Chapitre A — Motion & Animation

10 skills regroupés sous ce thème.

- motion-tokens
- motion-three-r3f
- reduced-motion-a11y
- motion-visual-regression
- motion-perf-audit
- motion-framer-patterns
- motion-gsap-scrolltrigger
- motion-view-transitions
- view-transitions
- motion-system

motion-tokens

motion-tokens

> *Catalogue généré le 2026-05-11*

En une phrase

Génère un jeu de tokens d'animation (durées, easings, distances) versionnables en JSON + variables CSS, avec mapping prêt pour Motion (React) et GSAP.

Quand l'utiliser

- Au démarrage d'un projet quand on veut une grammaire d'animation cohérente
- Quand les animations existantes sont incohérentes (chaque dev choisit ses durées au pif)
- Pour unifier un design system qui couvre déjà couleurs/typo mais pas le mouvement
- Avant d'attaquer un site marketing avec plusieurs sections animées
- Pour préparer le terrain avant `motion-system` ou `motion-framer-patterns`

Comment l'invoquer

- **Slash command** : `/motion-tokens`
- **Auto-invocation** : □ Oui — dès qu'un design system ou theming motion est nécessaire

Description détaillée

Ce skill produit un fichier `tokens.motion.json` avec 5 à 7 durées de base (par exemple `fast: 120ms`, `base: 200ms`, `slow: 320ms`), 3 à 5 courbes d'easing nommées (`standard`, `accelerate`, `decelerate`) et un set de distances pour les translations.

Il génère ensuite trois sorties exploitables tout de suite : le JSON pour versionner dans Git, des variables CSS (`--motion-duration-base`, `--motion-ease-standard`) à poser dans `:root`, et un mapping côté JS pour Motion ou GSAP afin que tu puisses écrire `transition={{ duration: motion.duration.base, ease: motion.ease.standard }}` au lieu de coder des nombres en dur.

Il finit par poser des règles d'usage : quelle durée pour un hover, laquelle pour une ouverture de modal, quel easing pour quel type de mouvement. Pour Thymon : c'est la fondation à poser avant d'animer sérieusement quoi que ce soit. Une fois ces tokens en place, tu peux changer le ressenti motion de tout le site en modifiant un seul fichier.

Source

- **Plugin** : `perso (standalone)`
- **Nom interne** : `motion-tokens`
- **Fichier** : `/home/thymon/.claude/skills/motion-tokens/SKILL.md`

motion-three-r3f

motion-three-r3f

> *Catalogue généré le 2026-05-11*

En une phrase

Guide pratique pour intégrer de la 3D web avec Three.js et React Three Fiber sans plomber la performance, en maîtrisant la boucle de rendu et les règles de cleanup.

Quand l'utiliser

- Quand on veut un hero WebGL animé sur une landing
- Pour ajouter un visuel 3D interactif (carte, produit, scène) à une page
- Quand un projet Three.js existant rame ou chauffe le CPU
- Avant d'attaquer une scène complexe pour valider qu'on a vraiment besoin de 3D
- Pour structurer proprement une intégration React + Canvas avec gestion du frameloop

Comment l'invoquer

- **Slash command** : `/motion-three-r3f`
- **Auto-invocation** : Oui — quand un projet implique de la 3D dans le navigateur

Description détaillée

Ce skill commence par te poser la question qui sauve : as-tu vraiment besoin de Three.js, ou est-ce qu'une image SVG animée ferait l'affaire ? La 3D web est lourde et la majorité des intégrations gagneraient à être remplacées par autre chose. Si la 3D est justifiée, il passe à la mise en place.

Il configure d'abord la boucle de rendu en mode `frameloop="demand"` plutôt qu'en rendu permanent : la scène ne se redessine que quand quelque chose change, ce qui peut diviser la consommation CPU par dix sur un hero statique. Il centralise la logique d'animation dans un seul `useFrame` au lieu de multiplier les `requestAnimationFrame` parallèles. Et il insiste sur le cleanup et le `dispose()` des géométries/textures au démontage, sinon le navigateur garde tout en mémoire.

Tu repars avec une architecture recommandée pour ton Canvas, des snippets prêts à copier pour `frameloop` + `invalidate` + `useFrame`, et une checklist perf à dérouler avant la mise en prod. Pour Thymon : c'est le skill à utiliser dès que le mot "WebGL" ou "3D" apparaît dans un brief, pour éviter de partir dans un truc impossible à maintenir.

Source

- **Plugin** : `perso (standalone)`
- **Nom interne** : `motion-three-r3f`
- **Fichier** : `/home/thymon/.claude/skills/motion-three-r3f/SKILL.md`

reduced-motion-a11y

reduced-motion-a11y

> *Catalogue généré le 2026-05-11*

En une phrase

Conçoit une stratégie `prefers-reduced-motion` complète pour que les utilisateurs sensibles au mouvement (vertige, migraines) puissent profiter du site sans casser le design.

Quand l'utiliser

- Sur tout site avec des animations un peu marquées (parallax, transitions de page, scroll-triggered)
- Avant la mise en prod pour cocher la case accessibilité WCAG
- Quand un utilisateur signale du motion sickness sur une page
- Pour structurer une variante "reduce" qui reste élégante (pas juste tout couper)
- Sur un projet où le motion est central à l'identité (sinon trop d'animations à gérer)

Comment l'invoquer

- **Slash command** : `/reduced-motion-a11y`
- **Auto-invocation** : □ Oui — pour mettre en place une stratégie reduced-motion

Description détaillée

Le réflexe naïf face à `prefers-reduced-motion: reduce`, c'est de tout désactiver d'un coup avec une media query qui assassine toutes les transitions. Ce skill propose mieux : une matrice où tu décides

animation par animation ce qui est essentiel (un chargement qui doit rester visible), ornemental (un parallax qu'on peut couper) ou à remplacer par une variante apaisée (un slide qui devient un fade).

Il génère ensuite l'implémentation CSS via la media query plus un attribut data (`data-motion="reduce"`) pour pouvoir tester en local sans changer les préférences système, et passe en revue les pièges classiques : le focus qui saute pendant une transition de page, les modales qui s'ouvrent sans animation visible donc semblent surgir, les drawers qui ne préviennent plus l'utilisateur.

Tu repars avec la matrice normal vs reduce, des snippets CSS et utilitaires JS, et une checklist QA pour vérifier que le site reste utilisable en mode reduced-motion. Pour Thymon : à utiliser sur tout projet où il y a plus de trois animations significatives, surtout les sites marketing où les utilisateurs ne s'attendent pas à un crash de leur oreille interne.

Source

- **Plugin** : `perso (standalone)`
- **Nom interne** : `reduced-motion-a11y`
- **Fichier** : `/home/thymon/.claude/skills/reduced-motion-a11y/SKILL.md`

motion-visual-regression

motion-visual-regression

> *Catalogue généré le 2026-05-11*

En une phrase

Met en place des tests visuels Playwright (golden screenshots) pour détecter automatiquement quand une animation casse ou quand un composant change visuellement sans qu'on le veuille.

Quand l'utiliser

- Sur un projet où les régressions visuelles coûtent cher (site client, portfolio public)
- Quand une équipe travaille à plusieurs et qu'on veut un filet de sécurité en CI
- Après avoir polish un motion system pour figer le rendu attendu
- Quand on a déjà eu un bug "ça marche en local mais pas en prod" sur du visuel
- Sur les pages critiques d'un site (hero, checkout, formulaires)

Comment l'invoquer

- **Slash command** : `/motion-visual-regression`
- **Auto-invocation** : Oui — pour stabiliser le motion et détecter les régressions visuelles

Description détaillée

L'idée : Playwright prend une capture d'écran de référence (le "golden") quand tout va bien, puis à chaque commit il recompare la page actuelle à la référence. Si les pixels diffèrent au-delà d'un seuil, le test échoue et tu sais qu'un changement de CSS ou un refactor a impacté le visuel.

Ce skill installe Playwright, écrit deux premiers tests `toHaveScreenshot()` (un état idle + un état stable après animation), puis t'arme contre la "flakiness" — c'est-à-dire les faux positifs où le test échoue sans raison à cause d'une animation qui n'est pas finie au moment de la capture. Pour ça il propose de désactiver les animations pendant les tests, de masquer les zones dynamiques (timestamp, données live), et de fixer des seuils de tolérance pixel pertinents.

Tu repars avec des scripts npm prêts à lancer, deux tests qui fonctionnent dès le premier `npx playwright test`, et un set de recommandations pour éviter que ta suite de tests visuels devienne un cauchemar à maintenir. Pour Thymon : utile si tu déploies souvent et que tu veux dormir tranquille la nuit, moins pertinent sur un projet perso en solo où tu vois tout de suite si quelque chose casse.

Source

- **Plugin** : `perso (standalone)`
- **Nom interne** : `motion-visual-regression`
- **Fichier** : `/home/thymon/.claude/skills/motion-visual-regression/SKILL.md`

motion-perf-audit

motion-perf-audit

> *Catalogue généré le 2026-05-11*

En une phrase

Diagnostic ciblé d'une animation qui rame ou d'un scroll saccadé, avec identification de la cause (compositing, layout thrashing, will-change mal posé) et plan de correction priorisé.

Quand l'utiliser

- Quand une animation visiblement saccade ou "lag" sur certains appareils
- Quand le scroll d'une page sent le pâteux et qu'on ne sait pas pourquoi
- Si le CPU s'envole pendant une animation (ventilateur du laptop qui démarre)
- Après une mise en prod où une animation tournait bien en dev mais plus en réel
- Pour valider qu'un hero animé est viable avant de le shipper

Comment l'invoquer

- **Slash command** : `/motion-perf-audit`
- **Auto-invocation** : Oui — pour auditer une animation laggy

Description détaillée

La performance motion est un sujet contre-intuitif : ce n'est pas le nombre d'animations qui pose problème, c'est le type de propriétés animées. Animer `transform` et `opacity`, c'est gratuit (le navigateur délègue au GPU). Animer `top`, `left`, `width` ou `height`, c'est cher (chaque frame

relance un calcul de layout).

Ce skill commence par classer le symptôme observé (jank ponctuel, scroll lourd, jank permanent, CPU haut), puis enquête sur les causes probables : propriétés animées qui forcent du layout, "thrashing" où on lit et écrit le DOM en boucle, plusieurs `requestAnimationFrame` qui se marchent dessus, observers (`IntersectionObserver`, `ResizeObserver`) qui se déclenchent en cascade, ou `will-change` posé partout "au cas où" (ce qui consomme de la mémoire GPU sans aider).

Il produit ensuite un tableau symptôme → cause → preuve → fix, un top 5 d'actions priorisées P0/P1/P2 avec snippets de code, et une Definition of Done mesurable (par exemple "scroll à 60fps sur Chrome desktop, 30fps minimum sur mobile mid-range"). Pour Thymon : c'est l'enquêteur à appeler quand un truc rame sans raison apparente, avant de jeter l'animation entière.

Source

- **Plugin** : `perso (standalone)`
- **Nom interne** : `motion-perf-audit`
- **Fichier** : `/home/thymon/.claude/skills/motion-perf-audit/SKILL.md`

motion-framer-patterns

motion-framer-patterns

> *Catalogue généré le 2026-05-11*

En une phrase

Snippets et patterns prêts à l'emploi pour la librairie Motion (anciennement Framer Motion) en React : animations d'entrée/sortie, layout animations, shared elements, gestures et reduced-motion.

Quand l'utiliser

- Pour ajouter une animation d'apparition propre à un composant React
- Quand tu veux qu'un élément "glisse" élégamment d'une grille vers un détail (layout animation)
- Pour faire un effet de partage entre deux pages (shared element transition)
- Quand un site React commence à avoir trop d'animations désordonnées
- Pour avoir une référence rapide des bonnes pratiques sans relire la doc de Motion

Comment l'invoquer

- **Slash command** : `/motion-framer-patterns`
- **Auto-invocation** : □ Oui — pour des patterns d'animation React avec Motion

Description détaillée

Motion (anciennement Framer Motion) est la librairie d'animation la plus utilisée en React. Elle est très puissante mais facile à mal utiliser, ce qui dégrade la performance et casse le ressenti.

Ce skill commence par t'aider à choisir le pattern adapté à ton besoin : `enter/exit` pour une apparition simple, `layout` pour qu'un élément s'adapte tout seul quand son conteneur change, `shared` pour qu'un élément "voyage" entre deux contextes (par exemple une carte qui s'ouvre en modal), `scroll` pour le scroll-triggered. Il applique ensuite les tokens du projet (durées, easings) pour rester cohérent avec le reste du motion system, et implémente la variante reduced-motion en parallèle.

Il finit par une checklist QA qui rappelle les pièges : ne pas avoir trop de `layout` animations simultanées (chacune coûte en perf), préférer `transform` à `top/left`, nettoyer les listeners au démontage. Tu repars avec trois snippets prêts à coller (un enter/exit, un layout, un shared), 5 do/don't maximum pour ne pas surcharger, et une note explicite sur le reduced-motion. Installation : `npm install motion`. Pour Thymon : c'est ta référence rapide pour ne pas perdre 30 minutes à chaque animation React.

Source

- **Plugin** : `perso (standalone)`
- **Nom interne** : `motion-framer-patterns`
- **Fichier** : `/home/thymon/.claude/skills/motion-framer-patterns/SKILL.md`

motion-gsap-scrolltrigger

motion-gsap-scrolltrigger

> *Catalogue généré le 2026-05-11*

En une phrase

Snippets et bonnes pratiques GSAP + ScrollTrigger pour animer au scroll : reveals progressifs, pinning d'éléments, scrub lié au scroll, timelines complexes, cleanup et refresh sur changements de layout.

Quand l'utiliser

- Pour un site marketing ou portfolio avec une narration scrollytellante
- Quand tu veux un élément qui reste collé pendant qu'un autre défile (pinning)
- Pour synchroniser une animation à la position exacte de scroll (scrub)
- Sur une landing où plusieurs sections doivent "se révéler" en cascade au scroll
- Pour fiabiliser un projet GSAP existant qui a des animations qui ne se déclenchent plus après un changement de layout

Comment l'invoquer

- **Slash command** : `/motion-gsap-scrolltrigger`
- **Auto-invocation** : Oui — pour des patterns GSAP + ScrollTrigger

Description détaillée

GSAP est la librairie d'animation la plus puissante du web, et son plugin ScrollTrigger est ce qu'on utilise dès qu'on veut faire du "scroll storytelling" (les sites Apple, Stripe Press, et la plupart des landings premium).

Ce skill couvre les trois grands patterns ScrollTrigger : `reveal` (un élément qui apparaît quand il entre dans la viewport), `scrub` (une animation dont le timeline est piloté directement par la position du scroll, ce qui crée des effets de "défilement contrôlé"), et `pin` (un élément qui reste en place pendant que le contenu défile derrière, technique signature des sites éditoriaux). Il insiste sur les pièges spécifiques à GSAP : ne pas créer des instances ScrollTrigger en double (la consommation explose vite), penser à appeler `ScrollTrigger.refresh()` quand le DOM change après le chargement (sinon les triggers sont calculés sur l'ancien layout), nettoyer toutes les animations au démontage du composant React/Vue pour éviter les fuites mémoire.

Il fournit aussi la variante `reduced-motion` : sur les patterns scrub/pin, on remplace généralement par un simple fade ou par un affichage instantané. Installation : `npm install gsap`. Pour Thymon : ce skill est ton kit de survie quand tu attaques un site avec du scroll narratif fort, comme un portfolio ou une landing premium.

Source

- **Plugin** : `perso (standalone)`
- **Nom interne** : `motion-gsap-scrolltrigger`
- **Fichier** : `/home/thymon/.claude/skills/motion-gsap-scrolltrigger/SKILL.md`

motion-view-transitions

motion-view-transitions

> *Catalogue généré le 2026-05-11*

En une phrase

Implémente les View Transitions natives du navigateur (SPA ou MPA) avec shared elements, en progressive enhancement — variante reduced-motion incluse et focus géré proprement.

Quand l'utiliser

- Pour ajouter des transitions natives entre pages d'un site multi-pages (MPA)
- Sur une SPA Vue/React quand on veut des transitions plus fluides que celles de la librairie
- Quand tu veux qu'un élément "voyage" entre deux pages (image qui s'agrandit en passant au détail)
- Pour donner un ressenti premium à un portfolio ou un site marketing
- Si tu veux les bénéfices visuels sans la dette technique d'une libraire externe

Comment l'invoquer

- **Slash command** : `/motion-view-transitions`
- **Auto-invocation** : Oui — pour implémenter des View Transitions avec shared elements

Description détaillée

L'API View Transitions est la nouveauté navigateur qui change la donne : avec quelques lignes de CSS et un appel à `document.startViewTransition()`, on obtient des transitions de pages fluides sans librairie, et même des animations d'éléments partagés entre deux états (le fameux "shared element transition" des apps mobiles).

Ce skill démarre par la stratégie de support et le fallback : tous les navigateurs ne la supportent pas encore (Safari traîne). Il met en place un progressive enhancement où les navigateurs récents profitent des transitions, et les autres ont une navigation classique qui marche normalement. Il configure ensuite la transition globale (fade ou slide subtil) puis les transitions d'éléments partagés via `view-transition-name` en CSS.

Deux points critiques que le skill couvre : la gestion du **focus**, qui doit suivre logiquement la transition pour ne pas perdre les utilisateurs au clavier, et la **variante reduced-motion** qui désactive l'animation tout en gardant le changement de page propre. Il termine par une checklist QA pour vérifier que la navigation back/forward du navigateur fonctionne aussi (souvent oublié).

Tu repars avec une stratégie progressive enhancement claire, un snippet minimal JS + CSS qui marche, la variante reduced-motion et la checklist de QA. Pour Thymon : c'est la version "motion" du skill, complémentaire de `view-transitions` qui est plus orientée perception utilisateur. Utilise celui-ci quand tu veux du shared element fluide.

Source

- **Plugin** : `perso (standalone)`
- **Nom interne** : `motion-view-transitions`
- **Fichier** : `/home/thymon/.claude/skills/motion-view-transitions/SKILL.md`

view-transitions

view-transitions

> *Catalogue généré le 2026-05-11*

En une phrase

Implémente l'API View Transitions du navigateur en progressive enhancement pour ajouter des transitions de pages fluides (fade, slide, shared element) sans casser l'accessibilité ni la performance.

Quand l'utiliser

- Sur un portfolio où la navigation entre projets doit avoir un ressenti premium
- Pour une landing marketing où on veut un effet "wow" subtil entre sections
- Sur une SPA Next/Nuxt/Astro qui veut des transitions sans librairie tierce
- Quand le client demande des transitions "comme les apps mobiles"
- Pour ajouter une couche de polish à un site déjà fonctionnel sans tout refaire

Comment l'invoquer

- **Slash command** : `/view-transitions`
- **Auto-invocation** : Oui — pour navigation premium, portfolios, marketing, apps

Description détaillée

C'est la version "perception utilisateur" du skill, là où `motion-view-transitions` est plus orientée motion designer. L'objectif ici est plus pragmatique : ajouter des transitions de pages **uniquement**

si elles apportent un bénéfice clair (perception de fluidité, continuité visuelle entre deux états), pas juste pour faire joli.

Le skill commence par vérifier la compatibilité navigateur et définit une stratégie de fallback : sur les navigateurs qui ne supportent pas l'API (Safari traîne, Firefox arrive), la navigation reste classique. Il implémente ensuite des transitions simples — généralement un fade ou un slide subtil — et propose une variante "shared element" pour les cas où un élément doit visuellement persister entre deux pages.

Deux préoccupations transverses guident l'implémentation : **préserver le focus** (l'utilisateur clavier ne doit pas être perdu après une transition) et **respecter** `prefers-reduced-motion` (transition désactivée pour les utilisateurs sensibles au mouvement). Tu repars avec une stratégie support+fallback claire, un snippet minimal JS + CSS, les variantes fade/shared element, et les notes d'accessibilité associées.

Pour Thymon : c'est le skill rapide à dégainer quand tu veux ajouter une touche premium à un projet sans installer Motion ou GSAP. Plus léger, plus moderne, suffisant dans 80 % des cas.

Source

- **Plugin** : `perso (standalone)`
- **Nom interne** : `view-transitions`
- **Fichier** : `/home/thymon/.claude/skills/view-transitions/SKILL.md`

motion-system

motion-system

> *Catalogue généré le 2026-05-11*

En une phrase

Conçoit un motion system complet pour un projet : principes directeurs, tokens (durations/easings/distances), 6 patterns réutilisables (enter, hover, focus, modal, list, page) et règles reduced-motion intégrées.

Quand l'utiliser

- Sur un projet ambitieux où le motion fait partie de l'identité (portfolio, marketing premium, app produit)
- Quand `motion-tokens` ne suffit plus et qu'il faut formaliser les patterns
- Pour aligner une équipe sur "quelle animation pour quel usage" avant de commencer à coder
- Avant de générer la doc d'un design system pour avoir un chapitre motion solide
- Sur une refonte de site où le mouvement est un levier de différenciation

Comment l'invoquer

- **Slash command** : `/motion-system`
- **Auto-invocation** : Oui — pour produire une grammaire motion cohérente d'un projet

Description détaillée

Un motion system, c'est l'équivalent du design system mais pour le mouvement. Au lieu d'avoir 47 animations différentes choisies au feeling, on a un vocabulaire restreint et cohérent qui se réutilise partout — ce qui rend le site mémorable et facile à maintenir.

Ce skill produit le motion system en 4 temps. D'abord les **intentions** : pour quoi anime-t-on ? Hiérarchie (un élément attire l'œil), causalité (un clic ici déclenche un changement là), feedback (l'app me dit qu'elle a compris), émotion (un détail délicieux). Ensuite les **tokens** : 5-7 durations et 3-5 easings nommés sémantiquement, plus des distances standard. Puis 6 **patterns** prêts à l'emploi : enter (apparition à l'arrivée d'un élément), hover (réaction au survol), focus (signal clavier), modal (ouverture/fermeture), list (animations en série sur une liste), page (transition d'écran). Enfin les **règles reduced-motion**, déclinées pattern par pattern (pas juste un kill switch global).

À la sortie, tu as 6-10 lignes de principes directeurs, les tokens en JSON + CSS variables, les 6 patterns avec snippets fonctionnels, et une matrice normal vs reduce pour chacun. Conseil intégré : préférer `transform` et `opacity` (gratuits pour le GPU), éviter `width`/`height`/`top`/`left` qui forcent du layout. Pour Thymon : c'est le skill "boss" du motion. Une fois ce système posé, tous les autres skills motion (framer-patterns, gsap, view-transitions) viennent s'y appuyer naturellement.

Source

- **Plugin** : `perso (standalone)`
- **Nom interne** : `motion-system`
- **Fichier** : `/home/thymon/.claude/skills/motion-system/SKILL.md`