
```

Creating a Simple HTML Interface

Step 1: Add HTML Op

1. Add a `MainLoop` op
2. Add an `HTML` op
3. Connect `MainLoop` -> `HTML`

Step 2: Define HTML Content

In the HTML op's "HTML" parameter, enter your HTML:

```
<div id="myInterface">
    <h1>My Interface</h1>
    <button id="myButton">Click Me</button>
    <input type="range" id="mySlider" min="0" max="100" value="50">
    <p id="myText">Value: <span id="valueDisplay">50</span></p>
</div>
```

Step 3: Add CSS Styling

In the HTML op's "CSS" parameter:

```
#myInterface {
    position: absolute;
    top: 20px;
    left: 20px;
    background: rgba(30, 30, 30, 0.9);
    padding: 20px;
    border-radius: 8px;
    color: white;
    font-family: Arial, sans-serif;
    z-index: 1000;
}

#myButton {
    background: #4a9eff;
    color: white;
    border: none;
    padding: 10px 20px;
    border-radius: 4px;
    cursor: pointer;
    font-size: 16px;
}

#myButton:hover {
    background: #5aaeff;
}
```

```
#mySlider {
    width: 200px;
    margin: 10px 0;
}

#myText {
    margin-top: 10px;
    font-size: 14px;
}
```

Step 4: Position the Interface

Set the HTML op's position parameters: - `X`: 0 (or desired x position) - `Y`: 0 (or desired y position) - `Width`: 300 - `Height`: 200

11.3.3 Connecting HTML to Patch Logic

Using JavaScript Custom Op for Interaction

To make HTML elements interactive with your patch, use a JavaScript custom op:

```
// Custom Op: HTML Controller
const inTrigger = op.inTrigger("Render");
const outSliderValue = op.outNumber("Slider Value");
const outButtonClicked = op.outTrigger("Button Clicked");

let sliderValue = 50;
let buttonClicked = false;

// Access DOM elements
op.onInit = function() {
    const slider = document.getElementById("mySlider");
    const button = document.getElementById("myButton");
    const display = document.getElementById("valueDisplay");

    if (slider) {
        slider.addEventListener("input", function(e) {
            sliderValue = parseFloat(e.target.value);
            if (display) {
                display.textContent = sliderValue;
            }
            outSliderValue.set(sliderValue);
        });
    }

    if (button) {
        button.addEventListener("click", function() {
            buttonClicked = true;
            outButtonClicked.trigger();
        });
    }
};

inTrigger.onTriggered = function() {
    outSliderValue.set(sliderValue);
    if (buttonClicked) {
        buttonClicked = false;
    }
};
```

Complete Example: Interactive Control Panel

```
+--------------------------------------------------------------+
|                    INTERACTIVE SETUP                           |
+--------------------------------------------------------------+
|                                                                |
|  MainLoop                                                      |
|    +-> HTML Op (UI elements)                                   |
|    +-> Custom Op (JavaScript controller)                       |
|         +-> Reads DOM events                                   |
|         +-> Outputs: Slider Value                             |
|         +-> Outputs: Button Trigger                           |
|         +-> Outputs: Text Input                               |
|              |                                                 |
|         Patch Logic (uses values)                             |
|              |                                              |
|         Visual Output (canvas)                             |
|                                                             |
+-------------------------------------------------------------+
```

HTML Content:

```
<div id="controlPanel">
    <h2>Animation Controls</h2>

    <div class="control-group">
        <label>Speed:</label>
        <input type="range" id="speedSlider" min="0.1" max="5" step="0.1" value="1">
        <span id="speedValue">1.0</span>
    </div>

    <div class="control-group">
        <label>Color:</label>
        <input type="color" id="colorPicker" value="#4a9eff">
    </div>

    <div class="control-group">
        <label>Mode:</label>
        <select id="modeSelect">
            <option value="normal">Normal</option>
            <option value="fast">Fast</option>
            <option value="slow">Slow</option>
        </select>
    </div>

    <button id="resetButton">Reset</button>
    <button id="playButton">Play/Pause</button>
</div>
```

CSS Styling:

```
#controlPanel {
    position: fixed;
    top: 20px;
    right: 20px;
    width: 280px;
    background: linear-gradient(135deg, #1e1e1e 0%, #2d2d2d 100%);
    padding: 24px;
    border-radius: 12px;
    box-shadow: 0 8px 32px rgba(0, 0, 0, 0.4);
    border: 1px solid rgba(255, 255, 255, 0.1);
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    z-index: 1000;
}

#controlPanel h2 {
    margin: 0 0 20px 0;
    color: #ffffff;
    font-size: 20px;
    font-weight: 600;
    border-bottom: 2px solid #4a9eff;
    padding-bottom: 10px;
}

.control-group {
    margin-bottom: 20px;
}

.control-group label {
    display: block;
    color: #b0b0b0;
    font-size: 14px;
    margin-bottom: 8px;
    font-weight: 500;
}

#speedSlider {
    width: 100%;
    height: 6px;
    border-radius: 3px;
    background: #3a3a3a;
    outline: none;
    -webkit-appearance: none;
}

#speedSlider::-webkit-slider-thumb {
    -webkit-appearance: none;
    appearance: none;
    width: 18px;
    height: 18px;
    border-radius: 50%;
    background: #4a9eff;
    cursor: pointer;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.3);
}

#speedSlider::-moz-range-thumb {
    width: 18px;
    height: 18px;
    border-radius: 50%;
    background: #4a9eff;
    cursor: pointer;
    border: none;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.3);
}

#speedValue {
    color: #4a9eff;
    font-weight: 600;
    margin-left: 10px;
}

#colorPicker {
    width: 100%;
    height: 40px;
    border: 2px solid #3a3a3a;
    border-radius: 6px;
    cursor: pointer;
    background: transparent;
}

#modeSelect {
    width: 100%;
    padding: 10px;
    background: #3a3a3a;
    color: #ffffff;
    border: 2px solid #3a3a3a;
    border-radius: 6px;
    font-size: 14px;
    cursor: pointer;
}

#modeSelect:hover {
    border-color: #4a9eff;
}

#modeSelect:focus {
    outline: none;
    border-color: #4a9eff;
}

button {
    width: 100%;
    padding: 12px;
    margin-top: 10px;
    background: #4a9eff;
    color: white;
    border: none;
    border-radius: 6px;
    font-size: 14px;
    font-weight: 600;
    cursor: pointer;
    transition: all 0.2s ease;
}

button:hover {
    background: #5aaeff;
    transform: translateY(-1px);
    box-shadow: 0 4px 12px rgba(74, 158, 255, 0.3);
}

button:active {
    transform: translateY(0);
}
```

11.3.4 Advanced HTML Interface Patterns

Pattern 1: Responsive Overlay

```
#myInterface {
    position: fixed;
    top: 0;
    left: 0;
    width: 100vw;
    height: 100vh;
    background: rgba(0, 0, 0, 0.8);
    display: flex;
    align-items: center;
    justify-content: center;
    z-index: 10000;
}

#myInterface .content {
    background: #2d2d2d;
    padding: 40px;
    border-radius: 12px;
    max-width: 500px;
    width: 90%;
}
```

Pattern 2: Sidebar Panel

```
#sidebar {
    position: fixed;
    top: 0;
    right: 0;
    width: 300px;
    height: 100vh;
    background: #1e1e1e;
    box-shadow: -4px 0 16px rgba(0, 0, 0, 0.3);
    padding: 20px;
    overflow-y: auto;
    z-index: 1000;
    transform: translateX(0);
    transition: transform 0.3s ease;
}

#sidebar.hidden {
    transform: translateX(100%);
}
```

Pattern 3: HUD (Heads-Up Display)

```
#hud {
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    pointer-events: none;
    z-index: 100;
}
```

```
#hud .info {
    position: absolute;
    top: 20px;
    left: 20px;
    color: white;
    font-family: monospace;
    font-size: 14px;
    text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.8);
}

#hud .crosshair {
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    width: 20px;
    height: 20px;
    border: 2px solid rgba(255, 255, 255, 0.5);
    border-radius: 50%;
}
```

11.4 Native Sidebar Interface Ops

11.4.1 Overview

Cables.gl provides native interface operators that create UI elements directly in the sidebar. These are faster to set up and integrate seamlessly with the patch system.

11.4.2 Available Interface Ops

```
+-------------------------------------------------------------+
|              NATIVE INTERFACE OPS                            |
+-------------------------------------------------------------+
|                                                              |
|  • Slider        - Numeric input with range                |
|  • Button        - Clickable trigger                         |
|  • Toggle        - Boolean on/off switch                     |
|  • Text Input    - String input field                       |
|  • Color Picker  - Color selection                          |
|  • Dropdown      - Selection from options                   |
|  • Number Input  - Direct numeric input                     |
|  • Text Display  - Display text/values                      |
|                                                              |
+-------------------------------------------------------------+
```

11.4.3 Basic Interface Op Setup

Example: Simple Control Panel

```
+-------------------------------------------------------------+
|                    SIDEBAR INTERFACE                         |
+-------------------------------------------------------------+
|                                                              |
|  Slider Op (Speed)                                          |
|    +-> Min: 0.1                                              |
|    +-> Max: 5.0                                              |
|    +-> Default: 1.0                                          |
|    +-> Output: Speed Value                                   |
|         |                                                    |
|    Patch Logic                                               |
|                                                              |
|  Button Op (Reset)                                           |
|    +-> Output: Trigger                                       |
|         |                                                    |
|    Reset Logic                                               |
|                                                              |
|  Toggle Op (Enabled)                                        |
|    +-> Output: Boolean                                        |
|         |                                                    |
|    Conditional Logic                                         |
|                                                              |
+-------------------------------------------------------------+
```

Step-by-Step: Creating a Sidebar Interface

Step 1: Add Interface Ops

1. Add a `Slider` op for speed control
2. Add a `Button` op for actions
3. Add a `Toggle` op for enable/disable
4. Add a `ColorPicker` op for color selection

Step 2: Configure Each Op

Slider Op: - Name: "Speed" - Min: 0.1 - Max: 5.0 - Default: 1.0 - Step: 0.1

Button Op: - Name: "Reset" - Label: "Reset Animation"

Toggle Op: - Name: "Enabled" - Default: true

ColorPicker Op: - Name: "Base Color" - Default: #4a9eff

Step 3: Connect to Patch

```
Speed Slider -> Multiply -> Animation Speed
Reset Button -> SetValue -> Reset Position
Enabled Toggle -> If -> Conditional Execution
ColorPicker -> SetColor -> Material Color
```

11.4.4 Styling Native Sidebar with CSS

This is a powerful technique that allows you to customize the appearance of native sidebar interface ops using CSS.

Understanding the Sidebar Structure

The sidebar interface ops render in a specific DOM structure that you can target with CSS:

```
+--------------------------------------------------------------+
|              SIDEBAR DOM STRUCTURE                           |
+--------------------------------------------------------------+
|                                                              |
|  <div class="cables-sidebar">                               |
|    <div class="cables-sidebar-content">                     |
|      <div class="cables-op-slider" data-op-name="Speed">     |
|        <label>Speed</label>                                  |
|        <input type="range" ...>                               |
|        <span class="value">1.0</span>                        |
|      </div>                                                   |
|      <div class="cables-op-button" data-op-name="Reset">     |
|        <button>Reset</button>                                 |
|      </div>                                                   |
|      ...                                                      |
|    </div>                                                     |
|  </div>                                                       |
|                                                               |
+--------------------------------------------------------------+
```

Method 1: Global CSS Injection

Use an HTML op to inject CSS that styles the entire sidebar:

HTML Op Setup:

```
<style id="sidebar-styles">
/* Sidebar styling will go here */
</style>
```

CSS Content:

```
/* Target the entire sidebar */
.cables-sidebar {
    background: linear-gradient(180deg, #1a1a1a 0%, #2d2d2d 100%);
    border-left: 2px solid #4a9eff;
}

/* Style all interface ops */
.cables-sidebar-content > div {
    background: rgba(255, 255, 255, 0.05);
    border-radius: 8px;
    padding: 16px;
    margin-bottom: 12px;
    border: 1px solid rgba(255, 255, 255, 0.1);
    transition: all 0.2s ease;
}

.cables-sidebar-content > div:hover {
    background: rgba(255, 255, 255, 0.08);
    border-color: #4a9eff;
}

/* Style slider ops specifically */
.cables-op-slider {
    /* Custom slider container */
}

.cables-op-slider label {
    color: #b0b0b0;
    font-size: 14px;
    font-weight: 500;
    margin-bottom: 8px;
    display: block;
    text-transform: uppercase;
    letter-spacing: 0.5px;
}

.cables-op-slider input[type="range"] {
    width: 100%;
    height: 6px;
    border-radius: 3px;
    background: #3a3a3a;
    outline: none;
    -webkit-appearance: none;
    margin: 10px 0;
}

.cables-op-slider input[type="range"]::-webkit-slider-thumb {
```

```css
    -webkit-appearance: none;
    appearance: none;
    width: 20px;
    height: 20px;
    border-radius: 50%;
    background: #4a9eff;
    cursor: pointer;
    box-shadow: 0 2px 8px rgba(74, 158, 255, 0.4);
    transition: all 0.2s ease;
}

.cables-op-slider input[type="range"]::-webkit-slider-thumb:hover {
    background: #5aaeff;
    transform: scale(1.1);
    box-shadow: 0 4px 12px rgba(74, 158, 255, 0.6);
}

.cables-op-slider input[type="range"]::-moz-range-thumb {
    width: 20px;
    height: 20px;
    border-radius: 50%;
    background: #4a9eff;
    cursor: pointer;
    border: none;
    box-shadow: 0 2px 8px rgba(74, 158, 255, 0.4);
}

.cables-op-slider .value {
    color: #4a9eff;
    font-weight: 600;
    font-size: 16px;
    float: right;
    margin-top: -24px;
}

/* Style button ops */
```

```
.cables-op-button button {
    width: 100%;
    padding: 12px 24px;
    background: linear-gradient(135deg, #4a9eff 0%, #3a8eef 100%);
    color: white;
    border: none;
    border-radius: 6px;
    font-size: 14px;
    font-weight: 600;
    cursor: pointer;
    transition: all 0.2s ease;
    text-transform: uppercase;
    letter-spacing: 1px;
    box-shadow: 0 4px 12px rgba(74, 158, 255, 0.3);
}

.cables-op-button button:hover {
    background: linear-gradient(135deg, #5aaeff 0%, #4a9eff 100%);
    transform: translateY(-2px);
    box-shadow: 0 6px 16px rgba(74, 158, 255, 0.4);
}

.cables-op-button button:active {
    transform: translateY(0);
    box-shadow: 0 2px 8px rgba(74, 158, 255, 0.3);
}

/* Style toggle ops */
.cables-op-toggle {
    display: flex;
    align-items: center;
    justify-content: space-between;
}

.cables-op-toggle label {
    color: #b0b0b0;
    font-size: 14px;
    font-weight: 500;
}

.cables-op-toggle input[type="checkbox"] {
    width: 50px;
    height: 26px;
    -webkit-appearance: none;
    appearance: none;
    background: #3a3a3a;
    border-radius: 13px;
    position: relative;
    cursor: pointer;
    transition: background 0.3s ease;
    border: 2px solid #2a2a2a;
}

.cables-op-toggle input[type="checkbox"]:checked {
    background: #4a9eff;
    border-color: #4a9eff;
}

.cables-op-toggle input[type="checkbox"]::before {
    content: '';
    position: absolute;
    width: 20px;
    height: 20px;
    border-radius: 50%;
    background: white;
    top: 1px;
    left: 1px;
    transition: transform 0.3s ease;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.3);
}

.cables-op-toggle input[type="checkbox"]:checked::before {
    transform: translateX(24px);
}

/* Style color picker ops */
.cables-op-colorpicker {
    display: flex;
    align-items: center;
    gap: 12px;
}

.cables-op-colorpicker label {
    color: #b0b0b0;
    font-size: 14px;
    font-weight: 500;
    flex: 1;
}

.cables-op-colorpicker input[type="color"] {
    width: 60px;
    height: 40px;
    border: 2px solid #3a3a3a;
    border-radius: 6px;
    cursor: pointer;
    background: transparent;
    transition: border-color 0.2s ease;
}

.cables-op-colorpicker input[type="color"]:hover {
    border-color: #4a9eff;
}

/* Style text input ops */
.cables-op-textinput input[type="text"] {
    width: 100%;
    padding: 10px 12px;
    background: #3a3a3a;
    color: #ffffff;
    border: 2px solid #3a3a3a;
    border-radius: 6px;
    font-size: 14px;
    transition: all 0.2s ease;
}

.cables-op-textinput input[type="text"]:focus {
    outline: none;
    border-color: #4a9eff;
    background: #404040;
    box-shadow: 0 0 0 3px rgba(74, 158, 255, 0.1);
}

/* Style dropdown ops */
.cables-op-dropdown select {
    width: 100%;
    padding: 10px 12px;
    background: #3a3a3a;
    color: #ffffff;
    border: 2px solid #3a3a3a;
    border-radius: 6px;
    font-size: 14px;
    cursor: pointer;
    transition: all 0.2s ease;
}

.cables-op-dropdown select:hover {
    border-color: #4a9eff;
}

.cables-op-dropdown select:focus {
    outline: none;
    border-color: #4a9eff;
    box-shadow: 0 0 0 3px rgba(74, 158, 255, 0.1);
}
```

Method 2: Targeted Op Styling

Style specific ops by their data attributes:

```
/* Style a specific slider by op name */
.cables-op-slider[data-op-name="Speed"] {
    background: rgba(74, 158, 255, 0.1);
    border: 2px solid #4a9eff;
}

.cables-op-slider[data-op-name="Speed"] label {
    color: #4a9eff;
    font-weight: 600;
}

/* Style a specific button */
.cables-op-button[data-op-name="Reset"] button {
    background: linear-gradient(135deg, #ff4a4a 0%, #ef3a3a 100%);
}

.cables-op-button[data-op-name="Reset"] button:hover {
    background: linear-gradient(135deg, #ff5a5a 0%, #ff4a4a 100%);
}
```

Method 3: Dynamic CSS with JavaScript Custom Op

Create a custom op that injects CSS based on patch state:

```
// Custom Op: Dynamic Sidebar Styling
const inTheme = op.inSwitch("Theme", ["dark", "light", "neon"], "dark");
const inAccentColor = op.inString("Accent Color", "#4a9eff");

let currentTheme = "dark";
let currentAccent = "#4a9eff";

function updateStyles() {
    const theme = inTheme.get();
    const accent = inAccentColor.get();

    if (theme === currentTheme && accent === currentAccent) return;

    currentTheme = theme;
    currentAccent = accent;

    let styleElement = document.getElementById("dynamic-sidebar-styles");
    if (!styleElement) {
        styleElement = document.createElement("style");
        styleElement.id = "dynamic-sidebar-styles";
        document.head.appendChild(styleElement);
    }

    let css = "";

    if (theme === "dark") {
        css = `
            .cables-sidebar {
                background: linear-gradient(180deg, #1a1a1a 0%, #2d2d2d 100%);
            }
            .cables-sidebar-content > div {
                background: rgba(255, 255, 255, 0.05);
                border-color: rgba(255, 255, 255, 0.1);
            }
        `;
    } else if (theme === "light") {
        css = `
            .cables-sidebar {
                background: linear-gradient(180deg, #f5f5f5 0%, #e0e0e0 100%);
            }
            .cables-sidebar-content > div {
                background: rgba(0, 0, 0, 0.05);
                border-color: rgba(0, 0, 0, 0.1);
            }
            .cables-op-slider label,
            .cables-op-button label {
                color: #333;
            }
        `;
    } else if (theme === "neon") {
        css = `
            .cables-sidebar {
                background: #0a0a0a;
                border-left: 2px solid ${accent};
                box-shadow: -4px 0 20px ${accent}40;
            }
            .cables-sidebar-content > div {
                background: rgba(0, 0, 0, 0.5);
                border: 1px solid ${accent}40;
                box-shadow: 0 0 10px ${accent}20;
            }
        `;
    }

    // Apply accent color
    css += `
        .cables-op-slider input[type="range"]::-webkit-slider-thumb {
            background: ${accent};
            box-shadow: 0 2px 8px ${accent}60;
        }
        .cables-op-button button {
        background: linear-gradient(135deg, ${accent} 0%, ${adjustBrightness(accent, -
20)} 100%);
        }
        .cables-op-toggle input[type="checkbox"]:checked {
            background: ${accent};
        }
    `;

    styleElement.textContent = css;
```

```
}

function adjustBrightness(color, percent) {
    // Simple brightness adjustment (simplified)
    const num = parseInt(color.replace("#", ""), 16);
    const r = Math.max(0, Math.min(255, (num >> 16) + percent));
    const g = Math.max(0, Math.min(255, ((num >> 8) & 0x00FF) + percent));
    const b = Math.max(0, Math.min(255, (num & 0x0000FF) + percent));
    return "#" + ((r << 16) | (g << 8) | b).toString(16).padStart(6, "0");
}

inTheme.onChange = updateStyles;
inAccentColor.onChange = updateStyles;

op.onInit = function() {
    updateStyles();
};
```

11.4.5 Complete Styling Example: Professional Control Panel

Here's a complete example that styles all interface ops with a cohesive, professional design:

HTML Op (CSS Injection):

```
<style id="professional-sidebar-styles">
/* Professional Sidebar Styling */

/* Sidebar Container */
.cables-sidebar {
    background: linear-gradient(180deg,
        #1a1a1a 0%,
        #1e1e1e 50%,
        #2d2d2d 100%);
    border-left: 3px solid #4a9eff;
    box-shadow: -4px 0 24px rgba(0, 0, 0, 0.5);
    font-family: 'Inter', 'Segoe UI', system-ui, sans-serif;
}
```

```
/* Sidebar Header (if exists) */
.cables-sidebar-header {
    padding: 20px;
    border-bottom: 2px solid rgba(74, 158, 255, 0.2);
    background: rgba(74, 158, 255, 0.05);
}

.cables-sidebar-header h2 {
    margin: 0;
    color: #ffffff;
    font-size: 18px;
    font-weight: 600;
    text-transform: uppercase;
    letter-spacing: 1px;
}

/* Content Container */
.cables-sidebar-content {
    padding: 16px;
}

/* All Interface Op Containers */
.cables-sidebar-content > div {
    background: rgba(255, 255, 255, 0.03);
    border: 1px solid rgba(255, 255, 255, 0.08);
    border-radius: 10px;
    padding: 18px;
    margin-bottom: 16px;
    transition: all 0.3s cubic-bezier(0.4, 0, 0.2, 1);
    position: relative;
    overflow: hidden;
}

.cables-sidebar-content > div::before {
    content: '';
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 2px;
    background: linear-gradient(90deg,
        transparent 0%,
        #4a9eff 50%,
        transparent 100%);
    opacity: 0;
    transition: opacity 0.3s ease;
}

.cables-sidebar-content > div:hover {
    background: rgba(255, 255, 255, 0.06);
    border-color: rgba(74, 158, 255, 0.3);
    transform: translateX(4px);
    box-shadow: 0 4px 16px rgba(0, 0, 0, 0.3);
}

.cables-sidebar-content > div:hover::before {
    opacity: 1;
}

/* Slider Styling */
.cables-op-slider label {
    display: block;
    color: #b0b0b0;
    font-size: 12px;
    font-weight: 600;
    margin-bottom: 10px;
    text-transform: uppercase;
    letter-spacing: 0.5px;
}

.cables-op-slider input[type="range"] {
    width: 100%;
    height: 8px;
    border-radius: 4px;
    background: linear-gradient(90deg,
        #2a2a2a 0%,
        #3a3a3a 100%);
    outline: none;
    -webkit-appearance: none;
    margin: 12px 0;
    position: relative;
}

.cables-op-slider input[type="range"]::-webkit-slider-thumb {
    -webkit-appearance: none;
    appearance: none;
    width: 24px;
    height: 24px;
    border-radius: 50%;
    background: linear-gradient(135deg, #4a9eff 0%, #3a8eef 100%);
    cursor: pointer;
    box-shadow:
        0 2px 8px rgba(74, 158, 255, 0.4),
        0 0 0 4px rgba(74, 158, 255, 0.1),
        inset 0 1px 0 rgba(255, 255, 255, 0.2);
    transition: all 0.2s ease;
    border: 2px solid rgba(255, 255, 255, 0.1);
}

.cables-op-slider input[type="range"]::-webkit-slider-thumb:hover {
    background: linear-gradient(135deg, #5aaeff 0%, #4a9eff 100%);
    transform: scale(1.15);
    box-shadow:
        0 4px 12px rgba(74, 158, 255, 0.6),
        0 0 0 6px rgba(74, 158, 255, 0.15),
        inset 0 1px 0 rgba(255, 255, 255, 0.3);
}
```

```
.cables-op-slider input[type="range"]::-webkit-slider-thumb:active {
    transform: scale(1.05);
}

.cables-op-slider input[type="range"]::-moz-range-thumb {
    width: 24px;
    height: 24px;
    border-radius: 50%;
    background: linear-gradient(135deg, #4a9eff 0%, #3a8eef 100%);
    cursor: pointer;
    border: 2px solid rgba(255, 255, 255, 0.1);
    box-shadow:
        0 2px 8px rgba(74, 158, 255, 0.4),
        0 0 0 4px rgba(74, 158, 255, 0.1);
}

.cables-op-slider .value {
    color: #4a9eff;
    font-weight: 700;
    font-size: 18px;
    float: right;
    margin-top: -32px;
    font-variant-numeric: tabular-nums;
    text-shadow: 0 0 8px rgba(74, 158, 255, 0.5);
}

/* Button Styling */
.cables-op-button button {
    width: 100%;
    padding: 14px 24px;
    background: linear-gradient(135deg, #4a9eff 0%, #3a8eef 100%);
    color: white;
    border: none;
    border-radius: 8px;
    font-size: 14px;
    font-weight: 600;
    cursor: pointer;
    transition: all 0.3s cubic-bezier(0.4, 0, 0.2, 1);
    text-transform: uppercase;
    letter-spacing: 1.2px;
    box-shadow:
        0 4px 12px rgba(74, 158, 255, 0.3),
        inset 0 1px 0 rgba(255, 255, 255, 0.2);
    position: relative;
    overflow: hidden;
}

.cables-op-button button::before {
    content: '';
    position: absolute;
    top: 50%;
    left: 50%;
    width: 0;
    height: 0;
    border-radius: 50%;
    background: rgba(255, 255, 255, 0.3);
    transform: translate(-50%, -50%);
    transition: width 0.6s, height 0.6s;
}

.cables-op-button button:hover {
    background: linear-gradient(135deg, #5aaeff 0%, #4a9eff 100%);
    transform: translateY(-2px);
    box-shadow:
        0 6px 20px rgba(74, 158, 255, 0.4),
        inset 0 1px 0 rgba(255, 255, 255, 0.3);
}

.cables-op-button button:hover::before {
    width: 300px;
    height: 300px;
```

```
}

.cables-op-button button:active {
    transform: translateY(0);
    box-shadow:
        0 2px 8px rgba(74, 158, 255, 0.3),
        inset 0 1px 0 rgba(255, 255, 255, 0.1);
}

/* Toggle Styling */
.cables-op-toggle {
    display: flex;
    align-items: center;
    justify-content: space-between;
}

.cables-op-toggle label {
    color: #b0b0b0;
    font-size: 14px;
    font-weight: 500;
    flex: 1;
}

.cables-op-toggle input[type="checkbox"] {
    width: 56px;
    height: 30px;
    -webkit-appearance: none;
    appearance: none;
    background: #2a2a2a;
    border-radius: 15px;
    position: relative;
    cursor: pointer;
    transition: all 0.3s cubic-bezier(0.4, 0, 0.2, 1);
    border: 2px solid #1a1a1a;
    box-shadow: inset 0 2px 4px rgba(0, 0, 0, 0.3);
}
```

```
.cables-op-toggle input[type="checkbox"]:checked {
    background: linear-gradient(135deg, #4a9eff 0%, #3a8eef 100%);
    border-color: #4a9eff;
    box-shadow:
        inset 0 2px 4px rgba(0, 0, 0, 0.2),
        0 0 12px rgba(74, 158, 255, 0.4);
}

.cables-op-toggle input[type="checkbox"]::before {
    content: '';
    position: absolute;
    width: 24px;
    height: 24px;
    border-radius: 50%;
    background: linear-gradient(135deg, #ffffff 0%, #f0f0f0 100%);
    top: 1px;
    left: 1px;
    transition: transform 0.3s cubic-bezier(0.4, 0, 0.2, 1);
    box-shadow:
        0 2px 6px rgba(0, 0, 0, 0.3),
        inset 0 1px 0 rgba(255, 255, 255, 0.5);
}

.cables-op-toggle input[type="checkbox"]:checked::before {
    transform: translateX(26px);
}

/* Color Picker Styling */
.cables-op-colorpicker {
    display: flex;
    align-items: center;
    gap: 16px;
}

.cables-op-colorpicker label {
    color: #b0b0b0;
    font-size: 14px;
    font-weight: 500;
    flex: 1;
}

.cables-op-colorpicker input[type="color"] {
    width: 70px;
    height: 50px;
    border: 3px solid #3a3a3a;
    border-radius: 8px;
    cursor: pointer;
    background: transparent;
    transition: all 0.3s ease;
    box-shadow: 0 2px 8px rgba(0, 0, 0, 0.3);
}

.cables-op-colorpicker input[type="color"]:hover {
    border-color: #4a9eff;
    transform: scale(1.05);
    box-shadow:
        0 4px 12px rgba(0, 0, 0, 0.4),
        0 0 0 4px rgba(74, 158, 255, 0.1);
}

/* Text Input Styling */
.cables-op-textinput label {
    display: block;
    color: #b0b0b0;
    font-size: 12px;
    font-weight: 600;
    margin-bottom: 8px;
    text-transform: uppercase;
    letter-spacing: 0.5px;
}
```

```
.cables-op-textinput input[type="text"] {
    width: 100%;
    padding: 12px 16px;
    background: #2a2a2a;
    color: #ffffff;
    border: 2px solid #3a3a3a;
    border-radius: 8px;
    font-size: 14px;
    transition: all 0.3s ease;
    box-sizing: border-box;
}

.cables-op-textinput input[type="text"]:focus {
    outline: none;
    border-color: #4a9eff;
    background: #333333;
    box-shadow:
        0 0 0 4px rgba(74, 158, 255, 0.1),
        inset 0 2px 4px rgba(0, 0, 0, 0.2);
}

/* Dropdown Styling */
.cables-op-dropdown label {
    display: block;
    color: #b0b0b0;
    font-size: 12px;
    font-weight: 600;
    margin-bottom: 8px;
    text-transform: uppercase;
    letter-spacing: 0.5px;
}

.cables-op-dropdown select {
    width: 100%;
    padding: 12px 16px;
    background: #2a2a2a;
    color: #ffffff;
    border: 2px solid #3a3a3a;
    border-radius: 8px;
    font-size: 14px;
    cursor: pointer;
    transition: all 0.3s ease;
    appearance: none;
   background-image: url("data:image/svg+xml,%3Csvg xmlns='http://www.w3.org/2000/svg' width='12' h
    background-repeat: no-repeat;
    background-position: right 12px center;
    padding-right: 40px;
}

.cables-op-dropdown select:hover {
    border-color: #4a9eff;
    background-color: #333333;
}

.cables-op-dropdown select:focus {
    outline: none;
    border-color: #4a9eff;
    box-shadow:
        0 0 0 4px rgba(74, 158, 255, 0.1),
        inset 0 2px 4px rgba(0, 0, 0, 0.2);
}

/* Number Input Styling */
.cables-op-numberinput {
    display: flex;
    align-items: center;
    gap: 12px;
}

.cables-op-numberinput label {
    color: #b0b0b0;
    font-size: 14px;
    font-weight: 500;
    flex: 1;
}

.cables-op-numberinput input[type="number"] {
    width: 100px;
    padding: 10px 12px;
    background: #2a2a2a;
    color: #ffffff;
    border: 2px solid #3a3a3a;
    border-radius: 6px;
    font-size: 14px;
    text-align: center;
    transition: all 0.3s ease;
}

.cables-op-numberinput input[type="number"]:focus {
    outline: none;
    border-color: #4a9eff;
    background: #333333;
    box-shadow: 0 0 0 3px rgba(74, 158, 255, 0.1);
}

/* Text Display Styling */
.cables-op-textdisplay {
    padding: 12px;
    background: rgba(74, 158, 255, 0.1);
    border: 1px solid rgba(74, 158, 255, 0.3);
    border-radius: 6px;
    color: #4a9eff;
    font-family: 'Courier New', monospace;
    font-size: 14px;
    text-align: center;
    font-weight: 600;
}
```

```
/* Responsive adjustments */
@media (max-width: 768px) {
    .cables-sidebar {
        width: 100% !important;
        height: auto !important;
        position: relative !important;
    }
}
</style>
```

11.4.6 Advanced CSS Techniques

Technique 1: Animated Transitions

```
.cables-sidebar-content > div {
    animation: slideIn 0.3s ease-out;
}

@keyframes slideIn {
    from {
        opacity: 0;
        transform: translateX(-20px);
    }
    to {
        opacity: 1;
        transform: translateX(0);
    }
}

/* Stagger animation delays */
.cables-sidebar-content > div:nth-child(1) { animation-delay: 0.05s; }
.cables-sidebar-content > div:nth-child(2) { animation-delay: 0.10s; }
.cables-sidebar-content > div:nth-child(3) { animation-delay: 0.15s; }
.cables-sidebar-content > div:nth-child(4) { animation-delay: 0.20s; }
```

Technique 2: Custom Scrollbar

```
.cables-sidebar-content::-webkit-scrollbar {
    width: 8px;
}

.cables-sidebar-content::-webkit-scrollbar-track {
    background: #1a1a1a;
    border-radius: 4px;
}

.cables-sidebar-content::-webkit-scrollbar-thumb {
    background: #4a9eff;
    border-radius: 4px;
    border: 2px solid #1a1a1a;
}

.cables-sidebar-content::-webkit-scrollbar-thumb:hover {
    background: #5aaeff;
}
```

Technique 3: Glassmorphism Effect

```
.cables-sidebar {
    background: rgba(30, 30, 30, 0.7);
    backdrop-filter: blur(20px);
    -webkit-backdrop-filter: blur(20px);
    border-left: 1px solid rgba(255, 255, 255, 0.1);
}

.cables-sidebar-content > div {
    background: rgba(255, 255, 255, 0.05);
    backdrop-filter: blur(10px);
    -webkit-backdrop-filter: blur(10px);
    border: 1px solid rgba(255, 255, 255, 0.1);
}
```

11.5 Combining HTML and Native Interfaces

You can combine both approaches for maximum flexibility:

```
+--------------------------------------------------------------+
|              HYBRID INTERFACE APPROACH                       |
+--------------------------------------------------------------+
|                                                               |
|  Native Sidebar Ops                                          |
|    +-> Quick controls (sliders, buttons)                     |
|    +-> Styled with CSS                                         |
|                                                               |
|  HTML Overlay                                                |
|    +-> Complex UI elements                                    |
|    +-> Custom layouts                                         |
|    +-> Interactive components                                   |
|                                                               |
|  JavaScript Custom Op                                         |
|    +-> Bridges both systems                                   |
|    +-> Syncs values                                           |
|    +-> Handles interactions                                   |
|                                                               |
+--------------------------------------------------------------+
```

11.6 Best Practices

11.6.1 1. Performance

- **Minimize DOM Manipulation**: Cache element references
- **Use CSS Transforms**: For animations instead of position changes
- **Debounce Inputs**: For sliders and text inputs that trigger heavy computations

11.6.2 2. Accessibility

- **Labels**: Always provide clear labels for controls
- **Keyboard Navigation**: Ensure keyboard accessibility
- **Color Contrast**: Maintain sufficient contrast ratios

- **Focus States**: Provide visible focus indicators

11.6.3 3. Responsive Design

```css
/* Mobile-first approach */
.cables-sidebar {
    width: 100%;
    height: auto;
    position: relative;
}

@media (min-width: 768px) {
    .cables-sidebar {
        width: 320px;
        height: 100vh;
        position: fixed;
    }
}
```

11.6.4 4. Organization

- **Group Related Controls**: Use visual grouping
- **Clear Hierarchy**: Use size, color, and spacing
- **Consistent Spacing**: Maintain uniform margins and padding

11.7 Practical Examples

11.7.1 Example 1: Animation Control Panel

Create a comprehensive control panel for animation parameters:

```
Speed Slider -> Animation Speed
Color Picker -> Material Color
Toggle (Loop) -> Loop Animation
Button (Reset) -> Reset Animation
Text Display -> Current Frame
```

11.7.2 Example 2: Game UI Overlay

HTML overlay for game-like interface:

```
<div id="gameUI">
    <div class="hud-top">
        <div class="score">Score: <span id="score">0</span></div>
        <div class="health">Health: <span id="health">100</span></div>
    </div>
    <div class="hud-bottom">
        <button id="pauseBtn">Pause</button>
        <button id="menuBtn">Menu</button>
    </div>
</div>
```

11.7.3 Example 3: Data Visualization Dashboard

Combine native ops with HTML for a data dashboard:

- Native sliders for filtering
- HTML charts and graphs
- Real-time data display

11.8 Debugging Interface Issues

11.8.1 Common Issues

1. **CSS Not Applying**
    - Check selector specificity
    - Verify CSS is injected after sidebar renders
    - Use `!important` sparingly
2. **Elements Not Visible**
    - Check z-index values
    - Verify position properties
    - Check for overflow: hidden
3. **Events Not Firing**
    - Ensure JavaScript runs after DOM is ready

- Check event listener attachment
- Verify element selectors

11.8.2 Debugging Tools

```
// Log sidebar structure
console.log(document.querySelector('.cables-sidebar'));

// Check computed styles
const element = document.querySelector('.cables-op-slider');
console.log(window.getComputedStyle(element));

// Monitor style changes
const observer = new MutationObserver((mutations) => {
    console.log('DOM changed:', mutations);
});
observer.observe(document.querySelector('.cables-sidebar'), {
    childList: true,
    subtree: true,
    attributes: true
});
```

11.9 Exercises

1. **Basic HTML Interface**: Create a simple HTML overlay with a button and slider that control patch parameters
2. **Styled Sidebar**: Style native sidebar ops with a cohesive color scheme and modern design
3. **Responsive Panel**: Create a sidebar that adapts to different screen sizes
4. **Interactive Dashboard**: Build a complete control panel combining HTML and native ops
5. **Theme Switcher**: Create a custom op that dynamically changes sidebar styling based on user selection
6. **Advanced Styling**: Implement glassmorphism or other modern design trends in your sidebar

---

12 Export & Deployment in Cables.gl

12.1 Introduction

Once you've created your cables.gl project, you'll want to share it with the world. This chapter covers all the ways to export and deploy your creations.

12.2 Export Options

12.2.1 1. Public Patch Link

The simplest way to share - just make your patch public and share the URL.

Pros: - Instant sharing - Always up-to-date - No hosting needed

Cons: - Requires internet - Cables.gl branding - Limited customization

12.2.2 2. Embedded iframe

Embed your patch in any website:

```
<iframe
    src="https://cables.gl/view/YOUR_PATCH_ID"
    width="800"
    height="600"
    frameborder="0"
    allowfullscreen>
</iframe>
```

12.2.3 3. Standalone Export

Download your patch as a standalone web application.

Includes: - HTML file - JavaScript bundle - Assets (textures, models, audio) - No cables.gl dependency

12.2.4 4. npm Package Export

Export as an npm package for integration with other JavaScript projects.

12.3 Standalone Export Process

12.3.1 Step 1: Prepare Your Patch

1. Test thoroughly in the editor
2. Optimize assets (compress images, reduce model complexity)
3. Remove unused ops and connections
4. Set default camera/view position

12.3.2 Step 2: Export

1. Click the export/download button in the editor
2. Choose "Standalone" export
3. Configure options:
    - Include minified code
    - Include source maps (for debugging)
    - Asset optimization level

12.3.3 Step 3: Download

You'll receive a ZIP file containing:

```
exported-patch/
+-- index.html          # Main HTML file
+-- js/
|   +-- cables.min.js   # Cables runtime
|   +-- ops.js          # Your patch's operators
|   +-- patch.js        # Patch configuration
+-- assets/
|   +-- textures/       # Image files
|   +-- audio/          # Sound files
|   +-- models/         # 3D models
+-- css/
```

```
    +-- style.css       # Optional styles
```

12.3.4 Step 4: Test Locally

```
# Using Python
python -m http.server 8000

# Using Node.js
npx serve .

# Using PHP
php -S localhost:8000
```

Then open `http://localhost:8000` in your browser.

12.4 Customizing the Export

12.4.1 Custom HTML Template

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>My Cables Project</title>
    <style>
        body { margin: 0; overflow: hidden; }
        #cables-container { width: 100vw; height: 100vh; }
    </style>
</head>
<body>
    <div id="cables-container"></div>

    <script src="js/cables.min.js"></script>
    <script src="js/ops.js"></script>
    <script>
        CABLES.patch = new CABLES.Patch({
            patchFile: 'js/patch.js',
            prefixAssetPath: 'assets/',
            glCanvasId: 'cables-container',
            onFinishedLoading: function() {
                console.log('Patch loaded!');
            }
        });
    </script>
</body>
</html>
```

12.4.2 Configuration Options

```
new CABLES.Patch({
    patchFile: 'js/patch.js',
    prefixAssetPath: 'assets/',
    glCanvasId: 'myCanvas',
    glCanvasResizeToWindow: true,
    onFinishedLoading: callback,
    onError: errorCallback,
    variables: {
        // Pass custom variables to the patch
        customColor: '#ff0000',
        userName: 'Guest'
    }
});
```

12.5 Communicating with Your Patch

12.5.1 Setting Variables from JavaScript

```
// Get the patch instance
const patch = CABLES.patch;

// Set a variable
patch.setVariable('myValue', 42);
patch.setVariable('myColor', [1, 0, 0, 1]);
```

12.5.2 Getting Values from the Patch

```
// Get a variable
const value = patch.getVariable('myValue');

// Listen for variable changes
patch.on('variableChanged', function(name, value) {
    console.log(name, 'changed to', value);
});
```

12.5.3 Triggering Events

```
// Trigger an op
patch.getOpById('YOUR_OP_ID').trigger();

// Or use variables as triggers
patch.setVariable('doSomething', true);
```

12.6 Advanced Embedding & Integration

When cables.gl becomes part of a larger website/app, you want the embed to be **robust**:

- correct sizing and device pixel ratio handling
- pause/resume behavior when the tab is hidden
- a clean integration API (events in, telemetry out)
- predictable asset paths across dev/staging/prod

12.6.1 Responsive Canvas: Beyond Width/Height

If you embed into dynamic layouts (resizable panels, CSS grid, etc.), treat resize as a first-class event:

- call your resize function on load
- call it on `resize`
- call it when layout changes (route changes, UI toggles, etc.)

12.6.2 Pausing When Not Visible

For performance and battery life, consider pausing expensive animation when the page is hidden:

```
document.addEventListener("visibilitychange", () => {
  if (!window.CABLES || !CABLES.patch) return;
  // Depending on your patch/runtime, you may gate updates via a variable:
  CABLES.patch.setVariable("isVisible", !document.hidden);
});
```

Then in your patch, use `isVisible` to reduce workload (lower particle count, skip effects, etc.).

12.6.3 postMessage Integration (iframe Control)

If you embed via iframe, `postMessage` is the clean way to send commands and data.

Parent page -> iframe:

```
const iframe = document.getElementById("cablesFrame");
iframe.contentWindow.postMessage(
  { type: "CABLES_SET", name: "myValue", value: 0.75 },
  "*"
);
```

Inside the exported patch wrapper page:

```
window.addEventListener("message", (event) => {
  const msg = event.data;
  if (!msg || !window.CABLES || !CABLES.patch) return;

  if (msg.type === "CABLES_SET") {
    CABLES.patch.setVariable(msg.name, msg.value);
  }
});
```

12.6.4 Environment-Specific Configuration (dev / test / prod)

Keep environment differences in **configuration**, not in the patch logic:

- dev: verbose logging, source maps, local asset path
- test/staging: production-like hosting + debug overlays
- prod: minified, caching enabled, stable URLs

Common patterns:

- query string flags: `?debug=1`
- separate `config.json` loaded at runtime
- environment variables handled by the site that embeds the patch

12.6.5 Asset Path Gotchas

Most "works locally but not in prod" issues come down to:

- wrong `prefixAssetPath`
- case-sensitive paths on Linux hosts
- missing assets in the exported zip upload

If you deploy under a sub-path (e.g., `https://site.com/myproject/`), ensure all paths are relative or correctly prefixed.

12.7 Hosting Options

12.7.1 Static Hosting

Your exported patch is static files - host anywhere:

- **GitHub Pages** - Free, great for projects
- **Netlify** - Free tier, easy deployment
- **Vercel** - Free tier, automatic deploys
- **Amazon S3** - Scalable, pay-per-use
- **Any web server** - Apache, Nginx, etc.

12.7.2 GitHub Pages Deployment

```
# Create a gh-pages branch
git checkout -b gh-pages

# Add your exported files
git add .
git commit -m "Deploy cables patch"

# Push to GitHub
git push origin gh-pages
```

Enable GitHub Pages in repository settings.

12.7.3 Netlify Deployment

1. Connect your GitHub repository
2. Set build command: (none needed for static)
3. Set publish directory: / or your export folder
4. Deploy!

12.8 Embedding in Existing Websites

12.8.1 As a Background

```
<style>
    #cables-bg {
        position: fixed;
        top: 0;
        left: 0;
        width: 100%;
        height: 100%;
        z-index: -1;
    }
</style>
<canvas id="cables-bg"></canvas>
<script>
    CABLES.patch = new CABLES.Patch({
        patchFile: 'patch.js',
        glCanvasId: 'cables-bg'
    });
</script>
```

12.8.2 As a Hero Section

```
<section class="hero">
    <div id="cables-hero"></div>
    <div class="hero-content">
        <h1>Welcome</h1>
        <p>Your content here</p>
    </div>
</section>
```

12.8.3 Responsive Embedding

```
function resizeCables() {
    const container = document.getElementById('cables-container');
    container.style.width = window.innerWidth + 'px';
    container.style.height = window.innerHeight + 'px';

    // Notify cables of resize
    if (CABLES.patch) {
        CABLES.patch.cgl.setSize(window.innerWidth, window.innerHeight);
    }
}

window.addEventListener('resize', resizeCables);
resizeCables();
```

12.9 Performance Optimization

12.9.1 Before Export

1. **Remove unused ops** - Clean up your patch
2. **Optimize textures** - Use appropriate sizes

3. **Reduce polygon count** - Simplify 3D models
4. **Minimize audio files** - Compress audio

12.9.2 Asset Optimization

Images: - Use WebP format when possible - Use power-of-2 dimensions - Compress with tools like TinyPNG

3D Models: - Use glTF/GLB format - Remove unnecessary detail - Use Draco compression

Audio: - Use MP3 or OGG - Compress appropriately - Consider streaming for long files

12.9.3 Loading Optimization

```
// Show loading progress
CABLES.patch = new CABLES.Patch({
    patchFile: 'patch.js',
    onLoadingProgress: function(percent) {
        document.getElementById('loader').style.width = percent + '%';
    },
    onFinishedLoading: function() {
        document.getElementById('loader').style.display = 'none';
    }
});
```

12.10 Deployment Checklist (The Stuff That Breaks at the Worst Time)

Before you publish, run through this list:

- **Loading**: Do you show a loader/progress bar for heavy patches?
- **Autoplay policies**: If you use audio/video/webcam, do you require a user click?
- **Mobile sanity**: Does it run on a mid-tier phone without overheating?
- **Resize**: Does it handle orientation changes and dynamic layout resizing?
- **Asset paths**: Are all assets included and paths correct on a case-sensitive host?
- **Cache behavior**: Are you accidentally serving old JS after updates?
- **Console**: Is the browser console clean (no noisy logs, no repeated warnings)?

12.10.1 Cache Busting and Versioning

Static hosts cache aggressively. If you deploy a new version and still see the old one:

- add a version/hash to filenames (e.g. `ops.v123.js`)
- or configure cache headers (short cache for HTML, long cache for hashed assets)

12.10.2 MIME Types (Especially for Wasm / Binary Assets)

Some servers mis-serve file types. If a resource fails to load, check response headers:

- `.wasm` should be served as `application/wasm`
- `.json` as `application/json`
- textures as correct image mime types

12.10.3 CORS (Cross-Origin Assets)

If you load assets from another domain:

- ensure that server sends correct CORS headers
- prefer hosting assets alongside the patch when possible (simpler)

12.10.4 Content Security Policy (CSP)

If your patch is embedded into a site with strict CSP, you may need to allow:

- fetching assets from required domains
- media playback sources

When possible, avoid "unsafe-inline" and instead rely on your host app's approved patterns.

12.11 CI/CD Ideas (Optional, But Great for Teams)

If you repeatedly export and deploy:

- treat the export zip as a build artifact
- deploy to staging on every change
- promote to prod when approved

Even a simple workflow that publishes static files to GitHub Pages can save time and reduce mistakes.

12.12 Offline/PWA

Make your patch work offline as a Progressive Web App:

12.12.1 manifest.json

```json
{
    "name": "My Cables App",
    "short_name": "CablesApp",
    "start_url": "/",
    "display": "standalone",
    "background_color": "#000000",
    "theme_color": "#000000",
    "icons": [
        {
            "src": "icon-192.png",
            "sizes": "192x192",
            "type": "image/png"
        },
        {
            "src": "icon-512.png",
            "sizes": "512x512",
            "type": "image/png"
        }
    ]
}
```

12.12.2 Service Worker

```javascript
// sw.js
const CACHE_NAME = 'cables-app-v1';
const urlsToCache = [
    '/',
    '/index.html',
    '/js/cables.min.js',
    '/js/ops.js',
    '/js/patch.js',
    // Add your assets
];

self.addEventListener('install', event => {
    event.waitUntil(
        caches.open(CACHE_NAME)
            .then(cache => cache.addAll(urlsToCache))
    );
});

self.addEventListener('fetch', event => {
    event.respondWith(
        caches.match(event.request)
            .then(response => response || fetch(event.request))
    );
});
```

12.13 Electron Desktop Applications

For a truly native desktop experience, you can package your cables.gl export as an Electron application. Electron allows you to create cross-platform desktop apps using web technologies, perfect for distributing your cables.gl creations as standalone applications.

12.13.1 Why Electron?

Advantages: - Native desktop experience (menus, system tray, notifications) - Full file system access - Better performance control - No browser UI chrome - Can work offline completely - Access to native OS APIs - Professional distribution via installers

Considerations: - Larger app size (~100-200MB) - Requires code signing for distribution - More complex build process - Platform-specific considerations

12.13.2 Getting Started with Electron

Project Structure

After exporting your cables.gl patch, set up an Electron project:

```
electron-app/
+-- package.json
+-- main.js              # Main Electron process
+-- preload.js           # Preload script (optional)
+-- renderer/
|   +-- index.html       # Your exported cables HTML
|   +-- js/
|   |   +-- cables.min.js
|   |   +-- ops.js
|   |   +-- patch.js
|   +-- assets/          # Your exported assets
+-- assets/
|   +-- icon.ico         # Windows icon
|   +-- icon.icns        # macOS icon
|   +-- icon.png         # Linux icon
+-- build/               # Build configuration
    +-- mac/
    +-- win/
    +-- linux/
```

Initial Setup

package.json:

```
{
  "name": "my-cables-app",
  "version": "1.0.0",
  "description": "My Cables.gl Desktop App",
  "main": "main.js",
  "scripts": {
    "start": "electron .",
    "build": "electron-builder",
    "build:mac": "electron-builder --mac",
    "build:win": "electron-builder --win",
    "build:linux": "electron-builder --linux"
  },
  "build": {
    "appId": "com.yourcompany.cablesapp",
    "productName": "My Cables App",
    "directories": {
      "output": "dist"
    },
    "files": [
      "main.js",
      "preload.js",
      "renderer/**/*"
    ],
    "mac": {
      "icon": "assets/icon.icns",
      "category": "public.app-category.graphics-design"
    },
    "win": {
      "icon": "assets/icon.ico",
      "target": ["nsis", "portable"]
    },
    "linux": {
      "icon": "assets/icon.png",
      "target": ["AppImage", "deb"]
    }
  },
  "devDependencies": {
    "electron": "^28.0.0",
    "electron-builder": "^24.9.1"
  }
}
```

Install dependencies:

```
npm install --save-dev electron electron-builder
```

12.13.3 Main Process (main.js)

The main process controls the application lifecycle and creates windows:

```
const { app, BrowserWindow, Menu, ipcMain, dialog, shell } = require('electron');
const path = require('path');
const fs = require('fs').promises;

// Keep a global reference of the window object
let mainWindow;
let splashWindow;

// Determine if we're in development
const isDev = process.env.NODE_ENV === 'development' || !app.isPackaged;

function createSplashWindow() {
  splashWindow = new BrowserWindow({
    width: 400,
    height: 300,
    frame: false,
    transparent: true,
    alwaysOnTop: true,
    resizable: false,
    webPreferences: {
      nodeIntegration: false,
      contextIsolation: true
    }
  });

  // Load splash screen HTML
  splashWindow.loadFile('splash.html');

  // Center the window
  splashWindow.center();

  return splashWindow;
}

function createMainWindow() {
  // Create the browser window
```

```
mainWindow = new BrowserWindow({
  width: 1280,
  height: 720,
  minWidth: 800,
  minHeight: 600,
  show: false, // Don't show until ready
  frame: true,
  titleBarStyle: process.platform === 'darwin' ? 'hiddenInset' : 'default',
  backgroundColor: '#000000',
  icon: getIconPath(),
  webPreferences: {
    nodeIntegration: false, // Security: don't expose Node.js
    contextIsolation: true, // Security: isolate context
    preload: path.join(__dirname, 'preload.js'), // Preload script
    webSecurity: !isDev, // Disable in dev for easier debugging
    enableRemoteModule: false
  }
});

// Load your exported cables.gl patch
if (isDev) {
  mainWindow.loadFile('renderer/index.html');
  // Open DevTools in development
  mainWindow.webContents.openDevTools();
} else {
  mainWindow.loadFile(path.join(__dirname, 'renderer/index.html'));
}

// Show window when ready to prevent visual flash
mainWindow.once('ready-to-show', () => {
  if (splashWindow) {
    splashWindow.close();
    splashWindow = null;
  }
  mainWindow.show();
```

```
    // Focus the window
    if (isDev) {
      mainWindow.focus();
    }
  });

  // Handle window closed
  mainWindow.on('closed', () => {
    mainWindow = null;
  });

  // Handle external links
  mainWindow.webContents.setWindowOpenHandler(({ url }) => {
    shell.openExternal(url);
    return { action: 'deny' };
  });

  // Prevent navigation to external URLs
  mainWindow.webContents.on('will-navigate', (event, navigationUrl) => {
    const parsedUrl = new URL(navigationUrl);

    if (parsedUrl.origin !== 'file://') {
      event.preventDefault();
      shell.openExternal(navigationUrl);
    }
  });

  return mainWindow;
}

function getIconPath() {
  if (process.platform === 'win32') {
    return path.join(__dirname, 'assets/icon.ico');
  } else if (process.platform === 'darwin') {
    return path.join(__dirname, 'assets/icon.icns');
  } else {
    return path.join(__dirname, 'assets/icon.png');
  }
}

function createMenu() {
  const template = [
    {
      label: 'File',
      submenu: [
        {
          label: 'Load Settings',
          accelerator: 'CmdOrCtrl+O',
          click: async () => {
            const result = await dialog.showOpenDialog(mainWindow, {
              properties: ['openFile'],
              filters: [
                { name: 'JSON Files', extensions: ['json'] },
                { name: 'All Files', extensions: ['*'] }
              ]
            });

            if (!result.canceled && result.filePaths.length > 0) {
              mainWindow.webContents.send('load-settings', result.filePaths[0]);
            }
          }
        },
        {
          label: 'Save Settings',
          accelerator: 'CmdOrCtrl+S',
          click: async () => {
            const result = await dialog.showSaveDialog(mainWindow, {
              filters: [
                { name: 'JSON Files', extensions: ['json'] },
                { name: 'All Files', extensions: ['*'] }
              ],
              defaultPath: 'settings.json'
            });

            if (!result.canceled) {
              mainWindow.webContents.send('save-settings', result.filePath);
            }
          }
        },
        { type: 'separator' },
        {
          label: 'Exit',
          accelerator: process.platform === 'darwin' ? 'Cmd+Q' : 'Ctrl+Q',
          click: () => {
            app.quit();
          }
        }
      ]
    },
    {
      label: 'Edit',
      submenu: [
        { role: 'undo', label: 'Undo' },
        { role: 'redo', label: 'Redo' },
        { type: 'separator' },
        { role: 'cut', label: 'Cut' },
        { role: 'copy', label: 'Copy' },
        { role: 'paste', label: 'Paste' },
        { role: 'selectAll', label: 'Select All' }
      ]
    },
    {
      label: 'View',
      submenu: [
        { role: 'reload', label: 'Reload' },
        { role: 'forceReload', label: 'Force Reload' },
        { role: 'toggleDevTools', label: 'Toggle Developer Tools' },
        { type: 'separator' },
        { role: 'resetZoom', label: 'Actual Size' },
        { role: 'zoomIn', label: 'Zoom In' },
        { role: 'zoomOut', label: 'Zoom Out' },
        { type: 'separator' },
        { role: 'togglefullscreen', label: 'Toggle Fullscreen' }
      ]
    },
    {
      label: 'Window',
      submenu: [
        { role: 'minimize', label: 'Minimize' },
        { role: 'close', label: 'Close' }
      ]
    },
    {
      label: 'Help',
      submenu: [
        {
          label: 'About',
          click: () => {
            dialog.showMessageBox(mainWindow, {
              type: 'info',
              title: 'About',
              message: 'My Cables App',
              detail: 'Version 1.0.0\nBuilt with cables.gl and Electron'
            });
          }
        }
      ]
    }
  ];

  // macOS specific menu adjustments
  if (process.platform === 'darwin') {
    template.unshift({
      label: app.getName(),
      submenu: [
        { role: 'about', label: 'About ' + app.getName() },
        { type: 'separator' },
        { role: 'services', label: 'Services' },
        { type: 'separator' },
        { role: 'hide', label: 'Hide ' + app.getName() },
        { role: 'hideOthers', label: 'Hide Others' },
        { role: 'unhide', label: 'Show All' },
        { type: 'separator' },
        { role: 'quit', label: 'Quit ' + app.getName() }
      ]
    });

    // Window menu
    template[4].submenu = [
      { role: 'close', label: 'Close' },
      { role: 'minimize', label: 'Minimize' },
      { role: 'zoom', label: 'Zoom' },
      { type: 'separator' },
      { role: 'front', label: 'Bring All to Front' }
    ];
  }

  const menu = Menu.buildFromTemplate(template);
  Menu.setApplicationMenu(menu);
}

// IPC Handlers for inter-process communication
function setupIpcHandlers() {
  // Handle file reading
  ipcMain.handle('read-file', async (event, filePath) => {
    try {
      const data = await fs.readFile(filePath, 'utf-8');
      return { success: true, data: JSON.parse(data) };
    } catch (error) {
      return { success: false, error: error.message };
    }
  });

  // Handle file writing
  ipcMain.handle('write-file', async (event, filePath, data) => {
    try {
      await fs.writeFile(filePath, JSON.stringify(data, null, 2), 'utf-8');
      return { success: true };
    } catch (error) {
      return { success: false, error: error.message };
    }
  });

  // Get app version
  ipcMain.handle('get-app-version', () => {
    return app.getVersion();
  });

  // Get user data path
  ipcMain.handle('get-user-data-path', () => {
    return app.getPath('userData');
  });

  // Window control
  ipcMain.on('window-minimize', () => {
    if (mainWindow) mainWindow.minimize();
  });

  ipcMain.on('window-maximize', () => {
    if (mainWindow) {
      if (mainWindow.isMaximized()) {
        mainWindow.unmaximize();
      } else {
        mainWindow.maximize();
      }
    }
  });

  ipcMain.on('window-close', () => {
    if (mainWindow) mainWindow.close();
  });
}

// App event handlers
app.whenReady().then(() => {
  // Create splash screen
  createSplashWindow();

  // Create main window after a short delay (simulate loading)
  setTimeout(() => {
    createMainWindow();
    createMenu();
    setupIpcHandlers();
  }, 1500);

  app.on('activate', () => {
    // On macOS, re-create window when dock icon is clicked
    if (BrowserWindow.getAllWindows().length === 0) {
      createMainWindow();
    }
  });
});

app.on('window-all-closed', () => {
  // On macOS, keep app running even when all windows are closed
  if (process.platform !== 'darwin') {
    app.quit();
  }
});

// Security: Prevent new window creation
app.on('web-contents-created', (event, contents) => {
  contents.on('new-window', (event, navigationUrl) => {
    event.preventDefault();
    shell.openExternal(navigationUrl);
  });
});
```

12.13.4 Preload Script (preload.js)

The preload script safely exposes Node.js APIs to the renderer process:

```
const { contextBridge, ipcRenderer } = require('electron');

// Expose protected methods that allow the renderer process
// to use ipcRenderer without exposing the entire object
contextBridge.exposeInMainWorld('electronAPI', {
  // File operations
  readFile: (filePath) => ipcRenderer.invoke('read-file', filePath),
  writeFile: (filePath, data) => ipcRenderer.invoke('write-file', filePath, data),

  // App info
  getAppVersion: () => ipcRenderer.invoke('get-app-version'),
  getUserDataPath: () => ipcRenderer.invoke('get-user-data-path'),

  // Window control
  minimizeWindow: () => ipcRenderer.send('window-minimize'),
  maximizeWindow: () => ipcRenderer.send('window-maximize'),
  closeWindow: () => ipcRenderer.send('window-close'),

  // Listen for messages from main process
  onLoadSettings: (callback) => {
    ipcRenderer.on('load-settings', (event, filePath) => callback(filePath));
  },
  onSaveSettings: (callback) => {
    ipcRenderer.on('save-settings', (event, filePath) => callback(filePath));
  },
  // Remove listeners
  removeAllListeners: (channel) => {
    ipcRenderer.removeAllListeners(channel);
  }
});
```

12.13.5 Advanced Window Configuration

Window Options Deep Dive

```
const mainWindow = new BrowserWindow({
  // Size and position
  width: 1280,
  height: 720,
  minWidth: 800,
  minHeight: 600,
  maxWidth: 3840,
  maxHeight: 2160,
  x: undefined, // Center if undefined
  y: undefined,
  center: true, // Center on screen

  // Appearance
  frame: true, // Show window frame
  titleBarStyle: 'default', // 'default', 'hidden', 'hiddenInset', 'customButtonsOnHover'
  transparent: false, // Transparent window (performance impact)
  backgroundColor: '#000000', // Background color before content loads
  opacity: 1.0, // Window opacity (0.0 to 1.0)
  vibrancy: 'ultra-dark', // macOS only: 'appearance-based', 'light', 'dark', etc.
  visualEffectState: 'active', // macOS only: 'active', 'inactive', 'followsWindowActiveState'

  // Behavior
  show: false, // Don't show until ready
  alwaysOnTop: false, // Keep window on top
  fullscreen: false, // Start in fullscreen
  fullscreenable: true, // Allow fullscreen
  simpleFullscreen: false, // macOS simple fullscreen
  skipTaskbar: false, // Don't show in taskbar
  kiosk: false, // Kiosk mode (fullscreen, no exit)
  closable: true, // Allow closing
  minimizable: true, // Allow minimizing
  maximizable: true, // Allow maximizing
  resizable: true, // Allow resizing
  movable: true, // Allow moving
  focusable: true, // Can receive focus

  // Window state
  autoHideMenuBar: false, // Auto-hide menu bar
  useContentSize: false, // Use content size instead of window size
  title: 'My Cables App', // Window title

  // Icon
  icon: getIconPath(), // Window icon

  // Web preferences
  webPreferences: {
    nodeIntegration: false,
    contextIsolation: true,
    preload: path.join(__dirname, 'preload.js'),
    webSecurity: true,
    allowRunningInsecureContent: false,
    experimentalFeatures: false,
    enableBlinkFeatures: '',
    disableBlinkFeatures: '',
    sandbox: false, // Enable sandbox for extra security
    enableRemoteModule: false,
    backgroundThrottling: true, // Throttle when backgrounded
    offscreen: false, // Use offscreen rendering
    webviewTag: false // Disable webview tag
  }
});
```

Window State Persistence

Save and restore window position and size:

```
const Store = require('electron-store');

const store = new Store({
  name: 'window-state',
  defaults: {
    width: 1280,
    height: 720,
    x: undefined,
    y: undefined,
    isMaximized: false
  }
});

function createMainWindow() {
  const windowState = store.get('windowState', {});

  const mainWindow = new BrowserWindow({
    width: windowState.width || 1280,
    height: windowState.height || 720,
    x: windowState.x,
    y: windowState.y,
    // ... other options
  });

  // Restore maximized state
  if (windowState.isMaximized) {
    mainWindow.maximize();
  }

  // Save window state on move/resize
  const saveWindowState = () => {
    const bounds = mainWindow.getBounds();
    store.set('windowState', {
      width: bounds.width,
      height: bounds.height,
      x: bounds.x,
      y: bounds.y,
      isMaximized: mainWindow.isMaximized()
    });
  };

  mainWindow.on('moved', saveWindowState);
  mainWindow.on('resized', saveWindowState);
  mainWindow.on('maximize', () => {
    store.set('windowState.isMaximized', true);
  });
  mainWindow.on('unmaximize', () => {
    store.set('windowState.isMaximized', false);
  });

  return mainWindow;
}
```

Install electron-store:

```
npm install electron-store
```

12.13.6 Inter-Window Communication

Electron supports multiple windows with various communication patterns:

Method 1: IPC (Inter-Process Communication)

Main Process -> Renderer Process:

```
// In main.js
mainWindow.webContents.send('message-from-main', {
  type: 'update',
  data: { value: 42 }
});

// In renderer (index.html or your cables patch)
window.electronAPI.onMessage((data) => {
  console.log('Received:', data);
});
```

Renderer Process -> Main Process:

```
// In preload.js
contextBridge.exposeInMainWorld('electronAPI', {
  sendToMain: (channel, data) => {
    ipcRenderer.send(channel, data);
  },
  onMessage: (callback) => {
    ipcRenderer.on('message-from-main', (event, data) => callback(data));
  }
});

// In renderer
window.electronAPI.sendToMain('message-from-renderer', {
  action: 'save',
  data: { settings: {...} }
});
```

Method 2: Multiple Windows Communication

```
// In main.js
let windows = [];

function createWindow(id) {
  const window = new BrowserWindow({
    // ... window options
    webPreferences: {
      // ... web preferences
    }
  });
  window.id = id;
  windows.push(window);

  window.on('closed', () => {
    windows = windows.filter(w => w.id !== id);
  });

  return window;
}

// Broadcast to all windows
function broadcastToAllWindows(channel, data) {
  windows.forEach(window => {
    if (window && !window.isDestroyed()) {
      window.webContents.send(channel, data);
    }
  });
}

// Send to specific window
function sendToWindow(windowId, channel, data) {
  const window = windows.find(w => w.id === windowId);
  if (window && !window.isDestroyed()) {
    window.webContents.send(channel, data);
  }
}

// Example: Sync settings across windows
ipcMain.on('update-settings', (event, settings) => {
  // Save settings
  store.set('settings', settings);

  // Broadcast to all windows
  broadcastToAllWindows('settings-updated', settings);
});
```

Method 3: Shared Data via Main Process

```
// In main.js
let sharedData = {
  settings: {},
  state: {}
};

// Get shared data
ipcMain.handle('get-shared-data', (event, key) => {
  return sharedData[key];
});

// Set shared data
ipcMain.handle('set-shared-data', (event, key, value) => {
  sharedData[key] = value;
  // Notify all windows
  broadcastToAllWindows('shared-data-changed', { key, value });
  return true;
});
```

Method 4: Window-to-Window via Main Process

```
// Window A sends message to Window B
ipcMain.on('send-to-window', (event, targetWindowId, channel, data) => {
  sendToWindow(targetWindowId, channel, data);
});

// In preload.js
contextBridge.exposeInMainWorld('electronAPI', {
  sendToWindow: (targetWindowId, channel, data) => {
    ipcRenderer.send('send-to-window', targetWindowId, channel, data);
  },
  onWindowMessage: (callback) => {
    ipcRenderer.on('window-message', (event, data) => callback(data));
  }
});
```

12.13.7 Splash Screen Implementation

A professional splash screen improves perceived performance:

splash.html:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <style>
    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
    }

    body {
      width: 400px;
      height: 300px;
      background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
      display: flex;
      flex-direction: column;
      justify-content: center;
      align-items: center;
     font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, sans-serif;
      color: white;
      overflow: hidden;
    }

    .logo {
      width: 80px;
      height: 80px;
      margin-bottom: 20px;
      animation: pulse 2s ease-in-out infinite;
    }
    @keyframes pulse {
      0%, 100% { transform: scale(1); opacity: 1; }
      50% { transform: scale(1.1); opacity: 0.8; }
    }

    .app-name {
      font-size: 24px;
      font-weight: 600;
      margin-bottom: 10px;
    }

    .version {
      font-size: 12px;
      opacity: 0.8;
      margin-bottom: 30px;
    }

    .loader {
      width: 200px;
      height: 4px;
      background: rgba(255, 255, 255, 0.2);
      border-radius: 2px;
      overflow: hidden;
      position: relative;
    }

    .loader-bar {
      height: 100%;
      background: white;
      width: 0%;
      animation: loading 2s ease-in-out infinite;
      border-radius: 2px;
    }

    @keyframes loading {
      0% { width: 0%; }
      50% { width: 70%; }
      100% { width: 100%; }
    }

    .status {
      margin-top: 20px;
      font-size: 12px;
      opacity: 0.7;
    }
  </style>
</head>
<body>
  <div class="logo">
    <!-- Your logo SVG or image -->
    <svg viewBox="0 0 100 100" fill="white">
      <circle cx="50" cy="50" r="40" stroke="white" stroke-width="2" fill="none"/>
      <path d="M30 50 L45 65 L70 35" stroke="white" stroke-width="3" fill="none"/>
    </svg>
  </div>
  <div class="app-name">My Cables App</div>
  <div class="version">Version 1.0.0</div>
  <div class="loader">
    <div class="loader-bar"></div>
  </div>
  <div class="status" id="status">Loading...</div>

  <script>
    // Update status from main process
    const { ipcRenderer } = require('electron');

    ipcRenderer.on('splash-status', (event, message) => {
      document.getElementById('status').textContent = message;
    });

    ipcRenderer.on('splash-progress', (event, percent) => {
      document.querySelector('.loader-bar').style.width = percent + '%';
    });
  </script>
</body>
</html>
```

Enhanced main.js with splash screen:

```
function createSplashWindow() {
  splashWindow = new BrowserWindow({
    width: 400,
    height: 300,
    frame: false,
    transparent: true,
    alwaysOnTop: true,
    resizable: false,
    webPreferences: {
      nodeIntegration: true, // Needed for splash screen
      contextIsolation: false
    }
  });

  splashWindow.loadFile('splash.html');
  splashWindow.center();

  // Update splash screen status
  const updateSplashStatus = (message) => {
    if (splashWindow && !splashWindow.isDestroyed()) {
      splashWindow.webContents.send('splash-status', message);
    }
  };

  const updateSplashProgress = (percent) => {
    if (splashWindow && !splashWindow.isDestroyed()) {
      splashWindow.webContents.send('splash-progress', percent);
    }
  };
  // Simulate loading progress
  updateSplashStatus('Initializing...');
  updateSplashProgress(10);

  setTimeout(() => {
    updateSplashStatus('Loading assets...');
    updateSplashProgress(40);
  }, 300);

  setTimeout(() => {
    updateSplashStatus('Preparing renderer...');
    updateSplashProgress(70);
  }, 800);

  setTimeout(() => {
    updateSplashStatus('Almost ready...');
    updateSplashProgress(90);
  }, 1200);

  return { splashWindow, updateSplashStatus, updateSplashProgress };
}

// In app.whenReady()
app.whenReady().then(() => {
  const { splashWindow: splash, updateSplashStatus } = createSplashWindow();

  updateSplashStatus('Creating main window...');

  setTimeout(() => {
    createMainWindow();
    createMenu();
    setupIpcHandlers();

    // Close splash when main window is ready
    mainWindow.once('ready-to-show', () => {
      setTimeout(() => {
        if (splash && !splash.isDestroyed()) {
          splash.close();
        }
        mainWindow.show();
      }, 500); // Small delay for smooth transition
    });
  }, 1500);
});
```

12.13.8 JSON File Operations

Saving and loading JSON data is essential for app settings, user preferences, and state persistence:

Method 1: Using IPC Handlers (Recommended)

In main.js:

```
const fs = require('fs').promises;
const path = require('path');

// Get user data directory
const getUserDataPath = () => {
  return app.getPath('userData');
};

// Ensure directory exists
async function ensureDirectory(dirPath) {
  try {
    await fs.mkdir(dirPath, { recursive: true });
  } catch (error) {
    console.error('Error creating directory:', error);
  }
}

// IPC Handlers for JSON operations
ipcMain.handle('save-json', async (event, filename, data) => {
  try {
    const userDataPath = getUserDataPath();
    const filePath = path.join(userDataPath, filename);

    await ensureDirectory(path.dirname(filePath));
    await fs.writeFile(filePath, JSON.stringify(data, null, 2), 'utf-8');

    return { success: true, path: filePath };
  } catch (error) {
    console.error('Error saving JSON:', error);
    return { success: false, error: error.message };
  }
});

ipcMain.handle('load-json', async (event, filename) => {
  try {
    const userDataPath = getUserDataPath();
    const filePath = path.join(userDataPath, filename);

    const data = await fs.readFile(filePath, 'utf-8');
    return { success: true, data: JSON.parse(data) };
  } catch (error) {
    if (error.code === 'ENOENT') {
      // File doesn't exist, return default
      return { success: true, data: null };
    }
    console.error('Error loading JSON:', error);
    return { success: false, error: error.message };
  }
});

ipcMain.handle('delete-json', async (event, filename) => {
  try {
    const userDataPath = getUserDataPath();
    const filePath = path.join(userDataPath, filename);

    await fs.unlink(filePath);
    return { success: true };
  } catch (error) {
    if (error.code === 'ENOENT') {
      return { success: true }; // Already deleted
    }
    console.error('Error deleting JSON:', error);
    return { success: false, error: error.message };
  }
});

ipcMain.handle('list-json-files', async (event, directory = '') => {
  try {
    const userDataPath = getUserDataPath();
    const dirPath = path.join(userDataPath, directory);

    const files = await fs.readdir(dirPath);
    const jsonFiles = files.filter(file => file.endsWith('.json'));

    return { success: true, files: jsonFiles };
  } catch (error) {
    console.error('Error listing JSON files:', error);
    return { success: false, error: error.message };
  }
});
```

In preload.js:

```
contextBridge.exposeInMainWorld('electronAPI', {
  // JSON file operations
  saveJSON: async (filename, data) => {
    return await ipcRenderer.invoke('save-json', filename, data);
  },

  loadJSON: async (filename) => {
    return await ipcRenderer.invoke('load-json', filename);
  },
  deleteJSON: async (filename) => {
    return await ipcRenderer.invoke('delete-json', filename);
  },

  listJSONFiles: async (directory = '') => {
    return await ipcRenderer.invoke('list-json-files', directory);
  }
});
```

In your renderer (cables patch or HTML):

```
// Save settings
async function saveSettings(settings) {
  const result = await window.electronAPI.saveJSON('settings.json', settings);
  if (result.success) {
    console.log('Settings saved to:', result.path);
  } else {
    console.error('Failed to save settings:', result.error);
  }
}

// Load settings
async function loadSettings() {
  const result = await window.electronAPI.loadJSON('settings.json');
  if (result.success) {
    if (result.data) {
      console.log('Settings loaded:', result.data);
      return result.data;
    } else {
      // Return default settings
      return getDefaultSettings();
    }
  } else {
    console.error('Failed to load settings:', result.error);
    return getDefaultSettings();
  }
```

```
}

// Example usage with cables.gl patch
async function initializeApp() {
  // Load saved settings
  const settings = await loadSettings();

  // Apply settings to cables patch
  if (window.CABLES && window.CABLES.patch) {
    Object.keys(settings).forEach(key => {
      window.CABLES.patch.setVariable(key, settings[key]);
    });
  }

  // Listen for settings changes and auto-save
  if (window.CABLES && window.CABLES.patch) {
    window.CABLES.patch.on('variableChanged', async (name, value) => {
      const currentSettings = await loadSettings();
      currentSettings[name] = value;
      await saveSettings(currentSettings);
    });
  }
}

// Save cables patch state
async function savePatchState() {
  if (!window.CABLES || !window.CABLES.patch) return;

  const state = {
    timestamp: new Date().toISOString(),
    variables: {},
    camera: {
      position: window.CABLES.patch.cgl?.camera?.position || null,
      rotation: window.CABLES.patch.cgl?.camera?.rotation || null
    }
  };
  // Save all variables
  // (You'll need to track variable names or get them from your patch)
  const variableNames = ['color', 'speed', 'intensity']; // Your variable names
  variableNames.forEach(name => {
    state.variables[name] = window.CABLES.patch.getVariable(name);
  });

  await window.electronAPI.saveJSON('patch-state.json', state);
}

// Load patch state
async function loadPatchState() {
  const result = await window.electronAPI.loadJSON('patch-state.json');
  if (result.success && result.data) {
    const state = result.data;

    // Restore variables
    Object.keys(state.variables).forEach(name => {
      window.CABLES.patch.setVariable(name, state.variables[name]);
    });

    // Restore camera if available
    if (state.camera && window.CABLES.patch.cgl?.camera) {
      // Camera restoration depends on your cables setup
    }
  }
}
```

Method 2: Using electron-store (Simpler)

```
npm install electron-store
```

```
// In main.js
const Store = require('electron-store');

const store = new Store({
  name: 'app-settings',
  defaults: {
    theme: 'dark',
    windowState: {
      width: 1280,
      height: 720
    },
    cablesSettings: {
      color: [1, 0, 0, 1],
      speed: 1.0
    }
  }
});

// Expose store to renderer
ipcMain.handle('store-get', (event, key) => {
  return store.get(key);
});

ipcMain.handle('store-set', (event, key, value) => {
  store.set(key, value);
  return true;
});

ipcMain.handle('store-delete', (event, key) => {
  store.delete(key);
  return true;
});

ipcMain.handle('store-clear', () => {
  store.clear();
  return true;
});

ipcMain.handle('store-all', () => {
  return store.store;
});
```

```
// In preload.js
contextBridge.exposeInMainWorld('electronAPI', {
  store: {
    get: (key) => ipcRenderer.invoke('store-get', key),
    set: (key, value) => ipcRenderer.invoke('store-set', key, value),
    delete: (key) => ipcRenderer.invoke('store-delete', key),
    clear: () => ipcRenderer.invoke('store-clear'),
    all: () => ipcRenderer.invoke('store-all')
  }
});
```

```
// In renderer
// Get setting
const theme = await window.electronAPI.store.get('theme');

// Set setting
await window.electronAPI.store.set('cablesSettings.color', [0, 1, 0, 1]);

// Get all settings
const allSettings = await window.electronAPI.store.all();
```

12.13.9 Code Signing for Distribution

Code signing is essential for smooth app distribution on macOS and Windows. Unsigned apps trigger security warnings and may be blocked.

macOS Code Signing

Requirements: - Apple Developer Account ($99/year) - Valid code signing certificate - Notarization (required for macOS 10.15+)

package.json configuration:

```
{
  "build": {
    "appId": "com.yourcompany.cablesapp",
    "mac": {
      "icon": "assets/icon.icns",
      "category": "public.app-category.graphics-design",
      "target": [
        {
          "target": "dmg",
          "arch": ["x64", "arm64"]
        },
        {
          "target": "zip",
          "arch": ["x64", "arm64"]
        }
      ],
      "hardenedRuntime": true,
      "gatekeeperAssess": false,
      "entitlements": "build/mac/entitlements.mac.plist",
      "entitlementsInherit": "build/mac/entitlements.mac.plist"
    },
    "afterSign": "scripts/notarize.js",
    "notarize": {
      "teamId": "YOUR_TEAM_ID"
    }
  }
}
```

entitlements.mac.plist:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-
1.0.dtd">
<plist version="1.0">
<dict>
  <key>com.apple.security.cs.allow-jit</key>
  <true/>
  <key>com.apple.security.cs.allow-unsigned-executable-memory</key>
  <true/>
  <key>com.apple.security.cs.allow-dyld-environment-variables</key>
  <true/>
  <key>com.apple.security.cs.disable-library-validation</key>
  <true/>
</dict>
</plist>
```

scripts/notarize.js:

```
const { notarize } = require('@electron/notarize');

exports.default = async function notarizing(context) {
  const { electronPlatformName, appOutDir } = context;

  if (electronPlatformName !== 'darwin') {
    return;
  }

  const appName = context.packager.appInfo.productFilename;

  return await notarize({
    appBundleId: 'com.yourcompany.cablesapp',
    appPath: `${appOutDir}/${appName}.app`,
    appleId: process.env.APPLE_ID,
    appleIdPassword: process.env.APPLE_ID_PASSWORD,
    teamId: process.env.APPLE_TEAM_ID
  });
};
```

Environment variables (.env or export):

```
export APPLE_ID="your@email.com"
export APPLE_ID_PASSWORD="app-specific-password"
export APPLE_TEAM_ID="YOUR_TEAM_ID"
```

Build command:

```
npm run build:mac
```

Windows Code Signing

Requirements: - Code signing certificate (purchased from certificate authority) - Or use self-signed certificate for testing (not recommended for distribution)

package.json configuration:

```
{
  "build": {
    "win": {
      "icon": "assets/icon.ico",
      "target": [
        {
          "target": "nsis",
          "arch": ["x64", "ia32"]
        },
        {
          "target": "portable",
          "arch": ["x64"]
        }
      ],
      "signingHashAlgorithms": ["sha256"],
      "sign": "build/win/sign.js",
      "certificateFile": "path/to/certificate.pfx",
      "certificatePassword": "${env.CERTIFICATE_PASSWORD}"
    }
  }
}
```

build/win/sign.js:

```
const path = require('path');
```

```
exports.default = async function(configuration) {
  const { path: filePath } = configuration;

  // Only sign on Windows
  if (process.platform !== 'win32') {
    return;
  }

  // Use electron-builder's built-in signing
  // Or use signtool directly
  const { execSync } = require('child_process');

  const certPath = process.env.CERTIFICATE_PATH;
  const certPassword = process.env.CERTIFICATE_PASSWORD;

  if (!certPath || !certPassword) {
    console.warn('Certificate not configured, skipping signing');
    return;
  }

  try {
    execSync(
    `signtool sign /f "${certPath}" /p "${certPassword}" /t http://timestamp.digicert.com /d "My Cabl
      { stdio: 'inherit' }
    );
  } catch (error) {
    console.error('Signing failed:', error);
    throw error;
  }
};
```

Alternative: Using electron-builder's built-in signing:

```
{
  "build": {
    "win": {
      "certificateFile": "path/to/certificate.pfx",
      "certificatePassword": "${env.CERTIFICATE_PASSWORD}",
      "signingHashAlgorithms": ["sha256"],
      "signDlls": true
    }
  }
}
```

Build command:

```
npm run build:win
```

App Registration and Metadata

package.json - Complete build configuration:

```
{
  "name": "my-cables-app",
  "version": "1.0.0",
  "description": "My amazing Cables.gl application",
  "author": {
    "name": "Your Name",
    "email": "your@email.com"
  },
  "license": "MIT",
  "main": "main.js",
  "build": {
    "appId": "com.yourcompany.cablesapp",
    "productName": "My Cables App",
    "copyright": "Copyright © 2024 Your Company",
    "directories": {
      "output": "dist",
      "buildResources": "build"
    },
    "files": [
      "main.js",
      "preload.js",
      "renderer/**/*",
      "!renderer/**/*.map"
    ],
    "extraResources": [
      {
        "from": "assets/",
        "to": "assets/",
        "filter": ["**/*"]
      }
    ],
    "mac": {
      "icon": "assets/icon.icns",
      "category": "public.app-category.graphics-design",
      "minimumSystemVersion": "10.13",
      "darkModeSupport": true,
      "target": [
        {
          "target": "dmg",
          "arch": ["x64", "arm64"]
        }
      ],
      "hardenedRuntime": true,
      "entitlements": "build/mac/entitlements.mac.plist",
      "entitlementsInherit": "build/mac/entitlements.mac.plist"
    },
    "win": {
      "icon": "assets/icon.ico",
      "target": [
        {
          "target": "nsis",
          "arch": ["x64"]
        }
      ],
      "publisherName": "Your Company Name",
      "verifyUpdateCodeSignature": false
    },
    "linux": {
```

```
    "icon": "assets/icon.png",
    "target": [
      {
        "target": "AppImage",
        "arch": ["x64"]
      },
      {
        "target": "deb",
        "arch": ["x64"]
      }
    ],
    "category": "Graphics"
  },
  "nsis": {
    "oneClick": false,
    "allowToChangeInstallationDirectory": true,
    "createDesktopShortcut": true,
    "createStartMenuShortcut": true,
    "shortcutName": "My Cables App"
  },
  "dmg": {
    "title": "${productName} ${version}",
    "icon": "assets/icon.icns",
    "background": "build/mac/dmg-background.png",
    "contents": [
      {
        "x": 410,
        "y": 150,
        "type": "link",
        "path": "/Applications"
      },
      {
        "x": 130,
        "y": 150,
        "type": "file"
      }
      ],
      "window": {
        "width": 540,
        "height": 380
      }
    }
  }
}
```

12.13.10 Building and Distributing

Development Build

```
# Start in development mode
npm start
```

Production Build

```
# Build for current platform
npm run build

# Build for specific platforms
npm run build:mac
npm run build:win
npm run build:linux

# Build for all platforms (requires platform-specific tools)
npm run build:all
```

Distribution Checklist

Before Building: - [ ] Update version in package.json - [ ] Test app thoroughly - [ ] Optimize assets - [ ] Prepare code signing certificates - [ ] Set up environment variables - [ ] Test on target platforms

After Building: - [ ] Test installer on clean system - [ ] Verify code signing - [ ] Test auto-updater (if implemented) - [ ] Check file associations - [ ] Verify menu items work - [ ] Test file operations - [ ] Check window state persistence

12.13.11 Advanced Electron Features

Auto-Updater

```
npm install electron-updater
```

```
// In main.js
const { autoUpdater } = require('electron-updater');

autoUpdater.checkForUpdatesAndNotify();

autoUpdater.on('update-available', () => {
  dialog.showMessageBox(mainWindow, {
    type: 'info',
    title: 'Update Available',
    message: 'A new version is available. It will be downloaded in the background.',
    buttons: ['OK']
  });
});

autoUpdater.on('update-downloaded', () => {
  dialog.showMessageBox(mainWindow, {
    type: 'info',
    title: 'Update Ready',
    message: 'Update downloaded. The application will restart to apply the update.',
    buttons: ['Restart Now', 'Later']
  }).then(result => {
    if (result.response === 0) {
      autoUpdater.quitAndInstall();
    }
  });
});
```

System Tray

```
const { Tray, Menu } = require('electron');
const path = require('path');

let tray = null;

function createTray() {
  const iconPath = path.join(__dirname, 'assets', 'tray-icon.png');
  tray = new Tray(iconPath);

  const contextMenu = Menu.buildFromTemplate([
    {
      label: 'Show App',
      click: () => {
        mainWindow.show();
      }
    },
    {
      label: 'Quit',
      click: () => {
        app.quit();
      }
    }
  ]);

  tray.setToolTip('My Cables App');
  tray.setContextMenu(contextMenu);

  tray.on('click', () => {
    mainWindow.isVisible() ? mainWindow.hide() : mainWindow.show();
  });
}
```

Native Notifications

```
const { Notification } = require('electron');

```

```
function showNotification(title, body) {
  if (Notification.isSupported()) {
    new Notification({
      title: title,
      body: body,
      icon: getIconPath()
    }).show();
  }
}
```

12.13.12 Performance Optimization for Electron

1. **Disable Node Integration in Renderer** - Use contextBridge instead
2. **Enable Context Isolation** - Better security and performance
3. **Use Hardware Acceleration** - Enabled by default
4. **Optimize Asset Loading** - Lazy load when possible
5. **Throttle Background Processes** - Use `backgroundThrottling: true`
6. **Monitor Memory Usage** - Use DevTools memory profiler

12.13.13 Security Best Practices

1. **Never use `nodeIntegration: true`** - Use preload scripts instead
2. **Always use `contextIsolation: true`** - Isolates your code
3. **Validate all IPC messages** - Don't trust renderer input
4. **Use Content Security Policy** - Restrict resource loading
5. **Keep Electron updated** - Security patches are important
6. **Sanitize file paths** - Prevent directory traversal attacks

12.13.14 Troubleshooting Electron Issues

App won't start: - Check main.js for syntax errors - Verify all dependencies are installed - Check console for error messages

Window is blank: - Verify file paths are correct - Check DevTools for errors - Ensure renderer files are included in build

Code signing fails: - Verify certificate is valid - Check environment variables are set - Ensure certificate password is correct

App is slow: - Check for memory leaks - Optimize asset loading - Use performance profiling tools

12.14 Troubleshooting

12.14.1 Common Issues

"Assets not loading" - Check file paths are correct - Ensure CORS headers are set for cross-origin assets - Verify assets are included in export

"Blank screen" - Check browser console for errors - Verify all JavaScript files loaded - Test on a local server (not file://)

"Poor performance" - Reduce canvas resolution - Lower texture sizes - Simplify shaders - Check for memory leaks

"Works locally but not on server" - Check file paths (case-sensitive on Linux) - Verify all files uploaded - Check server MIME types

12.15 Featured Videos

```
https://youtu.be/hVxrxXhH7vQ
Title: Cables.gl Standalone (Offline) Build: Create Without Limits!
Author: Decode GL
Thumbnail: https://i.ytimg.com/vi/hVxrxXhH7vQ/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@Decode_gl
```

12.16 Exercises

1. Export a simple patch and host it on GitHub Pages
2. Embed a cables patch as a website background
3. Create a loading screen for your patch
4. Set up communication between your patch and external JavaScript
5. **Electron Exercise**: Package your cables.gl export as an Electron app with a custom splash screen
6. **Electron Exercise**: Implement JSON save/load functionality to persist your patch settings
7. **Electron Exercise**: Set up code signing for macOS or Windows (requires developer account/certificate)
8. **Electron Exercise**: Create a multi-window Electron app with inter-window communication

9. **Electron Exercise**: Implement window state persistence (save/restore window position and size)
10. **Electron Exercise**: Add a system tray icon with context menu for your Electron app

---

12.17 Congratulations!

You've completed the Cables.gl book! You now have the knowledge to:

- Create stunning 2D and 3D graphics
- Apply textures and materials
- Write custom shaders
- Build custom operators
- Create audio-reactive visuals
- Animate with timeline and code
- Export and deploy your creations

Keep experimenting, join the community, and share your creations!

Resources: - cables.gl - Official website - cables.gl/docs - Documentation - Discord - Community chat

13 Video Tutorials

This page is a **best-effort, heavily searched** index of cables.gl videos.

- **Source**: generated from multiple YouTube searches using `yt-dlp` (then de-duplicated and filtered).

13.1 Getting Started & Overviews

```
https://youtu.be/iXKo7mU422M
Title: Array from Numbers Operator tutorial - byte size
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/iXKo7mU422M/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/koLSrHFyIUY
Title: Arrays in cables - tutorial 03
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/koLSrHFyIUY/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/FRFfvVgWFcs
Title: Arrays in cables tutorial 01
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/FRFfvVgWFcs/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/F-CUdHq40Pc
Title: Basic material op tutorial - Byte size
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/F-CUdHq40Pc/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/EPFNHYah9F4
Title: cables gl introduction
```

```
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/EPFNHYah9F4/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/MOdVmJ6MYQE
Title: Creating your own cables.gl operators - custom and user ops tutorial
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/MOdVmJ6MYQE/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/EzV5CRAMyTA
Title: Depth texture op tutorial - Byte size
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/EzV5CRAMyTA/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/knGnukutZeM
Title: Lights and Shadows Operators - getting started - Video Tutorial
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/knGnukutZeM/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/7xlElfbMWgw
Title: MeshInstancer tutorial 01
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/7xlElfbMWgw/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/XvVBnPakE28
Title: Midi Input Device - intro to MIDI in cables -  Byte Size
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/XvVBnPakE28/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/Ds4fPcxyBvM
Title: Noise Texture Operator for generating color palettes for various design techniques -
Video Tutorial
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/Ds4fPcxyBvM/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/P6esDOFHM6w
Title: Particle system in cables tutorial 01
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/P6esDOFHM6w/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/Nre7LH0OVw4
Title: Particle system in cables tutorial 02
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/Nre7LH0OVw4/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/x2jKZgmFVq4
Title: Post processing tutorial for beginners
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/x2jKZgmFVq4/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/z1Qf9dE67-w
Title: Text Texture op tutorial - Byte size
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/z1Qf9dE67-w/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/mQN8VtVOltQ
Title: Texture2ColorArray op tutorial
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/mQN8VtVOltQ/mqdefault.jpg
```

```
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/wzpKR7vbCXg
Title: Timeline - Part 1: Overview
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/wzpKR7vbCXg/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/SaKWF6RnsyI
Title: Transform Vertex Operator tutorial (GPU vs CPU based animation) - byte size
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/SaKWF6RnsyI/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/B9GyRzov5Bg
Title: tutorial demo effect / render2textures world position target tricks
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/B9GyRzov5Bg/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/T0djoWQkBew
Title: Cables.GL: Introduction
Author: Creative Tech Talks
Thumbnail: https://i.ytimg.com/vi/T0djoWQkBew/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@webtvmaster
```

```
https://youtu.be/sbML3B3Vu4g
Title: Cables.GL: Tutorial
Author: Creative Tech Talks
Thumbnail: https://i.ytimg.com/vi/sbML3B3Vu4g/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@webtvmaster
```

```
https://youtu.be/kgXpXsLtv1M
Title: Assets (6/13) - Intro to Cables.gl
Author: Decode GL
```

```
Thumbnail: https://i.ytimg.com/vi/kgXpXsLtv1M/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@Decode_gl
```

```
https://youtu.be/vzWrCGfU7uw
Title: Control Flow (3/13) - Intro to Cables.gl
Author: Decode GL
Thumbnail: https://i.ytimg.com/vi/vzWrCGfU7uw/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@Decode_gl
```

```
https://youtu.be/2YFB4MuN8y8
Title: Data Types (2/13) - Intro to Cables.gl
Author: Decode GL
Thumbnail: https://i.ytimg.com/vi/2YFB4MuN8y8/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@Decode_gl
```

```
https://youtu.be/Z4gReZ34SHU
Title: Interactions (5/13) - Intro to Cables.gl
Author: Decode GL
Thumbnail: https://i.ytimg.com/vi/Z4gReZ34SHU/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@Decode_gl
```

```
https://youtu.be/VsS4gaJ7pMw
Title: Introduction to Cables.gl (1/13)
Author: Decode GL
Thumbnail: https://i.ytimg.com/vi/VsS4gaJ7pMw/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@Decode_gl
```

```
https://youtu.be/RhbId-kUWig
Title: Texture Effects (8/13) - Intro to Cables.gl
Author: Decode GL
Thumbnail: https://i.ytimg.com/vi/RhbId-kUWig/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@Decode_gl
```

```
https://youtu.be/qEno30S8CBc
Title: Glitch Art Tutorial using Cables.gl
```

```
Author: Jaalibandar
Thumbnail: https://i.ytimg.com/vi/qEno30S8CBc/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@jaalibandar
```

```
https://youtu.be/goO3PhuenBI
Title: First Steps in Cables.gl - Tutorial
Author: The Interactive & Immersive HQ
Thumbnail: https://i.ytimg.com/vi/goO3PhuenBI/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@TheInteractiveImmersiveHQ
```

13.2 Core Concepts & Workflow

```
https://youtu.be/lj6REnNZU0s
Title: converter ops
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/lj6REnNZU0s/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/M1A8S98UOuI
Title: how to reroute cables #gui #uxdesign #motiondesign
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/M1A8S98UOuI/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/ZCKrhswQiyc
Title: you can cut cables with the [Y] key #animation #motiondesign #design #web #3danimation
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/ZCKrhswQiyc/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/xawlfxKpxRQ
Title: you can replace cables that easy #animation #motiondesign #design #web
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/xawlfxKpxRQ/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/GQc6JF_jy6M
Title: Debug View in Cables.gl | Setting up multiple views in your patch
Author: Jaalibandar
Thumbnail: https://i.ytimg.com/vi/GQc6JF_jy6M/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@jaalibandar
```

```
https://youtu.be/uzqplBUGMWg
Title: 01 Jam Sessions : Generative Fluid Graphic in Cables.gl
Author: FahmiMursyid
Thumbnail: https://i.ytimg.com/vi/uzqplBUGMWg/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@fahmimursyid
```

```
https://youtu.be/wERboDg6zOI
Title: Impactful Transitions under 10 minutes using cables.gl | Genuary 04: Intersections
Author: Jaalibandar
Thumbnail: https://i.ytimg.com/vi/wERboDg6zOI/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@jaalibandar
```

```
https://youtu.be/_CltN9uQhoU
Title: Procedurally generated plants in Cables.gl #genuary
Author: Jaalibandar
Thumbnail: https://i.ytimg.com/vi/_CltN9uQhoU/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@jaalibandar
```

```
https://youtu.be/5Jc3woVozNc
Title: Cables.gl | Generative Poster 05
Author: Karthik Dondeti
Thumbnail: https://i.ytimg.com/vi/5Jc3woVozNc/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@karthikdondeti1672
```

```
https://youtu.be/DsSPcNSLyAw
Title: Cables.gl | Generative Poster 06
Author: Karthik Dondeti
Thumbnail: https://i.ytimg.com/vi/DsSPcNSLyAw/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@karthikdondeti1672
```

```
https://youtu.be/gRV0DqpSd-4
Title: Cables.gl | Generative Poster 09
Author: Karthik Dondeti
Thumbnail: https://i.ytimg.com/vi/gRV0DqpSd-4/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@karthikdondeti1672
```

```
https://youtu.be/3tZQtsEiicw
Title: February 2022 Release Chat - cables.gl updated - PBR, Geometry from Textures, Teams, EXR support
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/3tZQtsEiicw/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/a56wk9Xm9dY
Title: Using Vertex Displacement with Normal maps in cables.gl
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/a56wk9Xm9dY/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/NjG85Qbbl0w
Title: Vertex displacement op - byte size
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/NjG85Qbbl0w/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/lOMplXy_JV0
Title: Visualize any YouTube playlist in 3D with n8n.io & cables.gl (part 1)
Author: Decode GL
Thumbnail: https://i.ytimg.com/vi/lOMplXy_JV0/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@Decode_gl
```

```
https://youtu.be/AZrWNl3MwHQ
Title: Scrolling Terrain with UFO in 10 minutes using cables.gl
Author: Jaalibandar
Thumbnail: https://i.ytimg.com/vi/AZrWNl3MwHQ/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@jaalibandar
```

```
https://youtu.be/sbqE83ZHiTU
Title: Scrolling Terrain with UFO in 10 minutes using cables.gl
Author: Jaalibandar
Thumbnail: https://i.ytimg.com/vi/sbqE83ZHiTU/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@jaalibandar
```

```
https://youtu.be/3KSS1nrv6t0
Title: cables.gl web demo - realtime visualizer soundcloud globe | Exyl - Ping! Moai
Author: stobelights
Thumbnail: https://i.ytimg.com/vi/3KSS1nrv6t0/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@stobelights
```

13.3 3D / 3D Meshes

```
https://youtu.be/iqIXSb-kAws
Title: Importing GLTF 3D Scenes with Camera positions and animating them in cables.gl
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/iqIXSb-kAws/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/I_eD5nml_5A
Title: More GLTF operators - animated rig support, position data, separate animation timing -
Byte Size
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/I_eD5nml_5A/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/DW9U5tv1GHM
Title: Varying Mesh Instances with color, animation and textures - Video Tutorial
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/DW9U5tv1GHM/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/PrkdnENo8wQ
Title: Vertex Textures - Point Clouds and Mesh Instancing from Textures - Introduction
```

```
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/PrkdnENo8wQ/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

13.4 Textures / Post-Processing

```
https://youtu.be/uwoj7R52yU8
Title: PBR Material & PBR Environment Light Op - Byte Size - Physically Based Rendering in Cables
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/uwoj7R52yU8/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/Yf84KQc9jzU
Title: Copy Texture operator deep dive - basics and use cases
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/Yf84KQc9jzU/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/cc5Vlmvlq6A
Title: Pixel displace op - byte size
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/cc5Vlmvlq6A/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/rtDA2S9SPQ4
Title: Exploring Matcap Creator by bagoof - a new tool made with cables
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/rtDA2S9SPQ4/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

13.5 Shaders / Shadertoy / GLSL

```
https://youtu.be/Zfhn8xSM0SE
Title: Coding with cables - custom shader op
Author: cables_gl
```

```
Thumbnail: https://i.ytimg.com/vi/Zfhn8xSM0SE/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/j_ins4RW0c8
Title: Shadertoy to cables - part  01
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/j_ins4RW0c8/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/nil-HkZgNZ8
Title: Programmation d'un shadertoy avec Cables.gl Partie 8.
Author: Meletou1
Thumbnail: https://i.ytimg.com/vi/nil-HkZgNZ8/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@Meletou1
```

13.6 Audio / Music / MIDI

```
https://youtu.be/SFXvtm-vkvE
Title: Introduction to Generative Music and Audio Reactive Systems with Cables.gl
Author: Jaalibandar
Thumbnail: https://i.ytimg.com/vi/SFXvtm-vkvE/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@jaalibandar
```

```
https://youtu.be/h20ZH-xD8Ts
Title: Microphone Input & Audio Reactivity in Cables.gl - Tutorial
Author: The Interactive & Immersive HQ
Thumbnail: https://i.ytimg.com/vi/h20ZH-xD8Ts/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@TheInteractiveImmersiveHQ
```

```
https://youtu.be/uYk7-9dZ8Ys
Title: MidiFighter cables.gl Vjing
Author: Alberto Barrios L. (nahui-ocelotl.com)
Thumbnail: https://i.ytimg.com/vi/uYk7-9dZ8Ys/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@4183RtdF
```

```
https://youtu.be/KZbhVCIahv4
Title: Páginas WEB Interactivas con cables.gl | 13 Audio
Author: Alberto Barrios L. (nahui-ocelotl.com)
Thumbnail: https://i.ytimg.com/vi/KZbhVCIahv4/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@4183RtdF
```

```
https://youtu.be/3m-2F2T1f6w
Title: Audio analyzer op - audio reactive
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/3m-2F2T1f6w/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/68iSILnuLnA
Title: BiQuadFilter op- audio reactive tut
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/68iSILnuLnA/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/eDlaFD_d5lc
Title: Connecting Midi controllers to Cables
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/eDlaFD_d5lc/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/wKQN2BZPtyU
Title: Exploring Spatial Audio in Cables.gl
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/wKQN2BZPtyU/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/3owzsIzvkdQ
Title: Let's make some noise! Building a drum machine with Cables.gl.
Author: Kirell Benzi
Thumbnail: https://i.ytimg.com/vi/3owzsIzvkdQ/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@KirellB
```

```
https://youtu.be/KtREXHa9tS8
Title: Programmation Cables.gl Audio Analyzer Partie 7.
Author: Meletou1
Thumbnail: https://i.ytimg.com/vi/KtREXHa9tS8/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@Meletou1
```

```
https://youtu.be/TyEIawM-ll0
Title: Syncing Cables.gl with Bitwig Studio
Author: Stefan Sauer
Thumbnail: https://i.ytimg.com/vi/TyEIawM-ll0/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@StefanSauer
```

```
https://youtu.be/TlDHrXS06-A
Title: [animatic] Better! // bitwig studio, cables.gl
Author: vozh-kc
Thumbnail: https://i.ytimg.com/vi/TlDHrXS06-A/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@vozh-kc
```

```
https://youtu.be/S-KyCySVucM
Title: [HD] Sidereal Collapse // cables.gl, Bitwig Studio
Author: vozh-kc
Thumbnail: https://i.ytimg.com/vi/S-KyCySVucM/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@vozh-kc
```

13.7 Physics

```
https://youtu.be/hlmNf_42raY
Title: AmmoRaycast Operator - creating a simple 3D menu UI - Tutorial
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/hlmNf_42raY/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/TAhAqgY_EEs
Title: AmmoWorld and AmmoBody Operators - physics simulations in cables.gl -
Video Tutorial
```

```
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/TAhAqgY_EEs/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

13.8 Export / Deployment / Embedding

```
https://youtu.be/DX0slSkR_Hg
Title: Páginas WEB Interactivas con cables.gl | 20 Exportación
Author: Alberto Barrios L. (nahui-ocelotl.com)
Thumbnail: https://i.ytimg.com/vi/DX0slSkR_Hg/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@4183RtdF
```

```
https://youtu.be/J8yJtcd1Jeg
Title: CABLES Command Line export
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/J8yJtcd1Jeg/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/YUAyS_NcwTA
Title: embed a cables patch into a html website
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/YUAyS_NcwTA/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/B4M9FddXk1I
Title: Exporting your Project - .zip Export - Byte Size
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/B4M9FddXk1I/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/L5BGMs7vKuI
Title: Exporting your Project - Netlify export - Byte Size
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/L5BGMs7vKuI/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/hVxrxXhH7vQ
Title: Cables.gl Standalone (Offline) Build: Create Without Limits!
Author: Decode GL
Thumbnail: https://i.ytimg.com/vi/hVxrxXhH7vQ/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@Decode_gl
```

13.9 Hardware / External Tools

```
https://youtu.be/vebGfUp9vJ4
Title: Getting cables.gl to talk to hardware, using Chataigne!
Author: Rob Duarte
Thumbnail: https://i.ytimg.com/vi/vebGfUp9vJ4/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@RobDuarte
```

```
https://youtu.be/4YsuGFAEvEE
Title: Phidget Encoder in cables.gl
Author: wirmachenbunt
Thumbnail: https://i.ytimg.com/vi/4YsuGFAEvEE/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@wirmachenbunt
```

13.10 Talks / Meetups / Release Notes

```
https://youtu.be/FvC3Ec_38Jo
Title: Inércia 2023 | Seminar: Cables.gl as a demo making tool by anticore feat. liqube
Author: Associação Inércia
Thumbnail: https://i.ytimg.com/vi/FvC3Ec_38Jo/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@inercia_pt
```

```
https://youtu.be/xLBLo6O1kXg
Title: cables.gl october meetup
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/xLBLo6O1kXg/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/xRbg1Az0k8k
Title: November Update - cables.gl monthly meetup
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/xRbg1Az0k8k/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/C2FjpdRWPxw
Title: Updated Physically Based Rendering Operators - discussion with the developer AMajesticSeaFlapF
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/C2FjpdRWPxw/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/v4rYqHuT-0E
Title: Seminar: Making demos with cables.gl (speaker: pandur)
Author: psenough
Thumbnail: https://i.ytimg.com/vi/v4rYqHuT-0E/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@psenough
```

13.11 Showcases / Demos / Visualizers

```
https://youtu.be/oLPsJd0e4Gc
Title: antonymph - vylet pony (avoset remix; cables.gl visualiser)
Author: avoset
Thumbnail: https://i.ytimg.com/vi/oLPsJd0e4Gc/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@avoset
```

```
https://youtu.be/CfPJZMAxcTU
Title: Lines / Live experience with Cables.gl
Author: BoatBoat_Station
Thumbnail: https://i.ytimg.com/vi/CfPJZMAxcTU/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@Boatboat_station
```

```
https://youtu.be/Zr_7wRBmRmA
Title: Building a VJ patch mixer with cables.gl
Author: cables_gl
```

```
Thumbnail: https://i.ytimg.com/vi/Zr_7wRBmRmA/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/84pXsmJghdM
Title: demomaking with cables
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/84pXsmJghdM/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/M8Is131LSzE
Title: hydra - demo by mfx
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/M8Is131LSzE/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/R9-D4SxBd90
Title: Ninja de Gaia - Inércia 2023 - creating a demo with cables.gl
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/R9-D4SxBd90/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/auvD8oSxMew
Title: cables.gl - Drifting Apart (FXHash Project)
Author: Creative Exploration /w Purz
Thumbnail: https://i.ytimg.com/vi/auvD8oSxMew/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@PurzBeats
```

```
https://youtu.be/9vZzrXX_2jM
Title: cables.gl - purzOS - Low Poly Lavalamp (FXHash Project)
Author: Creative Exploration /w Purz
Thumbnail: https://i.ytimg.com/vi/9vZzrXX_2jM/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@PurzBeats
```

```
https://youtu.be/a0lJ8DF-v8o
Title: cables.gl - purzOS - Ring Worlds (Screensaver)
```

```
Author: Creative Exploration /w Purz
Thumbnail: https://i.ytimg.com/vi/a0lJ8DF-v8o/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@PurzBeats
```

```
https://youtu.be/EO3UdeBQ9m0
Title: EroLogo - Visual Demo Lenght 12:37 made with Cables.gl
Author: faktisProductions
Thumbnail: https://i.ytimg.com/vi/EO3UdeBQ9m0/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@faktisProductions
```

```
https://youtu.be/xba3e91Fum4
Title: Design Designs Design - "Smorp" (A Cables.gl demo for Evoke 2022)
Author: Jan-Jozef Tuigstra
Thumbnail: https://i.ytimg.com/vi/xba3e91Fum4/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@jan-jozeftuigstra2218
```

13.12 Unsorted (Still cables.gl-related)

```
https://youtu.be/1FqBKJ1RXdY
Title: Entornos virtuales WEB con programación visual en cables.gl Parte 1
Author: Alberto Barrios L. (nahui-ocelotl.com)
Thumbnail: https://i.ytimg.com/vi/1FqBKJ1RXdY/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@4183RtdF
```

```
https://youtu.be/oBoH_7uHv-E
Title: Páginas WEB Interactivas con cables.gl | 02 Enlace
Author: Alberto Barrios L. (nahui-ocelotl.com)
Thumbnail: https://i.ytimg.com/vi/oBoH_7uHv-E/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@4183RtdF
```

```
https://youtu.be/-9QrZSoAPpQ
Title: Páginas WEB Interactivas con cables.gl | 07 Botones
Author: Alberto Barrios L. (nahui-ocelotl.com)
Thumbnail: https://i.ytimg.com/vi/-9QrZSoAPpQ/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@4183RtdF
```

```
https://youtu.be/MTeI06T-kGw
Title: Páginas WEB Interactivas con cables.gl | 08 Menu
Author: Alberto Barrios L. (nahui-ocelotl.com)
Thumbnail: https://i.ytimg.com/vi/MTeI06T-kGw/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@4183RtdF
```

```
https://youtu.be/iFDD4tm7-Uw
Title: Páginas WEB Interactivas con cables.gl | 15 Valores Aleatorios
Author: Alberto Barrios L. (nahui-ocelotl.com)
Thumbnail: https://i.ytimg.com/vi/iFDD4tm7-Uw/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@4183RtdF
```

```
https://youtu.be/a2H8vk3Ko1M
Title: Páginas WEB Interactivas con cables.gl | 18 FPS
Author: Alberto Barrios L. (nahui-ocelotl.com)
Thumbnail: https://i.ytimg.com/vi/a2H8vk3Ko1M/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@4183RtdF
```

```
https://youtu.be/cVpC9IS6kI0
Title: Substitution Pattern / Testing / CABLES.GL /
Author: Antiguo Autómata Mexicano
Thumbnail: https://i.ytimg.com/vi/cVpC9IS6kI0/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@antiguoautomata
```

```
https://youtu.be/7BiDxNc7D7g
Title: Create awesome Visuals using OpenDAW and cables.gl!
Author: BeatMax_Prediction
Thumbnail: https://i.ytimg.com/vi/7BiDxNc7D7g/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@BeatMax2023
```

```
https://youtu.be/omIK1YOtb70
Title: Copy cat with cables- live stream - Inconvergent
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/omIK1YOtb70/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/tu49qg8BpBU
Title: copy cat(s) with cables live stream - Junkiyoshi
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/tu49qg8BpBU/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/Gr3iVMUs_hA
Title: Copycat with Cables - Tyler hobbs - Untitled
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/Gr3iVMUs_hA/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/hZQZsh5UHSE
Title: did you know, you can add multiple ops one go
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/hZQZsh5UHSE/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/jiOLZaMUH78
Title: Repeat op tut 01 - Byte size
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/jiOLZaMUH78/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/00Rvb749wrc
Title: Smooth Operator - Byte Size
Author: cables_gl
Thumbnail: https://i.ytimg.com/vi/00Rvb749wrc/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@cablesgl
```

```
https://youtu.be/8LfR8iLLbMA
Title: Infinite Looping Motion Graphic in 10 minutes using cables.gl
Author: Jaalibandar
Thumbnail: https://i.ytimg.com/vi/8LfR8iLLbMA/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@jaalibandar
```

```
https://youtu.be/WGoM1AmfW7g
Title: Getting data from an API with cables.gl - data-driven gradient from geo-located weather - part 1
Author: Kirell Benzi
Thumbnail: https://i.ytimg.com/vi/WGoM1AmfW7g/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@KirellB
```

```
https://youtu.be/G1HKysL8iVw
Title: Présentation du logiciel Cables.gl par les étudiants en UI/UX design
Author: L'École de design Nantes Atlantique
Thumbnail: https://i.ytimg.com/vi/G1HKysL8iVw/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@lecolenantes
```

```
https://youtu.be/4Op74uIzH5c
Title: Retour sur le programme Cables.gl
Author: Meletou1
Thumbnail: https://i.ytimg.com/vi/4Op74uIzH5c/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@Meletou1
```

```
https://youtu.be/tdbTTxDu7Qk
Title: Cables.gl
Author: Nathan Sonzogni
Thumbnail: https://i.ytimg.com/vi/tdbTTxDu7Qk/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@nathansonzogni
```

```
https://youtu.be/n4UPiZhbcRU
Title: StableDiffusion and ControlNet in Cables.gl via the WebUI
Author: Neight Allen
Thumbnail: https://i.ytimg.com/vi/n4UPiZhbcRU/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@CzechNeight
```

```
https://youtu.be/lImv9ZJshUE
Title: cables.gl and ollama API
Author: Tobias Hartmann
Thumbnail: https://i.ytimg.com/vi/lImv9ZJshUE/mqdefault.jpg
```

```
AuthorUrl: https://www.youtube.com/@tobiashartmann
```

```
https://youtu.be/vOVKpppw1ds
Title: Class 30: Learning how to mint a cables.gl patch on fx hash w/ Somaticbits
Author: VERTICAL
Thumbnail: https://i.ytimg.com/vi/vOVKpppw1ds/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@verticalcrypto
```

```
https://youtu.be/4PsWzWHsiV4
Title: cables.gl ink spill
Author: Video Art Duo
Thumbnail: https://i.ytimg.com/vi/4PsWzWHsiV4/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@videoartduo
```

```
https://youtu.be/9UR8upg0g54
Title: Pod005 - Flicker | Distortion | cables.gl
Author: zuggamasta
Thumbnail: https://i.ytimg.com/vi/9UR8upg0g54/mqdefault.jpg
AuthorUrl: https://www.youtube.com/@zuggamasta
```