Procedural

UI PRO

Complete Documentation & Runtime API Reference

Draw any UI shape. No sprites. No shaders. Just code and creativity.

Version 1.0 | Unity 2021.3+ | URP / Built-in / HDRP

Table of Contents

1. What is Procedural UI Pro?
2. Quick Start (5 Minutes)
3. Shape Types at a Glance
4. The Inspector — Module by Module
   4.1 Shape Settings
   4.2 Fill
   4.3 Outline & Stroke
   4.4 Effects (Glow, Shadow, Bevel, Blur, 3D Rotation)
   4.5 Mask
   4.6 Animation
5. Runtime Scripting API
   5.1 Getting Started with Code
   5.2 Changing Shapes & Colors
   5.3 Controlling Effects
   5.4 Playing Animations
   5.5 Style Presets
   5.6 Custom Shape Generators
6. Performance Guide
7. Common Recipes
8. Troubleshooting & FAQ

1. What is Procedural UI Pro?

Procedural UI Pro lets you build resolution-independent UI elements entirely from code-generated meshes. No sprite atlases to maintain. No shader graphs to untangle. Drop a single component onto any RectTransform and pick from 56 built-in shapes — rounded rectangles, stars, arrows, health bars, speech bubbles, gears, and more.

The Problem It Solves

Traditional Unity UI relies on pre-made sprites. Every new button corner radius, every icon variation, every different-sized panel means another texture in your atlas. Multiply that across screen sizes and color themes, and you end up with bloated build sizes, tedious asset pipelines, and designers waiting on artists for every small tweak.

How It Works

Procedural UI Pro generates triangle meshes directly on the CPU using Unity's VertexHelper system. It plugs into the standard Canvas rendering pipeline — same batching, same masking, same layout system you already know. Each shape is defined by a lightweight generator class that turns a few parameters (corner radius, star points, arrow thickness) into vertices and triangles every time the RectTransform changes.

On top of the shape mesh, a modular pipeline layers fill (solid, gradient, texture, pattern), outline (solid, dashed, with gradient/texture support), and effects (glow, shadow, bevel, blur, 3D rotation, masking). Everything is vertex-based — no extra draw calls, no render textures, no custom shaders required.

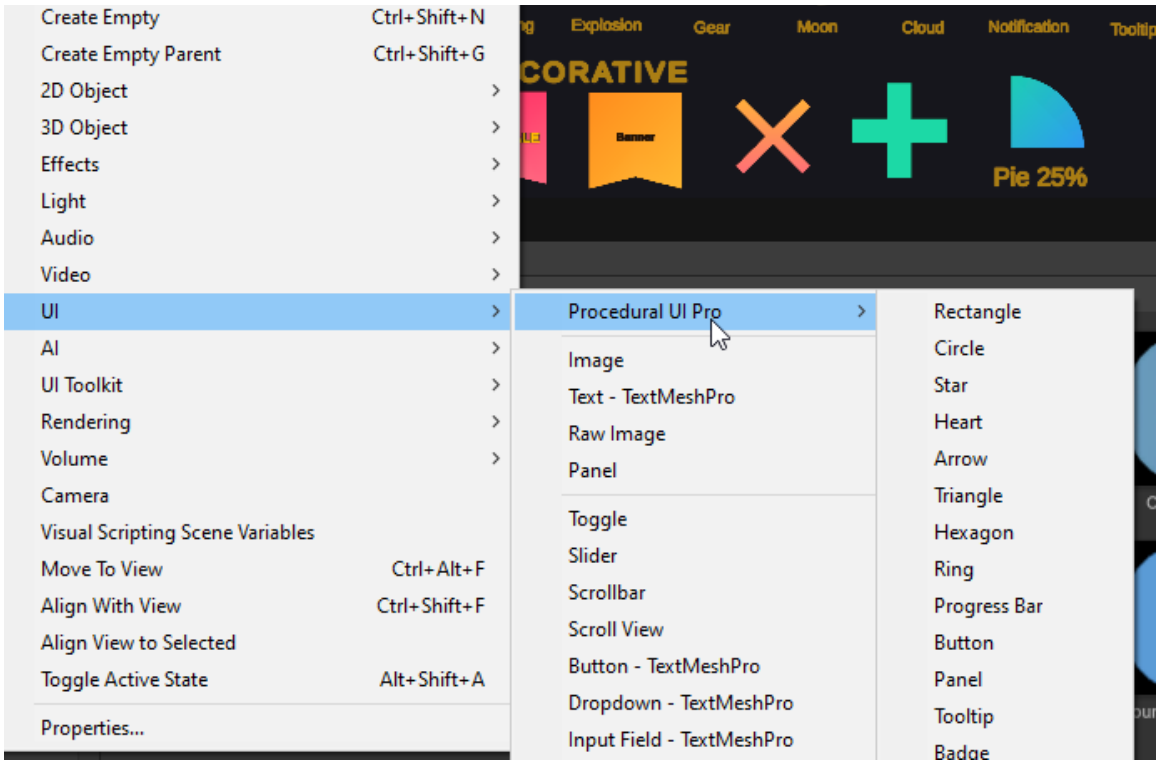
Key Features

- **56 shape** types across 5 categories (Basic, Directional, Game UI, Advanced, Custom)
- **6 fill modes**: Solid, Gradient (5 types), Texture, Sprite, Pattern (6 types)
- **Full outline system** with 6 stroke types, gradient/texture strokes, join & cap control
- Glow, Shadow (outer + inner), Bevel, Blur, 3D Rotation effects
- Soft Mask, Gradient Mask, Alpha Mask for advanced compositing
- **26 animation** presets with looping, easing curves, and speed control
- Style presets — save and reuse looks across your project
- **HDR color support** on every color field
- **Per-corner radius** on rounded rectangles
- Mesh quality levels (Low / Medium / High / Ultra) for performance tuning
- Works with Unity UI Mask, Layout Groups, and Canvas Scaler
- **Zero dependencies**. Pure C#. No shaders to import.

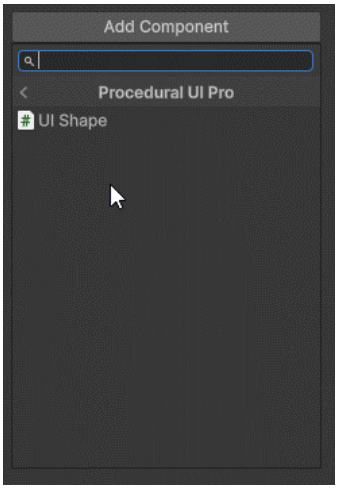
2. Quick Start (5 Minutes)

Step 1 — Add the Component

Right-click in your Hierarchy inside a Canvas, then go to UI > Procedural UI Pro > UI Shape. This creates a new GameObject with the UIShape component already attached.

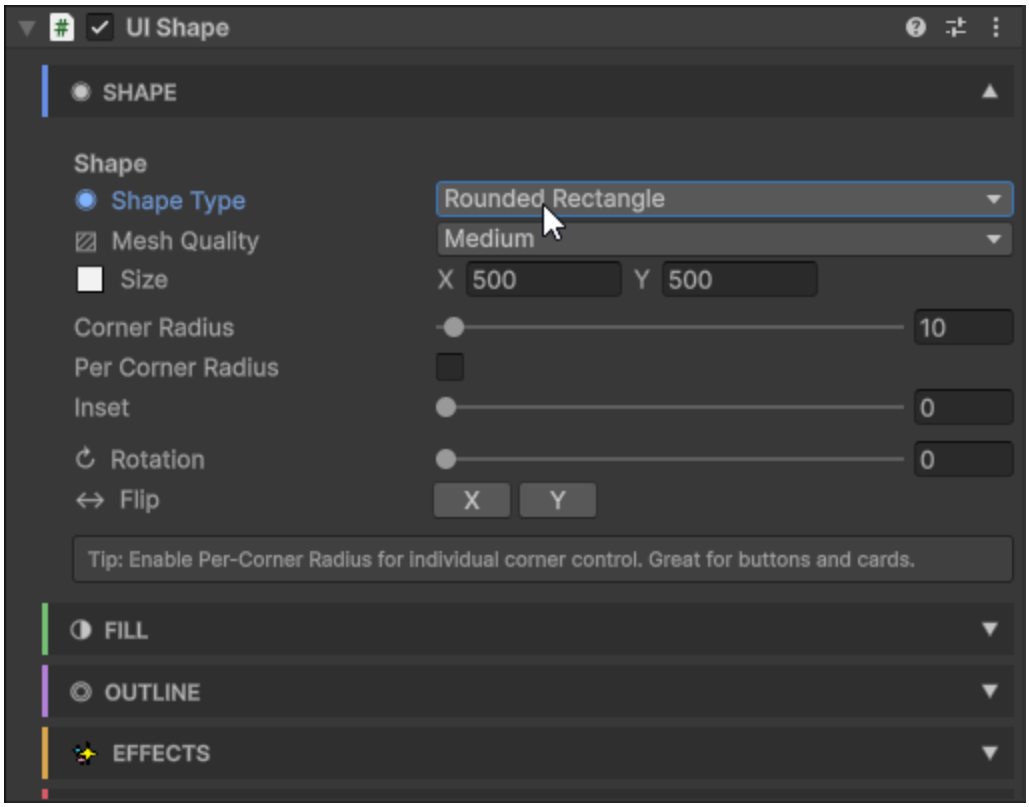


Alternatively, select any existing RectTransform and use Add Component > UI > Procedural UI Pro > UI Shape.



Step 2 — Pick a Shape

In the Inspector, open the Shape dropdown and choose one of the 56 built-in shapes. The preview updates immediately in the Scene view. Each shape type reveals its own set of parameters below the dropdown — corner radius for rectangles, point count for stars, tail size for speech bubbles, and so on.



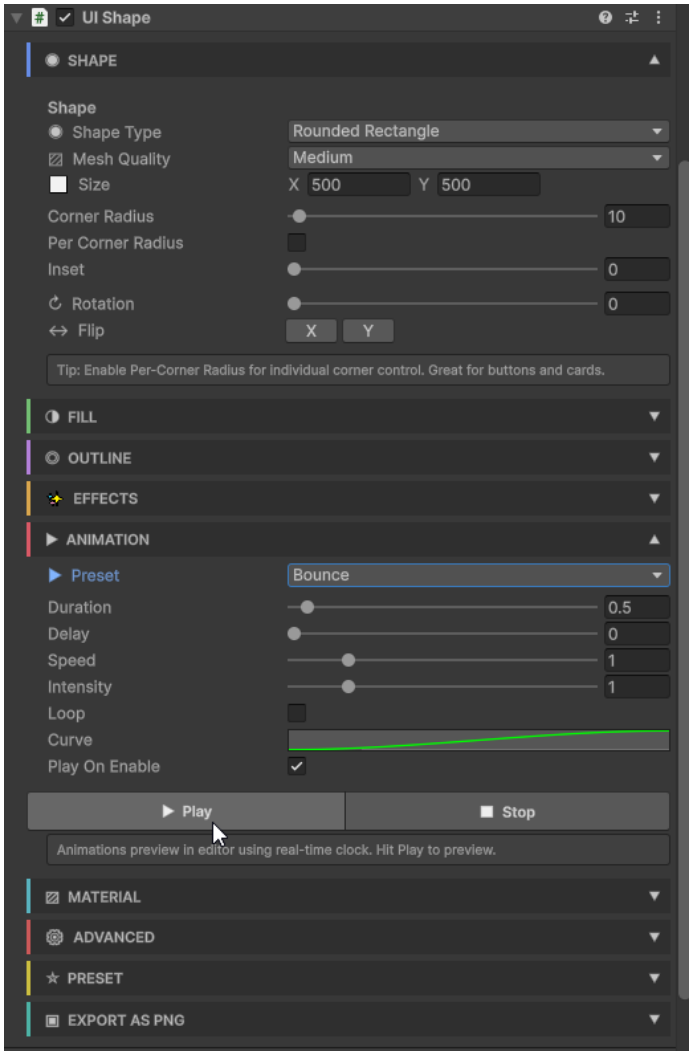
Step 3 — Style It

Expand the Fill section to pick a color, gradient, or texture. Toggle on the Outline to add a border. Flip on Glow or Shadow in the Effects section. Everything is live — tweak sliders and watch the shape update in real time.

Step 4 — Animate (Optional)

Open the Animation section and pick a preset like Pop, Bounce, or Heartbeat. Check "Play On Enable" and the shape will animate as soon as it appears. Or trigger it from code with shape.PlayAnimation().

Tip: Hold Alt while dragging a slider to get finer control. Right-click any color field to copy/paste hex values.



3. Shape Types at a Glance

Procedural UI Pro ships with 56 shapes organized into five categories. Every shape works with every fill mode, every outline style, and every effect.

Basic Shapes

Shape	Key Parameters	Best For
Rectangle	Inset	Panels, backgrounds, cards
RoundedRectangle	Corner radius (uniform or per-corner)	Buttons, cards, modals — the workhorse
Circle	Segments	Icons, avatars, radio buttons
Ellipse	Segments	Decorative elements, badges
Square	—	Grid cells, thumbnails
Triangle	Rotation	Directional hints, play buttons
Diamond	—	Decorative accents, card suits
Pentagon	—	Badges, labels
Hexagon	—	Honeycomb layouts, strategy game tiles
Octagon	—	Stop signs, industrial UI
Star	Points, inner/outer radius	Ratings, rewards, decorations
Heart	—	Health, favorites, love indicators
Cross / Plus	Arm thickness	Medical UI, add buttons
Line	Thickness	Dividers, connectors
Capsule	Corner radius	Pills, tags, search bars
SemiCircle	—	Gauges, decorative
Arc	Start/end angle	Progress rings, dials
Ring	Inner radius	Loading indicators, donuts
PieShape	Fill amount	Charts, skill cooldowns

Directional Shapes

Shape	Key Parameters	Best For
ArrowLeft / Right / Up / Down	Head width, head height, body thickness	Navigation, indicators
Chevron	Angle, thickness	Breadcrumbs, pagination
DoubleArrow	Same as Arrow	Resize handles, bidirectional flow
CurvedArrow	Same as Arrow	Undo/redo icons, tutorials

Game UI Shapes

Shape	Key Parameters	Best For
HealthBar	Fill amount, corner radius	HP bars, stamina, XP bars
ProgressBar	Fill amount, corner radius	Loading, download progress
Badge	Corner radius	Notification counts, labels
Tag	Corner radius	Category labels, filters
TooltipBubble	Tail size, tail position	Tooltips, popups
DialogBubble	Tail size, tail position	Chat bubbles, speech
ButtonShape	Corner radius	Styled buttons
PanelShape	Corner radius	Content containers
NotificationShape	Corner radius	Alerts, toasts

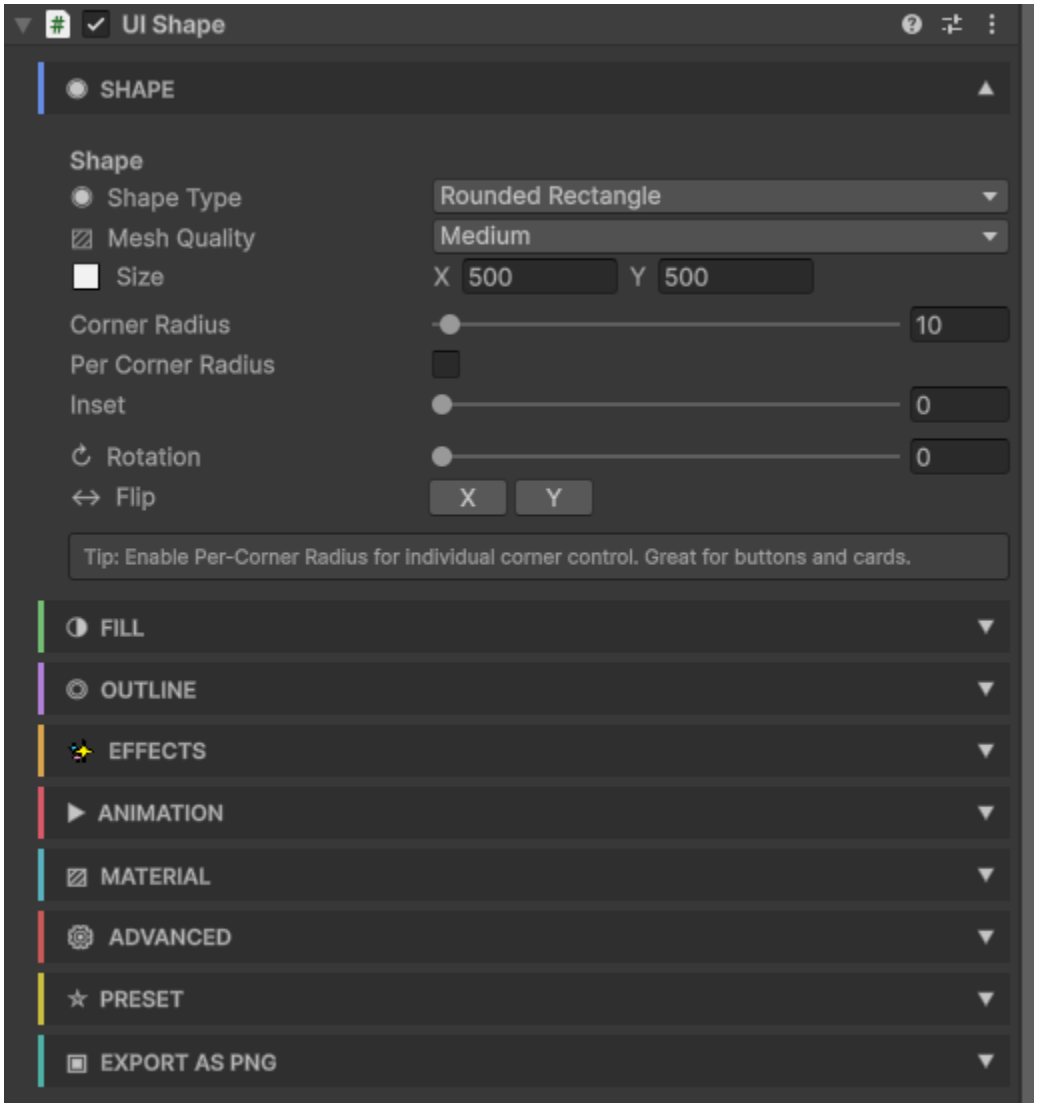
Advanced Shapes

Shape	Key Parameters	Best For
Cloud	—	Weather UI, thought bubbles
Lightning	—	Power-ups, electrical effects
Gear	Teeth count, inner radius	Settings icons, mechanical UI
Ribbon / Banner	—	Achievement banners, labels
Moon	Phase	Day/night cycles, theme toggles
Explosion	Spike count	Impact effects, sale badges
CustomPolygon	Vertex list	Anything you can imagine

4. The Inspector — Module by Module

4.1 Shape Settings

The top section of the inspector controls the shape type and its geometry parameters.



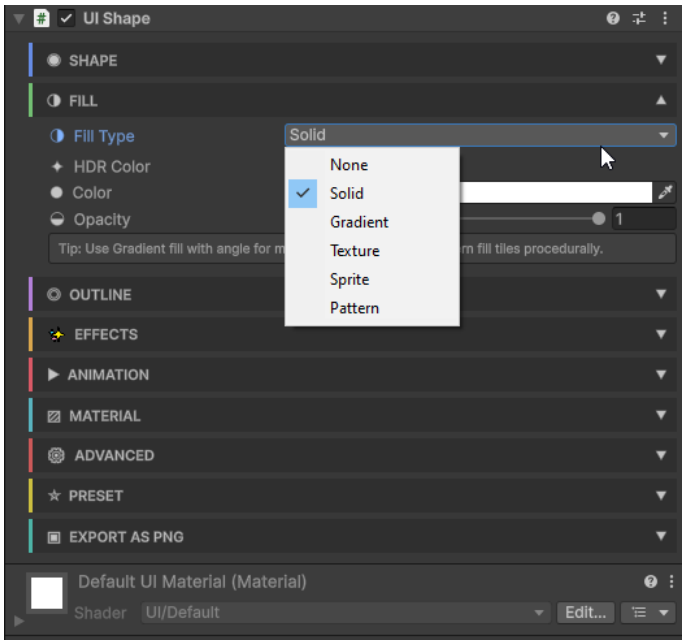
Parameter	Range	What It Does
Shape Type	56 options	Selects the base shape geometry
Mesh Quality	Low / Medium / High / Ultra	Controls curve smoothness vs. vertex count
Corner Radius	0 – 500	Rounds the corners (rectangles, panels, bars)
Per-Corner Radius	Toggle	Enables individual control of each corner
Segments	3 – 256	Subdivision count for circles, arcs, rings
Rotation	0 – 360	Rotates the shape inside its rect

Flip X / Flip Y	Toggle	Mirrors the shape horizontally or vertically

Tip: Start with Medium mesh quality. Only bump to High or Ultra for large hero elements where you can see jagged curves. On mobile, Low is often fine.

4.2 Fill

Fill controls what the inside of the shape looks like.



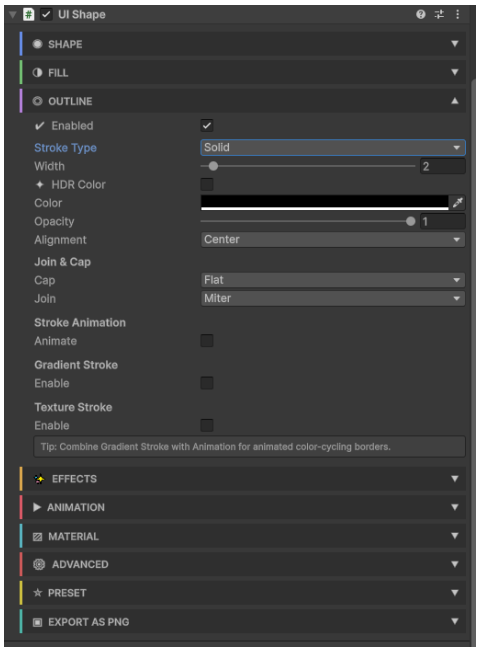
Fill Type	Description
None	Transparent interior — useful for outline-only shapes
Solid	Single flat color. Toggle HDR for bloom-ready intensity values.
Gradient	Blend between colors. Five gradient types: Linear, Radial, Angular, Diamond, Reflected. Control angle, offset, and scale.
Texture	Map any Texture2D onto the shape. Supports Stretch, Fit, Fill, Repeat, and Mirror modes. Control UV scale, offset, and rotation.
Sprite	Uses a Sprite asset, respecting its borders and pivot.

Pattern	Procedural patterns: Dots, Diagonal Lines, Grid, Noise, Zigzag, CrossHatch. Two-color system with adjustable scale.

Every fill type has an Opacity slider (0–1) that fades the entire fill without touching your color values. The HDR Color toggle switches the color picker to HDR mode, letting you push intensity above 1.0 for bloom and emissive effects.

4.3 Outline & Stroke

The outline module draws a border around your shape. It supports both simple expanded-shape outlines and advanced contour-ring strokes with gradient and texture mapping.



Parameter	Options / Range	Notes
Stroke Type	Solid, Dashed, Dotted, DashDot, LongDash, Custom	Controls dash pattern
Width	0 – 50 px	Stroke thickness
Alignment	Inside, Center, Outside	Where the stroke sits relative to the shape edge
Cap Type	Flat, Round, Square	How dash endpoints are drawn
Join Type	Miter, Round, Bevel	How corners are joined
Dash Length	1 – 50 px	Length of each dash segment
Gap Length	1 – 50 px	Space between dashes

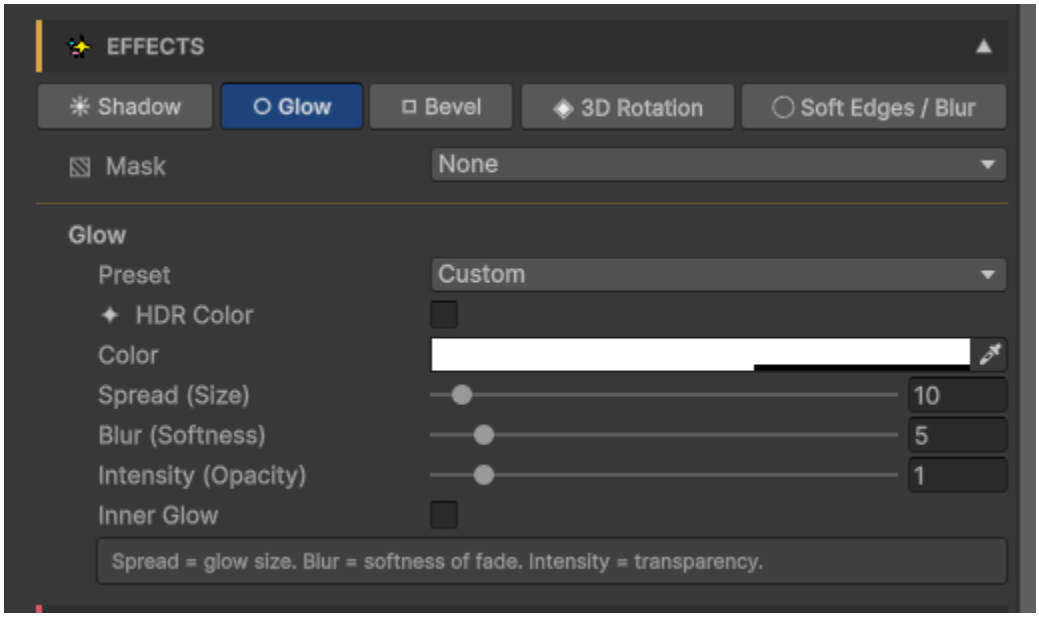
Gradient Stroke	Toggle + Gradient	Colors the stroke with a gradient along its length
Texture Stroke	Toggle + Texture	Maps a texture along the stroke with tiling control
Animate Stroke	Toggle + Progress 0–1	Reveals the stroke progressively (great for drawing effects)

Note: Gradient and Texture strokes use a contour-ring approach instead of the simple expanded-shape method. This means they work correctly on every shape, including compound shapes like speech bubbles.

4.4 Effects

Glow

Adds a colored aura around the shape. Built from concentric expanded copies with alpha falloff.

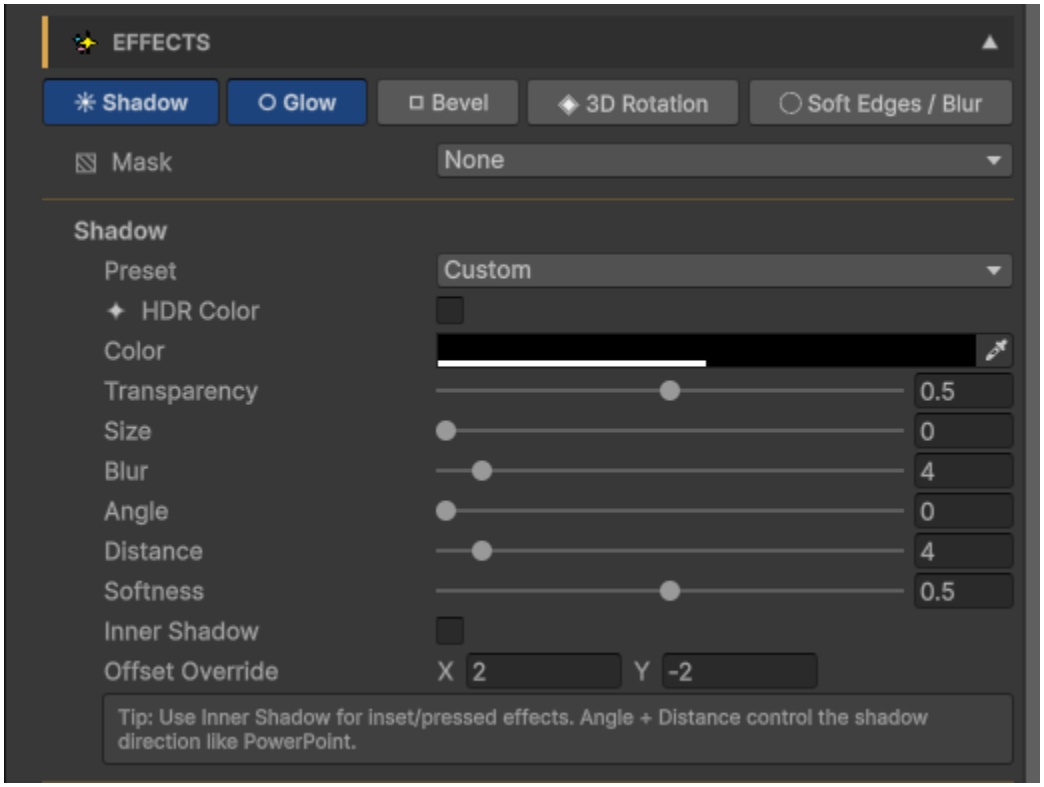


Parameter	Range	What It Controls
Spread	0 – 200	How far the glow extends from the shape edge (size)

Blur	0 – 50	How gradually the glow fades — low values give a tight halo, high values give a soft cloud
Intensity	0 – 10	Opacity multiplier — crank it up for neon, keep it low for subtle ambiance
Inner Glow	Toggle	Renders the glow inside the shape instead of outside

Shadow

PowerPoint-style shadow system with 15 presets covering outer, inner, and perspective shadows.

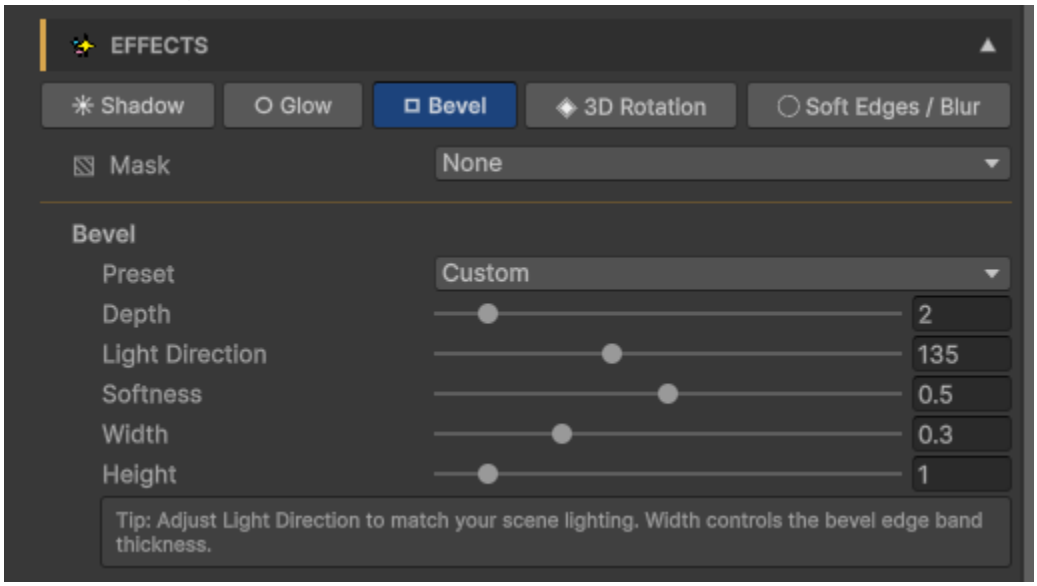


Parameter	Range	What It Controls
Angle	0 – 360	Direction the shadow falls (0 = right, 90 = up, 270 = below)
Distance	0 – 50	How far the shadow is offset from the shape
Blur	0 – 50	Shadow softness / feathering
Size	0 – 50	Extra expansion beyond the shape silhouette

Opacity	0 – 1	Shadow transparency
Softness	0 – 1	Edge falloff intensity
Inner Shadow	Toggle	Renders shadow inside the shape (inset shadow)

Bevel

Simulates 3D surface relief using contour-distance-based lighting. Unlike simple edge tinting, this computes actual surface normals from each vertex's distance to the nearest shape edge, then applies Lambertian shading against a configurable light direction. Works correctly on every shape — circles, stars, arrows, speech bubbles — not just rectangles.



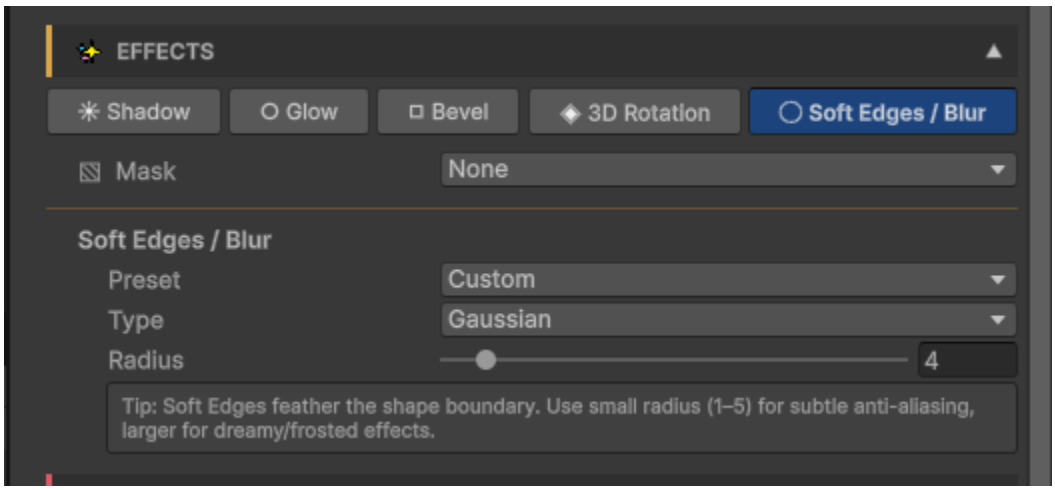
Parameter	Range	What It Controls
Depth	0 – 20	How pronounced the 3D relief looks
Light Direction	0 – 360	Angle of the virtual light source
Softness	0 – 1	Bevel profile: 0 = sharp hard edge, 1 = smooth rounded edge
Width	0.05 – 1	How wide the beveled rim extends inward
Height	0 – 10	Intensity multiplier for the lighting effect

Preset	Character

Circle	Soft, rounded rim — the classic look
RelaxedInset	Gentle, subtle depth
Cross	Sharp, defined edges
CoolSlant	Angled, modern feel
HardEdge	Extreme, industrial depth
ArtDeco	Elegant, decorative bevel

Blur

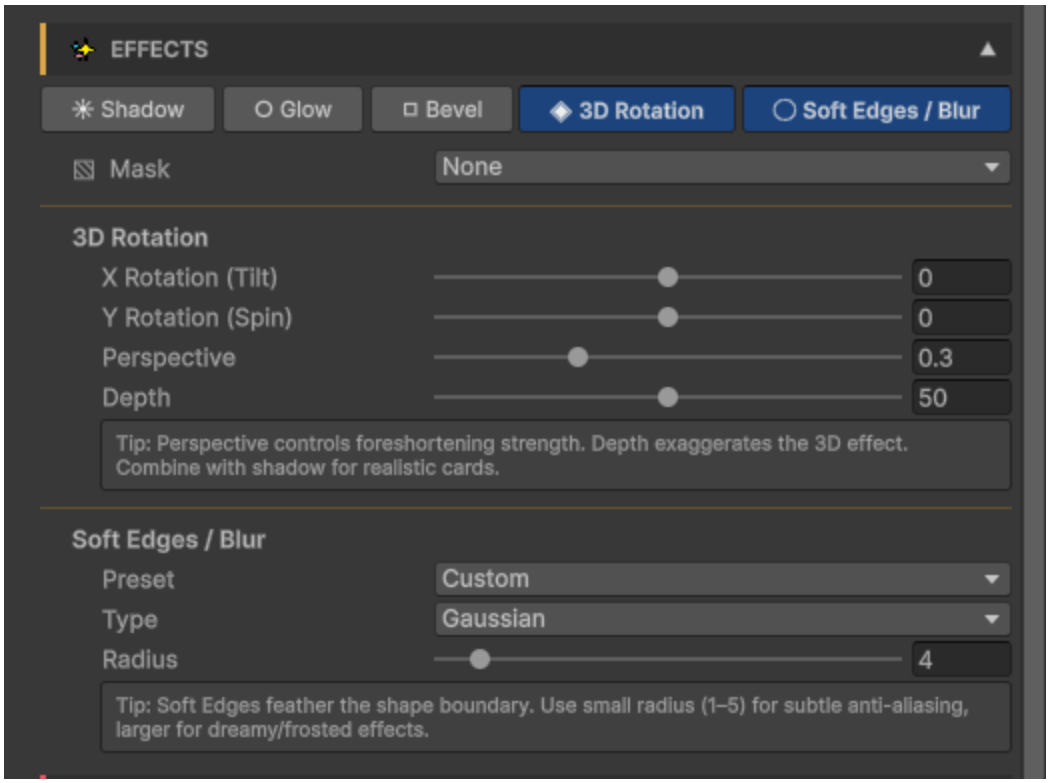
Softens the shape edges with a blur effect. Three algorithms available.



Type	Description
Gaussian	Uniform soft blur in all directions — the most common choice
Motion	Directional blur along a specified angle (great for speed effects)
Background	Reserved for future frosted-glass implementation

3D Rotation

Applies perspective rotation to the shape mesh, creating the illusion of a card or panel tilted in 3D space. This is a vertex transform, not a shader effect — it works with batching.



Parameter	Range	What It Controls
Rotation X	-180 – 180	Vertical tilt (like tilting a card forward/backward)
Rotation Y	-180 – 180	Horizontal spin (like turning a page)
Perspective	0 – 1	Foreshortening strength (0 = flat, 1 = dramatic)
Depth	0 – 100	Exaggeration factor (50 = natural)

4.5 Mask

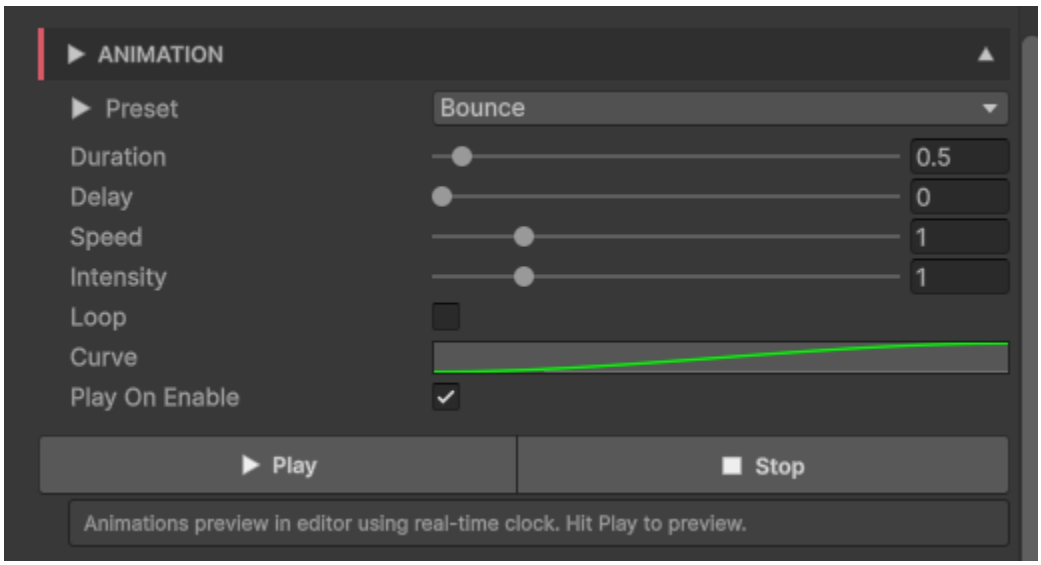
Masks are vertex-level alpha modifications applied as a post-process to the shape layers (fill + outline + edge AA + inner shadow). They do not affect the outer shadow or glow, which render behind the shape.

Mask Type	Parameter	Description
SoftMask	Edge Width (1–100 px)	Fades the shape edges to transparent. Great for vignette-style fading.

GradientMask	Direction Angle (0–360)	Applies a directional alpha gradient across the shape. Use for reveal effects.
AlphaMask	—	Uses the shape's own vertex alpha — plug in your own values from code.

4.6 Animation

26 built-in animation presets organized into five categories. Each preset modifies specific properties (scale, position, rotation, opacity, or fill) over time using the configured easing curve.



Category	Presets	What Changes
Fade	FadeIn, FadeOut, Blink	Fill opacity
Scale	Pop, Heartbeat, Bounce, Pulse, PingPong	Transform scale
Movement	Float, Shake, Slide, Hover, Bob	Transform position
Attention	Wiggle, Elastic, Jelly, Flash	Scale + rotation (varies)
Loading	Spinner, Rotate, CircularFill, Breathing	Rotation or pieFill

Setting	Range	Notes
Duration	0.05 – 10 sec	One full cycle length

Delay	0 – 5 sec	Wait before starting
Loop	Toggle	Repeat continuously
Speed	0.1 – 5x	Playback speed multiplier
Intensity	0.1 – 5x	How dramatic the motion is
Play On Enable	Toggle	Auto-start when the object activates
Curve	AnimationCurve	Custom easing (default: ease-in-out)

5. Runtime Scripting API

Everything you see in the Inspector is accessible from C# at runtime. The namespace is ProceduralUIPro. Add it at the top of your script:

```
using ProceduralUIPro;
```

5.1 Getting Started with Code

Grab a reference to the UIShape component like any other Unity component:

```
// From the same GameObject
UIShape shape = GetComponent<UIShape>();

// From a child or reference
[SerializeField] UIShape myButton;
```

After changing any property, call MarkDirty() to trigger a mesh rebuild on the next frame. The convenience setters (FillColor, CornerRadius) do this automatically.

5.2 Changing Shapes & Colors

Switch Shape Type

```
shape.ShapeType = ShapeType.Star;
shape.ShapeData.starPoints = 6;
shape.ShapeData.starInnerRadius = 0.4f;
shape.MarkDirty(DirtyFlags.Mesh);
```

Set Fill Color (Quick Way)

```
// Convenience property — auto-marks dirty
shape.FillColor = Color.red;
```

Set Fill Color (Full Control)

```
shape.FillData.fillType = FillType.Solid;
shape.FillData.color = new Color(0.2f, 0.6f, 1.0f);
shape.FillData.opacity = 0.8f;
shape.SetFillDirty();
```

Gradient Fill

```
shape.FillData.fillType = FillType.Gradient;
shape.FillData.gradientType = GradientType.Linear;
shape.FillData.gradientAngle = 45f;

var gradient = new Gradient();
gradient.SetKeys(
    new[] {
        new GradientColorKey(Color.blue, 0f),
        new GradientColorKey(Color.cyan, 1f)
    },
    new[] {
        new GradientAlphaKey(1f, 0f),
        new GradientAlphaKey(1f, 1f)
    }
);
shape.FillData.gradient = gradient;
shape.SetFillDirty();
```

Corner Radius

```
// Uniform corners
shape.CornerRadius = 20f;

// Per-corner (like CSS border-radius)
shape.ShapeData.perCornerRadius = true;
shape.ShapeData.cornerRadiusTL = 30f;  // top-left
shape.ShapeData.cornerRadiusTR = 30f;  // top-right
shape.ShapeData.cornerRadiusBL = 0f;   // bottom-left
shape.ShapeData.cornerRadiusBR = 0f;   // bottom-right
shape.MarkDirty(DirtyFlags.Mesh);
```

Health/Progress Bars

```
// Animate a health bar
shape.ShapeType = ShapeType.HealthBar;
shape.ShapeData.barFillAmount = currentHP / maxHP;
shape.MarkDirty(DirtyFlags.Mesh);
```

5.3 Controlling Effects

Enable Glow

```
shape.EffectData.glow.enabled = true;
shape.EffectData.glow.color = Color.yellow;
shape.EffectData.glow.spread = 15f;
shape.EffectData.glow.blur = 8f;
shape.EffectData.glow.intensity = 1.5f;
shape.SetEffectsDirty();
```

Drop Shadow

```
shape.EffectData.shadow.enabled = true;
shape.EffectData.shadow.color = new Color(0, 0, 0, 0.5f);
shape.EffectData.shadow.angle = 315f;    // bottom-right
shape.EffectData.shadow.distance = 6f;
shape.EffectData.shadow.blur = 8f;
shape.SetEffectsDirty();


// Or use a preset:
shape.EffectData.shadow.preset = ShadowPreset.OuterOffsetDiagonal;
shape.EffectData.shadow.ApplyPreset();
shape.SetEffectsDirty();
```

Inner Shadow

```
shape.EffectData.shadow.enabled = true;
shape.EffectData.shadow.innerShadow = true;
shape.EffectData.shadow.preset = ShadowPreset.InnerOffsetDiagonal;
shape.EffectData.shadow.ApplyPreset();
shape.SetEffectsDirty();
```

Bevel (3D Relief)

```
shape.EffectData.bevel.enabled = true;
shape.EffectData.bevel.preset = BevelPreset.SoftRound;
```

```
shape.EffectData.bevel.ApplyPreset();
shape.EffectData.bevel.lightDirection = 135f;
shape.SetEffectsDirty();
```

Blur

```
shape.EffectData.blur.enabled = true;
shape.EffectData.blur.blurType = BlurType.Gaussian;
shape.EffectData.blur.radius = 10f;
shape.SetEffectsDirty();
```

3D Rotation

```
shape.EffectData.rotation3D.enabled = true;
shape.EffectData.rotation3D.rotationX = 15f;
shape.EffectData.rotation3D.rotationY = -10f;
shape.EffectData.rotation3D.perspective = 0.4f;
shape.SetEffectsDirty();
```

Masking

```
// Soft edge fade
shape.EffectData.maskType = MaskType.SoftMask;
shape.EffectData.softMaskWidth = 30f;
shape.SetEffectsDirty();

// Directional fade (great for reveals)
shape.EffectData.maskType = MaskType.GradientMask;
shape.EffectData.gradientMaskAngle = 90f;  // fade from bottom to top
shape.SetEffectsDirty();
```

Outline

```
shape.OutlineData.enabled = true;
shape.OutlineData.width = 3f;
shape.OutlineData.color = Color.white;
shape.OutlineData.strokeType = StrokeType.Dashed;
shape.OutlineData.dashLength = 10f;
shape.OutlineData.gapLength = 5f;
shape.OutlineData.capType = CapType.Round;
shape.OutlineData.joinType = JoinType.Round;
shape.SetOutlineDirty();
```

Animated Stroke Reveal

```
// Draw the outline progressively over 2 seconds
shape.OutlineData.enabled = true;
shape.OutlineData.animateStroke = true;

IEnumerator RevealStroke()
{
   float duration = 2f;
   float elapsed = 0f;
   while (elapsed < duration)
   {
      elapsed += Time.deltaTime;
      shape.OutlineData.strokeProgress = elapsed / duration;
      shape.SetOutlineDirty();
      yield return null;
   }
}
```

5.4 Playing Animations

One-Shot Animation

```
shape.AnimationData.preset = AnimationPresetType.Pop;
shape.AnimationData.duration = 0.3f;
shape.AnimationData.loop = false;
shape.PlayAnimation();
```

Looping Animation

```
shape.AnimationData.preset = AnimationPresetType.Heartbeat;
shape.AnimationData.duration = 1f;
shape.AnimationData.speed = 1.2f;
shape.AnimationData.loop = true;
shape.PlayAnimation();

// Later, when done:
shape.StopAnimation();
```

Loading Spinner

```
shape.ShapeType = ShapeType.Arc;
shape.ShapeData.arcStartAngle = 0f;
```

```
shape.ShapeData.arcEndAngle = 270f;
shape.FillColor = Color.cyan;

shape.AnimationData.preset = AnimationPresetType.Spinner;
shape.AnimationData.speed = 2f;
shape.AnimationData.loop = true;
shape.PlayAnimation();
```

Custom Easing Curve

```
shape.AnimationData.curve = AnimationCurve.EaseInOut(0, 0, 1, 1);
// Or create a bouncy overshoot:
var keys = new Keyframe[] {
   new Keyframe(0f, 0f),
   new Keyframe(0.6f, 1.1f),
   new Keyframe(0.8f, 0.95f),
   new Keyframe(1f, 1f)
};
shape.AnimationData.curve = new AnimationCurve(keys);
```

Circular Fill Animation (Cooldown / Radial Wipe)

```
// Perfect for ability cooldowns
shape.ShapeType = ShapeType.PieShape;
shape.AnimationData.preset = AnimationPresetType.CircularFill;
shape.AnimationData.duration = 5f;  // 5-second cooldown
shape.AnimationData.loop = false;
shape.PlayAnimation();
```

5.5 Style Presets

Style presets are ScriptableObject assets that store a complete snapshot of shape settings. Use them to maintain consistent looks across your project, or to swap themes at runtime.

Create a Preset in the Editor

Right-click in the Project window > Create > Procedural UI Pro > Style Preset. Then design your shape in the Inspector and click "Capture" in the preset asset to save it.

Apply a Preset at Runtime

```
// Load from Resources folder
UIStylePreset preset = Resources.Load<UIStylePreset>("Styles/DangerButton");
shape.ApplyPreset(preset);
```

```
// Or reference it directly
[SerializeField] UIStylePreset hoverStyle;
[SerializeField] UIStylePreset normalStyle;

public void OnPointerEnter() => shape.ApplyPreset(hoverStyle);
public void OnPointerExit() => shape.ApplyPreset(normalStyle);
```

Save a Preset from Code (Editor Only)

```
#if UNITY_EDITOR
var preset = ScriptableObject.CreateInstance<UIStylePreset>();
preset.CaptureFrom(shape);
UnityEditor.AssetDatabase.CreateAsset(preset, "Assets/Presets/MyStyle.asset");
UnityEditor.AssetDatabase.SaveAssets();
#endif
```

5.6 Custom Shape Generators

You can extend Procedural UI Pro with your own shape types. Implement the IShapeGenerator interface and tag your class with the ShapeGeneratorAttribute to register it automatically.

```
using UnityEngine;
using UnityEngine.UI;
using ProceduralUIPro;

[ShapeGenerator(ShapeType.CustomPolygon)]
public class MyCustomShape : IShapeGenerator
{
    public void GenerateMesh(VertexHelper vh, Rect rect, UIShapeData data)
    {
        // Don't call vh.Clear() — the pipeline handles that.
        // Build your mesh using vh.AddVert() and vh.AddTriangle().

        Vector2 center = rect.center;
        float w = rect.width * 0.5f;
        float h = rect.height * 0.5f;

        // Example: simple diamond
        vh.AddVert(new Vector3(center.x, rect.yMax), Color.white, Vector2.zero);
```

```
        vh.AddVert(new Vector3(rect.xMax, center.y), Color.white, Vector2.zero);
        vh.AddVert(new Vector3(center.x, rect.yMin), Color.white, Vector2.zero);
        vh.AddVert(new Vector3(rect.xMin, center.y), Color.white, Vector2.zero);

        vh.AddTriangle(0, 1, 2);
        vh.AddTriangle(0, 2, 3);
    }
}
```

Note: Your generator class must be stateless — the factory caches a single instance and reuses it across all shapes of that type. All per-shape configuration should come from UIShapeData fields.

5.7 Full API Quick Reference

All public members on the UIShape class at a glance:

Member	Type	Access	Description
ShapeType	ShapeType	get / set	Current shape. Setting it auto-marks dirty.
ShapeData	UIShapeData	get	All geometry parameters (radius, segments, etc.)
FillData	FillData	get	Fill settings (type, color, gradient, texture, pattern)
OutlineData	OutlineData	get	Outline settings (stroke, width, caps, joins)
EffectData	EffectData	get	All effects (glow, shadow, bevel, blur, 3D, mask)
AnimationData	UIAnimationData	get	Animation preset and timing settings
StylePreset	UIStylePreset	get	Currently linked style preset asset
FillColor	Color	get / set	Shortcut for FillData.color + auto dirty
CornerRadius	float	get / set	Shortcut for ShapeData.cornerRadius + auto dirty
PlayAnimation()	void	method	Starts animation (adds Animator if needed)
StopAnimation()	void	method	Stops animation, restores original state
ApplyPreset(preset)	void	method	Applies a UIStylePreset to this shape
MarkDirty(flags)	void	method	Triggers rebuild for specified systems
SetFillDirty()	void	method	Marks fill + mesh for rebuild
SetOutlineDirty()	void	method	Marks outline + mesh for rebuild

SetEffectsDirty()	void	method	Marks effects + mesh for rebuild
GetMeshStats()	(int, int)	method	Returns (vertex count, triangle count)

Dirty Flags

```
DirtyFlags.Mesh       // Shape geometry changed
DirtyFlags.Fill       // Fill color, gradient, or texture changed
DirtyFlags.Outline    // Outline properties changed
DirtyFlags.Effects    // Any effect toggled or adjusted
DirtyFlags.Animation  // Animation settings changed
DirtyFlags.All        // Everything
```

6. Performance Guide

How It Batches

Procedural UI Pro uses Unity's standard UI batching. Shapes that share the same material (which is the default — all shapes use the default UI material unless overridden) and the same texture will batch together automatically. This means 50 solid-colored shapes on screen might result in just 1 draw call.

Vertex Counts

Each effect adds vertices. Here's a rough guide to help you budget:

Feature	Approximate Vertex Cost
Simple rectangle	4 vertices
Rounded rectangle (Medium quality)	40–80 vertices
Circle (64 segments)	65 vertices
Outline (solid)	2x the shape vertices
Glow (default spread)	4–20 rings, each = shape vertex count
Shadow	1x shape vertices (offset copy)
Edge anti-aliasing	2x perimeter vertices
Bevel	0 extra (modifies existing fill vertices)
3D Rotation	0 extra (transforms existing vertices)

Optimization Checklist

- Use Low or Medium mesh quality unless the shape is large and has visible curves.
- Disable effects you're not using. Even a disabled glow with spread > 0 costs nothing — the early-out check catches it.
- Reduce glow spread on small shapes. A 200px spread on a 50px icon is wasted work.
- Use solid outlines instead of gradient/texture strokes when the visual difference doesn't matter — the expanded-shape method is cheaper than contour-ring.
- Share style presets instead of duplicating settings. This doesn't affect performance directly but prevents accidental complexity.
- Profile with the built-in mesh stats display in the Inspector's debug section. It shows vertex and triangle counts for each shape.
- On mobile, stick to Low mesh quality and keep glow spread under 30. The vertex count from multiple glow rings adds up fast.

7. Common Recipes

Neon Button

```
var shape = GetComponent<UIShape>();
shape.ShapeType = ShapeType.RoundedRectangle;
shape.CornerRadius = 12f;
shape.FillData.fillType = FillType.Solid;
shape.FillData.color = new Color(0.1f, 0.1f, 0.15f);  // dark background
shape.FillData.useHDR = true;

shape.OutlineData.enabled = true;
shape.OutlineData.width = 2f;
shape.OutlineData.color = new Color(0f, 2f, 1f);  // HDR cyan
shape.OutlineData.useHDR = true;

shape.EffectData.glow.enabled = true;
shape.EffectData.glow.color = new Color(0f, 1f, 0.8f);
shape.EffectData.glow.spread = 20f;
shape.EffectData.glow.blur = 12f;
shape.EffectData.glow.intensity = 2f;

shape.MarkDirty(DirtyFlags.All);
```

Neumorphic Card

```
shape.ShapeType = ShapeType.RoundedRectangle;
shape.CornerRadius = 24f;
shape.FillColor = new Color(0.9f, 0.91f, 0.93f);  // light gray

// Light shadow (top-left highlight)
shape.EffectData.shadow.enabled = true;
shape.EffectData.shadow.color = new Color(1f, 1f, 1f, 0.7f);
shape.EffectData.shadow.angle = 135f;
shape.EffectData.shadow.distance = 6f;
shape.EffectData.shadow.blur = 10f;

// Add bevel for the raised surface feel
shape.EffectData.bevel.enabled = true;
```

```
shape.EffectData.bevel.preset = BevelPreset.SoftRound;
shape.EffectData.bevel.ApplyPreset();

shape.MarkDirty(DirtyFlags.All);
```

Chat Bubble with Animated Reveal

```
shape.ShapeType = ShapeType.DialogBubble;
shape.ShapeData.bubbleTailSize = 14f;
shape.ShapeData.bubbleTailPosition = 0.15f;
shape.FillColor = new Color(0.22f, 0.65f, 1f);  // messenger blue

shape.AnimationData.preset = AnimationPresetType.Pop;
shape.AnimationData.duration = 0.25f;
shape.PlayAnimation();
```

Ability Cooldown Ring

```
shape.ShapeType = ShapeType.Ring;
shape.ShapeData.ringInnerRadius = 0.75f;
shape.FillColor = Color.gray;

// Animate fill amount over cooldown duration
IEnumerator RunCooldown(float duration)
{
   shape.ShapeType = ShapeType.PieShape;
   float t = 0;
   while (t < duration)
   {
      t += Time.deltaTime;
      shape.ShapeData.pieFill = t / duration;
      shape.MarkDirty(DirtyFlags.Mesh);
      yield return null;
   }
}
```

Gradient Badge with Outline

```
shape.ShapeType = ShapeType.Badge;
shape.FillData.fillType = FillType.Gradient;
shape.FillData.gradientType = GradientType.Linear;
```

```
shape.FillData.gradientAngle = 135f;

shape.OutlineData.enabled = true;
shape.OutlineData.width = 2f;
shape.OutlineData.color = Color.white;
shape.OutlineData.strokeType = StrokeType.Solid;

shape.MarkDirty(DirtyFlags.All);
```

Theme Switcher with Presets

```
[SerializeField] UIStylePreset lightTheme;
[SerializeField] UIStylePreset darkTheme;
[SerializeField] UIShape[] allShapes;

public void SwitchTheme(bool isDark)
{
   var theme = isDark ? darkTheme : lightTheme;
   foreach (var shape in allShapes)
      shape.ApplyPreset(theme);
}
```

8. Troubleshooting & FAQ

Q: My shape isn't showing up.

A: Check that your GameObject is inside a Canvas and has a RectTransform with non-zero size. Make sure the Fill Type isn't set to None, and that the fill color alpha isn't zero. If you're creating the shape from code, call MarkDirty(DirtyFlags.All) after setting properties.

Q: I changed a property from code but nothing happened.

A: You need to call the appropriate dirty method after modifying data classes directly. Use shape.SetFillDirty() for fill changes, shape.SetOutlineDirty() for outline changes, shape.SetEffectsDirty() for effects, or shape.MarkDirty(DirtyFlags.All) if you changed multiple things. The convenience setters (FillColor, CornerRadius) handle this automatically.

Q: The outline has a diagonal line on my tooltip/dialog bubble.

A: This was a known issue in earlier versions. Make sure you're running the latest version where compound shapes (tooltip, dialog bubbles) use a single-polygon approach that eliminates seam artifacts in outlines.

Q: Gradient/texture strokes aren't showing on my outline.

A: Gradient and texture strokes require the contour-ring rendering path. Make sure your outline is enabled and that you've assigned a gradient (for gradient stroke) or a texture (for texture stroke). The system automatically falls back from the expanded-shape approach to contour-ring when these features are active.

Q: The glow looks like a transparent shadow instead of a glow.

A: Increase the Intensity slider to boost opacity, lower the Blur to get a tighter falloff, and make sure Spread is high enough to be visible. A good starting point: Spread 15, Blur 8, Intensity 1.5.

Q: Bevel looks weird on non-rectangular shapes.

A: The bevel system uses contour-distance-based surface normals, which works on all shapes. If the bevel looks flat, increase the Width parameter so the beveled rim extends further inward. If it looks too harsh, increase Softness for a smoother profile.

Q: My shape has jagged edges on curves.

A: Increase the Mesh Quality setting. Medium is the default and works for most sizes. Use High or Ultra for large hero elements. You can also increase the Segments parameter directly for circle-based shapes.

Q: Can I use this with TextMeshPro?

A: Yes. Place the UIShape on a sibling or parent GameObject behind your TMP text. They share the same Canvas and will render in hierarchy order. UIShape doesn't interfere with TMP.

Q: Does it work with Unity's Mask and RectMask2D?

A: Yes. UIShape extends MaskableGraphic, so it respects Unity's built-in masking system. Both Mask (stencil-based) and RectMask2D (scissor-based) work out of the box.

Q: Can I use this in World Space canvases?

A: Yes. Procedural UI Pro works with all Canvas render modes: Screen Space - Overlay, Screen Space - Camera, and World Space.

Q: How do I animate properties not covered by the built-in presets?

A: Use Unity's Animation window, DOTween, or a coroutine. Modify the property you want and call the appropriate dirty method each frame. The built-in presets cover the most common patterns, but any public field is animatable from code.

Q: Why doesn't my custom shader work with the shape?

A: Set the Custom Material field in the UIShape inspector to your material. Your shader needs to support the standard UI vertex format (position, color, UV0). The shape generates vertex colors, so make sure your shader reads them.

Q: How many shapes can I have on screen?

A: It depends on complexity. Simple solid-colored shapes are extremely cheap — hundreds are fine. Shapes with glow (high spread), blur, and high-segment curves are more expensive. Check the mesh stats in the Inspector's debug section to monitor vertex counts. As a rule of thumb: keep total vertex count under 50k for smooth performance on mobile.

Q: Is there SRP / URP / HDRP support?

A: Yes. Procedural UI Pro uses the standard VertexHelper mesh generation system, which is render-pipeline independent. It works with Built-in, URP, and HDRP. HDR colors will bloom automatically if you have post-processing with bloom enabled in URP/HDRP.

Need Help?

If you run into something not covered here, reach out through the Unity Asset Store page or the support email listed in the asset package. Include your Unity version, render pipeline, and a screenshot of the issue — it helps us get back to you faster.

Thanks for choosing Procedural UI Pro. We built it because we were tired of managing sprite sheets for UI elements that should have been code all along. We hope it saves you as much time as it saved us.