ChatGPT

Uso matemático y perceptual del color en Apple Liquid Glass y Material Design

Efectos de Liquid Glass en Apple iOS/macOS (Blur y Vibrancy)

Apple introdujo en iOS y macOS interfaces translúcidas tipo *"frosted glass"*, conocidas como **Liquid Glass**, que usan desenfoque (blur) y vibrancy (vibrancia) para superponer capas semitransparentes. Estos efectos se consiguen combinando cálculos matemáticos de imagen (convolución gaussiana) con modelos perceptuales para garantizar legibilidad y sensación de profundidad.

Desenfoque gaussiano (Blur): El desenfoque de fondo se implementa aplicando un filtro gaussiano a la imagen de fondo. Matemáticamente, esto se expresa como una convolución: cada píxel desenfocado $C'*{x,y}$ es un promedio ponderado de los colores C* en un vecindario definido por el radio del blur r. Por ejemplo, para un radio r:

$$C'_{x,y} = \sum_{u=-r}^{r} \sum_{v=-r}^{r} w(u,v) \cdot C_{x+u,y+v},$$

donde $w(u,v)$ son pesos dados por una función gaussiana (ej. $w(u,v)=\frac{1}{2\pi\sigma^2}\exp(-\frac{u^2+v^2}{2\sigma^2})$) que suman 1. En términos simples, el valor de cada píxel se reemplaza por el promedio de los píxeles a su alrededor [1], suavizando la imagen. Un radio de blur mayor (más vecinos considerados) produce una imagen más difuminada, lo que requiere más cálculo pero aporta un efecto de vidrio más "grueso".

Capas translúcidas y mezcla de color: Encima del fondo desenfocado, Apple coloca una capa *material* semi-transparente. La composición de color usa la mezcla alfa convencional:

$$\mathrm{ColorFinal} = \alpha \cdot C_{\mathrm{material}} + (1-\alpha) \cdot C_{\mathrm{fondoDesenfocado}},$$

donde α es la opacidad de la capa (dependiendo del "grosor" del material) y C_{material} puede incluir un tinte base. En iOS existe una gama de materiales de sistema predefinidos (ej. `.systemUltraThinMaterial`, `.systemThickMaterial`, etc.) que varían en **espesor**: los más delgados dejan ver más fondo, los más gruesos lo ocultan más [2] [3]. Un material *ultra thin* aplica un blur leve con alta transparencia, brindando contexto del fondo; un material *thick* aplica blur más fuerte y opacidad mayor, resultando en un fondo casi opaco y más contraste con el texto por encima [3]. Aunque Apple no publica fórmulas exactas para cada estilo, conceptualmente un material más grueso equivale a un blur de mayor radio y/o una capa con α más alta (más blanca u oscura según el modo) sobre el fondo.

Efecto *Vibrancy* (vibrancia): Para asegurar la legibilidad del contenido en primer plano sobre fondos translúcidos, Apple emplea el efecto de **vibrancy**, que *"tira" del color de fondo hacia el contenido de primer plano* [4]. En la práctica, vibrancy es una forma de mezcla de color más compleja que la simple transparencia: Apple la describe como *"un modo de mezcla especial"* entre el contenido y el fondo [5]. En macOS se implementó con blend modes similares a **Color Burn** o **Color Dodge**, según el caso, para

realzar el contraste [5]. Por ejemplo, texto blanco sobre un fondo vibrante oscuro puede renderizarse ligeramente más brillante o saturado usando el color de fondo difuminado, logrando que no luzca apagado ni excesivamente contrastado. Apple abstrae este cálculo bajo el API `UIVibrancyEffect` / `NSVisualEffectView`, sin exponer la fórmula exacta, pero en términos de composición se puede modelar como:

- En apariencias claras (*VibrantLight*), los píxeles de primer plano se mezclan mediante un modo tipo **Color Burn** con el fondo, oscureciéndose y tomando matices del fondo [5].
- En apariencias oscuras (*VibrantDark*), se usaría el análogo inverso (p. ej. **Color Dodge** u otro) para aclarar el contenido con la luz del fondo [6].

Apple introdujo paletas vibrantes con distintos niveles (primario, secundario, terciario, etc.) que controlan cuánta coloración de fondo se aplica al texto/gráficos [7]. Por ejemplo, un label con vibrancy primaria sobre material blur tomará bastante el tono de fondo, mientras vibrancy secundaria es más sutil. Esto se combina con **colores de sistema dinámicos**: en entornos vibrantes, colores como `labelColor` o `secondaryLabelColor` se definen dependiendo de la apariencia (clara u oscura) para que el sistema los mezcle adecuadamente [8] [9]. El resultado es que el texto y los iconos "flotan" integrados con el fondo, manteniendo suficiente contraste automáticamente.

Percepción de profundidad y elevación: Estos efectos no solo son estéticos, también comunican jerarquía en la interfaz. Un principio de percepción visual es que el ojo humano enfoca elementos cercanos y ve borrosos los lejanos; este desenfoque natural (acomodación) nos indica profundidad [10]. Apple aprovecha esto: cuando una vista aparece sobre otra (ej. el Centro de Control sobre la pantalla actual), el fondo se desenfoca para indicar que quedó atrás, manteniendo *contexto* pero sin distraer [11]. Un material más *grueso* (más blur/menos transparencia) sugiere que la ventana está más "cercana" al usuario, como si fuera un vidrio opaco más elevado sobre el fondo. En cambio, un material ultra fino apenas separa visualmente, indicando menor elevación. En la práctica no hay una métrica explícita de "profundidad en mm", pero los **niveles de material** funcionan como capas de elevación cualitativa. Por ejemplo, en Apple Maps se usa un fondo traslúcido delgado para un panel sobre el mapa cuando se quiere que el usuario siga percibiendo el mapa detrás (baja elevación), mientras que un pop-up modal importante podría usar material *chrome* (más denso) para aislarlo del contexto [12] [3].

En resumen, Apple logra el **Liquid Glass** combinando un modelo matemático de blur (convolución gaussiana) y blending (alfa y modos de mezcla) con consideraciones perceptuales: calibrando transparencia y vibrancy para realzar contraste y aprovechando el desenfoque para transmitir jerarquía espacial en 2D como si fuese profundidad 3D.

Color en Material Design (Material You): tonalidades, contraste y dinámicas

Google Material Design (específicamente **Material 3** o *Material You*) introduce un sistema de color altamente algorítmico, donde las paletas ya no son estáticas sino generadas dinámicamente a partir de unos pocos colores base (*seed*). Este sistema se apoya en una representación perceptual del color y fórmulas matemáticas para garantizar consistencia y accesibilidad en todas las tonalidades.

Espacio de color HCT (Hue-Chroma-Tone): Material 3 define todos los colores en un espacio llamado **HCT**, con tres dimensiones: **Hue** (matiz, 0–360º), **Chroma** (croma o saturación/viveza del color) y **Tone** (tono claro/oscuro, en una escala 0–100 del negro al blanco) [13]. HCT combina lo mejor de dos modelos perceptuales: utiliza CAM16 (un modelo de apariencia de color) para Hue y Chroma, y el componente *L* (luminosidad perceptiva) de CIELAB para el Tone [14]. En esencia, el **Tone HCT** equivale al valor de

claridad L *(donde 0 = negro absoluto, 100 = blanco puro). Esta elección tiene fundamento matemático: la escala L* es aproximadamente lineal con la percepción humana de luminosidad, a diferencia de la luminancia relativa Y (usada en el contraste WCAG) que es no lineal [15]. Gracias a esto, diferencias fijas en el tono HCT se corresponden con diferencias de contraste predecibles. En concreto, *una diferencia de 40 puntos de Tone garantiza una razón de contraste ≈ 3.0:1, y una diferencia de 50 garantiza ≥ 4.5:1* (el mínimo WCAG para texto normal) [16]. Material 3 establece incluso diferencias mayores para mejorar legibilidad: suele usar *mínimo 60 puntos* de separación de tone entre texto y fondo, logrando contrastes alrededor de 7:1 o superiores [17]. Esta linealidad permite sustituir cálculos complejos de luminancia por sencillas restas de tonos: por ejemplo, si un fondo tiene Tone 20, un texto podría asignarse Tone 80 para exceder el contraste requerido. (Para referencia, el contraste WCAG se define como $(L_1+0.05)/(L_2+0.05)$ con luminancias relativas; HCT simplifica esto trabajando en un eje lineal ligado a L^* [18]).

Paletas tonales y esquemas de color: En Material Design, cada **color base** genera una **paleta tonal**, que es la colección del mismo matiz y croma, pero en múltiples tonos (luminosidades) desde 0 hasta 100 [19]. Típicamente se definen 11 tonos clave (0, 10, 20, ..., 100) para cada color. Por ejemplo, si el color primario de marca es un azul con cierto hue, se calcularán variantes muy oscuras de ese azul (tone 20, 30), intermedias (40, 50, 60) y claras (80, 90, etc.), hasta el blanco (100) y negro (0) aproximados de ese matiz. Este conjunto se denomina *tonal palette*. Google proporciona utilidades para calcularlas: la librería **Material Color Utilities** puede crear una paleta tonal a partir de un color usando el modelo CAM16/HCT. En código, por ejemplo, se puede obtener la tonal palette así (Kotlin):

```
val camColor = Cam16.fromInt(colorInt)
val palette = TonalPalette.fromHueAndChroma(camColor.hue, max(48.0,
camColor.chroma))
val tones = listOf(0,10,20,...,100).map { tone -> palette.tone(tone) }
```

Este código convierte el color ARGB a CAM16 para extraer su matiz y croma, luego genera una paleta tonal manteniendo ese matiz y una croma mínima de 48 (Material a veces eleva colores muy apagados a un croma estándar) [20]. El resultado es un mapa de tonos: Tone 100 sería el color más claro (casi blanco matizado), Tone 0 el casi negro, Tone 50 el color en su saturación media, etc.

Un **Core Palette** (paleta central) en Material 3 agrupa varias paletas tonales para diferentes roles de color [19]. Normalmente, del color primario *seed* se derivan: la paleta primaria (más saturada), una paleta secundaria (un color de apoyo, a veces con el mismo hue pero croma reducido o un hue cercano complementario) y una paleta terciaria (otra acentuación, usada quizás para UI/ gráficos, con hue distinto), además de dos paletas neutras (Neutral y Neutral-variant) para fondos y superficies en tonos grisáceos derivados sutilmente del primario. El algoritmo dinámico elige estos colores de manera que tengan suficiente diferenciación. Por ejemplo, con un *seed* azul, el secundario podría ser un tono aguamarina o verde-azulado suave, y el terciario un rosa o naranja complementario, según las heurísticas de armonía [21] [22]. Todas estas paletas juntas forman la base para construir un **esquema de color completo** (*Color Scheme*).

Asignación de roles y contraste: Una vez calculadas las paletas tonales, Material asigna tonos específicos a cada **role** de la interfaz (primario, onPrimary, primariaContainer, fondo, superficie, error, etc.) optimizados para *light* y *dark*. Por convención de Material 3, en tema claro el **Primary** suele ser el tono 40 de la paleta primaria (un color relativamente vivo pero no demasiado oscuro), y el **On Primary** (texto sobre primario) es el tono 100 (blanco) [23]. El **Primary Container** (el fondo de botones o tarjetas coloreadas) suele ser muy claro – tono ~90 – para que el texto oscuro encima (On Primary Container, tono ~10) tenga alto contraste [23]. En tema oscuro, estas asignaciones se invierten: el Primary pasa a

un tono claro (ej. 80) de la paleta para sobresalir sobre fondo oscuro, y el texto On Primary sería tono 20 (casi negro) [24]. Un ejemplo extraído de la generación de esquemas en Kotlin:

```
// Esquema claro
color = palette.color40       // primario tone 40
onColor = palette.color100    // texto sobre primario tone 100 (blanco)
colorContainer = palette.color90   // contenedor claro tone 90
onColorContainer = palette.color10 // texto sobre contenedor tone 10 (casi
negro)

// Esquema oscuro
color = palette.color80       // primario tone 80 (más claro en dark)
onColor = palette.color20     // texto sobre primario tone 20 (oscuro)
colorContainer = palette.color30   // contenedor oscuro tone 30
onColorContainer = palette.color90 // texto claro sobre contenedor tone 90
```

Como se observa, la diferencia de tonos entre `color` y `onColor` es grande (p.ej. 40 vs 100 en claro, diferencia 60; ó 80 vs 20 en oscuro, diferencia 60), asegurando accesibilidad [17]. De hecho, el sistema ajusta automáticamente cualquier conflicto de contraste: si un color de fondo y de texto no alcanzan el mínimo, el algoritmo mueve el tone del texto o fondo hasta lograrlo. Esto es posible gracias a la continuidad del eje Tone. Por ejemplo, si por branding se requería usar un primario muy claro en tema claro, el sistema aumentaría ligeramente la oscuridad del texto (bajando su tone) o viceversa para mantener la legibilidad.

Generación automática de temas dinámicos: El gran aporte de Material You es que todo lo anterior sucede automáticamente a partir de un color *seed* proporcionado (por ejemplo, el usuario escoge un color, o se extrae de su fondo de pantalla). La función `ColorScheme.fromSeed(...)` de Flutter/Android ejemplifica este proceso: dado un `seedColor`, genera las paletas tonales y construye un esquema completo que puebla todos los roles de color [25] [26]. Es destacable que el color semilla en sí puede no aparecer tal cual en la interfaz; en su lugar, aparece a través de sus tonalidades derivadas [27]. El algoritmo ofrece incluso variantes de estilo: por defecto usa un esquema *tonalSpot* (que tiende a paletas pastel, reduciendo chroma si el seed es muy saturado) [28]. Existe otra variante *fidelity* que mantiene mayor fidelidad al color original (paletas más intensas similares al seed) [28]. Además, se permite ajustar la **intensidad de contraste global** mediante un parámetro – por ejemplo, `contrastLevel = 1.0` produce un esquema de "alto contraste" donde las diferencias de tone entre fondos y textos son mayores de lo normal, mientras `contrastLevel = -1.0` haría un esquema más tenue [29]. Internamente, esto afecta cálculos como la elección de tonos para `onSurface`, `inverseSurface`, etc., pero siempre respetando las reglas mínimas de legibilidad.

Elevación tonal en temas oscuros: Material Design aborda de forma interesante la representación de profundidad (*elevación*) en **tema oscuro** a través del color. En lugar de sombras (que son menos visibles sobre negro), usa una superposición *tonal*. El color de fondo base de las superficies oscuras es un gris muy oscuro (#121212 en Material 2). A medida que un componente tiene mayor elevación (está "más arriba"), se le aplica un **overlay blanco semitransparente** para aclararlo ligeramente [30]. Las guías oficiales definían porcentajes específicos: por ejemplo, *1dp = 5%* de blanco sobre #121212, *2dp = 7%*, 3dp = 8%, 4dp = 9%, 6dp = 11%, 8dp = 12%, hasta 24dp = 16% [31]. Esto significa que una tarjeta flotante a 8dp en modo oscuro tendría un color de fondo = mix(#121212, #FFFFFF, 12%). Matemáticamente, es similar a la fórmula de mezcla antes citada, mezclando el color de superficie con blanco según la opacidad dada por la elevación. Este "elevation overlay" se puede pre-calcular o aplicar en tiempo real; por ejemplo, en Android se ofrece la propiedad `?attr/elevationOverlayEnabled` que, si activa,

automáticamente mezcla el color de superficie con el overlay adecuado según la elevación del componente [32]. El resultado es que en oscuro, los elementos en distintos planos se distinguen porque los de mayor elevación aparecen más claros (como si recibieran algo de luz), simulando profundidad de forma sutil pero uniforme con la paleta. Material 3 continúa con este enfoque, aunque ajustando ligeramente los valores y permitiendo overlays tanto en dark como en light en ciertos casos para harmonizar con el nuevo esquema Material You [33].

Implementación en código y herramientas: Google ha proporcionado implementaciones multi-plataforma de estas utilidades de color (Java, Kotlin, Dart, Swift, Typescript, etc. [34]). Por ejemplo, usando la librería *MaterialColorUtilities* se puede convertir un color ARGB a HCT, modificar sus atributos y obtener el nuevo color. Un caso sencillo: aclarar un color oscuro incrementando su Tone:

```
val hctColor = Hct.fromInt(0xFF3700B3) // un morado oscuro
hctColor.tone = 90.0                   // subir a tone 90 (muy claro)
val lightColorInt = hctColor.toInt()   // obtener el ARGB resultante
```

En este fragmento, partiendo de un morado (H=282°, C≈70, Tone≈25), al asignar Tone 90 obtenemos un lila pastel claro manteniendo el mismo hue [35]. Operaciones similares son posibles para reducir chroma (desaturar) o ajustar hue, con métodos como `Hct.withChroma()` o `withHue()`, donde la librería recalcula el color válido más cercano dado que no cualquier combinación arbitraria de H, C, T es alcanzable en el gamut sRGB [36] [37].

En el ecosistema de desarrollo, estas utilidades se incorporan en APIs de alto nivel: por ejemplo, en Flutter la clase `ColorScheme.fromSeed` mencionada o en Android `MaterialColors` y `Scheme` hacen todo el trabajo debajo usando estas fórmulas. Para los diseñadores, Google ofrece herramientas como **Material Theme Builder** (Figma plugin y Web) que permiten ingresar un color semilla y obtener automáticamente el esquema completo, asegurando que todos los colores generados mantienen el contraste y la armonía tonal [38] [39].

Conclusión: El sistema de color de Material Design ejemplifica el uso de modelos perceptuales (HCT) y algoritmos matemáticos para automatizar paletas. Con fórmulas basadas en la ciencia del color, logra *esquemas dinámicos* donde, a partir de un color, se derivan decenas de colores coherentes. Se garantizan contrastes mínimos al mapear diferencias de tono a ratios de contraste [16], y se maneja la elevación tonal mediante cálculos de opacidad ajustados a la profundidad visual [40]. Todo esto es exportable a código y se refleja en utilidades listas para usar, cumpliendo con la visión de Material You: interfaces personalizadas, accesibles y adaptativas construidas sobre una base científica del color.

Fuentes: Las referencias utilizadas incluyen la documentación de Apple HIG y Developer (p. ej. vibrancy en WWDC2014 [5]), artículos técnicos sobre blur en iOS [1] [10], así como la documentación y código abierto de Material Design 3 [17] [16] [31] [23], entre otros. Todas ellas respaldan los detalles matemáticos y perceptuales descritos.

---

[1] [10] [11] UIVisualEffectView Tutorial: Getting Started | Kodeco
https://www.kodeco.com/16125723-uivisualeffectview-tutorial-getting-started

[2] [3] [4] [7] [12] Using Apple's Materials Blur & Vibrancy in Your App Design - Play · Design mobile apps with the power of iOS & SwiftUI
https://createwithplay.com/blog/best-practices-for-using-materials-properties

5 6 8 9 Adopting Advanced Features of the New UI of OS X Yosemite - WWDC 2014 - WWDC Index
https://nonstrict.eu/wwdcindex/wwdc2014/220/

13 14 17 18 A new color space called HCT - Material 3 Themes for Home Assistant
https://material3-themes-manual.amoebelabs.com/basics/m3-analysis-introduction/

15 16 36 37 Hct
https://docs.materialkolor.com/material-color-utilities/com.materialkolor.hct/-hct/index.html

19 21 22 34 38 39 GitHub - material-foundation/material-color-utilities: Color libraries for Material You
https://github.com/material-foundation/material-color-utilities

20 23 24 35 android jetpack compose - Material 3 custom colors - How to generate tonal palettes? - Stack Overflow
https://stackoverflow.com/questions/77163228/material-3-custom-colors-how-to-generate-tonal-palettes

25 26 27 28 29 ColorScheme.fromSeed constructor - ColorScheme - material library - Dart API
https://api.flutter.dev/flutter/material/ColorScheme/ColorScheme.fromSeed.html

30 31 32 40 Elevation
https://klwp.erikbucik.com/material/elevation

33 Update elevation overlay to apply in both light and dark theme #91607
https://github.com/flutter/flutter/issues/91607