

**Rapport de stage**Intégration de l'éclairement global dans la plateforme **HERA**  
Du 4 mars au 24 juillet 2025

Amélia Di Martino



**Tuteurs entreprise** :Jean-Philippe Farrugia, Fabrice Jaillet  
**Tuteur universitaire** : Raphaëlle Chaine

**Établissement** : Université Lyon 1  
**Année de formation** : M2 Informatique, ID3D  
**Entreprise d’accueil** : LIRIS

Année universitaire 2024-2025

**Sommaire**

[Remerciements 3](#_Toc1)

[1. Introduction 4](#_Toc2)

[1.1 Présentation de l’entreprise 4](#_Toc3)

[1.2 Contexte 4](#_Toc4)

[1.3 Sujet du stage 4](#_Toc5)

[1.4 Outils et environnement de travail 4](#_Toc6)

[2. Travail réalisé 4](#_Toc7)

[2.1 Gestion des matériaux 4](#_Toc8)

[2.2 Intégration de l’éclairement global 4](#_Toc9)

[2.2.1 Techniques 4](#_Toc10)

[2.2.2 Résultats 4](#_Toc11)

[3. Conclusion 4](#_Toc12)

[Sources 4](#_Toc13)

[Annexes 4](#_Toc14)

# Remerciements

Dans un premier temps, je tiens à remercier mes deux tuteurs de stage, Jean-Philippe Farrugia ainsi que Fabrice Jaillet, de m’avoir encadrée, suivie et conseillée tout au long de ce stage.  
Malgré la distance, vous avez toujours été présents, patients et de bon conseil pour m’aider à avancer lors de ce stage.

Je remercie Raphaëlle Chaine de m’avoir parlé de ce sujet de stage, sujet mêlant plusieurs passions qui me touchent.

Je tiens à remercier Noah Bertholon, avec qui j’ai pu échanger sur certaines problématiques de mon stage, dont la lumière fût souvent rafraîchissante.

De même, je tiens à remercier Jean-Claude Iehl pour ses conseils et les références bibliographiques qui m’ont grandement aidée lors de ce stage.

Ce stage a été pour moi très enrichissant, j’ai eu la chance de beaucoup apprendre, par moi-même et grâce aux autres.

# 1. Introduction

## 1.1 Présentation de l’entreprise

Le **LIRIS** (Laboratoire d’infoRmatique en Image et Système d’information) est une unité mixte de recherche qui mélange CNRS, INSA Lyon, l’UBCL Lyon 1, l’Université Lumière Lyon 2 ainsi que l’École Centrale de Lyon.

Il est ainsi de composé de 330 membres, divisés en plusieurs équipes avec leurs thématiques de recherche propres.

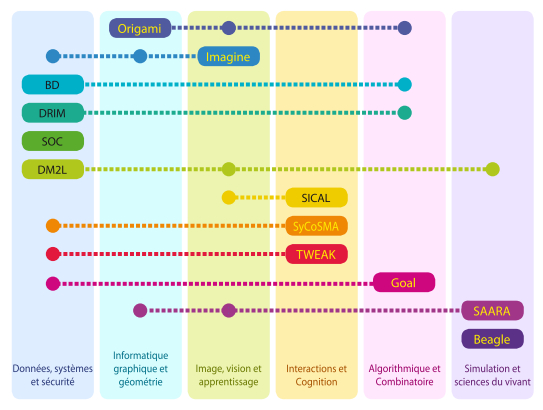


Figure 1 : Équipes et thématiques

Ce stage prend place au sein de l’équipe **Origami**, dont les thèmes de recherches gravitent autour de de l’informatique graphique et de la géométrie.

## 1.2 Contexte

Avant toute chose, il convient de présenter **HERA** (**H**istorically **E**nhanced **R**eality **A**pplication), l’application sur laquelle ce stage porte.  
Il s’agit d’une application web de réalité augmentée et réalité virtuelle, prévue pour des non-informaticiens afin de pouvoir créer et accompagner des visites guidées de sites historiques ou de musées.

L’application est codée en javascript + vue.js, afin de pouvoir l’utiliser de manière portative, sur téléphone, PC, tablette, casques VR...  
La partie affichage 3D utilise ThreeJS.

L’application est divisée en deux parties :

Un **éditeur**, pour permettre au guide de préparer la visite :  
Son objectif n’est pas de recréer Blender, mais il permet quelques opérations simples : rajouter des maillages (au format GLB), les bouger, changer les propriétés basiques des matériaux...  
Il est aussi possible de rajouter des labels (des zones de textes), ainsi que des timings pour gérer le moment où ils apparaîtront lors de la visite.

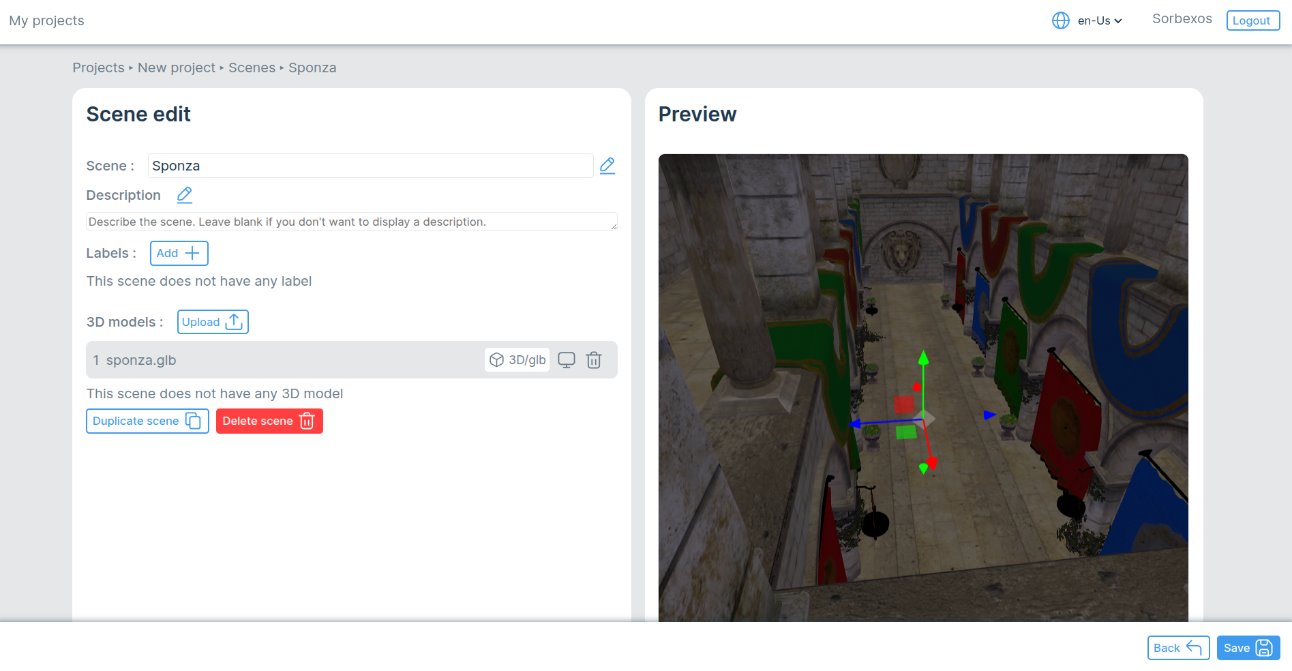


Figure 2 : Partie éditeur de HERA

Un **viewer**, permettant au visiteur de profiter de la visite. Il s’agit simplement de calibrer la scène en plaçant la caméra par rapport à un repère prévu, et ainsi profiter de la scène en réalité augmenté sur son appareil.

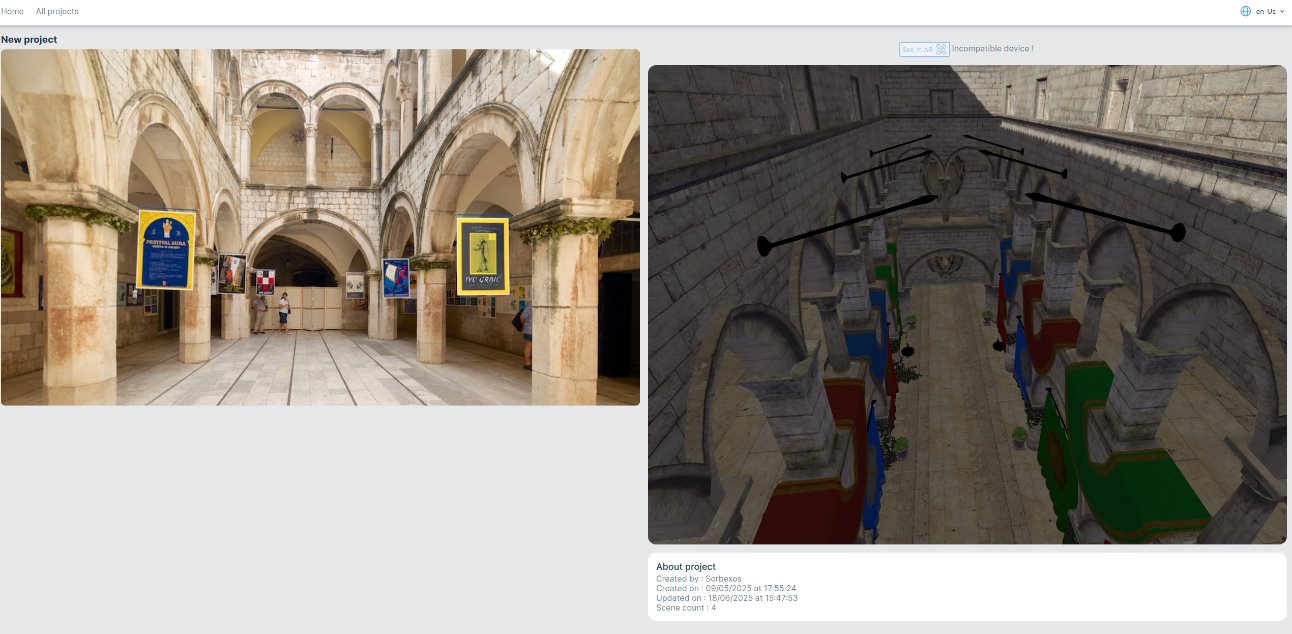


Figure 3 : Partie viewer de HERA

## 1.3 Sujet du stage

L’objectif premier de ce stage était de rajouter des fonctionnalités à l’application.

Plus largement, l’objectif de ce stage est d’utiliser les travaux réalisés afin de pouvoir étudier l’impact des techniques d’**illumination globale** sur la perception humaine.

Avant ce stage, l’éclairage de la scène restait sommaire : une lumière ambiante ainsi qu’une lumière directionnelle pour simuler le soleil.

Le but est donc de pouvoir créer des tests afin de comparer l’impact de l’illumination globale, ainsi que pouvoir avoir différents présets (jour, nuit...) d’éclairage.

## 1.4 Outils et environnement de travail

J’ai passé mon stage dans les boxs prévu pour les stagiaires.  
J’ai utilisé mon propre ordinateur portable, ayant besoin d’un ordinateur avec un peu de puissance de calcul.  
Pour ce qui est des outils informatique, j’ai principalement utilisé VScode, Chrome pour utiliser l’application, ainsi que Gitlab.

# 2. Travail réalisé

## 2.1 Gestion des matériaux

Dans l’objectif de me familiariser avec la base de code, j’ai passé les premières semaines à implémenter la possibilité de changer les propriétés des matériaux des mesh importés sur l’éditeur.

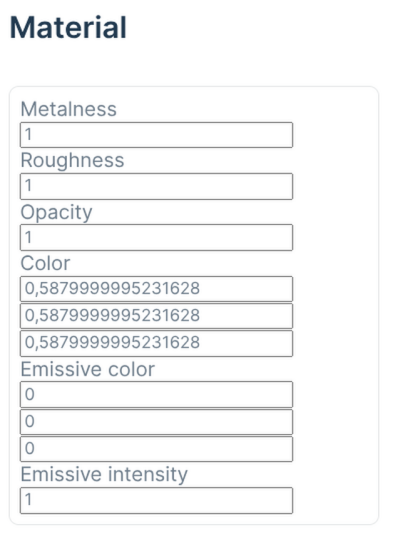


Figure 4 : Menu d'édition de matériaux

Cela était un exercice intéressant, sachant que la base de code n’était pas du tout prévue pour faire cela.  
En effet, chaque objet GLB était importé sous forme d’« asset ». Un asset est une arborescence de différents mesh, qui chacun possède son propre matériau.

Changer le matériau d’un objet impliquait donc de complètement modifier la manière dont les objets étaient importés, afin de pouvoir sélectionner un élément spécifique de la scène, et non toute la scène.

Lorsqu’on importe un asset, il faut traverser son arborescence de mesh afin de les ajouter à la scène, ainsi qu’à la base de donnée, afin de pouvoir stocker leurs spécificités (transformations géométriques et matériau).

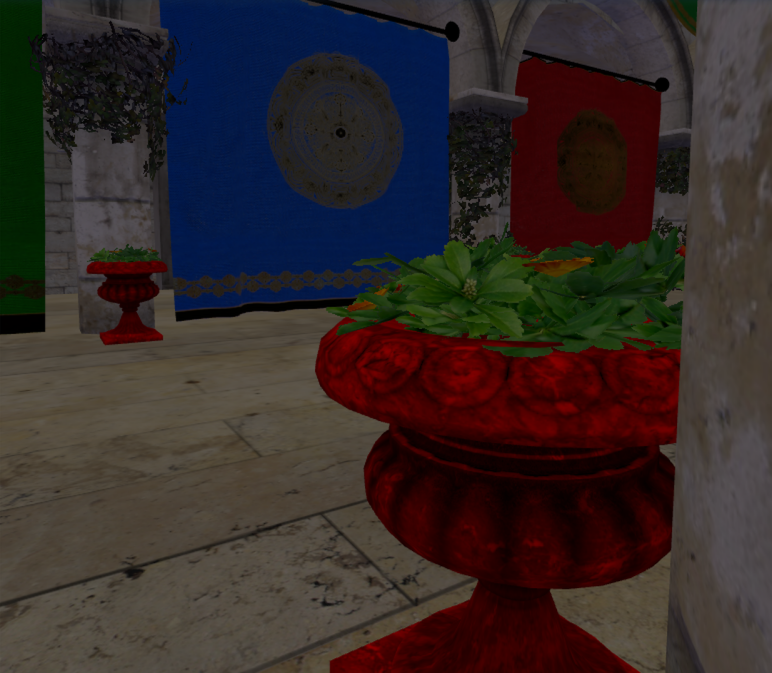
J’ai dû me familiariser avec la sauvegarde des éléments de la scène dans la base de données côté backend, ainsi que leur utilisation dans le frontend.

*(rajouter des schémas explicatifs comme t’avais dessiné sur ton carnet)*

Cette tâche convenait tout à fait à la découverte et la compréhension de la base de code.  
Il m’a fallu m’inspirer de ce qui existait déjà (enregistrement des assets, des labels de texte, des scènes...), et plus globalement du fonctionnement de l’API.  
De même, il m’a fallu comprendre le chemin des objets du chargement de la page à leur affichage dans ThreeJS.

Figure 6 : Pot de fleurs rouge

Figure 5 : Pot de fleurs gris



## 2.2 Intégration de l’éclairement global

Dans un premier temps, il convenait de choisir une technique d’illumination globale.  
En effet, les contraintes sont fortes : l’éclairage doit être réaliste, le tout en temps réel sur n’importe quel support (pc, tablette, téléphone...).

En vue des fortes contraintes, il fallu faire un travail important de recherche afin de cataloguer les différentes techniques pouvant convenir.

Nous présenterons dans un premier temps les quelques techniques qui ont retenu notre attention, et ensuite la technique principale que nous avons choisi.

### 2.2.1 Techniques intéressantes

**SSGI (Screen Space Global Illumination)**

Première technique présentée comme pouvant fonctionner en temps réel.  
Une [lib ThreeJS](https://github.com/0beqz/realism-effects) implémentant cette technique, il nous a paru pertinent de l’essayer.  
Le but n’était pas forcément n’implémenter une technique à partir de zéro : si quelque chose existait déjà et donnait des résultats intéressants, nous l’aurions utilisé.

L’idée est simple : prendre en compte uniquement les objets à l’écran : cela limite évidemment beaucoup les temps de calcul, même si on perd en réalisme, car uniquement ce qui apparaîtra à l’écran influera l’illumination de la scène.

Les résultats ne répondaient finalement pas à nos attentes : la technique, bien plus légère que d’autres, restait trop lourde pour être utilisée sans GPU, et mettait trop de temps à converger.



Figure 7 : SSGI sur la Sponza

Cette image (**figure 7**) a convergé en ~30 secondes, ce qui est bien trop long.  
Nous voudrions rester à 30-60 images par secondes, et nous sommes ici à 6 IPS.

Cette solution aurait aussi été avantageuse car elle permettait d’avoir de l’illumination globale sur ThreeJS, sans avoir à bidouiller dans les shaders de ce dernier.  
En effet, ThreeJS ne permet pas vraiment de modifier facilement sa pipeline de rendu.

**Instant radiosity (Virtual Lights)**

Cette technique a l’avantage d’être très simple à comprendre :

Elle remplace la contrainte de lancer beaucoup de rayons par pixel à chaque frame, par le fait d’avoir beaucoup de sources de lumières « secondaires » qui simulent les différents rebonds.

Dans un premier temps de baking, avant le rendu, on va créer des sources de lumières secondaire.  
En partant d’une source de lumière primaire (**L1 sur la figure 8)** on lance des rayons de manière aléatoire.  
Lorsqu’un rayon touche une surface, on va créer une source de lumière secondaire (**L1’, L2’... sur la figure 8)**.  
On peut répéter l’opération autant de fois que souhaité, tant qu’il reste de l’énergie dans notre rayon.

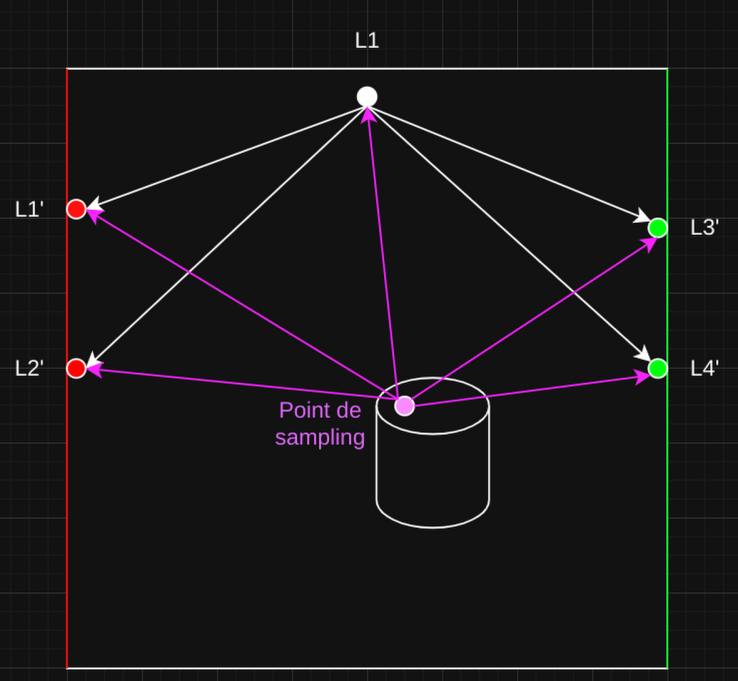


Figure 8 : Instant radiosity

Une fois nos lumières ponctuelles crées, il faut les transmettre au GPU.  
Et là...  
C’est le moment où nous nous arrêtons pour expliquer quelques contraintes liées à ThreeJS et à WebGL 2.0 : il n’y a **pas de buffer** dans l’API de ThreeJS. Cela signifie qu’il faudra passer par de textures afin de transmettre des données en quantité.  
En l’occurrence, ce que nous voulons transmettre ici, ce sont les positions des lumières, ainsi que leur puissance.  
(Par ailleurs, pas de compute shader non plus, même si cela n’est pas embêtant dans ce cas d’utilisation.)

Une fois cela compris, il ne manque plus qu’une étape : tester si oui ou non, sampler à chaque frame un nombre important de lumière ponctuelles est supportable pour l’iGPU de mon PC portable.

La réponse est : **non**, on commence à voir nos IPS divisés par deux à partir d’une cinquantaine de lumière, ce qui n’est pas suffisant du tout pour simuler de l’illumination globale dans une pièce de taille respectable.

**Area Lights**

Une dernière solution a été essayée avant de passer à autre chose : les area lights de ThreeJS.



Figure 9 : Rect Area Light de ThreeJS

Cette solution est intéressante car elle permet d’avoir des lumières douces à faible coût.  
De plus, elle est déjà implémentée dans ThreeJS, et représente donc un avantage non négligeable en terme de temps d’implémentation.

Il y a deux problèmes principaux :

* L’utilisateur devra placer lui-même les area lights
* Pas d’ombre

Cette solution a donc été abandonnée, même s’il aurait été possible d’approximer de ombres en rajoutant des lumières ponctuelles à la surface des Rect Area Lights.

### 2.2.2 Light Probe Volume

Une documentation scientifique experte, quasiment pas d’exemples d’implémentations disponibles en ligne, demande un bidouillage intensif de la pipeline de rendu de ThreeJS, gestion de beaucoup de cas particuliers...   
La solution parfaite non ?

Et si !  
Malgré une implémentation ardue, cette solution a été retenue pour plusieurs raisons :

* Tourne sans soucis en temps réel
* Permet des résultats réalistes
* Plutôt simple à comprendre
* Permet de mettre à jour l’illumination reçue par des objets de la scène lorsqu’ils se déplacent (nécessaire car des animations seront rajoutées à HERA à l’avenir)

### 2.2.3 Résultats

# 3. Conclusion

# Sources

Figure 1 : https://liris.cnrs.fr/liris

# Table des figures

[Figure 1 : Équipes et thémathiques 4](#_Toc1)

# Annexes