

High Dynamic Range Effects

Naaji Karim* Université de Montréal



Résumé

De nombreux effets sont réalisables en *post processing* suite à l'utilisation d'images en *High Dynamic Range*. Notamment car ce format permet d'utiliser une plage de couleurs plus conséquente, et ainsi de porter son attention sur des zones de forte luminance. Deux de ces effets, pour une implémentation en temps réel, vont être présentés dans cet article. Le premier effet, le *bloom*, va être en partie revisité de manière à se rapprocher de l'effet de *glare*. L'article présente également une méthode *lens flare approximation* utilisant des textures générées de manière procédurales.

Mots clefs : *high dynamic range, image based lighting, bloom, lens flare approximation, cubic lens distortion*

1 Introduction

L'utilisation du *High Dynamic Range* (HDR) permet de réaliser des effets que des images en *Low Dynamic Range* (LDR) ne permettent pas. Le fait que la dynamique des couleurs soit sur une plage bien plus importante, permet de singulariser les zones de forte luminance pour pouvoir ensuite réaliser des effets en les utilisant.

Un des effets les plus connus et certainement l'un des plus utilisé est le *bloom*. D'une approche physique, l'effet de *bloom* vient du fait que la lentille d'un objectif ne peut pas faire un focus parfait. Cette imperfection n'est pas visible lorsqu'il n'y a pas ou peu de fortes

sources d'intensité lumineuse. Cependant lorsqu'une source lumineuse est présente, cette imperfection devient visible, et aura pour conséquence d'étaler la source lumineuse autour de sa source. Les implementations de *bloom*, notamment une méthode présenté par Intel [7] sur laquelle cet article se base, permettent d'approximer de manière non physique cet effet. L'idée principale pour approximer cet effet est de singulariser les zones de fortes luminances pour ensuite les diminuer en échelle, leur appliquer un flou de type gaussien et les ajouter finalement à l'image originale.

Le résultat de cette méthode a l'inconvénient de donner un effet souvent trop lisse comparé à la réalité. On peut noter qu'en réalité un bruit peut être perceptible autour des sources lumineuses. Lorsque les photons issus d'une source lumineuse se déplacent dans un fluide, ceux-ci subissent des changements de directions suite à des collisions avec les différentes éléments constituant ce fluide. Un bon exemple qui permet de voir ce phénomène, est d'observer une forte source lumineuse par temps de brouillard. Pour approximer ce phénomène, un bruit de type *Fractional Brownian Motion* (FBM), sera utilisé pour tenter d'améliorer l'effet de *bloom* et s'approcher d'un effet de *glare* [5].

Les effets de type *Lens Flare* peuvent être réalisés en se basant sur un modèle physique [2] ce qui donne des résultats extrêmement proche de la réalité, tant que la connaissance de la structure de l'objectif que l'on souhaite représenter soit connue, ce qui n'est pas toujours le cas. Pour un effet plus modeste, et plus simple à mettre en place, une approximation peut être intéressante. Une méthode récente [1], donne des résultats convaincants. Une amélioration va être apporté en utilisant des textures générées de manières procédurales, notamment pour colorer les *ghosts* générés, ainsi que pour simuler l'effet de *sun starburst*. L'utilisation de ces textures donnent plus de maniabilité sur le résultat final de l'effet, et pourrait par exemple aisément être modifié pendant l'exécution. Pour améliorer la réalité de l'effet une distortion va être réalisée, notamment pour que les *flares* semblent épouser les bords d'une lentille, d'après un modèle de distortion [3].

Le *tone mapping* utilisé sera celui présenté par Reinhard et al. [4], pour pouvoir être utilisé en temps réel, un calcul de luminance moyenne à chaque *frame* est réalisé.

*e-mail: karim.naaji@gmail.com

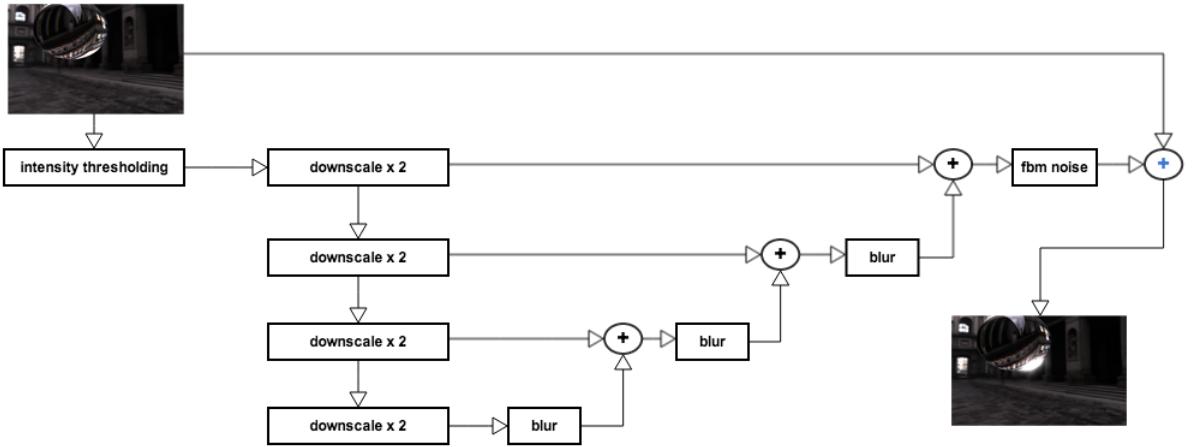


Figure 1 – Bloom pipeline, la dernière composition en bleue prend en considération un poids associé à l'image générée par le *bloom*. Le filtre de bruit FBM est optionnel.

2 Bloom

La *pipeline* de *bloom* se décompose entre trois étapes principales (Figure 1). La première est l’ *intensity thresholding*, qui va singulariser les zones dont la luminance dépasse un certain seuil. Dans un deuxième temps, ce résultat est réduit en échelle pour subir finalement dans une dernière partie un flou, ici gaussien. L’image résultante issue du filtre de flou est ajoutée au résultat de *downsampling* le précédent immédiatement dans la *pipeline*. L’intérêt du *downsampling* est d’aller chercher des valeurs éloignées autour de la zone de forte luminance. Le fait d’ajouter le résultat issu du filtre de flou gaussien de cette manière permet d’ajouter plus la douceur et de cohérence entre les différentes *frames*. Une des méthodes souvent utilisée est d’appliquer un flou sur chacune des images issues du *downsampling* et de toutes les ajouter. Cette méthode aura l’inconvénient de faire apparaître un scintillement entre deux *frames* successives.

Le bruit, pour un fragment positionné en x et y , est calculé de la manière suivante :

$$N(x,y) = \alpha_n L(x,y) \|fbm(x,y)\| \quad (1)$$

où α_n est un facteur de poids, $L(x,y)$ la luminance au point, et fbm la fonction de bruit. En multipliant par la luminance au point, on accentue le bruit autour de la zone désirée.

La composition finale de l’image C_f est effectuée ainsi :

$$C_f(x,y) = C_i(x,y) + (B(x,y) + N(x,y))\alpha_b \quad (2)$$

où C_i est l’image initiale, B l’image en sortie de la composition entre le dernier flou et l’image issue du premier *downscale* et α_b le poids associé au *bloom*.

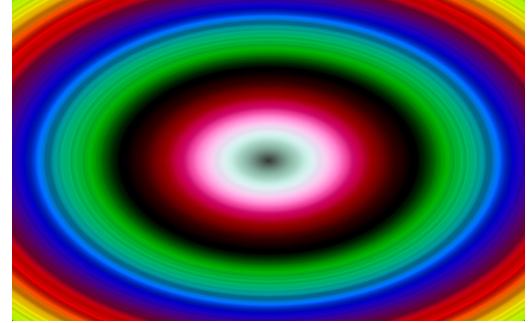


Figure 2 – Exemple texture générée pouvant être utilisée pour colorer les *lens flares*. Les valeurs des vecteurs sont $a = (0.2, 0.2, 0.2)$, $A = (0.6, 0.6, 0.6)$ et $\omega = (12.5, 20.5, 17.2)$

3 Lens Flare Approximation

L’approximation de *lens flare* se décompose en deux phases, premièrement la génération de *ghosts* et son échantillonnage de couleurs, ensuite la génération de halos. Dans un dernier temps, un effet de *sunburst*, ainsi qu’un *rgbshift* sont appliqués. L’image en entrée est de la même manière que pour le *bloom*, une image issue d’un filtre d’*intensity thresholding*.

Génération de ghosts. Le principe pour générer les *ghosts* est d’effectuer une dispersion centrée sur le centre de l’écran. Le vecteur de dispersion est calculé ainsi :

$$d = c - pf_d \quad (3)$$

où p correspond aux coordonées écran du pixel, c au centre de l’écran, et f_d la force de dispersion. Pour un pixel p , la valeur de *ghost* correspondante est la suivante :

$$G(p) = \sum_{n=1}^N I(\lfloor p + dn \rfloor) w_G \quad (4)$$

$$w_G = 1 - \frac{c - (p + dn)}{\|c\|} \quad (5)$$

où N est le nombre maximal de *ghosts* à faire apparaître, w_G un facteur de poids qui diminue lorsque p s'écarte du centre de l'écran, et I l'image en entrée qui a subit l'*intensity thresholding*. Le facteur de poids permet de diminuer la puissance des *ghosts* sur les bords de l'écran.

Echantillonage des couleurs. Une fois les *ghosts* générés, ceux-ci sont de la couleur de leur luminance. Pour pouvoir leur donner une couleur qui sera différente en fonction de la distance du centre de l'écran, on va générer une texture à une dimension (Figure 2). Pour générer la texture on va associer à chacune des longueurs d'onde un signal différent. Ce signal est de la forme :

$$C(p) = a + A \sin(\omega ||p - c||) \quad (6)$$

où a , A et ω sont des vecteurs 3 dimensions pour décrire les différents signaux pour chacune des longueurs d'onde. Ce signal évolue en fonction de la distance entre la position courante du pixel et le centre de l'écran ce qui va permettre de colorer le *lens flare* d'une manière cohérente. La couleur est échantillonée de la manière suivante :

$$l_p = ||1 - 2p||^2 \quad (7)$$

$$G_c(p) = (1 - f_{bm}(l_p, l_p))C(p) \quad (8)$$

où l_p dilate et centre les coordonnées du pixel sur $(0,0)$ pour pouvoir générer un bruit centré sur le centre de la texture et permettre diminuer l'uniformité de la combinaison de couleur.

Pour obtenir la valeur finale, on va multiplier G par la texture colorée G_c :

$$G_f(p) = G(p)G_c(p) \quad (9)$$

Halos. Les halos sont des artefacts visibles sur les bords de l'écran seulement.

$$h = \frac{\hat{d}}{2} \quad (10)$$

$$H(p) = I(p + h)w_H \quad (11)$$

$$w_H = \frac{||c - [p + h]||}{||c||} \quad (12)$$

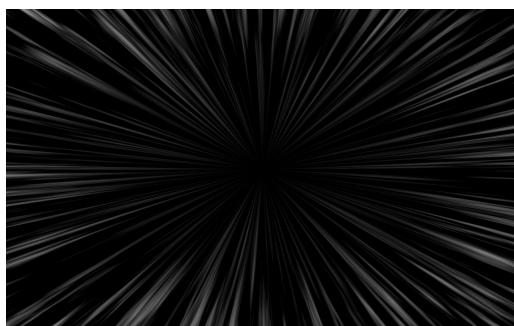


Figure 3 – Texture de bruit radial utilisé pour le *sunburst*, on peut voir ici l'influence du poids w_{SB} qui va éliminer les parties centrales.

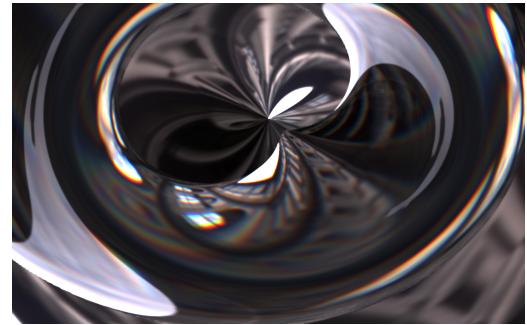


Figure 4 – Texture repliée sur elle-même par rapport au centre de l'écran. Cette texture est utilisée pour échantillonner le halo du *lens flare*.

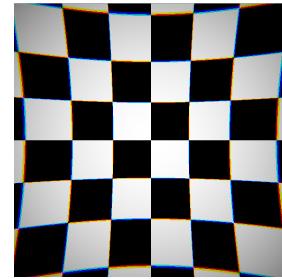


Figure 5 – Effet généré par la distortion de lentille [3] $k=-0.75$, et un effet de vignette

où h est une réduction du vecteur de dispersion normalisé, on récupère ensuite des valeurs dans une texture repliée sur elle-même autour de son centre (Figure 4), seulement sur les zones de forte luminance. Le poids w_H a pour effet d'exclure les valeurs centrales.

On va composer finalement notre *lens flare* de la manière suivante :

$$LF(p) = G_f(p) + H(p) \quad (13)$$

Sunburst. La texture de sunburst se calcule de cette façon :

$$r = ||p_c||^2 \quad (14)$$

$$SB(p) = LF(p)(1 - f_{bm}(r, atan(l_p))w_{SB}) \quad (15)$$

$$w_{SB} = ||c - p|| \quad (16)$$

où w_{SB} est un poids qui diminue en fonction de la distance entre le fragment et le centre de l'écran. Le bruit va être créé par rapport à l'angle orienté par les coordonnées du pixel par rapport au centre, ainsi que par la distance au centre. Ceci va générer un bruit radial centré autour du centre de l'écran (Figure 3).

Dans un dernier temps, un effet de *RGB shift* est réalisé sur le résultat du *sunburst* pour approximer l'effet visible. Celui-ci est plus conséquent sur les bords de l'écran. La distortion de lentille [3] qui est appliquée a été modifiée pour pouvoir effectuer une aberration chromatique. Ces échantillons sont prélevés de la même manière que lors d'une approximation de l'équation de

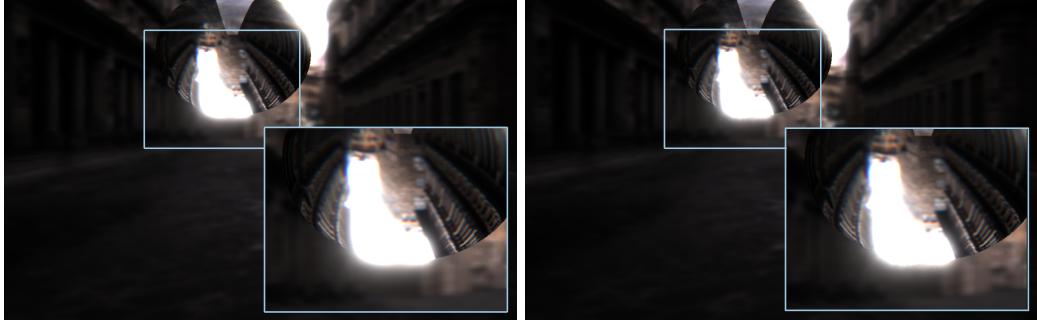


Figure 6 – Comparaison entre l’effet de *bloom* avec bruit (à droite) et sans bruit (à gauche).

Fresnel avec aberration chromatique [6] (Listing 14.4). L’effet est accentué et visible seulement sur les bords de l’écran (Figure 5).

4 Tone Mapping

L’opérateur de *tone mapping* utilisé est celui présenté par Reinhard et al. [4]. L’opérateur utilise la luminance moyenne. Il est possible de la calculer sur l’ensemble de l’*environment map*, cependant pour améliorer le résultat, il préférable de la calculer par *frame*. La méthode de réduction parallèle permet de calculer rapidement la moyenne d’une texture. Ainsi en fonction de la luminance moyenne de la scène, l’opérateur de *tone mapping* adaptera le rendu par *frame*.

5 Résultats

On peut voir l’influence du *starburst* sur le *lens flare* (Figure ??). L’inconvénient de cette approximation est que la forme des *flares* générés dépendent directement de la forme de la zone de luminance après seuillage. Le bruit FBM ajouté sur le *bloom* donne des résultats mitigés. En effet, lorsque la distance entre la caméra et la source de forte luminance est importante, le bruit donne de bons résultats. Lorsque celle-ci est proche, le bruit devient trop perceptible et le résultat très médiocre. Il a été envisagé d’utiliser un bruit radial (Figure 3) centré sur les zones de forte luminance. Mais il est difficile d’estimer à chaque *frame* le centre des zones de forte luminance sur lequel placer ce bruit radial. Mais le résultat serait grandement amélioré. Une solution envisageable au problème de la perception du bruit serait de réduire celui-ci en fonction de la position de la caméra.



Figure 7 – Résultats de *lens flare* avec bruit FBM ajouté sur le *bloom*.

6 Conclusion

Les *lens flare* ainsi que le *bloom* ne sont pas des effets basés sur des modèles physiques. Cependant, leur approximation peut donner d’assez bon résultat, et améliorer la perception d’une scène lorsque celle-ci est en HDR. L’avantage réside également dans le fait que ce sont des effets moins complexes à mettre en place.

Références

- [1] John Chapman. Pseudo lens flare. <http://john-chapman-graphics.blogspot.ca/2013/02/pseudo-lens-flare.html>.
- [2] Matthias B. Hullin, Elmar Eisemann, Hans-Peter Seidel, and Sungkil Lee. Physically-based real-time lens flare rendering. *ACM Trans. Graph. (Proc. SIGGRAPH 2011)*, 30(4) :108 :1–108 :9, 2011.
- [3] Andersson Technologies LLC. Syntheyes lens distortion algorithm. <http://www.ssontech.com/content/lensalg.htm>.
- [4] Erik Reinhard, Michael Stark, Peter Shirley, and James Ferwerda. Photographic tone reproduction for digital images. *ACM Trans. Graph.*, 21(3) :267–276, July 2002.
- [5] Tobias Ritschel, Matthias Ihrke, Jeppe Revall Frisvad, Joris Coppens, Karol Myszkowski, and Hans-Peter Seidel. Temporal Glare : Real-Time Dynamic Simulation of the Scattering in the Human Eye. In *Proceedings Eurographics 2009, Munich 30 March—3 April 2009*, 2009.
- [6] Randi J. Rost, Bill M. Licea-Kane, Dan Ginsburg, John M. Kessenich, Barthold Lichtenbelt, Hugh Malan, and Mike Weiblen. *OpenGL Shading Language (3rd Edition)*. Addison-Wesley Professional, 3 edition, 7 2009.
- [7] Intel Software. Compute shader hdr and bloom. <http://software.intel.com/en-us/articles/compute-shader-hdr-and-bloom>.