SwiftUI / Drawing and graphics

API Collection

Drawing and graphics

Enhance your views with graphical effects and customized drawings.

Overview

You create rich, dynamic user interfaces with the built-in views and Shapes that SwiftUI provides. To enhance any view, you can apply many of the graphical effects typically associated with a graphics context, like setting colors, adding masks, and creating composites.

When you need the flexibility of immediate mode drawing in a graphics context, use a `Canvas` view. This can be particularly helpful when you want to draw an extremely large number of dynamic shapes — for example, to create particle effects.

For design guidance, see Materials and Color in the Human Interface Guidelines.

Topics

Immediate mode drawing

{} Add rich graphics to your SwiftUI app

Make your apps stand out by adding background materials, vibrancy, custom graphics, and animations.

`struct Canvas`

A view type that supports immediate mode drawing.

`struct GraphicsContext`

An immediate mode drawing destination, and its current state.

Setting a color

`func tint(_:)`

Sets the tint color within this view.

`struct Color`

A representation of a color that adapts to a given context.

Styling content

`func border<S>(S, width: CGFloat) -> some View`

Adds a border to this view with the specified style and width.

`func foregroundStyle<S>(S) -> some View`

Sets a view's foreground elements to use a given style.

`func foregroundStyle<S1, S2>(S1, S2) -> some View`

Sets the primary and secondary levels of the foreground style in the child view.

`func foregroundStyle<S1, S2, S3>(S1, S2, S3) -> some View`

Sets the primary, secondary, and tertiary levels of the foreground style.

`func backgroundStyle<S>(S) -> some View`

Sets the specified style to render backgrounds within the view.

`var backgroundStyle: AnyShapeStyle?`

An optional style that overrides the default system background style when set.

`protocol ShapeStyle`

A color or pattern to use when rendering a shape.

`struct AnyShapeStyle`

A type-erased ShapeStyle value.

`struct Gradient`

A color gradient represented as an array of color stops, each having a parametric location value.

`struct MeshGradient`

A two-dimensional gradient defined by a 2D grid of positioned colors.

`struct AnyGradient`

A color gradient.

`struct ShadowStyle`

A style to use when rendering shadows.

`struct Glass`

A structure that defines the configuration of the Liquid Glass material.

Transforming colors

`func brightness(Double) -> some View`

Brightens this view by the specified amount.

`func contrast(Double) -> some View`

Sets the contrast and separation between similar colors in this view.

`func colorInvert() -> some View`

Inverts the colors in this view.

`func colorMultiply(Color) -> some View`

Adds a color multiplication effect to this view.

`func saturation(Double) -> some View`

Adjusts the color saturation of this view.

`func grayscale(Double) -> some View`

Adds a grayscale effect to this view.

`func hueRotation(Angle) -> some View`

Applies a hue rotation effect to this view.

`func luminanceToAlpha() -> some View`

Adds a luminance to alpha effect to this view.

`func materialActiveAppearance(MaterialActiveAppearance) -> some View`

Sets an explicit active appearance for materials in this view.

`var materialActiveAppearance: MaterialActiveAppearance`

The behavior materials should use for their active state, defaulting to `automatic`.

`struct MaterialActiveAppearance`

The behavior for how materials appear active and inactive.

Scaling, rotating, or transforming a view

`func scaledToFill() -> some View`

Scales this view to fill its parent.

`func scaledToFit() -> some View`

Scales this view to fit its parent.

`func scaleEffect(_:anchor:)`

Scales this view's rendered output by the given amount in both the horizontal and vertical directions, relative to an anchor point.

`func scaleEffect(x: CGFloat, y: CGFloat, anchor: UnitPoint) -> some View`

Scales this view's rendered output by the given horizontal and vertical amounts, relative to an anchor point.

`func scaleEffect(x: CGFloat, y: CGFloat, z: CGFloat, anchor: UnitPoint3D) -> some View`

Scales this view by the specified horizontal, vertical, and depth factors, relative to an anchor point.

`func aspectRatio(_:contentMode:)`

Constrains this view's dimensions to the specified aspect ratio.

`func rotationEffect(Angle, anchor: UnitPoint) -> some View`

Rotates a view's rendered output in two dimensions around the specified point.

`func rotation3DEffect(Angle, axis: (x: CGFloat, y: CGFloat, z: CGFloat), anchor: UnitPoint, anchorZ: CGFloat, perspective: CGFloat) -> some View`

Renders a view's content as if it's rotated in three dimensions around the specified axis.

`func perspectiveRotationEffect(Angle, axis: (x: CGFloat, y: CGFloat, z: CGFloat), anchor: UnitPoint, anchorZ: CGFloat, perspective: CGFloat) -> some View`

Renders a view's content as if it's rotated in three dimensions around the specified axis.

`func rotation3DEffect(Rotation3D, anchor: UnitPoint3D) -> some View`

Rotates the view's content by the specified 3D rotation value.

`func rotation3DEffect(_:axis:anchor:)`

Rotates the view's content by an angle about an axis that you specify as a tuple of elements.

`func transformEffect(CGAffineTransform) -> some View`

Applies an affine transformation to this view's rendered output.

`func transform3DEffect(AffineTransform3D) -> some View`

Applies a 3D transformation to this view's rendered output.

`func projectionEffect(ProjectionTransform) -> some View`

Applies a projection transformation to this view's rendered output.

`struct ProjectionTransform`

`enum ContentMode`

Constants that define how a view's content fills the available space.

Masking and clipping

`func mask<Mask>(alignment: Alignment, () -> Mask) -> some View`

Masks this view using the alpha channel of the given view.

`func clipped(antialiased: Bool) -> some View`

Clips this view to its bounding rectangular frame.

`func clipShape<S>(S, style: FillStyle) -> some View`

Sets a clipping shape for this view.

Applying blur and shadows

`func blur(radius: CGFloat, opaque: Bool) -> some View`

Applies a Gaussian blur to this view.

`func shadow(color: Color, radius: CGFloat, x: CGFloat, y: CGFloat) -> some View`

Adds a shadow to this view.

`struct ColorMatrix`

A matrix to use in an RGBA color transformation.

Applying effects based on geometry

`func visualEffect((EmptyVisualEffect, GeometryProxy) -> some VisualEffect) -> some View`

Applies effects to this view, while providing access to layout information through a geometry proxy.

`func visualEffect3D((EmptyVisualEffect, GeometryProxy3D) -> some VisualEffect) -> some View`

Applies effects to this view, while providing access to layout information through a 3D geometry proxy.

`protocol VisualEffect`

Visual Effects change the visual appearance of a view without changing its ancestors or descendents.

`struct EmptyVisualEffect`

The base visual effect that you apply additional effect to.

Compositing views

`func blendMode(BlendMode) -> some View`

Sets the blend mode for compositing this view with overlapping views.

`func compositingGroup() -> some View`

Wraps this view in a compositing group.

`func drawingGroup(opaque: Bool, colorMode: ColorRenderingMode) -> some View`

Composites this view's contents into an offscreen image before final display.

`enum BlendMode`

Modes for compositing a view with overlapping content.

`enum ColorRenderingMode`

The set of possible working color spaces for color-compositing operations.

`protocol CompositorContent`

`struct CompositorContentBuilder`

A result builder for composing a collection of `CompositorContent` elements.

`struct AnyCompositorContent`

Type erased compositor content.

Measuring a view

`struct GeometryReader`

A container view that defines its content as a function of its own size and coordinate space.

`struct GeometryReader3D`

A container view that defines its content as a function of its own size and coordinate space.

`struct GeometryProxy`

A proxy for access to the size and coordinate space (for anchor resolution) of the container view.

`struct GeometryProxy3D`

A proxy for access to the size and coordinate space of the container view.

`func coordinateSpace(NamedCoordinateSpace) -> some View`

Assigns a name to the view's coordinate space, so other code can operate on dimensions like points and sizes relative to the named space.

`enum CoordinateSpace`

A resolved coordinate space created by the coordinate space protocol.

`protocol CoordinateSpaceProtocol`

A frame of reference within the layout system.

`struct PhysicalMetric`

Provides access to a value in points that corresponds to the specified physical measurement.

`struct PhysicalMetricsConverter`

A physical metrics converter provides conversion between point values and their extent in 3D space, in the form of physical length measurements.

Responding to a geometry change

`func onGeometryChange(for:of:action:)`

Adds an action to be performed when a value, created from a geometry proxy, changes.

Accessing Metal shaders

`func colorEffect(Shader, isEnabled: Bool) -> some View`

Returns a new view that applies `shader` to `self` as a filter effect on the color of each pixel.

`func distortionEffect(Shader, maxSampleOffset: CGSize, isEnabled: Bool) -> some View`

Returns a new view that applies `shader` to `self` as a geometric distortion effect on the location of each pixel.

`func layerEffect(Shader, maxSampleOffset: CGSize, isEnabled: Bool) -> some View`

Returns a new view that applies `shader` to `self` as a filter on the raster layer created from `self`.

`struct Shader`

A reference to a function in a Metal shader library, along with its bound uniform argument values.

`struct ShaderFunction`

A reference to a function in a Metal shader library.

`struct ShaderLibrary`

A Metal shader library.

Accessing geometric constructs

`enum Axis`

The horizontal or vertical dimension in a 2D coordinate system.

`struct Angle`

A geometric angle whose value you access in either radians or degrees.

`struct UnitPoint`

A normalized 2D point in a view’s coordinate space.

`struct UnitPoint3D`

A normalized 3D point in a view’s coordinate space.

`struct Anchor`

An opaque value derived from an anchor source and a particular view.

`protocol DepthAlignmentID`

`struct Alignment3D`

An alignment in all three axes.

`struct GeometryProxyCoordinateSpace3D`

A representation of a `GeometryProxy3D` which can be used for `CoordinateSpace3D` based conversions.

See Also

Views

View fundamentals

Define the visual elements of your app using a hierarchy of views.

View configuration

Adjust the characteristics of views in a hierarchy.

View styles

Apply built-in and custom appearances and behaviors to different types of views.

Animations

Create smooth visual updates in response to state changes.

Text input and output

Display formatted text and get text input from the user.

- Images

  Add images and symbols to your app’s user interface.

- Controls and indicators

  Display values and get user selections.

- Menus and commands

  Provide space-efficient, context-dependent access to commands and controls.

- Shapes

  Trace and fill built-in and custom shapes with a color, gradient, or other pattern.